AN ABSTRACT OF THE DISSERTATION OF

Jinjin He for the degree of Doctor of Philosophy in Electrical and Computer Engineering presented on April 19, 2010.

Title: Efficient Decoder Design for Error Correction Codes

Abstract approved: _____

Huaping Liu Zhongfeng Wang

Error correction codes (ECCs) have been widely used in communication systems and storage devices. Nowadays, the rapid development of integrated circuit technologies makes feasible the implementation of powerful ECCs such as turbo code and low-density parity-check (LDPC) code. However, these high-performance codes require complex decoding algorithms, resulting in large hardware area and high power consumption. Furthermore, some of these decoders require an iterative decoding process, which leads to a long decoding latency. Therefore, low-complexity, low-power and high-speed very-large-scale integration (VLSI) architecture design for the ECC decoder is of great importance. This dissertation focuses on efficient VLSI implementation for the decoders of convolutional codes and two advanced coding schemes based on convolutional code: trellis-coded modulation (TCM) and convolutional turbo code (CTC).

The first part of this dissertation is dedicated to low-complexity, low-power decoders design for a 4-dimensional, 8-ary phase-shift keying (4-D 8PSK) TCM system. We propose a low-complexity architecture for the transition-metric unit (TMU) to reduce the hardware area without performance loss. Then, a power-efficient scheme by applying T-algorithm on branch metrics (BMs) is proposed for the Viterbi decoder (VD) embedded in the 4-D 8PSK TCM decoder. Unlike the conventional T-algorithm, the proposed scheme does not affect the clock speed of the decoder. Finally, a hybrid T-algorithm is developed by

applying T-algorithm on both BMs and path metrics (PMs), which reduces significantly more computations than the conventional T-algorithm applied on PMs.

The VLSI design for VDs has been an active research area for decades. In the second part of the dissertation, we extend our research to a more general topic of VDs, where novel architectures are explored to efficiently reduce the power consumption, while still maintaining a high decoding speed and a low decoding latency.

CTCs are constructed from parallel convolutional encoding of the same message in different sequences and have the error-correcting capability near the Shannon bound. Practical decoding schemes normally require an iterative decoding process employing the soft-in soft-out (SISO) decoder. The third part of this dissertation is focused on the SISO decoder design for double-binary (DB) CTCs. We propose a low-complexity, memory-reduced architecture by partitioning BMs into two independent portions: information metrics and parity metrics. Furthermore, high-speed recursion architectures for logarithm domain maximum *a posteriori* probability (log-MAP) algorithm are proposed to increase the decoding speed by algorithmic approximation and bit-level optimization.

Efficient Decoder Design for Error Correction Codes

by

Jinjin He

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented April 19, 2010
Commencement June 2010

Doctor of Philosophy dissertation of Jinjin He presented on April 19, 2010

APPROVED:

Co-Major Professor, representing Electrical and Computer Engineering

---

Director of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my dissertation will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my dissertation to any reader upon request.

Jinjin He, Author

ACKNOWLEDGEMENTS

First, I would like to express my sincere gratitude to my advisors, Prof. Huaping Liu at Oregon State University and Dr. Zhongfeng Wang at Broadcom Corporation for their invaluable advice, incessant guidance, continuous encouragement, and financial support throughout my Ph.D. career at Oregon State University. Their broad knowledge, rigorous working attitude and eagerness for new technology are always my model to follow. I am also very grateful to their understanding and caring to my study and life. I can not finish my dissertation successfully without their help.

I am deeply indebted to Dr. Bella Bose, Dr. Ben Lee, Dr. Patrick Chiang, Dr. Kagan Tumer and Dr. Thomas Schmidt for their great efforts and significant amount of time to serve on my Ph.D. committee.

I would like to thank all my friends and colleagues at Oregon State University for their friendships and support. My special thanks go to Dr. Zhiqiang Cui, Dr. Qingwei Li, Lupin Chen and Ruiqing Ye for many useful discussions and help.

Finally, I would like to express my deepest love and appreciation to my parents and my husband, to whom this dissertation is dedicated, for their constant encouragement, support and unconditional love.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF TABLES

EFFICIENT DECODER DESIGN FOR ERROR CORRECTION CODES

1. INTRODUCTION

1.1. Overview

Error correction codes (ECCs) have been widely used in a variety of modern telecommunication and electronic systems [1]. Historically, very-large-scale integration (VLSI) technique plays an important role for the application of ECCs. In the mid-20th century, two powerful ECCs – Reed-Solomon codes [2] and low-density parity-check (LDPC) codes [3]– were discovered, both of which require very complicated decoding algorithms. However, at that time, no hardware was capable of handling such complex computations. Consequently, these codes have been forgotten for a long time. With the rapid development of VLSI technology later last century, implementation of very complex algorithms becomes feasible. Thus, these codes were rediscovered in the late 1990s [1]. Since then, the VLSI architecture design for the decoders of ECCs has become a hot research area.

Our research focuses on efficient VLSI architecture design and implementation for the decoders of convolutional codes, trellis coded modulation and convolutional turbo codes.

Convolutional code is one of the two major groups of ECCs, for which the Viterbi algorithm is known to be the optimal decoding method. Therefore, Viterbi decoders are widely used in modern communication systems and storage systems. Research on Viterbi decoder architectures has focused on the add-compare-select unit (ACSU) and survivor

memory unit (SMU) [4]. ACSU is the bottleneck for high-speed implementations and SMU is the key unit in determining the overall power consumptions.

Trellis coded modulation, (TCM) which typically uses high-rate convolutional codes, is a modulation scheme that allows efficient transmission of information over band-limited channels. In the literature, very few papers have dealt with the implementation issues of TCM decoders. The transition metric unit (TMU) and the Viterbi decoder are the two major modules in a TCM decoder.

Convolutional turbo code (CTC) is a high-performance code that could approach the Shannon bound in theory. However, the iterative decoding for CTCs requires a large amount of memories to store intermediate results. Therefore, memory-reduced architectures for CTC decoders are of great interest. On the other hand, high-speed recursion architecture design is very important for CTC decoders since it is the bottleneck for high-speed implementation. Extensive work has been done on efficient recursion architectures with sub-optimal decoding methods [5–7].

In this dissertation, we address various VLSI design issues and develop efficient approaches to reduce the memory requirement, lower the computational complexity/power consumption, and increase the decoding speed of the decoders for the aforementioned ECCs.

1.2. Summary of Contributions

The main contributions of this dissertation are summarized as follows:

1.2.1 A low-complexity TMU for a 4-dimensional TCM decoder

The main difference between the trellis diagrams of a TCM system and a normal convolutional coded system is that, in the former, there are parallel paths between state transitions, which extend from one state and end at the same state. Therefore, before

performing Viterbi decoding, a transition metric unit (TMU) is used to find the optimal paths among each group of the parallel paths as the branch metrics (BMs). Theoretically, the BM for a soft-decision VD in multi-dimensional TCM is represented as the sum of the Euclidian distance from each dimension of the signal sets [8]. In a multi-dimensional TCM decoder, the TMU could be very complex. We propose a new architecture for TMU, which reduces the computational complexity to about 1/3 of an existing low-complexity architecture presented in [9] while the bit-error-rate (BER) performance is not compromised [10].

1.2.2 The *T*-algorithm on branch metrics

TCM typically employs high-rate convolutional codes. Unlike low-rate codes (e.g., 1/2, 1/3) that have 2 incoming paths for each state, high-rate convolutional codes generate many paths in state transitions. When applying Viterbi decoding on these codes, more branch metrics (BMs) are involved. Motivated by the conventional T-algorithm, we found that eliminating redundant additions is an effective way to reduce computational complexity. Therefore, the T-algorithm could be applied on BMs, instead of on path metrics (PMs) [11]. Compared with conventional T-algorithm on PMs, which will significantly reduce the clock speed due to a maximum/minimum value searching in the add-compare-select (ACS) loop, the proposed T-algorithm on BMs removes the searching operation from the feed-back loop so that the clock speed is not affected.

1.2.3 The hybrid *T*-algorithm

The proposed T-algorithm on BMs not only provides an alternative to the conventional T-algorithm on PMs, but also gives insights leading to techniques that can further reduce computations. Based on this, we develop a hybrid T-algorithm for VD by applying T-algorithm on both BMs and PMs [11]. Simulation results show that computations can be reduced by as much as 50% from the conventional T-algorithm applied on PMs.

1.2.4 A high-speed, low-power Viterbi decoder for high-rate convolutional codes

As we mentioned in Section 1.2.2, conventional T-algorithm on PMs suffers from a decreased clock speed. There have been several papers that suggest using the estimated value, instead of finding the accurate one, during each cycle to overcome the speed drawback of conventional T-algorithm. However, these schemes suffer from a significant performance loss when applied on high-rate convolutional codes. Thus we propose a precomputation architecture to speed up the decoding process with no performance loss from the conventional T-algorithm, at the expense of a small computation overhead [12]. ASIC synthesis and power estimation of the complete VD have verified the low-power property of the proposed architecture [13].

1.2.5 Optimal memory organization for Viterbi decoders

In the Viterbi decoder, the SMU is the most power-consuming module. For low power applications, a trace-back approach (TBA) is usually employed in SMU. However, TBA suffers long latency and low throughput. Employing multiple memory banks could resolve the throughput issue to a great extent at the cost of a higher power consumption. We present two efficient schemes to improve the latency issue of conventional TBA by exploiting the pre-trace-back method [14]. We also adopt a buffer-based TBA method to reduce memory access times, thus reducing power consumption significantly. Simulation results show that the proposed decoding schemes cause either zero or negligible performance loss.

1.2.6 A low-complexity, memory-reduced SISO decoder for double-binary turbo code

When extending CTCs from single-binary (SB) to double-binary (DB), the number of BMs increases exponentially. Therefore, in a soft-in soft-out (SISO) decoder for a DB CTC, BMs, rather than the forward metrics, become the dominant factor in determining

the overall memory size. We propose a memory-reduced method by decomposing each BM into an information metric and a parity metric, reducing the memory size for BMs by half. Then the MAP algorithm is modified accordingly. The modified MAP algorithm reveals that the extrinsic values exchanged between SISO decoders are only related to the forward metrics, the backward metrics, and the parity metrics. Therefore, unnecessary computation can be removed from the decoder. We thus develop an architecture based on BM partitioning and the modified MAP algorithm, a promising solution for low-power implementation [15].

1.2.7 A high-speed recursion architecture for log-MAP and TL-log-MAP algorithms

As the number of paths reaching each state is increased from 2 to 4, the length of the critical path as well as the computational complexity is increased drastically. A simplified algorithm [16] proposes to find the two largest values within all the candidates first. Then the log-MAP algorithm is applied to the two largest value only, rather than to all candidates. When this two-largest (TL) log-MAP algorithm is employed for DB CTCs, simulation result shows that it has no observable performance loss compared with log-MAP algorithm. However, a straightforward implementation of the TL log-MAP algorithm will dramatically reduce the decoding speed due to the sorting process to determine the two largest candidates. We propose two high-speed recursion architectures for TL log-MAP algorithm [17], where the decoding speed is improved by 50% through algorithmic approximation and bit-level optimization with negligible BER performance loss.

1.3. Organization of the Dissertation

This dissertation is outlined as follows.

Chapter 2 reviews some important information theory backgrounds for convolutional

codes, trellis coded modulation, and convolutional turbo codes.

Chapter 3 focuses on the high-speed low-complexity VLSI architecture design for a 4-D 8PSK TCM codec. We first give the system overview. Then the encoder design is discussed. After that, the proposed low-complexity TMU, the T-algorithm on branch metrics and the hybrid T-algorithm are presented in the decoder design section.

In Chapter 4, we discuss the low-power implementation for Viterbi decoders. We first present the high-speed, low-power VD design. Then a general optimization for memory organization in Viterbi decoders is discussed.

Chapter 5 is dedicated to the high-speed, memory-reduced VLSI design for double-binary CTCs.

Chapter 6 gives a brief conclusion and some potential future research topics.

2. ERROR CORRECTION CODES

Error correction codes (ECC) are employed in most electronics devices that people use everyday. For example, the International Standard Book Number (ISBN) system identifies every book with a ten-digit number, such as 7-5053-7048-0, where the first nine digits are the actual number and the tenth is a redundant number derived from a mathematical formula based on the first nine. If a single one of the digits is changed, as in a misprint when ordering a book, a simple check verifies that something is wrong. In brief, the idea of ECC is to add redundant information to the message in a controlled way. When error occurs during the transmission of the coded data, the receiver can recover the original message by exploiting the redundant information. For binary data transmissions, a theoretical way to evaluate an ECC's performance is to measure the minimum Hamming distance between all the possible codewords (coded data), where Hamming distance is the number of positions at which the corresponding symbols are different between two codewords. For example, the Hamming distance between codewords "010" and "100" is 2. The minimum Hamming distance of an ECC is also defined as the "minimum free distance". The larger the minimum free distance, the better the bit-error-rate (BER) performance, or the stronger the error-correcting capability.

The two basic groups of ECCs are block codes and convolutional codes [18]. Advanced codes such as Reed-Solomon codes [2] and LDPC codes [3] belong to the category of block codes, while CTCs [19] are based on convolutional codes. This dissertation is focusing on VLSI architecture design for decoding convolutional codes and two convolutional-code-based schemes: TCM and CTC. This chapter briefly summarizes the background information that is required for development in later chapters.

2.1. Convolutional Codes

Convolutional codes are adopted in many modern communication standards. Unlike block codes, whose codewords only depend on the current input, the output of convolutional codes depend on both current and past data bits sent to the encoder. In other words, convolutional codes are codes with memories, where a parameter called constraint length K is used to represent the length of convolved memory. The encoder for a convolutional code with a constraint length of K contains $K-1$ delay elements. Just as other ECCs, the code rate of the convolutional codes is defined as the number of input bits over the number of output bits. An example of a rate-1/3 convolutional code encoder is shown in Fig. 2.1, where the constraint length K equals 3. For convenience of analysis, convolutional codes are commonly described in the form of a trellis diagram. The trellis diagram of the code shown in Fig. 2.1 is given in Fig. 2.2, where the nodes in a column are the 4 states represented by the delay elements in the encoder[1]. The basic trellis diagram shown in Fig. 2.2(a) describes the state transition based on different input data, while the extended trellis in Fig. 2.2(b) helps illustrate the encoding process: the encoded sequence is a path traveling through the extended trellis from time slot 0 to time slot n.

In 1968, Heller derived an upper bound on the minimum free distance of a rate-$1/n$ convolutional code. It is given by [20]

$$d_{free} \leq \min_{l \geq 1} \lfloor \frac{2^{l-1}}{2^l - 1}(K + l - 1)n \rfloor \tag{2.1}$$

where$\lfloor \cdot \rfloor$ denotes the floor function. Eq. (2.1) reveals that the longer the constraint length K, the better the performance. Table 2.1 lists the optimal rate-1/2 convolutional codes with different constraint lengths. These codes were obtained by computer search and are optimal in the sense that, for a given constraint length, they have the largest possible d_{free}.

[1]In this dissertation, the input and output data of the encoders are assumed to be binary, unless explicitly stated otherwise

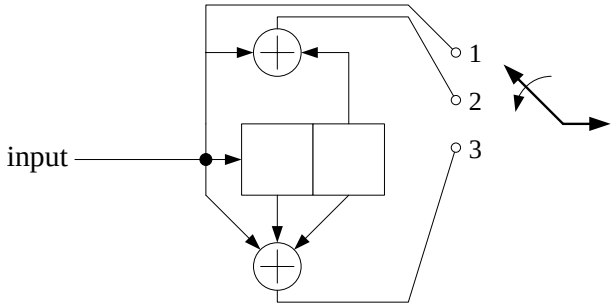
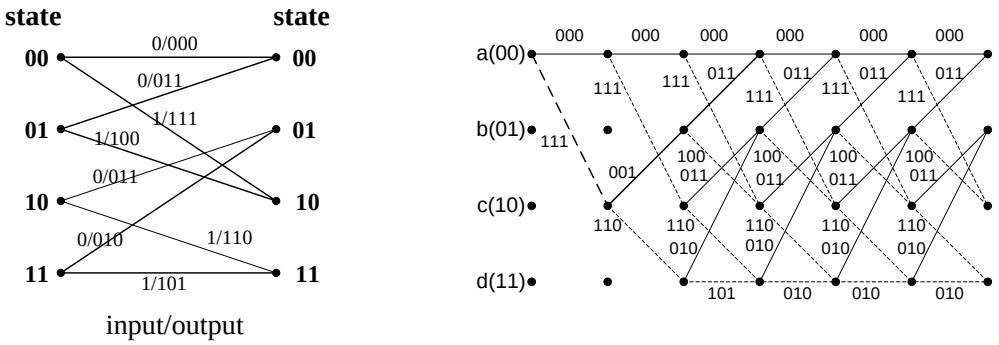


FIGURE 2.1: Encoder for a rate-1/3 convolutional code.



(a). Basic trellis. (b). Extended trellis.

FIGURE 2.2: Trellis diagram of a rate-1/3 convolutional code.

Although the minimum free distance of the listed optimal codes do not always match the theoretical upper bound in Eq. (2.1), both of them show the trend that increasing the constraint length does improve the performance.

In some practical applications, there is a need to employ high-rate convolutional codes (*i.e.*, rates of $(n-1)/n, n = 2, 3, 4...$). The trellis for such high-rate codes has $2^{(n-1)}$ branches that enter each state. Consequently, there are $2^{(n-1)}$ metric computations per state and many comparisons for selecting the best path. The implementation of the decoder of a high-rate code can be very complex. An alternative way to design a high-rate

TABLE 2.1: Rate 1/2 maximum free distance codes

Constraint length K	Generators in octal		d_{free}	Upper bound on d_{free}
3	5	7	5	5
4	15	17	6	6
5	23	35	7	8
6	53	75	8	8
7	133	171	10	10
8	247	371	10	11
9	561	753	12	12

convolutional code is to delete some of the coded bits. The deletion of selected coded bits at the output of a convolutional encoder is called puncturing, and the resulting code is called a punctured convolutional code. Punctured convolutional codes are widely used for constructing high-rate CTC.

For decoding convolutional codes, the Viterbi algorithm [21] is known as the optimal decoding method in the sense of maximum likelihood sequence detection (MLSD). By exploiting the extended trellis, the Viterbi algorithm implements the maximum likelihood algorithm with a drastically reduced computational complexity. Consider the extended trellis shown in Fig. 2.2(b): the MLSD compares the received sequence with all the possible paths traveling through the trellis, where the differences between the received one and the candidate paths are calculated. The candidate path with the minimum difference is selected as the decoded sequence. In the Viterbi algorithm, the path difference is defined as the path metric (PM) and is derived from accumulating the branch metrics (BMs) on the PMs in previous time slot. A BM represents the difference between an input codeword and the transition branch connecting the states of adjacent time slots. Note that in Fig. 2.2(b), at each time slot, all the paths will reach one of the 4 states. Therefore, for each state, the

Viterbi algorithm compares all the incoming paths. The optimal one with minimum PM is kept and others are discarded. It is easy to prove that the maximum likelihood path would not be any of the discarded one. A Viterbi decoder typically contains 4 modules as shown in Fig. 2.3:

- branch metric unit (BMU) – converts received symbols into branch metrics for ACSU;
- add-compare-select unit (ACSU) – recursively updates new path metrics for each state;
- survivor memory unit (SMU) – stores the selected path for each state;
- path metrics unit (PMU) – buffers path metrics generated by ACSU.

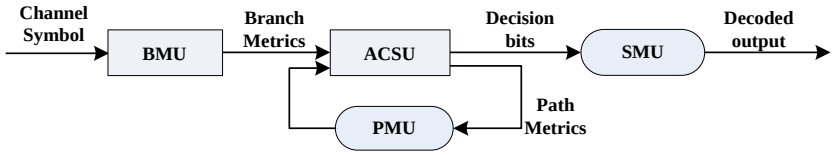


FIGURE 2.3: Block diagram of a Viterbi decoder.

There are two types of Viterbi decoders: hard Viterbi decoder and soft Viterbi decoder. In hard Viterbi decoder, the input is hard-decision binary codewords from the demodulation module, and the BMs are Hamming distances between codewords associated to the states transition lines in the trellis diagram and an input codeword. Soft Viterbi decoder directly takes the received symbols as the input, and the BMs are the Euclidean distance between the received symbols and constellation points. Soft Viterbi decoding gains about 2 dB over the hard decoding scheme in an additive white Gaussian noise (AWGN) channel.

From the discussion above, we know the Viterbi algorithm requires the computation of $2^{(K-1)} \times 2^{(n-1)}$ metrics each cycle, among which only $2^{(K-1)}$ would be stored.

Also, 2^{K-1} surviving sequences are saved, each of which is about $6K$ long. The computational burden and the storage requirement for the Viterbi decoder make it impractical to decode convolutional codes with a very large constraint length. Other non-optimal, low-complexity decoding algorithms for convolutional codes include sequential decoding, stack algorithm, and feedback decoding. Interested readers can refer to [20] for details.

2.2. Trellis Coded Modulation

When ECCs are employed in a system, the data rate for the information bits is reduced. Generally, there exist two possibilities to compensate for the rate loss: increasing the modulation rate if the channel bandwidth allows, or enlarging the signal set of the modulation system if the channel is band-limited. The latter approach leads to the use of nonbinary modulation [22]. However, given a fixed signal power, increasing the constellation size usually leads to an increasing of bit-error rate (BER) because of the smaller spacing between signal points. Therefore, performance improvement gained by encoding is counteracted by the performance loss in the modulation stage. Moreover, in the traditional communication systems, encoding and modulation are treated as two independent processes, and increasing the Hamming distance at the encoding stage does not guarantee a larger Euclidian distance at the modulation stage (Euclidian distance is the distance between two constellation points; just as the Hamming distance, a larger Euclidian distance indicates a better BER performance).

The coded modulation scheme is proposed by G. Ungerboeck [22], which gains noise immunity over uncoded transmission without expanding the signal bandwidth or increasing the transmission power. The encoding process will directly increase the Euclidian distance rather than the Hamming distance. Ungerboeck concentrated his efforts on finding trellis-based signaling schemes that use signal sets of size 2^{m+1} for transmission of m

bits per modulation interval. The following is an example to illustrate the increase of the Euclidian distance.

Consider a sequence of uncoded data modulated using quadrature phase-shift keying (QPSK). Every 2 bits are grouped together as (2,2) block code and modulated into 1 signal symbol. The minimum squared Euclidian distance (MSED) between QPSK signal points is $2E_s$, where E_s is the energy per symbol. For convenience of calculation, E_s is normalized as 1. In trellis-coded modulation, rate-1/2 convolutional encoding is performed only on 1-bit of the original 2. Then, the 2-bit coded data, together with another uncoded bit are modulated into 8 phase-shift keying (8PSK) signals. Fig. 2.4 shows the trellis diagram of the code. Because of the uncoded bit, the transitions in the trellis diagram occur in pairs of two parallel paths. The distance property of the example is illustrated in Fig. 2.5. For any unparallel path, the minimum free squared Euclidean (MFSE) d^2_{free} distance is defined as the MSED between any two signal paths in the trellis that diverge in one state and re-merge in the same state after more than one transition. In the example, $d^2_{free} = d^2_1 + d^2_0 + d^2_1 = 4.585$. The MSED between two parallel paths in this example is 4. Therefore, compared with the uncoded QPSK system, TCM increases the MSED from 2 to 4!

From the above example, it is clear that the methodology of mapping codewords to constellation points is essential to TCM. This process is called partitioning of the constellation by Ungerboeck. A systematic approach for this partitioning is developed in [8], which is based on three heuristic rules:

- Use all subsets with equal frequency in the trellis;
- Transitions originating from the same state or merging into the same state in the trellis are assigned subsets that are separated by the largest Euclidean distance;
- Parallel state transitions (when they occur) are assigned signal points separated by the largest Euclidean distance. Parallel transitions in the trellis are characteristic of

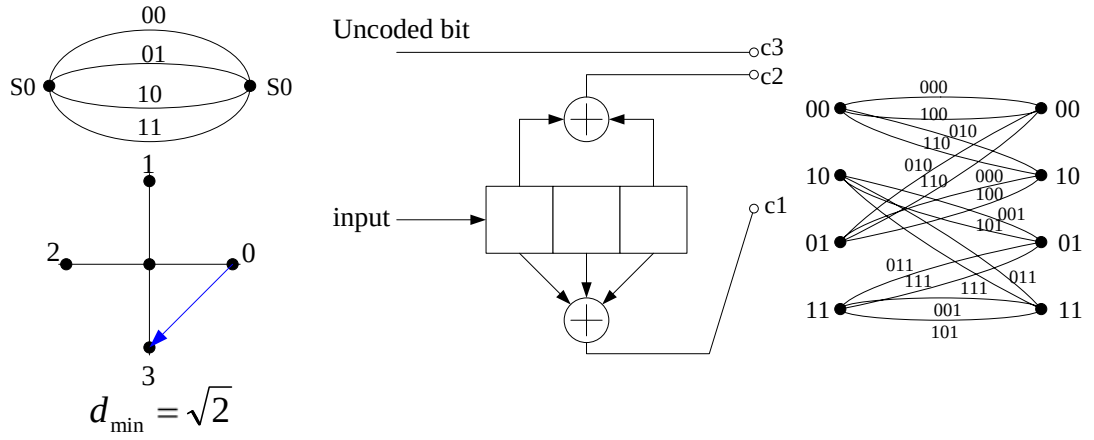


FIGURE 2.4: An example of TCM based on convolutional encoding.

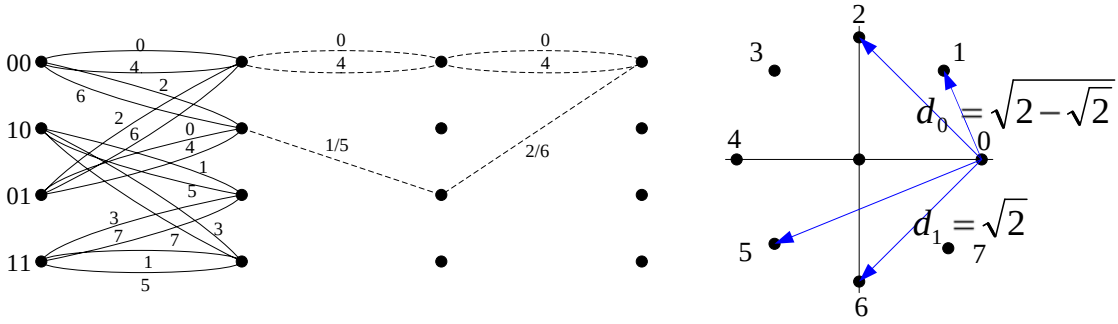


FIGURE 2.5: Free distance of the TCM.

TCM that contains one or more uncoded information bits.

The partitioning of the 8PSK signals is shown in Fig. 2.6. For the TCM system illustrated in Fig. 2.4, the coded bits select one of the four signal subsets at level 2 in Fig. 2.6, and the uncoded bit selects one of the constellation points in a subset.

Later on, Ungerboeck extended TCM schemes to multi-dimensional constellations, which further improves the BER performance [23]. The use of multidimensional modula-

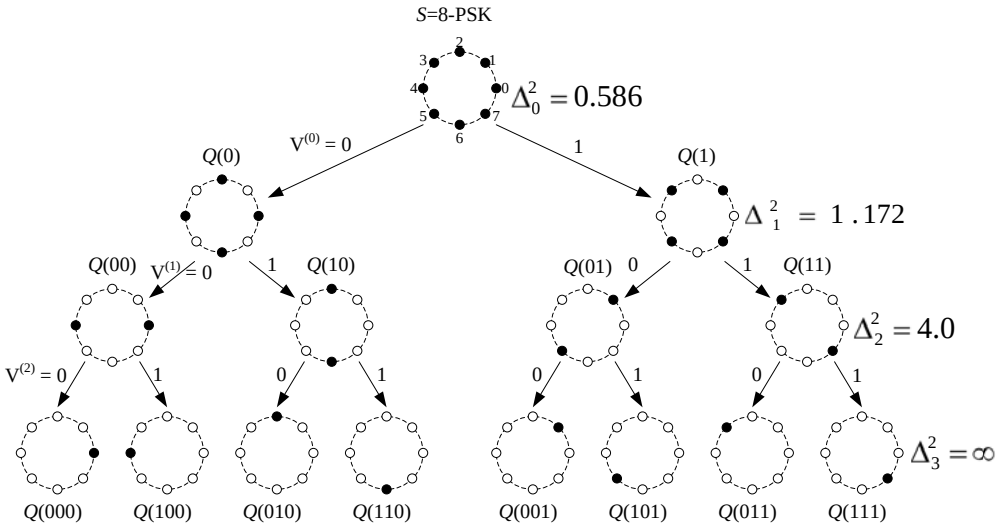


FIGURE 2.6: Partitioning of 8PSK signal.

tions has several advantages over traditional one-dimensional TCM:

- Flexibility in achieving a variety of fractional information rates;
- Codes are partially or completely transparent to discrete phase rotations of the signal set;
- Suitability for use as inner codes in a concatenated coding system
- Higher decoder speeds resulting from the high rate codes used (rate $k/(k+1)$ with k up to 15).

Partitioning of the multi-dimensional signal sets follows the same rules discussed above, though the process is more complex. Multi-dimensional partitioning for two common modulations: 2×8-PSK and 4×8-PSK is discussed in *Appendix*. The method can be further applied to higher dimensions. Interested readers can refer to [23] for more information. A more general encoder system of TCM is shown in Fig. 2.7, which consists of three principal sections: a differential encoder, a binary convolutional encoder, and a

multidimensional mapper. During the transmission, the signal phase may be rotated by the channel, and the rotation may lead to a wrong decision in demodulation. Differential decoding is a commonly used method to overcome the phase rotation. In a TCM system, differential encoding is only applied to several important bits. Then a subset is chosen from the constellation points through convolutional encoding and the uncoded data chose one point in the subset by the mapper.

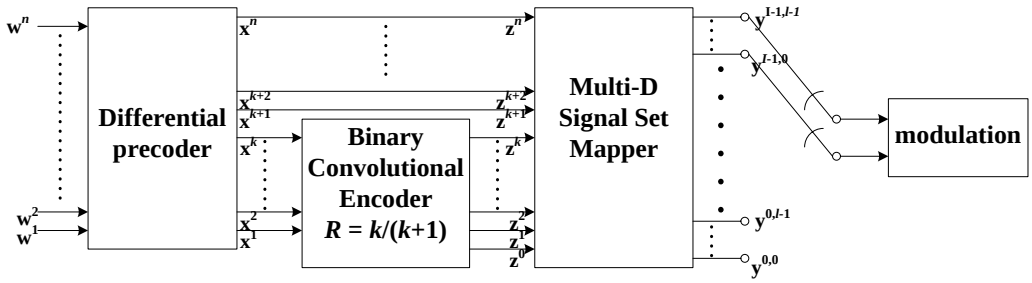


FIGURE 2.7: A general encoder system for TCM.

Decoding for a TCM system is based on the Viterbi algorithm since convolutional codes are involved. There are two general methods: "brute-force" method [9] and a simpler approach based on "auxiliary trellis" [8]. The first one is straightforward but has a high complexity; thus, it is rarely used in practice. The second one employed a"auxiliary trellis" to perform a low-complexity computation for finding the best path among all the parallel paths first. Then a regular Viterbi algorithm is used to further decode the signal. Details of "auxiliary trellis" will be discussed in Chapter 3.

2.3. Turbo Codes

Turbo codes have been considered as one of the most significant technology breakthroughs in the information theory in 1990s, because they are the first practical codes to

closely approach the Shannon bound. The code was first introduced by Berrou, Glavieus and Thitimajshima in 1993 in their paper: "Near Shannon Limit Error-correcting Coding and Decoding: Turbo-codes", published in the Proceeding of IEEE International conference on Communications [19]. The encoder of a convolutional turbo code (CTC) is illustrated in Fig. 2.8, where two recursive systematic convolutional (RSC) encoders work in parallel. One encoder encodes the original message and the other works on the permutated message sequence.

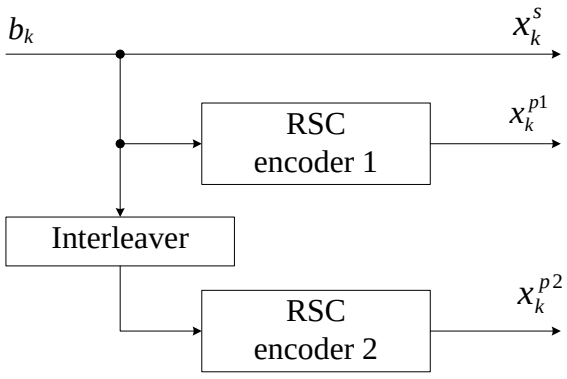


FIGURE 2.8: Turbo encoder structure.

The CTC decoder is built in a similar way as the encoder, where two basic decoders are interconnected in series as shown in Fig. 2.9. Optimal decoding algorithm for CTC requires iterative decoding, where the soft information of the received sequence is passed from a previous decoding iteration to the current iteration. Therefore, DEC1 and DEC2 should be capable of generating the soft outputs, which represent the probability of a bit being '1' or '0' as expressed in Eq. (2.2). Since the conventional Viterbi decoder is unable to calculate the probability value, it can not be directly used in DEC1 or DEC2. The soft-input soft-output (SISO) decoder employs the modified BCJR algorithm, which is also known as the maximum *a posteriori* probability (MAP) algorithm.

The MAP algorithm can be summarized as follows. Assume the output of the

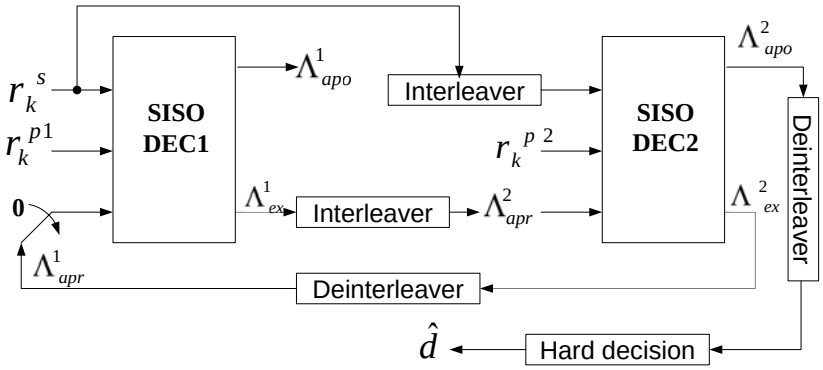


FIGURE 2.9: Turbo decoder structure.

Gaussian memoryless channel is a sequence $R_1^N = R_1 \cdots R_k \cdots R_N$, where k is the time index and $1 \leq k \leq N$. The MAP algorithm evaluates the probability of each bit to be 1 or 0 under the condition of the received sequence R_1^N. When decoding a specific bit at time slot k, the process turns out to be simplified if we take advantage of the trellis structure. Again, considering the trellis diagram shown in Fig. 2.2 (a), we use m' to represent the state at time slot $k-1$ and m for state at time slot k. Thus, the probability of the bit to be 1 equals $\sum_m \sum_{m'} Pr\{S_k = m\} Pr\{S_{k-1} = m'\} Pr\{d_k = 1, S_k = m, S_{k-1} = m'\}$. $Pr\{S_k = m\}$ is called the backward state metric, which represents the probability that the state at time slot k to be m and is related to received symbols R_{k+1}^N. $Pr\{S_{k-1} = m'\}$ is known as the forward state metric, which represents the probability that the state at time slot $k-1$ to be m' and is related to received symbols R_1^{k-1}. $Pr\{d_k = 1, S_k = m, S_{k-1} = m'\}$ is the probability the previous state being m' and current state being m under the condition that the current received symbol is R_k and the decoded bit is 1. In the iterative decoding, the soft input will contain another portion called the *a priori likelihood ratio*, which comes from the soft output of the last decoding iteration. The mathematic expression of the MAP algorithm is given in Eq. (2.2)

$$\begin{aligned}\lambda(d_k) &= \frac{p(d_k=1)}{p(d_k=0)} \\ &= \frac{\sum_m \sum_{m'} \gamma_1(R_k, m', m)\alpha_{k-1}(m')\beta_k(m)}{\sum_m \sum_{m'} \gamma_0(R_k, m', m)\alpha_{k-1}(m')\beta_k(m)},\end{aligned} \tag{2.2}$$

where γ, α and β are the branch metrics, forward state metrics and backward state metrics respectively, and are defined through Eq. (2.3) to Eq. (2.5), where $i = 0, 1$.

$$\begin{aligned}\gamma_i(R_k, m', m) &= p(R_k/d_k = i, S_k = m, S_{k-1} = m') \cdot q(d_k = i/S_k = m, S_{k-1} = m') \cdot \\ &\quad \pi(S_k = m/S_{k-1} = m'); \qquad (2.3)\\ \alpha_k(m) &= \sum_{m'} \gamma_i(R_k, m', m)\alpha_{k-1}(m'); \qquad (2.4)\\ \beta_k(m') &= \sum_m \gamma_i(R_{k+1}, m', m)\beta_{k+1}(m). \qquad (2.5)\end{aligned}$$

In Eq. (2.3), $p(\cdot/\cdot)$ is the transition probability of the channel, $q(\cdot/\cdot)$ is the *a priori likelihood ratio*, and $\pi(\cdot/\cdot)$ is a fixed probability derived from the trellis diagram. Usually the input at the encoder is a random data sequence with even probability of 1 and 0. Therefore, for most of the cases, $\pi(\cdot/\cdot)$ is a constant for all existing transition lines in the trellis and can be removed from the equation.

For simplicity of hardware implementation, log-MAP algorithm is used in practical design, where all the computation is transformed into logarithm domain. Therefore, branch metrics, forward state metrics, and backward state metrics should be re-defined as Eq. (2.6) to Eq. (2.10), and the multiplication operations are converted to addition operations in the new algorithm. According to Fig. 2.9, current Λ_{apr} in Eq. (2.6) is Λ_{ex} from previous decoding process after interleaving or de-interleaving.

$$\gamma_i(R_k, m', m) = \log(p(R_k/d_k = i, S_k = m, S_{k-1} = m')) + \Lambda_{apr}(k); \quad (2.6)$$

$$\alpha_k(m) = \log(\sum_{m'} \exp(\gamma_i(R_k, m', m) + \alpha_{k-1}(m'))); \quad (2.7)$$

$$\beta_k(m') = \log(\sum_{m} \exp(\gamma_i(R_{k+1}, m', m) + \beta_{k+1}(m))); \quad (2.8)$$

$$\Lambda_{apo}(k) = \log(\sum_{m}\sum_{m'} \exp(\gamma_1(R_k, m', m) + \alpha_{k-1}(m') + \beta_k(m))) - \log(\sum_{m}\sum_{m'} \exp(\gamma_0(R_k, m', m) + \alpha_{k-1}(m') + \beta_k(m))); \quad (2.9)$$

$$\Lambda_{ex}(k) = \Lambda_{apo}(k) - \Lambda_{apr}(k). \quad (2.10)$$

The first CTC only considers the single binary input. As a result, there are only two paths merging into one state. Mathematically, Eq. (2.7) and Eq. (2.8) can be treated as the same class of equation: $S = \log(\exp(a) + \exp(b))$. Note that,

$$\begin{aligned} S &= \log(\exp(a) + \exp(b)) \quad (2.11)\\ &= \log(\exp(\max(a, b)) \cdot (1 + \exp(-|a - b|))) \\ &= \max(a, b) + \log(1 + \exp(-|a - b|)). \end{aligned}$$

When the second term of Eq. (2.12) is ignored, the calculation for α reduces to the conventional Viterbi algorithm, and the computation for β can be treated as a backward Viterbi process. The simplified log-MAP is also known as the MAX-log-MAP algorithm, which leads to lower computational complexity, but with some performance loss. We can also maintain the high performance of log-MAP scheme by implementing $\log(1+\exp(-|a-b|))$ as a look-up-table (LUT).

In 2005, the inventor of the basic single-binary turbo codes, Claude Berrou, proposed the turbo codes based on multiple-input convolutional codes [24]. These codes offer several advantages over classical single binary CTC:

- Better convergence of the iterative process;

- Larger minimum distance;
- Less puncturing for a given rate;
- Higher throughput and lower latency;
- More robust decoder.

Recently, double-binary (DB) CTC has attracted significant research interest and has been adopted in wireless communication standards such as digital video broad-casting-return channel over satellite and terrestrial (DVB-RCS and DVB-RCT) [25] and WiMAX [26]. The decoding process for DB CTC is similar to the conventional binary CTC, though it requires more computational efforts. Details about decoding DB CTC will be introduced in Chapter 5.

2.4. Conclusion

In this chapter, we have reviewed the background information of convolutional codes, TCM system, and turbo codes. The decoding algorithm for each scheme is introduced. This information is the basis for the readers to understand the materials in the following chapters.

3. AN EFFICIENT VLSI ARCHITECTURE FOR THE 4-D TCM DECODER

3.1. System Overview

TCM techniques have been employed in communication systems emphasizing on the bandwidth efficiency. For example, the 4-dimentional (4-D) 8PSK TCM system studied in our research is considered by National Aeronautics and Space Administration (NASA) for space data transmission [27].

The convolutional encoder and constellation mapper for the 4-D 8PSK TCM system are specified in [27]:

- Size of the constellation: $M = 8$ phase states (8PSK);
- Number of signal set constituent is 4 (4-D);
- Number of states for the trellis encoder is 64;
- Rate of the convolutional encoder used for the construction of the trellis is 3/4;
- Rate of modulation: Rm=$m/(m+1)$, m =8, 9, 10 or 11.
- Efficiency of the modulation:
  - $R_{eff} = 2.0$ bits per channel-symbol (Rm=8/9);
  - $R_{eff} = 2.25$ bits per channel-symbol (Rm=9/10);
  - $R_{eff} = 2.5$ bits per channel-symbol (Rm=10/11);
  - $R_{eff} = 2.75$ bits per channel-symbol (Rm=11/12);

The TCM system was modeled in Matlab and the BER performance is simulated as shown in Fig. 3.1. From the figure, we can observe that, near the BER of 10^{-4}, with

the same modulation efficiency of 2 bits/symbol, the 4-D 8PSK TCM (Rm=8/9) achieve 2 dB gain over the uncoded QPSK system. Compared with 8PSK signal which transmit 3 bits/symbol, the TCM system with a modulation rate of 11/12 gains 3 dB at the cost of a very small loss of modulation efficiency.

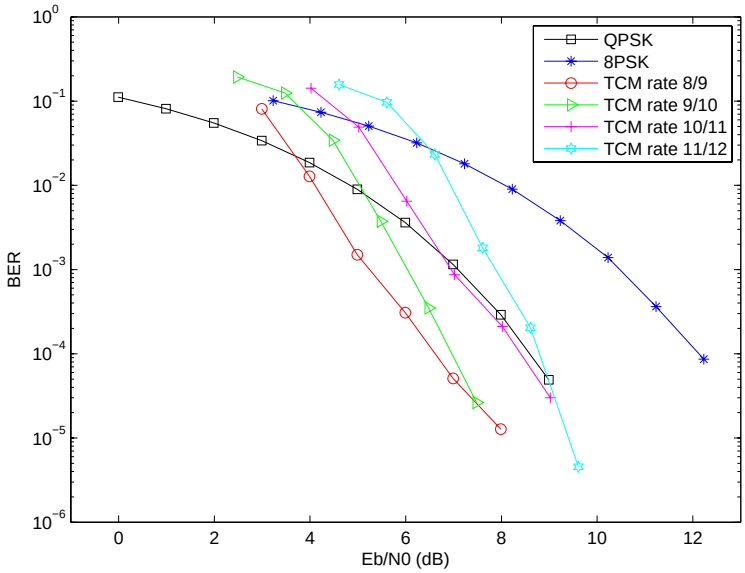


FIGURE 3.1: BER performance of the 4-D 8PSK TCM system.

3.2. Encoder Design

Regardless of the modulation rate, the 4-D 8PSK TCM encoder uses the same convolutional code shown in Fig. 3.2, while the constellation mapper varies from case to case, which is shown through Eq. (3.1) to Eq. (3.4). A complete understanding of partitioning multi-dimensional constellation points requires extensive mathematic background, which is beyond the scope of this dissertation. Interested readers may refer to the *Appendix* or [8] for more information.

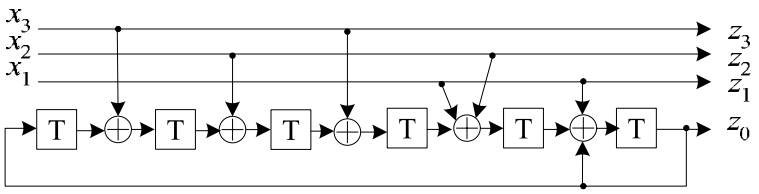


FIGURE 3.2: Rate-3/4 convolutional encoder.

$R_{eff} = 2.0$ bits/T, Rm=8/9:

$$\begin{pmatrix} Z_0 \\ Z_1 \\ Z_2 \\ Z_3 \end{pmatrix} = \left\{ (4x^{(8)} + 2^{(5)} + x^{(1)}) \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} + 4 \begin{pmatrix} 0 \\ x^{(7)} \\ x^{(6)} \\ x^{(7)} + x^{(6)} + x^{(4)} \end{pmatrix} + 2 \begin{pmatrix} 0 \\ x^{(3)} \\ x^{(2)} \\ x^{(3)} + x^{(2)} + x^{(0)} \end{pmatrix} \right\} (mod 8), \tag{3.1}$$

$R_{eff} = 2.25$ bits/T, Rm=9/10:

$$\begin{pmatrix} Z_0 \\ Z_1 \\ Z_2 \\ Z_3 \end{pmatrix} = \left\{ (4x^{(9)} + 2^{(6)} + x^{(2)}) \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} + 4 \begin{pmatrix} 0 \\ x^{(8)} \\ x^{(7)} \\ x^{(8)} + x^{(7)} + x^{(5)} \end{pmatrix} + 2 \begin{pmatrix} 0 \\ x^{(4)} \\ x^{(3)} \\ x^{(4)} + x^{(3)} + x^{(1)} \end{pmatrix} + \begin{pmatrix} 0 \\ x^{(0)} \\ 0 \\ x^{(0)} \end{pmatrix} \right\} (mod 8), \tag{3.2}$$

$R_{eff} = 2.5$ bits/T, Rm=10/11:

$$\begin{pmatrix} Z_0 \\ Z_1 \\ Z_2 \\ Z_3 \end{pmatrix} = \left\{ (4x^{(10)} + 2^{(7)} + x^{(3)}) \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} + 4 \begin{pmatrix} 0 \\ x^{(9)} \\ x^{(8)} \\ x^{(9)} + x^{(8)} + x^{(6)} \end{pmatrix} + 2 \begin{pmatrix} 0 \\ x^{(5)} \\ x^{(4)} \\ x^{(5)} + x^{(4)} + x^{(2)} \end{pmatrix} + \begin{pmatrix} 0 \\ x^{(1)} \\ x^{(0)} \\ x^{(1)} + x^{(0)} \end{pmatrix} \right\} (mod 8), \tag{3.3}$$

$R_{eff} = 2.75$ bits/T, Rm=11/12:

$$\begin{pmatrix} Z_0 \\ Z_1 \\ Z_2 \\ Z_3 \end{pmatrix} = \left\{ (4x^{(11)} + 2^{(8)} + x^{(4)}) \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} + 4 \begin{pmatrix} 0 \\ x^{(10)} \\ x^{(9)} \\ x^{(10)} + x^{(9)} + x^{(7)} \end{pmatrix} + 2 \begin{pmatrix} 0 \\ x^{(6)} \\ x^{(5)} \\ x^{(6)} + x^{(5)} + x^{(3)} \end{pmatrix} + \begin{pmatrix} 0 \\ x^{(2)} \\ x^{(1)} \\ x^{(2)} + x^{(1)} + x^{(0)} \end{pmatrix} \right\} (mod 8), \tag{3.4}$$

We can design 4 different constellation mappers and enable the right one according to its modulation rate. However, this approach is very inefficient in terms of hardware area. By carefully examining the common features of these mapping equations, we have developed a universal encoder architecture shown in Fig. 3.3, where the router block takes 4 bits input and outputs 4 bits in a proper order based on the given data rate, and the MUX maps the proper data to the input of the convolutional encoder. This architecture only requires one single constellation mapper to implement Eq. (3.4). The structure of the mapper is shown in Fig. 3.4.

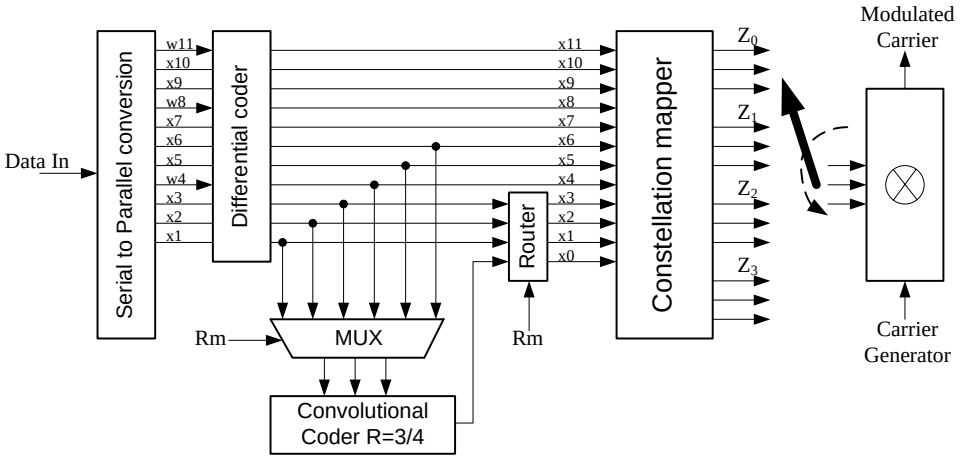


FIGURE 3.3: Encoder of the 4-D 8PSK TCM system.

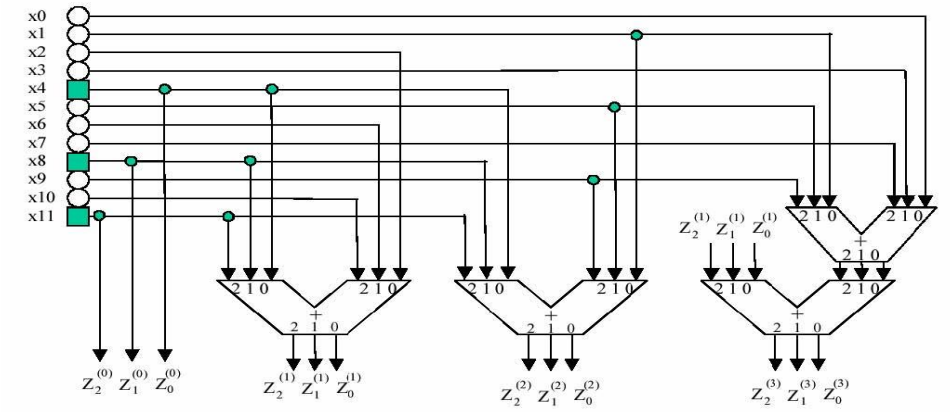


FIGURE 3.4: Constellation mapper.

3.3. Decoder Design

Before providing the details of decoder design, it is necessary to evaluate the influence from signal quantization. Fig. 3.5 shows the simulation results with various quantization levels for the 4-D 8PSK TCM signal with a modulation rate of 11/12. The signal-to-noise ratios (SNRs) to achieve a BER of 10^{-4} for the floating-point case, 8-bit

quantization, 7-bit quantization, and 6-bit quantization are 12.88 dB, 13.05 dB, 13.20 dB and 13.82 dB, respectively. In principle, on the premise of an acceptable performance, the length of bits for the signal quantization should be as small as possible. The 7-bit quantization is adopted in our design, since the performance loss is less than 0.5 dB compared with the floating-point case. When the frond-end analog-to-digital converter (ADC) can not provide so many bits per symbol, a 6-bit quantization is still usable. Quantization with fewer than 6 bits should not be considered due to its poor performance as shown in Fig. 3.5.

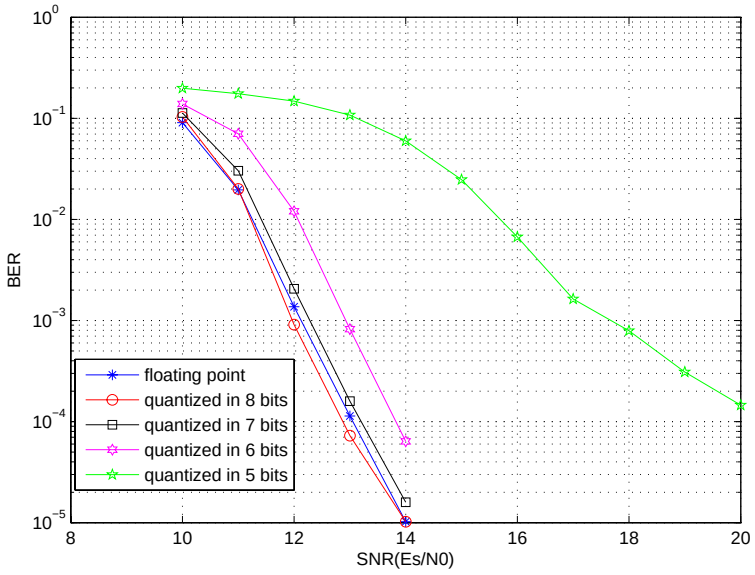


FIGURE 3.5: Influence of the signal quantization.

3.3.1 Literature review

The block diagram of the decoder for 4-D 8PSK TCM signal is shown in Fig. 3.6, where the delay chain is used to buffer the paths associated with each BM generated by the TMU. The "path" refers to the 12 bits of information from the 4 dimensions of 8PSK signals. Since there are only 16 BMs [9] in this case, using a 4-bit index is more

efficient than using the 12-bit path information inside the VD. When the VD outputs the decoded index, the corresponding path will be sent to the de-mapping unit (DMU) and differential decoder (DFD) for further processing. Implementation of the DMU and DFD is straightforward; the most challenging and hardware-consuming parts are the TMU and VD.

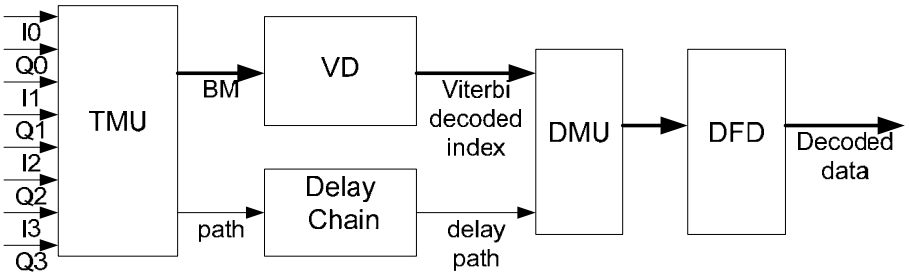


FIGURE 3.6: 4-D 8PSK TCM decoder diagram.

Parallel paths in trellis transition distinguish TCM schemes with conventional convolutional encoding. The TMU is used to find the optimal path among all the parallel ones as the BM to the Viterbi decoder. As indicated in Fig. 3.3, for any encoded sequence "$x3x2x1x0$", there are 256 different 4-D 8PSK signal groups (combinations of $x11 - x4$) which indicate 256 parallel paths for each state transition in the trellis diagram. Therefore, each BM has 256 candidates. A straightforward implementation of TMU leads to very high computational complexity, which is even higher than that of the ACSU in the Viterbi decoder. The structure of the auxiliary trellis has been presented in [9] to remove the redundant operations in TMU by sharing intermediate results among all the BM candidates. However, the computational complexity is still considerably high. A new architecture is proposed in this chapter to further remove the redundant computations.

For VD implementation, research efforts have focused on ACSU and SMU. ACSU implementation is critical because the feedback loop makes it the bottleneck for high speed applications. Additionally, as the constraint length K increases, the number of

states and computation in the ACSU increase exponentially. Several algorithms such as M-algorithm [28] and T-algorithm [29], [30] have been proposed to reduce the computation in ACSU. In the M-algorithm, this is achieved by keeping M optimal states and purging the rest. In the T-algorithm, a threshold T is set and the difference between each PM and the optimal one is calculated. The algorithm compares the differences with T; only the states with a difference less than T survive and are used for the calculation in the next cycle. Both methods have the potential for significant power savings. In practical implementation, however, T-algorithm is more popular than M-algorithm for two reasons: 1) it can adaptively adjust the number of purged states according to run-time channel conditions and 2) it has lower computational complexity since T-algorithm only needs to find out the optimal PM (maximum or minimum value, depending on different implementations), while M-algorithm involves a sorting process for $2^{(k-1)}$ values. However, T-algorithm still requires the search of the optimal PM in the ACS loop. The extra comparison operation will affect the clock speed of the entire design. Although it is possible to implement a fully parallel structure for the comparison operation, it will cause a significant hardware overhead. Recently, several novel VLSI architectures are proposed to remove or to reduce the loop latency with reasonable hardware overhead. For example, in [31], instead of finding the exact optimal PM, an estimated value is obtained. Then, a compensation scheme outside the loop is used to adjust the estimated optimal PM.

There are two basic schemes for SMU implementation: the register-exchange (RE) scheme [32] and the trace-back (TB) scheme [33]. Because of the maximum likelihood (ML) path searching in SMU, VD usually causes a decoding latency proportional to the constraint length K. The RE scheme is well suited for low-latency, high-speed applications, while the TB scheme consumes less power. Although there exists extensive work on reducing the latency of the TB scheme [4, 34–36], none of them could achieve the latency of the RE scheme even at the expense of a huge power consumption overhead. Another

advantage of using RE scheme is that T-algorithm on PM can be efficiently exploited in RE design. Thus, the RE scheme is adopted in our design.

The algorithms discussed above are general solutions for VDs. TCM schemes usually employ high-rate codes (e.g., the rate-3/4 codes in our case), and the unique properties of high-rate codes can be exploited to construct more efficient architectures for the corresponding VD.

3.3.2 Low-complexity TMU design

There are two common ways to design TMU: 1) ROM based approach, and 2) arithmetic (on-line) computation based solution. In order to reduce hardware as well as to pipeline the computation unit for high clock speeds, the second approach is adopted in our design. Exploiting the auxiliary trellis [9], we design two efficient methods to reduce the computational complexity.

A. Simplified computation of Euclidian metrics

Given a received 8PSK symbol (I_r, Q_r) and a target constellation point A: (I_s, Q_s) $(s = 0, 1, 2, \cdots, 7$ for 8 PSK signals), the Euclidian distance is computed as:

$$\begin{aligned} d_s &= (I_r - I_s)^2 + (Q_r - Q_s)^2 \\ &= (I_r^2 + I_s^2 + Q_r^2 + Q_s^2) - 2I_rI_s - 2Q_rQ_s. \end{aligned} \tag{3.5}$$

Note that for all d_s, ($I_r^2 + I_s^2 + Q_r^2 + Q_s^2$) remains the same. Finding the minimum of $(d_0, d_1, d_2..., d_7)$ (the closest constellation point to the received signal) is equivalent to finding the maximum of $(I_rI_0 + Q_rQ_0, I_rI_1 + Q_rQ_1, I_rI_2 + Q_rQ_2, ..., I_rI_7 + Q_rQ_7$). Thus, we only need to consider the following distance:

$$d'_s = I_rI_s + Q_rQ_s. \tag{3.6}$$

Furthermore, because all the possible constellation points are equally spaced on a circle with the phase of $s\pi/4, s = 0, 1, 2...7$, it has been found in [9] that I_s and Q_s satisfy the

following conditions

$$\begin{aligned} I_s &= -I_{(s+4)mod8} \\ Q_s &= -Q_{(s+4)mod8}. \end{aligned}$$

Hence, only four sets of Eq. (3.6) need to be calculated. The Euclidean metrics C_i ($i = 0, 1, 2, 3$) are computed as

$$\begin{aligned} C_0 &= |d'_0| = |I_r| \\ C_1 &= |d'_1| = |(I_r + Q_r) \times 0.707| \\ C_2 &= |d'_2| = |Q_r| \\ C_3 &= |d'_3| = |(Q_r - I_r) \times 0.707|. \end{aligned} \tag{3.7}$$

It is clear from Eq. (3.7) that only 2 multiplications are required.

B. Sharing of branch metric computations

In the 4-D 8PSK TCM decoder, 16 BMs are selected from 4096 (*i.e.*, 8^4) candidates every cycle. From Fig. 1, the 16 BMs are related to the 16 combinations of $x3, x2, x1$ and $x0$. Let BM 0000 denote the BM related to $(x3, x2, x1, x0) = (0, 0, 0, 0)$; BM 0001 denote the BM related to $(x3, x2, x1, x0) = (0, 0, 0, 1)$, and so on. Each candidate is the sum of the 4 Euclidian metrics of the received signal set (Z_0, Z_1, Z_2, Z_3). By applying the above simplified Euclidian metrics to BMs, the number of candidates is reduced from 4096 to 256 (*i.e.*, 4096/16), since the number of Euclidian metrics from each signal set is reduced by half. Now each BM is selected among only 16, rather than 256, candidates. A straightforward implementation requires 3 serial addition stages to compute the candidates and several comparison steps to find the BMs. Since the results at each addition stage can be shared for further computation, using auxiliary trellis [9] can eliminate redundant additions.

Fig. 3.7 is an example that illustrates the process of computing BM 0000. From the left to the right, C_i $(i = 0, 1, 2, 3)$ in each step denote the Euclidean metrics of the received symbols from Z_0, Z_1, Z_2 and Z_3, respectively. The same rule applies to all equations and figures in the rest of the Chapter. The 16 candidates for BM 0000 are:$C_0 + C_0 + C_0 + C_0, C_1+C_1+C_1+C_1, C_0+C_0+C_2+C_2, C_1+C_1+C_3+C_3, C_0+C_2+C_0+C_2, C_1+C_3+C_1+C_3, C_0+C_2+C_2+C_0, C_1+C_3+C_3+C_1, C_2+C_2+C_2+C_2, C_3+C_3+C_3+C_3, C_2+C_2+C_0+C_0, C_3+C_3+C_1+C_1, C_2+C_0+C_2+C_0, C_3+C_1+C_3+C_1, C_2+C_0+C_0+C_2, C_3+C_1+C_1+C_3$.

By using the auxiliary trellis, the total number of additions is reduced from 768 (*i.e.*, $3 \times 16 \times 16$) with the straightforward implementation to 336 (*i.e.*, $4^2 + 4^3 + 4^4$). However, the number of comparisons remains the same. In this work, we propose a new approach that further reduces the number of additions by half. In addition, about 2/3 of the comparisons are removed.

The key idea of the improvement is as follows. Let us again take BM 0000 as an example. In the auxiliary trellis approach as shown in Fig. 3.7, to compute each candidate branch metric, 3 serial addition stages are performed to compute 16 candidates in parallel. Then, one survival BM is chosen from the 16 candidates. A careful examination of the added Euclidian distances reveals that in the last addition stage, there are only 4 different values to be added onto the 16 candidates as indicated in Table 3.1.

This property can be exploited to reduce the number of additions in the last stage by rearranging the calculation flow in Fig. 3.7. In the proposed approach, the 16 candidates are divided at the end of the second addition stage into 4 groups, which is represented by $G_k, k = 0, 1, 2, 3$, in Table 3.1. The 4 candidates in each group are supposed to add the same Euclidian metric of Z_3 later. Before the last addition operation, one survival candidate is chosen from each group. Then in the last addition stage, only 4 additions, instead of 16, are required. Based on the two-step comparison method, an algorithmic strength reduction scheme is introduced, which significantly reduces the needed comparisons. As

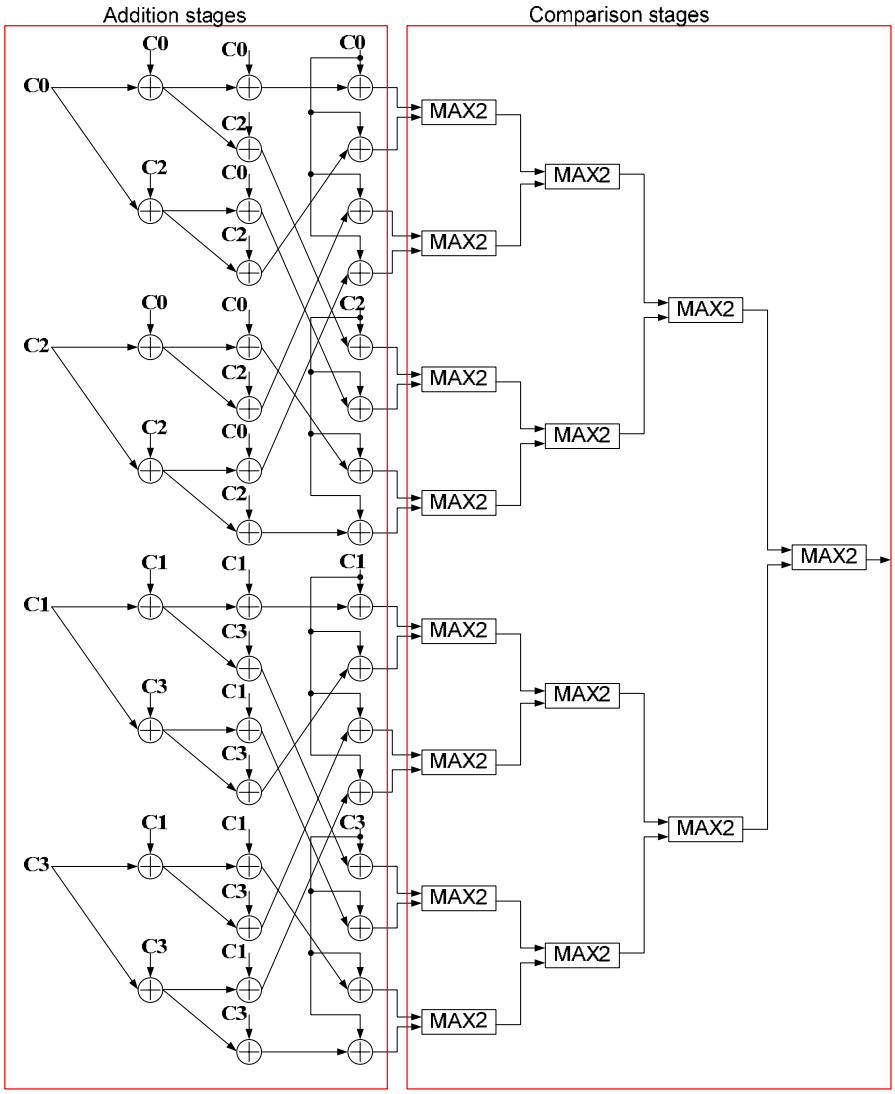


FIGURE 3.7: An example of auxiliary trellis to simplify the BM calculation.

TABLE 3.1: Grouping of the candidates for BM 0000

	Z_0	Z_1	Z_2	Z_3		Z_0	Z_1	Z_2	Z_3
G_1	C_0	C_0	C_0	C_0	G_2	C_1	C_1	C_1	C_1
	C_0	C_2	C_2			C_1	C_3	C_3	
	C_2	C_2	C_0			C_3	C_3	C_1	
	C_2	C_0	C_2			C_3	C_1	C_3	
G_3	C_0	C_0	C_2	C_2	G_4	C_1	C_1	C_3	C_3
	C_0	C_2	C_0			C_1	C_3	C_1	
	C_2	C_2	C_2			C_3	C_3	C_3	
	C_2	C_0	C_0			C_3	C_1	C_1	

shown in Fig. 3.8, the computation procedure is rearranged: two serial comparison operations are inserted between the second and the third addition operations. Furthermore, the survival candidates after the comparison step 1 can be shared by 4 BM computations (*i.e.*, BM 0000, BM 0001, BM 1000 and BM 1001 share the same survival candidates, as listed in Table 3.1 through Table 3.4).

The other 12 BMs have the same property and can be categorized into 3 clusters; in each cluster, 4 BMs share the survivor candidates after the comparison step 1. The total number of additions of the proposed architecture is $4^2 + 4^3$ from the first two addition stages plus 4×16 from the third addition stage. Meanwhile, the comparison operations are also reduced. The computational complexities of the 3 methods are summarized in Table 3.5.

The same structure can be extended to the cases of Rm = 10/11 and Rm = 9/10, in which the same method could be employed. For the case of Rm = 8/9, since there are only two candidates for each BM, the structure based on the conventional auxiliary trellis [9] is as efficient as the proposed approach. A detailed comparison of computational

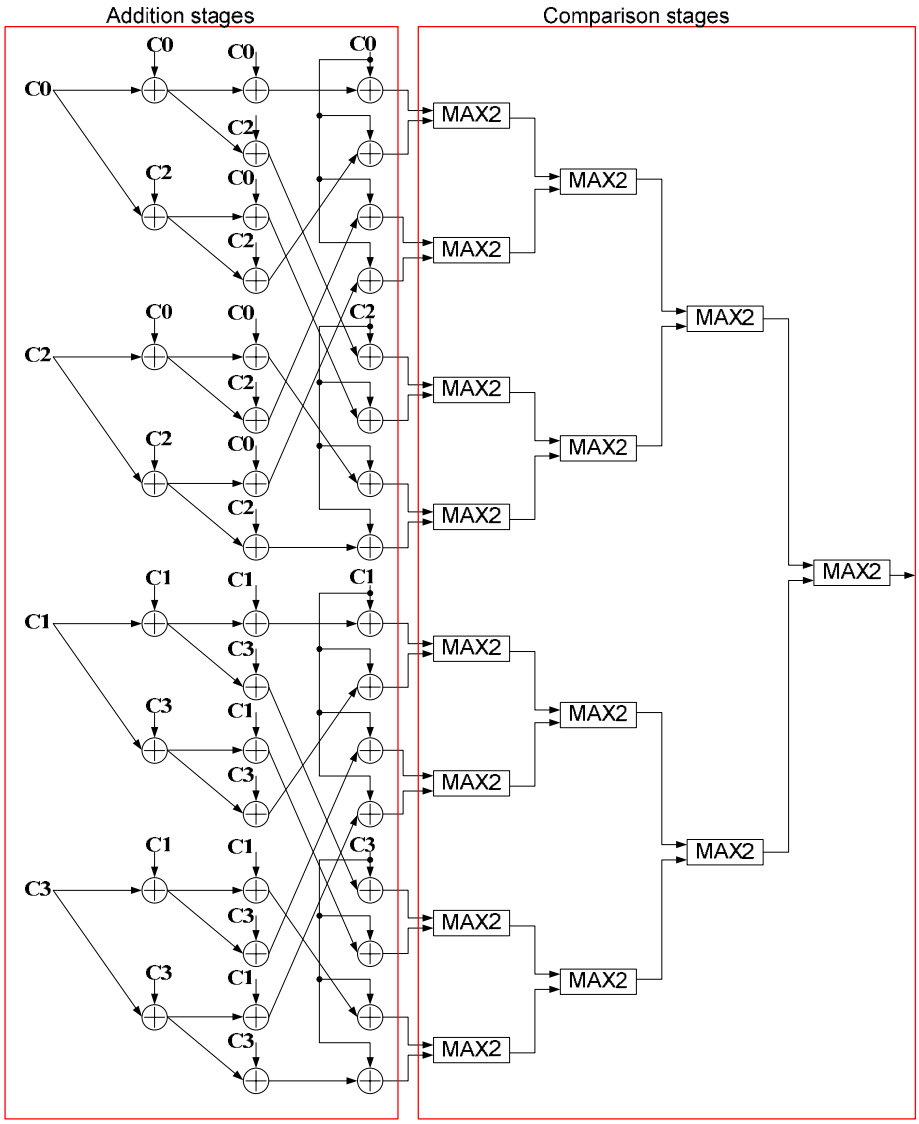


FIGURE 3.8: Computation of BM 0000 using the proposed 2-step comparison method.

TABLE 3.2: Grouping of the candidates for BM 0001

	Z_0	Z_1	Z_2	Z_3		Z_0	Z_1	Z_2	Z_3
G_1	C_0	C_0	C_0	C_1	G_2	C_1	C_1	C_1	C_2
	C_0	C_2	C_2			C_1	C_3	C_3	
	C_2	C_2	C_0			C_3	C_3	C_1	
	C_2	C_0	C_2			C_3	C_1	C_3	
G_3	C_0	C_0	C_2	C_3	G_4	C_1	C_1	C_3	C_0
	C_0	C_2	C_0			C_1	C_3	C_1	
	C_2	C_2	C_2			C_3	C_3	C_3	
	C_2	C_0	C_0			C_3	C_1	C_1	

complexity is provided in Table 3.6.

C. FPGA implementation result

In our design, the decoders for four different modulation rates are integrated into a reconfigurable 4-D 8PSK TCM decoders. The entire design is first modeled in Matlab for performance simulation and finite wordlength analysis. As we analyzed before, quantization of 7 bits is a good choice for maintaining an acceptable BER performance. Therefore, the 4-D 8PSK TCM decoder with 7-bit quantized input is implemented with Verilog HDL, simulated in Modelsim and synthesized using Xilinx Virtex-4 FPGA (XC4VLX160). The TMUs with both the conventional auxiliary trellis structure and the proposed 2-step comparison architecture are implemented. Both TMUs are designed to have the capability to deal with 4 different modulation rates. In addition, 3 pipeline stages are used for both cases to increase the clock speed. Synthesis results are shown in Table 3.7, where the utilization of the device on FPGA is also shown in percentage.

Although we concluded above that the proposed architecture could reduce the computational complexity by 67%, due to the similar number of registers used for pipelining

TABLE 3.3: Grouping of the candidates for BM 1000

	Z_0	Z_1	Z_2	Z_3		Z_0	Z_1	Z_2	Z_3
G_1	C_0	C_0	C_0	C_2	G_2	C_1	C_1	C_1	C_3
	C_0	C_2	C_2			C_1	C_3	C_3	
	C_2	C_2	C_0			C_3	C_3	C_1	
	C_2	C_0	C_2			C_3	C_1	C_3	
G_3	C_0	C_0	C_2	C_0	G_4	C_1	C_1	C_3	C_1
	C_0	C_2	C_0			C_1	C_3	C_1	
	C_2	C_2	C_2			C_3	C_3	C_3	
	C_2	C_0	C_0			C_3	C_1	C_1	

the circuits for both cases, Table 3.7 only shows half of the hardware reduction.

3.3.3 *T*-algorithm on BMs

A. Applying T-algorithm to BMs

In the 4-D 8PSK TCM system, signals with different modulation rates use the same convolutional code as shown in Fig. 3.2.

Unlike low-rate codes (e.g., 1/2, 1/3) that have 2 incoming paths for each state, high-rate convolutional codes generate many more paths in state transitions (8 in the our case due to 3 input bits). When applying Viterbi decoding on these codes, more BMs are involved (16 in the above case because of the 4 output bits), which increase the computational complexity of the decoder. For example, the computational complexity of the Viterbi decoder for decoding the rate-3/4 convolutional code employed in this 4D TCM system is equivalent to the Viterbi decoder for a rate-1/2 convolutional code with 256 states. Therefore, it is necessary to use some low-power scheme in the design. Motivated by the conventional T-algorithm, we found that eliminating redundant additions is an

TABLE 3.4: Grouping of the candidates for BM 1001

	Z_0	Z_1	Z_2	Z_3		Z_0	Z_1	Z_2	Z_3
G_1	C_0	C_0	C_0	C_3	G_2	C_1	C_1	C_1	C_0
	C_0	C_2	C_2			C_1	C_3	C_3	
	C_2	C_2	C_0			C_3	C_3	C_1	
	C_2	C_0	C_2			C_3	C_1	C_3	
G_3	C_0	C_0	C_2	C_1	G_4	C_1	C_1	C_3	C_2
	C_0	C_2	C_0			C_1	C_3	C_1	
	C_2	C_2	C_2			C_3	C_3	C_3	
	C_2	C_0	C_0			C_3	C_1	C_1	

TABLE 3.5: Computational complexity comparision for RM=11/12 case

Methods	Additions	Comparisons
Straightforward	768	240
Auxiliary trellis	336	240
2-step comparison	144	96

effective way to reduce computational complexity. In our case, since the number of BMs is so large, BMs also play a very important role in determining the overall complexity. Since PMs and BMs are evenly distributed in ACSU calculations, purging a BM is equivalent to purging 4 PMs in terms of equivalent number of additions being eliminated. Therefore, the T-algorithm could be applied on BMs, instead of PMs. The process is the same as in the conventional T-algorithm applied on PMs: first, finding the optimal BM - the maximum value of all 16 BMs in our case - and setting up a threshold; then, comparing the difference between each BM and the maximum value. If the difference is larger than the threshold, the corresponding BM is purged, which means that in the ACSU, the additions involving

TABLE 3.6: Computational complexity comparision for all cases

Rm		8/9	9/10	10/11	11/12
Straightforward	Additions	96	192	384	768
implement	Comparisons	16	48	112	240
Conventional	Additions	56	112	208	336
auxiliary trellis	Comparisons	16	48	112	240
Proposed 2-step	Additions	56	80	112	144
comparison	Comparisons	16	32	64	96

TABLE 3.7: Synthesis results for TMU

Xc4vlx160-12ff1148 Virtex 4, speed scale 12		
	Number of 4-input LUT	Number of Slice Flip-flop
Auxiliary trellis	14686(10%)	3939(2%)
2-step comparison	7912(5%)	1280(0%)

the purged BMs are not performed.

The benefit of applying T-algorithm on BMs is obvious. In our case, finding the maximum value of 16 BMs is more efficient than searching for the maximum value of 64 PMs, not only because the number of BMs is less than the number of PMs, but also due to the fact that each BM is the sum of the Euclidian metrics from 4 signal sets. The maximum value of BMs is equivalent to the sum of the maximum Euclidian metrics from each signal set. Finding 4 maximum values and then adding them is much more convenient than finding the maximum value of 16 arbitrary data. Additionally, the processes of finding the maximum value of BMs and calculating 16 BMs could take place in parallel, since the maximum BM is derived directly from the Euclidian metrics. This property can be

explored to eliminate the latency caused by the process of calculating the maximum value in a normal case which needs to obtain the values of all BMs first.

A more important advantage of the proposed method is the improvement of clock speed. The computation of the BMs is a feed-forward process that can be theoretically pipelined to achieve a clock speed as high as needed, and there is no additional process or calculation in the ACS loop. Comparing Fig. 2.3, Fig. 3.9 and Fig. 3.10, it is seen that the VD with T-algorithm applied on BMs maintain almost the same clock speed as the regular VD in Fig. 2.3.

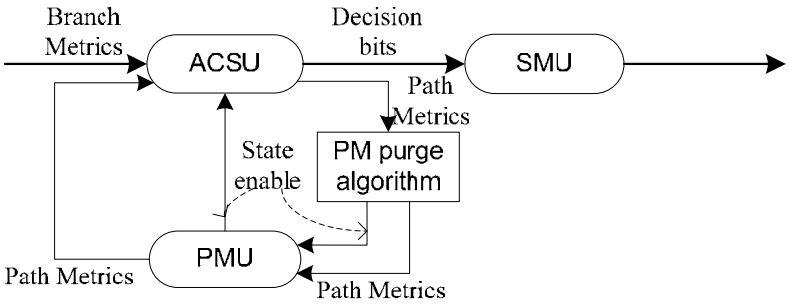


FIGURE 3.9: Functional diagram of a Viterbi decoder with T-algorithm on PM.

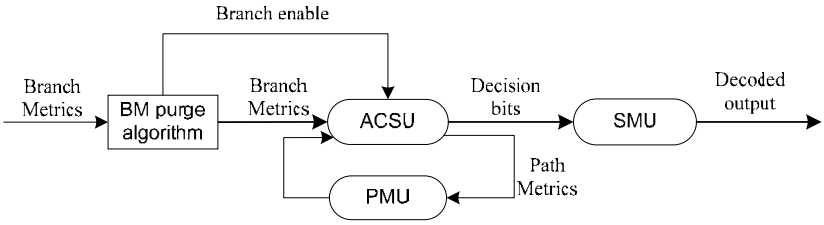


FIGURE 3.10: Functional diagram of a Viterbi decoder with T-algorithm on BM.

We have pointed out in Section 3.3.1 that conventional T-algorithm applied on PM could perform well for the RE scheme. When a state is purged in ACSU, the corresponding registers in SMU are not updated; thus, the power consumption is reduced. Generally, the scheme can be implemented by clock gating. For T-algorithm on BMs, when the BMs

involved for calculating a new PM are all purged and the PM can not get a valid value, the corresponding state can be treated as "purged". Again, the registers associated with the "purged" state will not be updated in the SMU.

B. Simulation results

It is necessary to examine the BER performance and computational complexity of the conventional VD with T-algorithm applied on PMs and the proposed VD with T-algorithm applied on BMs. Fig. 3.11 and Fig. 3.12 show the BER performance of these two methods for the case of Rm=11/12. When the threshold drops to below 0.3, a dramatic performance loss for both cases is observed; at a threshold of 0.3, the BER performance of the proposed scheme is slightly better than that of the conventional scheme with T-algorithm applied on PMs.

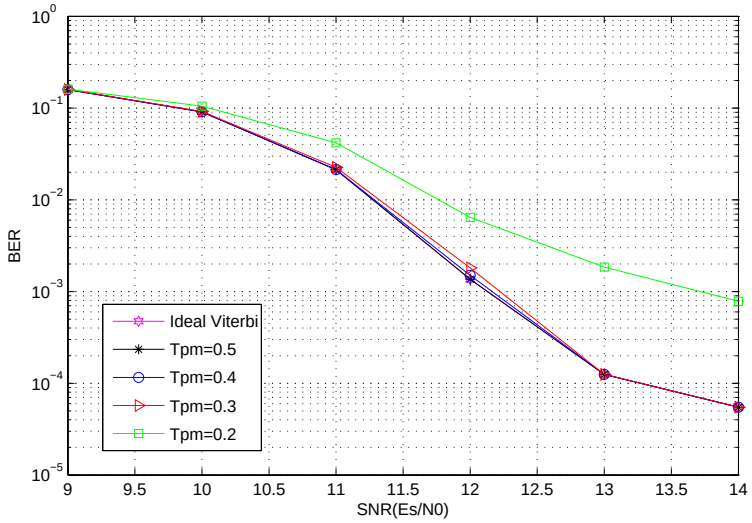


FIGURE 3.11: BER performance of T-algorithm on PMs.

The corresponding average computational complexity for the case with a threshold of 0.3 is shown in Fig. 3.13. The average computational complexity refers to the number of additions in each cycle. For a regular Viterbi decoder, the computational complexity is

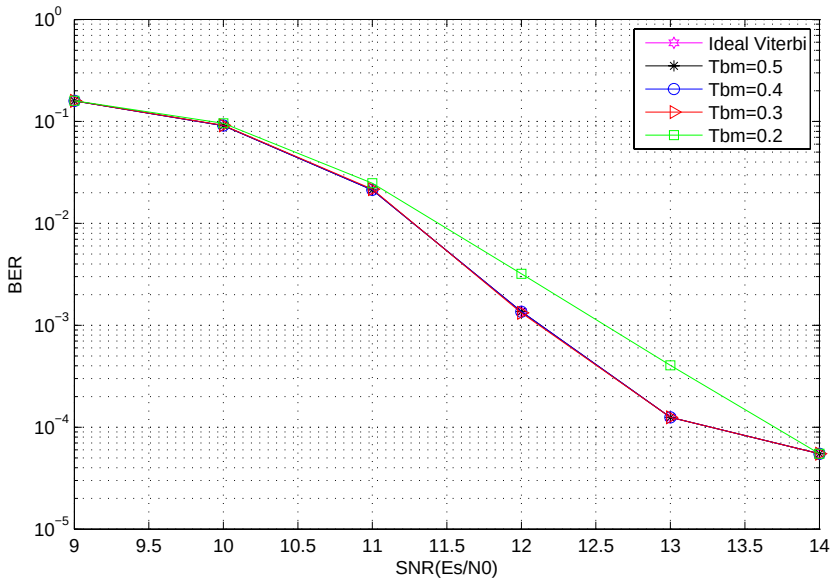


FIGURE 3.12: BER performance of T-algorithm on BMs.

64 (number of states) $\times$ 8 (number of input BMs per state) = 512. From the simulation result shown in Fig. 3.13, we observe that, as the SNR changes, the number of purged computation changes accordingly in the case of T-algorithm on PMs, whereas for the case of T-algorithm on BMs, the number of purged computation remains almost the same regardless of the channel condition. The performance of T-algorithm on BMs behaviors more like the conventional M-algorithm, rather than a T-algorithm. Therefore, when SNR is low, the computational complexity of the proposed scheme is lower than that of the conventional scheme. In the high-SNR region, however, the conventional scheme has a lower complexity than the proposed one. Although the proposed T-algorithm on BMs can not save as many computations as the conventional T-algorithm on PMs in the high SNR region, it still has the advantages of lower complexity, higher throughput, and better BER performance.

C. FPGA implementation result

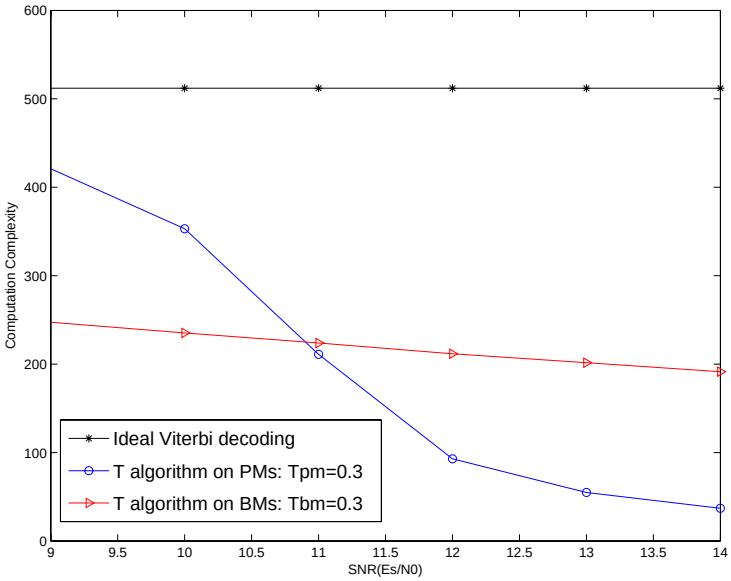


FIGURE 3.13: Computational complexities of T-algorithm on PMs and BMs.

Since the new scheme is proposed only for ACSU in VD, it can be combined with any type of SMU. To demonstrate the improvements of the proposed T-algorithm on BMs, ACSUs with different algorithms are synthesized. The results are shown in Table 3.8.

TABLE 3.8: Synthesis results for ACSU with several schemes

Xc4vlx160-12ff1148 Virtex 4, speed scale 12			
	Number of 4-input LUT	Number of Slice Flip-flop	Max clock speed(MHz)
Full trellis	18522(13%)	576(0%)	103.8
T-algorithm on PMs	23326(17%)	576(0%)	42.5
T-algorithm on BMs	18626(13%)	576(0%)	102.1

In Table 3.8, "Full trellis" refers to the conventional Viterbi algorithm without any state-reducing scheme, T-algorithm on PMs is the conventional T-algorithm in [31],

and T-algorithm on BMs is the first proposed scheme. Compared with the "Full trellis" case, T-algorithm on BMs requires a slightly more hardware, while the conventional T-algorithm on PMs leads to about 30% hardware overhead. Also, the the proposed T-algorithm on BMs can keep nearly the same clock speed as the "Full trellis" VD, which is more than twice of that for VD employing T-algorithm on PMs.

3.3.4 Hybrid T-algorithm

From Fig. 3.9 and Fig. 3.10, we observe that implementing T-algorithm on BMs does not require any change to the architecture of the conventional T-algorithm on PMs. This feature makes it possible to combine the two schemes together and purge more computations in ACSU. The functional diagram of the resulting scheme is shown in Fig. 3.14, which is named as "hybrid T-algorithm". The details of the hybrid T-algorithm are described as follows. The PM at time slot n is calculated from the sum of PMs at time slot $n-1$ and BMs at time slot n as:

$$\begin{aligned} PM_n^j &= \max\Big(PM_{n-1}^{j1}+BM_n^{j1}; PM_{n-1}^{j2}+BM_n^{j2}; \\ &\quad PM_{n-1}^{j3}+BM_n^{j3}; PM_{n-1}^{j4}+BM_n^{j4}; \\ &\quad PM_{n-1}^{j5}+BM_n^{j5}; PM_{n-1}^{j6}+BM_n^{j6}; \\ &\quad PM_{n-1}^{j7}+BM_n^{j7}; PM_{n-1}^{j8}+BM_n^{j8}\Big), \end{aligned}$$

where the subscript denotes the time slot and the superscript j denotes the j^{th} state. Each PM is selected from 8 candidates. Conventional T-algorithm on PMs checks the purging flags for states. For example, if PM^{j1} is purged at time slot $n-1$, the ACSU will not perform $PM_{n-1}^{j1}+BM_n^{j1}$ at time slot n. T-algorithm on BMs applies the same rule on BMs. The hybrid T-algorithm checks the purging flags for both states and BMs. The addition operation is performed only if both the PM and BM involved in the operation are retained.

Let us again focus on the example of Rm=11/12. The BER performances of the

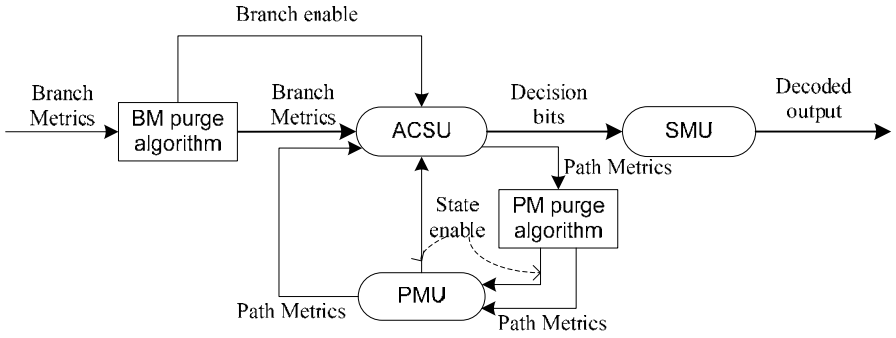


FIGURE 3.14: Functional diagram of a Viterbi decoder with hybrid T-algorithm.

hybrid T-algorithm with different parameters are shown in Fig. 13. Since the hybrid T-algorithm is developed based on conventional T-algorithm on PMs, we keep the threshold of 0.3 for PMs. The threshold for purging BMs is selected to be equal to or greater than 0.3 according to the simulation result in Fig. 3.12. It is observed from Fig. 3.15 that when T-bm (the threshold of BMs) is not less than 0.4, the BER performance of the hybrid T-algorithm on VD is identical to that of the conventional T-algorithm on PM; thus, the three curves nearly overlap in Fig. 3.15. When T-bm is equal to 0.3, a slight difference of less than 0.02 dB is observed in the region of SNR$<$ 11 dB.

Fig. 3.16 shows the computational complexity of proposed hybrid T-algorithm. For comparison, the complexity of T-algorithm on PMs and T-algorithm on BMs is also shown as the references. Compared with the conventional T-algorithm on PMs, the computational complexity is reduced by more than 50% when the T-bm is set to 0.3.

These simulation results indicate that with the hybrid T-algorithm more calculations can be eliminated while its BER performance is almost the same as the conventional T-algorithm on PMs. However, a simple straightforward combination of the conventional T-algorithm on PMs with the proposed T-algorithm on BMs will lose the advantage of speed of the latter.

To retain the advantage of speed, we need a novel architecture. Here we take

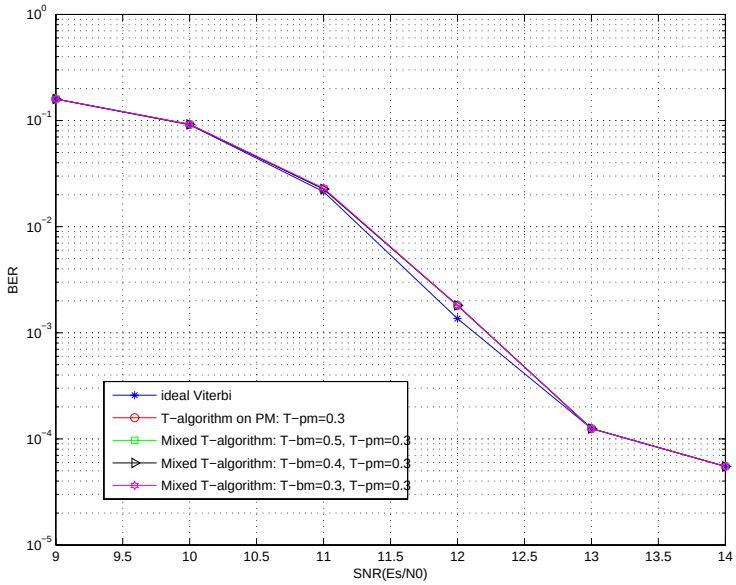


FIGURE 3.15: BER performance of hybrid T-algorithm.

advantage of the SPEC-T algorithm proposed in [31] to reduce the length of critical path. The key idea of SPEC-T algorithm is to use an estimated optimal PM value derived from the optimal BM value, instead of searching for the optimal PM in each cycle. Since the T-algorithm on BMs will generate an optimal BM anyway, very little additional logic is needed. The functional diagram of the resultant scheme is shown in Fig. 3.17. The optimal BM value is forwarded to the PM purging unit to calculate the estimated optimal PM value. For most of the time, the VD uses the estimated optimal PM value to perform T-algorithm on PM. Meanwhile, some compensation schemes are needed to adjust the estimated optimal value periodically. A straightforward compensation scheme is to find the real optimal PM in a number of cycles and compute the estimation error for next adjustment. The clock speed will not be affected by this searching process since it can be pipelined into several clock cycles outside the ACSU loop. However, When SPEC-T algorithm is adopted for decoding high-rate codes, it decreases the performance because

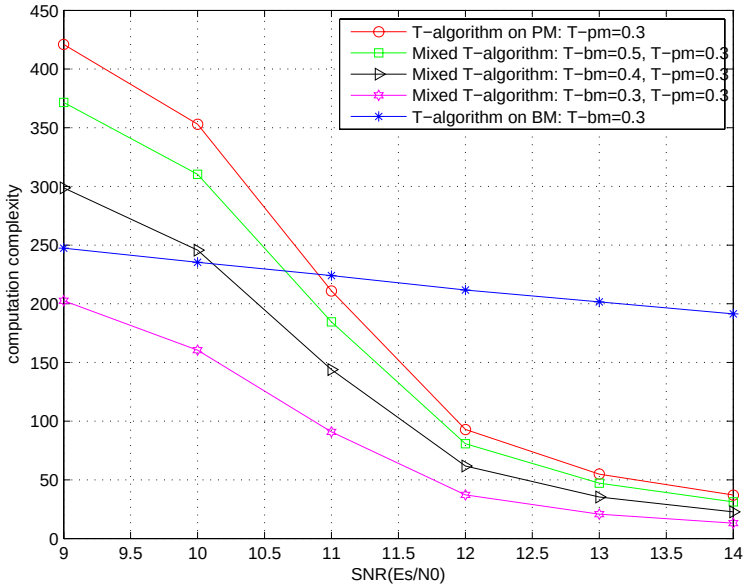


FIGURE 3.16: Computational complexities of hybrid T-algorithm.

of the error between the actual optimal PM and the estimated value. In the next chapter, a novel architecture is proposed to speed up the decoding process when T-algorithm on PMs is employed in a VD without sacrificing BER performance.

It is necessary to mention that, when the number of BMs is not very large (e.g., for a rate-1/2 code), purging BMs may cause considerable performance loss. Thus, the proposed T-algorithm on BMs and hybrid T-algorithm are only suitable for dealing with high-rate codes or codes that generate many BMs (e.g., codes of rate-1/3 or lower).

3.4. An FPGA implementation for the decoder

The 4-D 8PSK TCM decoder employs the proposed 2-step comparison architecture for TMU and the hybrid T-algorithm for the Viterbi decoder. The decoder is reconfigurable and can decode signals with 4 different modulation rates. Table 3.9 summarizes

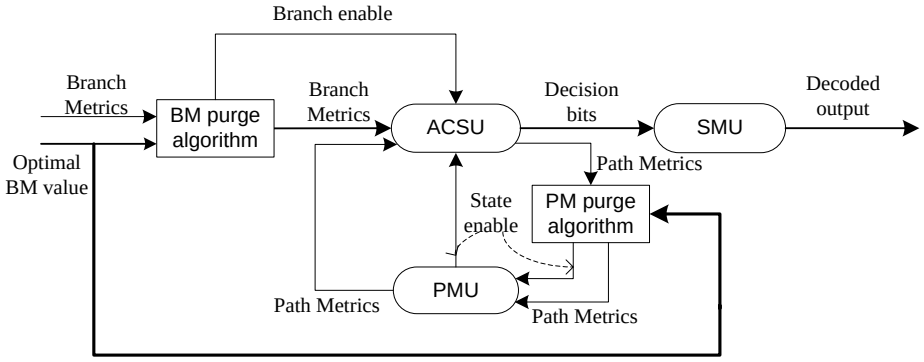


FIGURE 3.17: The hybrid T-algorithm Viterbi decoder with a SPEC-T algorithm.

the synthesis results.

TABLE 3.9: Synthesis results for TCM decoder

Xc4vlx160-12ff1148 Virtex 4, speed scale 12			
	4-input LUT	Slice Flip-flop	Max clock speed(MHz)
Number of devices	110804	14743	78.9
Utilization	80%	10%	–

Because of the heavy device utilization of the entire 4-D 8PSK TCM decoder, the delay due to wire routing is high. But it is still much faster than the conventional T-algorithm on PMs. The only work we found from the literature that is similar to our design is [9], in which the decoder for 8/9 modulation rate with auxiliary trellis structure for TMU and a full trellis VD is implemented on FPGA. Note that in the implementation in [9], the highest data rate reported is 460 Mbits/s. In our design, the data rate for the modulation rate of 8/9 is up to $78.9 \times 8 = 632.1$ Mbits/s, which is much faster than the work reported in [9].

3.5. Conclusion

In this chapter, we presented the design of a codec for a 4-D 8PSK TCM system. First, we developed a configurable encoder which shares the same convolutional encoder and constellation mapper with 4 cases of different modulation rates. Then we proposed an efficient architecture for 4-D 8PSK TCM decoders. Design efforts are focused on the two most complex parts of the decoder: TMU and VD. For TMU, the proposed architecture reduces the computational complexity to about 1/3 of an existing low complexity architecture presented in [9] while the BER performance is not compromised. For VD, we first proposed a new structure for T-algorithm applied on BMs. Compared with the conventional T-algorithm applied on PMs, the new scheme has the advantages of lower complexity and higher throughput. Based on the proposed T-algorithm on BMs and the conventional T-algorithm on PMs, we then designed a hybrid T-algorithm scheme. Simulation results show that the hybrid T-algorithm can further reduce more than half of the computations compared with the conventional T-algorithm while still maintaining the same BER performance. Finally, we proposed a new architecture for hybrid T-algorithm VD based on SPEC-T algorithm that can retain the speed advantage of the conventional scheme. Implementation results in FPGA validated the excellent performance of the proposed algorithms.

4. LOW-POWER VITERBI DECODER DESIGN

In this chapter, we will extend our research to a more general topic of Viterbi decoder design. Power consumption is a major issue in designing Viterbi decoders. Power could be reduced by employing puncturing schemes like T-algorithm or by appropriately organizing of SMU. In practical, however, the architecture optimization involves a trade-off among several design parameters. For example, T-algorithm would in general reduce the decoding speed; low-power SMU would increase the decoding latency. In this Chapter, we first propose a new architecture for Viterbi decoders using T-algorithm. This architecture reduces the power consumption while maintaining a high clock speed and good BER performance. Then, we discuss the optimal memory organization for SMU, reaching a solution where both the power consumption and the decoding latency are moderate.

4.1. A High-Speed Low-power Viterbi Decoders for High-Rate Codes

In this section, we propose an efficient architecture based on pre-computation for Viterbi decoders incorporating T-algorithm. The new architecture greatly shortens the long critical path introduced by the conventional T-algorithm. We provide a design example to demonstrate the significant improvement in clock speed with a small computation overhead while maintaining a good BER performance and low-power.

4.1.1 Literature review

Nowadays, it has been a common practice to use T-algorithm to prune significant portions of the trellis states to dramatically reduce power consumption. While T-algorithm has been proved to generally maintain a good decoding performance, the operation of searching for the best path metrics in the add-compare-select loop significantly

limits the clock speed.

To achieve a high speeds, it is possible to implement a $2^{(k-1)}$-input comparator with a fully parallel architecture. However, it will cause significant hardware overhead, which conflicts with the design goal of less computation and low power consumption. As we mentioned in Chapter 3, new schemes like SPEC-T for high speed T-algorithm implementation have been proposed in [31,37], which use the estimated (or approximated) optimal PM derived from the optimal BM, instead of finding the accurate value in each cycle. Also, compensation schemes should be employed in these methods to ensure that the estimated value is not drifting too far away from the accurate one.

These methods combined with compensation algorithms have shown good results on low-rate ($1/R,\ R = 2, 3, 4 \cdots$) codes. However, when these schemes are applied on high-rate ($(R-1)/R, R > 2$) codes, serious problems that significantly affect the BER performance arise. Moreover, these methods usually require extra parameters for the compensation algorithms, which can only be obtained from simulations and cannot be adjusted based on real-time channel information, making it less attractive in practical implementation.

We propose a new architecture for the Viterbi decoder using T-algorithm. Unlike existing works such as [31, 37], an accurate optimal PM is guaranteed to be found at each cycle and no extra parameters are needed. Since the optimal PM is accurate in the proposed architecture, the new architecture keeps the same BER performance as the conventional T-algorithm, and is well suited for high-rate codes.

4.1.2 The pre-computation algorithm

The basic idea of pre-computation is as follows. Consider a VD for a convolutional code with a constraint length of k, where each state receives p candidate paths. If the branch metrics are calculated based on the Euclidean distance, the optimal PM becomes the minimum value of all the PMs as shown in Eq. (4.1).

$$
\begin{aligned}
PM_{opt}(n) &= \min\left\{PM_0(n), PM_1(n), ..., PM_{2^k-1}(n), \right\} \\
&= \min\{\min[PM_{0,0}(n-1) + BM_{0,0}(n), \\
&\qquad PM_{0,1}(n-1) + BM_{0,1}(n), ..., \\
&\qquad PM_{0,p}(n-1) + BM_{0,p}(n)], \\
&\quad \min[PM_{1,0}(n-1) + BM_{1,0}(n), \\
&\qquad PM_{1,1}(n-1) + BM_{1,1}(n), ..., \\
&\qquad PM_{1,p}(n-1) + BM_{1,p}(n)], \\
&\quad ....., \\
&\quad \min[PM_{2^k-1,0}(n-1) + BM_{2^k-1,0}(n), \\
&\qquad PM_{2^k-1,1}(n-1) + BM_{2^k-1,1}(n), ..., \\
&\qquad PM_{2^k-1,p}(n-1) + BM_{2^k-1,p}(n)]\} \\
&= \min\{PM_{0,0}(n-1) + BM_{0,0}(n), \\
&\qquad PM_{0,1}(n-1) + BM_{0,1}(n), ..., \\
&\qquad PM_{0,p}(n-1) + BM_{0,p}(n), \\
&\qquad PM_{1,0}(n-1) + BM_{1,0}(n), \\
&\qquad PM_{1,1}(n-1) + BM_{1,1}(n), ..., \\
&\qquad PM_{1,p}(n-1) + BM_{1,p}(n), \\
&\qquad ....., \\
&\qquad PM_{2^k-1,0}(n-1) + BM_{2^k-1,0}(n), \\
&\qquad PM_{2^k-1,1}(n-1) + BM_{2^k-1,1}(n), ..., \\
&\qquad PM_{2^k-1,p}(n-1) + BM_{2^k-1,p}(n)\}. \qquad (4.1)
\end{aligned}
$$

The trellis butterflies for a VD usually have a symmetric structure. In other words, the states can be grouped into m clusters, where all the clusters have the same number

of states and all the states in the same cluster will be extended by the same BMs. Thus, Eq. (4.1) can be re-written as Eq. (4.2).

$$
\begin{aligned}
& PM_{opt}(n) \\
= \; & \min\{ \\
& \min(PMs(n-1)\text{in cluster 1}) + \min(BMs(n)\text{for cluster 1}), \\
& \min(PMs(n-1)\text{in cluster 2}) + \min(BMs(n)\text{for cluster 2}), \\
& ....., \\
& \min(PMs(n-1)\text{in cluster m}) + \min(BMs(n)\text{for cluster m}) \\
& \}. \qquad (4.2)
\end{aligned}
$$

The value of min(BMs) for each cluster can be easily obtained from the BMU, and min(PMs) at time $n-1$ in each cluster can be pre-calculated at the same time when the ACUS is updating the new PMs for time n. Therefore, a look-ahead architecture is formed here to calculate the accurate optimal PM at time n. Theoretically, when we continuously decompose PMs$(n-1)$, PMs$(n-2), \cdots$, the pre-computation scheme can be extended to q steps, where q is any positive integer that is less than n. Hence, $\text{PM}_{\text{opt}}(n)$ can be calculated directly from PM$(n-q)$ in q cycles.

Then, the VDs consider the inequality: PM-PM_{opt} > threshold, or PM > PM_{opt}+ threshold. The states satisfying the relationship will be purged.

For low-rate convolutional codes, pre-computation is usually inefficient because the number of states in the VD is much greater than that of BMs. In this case, at least 3 or 4 steps of pre-computation are needed to maintain an acceptable clock speed, which will cause large amount of hardware and computation overhead. However, for high-rate codes, the number of BMs is also large and each state receives more than 2 candidate paths. In this case, one to two steps of pre-computation are enough since regular update

of new PMs also takes a long time. In Section 4.1.3, we provide an example of a rate-3/4 code that employs 1 or 2 steps of pre-computation. The clock speed could approach the largest value in theory. In addition, the BER performance of the proposed scheme is the same as that of the conventional T-algorithm and the new scheme is more reliable than the scheme in [31].

4.1.3 The pre-computation architecture

Again, let us consider the rate-3/4 code shown in Fig. 3.2. Although there are only 64 states in the corresponding VD for this code, the number of BMs is 16 and each state receives 8 candidate paths. The computation complexity of this VD is equivalent to that of a rate-1/2 code with 256 states. Therefore, applying T-algorithm could effectively reduce the overall computational complexity. Computational complexity refers to the number of average candidate paths generated by the ACSU each cycle. For a regular VD, there are 512 (*i.e.*, 64×8) paths. It is observed from Fig. 3.13 that, as we lower the threshold "Tpm", the number of average enabled states as well as the enabled additions reduces. At the high-SNR region (SNR $\geq$ 12 dB), the number of enabled additions can be reduced to 1/10 of that for a regular VD (when Tpm = 0.3), which indicates a dramatic reduction of power consumption. Although the implementation of T-algorithm itself will introduce additional operations, compared with the saving computation, it is a small portion.

A crucial issue with implementing T-algorithm is how to quickly find out the optimal PM (the minimum value). The shortest critical path we could achieve is from the regular ACSU without T-algorithm. That's the amount of time each state needs to update its state value, as shown below:

$$T_{full\_trellis} = T_{adder} + T_{8-in\_comp}.$$

A fully parallel 8-input comparator needs 28 adders and a large look-up table. To achieve a balanced trade-off between hardware area and clock speed, the architecture of

two 4-input comparators connected with one 2-input comparator is used in this work. The critical path now becomes

$$T_{full\_trellis} = T_{adder} + T_{4-in\_comp} + T_{2-in\_comp}. \tag{4.3}$$

Next, let us consider the VD with T-algorithm. The general functional diagram is shown in Fig. 4.1, where T-algorithm is implemented in the "PM purge algorithm" unit (PPAU), which provides the information about which states are purged and which are enabled.

In a VD with conventional T-algorithm implementation, the optimal PM is calculated from the 64 newly updated PMs. To find the minimum value of the 64 PMs, we use an architecture consisting of 3-stage 4-input comparators as shown in Fig. 4.2.

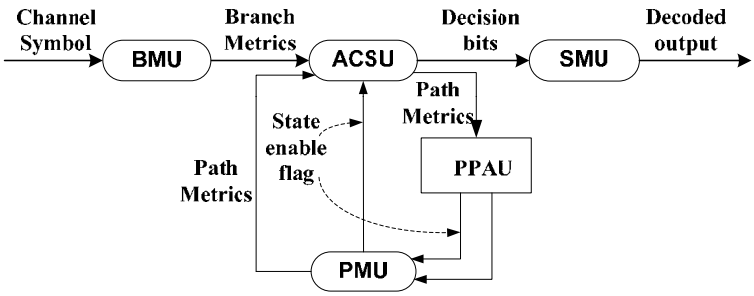


FIGURE 4.1: Functional block of a Viterbi decoder with conventional T-algorithm.

The critical path for the conventional T-algorithm implementation is computed by Eq. (4.4), where an extra delay of a 2-input comparator for comparing ($\mathrm{PM_{opt}}$ + threshold) with all the 64 PMs is included.

$$\begin{aligned} & T_{conv\_T-alg} \\ = \; & T_{adder} + T_{4-in\_comp} + T_{2-in\_comp} + 3T_{4-in\_comp} + T_{adder} + T_{2-in\_comp} \\ = \; & 2T_{adder} + 4T_{4-in\_comp} + 2T_{2-in\_comp}. \end{aligned} \tag{4.4}$$

When applying SPEC-T algorithm [31], the value of ($\mathrm{PM_{opt}}$ + threshold) can be obtained during the period when ACSU updates new PMs. The only extra calculation

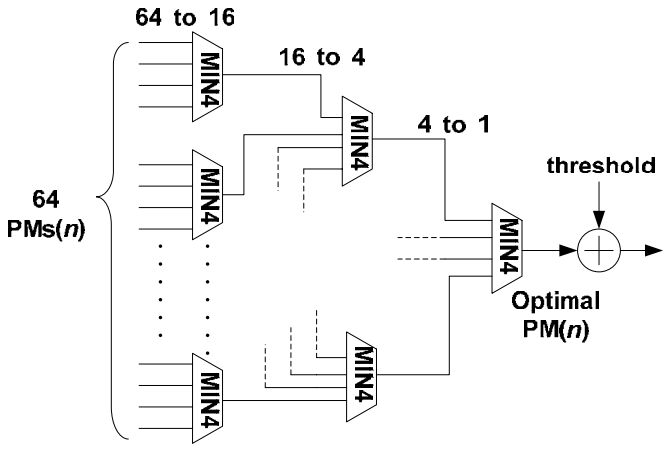


FIGURE 4.2: The implementation of PPAU for conventional T-algorithm.

for SPEC-T algorithm is the comparison between ($\mathrm{PM_{opt}}$ - threshold) and all the PMs. Therefore, the critical path for the SPEC-T scheme is greatly shortened as shown in Eq. (4.5).

$$T_{SPEC-T\_alg} = T_{adder} + T_{4-in\_comp} + 2T_{2-in\_comp}. \tag{4.5}$$

T_{SPEC-T_alg} is also the iteration bound we can get for this VD when T-algorithm is employed. The functional block of SPEC-T algorithm is slightly different from the one shown in Fig. 4.1, where the minimum BM is sent to the PPAU from the BMU. Since the estimated optimal PM is calculated each cycle, an accurate optimal PM is also needed every 6 to 7 cycles to compensate for the estimated one. For example, at time slot n, the decoder stores $\mathrm{PM_{opt\_esti}}(n)$ and PMs(n). After 7 cycles, ($\mathrm{PM_{opt\_accu}}(n)$-$\mathrm{PM_{opt\_esti}}(n)$) is added to $\mathrm{PM_{opt\_esti}}(n+7)$. The problem with this compensation scheme is that the error between $\mathrm{PM_{opt\_esti}}$ and $\mathrm{PM_{opt\_accu}}$ accumulates over at least 7-cycles due to the inherent delay of the scheme itself. Furthermore, our simulation shows that if the estimated value is adjusted every 3 or more cycles, there is a high probability ($> 3\%$ at a BER near 10^{-3}) that the decoder will lose all the survival paths during the decoding process due to the purging scheme according to the threshold and

the estimated $\mathrm{PM_{opt}}$. When the adjustment frequency reduces to once every 2 cycles, the probability drops to 0.1%, which is still not acceptable for practical systems. Therefore, the SPEC-T algorithm must adjust the estimated value each cycle, which is equivalent to the conventional T-algorithm.

A. One-step pre-computation

For convenience of discussion, we define the left-most register in Fig. 3.2 as the most-significant-bit (MSB) and the right-most register as the least-significant-bit (LSB). The 64 states and PMs are labeled from 0 to 63. A careful study reveals that the 64 states can be partitioned into two groups: odd-numbered PMs (when the LSB is '1') and even-numbered PMs (when the LSB is '0'). The odd PMs are all extended by odd BMs (when Z_0 is '1') and the even PMs are all extended by even BMs (when Z_0 is '0'). The minimum PM becomes:

$$\begin{aligned} & PM_{opt}(n) \\ = \; & \min\{ \\ & \min(\mathrm{even}PMs(n-1)) + \min(\mathrm{even}BMs(n)), \\ & \min(\mathrm{odd}PMs(n-1)) + \min(\mathrm{odd}BMs(n)) \}. \end{aligned} \tag{4.6}$$

The functional diagram of the 1-step pre-computation scheme is shown in Fig. 4.3. Notice that, in Fig. 4.1, the PPAU must wait for the new PMs from the ACSU to calculate the optimal PM, while in Fig. 4.3 the optimal PM is calculated directly from PMs in the previous cycles at the same time when the ACSU is calculating the new PMs. The details of the PPAU are shown in Fig. 4.4.

The critical path of the 1-step pre-computation scheme is

$$T_{1-step\_pre\_T} = 2T_{adder} + 2T_{4-in\_comp} + 3T_{2-in\_comp}, \tag{4.7}$$

which is much shorter than that in Eq. (4.4). Note that compared with Fig. 4.2, the hardware overhead of the 1-step pre-computation scheme is about 4 adders. Considering

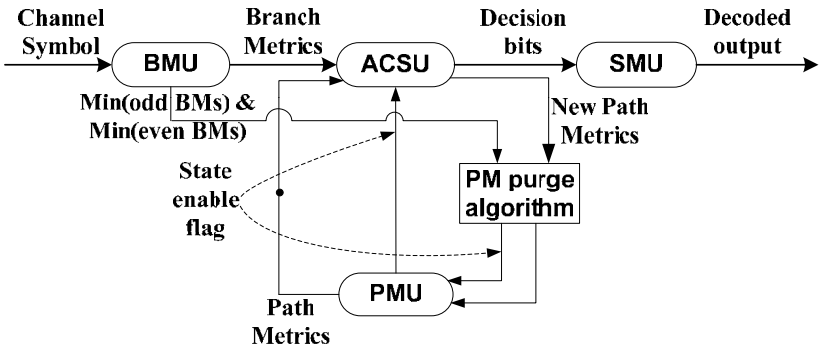


FIGURE 4.3: Functional block of a Viterbi decoder with 1-step pre-computation T-algorithm.

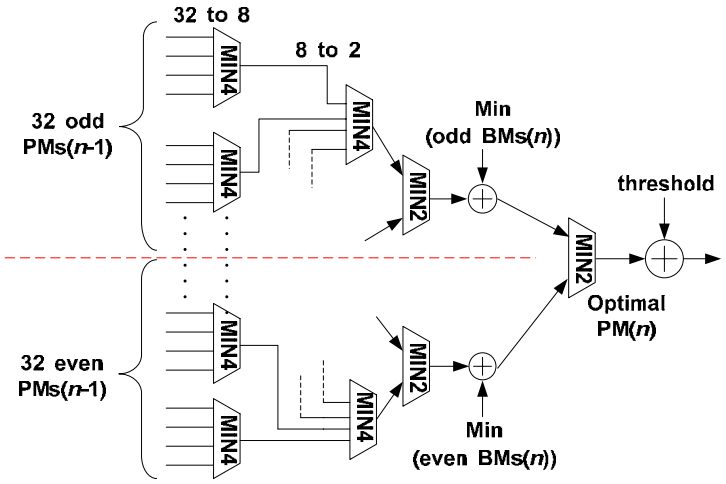


FIGURE 4.4: Block diagram of 1-step pre-computation T-algorithm..

the parallel structure of the ACSU, which contains 8×64 adders, the hardware overhead is really a small portion. Compared with the SEPC-T algorithm, however, the critical path of the 1-sept pre-computation scheme is still long. In order to further shorten the critical path, we explore the 2-step pre-computation design next.

B. Two-step pre-computation

We again need to analyze the trellis transition of the original code. In the 1-step

pre-computation architecture, we have pointed out that for the particular code shown in Fig. 3.2, odd-numbered states are extended by odd BMs, while even-numbered states are extended by even BMs. Furthermore, the even states all extend to states with higher indices (the MSB in Fig. 2 is '1') in the trellis transition, while the odd states extend to states with lower indices (the MSB is '0' in Fig. 3.2). This information allows us to obtain the 2-step pre-computation data path. This process is straightforward, although the mathematical details are tedious. For clarity, we only provide the main conclusion here.

The states are further grouped into 4 clusters as described by Eq. (4.8).

$$\begin{aligned} \text{cluster0} &= \{\text{PM}_m | 0 \le m \le 63, m \bmod 4 = 0\} \\ \text{cluster1} &= \{\text{PM}_m | 0 \le m \le 63, m \bmod 4 = 2\} \\ \text{cluster2} &= \{\text{PM}_m | 0 \le m \le 63, m \bmod 4 = 1\} \\ \text{cluster3} &= \{\text{PM}_m | 0 \le m \le 63, m \bmod 4 = 3\} \end{aligned} \tag{4.8}$$

The BMs are categorized in the same way and are described by Eq. (4.9).

$$\begin{aligned} \text{BMG0} &= \{\text{BM}_m | 0 \le m \le 15, m \bmod 4 = 0\} \\ \text{BMG1} &= \{\text{BM}_m | 0 \le m \le 15, m \bmod 4 = 2\} \\ \text{BMG2} &= \{\text{BM}_m | 0 \le m \le 15, m \bmod 4 = 1\} \\ \text{BMG3} &= \{\text{BM}_m | 0 \le m \le 15, m \bmod 4 = 3\} \end{aligned} \tag{4.9}$$

The optimal PM at time n is calculated as Eq. (4.10).

$$
\begin{aligned}
& PM_{opt}(n) \\
= \; & \min\left[\min\{\right. \\
& \min(\text{cluster0}(n-2)) + \min(\text{BMG0}(n-1)), \\
& \min(\text{cluster1}(n-2)) + \min(\text{BMG1}(n-1)), \\
& \min(\text{cluster2}(n-2)) + \min(\text{BMG3}(n-1)), \\
& \min(\text{cluster3}(n-2)) + \min(\text{BMG2}(n-1)), \\
& \}+\min(\text{even BMs}(n)), \\
& \min\{ \\
& \min(\text{cluster0}(n-2)) + \min(\text{BMG1}(n-1)), \\
& \min(\text{cluster1}(n-2)) + \min(\text{BMG0}(n-1)), \\
& \min(\text{cluster2}(n-2)) + \min(\text{BMG2}(n-1)), \\
& \min(\text{cluster3}(n-2)) + \min(\text{BMG3}(n-1)), \\
& \left.\}+\min(\text{odd BMs}(n))\right]. \qquad (4.10)
\end{aligned}
$$

An intuitive illustration of the process is shown in Fig. 4.5. The "MIN 16" unit for finding the minimum value in each cluster is constructed with 2 stages of 4-input comparators.

Calculating $\text{PM}_{\text{opt}}(n)$ from $\text{PM}(n-2)$ means that the calculation can be completed within 2 cycles . Thus, the process is pipelined as two stages as indicated by the dashed line in Fig. 4.5. Again, we need to examine the critical path of each stage. Remember that there is another comparison operation after the optimal PM is found. The critical paths of the first stage (left side of the dashed line in Fig. 4.5) and the second stage (right side of the dashed line) are expressed in Eq. (4.12) and Eq. (4.12), respectively.

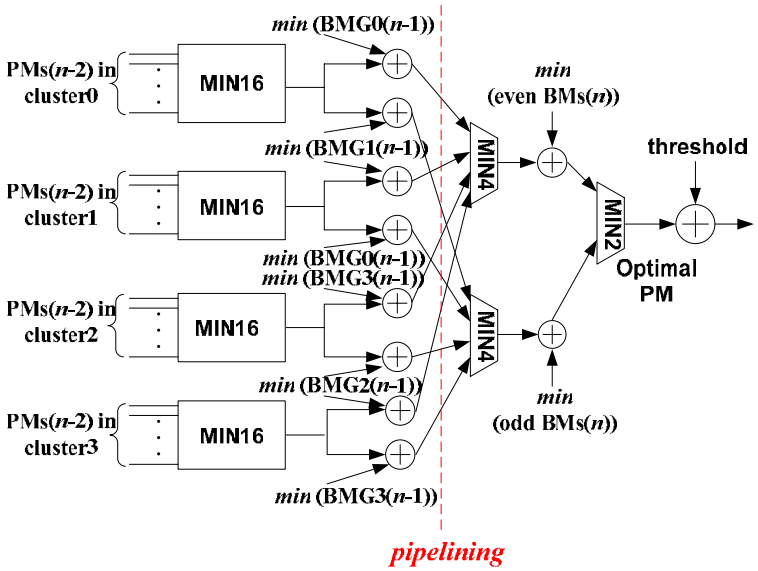


FIGURE 4.5: The implementation of 2-step pre-computation T-algorithm.

$$T(stage1)_{2-step\_pre\_T} = T_{adder} + 2T_{4-in\_comp}; \quad (4.11)$$

$$T(stage2)_{2-step\_pre\_T} = 2T_{adder} + T_{4-in\_comp} + 2T_{2-in\_comp}. \quad (4.12)$$

Comparing the above equations with Eq. (4.5), the shortest path needed for T-algorithm, we find that $T(stage1)_{2-step\_pre\_T}$ is shorter since the gate delay of the 2-stage 2-input comparator is slightly longer than that of a fully parallel 4-input comparator. On the other hand, $T(stage2)_{2-step\_pre\_T}$ is longer than T_{SPEC-T_alg} by the delay of an adder. However, by re-arranging the process and introducing one redundant adder, we can further reduce $T(stage2)_{2-step\_pre\_T}$ to that in Eq. (4.5). The details are shown in Fig. 4.6.

In Fig. 4.6, the left side of the dashed line remains the same. However, at the right side, the threshold value is added to both the min (even BMs) and min (odd BMs) before $\mathrm{PM_{opt}}$ is found out. Now, the critical path of the PPAU for the 2-step pre-computation scheme is the same as the iteration bound in Eq. (4.5). Compared with the conventional

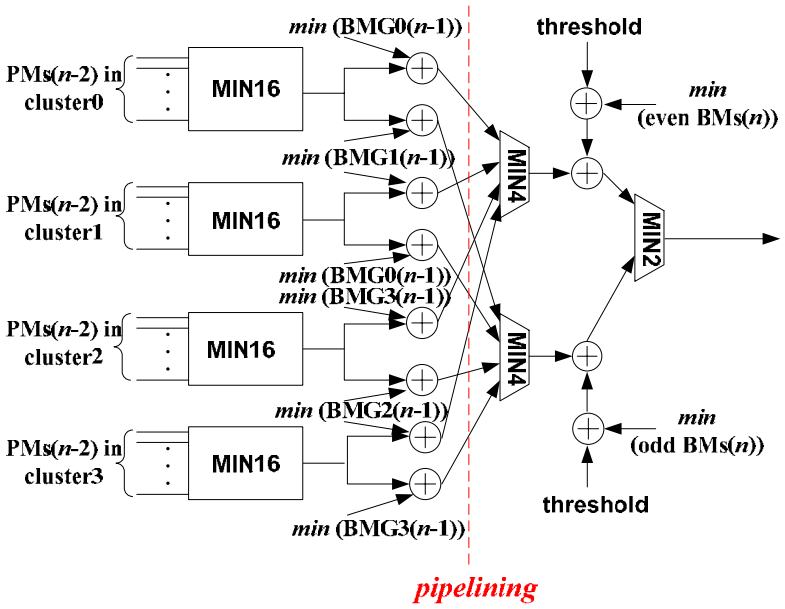


FIGURE 4.6: High-speed architecture for 2-step pre-computation T-algorithm.

implementation for T-algorithm, the hardware overhead of the architecture in Fig. 4.6 is 11 adders, a 4-input comparator and a 2-input comparator, which is about the size of the ACS circuits for one state.

More steps of pre-computation can be further achieved with larger hardware overhead; however, it is generally unnecessary in this case.

C. FPGA implementation

We implemented, in FPGA, the ACSU using several different schemes: 1) conventional implementation of T-algorithm, 2) the proposed 1-step pre-computation scheme, and 3) the proposed 2-step pre-computation scheme. SPEC-T algorithm is not considered here since it reduces to the conventional T-algorithm. The synthesis results are summarized in Table 4.1.

Table 4.1 shows that by applying the 2-step pre-computation architecture, the clock speed is doubled compared with the conventional implementation of T-algorithm. It is also

TABLE 4.1: Synthesis results of pre-computation scheme

Xc4vlx160-12ff1148 Virtex 4, speed scale 12			
	Number of 4-input LUT	Number of Slice Flip-flop	Maximum speed(MHz)
T-algorithm	23326(17%)	576(0%)	39.70
1-step pre-comp	21570(15%)	648(0%)	61.50
2-step pre-comp	22258(16%)	686(9%)	81.97

observed that, the pre-computation architecture requires such a small hardware overhead that it is not even evident in FPGA synthesis result.

4.1.4 Modification of SMU

SMU is the key module for low-power implementation of VDs. When T-algorithm is applied, RE-based SMU has an advantage because the register associated with the purged states can be powered down by clock gating [31], while for TB-based SMU, small amount of power can be saved due to relatively inflexible construction of the memory units. In this work, we adopt the clock gated RE architecture for the SMU.

In this section, we address another issue regarding SMU design when T-algorithm is employed. In the regular VD without any low-power schemes, the RE based SMU always outputs the decoded data from a fixed state (arbitrarily selected in advance) for low-complexity purpose. When T-algorithm is employed, relying only on a fixed state is impossible. In conventional implementation of T-algorithm, the decoded data are output from the register associated to the optimal state (state with $\mathrm{PM}_{\mathrm{opt}}$), because the optimal state is always enabled and the process of searching for $\mathrm{PM}_{\mathrm{opt}}$ could find the index of the optimal state as a byproduct. However, when the estimated $\mathrm{PM}_{\mathrm{opt}}$ is used [31], or in our case where $\mathrm{PM}_{\mathrm{opt}}$ is calculated from PMs at the previous time slot, it is difficult to find the index of the optimal state.

In [31], it is suggested to determine the decision symbols by a majority vote unit (MVU), which only counts the decision symbols from unpurged states, or use a $2^{(k-1)}$-to-$(k-1)$ priority encoder to find the index of an enabled state. The decoded data of low-rate codes must be either 0 or 1. Thus the design of the MVU is simple. For high-rate codes as shown in Fig. 3.2, the output from each register array is a 3-bit codeword with 8 possible values. The design of such a MVU is complex. Therefore, we choose the priority encoder method to determine the decoded codewords. We have labeled the states from 0 to 63. Then the output of the priority encoder would be the unpurged state with the lowest index. Assuming the purged states have the flag '0' and other states are assigned the flag '1', the truth table of such a priority encoder is shown in Table 4.2, where 'flag' is the input and 'index' is the output.

TABLE 4.2: Truth table of 64-to-6 priority encoder

flag[63:0]	index[5:0]
xxx.....xxxxx1	0 0 0 0 0 0
xxx.....xxxx10	0 0 0 0 0 1
xxx.....xxx100	0 0 0 0 1 0
xxx.....xx1000	0 0 0 0 1 1
xxx.....x10000	0 0 0 1 0 0
⋮	⋮
x10.....000000	1 1 1 1 1 0
100.....000000	1 1 1 1 1 1

Implementation of such a table is not trivial. In our design, we employ an efficient architecture for the 64-to-6 priority encoder based on three 4-to-2 priority encoders, as shown in Fig. 4.7. The 64 flags are first divided into 4 groups, each of which contains 16 flags. The priority encoder at level 1 detects which group contains at least one '1' and

determines 'index[5:4]'. Then MUX2 selects one group of flags based on 'index[5:4]'. The input of the priority encoder at level 2 can be computed from the output of MUX2 by 'OR' operations. We can also reuse the intermediate results by introducing another MUX (MUX1). The output of the priority encoder at level 2 is 'index[3:2]'. Again, 'index[3:2]' selects 4 flags (MUX3) as the input of the priority encoder at level 3. Finally, the last encoder will determine 'index[1:0]'.

Implementation of 4-to-2 priority encoder is much easier. Its truth table is shown in Table 4.3, and the corresponding logics are shown in Eq. (4.14) and Eq. (4.14).

TABLE 4.3: Truth table of 4-to-2 priority encoder

input(I[3:0])	output(O[5:0])
xxx1	0 0
xx10	0 1
x100	1 0
1000	1 1

$$O[0] = \overline{I[0]} \cdot (I[1] + I[3] \cdot \overline{I[2]} \cdot \overline{I[1]}) = \overline{I[0]} \cdot (I[1] + I[3]\overline{I[2]}); \quad (4.13)$$

$$O[1] = \overline{I[0]} \cdot \overline{I[1]} \cdot (I[2] + \overline{I[2]}I[3]) = \overline{I[0] + I[1]} \cdot (I[2] + I[3]). \quad (4.14)$$

The logic delay of the 4-to-2 priority encoder is less than a 1-bit full adder. Thus the logic delay of the 64-to-6 priority encoder is less than three 1-bit full adders plus two 4-input OR gate, which is smaller than the iteration bound.

4.1.5 Power estimation

To verify the low-power property of the proposed architecture, the full-trellis VD and the VD with two-step pre-computation are synthesized using TSMC 90nm CMOS standard cell. The power consumption of the two designs is measured with Synopsys

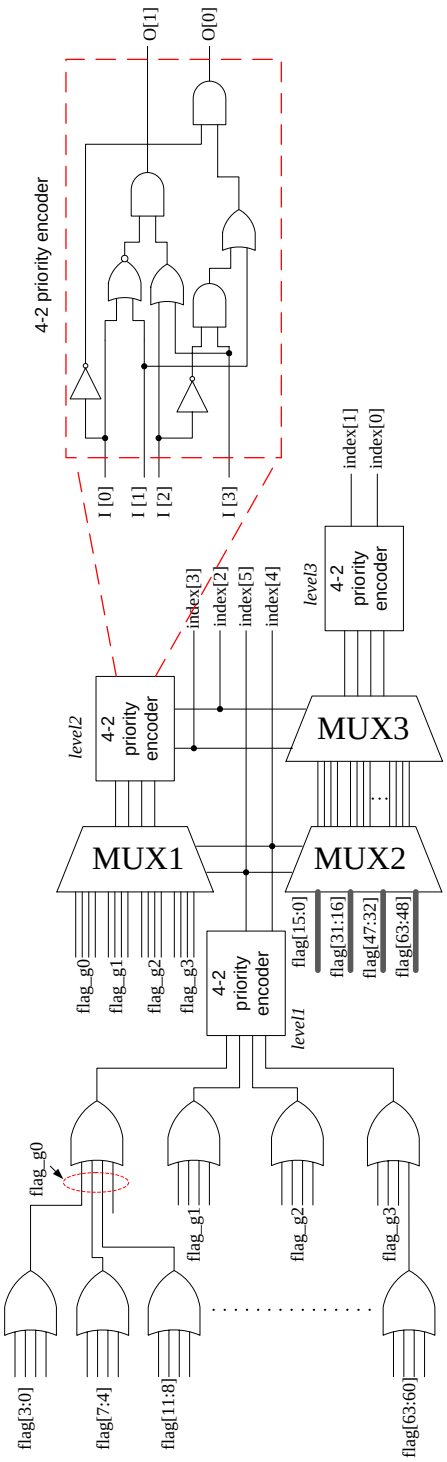


FIGURE 4.7: Architecture of 64-to-6 priority encoder.

Prime Power under the clock speed of 200Mbps (power supply of 1.0 V). 1133 received symbols (12,000 bits) are simulated. The results are shown in Table 4.4.

TABLE 4.4: Power Estimation results

	Power (mw)	
Full-trellis VD of	21.473(100%)	
two-step pre-computation VD	T=0.75	20.069 (93.5%)
	T=0.625	17.186 (80.0%)
	T=0.5	11.754 (54.7%)
	T=0.375	6.6127 (30.8%)

With the finite word-length implementation, the threshold can only be changed by a step of 0.125. Therefore, to maintain a good BER performance, the minimum threshold we chose is 0.375. Table 4.4 shows that, as the threshold decreases, the power consumption of the proposed VD reduces accordingly. In order to achieve the same BER performance, the proposed two-step pre-computation VD only consumes 30.8% the power of the full-trellis VD.

4.2. Optimal Memory Organization for Viterbi Decoders

In this section, we present efficient schemes to reduce the latency of conventional TB scheme by exploiting a pre-trace-back method. In the meantime, we adopt a buffer-based TB method to reduce memory access times, reducing power assumption significantly. Simulation results show that the proposed decoding schemes cause either zero or negligible performance loss.

4.2.1 Literature review

As introduced in Chapter 3, there are two basic schemes for SMU: register-exchange scheme (RE) and trace-back (TB) scheme. RE is the most straightforward method to extract the information bits from the encoded bit stream. It consists of a two dimension register array between which is a column of multiplexors, as shown in Fig. 4.8. It is attractive in terms of the regularity of circuit structure and decoding latency. However, for VD dealing with convolutional codes with a large constraint length K, RE becomes impractical due to its high power consumption and large routing overhead.

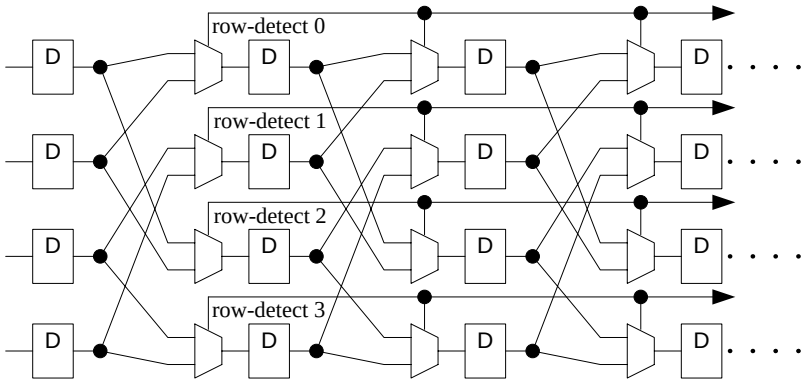


FIGURE 4.8: A 4-state RE SMU.

On the other hand, TB scheme searches for the maximum likelihood path through the memory which contains the decision information from ACSU. The classical implementation of the TB scheme is shown in Fig. 4.9, where WT is the writing process, TB is the trace back process and DC is the decoding process. Four banks of memories with length L each are required. L, the number of steps that all states can trace back to the same position, is usually $5K$ or $6K$. The total decoding latency is $4L$ since a last-in first-out (LIFO) stack is required to perform bit order reversal after DC (*i.e.*, the decoded bits in DC is written from right to left in Fig. 4.9, but the data are actually transmitted from left to right). Compared with RE scheme, TB scheme consumes less power but has a larger

latency that is proportional to K. Finding a good trade-off between power and latency has been an area of research focus for sometime. In [4,34,36], variations of TB algorithms have been proposed to either lower the power consumption or reduce the latency. A brief review of these scheme is given in below.

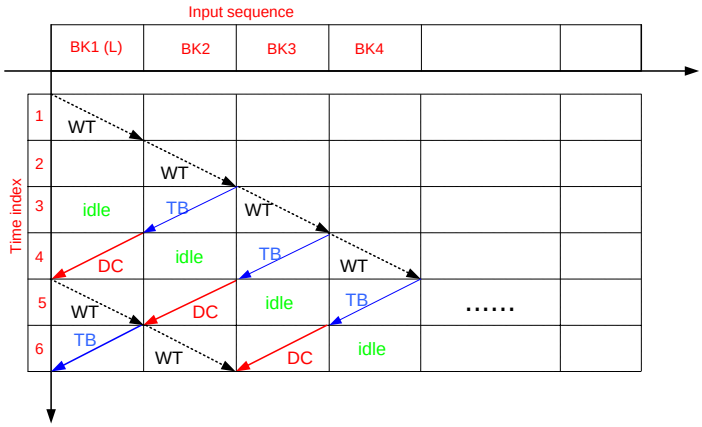


FIGURE 4.9: The conventional memory based TB approach.

A. The i-pointer algorithm

The i-pointer even algorithm is first proposed in [34], where i is the number of read pointers to access the SMU. The maximum of i is $L+1$. By increasing i, the decoding process is accelerated and both the latency and required memory are reduced to $2i \times L/(i-1)$ (*i.e.*, $2i$ banks with the length of $L/(i-1)$ each). The illustration of a 3-pointer even algorithm is shown in Fig. 4.10. At each time index, there are three pointers reading data from memory (*i.e.*, 2 for the TB and 1 for DC). And each bit stored in the memory will be read three times, while in the classical 2-pointer TB, it is only accessed twice. Obviously, each bit will be read i times in the i-pointer algorithm, which consumes more power. Here, power consumption is sacrificed for latency. This scheme is a good example of trading-off power for latency.

B. The pre-traceback algorithm

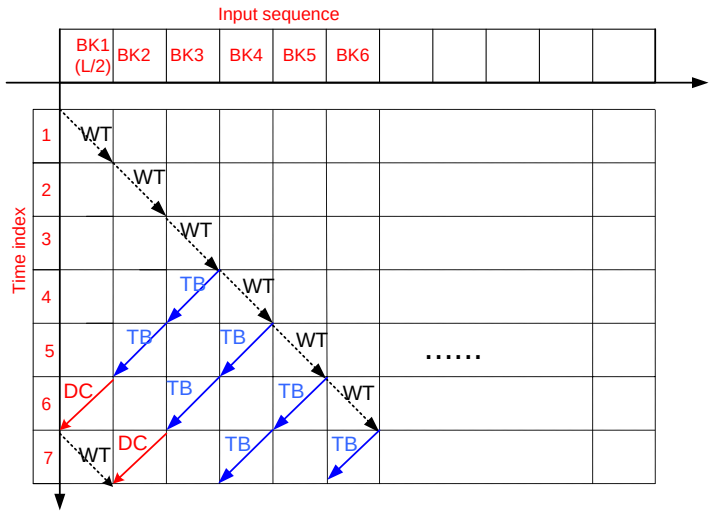


FIGURE 4.10: The 3-pointer even TB scheme.

The basic idea is to use registers to store the initial start point of each state. When decision bits are written into the memory each clock cycle, the register also updates the associated start point information. Thus, when a bank of memory is fully filled, the initial point can be known immediately and the conventional trace-back process can be removed. The Pre-traceback scheme is shown in Fig. 4.9. The details of updating start point information in the register can be found in [4]. Compared with the classical TB in Fig. 4.9, the pre-traceback scheme only leads to $3L$ decoding latency, and the saved decision bits are only read once during the decoding process.

C. The buffer based TB

This scheme targets on low power applications. The classical TB scheme in Fig. 4.9 is also called 2-pointer algorithm since at each clock cycle, there are two pointers accessing the memories. It also indicates that each bit stored in the memories will be read twice during the decoding process. In fact, the TB process and DC process are very similar. In the high SNR environment, when tracing back from the state with the optimal PM,

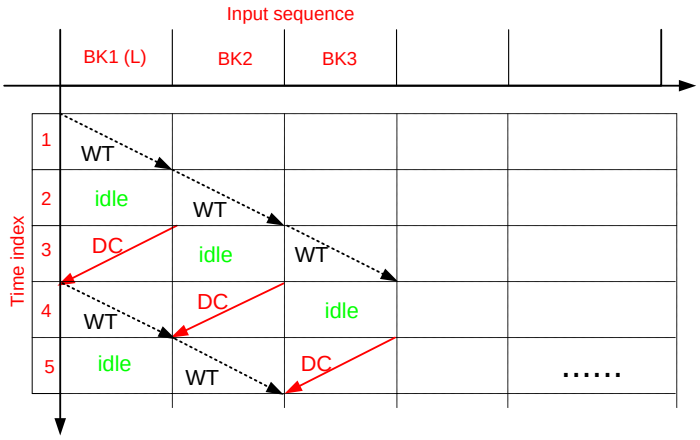


FIGURE 4.11: The Pre-traceback scheme.

DC usually repeats the same process of TB. Therefore, in [36], a buffer (bank of registers) with the same length of memory and a $K-1$ bits head register are used for each memory bank. At the beginning of TB, a head register is used to memorize the TB start point. Then, the decision bits for the TB path are saved into the buffer. During the DC process later, TB path is recovered from the buffer at the same time. The decoder compares the decoded path and that recovered from the buffer each step. When they become identical, the decoded data can be directly read out from the buffer afterwards. The latency of the scheme remains the same as the classical method, and the power due to reading the memories can be saved with some hardware penalty.

4.2.2 Proposed scheme-I

In the first trace-back scheme (denoted as Scheme-I), the length of each memory bank is set to $L/2$ as the 3-pointer TB scheme. The key points of the Scheme-I are as follows.

1. In order to maintain the same performance as the conventional memory based TB scheme, a TB process continuously goes through two memory banks for a trace-back length of L. For the first bank, a set of registers are used to pre-track the initial

starting point of each state. When the best state is known, the initial point can be found immediately. For the second memory bank, the conventional TB is employed. As a result, the decoding latency is reduced to $2.5L$.

2. It has been pointed out in [36] that, in DC process, after a number of steps, SMU trends to trace the same path which has been traced in the TB process. Therefore, in order to eliminate redundant operations, a buffer is assigned to each bank of memories to save the trace-back starting states and the decision bits generated in the TB process. When the same bank of data is decoded later, the previous TB path is recovered from the buffer. The decoded path and the recovered TB path are compared at each step. When they reach the same state, the decoded data can be directly read out from the buffer immediately. By using buffers, memory access operations are significantly reduced. The decoding procedures are shown in Fig. 4.12.

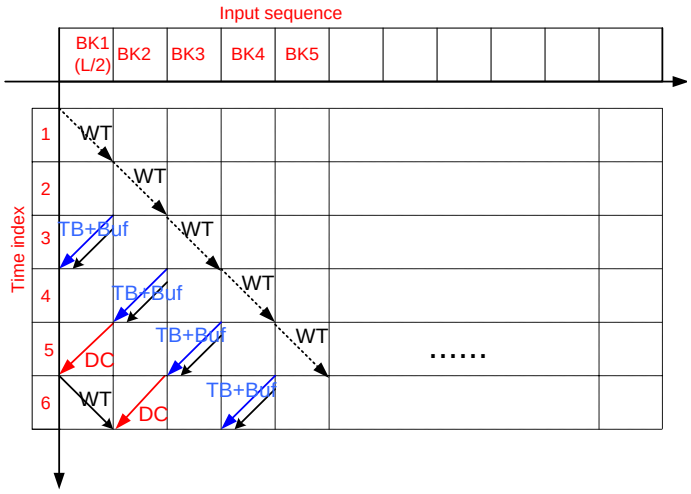


FIGURE 4.12: The proposed scheme-I.

The details of the scheme are explained in the following.

1. $0 \leq t < L/2$: ACSU writes decision bits into bank (BK) 1 in forward direction. At

$t = L/2 - 1$, initiate the pre-trace-back registers.

2. $L/2 \leq t < L$: ACSU writes decision bits into BK2. Meanwhile, pre-traceback is performed for BK2 by updating the registers to track the initial point of every state. At $t = L-1$, the pre-TB state for BK2 can be immediately obtained from the register corresponding to the current best state. Then, re-initiate the pre-TB registers.

3. $L \leq t < 3L/2$: SMU pre-traces back BK3 while ACSU writes decision bits into BK3. TB process is performed for BK1 in backward direction from the pre-TB state obtained in step 2. Meanwhile, SMU records the trace-back starting state and stores the temporal decoded bits from the TB process into a buffer. At $t = 3L/2-1$, obtain the pre-TB state from the register corresponding to the current best state for BK2. Then, re-initiate the pre-TB registers.

4. $3L/2 \leq t < 2L$: BK4 is written and a corresponding pre-trace-back is performed. Meanwhile, SMU traces back BK2 and records decision bits into the buffer associated with BK2. At $t = 2L - 1$, the initial state for BK4 can be found. Then the pre-TB registers are re-initiated.

5. $2L \leq t < 5L/2$: SMU recovers the TB path from the buffer associated with BK1 and decodes for BK1. The path obtained in DC process and the path recovered from buffer are compared at each step until they reach the same state. Then the buffer should contain the decoded data that can be sent out. There's no need to access BK1 any more. Similar to previous time periods, SMU writes decision bits to BK5. The corresponding pre-trace-back for BK5 is performed. TB process and decision bits recording are continued for BK3.

In the 3-pointer TB scheme, a last-in-first-out (LIFO) of length L/2 is needed to reverse the order of decoded bits. In the proposed Scheme-I, the buffer can be used as a LIFO.

4.2.3 Proposed scheme-II

A close examination of the survivor path in DC process and the path in TB process for the same bank of memory reveals that the two paths merge after only a few steps. In our study, a rate-1/2 convolutional code is simulated. The constraint length is 9 and the generator polynomial is (561, 735). 40,000 banks of data (1,080,000 bits) are simulated at SNRs of 0.3 dB, -0.7 dB, -1.7 dB. Fig. 4.13 shows the distribution of the number of steps that DC process is performed before the two paths merge. At SNR=0.3 dB, the two paths merge within 13 steps (*i.e.*, $L/4$ steps) in all cases. Within L/4 steps, two paths merge in 97% simulated cases at SNR=-1.7 dB.

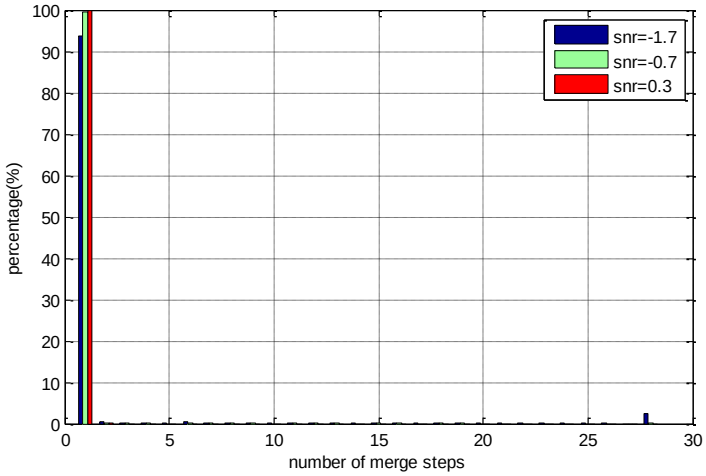


FIGURE 4.13: The number of steps needed for trace-back paths merging.

By exploiting this characteristic, the decoding latency can be further reduced. Similar to Scheme-I, the length of each bank of memory is $L/2$ in the second scheme. However, the previous DC process is modified in the new scheme. The first half of DC process is the same as before. In this period, we update the buffer as well. But in the meantime, we start to output the decoded bits from the corresponding buffer in a LIFO way. When we reach the middle point of the bank, we will output decoded bits from the newly updated

(2nd half) buffer. As a result, the decoding latency is reduced to $2L$. This proposed scheme is denoted as Scheme-II in this thesis. Its details are illustrated in Fig. 4.14.

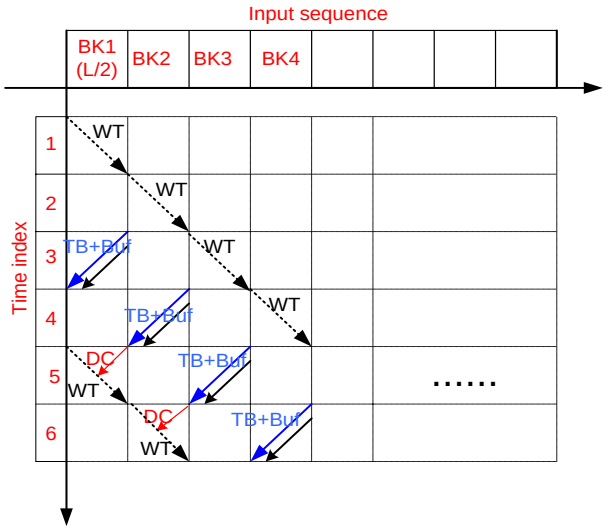


FIGURE 4.14: The proposed scheme-II.

From $t = 0$ to $t = 2L - 1$, Scheme-II has the same processes as Scheme-I. During $[2L, 5L/2]$, the decoded bits recorded in the buffer corresponding to BK1 are sent out in an increasing address order while the DC process revises the temporal decoded bits in a decreasing address order. In addition, new decision bits from ACSU are written into BK1.

4.2.4 BER performance

A rate-1/2 convolutional code is simulated. The constraint length is 9 and the generator polynomial is (561, 735). An AWGN channel is assumed and data are represented in floating-point precision. Four decoding schemes are simulated: the optimal Viterbi decoding [38], the conventional TB scheme, the proposed Scheme-I, and the proposed Scheme-II. For the optimal Viterbi decoding, the TB process starts after the whole signal sequence is received. For the conventional TB decoder, L is selected as $6K$, where $K = 9$ is the constrain length of the simulated conventional code.

Fig. 4.15 shows the BER performance of the decoding schemes. The optimal Viterbi decoding has the best performance. The decoding performance of TB scheme and the proposed Scheme-I are the same. It is not a surprise because the two approaches have the same trace-back length. For the proposed Scheme-II, a slight performance loss is caused if the survivor path in DC process and the path in TB process do not merge in the first $L/4$ steps. In Fig. 4.15, no performance losses are observed in the high-SNR region. At low-SNR region, e.g., close to -1.7 dB, less than 0.02 dB performance loss is introduced compared to the original TB scheme.

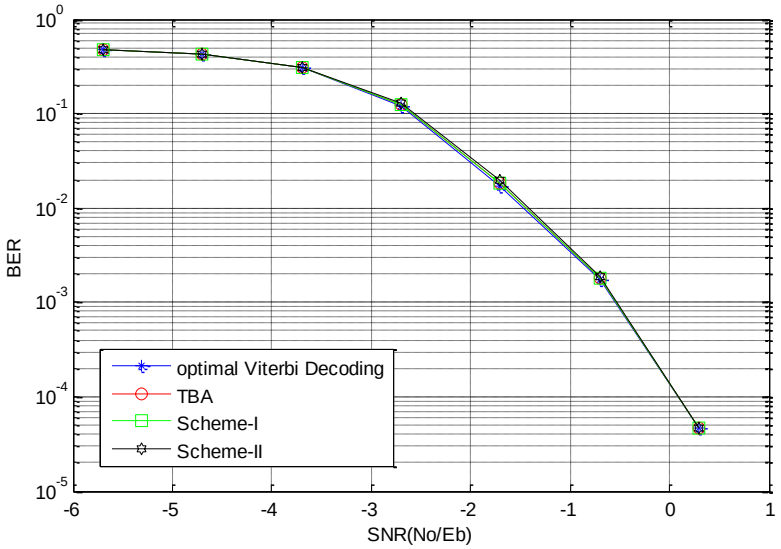


FIGURE 4.15: BER performance comparisons for several TB schemes.

4.2.5 Complexity analysis

The decoding latency and implementation complexity of different trace-back schemes for SMU are summarized in Table- I. The pipeline latency from ACSU is not included since ACSU is independent of SMU. Besides, each buffer used for storing trace-back path in Scheme-I and Scheme-II can be implemented as a $(K-1) \times L/2$-bit dual-port memory. This part of hardware is neglected in the comparison as it is much smaller compared to

other parts. From Table 1, it is observed that the conventional TB scheme leads to the longest decoding latency of $4L$. The 3-pointer TB and Pre-TB schemes have the same decoding latency of $3L$. However, Pre-TB scheme requires extra registers. The proposed Scheme-I reduces the latency to $2.5L$ without performance losses. It requires the same amount of registers but less memory than the Pre-TB scheme [4]. Compared with the 3-pointer TB scheme, the proposed Scheme-I requires less memory but more registers. The overall hardware overhead is very small (*i.e.*, it reduces 27×256-bit memory while adding 256×8-bit registers). The proposed Scheme-II has the shortest decoding latency among all the low power trace-back schemes. The implementation complexity of Scheme-II is less than that of the Pre-TB scheme and the Scheme-I. Compared to the 3-pointer TB scheme, the Scheme-II reduces significant amount of decoding latency with similar implementation complexity (*i.e.*, saving 54×256-bit memory while adding 256×8-bit registers).

TABLE 4.5: Decoding latency and the implementation complexity for SMU

	Decoding Latency	Memory Width	Memory Depth	Number of Memories	Register
TB	216 ($4L$)	256	54 (L)	4	0
3-pointer TB	162 ($3L$)	256	27 ($L/2$)	6	0
Pre-TB	162 ($3L$)	256	54 (L)	3	256×8
Scheme-I	135 ($2.5L$)	256	27 ($L/2$)	5	256×8
Scheme-II	108 ($2L$)	256	27 ($L/2$)	4	256×8

It is possible to apply the pre-TB method to the 3-pointer TB scheme to reduce the latency down to 2L [4]. However, 2 sets of registers, each of which includes 256×8-bit registers, are required to perform parallel pre-trace-back processes. The 2-parallel pre-TB architecture needs extra 256×8-bit registers compared with the proposed Scheme-II. Furthermore, updating 256×8 registers at each cycle consumes significant power. On the

other hand, by increasing the number of parallel TB processes in i-pointer TB scheme, the latency can also be reduced to the minimum of $2L$ when $i = L + 1$ [34]. However, it's impractical for real applications due to the huge amount of power consumption from L parallel TB processes. Hence, Scheme-II is a better choice in the sense of low power consumption.

4.3. Conclusion

This chapter deals with some of the technical challenges for designing Viterbi decoders. First, we proposed a pre-computation scheme with its associated hardware architecture for Viterbi decoders employing T-algorithm. Compared with existing schemes that target to efficiently implement T-algorithm, the proposed scheme is more reliable for high-rate codes. The analysis of the critical path reveals that the pre-computation scheme could achieve the iteration bound for Viterbi decoders employing T-algorithm with negligible hardware overhead. Simulation results show that the proposed scheme maintains the same BER performance as the conventional T-algorithm while other schemes could lose all the survive paths during the decoding process. Synthesis results in FPGA verify the speed advantages of the proposed pre-computation scheme. Power consumption of the proposed architecture is evaluated through its ASIC implementation. It shows that the proposed Viterbi decoder only consumes 30.8% of the power compared with a full-trellis Viterbi decoder.

Then, we presented two efficient trace-back schemes to improve the decoding latency of conventional trace-back approach. The proposed scheme reduces the decoding latency from $4L$ to $2L$ at the expense of a small hardware overhead and a slightly increased power consumption. The power consumption overhead is mostly compensated by adopting a buffer-based trace-back scheme. Compared with the conventional RE, the latency of the

proposed Scheme-II is still L cycles longer but its total power consumption is significantly smaller.

5. HIGH-SPEED MEMORY-REDUCED SISO DESIGN FOR DUO-BINARY TURBO CODE

5.1. literature Review

Nonbinary CTC were first introduced by Claude Berrou and M. Jezequel [39], which has superior decoding performance compared with the single-binary (SB) CTC. Recently, dou-binary (DB) CTC has attracted significant research interest and is adopted in wireless communication standards such as digital video broadcasting return channel over satellite and terrestrial (DVB-RCS and DVB-RCT) [25] and WiMAX [26].

An optimal decoding method for CTC is the iterative decoding using the maximum *a posteriori* probability (MAP) algorithm [40]. For simplicity of implementation, a common practice is to execute the computation in the logarithmic domain, which is known as the log-MAP algorithm [41]. Existing work regarding the implementation of log-MAP algorithm has focused on low latency design [42, 43], memory-reduced schemes [43–46], and sub-optimal low-complexity decoding approaches [5–7]. Low latency designs are usually realized by a parallel decoding process. The memory-reduced schemes focused on the reverse calculation of the forward state metrics α, in which the decoder only saves some of the forward state metrics during the forward calculation process and recovers the rest of them later in the backward calculation process. Sub-optimal decoding algorithms refer to schemes like Max-log-MAP algorithm [5], enhanced Max-log-MAP [6] and linear approximation [7], which sacrifice the BER performance for low-complexity implementation.

When a CTC is extended from SB to DB, the basic decoding method remains the same. However, the computational complexity will increase significantly. The above

mentioned power-reduced or complexity-reduced techniques can be applied on DB CTC directly. In addition, special schemes have been proposed to further reduce the memory accessing rate in DB CTC decoder [47–49]. In [47], border metrics β of each sliding window from previous decoding iterations are saved. In the current decoding iteration, the *warm-up* backward calculation is removed, and the saved border metrics are used as the initial probability information for each sliding window. In a more recent work [48], the extrinsic information is expressed at the bit-level rather than the symbol-level. As a result, the memory used for interleaving is reduced by half. In [49], a trace-back MAP decoding method is proposed. To reduce the memory for storing the forward metrics α, only the 6 difference metrics are saved. Then during the backward process to calculate extrinsic metrics and the *a posteriori* log-likelihood ratio (LLR), 8 forward metrics α are recovered from those difference metrics in the reverse order by a trace-back module. Although extra computation logic is required to re-generate α, the authors claim that there is a 7% power reduction. In this chapter, we will first show that the BMs γ, rather than α, dominate the overall memory size in the soft-input soft-output (SISO) decoder for DB CTC since the number of BMs is dramatically increased in this case. In general, a BM can be expressed as the sum of the information metric (IM) and the parity metric (PAM). Based on this observation, we propose a memory-reduced architecture that reduces the memory for SISO decoder to less than half of the conventional way by saving the IMs and PAMs, instead of BMs. In principle, this technique can be applied to any type of CTCs. For SB CTCs, it will not make any difference to the conventional scheme which stores BMs directly; for the decoding of non-binary CTCs, considerable amount of memory can be reduced. Furthermore, as the BMs are re-defined, the computation for the extrinsic metrics Λ_{ex} and the LLR Λ_{apo} can be simplified, and unnecessary computations are removed from the decoder.

The recursion architecture to calculate the state metrics is another issue for decod-

ing DB CTCs, because it is the high-speed bottleneck for the decoder. As the number of candidate paths reaching each state increases from 2 to 4, the length of the critical path as well as the computational complexity is increased. To reduce the computational complexity while still maintaining a good BER performance, two-largest (TL) log-MAP algorithm becomes attractive for DB CTC decoding. TL log-MAP algorithm was originally proposed for decoding turbo TCM codes [50], which suggested that when the number of candidate paths reaching the same state is larger than 2, the decoder finds the two largest values within all the candidate first; then the log-MAP algorithm is only applied to the two largest values. When this TL log-MAP algorithm is employed for DB CTCs, simulation result shows it has no observable performance loss compared with the log-MAP algorithm. However, a straightforward implementation of the TL log-MAP algorithm will dramatically reduce the decoding speed due to the sorting process to find out the two largest candidates. In this chapter, we will explore two high-speed recursion architectures for TL log-MAP algorithm, where the critical path is reduced through algorithmic approximation and bit-level optimization with negligible BER performance loss.

5.2. System Overview

Fig. 5.1 shows the DB CTC encoder used in WiMAX standards, where the constraint length v is 4 and the data sequence is fed to the encoder in pairs. A special trellis-termination technique, *i.e.*, circular coding [51,52], is used in DB CTC to drive the encoder to a known state S_c at the end of the encoding state. Since the value of S_c depends on the length and contents of the data frame, pre-encoding for both the original and the permuted data is required. Pre-encoding initializes the encoder to the all zero state. Then, the entire data frame goes through the encoder and ends at state S_p. S_c is obtained from the expression $S_c = (I + G^N)^{-1} S_p$ [53], where I is the identity matrix, G is the

generator matrix of the encoder [54] and N is the length of the frame. Obviously, in order to get a valid S_c, N should not be the period of G. When the encoder is initialized as S_c, the encoder will return to S_c after encoding the data again.

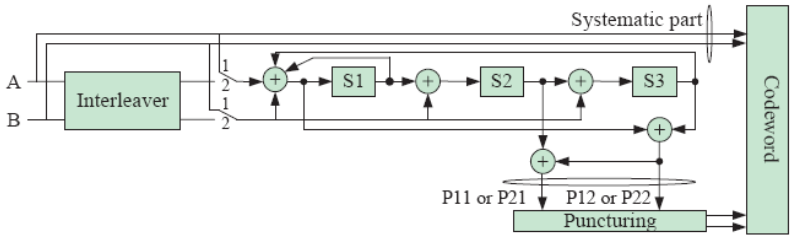


FIGURE 5.1: Duo-binary convolutional turbo code adopted in WiMAX.

Let j be the address of the original data stream and i be the address of the permutated data. Interleaving is performed at two levels: For $j = 0, \ldots, N-1$,

- Level 1: inversion of A_j and B_j in the data pairs, if $(j \bmod 2) = 0$;
- Level 2: $i = (P \times j + Q(j) + 3) \bmod N$, with

$$\begin{aligned} Q(j) &= 0 && \text{if } (j \bmod 4) = 0 \\ Q(j) &= Q_1 && \text{if } (j \bmod 4) = 1 \\ Q(j) &= 4Q_0 + Q_2 && \text{if } (j \bmod 4) = 2 \\ Q(j) &= 4Q_0 + Q_3 && \text{if } (j \bmod 4) = 3\ , \end{aligned}$$

where Q_0, Q_1, Q_2 and Q_3 are determined according to the frame length N.

The iterative decoder for DB CTC is shown in Fig. 5.2, where the Max-log-MAP algorithm for SISO decoder can be expressed as Eq. (5.1) [49].

$$\gamma_k^z(s_{k-1}, s_k) = \Lambda_{apr,k}^z(u_k) + 2\sum_{i=1}^{2} y_k^{si} x_k^{si} + 2\sum_{i=1}^{2} y_k^{pi} x_k^{pi}; \tag{5.1a}$$

$$\alpha_k(s_k) = \max_{s_{k-1}}(\gamma_k^z(s_{k-1}, s_k) + \alpha_{k-1}(s_{k-1})); \tag{5.1b}$$

$$\beta_k(s_k) = \max_{s_{k+1}}(\gamma_{k+1}^z(s_k, s_{k+1}) + \beta_{k+1}(s_{k+1})); \tag{5.1c}$$

$$\begin{aligned}\Lambda_{apo,k}^z(u_k) = & \max_{\substack{s_{k-1}, s_k, \\ u_k = z}} (\alpha_{k-1}(s_{k-1}) + \gamma_k^z(s_{k-1}, s_k) + \beta_k(s_k)) \\ & - \max_{\substack{s_{k-1}, s_k, \\ u_k = 00}} \Big(\alpha_{k-1}(s_{k-1}) + \gamma_k^{(00)}(s_{k-1}, s_k) \\ & + \beta_k(s_k)\Big);\end{aligned} \tag{5.1d}$$

$$\Lambda_{ex,k}^z(u_k) = \delta(\Lambda_{apo,k}^z(u_k) - \Lambda_{apr,k}^z(u_k) - \Lambda_{in,k}^z(u_k)); \tag{5.1e}$$

$$\begin{cases} \Lambda_{in,k}^{(00)}(u_k) = 0, \\ \Lambda_{in,k}^{(01)}(u_k) = 4y_k^{s1}, \\ \Lambda_{in,k}^{(10)}(u_k) = 4y_k^{s2}, \\ \Lambda_{in,k}^{(11)}(u_k) = 4(y_k^{s1} + y_k^{s2}) \end{cases} \tag{5.1f}$$

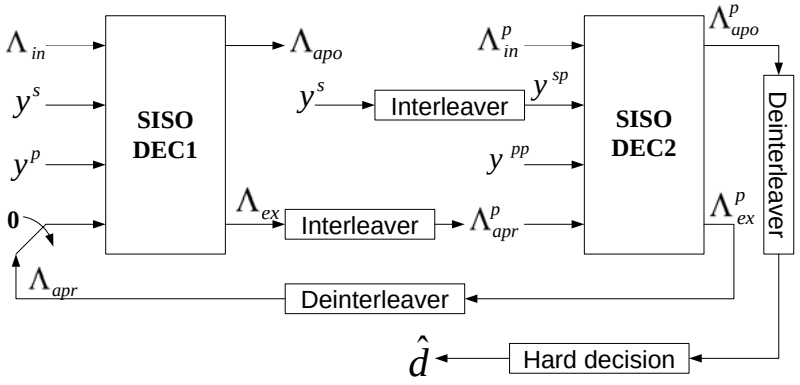


FIGURE 5.2: Iterative decoder for DB CTC.

In Eq. (5.1), k is the time instant, u_k is the decoded symbol containing 2-bit information, δ is a scaling factor between (0, 1], which can enhance the decoding performance [6], z is the fixed binary bits taking on the values of 00, 01, 10, 11, x_k^{si}/x_k^{pi} are the transmitted

codewords (1 or -1), y_k^{si}/y_k^{pi} are the received soft codewords, and the superscript si and pi could be 0 or 1. Due to the circular coding technique discussed above, the initial values of α and β are not explicitly specified, because the circular state S_c is not delivered to the decoder. To determine the initial values, in [55], it is proposed to use the ending metric values from previous iteration as the initial values of the current decoding iteration. Specifically, for the first iteration, the initial values of α and β are set to be identical for all states; during the following decoding iterations, initial values are taken from the ending values of previous iteration, since the starting and ending states of the frame are the same.

When the max operations in Eq. (5.1) are replaced by max* in Eq. (5.2), the Max-log-MAP becomes log-MAP algorithm:

$$\max^*(x, y, z, \dots) = \ln(e^x + e^y + e^z + \dots). \tag{5.2}$$

Eq. (5.1) reveals that, to decode CTC, 2^{v-1} forward and 2^{v-1} backward state metrics are computed in reverse orders. Memories are required to store either forward or backward state metrics as well as the BMs. In order to reduce the memory size, sliding window (SW) MAP [56] decoding is commonly used in practical implementation, where the whole frame is partitioned into small blocks. The decoding flow for sliding window MAP is illustrated in Fig. 5.3, where the length of each block is D. Three computation units work on 3 different blocks simultaneously: the "*warm-up*" dummy calculation for β, the computation for forward state metrics α, and the computation for backward state metrics β. When both α and β are available for a block, the decoder calculates and outputs the soft-decision results.

The VLSI architecture of the SISO decoder employing the SW log-MAP algorithm is shown in Fig. 5.4, where α Unit calculates the forward state metrics, $\beta 1$ Unit calculates the backward state metrics, and $\beta 0$ Unit is responsible for the "*warm-up*" process. Also,

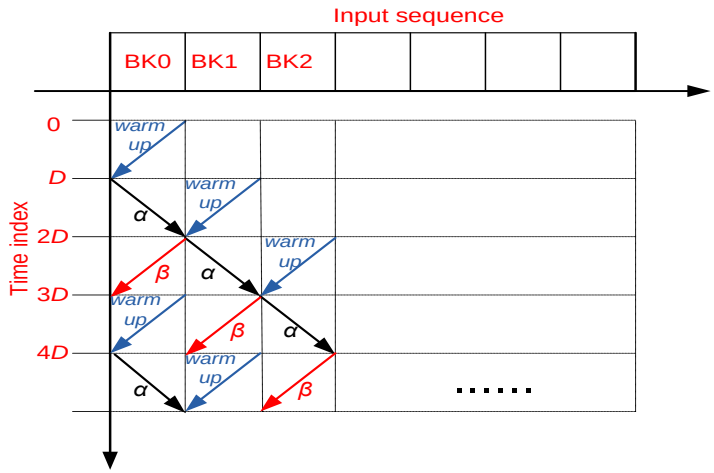


FIGURE 5.3: The decoding flow of sliding window MAP.

in this figure, there are 3 banks of memory for the interleaving process, and for storing BMs and α. To reduce the size of memory and the power consumption for accessing the memory, many schemes have been proposed as discussed in Section 5.1. However, previous research has only focused on memory organizations for forward state metrics α and the interleaving process. In practice, a DB CTC decoder generates much more γ than β (*i.e.*, 16 γ and 8 β each cycle), and γ will dominate the overall memory size inside the SISO decoder. Therefore, memory optimization for γ becomes more important in this case.

Another interesting thing that we observe from both Eq. (5.1) and Fig. 5.4 is that, this algorithm requires Λ_{apo} be calculated first to generate Λ_{ex}. However, as shown in Fig. 5.2, Λ_{apo} might not be needed in the first 2 or 3 decoding iterations, while Λ_{ex} is required in all decoding iterations. In our proposed SISO decoder, we will try to eliminate some redundant calculation and make the computation of Λ_{ex} independent of Λ_{apo}.

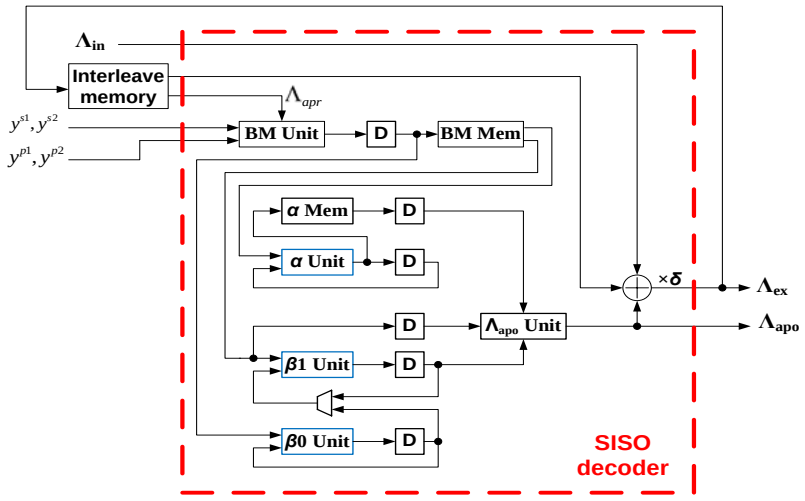


FIGURE 5.4: The VLSI architecture for SISO decoder.

5.3. Memory-Reduced SISO Decoder

5.3.1 Analysis of branch metrics

Usually, γ consists of two portions: the received values ($i.e.$,$2\sum_{i=1}^{2} y_k^{si} x_k^{si} + 2\sum_{i=1}^{2} y_k^{pi} x_k^{pi}$) and the *a priori* probability values $\Lambda^z_{apr,k}(u_k)$. In the VLSI implementation, it is more convenient to decompose γ as IM ($\tilde{\gamma}^z_{in,k}$) and PAM ($\tilde{\gamma}^m_{pa,k}$), as shown bellow, where m is the index that refers to the 4 combinations of $p1$ and $p2$.

$$\begin{aligned} \tilde{\gamma}^z_{in,k} &= \Lambda^z_{apr,k}(u_k) + 2y_k^{s1}x_k^{s1} + 2y_k^{s2}x_k^{s2}; \\ \tilde{\gamma}^m_{pa,k} &= 2y_k^{p1}x_k^{p1} + 2y_k^{p2}x_k^{p2}. \end{aligned}$$

In IMs, indices $s1$ and $s2$ always match with z. For example, when $s1 = s2 = 0$, z equals 00; when $s1 = 1$ and $s2 = 0$, z equals 01 and so on. On the other hand, z and m are independent. Therefore, for each group of input data, there are 4 IMs and 4 PAMs, which results in 16 BMs. Since turbo codes employ systematic convolutional codes, a BM can always be partitioned into an IM and a PAM. During the decoding process, only the

differences between BMs influence the decoding result. Therefore, we can further define IMs and PAMs as Eq. (5.3), which makes the number of meaningful values of IMs or PAMs only 3 instead of 4, because $\gamma_{in,k}^{(00)} = \gamma_{pa,k}^{(00)} = 0$.

$$\gamma_{in,k}^{z} = \tilde{\gamma}_{in,k}^{z} - \tilde{\gamma}_{in,k}^{(00)}; \tag{5.3a}$$

$$\gamma_{pa,k}^{m} = \tilde{\gamma}_{pa,k}^{m} - \tilde{\gamma}_{pa,k}^{(00)}. \tag{5.3b}$$

5.3.2 Modification of MAP algorithm

In Section 5.2, the BMs are expressed as the sum of IMs and PAMs. The MAP algorithm in Eq. (5.1) can be modified accordingly to eliminate redundant computations. The improvements are enabled by the fact that a common part for several candidates can be extracted out: given p to be the common part, for the Max-log-MAP algorithm,

$$\max(x+p, y+p, z+p, \dots) = p + \max(x, y, z, \dots); \tag{5.4}$$

while for the log-MAP algorithm

$$\begin{aligned} & \max^*(x+p, y+p, z+p, \dots) \\ = \; & \ln(e^{x+p} + e^{y+p} + e^{z+p} + \dots) \\ = \; & \ln(e^p(e^x + e^y + e^z + \dots)) \\ = \; & \ln(e^p) + \ln(e^x + e^y + e^z + \dots) \\ = \; & p + \max^*(x, y, z, \dots). \end{aligned} \tag{5.5}$$

First, we will simplify the calculation of the *a posterior* LLR. The candidates for

each *a posteriori* LLR contain the same information metric $\gamma^z_{in,k}$. For example,

$$\Lambda^{(11)}_{apo,k}(u_k = 11)$$

$$= \max \left\{ \begin{array}{l} \alpha_{(k-1)}(0) + \gamma^{(11)}_{in,k} + \gamma^{(00)}_{pa,k} + \beta_k(6), \\ \alpha_{(k-1)}(1) + \gamma^{(11)}_{in,k} + \gamma^{(11)}_{pa,k} + \beta_k(5), \\ \alpha_{(k-1)}(2) + \gamma^{(11)}_{in,k} + \gamma^{(01)}_{pa,k} + \beta_k(2), \\ \alpha_{(k-1)}(3) + \gamma^{(11)}_{in,k} + \gamma^{(10)}_{pa,k} + \beta_k(1), \\ \alpha_{(k-1)}(4) + \gamma^{(11)}_{in,k} + \gamma^{(00)}_{pa,k} + \beta_k(7), \\ \alpha_{(k-1)}(5) + \gamma^{(11)}_{in,k} + \gamma^{(11)}_{pa,k} + \beta_k(4), \\ \alpha_{(k-1)}(6) + \gamma^{(11)}_{in,k} + \gamma^{(01)}_{pa,k} + \beta_k(3), \\ \alpha_{(k-1)}(7) + \gamma^{(11)}_{in,k} + \gamma^{(10)}_{pa,k} + \beta_k(0) \end{array} \right\} - \max \left\{ \begin{array}{l} \alpha_{(k-1)}(0) + \gamma^{(00)}_{in,k} + \gamma^{(00)}_{pa,k} + \beta_k(0), \\ \alpha_{(k-1)}(1) + \gamma^{(00)}_{in,k} + \gamma^{(11)}_{pa,k} + \beta_k(3), \\ \alpha_{(k-1)}(2) + \gamma^{(00)}_{in,k} + \gamma^{(01)}_{pa,k} + \beta_k(4), \\ \alpha_{(k-1)}(3) + \gamma^{(00)}_{in,k} + \gamma^{(10)}_{pa,k} + \beta_k(7), \\ \alpha_{(k-1)}(4) + \gamma^{(00)}_{in,k} + \gamma^{(00)}_{pa,k} + \beta_k(1), \\ \alpha_{(k-1)}(5) + \gamma^{(00)}_{in,k} + \gamma^{(11)}_{pa,k} + \beta_k(2), \\ \alpha_{(k-1)}(6) + \gamma^{(00)}_{in,k} + \gamma^{(01)}_{pa,k} + \beta_k(5), \\ \alpha_{(k-1)}(7) + \gamma^{(00)}_{in,k} + \gamma^{(10)}_{pa,k} + \beta_k(6) \end{array} \right\}$$

$$= \max \left\{ \begin{array}{l} \alpha_{(k-1)}(0) + \gamma^{(00)}_{pa,k} + \beta_k(6), \\ \alpha_{(k-1)}(1) + \gamma^{(11)}_{pa,k} + \beta_k(5), \\ \alpha_{(k-1)}(2) + \gamma^{(01)}_{pa,k} + \beta_k(2), \\ \alpha_{(k-1)}(3) + \gamma^{(10)}_{pa,k} + \beta_k(1), \\ \alpha_{(k-1)}(4) + \gamma^{(00)}_{pa,k} + \beta_k(7), \\ \alpha_{(k-1)}(5) + \gamma^{(11)}_{pa,k} + \beta_k(4), \\ \alpha_{(k-1)}(6) + \gamma^{(01)}_{pa,k} + \beta_k(3), \\ \alpha_{(k-1)}(7) + \gamma^{(10)}_{pa,k} + \beta_k(0) \end{array} \right\} + \gamma^{(11)}_{in,k} - \max \left\{ \begin{array}{l} \alpha_{(k-1)}(0) + \gamma^{(00)}_{pa,k} + \beta_k(0), \\ \alpha_{(k-1)}(1) + \gamma^{(11)}_{pa,k} + \beta_k(3), \\ \alpha_{(k-1)}(2) + \gamma^{(01)}_{pa,k} + \beta_k(4), \\ \alpha_{(k-1)}(3) + \gamma^{(10)}_{pa,k} + \beta_k(7), \\ \alpha_{(k-1)}(4) + \gamma^{(00)}_{pa,k} + \beta_k(1), \\ \alpha_{(k-1)}(5) + \gamma^{(11)}_{pa,k} + \beta_k(2), \\ \alpha_{(k-1)}(6) + \gamma^{(01)}_{pa,k} + \beta_k(5), \\ \alpha_{(k-1)}(7) + \gamma^{(10)}_{pa,k} + \beta_k(6) \end{array} \right\} - \gamma^{(00)}_{in,k}.$$

Therefore, Eq. (5.1d) can be modified as:

$$\begin{aligned} \Lambda^z_{apo,k}(u_k) &= \max_{\substack{s_{k-1},s_k, \\ u_k=z}} (\alpha_{k-1}(s_{k-1}) + \gamma^m_{pa,k} + \beta_k(s_k)) + \gamma^z_{in,k} \\ &\quad - \max_{\substack{s_{k-1},s_k, \\ u_k=00}} \Big(\alpha_{k-1}(s_{k-1}) + \gamma^{(00)}_{pa,k} + \beta_k(s_k) \Big) - \gamma^{(00)}_{in,k} \\ &= \tilde{\Gamma}^z_k - \tilde{\Gamma}^{(00)}_k + \gamma^z_{in,k}, \end{aligned}$$

where $\tilde{\Gamma}^z_k = \max\limits_{\substack{s_{k-1},s_k, \\ u_k=z}} (\alpha_{k-1}(s_{k-1}) + \gamma^z_k(s_{k-1}, s_k) + \beta_k(s_k))$

Notice that $\gamma_{in,k}^{(00)} = 0$. Therefore, comparing the modified $\Lambda_{apo,k}^{z}(u_k)$ shown above with Eq. (5.1d), the newly defined $\Lambda_{apo,k}^{z}(u_k)$ introduces 3 extra addition operations. However, the computation of $\tilde{\Gamma}_k^z$ can be further simplified by extracting $\gamma_{pa,k}^{m}$ out. For example,

$$\tilde{\Gamma}_k^{(00)} = \max \left\{ \begin{array}{l} \max[\alpha_{(k-1)}(0) + \beta_k(0), \alpha_{(k-1)}(4) + \beta_k(1)] + \gamma_{pa,k}^{(00)}, \\ \max[\alpha_{(k-1)}(1) + \beta_k(3), \alpha_{(k-1)}(5) + \beta_k(2)] + \gamma_{pa,k}^{(11)}, \\ \max[\alpha_{(k-1)}(2) + \beta_k(4), \alpha_{(k-1)}(6) + \beta_k(5)] + \gamma_{pa,k}^{(01)}, \\ \max[\alpha_{(k-1)}(3) + \beta_k(7), \alpha_{(k-1)}(7) + \beta_k(6)] + \gamma_{pa,k}^{(10)} \end{array} \right\},$$

where 4 addition operations are removed from each $\tilde{\Gamma}_k^z$. An extra addition operation can be saved because $\gamma_{pa,k}^{(00)} = 0$. Therefore, the overall computational complexity of the modified equation for *a posteriori* LLR is lower than Eq. (5.1d) by 17 addition operations (*i.e.*, 5 additions $\times$ 4 metrics-3additions).

Just as we developed IMs and PAMs in Section 5.3.1, $\tilde{\Gamma}_k^z - \tilde{\Gamma}_k^{(00)}$ is further defined as Γ_k^z, and Λ_{apo} becomes

$$\Lambda_{apo,k}^{z}(u_k) = \Gamma_k^z + \gamma_{in,k}^{z}. \tag{5.6}$$

Next, we will drive a simplified expression for the extrinsic LLR. From Eq. (5.1), we know that $\Lambda_{apo,k}^{(00)}(u_k) = \Lambda_{ex,k}^{(00)}(u_k) = 0$. Substituting $\Lambda_{apo,k}^{z}(u_k)$ given by Eq. (5.6) and $\gamma_{in,k}^{z}$ given by Eq. (5.3b) into Eq. (5.1e) results in

$$\begin{aligned}
\Lambda^z_{ex,k}(u_k) &= \delta(\Lambda^z_{apo,k}(u_k) - \Lambda^z_{apr,k}(u_k) - \Lambda^z_{in,k}(u_k)) \\
&= \delta(\Gamma^z_k + \gamma^z_{in,k} - \Lambda^z_{apr,k}(u_k) - \Lambda^z_{in,k}(u_k)) \\
&= \delta\left(\Gamma^z_k + \Lambda^z_{apr,k}(u_k) + 2y^{s1}_k x^{s1}_k + 2y^{s2}_k x^{s2}_k \right. \\
&\quad \left. -(\Lambda^{(00)}_{apr,k} - 2y^{s1}_k - 2y^{s2}_k) - \Lambda^z_{apr,k}(u_k) - \Lambda^z_{in,k}(u_k)\right) \\
&= \delta(\Gamma^z_k + 2(y^{s1}_k x^{s1}_k + y^{s2}_k x^{s2}_k + y^{s1}_k + y^{s2}_k) - \Lambda^z_{in,k}(u_k)) \\
&= \delta(\Gamma^z_k + 2(y^{s1}_k(x^{s1}_k + 1) + y^{s2}_k(x^{s2}_k + 1)) - \Lambda^z_{in,k}(u_k));
\end{aligned}$$

where

$$2(y^{s1}_k(x^{s1}_k + 1) + y^{s2}_k(x^{s2}_k + 1)) = \left\{ \begin{array}{ll} 0 & (z = 00) \\ 4y^{s1}_k & (z = 01) \\ 4y^{s2}_k & (z = 10) \\ 4y^{s2}_k + 4y^{s1}_k & (z = 11) \end{array} \right\} = \Lambda^z_{in,k}.$$

Therefore,

$$\Lambda^z_{ex,k}(u_k) = \delta\Gamma^z_k. \tag{5.7}$$

Eq. (5.7) reveals that the extrinsic LLR is only related to the forward metrics α, the backward metrics β, and the parity metrics $\gamma^m_{pa,k}$. When the BMs are partitioned into IMs and PAMs, it is more convenient to calculate Γ^z_k first. Then both $\Lambda^z_{ex,k}$ and $\Lambda^z_{apo,k}$ can be easily generated. Since the computation of Γ^z_k is already included in Eq. (5.6), Eq. (5.7) removes all the addition operations in Eq. (5.1e) and results in a computation reduction of 8 substractions in calculating $\Lambda^z_{ex,k}$.

Although the above discussion is focused on Max-log-MAP algorithm, the same technique can be applied to log-MAP algorithm by using Eq. (5.5).

5.3.3 Complexity analysis

The modified log-MAP algorithm is summarized in Eq. (5.8), where the calculation for all the state metrics and output LLR are expressed as a function of IMs and PAMs.

$$\begin{cases} \tilde{\gamma}^z_{in,k} = \Lambda^z_{apr,k}(u_k) + 2y^{s1}_k x^{s1}_k + 2y^{s2}_k x^{s2}_k; \\ \gamma^z_{in,k} = \tilde{\gamma}^z_{in,k} - \tilde{\gamma}^{(00)}_{in,k}; \end{cases} \tag{5.8a}$$

$$\begin{cases} \tilde{\gamma}^m_{pa,k} = 2y^{p1}_k x^{p1}_k + 2y^{p2}_k x^{p2}_k; \\ \gamma^m_{pa,k} = \tilde{\gamma}^m_{pa,k} - \tilde{\gamma}^{(00)}_{pa,k}; \end{cases} \tag{5.8b}$$

$$\alpha_k(s_k) = \max_{s_{k-1}}(\gamma^z_{in,k}(s_{k-1}, s_k) + \gamma^m_{pa,k}(s_{k-1}, s_k) + \alpha_{k-1}(s_{k-1})); \tag{5.8c}$$

$$\beta_k(s_k) = \max_{s_{k+1}}(\gamma^z_{in,k+1}(s_k, s_{k+1}) + \gamma^m_{pa,k+1}(s_k, s_{k+1}) + \beta_{k+1}(s_{k+1})); \tag{5.8d}$$

$$\begin{cases} \tilde{\Gamma}^z_k = \max\limits_{\substack{s_{k-1}, s_k, \\ u_k = z}} (\alpha_{k-1}(s_{k-1}) + \gamma^z_k(s_{k-1}, s_k) + \beta_k(s_k)); \\ \Gamma^z_k = \tilde{\Gamma}^z_k - \tilde{\Gamma}^{(00)}_k; \end{cases} \tag{5.8e}$$

$$\Lambda^z_{apo,k}(u_k) = \Gamma^z_k + \gamma^z_{in,k}; \tag{5.8f}$$

$$\Lambda^z_{ex,k}(u_k) = \delta\ \Gamma^z_k; \tag{5.8g}$$

It is necessary to evaluate the computational overhead of the modified log-MAP algorithm over the conventional scheme described by Eq. (5.1). According to Fig. 5.3, for the calculation of α and β, the decoder needs to regenerate 16 BMs. In fact, as we defined in Eq. (5.3), one BM must equal 0, which means that at least one addition operation can be saved when computing the values of new α or β. Furthermore, among the other 15 BMs, only 9 of them really need adders to compute. The overall computational overhead is $8 \times 2 = 16$ addition operations. Table I lists the computational overhead of each type of metrics or process in the modified MAP algorithm in comparison with Eq. (5.1), where a substraction is counted as an addition operation. The overall computational complexity of the modified MAP algorithm is less than that of the conventional one.

TABLE 5.1: Summary of the computational overhead

	Computational overhead (addition operations)
IMs	3
PMs	3
a posteriori LLR	-17
extrinsic LLR	-8
warm up	0
α	8
β	8
in totall	-3

5.3.4 Proposed architecture for low-complexity SISO decoder

Just as the conventional SW MAP decoding flow shown in Fig. 5.3, the proposed scheme has 3 major stages:

1. Time slot 0 to $D-1$: the IMs and PAMs for Block 0 are computed by using Eq. (5.3) and then stored in the memory; the decoder performs the "*warm-up*" process with Eq. (5.8d) for Block 0;

2. Time slot D to $2D-1$: "warm-up" process is further performed for Block 1, and the IMs and PAMs for Block 1 are saved; in the mean time, the decoder calculates α in the forward order for Block 0 from the stored IMs and PAMs using Eq. (5.8c), and α are then stored in the memory;

3. Time slot $2D$ to $3D-1$: "warm-up" Block 2, and the new IMs and PAMs are stored; α for Block 1 are calculated and saved; the backward metrics β for Block 0 are computed from the saved IMs and PAMs in the reverse order by using Eq. (5.8d)

again; then either $\Lambda^z_{ex,k}$ or $\Lambda^z_{apo,k}$ for Block 0 is computed by Eq. (5.8g) or Eq. (5.8f).

The proposed VLSI architecture for the SISO decoder with modified log-MAP algorithm is shown in Fig. 5.5. In conventional MAP decoding shown in Fig. 5.4, the *a posteriori* LLRs $\Lambda^z_{apo,k}$ are generated first as given by Eq. (5.1d); then the extrinsic metrics $\Lambda^z_{ex,k}$ are obtained from by using Eq. (5.1e). In the proposed architecture, $\Lambda^z_{ex,k}$ are calculated directly from Γ^z_k. Therefore, the circuits for computing $\Lambda^z_{apo,k}$ (circled by the dash line in Fig. 5.5) could be powered down in the first 2 or 3 decoding iterations, and are only enabled when the decoder is going to output the hard decision.

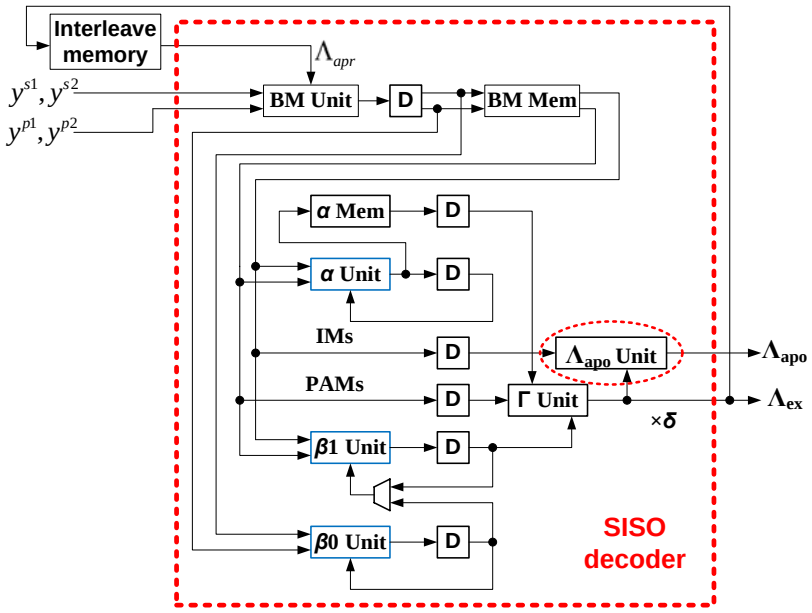


FIGURE 5.5: Proposed VLSI architecture for SISO decoder.

5.3.5 Memory organization

Given the length of a data frame N, the maximum value of N for WiMAX system is 2400, which results in 1200 pairs of data. Since the received symbols will be reused during the iterative MAP decoding, the decoder needs to save all of them.

According to the decoding flow in Fig. 5.3, the conventional SISO decoder shown in Fig. 5.4 requires memories with a length of $2D$ to save α and $3D$ for BMs. Usually, the bit-lengths of the state metrics and the BMs are set to be the same. For example, in [47]and [49] the word-length of all the metrics is 10 bits. Therefore, for conventional MAP algorithm the size of "BM Mem" is $3D \times 16 \times 10$ and the size of "α Mem" is $2D \times 8 \times 10$. The tracing back technique for α [49] can only save limited hardware, and the size of "α Mem" is reduced to $2D \times 6 \times 10$ by saving 6 difference metrics, instead of 8 BMs. However, the "BM Mem" remains the same. In the proposed architecture, the forward state metrics are still quantized as 10 bits. The IMs and PAMs have smaller word-lengths of 9 bits and 6 bits, respectively, since each of them is only a portion of BM. Recall that the number of virtual values of IMs or PAMs is only 3 as indicated in Eq. (5.3). the size of "BM Mem" is reduced to $3D \times 3 \times 15$, which is less than 1/3 of the "BM Mem" in Fig. 5.4. The details of memory organization for DB CTC decoder are summarized in Table 5.2.

TABLE 5.2: Summary of the memory size

	Branch Metrics	Forward Metrics	Total
conventional architecture	$3D \times 16 \times 10$	$2D \times 8 \times 10$	$D \times 640$
Architecture in [49]	$3D \times 16 \times 10$	$2D \times 6 \times 10$	$D \times 600$
Proposed architecture	$3D \times 3 \times 9+$ $3D \times 3 \times 6$	$2D \times 8 \times 10$	$D \times 295$

Table 5.2 shows that the proposed architecture reduces more than 1/2 of the memory inside the SISO decoder by decomposing BMs, while the tracing back architecture proposed in [49] can only save 1/16 of the memory compared with the conventional method.

The proposed method can in general be applied to other non-binary CTC decoders for memory-reduced architecture design. Moreover, by incorporating with other memory reduced techniques such as those described in [43–46] and [47–49], even more efficient architecture could be constructed.

In principle, the memory for storing BMs can be removed completely. When computing β and Γ_k^z, $\gamma_{in,k}^z$ and $\gamma_{pa,k}^m$ can be recovered directly from y_k^{si}, y_k^{pi} and $\Lambda_{apo,k}^z$. However, the recovering process will cause a considerable computational overhead and a much longer logic delay in the critical path for recursively generating β, which reduces the clock speed of the entire decoder. Therefore, it is rarely adopted in practical implementations.

5.4. High-speed Architecture for TL log-MAP algorithm

In this section, we focus on the recursion architecture design for calculating α and β (see α and $\beta 1$ units in Fig. 5.5), since it is the high-speed bottleneck of the SISO decoder. Mathematically, the two units implement the same function: the log-MAP function or its variations. Therefore, in the rest of the section, we will only discuss the design for α unit; the proposed scheme can be directly applied to the other one.

In the proposed SISO decoder shown in Fig. 5.5, the critical path of α unit is long for two reasons: 1) in general, the critical path of a SISO decoder for the DB CTC is inherently longer than that for the SB CTC because the number of candidate paths reaching the same state is increased; 2) in the modified log-MAP algorithm expressed in Eq. (5.8), in the worst case, the decoder needs an extra adder in the recursion loop to recover a BM from the IM and PAM, which makes the critical path even longer. We will explore the high-speed solution, which can overcome both problems.

Among all the decoding algorithms we mentioned in Section 5.2, the MAX-log-MAP algorithm has the lowest computational complexity. However, being a sub-optimal algo-

rithm, with the same number of decoding iterations, it has considerable BER performance loss compared with the log-MAP algorithm. With the scaling factor δ, the performance of MAX-log-MAP algorithm can be improved, though the loss is still obvious (at least 0.1 dB in general). Therefore, the log-MAP algorithm is preferred for high-performance systems. The essential part of log-MAP decoding is the max* operation shown in Eq. (5.2). Efficient implementation of the max* operation for SB CTC decoder has been widely studied. Since the number of inputs to max* operation in SB CTC decoder is only 2, Eq. (5.2) can be simplified as

$$\begin{aligned} \max^*(a,b) &= \ln(e^a + e^b) \\ &= \ln(e^{\max(a,b)} \times (1 + e^{-|a-b|})) \\ &= \max(a,b) + \ln(1 + e^{-|a-b|}), \end{aligned} \tag{5.9}$$

where the second term is nonlinear and is usually implemented by a look-up-table (LUT). The corresponding VLSI architectmax*(a,b) operation is shown in Fig. 5.6

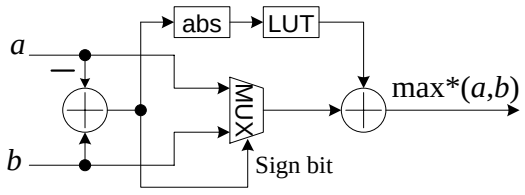


FIGURE 5.6: VLSI architecture for max*(a,b) operation.

As we introduced in Section 5.1, TL log-MAP scheme is a good approximation of log-MAP algorithm, since it introduces almost no performance loss when an enhanced scaling factor is employed. The VLSI architecture of TL log-MAP algorithm is shown in Fig. 5.7, where four 2-input comparators are used to determine the largest and the second largest values among 4 candidates. This straightforward implementation is not a good

choice for practical applications, because finding the two largest candidates usually takes a long time.

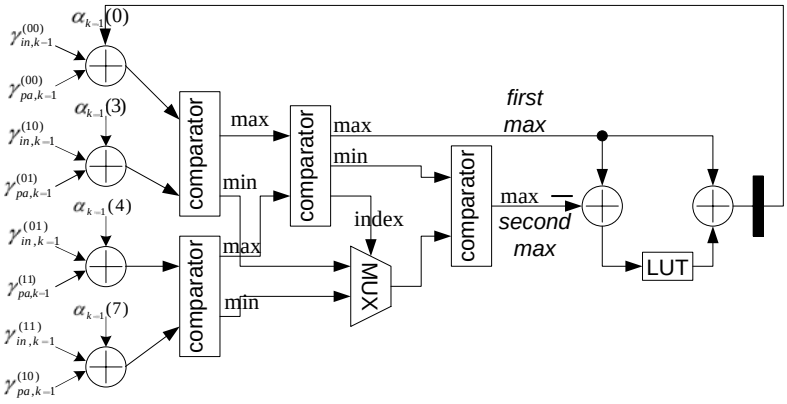


FIGURE 5.7: VLSI architecture for TL log-MAP algorithm.

5.4.1 Proposed Scheme-I

There are two algorithms that can be used as an approximation to TL log-MAP algorithm. The first one is proposed by Lucent [57] shown in Eq. (5.10), which is originally used for Radix-4 decoding of SB CTCs. In Eq. (5.10), the 4 candidates are divided into two groups and the maximum values of each group are treated as the "two largest values". Compared with the real TL log-MAP algorithm, this approximation still selects the real largest value, but the other value could be the second or the third largest one. This approximation has almost the same BER performance as TL log-MAP algorithm, while the corresponding VLSI architecture has a shorter logic delay since a comparator to find the real "second largest value" is removed from the critical path.

$$\max^*(a,b,c,d) = \max^*(\max(a,b),\max(c,d)). \tag{5.10}$$

The first high-speed recursion architecture we proposed is based on Eq. (5.10). However, it is very different from the straightforward implementation of Eq. (5.10). This

architecture, as shown in Fig. 5.8, employs both existing schemes and novel structures which further reduce the critical path. First, we replace the abs module and LUT with a generalized LUT (GLUT) module [58], which avoids the computation of the absolute value. Then we split each state metric into two portions [58]. For example, $\alpha_k(0) = \alpha'_k(0) + \delta_0$, where $\alpha'_k(0)$ is the comparison result from MUX and δ_0 is the nonlinear portion from GLUT. In Fig. 5.8, the recursion architecture feeds back $\alpha'_k(0)$ and δ_0 to the front end separately, which enables the data compression at the front end and moves the addition of $\alpha'_k(0)$ and δ_0 outside of the loop.

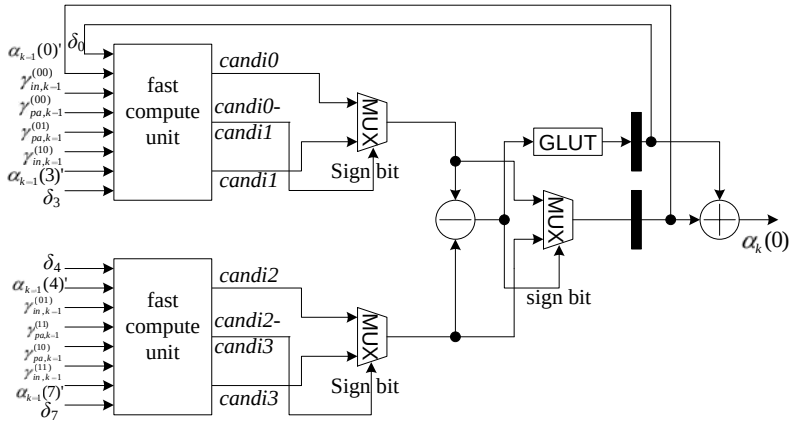


FIGURE 5.8: High-speed architecture for TL MAP algorithm: Scheme-I.

In order to further reduce the critical path, we propose a "fast compute unit", which computes two candidates and generates the difference between them. The block diagram for the "fast compute unit" is shown in Fig. 5.9. Due to the modification of log-MAP algorithm and the splitting of the state metric, each candidate now has 4 input data and requires 3 adders to calculate as shown in Fig. 5.9. Usually, to add 4 data, the logic delay equals two n-bits adder, where n is the word length of the input. In Fig. 5.9, the "4:2 compress A" module will compress the 4 inputs to two numbers with a delay of only two 1-bit full adders. The corresponding VLSI architecture of the module is shown in

Fig. 5.10, where HA stands for a half adder and FA stands for a full adder.

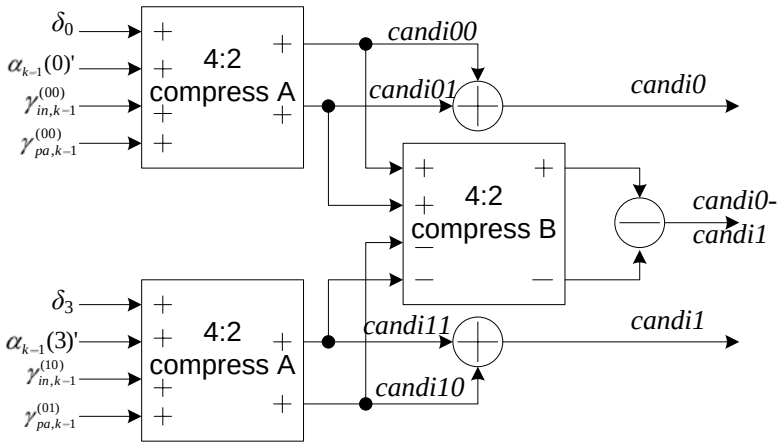


FIGURE 5.9: Proposed fast compute unit.

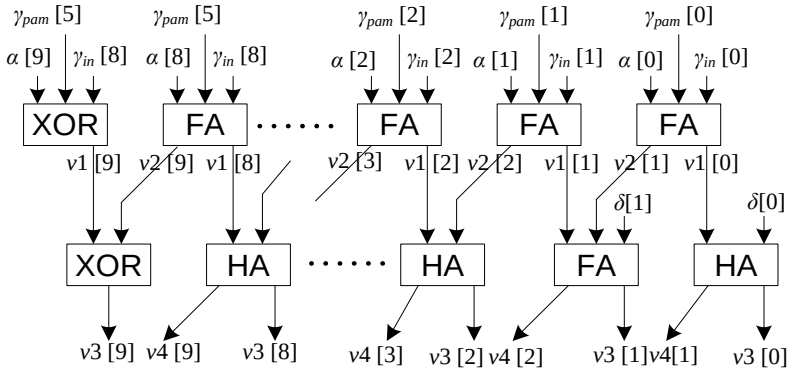


FIGURE 5.10: The "4:2 compress A" module.

Usually, the comparator between *candi*0 and *candi*1 should wait until the two numbers are available. Then, the difference between the two is calculated, and the decision is made based on the sign of the difference. Here, we explore a new architecture that can calculate the difference between *candi*0 and *candi*1 without knowing *candi*0 and *candi*1. We observe that, the outputs of the two "4:2 compress A" modules form another group of 4 numbers, and $candi0 - candi1 = candi00 + candi01 - candi10 - candi11$. Hence

we use a different 4 to 2 compressor - "4:2 compress B" to compress the four numbers; then the difference is calculated by a substractor as shown in Fig. 5.9. The delay of "4:2 compress B" is so small (no more than a 2-bit adder or the delay of 4 XOR gates), that the computation of $candi0 - candi1$ is executed almost at the same time with the calculation of $candi0$ and $candi1$. Therefore, the logic delay in the recursion architecture is further reduced.

A straightforward architecture of the "4:2 compress B" module is shown in Fig. 5.11, which consists of the borrow-saving structure with PPM and MMP cells [59]. The PPM cell allows one to compress 3 bits operation into 2 bits as follows:

$$a[i] + b[i] - c[i] = 2 \cdot cout[i+1] - sum[i]. \tag{5.11}$$

Similarly, MMP cells precess the following operation:

$$-a[i] - b[i] + c[i] = -2 \cdot cout[i+1] + sum[i]. \tag{5.12}$$

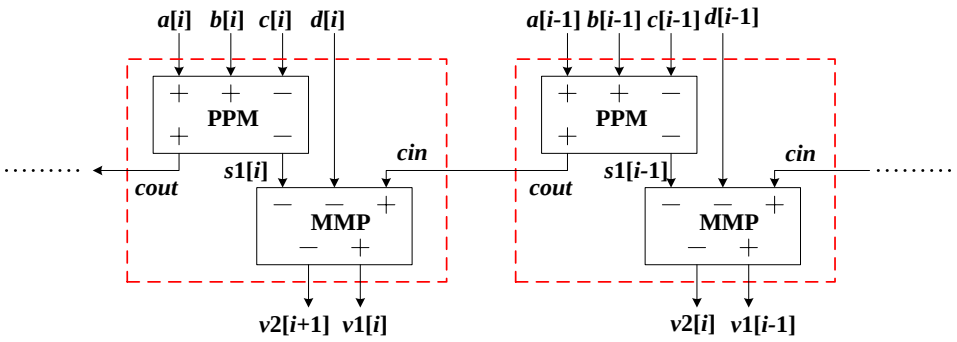


FIGURE 5.11: The "4:2 compress B" module built with PPM and MMP.

Note that, logically Eq. (5.11) and Eq. (5.12) are virtually identical. Therefore, at the gate level, MMP and PPM cells have the same logic that,

$$sum = a \oplus b \oplus c; \qquad cout = (a+b)\bar{c} + ab; \tag{5.13}$$

where the logic delay of "*sum*" is 2XOR, and the logic delay for "*cout*" is about 1.5XOR. Consequently, the overall logic delay of Fig. 5.11 is 4XOR (4XOR for $v1[i]$ and 3.5XOR for $v2[i+1]$).

Here, we propose a high-speed architecture for "4-2 compress B" module by re-designing the bit-level cell. First, we re-express the *cout* as:

$$cout = (a+b)\bar{c} + ab = (a \oplus b)\bar{c} + \overline{a \oplus b} \cdot b; \tag{5.14}$$

Then, $v2[i+1]$ and $v1[i]$ are derived by:

$$p[i] = s1[i] \oplus d[i] = (a[i] \oplus b[i]) \oplus (c[i] \oplus d[i]); \tag{5.15a}$$

$$v1[i] = s1[i] \oplus d[i] \oplus cin = p[i] \oplus cin; \tag{5.15b}$$

$$v2[i+1] = (s1[i] \oplus d[i])\overline{cin} + \overline{s1[i] \oplus d[i]} \cdot d[i] = p[i]\overline{cin} + \overline{p[i]} \cdot d[i]. \tag{5.15c}$$

The corresponding bit-level logic cell is illustrated in Fig. 5.12, where the overall logic delay is only 3XOR (3XOR for $v2[i+1]$ and 2.5XOR for $v1[i]$).

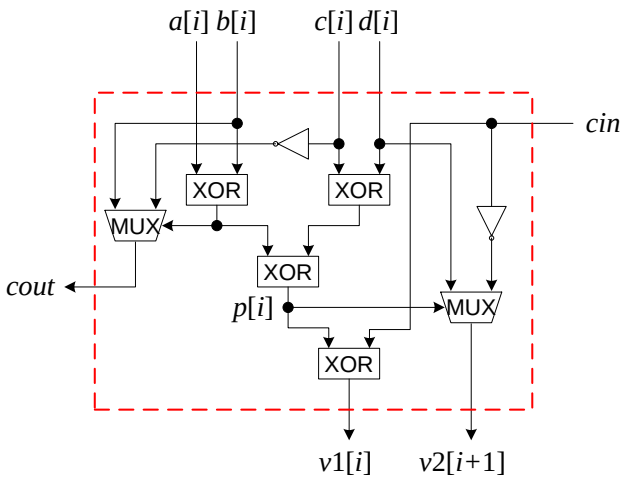


FIGURE 5.12: Proposed bit-level cell for the "4:2 compress B" module.

It is worth mentioning that, in reality the substractor shown in Fig. 5.8 is implemented with a standard adder in the library, where the inverse of the minuend $v2[i+1]$

is more welcomed. In that case, the output of $\overline{v2[i+1]}$ should be used. Eq. (15b) can be modified as $\overline{v2[i+1]} = \overline{p[i]cin} + p[i] \cdot d[i]$, where the logic delay is still 3 XOR.

5.4.2 Proposed Scheme-II

The second high-speed recursion architecture is built from the approximation proposed by Wang [58], which is expressed as

$$\max^*(a,b,c,d) = \max(\max^*(a,b), \max^*(c,d)). \tag{5.16}$$

Just as Scheme-I, simulation result of this approximation does not show observable performance loss from the real TL log-MAP algorithm. Compared with approximation I given by Eq. (5.8), this scheme requires a slightly larger hardware area because there are two max* operations in the approximation while scheme I has only one. On the other hand, by further approximation, the recursion architecture of Scheme-II will have a shorter critical path than Scheme-I. The proposed Scheme-II based on Eq. (5.16) is shown in Fig. 5.13.

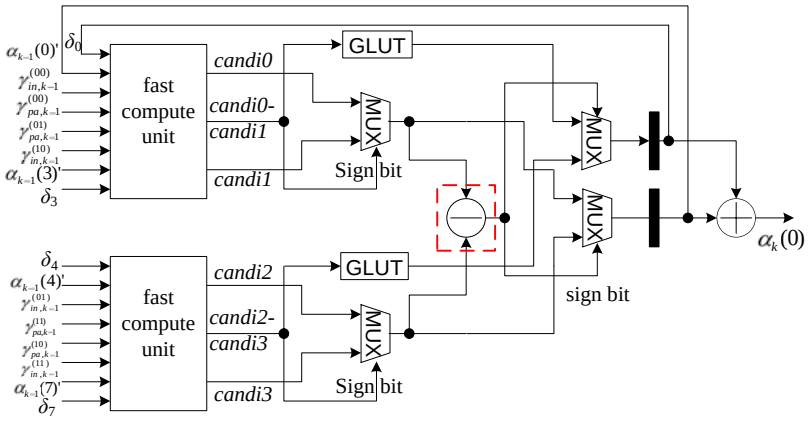


FIGURE 5.13: High-speed architecture for TL MAP algorithm: Scheme-II.

In Fig. 5.13, we employed the GLUT and the fast computation unit adopted for Scheme-I, which improves the speed without changing the function. However, the ar-

chitecture is an approximated version of Eq. (5.16). When we calculate the maximum value between $\max^*(a,b)$ and $\max^*(c,d)$ in Fig. 5.13, actually, the decision criterion is "$\max(a,b) - \max(c,d)$", rather than "$\max^*(a,b) - \max^*(c,d)$". By this approximation, we remove two adders from the recursion architecture. In the mean time, the critical path can be further reduced. Compared with max(a,b), the nonlinear portion from GLUT is so small (usually quantized into 2 bits) that whether or not including it in the decision criterion will generally not change the decision result, or in case there is a wrong decision, the values of the two candidates must be very close that the wrong decision will not affect the final decoding result.

5.4.3 Implementation result

For comparison of hardware complexity and decoding speed, we use Fujitsu 0.13-μm standard cells to synthesize the α unit in Fig. 5.5 with three different recursion architectures: 1) The straightforward implementation of TL log-MAP algorithm shown in Fig. 5.7; 2) the proposed high-speed architecture based on Lucent's approximation shown in Fig. 5.8: Scheme-I; and 3) the proposed high-speed architecture based on Wang's approximation in Fig. 5.13: Scheme-II. All the state metrics are quantized as 10 bits, the IMs are quantized as 9 bits, and the PAMs are quantized as 6 bits. Synthesis is optimized for speed, and the results are shown in Table 5.3.

TABLE 5.3: Comparison for several architectures

	Max Clock Freq. (MHz)	Hardware (gate count)
TL log-MAP	220	28075
Proposed Scheme-I	327	24436
Proposed Scheme-II	330	24484

Table 5.3 clearly shows that, the proposed high-speed architectures can improve the clock speed up to 50% compared with the straightforward implementation of the TL

log-MAP algorithm. We also notice that the hardware of the two proposed schemes is about 14% smaller than the TL log-MAP algorithm. The results make sense since the sorting process in the TL log-MAP algorithm is removed from the recursion architecture. The synthesis results between the two proposed schemes basically match our expectation, though the clock speed of Scheme-II is only slightly higher than that of Scheme-I. In brief, both of the proposed schemes can be used in high-speed decoders for DB CTCs.

5.5. Conclusion

In this Chapter, several novel ideas for VLSI design of the DB CTC decoders are presented. First, we proposed a memory-reduced VLSI architecture as well as a modified MAP algorithm without performance loss and computational overhead for DB CTC. The key idea is to decompose the BMs into IMs and PAMs. By storing IMs and PAMs, rather than BMs, to the metrics memory, the overall saving of memory inside the SISO decoder is more than 50%, while other schemes focusing on forward metrics can only achieve 1/12 memory reduction. As the BMs are decomposed, the MAP algorithm is modified to be expressed by IMs and PAMs, instead of BMs. The newly defined MAP algorithm eliminates unnecessary computation. As a result, the extrinsic metrics become independent of the *a posteriori* LLR, while in existing works, the decoders generate extrinsic metrics from the *a posteriori* LLR. The corresponding VLSI architecture for the modified MAP algorithm has a good potential for low-power implementation. We also explored two high-speed recursion architectures for DB CTC decoders employing TL log-MAP algorithm by algorithmic approximation and bit level optimization. Synthesis results have shown that, compared with the TL log-MAP algorithm, the two proposed schemes could improve the clock speed by 50% with less hardware.

6. CONCLUSIONS AND FUTURE WORK

6.1. Conclusions

This dissertation focuses on efficient VLSI implementation of decoders for ECCs. Specifically, high-speed and low-complexity (low-power) schemes are proposed for the Viterbi decoder, TCM decoder, and DB turbo decoder.

In Chapter 3, we presented the VLSI design of a codec for a 4-D 8PSK TCM system. We first developed a configurable encoder which uses the same convolutional encoder and constellation mapper to deal with 4 different modulation rates. Then more focus was put on the efficient VLSI design of the 4-D 8PSK TCM decoder. For TMU, the proposed architecture reduces the computational complexity to about 1/3 of an existing low complexity architecture presented in [9] while the BER performance is not compromised. For VD, we proposed a new structure for T-algorithm applied on BMs. Compared with the conventional T-algorithm applied on PMs, the new scheme has the advantages of lower complexity and higher throughput. Based on the proposed T-algorithm on BMs and the conventional T-algorithm on PMs, we further designed a hybrid T-algorithm scheme. Simulation results show that the hybrid T-algorithm could further reduce more than half of the computations compared with the conventional T-algorithm while still maintaining the same BER performance. Finally, we proposed a new architecture for hybrid T-algorithm VD based on SPEC-T algorithm that can retain the speed advantage of the conventional scheme. Implementation results in FPGA validated the excellent performance of the proposed algorithms.

In Chapter 4 we extended our research to a more general topic of Viterbi decoder

design. We proposed a pre-computation scheme and the associated hardware architecture for Viterbi decoders employing T-algorithm. Compared with existing schemes that target to efficiently implement T-algorithm, the proposed scheme is more reliable for high-rate codes. An analysis of the critical path reveals that the pre-computation scheme can achieve the iteration bound for Viterbi decoders employing T-algorithm with negligible hardware overhead. Simulation results show that the proposed scheme maintains the same BER performance as the conventional T-algorithm while other schemes could lose all the survivor paths during the decoding process. Synthesis results in FPGA verify the speed advantages of the proposed pre-computation scheme, and power estimation result through ASIC synthesis verifies the low-power property of T-algorithm. We also presented two efficient trace-back schemes to improve the decoding latency of the conventional trace-back approach. It is shown that the decoding latency can be reduced from $3L$ to $2L$ at the expense of a small hardware overhead and a slightly increased power consumption. However, the power consumption overhead could be largely compensated by adopting a buffer-based trace-back scheme. Compared to the conventional RE, the latency of the proposed Scheme-II is still L cycle longer while the overall power consumption of our design is significantly lower.

Chapter 5 of this dissertation is dedicated to high-speed, memory reduced SISO decoder design for DB turbo codes. We proposed a memory-reduced VLSI architecture as well as a modified MAP algorithm without performance loss and computational overhead for DB CTC. The key idea is to decompose the BMs into IMs and PAMs. By storing IMs and PAMs, rather than BMs, to the metrics memory, the overall saving of memory inside the SISO decoder is more than 50%, while other schemes focusing on forward metrics could achieve 1/12 memory reduction only. As the BMs are decomposed, the MAP algorithm is modified to be expressed by IMs and PAMs, instead of BMs. The newly defined MAP algorithm eliminates unnecessary computation. As a result, the extrinsic metrics become

independent of the *a posteriori* LLR, while in conventional schemes, the decoders generate extrinsic metrics from the *a posteriori* LLR. The corresponding VLSI architecture for the modified MAP algorithm has a good potential for low-power implementation. We have also explored two high-speed recursion architectures for DB CTC decoders employing TL log-MAP algorithm by algorithmic approximation and bit-level optimization. Synthesis results have shown that, compared with the TL log-MAP algorithm, the two proposed schemes can improve the clock speed by 50% with less hardware.

6.2. Future Work

Soft-output decoding of convolutional codes has in general been applied to only CTCs, which require an iterative decoding process. For low-complexity decoding, CTCs usually employ convolutional codes with short constraint lengths (e.g., less than 5). For the decoding of a single convolutional code or TCM, the Viterbi algorithm is still the most popular method because it only introduces very little performance loss compared with the MAP algorithm which provides soft-output, while its computational complexity is much lower than the latter. Recently, concatenated forward error correction codes are commonly used, which connect two different types of codes in series to enhance the error-correcting capability. For example, in the WiMAX standard, the concatenated FEC is based on a Reed-Solomon outer code and a rate-compatible TCM inner code (constraint length equals 7). As we introduced in Chapter 2, soft-input decoding in general would gain 2 to 3 dB over the hard-input decoding. Thus, it is sometimes desirable for the decoder of the inner code to output soft information. However, a straightforward implementation of a SISO decoder for convolutional codes with long constraint lengths would be very complex. Our research work can be extended to the low complexity decoding of convolutional codes or TCM with long constraint lengths. Improvements could be achieved at both the algorithm

and architecture levels. Extensive efforts are needed for such design.

BIBLIOGRAPHY

1. Shu Lin, and Daniel J. Costello, *Error Control Coding (2nd Edition).* Englewood Cliffs, NJ: Prentice-Hall, 2004.

2. I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Andustrial and Applied Mathematics*, vol. 8, pp. 300–304, June 1960.

3. R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, vol. IT-8, pp. 21–28, Jan. 1962.

4. Y. Gang, A. T. Erdogan and T. Arslan, "An efficient pre-traceback architecture for the viterbi decoder targeting wireless communication applications," *IEEE Trans. Circuits syst. I, Reg. Papers.*, vol. 53, pp. 1918–1927, Sept. 2006.

5. J. P. Woodard and L. Hanzo, "Comparative study of turbo decoding techniques: An overview," *IEEE Trans. Veh. Technol.*, vol. 49, pp. 2208–2233, Nov. 2000.

6. J. Vogt and A. Finger, "Improving the max-log-map turbo decoder," *Electronics Lett.*, vol. 36, pp. 1937–1938, Nov. 2000.

7. K. L. Hsiung, S. J. Kim, and S. Boyd, "Tractable approximate robust geometric programming," *Optimization and Engineering, 9(2)*, pp. 95–118, June 2008.

8. S. S. Pietrobon and R. H. Deng, "Trellis-coded multidimensional phase modulation," *IEEE Trans. Inf. Theory.*, vol. 36, pp. 63–89, Jan. 1990.

9. D. Servant, "Design and analysis of a multidimensional trellis coded demodulator," 2004. Master Thesis, Department of Signals, Sensors and Systems, KTH, Stockholm, Sweden.

10. J. He, Z. Wang, and H. Liu, "Low-complexity high-speed 4-D 8PSK TCM decoder," in *Proc. IEEE 2008 SiPS*, pp. 3030–3033, Oct. 2008.

11. J. He, Z. Wang, and H. Liu, "An efficient 4-D 8PSK TCM decoder architectures," to appear in *IEEE Trans. Very Large Scale Intergr. (VLSI) Syst.*, 2010.

12. J. He, H. Liu, and Z. Wang, "A fast acsu architecture for viterbi decoder using t-algorithm," in *Proc. 43th IEEE Asilomar Conf. on Signals, Systems, and Computers*, (Asilomar CA), Nov. 2009.

13. J. He, H. Liu, Z. Wang, Xinming Huang, and Kai Zhang, "A High-Speed Low-Power Viterbi Decoder for High-Rate Convolutional Codes," submitted to *IEEE Trans. Circuits. Syst. II.*

14. J. He, Z. Cui, Z. Wang, and L. Li, "Towards an optimal trade-off of Viterbi decoder design," in *Proc. IEEE ISCAS 2009*, pp. 216–220, May 2009.

15. J. He, Z. Wang, and H. Liu, "Low-complexity memory-reduced map decoding for double-binary convolutional turbo code," to appear in *Proc. IEEE ISCAS 2010*, May, 2010.

16. M. Sybis, P. Tyczka, S. Papaharalabos, and P. T. Mathiopoulos, "Reduced-complexity algorithms for near-optimal decoding of turbo tcm codes," *Electronics Lett.*, vol. 45, pp. 278–279, Feb. 2009.

17. J. He, Z. Wang, and H. Liu, "High-speed memory-reduced soft-in soft-out decoder for double-binary convolutional turbo code," submitted to *IET on Circuits , systems and Devices*

18. S. G. Wilson, *Digital Modulation and Coding.* Englewood Cliffs, NJ: Prentice-Hall, 1996.

19. C. Berrou, A. Clavieux and P. Thitimajshia, "Near Shannon limit error-correcting coding and decoding: Turbo codes," in *Proc. IEEE ICC'93*, vol. 2, pp. 1064–1070, Jan. 1993.

20. J. G. Proakis, *Digital Communications, Fourth Edition.* New York: McGraw-Hill, 2001.

21. G. D. Forney. Jr., "The Viterbi algorithm," in *Proc. IEEE*, vol. 61, no. 3 pp. 268–278, Mar. 1973.

22. G. Ungerboeck, "Trellis-coded modulation with redundant signal sets part I: Introduction," *IEEE Trans. Commun.*, vol. 25, pp. 5–11, Feb. 1987.

23. G. Ungerboeck, "Channel coding with multilevel/phase signals," *IEEE Trans. Inf. Theory.*, vol. IT-28, pp. 55–67, Jan. 1990.

24. C. Douillard and C. Berrou, "Turbo codes with rate-$m/(m+1)$ constituent convolutional codes," *IEEE Trans. Commun.*, vol. 53, pp. 1630–1638, Oct. 2995.

25. "Digital video broadcasting (dvb)." [Online]. Available: http://www.dvb.org/.

26. "IEEE standard for local and metropolitan area networks - part 16: Air interface for fixed broadband wireless access systems," Oct. 2004.

27. "Bandwith-efficient modulations," Apr. 2003. CCSDS 401(3.3.6) Green Book.

28. C. F. Lin and J. B. Anderson, "M-algorithm decoding of channel convolutional codes," in *Proc. Princetion Conf. Inform. Sci. Syst.*, (Princeton, NJ), Mar. 1986.

29. S. J. Simmons, "Breadth-first trellis decoding with adaptive effort," *IEEE Trans. Commun.*, vol. 38, no. 1, pp. 3–12, 2005.

30. F. Chan and D. Haccoun, "Adaptive viterbi decoding of convolutional codes over memoryless channels," *IEEE Trans. Commun.*, vol. 45, pp. 1389–1400, Nov. 1997.

31. F. Sun and T. Zhang, "Parallel high-throughput limited search trellis decoder vlsi design," *IEEE Trans. Very Large Scale Intergr. (VLSI) Syst.*, vol. 13, pp. 1013–1022, Sept. 2005.

32. R. M. Orndorff, P. C. Chou, J. D. Kromatick, T. W. Colesworthy, T. W. Doak, and R. Koralek, "Viterbi decoder vlsi integrated circuit for bit error correction," in *Proc. Nat. Telecommun. Conf.*, (New Orleans, LA), Dec. 1981.

33. "Memory management in traceback Viterbi deocders," Nov. 1989. DA Progress Report 42-99, Jet Propulsion Lab., Pasadena, CA.

34. G. Feygin and P. G. Gulak, "Architectureal tradeoffs for survivor sequence memory management in viterbi decoders," *IEEE Trans. Commun.*, vol. 41, pp. 425–429, Mar. 1993.

35. R. C. S. Morling and N. Haron, "Novel low-latency low-power viterbi decoder traceback memory architecture," in *Proc. IEEE MELECON*, (Croatia), May 2004.

36. C.-C. Lin, Y.-H. Shih, H.-C. Chang, and C.-Y. Lee, "Design of a power-reduction viterbi decoder for WLAN applications," *IEEE Trans. Circuits syst. I, Reg. Papers.*, vol. 52, pp. 1148–1156, June 2005.

37. F. Sun and T. Zhang, "Low power state-parallel relexed adaptive viterbi decoder design and implementation," in *Proc. IEEE ISCAS*, May 2006.

38. A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inf. Theory.*, vol. IT-13, No. 2, pp. 260-269, Apr. 1967.

39. C. Berrou and M. Jezequel, "Non-binary convolutional codes for turbo coding," *Electronics Lett.*, vol. 35, pp. 39–40, Jan. 1999.

40. L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inf. Theory.*, vol. IT-20, No. 2, pp. 284-287, Mar. 1974.

41. P. Robertson E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal map decoding algorithms operation in the log domain," in *Proc. IEEE Int. Conf. Commun. (ICC)*, (Seattle, WA, USA), June 1995.

42. C. Schurgers F. Catthoor, and M. Engels, "Memory optimization of map turbo decoder algorithms," *IEEE Trans. Very Large Scale Intergr. (VLSI) Syst.*, vol. 9, pp. 305–312, Sept. 2001.

43. Z. Wang Z. Chi, and K. K. Parhi, "Area-efficient high-speed decoding schemes for turbo decoders," *IEEE Trans. Very Large Scale Intergr. (VLSI) Syst.*, vol. 10, pp. 902–912, Dec. 2002.

44. T.-H. Tsai, C.-H. Lin, and A.-Y. Wu, "A memory-reduced log-map kernel for turbo decoder," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, vol. 2, pp. 1032–1035, May 2005.

45. D.-S. Lee and I.-C. Park, "Low-power log-map decoding based on reduced megtric memory access," *IEEE Trans. Circuits syst. I, Reg. Papers.*, vol. 53, pp. 1244–1253, June 2006.

46. H.-M. Choi, J.-H. Kim, and I.-C. Park, "Low-power hybrid turbo decoding based on reverse calculation," *IEICE Trans. Fund. Electrons. Comm. Comput. Sci.*, vol. E89-A, pp. 782–789, Mar. 2006.

47. J.-H. Kim and I.-C. Park, "Double-binary circular turbo decoding based on border metric encoding," *IEEE Trans. Circuits. Syst. II, Briefs*, vol. 55, pp. 79–83, Jan. 2008.

48. J.-H. Kim and I.-C. Park, "Bit-level extrinsic information exchange method for double-binary turbo codes," *IEEE Trans. Circuits. Syst. II,Briefs*, vol. 56, pp. 81–85, Jan. 2009.

49. C.-H. Lin, C.-Y. Chen, A.-Y. Wu, and T.-H. Tsai, "Low-power memory-reduced traceback map decoding for double-binary convolutional turbo decoder," *IEEE Trans. Circuits syst. I, Reg. Papers.*, vol. 56, pp. 1005–1015, May 2009.

50. M. Sybis, P. Tyczka, S. Papaharalabos, and P. T. Mathiopoulos, "Reduced-complexity algorithms for near-optimal decoding of turbo tcm codes," *Electronics Lett.*, vol. 45, pp. 278–279, Feb. 2009.

51. C. Weiss, C. Bettstetter, S. Riedel, and D. J. Costello, "Turbo decoding with tailbiting trellises," in *Proc. IEEE Int. Symp. Signals, Syst. Electron.*, (Pisa, Italy), vol. 2, pp. 343–348, Oct. 1998.

52. R. Johannesson and K. S. Zigangirov, *Fundamentals of Convolutional Coding.* New York: IEEE Press. Sponsored by IEEE Communications Society, IEEE Information Theory Society, IEEE Vehicular Technology Society.

53. Y. Ould-Cheikh-Mouhamedou, P. Kabal, and P. Guinand, "Enhanced max-log-app and enhanced log-app decoding for dvb-rcs," in *Proc. 3rd Int. Symp, Turbo Codes*, (Brest, France), pp. 259–262, Sept. 2003.

54. C. Douillard and C. Berrou, "Turbo codes with rate-$m/(m+1)$ consituent convolutional codes," *IEEE Trans. Commun.*, vol. 53, pp. 1630–1638, Oct. 2005.

55. J. B. Anderson and S. M. Hladik, "Tailbiting map decoders," *IEEE J. Sel., Areas Commun.*, vol. 16, pp. 297–302, Feb. 1998.

56. S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Algorithm for continuous decoding of turbo codes," *Electronics Lett.*, vol. 32, pp. 314–315, Feb. 1996.

57. M. Bickerstaff, L. Davis, C. Thomas, D. Garret, and C. Nicol, "A 24 mb/s radix-4 logMAP turbo decoder for 3 GPP-HSDPA mobile wireless," *IEEE ISSCC Dig. Tech. Papers*, vol. 1, pp. 150–151, 2003.

58. Z. Wang, "High-speed recursion architectures for map-based turbo decoders," *IEEE Trans. Very Large Scale Intergr. (VLSI) Syst.*, vol. 15, pp. 902–912, Apr. 2007.

59. J. C. Bajard, J. Duprat, S. Kla, and J. M. Muller, "Some operators for on-line radix-2 computations," *Journal of Parallel and Distributed Computing*, vol. 22, pp. 336–345, Aug. 1994.

Appendix [9]

Multi-Dimensional Signal Set Partitioning

For convenience of discussion, the partitioning of 8PSK signal discribed in Section 2.2. is duplicated in Fig. A.1, where each subset is equally divided into two smaller subsets in the next level to maximize the minimum squared subset distance (MSSD) in each smaller subset.

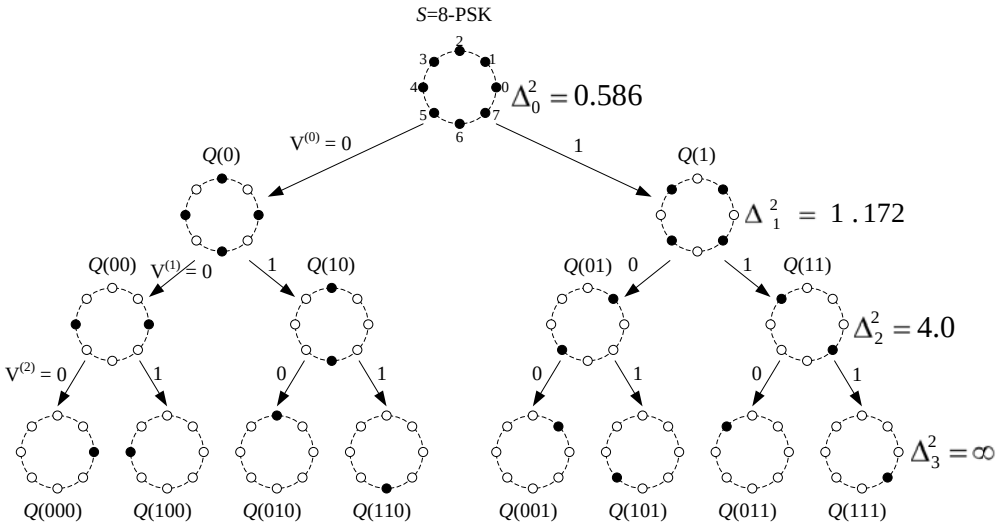


FIGURE A.1: Signal set partitioning of 8PSK.

Partitioning of 2×8PSK signal set (2 dimensional modulation)

Similarly to the 8PSK modulation, partitioning can be applied in the same way to the 2×8PSK signals. Integer $y1$ indicates the 8PSK points of one dimension and $y2$ indicates the points of another dimension, as shown in Fig. A.2. Then a 2×8PSK signal point can be represented by a 2×3 binary matrix:

$$\mathbf{y} = \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{bmatrix} = \begin{bmatrix} y_1^2 & y_1^1 & y_1^0 \\ y_2^2 & y_2^1 & y_2^0 \end{bmatrix}$$

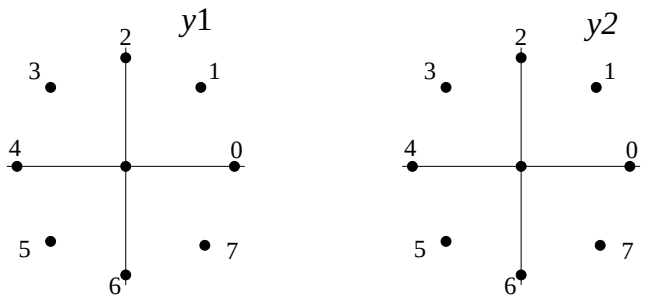


FIGURE A.2: 2×8PSK signal set.

Since 6 bits select a signal point, the unpartitioned signal set (indicated by Ω^0 and also called partitioning level $p = 0$) has a total of $2^6 = 64$ points. It can be seen that the MSSD at partition level $p = 0$ is $\Delta_0^2 = 2\sin^2(\pi/8) = 0.586$. At partition level $p = 1$, Ω^0 is divided into two subsets of 32 points each: $\Omega^1(0)$ and its coset $\Omega^1(1)$. In forming these two subsets, we would like to get their MMSD Δ_1^2 larger than Δ_0^2. If this were not possible, then we should find a partitioning that leads to a maximum reduction in the number of nearest neighbors within the smaller subset.

Before continuing the partitioning method, we introduce some notations. The column vector $\mathbf{y}$, *i.e.*, $\mathbf{y}^i = [y_1^i, y_2^i]$, for $0 \leq i \leq 2$, is required to be a block code. We define C_{mi} as the block code containing column vectors $\mathbf{y}^i$. Thus, C_{m0} contains the least significant bits of $y1$ and $y2$, C_{m1} contains the middle bits of $y1$ and $y2$, and C_{m2} contains the most significant bits of $y1$ and $y2$ (if $L = 2$ with L being the dimension of the modulation). The value of C_{mi} indicates which block code is being used. As an example, for $L = 2$, only three block codes are used: C_0 associated to codewords $[00]^T$, $[01]^T$, $[11]^T$ and $[10]^T$ with Hamming distance $d_0 = 1$; C_1 associated to code-words $[00]^T$ and $[11]^T$ with Hamming distance $d_1 = 2$; C_2 associated to code-words $[00]^T$ with Hamming distance $d_2 = \infty$.

To indicate which column vector will be modified at each step of the partitioning, the notation $\Omega\ (C_{m0}, C_{m1}, C_{m2})$ will be used in the following part. Thus, $\Omega^0 = \Omega(C_0, C_0, C_0)$

since C_0 has all possible vectors.

To obtain the next partition (at level $p = 1$), $\Omega^1 = \Omega(C_0, C_0, C_1)$ is constructed accrodingly to the previous rule. Since C_0 has only two code words, Ω^1 has 32 points and all points of Ω^1 belong to $\Omega^0(\Omega^1 \subset \Omega^0)$. Generally, this property is satisfied at each partitioning level: for two partitioning levels p and p', with $p' = p + 1, \ldots C_{m'_i} \subseteq C_{m_i}$ and $\Omega^{p+1} \subset \Omega^p$.

Fig. A.1 shows that for partition level $p = 1$, the MSSD is $\Delta_1^2 = 2\Delta_0^2 = 1.172$. In fact, we can use a more general expression from [8] that gives a lower bound on the MSSD at level partitioning p:

$$\Delta_p^2 \geq \min(\delta_{I-1}^2 d_{m_{I-1}}, \ldots, \delta_1^2 d_{m_1}, \delta_0^2 d_{m_0})$$

where d_{m_i} is the Hamming distance of the code C_{m_i}, δ_i^2 is the MSSD of $1\times$ MPSK ($L = 1$) modulation and $I = \log_2 M$ is the number of required bits to represent an MPSK point. Applying the above inequality to the $2\times$ 8PSK modulation, we obtain:

$$\Delta_p^2 \geq \min(4d_{m_2}, 2d_{m_1}, 0.586 d_{m_0}).$$

This inequality shows why Ω^1 is $\Omega(C_0, C_0, C_1)$ and is neither $\Omega(C_0, C_1, C_0)$ nor $\Omega(C_1, C_0, C_0)$: in both cases $\Delta_1^2 = 0.586$ and since the MSSD must be maximized, that leads to our choice at level partitioning $p = 1$. For all following partitions, we want to make Δ_p^2 as large as possible. Therefore, we use the following rule: the C_{m_i} that we partition into C_{m_i+1} from level p to level $p + 1$, should be the i corresponding to the smallest $\delta_i^2 d_{m_i}$ at partition level p. If several $\delta_i^2 d_{m_i}$ have the smallest value, we choose the one with the smallest i. This rule has been applied to find the partitioning of $2\times$ 8PSK modulation listed in Table AII.

According to Table AII, the partitioning generator equation to partition the signal set at level 6 is:

TABLE AI: 2×8PSK signal set partitioning

partition level(p)	Ω^p	Minimum Squared Subset Distance (Δ_p^2)	Generating vector $(t^p)^T$
0	$\Omega(C_0, C_0, C_0)$	$\min(4, 2, 0.586) = 0.586$	$[01]^T$
1	$\Omega(C_0, C_0, C_1)$	$\min(4, 2, 1.172) = 1.172$	$[11]^T$
2	$\Omega(C_0, C_0, C_2)$	$\min(4, 2, \infty) = 2.0$	$[02]^T$
3	$\Omega(C_0, C_1, C_2)$	$\min(4, 4, \infty) = 4.0$	$[22]^T$
4	$\Omega(C_0, C_2, C_2)$	$\min(4, \infty, \infty) = 4.0$	$[04]^T$
5	$\Omega(C_1, C_2, C_2)$	$\min(8, \infty, \infty) = 8.0$	$[44]^T$
6	$\Omega(C_2, C_2, C_2)$	$\min(\infty, \infty, \infty) = \infty$	–

$$
\begin{aligned}
\mathbf{y} &= \Omega^6(z) \\
&= z^5 \begin{pmatrix} 4 \\ 4 \end{pmatrix} + z^4 \begin{pmatrix} 0 \\ 4 \end{pmatrix} + z^3 \begin{pmatrix} 2 \\ 2 \end{pmatrix} + z^2 \begin{pmatrix} 0 \\ 2 \end{pmatrix} + z^1 \begin{pmatrix} 1 \\ 1 \end{pmatrix} + z^0 \begin{pmatrix} 0 \\ 1 \end{pmatrix} (mod 8) \\
&= \left\{ (z^1 + 2z^3 + 4z^5) \begin{pmatrix} 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ z^0 + 2z^2 + 4z^4 \end{pmatrix} \right\} (mod 8).
\end{aligned}
$$

Partitioning of 3×8PSK signal set (3 dimensional modulation)

As we indicated before, some signal sets may have more than one ways of partitioning, all of which satisfy the rules discussed above for 2×8PSK signal set. For example, there are three different ways of partitioning 3×8PSK signal set for a TCM system. All of them have 9 levels of partitioning, and will lead to the infinite MSSD at the 9th level. However, the increasing of MSSD from the lower level to the higher one is different from case to case. Interest readers can refer to [8] for details.

Partitioning of 4×8PSK signal set (4 dimensional modulation)

4×8PSK signal set is used for the system we discussed In Chapter 3. In a similar fashion to 2×8PSK described above, there is only one good way to partitioning the 4D signal set. The final result is summarized in Table AII.

TABLE AII: 4×8PSK signal set partitioning

partition level(p)	Ω^p	Minimum Squared Subset Distance (Δ_p^2)	Generating vector $(t^p)^T$
0	$\Omega(C_0, C_0, C_0)$	$\min(4, 2, 0.586) = 0.586$	$[0001]^T$
1	$\Omega(C_0, C_0, C_1)$	$\min(4, 2, 1.172) = 1.172$	$[0011]^T$
2	$\Omega(C_0, C_0, C_2)$	$\min(4, 2, 1.172) = 1.172$	$[0101]^T$
3	$\Omega(C_0, C_0, C_3)$	$\min(4, 2, 2.343) = 2.0$	$[0002]^T$
4	$\Omega(C_0, C_1, C_3)$	$\min(4, 4, 2.343) = 2.343$	$[1111]^T$
5	$\Omega(C_0, C_1, C_4)$	$\min(4, 4, \infty) = 4.0$	$[0022]^T$
6	$\Omega(C_0, C_2, C_4)$	$\min(4, 4, \infty) = 4.0$	$[0202]^T$
7	$\Omega(C_0, C_3, C_4)$	$\min(4, 8, \infty) = 4.0$	$[0004]^T$
8	$\Omega(C_1, C_3, C_4)$	$\min(8, 8, \infty) = 8.0$	$[2222]^T$
9	$\Omega(C_1, C_4, C_4)$	$\min(8, \infty, \infty) = 8.0$	$[0044]^T$
10	$\Omega(C_2, C_4, C_4)$	$\min(8, \infty, \infty) = 8.0$	$[0404]^T$
11	$\Omega(C_3, C_4, C_4)$	$\min(16, \infty, \infty) = 16.0$	$[4444]^T$
12	$\Omega(C_4, C_4, C_4)$	$\min(\infty, \infty, \infty) = \infty$	–

Again, according to Table AII, we could develop the generator equations through Eq. (3.1) to Eq. (3.4), where Eq. (3.1) is based on the 9^{th} level partitioning, Eq. (3.2) is based on the 10^{th} level, Eq. (3.3) is based on the 11^{th} level, and Eq. (3.4) is based on the 12^{th} level.