# Turbo Decoder VLSI Implementations for Multi Standards Wireless Communication Systems

*Jong Hun Han*


A thesis submitted for the degree of Doctor of Philosophy
**The University of Edinburgh**
December 2006

# Abstract

Turbo codes can provide a large coding gain through an iteration process as compared with a gain achieved by a traditional channel coding method. This has led to the adoption of turbo codes as standard in a variety of wireless communication systems, despite their higher computational complexity than a traditional decoder like a Viterbi decoder.

This thesis presents a number of high performance turbo decoder architectures for VLSI implementation in terms of area, power, and critical path delay. A Max-Log-MAP (MLMAP) algorithm is used to implement the turbo decoder with sliding window (SW) method to reduce the latency. Low power and area efficient turbo decoder implementation is achieved by reducing the memory blocks required by the SW method and to store the branch metrics used for computing log-likelihood-ratio (LLR). Retiming and reordering methods are applied to the computational units for computing the LLR and the state metrics.

A novel method is proposed to achieve high speed turbo decoder implementation for high throughput without significant area and power overheads. The proposed method addresses the inherent critical path delay problem in the state metric computation process by normalizing the branch metrics. While increasing the maximum speed of the turbo decoder, it also saves area and power of the state metric computation units.

A two-step soft-output Viterbi algorithm (TSOVA) based turbo decoder is implemented exploiting a novel concept for implementing a traceback algorithm (TBA) to achieve low area and low power turbo decoder implementations as compared to the MLMAP turbo decoder without any significant BER performance degradation.

Two reconfigurable application specific turbo decoders are implemented to support variable constraint length and binary and double-binary turbo codes for targeting various wireless communication systems. The reconfigurable turbo decoder architectures are realized by a proposed mapping method applied to the process for computing the state metrics and the LLR values. It is found that radix-4 based turbo decoder architecture can be exploited to implement the reconfigurable turbo decoder for binary and double-binary turbo codes.

# Declaration of originality

I hereby declare that the research recoded in this thesis and the thesis itself was composed and originated entirely by myself in the School of Engineering and Electronics at the University of Edinburgh.

# Acknowledgements

I would like to thank Dr. Ahmet Erdogan and Dr. Tughrul Arslan for their support and supervising. I also want to thank my lab colleagues for their help throughout my research work.

Thanks to my parents and my brother for their support.

Thanks to my wife, SY, and SH.

# Contents

# List of Figures

# List of Tables

# Acronyms and abbreviations

| | |
|---|---|
| ACS | Add compare select |
| ACSN | Add compare select normalization |
| APP | *A posteriori* probability |
| ASIC | Application specific integrated circuit |
| BER | Bit error rate |
| BM | Brach metric |
| BMCU | Backward metric computation unit |
| BMNU | Branch metric normalization unit |
| BMU | Branch metric unit |
| BPSK | Binary phase shift keying |
| BSM | Backward state metric |
| BTC | Binary turbo codes |
| BTE | Binary turbo encoder |
| CBG | Competing bit generator |
| CONST | Constant |
| CRSC | Circular recursive systematic convolutional |
| CS | Compare select |
| DAG | De-interleaver address generator |
| DMCU | Dummy-backward metric computation unit |
| DMD | Demodulation |
| DNT | Deinterleaver |
| DSM | Dummy-backward state metric |
| DTC | Double-binary turbo codes |
| DTE | Double-binary turbo encoder |
| FSM | Forward state metric |
| FMCU | Forward metric computation unit |
| IAG | Interleaver address generator |
| INT | Interleaver |
| LCU | LLR computation unit |

| | |
|---|---|
| LMAP | Log-MAP |
| LLR | Log likelihood ratio |
| MAP | Maximum *a posterior* |
| MAU | Mapping unit |
| MCU | Metric computation unit |
| MEM | Memory |
| ML | Maximum likelihood |
| MLMAP | Max-Log-MAP |
| MOD | Modulation |
| MUX | Multiplexer |
| PCU | Path comparison unit |
| QoS | Quality of service |
| R2 | Radix-2 |
| R4 | Radix-4 |
| REA | Register exchange algorithm |
| REASIC | Reconfigurable application specific integrated circuit |
| REASTD | Reconfigurable application specific turbo decoder |
| RSC | Recursive systematic convolutional |
| RTL | Register transfer level |
| SM | State metric |
| SMU | Survivor memory unit |
| SMCU | State metric computation unit |
| SOVA | Soft-output Viterbi algorithm |
| SW | Sliding window |
| TBA | Traceback algorithm |
| TBU | Traceback unit |
| TC | Turbo codes |
| TD | Turbo decoder |
| TE | Turbo encoder |
| TMU | Transition metric unit |
| TSOVAREA | Two-step SOVA based on REA |
| TSOVATBA | Two-step SOVA based on TBA |
| UPU | Update process unit |

# Nomenclature

| | |
|---|---|
| A, B | Systematic bits of double-binary turbo codes |
| C, D | Parity bits of double-binary turbo codes |
| X | A systematic bit of binary turbo codes |
| Y | A parity bit of binary turbo codes |
| $L_e$ | Extrinsic information |
| $L_{lr}$ | Log likelihood ratio |
| $k$ | Time |
| $y_s$ | Symbol data of systematic bits |
| $y_p$ | Symbol data of parity bits |
| $\gamma$ | Branch metric |
| $\alpha$ | Forward state metric |
| $\beta$ | Backward state metric |
| $\mathbf{y}$ | A vector of received data |
| $\mathbf{x}$ | A vector of transmitted data |
| $\mathbf{n}$ | Noise |

# Chapter 1

# Introduction

## 1.1 Motivation

Since wireless communication systems evolved from an analogue mode to a digital, there have been intensive studies of digital signal processing, leading to development of a number of digital signal processing techniques to deliver quality of services (QoS). It is well known that digital signal processing has many advantages over analogue [1]. In the past, voice communication was the main purpose of the systems. Nowadays, with the developed techniques, wireless communication is regarded as a way of not only voice communication, but also delivering various services such as text messaging, web browsing, gaming, entertaining multi-media. These services are possible due to the development of high performance digital signal processing techniques.

A number of digital signal processing techniques are required to construct wireless communication systems. They are needed to transmit and to receive correct information without errors, such as those which happen due to the effect of noise and interference when a signal is transmitted over the channel. Among the techniques, the channel encoding and decoding processes are one of the processes most necessary to avoid the error in current wireless communication. Several channel coding techniques are available for the purpose. In general, a decoder in receiver systems performs an exhaustive search process to obtain error free information. For that reason, it is regarded as one of the highest computational units in base station systems [2].

Turbo codes introduced by Berrou *et al.* [3] can provide a large coding gain through iterative process. Their outstanding bit error rate (BER) performance has been paid a lot of attention by many researchers, and their contributions have led turbo codes to be adopted as standard for channel coding of various wireless communication systems such as 3GPP, IEEE 802.16, DVB-RCS, etc. Since then, turbo codes have been researched and their performance in such

systems evaluated [4-10]. The use of turbo codes for achieving better QoS is expected to continuously increase. However, some obstacles are needed to be overcome to achieve a practical turbo decoder implementation, due to one's intensive computation process and high latency occurring by an iteration process, which cause of increasing hardware costs and reducing data rates, respectively. In addition, the use of different types of turbo encoding schemes in the systems can decrease compatibility and reusability of turbo decoders in the receivers. This can be addressed by implementing a turbo decoder that can be configured different turbo codes. Thus, these issues were the motivation of this thesis, aiming at the development of high performance in terms of area, power, and throughput and dynamically reconfigurable turbo decoder VLSI implementations.

## 1.2 An Overview of State-of-Art Turbo Decoder Implementations

Since appearing turbo codes, many researchers have studied turbo codes to understand their decoding principles and mechanisms in order to achieve better performance in terms of BER. Also, implementing practical turbo decoders for wireless communication systems has been paid a lot of attention due to their high complexity and latency as mentioned in the previous section. In recent, turbo decoders targeting practical applications have been realized on a silicon chip [11-14].

Two algorithms, Maximum a posterior (MAP) and soft-output Viterbi algorithm (SOVA), are available to implement a turbo decoder. The MAP algorithm proposed with turbo codes [3] has been simplified by the approximation methods called Log and Max-Log MAP algorithms [15]. Also, the sliding window method for the MAP based turbo decoders suggested by [16] reduces turbo decoding latency. In addition, two-step SOVA presented in [17] improves the implementation complexity of the original SOVA. Since these techniques, many works implementing practical turbo decoders have proposed suggesting several techniques. Among the techniques, the look-ahead computation [18-20], pipelining [11], and parallel processing techniques [21-28] are proposed for high data throughput turbo decoders based on LMAP and MLMAP with SW methods. Also, in [29-32], low power

implementations for turbo decoders have recently been investigated for wireless applications. On the other hand, a turbo decoder based on SOVA can provide better throughput than MAP based turbo decoder, due to low latency. In addition, less complexity of SOVA than the MAP algorithm has been attracted many researchers to implement an efficient SOVA based turbo decoder [33-42]. In the literature, low power [35-38] and high rate [33-34, 39-41] implementations for SOVA have been proposed.

Furthermore, recent wireless communication systems have adopted several channel coding schemes as a standard. Thus, designing highly compatible and flexible decoders for the systems is becoming a crucial issue more and more to save the costs of the systems. This issue can be addressed by designing a turbo decoder to be configured for the systems. The reconfigurable turbo decoders [43-53] recently have been proposed for supporting various wireless communication systems.

## 1.3 Contributions

This thesis contributes to development of high performance and reconfigurable turbo decoder architecture targeting mobile and portable applications requiring low power consumption and area usage. The architecture is realized by using retiming and reordering methods that are used to implement efficient modules for computing the state metrics and the log-likelihood-ratio. A turbo decoder for high data rate is implemented using a novel technique that addresses the inherent critical path delay problem in order to achieve high speed turbo decoder. The turbo decoders are implemented based on Max-Log-MAP algorithm. For achieving better area and power saving, in this thesis, a novel structure for implementing TSOVA based turbo decoder architecture is introduced and compared with the Max-Log-MAP based turbo decoders. Also, based on the improved turbo decoder architectures, reconfigurable application specific turbo decoders to support various wireless communication systems, saving the costs needed for ASIC implementations, are proposed.

The thesis starts by investigating turbo decoder at high level to verify the decoding functionality through the BER performance evaluation. After completing the verification, a conventional turbo decoder based on Max-Log-MAP algorithm has been designed at RTL

level for the hardware implementation. Parameters for designing the hardware have been chosen based on the results of the high level simulations.

In the conventional turbo decoder architecture, it has been found that memory blocks can be reduced by modifying the input data transaction method. This resulted in saving up to 30 % in total area and 18 % in total power. Also, retiming and reordering methods for the state metric and LLR value computation process achieved saving 21 % in area and 29 % in power. The high-speed turbo decoder has been realized by using a novel method that normalizes the branch metrics to reduce the critical path delay in the turbo decoder architecture. Simulation results show that the method can reduce the critical path delay up to 42 % as compared to the critical path delay of the conventional turbo decoder architecture.

After presenting the efficient implementation techniques applied to Max-Log MAP based turbo decoder implementations, an alternative algorithm, called two-step soft-output Viterbi algorithm (TSOVA), to implement a turbo decoder has been presented. The TSOVA turbo decoder architecture has been designed with a novel concept for implementing the traceback algorithm (TBA). Simulation results of the architecture have shown that the TSOVA turbo decoder architecture can save 72% in area and 52% in power compared to the MLMAP turbo decoder architecture.

To support multi-standard wireless communication systems, two reconfigurable turbo decoder architectures have been presented in this thesis. One of these architectures has been designed for supporting different constraint lengths from 3 to 5. For this reconfigurable implementation, a mapping method is presented for designing the units for computing the state metrics and LLR values. The reconfigurable architecture is implemented with a clock gating method to minimize the power overhead as the architecture is configured for different constraint lengths.

The other reconfigurable turbo decoder has been designed for different sorts of turbo codes called binary and duo-binary turbo codes. It is found that radix-4 binary and double binary turbo decoders can share many hardware resources. The design of the reconfigurable architecture is based on a radix-4 turbo decoder, and can be configured for binary and double binary turbo codes. Its hardware performance has been compared with a convention turbo

decoder implementation. The comparison of the performance results shows that the reconfigurable architecture can save more energy than the conventional architecture.

## 1.4 Thesis Contents

The thesis consists of nine chapters including this chapter. It is organized as follows.

Chapter 2 reviews turbo codes and their encoding and decoding techniques. Two kinds of turbo codes, binary and duo-binary turbo codes, are described with their encoder structures for different constraint length. Maximum *a posterior* (MAP) and SOVA are introduced. In the literature, several techniques are proposed to achieve an efficient turbo decoder implementation. This chapter reviews and summarizes the techniques, describing how they are used to reduce implementation complexity and hardware costs.

Chapter 3 presents MLMAP based turbo decoder architecture. The architecture is implemented with sliding window (SW) method in order to reduce latency and to save hardware costs. The architecture consists of three metric computation units, a log-likelihood-ratio computation unit, a data scheduling unit, and memory blocks. Implementation methods and functions of those units are described in detail. Through high level simulations, before estimating hardware performance, BER performance of the turbo decoder based on MAP algorithm is evaluated with a variety of parameters to verify the turbo decoding algorithm. MAP and MLMAP based turbo decoders are investigated to find the optimal parameters for efficient hardware implementation of turbo decoders. Following a design flow, the MLMAP turbo decoder is implemented on hardware, and the hardware performance in terms of area usage, power consumption, and critical path delay is evaluated. The results obtained in this chapter are regarded as conventional turbo decoder results. Thus, these results are used as reference and compared with the results obtained with proposed turbo decoders presented in later chapters.

Chapter 4 introduces a novel technique to implement high performance MLMAP based turbo decoders. Two main techniques are proposed in this chapter. The first is suggested in order to achieve high speed turbo decoder implementations for high throughput by reducing the

critical path delay. This is realized by addressing the inherent critical path delay problem using a novel branch metric normalization method. The second is for saving area and power of the turbo decoder by reducing memory blocks for controlling SW method and storing the branch metrics. Moreover, an efficient unit used for the state metric and the LLR computational processes is proposed to save the hardware costs. The turbo decoder hardware performance is investigated and compared with the conventional turbo decoder results.

Chapters 5 and 6 present an alternative turbo decoding algorithm called soft-output Viterbi algorithm (SOVA) and its hardware architecture. Specifically, two-step SOVA (TSOVA) is implemented to evaluate BER and hardware performance. In this chapter, a novel method for traceback algorithm (TBA) based TSOVA turbo decoder hardware implementation is proposed to save area and power. The TBA based TSOVA turbo decoder architecture is described and compared with the register exchange algorithm (REA) based TSOVA described in the literature. The results of the TBA based SOVA turbo decoder are compared with the MLMAP based turbo decoder results in terms of area and power.

Chapter 7 presents a reconfigurable turbo decoder architecture that is implemented based on MLMAP algorithm with SW method. The architecture is designed to be used with various wireless communication systems. In this chapter, two different reconfiguration turbo decoders are proposed. One reconfigurable turbo decoder is designed to support different constraint lengths from K=3 to 5. For the reconfigurable implementation, a mapping method for computing state metrics and LLR values is proposed, and a clock gating method is applied to saving power consumption as the architecture is configured for K=3 and 4. Another reconfigurable turbo decoder that can be configured for binary and double-binary turbo codes with fixed constraint length K=4. For this architecture, a radix-4 turbo decoder for binary turbo codes, showing similarities between turbo decoders for radix-4 binary turbo codes and double-binary turbo codes, is investigated and implemented. Thus, the radix-4 method is exploited to implement the reconfigurable turbo decoder. The turbo decoders for radix-4 binary turbo codes and duo-binary turbo codes are implemented in ASIC to compare the hardware performance with the reconfigurable turbo decoders. The results of the two reconfigurable turbo decoders are also compared with conventional ASIC turbo decoder implementations.

Finally, the thesis is concluded in chapter 8, describing what should be researched in the future.

# Chapter 2

# Turbo Codes Encoding-Decoding Algorithms and Implementation Techniques

## 2.1 Introduction

Turbo codes introduced in [3] provide a large coding gain close to the Shannon's limit [54]. Thus a turbo decoder can achieve a very low bit error rate (BER), by performing an iterative process as shown in [3]. Since then, the principles of turbo encoding and decoding have been studied intensively in order to improve BER performance. The authors in [56-60] describe the turbo decoding algorithm in detail. A number of contributions to the development of the turbo decoder for practical applications has led to the adoption of turbo codes as a standard in various wireless communication systems, such as wideband code division multiple access (WCDMA) [61], 3GPP [62], Consultative Committee for Space Data Systems (CCSDS) [63], IEEE 802.16 [64] (also known as WiMax), and digital video broadcasting returned channel over satellite (DVB-RCS) [65]. Specifications of turbo codes required by the systems are given in Appendix B and C.

This chapter introduces turbo encoding and decoding schemes for different constraint lengths. First, a turbo encoder used to generate binary and non-binary turbo codes is presented, and an encoder structure described. Then, a decoding algorithm for turbo codes is presented, and a turbo decoder structure described. Two sorts of turbo codes are widely used in wireless communication systems. Two algorithms, maximum a posterior (MAP) [3] also known as BCJR algorithm [66], and soft output Viterbi algorithm (SOVA) [67], are available for the turbo decoder implementations. It is well known that while MAP based turbo decoders provide better BER performance, a turbo decoder based on SOVA can be implemented with less computational complexity. However, the original turbo decoding algorithms based on MAP and SOVA are not suitable for establishing an efficient turbo decoder implementation.

Techniques needed to implement an efficient turbo decoder for practical systems are reviewed, and the techniques used to improve performance of turbo decoders are described.

Detailed discussion of fundamental turbo encoding and decoding theory is beyond the scope of this thesis. The explanations and discussion set out in this chapter, and in later sections, are based on practical implementation, with the aim of understanding the encoding and decoding principles. Detailed principles of turbo codes can be consulted elsewhere [68-40].

This chapter is organized as follows. Section 2.2 and 2.3 describe binary and non-binary turbo codes encoding techniques showing the encoder structures. Two kinds of turbo codes, known as binary and non-binary turbo codes, are explained, including how they are generated. The interleaving method used in the performance evaluation is described in section 2.4. Section 2.5 shows a general turbo decoder structure. The MAP based turbo decoding algorithm is presented in Section 2.6. In section 2.7, SOVA is introduced, and how it is used for a turbo decoding algorithm described. Techniques for an efficient turbo decoder implementation are reviewed in section 2.8. Section 2.9 summarizes the chapter.

# 2.2 Encoding Binary Turbo Codes

## 2.2.1 Turbo Encoder Structure

The turbo encoder structure is based on the convolutional encoder structure. Figures 2.1 (a) and (b) illustrate conventional encoder structures for convolutional and turbo codes with constraint length K=4 and polynomial generators $13_8$ and $15_8$. The encoding process starts after initializing $D_0$, $D_1$, and $D_2$, into zero.

The convolutional encoder of Figure 2.1 (a) generates a code word of two bits, $X_1$ and $X_2$, in a half code rate with constraint length K=4. A convolutional encoder can be implemented with different constraint length and code rate depending on the application.

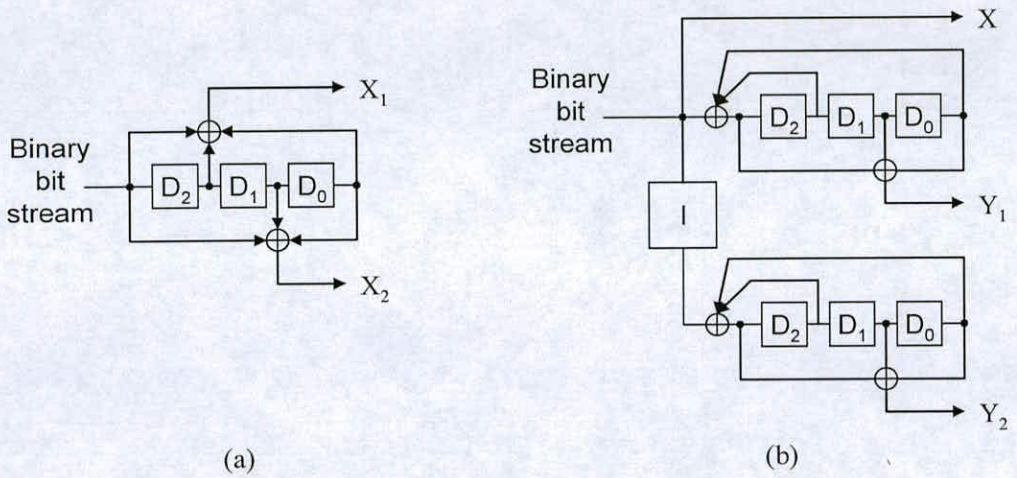



Figure 2.1 (a) Convolutional and (b) turbo encoder structures for K=4.

The turbo encoder structure illustrated in Figure 2.1 (b) consists of two recursive systematic convolutional (RSC) encoders and one interleaver (I) with a basic code rate of 1/3. X of the code word is the same with input binary bits, and is called a systematic bit. $Y_1$ and $Y_2$ are parity bits produced by 1st and 2nd RSC encoders, respectively. Code rate of the turbo encoder can be adjusted by puncturing the code word. Turbo codes generated by the encoder are called parallel concatenated convolutional codes (PCCC). Serial concatenated convolutional codes (SCCC) [71] also exist. In this work, turbo codes for BER performance evaluation are generated by the structure of Figure 2.1 (b). In general, binary turbo codes terminate with tail bits. The tail bits make the final state into zero, which is crucial to obtain a proper BER performance in the turbo decoding process [72-74]. The tailing bits for the simulation are made following the encoding method described in 3GPP [62]. .

Figures 2.2 (a) and (b) illustrate turbo encoder structures for constraint length K=3 and 5, in which polynomial generators are $7_8 5_8$ and $23_8 35_8$. These encoders also generate block code including the tailing bits, which are generated by the same method used in the encoder of Figure 2.1 (b).

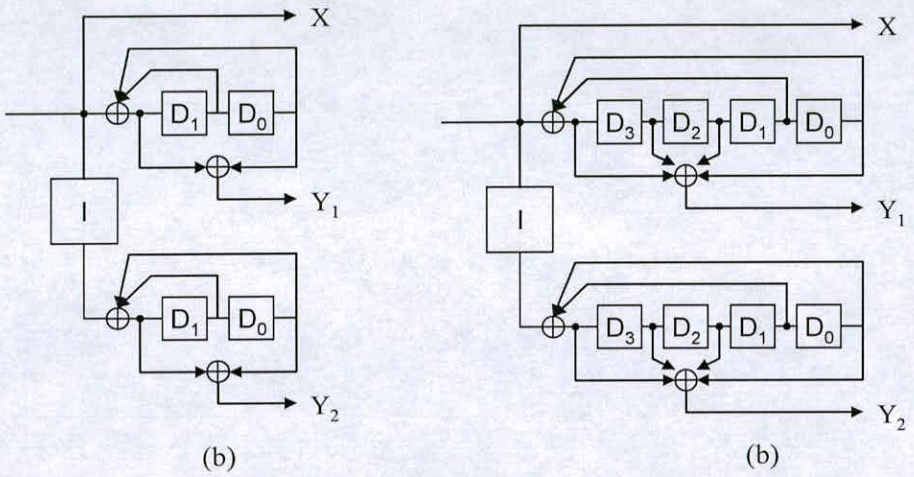



Figure 2.2 Turbo encoder structures for (a) K=3 and (b) K=5.

## 2.2.2 Trellis State Diagram

The encoder behaviour can be represented with the help of a trellis state transition diagram method. A trellis state diagram represents the encoding process in the state transition paths following the input sequences coming to the encoder. The state transition also gives information of the output code word.

Figure 2.3 illustrates the trellis state diagram of the turbo encoder shown in Figure 2.1 (b), as an example. The number of states corresponds to $2^{K-1}$ for constraint length K. The state transition paths represented in different line styles indicate the code word, X and $Y_1$ (or $Y_2$) generated by the RSC encoder, illustrated in Figure 2.1 (b). Each state is represented by the information accumulated in each of the Ds of the encoder, which are initialized to 0 at the beginning of every input block. Thus the initial state starts with 0, and then the next state depends on the input data. This procedure is performed until the end of the block. As already described, the turbo encoders drive the end state to converge to state 0 using the tail bits. Therefore, the turbo decoder at a receiver can perform the decoding process with the initial and final state information available.

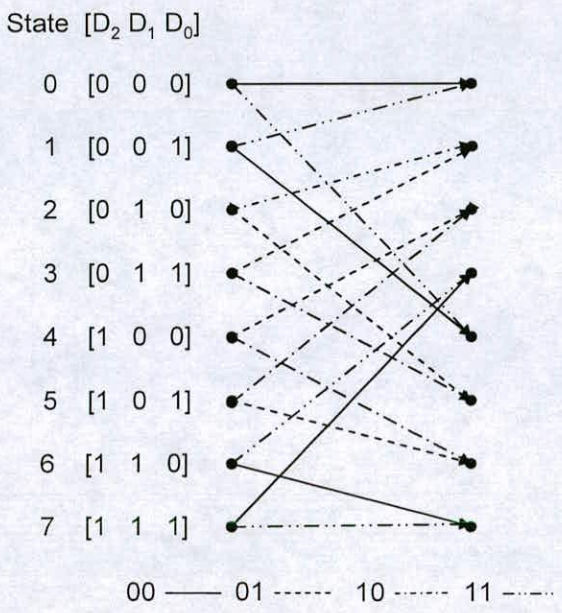



Figure 2.3 A trellis state diagram of a turbo encoder for K=4.

# 2.3 Encoding Double-Binary Turbo Codes

## 2.3.1 Encoder Structure for Double-Binary Turbo Codes

The turbo encoder in the previous section accepts binary bit data. Non-binary data can also be encoded by a turbo encoder that generates non-binary turbo codes [75-76]. It is known that non-binary turbo codes are suitable for a higher code rate than 1/2, while binary turbo codes are suitable for lower than 1/2 [77-78]. Thus non-binary turbo codes could be used for better throughput. This section describes the encoder structure for double-binary turbo codes (DTC), which are adopted as a standard encoding scheme in IEEE 802.16 [64] and DVB-RCS [65].

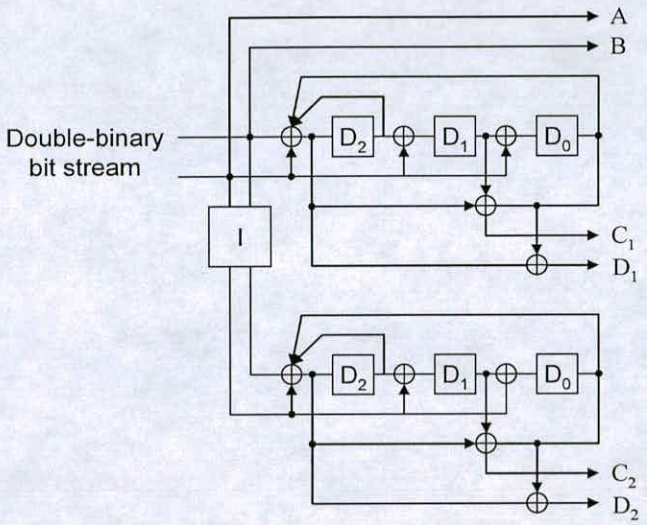



Figure 2.4 A double-binary encoder structure for K=4.

Figure 2.4 illustrates the double-binary turbo encoder (DTE) structure for constraint length K=4 with the polynomial generators, $13_8$ and $15_8$. As can be seen, two binary bits are used as source data for the encoder. The two input bits are then encoded as systematic bits, A and B, and two parity bits, $C_1$ and $D_1$. In the same way as the binary turbo encoder, the input bits are then interleaved to provide the other parity bits, $C_2$ and $D_2$ with the second RSC encoder. The code rate of the encoder is 1/3. However, it can be adjusted by puncturing the output code word. The double-binary encoding scheme used in this thesis refers to the encoder in [64] and [65].

### 2.3.2 CRSC Turbo Codes

The DTC are often called circular recursive systematic convolutional (CRSC) turbo codes [77-79], which are terminated without tail bits, unlike BTC. In general, BTC are generated with tail bits at the end of the block in order to make the final state same with the initial state. In contrast to BTC, the initial state of double-binary turbo codes can be any state, and the final state must be the same as the initial state. Thus, CRSC turbo codes seem to be circular

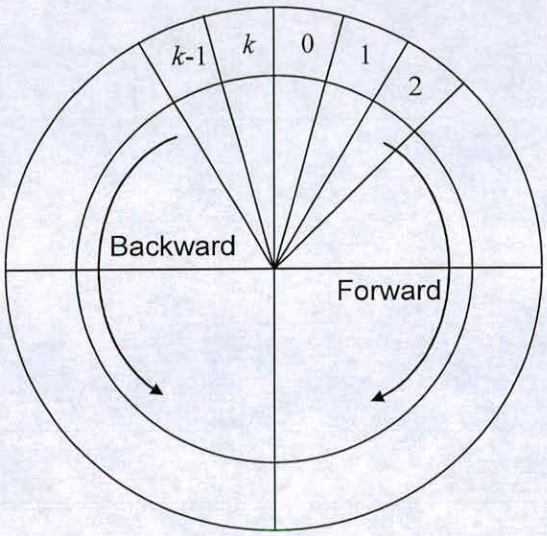



Figure 2.5 A diagram of circular recursive systematic codes.

as illustrated in Figure 2.5. It is assumed that a block of DTC is started at time 0, and ended at time $k$. Since the initial and final states are the same, the DTC can be circulated without terminating to a particular state. To generate the circular codes, an additional encoding process (called a training process) is needed. At first, DTE starts with initial state 0, and then the final state is found when completing the encoding process. DTE starts the encoding process again with a state that is determined following the final state found in the training process. For this determination, a look-up table is required, as described in [64] and [65]. After completing this second encoding process, the initial and final states of the DTC become the same. This process is summarized as follows :

1) Initialize the initial state to 0 ($D_2$=0, $D_1$=0, $D_0$=0).

2) Encode an input block with the double binary encoder.

3) Find the final state of the input block.

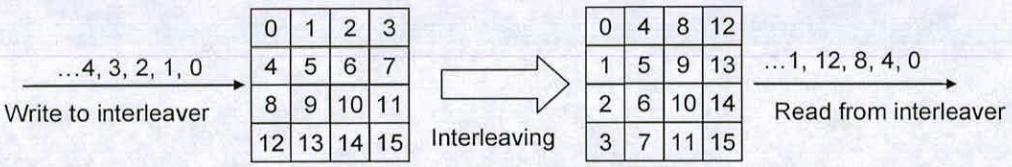
4) Initialize the initial state into a state determined by the final state found in 3)

5) Encode the same input block again with the double-binary encoder from the newly determined initial state.

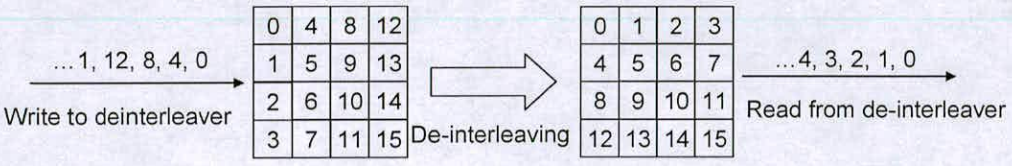
## 2.4 Interleaving Method

An interleaving method is used to reduce burst errors caused by transmitting data over the channel. It is already widely used to construct wireless communication systems for achieving better performance. Interleaving method and size strongly affect the turbo decoder performance [80-82]. A number of interleaving methods have been suggested in order to achieve better BER performance.

In this thesis, a block interleaving method [83] is used to evaluate the BER performance of the turbo decoders. Figure 2.6 illustrates the block interleaving method as an example. In this case, a square block is employed as the block interleaver. First, the block is filled with a series of data, after which the addresses of data in the block are switched. Then, the data are




(a)




(b)

Figure 2.6 Examples of block (a) interleaving and (b) de-interleaving.

read out from the interleaver in different order from the input order. De-interleaving is performed in the same way as the interleaving. The de-interleaved data are filled in with the original addresses after completing the de-interleaving process.

## 2.5 Turbo Decoder Structure

Figure 2.7 illustrates a turbo decoder structure which consists of two soft-input soft-output (SISO) decoders, two interleavers (I1 and I2), and two de-interleavers (D1 and D2), as introduced in the literature [3]. The turbo decoding process is iteratively performed by two SISO decoders via I and D. An increase of the number of iteration gives better performance in terms of BER. The performance improvement is saturated after completing several numbers of iteration depending on the channel conditions. The input symbols, $y_s$ and $y_{p1}$, and the extrinsic information, $L'_{e1}$, are used for the turbo decoding process in the 'SISO Decoder 1' that produces log-likelihood ratio, $L_{lr1}$, and a priori value, $L_{e2}$. Then the input symbols, $y_s$ (via interleaver) and $y_{p2}$, and the extrinsic information, $L'_{e2}$ (interleaved value of $L_{e2}$), are used in the 'SISO Decoder 2' that generates $L_{e1}$ for the 1st SISO decoder and soft-output value, $L_{lr2}$., which is converted into the output for decoding information after being de-interleaved by D2. These input symbols, $y_s$, $y_{p1}$, and $y_{p2}$, correspond to the code word, X, Y1,

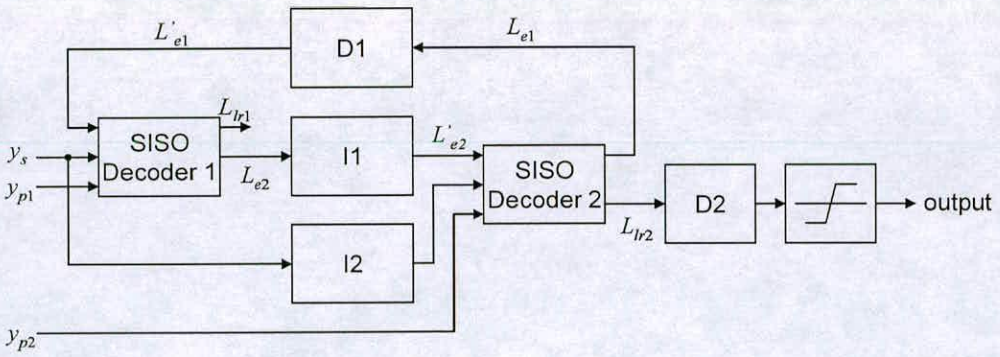



Figure 2.7 A block diagram of a turbo decoder structure.

and Y2, generated by the turbo encoder in Figure 2.1. The decoding processes are iteratively performed in order to achieve a better BER performance. One iteration is finished when the decoding process of the two SISO decoders is completed. The structure in Figure 2.7 is for BTC, but it can also be used for DTC with the increased number of input symbol data.

## 2.6 Maximum *a posterior* Algorithm for Turbo Decoding Process

The decoding principle of MAP based turbo decoders is different than traditional decoders like Viterbi decoders [84]. While Viterbi decoders, in general, find Maximum-Likelihood (ML) paths to determine the hard-decision during the decoding process, MAP based turbo decoders compute all state metrics of the forward and backward processes to generate the hard-decision. These processes start at the beginning and the ending of the block as illustrated in Figure 2.8. Then, soft-output, called LLR, is calculated from all the state and branch metrics.

The MAP algorithm suggested in [67] is modified for use in the turbo decoding process. A turbo decoder based on the MAP algorithm generates soft-output from the received symbol data. The soft-output contains information on the hard-decision and its reliability value. It is represented by log-likelihood ratio (LLR) of *a posteriori* probability (APP) as follows :

$$\begin{aligned} L_{lr}(k-1) &= \ln \frac{\sum_{s_k} \sum_{s_{k-1}} \gamma_1(s_{k-1}, s_k)\alpha_{k-1}(s_{k-1})\beta_k(s_k)}{\sum_{s_k} \sum_{s_{k-1}} \gamma_0(s_{k-1}, s_k)\alpha_{k-1}(s_{k-1})\beta_k(s_k)} \\ &= L_{lr1}(k-1) - L_{lr0}(k-1) \end{aligned} \tag{2.1}$$

where $L_{lr}$ is the LLR value, $\gamma$ is the branch metric, $\alpha$ is the forward state metric, and $\beta$ is the backward state metric at time $k$ and state $s$. These metrics can be represented in logarithm form as follows:

$$\bar{\gamma}(s_{k-1}, s_k) = \ln \gamma(s_{k-1}, s_k) = 1/2[L_c y_{s,k} u_{s,k} + L_c y_{p1,k} u_{p1,k} + L_{e1} u_{s,k}] \tag{2.2}$$

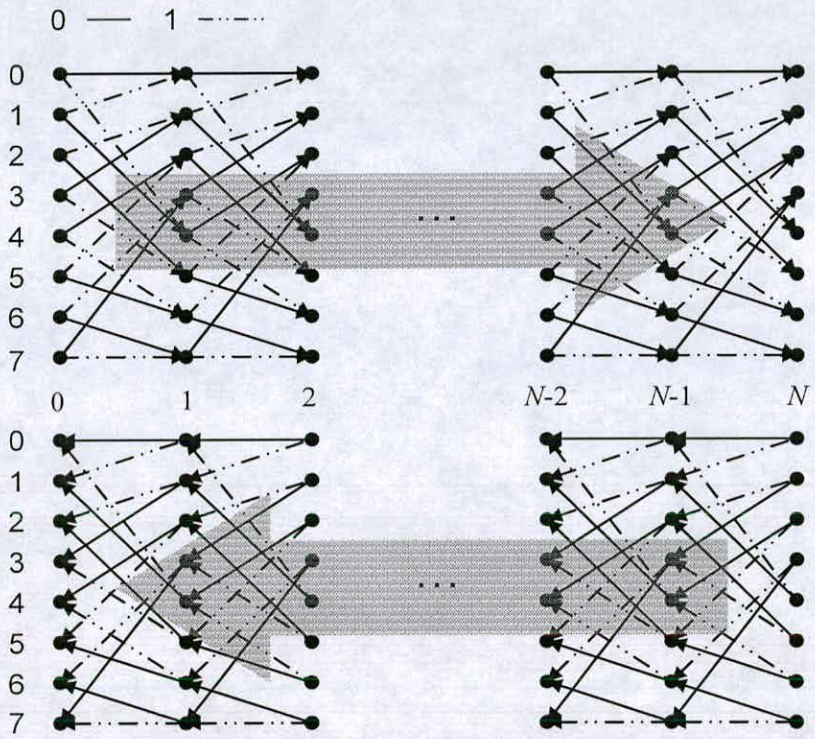



Figure 2.8 The forward and backward process flow of the MAP based turbo decoder.

$$\overline{\alpha}(s_k) = \ln \alpha(s_k) = \ln \sum_{s_{k-1}} \exp[\overline{\gamma}(s_{k-1}, s_k) + \overline{\alpha}(s_{k-1})] \tag{2.3}$$

$$\overline{\beta}(s_k) = \ln \beta(s_k) = \ln \sum_{s_{k+1}} \exp[\overline{\gamma}(s_{k+1}, s_k) + \overline{\beta}(s_{k+1})] \tag{2.4}$$

$$\alpha(s_0) = \begin{cases} \infty & for\ s = 0 \\ 0 & for\ s \neq 0 \end{cases} \qquad \overline{\beta}(s_N) = \begin{cases} \infty & for\ s = 0 \\ 0 & for\ s \neq 0 \end{cases} \tag{2.5}$$

where $y_{s,k}$ and $y_{p1,k}$ are the input symbol for systematic bit ($u_{s,k}$) and parity bit ($u_{p1,k}$), $L_c$ is the channel reliability value, which is fixed to 2 in this work, and $L_{el}$ is the extrinsic information. The forward and backward state metric computation process is illustrated in Figure 2.9 for

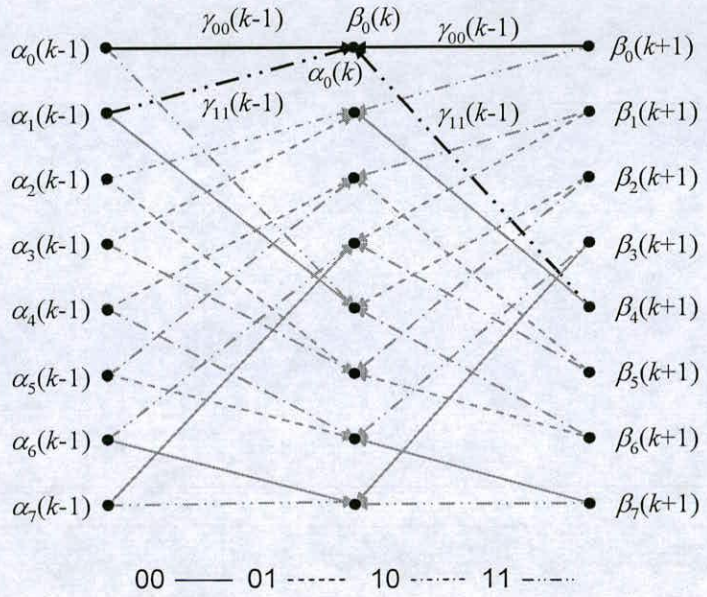



Figure 2.9 An example of the forward and backward state metric computation processes.

constraint length K=4. As can be seen, a forward state metric at time k is computed from two of the previous state metrics at time $k$-1 and two branch metrics. A backward state metric at time $k$ is computed from two backward state metrics at time $k$+1 and two branch metrics. These computation processes are performed for all the states at each time, initializing the forward and backward state metrics by equation (2.5) at the beginning and end of the block. Then, LLR is calculated from the state and branch metrics obtained by equations (2.2), (2.3), and (2.4). It can be represented as follows:

$$L_{lr}(k) = \ln \sum_{s_k} \sum_{s_{k-1}} \exp(\bar{\gamma}_1(s_{k-1}, s_k) + \bar{\alpha}_{k-1}(s_{k-1}) + \bar{\beta}_k(s_k)) - \ln \sum_{s_k} \sum_{s_{k-1}} \exp(\bar{\gamma}_0(s_{k-1}, s_k) + \bar{\alpha}_{k-1}(s_{k-1}) + \bar{\beta}_k(s_k)) \tag{2.6}$$

The hard-decision is determined by whether the result of equation (2.6) is positive or negative. The extrinsic information for the next decoding process is obtained from the LLR after subtracting input systematic symbol and extrinsic information as follows:

$$L_{e2}(k) = L_{lr}(k) - L_c y_s(k) - L_{e1}(k) \tag{2.7}$$

where $L_{e2}$ is used as an input for the next iteration process, as shown in the turbo decoder structure illustrated in Figure 2.7.

## 2.7 Sliding Window Method

The MAP based turbo decoder performs the forward and backward processes to generate LLR, as described in section 2.6. While the forward process starts at the beginning of the input block, the backward process starts at the end of the block. These processes cause high latency, because LLR can only be generated after completing the forward and backward processes. The latency depends on the input block size, which is significantly increased if the block size is large. This may result in a lowering of throughput.

The SW method introduced in [16] can help to solve the latency problem by dividing the input block into several sub-blocks (called a "window"). The turbo decoder then performs the decoding process, sub-block by sub-block, rather than for the whole block. The size of each sub-block can be uniformly determined, regardless of input block size. The SW method cannot only reduce the latency, but also reduce hardware costs by reducing the amount of memory required to store the forward or backward state metrics which cause delay, before computing LLR. Thus it is necessary to achieve an efficient MAP-based turbo decoder implementation. In the literature, a number of SW based turbo decoders have been implemented to reduce power and area [21, 25, 26, 28], and to achieve high throughput [90-92].

Figure 2.10 shows the backward and forward process flow in the SW method. In this case, the forward process is not affected by the SW method. The dummy-backward process starts at the end of the second sub-block. This process produces error-free backward state metrics

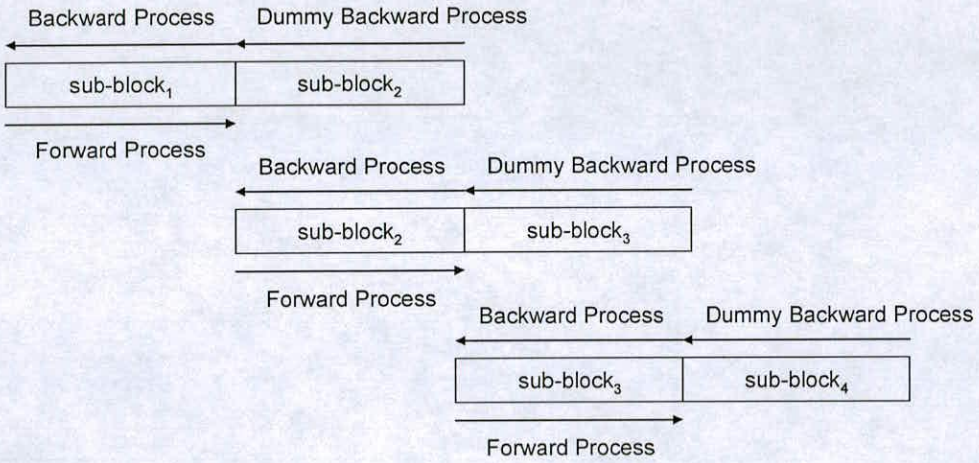



Figure 2.10 Sub-frame by sub-frame turbo decoding process in the SW method.

for the backward process of the first sub-block. After completing the dummy-backward process, the backward state metrics generated from the dummy-backward process are used as the initial state metrics of the backward process. During that time, the forward process is performed to generate the forward state metrics, and they are stored in memory until the backward process starts. Thus, LLR is calculated from the backward state metrics generated when the backward process starts, and the stored forward state metrics. In the process, the role of the dummy-backward process is similar to the traceback process in a Viterbi decoder [139]. While the traceback process finds an error-free state before starting the decoding process for generating a hard decision, the dummy-backward process generates the initial state metrics for the backward process, which provides the backward state metrics for LLR computation process.

The data flow for the turbo decoding process with the SW method is also illustrated as a graph with time and block axes, in Figure 2.11. The solid line without notations indicates input data, while the dashed line with $\beta_d$ represents the dummy backward calculation process; the number identifies the sub-block.

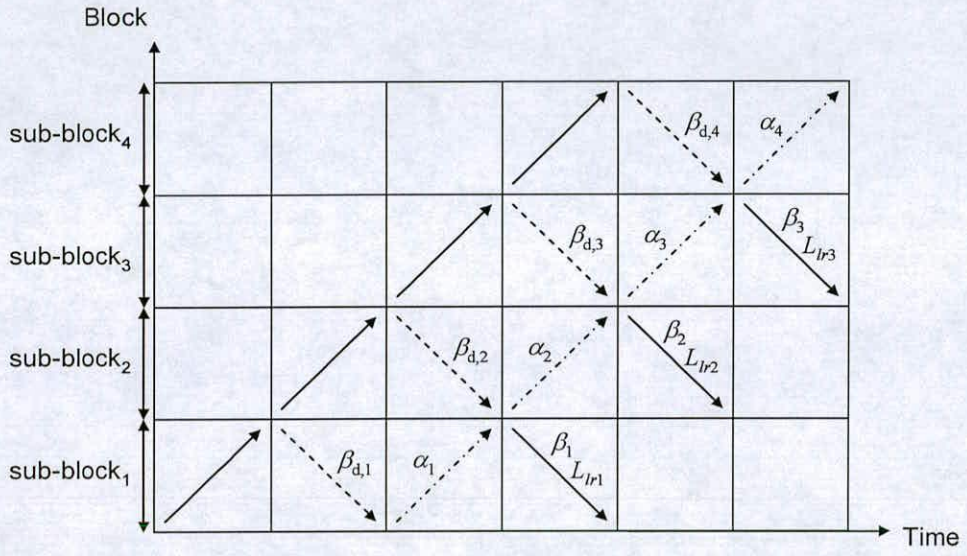



Figure 2.11 A graph of sub-block data with sliding window method.

As stated above, the dummy backward process of the dashed line is not for computing LLR. The backward process for the LLR values is performed on the solid line denoted with $\beta$ and $L$. On the solid line, the backward process and LLR computation are executed simultaneously. The forward process (represented by the dashed-dotted line with $\alpha$), is done before executing the backward and LLR computation processes.

## 2.8 Soft-Output Viterbi Algorithm for Turbo Decoding Process

SOVA introduced in [67] generates the soft-decision (called soft-output or LLR) rather than the hard-decision, finding the maximum likelihood (ML) paths during the forward process. The soft-output is represented by APP, as shown below.

$$L_{lr}(k) = \log \frac{P(u(k) = + \mid y)}{1 - P(u(k) = + \mid y)} = u(k)L(k) \tag{2.8}$$

where $u(k)$ is the hard-decision and $L(k)$ is the reliability value of the hard-decision at time $k$. In MAP-based turbo decoders, $L_{lr}$(k) is calculated from the branch metrics, and all forward and backward state metrics. However, $u(k)$ and $L(k)$ in equation (2.8) are independently obtained from the forward process only, as in a Viterbi decoder. In SOVA, the hard-decision can be obtained by using traceback or register-exchange algorithms (TBA and REA) as used for traditional decoders [85-88]. On the other hand, the reliability value is obtained from the difference between the state metrics after completing the updating process. The reliability value, $\Delta(s_{k+1})$, before the updating process is represented as follows :

$$\Delta(s_{k+1}) = \left| \bar{\gamma}(s_k, s_{k+1}) + \bar{\alpha}(s_k) - \bar{\gamma}(s_k^{'}, s_{k+1}) - \bar{\alpha}(s_k^{'}) \right| \tag{2.9}$$

where $\bar{\gamma}$ and $\bar{\alpha}$ are the branch and forward state metrics, respectively. They can be obtained from the same equations (2.2) and (2.3) described in the previous section. Therefore this computation process is the same as the forward process of the MAP algorithm. However, in SOVA, the reliability value, $\Delta(s_{k+1})$, is obtained from an absolute value of the difference between the state metrics, as shown in equation (2.9). These values are accumulated and updated at each state through the survivor and update processes [67] until the final reliability value for the soft-output is decided. Then, the extrinsic information for the next iteration process can be calculated using the equation (2.7). Thus, SOVA can be used to the iteration process for the turbo decoding process, exploiting the soft-output to obtain the extrinsic information.

## 2.9 Techniques for Turbo Decoder Implementations

The MAP and SOVA described in Sections 2.6 and 2.7 are not suitable and less efficient to implement a turbo decoder in practical systems, because of their complexities. Since turbo codes were introduced, much research has focused on the implementation of the turbo decoder in order to achieve high performance in terms of area, power, throughput, and so on.

This section reviews and summarizes techniques used in the literature to implement turbo decoder hardware based on either MAP or SOVA.

### 2.9.1 Optimum and Sub-Optimum Algorithms

Equations described in Section 2.6 for the MAP algorithm are not suitable for an efficient turbo decoder implementation because of the non-linear functions. These functions can increase computational complexity, which might lead to high area usage and power consumption in hardware implementations. This problem can be addressed by using optimum and sub-optimum algorithms suggested in [15]. These algorithms are Log-MAP (LMAP) and Max-Log-MAP (MLMAP). The equations (2.2), (2.3), and (2.4) include logarithm and exponential functions which can be simplified by an approximation form (known as the Jacobi logarithm) represented as follows :

$$\ln[\exp(x)+\exp(y)] = \max[x,y] + \ln[\exp(-|x-y|)] \tag{2.10}$$

where the second term of the right hand side is the correction term that can be implemented with a simple look-up table [81]. The difference between LMAP and MLMAP algorithms is determined by the inclusion of this term. While this term is ignored in the MLMAP algorithm, it is implemented in the LMAP algorithm.

### 2.9.2 Two-Step SOVA

Two-step SOVA (TSOVA) suggested in the literature [93] was developed to reduce the computational complexity of the original SOVA described in [67]. TSOVA divides the decoding process into the survivor and update processes to reduce the complexity. These processes are executed simultaneously in the original SOVA. The survivor process performs the same function as the traceback process in a Viterbi decoder [85-86] in order to find a state for beginning the decoding process to generate the hard-decision. Thus in the TSOVA, the survivor process finds the ML path for each state to determine the state at which all ML paths converge. Then, the update process starts from the converged state to update the reliability values by following the hard-decisions provided by the survivor and competing paths. The reliability values are updated at the point where the two hard-decisions of the

survivor and competing paths are different. The update rule described in [67] is used for the update process. Since the introduction of TSOVA in [67], it has been used not only for turbo codes [94-96], but also for other applications [97-98].

### 2.9.3 Quantization Method

A fixed-point method is useful to reduce the hardware complexity. The real value transmitted over the channel can be represented in an integer value using a fixed-point representation method to save the resources needed to implement digital signal processors. However, the fixed-point level can affect both BER performance and hardware costs. This represents a trade-off between performance and cost.

In the literature [99-104], fixed-point turbo decoders have been investigated through the BER performance, varying the fixed-point level of soft-input symbol data to find an appropriate level without significant performance degradation. This fixed-point can be represented as follows [89]:

$$y_{fix} = \lfloor 2^{p} y + 0.5 \rfloor \tag{2.11}$$

where $y$ is the symbol data for the turbo decoder, $y_{fix}$ is the fixed-point representation of the $y$, and $p$ is the number of the precision bit in the fixed-point representation. In the equation, '$\lfloor y \rfloor$' means 'integer part of $y$'. If the total number of bits is $L$, the final fixed-point value, $y_L$ is

$$y_L = \begin{cases} \min(y_{fix},\ 2^{L-1} - 1) \\ \max(y_{fix},\ -2^{L-1}) \end{cases} \tag{2.12}$$

Thus the maximum or minimum fixed-point value is determined by the total number of bits.

### 2.9.4 Metric Wordlength Optimization

A digital signal processor is implemented with a finite wordlength for input, output, and internal metric representations. Thus it is clear that the size of wordlength directly affects

hardware costs. An optimized wordlength of the metrics for turbo decoder implementations without degrading BER performance has been considered in the literature [89, 101-104]. This optimization is associated with the quantization level described in the previous sub section. According to [15, 105], a total of 4-bits is the most suitable wordlength to represent soft-input symbol data. When the soft-input representation is determined, the metrics such as branch metrics, state metrics, and LLR, can be decided by high level simulation for evaluating turbo decoder performance. Larger wordlength than the 4-bits can be applied for turbo decoder implementations to achieve better performance, but that leads to increasing overheads in hardware performance in terms of area and power. Therefore the optimization should be determined by considering the trade-off between performance and hardware costs.

### 2.9.5 State Metric Normalization

A turbo decoder performs an iteration process in order to produce better performance. This process dramatically increases the state metrics during the decoding process. Thus a large wordlength is needed to avoid overflow of the state metrics in turbo decoders. However, the large wordlength clearly increases the hardware costs of turbo decoders, as described in the previous sub section. A Viterbi decoder also suffers from the overflow problem. As a result of using soft-input symbol data for the turbo decoding process, the increase rate of the state metrics in turbo decoders is much more rapid than in a Viterbi decoder. Several techniques to address this problem for Viterbi decoders have been suggested [106]. The state metrics normalization method, which is similar to the methods described in [11, 105, 107], can be used to reduce the wordlength of the state metrics, avoiding the overflow and saving the hardware costs. The normalization method is realized by subtracting a constant value from the computed state metrics when the state metrics are larger than the constant value. Thus the use of the state metric normalization can reduce the wordlength and save hardware costs.

### 2.9.6 Retiming Method

In a turbo decoder, the state metric computation unit has the highest computational complexity which increases exponentially with the constraint length (K). As a result, the number of states grows proportional to $2^K$. Also, the state metric computation process,

known as add-compare-select (ACS), is performed concurrently during the decoding process. Thus, the critical path delay of a turbo decoder is known to exist in the ACS unit. The retiming method reduces this delay by changing the order of the state metric computation process. Similar techniques to reduce the critical path delay are already employed for Viterbi decoders [108-110]. According to [34], the retiming method can reduce the critical path delay by 29%, but it leads to 99% and 89% overheads in area and power.

### 2.9.7 Adaptive Iteration Method

The iteration process of a turbo decoder corrects the errors introduced over the channel to improve BER performance, but this process can cause high latency and energy. In practical wireless communication systems, a turbo decoder performs the decoding process frame by frame. Each of the frames passed to a turbo decoder could have different error rates. Thus all the frames that have different error probabilities could be corrected by applying a different number of iterations rather than any fixed number of iterations.

The adaptive iteration method is used to reduce the number of iterations by observing the errors of each frame indirectly during the decoding process, so reducing latency and energy consumption. Several adaptive iteration methods have been suggested in the literature [111-113]. One of the methods is to observe the reliability value of LLR, during the decoding process. If all the reliability values are larger than the threshold value, the iteration process is stopped. In this way the adaptive iteration method can save energy and reduce latency.

### 2.9.8 High Radix Architecture

High radix architecture has been considered for a Viterbi decoder implementation in order to improve throughput of the decoder [114-117]. Although the high radix method increases hardware costs and complexities, it can raise throughput without high clock frequency. In [118], a high radix turbo decoder implementation based on LMAP algorithm was introduced showing how LLR and extrinsic information for the turbo decoding process are obtained from the high radix turbo decoder architecture. Several papers [11, 20] show the turbo decoder implemented with the high radix architecture in order to achieve better throughput.

### 2.9.9 Stopping Criteria

A turbo decoder performs an iteration process to achieve better performance in terms of BER. However, the iteration process causes of high latency and energy consumption. A stopping criterion is to address the issues by stopping the iteration process, which can prevent unnecessary computation and decoding delay. Thus, one can improve throughput rate and save energy consumption of turbo decoders. In [119-121], a method based cross entropy between the distributions of the estimates at the output of the decoders at each iteration has been proposed. In recent, several techniques for the stopping criteria [122-126] for an efficient turbo decoding process have been suggested. In the criteria, the most important thing is to minimize the additional algorithm used to stop the iteration without degrading the decoding performance.

### 2.9.10 Parallel Implementation

A throughput rate is always one of the most important issues in wireless and wired communication systems. One of the ways for achieving high throughput is to implement a turbo decoder in a parallel scheme. In [127-129], a parallel scheme for high throughput turbo decoder implementations has been proposed showing hardware overheads. There are several issues to implement parallel turbo decoders. They are designing interleavers [130-134] and parallel architectures [135-136] for parallel processing. Implementing a turbo decoder in parallel clearly leads to an increase of the hardware costs. Thus, there must be trade-off between throughput and performance in terms of area usage and power consumption. The performance overhead led by the parallel scheme should be acceptable in the systems requiring turbo codes. .

## 2.10 Summary

This chapter reviewed turbo encoding and decoding principles. Two sorts of turbo codes were introduced. The MAP based and SOVA based turbo decoding algorithms, and how they generate soft-output, were described and explained. For practical turbo decoder

implementations in hardware, the algorithms need to be modified in order to reduce their computational complexities. Several techniques for the turbo decoder implementations were described, including their methods for achieving high performance and efficient hardware. These techniques are utilized to implement a turbo decoder to be presented in later chapters.

# Chapter 3

# Max-Log-MAP Based Turbo Decoder Hardware Architecture

## 3.1 Introduction

Many researchers have been paying a lot of attention to the outstanding BER performance of the turbo decoder. However, the use of the turbo decoder was expected to increase the costs of receiver systems, as the authors in [3] stated that the turbo decoder's complexity is twice that of the Viterbi decoder. In general, a decoding processor is regarded as the most exhaustive processor in the systems. For this reason, the turbo decoder has been challenged by a number of researchers to reduce the computational complexity and thus save hardware costs. Among the many works described in the previous chapter, the development of Log-MAP (LMAP) and Max-Log-MAP (MLMAP) [15] contributes to a significant reduction in this complexity, and they are widely applied to turbo decoder implementations.

Another issue in turbo decoder implementations is that the iterative process increases the data output latency, which can lower the data throughput. The turbo decoder provides the LLR after computing all the forward and backward state metrics, as described in the previous chapter. This leads to a large latency, and also requires large amounts of memory to store either forward or backward state metrics until completing one of the state metric computation processes. Thus, the latency and memory size are dramatically increased, depending on the number of iterations and the block size. The sliding window (SW) method proposed in [25] can address the latency problem. In [137-138], several turbo decoding algorithms are compared in terms of implementation complexity. The results show that MLMAP with the SW method is the most suitable algorithm for the turbo decoder implementation.

This chapter presents MLMAP based turbo decoder architecture with the SW method, which is similar to the conventional SW based MLMAP turbo decoder architecture. The

architecture consists of memory blocks for controlling data input and output through the SW method, metric computation units (MCU) for the branch and state metrics, an LLR computation unit (LCU) for obtaining LLR and the extrinsic information, and memory blocks for storing input data and the metrics. This chapter discusses a detailed implementation method for the units and the design flow for the turbo decoder VLSI implementations, in order to evaluate their hardware performance in terms of area, power and critical path delay for the maximum speed. Before implementing the MLMAP turbo decoder, the turbo decoder is investigated at high level using Matlab™ to verify its functionality. For the investigation, various conditions and parameters are applied for evaluating BER performance of the turbo decoder.

The chapter is organised as follows. In Section 3.2, the MLMAP turbo decoder architecture, as well as the components incorporated into the architecture, is described in detail. Section 3.3 shows the turbo decoder performance results at high level. Hardware simulation results are discussed in Section 3.4, and Section 3.5 summarizes the chapter.

## 3.2 Max-Log-MAP and Sliding Window Method-Based Turbo Decoder Architecture

### 3.2.1 Max-Log-MAP Soft-Input Soft-Output Turbo Decoder Architecture

Figure 3.1 illustrates the MLMAP turbo soft-input soft-output (SISO) decoder architecture implemented with the SW method described in the previous chapter. The architecture consists of a data scheduling unit (DSU) for managing data input and output following the SW method, three metric computation units (MCU) for the branch and state metric computation; an LLR computation unit (LCU) for generating the hard decision and the extrinsic information, and memory blocks for delaying the soft-input (DS), the branch metrics (DB), the forward state metrics (DF) and the decoded information (DL). These

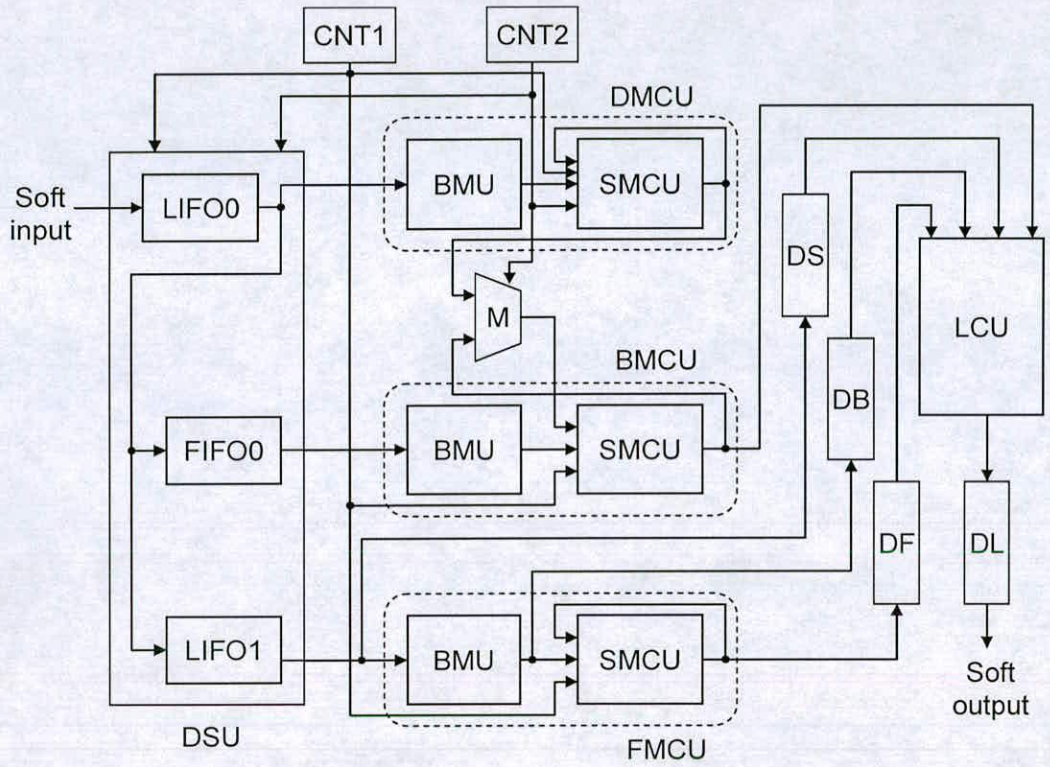



Figure 3.1 The Max-Log MAP turbo SISO decoder architecture, implemented with the SW method.

memory blocks are realised by a last-in first-out (LIFO) memory block. Two counters (CNT 1 and 2) give the DSU and each MCU the count numbers that indicate the beginning and end of the input and sub-block for the SW process.

As shown in Figure 3.1, the soft-input data is fed into the DSU and then it is distributed to each of the MCUs through first-in first-out (FIFO) and last-in, last-out (LIFO) blocks, which have the same depth with a window size of 40. A dummy-backward MCU (DMCU) computes the backward state metrics to provide the initial state metrics to backward MCU (BMCU). The forward state metrics are computed in forward MCU (FMCU), which are then delayed by DF before input to LCU, until the backward state metrics generated by BMCU

are passed to LCU. LCU calculates LLR from the branch metrics delayed by DB, the forward state metrics delayed by DF, and the backward state metrics computed by BMCU. After obtaining LLR, the extrinsic information for the next decoding process is computed by subtracting the soft-input delayed by DS from the computed LLR.

The output of LCU consists of nine bits, of which MSB is the hard decision and the rest is the extrinsic information. The DL converts the reverse ordered output into the right order. In the following subsections, a detailed structure and implementation method for the components is described.

Before designing the MLMAP turbo decoder in hardware, parameters for representing the metrics are determined from high-level simulation results described in the next section. Table 3.1 summarizes the wordlength for each parameter for MLMAP decoder implementation. The input symbols and the extrinsic information are represented with four bits and eight bits, respectively. On the other hand, the branch and state metrics are represented with eight bits and nine bits, respectively. The 1- and 0-bit LLR values for computing LLR are represented in 10 bits. All the parameters are represented in 2's complement. These parameters are similar to ones used in the literature [15, 104].

Table 3.1 Parameters used for the MLMAP turbo decoder hardware implementation.

| Metrics | Wordlength |
|---|---|
| Input Symbols | 4-bits |
| Extrinsic information | 8-bits |
| Branch Metric | 8-bits |
| Forward and Backward State Metrics | 9-bits |
| LLR 1- and 0-bit | 10-bits |

### 3.2.2 Metric Computation Unit

Figure 3.2 illustrates the MCU structure, which consists of a branch metric unit (BMU) and a state metric computation unit (SMCU). The MCU structure is used to implement the three MCUs incorporated in the MLMAP turbo decoder architecture, as illustrated in Figure 3.1. They can be implemented in a similar way. In the MCU, BMU computes the branch metrics (BM) from the soft-input data. SMCU, which consists of a number of parallel add-compare-select-normalization (ACSN) units, calculates the state metrics (SM) from the BM and previous SM values. The number of ACSN units incorporated into SMCU is given by $2^K$-1 for constraint length K.

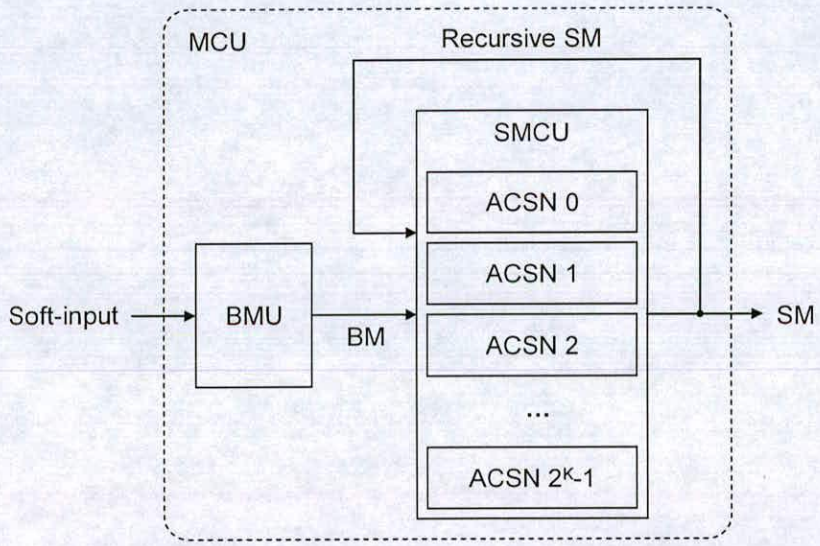



Figure 3.2 The metric computation unit structure.

#### 3.2.2.1 Branch Metric Unit

BMU calculates the branch metrics from the soft-input symbol data ($y_s$ and $y_p$) and the extrinsic information ($L_e$) provided by the previous decoding process, which is initialised to zero at the first decoding process. In the turbo decoding process, four branch metrics are needed to compute the state metrics.

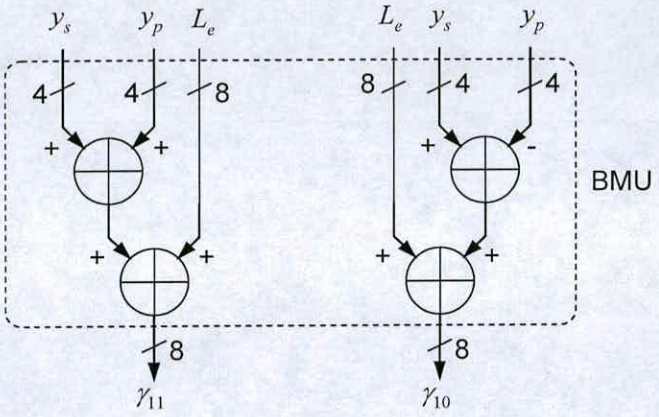



Figure 3.3 The branch metric unit structure.

Figure 3.3 illustrates the BMU structure, which consists of four adders. The BMU generates two branch metrics, $\gamma_{11}$ and $\gamma_{10}$ (the subscripts of $\gamma$ represents the code word of systematic and parity bits generated by a turbo encoder). The two branch metrics are negated to obtain the rest of the branch metrics, $\gamma_{00}$ and $\gamma_{01}$, for computing the state metrics. The BMU does not depend on the constraint length, and can be used in turbo decoder implementations with different constraint lengths.

### 3.2.2.2 Add Compare Select Normalization Unit

Most of the MCU area and power is clearly occupied and consumed by the parallel ACSN. The state metric computation process can be described by the trellis state diagram, as shown in the previous chapter. Figures 3.4 (a) and (b) show the trellis diagram of the forward and backward processes for K = 4, as an example. Each of the paths in the figure is represented in a different line style that corresponds to the code word. By initialising the forward and backward state metrics following the equation (2.5) in Chapter 2, next state metrics are calculated from the two branch and two state metrics represented by the transition paths, as shown in Figure 3.4. This procedure is recursively performed for the input block.

The structure of ACSN used in the turbo decoder implementation is similar to an ACS structure used in Viterbi decoders [114], except the state metric normalization process. In

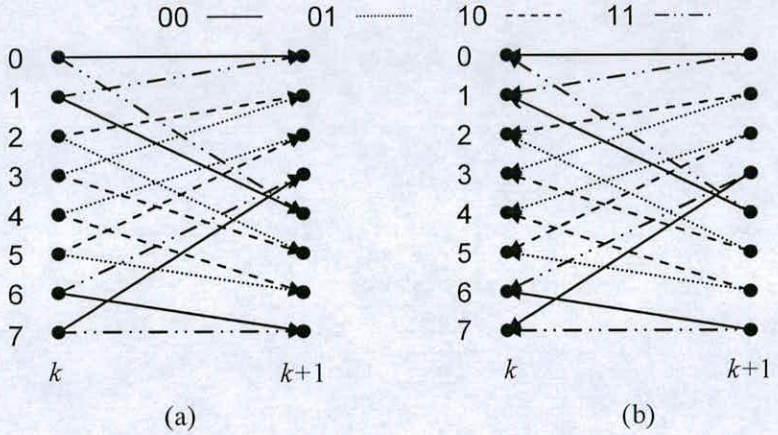



Figure 3.4 The trellis state transition diagrams of (a) the forward and (b) backward processes.

turbo decoder process, the state metrics could increase more rapidly than in a Viterbi decoder, due to the use of soft-input symbol data. Therefore; a large wordlength is required for the state metrics to avoid overflow; however, this may lead to large area and power overheads. Thus, the state metrics normalization process is necessary for an efficient SMCU implementation that has no overflow. The normalization process is performed by subtracting a constant value from all state metrics, after calculating the state metrics in every cycle, when all the computed state metrics are larger than the constant value [104].

Figure 3.5 illustrates the structure of the ACSN used for calculating the forward state metric of state 0 at time $k$, as an example. Two branch metrics generated by BMU, $\gamma_{00}(k)$ and $\gamma_{11}(k)$, and two state metrics of states 0 and 1, $\alpha_0(k)$ and $\alpha_1(k)$, are inputs of the ACSN. A pair of branch and state metrics is added using each of the adders (A), and then the results of each adder are compared to select (CS) the maximum between them. The comparison process is carried out by using an adder and a multiplexer (MUX). The most significant bit (MSB) of the adder output is used as a control signal for MUX, which selects the maximum one from its two inputs. The selected maximum value is used for next state metric computation process and for computing LLR after completing the normalization process (N). For the state metric normalization process, the selected maximum value is compared with a constant value

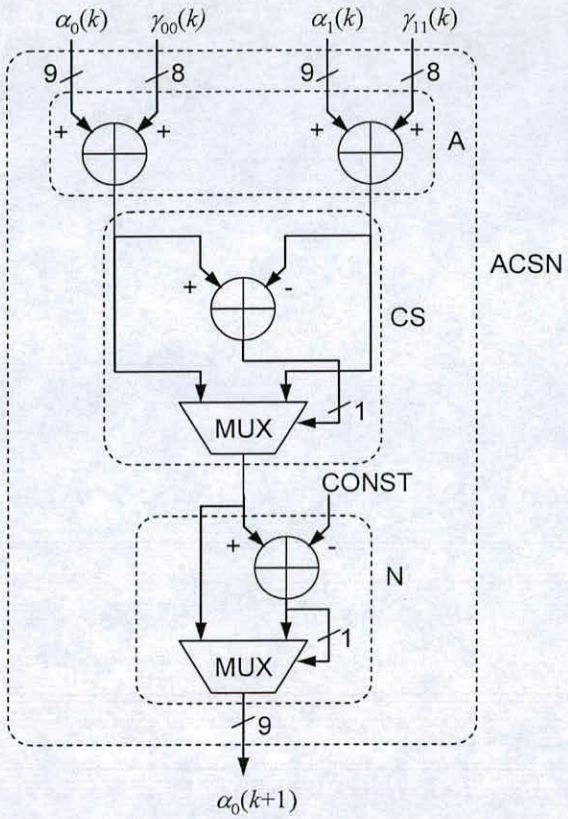



Figure 3.5 An example of the add-compare-select-normalization structure for computing the forward state metric at state 0.

(CONST) to determine whether the maximum value is larger than CONST or not. If the maximum value is larger than CONST, CONST is subtracted from the maximum value. If the maximum value is not larger than CONST, the maximum value becomes the state metric, $\alpha_0(k+1)$, without the normalization process.

The ACSN process is concurrently performed during the turbo decoding process, using the output SM as the input of the ACSN itself recursively. Thus, the whole ACSN process must be completed in one clock cycle concurrently, due to the recursive SM input to ACSN itself. In this case, the critical path delay of the MLMAP turbo decoder is determined by the ACSN [13], which can be approximated as follows:

$$d_{ACSN} = 3\times t_a + 2\times t_m, \tag{3.1}$$

where $t_a$ is the delay of the adders used in A, C and N processes, and $t_m$ is the delay of the MUX. This ACSN critical path delay, $d_{ACSN}$, determines the maximum speed of the turbo decoder for achieving maximum throughput. Other modules – such as BMU and LCU, which are described in the next subsection – can be implemented with pipelining method, to reduce the path delay. ACSN can also be implemented with pipelining method to reduce the delay. In this case, however, the pipelining for ACSN does not help to improve the throughput, since the output is also delayed by the number of pipelined stages of ACSN.

### 3.2.3 Log-Likelihood Ratio Computation unit

After obtaining the branch metrics and all the forward and backward state metrics, LLR can be calculated. Before computing the LLR, the computation process of the 1- and 0-bit LLR values, which are the numerator and the denominator in equation (2.1), respectively, can be described by the trellis state diagram illustrated in Figures 3.6 (a) and (b). The figures illustrate the state transition paths denoting the code words in different line styles to distinguish the computation of 1-bit and 0-bit LLR ($L_{lr1}$ and $L_{lr0}$) for constraint length K = 4.

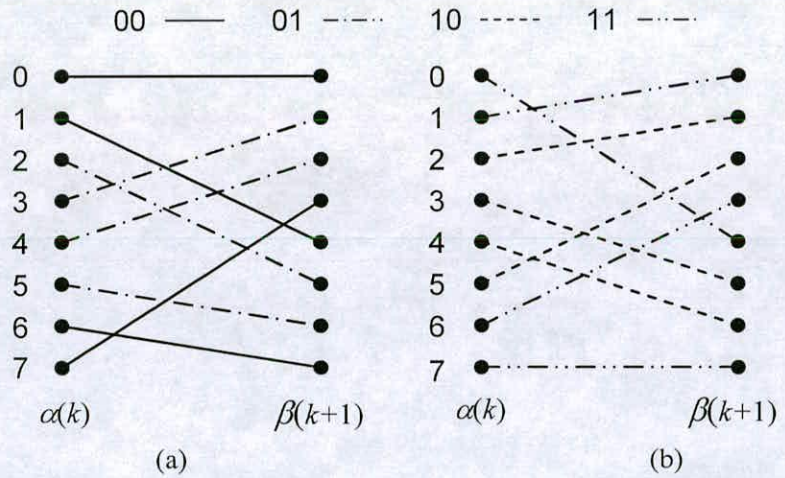



Figure 3.6 The trellis state transition paths for computing (a) $L_{lr0}$ and (b) $L_{lr1}$.

For instance, Figure 3.6 (b) shows the paths with the code words of 11 and 10, in which the first digit, 1, implies the systematic bits. Thus, $L_{lr1}$ values are calculated from the two-branch metrics ($\gamma_{11}$, $\gamma_{10}$) and the forward state metric at time $k$, $\alpha(k)$, and the backward state metric at time $k$+1, $\beta(k+1)$, indicated by the ends of the paths shown in Figure 3.6 (b). In the case of K = 4, 8 $L_{lr1}$ values are computed, and then the maximum among them is selected to calculate the LLR. In the same way, the maximum $L_{lr0}$ value can be obtained using the diagram shown in Figure 3.6 (a). Thus, the LLR value is obtained by subtraction as follows:

$$L_{lr} = \max[L_{lr1,0} \ldots L_{lr1,7}] - \max[L_{lr0,0} \ldots L_{lr0,7}] \qquad \text{for} \quad K = 4 \tag{3.2}$$

Figure 3.7 illustrates the LCU structure for calculating the LLR and extrinsic information. It consists of a LCU0, a LUC1 and adders. The LCU inputs are the $2^{K-1}$ numbers of the forward and backward state metrics for constraint length K, and the four branch metrics. LCU0 and LCU1 compute the $L_{lr0}$ and $L_{lr1}$ values. Then, $L_{lr}$ is obtained by subtracting $L_{lr0}$ from $L_{lr1}$. After that, the extrinsic information, $L_e$, is obtained by subtracting the soft-input from $L_{lr}$.

The structure illustrated in Figure 3.8 is commonly used to implement the LCU1 and LCU0 with different input metrics. The structure, given as an example, generates $L_{lr1}$ for constraint

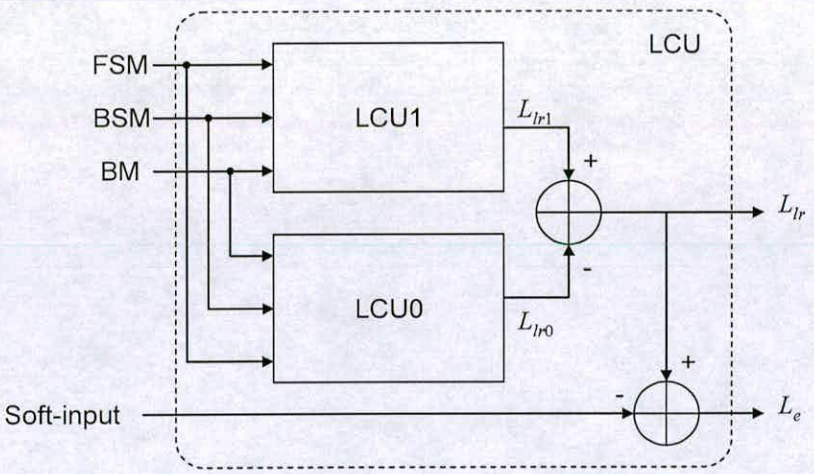



Figure 3.7 The LLR computation unit structure.

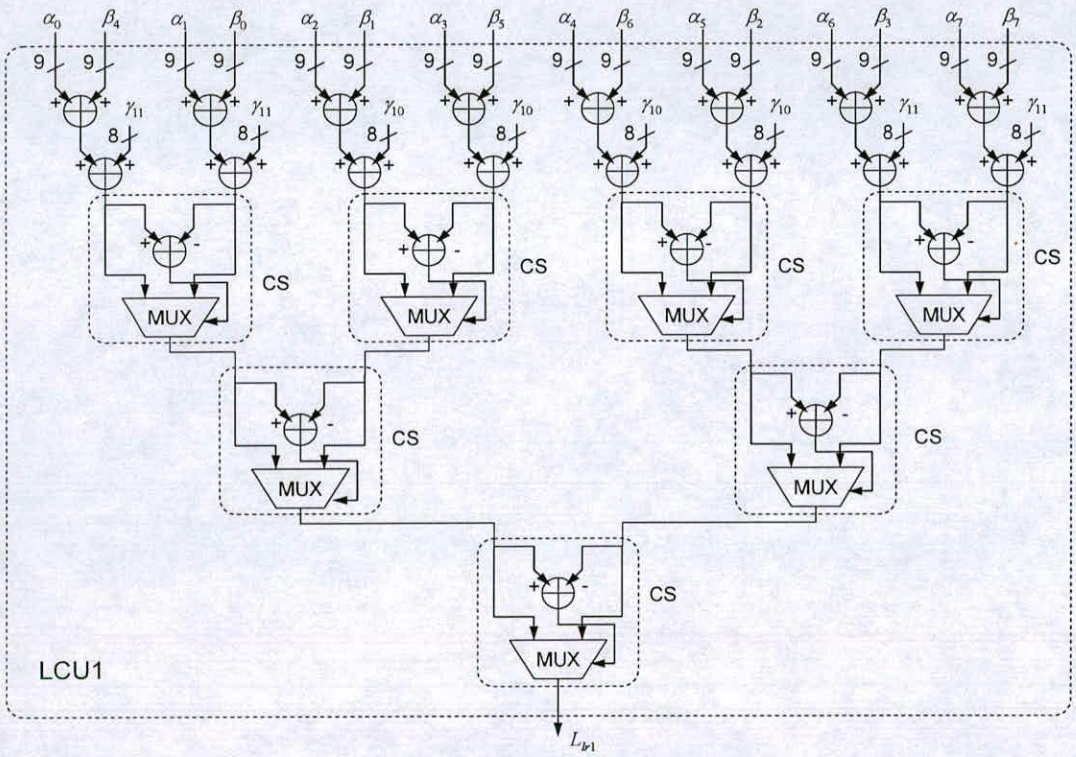



Figure 3.8 The LCU1 structure for constraint length K = 4.

length K = 4. In this structure, eight numbers of the forward ($\alpha_0$ - $\alpha_7$) and backward ($\beta_0$ - $\beta_7$) state metrics and two branch metrics ($\gamma_{11}$ - $\gamma_{10}$) are input to LCU1. The first two adder rows generate eight $L_{lr1}$ values, then the maximum among them is determined by repeating the compare-select (CS) processes. After completing the CS process, $L_{lr1}$ is finally output from the LCU1.

The critical path delay of the LCU1 (or LCU0) structure can be given as follows:

$$d_{\mathrm{LCU1}} = 5\times t_{\mathrm{a}} + 3\times t_{\mathrm{m}}, \tag{3.3}$$

where $t_{\mathrm{a}}$ is the delay of the adder and $t_{\mathrm{m}}$ is the delay of the MUX. As can be seen, $d_{\mathrm{LCU1}}$ is larger than the delay of ACSN ($d_{\mathrm{ACSN}}$) given by equation (3.1). In contrast to ACSN,

however, this delay can be reduced by using pipelining method. The number of pipelining stages varies with the constraint length, K, due to the different number of CS processes involved.

# 3.3 High Level Simulation Results

## 3.3.1 Simulation Specifications and Systems

The turbo decoder simulation model for high level simulation can be represented as follows:

$$\mathbf{y}(k) = \mathbf{x}(k) + \mathbf{n}(k) \tag{3.4}$$

where $\mathbf{x}(k)$ is the encoded data for transmission, $\mathbf{n}(k)$ is the additive noise, and $\mathbf{y}(k)$ is the received data for the decoding process. Figure 3.9 illustrates a block diagram for the simulation. The binary source data are generated at random. The turbo encoder encodes the source data, and then, the encoded data are modulated in the binary phase shift keying

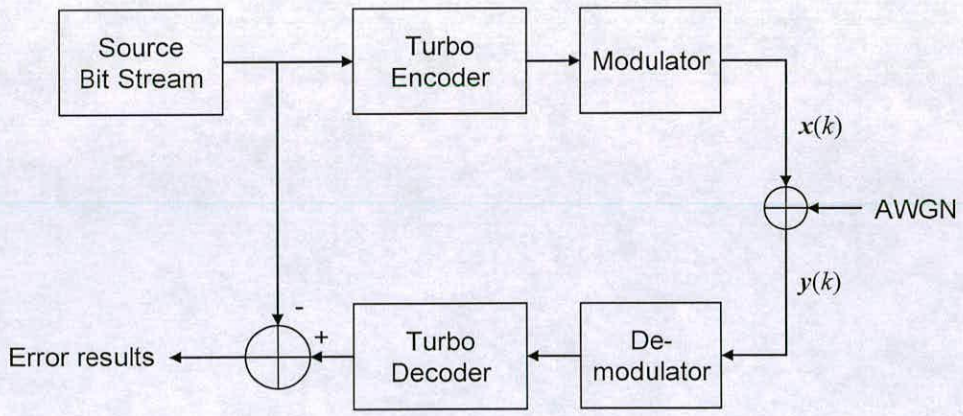



Figure 3.9 A block diagram of a test system for turbo decoder BER evaluation.

(BPSK) modulation method. The modulated data are transmitted over the additive white Gaussian noise (AWGN) channel model. After the received data are demodulated, they are decoded by the turbo decoder. The decoder output is compared with the source data to evaluate BER performance.

In the simulation, turbo codes are generated for constraint length K=3, 4, and 5 using the encoder illustrated in the previous chapter. It is assumed that an additive white Gaussian noise (AWGN) channel model with mean 0 and variance 1, code rate 1/3, and a block interleaver of 1024 bits with three tail bits are used.

### 3.3.2 Performance for Varying Block Size

There are various factors that affect the BER performance of a turbo decoder. One of them is the block size for the turbo decoding process, which in turn effects the interleaving. When the block size is large, the distance between two neighboring symbols can be increased by using the interleaving process. This can help to reduce the burst errors. In contrast, a small

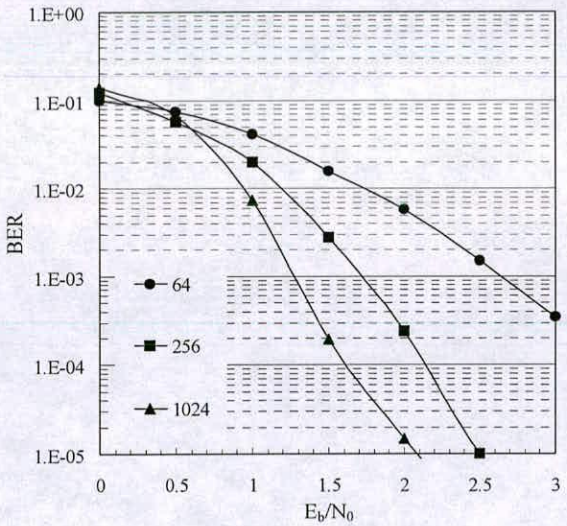



Figure 3.10. BER performance for different size of block interleaver.

block size decreases the interleaving effect as a result of the reduced distance, and provides a lower BER performance than a large block size.

Figure 3.11 shows the BER performance results for different block sizes, 64, 256, and 1024. The simulation results are based on K=4 MAP based turbo decoder with the parameters and conditions described in the previous subsection, after completing 8 iterations for each block. A block interleaving method described in Section 2.4 of Chapter 2 was applied for performing the simulations. As can be seen, the results show that the increase in block size improves the turbo decoder BER performance. In next simulations, the 1024 size block interleaver is used to evaluate BER performance of the turbo decoder.

### 3.3.3 Fixed-Point Implementation

Following the fixed-point representation method described in the previous chapter, BER performance of the MLMAP based turbo decoder for constraint length K=4 was investigated by varying the number of total and precision numbers of the bits. The parameters and simulation conditions described in Section 3.4.1 were used. Figures 3.11 (a), (b), and (c) illustrate the BER performance results after completing 8 iterations for (4.1), (4.2), and (4.3), respectively, in which 4 is the total bit number and 1, 2, and 3 are the precision bit numbers. In the simulation results, (4.2) provides the best BER performance of the fixed-point representations. For further investigation, the simulations were repeated with increasing the total number of bits to 5. Figure 3.12 shows the BER performance results of the 8th iteration for (4.1), (4.2), (4.3), (5.2), and (5.3). In the results, (5.3) provides the best BER performance followed by (5.2). However, it can be seen that the performance degradation of (4.2) is not significant as compared with the performance obtained from the fixed-point implementation with a total of 5-bits. Therefore, in this work, the soft-input will be represented with the (4.2) fixed-point representation scheme.

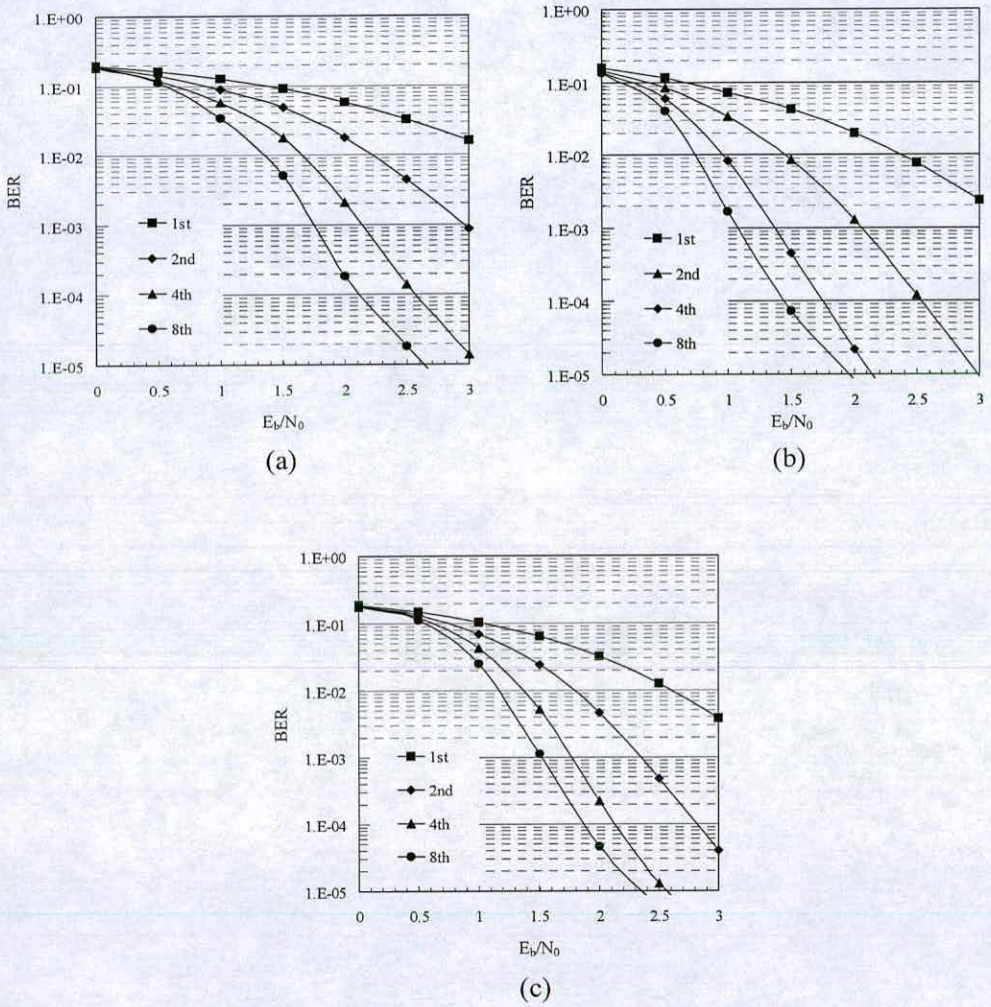



Figure 3.11 BER performance for different fixed-point representations. (a) (4.1), (b) (4.2), and (c) (4.3).

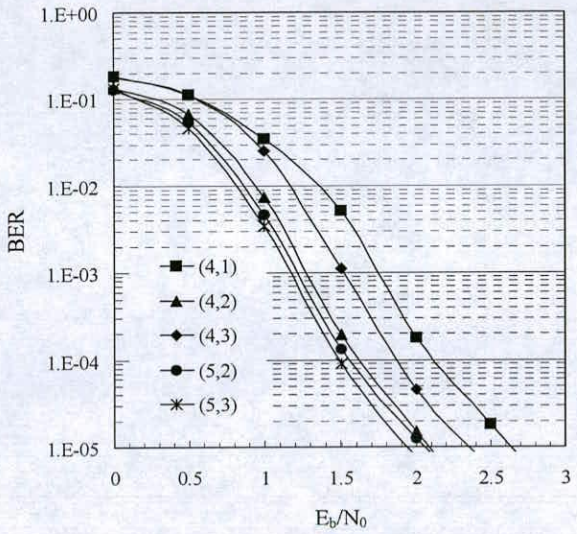



Figure 3.12. BER performance comparison for different fixed-point representation.

### 3.3.4 Performance with Varying Window Size

It is well known that the BER performance of the SW based MLMAP turbo decoders depends on the window size. If the window size is small, the state metrics computed by the dummy-backward process could be wrong, which can degrade the performance. If the window is large enough, there is no performance degradation, but latency could be increased.

The BER performance has been evaluated by varying the window size for different constraint lengths. The simulation was carried out with the same conditions used in the previous sub sections. Figures 3.13 (a), (b) and (c) show the BER performance results when the window size was varied, after completing eight iterations for constraint length K = 3, 4 and 5, respectively. The simulation results were obtained for window sizes from 4 to 20, with a step of 4, and 40. As can be seen, the performance is almost saturated at the window size of 12, 16 and 20 for K = 3, 4, and 5, respectively. These results show that the window size for achieving proper BER performance is strongly related to the constraint length.

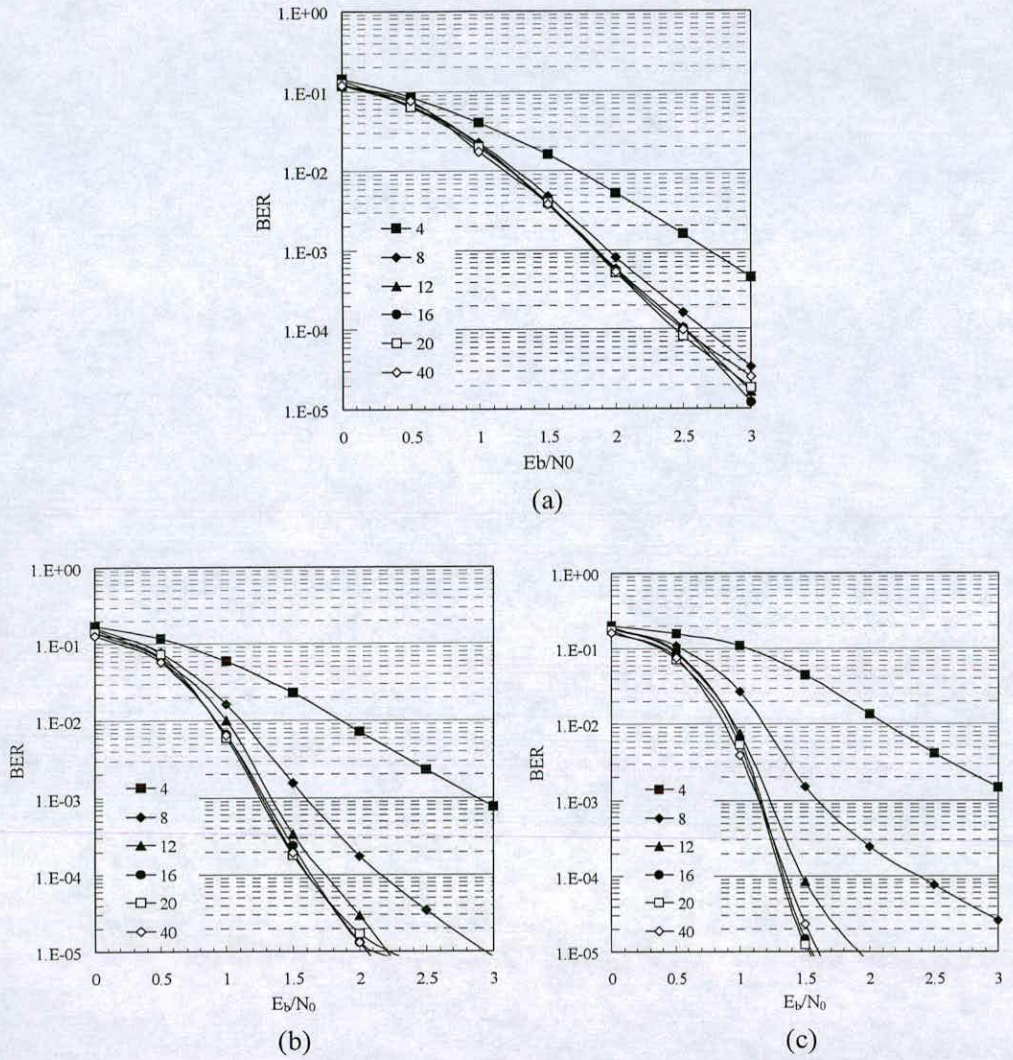



Figure 3.13 BER performance upon varying the window size: (a) K = 3, (b) K = 4 and (c) K = 5.

In this work, the window size is determined by considering of the minimum block size of 40 specified for turbo codes in 3GPP [62].

### 3.3.5 BER Performance Comparisons

This subsection shows the turbo decoder BER performance results for constraint lengths K=3, 4, and 5. Two turbo decoders, based on MAP and MLMAP algorithms, have been implemented in order to investigate the performance difference between the two algorithms. MAP and MLMAP turbo decoders were simulated with both real and fixed-point values. Thus, the BER simulation results can reflect the performance gap between the algorithms and also the representation methods, as well.

Figures 3.14, 3.15, and 3.16 illustrate the BER simulation results for constrain length K=3, 4, and 5, respectively, after completing 1, 2, 4, and 8 iterations. As in the previous subsections, the parameters described in Section 3.4.1 were used in these simulations. The results show a feature of the turbo decoder performance that the performance improves with increasing the number of iterations. In the figure, the real value MAP and the fixed-point MLMAP based turbo decoder results are represented in dashed and solid lines, respectively. As expected,

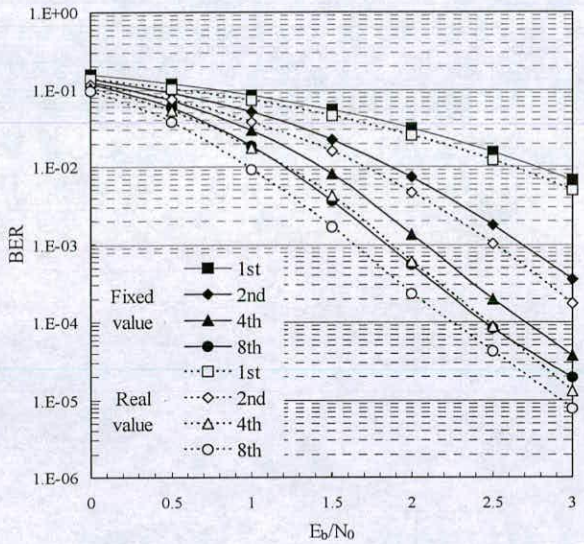



Figure 3.14 BER performance comparison between real and fixed-point values for K=3.

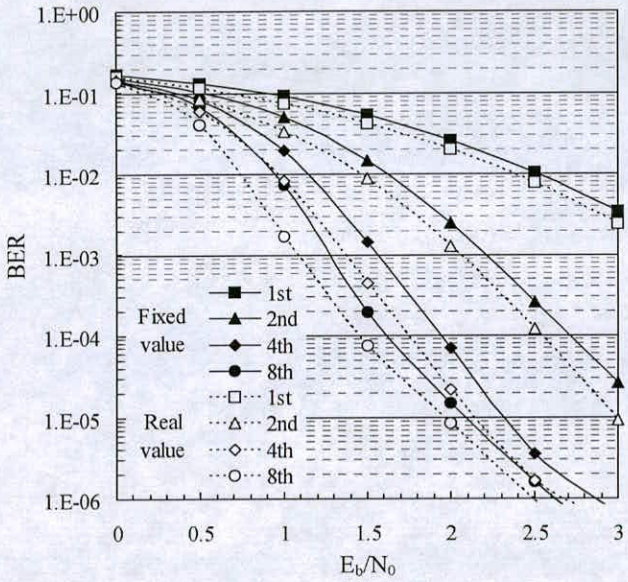



Figure 3.15 BER performance comparison between real and fixed-point values for K=4.

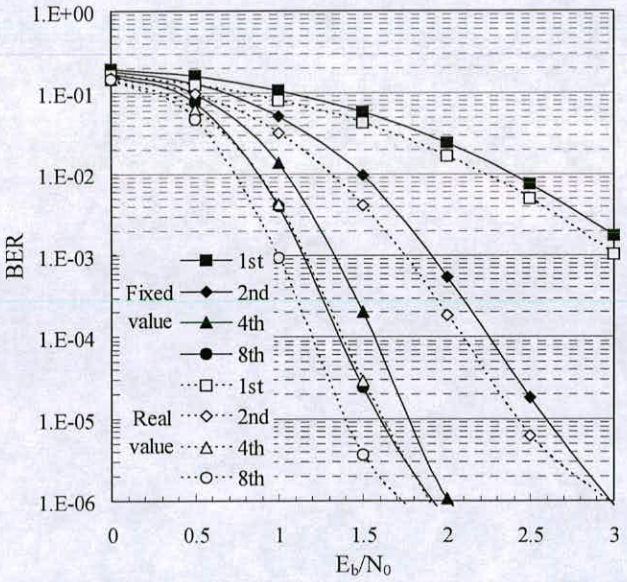



Figure 3.16 BER performance comparison between real and fixed-point values for K=5.

the real value MAP turbo decoder provides better performance than the fixed-point MLMAP based turbo decoder, due to the approximation. However, as can be seen, there is no significant performance improvement after completing 4 iterations. The simulation results show that the fixed-point representation and MLMAP algorithm lose a coding gain of around 0.25, 0.15, and 0.15 in $E_b/N_0$ at a BER of $10^{-4}$ for K=3, 4, and 5, respectively, after completing 8 iterations.

# 3.4 Hardware Design and Results

## 3.4.1 Turbo Decoder Hardware Design Flow

This section shows the design flow utilised in this thesis for the turbo decoder hardware implementation. Figure 3.17 shows high level simulation flow for verifying turbo decoding algorithm. It starts by determining the specifications of the turbo encoder and decoder. At this stage, the algorithm used to implement the turbo decoder is verified by evaluating its BER performance. This step is important in verifying the functionality of the turbo decoder. Figure 3.18 illustrates how the turbo decoder hardware is designed, verified, and evaluated in this thesis. The hardware is designed at a register transfer level (RTL) using the Verilog hardware description language (HDL). Some of the parameters needed to design optimised hardware are provided by high-level simulation results. The hardware is then synthesised using a standard CMOS technology library.

Gate-level netlist generated by the synthesis process are simulated with the timing information to verify the functionality. After completing the synthesis, the power consumption of the hardware can be estimated by using the capacitance obtained from the synthesis and the switching activity generated during the netlist simulation. The physical synthesis process for generating layout can then be performed with the netlist and the information generated by the synthesis process. The layout of the hardware is produced through floor planning, placing and routing processes. After completing the layout generation, the hardware power can also be estimated in the same way with the gate-level simulation.

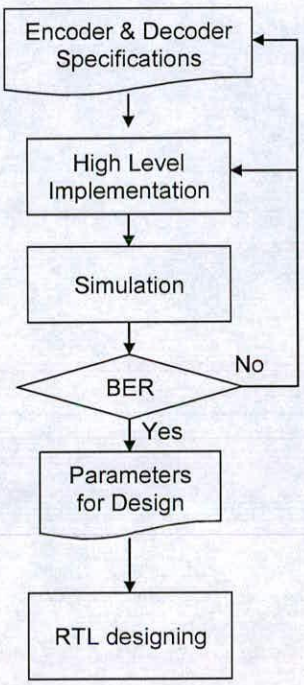



Figure 3.17 The design flow for high level simulation.

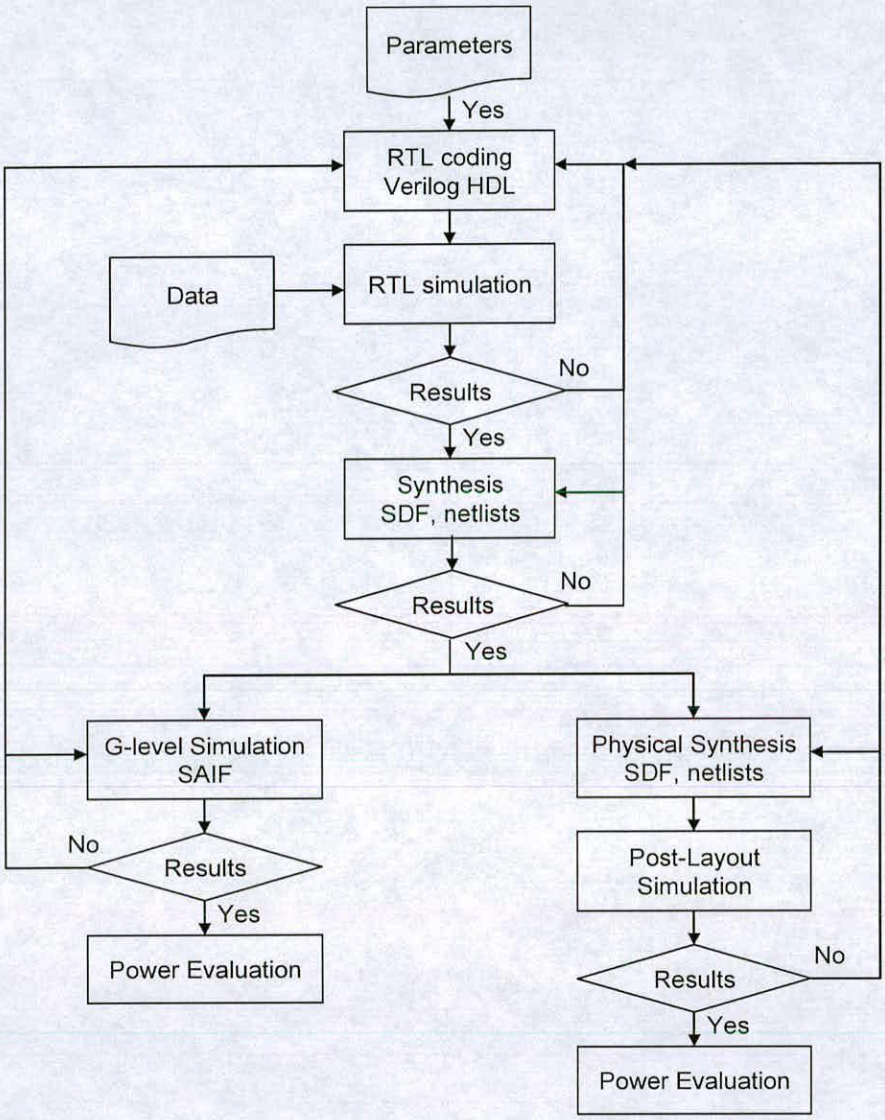



Figure 3.18 The design flow for the turbo decoder hardware implementation.

### 3.4.2 Hardware Test Environment

A test system for the MLMAP turbo decoder hardware functional verification has been built using Matlab™ and Verilog HDL. Figure 3.19 illustrates the test system for the verification. The data fed into the testbench for hardware verification is generated in Matlab™, as used in the high-level simulation described in Section 3.4. The source bits generated at random are encoded, and then the symbol data transmitted over the AWGN channel is converted into the fixed-point representation, so that it can be provided to the testbench. At the same time, the interleaver and deinterleaver addresses generated by the interleaver and deinterleaver address generators (IAG/DAG) are also sent to memory in the testbench. The turbo encoders for constraint lengths K = 3, 4 and 5 described in Chapter 2 are used, and the parameters for the data generation are the same as used in Section 3.4.

The testbench for the MLMAP turbo decoder hardware is designed using Verilog HDL. It incorporates the memory blocks for the interleaver and deinterleaver with the input data. The sequences for the turbo decoding process and the number of iterations are controlled and generated by the testbench, which can be used not only for the RTL-level simulation, but also the gate-level simulation.

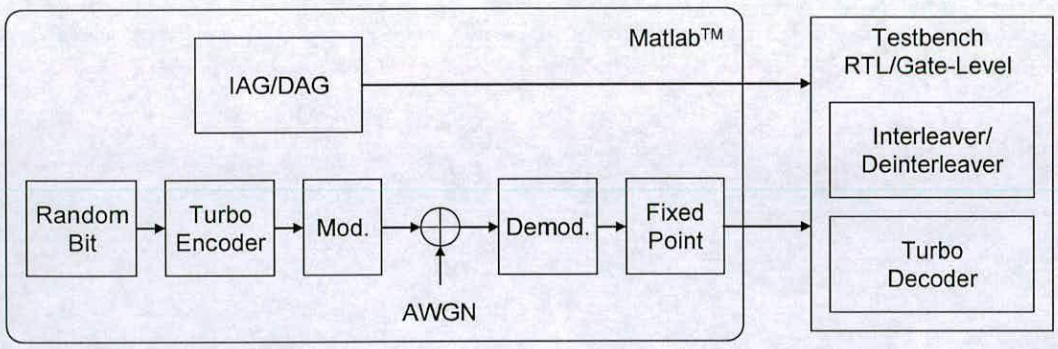



Figure 3.19 The test system for the MLMAP turbo decoder architecture.

### 3.4.3 Evaluating Power

The sources of power consumption in CMOS circuits are switching, short-circuit and leakage powers [140-141]. The switching power is the power consumed in charging and discharging the load capacitance; it accounts for most of the total power consumption in the circuits. The switching power, $P_{sw}$, can be represented as follows:

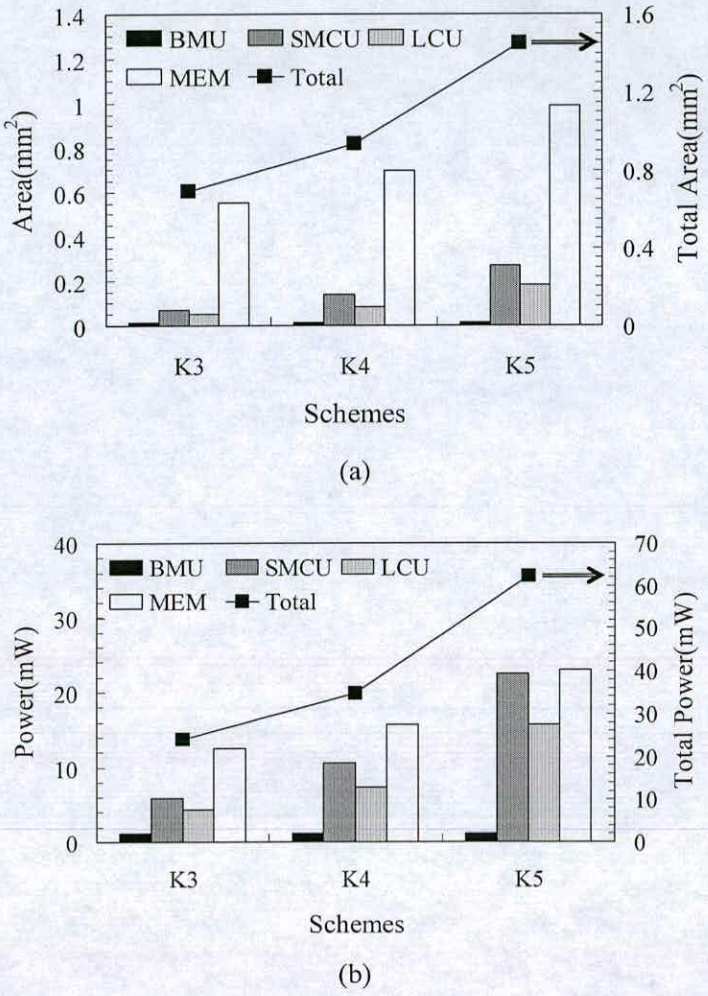
$$P_{sw} = \alpha_{sw} C_{load} V_{dd}^2 f \tag{3.4}$$

where $V_{dd}$ is the supply voltage; $f$ is the clock frequency; $C_{load}$ is the load capacitance of the gate; and $\alpha_{sw}$ is the switching activity factor, which is defined as the average number of gate transitions ( $1 \rightarrow 0$ or $0 \rightarrow 1$).

### 3.4.4 Post-Synthesis Results

As shown in the design flow in Figures 3.17 and 3.18, the MLMAP turbo decoders for constraint lengths K = 3, 4 and 5 were designed at RTL level using Verilog HDL, after completing the high-level simulations discussed in the previous section. The correct functionality of the Verilog code at RTL level was tested with Cadence Verilog-XL™, with the testbench shown in Figure 3.19. Then, the turbo decoder was synthesised using the UMC 0.18μm standard CMOS library with Synopsys DesignCompiler™ to obtain gate-level netlists. These gate-level netlists were simulated with Cadence Verilog-XL™, using the timing data produced by the synthesis. During the gate-level simulation, the switching activities of all the circuit nets of the MLMAP turbo decoder hardware were captured to be used for the power consumption using Synopsys PowerComplier™. These processes were carried out at a clock frequency of 50 MHz.

Figures 3.20 (a) and (b) illustrate the total and breakdown area and power results of the MLMAP turbo decoders, respectively, for the constraint lengths K = 3, 4 and 5. In the graphs, the primary and secondary y axes show results for individual components and the whole design, respectively. As can be seen, the increase in total area and power is not linear with




Figures 3.20 (a) Area and (b) power simulation results of the MLMAP turbo decoders for K = 3, 4 and 5.

the increase of constraint length, K. This could be explained with the exponential increase of the number of states given by $2^{K-1}$.

The area results show that MEM occupied the most area with 80%, 74% and 64% for K = 3, 4 and 5, respectively. This is followed by SMCU, LCU, and BMU. On the other hand, the power results show different aspects than the area results. The power consumed by MEM is equal to or less than the total power of SMCU, LCU, and BMU for K=3, 4, and 5. As K

Table 3.2 The MLMAP turbo decoder hardware performance results for constraint lengths K = 3, 4 and 5. The power consumption results are evaluated at a clock frequency of 50MHz.

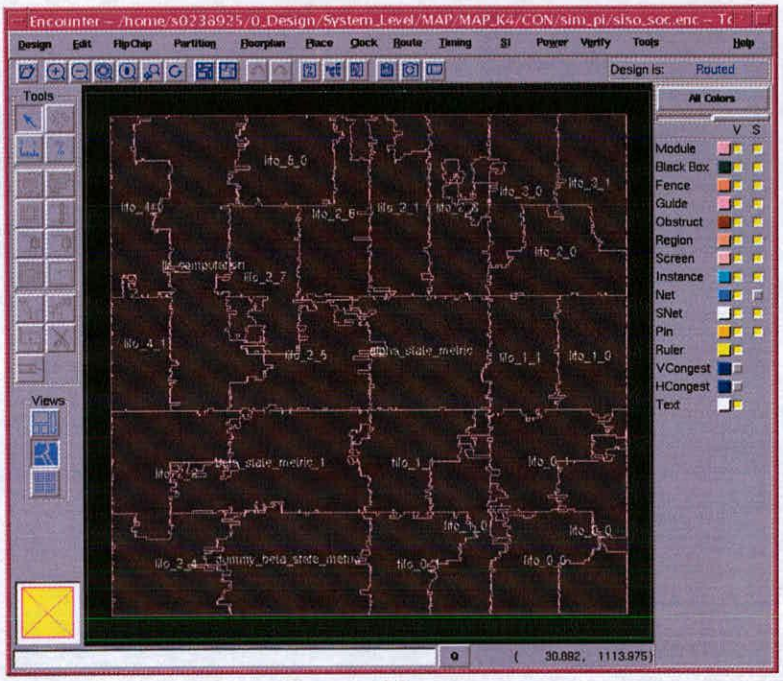
| | K3 | | K4 | | K5 | |
|---|---|---|---|---|---|---|
| | Area ($mm^2$) | Power (mW) | Area ($mm^2$) | Power (mW) | Area ($mm^2$) | Power (mW) |
| BMU | 0.014 | 1.21 | 0.014 | 1.21 | 0.014 | 1.20 |
| SMCU | 0.071 | 5.9 | 0.137 | 10.54 | 0.268 | 22.46 |
| LCU | 0.049 | 4.32 | 0.083 | 7.35 | 0.180 | 15.65 |
| MEM | 0.553 | 12.51 | 0.698 | 15.66 | 0.988 | 22.99 |
| Total | 0.689 | 23.95 | 0.934 | 34.78 | 1.452 | 62.32 |

increases, the power of SMCU and LCU starts to dominate the overall power consumption, due to increased computational complexity.

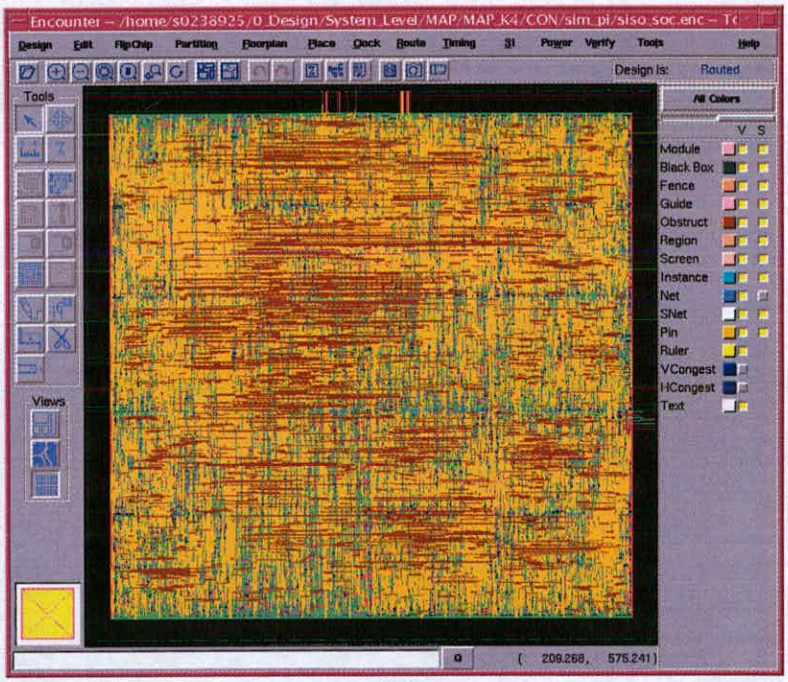
The critical path delay of each MLMAP turbo decoder architecture was investigated, and the results are depicted in Table 3.3. In all cases, the critical path delay was in the ACSN unit, whose structure does not change with different constraint lengths. Although the critical path delay for K = 3 is the shortest, there is no significant difference among the results. It can be assumed that the differences were led by the different input and output interfaces of the SMCU for the different constraint lengths.

Table 3.3 The MLMAP-based turbo decoder critical path delay for constraint lengths K = 3, 4 and 5.

| | K = 3 | K = 4 | K = 5 |
|---|---|---|---|
| Critical path delay | 4.49nsec | 5.02nsec | 4.94nsec |




(a)




(b)

Figures 3.21 (a) Floor planning and (b) placed and routed results of the MLMAP turbo decoder architecture for K = 4.

### 3.4.5 Post Layout Results

The gate-level netlists and timing information obtained from the synthesis were used to generate the layout of the MLMAP turbo decoder by using Cadence SOC Encounter™. Figures 3.21 (a) and (b) show the floor planning and place routing results, respectively, for K = 4. Table 3.4 shows performance results comparing with state-of-art turbo decoder implementations. In the table, the area results are core area without including an area of pads for inputs and outputs. The total number of gates for each turbo decoder is equivalent to the total number of NAND gates. Each throughput of the turbo decoders is estimated after 8 numbers of iterations. The turbo decoder hardware performance in terms BER has been justified by comparing its outputs with the high level simulation results. It also shows that the power consumption results of the turbo decoders are similar to the results shown in Table 3.2.

Table 3.4 Performance comparison with state-of-art turbo decoder implementations.

| | This work | | | [11] | [13] | [12] |
|---|---|---|---|---|---|---|
| | K = 3 | K = 4 | K = 5 | | | |
| Area ($mm^2$) | 0.69 | 0.93 | 1.45 | 1.5 | 14.5 | 10 |
| No. of gates | 59K | 80K | 124K | 150K | 410K | - |
| Max. Speed | 200MHz | 180MHz | 180MHz | 285MHz | 145MHz | 352MHz |
| Throughput (Mb/s) | 25 | 22.5 | 22.5 | - | 24 | 352 |
| Technology | 0.18μm | 0.18μm | 0.18μm | 0.18μm | 0.18μm | 0.13μm |
| Note | - | - | - | K=3 | K=4 Radix-4 | K=5 |

## 3.5 Summary

This chapter presented MLMAP-based turbo decoder hardware architectures with the SW method for K = 3, 4 and 5. In the architecture, the SW method was realised by using LIFO and FIFO blocks. A detailed structure of the decoding process units incorporated in the architecture was described and the implementation method and the functionality for the decoding process were explained. In the simulation results, the turbo decoder performance in terms of BER was examined in various ways to justify the algorithms and other factors that affect the turbo decoder performance. The real MAP and the fixed-point MLMAP based turbo decoders have been compared in terms of BER performance in order to observe the performance differences between the two turbo decoder implementations. After completing high level simulations, the MLMAP turbo decoder architectures have been designed using Verilog HDL and then synthesised using UMC 0.18μm standard CMOS technology. The area, power, and critical path delay results of the MLMAP turbo decoder architecture for different constraint lengths were estimated, and the total number of gates was obtained after generating the layout. In this thesis, the hardware results obtained in this chapter are considered consistent with those of a conventional MLMAP turbo decoder, and so the results are compared with those of proposed turbo decoders described in later chapters.

# Chapter 4

# High Performance Max-Log-MAP Turbo Decoder Architecture

## 4.1 Introduction

This chapter presents high performance MLMAP based turbo decoder architecture in terms of low power, efficient area and high speed for high throughput. In MLMAP turbo decoder VLSI implementations, power and area are always one of the most important issues, if they target portable and mobile equipment operated by a battery. Many works have tried to achieve low power and area efficient turbo decoder implementations [101-105, 107]. Moreover, another important issue in wireless communication systems is throughput to support a variety of services requiring high data rate, such as multi-media broadcasting. A high throughput turbo decoder can be realized by the turbo decoders in a parallel scheme [106, 142-147]. However, parallel implementations might lead to large area and power overheads, and would not be suitable to be employed by portable and mobile systems. Without these overheads, one of the ways to achieve high throughput is to implement a high speed turbo decoder. The literature, however, considers high speed turbo decoder implementations less than parallel implementations.

This chapter presents a novel method for achieving high speed turbo decoder implementations to provide high throughput data rate without significant overheads of hardware costs. The novel method described in this chapter is to normalize branch metrics before using in the state metric computation process. It reduces a critical path delay of a turbo decoder in a unit for computing the state metrics. That results in increasing the maximum speed of a turbo decoder. Furthermore, area and power for the high throughput turbo decoder are improved by reducing memory blocks and implementing efficient computational logics. A detailed structure and implementation method for improving the

performance is described. The hardware performance results are compared to the conventional turbo decoder results given in the previous chapter.

This chapter is organized as follows. Section 4.2 describes the method to increase the MLMAP turbo decoder speed for high throughput. A detailed implementation method and structure for that are shown in Section 4.3. The architecture for low power and efficient area is described in Section 4.4. The turbo decoder simulation results are given in Section 4.5. Section 4.6 summarizes the chapter.

## 4.2 High Speed Turbo Decoder Implementation Method

### 4.2.1 Critical Path Delay in Turbo Decoder

As already stated in the previous chapter, it is well know that the process of computing the state metrics is the bottleneck for achieving high speed turbo decoder implementation. In addition, the state metrics grow dramatically and need to be represented by a large wordlength to prevent overflow. The same problem also exists with more traditional decoders, such as a Viterbi decoder. Several techniques for preventing the overflow in a Viterbi decoder were suggested in [106]. These techniques not only reduce the critical path delay, but also conserve hardware resources. A detailed data path for a Viterbi decoder has been studied to reduce the critical path delay in add-compare-select (ACS) units [148]. These methods could be applied to turbo decoder implementations as well. However, the critical path delay in turbo decoders is much longer than in Viterbi decoders because MAP based turbo decoders use soft-input information and require the computation of all state metrics for forward and backward processes. One popular technique to address the overflow problem is to normalize the state metrics by subtracting a constant value from all state metrics when one of the computed state metrics is larger than a threshold value [105, 107]. This technique was also described for Viterbi decoder implementations in [106]. The state metric normalization process has been applied to the conventional turbo decoder presented in the previous chapter

for implementing add-compare-select-normalization (ACSN). In this chapter another method will be introduced for achieving high throughput turbo decoder implementations.

### 4.2.2 Branch Metric Normalization

A branch metric normalization (BMN) method is proposed to address the inherent critical path delay problem. The BMN method aims to increase the MLMAP turbo decoder maximum speed by reducing the critical path delay. The method removes the state metric normalisation process of ACSN, so only ACS is used for turbo decoder implementations. This method can be applied not only to turbo decoders, but also Viterbi decoders, if the state metric computation process incorporates the state metric normalisation process. The rule of BMN is summarized as follows :

$$
\begin{aligned}
&\text{For } n = 0 \ldots N \\
&\text{if all state metrics} > V_d \\
&\quad \eta_n(k) = \gamma_n(k) - \max[\gamma_0(k), \ldots, \gamma_n(k)] \\
&\text{else if all state metrics} < V_d \\
&\quad \eta_n(k) = \gamma_n(k) - \min[\gamma_0(k), \ldots, \gamma_n(k)] \\
&\text{else} \\
&\quad \eta_n(k) = \gamma_n(k)
\end{aligned}
\tag{4.1}
$$

where $n$ denotes the total number of branch metrics, $\gamma_n(k)$ is the branch metric at time $k$, $\eta_n(k)$ is the normalized branch metric, and $V_d$ is a constant value. The state metrics are used to determine whether the branch metrics need to be normalized by the maximum or minimum branch metric. If all state metrics are larger than $V_d$, the maximum branch metric is used for the normalization. However, if all state metrics are less than $V_d$, then the minimum branch metric is used for the normalization. If these conditions are not satisfied, the branch metrics are passed on to SMCU without normalization. From these conditions, we can see that the normalized branch metrics are either equal to or less than zero if all state metrics are larger than $V_d$. Similarly, normalized branch metrics are either equal to or greater than zero if all state metrics are less than $V_d$.

### 4.2.3 Variation of State Metrics with BMN

Employing the BMN method does not affect BER performance of MLMAP turbo decoders. During the simulations, all of the forward and backward state metrics have been observed in order to investigate their variation. The BMN process relies on a constant value of $V_d$ to determine when the branch metrics need to be normalized. The simulations were repeated for

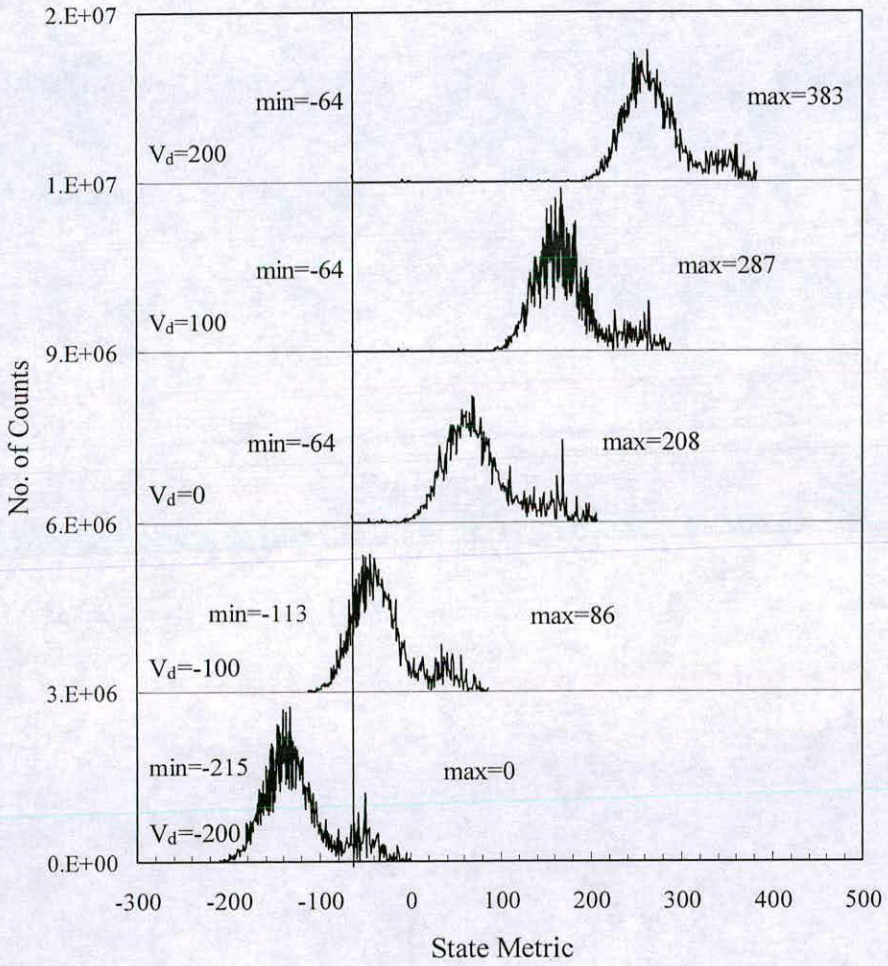



Figure 4.1 The state metric variations of the MLMAP turbo decoder with the BMN method for different $V_d$ values when the initial state metrics are set to -64.

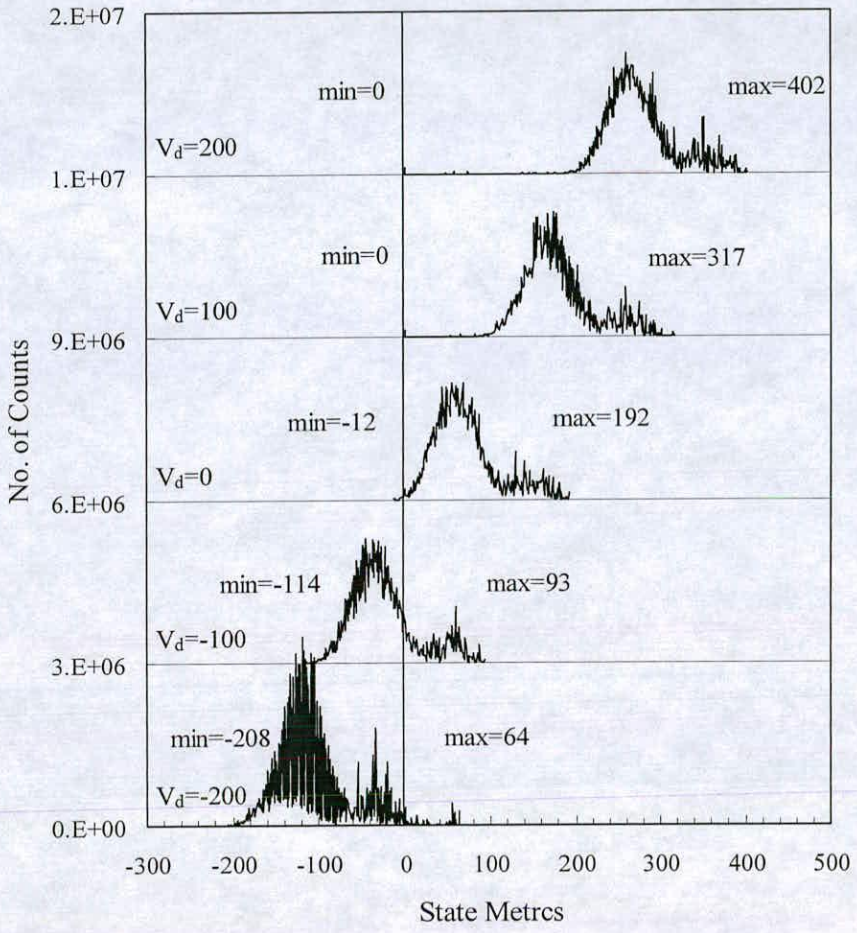



Figure 4.2 The state metric variations of the MLMAP turbo decoder with the BMN method for different $V_d$ values when the initial state metrics are set to 0.

different $V_d$ ranging from -200 to 200 with a step size of 100. The simulation results are shown in Figures 4.1 and 4.2, where horizontal and vertical axes represent the state metrics and the number of times each state metric encountered, respectively. To find the effect of the minimum state metrics, different initial minimum state metrics of -64 and 0 were used to obtain the results of Figures 4.1 and 4.2, respectively. The minimum and maximum state metrics for each $V_d$ can also be seen in these figures. The minimum and maximum state

metrics change with $V_d$ and the main distribution is shifted to the right with increases in $V_d$. The simulation results of Figure 4.1 also show that the minimum state metric values for $V_d$ = 0, 100 and 200 are the same. For practical implementations, the initial state metrics for the forward and backward processes are a finite value. Thus, the minimum state metric value, -64, for $V_d$ = 0, 100 and 200 can be used as the initial state metric, which means that if $V_d$ is larger than the initial state metric value, the minimum state metric value is the same as the initial value. The simulation results show that $V_d$ = -100 leads to a more balanced distribution of the state metrics where the minimum and maximum state metrics are -113 and 86, respectively. The results of Figure 4.2 are similar to Figure 4.1. However, due to the initial minimum state metrics of 0, the same minimum state metrics of 0 are observed when $V_d$ is 100 and 200. As in Figure 4.1, $V_d$=-100 leads to a more balanced distribution of the state metrics.

It is interesting to note that the results shown in Figures 4.1 and 4.2 coincide with the results in [15, 105], which suggested 8-bits for the state metrics if 4-bits were used for input symbols. Therefore, we can see that the BMN method drives the state metrics to the level required for the decoding process. However, in practical implementations, 9-bits are used to represent the state metrics in the ACSN unit to prevent overflows [105]. The same wordlength was used to implement ACS with BMN in this work.

## 4.3 High Speed Turbo Decoder Hardware Architecture

### 4.3.1 High Speed Turbo Soft-Input Soft-Output Decoder Architecture

The high speed MLMAP turbo SISO decoder (HSMLTBD) architecture improves the MLMAP turbo decoder (MLTBD) architecture presented in the previous chapter. The HSMLTBD architecture uses the BMN method described in the previous section to reduce the critical path delay in order to achieve high speed turbo decoder implementation for high throughput. Figure 4.3 illustrates the HSMLTBD architecture. It is configured similarly to

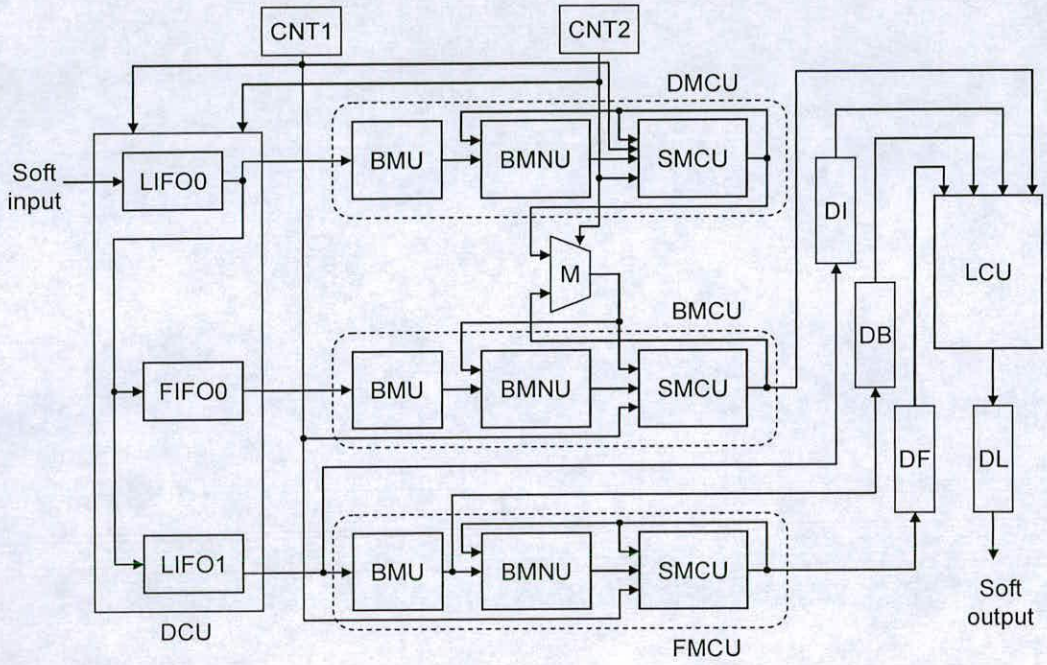



Figure 4.3 The HSMLTBD architecture incorporating the BMNU.

the MLTBD architecture. One difference is the use of a branch metric normalization unit (BMNU) between BMU and SMCU in each MCU for the forward (FMCU), backward (BMCU), and dummy-backward (DMCU) processes. In the architecture, the branch metrics generated by BMU are passed to SMCU after they are normalized by BMNU. Then, the normalized branch metrics and the state metrics are sent to SMCU, which consists of a parallel ACS without the state metric normalization process. Thus, SMCU in the HSMLTBD architecture is much more simplified than the SMCU incorporated in the MLTBD architecture. This results in a reduction of the SMCU critical path delay, which can increase the turbo decoder maximum speed.

As in the previous chapter, the other modules are implemented the same way as in the MLTBD implementation. However, due to the reduction of the critical path delay of SMCU, path delays of other modules need to be reduced as well. Thus, the modules implement more

pipelining stages. The following sections describe in detail the structure and implementation method of each module.

### 4.3.2 Metric Computation Unit with Branch Metric Normalization Unit

Figure 4.4 illustrates the structure of the metric computation unit (MCU). The MCU for the forward and backward processes consists of a BMU for computing the branch metrics, a BMNU for normalizing the branch metrics and an SMCU consisting of a parallel ACS. The soft-input data is first processed by BMU to provide the two branch metrics (BM) needed by BMNU. Then, BMNU normalizes these branch metrics based on the state metrics (SM) provided by SMCU. Finally, the four normalized branch metrics (NBM) are input to SMCU with the Recursive SM to compute the new state metrics.

The BMU incorporated in the MCU uses the BMU illustrated in Figure 3.5. On the other hand, SMCU consists of a parallel ACS without the state metric normalization process. As an example, Figure 4.5 illustrates the ACS structure for calculating the forward state metric of state 0 for constraint length K=4. The ACS performs add (A) and compare select (CS) processes to calculate the next state metrics. Two NBMs, $\eta_{00}(k)$ and $\eta_{00}(k)$, and two forward state metrics, $\alpha_0(k)$ and $\alpha_1(k)$, are the ACS inputs for computing the state metric, $\alpha_0(k+1)$.

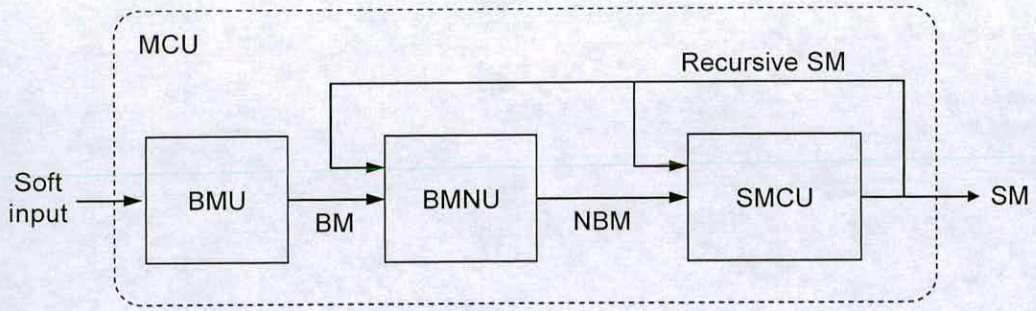



Figure 4.4 The metric computation structure.

Therefore, the ACS can generate the state metric with the same wordlength used in ACSN without the need for the state metric normalization process, while still avoiding overflow problems. The ACS critical path delay can be represented as follows:

$$d_{ACS} = 2 \times t_a + t_m \tag{4.2}$$

where $t_a$ is the delay of the adder, and $t_m$ is the MUX's delay. As can be seen, this delay is less than the delay of ACSN ($d_{ACSN}$) derived in the previous chapter (see equation (3.1)). Thus, employing the BMNU can simplify SMCU implementation while achieving high speed turbo decoder implementations.

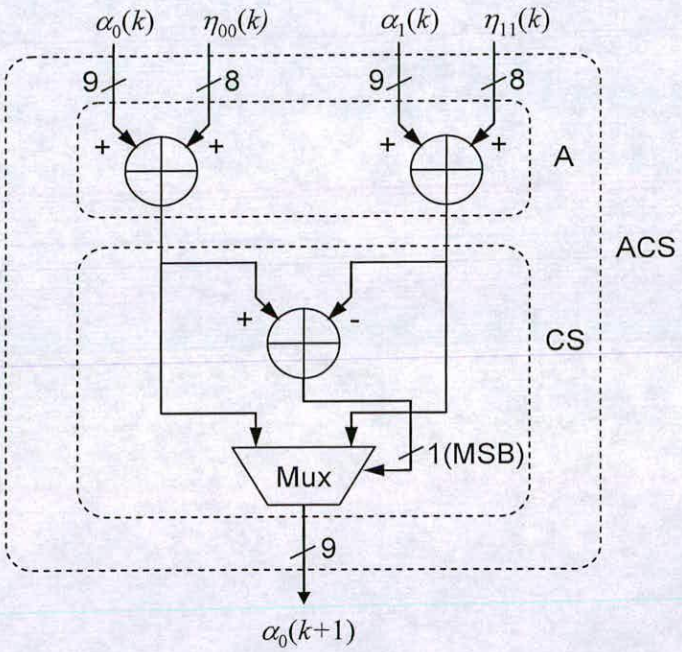



Figure 4.5 The add-compare-select structure without the state metric normalization process.

### 4.3.2.1 Branch Metric Normalization Unit

Figure 4.6 illustrates the BMNU structure employed in the HSMLTBD architecture. The BMNU inputs are the two branch metrics provided by BMU. These branch metrics are first converted into absolute values (ABS). The absolute values are compared to select the maximum of them, max[$\gamma(k)$], using the compare-select (CS) unit as used in ACS and ACSN. Then, the sign of the maximum branch metric is determined by the state metric condition (STC) input to the sign conversion unit (SCU). The sequence of STC is generated by using state metrics of ACS, and is described in next sub section how it is obtained from them.

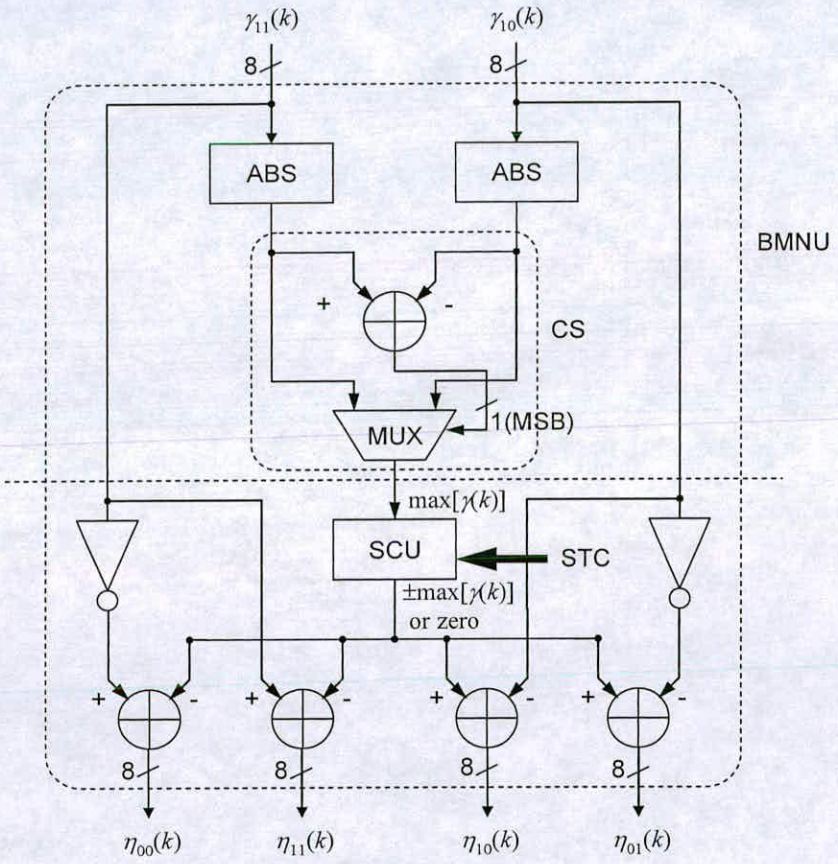



Figure 4.6 The branch metric normalization unit structure.

After determining the sign of max[$\gamma(k)$], the output of the SCU is used to normalize the branch metrics by using the four adders. The results of the four adders can be categorized as all four branch metrics are i) larger than or equal to zero, ii) less than or equal to zero, iii) the same as the original branch metrics, which means that the branch metrics did not need to be normalized.

In the BMNU, the data path delay can be represented as follows:

$$d_{BMNU} = 3 \times t_a + t_m + t_{SCU} \tag{4.3}$$

where $t_a$ is the delay of the adder, $t_m$ is the delay of the MUX; and $t_{SCU}$ is the delay of the SCU, which could be assumed to be same as $t_m$. Clearly, $d_{BMNU}$ is larger than $d_{ACS}$ delay. In this case, the turbo decoder critical path delay is in BMNU. In order to reduce the critical path delay of BMNU, it can be pipelined as shown in Figure 4.6.

When the conventional MCU and MCU with BMN methods are compared for the number of adders required to compute branch and state metrics, BMN requires fewer adders as the constraint length K increases. Table 4.1 summarizes the comparisons for constraint lengths from K=3 to 5. As can be seen, the number of adders for BMU and BMNU is not affected by the constraint length. Although the BMN method needs seven additional adders for BMNU, it uses a less complex SMCU, which leads to a decrease in the overall number of adders as K increases. In addition to the number of adders, BMNU needs two multiplexors for the normalization process.

Table 4.1 Comparison of the number of adders for conventional MCU and MCU with BMN.

| | Conventional MCU | | | MCU with BMN | | |
|---|---|---|---|---|---|---|
| | K=3 | K=4 | K=5 | K=3 | K=4 | K=5 |
| BMU | 4 | 4 | 4 | 4 | 4 | 4 |
| BMNU | - | - | - | 7 | 7 | 7 |
| SMCU | 16 | 32 | 64 | 12 | 24 | 48 |
| Total | 20 | 36 | 68 | 23 | 35 | 59 |

### 4.3.2.2 An Efficient BMNU

This section describes an efficient BMNU implementation method. A conventional method for obtaining the STC sequence may use the same state number of adders, $2^{K-1}$, as a comparator for the determination described in (4.1). However, the sequence can be generated by simple circuits without using the adders. The previous section showed that $V_d$=-100 led to well balanced state metrics. The distribution of the state metrics for $V_d$=-128 was also investigated. As already shown in Figures 4.1 and 4.2, although the distribution range for $V_d$=-128 was slightly shifted to the left, the wordlength of 9-bits was sufficient to represent the state metrics.

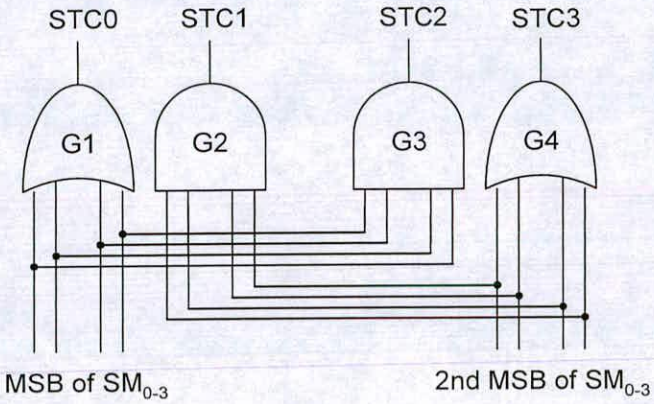



Figure 4.7 Circuit for generating STC signals for BMNU with K=3.

Figure 4.7 illustrates a simple circuits for generating four STC sequences for constraint length K=3. The inputs of the AND and OR gates are MSB and the second MSB of the state metrics generated by SMCU illustrated in Figure 4.4. Using these STC signals in the figure, the BMN rule shown in (4.1) can be written as follows :

$$
\begin{aligned}
&\text{For } n = 0 \ldots N \\
&\text{if STC0 = '0' or \{STC1, STC2\} = '11'} \\
&\quad \eta_n(k) = \gamma_n(k) - \max[\gamma_0(k), \ldots, \gamma_n(k)] \\
&\text{else if \{STC2,STC3\} = '10'} \\
&\quad \eta_n(k) = \gamma_n(k) - \min[\gamma_0(k), \ldots, \gamma_n(k)]
\end{aligned}
\tag{4.4}
$$

```
else
```

$$\eta_n(k) = \gamma_n(k)$$

Here, the first and second conditions imply that all state metrics are larger or less than $V_d$=-128. If these conditions are not satisfied, the branch metrics are passed to the ACS units without the normalization process.

### 4.3.3 Log-Likelihood Ratio Computation unit

The basic LCU structure used in the MLMAP turbo decoder architecture is not affected by employing the BMNU. However, due to the reduced ACS critical path delay, the LCU might be implemented with more pipelined stages than were used before in the MLMAP turbo

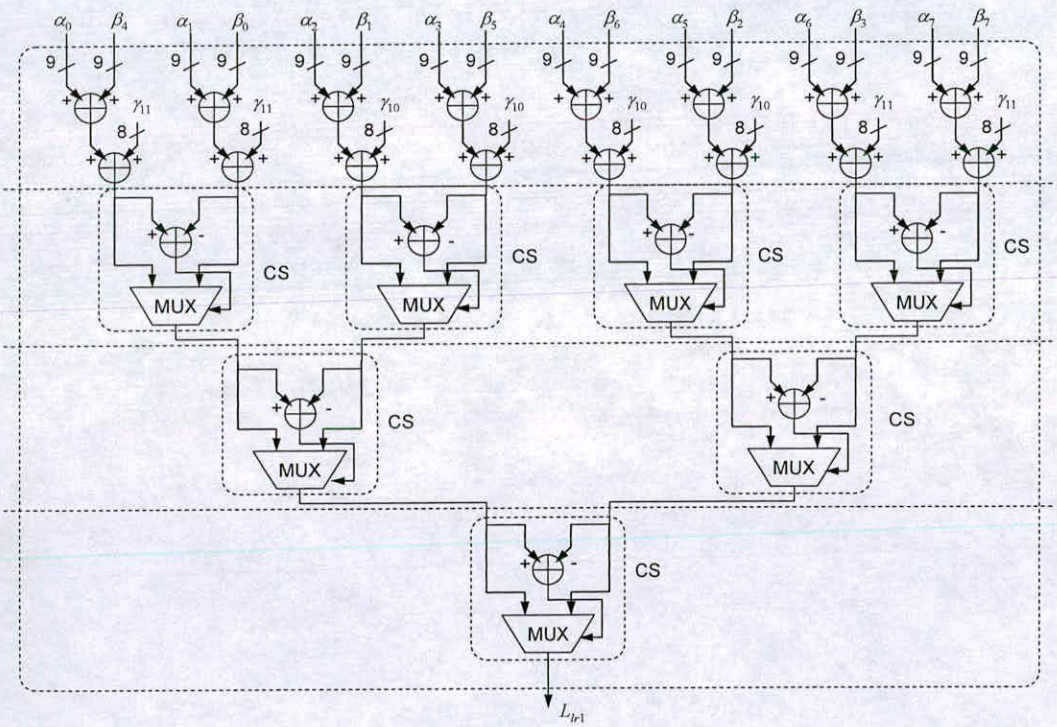



Figure 4.8 The structure of LLR computation unit with high level of pipelining.

decoder architecture to avoid a path delay longer than the critical path delay. This can be realised by inserting pipeline registers. Figure 4.8 illustrates an LCU structure for calculating $L_{lr1}$ for constraint length K=4. As can be seen, the LCU structure is implemented in 3 pipelined stages to reduce its delay to less than the ACS delay. The additional pipelining stages could increase area and power due to the inserted registers. To address this problem, the next section describes a more efficient LCU implementation method.

## 4.4 Low Power and Area Efficient Max-Log MAP Turbo Decoder Hardware Architecture

### 4.4.1 Low Power and Efficient Max-Log MAP Turbo Soft-Input Soft-Output Decoder Architecture

This section presents a low power and area efficient MLTBD (LAMLTBD) architecture incorporating the BMNU. The previous chapter showed that the memory blocks had the largest area in the MLTBD implementations. The LAMLTBD architecture improves the area usage and power consumption by reducing the size of the memory required to control the input data for the SW method and to delay the soft-input data passed to the LCU. Moreover, the LCU is more efficiently implemented by exploiting retiming and a four-input compare select (4-CS) unit to reduce the data path and hardware resources.

Figure 4.9 illustrates the LAMLTBD architecture. As the figure shows, the data schedule unit (DSU) of the MLTBD illustrated in the previous chapter is simplified by using only a single FIFO block. The FIFO block is used to delay the soft-input data before fed into BMCU, and its depth is the same as the window size, 40. To eliminate the LIFO blocks used in the DSU of the MLTBD, a triple read and single write memory (TRMEM) block is employed to store soft-input data. Once the window size is determined, the data addresses for SW method can be decided for performing the forward, backward, and dummy-backward processes. Thus, following the SW method with the determined addresses, the three inputs, "Soft-input 0, 1 and 2," corresponding to each of TRMEM read ports can be controlled to pass them to the LAMLTBD architecture. The soft-input data flow is illustrated in Figure

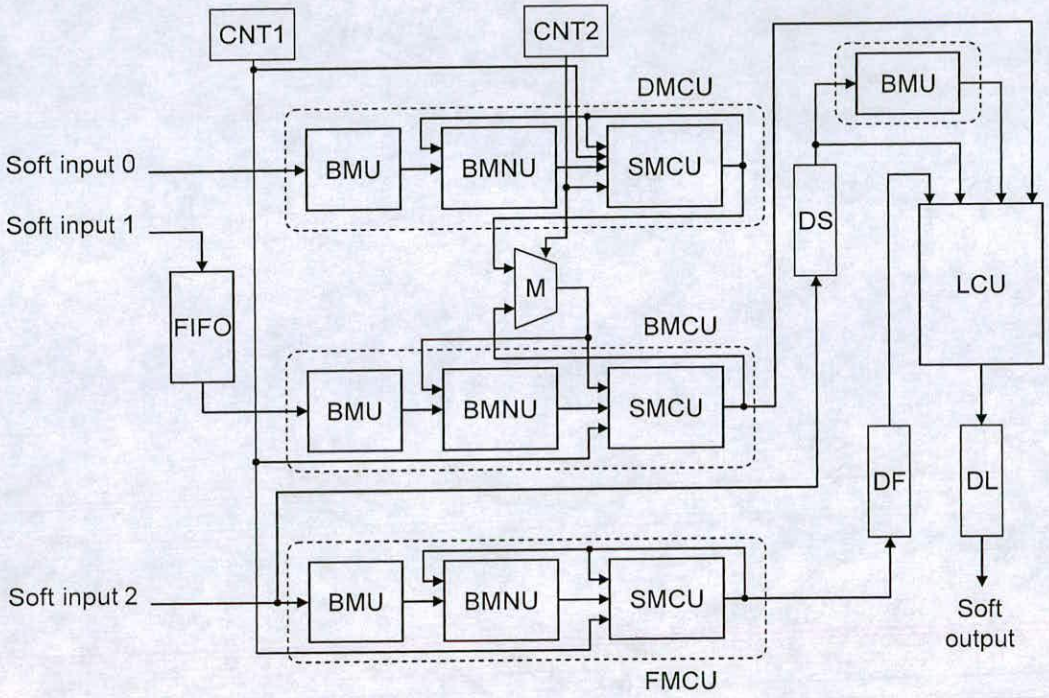



Figure 4.9 The LAMLTBD architecture with the reduced memory.

4.10, sub-block by sub-block. While the reverse ordered "Sub-block 2" data are passed to DMCU as "Soft-input 0" to calculate the dummy backward state metrics, the reverse and right ordered "Sub-block 1" data are input to the FIFO for delaying and the FMCU for computing the forward state metrics, respectively. The reverse ordered "Sub-block 1" data are delayed in the FIFO block until the completion of the dummy backward state metric computation process for the reverse ordered "Sub-block 2." When the dummy backward state metric computation is completed, the backward state metrics and the delayed forward state metrics in DF are passed to LCU to compute the LLR value, as illustrated in Figure 4.3. The sub-block data addressing can be controlled by TRMEM without significant overhead in area and power.

For further area savings, the memory for delaying the branch metrics passed to LCU is replaced by an additional BMU. The DB in Figure 4.3 is used to delay the branch metrics

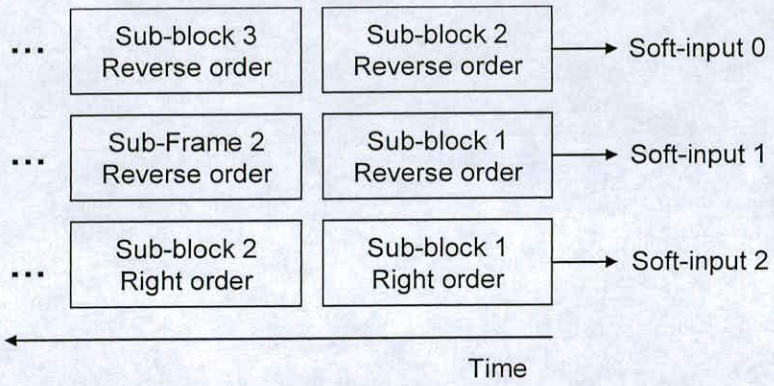



Figure 4.10 The soft-input data stream input to the LAMLTBD architecture.

that are required by LCU to calculate LLR. The LAMLTBD architecture eliminates the DB. Instead, the branch metrics for computing LLR are generated by an additional BMU (placed before LCU in Figure 4.3) from the soft-input data delayed by the DS. The soft-input data is also used for obtaining the extrinsic information by subtracting it from LLR. The added BMU is the same as other BMUs used in MCU. Thus, the replacement of DL with an additional BMU contributes to further area savings.

### 4.4.2 An Efficient LCU implementation

The previous section showed that the BMN method can reduce the turbo decoder critical path delay. However, this reduction may lead to high level of pipelining in LCU implementations for making the LCU critical path delay shorter than the ACS critical path delay. In general, high level of pipelining increases area and power due to an excessive use of the registers incorporated. To address this problem, this section describes an implementation method for reducing the number of pipelining stages in LCU for an efficient implementation in terms of area and power.

Figure 4.11 illustrates the new LCU structure for constraint length K=4, whose output sequence is the same as the LCU output illustrated in Figure 4.8. The new LCU is implemented with retiming method and a four-input compare select unit (CS). In the LCU

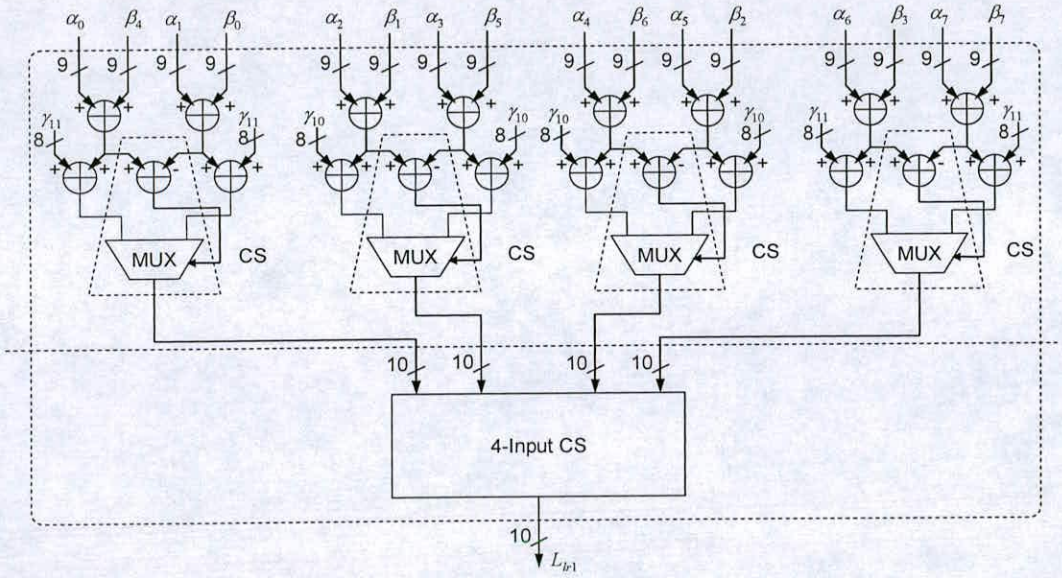



Figure 4.11 The efficient LLR computation unit structure.

structure shown in Figure 4.8, the first CS process is performed after completing the additions of two state and one branch metrics. However, the new LCU performs the first CS after finishing the addition of the forward and backward state metrics. The first CS and the addition of the branch metric to the results of the addition of the two state metrics are executed simultaneously. This is possible because the same branch metrics are added to the first and second state metrics' addition results, as shown in Figure 4.10. Thus, in the new LCU, the first CS compares two state metric addition results, and then selects one of them after adding the branch metric during the comparison process. The four selected values are then input to a four-input CS to determine the maximum among them which corresponds to $L_{lr1}$.

Figure 4.12 illustrates the four-input CS structure that consists of six adders for the comparison process, and a unit for selecting the maximum input. It is designed to minimise its critical path delay. As can be seen, the 4-CS delay is given by the delay of an adder and the four-input selector. The four-input selector is processed as follows :

```
If {s0, s1, s2} = 3'b000                    (4.5)
    Llr1 = A
else if {s0, s1, s2} = 3'100
    Llr1 = B
else if {s0, s3, s4} = 3'110
    Llr1 = C
else
    Llr1 = D
```

While the LCU in Figure 4.8 needs four pipelined stages, the new LCU is realized with only two pipelined stages. However, the number of adders in the new LCU is increased by employing the 4-CS. The next section shows area and power comparison results of the LCU implementations.

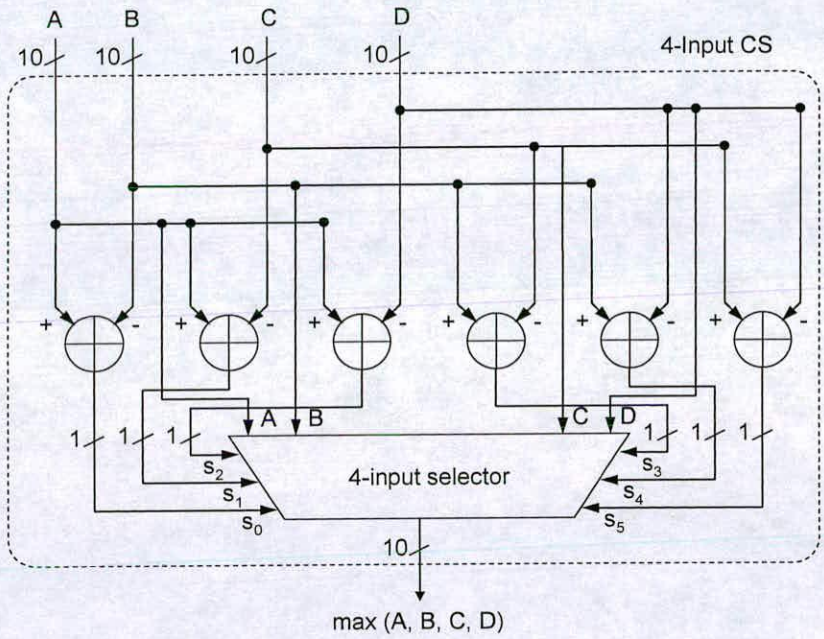



Figure 4.12 The four-input compare select unit structure.

## 4.5 Results

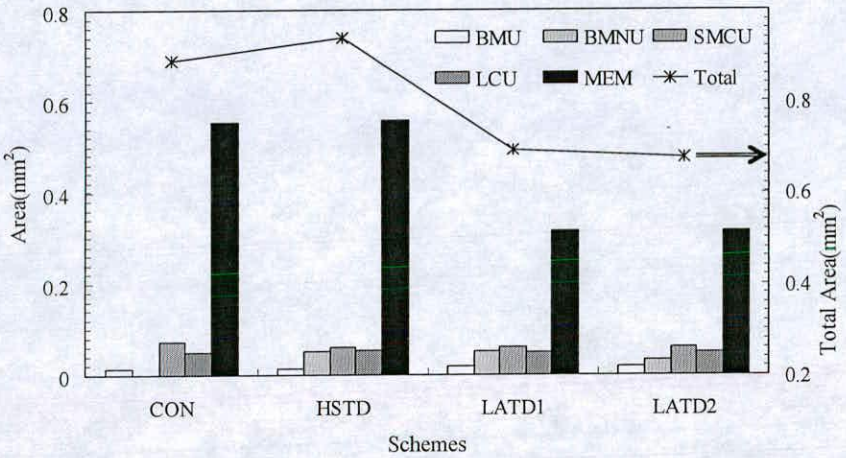
The HSMLTBD and LAMLTBD architectures were designed at RTL using Verilog HDL. These designs were verified by the same test systems used in the previous chapter. After completing the hardware verification, they were synthesized using the 0.18μm UMC standard CMOS library with Synopsys DesignCompiler™. The netlists were simulated using Cadence Verilog-XL™ with the timing constraints obtained from the synthesis to generate the switching activities of all the circuit nets. Then, the power consumption of both architectures was evaluated using Synopsys PowerCompiler™. A clock frequency of 50MHz was used in the hardware performance evaluation process.

Table 4.2 List of schemes for hardware performance evaluation.

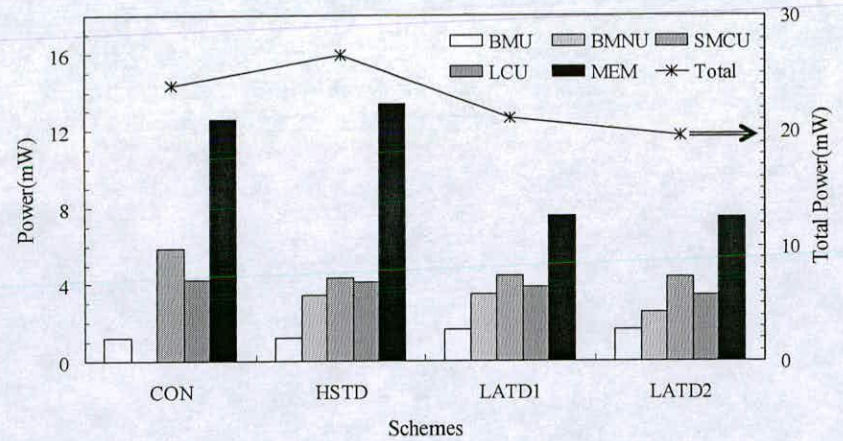
| List of Schemes | Descriptions |
|---|---|
| CON | Conventional |
| HSTD | HSMLTBD |
| LATD1 | LAMLTBD with efficient LCU |
| LATD2 | LAMLTBD with efficient LCU and BMNU |

After investigating various turbo decoder schemes in order to decide on an optimized configuration for the turbo decoder hardware implementation, three different schemes for constraint lengths from K=3 to K=5 have been implemented to evaluate their hardware performance in terms of area, power and critical path delay for maximum speed. These schemes are listed in Table 4.2. CON, called conventional, is the MLTBD implemented in the previous chapter. HSTD is the HSMLTBD illustrated in Figure 4.3 for high speed turbo decoder implementations. LATD1 and LATD2 are based on the LAMLTBD architecture illustrated in Figure 4.9 for low power and area efficient turbo decoder implementations. The difference between LATD1 and LATD2 is in their BMNU implementations. The circuit illustrated in Figure 4.7 for BMNU has been incorporated in LATD2, while LATD1 has been implemented with a conventional BMNU shown in Figure 4.6.

Figures 4.13-15 and Tables 4.3-8 show the hardware performance evaluation results for K=3, 4 and 5, respectively. The overall area and power of HSTD was increased by around 7%, 7% and 5% and 10%, 8% and 0% for each constraint length, respectively, when compared to CON. On the other hand, LATD1 and LATD2 can save up to 30% in area and 19% in power as compared to CON. These results show that the BMNU leads to an increase in hardware costs in HSTD. Despite including the BMNU, eliminating the memory blocks for the SW




(a)




(b)

Figure 4.13 (a) Area and (b) power comparisons for K=3.

method and the efficient LCU and BMNU implementations can save the area and power of LATD1 and LATD2.

Figures 4.13 (a) and (b) illustrate the area and power results for K=3, respectively. In these figures, the primary and secondary Y-axes indicate the results of each component and total area or power, respectively. These results are summarized in Tables 4.3 and 4.4. The area

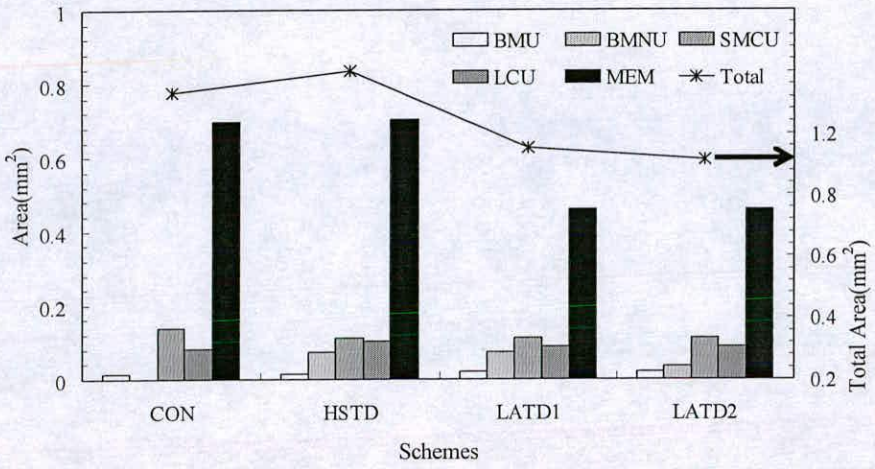
Table 4.3 List of area results for K=3 turbo decoders.

| | K=3 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | CON | | HSTD | | LATD1 | | LATD2 | |
| | Area ($mm^2$) | Save (%) | Area ($mm^2$) | Save (%) | Area ($mm^2$) | Save (%) | Area ($mm^2$) | Save (%) |
| BMU | 0.014 | - | 0.014 | 0 | 0.019 | -33.13 | 0.019 | -33.13 |
| BMNU | - | - | 0.051 | - | 0.051 | - | 0.033 | - |
| SMCU | 0.071 | - | 0.060 | 15.96 | 0.060 | 15.96 | 0.060 | 15.96 |
| LCU | 0.049 | - | 0.053 | -8.88 | 0.048 | 1.70 | 0.048 | 1.70 |
| MEM | 0.553 | - | 0.558 | -0.94 | 0.314 | 43.18 | 0.314 | 43.18 |
| Total | 0.689 | - | 0.738 | -7.14 | 0.493 | 28.33 | 0.475 | 30.95 |

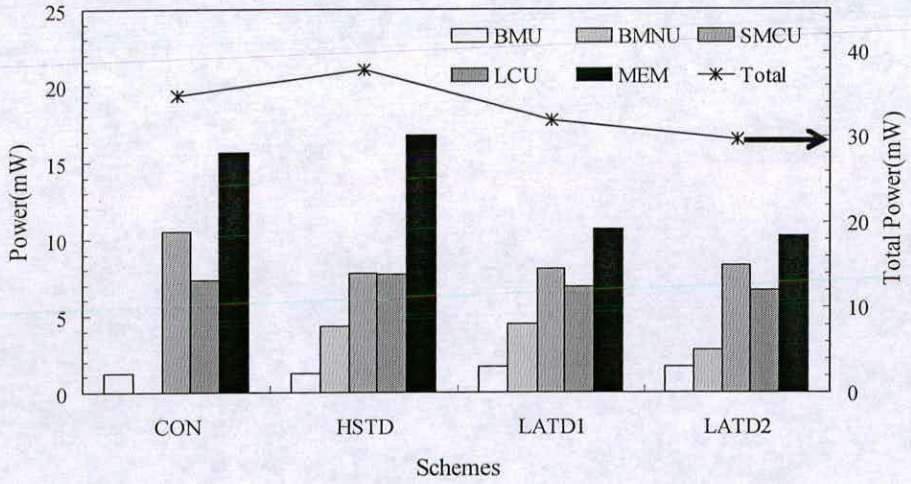
Table 4.4 List of power results for K=3 turbo decoders.

| | K=3 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | CON | | HSTD | | LATD1 | | LATD2 | |
| | Power (mW) | Save (%) | Power (mW) | Save (%) | Power (mW) | Save (%) | Power (mW) | Save (%) |
| BMU | 1.21 | - | 1.22 | -0.66 | 1.66 | -37.48 | 1.66 | -37.48 |
| BMNU | - | - | 3.44 | - | 3.48 | - | 2.56 | - |
| SMCU | 5.9 | - | 4.33 | 26.59 | 4.46 | 24.27 | 4.41 | 25.18 |
| LCU | 4.32 | - | 4.10 | 2.97 | 3.86 | 8.76 | 3.45 | 18.51 |
| MEM | 12.51 | - | 13.42 | -6.45 | 7.59 | 39.79 | 7.54 | 40.18 |
| Total | 23.95 | - | 26.52 | -10.74 | 21.07 | 12.01 | 19.63 | 18.04 |

and power increase of BMN in LATD1 and LATD2 is due to the additional BMU needed to replace the memory blocks for generating the branch metrics used to compute LLR in LCU. Clearly, the SMCU of HSTD, LATD1 and LATD2 is reduced by up to 16% in area and 26% in power by eliminating the state metric normalization process. The efficient LCU incorporated in LATD1 and LATD2 saves area and power. A significant change in area and




(a)




(b)

Figure 4.14 (a) Area and (b) power comparisons for K=4.

power is indeed seen in the MEM of LATD1 and LATD2. The reduced memory needed to control data input following the SW method saves the total area and power dominantly.

The area and power results for K=4 are illustrated in Figures 4.14 (a) and (b), and are summarized in Tables 4.5 and 4.6. The BMU results are almost the same as the results of K=3. In the results, the LCU of HSTD is increased by 23% in area, which is due to the

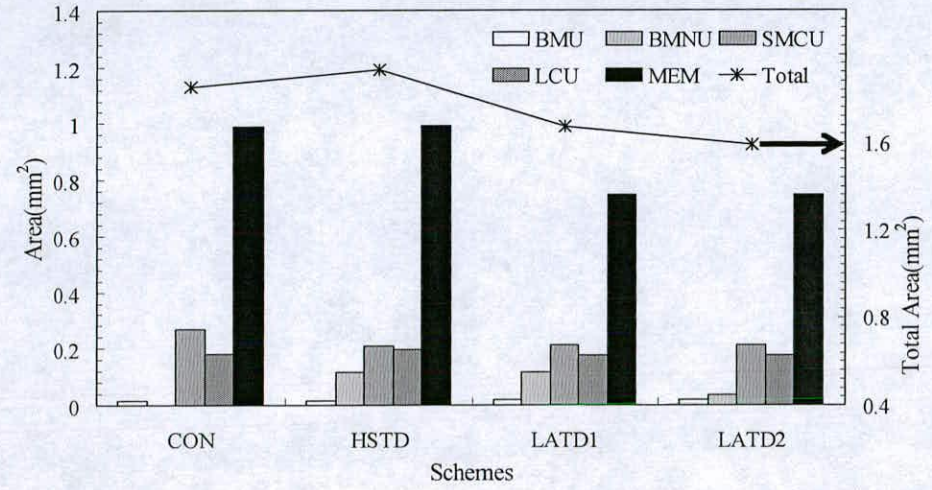
Table 4.5 List of area results for K=4 turbo decoders.

| | K=4 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | CON | | HSTD | | LATD1 | | LATD2 | |
| | Area ($mm^2$) | Save (%) | Area ($mm^2$) | Save (%) | Area ($mm^2$) | Save (%) | Area ($mm^2$) | Save (%) |
| BMU | 0.014 | - | 0.014 | 0 | 0.019 | -33.13 | 0.019 | -33.13 |
| BMNU | - | - | 0.072 | - | 0.072 | - | 0.035 | - |
| SMCU | 0.137 | - | 0.110 | 19.56 | 0.110 | 19.52 | 0.110 | 19.52 |
| LCU | 0.083 | - | 0.102 | -23.03 | 0.089 | -7.03 | 0.089 | -7.03 |
| MEM | 0.698 | - | 0.703 | -0.74 | 0.459 | 34.22 | 0.459 | 34.22 |
| Total | 0.934 | - | 1.004 | -7.51 | 0.751 | 19.52 | 0.714 | 23.47 |

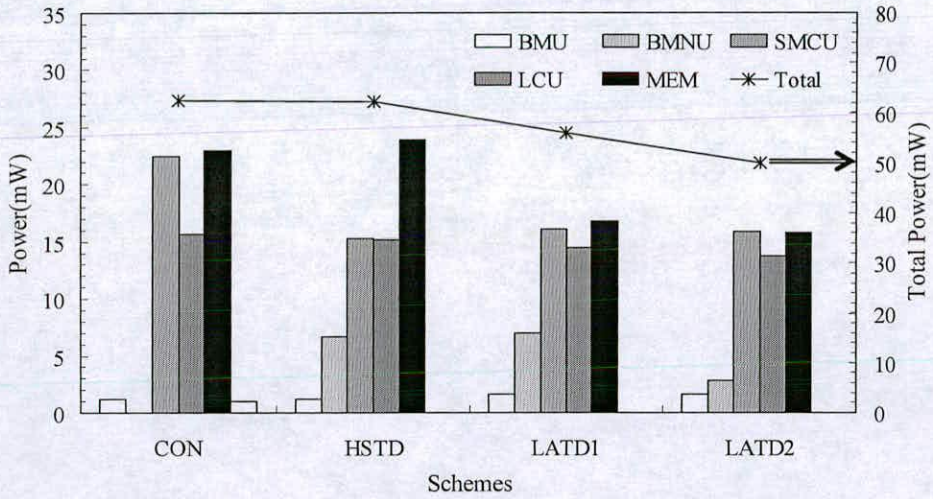
Table 4.6 List of power results for K=4 turbo decoders.

| | K=4 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | CON | | HSTD | | LATD1 | | LATD2 | |
| | Power (mW) | Save (%) | Power (mW) | Save (%) | Power (mW) | Save (%) | Power (mW) | Save (%) |
| BMU | 1.21 | - | 1.21 | 0.08 | 1.66 | -36.78 | 1.66 | -36.61 |
| BMNU | - | - | 4.36 | - | 4.51 | - | 2.77 | - |
| SMCU | 10.54 | - | 7.77 | 26.27 | 8.1 | 23.17 | 8.30 | 21.26 |
| LCU | 7.35 | - | 7.74 | -5.29 | 6.94 | 5.58 | 6.72 | 8.52 |
| MEM | 15.66 | - | 16.73 | -6.79 | 10.63 | 32.14 | 10.22 | 34.76 |
| Total | 34.78 | - | 37.83 | -8.76 | 31.85 | 8.42 | 29.68 | 14.64 |

highly pipelined implementation, compared to CON. On the other hand, the efficient LCU area of LATD1 and LATD2 is slightly increased compared to CON. Moreover, the efficient LCU can save 8% in power despite the area increase. Similar to the K=3 results, it is clear




(a)




(b)

Figure 4.15 (a) Area and (b) power comparisons for K=5.

from the results in Tables 4.5 and 4.6 that area and power savings of MEM dominantly contribute to reduce the total area and power.

Finally, Figures 4.15 (a) and (b) and Tables 4.7 and 4.8 illustrate and summarize the area and power results of K=5 turbo decoder hardware implementations. Inserting BMNU saves 21% in area and 31% in power of the SMCU of HSTD, LATD1 and LATD2. Moreover, the

Table 4.7 List of area results for K=5 turbo decoders.

| | K=5 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | CON | | HSTD | | LATD1 | | LATD2 | |
| | Area ($mm^2$) | Save (%) | Area ($mm^2$) | Save (%) | Area ($mm^2$) | Save (%) | Area ($mm^2$) | Save (%) |
| BMU | 0.014 | - | 0.014 | 0 | 0.019 | -33.1 | 0.019 | -33.1 |
| BMNU | - | - | 0.115 | - | 0.115 | - | 0.035 | - |
| SMCU | 0.268 | - | 0.211 | 21.3 | 0.211 | 21.3 | 0.211 | 21.3 |
| LCU | 0.180 | - | 0.197 | -9.6 | 0.176 | 2.0 | 0.176 | 2.0 |
| MEM | 0.988 | - | 0.993 | -0.5 | 0.749 | 24.1 | 0.749 | 24.1 |
| Total | 1.452 | - | 1.532 | -5.5 | 1.272 | 12.3 | 1.192 | 17.9 |

Table 4.8 List of power results for K=5 turbo decoders.

| | K=5 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | CON | | HSTD | | LATD1 | | LATD2 | |
| | Power (mW) | Save (%) | Power (mW) | Save (%) | Power (mW) | Save (%) | Power (mW) | Save (%) |
| BMU | 1.20 | - | 1.20 | -0.3 | 1.65 | -37.17 | 1.65 | -37.0 |
| BMNU | - | - | 6.65 | - | 6.93 | - | 2.83 | - |
| SMCU | 22.46 | - | 15.32 | 31.7 | 16.09 | 28.34 | 15.91 | 29.1 |
| LCU | 15.65 | - | 15.19 | 2.9 | 14.45 | 7.66 | 13.78 | 11.9 |
| MEM | 22.99 | - | 23.92 | -4.0 | 16.77 | 27.07 | 15.76 | 31.4 |
| Total | 62.32 | - | 62.30 | 0.0 | 55.91 | 10.28 | 49.94 | 19.8 |

efficient BMNU implementation of LATD2 saves 69% and 59% in area and power, respectively, compared to LATD1. Also, the efficient LCU of LATD1 and LATD2 reduces 2% and 11% in area and power, respectively. With the area and power savings of MEM, LATD2 has achieved 17% and 19% savings in total area and power, respectively.

One of the crucial results in this chapter is the critical path delay, which is compared with the results in Table 3.3 in Chapter 3. The proposed BMNU addresses the inherent critical path delay problem to achieve the high speed turbo decoder implementation. Thus, the critical path delay of each scheme was measured. Table 4.9 lists the results comparing with the conventional architecture (CON) obtained in the previous chapter and state-of-art turbo decoders. In the table, total number of gates is equivalent to the number of NAND gates. As is apparent, HSTD reduces the critical path delay for K=3, 4 and 5 by 32%, 42% and 41%, respectively, compared with the delays of CON. This leads to the increase of the maximum speed of HSTD and LATD2. All the delays measured are in SMCU of the architectures. Thus, HSTD can be used to provide higher throughput than the conventional architecture. Also, the critical path delay results of LATD1 and LATD2 are the same with HSTD.

Table 4.9 Performance comparison with state-of-art implementations.

| | CON | HSTD | LATD2 | [11] | [13] | [12] |
|---|---|---|---|---|---|---|
| Area($mm^2$) | 0.934 | 1.004 | 0.714 | 1.5 | 14.5 | 10 |
| No. of gates | 59K | 80K | 124K | 150K | 410K | - |
| Power(mW) | 34 | 37 | 29 | 330† | 1450† | 2464† |
| Max. Speed | 180MHz | 300MHz | 300MHz | 285MHz | 145MHz | 352MHz |
| Throughput (Mb/s) | 22.5 | 37 | 37 | - | 24 | 352 |
| Technology | 0.18μm | 0.18μm | 0.18μm | 0.18μm | 0.18μm | 0.13μm |
| Note | K=4 | K=4 | K=4 | K=3 | K=4 Radix-4 | K=5 |

† Power consumption at the maximum clock speed.

## 4.6 Summary

This chapter presented a high performance MLTBD architecture based on a novel scheme in order to save area and power, and to improve the critical path delay. To achieve low power and area efficient MLTBD implementations, the memory blocks needed for the SW method were reduced by using a triple read port based memory. This helped to reduce both total power consumption and area usage. The triple read port memory was designed in this thesis for the hardware simulation using without synthesizing. Also, more efficient LCU and BMNU implementations contributed to save power and area. The BMN method addressed the inherent critical path delay problem to achieve high speed turbo decoders for high throughput. The next two chapters present an alternative turbo decoder algorithm, which is called soft-output Viterbi algorithm (SOVA), and architecture, and introduce a novel architecture for the turbo decoder implementations.

# Chapter 5

# Soft-Output Viterbi Algorithm Based Turbo Decoding Process

## 5.1 Introduction

In previous chapters, the MAP based turbo decoders have been investigated at algorithmic and architectural levels. This chapter presents an alternative turbo decoding algorithm, called a soft-output Viterbi algorithm (SOVA), which was introduced by Hagenauer and Hoeher [67]. SOVA was suggested before turbo codes appeared, which were developed to provide outer-decoder soft-output (also known as soft-decision or LLR) in order to achieve better performance. With increased understanding of the principle of the iterative turbo decoding process, it was found that the extrinsic information for the iterative decoding process could be obtained from the soft-output of SOVA in a way that used a MAP based turbo decoding algorithm [59, 150–151]. Since then, there were some attempts to implement a turbo decoder based on SOVA. However, the decoding complexity of the SOVA was very high, making it difficult to apply to practical applications. To address the complexity problem, a two-step SOVA (TSOVA) was suggested, which divided the decoding process into the survivor and update processes [93, 152]. Since the introduction of TSOVA, the TSOVA based turbo decoders have been widely researched for achieving better performance [33-34, 98, 152-161].

This chapter describes SOVA and TSOVA based turbo decoders, investigating their BER performance for different constraint lengths, K=3, 4, and 5. In a TSOVA based turbo decoder, the survivor and update depths strongly affect BER performance. They also impact the hardware costs required to design the TSOVA based turbo decoder hardware in an optimised implementation. In the literature [98, 153-156], different depths of the survivor and update processes are suggested for TSOVA turbo decoder implementations. Thus, this chapter discusses BER performance relative to varying TSOVA survivor and update depths,

and then, two different depths are chosen for evaluating overall BER performance with TSOVA turbo decoders.

The chapter is organised as follows. Section 5.2 describes the original SOVA and how it is used in turbo decoder implementations. A two-step SOVA that reduces SOVA complexity is presented in Section 5.3. Section 5.4 shows the simulation results of the two-step SOVA-based turbo decoders, giving BER performance with optimised survivor and update depths. Section 5.5 summarizes the chapter.

## 5.2 Soft-Output Viterbi Algorithm for Turbo Decoding Process

The basic decoding principle of SOVA is not different than the conventional Viterbi decoding process [84, 139]. While a Viterbi decoder provides a hard-decision in a bit sequence, a SOVA-based decoder outputs a soft-decision including the hard-decision and its reliability value. The SOVA decoder output is exploited to obtain the extrinsic information needed in the turbo decoding process. While the LLR value of MAP-based turbo decoders is calculated from all the metrics generated during the forward and backward processes, the soft-output of the SOVA-based turbo decoders is obtained by multiplying the hard-decision of the survivor path with the reliability value of the LLR. Both of these values (hard-decision and reliability value) are generated independently during the decoding process. The extrinsic information is passed to the next decoding process in an iterative process and can be obtained from the soft-output after subtracting the systematic symbol and the extrinsic information from it as described in Section 2.6 of Chapter 2. Thus, this allows SOVA to perform the turbo decoding process.

According to [67], SOVA generates a soft-output rather than a hard-decision, finding the maximum likelihood (ML) paths in the forward process. The soft-output is represented by *a posterior* probability (APP) as LLR as shown below.

$$L_{lr}(k) = \log \frac{P(u(k) = + \mid y)}{1 - P(u(k) = + \mid y)} = u(k)L(k) \tag{5.1}$$

where $u(k)$ is the hard-decision and $L(k)$ is the reliability value of the hard-decision at time $k$. In MAP based turbo decoders, $L_{lr}$(k) is calculated from the branch metrics, and all forward and backward state metrics. However, $u(k)$ and $L(k)$ in equation (5.1) are independently obtained from only the forward process, as in a Viterbi decoder.

The turbo decoder structure illustrated in Figure 2.7 in Chapter 2 can also be used for the SOVA-based turbo decoding process. In order to describe the obtaining of $L_{lr}(k)$ for SOVA decoders, let's note that $y_s(k)$ and $y_p(k)$ are input symbols of the systematic and parity bits at time $k$, respectively. Then, the branch metric, $bm(k)$, is

$$bm(k) = \frac{1}{2}\left(L_c y_s(k) u_s + L_c y_p(k) u_p + L_e(k) u_s\right) \tag{5.2}$$

where $L_c$ is the channel reliability value, but is fixed to 2 in this work, $u_s$ and $u_p$ are associated with the code word of the systematic and the parity bits, respectively, and $L_e(k)$ is the extrinsic information, which is provided by previous decoding process and initialised at zero for the first decoding process.

The state metric computation process is illustrated using a trellis diagram in Figure 5.1. The trellis diagram shows the forward process, representing the transition paths in different line styles with code words. As an example in Figure 5.1, the transition paths for obtaining the state metrics of state 0 at time $k$ and $k$+1 are represented with a bold-style line. In the figure, $bm_{00}(k)$ and $sm_0(k)$ are the branch metric of code word 00 and the state metric of state 0 at time $k$, respectively. The difference of two paths converged to one state is the reliability value, $\Delta(k)$, at time $k$. Specifically, with the branch metrics computed by the equation (5.2), a state metric of each state, $sm$, can be represented as follows:

$$sm_s(k+1) = \max[\, bm(s_k, s_{k+1}) + sm_s(k), bm(s_k', s_{k+1}) + sm_{s'}(k)] \tag{5.3}$$

where $sm_s(k)$ is the state metric at the state $s$ and the time $k$, $bm(s_k, s_{k+1})$ is the branch metric of the state transition $s_k$ to $s_{k+1}$. The ML path corresponding to the decision bit at each state is determined from the maximum state metric obtained by comparing the two state metrics. Then, the absolute value of the two state metric differences, $\Delta$, is used as the reliability value of the soft-output as follows:

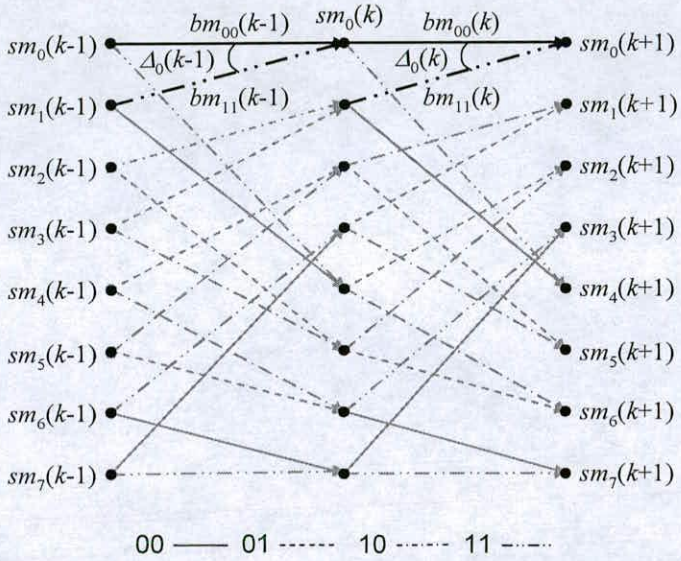



Figure 5.1 A trellis diagram for computing the state metrics for constraint length K=4.

$$\Delta(k+1) = \frac{1}{2}\left|bm(s_k, s_{k+1}) + sm_s(k) - bm(s_k', s_{k+1}) - sm_{s'}(k)\right| \tag{5.4}$$

These values are accumulated and updated at each state through the survivor and update processes until deciding the final reliability value for the soft-output. In these processes, the update rule is:

For $j$=0 to U (5.5)
if $u_s(j) \neq u_c(j)$ then $L(j) = \min(L(j), \Delta_s(k))$
else $L(j) = L(j)$.

where U is the depth of update process, $u_s(j)$ and $u_c(j)$ are the hard-decisions of the survivor and competing paths, $L(j)$ is the reliability value of LLR, and $\Delta_s(k)$ is the reliability value for state $s$ at time $k$. Therefore, LLR is finally represented by the multiplication of $u_s(j)$ and $L(j)$.

## 5.3 Two-step Soft-Output Viterbi algorithm

As already stated, the two-step SOVA (TSOVA) suggested in the literature [93] was developed to reduce the computational complexity of the original SOVA. TSOVA divides the decoding process into the survivor and update processes, which are executed simultaneously in the original SOVA. The survivor process performs the same function as the traceback process in a Viterbi decoder, that is, to find an appropriate state for starting the decoding process to generate the hard-decision. Figure 5.2 illustrates a trellis state diagram for the TSOVA decoding process for constraint length K=4 as an example. The survivor process finds the ML paths from time $k$ to $k$-D (D is the depth of the survivor process) to determine the state at which all ML paths are converged. The update process updates the reliability values starting from the converged state by following the hard-decisions provided by the survivor and competing paths. In Figure 5.2, the two paths are shown with different line styles corresponding to the hard-decisions, 0 and 1. The reliability values are updated where the two paths are represented in different line styles, which mean that the hard-decisions of the survivor and competing paths are different. The update rule given in equation (5.5) is used for the update process.

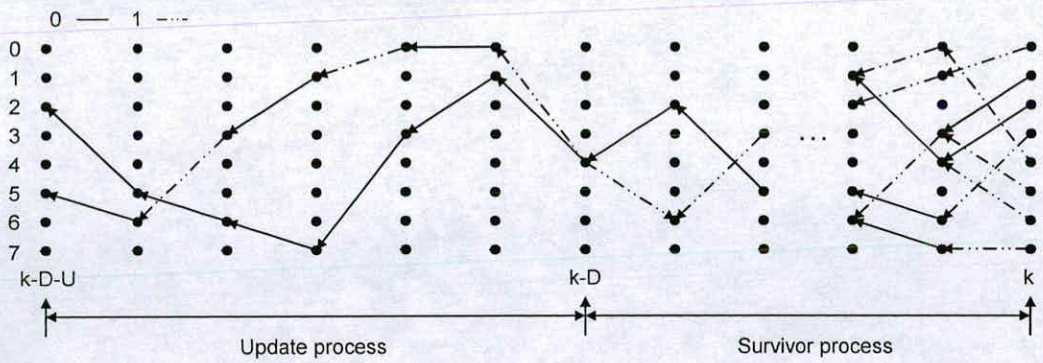



Figure 5.2 A trellis diagram of the survivor and update processes in two-step SOVA for constraint length K=4.

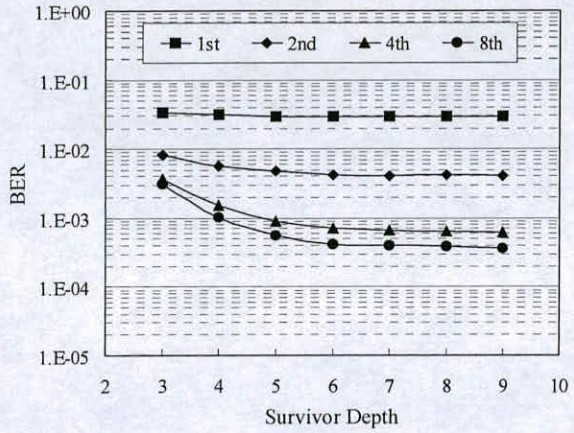
## 5.4 Results

TSOVA turbo decoder BER performance has been evaluated for constraint lengths K=3, 4, and 5. Turbo codes for performance evaluation were generated by the encoders illustrated in Figures 2.1 and 2.2 of Chapter 2. In this simulation, additive white Gaussian channel (AWGN), binary phase shift keying (BPSK) modulation, 1/3 code rate, and 1024-size block interleaver are assumed. The received data is represented in a fixed-point format with 4-bits total and 2-bits precision as described in Chapter 2 for evaluating the MAP based turbo decoder performance.

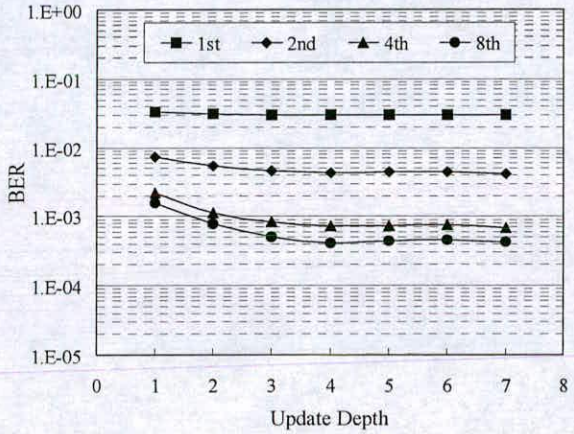
At first, BER performance was investigated varying the survivor and update depths of the TSOVA turbo decoders, to determine the optimised depths for each constraint length. Then, the BER performance of the turbo decoders was evaluated with the optimal survivor and update depths. Optimizing the depths is crucial not only to achieve better performance, but also to save area usage and power consumption in hardware implementation. The depths determined in this chapter will be applied to the TSOVA hardware implementations in the next chapter.

### 5.4.1 Determination of Survivor and Update Depths

Many papers [152-156] have discussed the survivor and update depths and how they affect TSOVA decoder performance. As already described in the previous section, the decoding process of TSOVA is divided into the survivor and update processes. The survivor process is the same as the traceback process in a Viterbi decoder, and is performed before the decoding process to generate a hard-decision. Thus, the depth of the survivor process primarily affects BER performance. It is well known that the traceback length in Viterbi decoder implementations must be at least 5 times of the constraint length, K. If the length is large enough, there is no performance degradation. However, a proper depth of the survivor process needs to be determined in order to achieve BER performance with optimal hardware performance. The same rule used to determine the traceback length of a Viterbi decoder could be applied to determine the survivor depth of the TSOVA decoder as well. The BER simulation results described in [66] show that the optimised survivor and update depths for convolutional codes are approximately $K \times 6$ and $K \times 3$, respectively. On the other hand, in




(a)

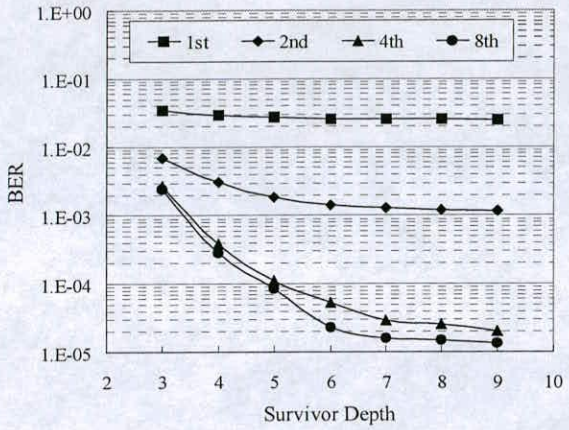



(b)

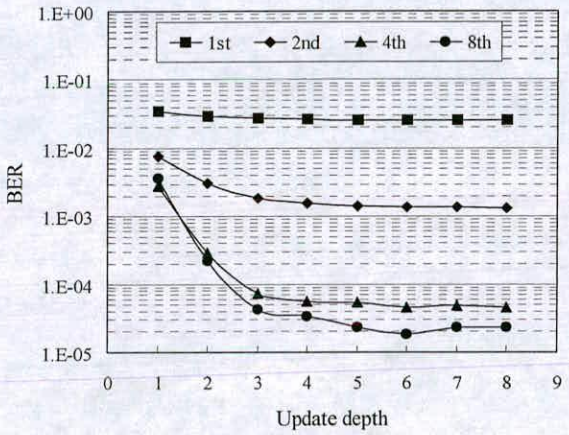
Figure 5.3 BER results for varying (a) the survivor and (b) update depths for K=3.

the literature [33], the depths of the survivor and update processes for turbo codes are suggested to be approximately K (=3) × 6 and K (=3) × 8, respectively.

In this sub section, the BER performance of the TSOVA turbo decoder for constraint lengths K=3, 4, and 5 has been evaluated by varying the survivor and update depths for $E_b/N_0$=2dB. A total of 5M bits of source data were used to obtain the simulation results. Figures 5.3 to 5.5 illustrate the simulation results for constraint lengths K=3, 4, and 5, respectively, after completing 1, 2, 4, and 8 iterations. The improvement in BER performance was almost



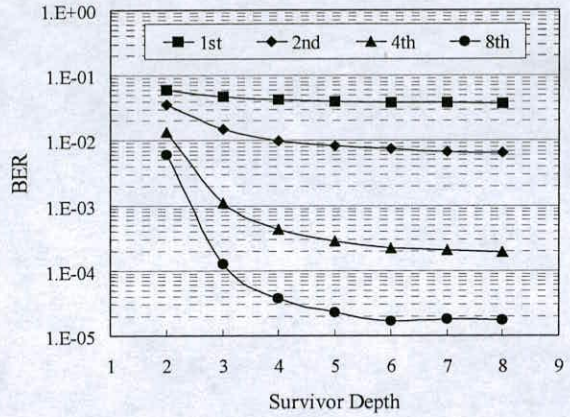

(a)



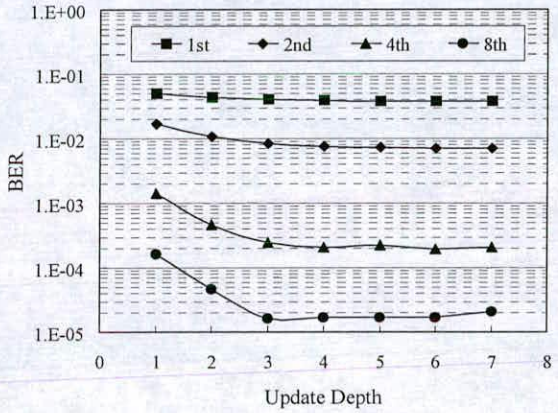

(b)

Figure 5.4 BER results for varying (a) the survivor and (b) update depths for K=4.

saturated after 4 iterations. In these simulations, while the performance for different survivor depths is evaluated, the update depths are fixed to K×5 for each constraint length. On the contrary, during the investigation of the performance for varying the update depths, the survivor depths are fixed to K×8 for each constraint length. The simulation results show that BER performance is almost saturated at near K×6 for the survivor depth and at near K×4 for the update depth. These results are similar to the results described in [93].




(a)




(b)

Figure 5.5 BER results for varying (a) the survivor and (b) update depths for K=5.

## 5.4.2 BER Performance Results

This subsection shows the TSOVA turbo decoder BER performance results for constraint lengths K=3, 4, and 5 with fixed survivor and update depths. The survivor and update depths were determined based on the results obtained in the previous subsection. The survivor and update depths used for the evaluation were:

$$D = K \times 6 \text{ and } U = 0.5 \times D + K \rightarrow D6U4 \tag{5.6}$$

$$D = K \times 8 \text{ and } U = 0.5 \times D + K \rightarrow D8U5 \tag{5.7}$$

Figures 5.6, 5.7 and 5.8 illustrate the BER performance results for K=3, 4, and 5, respectively, after completing 1, 2, 4, and 8 iterations. In these figures, the BER performance of the TSOVA turbo decoder with D6U4 is represented by solid lines, while the dotted lines represent the results with D8U4. As can be seen, the TSOVA turbo decoder for K=4 improved the performance by approximately 1.0dB compared to the turbo decoders for K=3 at a BER of $10^{-4}$. On the other hand, the performance of K=5 TSOVA decoders was not much improved as compared with the K=4 TSOVA turbo decoder results.

Figure 5.6 shows the BER performance results obtained from the depths defined in equation (5.6) and (5.7). In the case of K=3, the TSOVA turbo decoders have been implemented with

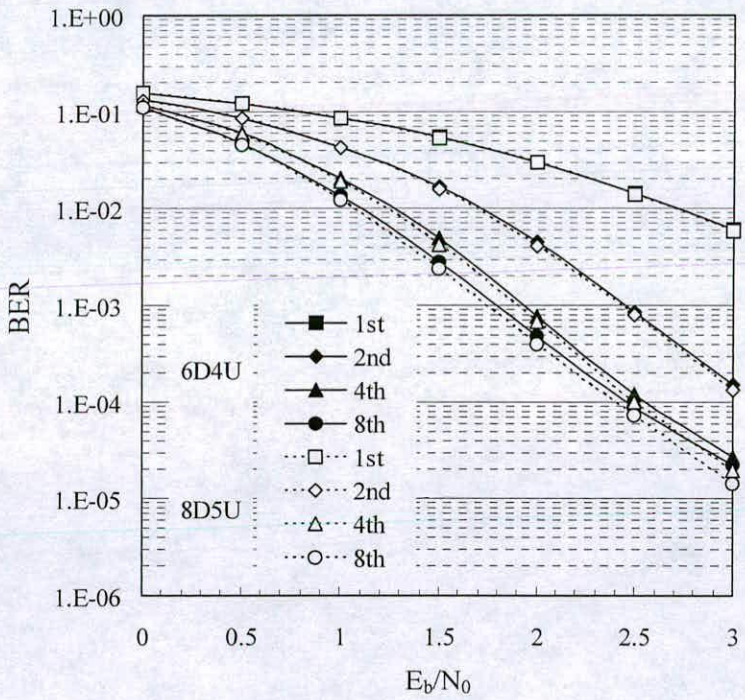



Figure 5.6 The D6U4 and D8U5 TSOVA turbo decoder BER results for K=3.

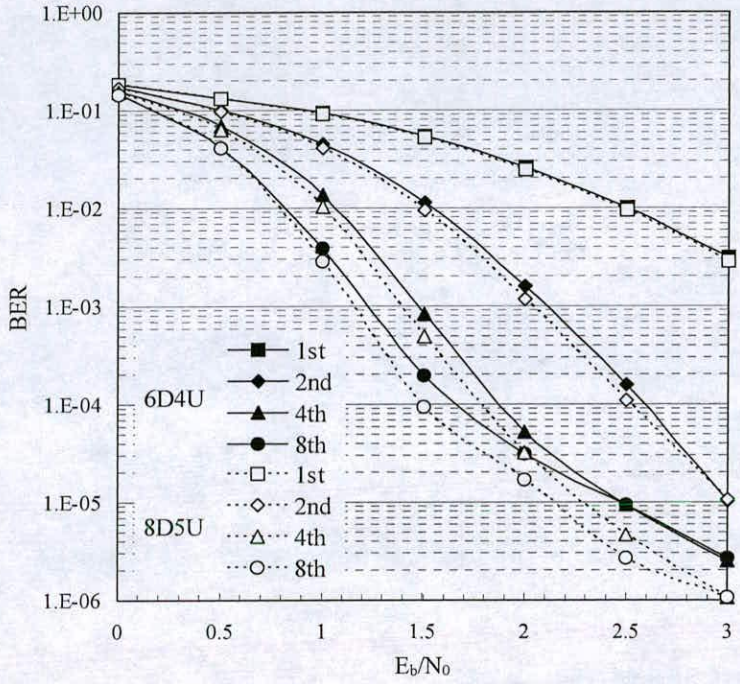



Figure 5.7 The D6U4 and D8U5 TSOVA turbo decoder BER results for K=4.

the survivor depths of 18 and 24, and the update depths of 12 and 15. There was no significant difference in BER performance between these trials.

Figure 5.7 illustrates the BER performance results of the TSOVA turbo decoders for K=4. The TSOVA turbo decoders have been implemented with the survivor depths of 24 and 32, and the update depths of 16 and 20, respectively. It can be shown that after 8 iterations, D8U5 provides about 0.2dB improvement at BER of $10^{-4}$ compared to D6U4.

The BER performance of the TSOVA turbo decoder for K=5 is illustrated in Figure 5.8. In this case, the survivor depths of 30 and 40, and the update depths of 20 and 25 were used to evaluate the TSOVA turbo decoder performance. There was no significant difference in the BER performance between these trials.

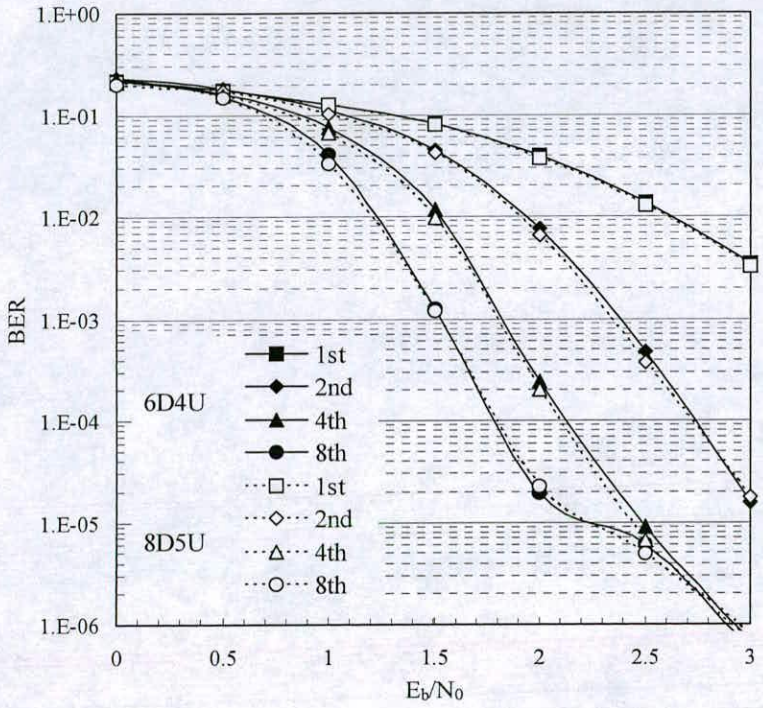



Figure 5.8 The S6U4 and S8U5 TSOVA turbo decoder BER results for K=5.

The performance results of the TSOVA turbo decoders have been compared with the performance of the MLMAP turbo decoders described in Chapter 3. Table 5.1 summarizes the comparisons between the two turbo decoders. Among the results, the largest difference was found between the turbo decoders for K=5. However, the overall results show that turbo decoders based on the two algorithms can provide similar performance. According to [139], MLMAP and TSOVA turbo decoders show almost same BER performance. The results listed in Table 5.1 supports this observation.

Table 5.1 $E_b/N_0$ comparisons at $10^{-4}$ BER. .

| | MLMAP | TSOVA with S8U5 |
|---|---|---|
| K=3 | 2.4 | 2.5 |
| K=4 | 1.6 | 1.5 |
| K=5 | 1.4 | 1.7 |

## 5.5 Summary

This chapter described the SOVA and TSOVA algorithms that are alternatively used for the turbo decoding process. In this chapter, the TSOVA based turbo decoder has been introduced. The survivor and update depths of the TSOVA turbo decoders were investigated to determine an optimised depth. The simulation results have shown that the BER performance was almost saturated when the survivor and update depths were K×6 and K×4, respectively. The TSOVA turbo decoders provided slightly better performance with larger survivor and update depths. Based on these results, the TSOVA turbo decoders for K=3, 4, and 5 were simulated to evaluate their BER performance. Furthermore, the performance results were compared with the results of the MLMAP based turbo decoders at a BER of $10^{-4}$. The comparisons showed that the TSOVA turbo decoder can produce BER performance close to the performance of the MLMAP turbo decoder without significant performance degradations. In the next chapter, a novel TSOVA based turbo decoder architecture will be presented.

# Chapter 6

# Two-Step Soft-Output Viterbi Algorithm Turbo Decoder Architecture

## 6.1 Introduction

In the previous chapter, the two-step SOVA (TSOVA) turbo decoder BER performance has been evaluated and the optimal survivor and update depths investigated. This chapter presents TSOVA turbo decoder architecture with a novel implementation method for implementing the survivor and update processes, designed according to the results obtained by the high level simulations of Chapter 5.

As described in the previous chapter, since TSOVA reduced the computational complexity of the original SOVA, it has been widely used not only for turbo decoder implementations, but also for other applications such as magnetic recoding, etc. [34, 157]. Two algorithms known as register exchange and traceback algorithms (REA and TBA) [85–88] are available to implement the SOVA turbo decoder. These algorithms are already widely used in Viterbi decoder implementations. It is well known that REA can provide high throughput and that TBA is suitable for low power implementations. However, if constraint length, K, is long, REA is not suitable due to the number of registers dramatically increased with K. In general, turbo encoders use constraint lengths up to K=5, as discussed in Chapter 2. Thus, the REA is widely used in TSOVA based turbo decoder implementations [160–162]. However, although TSOVA reduced the computational complexity of the original SOVA, the registers incorporated in REA based TSOVA (TSOVAREA) turbo decoders might increase hardware costs.

To improve the REA based TSOVA turbo decoders, this chapter uses TBA to implement the TSOVA (TSOVATBA) turbo decoder. TBA is implemented with a novel implementation method to save the hardware costs by reducing the number of registers required. Also, the novel method addresses the latency problem known to be a problem of TBA based decoders.

The chapter gives detailed descriptions of TSOVATBA turbo decoder implementations, and shows the performance results compared to the TSOVAREA and MLMAP based turbo decoders.

This chapter is organized as follows. Section 6.2 reviews the TSOVAREA decoder based on the presentation of past literature [93]. Section 6.3 describes the proposed TSOVATBA decoder architecture and its modules. An efficient area architecture is introduced in Section 6.4. Hardware performance results are given in Section 6.5. Section 6.6 summarizes the chapter.

## 6.2 Register-Exchange Algorithm Based Two-Step SOVA Decoder

This section reviews the TSOVAREA decoder architecture [93] before presenting the proposed TSOVATBA decoder architecture. The TSOVATBA architecture is modified from the TSOVAREA architecture. The two architectures have many similarities. However, the main components for the decoding process of each architecture are designed in a different

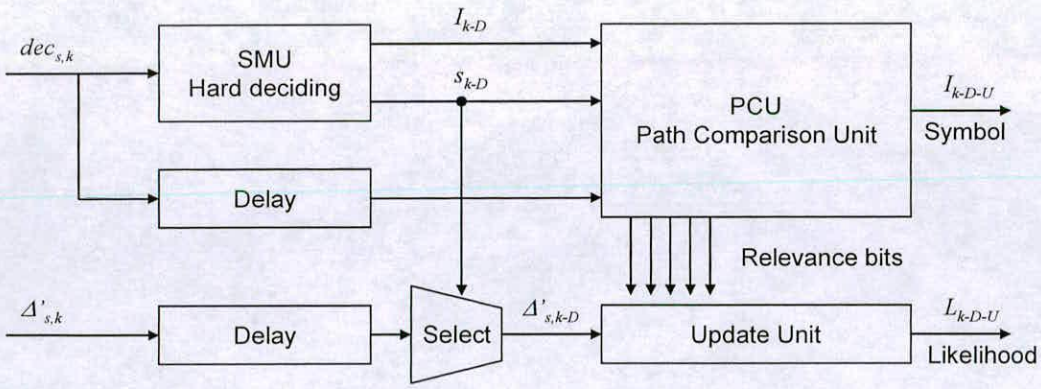



Figure 6.1 REA based TSOVA decoding processor architecture [93]

way. Thus, it is necessary to explain the TSOVAREA architecture and part of the architecture compared to the TSOVATBA architecture in the next section.

Figure 6.1 illustrates the TSOVAREA decoder architecture for the survivor and update processes. The architecture consists of a survivor memory unit (SMU) for generating the hard-decision and the merged state, a path comparison unit (PCU) for comparing the survivor and update paths, an update unit (UPU) for updating and generating the reliability values, and memory blocks for delaying the decision bits and the reliability values. The decision bits ($dec_{s,k}$) and reliability values ($\Delta'_{s,k}$) generated by the transition metric unit (TMU) are input to SMU and Delays, in which one of the Delays is for the decision bits, and the other is for the reliability values. SMU generates the hard-decision, and finds the merged state ($s_{k-D}$) to determine the initial state in PCU. The merged state is also used to select one of the reliability values stored in the Delay. The hard-decision ($I_{k-D}$) is passed to PCU to compare with the hard-decision generated by the competing path. The compared results

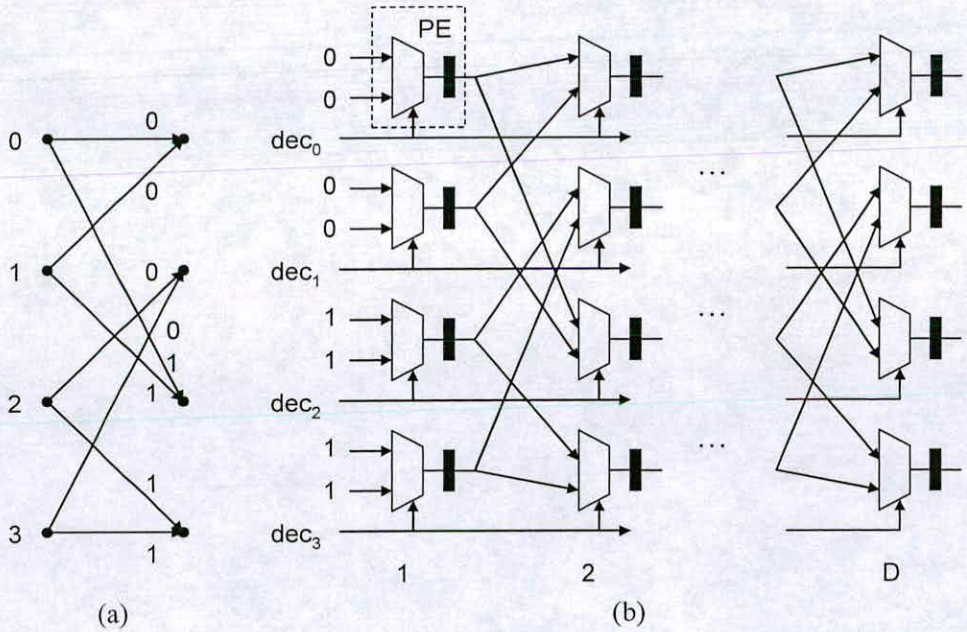



Figure 6.2 (a) The trellis diagram for the convolutional codes K=3. (b) The register exchange algorithm based survivor memory unit structure [93].

obtained by PCU are input to UPU for the updating process. The updating process is performed following the updating rule described by equation (5.5) of Chapter 5.

In the architecture illustrated in Figure 6.1, SMU and PCU are implemented based on the REA. Figure 6.2 illustrates the REA-based SMU structure for generating hard-decision for a trellis diagram with a constraint length K=3 [93]. This structure can also be used to implement PCU. The structure is comprised of an array of process elements (PE) for exchanging the hard-decisions in every clock cycle. The output of the PE is determined by the input decision bits (*dec*) sent from TMU. As can be seen in Figure 6.2, the initial inputs of the first column PEs are the same with the code words of the trellis diagram. The number of columns is the same with the survivor depth. All outputs of the last column PEs are the same and represent the hard-decision.

## 6.3 Traceback Algorithm Based Two-step SOVA Turbo Decoder Architecture

This section presents the proposed TSOVATBA turbo SISO decoder architecture. The architecture is described in comparison with the TSOVAREA decoder architecture described in the previous section, and the components incorporated in the architecture are described in detail with an explanation of their functions in the decoding process.

### 6.3.1 Two-Step SOVA Traceback Turbo SISO Architecture

TSOVATBA turbo decoder architecture is proposed to improve hardware performance in terms of area and power as compared to TSOVAREA and MLMAP turbo decoders. TBA is widely employed in Viterbi decoder implementations due to its advantage of low power consumption as compared to REA-based decoders. In general, the registers for REA cause large area usage and power consumption. Despite low power and area, less attention has been paid to TBA for implementing TSOVA turbo decoders because TBA is regarded as high-latency, which lowers the throughput. The architecture introduced in this section

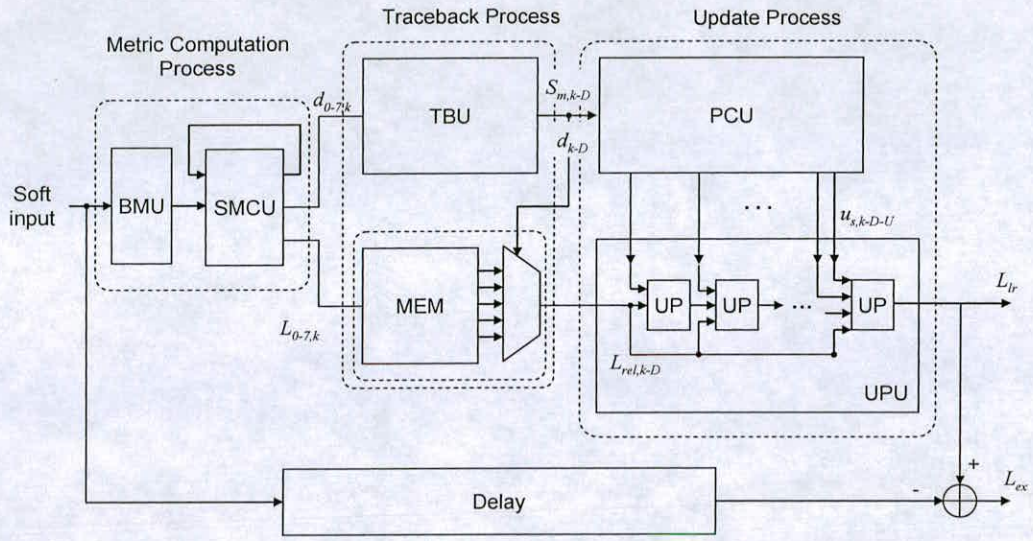



Figure 6.3 TSOVATBA turbo decoder architecture.

does not increase the latency as compared to TSOVAREA decoder, while retaining the low power advantage of TBA.

Figure 6.3 illustrates the TSOVATBA turbo SISO decoder architecture to be implemented in this work. This architecture consists of three main processes, which are metric computations, traceback and update processes. In the metric computation process, the soft-input is computed by branch and state metric computation units (BMU and SMCU) to generate the decision bits, $d_{0\text{-}7,k}$, and the reliability values, $L_{0\text{-}7,k}$, from all the states at time $k$. $d_{0\text{-}7,k}$ and $L_{0\text{-}7,k}$, are passed to the traceback unit (TBU) and memory (MEM) blocks, respectively. TBU finds the merged state, $S_{m,k\text{-}D}$, and one of the reliability values stored in MEM is selected with $S_{m,k\text{-}D}$. The merged state is then sent to the path comparison unit (PCU) that provides the path comparison results to the update process unit (UPU) to output the soft-output. Thus, the role of the traceback and update processes in the architecture is the same as the survivor and update processes in TSOVA decoder introduced in [93]. The extrinsic information for the next decoding process is obtained after subtracting the delayed soft-input from the soft-output.

In the architecture, the memory blocks (MEM and Delay) are realised by using a first-in first-out (FIFO) memory block. MEM consists of parallel FIFO blocks. The number of FIFO blocks is $2^{K-1}$ for constraint length K, and the depth of the FIFO is the same as the survivor depth. The Delay is realized by a FIFO, the depth of which is the same as the total depth of the survivor and update processes.

### 6.3.2 Metric Computation Unit

The metric computation unit (MCU) in the TSOVATBA turbo decoder architecture is similar to the MCU in the MLMAP turbo decoder architecture, shown in Chapter 3. While the MLMAP turbo decoder architecture requires three MCUs, the TSOVATBA turbo decoder architecture needs only one MCU, corresponding to the MCU for the forward process in the MLMAP-based turbo decoding process. Figure 6.4 illustrates the MCU structure incorporated in the TSOVATBA architecture. It consists of a branch metric unit (BMU) for generating the branch metrics, and a state metric computation unit (SMCU), which is realised by a number of parallel add-compare-select-normalization (ACSN) units. The BMU in this architecture is the same with the BMU illustrated in Figure 3.5 of Chapter 3. However, the output sequence of the ACSN differs from the output of the ACSN used in the MLMAP turbo decoder architecture. While the ACSN for the MAP based turbo decoding process produces only the state metrics, the ACSN illustrated in Figure 6.4 for the TSOVATBA turbo decoding process generates the decision bit (DB), the reliability value

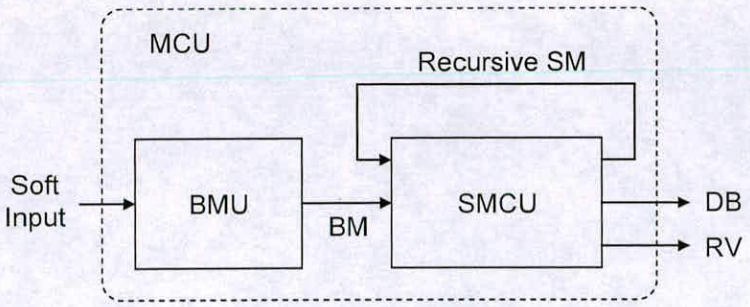



Figure 6.4 The metric computation unit structure in the TSOVATBA architecture.

(RV), and the state metric (SM), which is recursively input to the ACSN itself. DB and RV are passed to TBU and MEM, respectively.

$2^{K-1}$ numbers of parallel ACSN recursively compute the state metrics from the BM and the recursive SM to produce the DB and RV. Figure 6.5 illustrates the ACSN structure used in the TSOVATBA turbo decoder architecture. This ACSN structure generates the state metric, $sm_0(k+1)$, of state 0 at time $k+1$. For obtaining $sm_0(k+1)$, two branch ($bm_{00}(k)$, $bm_{11}(k)$) and two state metrics ($sm_0(k)$, and $sm_1(k)$) at time $k$ are input to the ACSN. First, the first two adders (A) compute two new state metrics, which then are compared in order to select (CS) the maximum state metric. In the CS process, a subtractor, which is represented with an

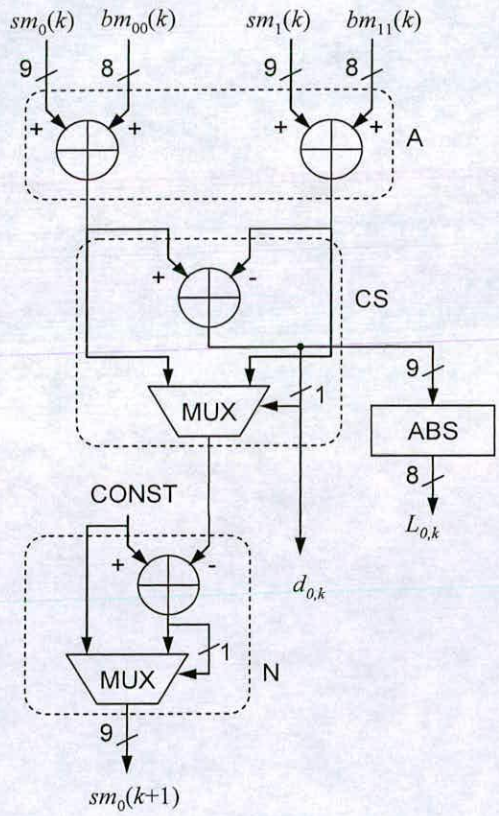



Figure 6.5 The add-compare-select-normalization structure in the TSOVATBA turbo decoder architecture.

adder in Figure 6.5, is used for the comparing process. The most significant bit (MSB) of the subtractor output is input to the multiplexer (MUX) to select the maximum state metric, and is associated with the decision bit ($d_{0,k}$) passed to the survivor process. The reliability value ($L_{0,k}$) sent to MEM is the absolute value of the subtractor output. In this process, 8-bits wordlength was enough to represent the absolute value, which is obtained from simply discarding the second MSB of the absolute value before passing to MEM.

In the turbo decoding process, the state metrics increase rapidly due to the use of soft-input symbol data and extrinsic information. This requires a large wordlength to represent the state metrics. To reduce the size of the wordlength without overflowing, the state metrics normalization process (N) is necessary as illustrated in Figure 6.5. This process is performed when the maximum state metric is larger than a constant (CONST), as described in [105].

### 6.3.3 Traceback Process Unit

The survivor process is performed to determine the merged state, from which the selected reliability value is passed to the update process. While the merged state is searched by the traceback process unit (TBU), the reliability values generated by SMCU are stored in MEM before selecting one of the reliability values using the merged state. The TBU is one of the main units in the TSOVATBA turbo decoder architecture. It uses the decision bits generated by SMCU to determine the merged state by tracing in backward as shown in Figure 5.2 of Chapter 5. Thus, the role of TBU is the same as the SMU of the TSOVAREA turbo decoder described in the previous section, but the TBU operates in a different way to find the merged state.

Figure 6.6 illustrates a novel TBU structure proposed in this thesis. In this structure, a process element (PE) is placed on each state as the trellis state diagram for constraint length K=4. Thus, the TBU consists of $2^{K-1}$ numbers of PE row and D numbers of PE columns. The structure shown in Figure 6.6 appears to be similar to the structure of the SMU introduced in [93] for the TSOVAREA decoder implementation. However, there are some significant differences in the implementation methods described below.

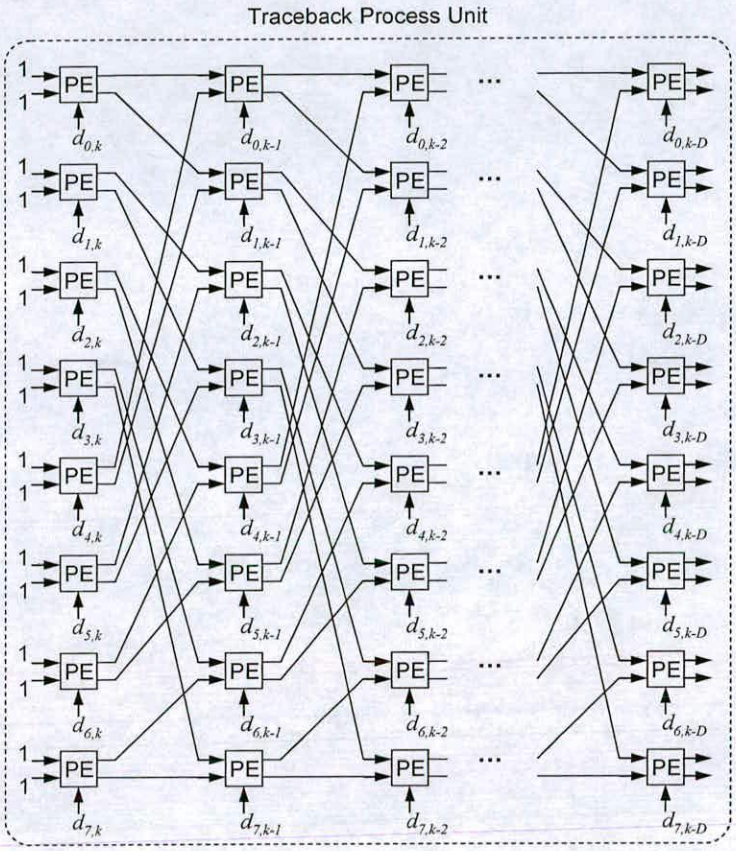



Figure 6.6 The traceback process unit structure for searching the merged state.

First of all, PEs incorporated in TBU are implemented with only four gates and have no registers, as shown in Figure 6.7. As illustrated in Figure 6.6, each of the PEs are connected by identical wired connections with the transition paths of the trellis diagram illustrated in Figure 5.2. On the other hand, PEs used for the SMU contain a register for exchanging the hard-decision. Thus, while the inputs of the first column PEs of the SMU are initialized with the code words corresponding to the hard-decisions, all first column PE inputs in TBU are initialized to '1' to make all state in the first column survivor states. Here '1' denotes a survivor path such that only one of the 16 outputs at the last column of PEs becomes '1' after completing the survivor process. However, all outputs of the SMU are the same, representing the hard-decision. Hence, the SMU needs an accumulator for state information in order to

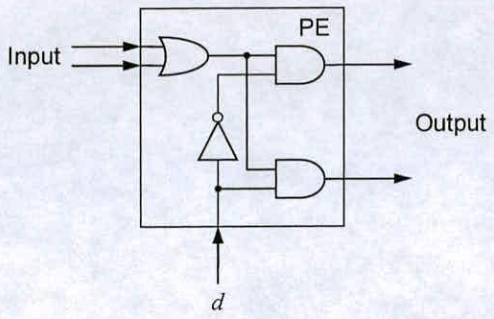



Figure 6.7 The process element structure.

determine the merged state, while the hard-decision and merged state in the TBU can be represented by using one of the outputs of the PEs in the last column without a memory for storing the state information.

Furthermore, in the SMU, the same decision bits generated by TMU are input to each of the PEs in the same row in every clock cycle. However, as shown in Figure 6.6, the decision bit input to all the PEs of the TBU are different, and are controlled by shift registers. The decision bits in the TBU are used for determining which of PE outputs is '1' or '0', in which '1' indicates a survivor path of a state. If the input sequences of PE are '0', the outputs of PE become '0' no matter what the input decision bit is. The survivor paths are finally merged to one PE at time $k$-D, which is the merged state, $S_{m,k\text{-D}}$. During the decoding process, the TBU provides only the merged state to the PCU without the sequences of the hard-decision, while the SMU generates the hard-decision and merged states for the update process.

In our TBU, a potential issue could be a critical path delay occurring due to the wired connections between the PEs. Whereas, in REA-based SMU, a critical path delay is no problem at all. The critical path delay due to the wired connections may be a cause of speed reduction of the turbo decoder to achieve maximum throughput. In general, it is well-known that the critical path delay of a turbo decoder is in ACSN [13]. Critical paths of the TBU and the ACSN in Figure 6.5 are investigated to determine which path affects the determination of the maximum speed of TSOVATBA turbo decoder implementation.

### 6.3.4 Path Comparison and Update Process Units

The update process is performed to generate the soft-output, including the hard-decision and the reliability value and the extrinsic information for the next decoding process. Figure 6.8 illustrates the path comparison and update process units (PCU and UPU) for performing the update process. PCU generates the comparison sequences of the survivor and competing paths, which are passed to REU to perform the updating process following the update rule in (5.5). The inputs of PCU are the decision bits, the merged state information, and the selected reliability value. The decision bits and the merged state are sent from TBU. The selected reliability value is one of the outputs delayed by the memory block.

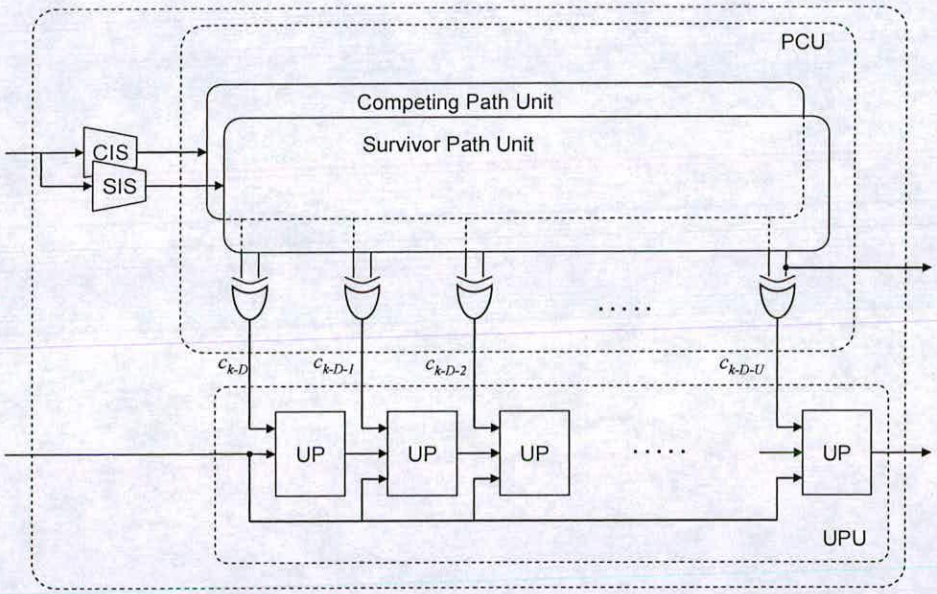



Figure 6.8 The path comparison and update process unit structures

Figure 6.8 illustrates the UPU structure, in which the PCU consists of a competing path unit (CPU) and a survivor path unit (SPU). While the inputs of TBU are initialized to '1', the inputs of the CPU and SPU are initialized by the merged state, which indicates the initial state to start the survivor and update processes. The CPU and SPU generate the comparison

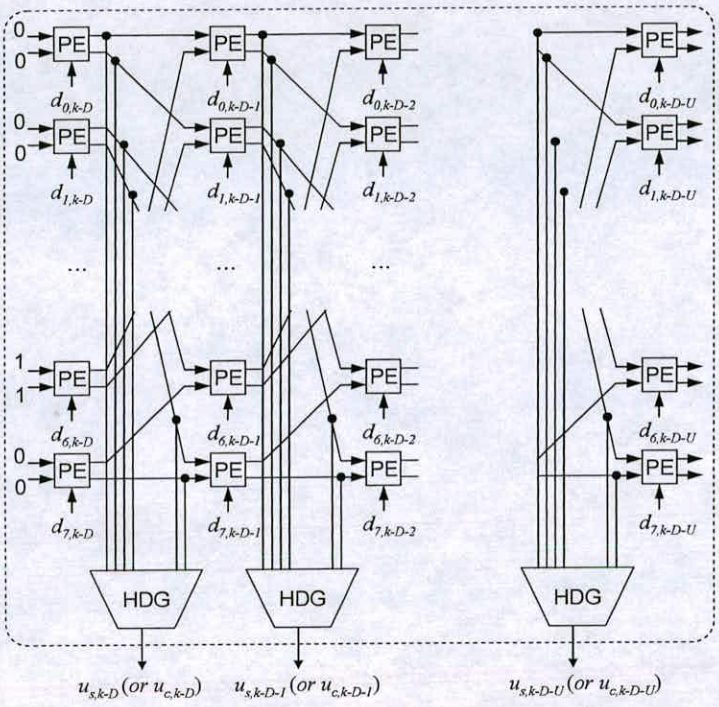



Figure 6.9 The structure of CPU and SPU.

bits. These are then processed by the update processors (UP), which are realized based on the update rule given by equation (5.5) in Chapter 5.

Figure 6.9 illustrates the structure of CPU and SPU, which is almost the same with the structure of TBU, shown in Figure 6.6. The number of the column PEs is the same as the update length, U. While the inputs of all first column PEs in TBU are initialized to '1', the inputs of only one PE in PCU are initialized to '1'. The initial inputs of CPU and SPU are determined by the merged state obtained by the TBU. During the update process, only one output path of each of the column PEs becomes '1', and the rest of their outputs are '0'. From this output sequence, hard-decisions ($u_{s,k}$ and $u_{c,k}$) of the survivor and competing paths can be generated by a hard-decision generator (HDG), and then they are compared to determine whether the reliability value needed to be updated or not following the rule in (5.5). The XORs shown in Figure 6.8 generate the comparison results ($c_k$) that are input to

the UPU. As shown in Figure 6.8, the UPU is implemented with pipelined stages, in which the stage number is also the same as the update depth, U.

## 6.4 Area-Efficient Traceback Two-Step SOVA Turbo SISO Decoder Architecture

Figure 6.10 illustrates the area-efficient TSOVATBA turbo SISO decoder architecture. In this architecture, the MEM in the architecture illustrated in Figure 6.3 is replaced with an additional MCU incorporating BMU and SMCU, which is similar to the technique suggested in [152]. In addition, the Delay block for delaying the soft-input is divided into two blocks. The MEM area depends on the survivor depth and the constraint length. On the other hand, the area of MCU replacing MEM depends only on the constraint length. The separated Delay0 has the same depth as the survivor depth. The total depth of Delay0 and Delay1 is

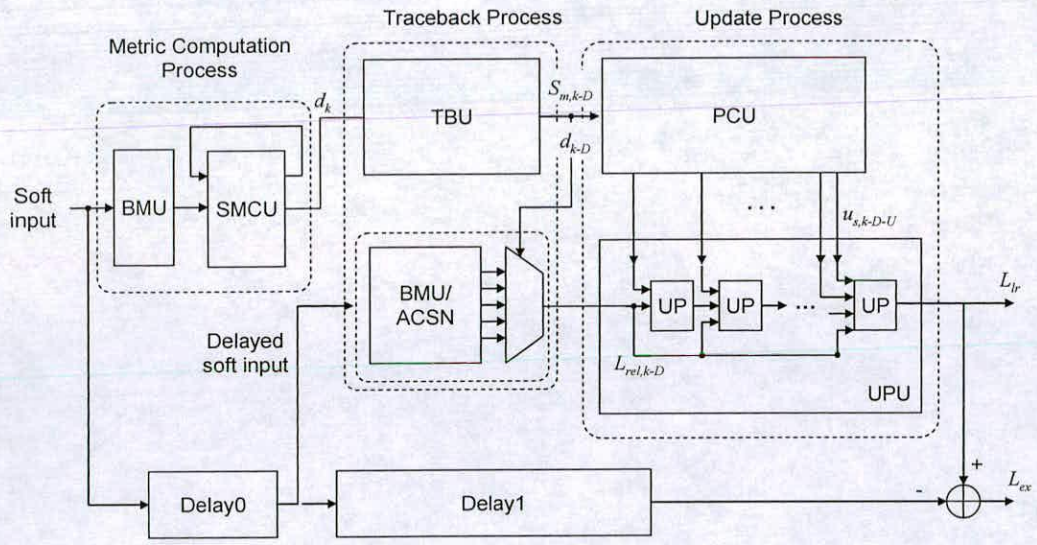



Figure 6.10 The area efficient TSOVA turbo decoder architecture

almost same as the depth of the Delay of the architecture illustrated in Figure 6.3. The two architectures described in Figure 6.3 and 6.10 are compared for area usage and power.

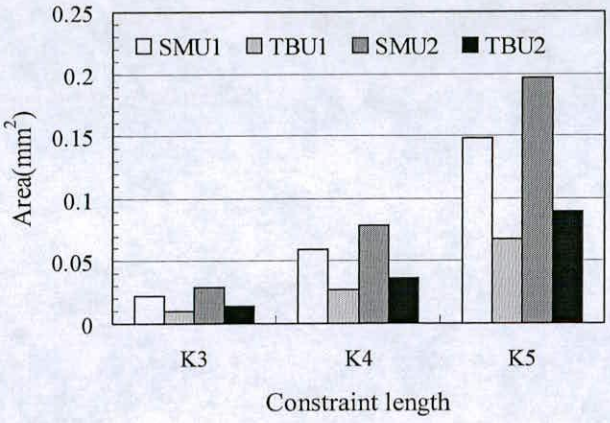
## 6.5 Results

The TBA based TSOVA turbo SISO decoders have been designed at RTL using Verilog HDL. After completing RTL simulations with Cadence Verilog-XL™, they were synthesized with Synopsys DesignCompiler™ using the UMC 0.18μm standard CMOS technology. The gate-level netlists obtained from the synthesis were simulated with Cadence Verilog-XL™. Power consumption of the TSOVATBA decoders was evaluated by Synopsys PowerCompiler™ with the switching activities obtained during the gate-level netlists simulation. The test systems used in the simulation were the same with those used in Chapters 3 and 4 for the MLMAP turbo decoder simulation. The block size and other parameters, such as code rates, fixed-point representation for soft-input, etc., were also the same.

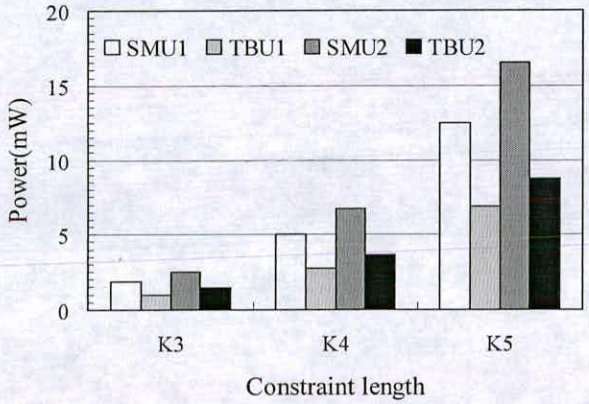
### 6.5.1 REA and TBA Results Comparisons

In TSOVA based turbo decoder implementations, as already stated in previous sections, one of the main processes is the survivor process, which can be implemented with REA or TBA. REA has been popularly employed for TSOVA decoder implementations using the method described in Section 6.2. The main differences between TSOVAREA and TSOVATBA decoders are in the modules that perform the survivor and update processes, while MCU can be commonly used in both decoder implementations. Thus, by comparing the SMU incorporated in TSOVAREA with the TBU used in TSOVATBA, we can forecast how much difference exists between the hardware performances of these decoders. For this comparison, REA based SMU following the structure described in [93] has been implemented and compared with the area usage and power consumption of the proposed TBU illustrated in Figure 6.5. Note that in this comparison SMU and TBU have been compared without the inclusion of memory blocks for storing the reliability values. Two different survivor depths,

K×6 and K×8, for constraint lengths K=3, 4, and 5, were considered for these performance




(a)




(b)

Figure 6.11 (a) Area and (b) power comparisons.

comparisons. The update depths were fixed to (0.5×the survivor depth + K). The depths were determined based on the decoder performance results obtained in the previous chapter.

Figures 6.11 (a) and (b) illustrate the area and power results of the SMU and TBU for K=3, 4, and 5. In these figures, SMU1 and TBU1 imply the survivor depth K×6 and the update depth K×4, while SMU2 and TBU2 imply the survivor depth K×8 and the update depth K×5. Table

6.1 summarizes the results. It is clear from the results that the area and power depends

Table 6.1 List of the SMU and TBU area and power results.

| | | D=K×6, U=K×4 | | | D=K×8, U=K×5 | | |
|---|---|---|---|---|---|---|---|
| | | SMU1 | TBU1 | Save (%) | SMU2 | TBU2 | Save (%) |
| K=3 | Area ($mm^2$) | 0.022 | 0.0103 | 53.7 | 0.029 | 0.013 | 53.9 |
| | Power (mW) | 1.85 | 0.96 | 47.8 | 2.48 | 1.41 | 42.9 |
| K=4 | Area ($mm^2$) | 0.059 | 0.027 | 53.9 | 0.078 | 0.036 | 54.1 |
| | Power (mW) | 5.02 | 2.71 | 45.8 | 6.69 | 3.67 | 45.1 |
| K=5 | Area ($mm^2$) | 0.147 | 0.067 | 54.1 | 0.196 | 0.089 | 54.2 |
| | Power (mW) | 12.45 | 6.85 | 44.9 | 16.51 | 8.71 | 47.2 |

strongly on the constraint length, as well as the survivor and update depths. The results also show that TBU can save 54% in area and 47% in power as compared to SMU. The area and power savings of TBU were almost same with different constraint lengths, survivor and update depths. While SMU is implemented with a number of registers used for REA, TBA used in TBU requires only wired connections which contribute to much of the area and power savings of TBA compared to REA.

### 6.5.2 TSOVATBA Turbo Decoder Hardware Performance Results

Two TSOVATBA based turbo decoders have been implemented for the evaluation of their hardware performance. The two decoders are illustrated in Figures 6.3 and 6.9, and are called the TSOVATBA turbo decoder (TBTD) and the area efficient TSOVATBA turbo decoder (AETD), respectively.

First, the latency was investigated, which affects the throughput performance. The iterative process in a turbo decoder results in a large latency that reduces the throughput. In general, the latency of a MAP based turbo decoder with sliding window (SW) method is known to be

4 times the window size in the first half iteration [16]. On the other hand, the latency of TSOVATBA can be defined as the total depth of the survivor and update processes. If it is assumed that the survivor depth is the same as the window size, the latency of the TSOVATBA becomes less than half of the MAP based turbo decoder latency. It must also be noted that TSOVATBA and TSOVAREA have the same latency. The latency of the TSOVATBA turbo decoders could be given as follows:

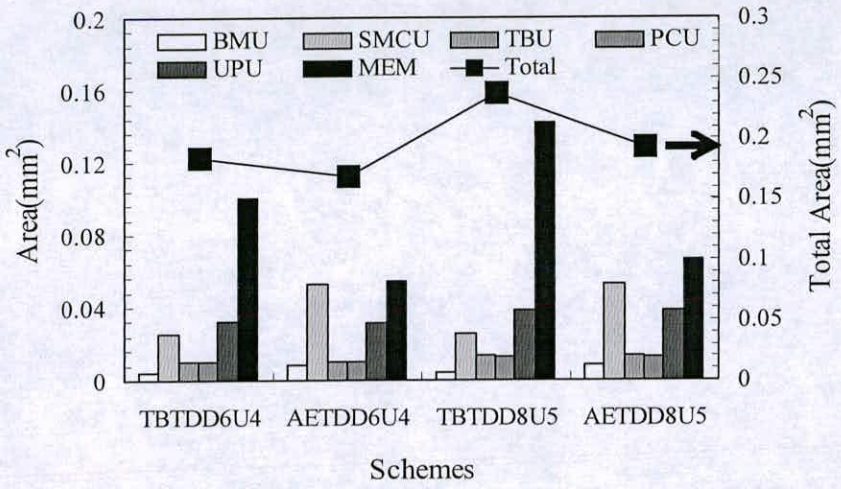
$$\text{Latency} = (D + U + P) \times \text{a clock period} \tag{6.1}$$

where D is the survivor depth, U is the update depth, and P is the number of pipeline stages in the TSOVATBA turbo decoder implementation. P depends on the design of the decoder and is typically much less than U. As can be seen, even though D is the same as the window size in the MAP based turbo decoders, the latency of the TSOVATBA turbo decoders is less than that of the MAP based turbo decoders.

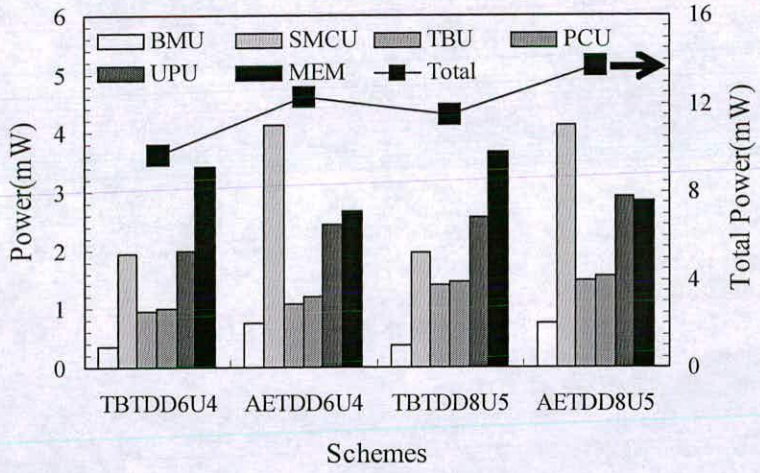
The area usage and power consumption results for TBTD and AETD are illustrated in Figures 6.12–6.14, and summarized in Table 6.2–6.7 for constraint lengths K=3, 4, and 5. In the figures, the primary and secondary Y-axes indicate the breakdown component and total results, respectively. As before, two different survivor and update depths are considered in the implementation of the decoders. The results show that the overall area of AETD for K=3 and 4 is saved by up to 26% as compared with TBTD area results. On the contrary, the overall power of AETD for K=3 and 4 increased by up to 27% as compared to TBTD. However, for K=5, AETD saves up to 36% in area and 6% in power. From these results, it is clear that the replacement of the memory for the reliability values with an additional MCU can achieve significant savings in overall area, while increasing the overall power consumption for K=3 and 4 due to high switching activities led by the replaced MCU. On the other hand, the memory size for the reliability values is increased proportional to the number of states, $2^{K-1}$. Thus, for K=5, the results show that the dramatically increased memory blocks consume more power than the computational logics of MCU. The results for each constraint length are discussed below.

Figures 6.12 (a) and (b) illustrates the area and power results for TBTD and AETD, respectively, for K=3. The results are summarized in Tables 6.2 and 6.3. AETD D6U4 and D8U5 save 8% and 18% in area, respectively, as compared to TBTD D6U4 and D8U5. The

replacement of the memory block for storing the reliability values leads significantly to area




(a)




(b)

Figure 6.12 (a) Area and (b) power comparisons for K=3.

reduction. In AETD, the area of BMU and SMCU is increased to two times that of the BMU and MCU of TBTD. This is because the BMU and SMCU area results of AETD include the area of the added MCU for generating the reliability values for the UPU. This results in the

power increase of AETD, despite of the area reduction. Thus, it is clear that although the

Table 6.2 List of area results for K=3

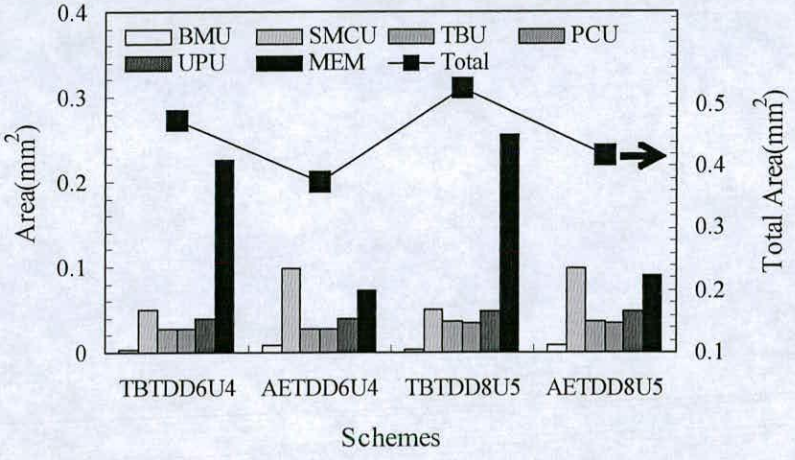
| | D6U4 | | | D8U5 | | |
|---|---|---|---|---|---|---|
| | TBTD ($mm^2$) | AETD ($mm^2$) | Save (%) | TBTD ($mm^2$) | AETD ($mm^2$) | Save (%) |
| BMU | 0.004 | 0.008 | -100 | 0.004 | 0.008 | -100 |
| SMCU | 0.025 | 0.052 | -105.6 | 0.025 | 0.052 | -105.6 |
| TBU | 0.010 | 0.010 | 0 | 0.013 | 0.013 | 0 |
| PCU | 0.010 | 0.010 | 0 | 0.013 | 0.013 | 0 |
| UPU | 0.032 | 0.031 | 2.8 | 0.038 | 0.038 | 0 |
| MEM | 0.100 | 0.054 | 46.0 | 0.141 | 0.066 | 53.2 |
| Total | 0.183 | 0.168 | 8.5 | 0.236 | 0.192 | 18.6 |

Table 6.3 List of power results for K=3

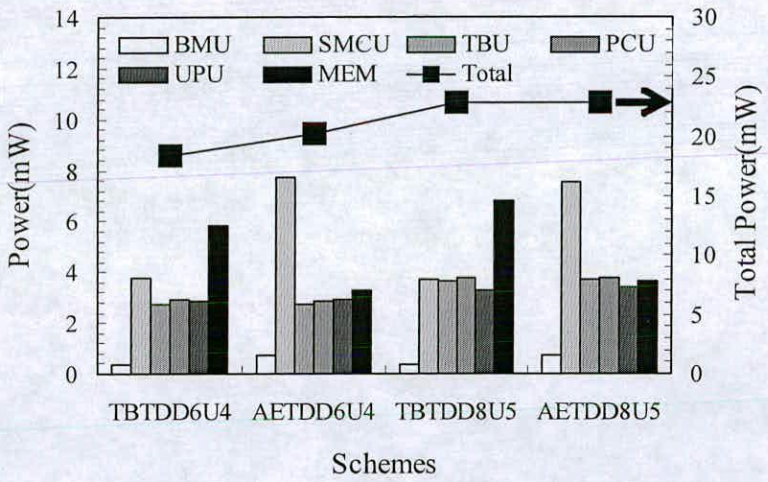
| | D6U4 | | | D8U5 | | |
|---|---|---|---|---|---|---|
| | TBTD (mW) | AETD (mW) | Save (%) | TBTD (mW) | AETD (mW) | Save (%) |
| BMU | 0.36 | 0.75 | -106.8 | 0.37 | 0.74 | -100 |
| SMCU | 1.93 | 4.14 | -114.0 | 1.96 | 4.13 | -109.9 |
| TBU | 0.96 | 1.09 | -12.6 | 1.41 | 1.49 | -5.0 |
| PCU | 1.00 | 1.21 | -20.6 | 1.46 | 1.56 | -7.0 |
| UPU | 1.98 | 2.43 | -22.9 | 2.58 | 2.91 | -12.8 |
| MEM | 3.43 | 2.66 | 22.2 | 3.67 | 2.85 | 22.2 |
| Total | 9.69 | 12.31 | -27.0 | 11.47 | 13.7 | -19.3 |

computational logic area is less than the memory area, it can consume more power due to a high switching activity led by the metric computation process.

Figures 6.13 (a) and (b) and Tables 6.4 and 6.5 illustrate and summarize the area and power results of TBTD and AETD for K=4. Similar to the results for K=3, AETD saves up to 26%




(a)




(b)

Figure 6.13 (a) Area and (b) power comparisons for K=4.

in area by eliminating the memory for the reliability values. The power of AETD is slightly increased by 9% as compared to TBTD. This power increase rate is smaller than the increase

seen for K=3. The number of states for K=4 is doubled compared to K=3. This leads to an increase of computational complexity and memory size. Thus, it is clear from the results for

Table 6.4 List of area results for K=4.

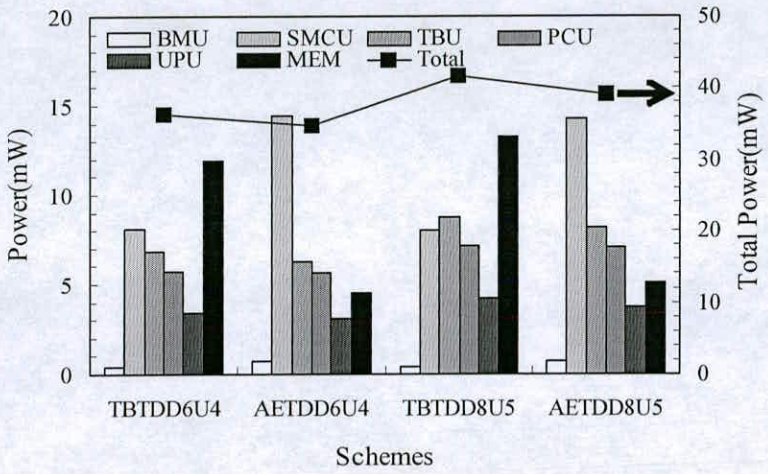
| | D6U4 | | | D8U5 | | |
|---|---|---|---|---|---|---|
| | TBTD ($mm^2$) | AETD ($mm^2$) | Save (%) | TBTD ($mm^2$) | AETD ($mm^2$) | Save (%) |
| BMU | 0.004 | 0.008 | -100 | 0.004 | 0.008 | -100 |
| SMCU | 0.049 | 0.099 | -101.39 | 0.049 | 0.099 | -101.39 |
| TBU | 0.027 | 0.027 | 0 | 0.036 | 0.036 | 0 |
| PCU | 0.027 | 0.027 | 0 | 0.034 | 0.034 | 0 |
| UPU | 0.039 | 0.039 | 0 | 0.048 | 0.048 | 0.31 |
| MEM | 0.226 | 0.073 | 67.70 | 0.255 | 0.090 | 64.67 |
| Total | 0.374 | 0.275 | 26.46 | 0.427 | 0.316 | 25.94 |

Table 6.5 List of power results for K=4

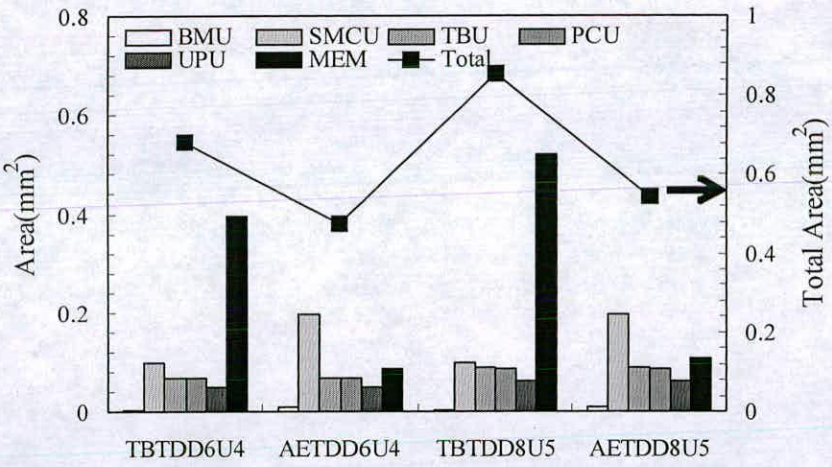
| | D6U4 | | | D8U5 | | |
|---|---|---|---|---|---|---|
| | TBTD (mW) | AETD (mW) | Save (%) | TBTD (mW) | AETD (mW) | Save (%) |
| BMU | 0.37 | 0.74 | -101.61 | 0.37 | 0.75 | -101.34 |
| SMCU | 3.79 | 7.74 | -104.05 | 3.74 | 7.53 | -101.30 |
| TBU | 2.71 | 2.71 | 0.18 | 3.67 | 3.70 | -0.89 |
| PCU | 2.89 | 2.86 | 0.89 | 3.78 | 3.76 | 0.29 |
| UPU | 2.83 | 2.89 | -1.93 | 3.31 | 3.43 | -3.68 |
| MEM | 5.81 | 3.26 | 43.94 | 6.80 | 3.64 | 46.45 |
| Total | 18.43 | 20.23 | -9.73 | 22.82 | 22.83 | -0.05 |

K=4 that the increased memory for the reliability values contributes to the total power as well as the total area as compared to the results for K=3 decoder results. This difference is even greater in the results for K=5.

The area and power results for K=5 are illustrate in Figures 6.14 (a) and (b), and summarized in Tables 6.6 and 6.7. The results for K=5 show different aspects than the results for K=3




(a)




(b)

Figure 6.14 (a) Area and (b) power comparisons for K=5.

and 4. In the TSOVATBA turbo decoder architecture, the memory size for the reliability values is higher than that of K=3 and K=4 results. Thus, AETD can achieve savings in both

area and power as compared to TBTD. The area of the memory is much more reduced compared to K=3 and 4, which also leads to the overall power reduction of AETD.

Table 6.6 List of area results for K=5

| | D6U4 | | | D8U5 | | |
|---|---|---|---|---|---|---|
| | TBTD ($mm^2$) | AETD ($mm^2$) | Save (%) | TBTD ($mm^2$) | AETD ($mm^2$) | Save (%) |
| BMU | 0.004 | 0.008 | -100 | 0.004 | 0.008 | -100 |
| SMCU | 0.096 | 0.195 | -102.1 | 0.096 | 0.195 | -102.1 |
| TBU | 0.067 | 0.067 | - | 0.089 | 0.089 | - |
| PCU | 0.068 | 0.068 | - | 0.084 | 0.084 | - |
| UPU | 0.048 | 0.048 | 0.3 | 0.059 | 0.059 | 0.2 |
| MEM | 0.396 | 0.084 | 78.6 | 0.519 | 0.106 | 79.4 |
| Total | 0.682 | 0.473 | 30.6 | 0.855 | 0.544 | 36.2 |

Table 6.7 List of power results for K=5.

| | D6U4 | | | D8U5 | | |
|---|---|---|---|---|---|---|
| | TBTD (mW) | AETD (mW) | Save (%) | TBTD (mW) | AETD (mW) | Save (%) |
| BMU | 0.36 | 0.74 | -106.0 | 0.36 | 0.73 | -104.7 |
| SMCU | 8.10 | 14.48 | -78.6 | 7.99 | 14.29 | -78.7 |
| TBU | 6.85 | 6.29 | 8.1 | 8.71 | 8.19 | 5.9 |
| PCU | 5.71 | 5.64 | 1.2 | 7.16 | 7.08 | 1.0 |
| UPU | 3.38 | 3.05 | 9.8 | 4.22 | 3.69 | 12.6 |
| MEM | 11.88 | 4.50 | 62.0 | 13.28 | 5.08 | 61.7 |
| Total | 36.30 | 34.72 | 4.3 | 41.74 | 39.1 | 6.3 |

### 6.5.3 Comparing with MLTBD

The TSOVATBA turbo decoder results shown in the previous sub section are compared with the conventional MLTBD turbo decoder performance results obtained in Chapter 3. In the literature, an REA based SOVA or TSOVA turbo decoder does not reduce the computational complexity or improve hardware performance as compared to a MLTBD turbo decoder. This chapter shows that the proposed TBA based TSOVA turbo decoder can save more area and power than the REA based TSOVA turbo decoder. Thus, this section shows performance comparison between the TSOVATBA and the MLTBD turbo decoders for different constraint lengths, K=3, 4, and 5. Table 6.8 lists the area and power comparison results between the MLTBD, the TBTD, and the AETD. The TBTD and AETD results were obtained from the survivor and update depths of D8U5. From the results, the TBTD and AETD for K=3 can save up to 65% and 72% in area and 52% and 42% in power as compared to the MLTBD for K=3, respectively. For K=4 and 5, the area and power savings of the TBTD and AETD are slightly less than for K=3, but the area and power savings still reach up to 66% and 34% for K=4 and 62% and 37% for K=5, respectively.

Table 6.8 Area and power comparisons for the MLMAP and TSOVATBA based turbo decoder schemes.

| | | MLTBD | TBTD | Save (%) | AETD | Save (%) |
|---|---|---|---|---|---|---|
| K=3 | Area ($mm^2$) | 0.689 | 0.236 | 65.7 | 0.192 | 72.1 |
| | Power (mW) | 23.9 | 11.4 | 52.3 | 13.7 | 42.6 |
| K=4 | Area ($mm^2$) | 0.93 | 0.427 | 54.0 | 0.316 | 66.0 |
| | Power (mW) | 34.78 | 22.82 | 34.3 | 22.83 | 34.3 |
| K=5 | Area ($mm^2$) | 1.452 | 0.855 | 41.1 | 0.544 | 62.5 |
| | Power (mW) | 62.3 | 41.7 | 33.0 | 39.1 | 37.2 |

## 6.6 Summary

This chapter presented a TSOVA turbo decoder based on TBA. The decoder architecture was described in comparison with the TSOVAREA architecture suggested in the literature. The hardware simulation results have shown that the TSOVATBA turbo decoder implementations can save more area and power than TSOVAREA turbo decoders. The savings are due to eliminating the registers required by the survivor and update processes of the TSOVAREA turbo decoders. The TSOVATBA turbo decoders were also compared with the MLMAP turbo decoders in terms of area and power. The results of this comparison showed that the TSOVATBA turbo decoders can reduce area and power more so than the MLMAP turbo decoders. Thus, the TSOVATBA turbo decoders could be more suitable for mobile and portable wireless communication systems requiring low power consumption and area usage without significant BER performance degradation, as shown in the previous chapter. In the next two chapters, reconfigurable application specific turbo decoder architectures targeting multi standard wireless communication systems are presented. The chapters present a turbo decoder that can be reconfigured for different constraint lengths and different type of turbo codes.

# Chapter 7

# Reconfigurable Turbo Decoder Architecture

## 7.1 Introduction

Wireless communication systems are constructed with a number of digital signal processing techniques. One trend for future wireless communication systems is that of converging to one system, which is compatible and flexible for different standardized systems [162-165]. In order to make the systems compatible and flexible, digital signal processors incorporated in these systems must be reconfigurable and reusable for different standards.

In wireless communication systems, the adoption of different encoding techniques has led to the development of a reconfigurable channel decoder, like the reconfigurable Viterbi decoder [166-168]. After the emergence of turbo codes, a dual mode decoder, which can be reconfigured as Viterbi and turbo decoders, has been proposed [13, 169-170] for supporting convolutional and turbo codes. However, a reconfigurable turbo decoder is considered less in the literature, and the demand of the reconfigurable turbo decoder is continuously expected to be increased in the future by requiring the compatibility and reusability with a very low BER. Table 7.1 summarizes the turbo code specifications for different applications [171-172]. As can be seen, the encoding schemes and parameters of turbo codes vary with the individual applications. Thus, this chapter presents a reconfigurable application specific integration circuit (REASIC) turbo decoder architecture that is designed to support turbo codes generated by different constraint lengths.

The reconfigurable application specific turbo decoder (REASTD) architecture presented in this chapter, which can be configured dynamically, is designed based on the K=5 MLMAP turbo decoder proposed in Chapter 4. This architecture can be reconfigured for K=3 and 4 with a proposed mapping method, which is used for implementing the state metric and LLR value computation units. The proposed mapping method involves minimized area overhead

Table 7.1 List of turbo codes for different applications [171-172].

| Applications | Turbo code | Termination | Polynomials | Rates |
|---|---|---|---|---|
| CCDSD | Binary, K=5 | Tail bits | 23, 33, 25, 37 | 1/6, 1/4, 1/3. 1/2 |
| UMTS, CDMA2000 | Binary, K=4 | Tail bits | 13, 15, 17 | 1/4, 1/3, 1/2 |
| DVB-RCS | Double-binary, K=4 | Circular | 13, 15 | 1/3 up to 6/7 |
| DVB-RCT | Double-Binary, K=5 | Circular | 13, 15 | 1/2, 3/4 |
| M4 | Binary, M=5 | None | 23, 35 | 1/2 |
| Skyplex | Double -Binary, K=4 | Circular | 13, 15 | 4/5, 6/7 |
| IEEE802.16, WiMax | Double -Binary, K=4 | Circular | 13, 15 | 1/2 up to 7/8 |

for the reconfigurable implementation. In order to achieve low power reconfigurable turbo decoder implementation, a clock gating method is employed in the memory blocks for storing the state metrics, and the units for computing the state metrics and the LLR values. With the help of simulation results, the hardware performance of the reconfigurable turbo decoder is compared with the ASIC turbo decoder implementations.

In addition, as shown in Table 7.1, there are two types of turbo codes adopted as a standard in wireless communication systems. They are binary and double-binary turbo codes (BTC and DTC). As shown in Chapter 2, BTC and DTC are generated by different encoding schemes. Thus, the reconfigurable turbo decoder for supporting the two turbo codes is desirable for compatible and flexible systems.

This chapter also proposes a reconfigurable application specific turbo decoder (RDASTD) architecture that can be configured for BTC and DTC with a fixed constraint length K=4. For supporting both turbo codes, BTC and DTC, the RDASTD needs different design strategies due to the different number of inputs and outputs required by BTC and DTC. A radix-4 (R4) BTC turbo decoding process [14, 20] is investigated; and it is found that the decoding

process can be exploited to implement the RDASTD. The high radix method is already employed by Viterbi decoder implementations in order to achieve high throughput [114-117]. The high radix method can also be used to increase turbo decoder throughput.

When the turbo decoding process for DTC is compared to the R4 process, it shows that many hardware resources can be shared for decoding both turbo codes. The complexity of a R4 based turbo decoder is clearly higher than a general radix-2 (R2) turbo decoder. However, the overhead led by the R4 method might be traded off with the compatibility and flexibility of wireless systems. Detailed simulation results are provided to compare with the performance of the ASIC turbo decoder implementations.

This chapter is organized as follows. Section 7.2 describes the mapping method for the reconfigurable implementations. The REASIC turbo decoder architecture and the incorporated components are illustrated in Section 7.3. Section 7.4 shows the R4 and DTC decoding methods used to implement the RDASTD. The detailed architecture and structure of each component are explained in Section 7.5. Simulation results in terms of area and power compared with ASIC turbo decoders are given in Section 7.6. Section 7.7 summarizes the chapter.

## 7.2 Mapping Method for Reconfigurable Turbo Decoder Implementation

The reconfigurable turbo decoder, presented in this chapter, is designed based on the MLMAP decoder architecture for constraint length K=5. The units for computing the state metrics and LLR are the main components in the reconfigurable turbo decoder implementations. They might be reconfigured for constraint lengths from K=3 to 5. A mapping method in this work is to reallocate the forward and backward state metrics based on constraint length K=5. This method can be described with the trellis state diagram for K=5 turbo codes.

Figures 7.1 and 7.2 illustrate the trellis diagrams of the forward and backward processes for K=5, respectively. There are 16 states given by $2^{K-1}$ for constraint length K=5. The arrow

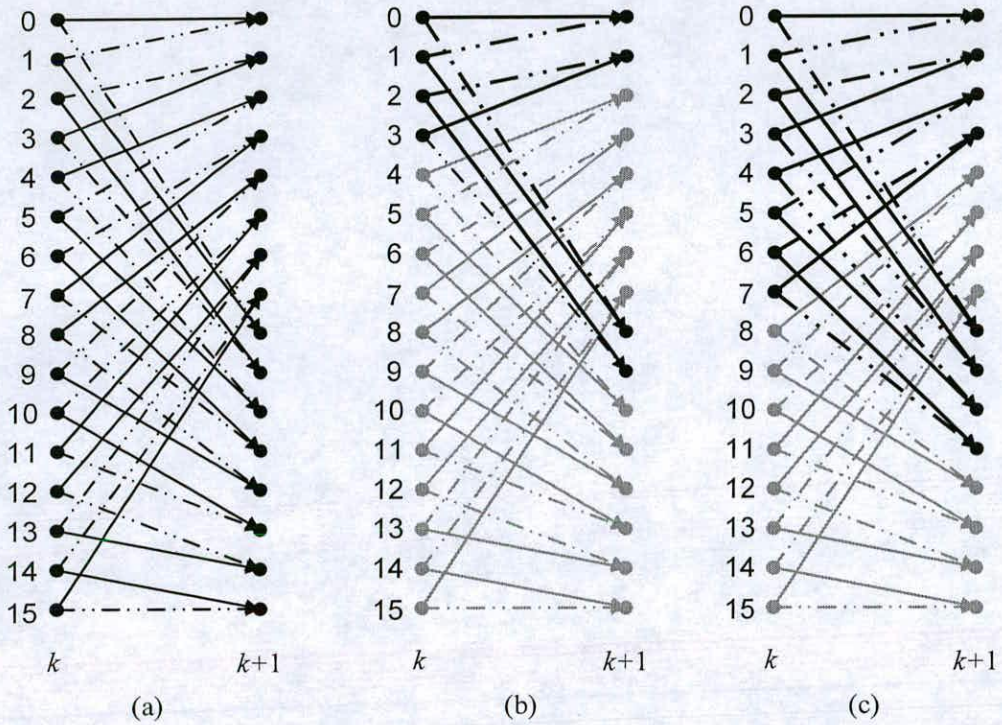



Figure 7.1 The forward process trellis diagrams for (a) K=5, (b) K=3 based on K=5, and (c) K=4 based on K=5.

indicates the process direction in different line styles corresponding to a systematic bit of the code words. Figures 7.1 (a) and 7.2 (a) are the trellis diagrams for K=5. Figures (b) and (c) show the state transitions in a bold line as required by the configurations for K=3 and 4 based on the K=5 trellis state diagram.

In the case of fixed constraint length turbo decoder implementations, the number of state metrics does not change. However, the reconfigurable turbo decoder is designed for reallocating the output state metrics before computing the new state metrics based on K=5 turbo decoder. For instance, when the constraint length is set to K=3, four states of the 16 states are needed for the turbo decoding process, as illustrated with the bold lines in Figures 7.1 (b) and 7.2 (b) for the forward and backward processes. Then, the output state metrics for each process are reordered as follows.

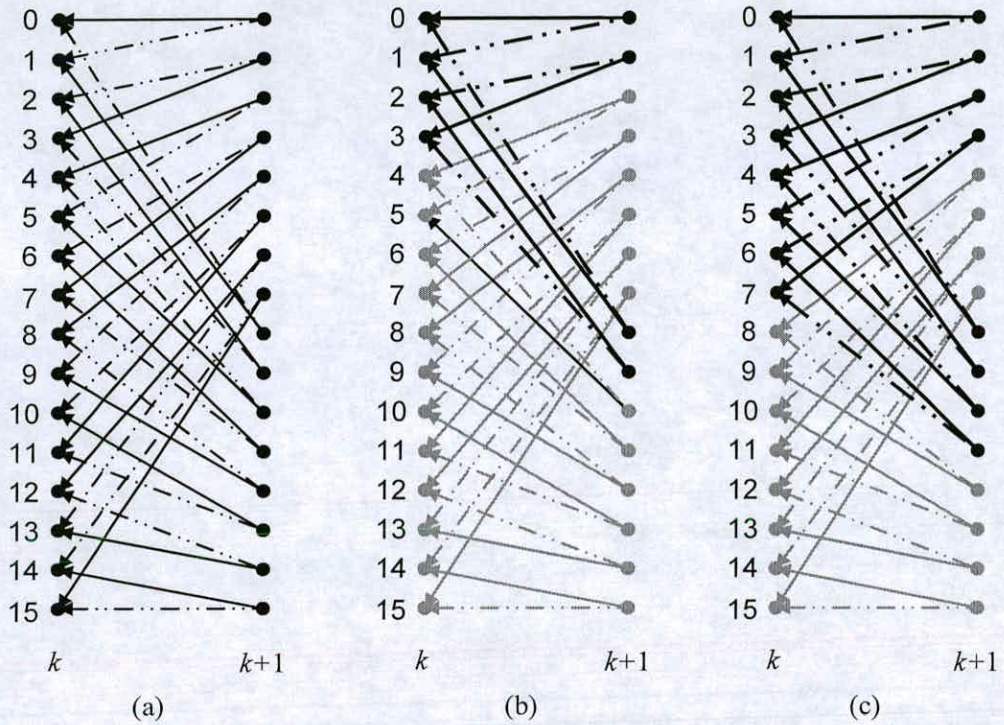



Figure 7.2 The backward process trellis diagram for (a) K=5, (b) K=3 based on K=5, and (c) K=4 based on K=5.

$$\alpha_0(k+1) \rightarrow \alpha_0(k+1), \quad \alpha_1(k+1) \rightarrow \alpha_1(k+1), \tag{7.1}$$

$$\alpha_8(k+1) \rightarrow \alpha_2(k+1), \quad \alpha_9(k+1) \rightarrow \alpha_3(k+1),$$

$$\text{Others} \rightarrow 0$$

In a similar way, the state metrics for the backward process is represented as follows:

$$\beta_0(k+1) \rightarrow \beta_0(k+1), \quad \beta_1(k+1) \rightarrow \beta_1(k+1), \tag{7.2}$$

$$\beta_2(k+1) \rightarrow \beta_8(k+1), \quad \beta_3(k+1) \rightarrow \beta_9(k+1),$$

Others $\rightarrow 0$

While the state metrics of states 8 and 9 are input to state metrics 2 and 3 for the forward state metric computation, the state metrics are input to the state metrics 8 and 9 for the backward state metric computation. Four state metrics are only passed to LCU in order to calculate LLR.

On the other hand, of the 16 state metrics, the mapping method for K=4 reallocates eight state metrics. The mapping can be represented for the forward and backward processes as follows:

Mapping forward state metrics for K=4:

$$\alpha_0(k+1) \rightarrow \alpha_0(k+1), \quad \alpha_1(k+1) \rightarrow \alpha_1(k+1), \qquad (7.3)$$

$$\alpha_3(k+1) \rightarrow \alpha_2(k+1), \quad \alpha_4(k+1) \rightarrow \alpha_3(k+1),$$

$$\alpha_8(k+1) \rightarrow \alpha_4(k+1), \quad \alpha_9(k+1) \rightarrow \alpha_5(k+1),$$

$$\alpha_{10}(k+1) \rightarrow \alpha_6(k+1), \quad \alpha_{11}(k+1) \rightarrow \alpha_7(k+1),$$

Others $\rightarrow 0$

Mapping backward state metrics for K=4:

$$\beta_0(k+1) \rightarrow \beta_0(k+1), \quad \beta_1(k+1) \rightarrow \beta_1(k+1), \qquad (7.4)$$

$$\beta_2(k+1) \rightarrow \beta_2(k+1), \quad \beta_3(k+1) \rightarrow \beta_3(k+1),$$

$$\beta_4(k+1) \rightarrow \beta_8(k+1), \quad \beta_5(k+1) \rightarrow \beta_9(k+1),$$

$$\beta_6(k+1) \rightarrow \beta_{10}(k+1), \quad \beta_7(k+1) \rightarrow \beta_{11}(k+1),$$

Others $\rightarrow 0$

Following the mapping methods described above, the output state metrics are computed after reordering their state numbers when the REASTD is configured for K=3 and 4.

## 7.3 Reconfigurable Turbo Decoder Hardware Architecture for Variable Constraint Lengths

This section presents the reconfigurable application specific turbo decoder (REASTD) architecture for supporting variable constraint length. A detailed structure of the units incorporated in the architecture is described. To save on power consumption of the REASTD hardware, gated clock is used to implement a reconfigurable state metric computation unit (RESMCU) and a reconfigurable LLR computation unit (RELCU).

### 7.3.1 Reconfigurable Turbo Decoder Architecture

The REASTD architecture is illustrated in Figure 7.3. The incorporated components are almost same with the MLMAP architecture presented in Chapter 3. The REASTD is implemented based on K=5 to support K=3 and 4 as well. It consists of three reconfigurable metric computation units for the forward (FREMCU), backward (BREMCU), and dummy-backward state metrics (DREMCU), a reconfigurable LLR computation unit (RELCU) to calculate LLR for K=3, 4, and 5, the memory blocks for a data scheduling unit (DSU) and delaying for the soft-input and the forward state metrics. The reconfigurable units can be configured for the mode (M), and operate with a gated clock (GC) to save on power consumption. In addition, the memory block, DF, for storing the forward state metrics is designed with 16 parallel LIFO blocks with GC. The 16 parallel LIFOs are for K=5, but 4 or 8 of them are used if K=3 or 4. The unused LIFO blocks are deactivated by the GC.

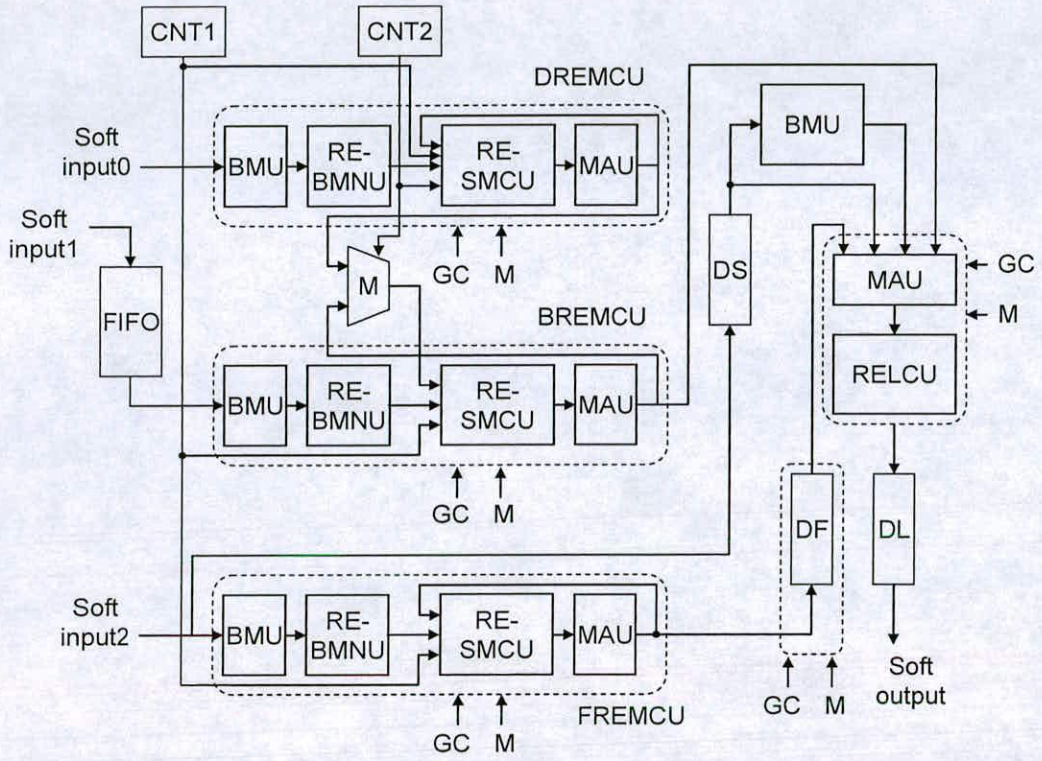



Figure 7.3 The reconfigurable turbo decoder architecture.

## 7.3.2 Clock Gating Method for Reconfigurable Turbo Decoder

It is well known that a clock gating method [140-141] is very useful for saving on power consumption. Figures 7.4 (a) and (b) illustrate an example of the use of the GC, and the GCs applied to the REASTD implementation, respectively. As shown in Figure 7.4 (a), a system clock is input to a register through an AND gate. If the register does not need to be activated, the 'Enable' input of the AND gate is set to '0' and hence the system clock is blocked.

Figure 7.4 (b) illustrates how the gated clocks are generated for the REASTD. Three gated clocks ($GC_0$, $GC_1$, and $GC_2$) are used in the REASTD architecture. The mode (M)

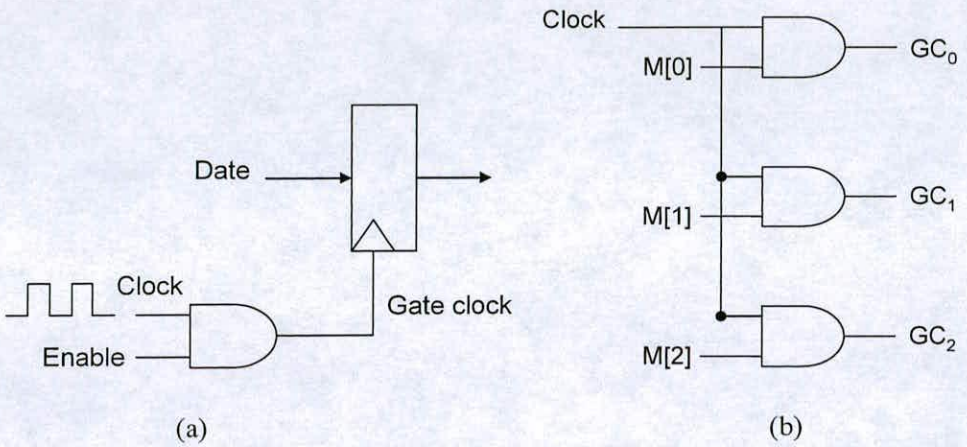



Figure 7.4 (a) Clock gating method. (b) The gated clocks applied to the reconfigurable architecture.

corresponding to the Enable in Figure 7.4 (a) is determined by the constraint length and represented in 3-bits. Thus, when K is 3, the least significant bit of M, M [0], is '1' that enables $GC_0$. When K is 4, M [1:0] enables $GC_0$ and $GC_1$. When K is 5, all GC signals are enabled.

### 7.3.3 Reconfigurable Metric Computation Unit

The reconfigurable metric computation unit (REMCU) consists of a BMU, a reconfigurable branch metric normalization unit (REBMNU), a reconfigurable state metric computation unit (RESMCU), and a mapping unit (MAU), as illustrated in Figure 7.5. It is used to compute the forward and backward state metrics. The soft-input data is fed into BMU to generate the branch metrics (BM). The BM is normalized by REBMNU with the state metrics (SM) providing by MAU. The normalized branch metrics (NBM) and the Recursive SM are passed to RESMCU for computing the next state metrics. The REBMNU is almost the same as the BMNU introduced in Chapter 4. The difference is that the number of SM inputs for determining the normalizing condition is controlled by the configuration of the constraint

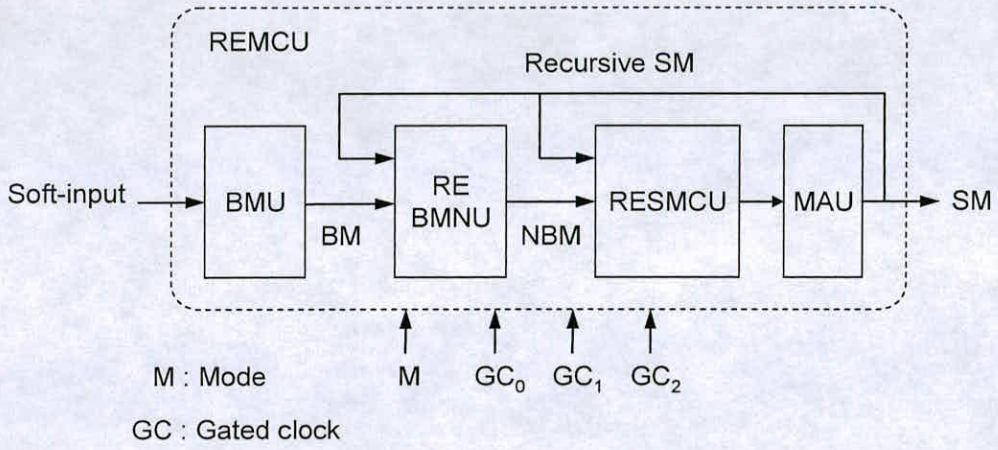



Figure 7.5 The reconfigurable metric computation unit structure.

length. For example, if REBMNU is configured for K=3, REBMNU accepts only four SM values.

RESMCU that is implemented based on K=5 consists of 16 ACSs in parallel and a MAU. The ACS structure is the same as the one used in Chapter 4. It operates with the mode (M) and three gated-clocks ($GC_0$, $GC_1$, $GC_2$) associated with the constraint length.

Figures 7.6 and 7.7 illustrate the configurations of RESMCU and MAU for computing the forward and backward SM, respectively. These figures are given as an example when they are configured for K=3. As described in the previous subsection, when K=3, $GC_0$ is enabled only, while the other two gated-clocks are blocked. In this case, four of the 16 ACS units are activated with $GC_0$, as illustrated in Figure 7.6. Thus, when K=3, the rest of the ACS units do not operate and their inputs are set to zero. In the figure, the unused ACSs are represented in grey coloured blocks. As can be seen, ACS 0, 1, 8, and 9 are activated by $GC_0$ and M [0]. Outputs of the four ACSs are reordered as $r\alpha_0$, $r\alpha_1$, $r\alpha_2$, and $r\alpha_3$ by MAU before recursively being input to the four ACS 0, 1, 8 and 9 of the RESMCU. The reordering method that is described in the previous section follows.

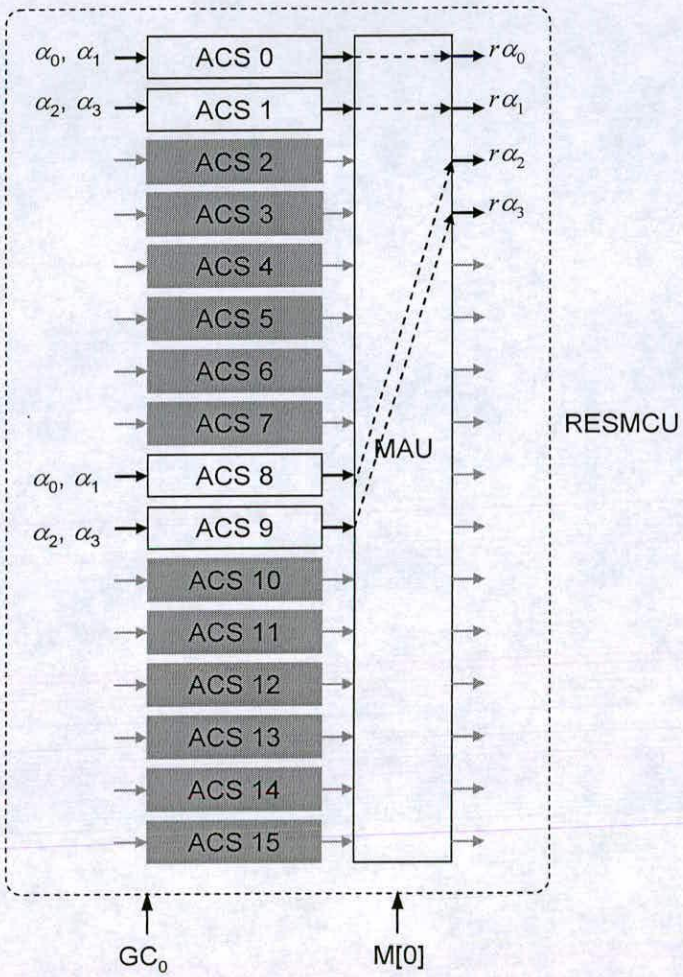



Figure 7.6 The reconfigurable state metric unit configuration for the forward process and K=3.

Figure 7.7 illustrates the configuration for the backward process that is performed in a similar way with the process illustrated in Figure 7.6. As before, $GC_0$ and M [0] are used to operate the four ACSs, 0, 1, 2, and 3. The rest of the ACSs are deactivated. Outputs of the four ACSs are reordered as $r\beta_0$, $r\beta_1$, $r\beta_8$, and $r\beta_9$ through the MAU. This backward SM reordering method is described in Section 7.2. The dummy-backward process is performed in the same way with the backward process.

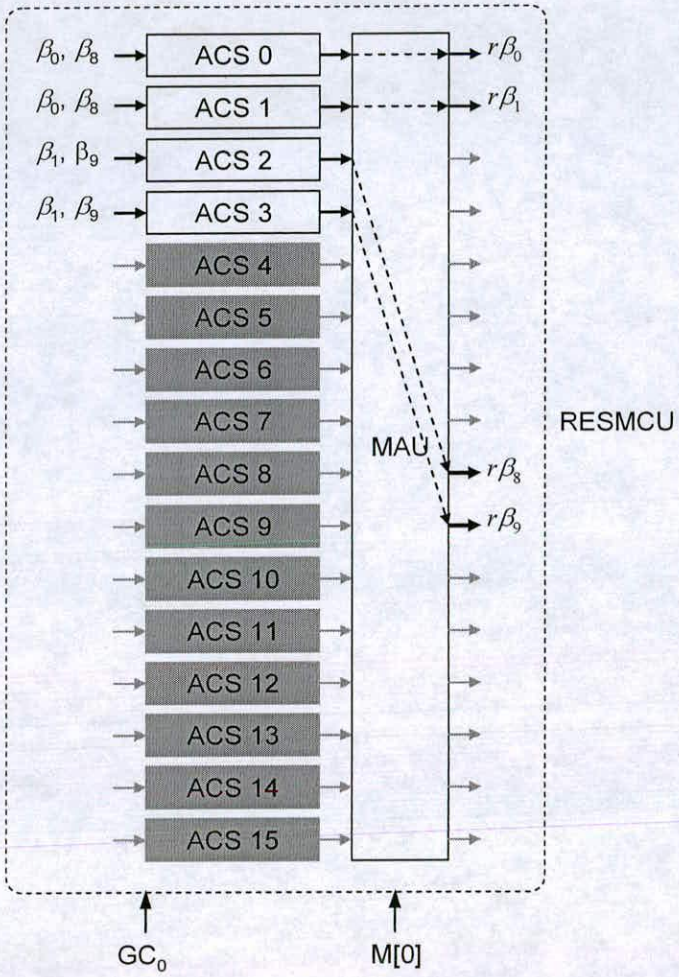



Figure 7.7 The reconfigurable state metric unit configuration for the backward process and K=3.

The configurations of the RESMCU for the constraint length K=4 can also be represented by using the method illustrated in Figure 7.6 and 7.7 with the mapping method described in section 7.2.

### 7.3.4 Reconfigurable Log-Likelihood Computation Unit

A reconfigurable log-likelihood computation unit (RELCU) is also designed based on the constraint length K=5 to support the constraint length K=3 and 4, as well. A gated-clock method is applied to the RELCU implementation to save power consumption as in the RESMCU implementation. The RELCU structure is illustrated in Figure 7.8, which consists of two mapping units (MAU) and two main components, LCU1 and LCU0, to generate 1- and 0- bit LLR values, $L_{lr1}$ and $L_{lr0}$. The extrinsic information is obtained from the differences of $L_{lr1}$ and $L_{lr0}$ after subtracting the soft-input. The forward and backward state metrics (FSM and BSM) and the branch metrics (BM) are input to RELCU, and then, they are distributed by MAU following the mode (M) that is associated with the constraint length. The three gated-clocks, $GC_0$, $GC_1$, and $GC_2$, are input to each of the LCU0 and LCU1 depending on M to save on the power consumption.

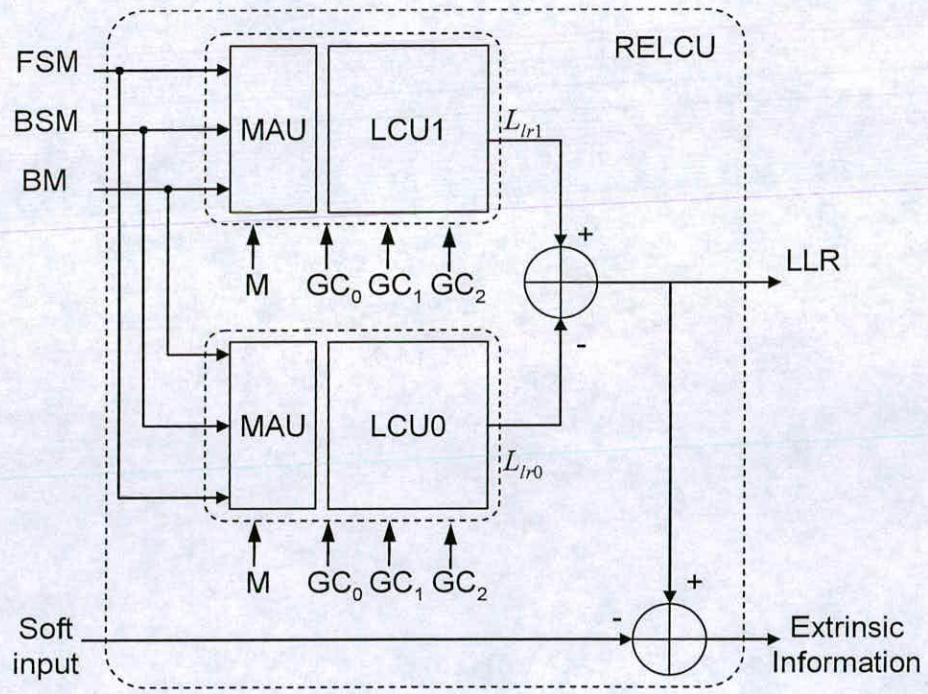



Figure 7.8 The reconfigurable LLR computation unit structure.

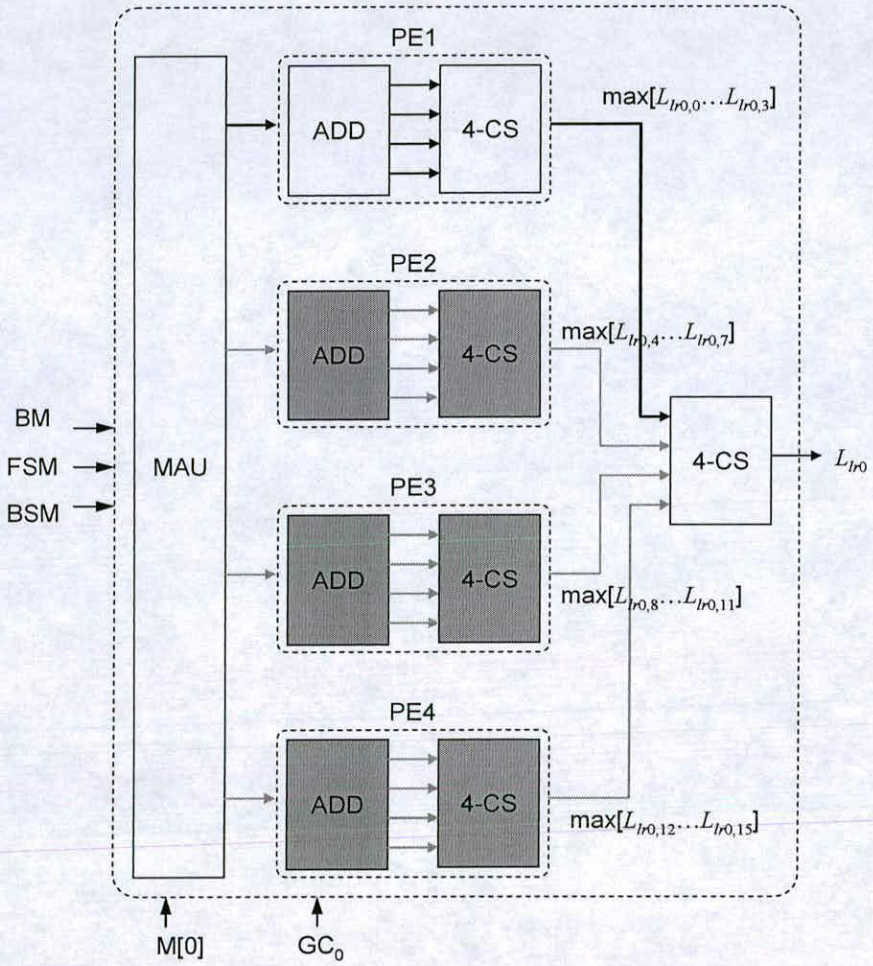



Figure 7.9 The reconfigurable $L_{lr0}$ or $L_{lr1}$ computation unit structure.

As an example, Figure 7.9 illustrates a structure that can be used for LCU0 or LCU1 with MAU. The figure is shown when it is configured for K=3. The process for computing the 1- or 0-bit LLR value is no different from that described in Chapters 3 and 4. In the structure, the four process elements (PE) are activated by the three gated-clocks. Each PE computes 4 1- or 0-bit LLR values from the input FSM, BSM, and BM, and then the LLR values are compared by 4-input compare-select (4-CS) to determine the maximum value, as represented by max[$L_{lr0,0}$...$L_{lr0,3}$] in PE1. The outputs of the four PEs are compared again to select the

final LLR value. However, when the LCU is configured for K=3, only PE1 is activated and its results are output through the 4-CS as the final value $L_{lr0}$, as shown in Figure 7.9, in which the grey coloured PEs are deactivated.

# 7.4 Turbo Decoder Implementation Methods for Binary and Double-Binary Turbo Codes

## 7.4.1 Radix-4 Turbo Decoding Method for Binary Turbo Codes

Radix-4 and higher radix decoders for convolutional codes were developed in order to achieve high throughput at the expense of increased implementation complexity [114]. An R4 turbo decoder VLSI implementation for BTC is described in [118]. Similarly, as with the decoder for convolutional codes, R4 turbo decoder processes the input data at times $k$ and $k$+1 concurrently. Figures 7.10 (a) and (b) illustrate R2 and R4 trellis diagrams for BTC with a constraint length K=4. In Figure 7.10 (b), four transition paths converge into one state,

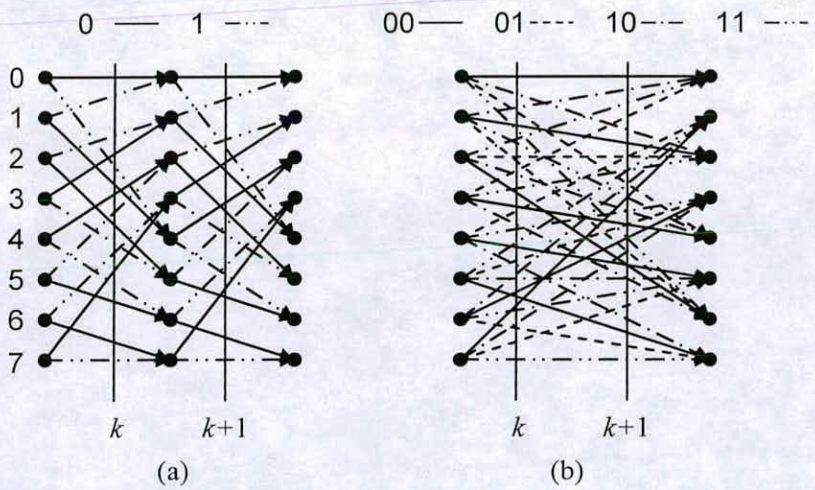



Figure 7.10 (a) Radix-2 and (b) radix-4 based trellis diagram for K=4 binary turbo codes.

represented in different line styles, which indicate two systematic bits of the code words at times $k$ and k+1. Thus, a state metric for each state is obtained from the four branch metrics calculated from the soft-input data at times $k$ and $k$+1, and the four state metrics. Therefore, the number of the state metrics for forward and backward processes is reduced to half of the number of the state metrics computed by the R2 turbo decoder implementations, which are the MLMAP turbo decoders described in previous chapters. However, two LLRs and extrinsic information must be obtained from the half number of the state metrics and the branch metrics generated at times $k$ and $k$+1. The two LLRs are concurrently calculated as follows:

$$L_{lr}(k) = \max[L_{lr11}(k,k+1), L_{lr10}(k,k+1)] - \max[L_{lr00}(k,k+1), L_{lr01}(k,k+1)] \quad (7.5)$$

$$L_{lr}(k+1) = \max[L_{lr11}(k,k+1), L_{lr01}(k,k+1)] - \max[L_{lr00}(k,k+1), Llr_{10}(k,k+1)] \quad (7.6)$$

where $L_{lr11}$, $L_{lr00}$, $L_{lr01}$, and $L_{lr10}$ are obtained from the state and branch metrics of the

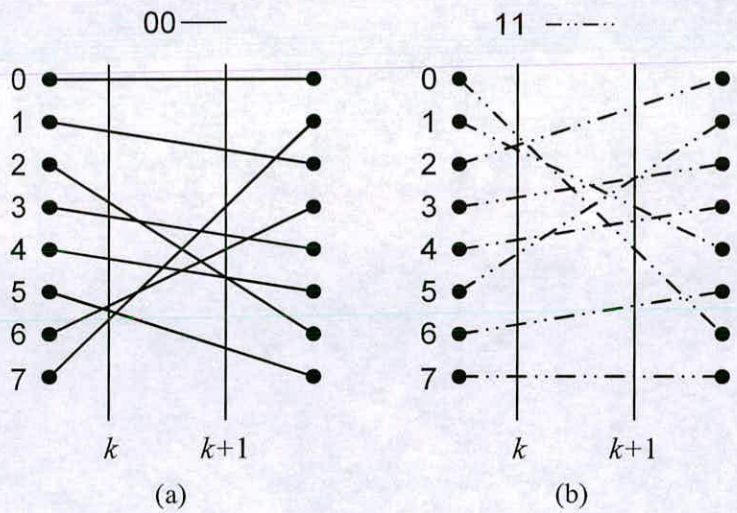



Figure 7.11 The trellis paths for computing (a) $L_{lr00}$ and (b) $L_{lr11}$ based on radix-4 binary turbo codes for K=4.

transition paths represented by the systematic bits of the code words, 11, 00, 01, and 10, respectively. Figures 7.11 (a) and (b) are given as an example to obtain $L_{lr00}$ and $L_{lr11}$, which are the maximum values of the LLRs calculated from the eight transition paths, respectively. The other two LLRs can be obtained using the same method. Then, the final LLR values in equations (7.5) and (7.6) can be obtained. The extrinsic information at times $k$ and $k$+1 can be provided to the next decoder by subtracting the systematic input symbol and the extrinsic information of each time from the LLR values obtained from equations (7.5) and (7.6).

### 7.4.2 Double-Binary Turbo Codes Decoding Method

As described in Chapter 2, the DTC is generated from a double-binary input data stream. Many papers have researched the turbo decoder implementations for BTC. However, the turbo decoders for DTC are relatively less considered in the literature from the hardware implementation point of view. Thus, in this work the DTC turbo decoding algorithm has been investigated. This investigation showed that the algorithm needs a different strategy from the BTC turbo decoding algorithm.

Basically, the same components are used to implement BTC and DTC turbo decoders. However, due to using different techniques in the encoding process, the initial and final state metrics for the forward and backward processes need to be treated differently in the

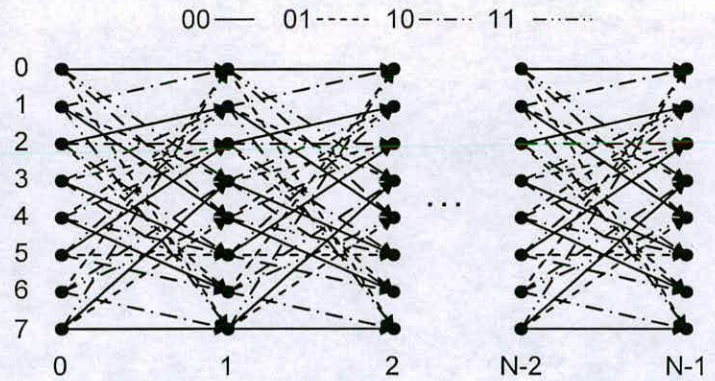



Figure 7.12 The forward process trellis diagram for K=4 double-binary turbo codes.

implementations, which require modifying the design of the state metric computation unit (SMCU).

The decoding algorithm for DTC has similarities, as shown in the trellis diagrams in Figure 7.12. Thus, calculating the branch and state metrics for the DTC decoding process is almost the same as that of the R4 BTC decoding process. However, while the initial state metrics for forward and backward processes for BTC are already known at the decoder side, due to using CRSC by its encoder, as described in Chapter 2, they may be determined and updated for DTC during the decoding process. For these processes, the initial state metrics for the forward and backward processes are initialized to zero at the first iteration as follows:

$$\overline{\alpha}(s_0) = 0, \ \overline{\beta}(s_{N-1}) = 0 \quad \textit{for all states} \tag{7.7}$$

Then, the final state metrics for the forward process and the backward process, $\overline{\alpha}(s_{N-1})$ and $\overline{\beta}(s_0)$, are updated to the initial state metrics in the next decoding process. This happens at every iteration for updating the new initial state metrics. Therefore, the final state metrics need to be accumulated until the next decoding process starts for the updating.

Then, two LLRs are computed using the same method described in (7.5) and (7.6). In addition, two extrinsic information data can be obtained as follows:

$$L_{e2}^{A}(k) = L_{lr}^{A}(k) - L_c y_k^{A} - L_{e1}^{A} \tag{7.8}$$

$$L_{e2}^{B}(k) = L_{lr}^{B}(k) - L_c y_k^{B} - L_{e1}^{B} \tag{7.9}$$

where $A$ and $B$ are the systematic bits as shown in Figure 2.4 in Chapter 2. Therefore, each set of extrinsic information, $L_{e2}^{A}$ and $L_{e2}^{B}$, corresponds to the input systematic symbols. This process is the same with the method used in the R4B turbo decoding process described in the previous subsection.

As can be seen, a R4 BTC turbo decoder structure can be exploited to implement the RDASIC turbo decoder that supports BTC and DTC.

# 7.5 Reconfigurable Turbo Decoder Hardware Implementations for BTC and DTC

## 7.5.1 Reconfigurable Turbo Decoder Architecture

In this section, the RDASTD architecture for R4B and DTC is described based on the architecture described in Figure 4.1 of Chapter 4. Figure 7.13 illustrates the reconfigurable architecture that consists of three R4 BTC and DTC metric computation units for forward (FRDMCU), backward (BRDMCU), and dummy-backward processes (DRDMCU), an R4 BTC and DTC LLR computation unit (RDLCU), memory blocks for data scheduling (DSU) and delaying the soft-input (DS), the forward state metrics (DF), and the soft-output (DL). These memory blocks can be realized by FIFO and LIFO blocks. The depth of the LIFO and

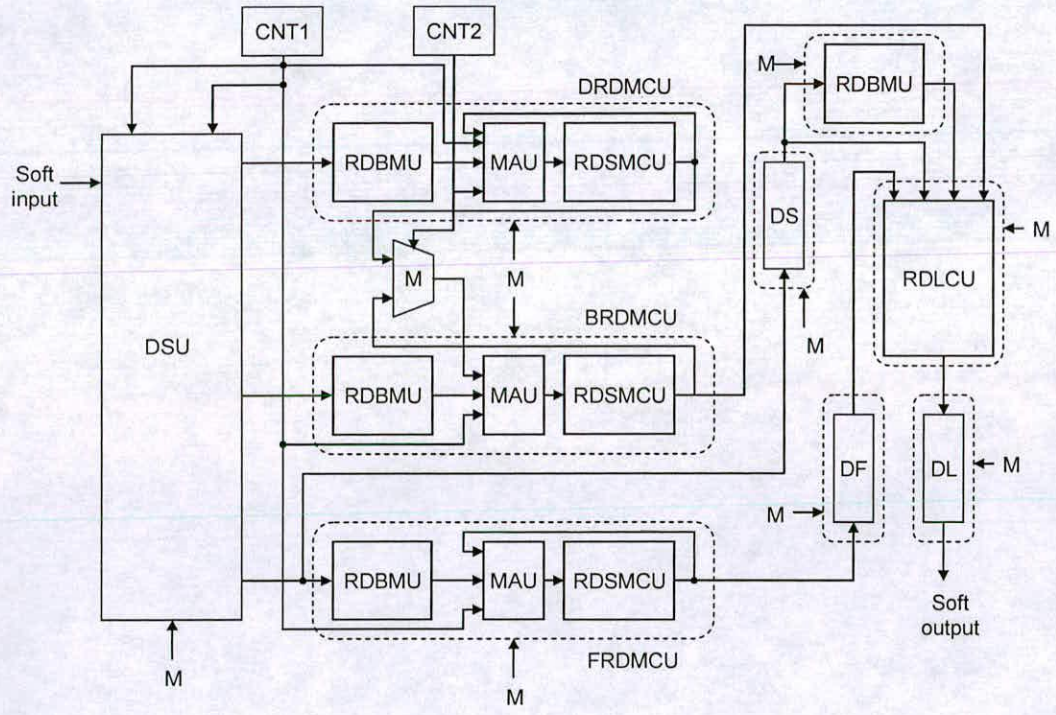



Figure 7.13 The reconfigurable turbo decoder architecture for radix-4 and double-binary turbo codes.

FIFO is related to the window size. In this architecture, the window size is determined by the minimum block size of 3GPP for BTC and IEEE 802.16 for DTC. Thus, the memory block size can be configured following the input mode (M) for BTC and DTC. The computational logic components, RDMCU and RDLCU, are designed for supporting the R4 BTC and DTC decoding process. The delay for the branch metrics input to RDLCU is replaced by an additional R4 BTC and DTC BMU (RDBMU) for saving area. All the components are configured by the mode (M), either for R4 BTC or DTC. Detailed implementation methods of each component are described in the following sections.

### 7.5.2 Reconfigurable Metric Computation Unit for Radix-4 BTC and DTC

Figure 7.14 illustrates the R4 BTC and DTC MCU (RDMCU) structure that consists of a reconfigurable R4 BTC and DTC branch metric unit (RDBMU) and a R4 BTC and DTC state metric computation unit (RDSMCU) that incorporates a mapping unit (MAU) and a parallel R4 BTC and DTC add-compare-select-normalization (RDACSN). The mode (M) input to RDMCU determines the algorithm (BTC or DTC) to be implemented. Before starting the decoding process, each component is configured for the selected mode. Then, MAU distributes the branch metrics (BM) obtained by RDBMU and the state metrics generated by RDSMCU in the previous process to calculate the new SMs.

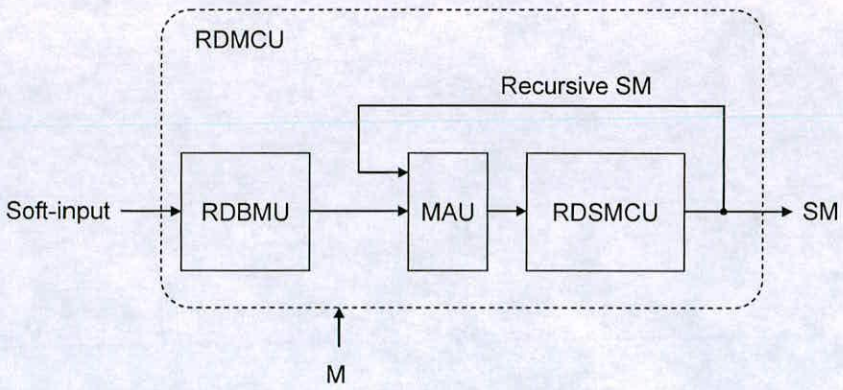



Figure 7.14 The reconfigurable metric computation unit structure.

### 7.5.2.1 R4 BTC and DTC Branch Metric Unit

Figure 7.15 illustrates the RDBMU structure used to implement the RDASTD architecture with R4 BTC and DTC. This figure is given as an example for R4 BTC. As can be seen, due to the high radix decoding process, the implementation complexity is clearly higher than the BMN implemented in previous chapters. While the BMU used in the previous MLTBD architectures generates the two branch metrics of the required four branch metrics, the RDBMU provides the eight branch metrics of the required 16 branch metrics from the six inputs as illustrated in Figure 7.15. The other eight metrics can simply be obtained by negating the outputs of RDBMU. Generally, the branch metric computation process does not significantly affect the overall turbo decoder performance in terms of area usage and power consumption. However, it was found that different weighting for the soft-input data and extrinsic information is needed for R4 BTC and DTC in order to achieve better BER

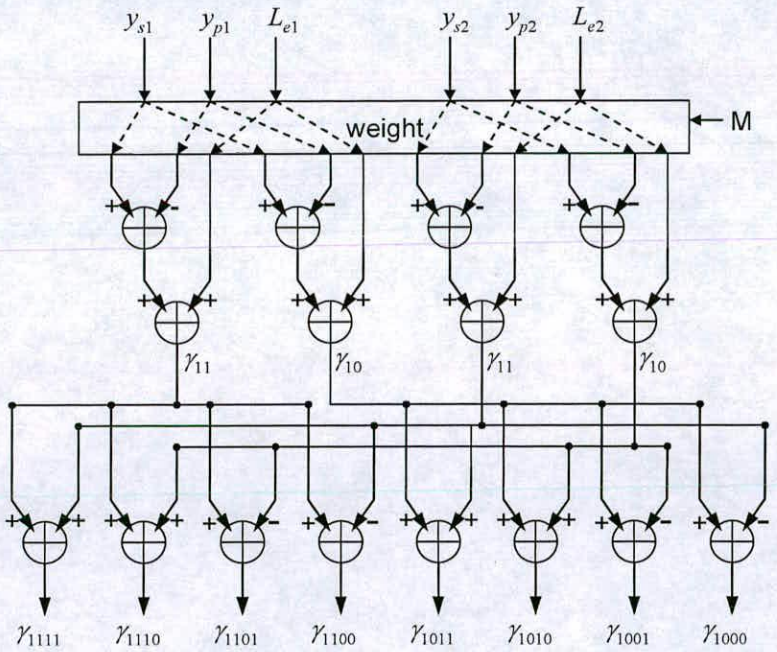



Figure 7.15 The branch metric unit structure for R4 BTC and DTC.

performance. Therefore, the weighting relies on the encoding style and it is implemented with shift registers.

#### 7.5.2.2 R4 BTC and DTC Add Compare Select Normalization Unit

In common turbo decoder implementations, an ACSN unit is regarded as one of the key components. Here, the ACSN for RDSMCU is separately described by whether it is for either R4 BTC or DTC. As an example, Figures 7.16 (a) and (b) show a block diagram of ACSN for R4 BTC and DTC, respectively. In Figure 7.16 (a), BM, SM, and dummy state metric (DSM) generated by DRDMCU shown in Figure 7.14 are input to ACSN. While SM is input recursively to ACSN through a multiplexer (MUX), the input recursive SM is updated in DSM at the beginning of every sub frame during the decoding process following the number of the counters (CNT1 and CNT2).

On the other hand, the ACSN illustrated in Figure 7.16 (b) requires an accumulator (ACM). All the functions and the data input sequence of Figure 7.16 (b) are the same as with Figure 7.16 (a). However, as already described in Section 7.4, while the initial state metrics for BTC

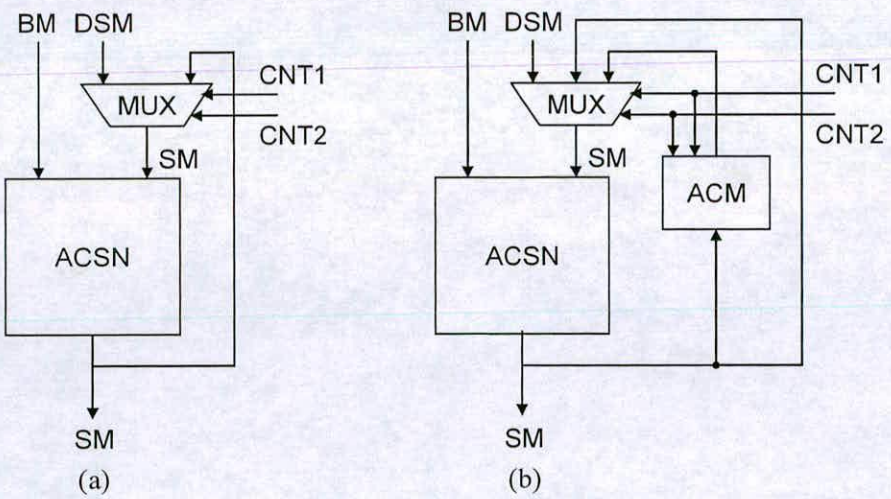



Figure 7.16 The add-compare-select-normalization block diagram for (a) binary and (b) double-binary turbo codes.

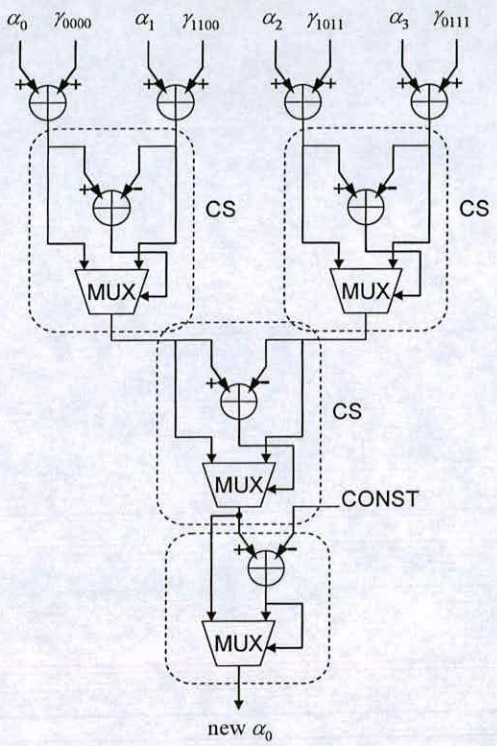



Figure 7.17 The raxid-4 based add-compare-select-normalization structure.

are known at the decoder side, the initial state metrics for DTC must be found for every iterative process after being initialized with zero for the first iteration. The initial state metrics are then updated when the next iteration is started. The updating takes place once in every frame. Therefore, the ACM is for storing the state metrics used for the initial state metrics in the next iteration process.

Figure 7.17 is a conventional ACSN structure, which can be used for R4 BTC and DTC. This structure is given as an example for calculating a state metric of state 0 for R4 BTC. The input branch and state metrics must be altered as they are for DTC. These inputs are controlled by MAU in Figure 7.14. Four branch ($\gamma_{0000}$, $\gamma_{1100}$, $\gamma_{1011}$, $\gamma_{0111}$) and state ($\alpha_0$, $\alpha_1$, $\alpha_2$, $\alpha_3$) metrics are added by using four adders and, then, the results of each adder is repeatedly compared for selecting the maximum state metric. The selected maximum state metric is

normalized by subtracting a constant (CONST) value to prevent overflows, as the state metric is larger than the constant. Finally, 'new $\alpha_0$' is generated and is recursively input to the ACSN itself.

It is well known that ACSN is the bottleneck in a high-speed turbo decoder and the critical path delay of a turbo decoder is in ACSN as was discussed in Chapter 3. The critical path delay, $d$, of the structure illustrated in Figure 7.17 can be given as follows:

$$d = 4 \times t_a + 3 \times t_m \tag{7.10}$$

where $t_a$ is the adder delay and $t_m$ is the multiplexer delay. In order to achieve a high-speed turbo decoder, the delay must be reduced. The retiming method used in [154] may be

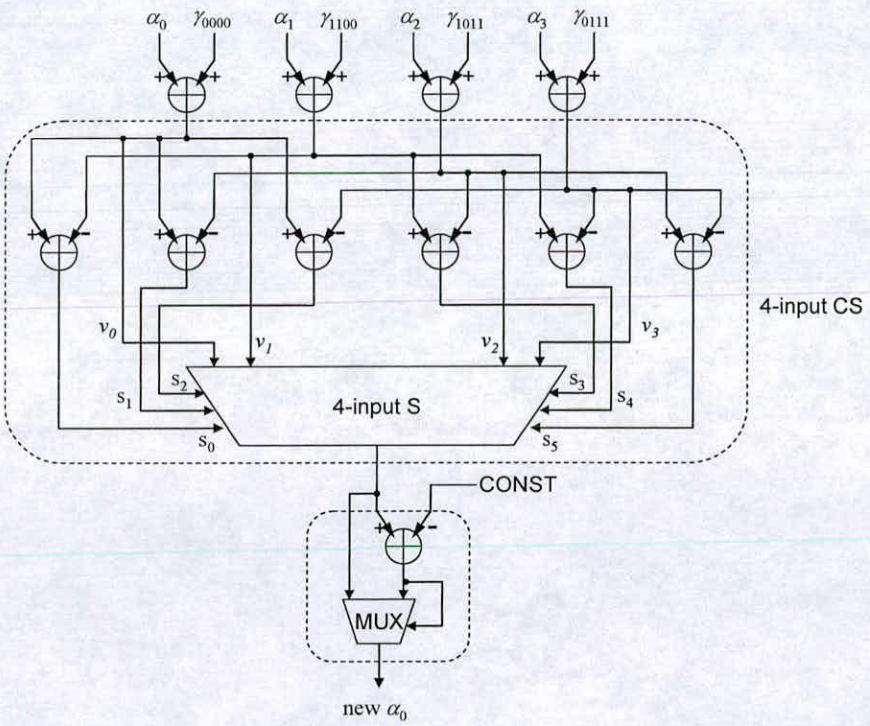



Figure 7.18 The radix-4 add-compare-select-normalization structure with 4-input compare select unit.

applied to the ACSN to reduce the delay. However, this method is not suitable for implementing the ACSN for R4 BTC and DTC due to large overheads in area usage and power consumption led by the method.

Figure 7.18 illustrates another ACSN structure for improving the critical path delay. Instead of the three CS stages in Figure 7.17, a 4-input CS introduced in Chapter 4 is employed in Figure 7.18. As can be seen, the four sets of results corresponding to the outputs of the first adder set are compared with each other at the same time by using six adders. Then, the six most significant bits (MSB), $s_{0\text{-}5}$, generated by the six adders, are used to select the maximum state metric as described in Chapter 4. Thus, the delay of the ACSN in Figure 7.18 is

$$d = 3 \times t_a + t_m + t_m' \tag{7.11}$$

where $t_m'$ is the delay of 4-input selector, which may be a little longer than $t_m$. With the 4-input CS, it is shown that the ACSN delay in (7.11) is roughly reduced by the delay of one adder and one MUX. However, the use of 4-input CS increases the number of adders that may lead to area and power overhead. The hardware performance between the two ACSN implementations is evaluated and compared in later sections.

### 7.5.3 R4 BTC and DTC Log-Likelihood Computation Unit

All of the branch metrics and the forward and backward state metrics generated by each RDMCU are used to calculate the LLR values. Figure 7.19 illustrates a block diagram of R4 BTC and DTC LLR computation unit (RDLCU) that consists of a mapping unit (MAP) for the distribution of the input branch metrics (BM), forward and backward state metrics (FSM and BSM), four sub LLR computation units (LLR00, 01, 10, and 11), and two LLR computation units (Sub-LCU0 and 1). The purpose of the mapping unit (MAU) is the same as that of the MAU shown in Figure 7.14. MAU reallocates the input BM, FSM, and BSM for computing LLR based on the selected operation mode (R4 BTC or DTC). After completing the distribution of all branch and state metrics, LLR00, 01, 10, and 11, calculate $L_{lr00}$, $L_{lr01}$, $L_{lr10}$, and $L_{lr11}$ following the method described in Section 7.4. These values are used to obtain two LLR values by Sub-LCU0 and Sub-LCU1 based on the methods in (7.6)

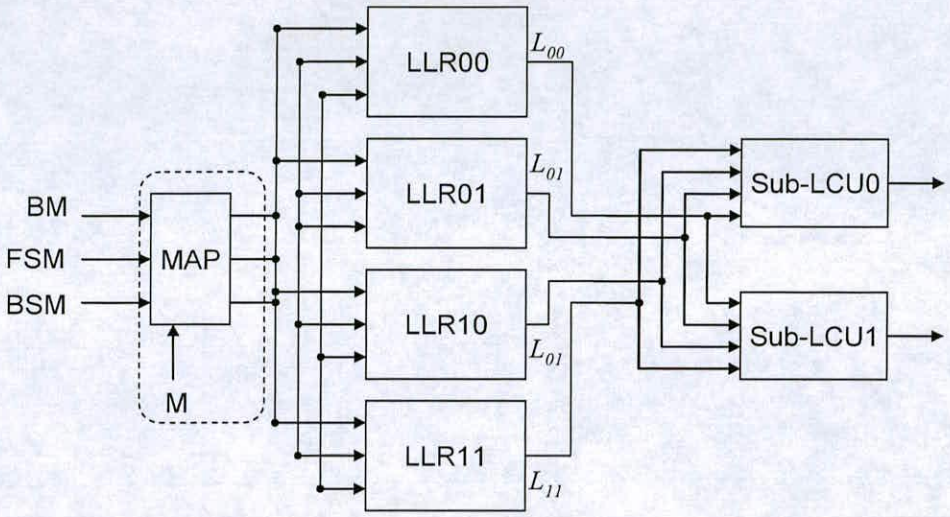



Figure 7.19 The LLR computation unit structure.

and (7.6). At the same time, the extrinsic information for the next iterative decoding process is also computed from the LLR values obtained by Sub-LCU0 and Sub-LCU1. The outputs of the RDLCU are represented by 9-bits, in which the most significant bit (MSB) represents the LLR and the other 8-bits represent extrinsic information. The MSB is the decoded bit information of the turbo decoder.

In Figure 7.11 (b), the trellis state diagram associated with computing $L_{lr11}$ for R4 BTC was given as an example, in which eight transition paths exist. Figure 7.20 illustrates a conventional structure of the LLR11 for computing $L_{lr11}$. Similarly, this structure can also be used to implement the sub-LCU00, 01, and 10 with different combinations of the input branch and state metrics. As shown in Figure 7.20, there are eight pairs of branch and state metrics corresponding to eight transition paths and forward/backward states. Note that the subscript of $\alpha_0$ and $\beta_0$ indicates state 0 of forward and backward state metrics. In addition, the subscript of $\gamma_{0000}$ denotes branch metric of 0000 code word generated by the encoder. In this case, after adding three input metrics, the $L_{11}$ is determined by selecting the maximum value by using seven CS units. In this structure, the critical path delay is

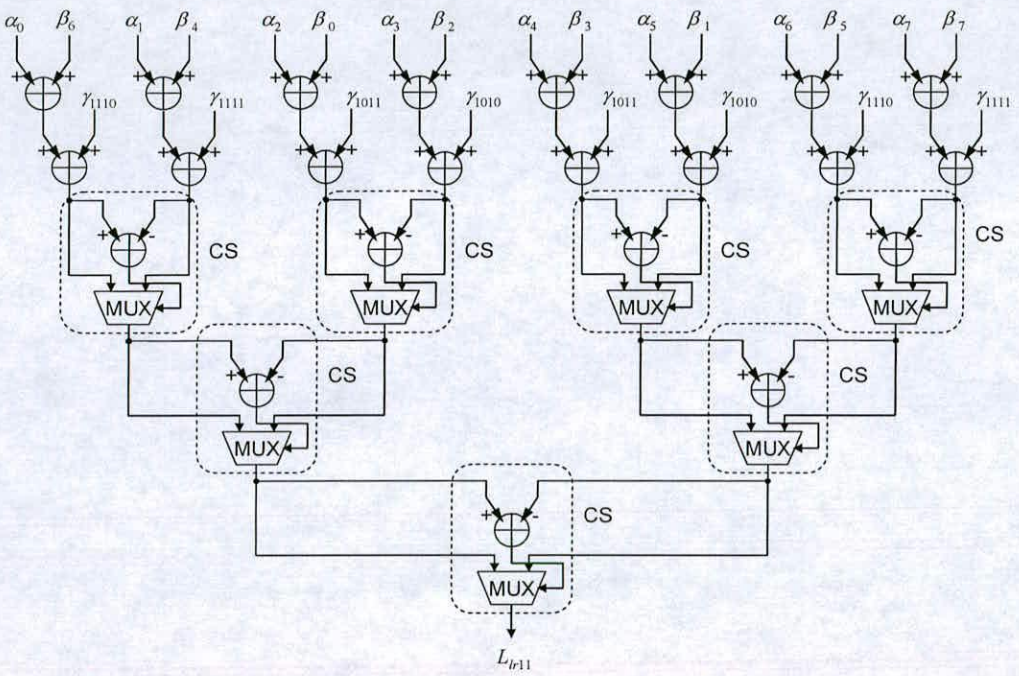



Figure 7.20 $L_{11}$ computation unit structure for radix-4 and double-binary turbo codes.

$$d = 5 \times t_a + 3 \times t_m \tag{7.12}$$

We can see that this delay, $d$, is longer than ACSN in (7.10). To reduce this critical path delay, the structure can be pipelined by inserting a set of registers. This cannot only reduce the critical path delay but also decrease the glitches. However, excessive pipelining may also lead to increased area usage and power consumption due to the insertion of registers.

Figure 7.21 illustrates an improved structure of Figure 7.20. When we look at the input branch metrics, we can see that the actual number of branch metrics is only four to calculate $L_{11}$. This is the same for computing $L_{01}$, $L_{10}$, and $L_{00}$ whether it is for BTC or DTC. This allows comparisons to be made before adding all the three metrics. Therefore, after adding forward and backward state metrics, the results are compared to select the maximum value and, at the same time, branch metrics are summed with the two state metrics. The four

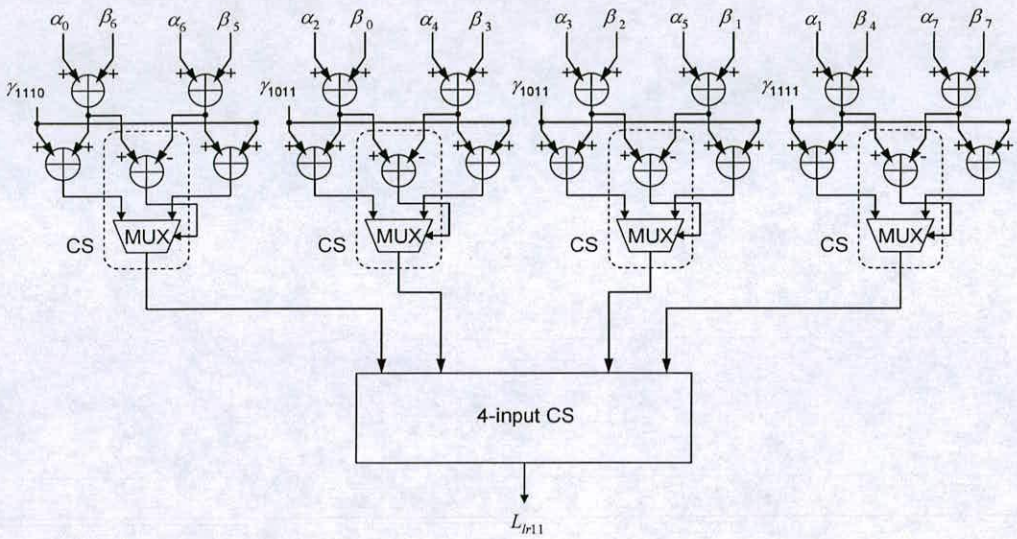



Figure 7.21 The $L_{11}$ computation unit structure with 4-input compare select.

outputs generated from the first stage comparisons are then used to compute the final output. Here, the 4-input CS as shown in Figure 7.18 is used to select the maximum LLR as the final output. The critical path delay of the structure in Figure 7.21 can be given as below:

$$d = 3 \times t_a + t_m + t_m' \tag{7.12}$$

Clearly, this delay is the same with (7.10). Therefore, the structure provides an efficient implementation for reducing the critical path delay without needing to employ pipelining. The structures of Figures 7.20 and 7.21 have been implemented and their performance compared in terms of area usage, power consumption, and delay, which are discussed in the next section.

## 7.6 Results

The reconfigurable ML-MAP turbo decoder has been designed at RTL using Verilog HDL and synthesized with the Synopsys DesignCompiler™ using the UMC 0.18μm standard CMOS cell library. RTL and gate level simulations were performed using Cadence Verilog-XL™. The switching activities generated from all circuit nets were obtained during the gate level simulation after eight numbers of iterations. The reconfigurable turbo decoder power consumption was evaluated with Synopsys DesignPower™ with the switching activity information generated by the gate-level simulation. These simulation processes were carried out at a clock frequency of 50MHz.

At first, Figure 7.22 illustrates the area results of the REASTD compared with the ASIC turbo decoder implementation results for each constraint length K=3, 4, and 5, which were given in Chapter 4. The primary and secondary axes in the graph indicate breakdown components and total area results. In the figure, K3A implies the K=3 MLMAP turbo decoder implemented as an ASIC. The comparisons of the area results are also depicted in Table 7.2. As expected, the area of the REASTD designed based on K=5 is larger than K=3 and K=4 ASIC implementations. Area overheads of the REASTD were 173% and 79% for K=3 and K=4 ASIC implementations, respectively. On the other hand, the area overhead compared to K=5 ASIC implementation was just 6%. When the results of each component of the REASTD are compared with the results of K5A, it is clear that the total area overhead is led by the reconfigurable components such as BMNU, SMCU, and LCU. The BMN does not include the reconfigurable feature, so its area remains unchanged.

The REASTD power results need to be analysed in a different way. The REASTD power was evaluated after it was configured for each constraint length K=3, 4, and 5. In this power evaluation, two cases were considered for the REASTD. One is the case of without the gated clock; the other is the case of with the gated clock. In the former case, the REASTD can be configured for each constraint length, but it operates without the gated clock. Although a clock signal is input to all the components, zero sequences are set to inputs of the unused components so that glitching can be reduced. Thus, the power results obtained from the two cases provide information about the effectiveness of the gated clock.

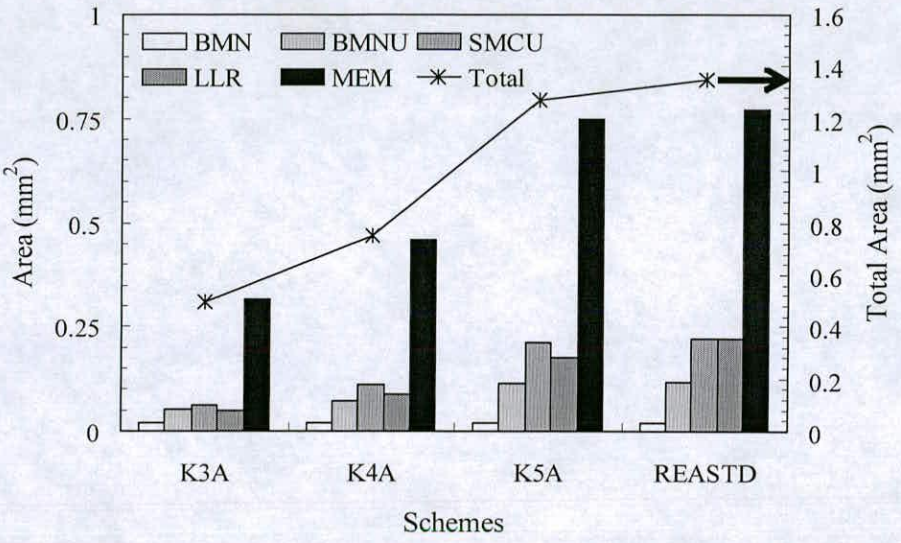



Figure 7.22 The area results and comparisons.

Table 7.2 List of area results.

| | K3A | K4A | K5A | REASTD | | | |
|---|---|---|---|---|---|---|---|
| | Area ($mm^2$) | Area ($mm^2$) | Area ($mm^2$) | Area ($mm^2$) | K3R Save (%) | K4R Save (%) | K5R Save (%) |
| BMU | 0.019 | 0.019 | 0.019 | 0.019 | 0 | 0 | 0 |
| BMNU | 0.051 | 0.072 | 0.115 | 0.115 | -126.4 | -59.3 | -0.3 |
| SMU | 0.060 | 0.110 | 0.211 | 0.220 | -265.5 | -99.3 | -4.4 |
| LCU | 0.048 | 0.089 | 0.176 | 0.223 | -361.5 | -149.4 | -26.1 |
| MEM | 0.314 | 0.459 | 0.749 | 0.771 | -145.2 | -67.8 | -2.8 |
| Total | 0.493 | 0.751 | 1.272 | 1.350 | -173.3 | -79.5 | -6.0 |

Figure 7.23 illustrates the power results of the REASTD with and without the gated clock, after being configured for each constraint length. The primary and secondary y-axes indicate

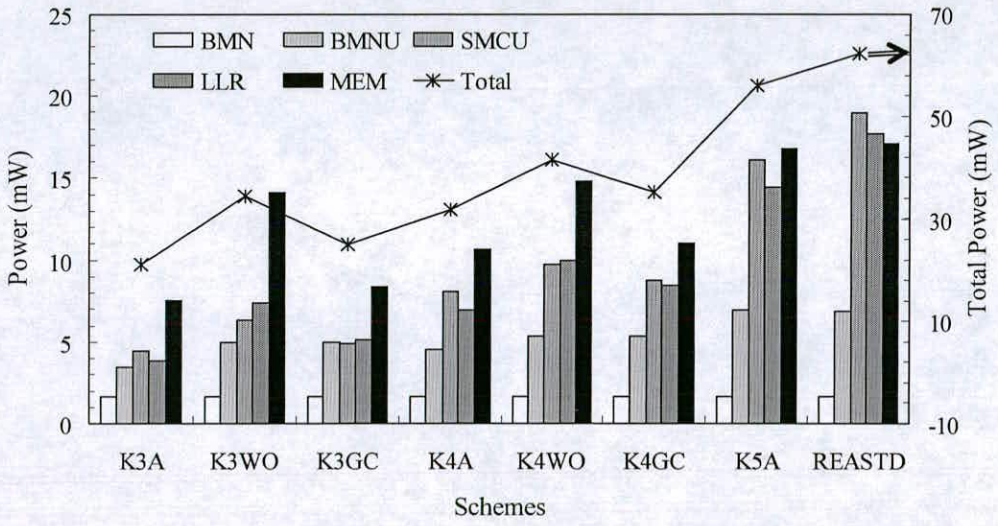



Figure 7.23 The power results and comparisons.

Table 7.3 List of power comparisons for K=3.

| | K3A | K3WO | | K3GC | |
|---|---|---|---|---|---|
| | Power (mW) | Power (mW) | Save (%) | Power (mW) | Save (%) |
| BMU | 1.6 | 1.65 | 0.9 | 1.6 | 0.9 |
| BMNU | 3.4 | 4.964 | -42.3 | 4.96 | -42.3 |
| SMCU | 4.4 | 6.328 | -41.6 | 4.88 | -9.3 |
| LCU | 3.8 | 7.4 | -91.5 | 5.12 | -32.5 |
| MEM | 7.5 | 14.097 | -85.7 | 8.37 | -10.3 |
| Total | 21.0 | 34.439 | -63.4 | 25.00 | -18.6 |

the breakdown component and total power results. In the figure, K3WO and K3GC imply that REASTD is configured for K=3 'without gated clock' and 'with gated clock', respectively.

Table 7.4 List of power comparisons for K=4.

| | K4A | K4WO | | K4GC | |
|---|---|---|---|---|---|
| | Power (mW) | Power (mW) | Save (%) | Power (mW) | Save (%) |
| BMU | 1.6 | 1.63 | 1.9 | 1.63 | 1.9 |
| BMNU | 4.5 | 5.33 | -18.2 | 5.33 | -18.2 |
| SMCU | 8.1 | 9.75 | -20.4 | 8.79 | -8.5 |
| LCU | 6.9 | 10 | -44.0 | 8.48 | -22.1 |
| MEM | 10.6 | 14.83 | -39.5 | 11.03 | -3.7 |
| Total | 31.8 | 41.56 | -30.5 | 35.27 | -10.7 |

Table 7.5 List of power comparisons for K=5.

| | K5A | REASTD | |
|---|---|---|---|
| | Power (mW) | Power (mW) | Save (%) |
| BMU | 1.6 | 1.66 | -0.8 |
| BMNU | 6.9 | 6.87 | 0.8 |
| SMCU | 16.0 | 18.95 | -17.7 |
| LCU | 14.4 | 17.69 | -22.3 |
| MEM | 16.7 | 17.05 | -1.7 |
| Total | 55.9 | 62.24 | -11.3 |

In the figure, the ASIC implementation power results are the results given in Chapter 4. Tables 7.3, 7.4 and 7.5 summarize the comparisons of the power results for constraint length K=3, 4, and 5, respectively. It is clear from the results that the clock gating method is very effective for conserving the power consumption of the REASTD when it is configured for K=3 and 4. While the power overheads of K3WO and K4WO were 63% and 30%, as compared to the ASIC turbo decoders, the power overheads using the clock gating method were 18% for K=3 and 10% for K=4. When the REASTD power for K=5 is compared with K5A, it is clear that the logics required for the reconfigurability resulted in an 11% increase

in power. As in the area results, most of the power overheads were led by the reconfigurable components, BMNU, SMCU, and LCU. Among them, the LCU power overhead dominantly affected the overall power overhead.

### 7.6.1 Hardware Test Systems for RDASTD

A test environment has been built for verifying and evaluating the reconfigurable turbo decoder at the RTL and gate levels. For this, two test systems for BTC and DTC were created. Figure 7.24 illustrates a block diagram of the test systems. The systems consist of two main blocks, one is for generating turbo codes and transmitting them to the decoder over the AWGN channel, and the other one is for performing and testing the turbo decoding process with the transmitted data. As can be seen, BTC and DTC codes were generated in a Matlab™ environment, and then, they were passed to the testbench after converting into a fixed-point representation. Additionally, the interleaver and deinterleaver addresses generated by the interleaver and deinterleaver address generator (IAG & DAG) were sent to

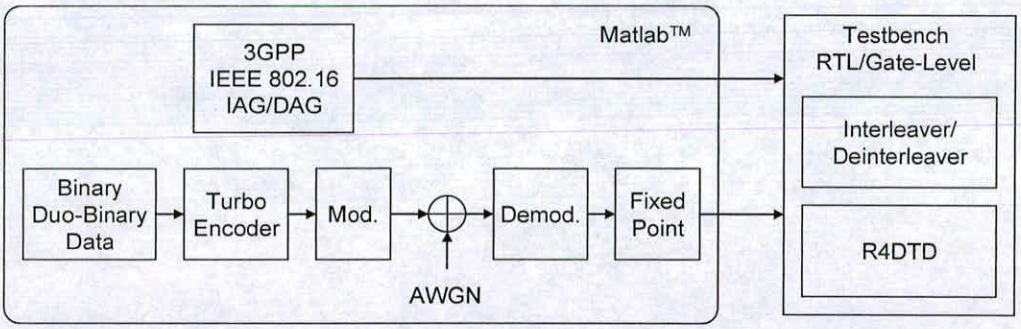



Figure 7.24 Test systems for the reconfigurable turbo decoder verification.

the testbench. The interleaver and deinterleaver addresses for binary and double-binary turbo codes were generated following the method described in 3GPP and IEEE 802.16. The testbench consists of memory blocks for accumulating all the transmitted data, interleaver and deinterleaver addresses, and control units for controlling data transactions between turbo decoders. From the tests, the latency of the turbo decoders was obtained as follows:

$$\text{Latency} = \text{clock period} \times (\text{window size} \times 4 + 3) \tag{7.13}$$

where 3 corresponds to the number of clock cycles due to the pipeline stages.

### 7.6.2 Hardware Performance Results

The RDASTD also designed using Verilog HDL has been synthesized with the Synopsys DesignCompiler™ using the UMC 0.18 um standard CMOS cell library. RTL and gate-level simulations of the turbo decoder hardware were performed using Cadence Verilog-XL™ with the test systems shown in Figure 7.24. After completing the gate-level simulation, power consumption was evaluated using the Synopsys PowerCompiler™ with the switching activities generated from all the circuit nets during the gate-level simulation. All the simulations were carried out at a clock frequency of 50MHz.

The RDASTD implementation has been investigated with various schemes in order to find an optimized hardware implementation. In addition, R4 BTC and DTC MLMAP turbo decoders have been implemented as ASIC in order to compare the hardware performance with the RDASTD. Figure 7.25 and Table 7.6 illustrate and summarize the area results for the different turbo decoder schemes. In the figure and table, CON is the conventional turbo decoder for constraint length K=4 BTC implemented in Chapter 3. R4A and DTA are turbo decoders for R4 BTC and DTC, respectively, implemented in this chapter. RDASTD implies the reconfigurable turbo decoder which is able to support R4 BTC and DTC. It is clear that the area of RDASTD is the largest among the schemes due to the additional logics for the reconfigurability. Area overhead of RDASTD was 40%, 21%, and 5% compared to CON, R4A, and DTA, respectively. In addition, the area of R4A is 15% more than the area of CON due to the increased complexity. As can be seen, most of the area increase was led by the increased computational complexity in the units for computing the branch and state metrics (BMU and SMCU), and LLR value (LCU). The R4 based turbo decoder requires a much more complex computation process, which results in the significant area increase of the computational logics. However, the area of memory blocks is less in R4A and DTA, and is slightly higher in RDASTD when compared to CON. This is because of the reduction of the memory blocks for storing the forward state metrics before they are fed to the LLR computation unit for calculating LLR. Although the computational complexity is increased

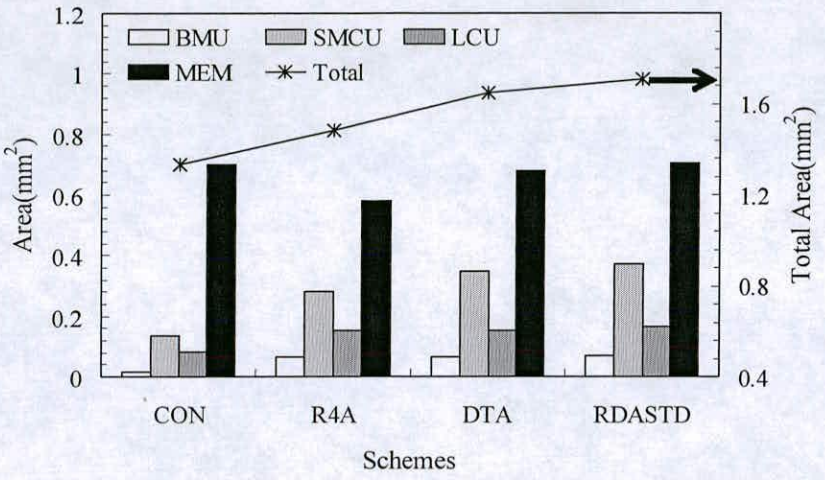



Figure 7.25 Area results and comparisons.

Table 7.6 List of area results

| | CON | R4A | | DTA | | RDASTD | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Area ($mm^2$) | Area ($mm^2$) | Save (%) CON | Area ($mm^2$) | Save (%) | Area ($mm^2$) | Save (%) CON | Save (%) R4D | Save (%) DTA |
| BMU | 0.014 | 0.067 | -354.1 | 0.067 | - | 0.070 | -377.0 | -5.0 | -5.0 |
| SMCU | 0.137 | 0.283 | -105.8 | 0.347 | - | 0.370 | -169.4 | -30.8 | -6.5 |
| LCU | 0.083 | 0.151 | -81.7 | 0.152 | - | 0.165 | -97.6 | -8.7 | -8.5 |
| MEM | 0.698 | 0.577 | 17.2 | 0.678 | - | 0.703 | -0.7 | -21.7 | -3.6 |
| Total | 0.934 | 1.079 | -15.5 | 1.245 | - | 1.309 | -40.2 | -21.2 | -5.1 |

by implementation based on R4, the size of the memory required for storing the forward state metrics can be reduced by almost half of the memory required by CON. In other words, if a turbo decoder is implemented based on radix-8, the memory size will be reduced by almost a quarter of the CON memory. In this case, however, the increased computational complexity would dramatically increase the overall area and power as well.

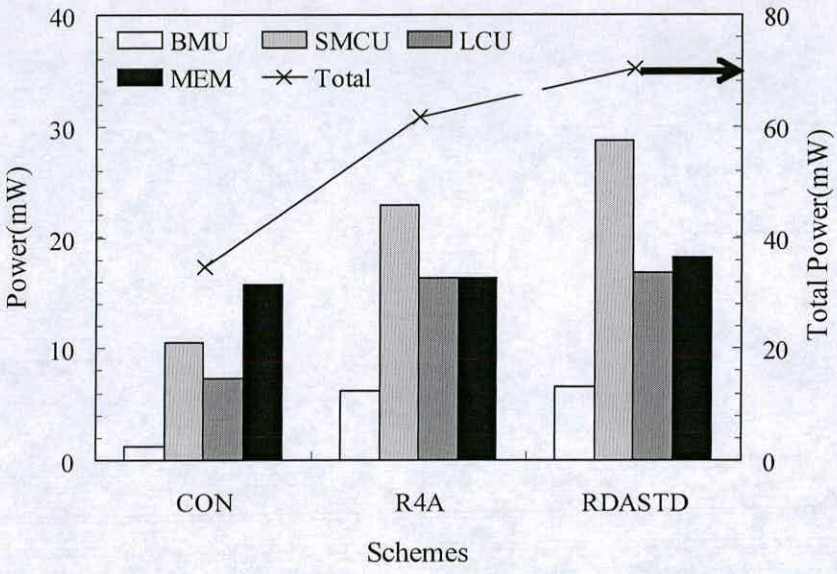



Figure 7.26 Power results and comparisons for binary turbo codes.

Table 7.7 List of power results for binary turbo codes

| | CON | R4A | | RDASTD | | |
|---|---|---|---|---|---|---|
| | Power (mW) | Power (mW) | Save (%) CON | Power (mW) | Save (%) CON | Save (%) R4A |
| BMU | 1.21 | 6.18 | -407.9 | 6.59 | -441.1 | -6.5 |
| SMCU | 10.54 | 22.89 | -117.1 | 28.69 | -172.1 | -25.3 |
| LCU | 7.35 | 16.36 | -122.5 | 16.79 | -128.4 | -2.6 |
| MEM | 15.66 | 16.28 | -3.96 | 18.15 | -15.8 | -11.4 |
| Total | 34.78 | 61.73 | -77.50 | 70.23 | -101.9 | -13.7 |

The power consumption results need to be compared differently than the area results because two different data for BTC and DTC are fed into the turbo decoders. Furthermore, the block sizes of BTC and DTC are not the same. Due to these considerations, RDASTD has been compared with BTC and DTC separately.

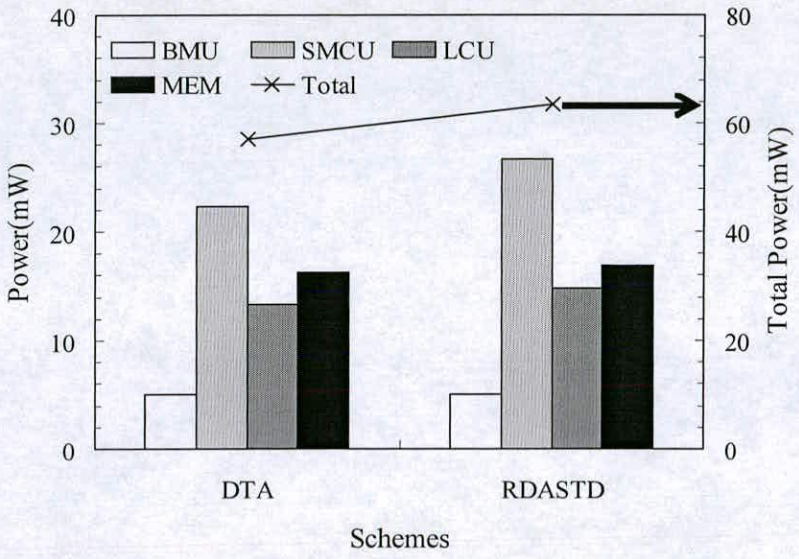



Figure 7.27 Power results and comparisons for double-binary turbo codes

Table 7.8 List of power results for double-binary turbo codes.

| | DTA | | RDASTD | |
|---|---|---|---|---|
| | Power (mW) | Save (%) | Power (mW) | Save (%) |
| BMU | 5.01 | - | 5.08 | -1.4 |
| SMCU | 22.38 | - | 26.65 | -19.0 |
| LCU | 13.27 | - | 14.86 | -11.9 |
| MEM | 16.27 | - | 16.95 | -4.1 |
| Total | 56.93 | - | 63.55 | -11.6 |

The power consumption results of the turbo decoders after completing eight iterations for BTC are given in Figure 7.26, where the primary and secondary Y-axis denotes the breakdown and overall power consumption of the turbo decoders. Table 7.7 lists the power results where CON and R4A are the ASIC turbo decoders for BTC and R4 BTC,

respectively. In the results, RDASTD is configured for BTC. The power results of the turbo decoders are obtained after simulations with the same data sets and number of iterations. A clock frequency of 50MHz is used. Total power consumption of R4A and RDASTD increased by 77% and 101% as compared with CON. On the other hand, the power overhead of RDASTD was only 13% as compared to R4A. From the results, it can be seen that the power overheads are due to the high complexity of the computational logics implemented based on R4. In R4A the power is five times the power of BMU and double the power of SMCU and LCU as compared to CON. Conversely, the power of MEM is not significantly increased. As in the area results, this is because the R4 based decoder implementations reduce the depth of the memory required to store the forward state metrics.

For better comparisons among the different turbo decoders, the total energy of the decoder might be considered because the time needed to complete eight iterations involves different CON and R4 based turbo decoders. R4 and RDASTD generate two LLR values, and take only half of the time required by CON to complete the decoding process. Thus, the results reveal that the total energy obtained by multiplying time by the power of the R4 and RDASTD is almost the same or less than the energy consumed by CON.

Figure 7.27 illustrates the power results of DTA and RDASTD configured for DTC. In this case, the power consumption results have been obtained from the same size of input blocks and decoding time for both the decoders, DTA and RDASTD. Table 7.8 summarizes the power results. Overall, the power overhead of RDASTD was 11% as compared to DTA. As can be seen, the results show that the power increase was mainly led by the reconfigurable logics incorporated in SMCU and LCU. The power increase in BMU and MEM of RDASTD was less significant than SMCU and LCU.

## 7.7 Summary

This chapter presented a reconfigurable turbo decoder architecture for supporting different constraint lengths, K=3, 4, and 5, turbo codes. For designing the reconfigurable architecture, a mapping method has been introduced for efficient implementation. The reconfigurable

architecture has been implemented based on the constraint length K=5. To save on the power consumption, the clock gating method has been applied to the implementation. The power results have shown that the clock gating method is very effective for saving on the power consumption of the reconfigurable architecture. The power overhead of the reconfigurable turbo decoder was around 18%, 10%, and 11% respectively, when compared with K=3, 4, and 5 turbo decoder ASIC implementations. Meanwhile, the reconfigurable turbo decoder increased just 6% in area as compared to the K=5 turbo decoder AISC implementation. This chapter also presented a reconfigurable turbo decoder architecture for constraint length K=4 BTC and DTC. Before designing the RDASTD, the radix-4 based turbo decoder for BTC has been investigated. This investigation has revealed that the radix-4 BTC and DTC turbo decoders had many similarities in their functions and implementation methods. The RDASTD, which exploited the radix-4 method for its hardware implementation, has been compared with the performance of ASIC turbo decoder implementations. For the comparison, the radix-4 BTC and DTC turbo decoders have been implemented as ASIC in this chapter. The simulation results have shown that the area overhead of the RDASTD was 40%. The power of the RDASTD has been analysed in two modes, BTC and DTC. When the RDASTD was set to the BTC mode, the power overhead was 101%. However, the total energy required to complete the turbo decoding process was less than the conventional turbo decoders.

# Chapter 8

# Conclusions

## 8.1 Introduction

This chapter concludes the thesis. Section 2 reviews each chapter specifying the results obtained. In Section 3, the achievements of the thesis are listed. Finally, Section 4 gives possible directions for future work related to this thesis work.

## 8.2 Review of Thesis Contents

Chapter 2 has described the turbo encoding and decoding principles for binary and double-binary turbo codes, describing their encoder and decoder structures. This chapter explained turbo decoding algorithms based up on MAP and SOVA, showing a turbo decoder structure for an iterative process. For a practical turbo decoder implementation, the complexity of the algorithms should be reduced. Techniques for efficient turbo decoder hardware implementations were reviewed and summarized. Some of the techniques were used in this thesis to reduce the turbo decoder computational complexity and to improve hardware performance in terms of area, power, and speed for throughput.

Chapter 3 has presented the MLMAP based turbo decoder hardware architectures with SW method for K=3, 4, and 5. In the architecture, the SW method was realised by using LIFO and FIFO blocks. A detailed structure of the decoding process units incorporated in the architecture was described and the implementation method and the functionality for the decoding process were explained. In the simulation results, the turbo decoder performance in terms of BER was examined in various ways to justify the algorithms and other factors that affect the turbo decoder performance. The real MAP and the fixed-point MLMAP based turbo decoders have been compared in terms of BER performance in order to observe the

performance differences between the two turbo decoder implementations. The comparisons showed that the MLMAP based turbo decoder can provide a BER performance without significant degradation. After showing the design flow, the MLMAP turbo decoder architectures have been designed using Verilog HDL and then synthesized using the UMC 0.18μm standard CMOS technology. Area, power, and critical path delay results of the MLTBD architecture for constraint lengths K=3, 4, and 5 were estimated. Also, the total number of gates of the architectures was obtained after generating their layouts. These hardware performance results were used to compare the improved turbo decoders introduced in later chapters.

Chapter 4 has proposed a high performance MLMAP turbo decoder architecture implemented base on a novel scheme in order to save area and power, and to reduce the critical path delay for high speed implementation. For achieving low power and area efficient MLMAP turbo decoder implementations, the memory blocks needed for the SW method were reduced by using a triple read port-based memory. Simulation results show that the memory reduction can achieve 30% reduction in area and 20% in power. In addition, efficient implementations of LCU and BMNU contributed to power and area savings. The BMN method addressed the inherent critical path delay problem to achieve high speed MLMAP turbo decoder for high throughput. Using this method, the critical path delay was reduced by up to 42%, as compared with the delay of the conventional turbo decoder.

Chapter 5 has described the SOVA and TSOVA algorithms that are alternatively used for the turbo decoding process. In this chapter, the TSOVA based turbo decoder has been introduced to evaluate the BER performance. Before the evaluation, the optimized survivor and update depths of the TSOVA turbo decoders were investigated. The simulation results have shown that the BER performance was almost saturated when the survivor and update depths were K×6 and K×4, respectively. Based on these results, the TSOVA turbo decoders for K=3, 4, and 5 have been simulated to evaluate their BER performance. The performance results were compared with the results of the MLMAP based turbo decoder. These comparisons showed that the TSOVA turbo decoder can produce a BER performance close to the MLMAP turbo decoder without significant performance degradations.

Chapter 6 has proposed a novel TSOVA turbo decoder based on TBA. This novel turbo decoder architecture was described and compared with the TSOVAREA architecture suggested in the literature. The hardware simulation results have shown that the TSOVATBA based turbo decoder implementations can save more area and power than the TSOVAREA based turbo decoders. In addition, the hardware performance was compared with the MLMAP based turbo decoder hardware. It was also shown that the TSOVATBA can save up to 72% in area and 52% in power compared to the MLMAP turbo decoder. These results indicate that the TSOVATBA turbo decoder is suitable to apply to mobile and portable wireless communication systems requiring low power consumption and area usage, without significant BER performance degradation.

Chapter 7 has presented a reconfigurable turbo decoder architecture for supporting different constraint lengths, K=3, 4, and 5, turbo codes. To improve the efficiency of the implementation, a novel mapping method was employed in the design of the reconfigurable architecture. The reconfigurable architecture has been implemented based on the constraint length K=5. To reduce the power consumption as the architecture is configured for constraint lengths less than K=5, the clock gating method has been applied to the implementation. The power results have shown that the clock gating method is very effective in reducing the power consumption of the reconfigurable architecture. The power overhead of the reconfigurable turbo decoder was around 18%, 10%, and 11% as compared with K=3, 4, and 5 turbo decoder ASIC implementations, respectively. Meanwhile, the reconfigurable turbo decoder area increased just by 6% as compared to K=5 turbo decoder AISC implementation. On this other hand, this chapter has presented another reconfigurable turbo decoder architecture which can be configured for BTC and DTC with K=4. Before designing the reconfigurable turbo decoder hardware, radix-4 based turbo decoder for BTC has been investigated. This revealed that the radix-4 based BTC and the DTC turbo decoders had many similarities in their functions and implementation methods. Therefore, the reconfigurable turbo decoder exploited the radix-4 for its hardware implementation. For comparison, radix-4 based BTC and DTC turbo decoders have also been implemented as an ASIC each. The simulation results have shown that the area overhead of the reconfigurable turbo decoder was 40%. The power of the reconfigurable has been analysed in two modes, BTC and DTC. When the reconfigurable turbo decoder was set to the BTC mode, the power overhead was 101%. However, the total energy required to complete the turbo decoding

process was less than the conventional turbo decoders, due to reducing the time required for the decoding process.

## 8.3 List of Achievements

This section specifies the achievements of this thesis.

- A novel technique for high speed turbo decoder implementation in order to achieve high throughput is proposed. The technique normalizes the branch metrics to reduce the inherent critical path delay in the state metric computation process. It achieves 42% reduction in critical path delay.

- A low power and area efficient turbo decoder is developed by reducing the memory block size required by sliding window method. This results in savings up to 30% in area and 20 % in power.

- A novel concept of SOVA turbo decoder implementation is presented. The new SOVA turbo decoder is based on TBA, which significantly reduces the power and area as compared with a conventional REA based SOVA turbo decoder implementation.

- Two reconfigurable turbo decoder implementations are presented to support multi standard wireless communication systems. The first turbo decoder is designed to be configured for different constraint lengths K=3, 4, and 5. This reconfigurable turbo decoder is implemented with a clock gating method in order to save power when it is configured for constant lengths less than K=5. The second reconfigurable turbo decoder supports binary and double-binary turbo codes is designed. To make it reconfigurable, the turbo decoder exploits a radix-4 binary turbo decoding method.

## 8.4 Future Research Directions

Wireless communication systems are evolving to provide better performance such as BER and throughput. The use of turbo codes in these systems is expected to be continuously increasing and hence the performance of these systems will be strongly affected by the turbo decoder they employ. Some of the issues related to turbo decoders are suggested below for future work.

High throughput is one of the important issues for current and future wireless communication systems. Due to their iterative process, the turbo decoders could be an obstacle to achieving high throughput. A parallel turbo decoder scheme is one of the solutions to address this problem. Several papers have suggested some parallel turbo decoder schemes in order to achieve high throughput. However, this will lead to large hardware overhead in terms of area and power. This could be an obstacle to employing the parallel turbo decoders for mobile applications. In the literature, there is little work on efficient implementation of parallel turbo decoder architectures. Thus, in order to achieve an efficient high throughput turbo decoder, the hardware implementation methods must be researched.

In the turbo decoding process, interleaving and de-interleaving require large memory blocks. Moreover, the addresses for the interleaving and de-interleaving also need to be stored in a large memory size. The memory size can be reduced by implementing an efficient address generator for interleaving and de-interleaving. In wireless communication systems, the address can be generated easily and efficiently if the input block size is known. However, if a parallel turbo decoder scheme is used, the multiple output of the turbo decoder can conflict with writing the output to the interleaver memory. Thus, the technique to avoid the data collision during the writing process should be studied. Another difficulty is that usually different interleaving methods between the wireless communication systems are employed to implement the address generator.

Less attention has been paid to a SOVA based turbo decoder when compared to a MAP based turbo decoder. The method proposed in this thesis has contributed to saving the hardware costs, as compared to the MLMAP based turbo decoder, without significant degradation in BER performance. Thus, the use of the SOVA based turbo decoder can contribute to reduce the costs of the receiver systems requiring high performance and

throughput. Also, a SOVA decoder for double-binary turbo codes is hardly considered in the literature. Therefore, a study of the SOVA turbo decoder architecture should be carried out for supporting different types of turbo codes.

# Appendix A. List of Publication

1. **J. H. Han**, A.T. Erdogan, and T. Arslan, "Exploiting Radix-4 Method in Max-Log-MAP Turbo Decoder Implementation for Duo-Binary Turbo Codes," submitted to *IEEE International Symposium on Circuits and Systems*, 27-30 May, 2007.

2. **J. H. Han**, A.T. Erdogan, and T. Arslan, "A Power and Area Efficient Maximum Lilkelihood Detector Implementation for High Throughput MIMO Systems," in *IEEE VLSI Design Conference*, 6 – 10 Jan., 2007.

3. **J. H. Han**, A.T. Erdogan, and T. Arslan, "Traceback Algorithm based Two-Step SOVA Turbo Soft-Input Soft-Output Decoder VLSI Architecture," submitted to *IEEE Transactions on VLSI systems*.

4. **J. H. Han**, A.T. Erdogan, and T. Arslan, "Normalization of Branch Metrics for High Speed Turbo Decoder Implementation," submitted to *IEEE Transactions on Circuits and Systems I*.

5. **J. H. Han**, A.T. Erdogan, and T. Arslan, "An Efficient Reconfigurable Max-Log-MAP Turbo SISO Decoder for Multi-Standards Wireless Communication Systems," submitted *to IEEE Transactions on VLSI systems*.

6. R. Zhang, **J. H. Han**, A.T. Erdogan, and T. Arslan, "Low Power CORDIC IP Core Implementation," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. III-956 – III-959, 21 – 24 May 2006.

7. T. Takahashi, A.T. Erdogan, T. Arslan, and **J. H. Han**, "Low Power Layered Space-Time Channel Detector for MIMO Systems," in *IEEE Computer Society Annual Symposium on VLSI*, 2 – 3 March, 2006.

8. **J. H. Han**, A.T. Erdogan, and T. Arslan, "A Low Power Pipelined Maximum Likelihood Detector for 4x4 QPSK MIMO Wireless Communication Systems," in *IEEE Computer Society Annual Symposium on VLSI*, 2 – 3 March, 2006.

9. **J. H. Han**, A. Erdogan, and T. Arslan, "Implementation of an Efficient Two-Step SOVA Turbo Decoder for Wireless Communication Systems," in *IEEE Global Telecommunications Conference*, pp. 2429 – 2433, 28 Nov. – 2 Dec., 2005.

10. **J. H. Han**, A. Erdogan, and T. Arslan, "Power and Area Efficient Turbo Decoder Implementation for Mobile Wireless Systems," in *IEEE 2005 Workshop on Signal Processing Systems*, pp. 705 – 709, 2 – 4 Nov., 2005.

11. **J. H. Han**, A. Erdogan, and T. Arslan, "A Power Efficient Reconfigurable Max-Log-MAP Turbo Decoder for Wireless Communication Systems," in *IEEE System on Chip Conference*, pp. 247-250, 25 – 28 Sep., 2005.

12. **J. H. Han**, A. Erdogan, and T. Arslan, "High Speed Max-Log-MAP Turbo SISO Decoder Implementation Using Branch Metric Normalization," in *IEEE Computer Society Annual Symposium on VLSI*, pp. 173-178, 11 –12 May, 2005.

# Appendix B. 3GPP Turbo Codes Specifications

## B.1 Turbo coder

The scheme of Turbo coder is a Parallel Concatenated Convolutional Code (PCCC) with two 8-state constituent encoders and one Turbo code internal interleaver. The coding rate of Turbo coder is 1/3. The structure of Turbo coder is illustrated in Figure B.1.

The transfer function of the 8-state constituent code for PCCC is:

$$G(D) = \left[1, \frac{g_1(D)}{g_0(D)}\right],$$

where

$$g_0(D) = 1 + D^2 + D^3,$$
$$g_1(D) = 1 + D + D^3.$$

The initial value of the shift registers of the 8-state constituent encoders shall be all zeros when starting to encode the input bits.

Output from the Turbo coder is

$$x_1, z_1, z'_1, x_2, z_2, z'_2, \ldots, x_K, z_K, z'_K,$$

where $x_1, x_2, \ldots, x_K$ are the bits input to the Turbo coder i.e. both first 8-state constituent encoder and Turbo code internal interleaver, and $K$ is the number of bits, and $z_1, z_2, \ldots, z_K$ and $z'_1, z'_2, \ldots, z'_K$ are the bits output from first and second 8-state constituent encoders, respectively.

The bits output from Turbo code internal interleaver are denoted by $x'_1, x'_2, \ldots, x'_K$, and these bits are to be input to the second 8-state constituent encoder.

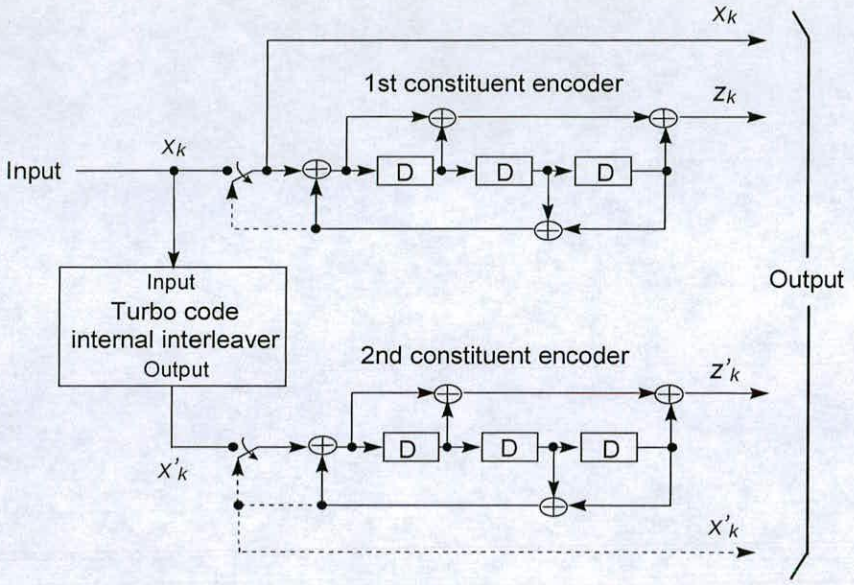



Figure B.1 Structure of rate 1/3 Turbo coder (dotted lines apply for trellis termination only)

## B.2 Trellis termination for Turbo coder

Trellis termination is performed by taking the tail bits from the shift register feedback after all information bits are encoded. Tail bits are padded after the encoding of information bits.

The first three tail bits shall be used to terminate the first constituent encoder (upper switch of figure B.1 in lower position) while the second constituent encoder is disabled. The last three tail bits shall be used to terminate the second constituent encoder (lower switch of Figure B.1 in lower position) while the first constituent encoder is disabled.

The transmitted bits for trellis termination shall then be:

$$x_{K+1}, z_{K+1}, x_{K+2}, z_{K+2}, x_{K+3}, z_{K+3}, x'_{K+1}, z'_{K+1}, x'_{K+2}, z'_{K+2}, x'_{K+3}, z'_{K+3}.$$

## B.3 Turbo code internal interleaver

The Turbo code internal interleaver consists of bits-input to a rectangular matrix with padding, intra-row and inter-row permutations of the rectangular matrix, and bits-output from the rectangular matrix with pruning. The bits input to the Turbo code internal interleaver are denoted by $x_1, x_2, x_3, \ldots, x_K$, where $K$ is the integer number of the bits and takes one value of $40 \leq K \leq 5114$. The relation between the bits input to the Turbo code internal interleaver and the bits input to the channel coding is defined by $x_k = o_{irk}$ and $K = K_i$.

The following subclause specific symbols are used in following sections :

| | |
|---|---|
| $K$ | Number of bits input to Turbo code internal interleaver |
| $R$ | Number of rows of rectangular matrix |
| $C$ | Number of columns of rectangular matrix |
| $p$ | Prime number |
| $v$ | Primitive root |
| $\langle s(j) \rangle_{j \in \{0,1,\cdots,p-2\}}$ | Base sequence for intra-row permutation |
| $q_i$ | Minimum prime integers |
| $r_i$ | Permuted prime integers |
| $\langle T(i) \rangle_{i \in \{0,1,\cdots,R-1\}}$ | Inter-row permutation pattern |
| $\langle U_i(j) \rangle_{j \in \{0,1,\cdots,C-1\}}$ | Intra-row permutation pattern of $i$-th row |
| $i$ | Index of row number of rectangular matrix |
| $j$ | Index of column number of rectangularmatrix |
| $k$ | Index of bit sequence |

### B.3.1 Bits-input to rectangular matrix with padding

The bit sequence $x_1, x_2, x_3, \ldots, x_K$ input to the Turbo code internal interleaver is written into the rectangular matrix as follows.

(1) Determine the number of rows of the rectangular matrix, $R$, such that:

$$R = \begin{cases} 5, \text{if } (40 \le K \le 159) \\ 10, \text{if } ((160 \le K \le 200) \text{ or } (481 \le K \le 530)) \\ 20, \text{if } (K = \text{any other value}) \end{cases}.$$

The rows of rectangular matrix are numbered 0, 1, …, $R$ - 1 from top to bottom.

(2) Determine the prime number to be used in the intra-permutation, $p$, and the number of columns of rectangular matrix, $C$, such that:

if $(481 \le K \le 530)$ then
  $p = 53$ and $C = p$.
else
Find minimum prime number $p$ from table 2 such that
  $K \le R \times (p+1)$,
and determine $C$ such that

$$C = \begin{cases} p-1 & \text{if } K \le R \times (p-1) \\ p & \text{if } R \times (p-1) < K \le R \times p \\ p+1 & \text{if } R \times p < K \end{cases}.$$

end if
The columns of rectangular matrix are numbered 0, 1, …, $C$ - 1 from left to right.

Table B.1 List of prime number $p$ and associated primitive root $v$.

| p | v | p | v | p | v | p | v | p | v |
|---|---|---|---|---|---|---|---|---|---|
| 7 | 3 | 47 | 5 | 101 | 2 | 157 | 5 | 223 | 3 |
| 11 | 2 | 53 | 2 | 103 | 5 | 163 | 2 | 227 | 2 |
| 13 | 2 | 59 | 2 | 107 | 2 | 167 | 5 | 229 | 6 |
| 17 | 3 | 61 | 2 | 109 | 6 | 173 | 2 | 233 | 3 |
| 19 | 2 | 67 | 2 | 113 | 3 | 179 | 2 | 239 | 7 |
| 23 | 5 | 71 | 7 | 127 | 3 | 181 | 2 | 241 | 7 |
| 29 | 2 | 73 | 5 | 131 | 2 | 191 | 19 | 251 | 6 |
| 31 | 3 | 79 | 3 | 137 | 3 | 193 | 5 | 257 | 3 |
| 37 | 2 | 83 | 2 | 139 | 2 | 197 | 2 | | |
| 41 | 6 | 89 | 3 | 149 | 2 | 199 | 3 | | |
| 43 | 3 | 97 | 5 | 151 | 6 | 211 | 2 | | |

(3) Write the input bit sequence $x_1, x_2, x_3, \ldots, x_K$ into the $R \times C$ rectangular matrix row by row starting with bit $y_1$ in column 0 of row 0:

$$\begin{bmatrix} y_1 & y_2 & y_3 & \cdots & y_C \\ y_{(C+1)} & y_{(C+2)} & y_{(C+3)} & \cdots & y_{2C} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ y_{((R-1)C+1)} & y_{((R-1)C+2)} & y_{((R-1)C+3)} & \cdots & y_{R\times C} \end{bmatrix}$$

where $y_k = x_k$ for $k = 1, 2, \ldots, K$ and if $R \times C > K$, the dummy bits are padded such that $y_k = 0 or 1$ for $k = K + 1, K + 2, \ldots, R \times C$. These dummy bits are pruned away from the output of the rectangular matrix after intra-row and inter-row permutations.

### B.3.2 Intra-row and inter-row permutations

After the bits-input to the $R \times C$ rectangular matrix, the intra-row and inter-row permutations for the $R \times C$ rectangular matrix are performed stepwise by using the following algorithm with steps (1) – (6):

(1) Select a primitive root $v$ from Table B.1 in previous subsection, which is indicated on the right side of the prime number $p$.

(2) Construct the base sequence $\langle s(j) \rangle_{j \in \{0,1,\cdots,p-2\}}$ for intra-row permutation as:

$s(j) = (v \times s(j-1)) \bmod p$, $\quad j = 1, 2, \ldots, (p - 2)$, and $s(0) = 1$.

(3) Assign $q_0 = 1$ to be the first prime integer in the sequence $\langle q_i \rangle_{i \in \{0,1,\cdots,R-1\}}$, and determine the prime integer $q_i$ in the sequence $\langle q_i \rangle_{i \in \{0,1,\cdots,R-1\}}$ to be a least prime integer such that g.c.d$(q_i, p - 1) = 1$, $q_i > 6$, and $q_i > q_{(i-1)}$ for each $i = 1, 2, \ldots, R - 1$. Here g.c.d. is greatest common divisor.

(4) Permute the sequence $\langle q_i \rangle_{i\in\{0,1,\cdots,\ -1\}}$ to make the sequence $\langle r_i \rangle_{i\in\{0,1,\cdots,R-1\}}$ such that

$r_{T(i)} = q_i$, $i = 0, 1, \ldots, R - 1$,

where $\langle T(i) \rangle_{i\in\{0,1,\cdots,R-1\}}$ is the inter-row permutation pattern defined as the one of the four kind of patterns, which are shown in Table B.2, depending on the number of input bits $K$.

Table B.2 Inter-row permutation patterns for Turbo code internal interleaver

| Number of input bits $K$ | Number of rows $R$ | Inter-row permutation patterns <$T(0)$, $T(1)$, …, $T(R - 1)$> |
|---|---|---|
| $(40 \le K \le 159)$ | 5 | <4, 3, 2, 1, 0> |
| $(160 \le K \le 200)$ or $(481 \le K \le 530)$ | 10 | <9, 8, 7, 6, 5, 4, 3, 2, 1, 0> |
| $(2281 \le K \le 2480)$ or $(3161 \le K \le 3210)$ | 20 | <19, 9, 14, 4, 0, 2, 5, 7, 12, 18, 16, 13, 17, 15, 3, 1, 6, 11, 8, 10> |
| $K$ = any other value | 20 | <19, 9, 14, 4, 0, 2, 5, 7, 12, 18, 10, 8, 13, 17, 3, 1, 16, 6, 15, 11> |

(5) Perform the $i$-th ($i = 0, 1, \ldots, R - 1$) intra-row permutation as:

if ($C = p$) then

$U_i(j) = s((j \times r_i) \bmod (p-1))$, $j = 0, 1, \ldots, (p - 2)$, and $U_i(p - 1) = 0$,

where $U_i(j)$ is the original bit position of $j$-th permuted bit of $i$-th row.

end if

if ($C = p + 1$) then

$U_i(j) = s((j \times r_i) \bmod (p-1))$, $j = 0, 1, \ldots, (p - 2)$. $U_i(p - 1) = 0$, and $U_i(p) = p$,

where $U_i(j)$ is the original bit position of $j$-th permuted bit of $i$-th row, and

if ($K = R \times C$) then

Exchange $U_{R\,1}(p)$ with $U_{R\,1}(0)$.

end if

end if

if ($C = p - 1$) then

$U_i(j) = s((j \times r_i) \bmod (p-1)) - 1$, $j = 0, 1, \ldots, (p - 2)$,

where $U_i(j)$ is the original bit position of $j$-th permuted bit of $i$-th row.

end if

(6) Perform the inter-row permutation for the rectangular matrix based on the pattern $\langle T(i) \rangle_{i \in \{0,1,\cdots,R-1\}}$,

where $T(i)$ is the original row position of the $i$-th permuted row.

### B.3.3 Bits-output from rectangular matrix with pruning

After intra-row and inter-row permutations, the bits of the permuted rectangular matrix are denoted by $y'_k$ :

$$\begin{bmatrix} y'_1 & y'_{(R+1)} & y'_{(2R+1)} & \cdots & y'_{((C-1)R+1)} \\ y'_2 & y'_{(R+2)} & y'_{(2R+2)} & \cdots & y'_{((C-1)R+2)} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ y'_R & y'_{2R} & y'_{3R} & \cdots & y'_{C\times R} \end{bmatrix}$$

The output of the Turbo code internal interleaver is the bit sequence read out column by column from the intra-row and inter-row permuted $R \times C$ rectangular matrix starting with bit $y'_1$ in row 0 of column 0 and ending with bit $y'_{CR}$ in row $R$ - 1 of column $C$ - 1. The output is pruned by deleting dummy bits that were padded to the input of the rectangular matrix before intra-row and inter row permutations, i.e. bits $y'_k$ that corresponds to bits $y_k$ with $k > K$ are removed from the output. The bits output from Turbo code internal interleaver are denoted by $x'_1, x'_2, \ldots, x'_K$, where $x'_1$ corresponds to the bit $y'_k$ with smallest index $k$ after pruning, $x'_2$ to the bit $y'_k$ with second smallest index $k$ after pruning, and so on. The number of bits output from Turbo code internal interleaver is $K$ and the total number of pruned bits is:

$$R \times C - K.$$

# Appendix C. IEEE 802.16 Turbo Codes Specifications

## C.1 CTC encoder

The Convolutional Turbo Code (CTC) defined in this subclause is designed to enable support of hybrid ARQ (HARQ). HARQ implementation is optional. The CTC encoder, including its constituent encoder, is depicted in Figure C.1. It uses a double binary Circular Recursive Systematic Convolutional code. The bits of the data to be encoded are alternately fed to $A$ and $B$, starting with the MSB of the first byte being fed to $A$. The encoder is fed by blocks of $k$ bits or $N$ couples ($k = 2*N$ bits). For all the frame sizes, $k$ is a multiple of 8 and $N$ is a multiple of 4. Further, $N$ shall be limited to: $8 \leq N/4 \leq 1024$

The polynomials defining the connections are described in octal and symbol notations as follows:

- For the feedback branch: 0xB, equivalently $1 + D + D^3$ (in symbolic notation)
- For the $Y$ parity bit: 0xD, equivalently $1 + D^2 + D^3$
- For the W parity bit: 0x9, equivalently $1 + D^3$

First, the encoder (after initialization by the circulation state $Sc1$) is fed the sequence in the natural order (position 1) with the incremental address $i = 0 \ldots N-1$. This first encoding is called $C_1$ encoding. Then the encoder (after initialization by the circulation state $Sc2$) is fed by the interleaved sequence (switch in position 2) with incremental address $j = 0, \ldots N-1$. This second encoding is called $C_2$ encoding. The order in which the encoded bit shall be fed into the subpacket generation block is:

$$A, B, Y_1, Y_2, W_1, W_2 =$$
$$A_0, B_0, \ldots, A_{N-1}, B_{N-1}, Y_{1,0}, Y_{1,1}, \ldots, Y_{1,N-1}, Y_{2,0}, Y_{2,1}, \ldots, Y_{2,N-1}, W_{1,0}, W_{1,1}, \ldots, W_{1,N-1}, W_{2,0}, W_{2,1}, \ldots, W_{2,N-1}$$

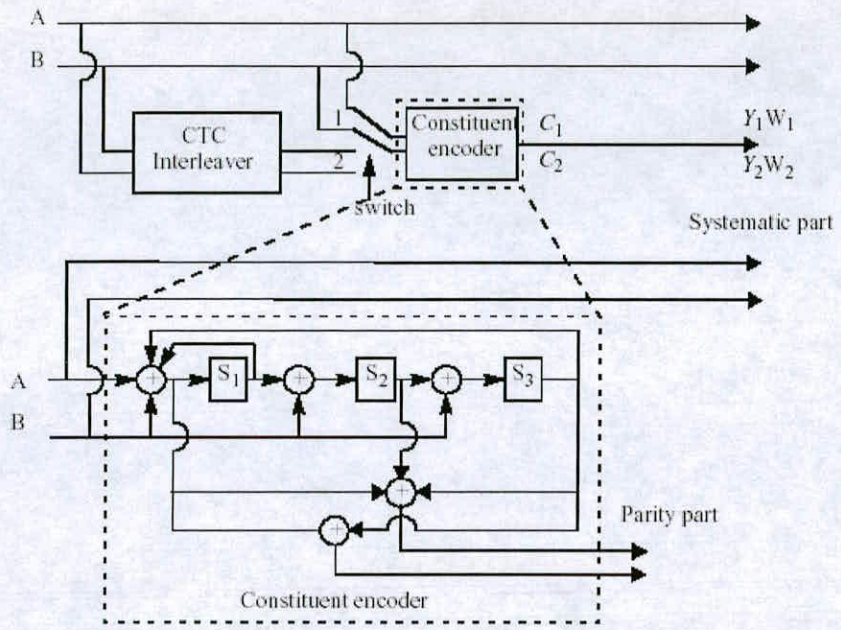



Figure C.1 CTC encoder

Note that the interleaver shall not be used when using CTC.

The encoding block size shall depend on the number of subchannels allocated and the modulation specified for the current transmission. Concatenation of a number of subchannels shall be performed in order to make larger blocks of coding where it is possible, with the limitation of not passing the largest block under the same coding rate (the block defined by 64-QAM modulation). Table C.1 specifies the concatenation of subchannels for different allocations and modulations. The concatenation rule shall not be used when using H-ARQ.

For any modulation and FEC rate, given an allocation of $n$ subchannels, the following parameters are defined:

$j$ parameter dependent on the modulation and FEC rate

$n$ number of allocated subchannels

$$k = \text{floor}(n/j)$$

$$m = n \bmod j$$

Table C.1 shows the rules used for subchannel concatenation :

Table C.1 Subchannel concatenation rule for CTC

| **Number of subchannels** | **Subchannels concatenated** |
|---|---|
| $n \leq j$<br><br>$n \neq 7$ | 1 block of n subchannels |
| $n = 7$ | 1 block of 4 subchannels<br>1 block of 3 subchannels |
| $n > j$ | (k-1) blocks of j subchannels<br>1 block of $L_{b1}$ subchannels<br>1 block of $L_{b2}$ subchannels<br><br>Where:<br>$L_{b1}$ = ceil($(m+j)/2$)<br>$L_{b2}$ = floor($(m+j)/2$)<br><br>If ($L_{b1} = 7$) or ($L_{b2} = 7$)<br>$Lb1 = Lb1 + 1$; $Lb2 = Lb2 - 1$; |

Table C.2 Encoding subchannel concatenation for different rates in CTC

| **Modulation and rate** | $j$ |
|---|---|
| QPSK 1/2 | 10 |
| QPSK 3/4 | 6 |
| 16-QAM 1/2 | 5 |
| 16-QAM 3/4 | 3 |
| 64-QAM 1/2 | 3 |
| 64-QAM 2/3 | 2 |
| 64-QAM 3/4 | 2 |
| 64-QAM 5/6 | 2 |

Table C.3 Optimal CTC channel coding per modulation

| Modulation | Data block size (bytes) | Encoded data block size (bytes) | Code rate | N | $P_0$ | $P_1$ | $P_2$ | $P_3$ |
|---|---|---|---|---|---|---|---|---|
| QPSK | 6 | 12 | 1/2 | 24 | 5 | 0 | 0 | 0 |
| QPSK | 12 | 24 | 1/2 | 48 | 13 | 24 | 0 | 24 |
| QPSK | 18 | 36 | 1/2 | 72 | 11 | 6 | 0 | 6 |
| QPSK | 24 | 48 | 1/2 | 96 | 7 | 48 | 24 | 72 |
| QPSK | 30 | 60 | 1/2 | 120 | 13 | 60 | 0 | 60 |
| QPSK | 36 | 72 | 1/2 | 144 | 17 | 74 | 72 | 2 |
| QPSK | 48 | 96 | 1/2 | 192 | 11 | 96 | 48 | 144 |
| QPSK | 54 | 108 | 1/2 | 216 | 13 | 108 | 0 | 108 |
| QPSK | 60 | 120 | 1/2 | 240 | 13 | 120 | 60 | 180 |
| QPSK | 9 | 12 | 3/4 | 36 | 11 | 18 | 0 | 18 |
| QPSK | 18 | 24 | 3/4 | 72 | 11 | 6 | 0 | 6 |
| QPSK | 27 | 36 | 3/4 | 108 | 11 | 54 | 56 | 2 |
| QPSK | 36 | 48 | 3/4 | 144 | 17 | 74 | 72 | 2 |
| QPSK | 45 | 60 | 3/4 | 180 | 11 | 90 | 0 | 90 |
| QPSK | 54 | 72 | 3/4 | 216 | 13 | 108 | 0 | 108 |
| 16-QAM | 12 | 24 | 1/2 | 48 | 13 | 24 | 0 | 24 |
| 16-QAM | 24 | 48 | 1/2 | 96 | 7 | 48 | 24 | 72 |
| 16-QAM | 36 | 72 | 1/2 | 144 | 17 | 74 | 72 | 2 |
| 16-QAM | 48 | 96 | 1/2 | 192 | 11 | 96 | 48 | 144 |
| 16-QAM | 60 | 120 | 1/2 | 240 | 13 | 120 | 60 | 180 |
| 16-QAM | 18 | 24 | 3/4 | 72 | 11 | 6 | 0 | 6 |
| 16-QAM | 36 | 48 | 3/4 | 144 | 17 | 74 | 72 | 2 |
| 16-QAM | 54 | 108 | 3/4 | 216 | 13 | 108 | 0 | 108 |
| 64-QAM | 18 | 24 | 1/2 | 72 | 11 | 6 | 0 | 6 |
| 64-QAM | 36 | 72 | 1/2 | 144 | 17 | 74 | 72 | 2 |
| 64-QAM | 54 | 108 | 1/2 | 216 | 13 | 108 | 0 | 108 |
| 64-QAM | 24 | 36 | 2/3 | 96 | 7 | 48 | 24 | 72 |
| 64-QAM | 48 | 72 | 2/3 | 192 | 11 | 96 | 48 | 144 |
| 64-QAM | 27 | 36 | 3/4 | 108 | 11 | 54 | 56 | 2 |

Table C.3 Optimal CTC channel coding per modulation *(continued)*

| Modulation | Data block size (bytes) | Encoded data block size (bytes) | Code rate | N | $P_0$ | $P_1$ | $P_2$ | $P_3$ |
|---|---|---|---|---|---|---|---|---|
| 64-QAM | 54 | 72 | 3/4 | 216 | 13 | 108 | 0 | 108 |
| 64-QAM | 30 | 36 | 5/6 | 120 | 13 | 60 | 0 | 60 |
| 64-QAM | 60 | 72 | 5/6 | 240 | 13 | 120 | 60 | 180 |

Table C.3 gives the block sizes, code rates, channel efficiency, and code parameters for the different modulation and coding schemes. As 64-QAM is optional, the codes for this modulation shall only be implemented if the modulation is implemented. Table C.4 shows code parameters for HARQ.

Table C.4 Optimal CTC channel coding per modulation when supporting H-ARQ

| Data block size (bytes) | N | P0 | P1 | P2 | P3 |
|---|---|---|---|---|---|
| 6 | 24 | 5 | 0 | 0 | 0 |
| 12 | 48 | 13 | 24 | 0 | 24 |
| 18 | 72 | 11 | 6 | 0 | 6 |
| 24 | 96 | 7 | 48 | 24 | 72 |
| 36 | 144 | 17 | 74 | 72 | 2 |
| 48 | 192 | 11 | 96 | 48 | 144 |
| 60 | 240 | 13 | 120 | 60 | 180 |
| 120 | 480 | 13 | 240 | 120 | 360 |
| 240 | 960 | 13 | 480 | 240 | 720 |
| 360 | 1440 | 17 | 720 | 360 | 540 |
| 480 | 1920 | 17 | 960 | 480 | 1440 |
| 600 | 2400 | 17 | 1200 | 600 | 1800 |

## C.2 CTC interleaver

The interleaver requires the parameters $P_0$ and $P_1$, shown in Table C.4.

The two-step interleaver shall be performed by:

**Step 1: Switch alternate couples**

for $j = 0...N - 1$

if ( $j_{\text{mod2}}$==0) let $(B,A) = (A,B)$ (i.e., switch the couple)

**Step 2: $P_i(j)$**

The function $P_i(j)$ provides the interleaved address $i$ of the consider couple $j$.

for $j = 0...N - 1$

switch $j_{\text{mod4}}$ :

case 0: $i = (P_0 . j + 1)_{\text{mod}N}$

case 1: $i = (P_0 . j + 1 + N/2 + P_1)_{\text{mod}N}$

case 2: $i = (P_0 . j + 1 + P_2)_{\text{mod}N}$

case 3: $i = (P_0 . j + 1 + N/2 + P_3)_{\text{mod}N}$

## C.3 Determination of CTC circulation states

The state of the encoder is denoted $S$ ($0 \le S \le 7$) with $S$ the value read binary (left to right) out of the constituent encoder memory. The circulation states $Sc1$ and $Sc2$ are determined by the following operations:

1) Initialize the encoder with state 0. Encode the sequence in the natural order for the determination of $Sc1$ or in the interleaved order for determination of $Sc2$. In both cases the final state of the encoder is $S0_{N-1}$;

2) According to the length $N$ of the sequence, use Table C.5 to find $Sc1$ or $Sc2$.

Table C.5 Circulation state lookup table (*Sc*)

| $N_{mod_3}$ | $S0_{N-1}$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| 1 | 0 | 6 | 4 | 2 | 7 | 1 | 3 | 5 |
| 2 | 0 | 3 | 7 | 4 | 5 | 6 | 2 | 1 |
| 3 | 0 | 5 | 3 | 6 | 2 | 7 | 1 | 4 |
| 4 | 0 | 4 | 1 | 5 | 6 | 2 | 7 | 3 |
| 5 | 0 | 2 | 5 | 7 | 1 | 3 | 4 | 6 |
| 6 | 0 | 7 | 6 | 1 | 3 | 4 | 5 | 2 |

# References


[1] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*, John Wiley and Sons,1999.

[2] L. Yuan, L. Hyunseok, M. Woh, Y. Harel, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner, "SODA: A Low-power Architecture For Software Radio," in *International Symposium on Computer Architecture*, pp. 89 – 101, June 2006.

[3] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1," in *IEEE International Conference on Communications*, vol. 2, pp. 1064-1070, May 1993

[4] P. Cherriman, T. Keller, and L. Hanzo, "Constant-rate turbo-coded orthogonal frequency division multiplex videophony over UMTS," in *IEEE Global Telecommunications Conference,* vol. 5, pp. 2848 – 2852, Nov. 1998.

[5] A. Gueguen, and D. Castelain, "Performance of frame oriented turbo codes on UMTS channel with various termination schemes," in *IEEE Vehicular Technology Conference*, vol. 3, pp. 1550 – 1554, Sep. 1999.

[6] Z. Jinyun, L. Ling, and T. Poon, "Turbo coded HSDPA systems with transmit diversity over frequency selective fading channels," in *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, vol. 1, pp. 90 – 94, Sep. 2002.

[7] M. C. Valenti, "Inserting turbo code technology into the DVB satellite broadcasting system," in *Military Communications Conference*, vol. 2, pp. 650 – 654, Oct. 2000.

[8] A. Bartolazzi, G. Cardarilli, A. Del Re, D. Giancristofaro, and M. Re, "Implementation of DVB-RCS turbo decoder for satellite on-board processing," in *IEEE International Conference on Circuits and Systems for Communications*, pp. 142 – 145, June 2002.

[9] L. N. Lee, A. R. Hammons Jr., S. Feng-Wen, and M. Eroz, "Application and standardization of turbo codes in third-generation high-speed wireless data services," *IEEE Transactions on Vehicular Technology*, vol. 49, pp. 2198 – 2207, Nov. 2000.

[10] F. Berens, A. Worm, H. Michel, and N. Wehn, "Implementation aspects of turbo-decoders for future radio applications," in *IEEE Vehicular Technology Conference*, vol. 5, pp. 2601 – 2605, Sep. 1999.

[11] L. Seok-Jun, N. R. Shanbhag, and A. C. Singer, "285-MHz pipelined MAP decoder in 0.18-/spl mu/m CMOS," *IEEE J. of Solid-State Circuits,* vol. 40, pp. 1718 – 1725, Aug. 2005.

[12] P. Urard, L. Paumier, M. Viollet, E. Lantreibecq, H. Michel, S. Muroor, B. Coates, B. Gupta, "A generic 350 Mb/s turbo-codec based on a 16-states SISO decoder," *IEEE Int. Solid-State Circuits Conf. Digest of Technical Papers,* pp. 424 – 536, Feb. 2004.

[13] M. A. Bickerstaff, D. Garrett, T. Prokop, C. Thomas, B. Widdup, Z. Gongyu, L. M. Davis, G. Woodward, C. Nicol, and Y. Ran-Hong "A unified turbo/Viterbi channel

decoder for 3GPP mobile wireless in 0.18-/spl mu/m CMOS," *IEEE J. of Solid-State Circuits,* vol. 37, no. 11, pp. 1555 - 1564, Nov. 2002.

[14] C. Thomas, M. A. Bickerstaff, L. M. Davis, T. Prokop, B. Widdup, Z. Gongyu, D. Garrett, C. Nicol, "Integrated circuits for channel coding in 3G cellular mobile wireless systems," *IEEE Communications Magazine,* vol. 41, no. 8, pp. 150 – 159, Aug. 2003.

[15] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," in *IEEE International Conference on Communications*, vol. 2, pp. 1009-1013, June 1995.

[16] H. Dawid, and H. Meyer, "Real-time algorithms and VLSI architectures for soft output MAP convolutional decoding," in *IEEE International Symposium on* Personal, Indoor and Mobile Radio Communications, vol. 1, pp. 193-197, Sep. 1995.

[17] O. Joeressen, M. Vaupel, and H. Meyr, "High-speed VLSI architectures for soft-output Viterbi decoding," in *IEEE International Conference on Application Specific Array Processors*, pp. 373 – 384, Aug. 1992.

[18] T. Miyauchi, K. Yamamoto, and T. Yokokawa, "High-performance programmable SISO decoder VLSI implementation for decoding turbo codes," in *Proc. IEEE Global Telecommunications Conf.*, vol. 1, 2001, pp. 305–309.

[19] M. El-Assal and M. Bayoumi, "A high-speed architecture for MAP decoder," in *Proc. IEEE Signal Processing Systems (SiPS): Design and Implementation*, pp. 69–74, Oct. 2002.

[20] M. Bickerstaff, L. Davis, C. Thomas, D. Garret, and C. Nicol, "A 24 Mb/s radix-4 logMAP turbo decoder for 3GPP-HSDPA mobile wireless," *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, pp. 150–151, 2003.

[21] Z. Wang, Z. Chi, and K. Parhi, "Area-efficient high-speed decodingschemes for turbo decoders," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 10, no. 6, pp. 902–912, Dec. 2002.

[22] J. Hsu and C. Wang, "A parallel decoding scheme for turbo codes," in *Proc. IEEE Int. Conf. Circuits and Systems*, vol. 4, 1998, pp. 445–448.

[23] B. Bougard, A. Ciulietti, L. V. d. Perre, and F. Catthoor, "A class ofpower efficient VLSI architectures for high speed turbo-decoding," in *Proc. IEEE Global Telecommunication Conf.*, vol. 1, 2002, pp. 553–549.

[24] B. Bougard *et al.*, "A scalable 8.7 nJ/bit 75.6 Mb/s parallel concatenatedconvolutional (turbo-) codec," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, 2003, pp. 152–153.

[25] A. Worm, H. Lamm, and N. Wehn, "A high-speed MAP architecturewith optimized memory size and power consumption," in *Proc. IEEEWorkshop on Signal Processing Systems (SiPS2000)*, 2000, pp. 265–274.

[26] A. Worm, H. Lamm, and N. Wehn,, "Design of low-power high-speed maximum *a priori* decoder architectures," in *Proc. Design, Automation and Test in Eur. Conf. Exhibition*, Apr. 2001, pp. 258–265.

[27] M. M. Mansour and N. R. Shanbhag, "Design methodology for highspeed iterative decoder architectures," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, vol. 3, 2002, pp. 3085–3088.

[28] M. M. Mansour and N. R. Shanbhag, "VLSI architectures for SISO-APP decoders," *IEEE Trans. Very Large Scale Integration (VLSI) Syst.*, vol. 11, no. 4, pp. 627–650, Aug. 2003.

[29] D. Garrett, B. Xu, and C. Nicol, "Energy efficient turbo decoding for 3G mobile," in *Proc. IEEE Int. Symp. Low Power Electronics Design (ISLPED'01)*, 2001, pp. 328–333.

[30] O. Leung, C. Yue, C. Tsui, and R. Cheng, "Reducing power consumption of turbo code decoder using adaptive iteration with variable supply voltage," in *Proc. IEEE Int. Symp. Low Power Electronics Design (ISLPED'99)*, 1999, pp. 36–41.

[31] P. H. Wu and S. M. Pisuk, "Implementation of a low complexity, low power, integer-based turbo decoder," in *Proc. Global Telecommunications Conf.*, vol. 2, 2001, pp. 946–951.

[32] S. Lee, N. Shanbhag, and A. Singer, "Low-power turbo equalizer architecture," *Proc. IEEE Signal Processing Systems (SiPS): Design and Implementation*, pp. 33–38, Oct. 2002.

[33] W. Zhongfeng and K. K. Parhi, "High performance, high throughput turbo/SOVA decoder design," *IEEE Trans. on Communications,* vol. 51, no. 4, pp. 570 –579, April 2003.

[34] Y. Engling; S. A. Augsburger, and W. R. Davis, and B. A. Nikolic, "A 500-Mb/s soft-output Viterbi decoder," *IEEE J. of Solid-State Circuits,* vol. 38, no. 7, pp. 1234 – 1241, July 2003.

[35] M. El-Assal, and M. Bayoumi, "Low power SOVA architecture using bi-directional scheme," in *Proc. IEEE Int. Symp. on Circuits and Systems,* vol. 1, 26-29 May 2002 pp. I-277 - I-280.

[36] D. Garrett, and M. Stan, "A 2.5 Mb/s, 23 mW SOVA traceback chip for turbo decoding applications," in *Proc. IEEE Int. Symp. on Circuits and systems,* vol. 4, 6-9 May 2001, pp. 61-64.

[37] W. Wang, T. Chi-Ying, and R. S. Cheng, "A low power VLSI architecture of SOVA-based turbo-code decoder using scarce state transition scheme," in *Proc. IEEE Int. Symp. on Circuits and Systems,* vol. 1, 28-31 May 2000, pp. 283 – 286.

[38] D. Garrett and M. Stan, "Low power architecture of the soft-output Viterbi algorithm," in *Proc. IEEE Int. Symp. Low Power Electronics Design*, 1998, pp. 262–267.

[39] E. Yeo, and S. A. Augsburger, W. R. Davis, and B. Nikolic, "Implementation of high throughput soft output Viterbi decoders," in *IEEE Workshop on Signal Processing Systems,* 16-18 Oct. 2002, pp. 146 – 151.

[40] A. Ghrayeb, and W. E. Ryan, "Performance of high rate turbo codes employing the soft-output Viterbi algorithm (SOVA)," in *Asilomar Conf. on Signals, Systems, and Computers,* vol. 2, 24-27 Oct. 1999, pp. 1665 – 1669.

[41] W. Zhongfeng, H. Suzuki, and K. K. Parhi, "Efficient approaches to improving performance of VLSI SOVA-based turbo decoders," in *Proc. IEEE Int. Symp. on Circuits and Systems,* vol. 1, 28-31 May 2000, pp. 287 – 290.

[42] W. Duanyi, and H. Kobayashi, "High-performance SOVA decoding for turbo codes over cdma2000 mobile radio," in *Proc. Military Communications Conference MILCOM 2000,* vol. 1, 22-25 Oct. 2000, pp. 189 – 193.

[43] A. La Rosa, L. Lavagno, and C. Passerone, "Implementation of a UMTS turbo decoder on a dynamically reconfigurable platform," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems,* vol. 24, no. 1, pp. 100 – 106, Jan. 2005.

[44] C. Chaikalis, and J. M. Noras, "Implementation of an improved reconfigurable SOVA/log-MAP turbo decoder" in *Proc. Int. Conf. on 3GPP3G Mobile Communication Technologies,* 8-10 May 2002, pp. 146 – 150.

[45] I. Ahmed, and T. Arslan, "A Reconfigurable Viterbi Traceback for Implemenation on Turbo Decoding Array," in *Proc. IEEE Int. SOC Conf.,* Sept. 2006, pp. 107 – 108.

[46] I. Ahmed, T. Arslan, S. Baloch, I. Underwood, R. Woodburn, "Domain Specific Reconfigurable Architecture of Turbo Decoder Optimized for Short Distance Wireless Communication," in *Proc. IEEE Int. Parallel and Distributed Processing Symp.,* 4-8 April 2005, pp. 166b - 166b.

[47] C. Thomas, M. A. Bickerstaff, L. M. Davis, T. Prokop, B. Widdup, Z. Gongyu, D. Garrett, and C. Nicol, "Integrated circuits for channel coding in 3G cellular mobile wireless systems," *IEEE Communications Magazine,* vol. 41, no. 8, pp. 150 – 159, Aug. 2003.

[48] S. Lingyan, T. Horigome, and B. V. K. V. Kumar, "A high-throughput, field programmable gate array implementation of soft output Viterbi algorithm for magnetic recording," *IEEE Trans. on Magnetics,* vol. 40, no. 4, pp. 3081 - 3083, July 2004.

[49] L. Cheng-Hung, L. Fan-Min, S. Xin-Yu Shi, and W. An-Yeu, "A Triple-Mode MAP/VA IP Design for Advanced Wireless Communication Systems," *Asian Solid-State Circuits Conf.*, Nov. 2005, pp. 221 – 224.

[50] L. Fan-Min, S. Pei-Ling, and W. An-Yeu, "Unified convolutional/turbo decoder architecture design based on triple-mode MAP/VA kernel," in *Proc. IEEE Asia-Pacific Conf. on Circuits and Systems,* vol. 2, 6-9 Dec. 2004, pp. 1073 – 1076.

[51] H. Kai, L. Fan-Min, S. Pei-Ling, an W. An-Yeu, "VLSI design of dual-mode Viterbi/turbo decoder for 3GPP," in *Proc. Int. Symp. on Circuits and Systems,* vol. 2, 23-26 May 2004, pp. II - 773-776.

[52] C. Pen-Hsin, Kai-Huang, H. Nai-Hsuan, W. An-Yeu, "Dual-mode convolutional/ SOVA based turbo code decoder VLSI design for wireless communication systems," in *Proc. IEEE Int. SOC Conf.,* 17-20 Sept. 2003, pp. 369 – 372.

[53] J. R. Cavallaro, and M. Vaya, "Viturbo: a reconfigurable architecture for Viterbi and turbo decodingAcoustics," *in Proc. IEEE Int. Conf. on Speech, and Signal Processing,* vol. 2, 6-10 April 2003, pp. II - 497-500.

[54] C. E. Shannon, "A Mathematical Theory of Communication," *The Bell System Technical Journal,* Vol. 27, pp. 379–423, 623–656, July, October, 1948. By

[55] J. Hagenauer, "The Turbo principle : Tutorial Introduction and State of the Art," in *Symposium on Turbo-Codes*, pp. 1-11, Sep. 1997.

[56] C. Berrou, and A. Glavieux, "Near optimum error correcting coding and decoding: turbo-codes." *IEEE Transactions on Communications*, vol. 44, pp. 1261 – 1271, Oct. 1996.

[57] A. J. Viterbi, "An intuitive justification and a simplified implementation of the MAP decoder for convolutional codes," *IEEE Journal on Selected Areas in Communications*, vol. 16, pp. 260 – 264, Feb. 1998.

[58] J. P. Woodard, and L. Hanzo, "Comparative study of turbo decoding techniques: an Overview," *IEEE Transactions on Vehicular Technology*, vol. 49, pp. 2208 – 2233, Nov. 2000.

[59] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Transactions on Information Theory*, vol. 42, pp. 429 - 445, Mar. 1996.

[60] "Guest Editorial the turbo principle: from theory to practice," *IEEE Journal on Selected Areas in Communications*, vol.19, May 2001.

[61] Japan's Proposal for Candidate Radio Transmission Technology on IMT-2000: W-CDMA. Japan Ass. of Radio Industries and Business (ARIB), http://www.arib.or.jp/IMT-2000/proponent

[62] 3GPP TS 25.212, *Multiplexing and channel coding (FDD)*, v.4.6.0, pp. 16-21, Sep. 2005.

[63] Consultative committee for space data systems (CCSDS), *Telemetry Channel Coding*, Blue Book 101.0-B-4, pp. 305–309, 1999.

[64] IEEE 802.16-2004, *Air Interface for Fixed Broadband Wireless Access Systems*, pp. 594-599.

[65] ETSI EN 390-790, *Digital Video Broadcasting (DVB); Interaction channel for satellite distribution systems*, v.2.4.1, pp. 23-26, Sep. 2005.

[66] L.R. Bahl, J. Cocke, F. Jelinek and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Transaction Information Theory*, IT-20, pp. 248-287, Mar. 1974.

[67] J. Hagenauer and P. Hoeher, "A Viterbi algorithm with soft-decision outputs and its applications," in *IEEE Global Telecommunications Conference*, vol. 3, pp. 1680 – 1686, Nov. 1989.

[68] L. Hanzo, T. H. Liew, and B. L. Yeap, *Turbo Coding, Turbo Equalisation, and Space-Time Coding for Transmission Over Fading Channels*, John Wiley and Sons Ltd., 2005.

[69] M. Bossert, *Channel Coding for Telecommunications*, .John Wiley and Sons Ltd., 1999.

[70] J. G. Proakis, *Digital Communications*, McGraw-Hill, 2001.

[71] S. Benedetto, D. Divsalar, G. Montorsi and F. Pollara, "Serial Concatenation of Interleaved Codes: Performance Analysis, Design, and Iterative Decoding," *The JPL TDA Progress Report*, pp. 42-126, Aug. 1996.

[72] W. J. Blackert, E. K. Hall, and S. G. Wilson, "Turbo code termination and interleaver conditions," *Electronics Letters*, vol. 31, pp. 2082 - 2084, Nov. 1995.

[73] L. Lang, and R. S. Cheng, "On the tail effect of SOVA-based decoding for turbo codes," in *IEEE Global Telecommunications Conference*, vol. 2, pp. 644 - 648, Nov. 1997.

[74] C. Weiss, C. Bettstetter, S. Riedel, and D. J. Costello Jr., "Turbo decoding with tail-biting trellises," in *International Symposium on Signals, Systems, and Electronics,* pp. 343 – 348, Sep. 1998.

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

[76] C. Douillard, and C. Berrou, "Turbo codes with rate-m/(m+1) constituent convolutional codes," *IEEE Transactions on Communications*, vol. 53, pp. 1630 – 1638, Oct. 2005.

[77] C. Berrou, "The ten-year-old turbo codes are entering into service," *IEEE Communications Magazine*, vol. 41, pp. 110 - 116, Aug. 2003.

[78] C. Berrou, M. Jezequel, C. Douillard, and S. Kerouedan, "The advantages of non-binary turbo codes," in *IEEE Information Theory Workshop*, pp. 61 – 63, Sep. 2001.

[79] C. Berrou, C. Douillard, and M. Jezequel, "Designing turbo codes for low error rates," in *IEE Colloquium*, pp. 6-7, Nov. 1999.

[80] S. Benedetto, and G. Montorsi, "Performance evaluation of turbo-codes," *Electronics Letters*, vol. 31, pp. 163 – 165, Feb. 1995.

[81] S. Benedetto, and G. Montorsi, "Design of parallel concatenated convolutional codes," *IEEE Transactions on Communications*, vol. 44, pp. 591 – 600, May 1996.

[82] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Analysis, design, and iterative decoding of double serially concatenated codes with interleavers," *IEEE Journal on Selected Areas in Communications*, vol. 16, pp. 231 – 244, Feb. 1998.

[83] J. G. Proakis, *Digital Communications*, McGraw-Hill, 2001.

[84] P. Gulak, and E. Shwedyk, "Viterbi decoder VLSI Structures for Viterbi Receivers: Part I--General Theory and Applications," *IEEE Journal on Selected Areas in Communications,* vol. 4, pp. 142 - 154, Jan. 1986.

[85] G. Feygin, and P. Gulak, "Architectural tradeoffs for survivor sequence memory management in Viterbi decoders," *IEEE Transactions on Communications*, vol. 41, pp. 425 – 429, March 1993.

[86] G. Feygin, P. Chow, P. G. Gulak,. J. Chappel, G. Goodes, O. Hall, A. Sayes, S. Singh, M. B. Smith, and S. Wilton, "A VLSI implementation of a cascade Viterbi decoder with traceback," in *IEEE International Symposium on Circuits and Systems*, vol. 3, pp. 1945 – 1948, May 1993

[87] C. Shiunn-Jang, and H. Li-Da, "A VLSI architecture of SMU for strongly connected Viterbi decoder," in *IEEE Asia-Pacific Conference on Circuits and Systems*, pp. 200 – 205, Dec. 1994.

[88] C. Yun-Nan, H. Suzuki, and K. K. Parhi, "A 2-Mb/s 256-state 10-mW rate-1/3 Viterbi decoder," *IEEE Journal of Solid-State Circuits*, vol. 35, pp. 826 – 834, June 2000.

[89] G. Montorsi, and S. Benedetto, "Design of fixed-point Iterative Decoders for Concatenated Codes with Interleavers," *IEEE Journal on Selected Areas in Communication*, vol. 19, pp. 871-882, May 2001.

[90] W. Chien-Ming, S. Ming-Der, W. Chien-Hsing H. Yin-Tsung, and C. Jun-Hong, "VLSI architectural design tradeoffs for sliding-window log-MAP decoders," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 13, pp. 439 – 447, April 2005.

[91] N. Engin, "A turbo decoder architecture with scalable parallelism," in *IEEE Workshop on Signal Processing Systems*, pp. 298 – 303, Oct. 2004

[92] E. Boutillon, W. J. Gross, and, P. G. Gulak, "VLSI architectures for the MAP algorithm," *IEEE Transactions on Communications*, vol. 51, pp. 175 – 185, Feb. 2003.

[93] O. Joeressen, M. Vaupel, and H. Meyr, "High-speed VLSI architectures for soft-output Viterbi decoding," in *IEEE International Conference on Application Specific Array Processors*, pp. 373 – 384, Aug. 1992.

[94] Z. Wang, and K. K. Parhi, "High performance, high throughput turbo/SOVA decoder design," *IEEE Transactions on Commun.*, vol. 51, no. 4, pp. 570 – 579, April 2003.

[95] Y. N. Chang, "Design of soft-output Viterbi decoders with hybrid trace-back processing," in *Proc. Int. IEEE Symp. Circuits and* Systems, vol. 2, May 2003, pp. II-69 - II-72.

[96] C. Berrou, P. Combelles, P. Penard, and B. Talibart, "An IC for turbo-codes encoding and decoding," in *Digest of Tech. Papers IEEE Int.* Solid-State Circuits Conf., Feb. 1995, pp. 90 – 91

[97] C. Ghrayeb, and X. Huang, "Improvements in SOVA-based decoding for turbo-coded storage channels," *IEEE Trans. on Magnetics*, vol. 41, no. 12, pp. 4435 – 4442, Dec. 2005.

[98] M. El-Assal, and M. Bayoumi, "Low power SOVA architecture using bi-directional scheme," in *Proc. Int. IEEE Symp. Circuits and Systems,* vol. 1, May 2002. pp. I-277 - I-280.

[99] W. Yufei, and B. D. Woerner, "The influence of quantization and fixed point arithmetic upon the BER performance of turbo codes," in IEEE *Vehicular Technology Conference*, vol. 2, pp. 1683 – 1687, May 1999.

[100] H. Michel, A. Worm, and N. Wehn, "Influence of quantization on the bit-error performance of turbo-decoders," in *IEEE Vehicular Technology Conference*, vol. 1, pp. 581 - 585, May 2000.

[101] T. K. Blankenship, and B. Classon, "Fixed-point performance of low-complexity turbo decoding algorithms," in *IEEE Vehicular Technology Conference*, vol. 2, pp. 483 – 1487, May 2001.

[102] J. Gibong, and H. Dan, "Optimal quantization for soft-decision turbo decoder," in *IEEE Vehicular Technology Conference*, vol. 3, pp. 1620 - 1624, Sep. 1999.

[103] H. Michel, and N. Wehn, "Turbo-decoder quantization for UMTS," *IEEE* Communications Letters, vol. pp. 55 – 57, Feb. 2001.

[104] L. Xizhong, M. Zhigang, and C. Yanmin, "Quantization issues in turbo decoding Communications," in International Conference on Communications, Circuits and Systems, vol. 1, pp. 35 – 39, May 2005.

[105] Z. Wang, H. Suzuki, and K. K. Parhi, "VLSI implementation issues of TURBO decoder design for wireless applications," in IEEE Workshop on Signal Processing Systems, pp. 503-512, Oct. 1999.

[106] C. B. Shung, P. H. Siegel, G. Ungerboeck, and H. K. Thapar, "VLSI architectures for metric normalization in the Viterbi algorithm," in IEEE Conference Communication, pp. 1723 – 1728, April 1990.

[107] L. Seok-Jun, N. R. Shanbhag, and A. C. Singer, "Area-efficient high-throughput VLSI architecture for MAP-based turbo equalizer," in *IEEE Workshop on Signal Processing Systems*, pp. 87 – 92, Aug. 2003.

[108] T. Gemmeke, M. Gansen, and T. G. Noll, "Implementation of scalable power and area efficient high-throughput Viterbi decoders." *IEEE Journal Solid-State Circuits*, vol. 37, no. 7, pp. 941 – 948, July 2002.

[109] T. Gemmeke, V. S. Gierenz, and T. G. Noll, "Scalable, power and area efficient high throughput Viterbi decoder implementations," in *Proceedings European Solid-State Circuits Conference*, pp. 474 – 477, Sep. 2001.

[110] P. J. Black, and T. H. Meng, "A 1-Gb/s, four-state, sliding block Viterbi decoder," *IEEE Journal of Solid-State Circuits,* vol. 32, no. 6, pp. 797 – 805, June 1997.

[111] O. Y. Leung, T. Chi-Ying, and R. S. Cheng, "Reducing power consumption of turbo-code decoder using adaptive iteration with variable supply voltage," *IEEE Transactions on VLSI Systems*, vol. 9, no. 1, pp. 34-41, Feb. 2001.

[112] W. Zhongfeng, and K. K. Parhi, "On-line extraction of soft decoding information and applications in VLSI turbo decoding," *IEEE Transactions on* Circuits and Systems II: Analog and Digital Signal Processing, vol. 49, no. 12, pp. 760-769, Dec. 2002.

[113] G. Jian, Z. Yi, Y. Dacheng, and L. Zhen, "Adaptive Iterative Turbo Decoding Algorithm," in *IEEE* Vehicular Technology Conference, vol. 3, pp. 1472-1476, 2006.

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

[115] A. K. Yeung, and J. M. Rabaey, "A 210 Mb/s radix-4 bit-level pipelined Viterbi decoder," in *IEEE International* Solid-State Circuits Conference, pp. 88 - 89, Feb. 1995.

[116] V. S. Gierenz, O. Weiss, T. G. Noll, I. Carew, J. Ashley, and R. Karabed, "A 550 Mb/s radix-4 bit-level pipelined 16-state 0.25-μm CMOS Viterbi decoder," in IEEE International Conference on Application-Specific Systems, Architectures, and Processors, pp. 342 – 345, Feb. 1999.

[117] T. Conway, "Implementation of high speed Viterbi detectors," Electronics Letters, vol. 35, pp. 2089 – 2090, Nov. 1999.

[118] C. Thomas, M. A. Bickerstaff, L. M. Davis, T. Prokop, B. Widdup, Z. Gongyu, D. Garrett, C. Nicol, "Integrated circuits for channel coding in 3G cellular mobile wireless systems," IEEE Commun. Mag., vol. 41, no. 8, pp. 150-159, Aug. 2003.

[119] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 429–445, Mar. 1996.

[120] M. Moher, "Decoding via cross entropy minimization," in *Proc. IEEEGlobecom Conf.*, Houston, TX, Dec. 1993, pp. 809–813.

[121] R. Y. Shao, S. Lin, and P. C. Marc, "Two Simple Stopping Criteria for Turbo Decoding," *IEEE Trans. on Communications*, vol. 7, no. 8, 1999

[122] M. Rovini, and A. Martinez, "Efficient stopping rule for turbo decoders," *Electronics Letters*, vol. 42, no. 4, pp. 235 – 236, 2006.

[123] L. Lei, W. Qin, and Y. L. Cheng, "A Novel Stopping Criterion for Turbo Decoding Innovative Computing," in *Int. Conf. Information and Control*, Aug. 2006, pp. 201 – 205.

[124] S. Byoung-Sup, J. Dae-Ho, L. Soon-Ja, and K. Hwan-Yong, "A new stopping criterion for turbo codes," in *Int. Conf. Advanced Communication Technology,* Feb. 2006, pp. 5.

[125] M. Zheng, F. Pingzhi, and H. M. Wai, "An effective stopping scheme for reduced-complexity iterative decoding of short-frame turbo codes," in *Proc. Int. Conf. Communications, Circuits and Systems,* May 2005, pp. 28 – 30.

[126] K. Gracie, S. Crozier, and P. Guinand, "Performance of an MLSE-based early stopping technique for turbo codes," in *IEEE Vehicular Technology Conference,* Sept. 2004, pp. 2287 – 2291.

[127] W. Zhongfeng, T. Yiyan, and W. Yuke, "Low hardware complexity parallel turbo decoder architecture," in *Proc. Int. Symp. Circuits and Systems,* May 2003, pp. II-53 - II-56.

[128] P. Ciao, G. Colavolpe, and L. Fanucci, "A parallel VLSI architecture for 1-Gb/s, 2048-b, rate-1/2 turbo Gallager code decoder," in *Euromicro Symp. on Digital System Design,* 31 Aug.-3 Sept. 2004, pp. 174 – 181.

[129] M. Mansour and N. R. Shanbhag, "VLSI architectures for SISO-APPdecoders," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, no. 2, pp. 627–650, Apr. 2003.

[130] R. Dobkin, M. Peleg, and R. Ginosar, "Parallel interleaver design and VLSI architecture for low-latency MAP turbo decoders," in *IEEE Trans. On Very Large Scale Integration (VLSI) Systems,* vol. 13, no. 4, pp. 427 – 438, 2005.

[131] L. Dinoi, and S. Benedetto, "Variable-size interleaver design for parallel turbo decoder architectures," *IEEE Trans. on Communications*, vol. 53, no. 11, pp. 1833 – 1840, 2005.

[132] L. Dinoi, A. Tarable, and S. Benedetto, "A permutation decomposition based algorithm for the design of prunable interleavers for parallel turbo decoder architectures," in *IEEE Int. Conf. Communications,* June 2006, pp. 1148 – 1153.

[133] A. Tarable, and S. Benedetto, "Mapping interleaving laws to parallel turbo decoder architectures," *IEEE Communications Letters,* vol. 8, no. 3, pp. 162 – 164, 2004.

[134] A. Giulietti, L. van der Perre, and M. Strum, "Parallel turbo coding interleavers: avoiding collisions in accesses to storage elements," *IEEE Commun. Lett.*, vol. 38, no. 2, pp. 232–234, Feb. 2002.

[135] A. Worm, H. Lamm, and N. Wehn, "VLSI architectures for high-speed MAP decoders," in *Proc. 14th Int. Conf. VLSI Design*, Jan. 2001, pp. 446–453.

[136] W. Zhongfeng, C. Zhipei, and K. K. Parhi, "Area-efficient high-speeddecoding schemes for turbo decoders," *IEEE Trans. Very Large ScaleIntegr. (VLSI) Syst.*, vol. 10, no. 6, pp. 902–912, Dec. 2002.

[137] J. Vogt, K. Koors, A. Finger, and G. Fettweis, "Comparison of different turbo decoder realizations for IMT-2000," in *IEEE Global Telecommunications Conference,* vol. 5, pp. 2704 – 2708, Dec. 1999.

[138] P. H. Wu, "On the complexity of turbo decoding algorithms," in *IEEE Vehicular Technology Conference,* vol. 2, pp. 1439 - 1443, May 2001.

[139] R. Cypher, and C. B. Shung, "Generalized trace back techniques for survivor memory management in the Viterbi algorithm," in *IEEE Global Telecommunications Conference*, vol. 2, pp. 1318 – 1322, Dec. 1990.

[140] A. Chandrakasan and R. Brodersen, *Low Power Digital CMOS design*, Kluwer, 1995.

[141] J. M. Rabaey and M. Pedram, Low power design methodologies, Kluwer, 1996.

[142] L. Seok-Jun, N. R. Shanbhag, and A. C. Singer, "A low-power VLSI architecture for turbo decoding," in *IEEE International Symposium on Low Power Electronics and Design,* pp. 366 – 371, Aug. 2003.

[143] G. Masera, M. Mazza, G. Piccinini, F. Viglione, and M. Zamboni, "Architectural strategies for low-power VLSI turbo decoders," *IEEE Transactions on* Very Large Scale Integration Systems, vol. 10, pp. 279 – 285, June 2002.

[144] D. Garrett, X. Bing, and C. Nicol, "Energy efficient turbo decoding for 3G mobile," in IEEE *International Symposium on* Low Power Electronics and Design, pp. 328 – 333, Aug. 2001.

[145] O. Y. Leung, T. Chi-Ying, and R. S. Cheng, "Reducing power consumption of turbo-code decoder using adaptive iteration with variable supply voltage," *IEEE Transactions on* Very Large Scale Integration Systems, vol. 9, pp. 34 – 41, Feb. 2001.

[146] M. Elassal, and M. Bayoumi, "A low power turbo decoder architecture," in *IEEE Workshop on* Signal Processing Systems, pp. 105 – 110, Aug. 2003.

[147] W. Zhongfeng, C. Zhipei and K. K. Parhi, "Area-efficient high speed decoding schemes for turbo/MAP decoders," in *IEEE International Conference on* Acoustics, Speech, and Signal Processing, vol. 4, pp. 2633 - 2636, May 2001.

[148] K. K. Parhi, "An Improved Pipelined MSB-First Add-Compare-Select Unit Structure for Viterbi Decoders," *IEEE Trans. On Circuits and Systems-I:Reg. Papers*, vol. 51, no. 3, pp. 504-511, Mar. 2004.

[149] J. Hagenauer, and L. Papke, "Decoding turbo-codes with the soft output Viterbi algorithm (SOVA)," in *IEEE* International *Symposium on Information Theory*, pp. 164, June 1994.

[150] M. P. C. Fossorier, F. Burkert, L. Shu, and J. Hagenauer, "On the equivalence between SOVA and max-log-MAP decodings," *IEEE Communications Letters*, vol. 2, pp. 137 – 139, May 1998.

[151] G. Colavolpe, G. Ferrari, and R. Raheli, "Extrinsic information in turbo decoding: a unified view," in *IEEE Global Telecommunications Conference*, vol. 1a, pp. 505 – 509, 1999.

[152] C. Berrou, P. Adde, E. Angui, and S. Faudeil, "A low complexity soft-output Viterbi decoder architecture," in *IEEE International Conference on Communications*, vol. 2, pp. 737 – 740, May 1993.

[153] L. Papke, P. Robertson, and E. Villebrun, "Improved decoding with the SOVA in a parallel concatenated (Turbo-code) scheme," in *IEEE International Conference on Communications*, vol. 1, pp. 102 – 106, June 1996.

[154] L. Lang, and R. S. Cheng, "Improvements in SOVA-based decoding for turbo codes," in *IEEE International Conference on Communications*, vol. 3, pp. 1473 – 1478, June 1997.

[155] W. Duanyi, and H. Kobayashi, "High-performance SOVA decoding for turbo codes over cdma2000 mobile radio," in *Military Communications Conference*, vol. 1, pp. 189 – 193, Oct. 2000.

[156] Y. Fahmy, H. A. G. Abdel Kader, and M. M. S. El-Soudani, "On the use of SOVA for iterative decoding," in *Mediterranean Electrotechnical Conference*, pp. 168 – 172, May 2002.

[157] S. Lingyan, T. Horigome, and B. V. K. V. Kumar, "A high-throughput, field programmable gate array implementation of soft output Viterbi algorithm for magnetic recording," *IEEE Transactions on Magnetics*, vol. 40, pp. 3081 – 3083, July 2004.

[158] D. Garrett, and M. Stan, "Low power architecture of the soft-output Viterbi algorithm," in *IEEE International Symposium on Low Power Electronics and Design*, pp. 262 – 267, Aug. 1998.

[159] K. W. Tae, B. S. Jae, K. S. Geun, and J. E. Kyeong, "Reduction of computational complexity in two-step SOVA decoder for turbo code," in *IEEE Global Telecommunications Conference*, vol. 3, pp. 1887 – 1891, Nov. 2000.

[160] D. Garrett, and M. A. Stan, "2.5 Mb/s, 23 mW SOVA traceback chip for turbo decoding applications," in *IEEE International Symposium on Circuits and Systems*, vol. 4, pp. 61 – 64, May 2001.

[161] C. Berrou, P. Combelles, P. Penard, and B. Talibart, "An IC for turbo-codes encoding and decoding," in *IEEE International Solid-State Circuits Conference*, pp. 90 – 91, Feb. 1995.

[162] J. Glossner, D. Iancu, L. Jin, E. Hokenek, and M. Moudgill, "A software-defined communications baseband design," *IEEE Communications Magazine*, vol. 41, pp. 120 – 128, Jan. 2003.

[163] M. Barnard, and S. McLaughlin, "Reconfigurable terminals for mobile communication systems," *Journal of Electronics & Communication Engineering*, vol. 12, pp. 281 – 292, Dec. 2000.

[164] W. H. W. Tuttlebee, "Software-defined radio: facets of a developing technology," *IEEE Personal Communications*, vol. 6, pp. 38 – 44, April 1999.

[165] M. Mehta, N. Drew, G. Vardoulias, N. Greco, and C. Niedermeier, "Reconfigurable terminals: an overview of architectural solutions," *IEEE Communications Magazine,* vol. 39, pp. 82 – 89, Aug. 2001.

[166] T. C. Reiner, and M. J. Lindsey, "VLSI development of a reconfigurable multi-user Viterbi decoder," in *IEEE Military Communications Conference,* vol. 1, pp. 244 – 248. Sep. 1990.

[167] P. H. Kelly, and P. M. Chau, "A flexible constraint length, foldable Viterbi decoder," in *IEEE Global Telecommunications Conference*, vol. 1, pp. 631 – 635, Nov. 1993.

[168] K. Chadha, and J. R. Cavallaro, "A reconfigurable Viterbi decoder architecture," in *Asilomar Conference on Signals, Systems and Computers,* vol. 1, pp. 66 – 71, Nov. 2001.

[169] J. R. Cavallaro, and M. Vaya, "Viturbo: a reconfigurable architecture for Viterbi and turbo decoding," *in IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2, pp. 497-500, April 2003.

[170] P. H. Chen, H. Kai, H. Nai-Hsuan, and W. An-Yeu, "Dual-mode convolutional/SOVA based turbo code decoder VLSI design for wireless communication systems," in *IEEE International Systems-on-Chip Conference*, pp. 369 – 372, Sep. 2003.

[171] C. Berrou, R. Pyndiah, P. Adde, C. Douillard, and R. Le Bidan, "An overview of turbo codes and their applications," in *The European Conference on Wireless Technology*, pp. 1 – 9, Oct. 2005.

[172] C. Berrou, "The ten-year-old turbo codes are entering into service," *IEEE Communications Magazine*, vol. 41, pp. 110 – 116, Aug. 2003.