FPGA Design and Implementation of Systolic Array-Based Viterbi Decoders

Man Guo

A Thesis

in

the Department

of

Electrical and Computer Engineering

Presented in Partial Fulfilment of the Requirements
for the Degree of Master of Applied Science at
Concordia University
Montreal, Quebec, Canada

September 2002

ABSTRACT

FPGA Design and Implementation of Systolic Array-Based Viterbi Decoders

Man Guo

The Viterbi algorithm is known to provide an efficient method for the maximum likelihood decoding of convolutional codes. The algorithm can be formulated by employing a matrix-vector computation and it can be implemented in hardware based on a systolic array architecture. It has been shown that in this implementation, the strongly-connected trellis decoding method can be used to improve the efficiency of the hardware utilization and the throughput of the decoding. However, the employment of the strongly-connected trellis decoding method results in an excessive amount of ACS (addition, comparison and selection) computations in the decoding process as the constraint length becomes large. Further, in the systolic array architecture, the hardware complexity increases exponentially with the constraint length of the code. This makes the adoption of the systolic array architecture not feasible in the design of the Viterbi decoder for decoding a convolutional code with a large constraint length.

In this thesis, a design and FPGA implementation of a Viterbi decoder with a constraint length of 9 and code rate of 1/2 is presented. In this design, a novel systolic array architecture with time multiplexing, arithmetic pipelining and clock-to-data skews tolerance is developed. Further, by modifying this Viterbi algorithm, an adaptive Viterbi algorithm that is based on strongly-connected trellis decoding is proposed.

Using the proposed adaptive algorithm, a design and FPGA implementation of a low-power adaptive Viterbi decoder with a constraint length of 9 and code rate of 1/2 is presented. The systolic array-based architecture used in this adaptive Viterbi decoder is a modified version of the architecture used for the non-adaptive Viterbi decoder in that the latter is modified to include the modules, which are needed for generating the survivor information and for eliminating the spurious toggles in the adaptive Viterbi decoding process. It is shown that the proposed algorithm can reduce up to 70% the average number of ACS (addition, comparison, and selection) computations over that by using the non-adaptive Viterbi algorithm, without a degradation in the error performance. This results in lowering the switching activities of the logic cells, with a consequent reduction in the dynamic power. Further, it is shown that the total power consumption in the implementation of the proposed algorithm can be reduced by up to 43% compared to that in the implementation of the non-adaptive Viterbi algorithm, with a negligible increase in the hardware.

ACKNOWLEGEMENTS

First and foremost, I would like to thank my supervisors, Dr. Omair Ahmad and Dr. M.N.S. Swamy, for giving me the opportunity to work on this thesis project. Their insight, valuable advice and guidance throughout this study have contributed significantly to this research. Moreover, without the financial support from my supervisors, it would have been extremely difficult to accomplish this investigation. I gratefully acknowledge the support of Micronet, a National Network of Centres of Excellence, through a research grant awarded to Drs. Ahmad and Swamy.

I am greatly indebted to my parents for their love and care, and for their constant support and encouragement throughout my education.

I am very grateful to my husband, Jiyao, and my son, Yiran, for their love and sacrifices, and for providing me encouragement and inspiration during my graduate study at Concordia University.

I would like to specially thank Ted Obuchowicz and Wojciech Galuszka for providing me the help and technical support in the VLSI lab. I would also like to express my fond appreciation to Yi Yang, Wei Wang, and many other friends and colleagues for support and encouragement during my graduate study at Concordia University.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

ASIC: Application Specific Integrated Circuit

AWGN: Additional White Gaussian Noise

ACS: Addition, Comparison, and Selection.

BER: Bit Error Rate

BPSK: Binary Phase Shift Key

CDMA: Code Division Multiple Access

DFF: D-type flip-flop

FPGA: Field Programmable Gate Array

FEC: Forward Error Correction.

FIFO: First-In First-Out

LUT: Look-Up Table

LIFO: Last-In First-Out

RAM: Random Access Memory

ROM: Read Only Memory

LIST OF SYMBOLS

$\mathbf{B_q}$: Adjacency matrix in the matrix-vector ACS computation

$b_q(i, j)$: The composite branch metric from state i of stage q-1 to state j of stage q of the strongly-connected trellis diagram

$\mathbf{C_{ij}}$: The codeword sequence corresponding to the composite branch

$C_{ij}[k]$: The k-th element of $\mathbf{C_{ij}}$

$\mathbf{C_{i0}}$: The codeword sequences with zero input sequence

$\mathbf{C_{0j}}$: The codeword sequences with zero encoder state

C_{OUTavg}: The average load capacitance

d_m^q: The minimum distance of all the survivor paths

d_n^S: The survivor decision of state S_n

$d_{n,n-1}$: The composite 2-bit radix-4 decision

$d_{n,\,n-1,\ldots,n-(K-2)}$: The composite $(K-1)$-bit radix-$2^{(K-1)}$ decision

F_{MAX}: Clock frequency

K: Constraint length of a convolutional code

k: The number of inputs in a convolutional encoder

K_P: A constant whose value depends on the family of FPGA

m: The number of stages in the shift register used for a convolutional encoder

min^{-1}: The operation of finding the state number in the (q-1)-th stage that yields the minimum distance of the paths at state j of state q

min: The operation of taking the minimum distance of the paths at state j of state q

n: The number of outputs in a convolutional encoder

N: The number of states in a trellis diagram

N_{LC}: The number of the logic cells used by the application, F_{MAX} the clock frequency

N_{OUT}: The number of outputs in the design

$\mathbf{P_q}$: Path metric row vector in the matrix-vector ACS computation

$p_q(j)$: The partial path metric from the initial state *0* to state j of stage q of the trellis diagram

P_{STAT}: Static power consumption resulting from leakage current by an inactive device connected to the power supply

P_{INT}: Internal power dissipation caused by the charging and discharging the capacitance on any internal nodes that are switched

P_{IO}: Input and output power dissipation resulting from the charging and discharging of the external load capacitors connected to device pins, and the pull-ups used on the inputs

r: Code rate of a convolutional code

$Sur_q(j)$: The survivor at state j of stage q

Vcc: Supply voltage

V_{swing}: Output swing voltage

S_n: An arbitrary state of stage n

$S_q(j)$: The survivor information at the q-th stage

Tog_{LC}: Average toggling rate at each clock

Tog_{OUT}: Average toggling rate of the output at each clock

$'\otimes'$: The ACS operation according to the likelihood criterion

Δ_{max}: The maximum dynamic range of path metric in the conventional Viterbi algorithm

λ_{max}: The maximum branch metric.

Γ_{bit}: The worldlength of a path metric

CHAPTER 1

INTRODUCTION

1.1 Introduction

Convolutional codes are a type of basic error correction codes whose encoding process can be viewed as convolving the message stream with the impulse response of the code. The Viterbi algorithm [1] is known to be an efficient method for maximum likelihood decoding of convolutional codes over a memoryless noise channel. Convolutional codes and the Viterbi algorithm provide a strong forward error correction (FEC) scheme, which has been widely utilized in digital communication applications. It has been shown that the larger the constraint length K used in a convolutional encoding process, the more powerful the code produced. Thus, a convolutional code with a relatively large constraint length of K = 9 is used for the IS-95 code-division multiple access (CDMA) wireless communication standard [2]. The use of a large constraint-length convolutional code, on one hand, can provide a better error correction performance. On the other hand, the complexity of the Viterbi decoding process, both in terms of computations and memory requirements, increases exponentially with the constraint length K of the code. As a result, it would be

difficult to have a hardware implementation of a Viterbi decoder with a large constraint length, such as K= 9, to meet the requirements of the power, speed and area. In recent year, Viterbi decoders have been mostly used in mobile systems that require portable battery operations, thus making the power consumption a critical concern to the designers.

The focus of this thesis is on a study of low-power design of a Viterbi decoder. In this chapter, first the currently available architectures for a Viterbi decoder are first reviewed, and then the challenges by employing these architecture to design and implement a large constraint-length Viterbi decoder are discussed. Next, the existing approaches for low-power design of the Viterbi decoders with K=9 are analyzed. Finally, the scope and organization of the thesis are presented.

1.2 Architectures for Implementation of a Viterbi Decoder

Designing an appropriate architecture plays a major role in achieving an efficient hardware implementation of a Viterbi decoder. In this section, two types of Viterbi decoder architectures, namely, the butterfly architecture and the systolic array architecture, are reviewed and discussed in order to propose a novel Viterbi decoder architecture in a later chapter.

1.2.1 Butterfly Architecture

Viterbi algorithm can be viewed as a process which finds the shortest path in a trellis diagram [3] to match the received sequence by a dynamic programming technique [4]. A trellis diagram is obtained by expanding the state diagram of a convolutional encoder in

time, and it can be decomposed into butterfly state-pairs in which each pair of its origin and destination states are interconnected together. Figure 1.1 shows an example of a trellis diagram with K=3 and its decomposed butterfly state-pairs. Decomposing a trellis into butterfly state-pairs is straightforward, and it does not change the interconnections of the trellis. Therefore, the commonly used Viterbi decoders are designed based on the butterfly architecture in which each butterfly state-pair in a trellis is processed by a module that performs three operations: addition, comparison, and selection (ACS). This module is usually called ACS butterfly module, shown in Figure 1.2 [5, 6]. In this module, each of the four

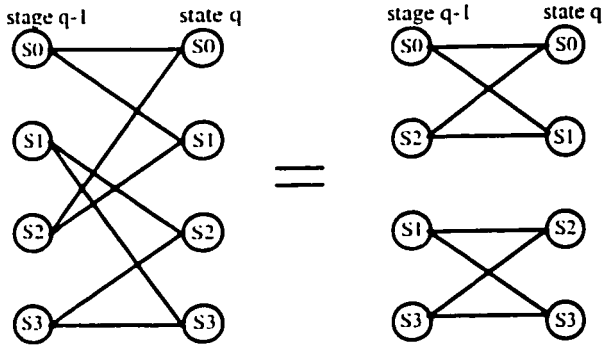
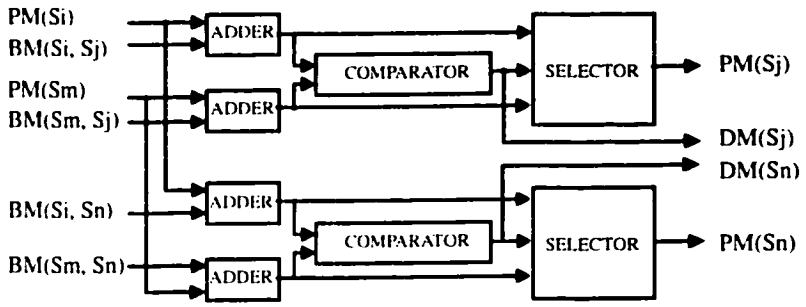


Figure 1.1 A trellis diagram (K =3) and two butterfly pairs

adders is used to compute one of the four path metrics at stage q+1 by adding the branch metric corresponding to one of the four butterfly interconnected branches to the corresponding path metric at stage q. Then, each of the two comparators is used to make a comparison between the two computed path metrics at the state of stage q+1. Finally, each of the two selectors is used to select the path metric with the shortest distance (Euclidean or

Hamming distance) at the state of stage q+1 depending on the comparison result by the corresponding comparator.



BM(Si, Sj), BM(Sm, Sj), BM(Si, Sn) and BM(Sm, Sn) : Branch Metric

PM(Si) , PM(Sm): Path Metric of state Si and state Sm

DM(Sj), DM(Sn): Decisions of state Sj and state Sn

Figure 1.2 Butterfly structure of an ACS module

The butterfly architecture-based Viterbi decoders can be categorized into two classes; serial and parallel. A serial Viterbi decoder sequentially processes the ACS computations at each trellis stage by iteratively using one ACS butterfly module. On the other hand, in a parallel Viterbi decoder, ACS butterfly modules corresponding to all butterfly state-pairs in a trellis diagram are designed to process the ACS computations simultaneously. Obviously, a serial Viterbi decoder can be advantageous from the viewpoint of power dissipation and hardware complexity. However, the number of states per trellis stage increases exponentially with the constraint length K of a code. If a serial architecture is used to implement a large constraint-length Viterbi decoder, there will be a large number of iterations for the ACS computations at each trellis stage. As a result, the speed of a serial Viterbi decoder will be greatly slowed down. On the contrary, a parallel Viterbi decoder is no

doubt able to achieve a high-speed decoding due to its parallel computations. However, for decoding a convolutional code with a large K, a large number of components should be involved to process ACS computations, and the interconnections between components will become very complex. Consequently, the amount of power dissipation and hardware resources (area) would increase exponentially.

1.2.2 Systolic Array Architecture

Unlike the serial and parallel architectures that are designed by decomposing states in a conventional low-connectivity trellis diagram into butterfly pairs, a systolic array architecture suggested by Chang and Yao [7] is derived by formulating the Viterbi algorithm employing matrix-vector multiplication. Since the adjacency matrix in this formulation is very sparse, the utilization of the array processors in this systolic architecture is rather poor. In order to resolve this problem, Chang and Yao also proposed the strongly-connected trellis decoding in [7]. As shown in Figure 1.3, a strongly-connected trellis diagram

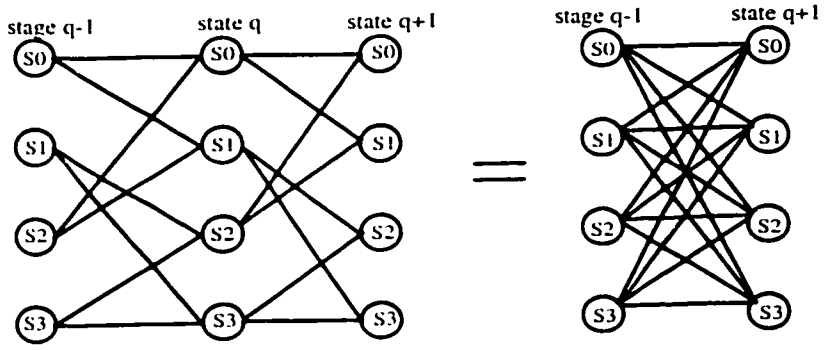


Figure 1.3 A strongly-connected trellis diagram

is obtained by merging the stages in each group of (K-1) stages in the original low-connectivity trellis, in such a way that every state in the (K-1)-th stage of the group is reachable from the states of the preceding (K-2) stages of the group. Obviously, by using strongly-connected trellis decoding method, the adjacency matrix in the matrix-vector formulation is no longer sparse. As a result, the utilization of systolic array processors as well as the throughput of the decoding can be enhanced

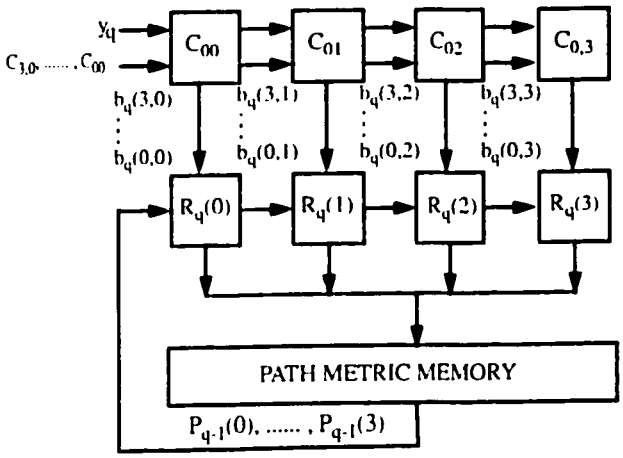


Figure 1.4 A systolic architecture for a Viterbi decoder with K=3

Figure 1.4 shows the systolic array architecture for a Viterbi decoder with K=3 presented in [7]. This is derived by employing the systolic array architecture for processing a matrix-vector multiplication [8]. The modification carried out in this architecture is that array processors do not perform multiplications, but process ACS computations in order to find the shortest path in a trellis diagram. The matrix-vector ACS computation in [7] is given as $P_q = P_{q-1} \otimes B_q$, where $\mathbf{P_q}$ represents the path metric row vector of the q-th stage, whose j-th element is denoted by $P_q(j)$, and $\mathbf{B_q}$ the adjacency matrix whose ij-th element is denoted by $b_q(i, j)$. This matrix-vector ACS computation is processed by two interconnected linear

systolic arrays. In the upper systolic array, the composite branch metric $b_q(i, j)$ is generated by calculating the Euclidean distance between the codeword C_{ij} and the received data y_q, where $C_{ij} = C_{i0} \oplus C_{0j}$, and $i, j = 0,..., 3$ [7]. The codeword C_{0j} stays inside each processor, and C_{i0} moves to the right. In the lower systolic array, the path metric $P_q(j)$ of stage q is computed. The intermediate ACS result denoted by $R_q(j)$ stays inside the j-th processor, whereas $P_{q-1}(j)$ of stage q-1 moves to the right. All data movements in the two systolic arrays are synchronized. This design can thus be viewed as a "result stay" systolic array architecture.

It is seen that the systolic array architecture presented in [7] is a class of pipelined array architectures that feature the properties of modularity, regularity, local interconnectivity, a high degree of pipelining, and highly synchronized multiprocessing. It can provide for different convolutional code structures, a general and local interconnection. However, the numbers of components and pipeline stages in the systolic array architecture presented in [7] increase exponentially with K. As a result, the design will be excessively hardware-resource and power consuming, and at the same time, clock-to-data skews caused by the deep pipelining will become critical for a proper functioning of the system. In addition, the processing times of the two operations, namely, the composite branch metric generation and the path metric update, will differ significantly as K becomes large. Consequently, it is a challenging task to ensure that the two types of processors with different processing times can be synchronized, without slowing down the clock-rate. Moreover, the employment of strongly-connected trellis decoding method carries a drawback that the amount of ACS computations becomes excessive as the constraint length K becomes

large. Consequently, this excessive amount of ACS computations will result in a large amount of power consumption in its hardware implementation.

1.3 Approaches to Reduce Power Consumption

In recent years, some attempts have been made for low-power design of Viterbi decoders with K=9. In this section, we discuss several features of such schemes.

The latest effort in the design of low power bit-serial Viterbi decoder, proposed by Chang, Suzuki and Parhi [9], is based on a parallel butterfly architecture with bit-serial arithmetic. In this design, the employment of bit-serial arithmetic for the ACS computations is the key in reducing the power consumption. By using the bit-serial arithmetic, the number of components and the complexity of the interconnections between components in the Viterbi decoder is reduced over that using a parallel butterfly architecture with parallel arithmetic. As a result, the power dissipation is reduced. A disadvantage for using bit-serial arithmetic technique is that extra memory elements are required to store intermediate results of the ACS computations. Since this design is targeted for application specific integrated circuit (ASIC) technology, this problem is solved at the circuit level by developing an area-efficient storage unit for the state metric, which is based on the first-in first-out (FIFO) architecture. In addition, since there are less switching activities generated in a FIFO register than in a conventional shift register, the use of the FIFO registers instead of the conventional shift registers in an ACS unit makes the design of the ACS unit power efficient.

The another low-power ASIC design of the Viterbi decoder has been proposed by Kang and Willson [10]. In this design, the power efficient serial butterfly architecture is obtained by combining a butterfly ACS unit with a flipped butterfly ACS unit. Furthermore, the gated-control scheme was employed to reduce spurious transitions on high-capacitance bus lines, and Gray code was used to reduce the number of transitions in path-memory addressing [10].

Beside the two low-power Viterbi decoders based on ASIC technology discussed above, two Viterbi decoders with a large constraint-length K=9 targeted for FPGA implementations have been proposed by Pandita [5] and by Park and Rho [6]. These two designs simply use a serial butterfly architecture, without using any other power reduction technique.

1.4 Scope of the Thesis

From the discussion above, it is clear that most low-power Viterbi decoders with K=9 are designed based on the serial butterfly architecture with little provision for high speed. There exists just one Viterbi decoder implementation based on the parallel butterfly architecture in which the employment of bit-serial arithmetic technique plays the role of moderately reducing the hardware complexity in order to provide a low-power design. At present, there is no implementation for a large constraint-length, for instance for K=9, Viterbi decoder using the systolic array architecture suggested by Chang and Yao [7]. The systolic array architecture in [7] can provide a moderate speed over the serial and parallel architectures due to its high degree of pipelining and multiprocessing. It can also provide, for different convolutional code structures, a general method for a simple adjacent cell

interconnection. It can be naturally mapped into an FPGA device, since (a) there are a large number of D-type flip-flops (DFFs) available in most commercial FPGA devices, such as the Xilinx VirtexII series, and (b) the primitive logic cells of FPGA devices are organized as an array. However, to design the Viterbi decoder for decoding a convolutional code with a large value of K, say, K=9 and a code rate of r=1/2, the complexity of the computations will render the adoption of the systolic array architecture in [7] not to be feasible.

In this thesis, first the conventional Viterbi algorithm is reformulated for the strongly-connected trellis decoding. Then, a design and implementation of the reformulated Viterbi algorithm for decoding the convolutional code with K=9 and r=1/2 using the Xilinx VirtexII-XC2V1000-4FG256 chip is presented. In this design, a novel systolic array-based architecture with arithmetic pipelining, time multiplexing and clock-to-data skews tolerance is developed. Further, an adaptive Viterbi algorithm that is based on strongly-connected trellis decoding is proposed by modifying the adaptive Viterbi algorithm based on low-connectivity trellis decoding. The objective of this proposed algorithm is to reduce the excessive amount of ACS computations resulting from the strongly-connected trellis decoding. By using this algorithm, the design and implementation of an adaptive Viterbi decoder for decoding the convolutional code with K=9 and r=1/2 using the Xilinx VirtexII-XC2V1000-4FG256 chip is presented. In this design, the proposed systolic array-based architecture for the Viterbi decoder is modified for the implementation of this adaptive algorithm. A process of spurious toggle reduction is developed to reduce power consumption in the design. Simulation results show that the proposed adaptive Viterbi

algorithm can reduce the large amount of ACS computations significantly while maintaining almost the same error performance as that of the non-adaptive Viterbi algorithm. This reduction in the amount of computations results in a substantial reduction in overall power consumption compared to the implementation of the non-adaptive Viterbi algorithm.

1.5 Organization of the Thesis

This thesis presents a study on low-power designs and implementations of systolic array-based Viterbi and adaptive Viterbi decoders. The thesis is organized as follows.

In Chapter 2, a review of the convolutional encoding process, the Viterbi decoding process, and the strongly-connected trellis decoding method is given. This is followed by the formulation of the Viterbi algorithm based on strongly-connected trellis decoding. In Chapter 3, a design and implementation of the Viterbi decoder for decoding the convolutional code with a constraint length of 9, and a code rate of 1/2 using the Xilinx VirtexII-XC2V1000-4FG256 chip is presented. A performance analysis of the decoder in terms of speed, power and FPGA resource utilization is carried out based on implementation results and estimation results by the VirtexII power estimator. In Chapter 4, an adaptive Viterbi algorithm that is based on strongly-connected trellis decoding is proposed, and a formulation of the proposed algorithm for hardware implementation of an adaptive Viterbi decoder is presented. A comparison between the proposed algorithm and the non-adaptive Viterbi algorithm in terms of the error performance and the amount of ACS computations based on the simulation results is presented. In Chapter 5, a design and implementation of a low-power adaptive Viterbi decoder with a constraint length of 9 and code rate of 1/2

using the Xilinx VirtexII-XC2V1000-4FG256 chip is presented. A performance analysis of this design in terms of speed, power and FPGA resource utilization is carried out, and the results of comparison with the Viterbi decoder are presented. Chapter 6 concludes the thesis by highlighting the findings of the investigation undertaken and by suggesting some possible future work in this area.

CHAPTER 2

A VITERBI ALGORITHM FOR THE STRONGLY-CONNECTED TRELLIS DECODING

2.1 Introduction

As noted in Chapter 1, convolutional codes and the Viterbi algorithm are widely used in digital communication systems as a forward error-correction scheme. Convolutional codes were first introduced by Elias [11] in 1955 as an alternative to block codes. Shortly thereafter, Wozencraft [12] proposed sequential decoding as an efficient decoding scheme for covolutional codes, and experimental studies soon began to appear. In 1963, Massey [13] proposed a less efficient but simpler-to-implement decoding method called threshold decoding. In 1967, Viterbi [14] introduced a decoding algorithm that was relatively easy to implement for convolutional codes with constraint length K not larger than 9, and has since become to be known as the Viterbi algorithm. Later, Omura [4] showed that the Viterbi algorithm was equivalent to a dynamic programming solution to the problem of finding the shortest path through a weighted graph. In 1973, Forney [1,3] recognized that it

was in fact a maximum likelihood decoding algorithm for convolutional codes, that is, the decoder output selected is always the codeword that gives the largest value of the log-likelihood function. Often, the process of Viterbi decoding has considered in the literature as equivalent to the solving the shortest path problem in a trellis diagram. In 1989, Chang and Yao showed that the Viterbi decoding process of finding the shortest path in a trellis diagram can be formulated by employing a general matrix-vector multiplication [7], and it can be realized by a systolic array. At the same time, they also presented a strongly-connected trellis decoding method in order to increase the throughput of the decoding and to improve the efficiency of the hardware utilization.

This chapter begins with describing the convolutional encoding and the process of the Viterbi decoding for finding the shortest path in a trellis. Then, the strongly-connected trellis decoding method is described. The processes of generating the composite branch metrics and updating the path metrics, ACS computations using modulo arithmetic, and radix-2^{K-1} trellis trace-back updating in the Viterbi decoding are formulated for the purpose of hardware implementation of a Viterbi decoder.

2.2 Encoding of Convolutional Codes

A convolutional code differs from block codes in that the encoder contains memory, and the encoder outputs at any given time depend not only on the inputs at that time unit but also on the previous input blocks. A convolutional code that is defined as (n, k, m) can be generated by an encoder with a k-input, n-output linear sequential circuit including m-stage shift registers. Each information bit remains in the encoder for up to m+1 time units,

and during each time unit can affect any of the n encoder outputs depending on the m-stage shift registers connections. Generally, K=m+1 is defined as the constraint length of a code, and k/n as the code rate of a code. In addition, connections from the m-stage shift register and the current inputs to each output is represented by a sequence called generator sequence. For example, an encoder for a (2, 1, 3) convolutional code is shown in Figure 2.1. Note that the encoder consists of a 3-stage shift register together with 2 modulo-2 adders and a multiplexer for serializing the encoder output. The three generator sequences in this encoder are G(0) = (1 1 1), G(1) = (1 1 1), and G(2)=(1 1 0). G(0) represents that there are three connections to the bit-v0 output from the current input u, the first stage, and the second stage of the shift register. G(1) represents that there are three connections to the bit-v1 output from the current input u, and the first and the second stages of the shift register. G(2) represents that there are only two connections to the bit-v3 output from the current input u and the first stage of the shift register.

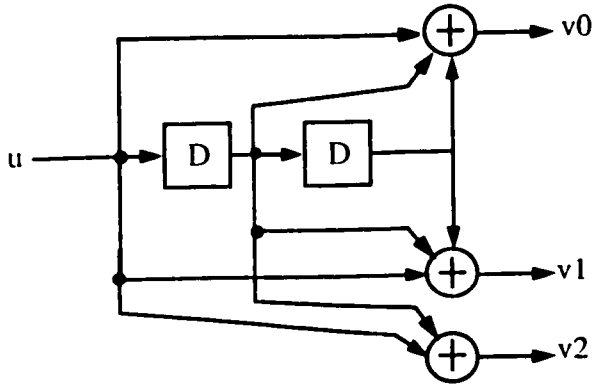


Figure 2.1 A (3, 1, 2) binary convolutional encoder

A generator sequence can be viewed as an impulse response of the encoder. Therefore, it follows that a convolutional code can be obtained by convolving the input sequence with the impulse response of the encoder. The number of stages m in the shift register and the generator sequences to be used affect the minimum Hamming distance of a code that determines the maximal number of correctable bits of a code. The use of a large constraint length K means that m has to be large. As a consequence, a convolutional code with a better error correction capability is produced.

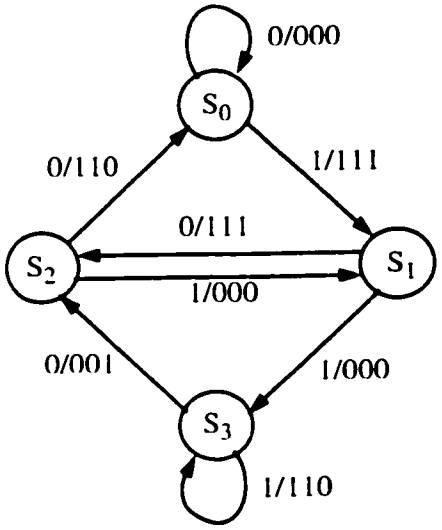


Figure 2.2 State diagram for an encoder

Since a convolutional encoder is a kind of sequential circuit, a convolutional encoding process can be described by a state diagram, in which the binary bits of its shift register represent a state of the encoder. Figure 2.2 shows a state diagram for a (3, 1, 2) code. As mentioned in Chapter 1, the state diagram of an encoder can be expanded in time, and resulting structure is called a trellis diagram. A trellis diagram for a (3, 1, 2) code with an information sequence of length L=5 is shown in Figure 2.3. In this case, the encoder

always starts from state S_0 and returns to state S_0. This is done by adding a reset sequence consisting of m bits of zeros to the information bits. In the first m (=2) time units the encoder departs from state S_0, and in the last m (=2) time units the encoder returns to state S_0. It is seen that not all the states can be reachable in the first m (=2) or the last m time units. However, in the central part of the

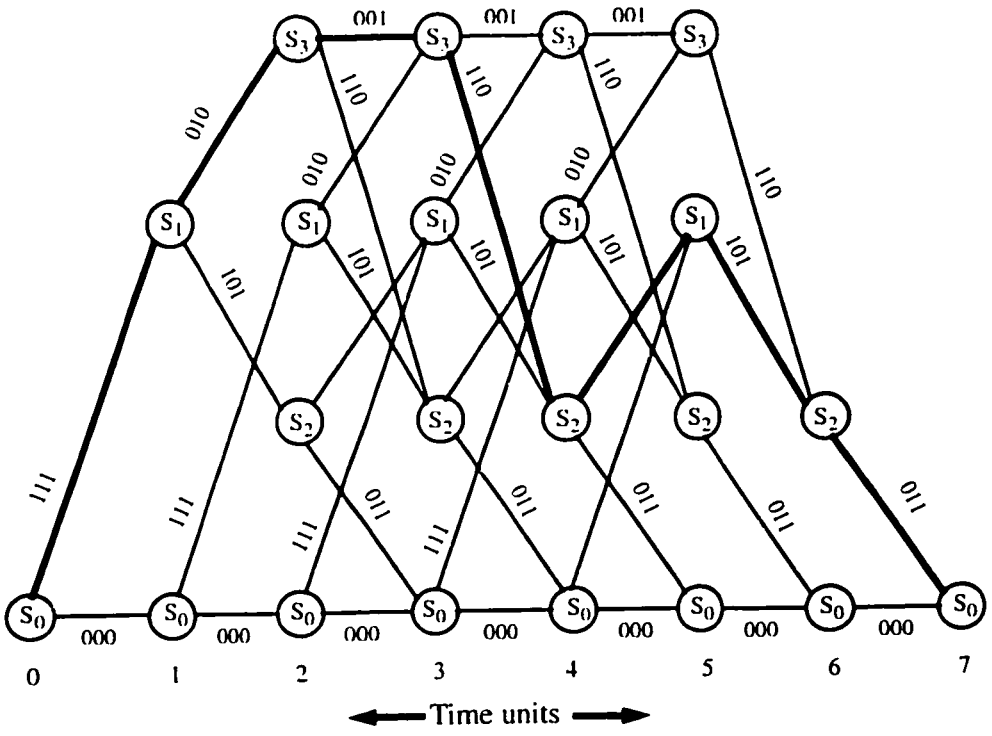


Figure 2.3 Trellis diagram for a (3, 1, 2) code with L = 5 [15]

trellis, all the states are reachable, and a trellis at each time unit represents a replica of the state diagram. There are $2^k(=2^1=2)$ branches leaving and entering each state. At each time unit (or stage), the upper branch leaving each state represents the input '1', while the lower branch represents the input '0'. Each branch is labeled with n (=3) corresponding outputs

called codeword, and each of the 2^L (= 32) codewords of length N=n(L+m)=21 corresponds to a unique path through the trellis. In the general case of an (n, k, m) code and an information sequence of length kL, there are 2^k branches leaving and entering each state, and 2^{kL} distinct paths through the trellis corresponding to the 2^{kL} code words.

2.3 Viterbi Decoding Process

The Viterbi algorithm is a maximum likelihood (ML) method for decoding of convolutional codes over an additive white Gaussian noise (AWGN) channel. Figure 2.4 shows a convolutional coded system on an AWGN channel [15]. At the transmitter end, the output of a convolutional encoder (v) is modulated into signal symbols to be transmitted through

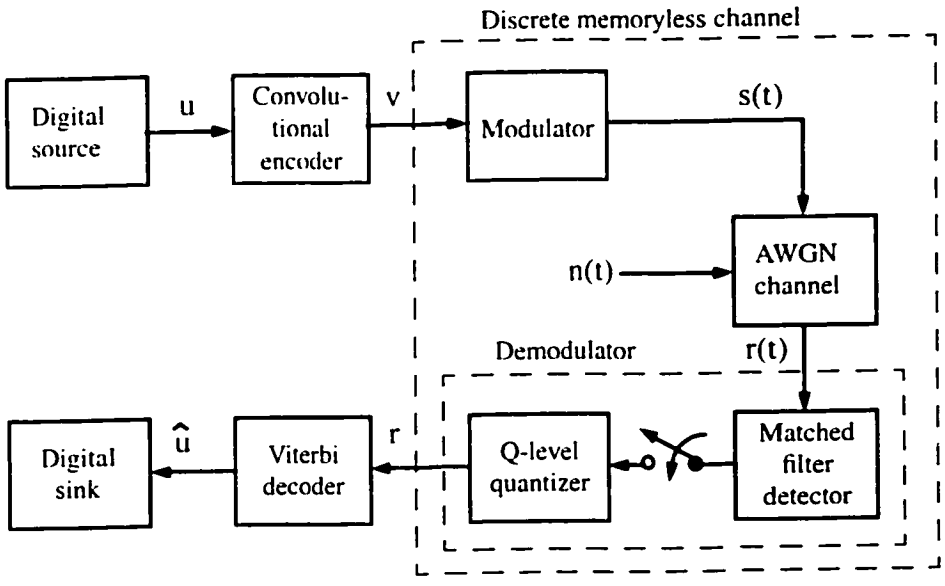


Figure 2.4 Convolutional coded system on an additive white gaussian noise channel

the AWGN channel. At the receiver end, the received signal symbols are demodulated and are quantized into several levels depending on using the hard decision or the soft decision. In the hard decision case, the received signal is quantized into two levels, either zero or

one, whereas in the soft decision case, it is quantized into more than two levels. The Viterbi decoder generates the estimate $\hat{u}$ of the information sequence u by ML method. In this method, all the codewords are assumed to be transmitted equally likely, and $P(r \mid v)$ that represents the probability of the received sequence r over the codeword sequence v to be transmitted is maximized. Further, if $\log P(r|v)$ is maximized, $P(r \mid v)$ is maximized. Thus, $\log P(r|v)$ is called log-likelihood function, and expressed as $\log P(r|v) = \sum_{i=0}^{N-1} \log P(r_i|v_i)$.

This can be interpreted with a trellis diagram. The log-likelihood function $\log P(r|v)$ is called metric associated with the path with the codeword sequence v in a trellis, whereas the term $\log P(r_i|v_i)$ is called metric corresponding to the i-th branch with codeword v_i along the path. It has been proved that if the Hamming distance or the Euclidean distance between the received sequence r and the codeword sequence v is minimized, $\log P(r|v)$ is maximized [15]. This distance can be obtained by accumulating the branch metrics that are represented by the Hamming or Euclidean distance between the received word r_i and the codeword v_i (i = 0, 1,..., N-1). Whether to compute the Hamming distance or the Euclidean distance in a ML method depends on whether hard demodulator decisions or soft demodulator decisions are made. The Hamming distance is computed with hard demodulator decisions, whereas the Euclidean distance is computed with soft demodulator decisions. Therefore, it is understandable that the ML decoding of convolutional codes can be viewed as a process of finding the shortest path in a trellis diagram.

In general, the Viterbi decoding process of finding the shortest path in a trellis can be divided into three parts: generation of the branch metrics, updating the path metrics and

making decisions for the survived paths, and the trace-back operation. Generation of the branch metrics computes the distance between two adjacent nodes (states) in a trellis diagram. Updating path metrics and making decisions for survived paths process the ACS computations at each state. At a given stage, all the path metrics entering a given state are computed by adding the branch metrics entering that state to the corresponding path metrics at the preceding stage, and then the computed path metrics of the state are compared and the shortest one selected to update the path metric of that state. At the same time, the survivor is decided corresponding to the path with the shortest distance at the state. In the trace-back operation, the decoded output sequence is produced by tracing the survivor decisions that are stored at each trellis stage. If a reset sequence is appended after each information sequence, the trace-back operation is performed from the initial state at stage m+L back to the initial state at stage 0. However, if a reset sequence is not used, the trace-back length should be at least 5K stages starting from any state of the current stage in a truncated trellis diagram.

2.4 Strongly-Connected Trellis Decoding Method

The strongly-connected trellis decoding means that the process of the Viterbi decoding is performed based on a strongly-connected trellis diagram. It has been shown in [7] that for a convolutional code of rate $1/n$ and constraint length K, groups each containing a minimum of $K-1$ stages are needed to obtain a strongly-connected trellis diagram from the original low-connectivity trellis diagram. As described in Chapter 1, a strongly-connected trellis diagram can be obtained by organizing the original low-connectivity trellis in

groups of (K-1) contiguous stages and merging the stages in each group. It means that, in a strongly-connected trellis diagram, every state in the (K-1)-th stage of the group is reachable from any state of the preceding (K-2) stages of the group. As a result, the adjacency matrix based on a strongly-connected trellis diagram will have no empty entry. As shown in Figure 1.3, in each group of (K-1) stages of the original low-connectivity trellis diagram, there is only one path between any state of the (K-1)-th stage of the group and any state of the preceding (K-2) stages of the group. This means that a strongly-connected trellis diagram contains all the possible paths corresponding to its original low-connectivity trellis diagram. Thus, when the Viterbi algorithm is used to find the shortest path in a strongly-connected trellis diagram, the feature of the maximum likelihood decoding can not be changed. This maintains the error correction capability of the low-connectivity trellis decoding in the strongly-connected trellis decoding. However, the computations in the strongly-connected trellis decoding process are somewhat different from those required in the low-connectivity trellis decoding process. In the next section, we will reformulate the conventional Viterbi algorithm based on the low-connectivity trellis decoding for strongly-connected trellis decoding.

2.5 Reformulation of the Conventional Viterbi Algorithm for the Strongly Connected Trellis Decoding

2.5.1 Generation of Composite Branch Metrics

Let K be the constraint length and r=1/n be the code rate of the given low connectivity trellis diagram. By combining (K-1) branches of the low-connectivity trellis diagram into

one single branch, we can convert it into a strongly-connected trellis diagram; the single branch in the latter is called a composite branch [7]. Let $b_q(i, j)$ represent the composite branch metric from state i of stage q-1 to state j of stage q of the strongly-connected trellis diagram, and $\boldsymbol{C_{ij}}$ a binary vector of $n(K$-$1)$ elements, representing the codeword sequence corresponding to the composite branch. Let $\boldsymbol{y_q}$ be a quantized vector of $n(K$-$1)$ elements, representing the received real-valued sequence at stage q. It has been shown in [7] that under the maximum likelihood criterion the evaluation of $b_q(i,j)$ can be expressed as

$$b_q(i, j) = f(\boldsymbol{y_q}, \boldsymbol{C_{ij}}) \tag{2.1}$$

where f is the likelihood function of $\boldsymbol{y_q}$ and $\boldsymbol{C_{ij}}$. For example, f evaluates the Euclidean distance between $\boldsymbol{C_{ij}}$ and $\boldsymbol{y_q}$ with soft demodulator decisions, and computes the Hamming distance between $\boldsymbol{C_{ij}}$ and $\boldsymbol{y_q}$ with hard demodulator decisions. In this study, binary phase shift keying (BPSK) modulation and soft demodulator decisions are assumed. Under the BPSK modulation, the minimum and maximum amplitude values -a and a can be used to respectively represent the symbol values '1' and '0' of a binary bit of $\boldsymbol{C_{ij}}$. The codeword symbol x[k] can then be expressed as

$$x[k] = \{2C_{ij}[k] - 1\}a, \quad k = 0, 1, \ldots, n(K\text{-}1)\text{-}1 \tag{2.2}$$

where $C_{ij}[k]$ represents the k th element of $\boldsymbol{C_{ij}}$. In this case, (2.1) can be written as

$$b_q(i, j) = (y_q[0] - (2C_{ij}[0] - 1)a)^2 + (y_q[1] - (2C_{ij}[1] - 1)a)^2 + \ldots$$
$$+ (y_q[n(K-1) - 1] - (2C_{ij}[n(K-1) - 1] - 1)a)^2 \tag{2.3}$$

This is equivalent to the Euclidean distance between the codeword symbols and the received symbols corresponding to the (K-1) branches in the original low-connectivity trellis diagram. Similar to the low-connectivity trellis decoding, in the strongly-connected trellis decoding, comparisons between the path metrics are required to determine the survivor path. Thus, the differences between the path metrics rather than the actual value of the path metrics are of concern. That is, one can arbitrarily add a constant to all the path metrics without changing the comparison result. This constant can be viewed as the one to be added to the composite branch metrics, since the path metrics are made up of the accumulated composite branch metrics. In (2.3), the received sequence $\boldsymbol{y_q}$ is used to generate the composite branch metrics of stage q so that each composite branch metric of stage q includes the constant $y_q[0]^2 + y_q[1]^2 +,..., + y_q[n(K\text{-}1)]^2$. Therefore, if this constant is subtracted from $b_q[i,j]$, the comparison result between the path metrics at stage q will not be changed. Furthermore, in (2.3) includes the term $[(2C_{ij}[0]\text{-}1)a]^2 + [(2C_{ij}[1]\text{-}1)a]^2 +,...,+ [(2C_{ij}[n(K\text{-}2)]\text{-}1)a]^2$, which equals to n(k-1) a^2 corresponding to $C_{ij}[k]$ = '0' or '1', k=0, 1,..., n(K-1). Likewise, the constant $n(k\text{-}1)a^2$ can be subtracted from $b_q[i,j]$ and the resulting value divided by 2a without altering the result of comparison between the path metrics of stage q. Therefore, (2.3) can be further simplified as

$$b_q[i,j] = -(2C_{ij}[0]-1)y_q[0] - (2C_{ij}[1]-1)y_q[1] - \cdots - (2C_{ij}[n(K-1)-1]-1)y_q[n(K-1)-1] \quad (2.4)$$

Thus, the computation of the Euclidean distance for a composite branch is simplified into signed additions depending on whether $C_{ij}[k]$ = '0' or '1', in the same way as presented in [9]. Note that the simplification could result in the value of the composite branch metric given by 2.4 to be positive or negative. As a result, the hardware implementation would become simple.

From (2.4), it is clear that before the composite branch metric $b_q[i,j]$ is evaluated, the codeword sequence C_{ij} should be generated. As noted in [7], a convolutional encoder can be viewed as a linear time-invariant system in which the total response can be decomposed into two parts, the zero-input response and the zero-state response. Thus, the codeword sequence $C_{ij,}$ with the initial encoder state i and the input sequence j, can be expressed as

$$C_{ij} = C_{i0} \oplus C_{0j} \tag{2.5}$$

where C_{i0} represents all the codeword sequences corresponding to $i = 0,1,...,$ $(2^{K-1}-1)$ with zero input sequence, C_{0j} represents all the codeword sequences corresponding to $j = 0,1,...,$ $(2^{K-1}-1)$ with zero encoder state, and the symbol $\oplus$ represents bitwise modulo 2 addition. Obviously, if all the codeword sequences C_{ij} with $i, j = 0,1,...,$ $(2^{K-1}-1)$ are stored, it will need a huge memory space of $N^2n(K-1)$ bits. In view of (2.5), only the minimum sets of the codeword sequences C_{i0} and C_{0j} with $i, j = 0,1,...,$ $(2^{K-1}-1)$ are required to be known and stored, and the other codeword sequences can simply be derived from these C_{i0} and C_{0j}.

2.5.2 Updating of Path Metrics

It is known that finding the shortest path in a trellis can be realized mainly by processing the ACS computation for the path metric at each state of each trellis stage. According to [7], the ACS computation corresponding to all the states of a trellis stage can be formulated into a matrix-vector computation. Given an (n, 1, m) convolutional code, the total number of states $N = 2^m$. Let $\boldsymbol{P}_q$ be a $1 \times N$ row vector, whose jth element is denoted by $p_q(j)$ representing the partial path metric from the initial state 0 to state j of stage q of the trellis diagram. Let $\boldsymbol{B}_q$ be an $N \times N$ adjacency matrix, whose ij-th element is denoted by $b_q(i, j)$ representing the branch metric from state i of stage $q-1$ to state j of stage q of the trellis diagram. Then, the viterbi algorithm can be formulated as

$$\boldsymbol{P}_q = \boldsymbol{P}_{q-1} \otimes \boldsymbol{B}_q \tag{2.6}$$

where the operator $\otimes$ denotes the ACS operation according to the likelihood criterion measured by the distance between the received data sequence and the codeword sequence. For example, the path metric at state j of stage q can be expressed as

$$\begin{aligned} p_q(j) = min\{ & p_{q-1}(0) + b_q(0, j), p_{q-1}(1) + b_q(1, j), \ldots, \\ & p_{q-1}(N-1) + b_q(N-1, j) \}, \end{aligned} \tag{2.7}$$

and the survivor at state j of stage q can be expressed as

$$\begin{aligned} Sur_q(j) = min^{-1}\{ & p_{q-1}(0) + b_q(0, j), p_{q-1}(1) + b_q(1, j), \ldots, \\ & p_{q-1}(N-1) + b_q(N-1, j) \} \end{aligned} \tag{2.8}$$

where *min* represents the operation of taking the minimum distance of the paths at state j of stage q, and min^{-1} represents the operation of finding the state number in the $(q-1)$-th stage that yields the minimum distance of the paths at state j of stage q.

Obviously, the adjacency matrix in the matrix-vector ACS computation can provide all the metrics corresponding to the branches from any state of stage $q-1$ to any state of stage q at a trellis stage. In the formulation based on the low-connectivity trellis decoding, there are only 2^k (i.e $k=1$) nonempty entries in each row or each column of the adjacency matrix, whereas its other 2^{m-k} entries are empty and do not appear in a regular form, such as a band matrix. As a result, the adjacency matrix is very sparse, and a hardware utilization of a corresponding systolic array is poor. However, by adopting the strongly-connected trellis decoding method, there is no empty entry in the adjacency matrix B_q. Therefore, the efficiency of hardware utilization can be improved and the throughput of the decoding can be increased.

2.5.3 Modulo Arithmetic for ACS

According to (2.7), the recursive path metric update at each state of a stage results in unbounded word growth due to the accumulations by a branch metric of the stage. To avoid normalization, which costs additional circuits in hardware implementation and increases the processing time, the modulo arithmetic approach [16, 17] is adopted here. It is known that for the original radix-2 trellis, the Viterbi algorithm inherently bounds the maximum dynamic range Δ_{max} of the path metric at each state as

$$\Delta_{max} \le \lambda_{max} \log_2 N \ , \tag{2.9}$$

where N is the number of states and λ_{max} is the maximum branch metric. In the Viterbi decoding process, any two updated path metrics at a state of stage q, $P_q[i,j]$ and $P_q[l,j]$, are compared using subtraction. It has been shown in [16, 17] that if $|P_q[i,j] - P_q[l,j]| < \Delta_{max}$, the comparison can be evaluated as $(P_q[i,j] - P_q[l,j]) mod 2\Delta_{max}$ without any ambiguity. Hence, the updated path metrics at a state of a stage can be computed modulo $2\Delta_{max}$. In this case, the wordlength of the path metric can be evaluated by

$$\Gamma_{bit} = \log_2 \Delta_{max} + 1 \tag{2.10}$$

The strongly-connected trellis can be viewed as a radix-$2^{(K-1)}$ trellis obtained from the original radix-2 low-connectivity trellis. In the strongly-connected trellis decoding, the dynamic range of the path metric increases, since a $(K-1)$-stage composite branch metric is added at each strongly connected stage. The wordlength of the strongly connected path metric can be deduced as

$$\Gamma_{bit} = \log_2 (\Delta_{max} + (K-1)\lambda_{max}) + 1 \tag{2.11}$$

2.5.4 Radix-2^{K-1} Trellis Trace-Back Update

The trace-back update can be viewed as a process to generate decoded outputs through tracing back the survivor decisions made according to the path metrics comparison result of each state at a stage, and stored along the trellis stages. The radix-$2^{(K-1)}$ trellis trace-

back update can be deduced starting from the trace-back update with the original radix-2 trellis [17].

Given a radix-2 trellis, let n represent the trace-back starting stage, S_n an arbitrary state of stage n, and d_n^S the survivor decision of state S_n. The state of the previous stage S_{n-1} from which the survived path at state S_n results is given by

$$S_{n-1} = d_n^S(S_n >> 1), \tag{2.12}$$

where S_n>>1 represents the right shifting the constant of S_n by one bit. Hence, the state S_{n-1} can be traced back by right shifting out the state S_n one bit at one end of a state register. Meanwhile the decision d_n^S is shifted in at the other end. If two trellis stages are merged together, an equivalent radix-4 trellis is derived. In such a case, the trace-back iterations from n-th stage to $(n-2)$-th stage can be expressed as

$$\begin{aligned} S_{n-2} &= d_{n-1}^S(S_{n-1} >> 1) \\ &= d_{n-1}^S(d_n^S(S_n >> 1) >> 1) \\ &= d_{n,n-1}^S(S_n >> 2), \end{aligned} \tag{2.13}$$

where $d_{n,n-1}$ is the composite 2-bit radix-4 decision. Similarly, if $(K-1)$ trellis stages are merged together, the strongly connected radix-$2^{(K-1)}$ trellis is obtained. In this case, the trace-back iterations from n-th to $(n-(K-1))$-th stage can be expressed as

$$S_{n-(K-1)} = d_{n,\, n-1,\ldots,n-(K-2)}(S_n >> (K-1)) \tag{2.14}$$

where $d_{n,\, n-1,\ldots,n-(K-2)}$ is the composite $(K-1)$-bit radix-$2^{(K-1)}$ decision. Obviously, instead of decoding only one bit per trace-back iteration in the original radix-2 low-connectivity trellis Viterbi decoding, $(K-1)$ bits can be decoded per trace-back iteration in the trace-

back decoding process based on the strongly connected radix-$2^{(K-1)}$ trellis. Thus, the strongly-connected trellis decoding can increase the throughput by a factor of $(K-1)$ compared to that of using the original low connectivity trellis decoding. Since any state of the radix-$2^{(K-1)}$ trellis diagram can be represented by $(K-1)$ binary bits, (2.14) shows that the bits representing a state of $[n-(K-1)]$-th stage is equivalent to the $(K-1)$-bit $d_{n,\, n-1,\ldots,n-(K-2)}$ decision.

2.6 Summary

In this chapter, the convolutional encoding, the process of Viterbi decoding, as well as the strongly-connected trellis decoding methods have been reviewed. Then, we have reformulated the conventional Viterbi algorithm, which is based on low-connectivity trellis decoding, for the case of a strongly-connected trellis decoding. The processes of generating of the composite branch metrics and updating of the path metrics, ACS computations using modulo arithmetic, and radix-2^{K-1} trellis trace-back updating in the Viterbi decoding have been formulated for the purpose of hardware implementation of a Viterbi decoder. It has been shown that a composite branch metric can be generated by evaluating the Euclidean distance between the codeword symbols and the received symbols when a soft demodulator decision is made. The evaluation for generating a composite branch metric has been simplified to process only signed additions instead of multiplications and additions under BPSK modulation. As a consequence, the complexity of the hardware implementation of the composite branch metrics can be reduced. In addition, the use of the modulo arithmetic for the ACS computations has avoided the process of normalization in

updating the path metrics, resulting in savings of the hardware and the processing time overhead. Since a strongly-connected trellis decoding increases the throughput of the decoding by a factor of (K-1) compared to that using the conventional low-connectivity trellis decoding, the process of trace-back updating of a radix-2^{K-1}strongly-connected trellis trace-back has been derived from that of radix-2 low-connectivity trellis.

CHAPTER 3

DESIGN AND IMPLEMENTATION OF A VITERBI DECODER

3.1 Introduction

In the previous chapter, we presented a formulation of the Viterbi algorithm based on the strongly-connected trellis diagram. As mentioned in Chapter 1, the systolic array architecture of the Viterbi decoder presented in [7] can be advantageous in hardware implementation due to its modularity, regularity, local and general interconnections, high degree of pipelining, and highly synchronized multiprocessing. However, the number of pipeline stages (and hence the number of array processors) in the architecture presented in [7] increases exponentially with K. For instance, for decoding a convolutional code with K=9 and r=1/2, there should be 256 processors in the corresponding 256 pipeline stages in both the systolic array for generating the composite branch metrics and the systolic array for processing the path metrics. As a result, the design will be excessively hardware-resource extensive and power consuming, and at the same time, clock-to-data skews

caused by the deep pipelining will become critical for proper functioning of the system. In addition, the processing times of the two operations, namely the composite branch metric generation and the path metric update, will differ significantly as K becomes large. For example, with K=9, the eight Euclidean or Hamming distances corresponding to the eight branches in the original low- connectivity trellis diagram should be computed and added together for generating a composite branch metric, whereas there are only three operations (ACS) in updating a path metric. Consequently, it is a challenging task to ensure that the two types of processors with different processing times can be synchronized, without slowing down the clock-rate.

In this chapter, the design and implementation of the Viterbi decoder for decoding a convolutional code with a constraint length of 9 and code rate of 1/2 using the Xilinx VirtexII-XC2V100-4FG256 chip is presented [18]. In this design, a novel systolic array architecture with arithmetic pipelining and time multiplexing is developed for updating the path metrics. In addition, a scheme for avoiding large clock-to-data skews and for providing a tolerance to low clock-to-data skews is devised to ensure that timings in the design are not violated. Moreover, the trace-back strategy and trace-back unit, the input buffer and one-stage branch metric generation, and the built-in testbench are designed to realize the complete functionality of the system. Finally, the performance analysis in terms of speed, power and FPGA resources utilization is presented based on the implementation results provided by the Xilinx implementation tool and estimation results by the Virtex-II power estimator.

3.2 Systolic Array Architecture for ACS Computations

This section presents a novel systolic array architecture for processing ACS computations for each stage, by exploiting the time multiplexing and arithmetic pipelining tecniques.

3.2.1 Adjacency Matrix Partitioning and Time Multiplexing

As shown in Chapter 2, updating of the path metrics at a given stage in the Viterbi decoding process can be formulated as a matrix-vector multiplication for ACS computation. For K=9, the matrix-vector ACS computation can be processed by the 256x256 adjacency matrix $\boldsymbol{B}_q$ and the 1x256 path metric vector $\boldsymbol{P}_{q-1}$. In order to process such a large matrix-vector ACS computation with limited hardware resources, we employed an idea that the solution of a large problem can be achieved by partitioning the problem into several smaller subproblems and then solving these sequentially. In this design, the adjacency matrix $\boldsymbol{B}_q$ is partitioned into four 256 x 64 sub-matrices, and the four submatrices along with the path metric vector $\boldsymbol{P}_{q-1}$ are used in (2.6) to update the corresponding 256 (= 4x64) path metrics of a given stage. This is achieved by the four pairs of interconnected linear systolic arrays shown in Figure 3.1. Each pair of the linear systolic arrays is formed as suggested in [7] and is shown in Figure 3.2. There are four processors in each of the top systolic array BM_4 and the bottom systolic array ACS_4. In each pair of systolic array, BM_4 is used to process the composite branch metrics given by (2.4) and (2.5), whereas ACS_4 is used to update the corresponding subset of the path metrics of a given stage according to (2.7) and make decisions on the survivor paths for the corresponding subset

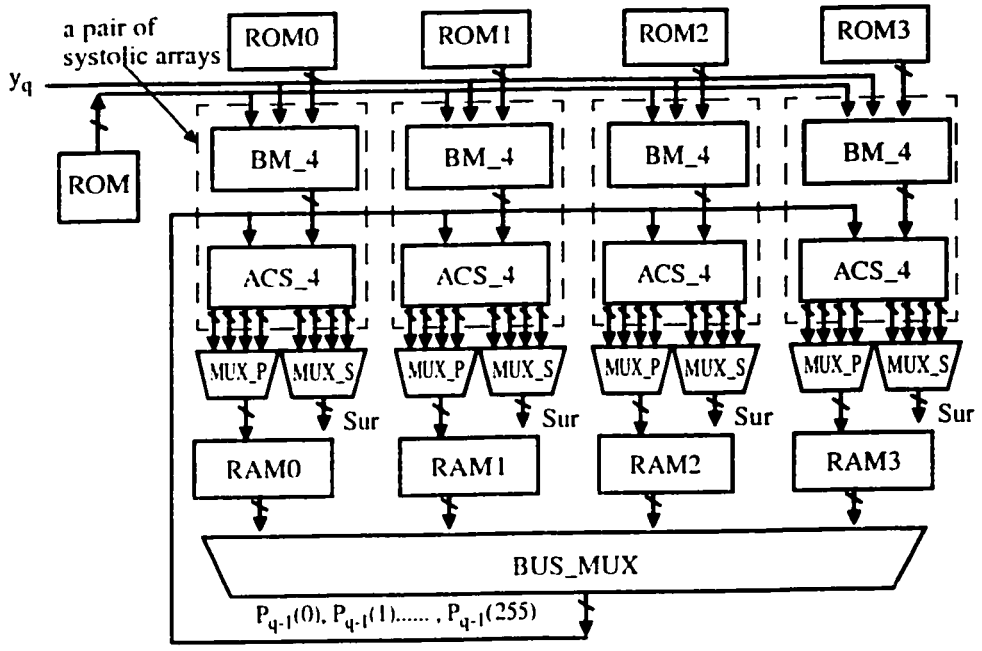


Figure 3.1 Systolic array architecture for path metrics update

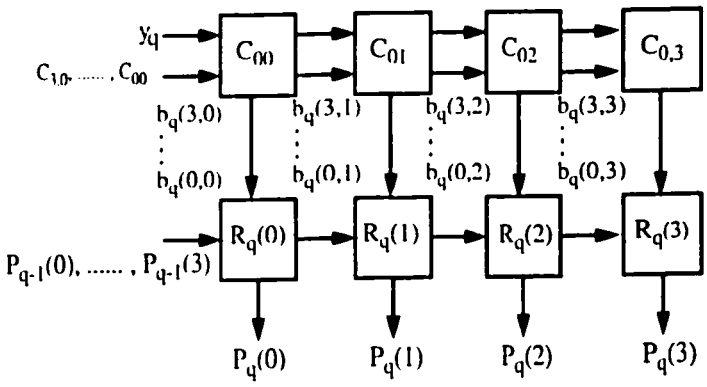


Figure 3.2 Architecture of a pair of systolic arrays

of the path metrics of the stage according to (2.8). For computing the corresponding four subsets of B_q at each stage, the codeword vector C_{0j} is divided into four 1x64 subvectors to be stored in four ROMs, ROM0 to ROM3, and the codeword vector C_{i0} stored in ROM. The updated path metric vectors from the four pairs of systolic arrays are selected by the

four multiplexers (MUX_P), and then written into the corresponding four RAMs, RAM1 to RAM3. At same time, the path metric vectors in the previous stage are read out from the four RAMs and time multiplexed to the inputs of the arrays by the multiplexer BUS_MUX. In this FPGA implementation, the ROMs and the RAMs are implemented by using the block RAMs provided by the FPGA chip. Globally, the four submatrix-vector ACS computations are carried out simultaneously by the corresponding four pairs of systolic arrays in Figure 3.1. Locally, inside each pair of the systolic arrays, the corresponding submatrix-vector ACS computation is time multiplexed. In this way, the large matrix-vector ACS computation can be implemented with four pairs of systolic arrays, each consisting of 8 processors. On the other hand, the architecture would require 512 processors, if the time multiplexing technique is not employed.

As mentioned in Section 3.1, in the systolic array architecture presented in [7], the number of pipeline stages increases with the constraint length K of a code, and for K=9, there are 256 pipeline stages. Even though it is possible to implement this large design with 256 array processors by using a large enough FPGA chip, the deep pipelining causes large clock-to-data skews, violating the timing constraint of the design. Therefore, the decoder would not function properly. On the contrary, in the proposed design, the four pairs of systolic arrays, with each array having 4 processors, are used to process the matrix-vector ACS computation, and only 4 pipeline stages are needed for each of the systolic array pairs. However, since only 4 systolic array pairs are used due to the limitation on the size of the FPGA chip, the time-multiplexing iterations are needed to process the large matrix-vector ACS computation. In general, the number of the pipeline stages does not change

with the constraint length K of a code. Even though in our implementation, the number of the time-multiplexing iterations increase exponentially with K, it has the flexibility to trade off the number of iteration with the number of the systolic array pairs. In our design of 4 systolic array pairs, there are only 16 time-multiplexing iterations.

3.2.2 Arithmetic-Pipelining Technique and Array Processors

It is seen from (2.4) that to generate a composite branch metric, one needs to process fifteen signed additions, whereas to compute a path metric only three ACS operations are needed. Usually, the global clock period has to be the higher of these two computational times, plus some safety margin so as to achieve the synchronization of the whole system. That is obviously undesirable from the point of view of speed. Fortunately, the use of a smaller clock period could be a solution not only to allow different operations in the array network take different time periods but also to speed up the design. Since one composite branch metric computation and one path metric computation consist of a number of arithmetic steps, the processors corresponding to the two types of computation can be designed with arithmetic-pipelining in which each arithmetic step is treated as a pipeline stage by adding appropriate registers. This implies that each pair of systolic arrays can be realized with a two-level pipelining, pipelining at the global array level from processor to processor and pipelining within a processor.

The arithmetic-pipelining processor for a BM_4 is shown as Figure 3.3. A composite branch metric can be viewed as the summation of the K-1=8 corresponding one-stage branch metrics in the original low-connectivity trellis decoding. Therefore, these one-

stage branch metrics can be generated before the corresponding composite branch metric is computed. In this design, the computation involved in (2.4) is divided into two parts, the

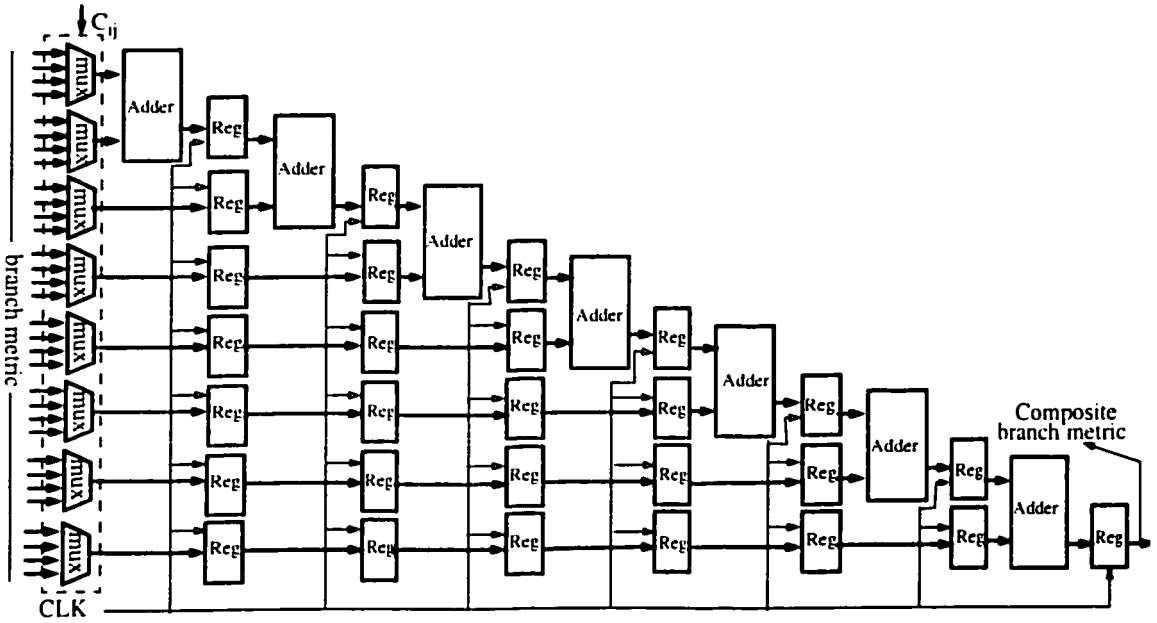


Figure 3.3 Arithmetic pipelining processor for a BM_4

computation of the eight one-stage branch metrics and the computation of the summation of those one-stage branch metrics. A module placed outside the arrays is designed to generate, by using a two-bit codeword, all the possible values for each of the eight one-stage branch metrics corresponding to a composite branch metric at a given stage. At a given stage, the array processors in a BM_4 first select, according to the codeword sequence, the eight computed one-stage branch metrics, and then perform seven signed additions to obtain the corresponding composite branch metric. Thus, the word length corresponding to the maximum composite branch metric can be 7 bits. In this way, an array processor for a BM_4 can be less complex compared to an array processor with fifteen signed additions

implemented by fifteen pipeline stages. As a result, in an FPGA implementation, the resource utilization will be reduced.

Path metric update processes the ACS computations with the composite branch metrics of the current stage and the path metrics of the previous stage. In general, a feedback is needed in path metric updating, since the path metrics of the previous stage should be fed back to the input of an ACS processor for updating the path metric at the current stage. In the butterfly architecture-based Viterbi decoders [5]-[10], ACS computations cannot be pipelined because of the feedback that would occur between the pipeline stages. However, in the proposed systolic array-based architecture, the ACS computations can be carried out in pipeline. In this design, each path metric update of the qth stage processes 256 ACS computations with 256 corresponding composite branch metrics of the qth stage and 256 path metrics of the (q-1)th stage. In view of this large number of inputs to be processed at a given stage, the proposed architecture is very suitable. Furthermore, the path metrics of the qth stage are fed back to the input of the corresponding processor at the same time as the corresponding composite branch metrics of the (q+1)th stage are generated. More importantly, since a latency of outcome exits in a pipelined design, during the 16th (final) iteration while updating the path metrics for the qth stage, the updating of the path metrics for the (q+1)th stage has already commenced. However, a two-level pipelining can ensure that there is no conflict between the updatings of the path metrics for the qth and the (q+1)th stages.

The arithmetic-pipelining processor for an ACS_4 is shown in Figure 3.4. This processor is implemented with two pipeline stages. The adder in the first pipeline stage is used to compute the 256 path metrics at any of the corresponding 16 states of a given stage,

whereas the adder in the second pipeline stage is used to make comparisons amongst the 256 path metrics computed by the first pipeline stage. According to (2.14) and (2.16), the

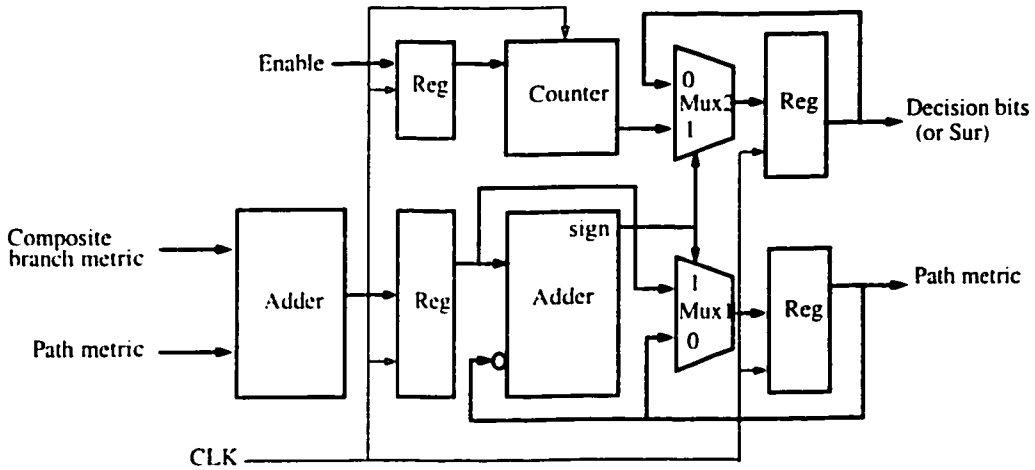


Figure 3.4. Arithmetic pipelining processor for a ACS-4

length of the adders in the two pipeline stages should be 10 bits in order to perform modulo arithmetic. In the second pipeline stage, the comparison between the path metrics computed at the current and previous clock cycles at the state of a given stage are carried out by adding the path metric at the current clock cycle to the compliment of the path metric at the previous clock cycle increased by unity. In this way, as presented in [16], the sign-bit of the adder can represent the result of the comparison between the two path metrics. If the sign-bit is '1', the path metric at the current clock cycle is smaller than the path metric at the previous clock cycle. If the sign-bit is '0', then the former is larger than the latter. In the meantime, the path metric with the shorter distance at the state of a given stage is selected by the multiplexer (Mux1) and sent to Register. In addition, the 8-bit

counter in the second pipeline stage is used to represent the 256 possible states of the previous stages from which the paths originate and terminate at various states of the present stage. One of these 256 paths would be the survivor path depending on the sign bit of the adder in the second pipeline stage, and is selected by the multiplexer (Mux2) at each clock cycle. Thus, at a given stage, the path metric and the survivor corresponding to any of the 16 states can be updated through 256 clock cycles.

If τ is the cycle time, the processing times corresponding to the computations in ACS_4 and BM_4 are 7τ and 2τ, respectively. Once an arithmetic-pipelining processor is full of data a new result is produced at every cycle instead of every 7 or 2 clock cycles, with an ordinary processor if no pipelining is involved. Therefore, employing the arithmetic-pipelinings in ACS_4 and BM_4 improves the throughput of the design. In addition, the synchronization between ACS_4 and BM_4 can be achieved by simply designing a timing scheme that makes them start at different time instants. This means that at a given stage the processors in an ACS_4 will not start their computations until the results from the processors in the corresponding BM_4 have been generated. In this way, throughput of the design will not be changed, but a latency between ACS_4 and BM_4 is introduced. This is in fact the latency of any processor in BM_4.

3.2.3 Clock-to-Data Skew and a Scheme for its Avoidance

The systolic array architecture described above provides a highly synchronized Viterbi decoder in which a source-synchronous clocking is used, and the clock frequency determines the rate of the decoding. Although the source-synchronous clocking can achieve signal integrity in the design, clock-to-data skew remains a problem to be solved. In an

FPGA implementation-based design, clock-to-data skew is caused by threshold and delay mismatch of driver output cells. edge-rate mismatch between the clock and the data output cells, and trace length mismatches between the clock and the data paths. Usually, in a

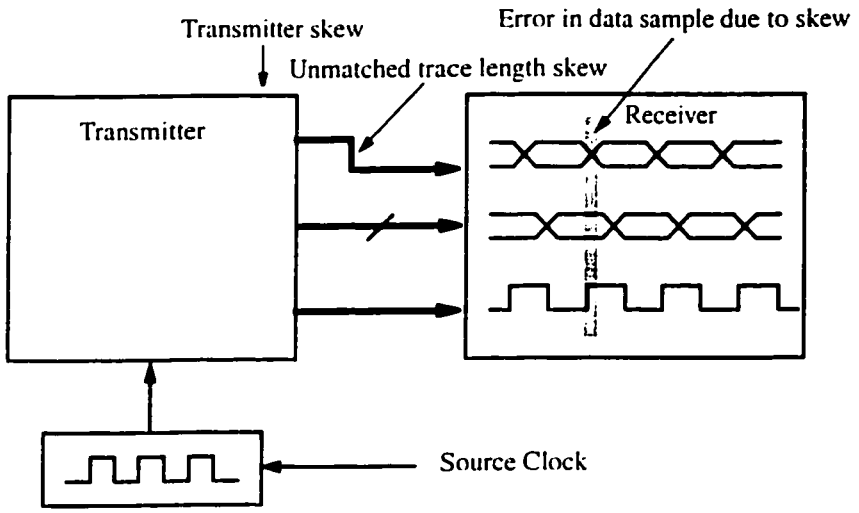


Figure 3.5 Trace length mismatches causing clock-to-data skew

synchronous design, data is transferred by the registers that are triggered by one or multiple clocks. For a register which is triggered by the rising or falling edge of the clock, data should be stable in a time phase before the rising edge or falling edge of a clock and in a time phase after the rising edge or falling edge of a clock. The former time phase is called setup time, while the latter time phase called hold time. Figure 3.5 shows an example of a trace length mismatch that causes a clock-to-data skew. It is seen that this clock-to-data skew violates the setup time so that at the rising edge of the clock, the receiver will obtain the indeterminate data. This is obviously undesirable. In this design, a scheme is proposed to avoid large clock-to-data skews and to provide a tolerance for small clock-to-data skews.

Because of the use of four pairs of systolic arrays with each array having 4 processors, instead of a pair of systolic arrays with each array having 16 processors, to process the large matrix-vector ACS computations, large clock-to-data skews have been avoided. Clock-to-data skews caused by delay mismatch between clock and data increases with the size of systolic array. The larger the systolic array, the more the pipeline stages involved, and thus the larger the clock-to-data skews generated. Therefore, clock-to-data skews caused by a delay mismatch between the clock and the data in a 16-processor systolic array can be larger than in a 4-processor systolic array. Thus, the use of four pairs of systolic arrays with each array having 4 processors is optimal in avoiding large clock-to-data skews.

A scheme for avoiding large clock-to-data skews is devised by using a 4-input look up table (LUT) provided in the FPGA device to implement the array processor for a BM_4, shown in Figure 3.3, In this array processor, there are seven pipeline stages, and each of the eight one-stage branch metrics is transferred and computed through the eight registers. If the clock skew registers of this array processor are implemented by using the conventional D flip-flops (DFFs) provided in the XC2V1000-4FG256 chip, a trace length from one DFF to another can be different depending on the placement and routing procedure in the FPGA implementation. As a result, there can be large clock-to-data skews caused by trace length mismatches between the clock and data paths in the design. However, the Xilinx VirtexII XC2V1000-4FG256 chip provides a large number of 4-input LUTs, that can be used to build a variable-tap shift register. Once the LUT-based variable-tap shift register is used to implement the clock skew registers in the array processor, a uniform trace length

from register to register through the pipeline stages can be attained. Thus, large clock-to-data skews caused by the trace length mismatches between the clock and data paths can be avoided.

On the other hand, since clock-to-data skews are inevitable in synchronous designs, there should be a scheme to provide a tolerance to the clock-to-data skews so that the timing violations caused by the clock-to-data skews can be avoided. Simulation studies have shown that once the design is implemented, the trace length mismatch between the clock and data paths is a crucial factor to cause clock-to-data skews. If each pair of the systolic arrays is triggered by a unique clock, the design for the four synchronized pipeline stages will not be tolerant to clock-to-data skews, as shown in Figure 3.6. As a result, once there

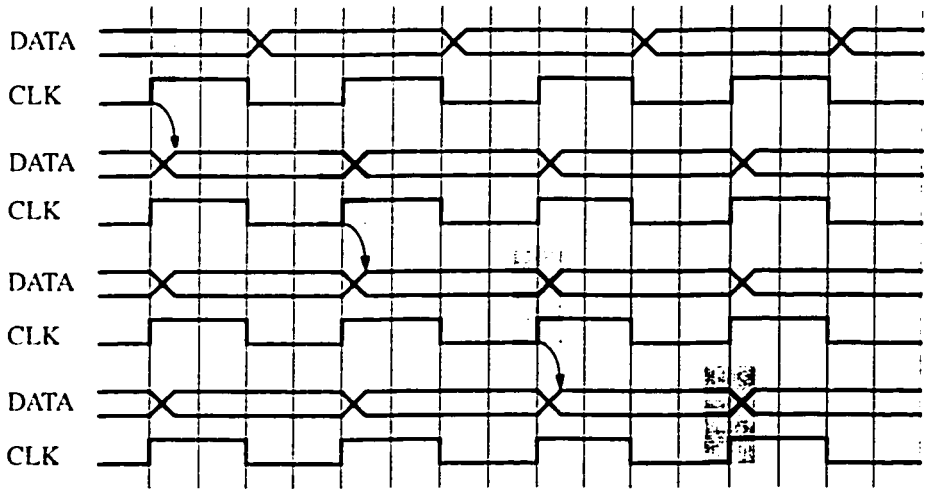


Figure 3.6 Timing diagram without clock-to-data skew tolerance

are trace length mismatches between the clock and data paths in the implementation, the setup time or hold time of the DFFs used in the pipeline register can be violated by the

clock-to-data skews, and the outputs corresponding to these mismatches between the clock and data paths at a pipeline stage will be indeterminate.

Fortunately, the digital clock manager (DCM) in VirtexII XC2V1000 provides multiple phases of the source clock, CLK0, CLK90, CLK180, and CLK270, which make it possible to exploit a timing scheme with four phase-shifted clocks to provide a tolerance to clock-to-data skews in the design. The relationship of the phase-shifted output clock to the period shift is shown in Table 3.1, and Figure 3.7 illustrates the characteristics of the

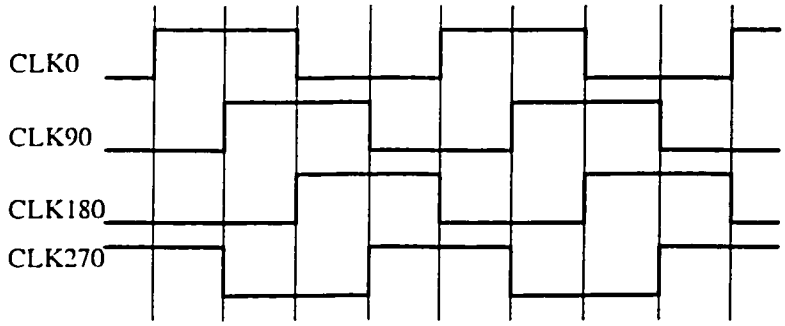


Figure 3.7 Clock outputs characteristics

Table 3.1 Relationship of phase-shifted output clock to period shift

Phase (degrees)	% Period Shift
0	0%
90	25%
180	50%
270	75%

clock outputs. The timing diagram with the four phase-shifted clocks is shown in Figure 3.8. CLK270 with the largest phase lag from the source clock is used to drive the first

pipeline stage, and CLK0 with zero phase lag is used to drive the last pipeline stage. Likewise, CLK90 is used to drive the second pipeline stage and CLK180 is used to drive the third pipe stage. Clearly, the four phase-shifted clocks can provide a 25% clock period safety margin for the data to be stable so that it can be retrieved at each pipeline stage when the clock is delayed, while they can provide almost a 75% clock period safety margin when the data is delayed. Thus, the four phase shifted clocks can compensate the clock-to-data skews at each pipeline stage, if a clock-to-data skew appears.

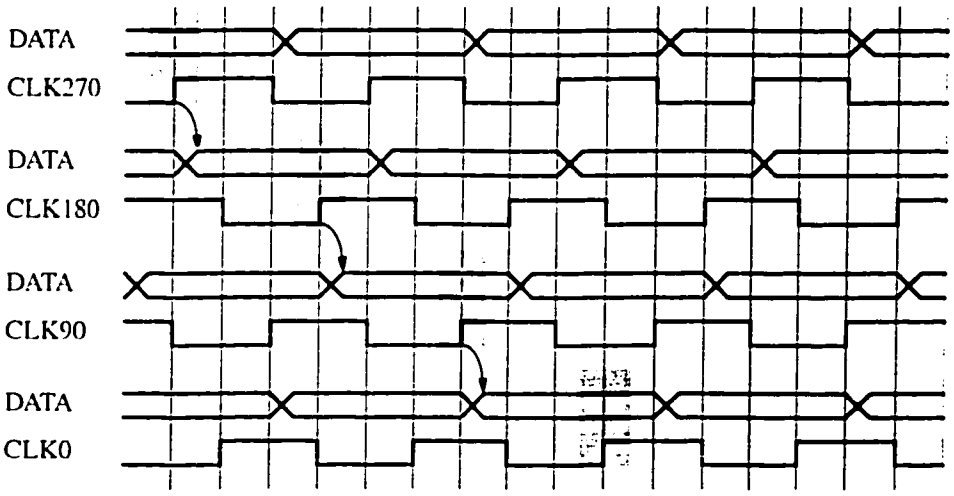


Figure 3.8 Timing diagram with clock-to-data skew tolerance

3.3. Trace-Back Strategy and Trace-Back Unit

In addition to the updating of the path metrics, described in the previous section, the trace-back updates should be carried out in the Viterbi decoder for retrieving the decoded data. In this section, a trace-back strategy used in the hardware implementation of a Vit-

erbi decoder is first introduced. Then, the trace-back unit designed for the Viterbi decoder is described.

3.3.1 Trace-Back Strategy

In the Viterbi decoding process, the decisions for the survivor paths are made and stored at each state of each stage. the decoded sequence can be retrieved from the storage of the survivor paths by trace-back operations. As explained in [1], when received data bits are very long or infinite, the survivor sequences can be truncated into a manageable length D. If the truncation depth D is chosen large enough, such as more than 5k to 6K trellis stages, all the survivors at the trellis stage k will merge to the same path with a high probability when the trace-back operation reaches the $(k\text{-}D)$-th trellis stage, and the path which those survivors merged to is the segment of the maximum-likelihood path. The decoding procedure based on this principle is called the truncated trace-back method. In this way, the decoded sequences can be retrieved with the fixed-size memory for the survivor paths.The

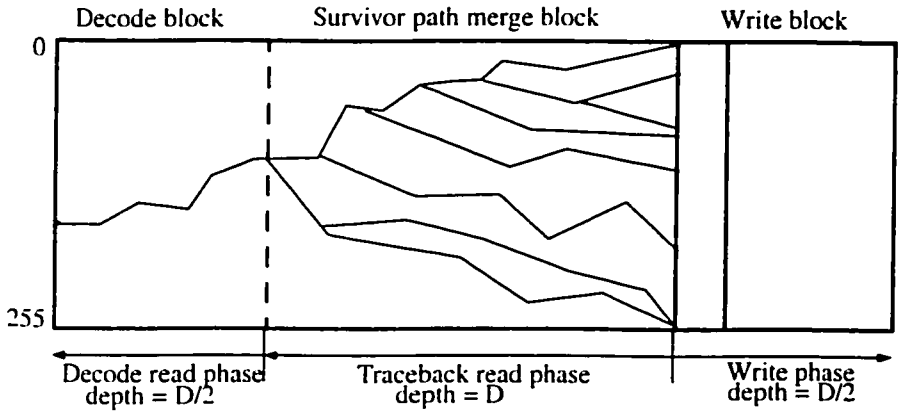


Figure 3.9 Organization of the memory for storing the survivor paths

memory for the survivor paths can be logically organized as shown in Figure 3.9. There are three operation phases in the management of the memory, Write New Data(WR), Traceback Read(TB), and Decode Read (DC). These three operations access the three logical blocks of the memory, respectively, by using 2D-circular memory addressing scheme [9],[19]. At WR phase, the survivor decisions made by the ACS computations at all the states of a given stage are written into the memory block that is just freed by the DR operation. The write pointer advances forward from stage to stage in the trellis. At TB phase, the decisions stored in the merge block of the memory are retrieved by the read pointer. It is noted from (2.14) that the retrieved data after each read operation can be treated as a pointer to indicate the corresponding state number of the previous stage. After tracing back D stages, all the survivor paths will be converged and the actual decoding takes place. At the DC phase, the decoded sequence is retrieved in the same way as the traceback read operation, and the location of the corresponding memory block is determined by the pointer value at the end of the TB phase. These retrieved data in the DC phase should then be rearranged in its original order.

The circular addressing can be realized by adopting the efficient method presented in [9] and [19] in which the traceback read and the decode read operations are performed by using one pointer instead of multiple ones, and the read operation rate should be three times that of the write operation. In this way, the two read-phases and one write- phase can be performed simultaneously within the same time period, as shown in Figure 3.10. At each traceback iteration, the decode memory block is overwritten after it has been retrieved, and the traceback operation starts from the memory block in which the new data

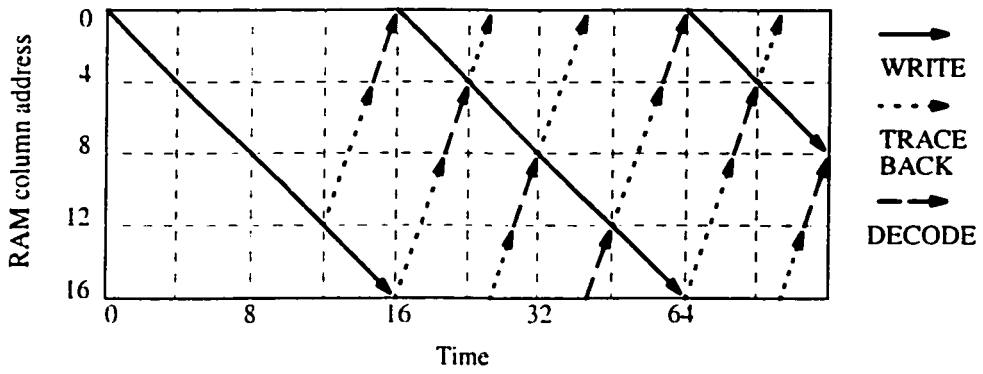


Figure 3.10 Trace-back strategy

has just been written. The logical memory block corresponding to the write operation phase is addressed circularly in the forward direction, whereas the logical memory blocks corresponding to the two read operation phases are addressed circularly in the backward direction. At the beginning of each iteration, the read pointer will meet the write pointer, and then they move towards to the opposite direction for next iteration.

In the FPGA based implementation, the memory for storing the survivor paths is divided into four memory banks, each of which stores 64 survivor paths. The four memory banks are implemented by using four block RAMs provided by the FPGA chip. The survivor path memory is organized as shown in Figure 3.11. The traceback depth of the design is chosen as $D = 8$. According to (2.14), at each strongly-connected trellis decoding, there are 8-bit binary data to be decoded in this design. Thus, the traceback depth of 8 is equivalent to the traceback depth of 8(K-1) = 64, K being the constraint length of the code in the original low connectivity trellis decoding. This is large enough to ensure the convergence

of the suvivor paths. The depth of each memory should be equal to *64 x 2D = 1024*, and its word length should be 8 bits.

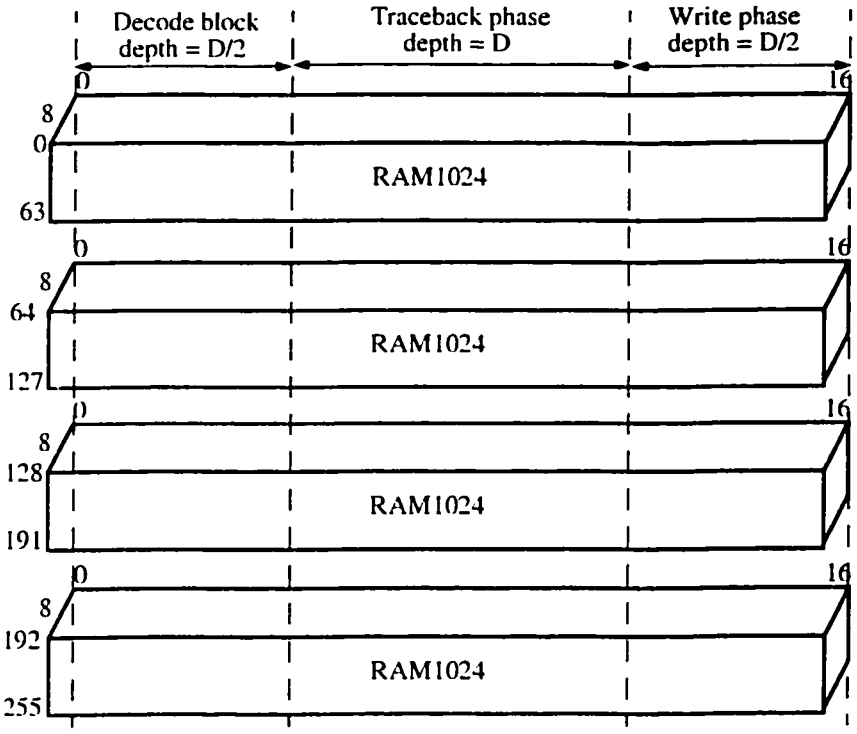


Figure 3.11 Survivor path memory structure

3.3.2 Trace-Back Unit

The architecture of the trace-back unit is shown in Figure 3.12. At each stage, the survivor decisions made by the four pairs of the systolic arrays in Figure 3.1 are written into the four corresponding memory banks, RAM1024s. At each traceback read, the previous retrieved data points to the current data to be read. The most significant two bits, s_7 and s_6, of the previous retrieved data in the register are connected to the 2/4 Decoder and BUS_MUX, are used to select the memory bank to be accessed and to have its stored data retrieved. The least significant 6 bits, s_5 to s_0, of the previous retrieved data in the register

are used to address the 64 decisions in one of the memory banks selected at each read operation. In other words, at every trace back operation, one of the 256 decisions (corresponding to the 256 survivor paths) stored in the four memory banks can be read by using

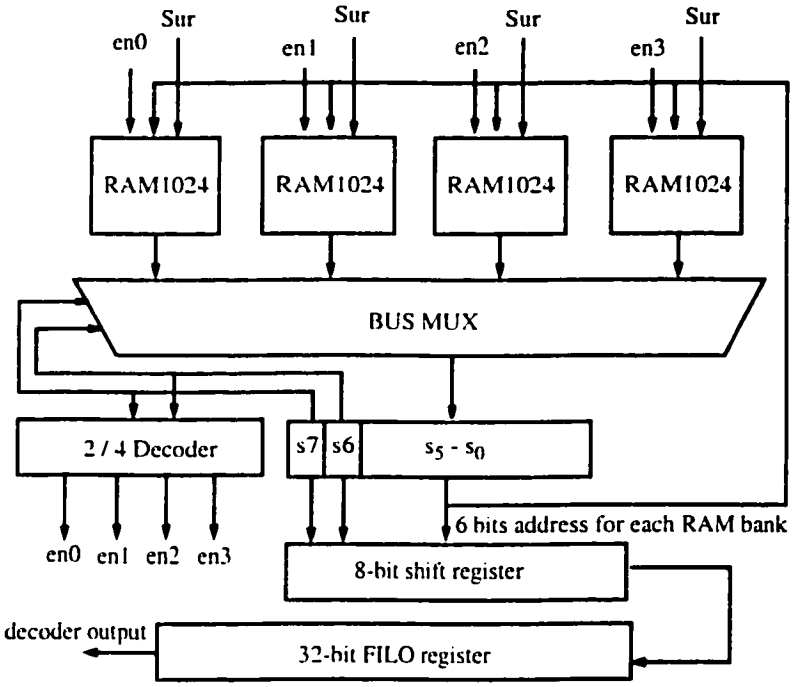


Figure 3.12. Architecture of the trace-back unit

the previous retrieved data as a pointer. After D tracebacks, corresponding to the eight strongly-connected trellis stages, the decoded data is loaded into the 8-bit shift register and then shifted to the first-in last-out register FILO. Finally, the decoded data is sent out in the reverse order from the FILO register.

3.4 Input Buffer and One-Stage Branch Metric Generation

The two inputs of the Viterbi decoder are eight-level quantized with soft demodulator decisions, and they can be represented by 3-bit binary data. The input buffer is designed to

gather the two soft input sequences corresponding to a given strongly-connected trellis stage so that the composite branch metrics of the stage can be generated. As shown in Figure 3.13, the two soft inputs are sent to the corresponding 3-bit registers depending on the

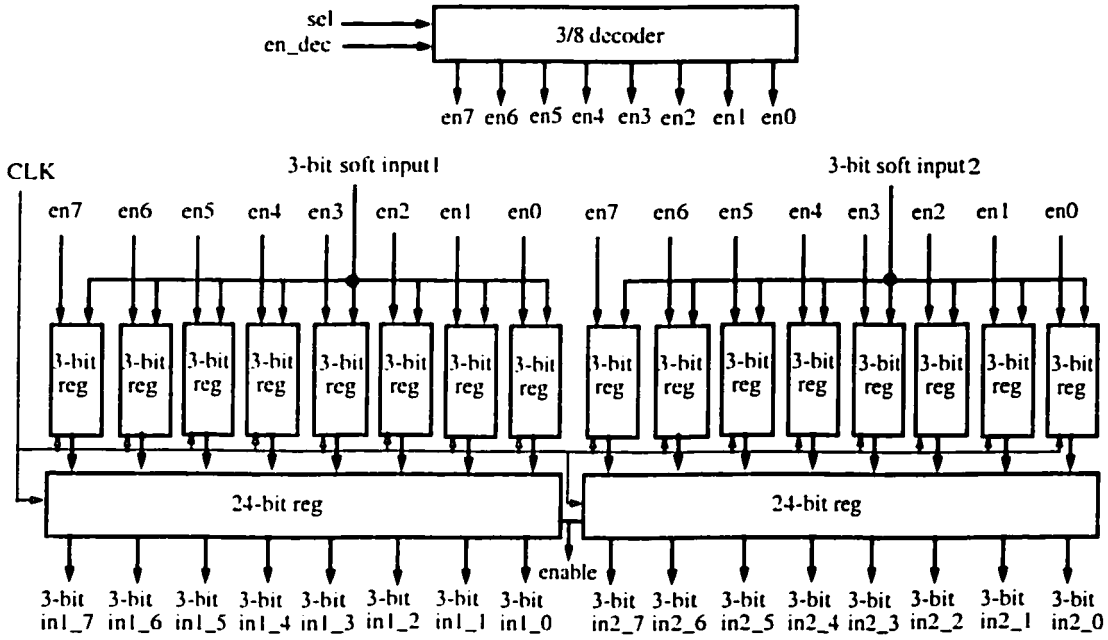


Figure 3.13. Input buffer for the two soft inputs

enable signal generated by the 3/8 decoder. The 3/8 decoder outputs the eight enable signals in a circular way so that the two soft input sequences corresponding to a strongly-connected trellis stage are loaded into the 3-bit registers. The sequences are then buffered into two 24-bit registers and used to compute the corresponding eight one-stage branch metrics.

As mentioned in Section 3.2.2, the eight one-stage branch metrics used to compute the correponding composite branch metric can be generated by a module placed outside the

array processors. Each of the eight one-stage branch metric has four possible results depending on the corresponding two-bit codeword. Table 3.2 summarizes the computation of a one-stage branch metric under the four possible values of the two-bit codeword, where *in1* and *in2* represent the two soft inputs of the low-connectivity trellis stage. The eight one-stage branch metrics corresponding to a composite branch metric are computed by the unit shown in Figure 3.14, in which each one-stage branch metric is computed under the four possible values of the two-bit codeword, and is output to the array processors in BM_4s of Figure 3.1.

Table 3.2. One-stage branch metric evaluation

Code word C_{ij}	One-stage branch metric
0 0	in1 + in2
0 1	in1 - in2
1 0	-in1 + in2
1 1	-in1 - in2

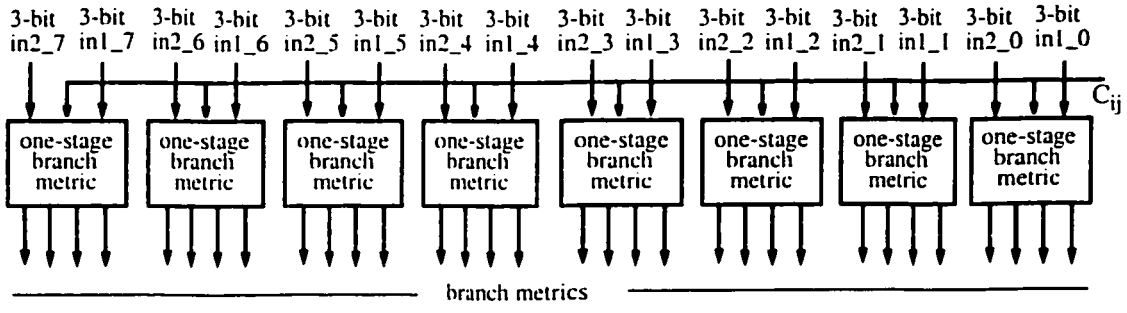


Figure 3.14 Eight one-stage branch metrics generation unit

3.5. Testability of the Design

In order to test the functionality of the Viterbi decoder offline, a testbench as shown in Figure 3.15 is built inside the system. The linear feedback shift register (LFSR) is used to generate a 32-bit random input sequence. The convolutional encoder is used to encode the random input sequence generated by the LFSR. The two soft-bit generators are used to convert the two codeword sequences generated by the encoder into the two 3-bit soft input

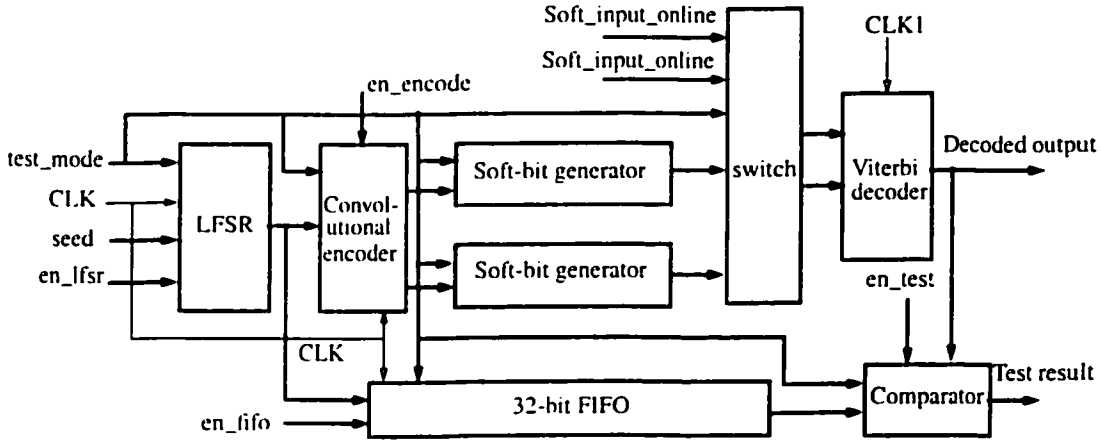


Figure 3.15. Architecture of the built-in testbench

sequences to be selected by the switch as the inputs to the Viterbi decoder. The 32-bit first-in first-out (FIFO) is used to buffer the 32-bit sequence generated by LFSR so that they can be used to make comparisons with the corresponding decoded 32 bits. In this way, the functionality of the Viterbi decoder can be tested. The system can work in either the test mode or the online mode controlled by the signal 'test_mode'. When 'test_mode' is '1', the system works in test mode. In this case, all the units used in this testbench are active, and two soft inputs from two soft-bit generators are transferred to the Viterbi decoder. On

the other hand, when 'test_mode' is '0', the system works in online mode. In this case, only the Viterbi decoder is active, and the two soft inputs from the channel are sent to the Viterbi decoder.

3.7 Performance Analysis

The design of the Viterbi decoder has been simulated, synthesized, and implemented based on the VHDL modeling (see Appendix). In this section, the performance analysis of the proposed systolic array-based Viterbi decoder in terms of speed, power, and FPGA resource utilization is carried out.

3.7.1 Speed

In general, speed of a Viterbi decoder is limited by the ACS computations in the process of updating path metrics. In this novel systolic array architecture, on one hand, the employment of time multiplexing technique slows down the speed of the decoder by 16 times, since there should be 16 time multiplexing iterations to process each submatrix-vector ACS computation at a given stage. On the other hand, the smaller clock period provided by arithmetic pipelining and the increased throughput by employing the strongly-connected trellis decoding can compensate the speed loss by time-multiplexing operations. In addition, the two-level pipelining that includes a pipelining from processor to processor and an arithmetic pipelining inside each processor, can avoid the use of an additional synchronization period to initialize the systolic arrays at the beginning of each time multiplexing iteration. Thus, a speed loss that could occur due to the initialization is avoided. The implementation result shows that the speed of the decoder is about 78 Kbps. Furthermore, this novel systolic array architecture-based design can be easily developed

further by involving more systolic array pairs to process the matrix-vector ACS computations. In this way, the number of time multiplexing iterations can be reduced, there by increasing the speed of decoding. For example, if the eight pairs of systolic arrays are used to implement the design, the speed of decoding can be doubled.

3.7.2 Power

Power consumption models for Virtex and Virtex-II series have not been published by Xilinx^{TM}. However, the power consumption model for the XC4000XL/EX/E FPGAs can be found in [20], and it has been shown in [21] that the power consumption for Virtex FPGAs can be estimated based on the same model as the one for the XC4000 series by assigning an appropriate technology-dependent parameter. Since Virtex-II series provides a significant structural similarity to Virtex series, the performance of the design, namely the conventional Viterbi decoder, can be analyzed based on the model presented in [20]. According to this model, the total power consumption (P_{TOT}) is the sum of three components, as given by

$$P_{TOT} = P_{STAT} + P_{INT} + P_{IO}, \tag{3.1}$$

where P_{STAT} is the static power consumption resulting from leakage current by an inactive device connected to the power supply, P_{INT} is the internal power dissipation caused by the charging and discharging the capacitance on any internal nodes that are switched, and P_{IO} is the input and output power dissipation resulting from the charging and discharging of the external load capacitors connected to device pins, and the pull-ups used on the inputs.

The internal power dissipation P_{INT} may be approximated as given by [20]

$$P_{INT} = V_{CC} \times K_P \times F_{MAX} \times N_{LC} \times Tog_{LC}, \quad (3.2)$$

where Vcc is the supply voltage (V), N_{LC} the number of the logic cells used by the application, F_{MAX} the clock frequency (Hz), Tog_{LC} the average toggling rate at each clock, and K_P a constant whose value (in Coulomb) depends on the family of FPGA.

The output power dissipation P_{OUT} depends on the capacitive load on each output as well as the frequency at which each output switches, and it can be evaluated as [20]

$$P_{OUT} = 0.5 \times C_{OUTavg} \times F_{MAX} \times Tog_{OUT} \times N_{OUT} \times V^2_{swing}, \quad (3.3)$$

where C_{OUTavg} is the average load capacitance (F), F_{MAX} the maximum clock frequency (Hz), Tog_{OUT} the average toggling rate of the output at each clock, N_{OUT} the number of outputs in the design, and V_{swing} the output swing voltage (V).

It is seen that the average toggle rate at each clock, which represents as to how often the switching activities are generated in the internal or external nodes of the system, can affect the dynamic power consumption.

The power consumption of the design is evaluated using the tool Virtex-II Power Estimator provided by Xilinx™. This tool allows the estimation the power consumed by Configurable Logic Blocks (LUT, DFF and CLB configured as RAM cell), embedded RAM cells, PLLs blocks and I/O cells. In this power estimation, all the data corresponding to the utilization of the FPGA resource are based on the implementation results provided by the Xilinx implementation tool. The average toggle rate at each clock for a module is obtained

by the simulation results and by referring to the data provided by Xilinx Virtex-II Power Estimator.

Table 3.3 Device quiescent power estimator result

V_{CCint} Subtotal (mW)	V_{CCaux} Subtotal (mW)
150	33

Table 3.4 CLB logic power estimator results

Module	Frequency (MHz)	CLB Slices	Flip/Flop or Latches	LUT		Average Toggle Rate %	Routing Amount	V_{CCint} Subtotal (mW)
				Shift Register	SelectRAM			
acsPEx16	40	448	608	0	0	50%	High	114
acsCNTx16	40	64	128	0	0	25%	Medium	6
brcPEx16	40	1488	832	480	0	50%	High	417
brcREGx16	40	32	64	0	0	55%	High	9
testPKG	40	48	56	2	0	50%	High	12
conLOGIC	40	206	165	0	0	50%	High	51
recBUF	40	220	256	0	0	12%	High	16
decOUT	40	20	40	0	0	12%	Medium	1
MUX_Ps & MUX_Ss	40	16	0	0	0	2%	High	0
							Total	629

Table 3.3 shows the results provided by the Power Estimatior regarding the static power consumed in this design. Table 3.4 shows the corresponding results for the CLB logic used in this design. It is seen that the average toggle rate at each clock for the multiplexers, MUX_Ps and MUX_Ss, used in the systolic array architecture shown in Figure 3.1, are

only about 2%. The dynamic power dissipation in these multiplexers are ignored by the tool. This is because these multiplexers are active under the enable signal generated only at the end of each time-multiplexing iteration. In this way, the intermediate results from

Table 3.5 Block selectRAM power estimator results

Module	Block RAM Cells	Port A				Port B				V_{CCint} Subtotal (mW)
		Freq. (MHz)	Width	Read Rate %	Write Rate %	Freq. (MHz)	Width	Read Rate %	Write Rate %	
ROM256	1	40	18	100%	0%	0	0	0%	0%	2
ROM64_0	1	40	18	2%	0%	0	0	0%	0%	0
ROM64_1	1	40	18	2%	0%	0	0	0%	0%	0
ROM64_2	1	40	18	2%	0%	0	0	0%	0%	0
ROM64_3	1	40	18	2%	0%	0	0	0%	0%	0
RAM128_0	1	40	18	0%	2%	40	18	100%	0%	2
RAM128_1	1	40	18	0%	2%	40	18	100%	0%	2
RAM128_2	1	40	18	0%	2%	40	18	100%	0%	2
RAM128_3	1	40	18	0%	2%	40	18	100%	0%	2
RAM1024_0	1	40	9	0%	2%	40	9	6%	0%	0
RAM1024_1	1	40	9	0%	2%	40	9	6%	0%	0
RAM1024_2	1	40	9	0%	2%	40	9	6%	0%	0
RAM1024_3	1	40	9	0%	2%	40	9	6%	0%	0
									Total	15

array processors cannot propagate through these multiplexers to generate switching activities, and thus the power dissipation is negligible. Table 3.5 shows the results given by the Power Estimator in block RAMs used in this decoder. Since the time-multiplexing computation results in a very low read rate on the four 64-depth ROMs and a very low write rate on the eight RAMs used in this decoder, there is little amount of the power dissipated in these memory units. In addition, Table 3.6 and Table 3.7 show the estimator results from

the DCM and the I/O cells. It is seen that the amount of power dissipation in the I/O cells and the DCM are only about 2 mW and 1 mW, respectively. Finally, the total power estimator results is shown in Table 3.8. The total power consumption is 830 mW, including 2 mW power consumption from I/O supply voltage V_{CCO}, 33 mW power consumption from

Table 3.6 DCM power estimator result

Module	Clock Input Frequency (MHz)	DCM Frequency Mode	V_{CCint} subtotal (mW)
DCM	40	Low	1

Table 3.7 Input/output power estimator result

Module	Frequency (MHz)	I/O Standard Type	Inputs	Outputs	Average Output Toggle Rate (%)	Average Output Load (pF)	V_{CCint} Subtotal (mW)	V_{CCO} Subtotal (mW)
allPKG	40	LVTTL_12	4	3	50%	50	0	2

Table 3.8 Power estimator results

Target Device		Estimated Design Power Values (mW)			
Device	Package	Total Power	V_{CCint} 1.5V	V_{CCaux} 3.3V	V_{CCO} 3.3V
XC2V1000	FG256	830	795	33	2

auxiliary circuits whose supply voltage is V_{CCaux}, and 795 mW power consumption from the supply voltage V_{CCint}. The power consumption from the core supply voltage V_{CCint} includes a static power consumption of 150 mW and a dynamic power consumption of

645 mW. Clearly, the dynamic power consumption is dominant in the design and it mainly results from the array processors. From Table 3.4, it is seen that the array processors denoted by acsPEx16 and brcPEx16 are mainly power consuming modules in this design, since they utilize a relatively large amount of FPGA hardware resource, and the switching activities in this decoder occur mainly in these modules. In order to reduce the amount of switching activities in the array processors and, therefore, to reduce the power consumption, an adaptive Viterbi algorithm and a low-power adaptive Viterbi decoder using this algorithm are presented in Chapters 4 and 5.

3.7.3 FPGA Resource Utilization

According to the implementation results provided by the Xilinx implementation tool, this design takes 2793 slices that is 54% of the total FPGA slices, 17 blockRAMs and one DCM. From Table 3.7.2, it is seen that the array processors used to generate composite branch metrics (brcPEX16) consume the largest part of the FPGA resources although these array processor have been simplified by processing the eight one-stage branch metrics for a composite branch metric using a module placed outside these array processors. This constitutes a drawback of the present design.

3.8 Summary

In this chapter, the design and an FPGA implementation of the Viterbi decoder with a constraint length of 9 and code rate of 1/2 has been presented. In this design, a novel systolic array-based architecture with time multiplexing and arithmetic pipelining has been

exploited. A large 256x256 matrix-vector ACS computation is partitioned into 4 submatrix-vector ACS computations, and they are processed by four pairs of systolic arrays with 4x2x4 processors rather than by one pair of systolic arrays with 1x2x256 processors. Globally, the four pairs of systolic arrays process the four submatrix-vector ACS computations simultaneously, whereas from the local point of view, each pair of systolic arrays processes the corresponding submatrix-vector ACS computation in time multiplexing. Partitioning the large systolic array pair into four pairs of systolic arrays provides benefits of minimizing clock-to-data skews as well as latencies of pipelinings. It also makes it possible to drive each systolic array pair by using the four-phase-shifted clocks provided by the DCM in XC2V1000-FG256. This provides a tolerance to low clock-to-data skews so that the timing violations can be avoided within a certain safety margin. The employment of time multiplexing technique has greatly reduced the FPGA resources utilization in the implementation of the design with a fixed-size FPGA chip. From a point view of the power, time multiplexing computation can provide very low toggle rate on the memorys and the multiplexers used in this architecture resulting in saving the dynamic power dissipation. In addition, the two-level pipelining has been exploited with the pipelining from processor to processor and the arithmetic pipelining inside each processor. In this way, the two types of array processors with different processing times are synchronized, and throughput of the design has been increased due to the use of a smaller clock period. From a point of view of the speed, the proposed systolic array architecture can easily be further developed by involving more systolic array pairs to process the matrix-vector ACS computation so that a smaller number of time-multiplexing iterations is needed. As a result,

the speed of the decoder can be increased. In the trace-back unit, the truncated trace-back strategy and 2D-circular memory addressing scheme has been employed to retrieve the decoded sequence from the survivor path memory. In order to achieve high-speed trace-back operation, the simple one read pointer approach is adopted instead of using different memory read and write access time or multiple read pointers, In this way, the access rate of read operation should be three times higher than that of write operation so that the three read operation and one write operation can be done at the same time period. Also, the input buffer and one-stage branch metric generation has been designed corresponding to the strongly-connected trellis decoding. A testbench has been built in this system and used to demonstrate the successful functionality of the decoder implemented.

It is noted that the array processors used for generating the composite branch metrics and updating the path metrics consume the largest amount of power. This is because these array processors take relatively a large amount of FPGA hardware resource, and switching activities in the decoding process are mainly generated by the ACS computations processed by them. In the next chapter, in order to reduce the amount of ACS computations in the strongly-connected trellis decoding process, an adaptive Viterbi algorithm is proposed. Using this adaptive algorithm, a low-power adaptive Viterbi decoder with reduced switching activities in its array processors is then proposed in Chapter 5. In view of this adaptive Viterbi algorithm and the corresponding adaptive decoder to be presented in Chapters 4 and 5, respectively, the reformulated Viterbi algorithm of Chapter 2 and the Viterbi decoder presented in this chapter are henceforth referred to as a non-adaptive Viterbi algorithm and a non-adaptive Viterbi decoder, respectively.

CHAPTER 4

AN ADAPTIVE VITERBI DECODING ALGORITHM

4.1 Introduction

As described in Chapter 2, a Viterbi algorithm [7] can be formulated employing a matrix-vector multiplication, whose hardware implementation can be realized using a systolic array architecture. Since the low-connectivity trellis decoding results in the adjacency matrix to become very sparse, the efficiency of the hardware utilization is rather poor for this formulation. In order to resolve this problem, a strongly-connected trellis-based Viterbi decoding method was also proposed in [7] for improving the utilization of the systolic array processors as well as for increasing the throughput of the decoding. A strongly-connected trellis is obtained by merging the stages in each group of the (K-1) contiguous stages of the original low-connectivity trellis so that every state in the (K-1)-th stage of the group is reachable from the states of the preceding (K-2) stages, where K is the constraint length. This, however, makes the amount of ACS computations in the strongly connected Viterbi decoding process to become very excessive. It is known that, in a Viterbi algo-

rithm, at each state, all possible distinct paths in the trellis are evaluated and the most likely one with the shortest Hamming or Euclidean distance is determined. Consequently, for the strongly-connected trellis Viterbi decoding, the amount of ACS computations can increase exponentially as the constraint length K increases. For instance, in the Viterbi decoder, for K= 9 and r=1/2, the path metric update at each state has 256 ACS computations. Since for a trellis with K=9, there are 256 states in a strongly-connected trellis, a total of 256 x 256 ACS computations for each stage are required. This contrasts with a total of 2 x 256 x 8 ACS computations for each group of 8 stages in a two-ACS per state computation of the corresponding low-connectivity trellis decoding. Thus, a major drawback of the strongly-connected trellis Viterbi decoding is its increased computational complexity, in that its number of ACS computations per stage is 16 times higher than that in the corresponding low-connectivity counterpart. In order to reduce the amount of computations in the Viterbi decoding process, an adaptive Viterbi algorithm was proposed by Chan and Haccoun [22]. This algorithm is a modified version of the conventional Viterbi algorithm and is based on a low-connectivity trellis decoding. However, it requires a time-consuming sorting operation to determine the most likely survivor paths among all the possible survivor paths.

In this chapter, an adaptive Viterbi algorithm based on a strongly-connected trellis decoding is presented [23]. The algorithm does not require a sorting operation to determine the most likely survivors and is very suitable for systolic array-based hardware implementation. A simulation study is carried out on the implementation of the proposed algorithm based on a systolic array architecture. The simulation results show that the pro-

posed adaptive Viterbi algorithm can reduce the number of ACS computations significantly while maintaining almost the same error performance as that of the non-adaptive Viterbi algorithm.

4.2 Formulation of the Proposed Adaptive Viterbi Algorithm

In this section, the adaptive Viterbi algorithm is reformulated as presented in [22] so that it is suitable for a systolic array-based hardware implementation of the algorithm. The most important modification introduced in the proposed algorithm is the employment of strongly-connected trellis decoding technique. This is done in order to achieve an increased utilization of hardware resources offered by the systolic array architecture and a higher throughput of the decoding. A bi-product of this change is in the elimination of the sorting operation needed in the low-connectivity technique of [22]. Similar to the Viterbi decoding process, the proposed adaptive Viterbi algorithm can be viewed as a process of finding the shortest path in a strongly-connected trellis diagram. This process includes mainly the generation of the composite branch metrics, updating of the path metrics, and the trace-back operation. The difference between the formulations of the Viterbi algorithm and the proposed algorithm is only in the updating of path metrics, all other parts for generation of composite branch metrics, the modulo arithmetic for ACS, and the radix-2^{K-1} trace-back update remaining the same.

In the adaptive Viterbi algorithm of [22], instead of keeping all the possible distinct paths in each trellis stage, only a number of the most likely paths are kept and all the others discarded. As a result, only the survivor paths are involved in the evaluations in each

trellis stage, and thus the amount of computations is reduced. The selection of the survivor paths at the trellis stage q is done by chosing for each state a path with the minimum metric distance and comparing it with $(d_m + T)$, where T is the discarding threshold value selected by the user, and d_m the minimum distance of all the survivor paths at the trellis stage $(q - 1)$. Assuming a convolutional code with the constraint length of K and a code rate of 1/n, in its strongly-connected trellis diagram there will be 2^{K-1} number of states at each stage. Further, at each state of a stage, 2^{K-1} branches enter and the same number of branches leave. For the q-th strongly-connected trellis stage, let $P_q(j)$ denote the survivor path metric and $Sur_q(j)$ the survivor state with the smallest path distance, where $j = 0, 1, \ldots, 2^{K-1}-1$ denote the different states. We denote by $b_q(i, j)$ the q-th stage composite branch metric [7] from state i of stage $(q-1)$ to state j of stage q. Also, let d_m^q represent the minimum distance of all the survivor paths, and $S_q(j)$ represent the survivor information at the q-th stage. The adaptive Viterbi algorithm can be formulated as follows.

At the start, the path metric $P_0(0)$, the survivor state $Sur_0(0)$, and the survivor information $S_0(j)$ are initialized as

$$P_0(0) = 0 \tag{4.1}$$

$$Sur_0(0) = 0 \tag{4.2}$$

$$S_0(j) = \begin{cases} 1 & for\ j = 0 \\ 0 & otherwise\ . \end{cases} \tag{4.3}$$

Let $j_0, j_1, \ldots, j_L$ represent the L survived states at the $(q-1)$ th stage. Then, $S_{q-1}(j)$ and d_m^{q-1} can be expressed as

$$S_{q-1}(j) = \begin{cases} 1 & for\ j = j_0, j_1, \ldots, j_L \\ 0 & otherwise \end{cases} \tag{4.4}$$

$$d_m^{q-1} = min\{P_{q-1}(0), P_{q-1}(1), \ldots, P_{q-1}(2^{K-1}-1)\}. \tag{4.5}$$

The survivor information $S_{q-1}(j)$ can be combined into the path metric value $P_{q-1}(j)$ and treated as a flag bit indicating the survivor of the j-th state in the (q-1)-th stage. We can now find $P_q(j)$ and $Sur_q(j)$ as follows:

$$P_q(j) = min\{P_{q-1}(j_0) + b_q(j_0, j), \ldots, P_{q-1}(j_L) + b_q(j_L, j)\}, \tag{4.6}$$

$$Sur_q(j) = min^{-1}\{P_{q-1}(j_0) + b_q(j_0, j), \ldots, P_{q-1}(j_L) + b_q(j_L, j)\}, \tag{4.7}$$

where j represents 2^{K-1} states of the q-th stage, *min* represents the operation of taking the minimum distance of the survivor paths at state j (of stage q), and min^{-1} represents the operation of finding the state number in the $(q-1)$-th stage that yields the minimum distance of the survivor paths at state j of stage q.

In general, just as the Viterbi algorithm can be formulated as a matrix-vector computation presented in Chapter 2, the path metric update here can be expressed as

$$\boldsymbol{P}_q = \boldsymbol{P}_{q-1} \otimes \boldsymbol{B}_q, \tag{4.8}$$

where $\boldsymbol{P}_q$ represents the path metric row vector of the q-th stage with 2^{K-1} elements, whose j-th element is denoted by $P_q(j)$, $\boldsymbol{B}_q$ represents 2^{K-1} x 2^{K-1} adjacency matrix, whose ij-th element is denoted by $b_q(i, j)$ representing the composite branch metric from state i of stage q-1 to state j of stage q, and $\otimes$ represents the ACS operation according to the likelihood criterion.

It is clear that (4.6) can be viewed as an example for path metric update at state j in (4.8), and it also shows that the ACS computations of the 2^{K-1} incoming paths at each state are performed selectively by the survivor information of the (q -1)-th stage. This means that only if the survivor information of the (q-1)-th stage is '1', the ACS computations at q-th stage are performed. The present survivor information $s_q(j)$ at each state can be determined by comparing $P_q(j)$ with $(d_m^{q-1} + T)$. If $P_q(j) < (d_m^{q-1} + T)$, then the q-th stage survivor information $s_q(j)$ is set as '1', otherwise as '0'. As pointed out in Chapter 2, the Euclidean distance for computing a composite branch metric has been simplified into signed additions in the same way as presented in [9]. Therefore, the composite branch metrics and hence, the path metrics could have negative values. Consequently, the minimum distance d_m at a given stage could also be negative. In this case, the most optimum value for the threshold value T should be negative.

The selection of the survivor paths and the survivor paths update can be repeated by using the procedure described above at each strongly-connected trellis stage. Finally, the decoded data can be obtained through a traceback of the survivor path.

From the description above, it is seen that the adaptive Viterbi algorithm based on a strongly-connected trellis diagram can be viewed as processing a matrix-vector ACS computation in which the survivor information at each state is combined into the state path metric value and treated as a flag bit. Its hardware implementation can be realized by using the systolic array architecture that will be discussed in the next chapter.

4.3 Comparisons with The Non-Adaptive Viterbi Algorithm

The performance of the proposed adaptive Viterbi algorithm based on the strongly-connected trellis decoding in the presence of additive white Gaussian noise is simulated using the C-language (see Appendix). This simulation study is carried out for decoding a convolutional code with a constraint length K = 9 and a code rate of 1/2, in which a BPSK modulation scheme and a 3-bit soft-decision are used. The computational precision is selected to conform with the requirements of the hardware computing units.

Figure 4.1 shows the error performance for both the convolutional coded binary data using the proposed algorithm and uncoded binary data. The error performance is expressed by the bit error rate (BER) as a function of E_b/N_o, where E_b denotes the signal energy and N_o the noise density. It can be seen from this figure that the proposed adaptive Viterbi algorithm based on the strongly-connected trellis decoding can provide at least a 5 dB coding gain at a bit-error-rate of 10^{-5}, when the threshold value T is chosen to be greater than -8.

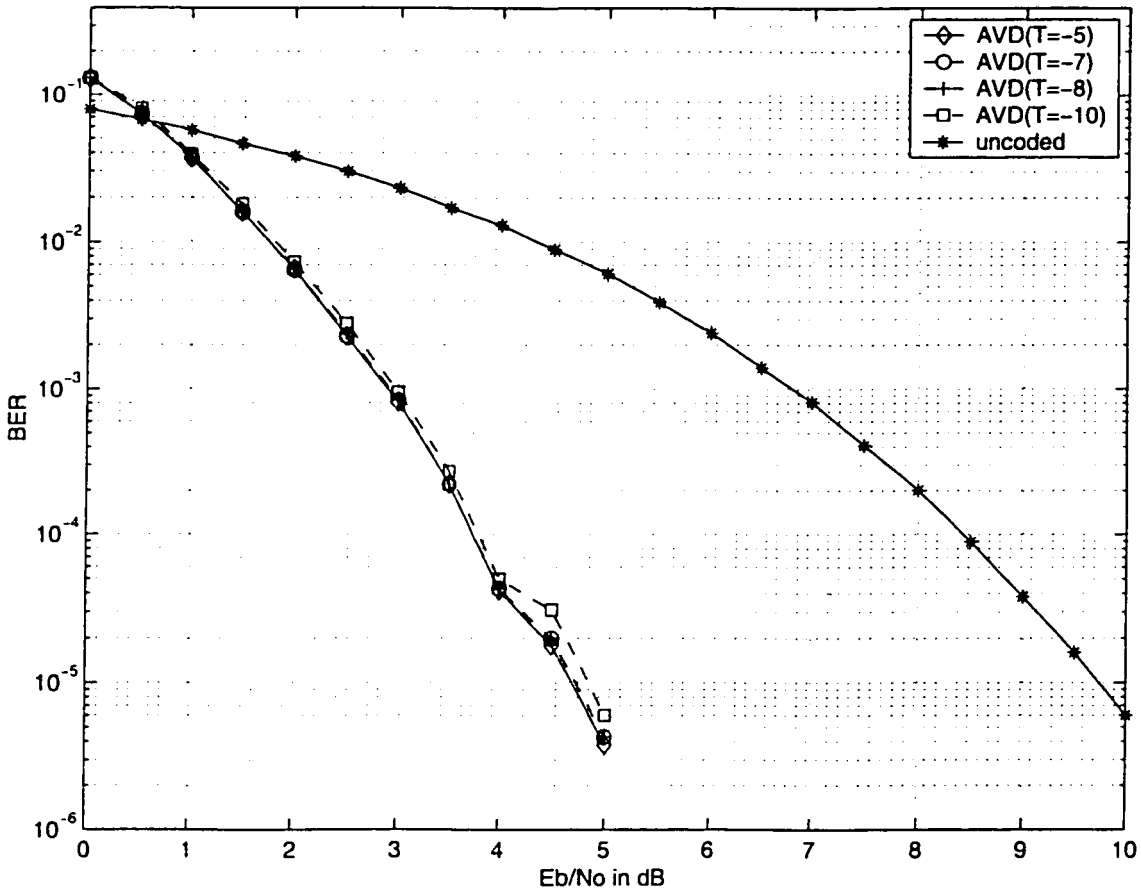
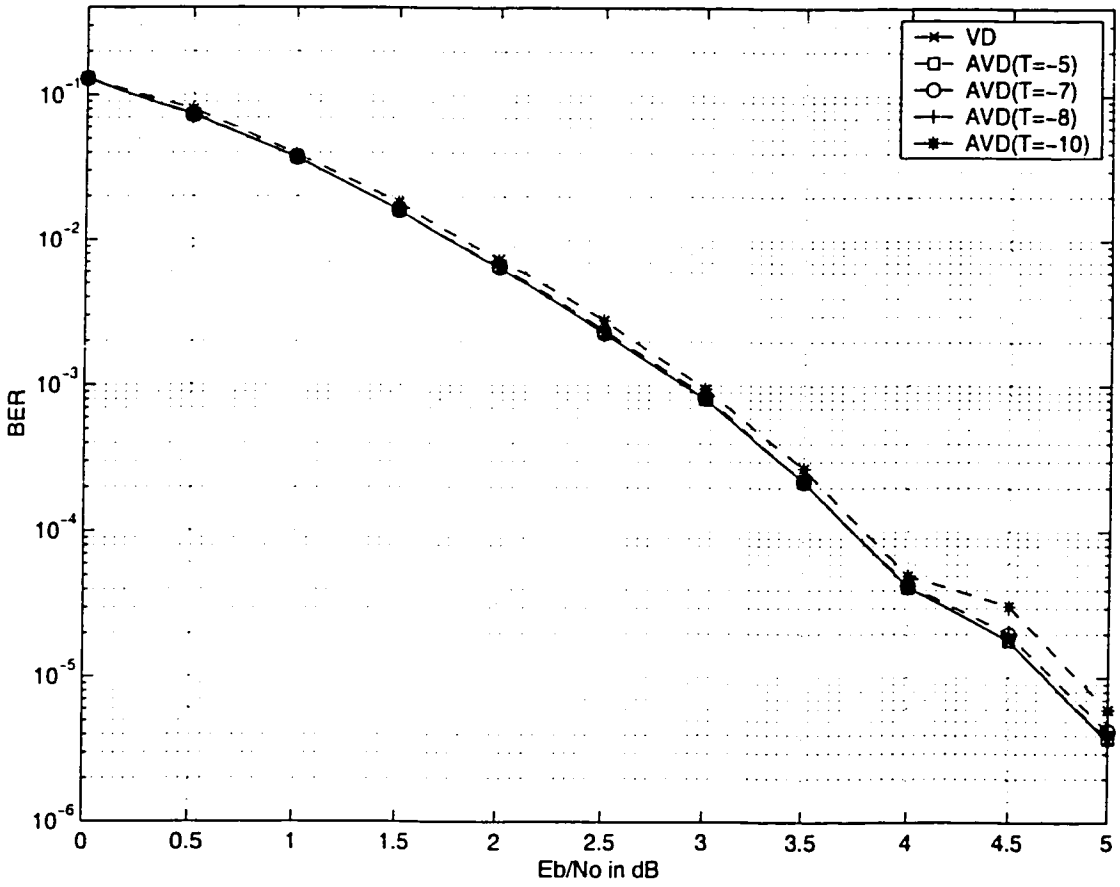


Figure 4.1. Error performance for coded and uncoded binary data

Figure 4.2 shows the error performance of the proposed algorithm for four different values of T as well as that of the non-adaptive Viterbi algorithm. Figure 4.3 and Figure 4.4 show the two expanded forms of Figure 4.2 in the ranges of E_b/N_o from 2.9 dB to 3.05 dB and from 3.8 dB to 5dB, respectively. When the threshold value T is chosen to be -5, the error performance curve for the proposed adaptive Viterbi algorithm completely coincides with that of the non-adaptive one, whereas when T = -7 and T=-8, there is a slight degra-

dation in the performance for E_l/N_o greater than 4 dB. However, for T=-10, this degradation is not negligible. Thus, the error performance can be brought close to that of the non-adaptive Viterbi algorithm by increasing the threshold value.



VD: Non-Adaptive Viterbi Decoder
AVD: Proposed adaptive Viterbi decoder
Figure 4.2. Performance of the bit-error (BER) as a function of Eb/No

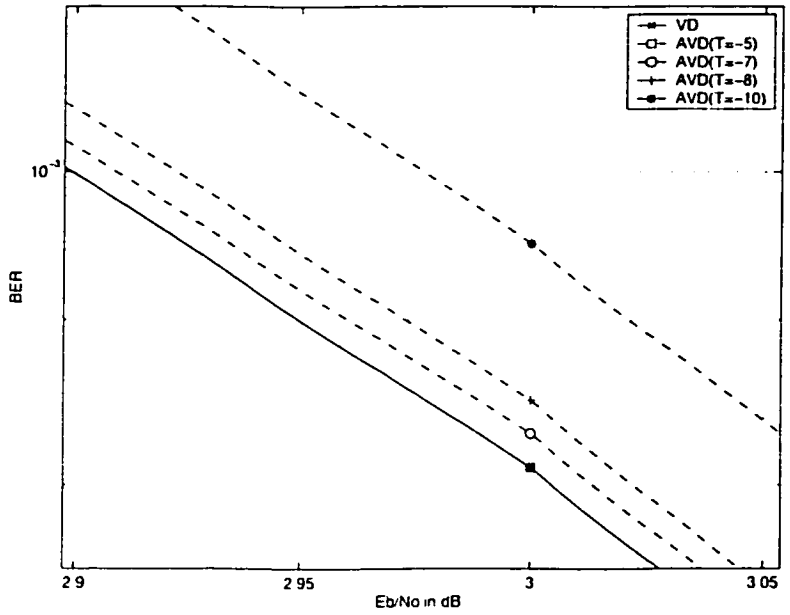


Figure 4.3 an expanded form of Figure 4.2 in range of E_b/N_o from 2.9 dB to 3.0

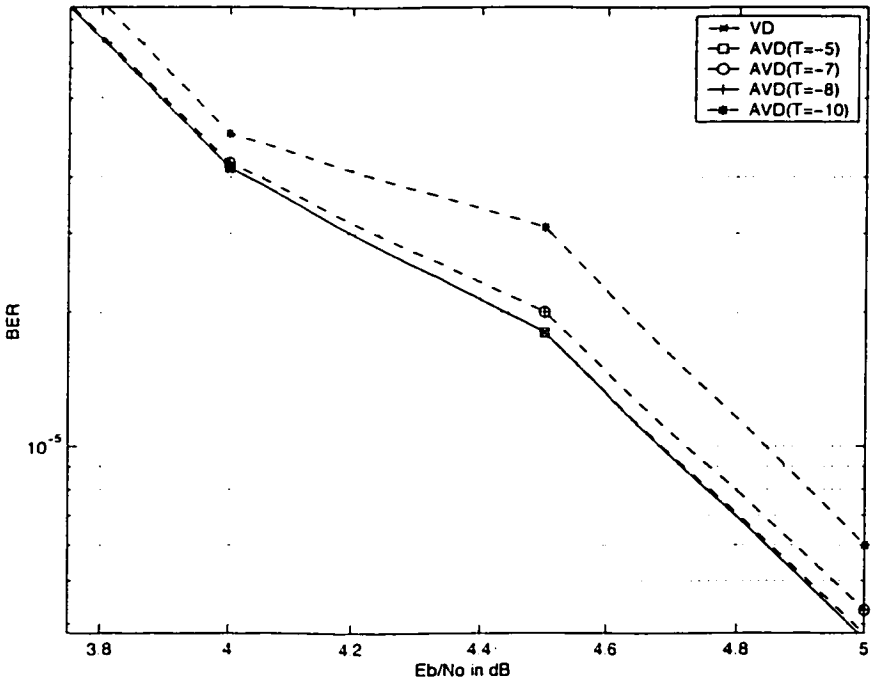


Figure 4.4 an expanded form of Figure 4.2 in range of E_b/N_o from 3.8 dB to 5 dB

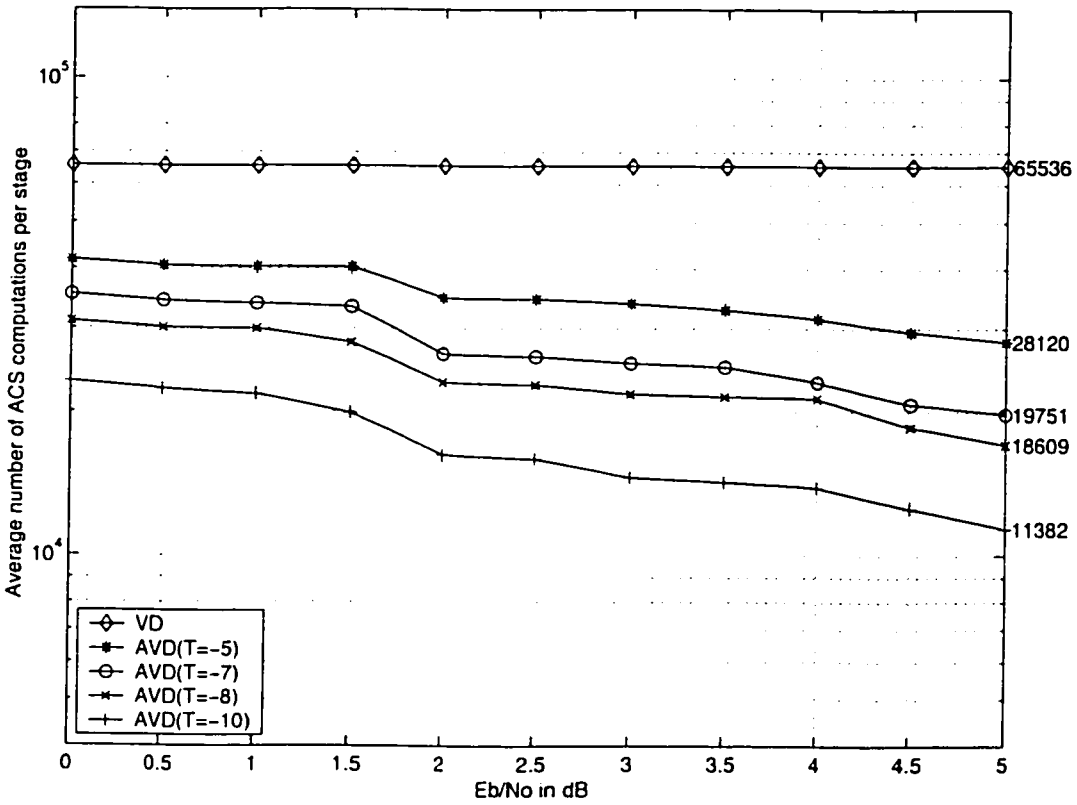


Figure 4.5 Average number of ACS computations per stage over Eb/No

The computational complexity expressed by the average number of ACS computation per stage as a function of E_b/N_o is shown in Figure 4.5 for the non-adaptive Viterbi algorithm as well as for the proposed algorithm for four values of T. It is seen that for a coding gain of 5 dB, the average number of ACS computations per stage using the proposed adaptive algorithm with T = -5 is 42.8% of that of the non-adaptive Viterbi algorithm for a strongly-connected trellis decoding. This number reduces to 30.1%, 28.4%, and 17.4% respectively for T = -7, T = -8, and T = -10. However, as pointed out earlier, there is a degradation in the error performance for T= -10.

Thus, it is seen that by using the proposed algorithm with a threshold value of -7, one can achieve a computational complexity of only 30% of that of the non-adaptive algorithm while maintaining virtually the same error performance.

4.4 Summary

In this chapter, an adaptive Viterbi algorithm based on the strongly-connected trellis decoding of binary convolutional codes has been presented. The proposed adaptive Viterbi algorithm has been developed by reformulating the adaptive Viterbi algorithm presented by Chan and Haccoun [22] so that it is suitable for a systolic array architecture-based hardware implementation. The most important feature that has been introduced in the proposed algorithm is the employment of the strongly-connected trellis decoding technique. This can provide an increased utilization of the hardware resources offered by the systolic array architecture and a higher throughput of the decoding. A bi-product of the use of the strongly-connected trellis decoding is the elimination of sorting operation needed in the low-connectivity technique. An extensive simulation study has been carried out in order to show the performance and the efficiency of the proposed algorithm. It has been shown that with a threshold value of -5, the proposed algorithm can yield the same error performance as that of the non-adaptive Viterbi algorithm based on the strongly-connected trellis decoding, but the amount of the ACS computations reduced to 42.8%. It has also been shown that this reduction in the ACS computations can be reduced to as low as 30% for a threshold value of -7, with a very slight degradation in the error performance. These reductions in the amount of computations by the proposed algorithm can provide the ben-

efits of a lower power dissipation if the algorithm is implemented based on a systolic array architecture.

CHAPTER 5

DESIGN AND IMPLEMENTATION OF AN ADAPTIVE VITERBI DECODER

5.1 Introduction

According to the power estimator results presented in Chapter 3, power consumption in the systolic array architecture-based Viterbi decoder results mainly from the array processors used for generating the composite branch metrics and updating the path metrics at a given stage. This is because these array processors constitute a significant part of the FPGA hardware resource of this design, whereas the operations carried out by the processors result in a dorminant switching activity of the decoding process. In this chapter, by using the adaptive Viterbi algorithm proposed in Chapter 4, a design and implementation of a low-power systolic array-based adaptive Viterbi decoder is presented [18]. It was seen in Chapter 4 that the proposed adaptive algorithm can reduce the amount of ACS computations significantly. It is shown in this chapter that this feature can result in reducing the

amount of switching activities in the array processors for the proposed adaptive Viterbi decoder. As a consequence, the dynamic power consumption of the decoder is reduced.

The block diagram of the proposed adaptive Viterbi decoder is shown in Figure 5.1 In this decoder, the Input buffer & one-stage branch metric generation unit is designed to collect the two soft input sequences and to compute all the possible branch metrics corresponding to the eight low-connectivity trellis stages. The path metric update unit is designed to generate the composite branch metrics and to process the matrix-vector ACS computation. The trace-back unit is designed to retrieve the decoded sequence from the survivor path memory through the trace-back strategy described in Chapter 3. The built-in testbench provides a functional testability to the design, and the control logic provides the timings for all the input signals to each unit. The difference between the implementations of the non-adaptive Viterbi algorithm and the proposed algorithm is only in the path metric update units, all other parts of the design being identical. The trace-back unit, the input

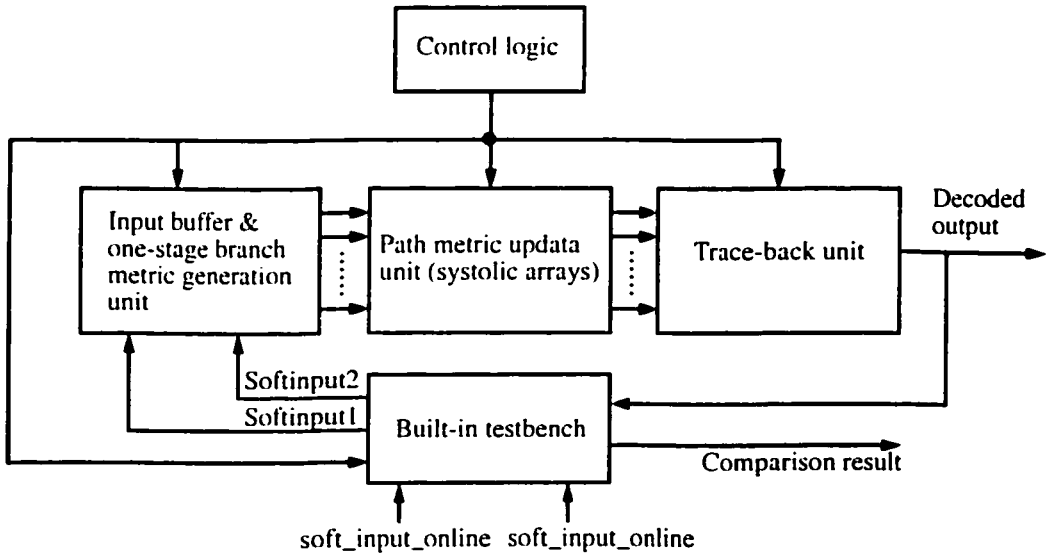


Figure 5.1 Block diagram of the adaptive Viterbi decoder

buffer, the one-stage branch metric generation unit, and the built-in testbench are shown respectively as in Figures 3.12, 3.13, 3.14 and 3.15. In the following section, a systolic array-based path metric update unit with a reduced spurious toggle is presented for the proposed adaptive Viterbi decoder. Furthermore, the performance of the adaptive Viterbi decoder in terms of speed, power and hardware resource utilization is analyzed and compared with that of the implementation of the non-adaptive Viterbi decoder.

5.2 Systolic Array-Based Architecture and Spurious Toggle Reduction

The systolic array architecture for the path metric update unit is shown in Figure 5.2. This unit is the same as the one shown in Figure 3.1 for the Viterbi decoder except that the former unit includes four CMPs, two multiplexers and a dm_plus_T block. In this design, time multiplexing, arithmetic pipelining, and a scheme for avoiding large clock-to-data skews and for providing a tolerance to low clock-to-data skews are employed in the same way as for the Viterbi decoder described in Chapter 3.

Similar to the Viterbi decoder, this path metric update unit processes the matrix-vector ACS computation as given in (4.8). For K=9, the 256 x 256 adjacency matrix $\boldsymbol{B}_q$ is partitioned into four 256 x 64 submatrices, and the four submatrices along with the path metric vector $\mathbf{P}_{q-1}$ are used in (4.8) to update the corresponding 256 (=4x64) path metrics of a given stage. This is done by the four pairs of interconnected linear systolic arrays shown in Figure 5.2. Each pair of the linear systolic arrays is the same as the one shown in Figure 3.2, and the two types of arithmetic-pipelining processors for an BM_4 and an ACS_4 are

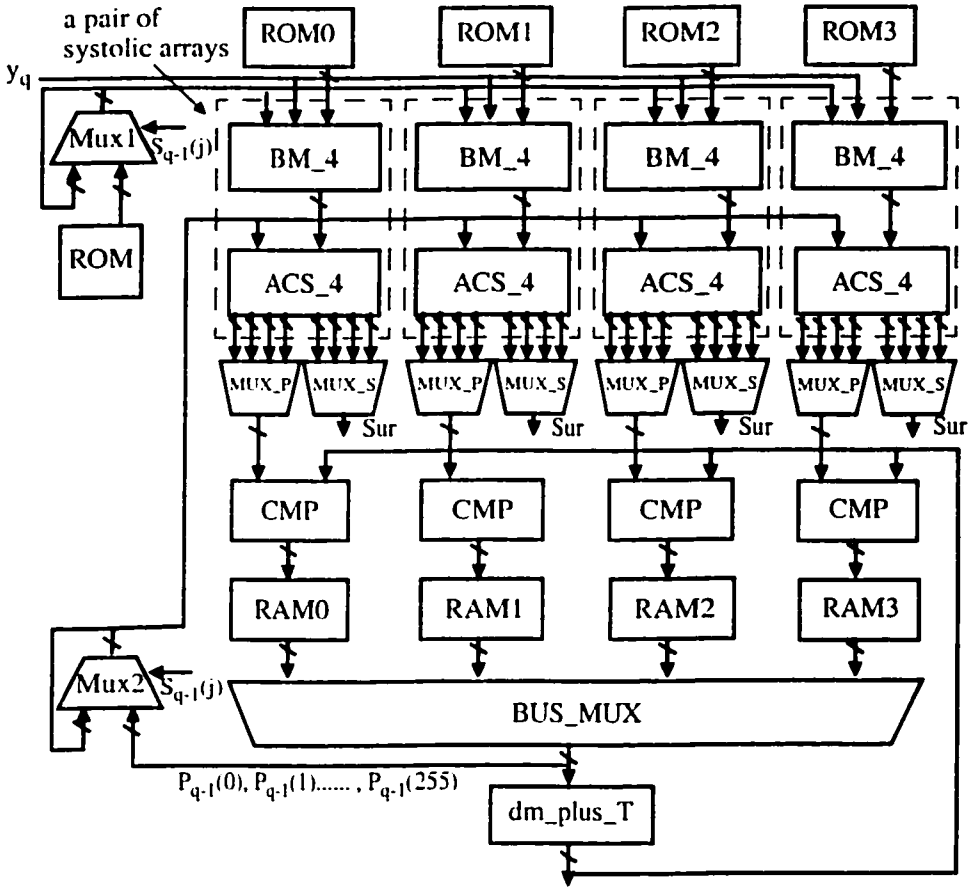


Figure 5.2 Systolic array-based architecture for an adaptive Viterbi decoder

the same as the ones shown in Figure 3.3 and Figure 3.4 respectively. In each pair of systolic arrays, BM_4 is used to process the composite branch metrics given by (2.4) and (2.5), whereas ACS_4 is used to update the corresponding subset of path metrics of a given stage according to (4.6) and to make decisions on the survivor paths for the corresponding subset of path metrics of the stage according to (4.7). For computing the corresponding four subsets of $\mathbf{B_q}$ at each stage, the codeword vector C_{0j} is divided into four 1x64 subvectors to be stored in four ROMs, ROM0 to ROM3, and the codeword vector C_{i0} stored in ROM. Globally, the four submatrix-vector ACS computation are carried out

simultaneously by the corresponding four pairs of systolic arrays of Figure 5.2. Locally, inside each pair of the systolic arrays the corresponding submatrix-vector ACS computation is time multiplexed.

In the four CMP units in Figure 5.2, the survivor information corresponding to the updated path metrics from the four pairs of systolic arrays are obtained by comparing $P_q(j)$ with $(d_m^{q-1} + T)$. In the dm_plus_T unit, d_m^{q-1} is obtained by taking the minimal value among the path metrics at the (q-1)-th stage, and $(d_m^{q-1} + T)$ is calculated. The updated path metric subvectors from the four pairs of systolic arrays are selected by the four multiplexers (MUX_P) and sent to the four CMP units. The path metric subvectors together with the survivor information are written into and read out from the four RAMs, RAM1 to RAM3. The two multiplexers, Mux1 and Mux2, are used to keep the input data streams to the pairs of systolic arrays unchanged from their previous values whenever the corresponding path metric to be computed is not survived, so that there is no switching activity in the arithmetic units of the systolic arrays. This process of eliminating the unnecessary switching is called spurious toggle reduction.

5.3 Performance Analysis and Comparison with the Non-Adaptive Viterbi Decoder

The design of the adaptive Viterbi decoder has been simulated, synthesized and implemented based on the VHDL modeling (see Appendix). In this section, the performance of the proposed adaptive Viterbi decoder is analyzed in terms of speed, power and hardware resource utilization and compared with that of the implementation of the non-adaptive Viterbi decoder.

5.3.1 Speed

From description in Section 5.2, compared to the Viterbi decoder presented in Chapter3, it is seen that a hardware overhead arising from the additional computation is introduced in the adaptive Viterbi decoder. This is because the four CMPs, the two multiplexers, and the dm_plus_T block are needed for the latter for generating the survivor information and for eliminating the spurious toggles. However, the speed of the design is not degraded by the computation overhead. As seen from Figure 5.2, each CMP unit can be viewed as a pipeline stage in the pipeline of the corresponding ACS_4 whose outputs are sent to the CMP unit through the multiplexer MUX_P. Similarly, the dm_plus_T unit works in parallel with the two multiplexers, mux1 and mux2, and can also be viewed as a pipeline stage in the pipeline of any ACS_4. In this way, the critical path delay of the design for the adaptive Viterbi decoder is not changed over that for the non-adaptive one. Therefore, the speed of the former decoder remains the same as that of the latter decoder.

5.3.2 Power

The power consumption is estimated using the Virtex-II power estimatior based on the implementation results provided by the Xilinx implementation tool. In this design, the FPGA resource utilization and power estimator results in memory units, the DCMs, and the I/O cells are the same as the ones for the non-adaptive Viterbi decoder, shown in Tables 3.5, 3.6, and 3.7 of Chapter 3. The comparison between the two systolic array-based decoders can be carried out by comparing the power dissipation in the CLB logic used by the two decoders. Table 5.1 shows the power estimator result for the CLB logic

used in the adaptive Viterbi decoder, and Table 5.2 shows the average toggle rate and P_{INT} in different modules of CLB logic used by the two decoders. It is seen from Table 5.2 that the average toggle rates in the array processors, acsPEx16, brcPEx16, and brcREGx16, for the proposed decoder are only about 30% of the corresponding rates in the non-adaptive

Table 5.1 CLB logic power estimator results for the adaptive Viterbi decoder

Module	Frequency (MHz)	CLB Slices	Flip/Flop or Latches	LUT		Average Toggle Rate(%)	Routing Amount	V_{CCint} Subtotal (mW)
				Shift Register	SelectRAM			
acsPEx16	40	448	608	0	0	15%	High	40
acsCNTx16	40	64	128	0	0	25%	Medium	6
brcPEx16	40	1488	832	480	0	15%	High	138
brcREGx16	40	32	64	0	0	17%	High	3
testPKG	40	48	56	2	0	50%	High	12
conLOGIC	40	206	165	0	0	50%	High	51
recBUF	40	220	256	0	0	12%	High	16
decOUT	40	20	40	0	0	12%	Medium	1
MUX_Ps & MUX_Ss	40	16	0	0	0	2%	High	0
cmpPEx4	40	104	48	0	0	2%	Medium	1
dm_plus_T	40	31	20	0	0	50%	Medium	4
mux1&mux2	40	14	0	0	0	15%	Medium	0
							Total	275

decoder. As explained in Section 5.2, this is due to the fact that spurious toggles have been eliminated in the adaptive Viterbi decoder. This results in a reduction in the dynamic power consumed in these three units in the adaptive Viterbi decoder as compared to the non-adaptive one. As can be seen from Table 5.2, the dynamic power consumed by these

three units in the adaptive Viterbi decoder is only about 35% of that in the non-adaptive decoder. It is seen that the hardware overhead by the CMP units (cmpPEx4), the dm_plus_T unit, and the multiplexers (Mux1 and Mux2) in the adaptive Viterbi decoder consume a power of only about 5 mW. As a consequence, the total internal power P_{INT} in the CLB logic used in the adaptive Viterbi decoder is about 43.7% of that in the non-adaptive decoder.

Table 5.2 Power and average toggle rate in CLB logic for the non-adaptive and adaptive Viterbi decoders

Module	Non-adaptive Viterbi Decoder		Adaptive Viterbi Decoder	
	Average Toggle Rate %	P_{INT} Subtotal (mW)	Average Toggle Rate %	P_{INT} Subtotal (mW)
acsPEx16	50%	114	15%	40
brcPEx16	50%	417	15%	138
brcREGx16	55%	9	17%	3
cmpPEx4	X	X	2%	1
dm_plus_T	X	X	50%	4
mux1&mux2	X	X	15%	0
acsCNTx16	25%	6	25%	6
testPKG	50%	12	50%	12
conLOGIC	50%	51	50%	51
recBUF	12%	16	12%	16
decOUT	12%	1	12%	1
MUX_Ps & MUX_Ss	2%	0	2%	0
	Total	629	Total	275

Table 5.3 shows the power estimator results for the total power. It is seen that the power consumption from the core supply voltage V_{CCint} is dorminant in this design. Table 5.4 shows the static power consumption, the dynamic power consumption, and the total power consumption in the two systolic array-based Viterbi decoder. It is seen that the dynamic

power consumption from V_{CCint} in the adaptive Viterbi decoder is only about 43% of that in the non-adaptive one. Consequently, the total power consumption in the adaptive Viterbi decoder is only about 57% of that in the non-adaptive decoder.

Table 5.3 Power estimator results for the adaptive Viterbi decoder

Target Device		Estimated Design Power Values (mW)			
Device	Package	Total Power	V_{CCint} 1.5 V	V_{CCaux} 3.3 V	V_{CCO} 3.3 V
XC2V1000	FG256	477	442	33	2

Table 5.4 Power consumption for the non-adaptive and adaptive Viterbi decoders

Non-adaptive Viterbi Decoder			Adaptive Viterbi Decoder		
Static Power V_{CCint} 1.5 V	Dynamic Power V_{CCint} 1.5 V	Total Power	Static Power V_{CCint} 1.5 V	Dynamic Power V_{CCint} 1.5 V	Total Power
150 mW	680 mW	830 mW	150 mW	292 mW	477 mW

5.3.3 FPGA Resources Utilization

As mentioned above, the adaptive Viterbi decoder requires a hardware overhead due to the inclusion of the four CMPs, the two multiplexers and the dm_plus_T block for generating the survivor information and for eliminating the spurious toggles. As shown in Figure 5.2 instead of using the four CMP units to process the four outputs from a ACS_4, one CMP unit is used to process these outputs selected by the corresponding multiplexer MUX_P. In this way, only four CMP units instead of 16 CMP units are used to generate the survivor information for the updated path metrics at a given stage. The FPGA resource

utilization with the four CMP units is obviously one-fourth of that with the 16 CMP units. By using four CMP units instead of 16 CMP units, the increase in hardware in the adaptive Viterbi decoder is very little. Table 5.5 shows the FPGA resource utilization for the two systolic array-based Viterbi decoders. There is a total of 5120 slices in the Xilinx VirtexII-XC2V1000-4FG256. It is seen that the adaptive Viterbi decoder takes 2934 slices, which is 57.3% the total number of slices available. On the other hand, the non-adaptive Viterbi decoder takes 2793 slices, which is 54.6% the total number of slices. Consequently, the hardware overhead in the adaptive Viterbi decoder is 5% compared to the hardware needed for the non-adaptive decoder.

Table 5.5 FPGA resources utilization for the non-adaptive and adaptive Viterbi decoders

Target Design	Slices	LUTs	FFs	Latches	BlockRAMs	DCM
Non-adaptive Viterbi Decoder	2793	4083	2693	4	17	1
Adaptive Viterbi Decoder	2934	4255	2777	4	17	1

5.4 Summary

In this chapter, the design and implementation of a low-power systolic array architecture-based adaptive Viterbi decoder has been presented. This adaptive decoder is different from the non-adaptive Viterbi decoder presented in Chapter 3 in terms of the path metric unit. The important change in the path metric update unit for the adaptive Viterbi decoder is the involvement of four CMPs, two multiplexers, and a dm_plus_T block. These units are needed for generating the survivor information and for eliminating the spurious tog-

gles in the array processors caused by the path metric updates corresponding to the paths that are not survived. It has been shown that because of the reduction in the spurious toggles, the average toggle rate at each clock for the array processors in the adaptive Viterbi decoder is only about 30% of that in the non-adaptive Viterbi decoder presented in Chapter 3. This reduced average toggle rate at each clock for the array processors has resulted in lowering the dynamic power consumption. The power estimator results have shown that at the expense of a 5% overhead in hardware, the total power consumption in the adaptive Viterbi decoder is only 57% of that in the non-adaptive decoder.

CHAPTER 6

CONCLUSION

6.1 Summary and Conclusions

The conventional Viterbi algorithm provides an efficient method for the maximum likelihood decoding of convolutional codes. It has been shown that this algorithm can be formulated by employing a matrix-vector computation and it can be implemented in hardware based on a systolic array architecture. It is also known that the strongly-connected trellis decoding method can be used to improve the efficiency of the hardware utilization in the systolic array architecture and the throughput of the decoding. However, the employment of the strongly-connected trellis decoding method results in an excessive amount of computations in the decoding process as the constraint length K becomes large. Moreover, the hardware complexity of the systolic array architecture increases exponentially with the constraint length of the code. This makes the adoption of such a systolic array architecture not feasible in the design of the Viterbi decoder for decoding a convolutional code with a large K.

In this thesis, a study on low-power designs and implementations of systolic array-based Viterbi and adaptive Viterbi decoders for decoding a convolutional code with a large K has been presented. First, the conventional Viterbi algorithm has been reformulated for the strongly-connected trellis decoding. Then, based on this algorithm, the design and implementation of a systolic array-based non-adaptive Viterbi decoder for decoding a convolutional code with the constraint length K=9 and the code rate r=1/2 using Xilinx VirtexII-XC2V1000-4FG256 have been presented. Further, an adaptive Viterbi algorithm that is based on strongly-connected trellis decoding has been proposed in order to reduce the amount of addition, comparison, and selection (ACS) computations. Using this algorithm, the design and implementation of a low-power systolic array-based adaptive Viterbi decoder with K=9 and r=1/2 employing Xilinx VirtexII-XC2V1000-4FG256 has been presented.

In the non-adaptive Viterbi algorithm based on the strongly-connected trellis decoding, the processes of generating the composite branch metrics, and updating the path metrics, the ACS computations using modulo arithmetic, and radix-2^{K-1} trellis trace-back updating in the Viterbi decoding have been reformulated. A composite branch metric is evaluated for updating the path metric using the Euclidean distance between the codeword sequence and the received data sequence under soft demodulator decisions. This computation for a composite branch metric has been simplified into several signed additions under the BPSK modulation. In this way, the multiplication operations for computing the Euclidean distance have been avoided, thus reducing the hardware complexity of the design. The updating of path metrics in the Viterbi decoding process has been formulated as a matrix-vector

ACS computation so that its hardware implementation is suitable for a systolic array architecture. Moreover, the adoption of the modulo arithmetic method has made the normalization for the ACS computations unnecessary so that the overheads of the hardware and the processing time due to the normalization have been avoided. In the trace-back decoding process based on the strongly connected radix-$2^{(K-1)}$ trellis, (K-1) bits can be decoded per trace-back operation. Thus, the strongly-connected trellis decoding has increased the throughput by a factor of (K-1) of that in the original low-connectivity trellis decoding.

In the proposed systolic array-based non-adaptive Viterbi decoder, a novel systolic array architecture that uses four pairs of systolic arrays with each array having four processors have been exploited. In this architecture, the time multiplexing has been employed to process the large matrix-vector ACS computations by using 32 array processors instead of 512 processors that are required if the time multiplexing is not used. In addition, the employment of the time multiplexing technique has lowered the toggle rate in the memory units and the multiplexers in this design so that the power dissipation in these units has been saved. However, a drawback of using the time multiplexing technique is that the speed of the decoder is one-sixteenth of of that without using the time multiplexing technique. This is due to the fact that 16 iterations of time multiplexing are needed to process the matrix-vector ACS computation at a given stage. However, this speed degradation has been compensated, since the strongly-connected trellis decoding has increased the throughput of the decoding and the arithmetic pipelining technique empioyed has increased the clock-rate of the design. The employment of the arithmetic pipelining has not only increased the clock-rate of the design, but it has also ensured that the multi-rate

array processors in this systolic array-based decoder are synchronized. Also, in this two-level pipelining architecture, there is no need of an additional period for initializing the systolic arrays at the beginning of each iteration. This has avoided the speed degradation of the decoder. Furthermore, a scheme for avoiding large clock-to-data skews and for providing a tolerance to low clock-to-data skews have been investigated to ensure that the timings in the design are not violated.

The proposed adaptive Viterbi algorithm is based on the strongly-connected trellis decoding and it is very suitable for a systolic array architecture-based hardware implementation. An extensive simulation study has been carried out in order to show the performance and the efficiency of the proposed algorithm. It has been shown that with a threshold value of -5, the proposed algorithm can yield the same error performance as that of the non-adaptive Viterbi algorithm based on a strongly- connected trellis decoding, but the amount of the ACS computations is reduced to 42.8%. It has also been shown that this reduction in the ACS computations can be lowered to as low as 30.1% for a threshold value of -7 with a negligible degradation in the error performance.

In the proposed systolic array-based adaptive Viterbi decoder, the time multiplexing has been employed to implement the large constraint-length Viterbi decoder with four pairs of arrays of a fixed size. The arithmetic pipelining has been applied to ensure that the processing times of the various array processors can be synchronized without sacrificing the speed of the decoder. The scheme for avoiding large clock-to-data skews and for providing a tolerance to low clock-to-data skews have been used to ensure that the timings in the design are not violated. Moreover, a process of spurious toggle reduction has been intro-

duced to eliminate the unnecessary switching activities in the array processors, caused by the path metric updates corresponding to the paths that have not survived.

The performances of both the non-adaptive and adaptive Viterbi decoders have been tested and estimated in terms of the speed, power, and FPGA resource utilization. It has been shown that the adaptive Viterbi decoder can achieve the same decoding speed and coding gain as the non-adaptive one. The decoding speed and coding gain for both are respectively 78 Kbps and 5 dB at a bit-error-rate of 10^{-5}. However, the total power dissipation for the non-adaptive Viterbi decoder is 830 mW, whereas for the adaptive one it is only 477 mW. The total number of slices for the non-adaptive Viterbi decoder is 2793, whereas for the adaptive one it is 2934. Thus, the total power consumption in the adaptive Viterbi decoder is only 57% of that in the non-adaptive decoder with a hardware overhead of only 5%.

6.2 Suggestions for Future Investigations

In this thesis, it has been demonstrated that the proposed adaptive Viterbi decoder has advantages in terms of power consumption compared to the non-adaptive one. However, the array processors for computing the composite branch metrics and the path metrics utilize a relatively large amount of hardware resources, and thus the power consumption of these array processors dominates the total power consumption of the decoder. If these array processors can be further simplified, the amount of hardware resource utilization as well as the amount of power consumption would be further reduced. On the other hand, if the array processors can be designed with less complexity, we can involve more pairs of

systolic arrays to process the matrix-vector ACS computation so that the number of time multiplexing iterations can be reduced. As a result, the speed of decoding would be improved. Therefore, design and implementation of an adaptive Viterbi decoder with simpler array processors can be investigated in order to lower the power consumption further and to increase the speed.

APPENDIX

The computer programs have been developed using C language and run on a SUN work-station for the simulations of the non-adaptive and adaptive Viterbi algorithms. The Viterbi decoders are modeled using VHDL, simulated using Synopsis VSS, synthesized using Synopsys Design Compiler, and implemented using Xilinx Design Manager. The simulation programs in C and the design models in VHDL, including the initial data for the memory units and the stimulus used for the simulations, are included in a CD-ROM with proper headings, as part of the thesis.

REFERENCES

[1] G. David Forney, JR., "The Viterbi Algorithm," *Proceedings of the IEEE*, Vol. 61, no. 3, pp. 268-278, March 1973.

[2] Vijay K. Gary, *Wireless Network Evolution 2G to 3G*. New York: Prentice hall, 2002.

[3] G. D. Forney, Jr., "Convolutional Codes II: Maximum Likelihood Decoding," *Inf. Control*, 25, pp. 222-266, July 1974.

[4] J. K. Omura, "On the Viterbi Decoding Algorithm," *IEEE Trans. Inf. Theory*, IT-15, pp. 177-179, January 1969.

[5] Bupesh Pandita and Subir K Roy, "Design and Implementation of a Viterbi decoder Using FPGAs," in *Proc. 12th International Conference on VLSI Design*, pp. 611-614, January 1999.

[6] Jang-Hyun Park and Yea-Chul Rho, "Performance Test of Viterbi Decoder for Wide-band CDMA System," in *Proc. IEEE Asp-Dac'97*, pp. 19-23.

[7] Chi-Yung Chang and Kung Yao, "Systolic Array Processing of the Viterbi Algorithm," *IEEE Trans. on Information Theory*, vol. 35, no. 1, pp. 76-86, January 1989.

[8] G. M. Megson, *An Introduction to Systolic Algorithm Design*. Oxford Science Publication.

[9] Yun-Nan Chang, Hiroshi Suzuki, Hiroshi Suzuki, and Keshab K. Parhi, "A 2-Mb/s 256-State 10-mW Rate-1/3 Viterbi Decoder," *IEEE J. Solid-State Circuits*, vol. 35, no. 6, pp. 826-834, June 2000.

[10] Inyup Kang and Alan N. Willson Jr., "Low-Power Viterbi Decoder for CDMA Mobile Terminals," *IEEE Journal of Solid-State Circuit*, vol. 33, no. 3. pp. 473-482, March 1998.

[11] P. Elias, "Coding for Noisy Channels," *IRE Conv. Rec.*, Part 4, pp. 37-47, 1955.

[12] J. M. Wozencraft and B. Reiffen, *Sequential Decoding*, MIT Press, Cambridge, Mass., 1961.

[13] J. L. Massey, *Threshold Decoding*, MIT Press, Cambridge, Mass., 1963.

[14] A. J. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," *IEEE Trans. Inf. Theory*, IT-13, pp. 260-269, April 1967.

[15] Shu Lin and Daniel J. Costello, Jr., *Error Control Coding*. Prentice-Hall, Inc. Englewood Cliffs, New Jersey 07632

[16] C. B. Shung, P. H. Siegel, G. Ungerboeck, and H. K. Thaper, "VLSI architecture for metric normalization in the Viterbi algorithm," in *Proc. int. Conf. Communications*, vol. 4, Atlanta, GA, pp. 1723-1728, Apr. 1990.

[17] Peter J. Black and Teresa H. Meng, "A 140-Mb/s, 32-State, Radix-4 Viterbi Decoder," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 12, pp. 1877-1885, December 1992.

[18] Man Guo, M. Omair Ahmad, M.N.S. Swamy, and Chunyan Wang "FPGA design and Implementation of a Low-Power Systolic Array-Based Adaptive Viterbi Decoder," to be submitted to *IEEE Trans. on Circuits and System-PartII.*

[19] Gennady Feygin and P. G. Gulak, "Architecture Tradeoffs for Survivor Sequence Memory Management in Viterbi Decoders," *IEEE Trans. on communications*, vol. 41, no. 3, pp. 425-429, March 1993.

[20] Xilinx Inc., "A Simple Method of Estimating Power in XC4000XL/EX/E FPGAs," *Application Brief* XBRF 014 June 30, 1997.

[21] K. Weiss, C. Oetker, I. Katchan, T. Steckstor and Prof. Dr. Wolfgang Rosenstiel, "Power Estimation Approach for SRAM-based FPGAs," *FPGA'2000*, pp. 195-202.

[22] Francois Chan and David Haccoun, "Adaptive Viterbi Decoding of Convolutional Codes over Memoryless Channels," *IEEE Trans. on Communication*, vol. 45, no. 11, pp. 1389-1400, November 1997.

[23] Man Guo, M. Omair Ahmad, M.N.S. Swamy, and Chunyan Wang "An Adaptive Viterbi ALgorithm Based on strongly-connected trellis Decoding," in *Proc. IEEE International Symposium on Circuits and Systems*, Scottsdale, Arizona, May, 2002.