Reconfigurable Architectures for Beyond 3G Wireless Communication Systems

Cheng Zhan

A thesis submitted for the degree of Doctor of Philosophy.
The University of Edinburgh.
August 2007

Abstract

Market requirements always influence the semiconductor industry. The coexistence of multiple standards, which exhibit distinct mobility and data rates, makes that a flexible convergence of current wireless standards and services is expected from beyond 3G systems. However, this trend needs a strong demand for underlying hardware architectures to achieve unprecedented performance, flexibility, low power consumption and time-to-market requirements.

Since forward error correction algorithms demand the most computational cost of the whole physical layer system, this thesis employs two forward error correction cases, Viterbi decoder and double binary circular Turbo decoder, to investigate three potential reconfigurable hardware architectures for beyond 3G wireless communication system.

Firstly, a domain specific reconfigurable Viterbi decoder fabric is introduced, which can support multiple Viterbi decoders with different constraint lengths and code rates. In addition, it also provides near ASIC performance in terms of power consumption and area. In order to further reduce the design and verification cost of this domain specific reconfigurable design, Chapter 4 presents another reconfigurable architecture which can be automatically generated and programmed by its associated CAD framework. Composed of heterogeneous coarse-grained processing units and a 2-D interconnection mesh, this reconfigurable architecture demonstrates significant power and area savings as compared with commercial FPGAs. RICA, reconfigurable instruction cell array, which is a dynamic reconfigurable architecture programmed by ANSI C, has been developed as a feasible solution for future wireless and multimedia applications. In Chapter 5, several advanced optimization approaches are proposed to efficiently implement the Viterbi decoder on RICA architecture. Furthermore, Chapter 6 and Chapter 7 demonstrate the implementation of a more complex application, double binary circular Turbo decoder. In Chapter 6, a system model is build to investigate the suitable decoding algorithm which can balance the decoding throughput and performance degradation. On the other hand, appropriate quantization scheme for the decoding implementation is devised based on a bit-true model. Finally, an optimized double binary circular Turbo decoder which can provide scalable decoding throughput is demonstrated on the RICA architecture.

Declaration of originality

I hereby declare that the research recorded 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, except when otherwise stated.

Cheng Zhan

Acknowledgments

The work described in this thesis could not have been accomplished without the help and support of others. Foremost, I would like to thank my research advisor Professor Tughrul Arslan for his guidance and support during the past three years. The vision, infrastructure, and resources you provided have been truly extraordinary. I am especially grateful to you, along with Dr. Iain Lindsay, for provide helpful comments on my research works. I would like to thanks Dr. John S Thompson for helping me to better understand many aspects of digital communication scenario. I would like to thank Dr. Ahmet T. Erdogan for teaching me how to use CAD tools, being so patient and helping whenever I had a problem with hardware design.

I would especially like to thank the following group members: Sami Khawam, Mark Milward, Ioannis Nousias, Yi Ying, Mark Muir, Zahid Khan, Shantnu Tiwari, Nazish Aslam. Thanks Sami for his helpful discussions on the reconfigurable architecture and design methodology. Thanks also to Mark Milward, Ioannis Nousias, Yi Ying and Mark Muir for their CAD tools, much help in preparing my thesis. Thanks Zahid Khan, Shantnu Tiwari and Nazish Aslam for their suggestions and cooperation on Wimax and DVB-T/H physical layer implementation.

I would also like to thank the fellow members of the system level integration group, also known as SLI group: Robert Graham, Evangelos F. Stefatos, David Fung, Zhenyu Liu, Adeoye Olugbon, Sajid Baloch, Imran Ahmed, Han Wei and Jong Hun Han. Thanks for their inputs and discussions on the digital circuit and system level designs.

I special appreciate Institute of Integrated Micro and Nano Systems and Spiral Gateway Ltd. sponsoring my PhD study and giving me a chance to fulfill my life goal.

Finally, I would like to express my deepest appreciation for my Father and Mother. Your unconditional love and continual encouragement have been a source of strength without which I would have never gotten to where I am today. I dedicate this work to you with all my love.

Table of Contents

List of figures

List of Tables

Glossary / Acronyms

2G	Second Generation
3G	Third Generation
3GPP	3rd Generation Partnership Project
4G	Fourth Generation
ACSU	Add-Compare-Select Unit
ACS	Add-Compare-Select
ALU	Arithmetic Logic Unit
ARQ	Automatic Repeat reQuest
ASIC	Application Specific Integrated Circuit
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BMU	Branch Metric Unit
BPSK	Binary Phase Shift Keying
CAD	Computer Aided Design
CLB	Configurable Logic Block
DVB-RCS	Digital Video Broadcasting - Return Channel Satellite
DVB-T	Digital Video Broadcasting - Terrestrial
FEC	Forward Error Correction
FER	Frame Error Rate
FFT	Fast Fourier Transform
FPGA	Field Program Gate Array
GPP	General Purpose Processor
GCC	GNU C Compiler
GSM	Global System for Mobile communication
HDL	Hardware Description Language
HSDPA	High Speed Downlink Packet Access
IC	Instruction Cell
IP	Intellectual Property
LDPC	Low Density Parity-check Code
LIFO	Last In First Out

LUT	Look Up Table
MAC	Multiplier ACcumulator
MAP	Maximum A Posterior
PU	Process
QoS	Quality of Service
RAM	Random Access Memory
RC	Reconfigurable Cell
REA	Register Exchange Algorithm
RICA	Reconfigurable Instruction Cells Array
RISC	Reduced Instruction Set Computer
RTL	Register Transfer Level
RSC	Recursive Systematic Convolutional
SIMD	Single Instruction Multiple Data
SISO	Soft Input Soft Output
SMU	Survivor Management Unit
SNR	Signal-to-Noise Ratio
SoC	System on Chip
SOVA	Soft Output Viterbi Algorithm
TBA	Trace Back Algorithm
TTA	Transport Triggered Architecture
ULIW	Ultra Long Instruction Word
VLIW	Very Long Instruction Word
WCDMA	Wideband Code Division Multiple Access
WiMAX	Worldwide Interoperability for Microwave Access
WLAN	Wireless Local Area Network

Chapter 1:

ntroduction

1.1 Motivation

Since the emergence of the second-generation cellular system, wireless communication has become one of the most remarkable sectors in the consumer semiconductor industry. The number of mobile subscribers has grown over 100-fold in the last decade, and the market analysts confidently predict that the current number of global mobile subscriptions will be doubled by 2008 [1].

Because of the huge commercial opportunity in the wireless communication market, the semiconductor manufacturers and telecom operators have struggled for providing customers with the most distinctive products which are smaller, cheaper, and have more features than current ones. Consequently, this innovation demand has become the major momentum to push the semiconductor and telecommunication technologies forward.

The basic trends for wireless communication are ubiquity and higher throughput [2]. The ubiquity of wireless communications indicates that users can use their mobile devices in different environments, like the home, the office, driving, and walking. With extremely high throughput, a great number of services, such as audio, video, internet access, etc, can be provided on a single portable device.

Stimulated by the market requirement, at present, various wireless communication standards have been deployed and commercialized focusing on different levels of mobility and data rate, which is shown in Figure 1.1. The coexistence of multiple standards makes the transition from third-generation (3G) wireless systems to fourth-generation (4G) systems is not likely to be as clear as the transition from second-generation (2G) systems to 3G.

With regard to the prediction the 4G wireless network, a flexible convergence of current wireless standards and services is expected [3]. This kind of convergence provides the opportunity to simultaneously use standards such as GSM, 3G and WLAN from one device. Thus, it not only fulfills customers' desire for faster and ubiquitous wireless connection, but also maintains the commercial infrastructural investment of the previous wireless generations.

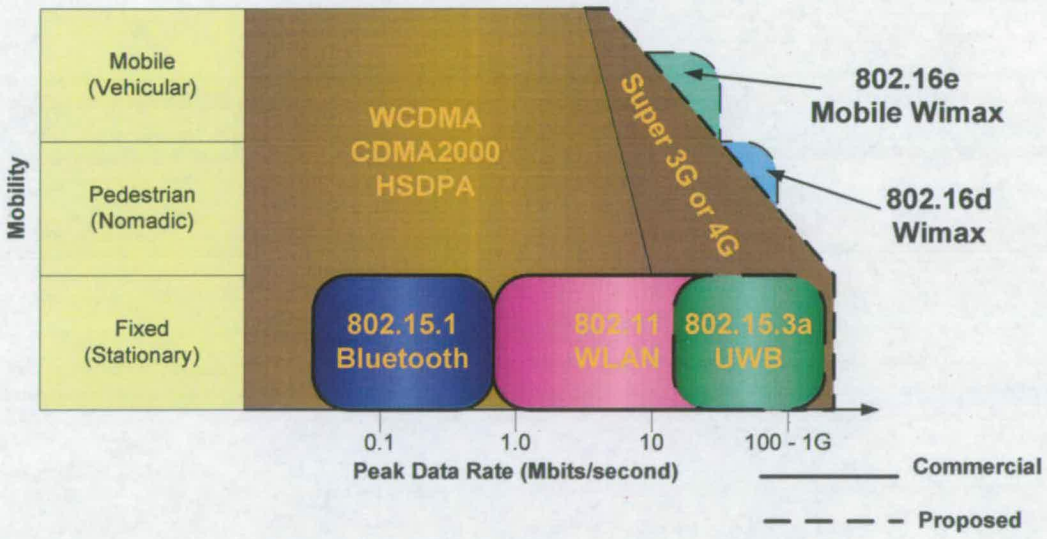


Figure 1.1 4G wireless communication standards [4]

However, the convergence concept of 4G relies heavily on the semiconductor industry to invent a novel and flexible hardware architecture that can be cost-efficient and provide acceptable performance [5]. The 4G hardware architecture will require unprecedented digital, analog, and software integration. The main obstacles of the hardware design for 4G exhibit

in four aspects:

- **Performance**: Mobile terminals require very accurate timing on reactive application and protocol processing, thus this kind of real-time constraint demands that the physical layer architecture must meet the data throughput and quality of service requested by different standards.

- **Flexibility**: The evolution of wireless standards is extremely fast. For instance, IEEE standardized eight different versions of wireless LAN applications from 1999 to 2004. The mobile terminal has to be flexible enough to accommodate changes in the standards or the introduction of new services.

- **Low power consumption**: Stringent limit on power consumption is an inherent challenge for mobile terminals. Minimizing the power consumption and extending the battery lifetime of mobile terminals are a key ultimate aim of every wireless network generation.

- **Time-to-market**: Because of the evolution of wireless communications and semiconductor technologies, the lifespan of mobile terminals has been shrunk and the manufacturers are forced to release new products at a short time to keep their competitive position in the market. This trend results in very tight time-to-market constraints for the hardware design.

However, the current architectures and design methodologies can hardly satisfy all these four requirements. Application specific integrated circuits (ASIC) can provide the best implementation for specific applications, typically several order of magnitude better than processors. However, ASICs have high non-recurring engineering (NRE) costs, expensive design tools, significant manufacturing risk and long periods to bring products to market. These limitations obstruct ASICs exhibiting themselves in the mainstream platform for future portable and network devices.

Digital signal processors (DSPs) are the dominant products in the 2G market. DSPs are not only strong in control flow applications and frequent branching, but are also optimized to support data flow oriented tasks. Unfortunately, with regard to the highly computational intensive tasks in 4G, even the highest-performance DSPs today cannot deliver the

demanded horsepower.

Field Programmable Gate Arrays (FPGAs) can offer the ASIC-like performance and DSP-like flexibility due to their fine-grained bit-level configuration architecture. However, FPGAs pay very high silicon area and power consumption penalties for their attractive performance and flexibility. Their area and power consumption make them unfeasible for the portable devices.

The limitations of the current architectures motivate us to investigate a novel reconfigurable architecture to bridge the performance gap between theory and practice, and also fulfill the flexibility, low power consumption and time-to-market requirements of the beyond 3G portable devices.

1.2 Objective

The objective of this dissertation is to make a contribution to the scenario of hardware architectures for beyond 3G portable devices. This dissertation investigates several reconfigurable architectures and design methodologies to study the underlying fabrics for beyond 3G portable devices. By employing the Viterbi decoder and double binary circular Turbo decoder as the study cases, this dissertation demonstrates the efficient solutions for the future portable devices.

1.3 Major contributions

The major contributions of the dissertation are split into five key aspects:

- A novel domain specific reconfigurable architecture for the Viterbi decoder is demonstrated. It not only supports the decoding process for constraint length from 3 to 9 and code rate 1/2 and 1/3 convolutional codes, but also offers a maximum 150Mbps decoding throughput with little area and power overhead.

- An automatic design methodology for a domain specific reconfigurable architecture is

proposed. This architecture is based on a synthesizable heterogeneous coarse-grained array and 2-D mixed interconnection mesh. In addition, the proposed software design flow can automatically generate the expected reconfigurable architecture and map applications on the proposed architecture.

- An efficient Viterbi decoder implementation on a reconfigurable instruction cell array (RICA) platform is demonstrated. Furthermore, several advanced optimization approaches for the Viterbi decoder on the RICA platform are proposed to boost its performance.

- A system model of M-binary circular Turbo codes and its practical application, double binary circular Turbo codes is established. The implementation design space on the algorithm level has been explored. The selection of suitable decoding algorithms with their simplification and optimization are investigated.

- Suitable parallel decoding schemes and fixed-point representation are investigated in order to obtain a high throughput double binary circular Turbo decoder implementation on the RICA. Moreover, the advanced optimization approaches for double binary circular Turbo decoder implementation are described.

1.4 Structure of this dissertation

The structure of this dissertation is presented as follows:

Chapter 2 contains a description of the literature reviews related to this work including basic concepts of wireless communication, channel coding, the representations of convolutional codes and the mathematical background of the Viterbi decoding algorithm. In addition, the implementation approaches for a Viterbi decoder are described. The details of current programmable and reconfigurable Viterbi architectures are also presented and compared.

A domain specific reconfigurable Viterbi decoder implementation is addressed in Chapter 3. The reconfigurable design approaches for the Viterbi decoder are presented. This chapter also proposes several power-saving schemes to reduce the power consumption of the whole fabric.

In Chapter 4, a design methodology which can automatically generate a heterogeneous coarse-grained reconfigurable architecture is proposed. By means of CAD tools, the design and verification time of a domain specific reconfigurable architecture can be dramatically reduced. Moreover, as compared with commercial FPGAs, the proposed architecture shows salient advantages in terms of power consumption and area.

A reconfigurable architecture, RICA, which can be programmed by high-level language, such as C and C++, is introduced in Chapter 5. The efficient implementation and optimization approaches for the Viterbi decoder are demonstrated on the RICA platform.

Chapter 6 and Chapter 7 address an efficient double binary circular Turbo decoder design from both algorithm level and implementation level. Chapter 6 focuses on investigating suitable decoder algorithms. Chapter 7 investigates the suitable implementations approaches, such as the parallel decoding scheme and fixed-point representation.

Finally, the thesis is concluded with the summary in Chapter 8.

Chapter 2:

Background

As one of the most successful sectors of the consumer electronics industry, wireless communication has become a major driving force behind the evolution of semiconductor technology. Beyond 3G wireless communication systems will not only offer faster and ubiquitous services for customers, but also need to keep the economical success of the previous system. In addition, the stringent requirements upon low power consumption, non-recurring engineering cost and time-to-market have to be fulfilled. These demands induce various challenging problems to the designers of next generation wireless communication systems.

This chapter introduces the background literatures relevant to the work in this thesis. This chapter commences with the introduction of basic concepts of wireless communication. In Section 2.2 and Section 2.3, channel coding and the representations of convolutional codes are elaborated. Section 2.4 gives an overview of the mathematical background of the Viterbi decoding algorithm. The implementation approaches for different blocks of a Viterbi decoder

is described in Section 2.5. The details of current programmable/reconfigurable Viterbi architectures are presented and compared in Section 2.6. The conclusion can be accessed in Section 2.7.

2.1 Digital Communication System

A typical digital communication system consists of a transmitter, receiver and the transmission media (channel). The transmitter takes digitized data from an information source, performs several processing steps, and transmits the signals through a communication channel. Because of the reflection, refraction and dispersion, the transmission waveform is disturbed by the noise in the wireless communication channel. To ensure a reliable quality of service (QoS), the receiver must appropriately process the received signal to recover the transmitted data. Figure 2.1 illustrates the basic blocks of a typical digital communication system [6].

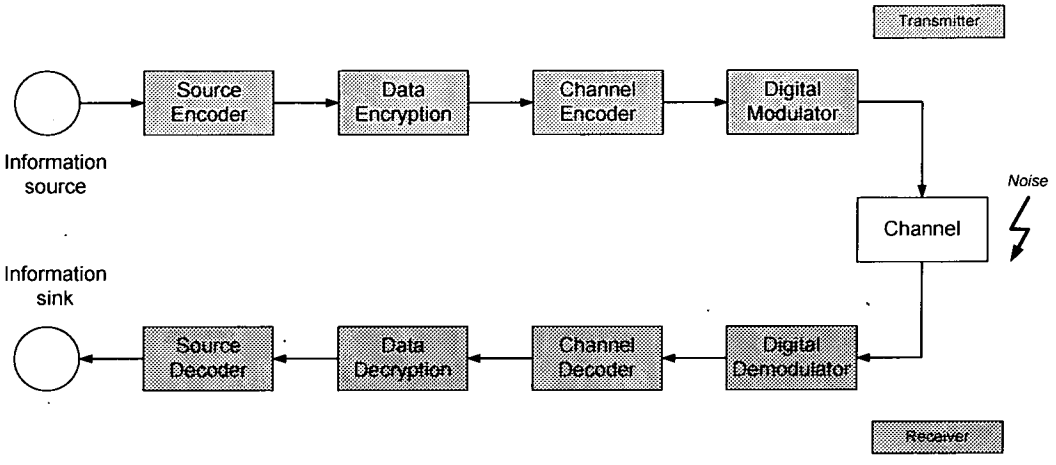


Figure 2.1 Digital communication system

In Figure 2.1, the information source is initially converted to binary digitals. During source encoding, these bits are grouped to form digital messages or message symbols. Then, data encryption can be used to ensure privacy. In a further step, error correction techniques are applied to protect the data against channel noise during transmission. This can be done either by forward error correction (FEC) or automatic repeat request (ARQ). Finally, these binary bits are modulated to analog signals which are suitable for the transmission channel. On the receiver side, corresponding data transformations have to be carried out in reverse order to recover the original information.

Due to different purposes and transmission environments, the techniques of each block may be varied for different standards. The techniques of channel coding and modulation in terms of different standards have been tabulated in Table 2.1.

Table 2.1 Different wireless communication standards

	Modulation	**FEC**	**System Peak Throughput**
GSM [7]	TDMA GSMK	Convolution codes K=5 R =1/2	270 Kbps
WCDMA [8]	CDMA BPSK/QPSK	Convolution codes K=9 R =1/2,1/3 Turbo codes K=4 R=1/3	2 Mbps
HSDPA [9]	CDMA QPSK/QAM16	Convolution codes K=9 R =1/2,1/3 Turbo codes K=4 R=1/3	14.4 Mbps
WLAN [10]	OFDM FFT size: 64 BPSK/QPSK/QAM16/QAM64	Convolution codes K=7 R =1/2	54 Mbps
WiMAX [11]	OFDM FFT size: 128/256/512/1024/2048 QPSK/QAM16/QAM64	Convolution codes K=7 R =1/2 Double binary Circular Turbo codes: K=4 R=1/2	75 Mbps
Mobile WiMAX [12]	OFDM FFT size: 512/1024 QPSK/QAM16/QAM64	Convolution codes K=7 R =1/2 Double binary Circular Turbo codes: K=4 R=1/2	31.68 Mbps
DVB-T [13]	OFDM FFT size: 2048/4096 QPSK/QAM16/QAM64	Convolution codes K=7 R =1/2 Double binary Circular Turbo codes: K=4 R=1/2	17.9 Mbps

2.2 Channel coding

Channel coding aims at protecting data against transmission errors, induced by noise and

path fading transmission channels. There are two basic channel coding approaches which are normally employed.

Forward Error Correction (FEC) [14], adds redundant information to the data sequence on the transmitter side. That redundancy is exploited on the receiver side to detect and correct errors. The rate of the remaining errors is a function of the channel characteristics, but the sequence is only transmitted once, thus the throughput stays constant.

It can be seen from Table 2.1 that the convolution encoder and decoder are the common FEC blocks for all the current wireless communication standards. According to the computational cost estimation and analysis in [15], the convolutional decoder demands the most computational complexity among physical layer blocks. Thus, this thesis selected the convolutional decoder as an application example to illustrate the reconfigurable architectures for beyond 3G wireless communication systems.

2.3 Convolutional Encoder

Convolutional codes were developed as an alternative to block codes due to their better error correction capability [14]. The error correction capabilities of convolutional codes result from the current coded symbols depending on past data values. Unlike the block encoder, where the transmitted message is coded in code words with a fixed length, a convolutional encoder operates serially on a bit-stream of arbitrary word length. A typical convolutional encoder is exhibited in Figure 2.2, where it can be seen that a convolutional encoder consists of shift registers and modulo-2 adders. At each time slot, one transmitted bit enters the first stage of shift registers. The coded symbols are generated by a modulo-2 adder which uses modulo-2 arithmetic to add the current bit with previous bits which are stored in the registers.

For each single input bit, there is a corresponding single output symbol, consisting of two output bits. Thus, the volume of the transmitted message is doubled after a convolutional encoder. These redundant bits will help the convolutional decoder detect and correct errors which occur during the transmission through the wireless communication channel.

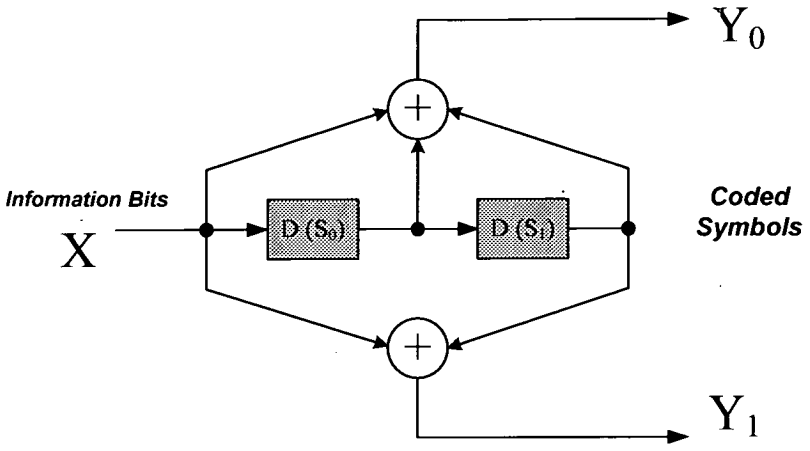


Figure 2.2 Constraint length 3, code rate 1/2 convolutional encoder

Convolutional codes have their own standard notation. There are three parameters to describe the types of convolutional codes, *constraint length K*, *code rate R* and *generator polynomials* [14].

Constraint length K represents the number of taps of the shift register in the convolutional encoder. For example, in Figure 2.2, *K* is equal to 3, since there are three taps in the encoder, one for current bit and two for previous bits.

A convolutional encoder with a code rate of $R = m/n$, means this encoder can transmit n output bits for every m input bits. Figure 2.2 illustrates a code rate $R=1/2$ convolutional encoder. Although any code rate is possible, $1/n$ systems are most widely used for the sake of decoding efficiency.

Generator polynomials are the mathematical description of the connections between the shift register taps and the module-2 adders. The generator polynomials for upper and lower connections of the encoder presented in Figure 2.2 can be described as follows: ($_8$ means octadic representation)

$$\begin{aligned} G(Y_0) &= \{1\ 1\ 1\} = 7_8 \\ G(Y_1) &= \{1\ 0\ 1\} = 5_8 \end{aligned} \tag{2.1}$$

The selection of the generator polynomials also can affect the performance of the convolutional coding system. The performance of the convolutional codes is defined as its

robustness to channel noises. The characteristic of convolutional codes which describes this robustness is called the free distance and is defined as the minimum Hamming distance between any two code words in the codes [14].

For different values of K and R, there are optimal values for generator polynomials which can provide the best performance. These have been experimentally determined in [6] and Table 2.2 shows the optimal values for code rate 1/2 and 1/3. This thesis only considered the 1/2 and 1/3 code rates, however the proposed reconfigurable fabrics can be easily extended to support other code rates, such as 1/4 and 1/5.

Table 2.2 Generator polynomials

Rate	K	Generator Polynomials
1/2	3	$\{7_8, 5_8\}$
	5	$\{33_8, 31_8\}$
	7	$\{171_8, 133_8\}$
	9	$\{753_8, 561_8\}$
1/3	3	$\{7_8, 7_8, 5_8\}$
	5	$\{37_8, 33_8, 25_8\}$
	7	$\{175_8, 145_8, 133_8\}$
	9	$\{711_8, 663_8, 557_8\}$

2.3.1 Finite state machine and the trellis representation

If the contents of the shift registers can be treated as the states of the convolutional encoder, the encoder also can be fully described as a finite state machine, which presents the state transitions and input/output symbols of a convolutional encoder. The state transition diagram of constraint length $K=3$ and code rate $R=1/2$ convolutional encoder with generator polynomials $\{7_8, 5_8\}$ is presented in Figure 2.3.

In Figure 2.3, the transition paths are drawn by solid lines and dashed lines, which denote the paths associated with the input bit 0 and 1, respectively. It can be seen that there are only two transitions emanating from each state, corresponding to the two possible input bits, 0 or 1. For example, if the present encoder state is 01, the only possibilities for the state at the next shift are 00 or 10.

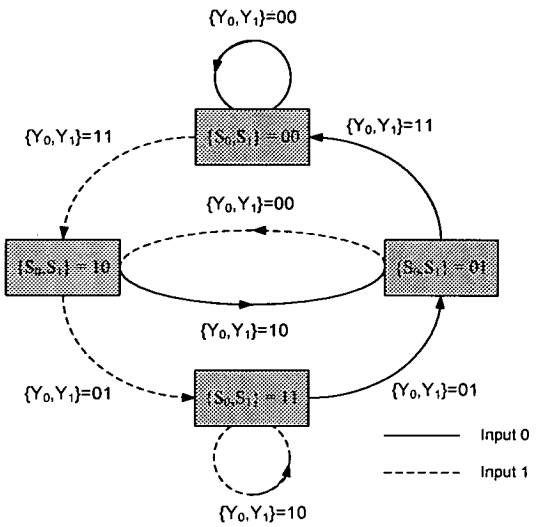


Figure 2.3 State transition diagram for convolutional encoder

From the state transition diagram, an important representation of convolutional codes, trellis diagram, can be obtained. Figure 2.4 shows the trellis diagram which is constructed from the state transition diagram, but also explicitly presents the passage of time. The trellis diagram depicts the interconnection of a set of stages which is indicated by a different time. The values of registers in the encoder are initialized to zero, and the trellis diagram starts from the 00 state. The numbers on the path called path states represent actual transmitted symbols which are the output of the encoder (Y_1 and Y_2). Similarly with the finite state machine representation, a solid line corresponds to a 0 input and a dashed line corresponds to a 1 input.

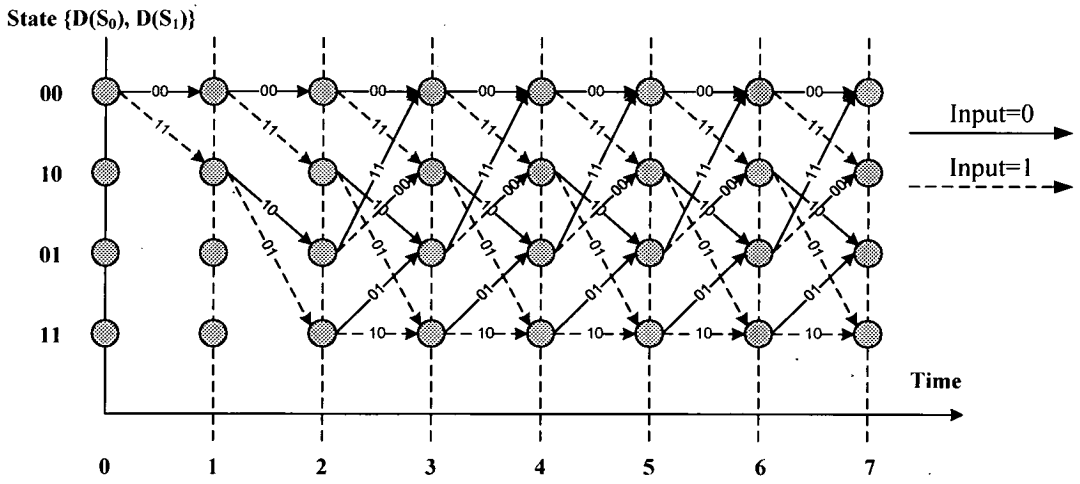


Figure 2.4 Trellis diagram

2.4 Viterbi decoding algorithm

At the receiver side, the channel decoder takes the received sequence D and regenerates an estimation of the transmitted codeword sequence E. The Viterbi decoding algorithm [21] is a maximum likelihood decoding algorithm based on the concept of the trellis diagram. Maximum likelihood decoder produces an output sequence with the maximum likelihood function, which is represented by:

$$P_{\max}(D|E) = MAX\left(\prod_{k=0}^{N-1} P(D_k|E_k)\right) \tag{2.2}$$

where D_k and E_k represent a symbol of the decoder received sequence and the transmitted codeword sequence, respectively, and N denotes the size of the sequence. Since logarithms are monotonically increasing, the likelihood function also can be estimated in a logarithmic domain to facilitate the implementation.

By means of the Viterbi algorithm, the logarithmic likelihood functions $\ln P(D_i|E_i)$ at time $t=i$ are accumulated following the trellis path and half of the current trellis path will be discarded based on the comparison of the likelihood function.

As shown in Figure 2.4, there are only two possible previous states for each trellis state. The accumulated logarithmic likelihood function following each possible path can be denoted as:

$$\begin{aligned} \ln P_1(D_{k<i+1}|E_{k<i+1}) &= \ln P_1(D_{k<i}|E_{k<i}) + \ln P(D_i|E_i^1) \\ \ln P_2(D_{k<i+1}|E_{k<i+1}) &= \ln P_2(D_{k<i}|E_{k<i}) + \ln P(D_i|E_i^2) \end{aligned} \tag{2.3}$$

The first term on the right side of Equation 2.3 indicates the accumulated logarithmic likelihood on each trellis node and the second term denotes the logarithmic likelihood for each trellis path, which are also named path metrics and branch metrics, respectively. It can be seen from Figure 2.5 that the path with smaller accumulated logarithmic likelihood is discarded and the other path is stored as the new state metric. Simultaneously, an indication of the survival path *(survivor bit)* has to be saved in the memory which is used to reconstruct transmission symbols. This iterative computation will follow the trellis till the end.

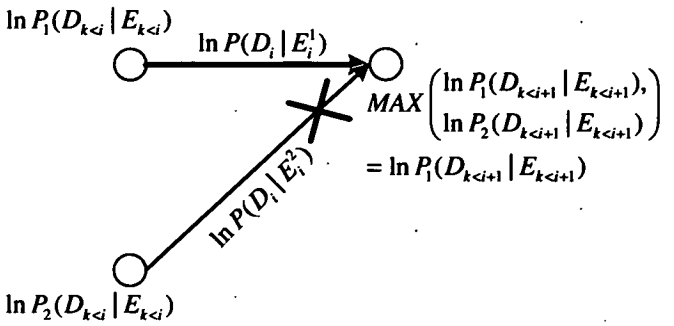


Figure 2.5 Path update following the trellis diagram

At the end of the trellis, the state with the maximum accumulated likelihood function can be obtained. Since each state has only one entering survivor path, tracing the trellis backwards can yield a unique path. This path is the maximum likelihood path and the bits retrieved during the trace back are the decoded output bits [21].

2.5 Viterbi decoder architecture

This section describes the hardware architecture for the Viterbi decoder in detail. The three major components in a Viterbi decoder, branch metric unit (BMU), add-compare-select unit (ACSU) and survivor management unit (SMU) are shown in Figure 2.6.

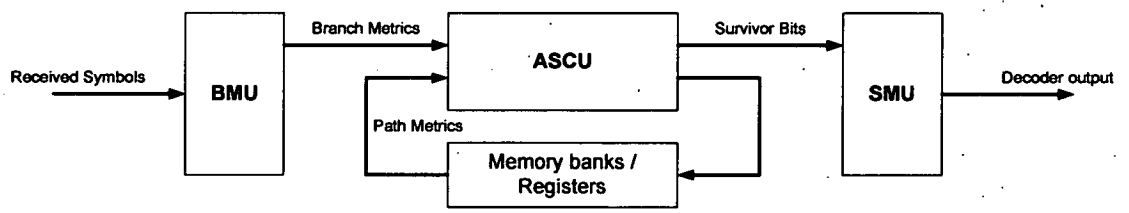


Figure 2.6 Diagram of Viterbi decoder architecture

2.5.1 Branch Metric Unit (BMU)

The BMU module takes charge of calculating branch metrics corresponding to the received sequence. According to the representation of the received sequence, there are two types of Viterbi decoder: hard decision Viterbi decoder and soft decision Viterbi decoder.

2.5.1.1 Hard decision

In the case of hard decision, each bit in an input symbol is represented by a single estimated bit. The calculation of maximum likelihood function $\ln P(D_i \mid E_i)$ is equal to look for the minimum Hamming distance between sequence D and E [14].

2.5.1.2 Soft decision

When each bit in a symbol in the received sequence is represented by multiple bits, then it is referred to as soft decision and the Euclidean distance is calculated.

In practical implementation, the Euclidean distance can be further simplified. If E_i is the expected codeword, D_i are the received soft decision symbol and R is the code rate, the Euclidean distance between E_i and D_i is defined as:

$$Euclidean = \sum_{n=0}^{1/R-1} \left(E_i^n - D_i^n\right)^2 \tag{2.4}$$

Expanding Equation 2.4, we can get:

$$Euclidean = \sum_{n=0}^{1/R-1} \left(\left(E_i^n\right)^2 - 2E_i^n D_i^n + \left(D_i^n\right)^2\right) \tag{2.5}$$

Since the comparison must be done, only the portions of Equation 2.5 which are different for each path need to be considered. Thus, the common terms $\left(E_i^n\right)^2$ and $\left(D_i^n\right)^2$ can be eliminated. Equation 2.5 can be further reduced to:

$$Euclidean = -2\sum_{n=0}^{1/R-1} \left(E_i^n D_i^n\right) \tag{2.6}$$

Moreover, the leading parameter *-2* can be removed, thus the minimum value of the Euclidean distance occurs when $\sum_{n=0}^{1/R-1} \left(E_i^n D_i^n\right)$ is a maximum. Since E_i is either 1 or -1, $\sum_{n=0}^{1/R-1} \left(E_i^n D_i^n\right)$ can be referred to as addition and subtraction operations.

According to previous works [22], soft decision data can provide 2.2 dB performance gain compared with hard decision data. In this thesis, without specific statement, a soft decision Viterbi decoder is considered.

2.5.2 Add-Compare-Select Unit (ACSU)

ACSU is a recursive computation unit which, taking in the branch metrics, calculates the path metrics and the survivor bits following the trellis paths. In every trellis stage, the updated path metrics are stored back to local registers which are served as inputs of ACSU in the next recursion whilst survivor bits are written to survivor memory.

Generally, for a constraint length K Viterbi decoder, 2^{K-1} path metrics have to be updated at every trellis stage, thus 2^{K-1} ACS operations are required to perform every recursion. With the increase of K, the number of ACS operations is exponentially increased. Therefore, ACSU becomes the most computational intensive unit of a Viterbi decoder and the decoding throughput is mainly determined by the execution time of ACSU. In the last decade, a lot of research contributions focus on how to implement ACSU more efficiently and effectively.

2.5.2.1 Butterfly Unit

According to the inherent symmetric property of the generator polynomials of convolutional codes, two adjacent ACS operations share the same inputs, path metrics and branch metrics. Therefore, in practical applications, two ACS units are bonded together [23], named butterfly units, shown in Figure 2.7. By means of this type of combination, the route delay inside an ACSU can be reduced and it can further also increase the modularity of the design.

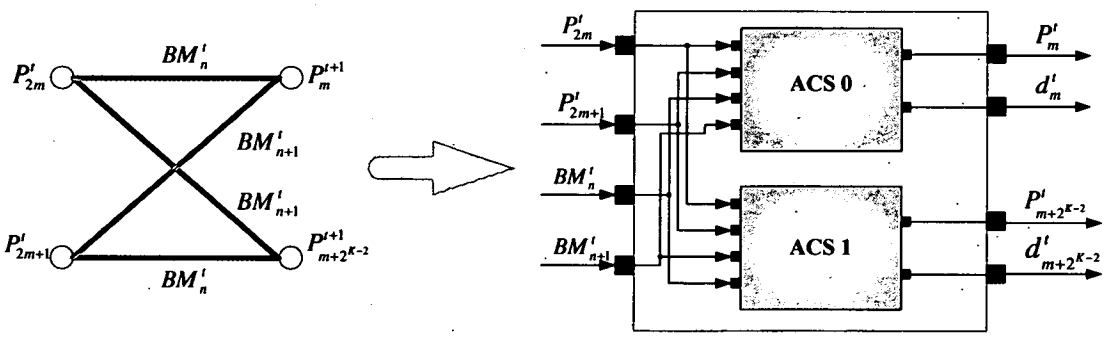


Figure 2.7 Butterfly Unit

2.5.2.2 Normalization

During the recursive ACS computation, path metrics will accumulate without bounds. Implementation without normalization requires large word length for path metrics to avoid overflow. On the other hand, increasing word length for path metrics leads to increasing power consumption and area of the whole architecture. In order to limit the word length of path metrics, normalization block has to be employed to prevent arithmetic overflow. There are two widely used approaches for Viterbi decoders, rescaling and modulo arithmetic.

2.5.2.2.1 Rescaling approach

Rescaling approach [24] is achieved by subtracting the minimum path metric from all other path metrics at the end of every ACS recursion, which is illustrated in Figure 2.8. This approach results in the minimum word length for path metrics. However, $2^{K-1}-1$ comparators working in a cascaded manner have to be integrated at the end of ACSU to look for the minimum path metric for each trellis stage. In addition, it can be seen from Figure 2.8 that all path metrics have to be subtracted by the minimum path metric to rescale them and avoid the overflow. The computational cost incurred by this approach is relatively huge and rescaling approach is infeasible for a high throughput Viterbi decoder design.

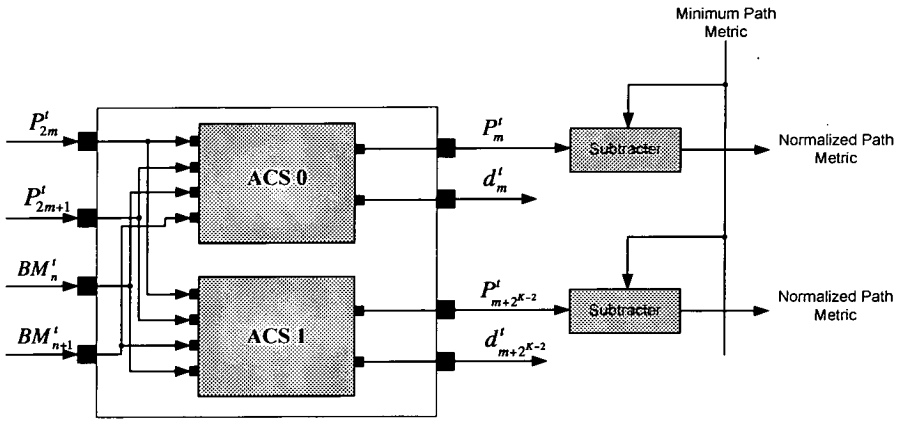


Figure 2.8 Rescaling approach for normalization

2.5.2.2.2 Modulo Arithmetic Approach

Modulo arithmetic was devised in [24], which exploits the fact that the difference of path metrics is always bounded by a constant value [25]. Modulo arithmetic approach replaces

the normal comparator by a modulo comparator and accommodates overflow in such a way that is does not affect the correctness of the results. Since the path metrics are implicitly normalized by modulo comparators, this approach can avoid the long comparison chain in the rescaling approach and the associated hardware overhead is extremely small, yielding a high throughput Viterbi decoder implementation.

The penalty for modulo arithmetic is requiring extra bits to represent path metrics as compared with the rescaling approach. According to [26], the number of bits required to represent path metrics can be deduced by:

$$w \geq [\log_2(2 \cdot K \cdot B + 1)] + 1 \tag{2.7}$$

where K denotes the constraint length and B is the upper bound for the absolute values of branch metrics. The minimum word length requirements of modulo arithmetic normalization scheme for constraint length 3, 5, 7 and 9, code rate 1/2 and 1/3 Viterbi decoder are tabulated in Table 2.3.

Table 2.3 Minimum word length requirement for path metrics

K	**3**		**5**		**7**		**9**	
R	**1/2**	**1/3**	**1/2**	**1/3**	**1/2**	**1/3**	**1/2**	**1/3**
Word length	7	7	7	8	8	8	8	8

From decoding throughput and computational cost point of view, modulo arithmetic is superior to rescaling approach and has been widely adopted in the high performance Viterbi decoder [26] [27] [28] [29].

2.5.2.3 Parallel and partial parallel ACSU

2.5.2.3.1 Parallel architecture

For a constraint length K Viterbi decoder, 2^{K-1} path metrics have to be updated at every trellis stage. If a dedicated ACS unit is used for every path metric calculation, this leads to a parallel Viterbi decoder architecture. Since 2^{K-1} path metrics and a complete vector of survivor bits are calculated every clock cycle, a high throughput Viterbi decoder is normally achieved by this architecture [27] [28] [29]. However, the parallel ACSU architecture has to

pay area penalty as the price for high throughput.

2.5.2.3.2 Partial parallel architecture

Partial parallel architecture is also named as in-place computation scheme, which was first introduced by [30]. Partial parallel architecture partitions the path metrics into several groups and makes each group share one ACS unit. However, additional memory banks and address controllers have to add to each group to store and schedule the temporary path metrics. For low and moderate throughput applications, partial parallel architecture can achieve high performance and low hardware cost [31] [32] [33]. However, the speed is limited by the folded ACSU architecture and obstructs its application for the high throughput Viterbi decoder.

2.5.3 Survivor Management Unit (SMU)

The SMU has the responsibility to reconstruct the information sequence from the survivor bits generated by the ACSU. In practice, there are two different algorithms to implement the survivor management unit, register exchange algorithm (REA) and traceback algorithm (TBA).

2.5.3.1 REA

In REA, a register must be assigned to each trellis state [34]. The registers record the survivor bits produced by ACSU along the trellis path from the initial stage to the final stage. At the end, the decoded output sequence is the one stored in the register which is assigned to the state with maximum likelihood function. Since REA does not require tracing trellis back, it is faster than TBA. However, REA incurs huge memory accessing bandwidth, and it is not suitable for a low power Viterbi decoder, especially for a large constraint length [35].

2.5.3.2 TBA

Unlike REA, TBA starts from an arbitrary state in final trellis stage to rebuild the most possible transmission path in a reversed way [36]. In TBA, the previous state S_{n-1} is decided

by current state S_n and associated survivor bits d_n, which is located in the bit line of survivor memory indexed by S_n. The previous state S_{n-1} is deduced by the following equation.

$$S_{n-1} = \{S_n << 1, d_n\} \tag{2.8}$$

Since traceback depends on ACSU finishing first, a huge decoding latency is introduced by TBA. In order to speed up the throughput, sliding window scheme is proposed in [37] to break the data dependence between ACSU and SMU.

2.5.3.3 Sliding window

Sliding window scheme is based on the rule that no matter from which state the TBA begins, after a sufficiently large number of recursive steps, all paths are merged at the same state [22]. Thus, traceback can start from the middle of a decoding sequence, rather than waiting till the end, thus the decoding latency can be shrunk.

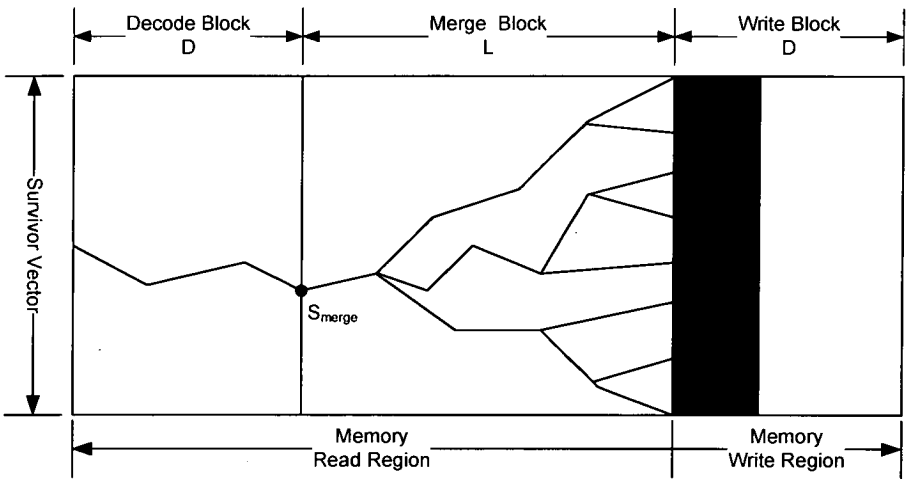


Figure 2.9 TBA with sliding window

TBA with sliding window scheme is illustrated in Figure 2.9. It can be seen that the whole procedure consists of three major operations: *write block, merge block* and *decode block* which are listed as follows:

Write block: In write block, the survivor bits generated by the ACSU are written to the survivor memory. Write block is the only memory writing operation and its address pointer

moves forward.

Merge block: In merge block, trellis is traced back based on Equation 2.8. The trace back operation starts from an arbitrary state. After L recursive processes, the trellis path reaches the merge state S_{merge}. This block reads the survivor vectors from the survivor memory to find the previous trellis state, but no outputs are generated in merge block. In order to ensure that the trellis path has arrived at S_{merge}, L is at least as long as five times the constraint length [22].

Decode block: The operation in this block is identical to the merge block except that the trace back starts from S_{merge} and the survivor bits produced every clock cycle will output as decoding sequence. Since the decode block produces the output in a reversed order, last-in-first-out (LIFO) buffers are exploited at the end to change the order of output sequence.

In the TBA approach, the decoder processes on a continuous sequence of incoming encoded data, splitting it into separate windows for operating. However, practically, the transmitted data is split into frames and regard each frame as an independent block. The disadvantage of trace back operation from an arbitrary state is that the bit error rate performance is degraded because the decoder does not know from which state to start the final window operation and has no extra data to be used as the training data to find the correct state for last window. There are several termination approaches which are proposed to solve this issue, such as zero force termination [38] and tail-biting termination [39]. However, this part of discussion is out of the scope of this thesis. Since the major design methodologies of different termination approaches are very similar, we select the trace back operation from an arbitrary state as an example to demonstrate the reconfigurable fabrics in the following chapters.

2.5.3.4 Memory arrangement

As can be seen from Figure 2.9, several memory banks are employed in the sliding window scheme. The size of memory banks and its read/write clock frequency have to be fulfilled by:

$$\frac{L}{\frac{f_R \bullet N_T}{f_W} - 1} = D \tag{2.9}$$

where f_R and f_W are read and write frequencies for the memory read and write regions. N_T is the number of read pointers of the read region. In [38], a SMU with $L = D$ and $f_R = f_W$ is presented, which is shown in Figure 2.10. It can be seen that four independent single port memory banks with size of D, are exploited. The operations for different memory banks are executed in parallel. Once a bank has been decoded, it will become the write region for the next phase.

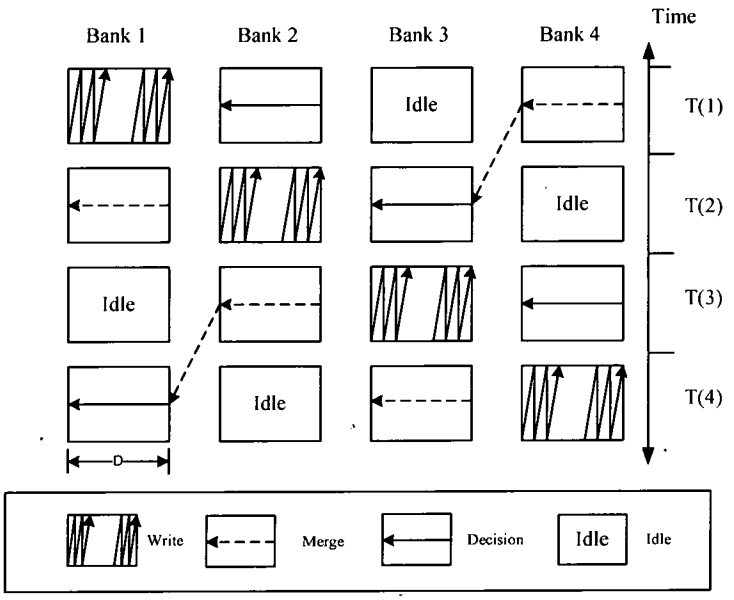


Figure 2.10 Operation of sliding window scheme with four memory banks

By increasing the number of the read pointers (N_T) [26] or the speed of the read pointer (f_R) [27], the total memory size could be reduced. The main point is that there is a tradeoff between the total memory size, the latency and the design complexity. The study of this tradeoff is out of the scope of this work, and this part of work can refer to [40]. We chose SMU architecture presented in [38] to illustrate our reconfigurable design methodology. However, this methodology proposed in this thesis also can be used by other sliding window cases.

2.6 Programmable/Reconfigurable Viterbi decoder implementation

ASIC implementation of a Viterbi decoder has been widely studied by previous works. ASICs are designed to exploit the parallelism of algorithms, also along with optimization in terms of power, speed and area. However, this inflexible implementation increases the cost of effort/time and shortens the lifetime of the products. In order to tailor to the dramatically changed wireless communication scenario, there are several different architectures which intend to balance the flexibility and hardware efficiency.

2.6.1 VLIW DSP

In the current market, the digital signal processor (DSP) is the dominating device for wireless communication products. In the contrast to general processor, the architecture of a DSP processor is usually optimized for a set of well-known applications by means of incorporating specific hardware components to provide high-performance arithmetical operation, such as the single-cycle multiply-accumulate (MAC), and high-efficient memory accessing, such as dedicated memory address generator.

By integrating more than one ALU, a very long instruction word (VLIW) DSP core can execute several instructions in parallel. A typical example of a VLIW DSP targeted for wireless communication applications is the Starcore SC140 from Fresscale Inc. [49]. The data arithmetic logic section of SC140 consists of four identical 32-bit ALUs and the address generation section contains two address arithmetic units. Thus an SC140 can execute up to six instructions at a time, four for arithmetic logic section and two for address arithmetic section.

2.6.1.1 Viterbi decoder on VLIW DSP

To accelerate the Viterbi decoding process, SC140 has designed specific instructions to provide efficient arithmetical operations, such as *ADD2*, *SUB2* and *MAX2VIT*, and suitable memory accessing instructions, such as *VSL.4F* and *VSL.4W* [50]. With four ALUs, eight butterflies can be executed in 10 clock cycles. Since the throughput of GSM is in the range

of several hundreds of *Kbps*, the extended instruction set VLIW DSP can perform very well in 2G infrastructures in terms of flexibility and performance.

However, its instruction fetching and dispatching manner becomes a major obstacle to a high throughput Viterbi decoder. When 3G emerged, purely software implementation on DSP can hardly fulfill up to 2Mbps system throughput requirement. Both TI [51] and Freescale launched their latest DSP families with a Viterbi co-processor to provide a high speed Viterbi decoding process. From their public reports [49] [51], 4Mbps and 4.88Mbps decoding throughput could be achieved by TI and Freescale DSPs, respectively.

2.7 Reconfigurable Viterbi decoder implementation

Due to the progress of silicon technology which allows more and more components to be implemented on a single chip, the era of reconfigurable hardware flourishes. Reconfigurable hardware combines the post-fabrication programmability of processors with a sequential computation style and the parallel computation style of application specific integrate circuits (ASIC).

In the case of the Viterbi decoder which is employed by lots of wireless communication system to overcome the variable deterioration in the reliability of wireless transmission channels, every system provides a unique service and requires different coding performance (constraint length and code rate) at different data throughput. Therefore, for beyond 3G system, it is extremely important to develop a reconfigurable Viterbi decoder which can be configured at run-time to operate over a range of standards (different constraint lengths and code rates) and also provide the targeted data throughput.

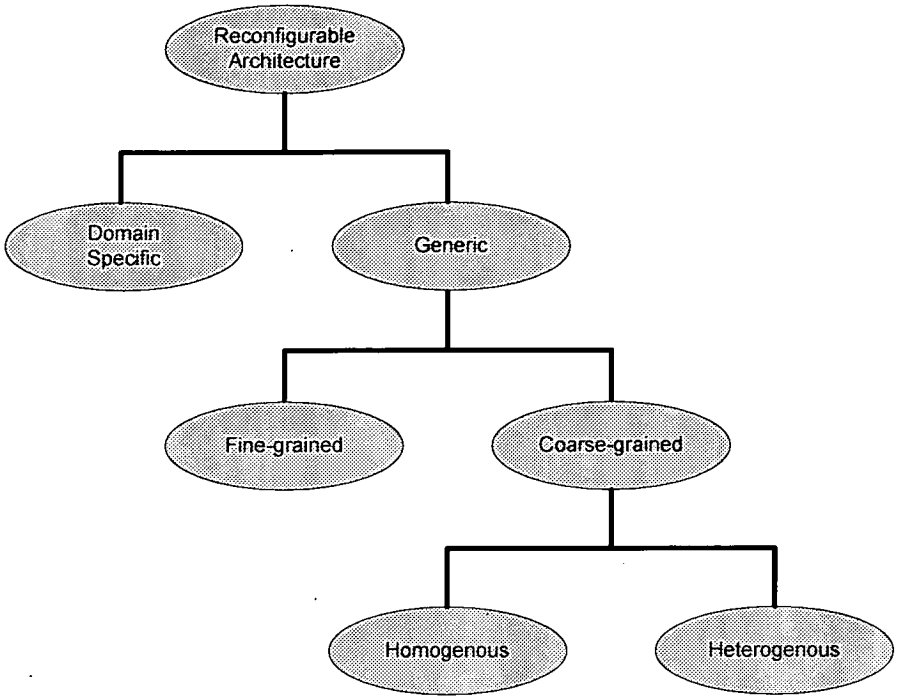


Figure 2.11 Categories of reconfigurable architectures

The categories of reconfigurable architectures are shown in Figure 2.11. In terms of the application domain of the fabric, the reconfigurable architectures can be split into domain specific reconfigurable architecture and generic reconfigurable architecture. According to the granularity of the function blocks inside the architecture, generic reconfigurable architecture can be further divided into fine-grained reconfigurable architecture and coarse-grained reconfigurable architecture. The previous works about the implementations of Viterbi decoder on the reconfigurable architecture were summarized as follows.

2.7.1 Domain Specific Reconfigurable Viterbi Decoder Implementation

By making hardware architectures less generic and specific to the Viterbi decoding domain, domain specific reconfigurable Viterbi decode leads to a combination of power efficiency, high throughput, area efficiency and flexibility.

[41] demonstrates reconfigurable architecture which can be configured to any Viterbi decoder type among K=3-7 and R=1/3-1/2. This design only considers the hard decision Viterbi decoder and hamming distance is calculated in the BMU. It also employed a full

parallel ACSU to speed up the decoding throughput. However, it only exploited one RAM bank to save the survivor bits resulting in a bottleneck for SMU and the maximum decoding throughput can only achieve 20Mbps.

A reconfigurable Viterbi decoder based on a partial parallel ACSU architecture is illustrated in [42]. It can be configured to a Viterbi decoder with K=7-10 and R=1/2. This fabric exploits four parallel butterfly units with 5-level pipelining which are scheduled to process all the trellis states over time. In the end, the fabric was implemented on Xilinx Virtex FPGA device. 12.625Mbps and 1.578Mbps decoding throughput are achieved in the case of constraint length 7 and constraint length 10, respectively.

[43] presents a reconfigurable Viterbi decoder supporting constraint length K= 4-14 and code rate 1/2 with an adaptive decoding algorithm. Instead of computing and retaining all 2^{K-1} possible paths, only those paths which satisfy certain cost conditions are retained at each trellis node. The adaptive decoding algorithm leads to various butterfly operations in ACSU. However the adaptive Viterbi algorithm requires additional decision circuits for each state to decide whether to retain the state or not, which relied on extra computational cost. As reported in [43], the decoding throughput only can achieve hundreds of *Kbps*.

Domain specific reconfigurable Viterbi decoders can achieve attractive performance for each configuration of Viterbi decoder with different constraint lengths and code rates. However, its application cannot stretch out the Viterbi decoding domain.

2.7.2 Viterbi decoder on generic reconfigurable architectures

There is another trend of reconfigurable architecture. Compared with domain specific architectures, this type of architecture is more generic than domain specific architecture and it aims to target the wider application domain. However, its performance is degraded as the cost of its flexibility.

According to the granularity of the function blocks, generic reconfigurable architecture can be classified into fine-grained reconfigurable architecture which corresponds with bit-level manipulation of data and coarse-grained reconfigurable architecture which implies word level operations.

2.7.2.1 Fine-grained Reconfigurable Architecture

The field programmable gate array (FPGA) device, which is depicted in Figure 2.12, drops into the fine-grained reconfigurable category. The operational elements of an FPGA are CLBs which are dominated by LUTs. A LUT is a 1-bit wide memory array in which the inputs to the LUT are the address lines and the LUT output corresponds with the 1-bit output of the memory. Any type of Boolean functions can be formed by a LUT. Commercial FPGAs, such as [44] and [45], employ 4-bits LUT as the basic element which consists of 16x1 SRAM cells to store the truth-table.

Practically, several LUTs are combined together to form a CLB in order to reduce the number of routing resources required to connect them. Figure 2.12 shows a typical CLB of Xilinx Virtex-E family [46], where each CLB contains four LUTs, organized in two similar slices.

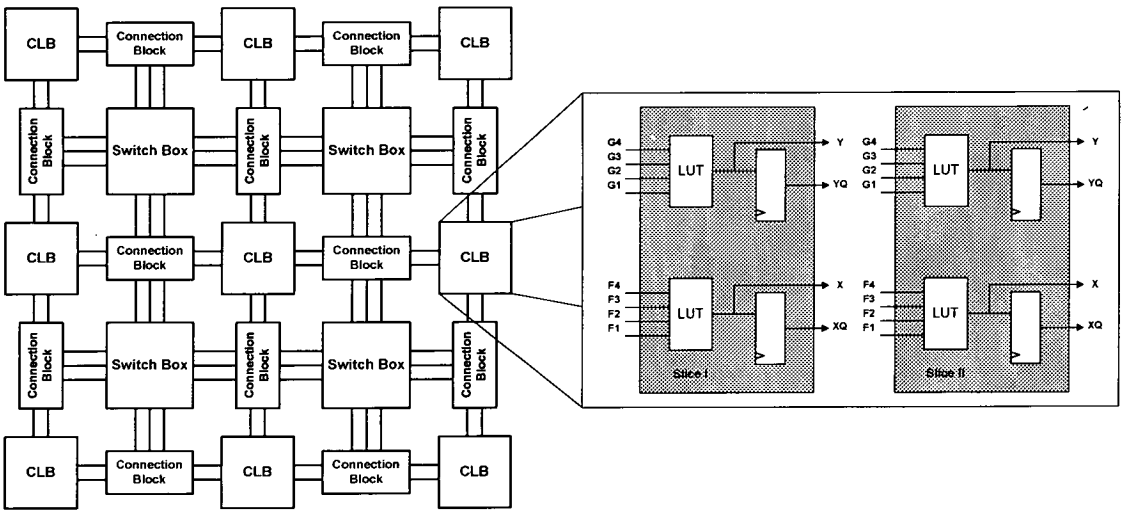


Figure 2.12 Fine-grained FPGA architecture

Routing resources of the fine-grained architecture account for the majority of the area, delay, and power dissipation. These routing resources, depicted in Figure 2.13, are used to connect CLBs to other CLBs and to I/O blocks. The architecture of the routing resources can be separated into three components: connection tracks, switch blocks, and connection blocks.

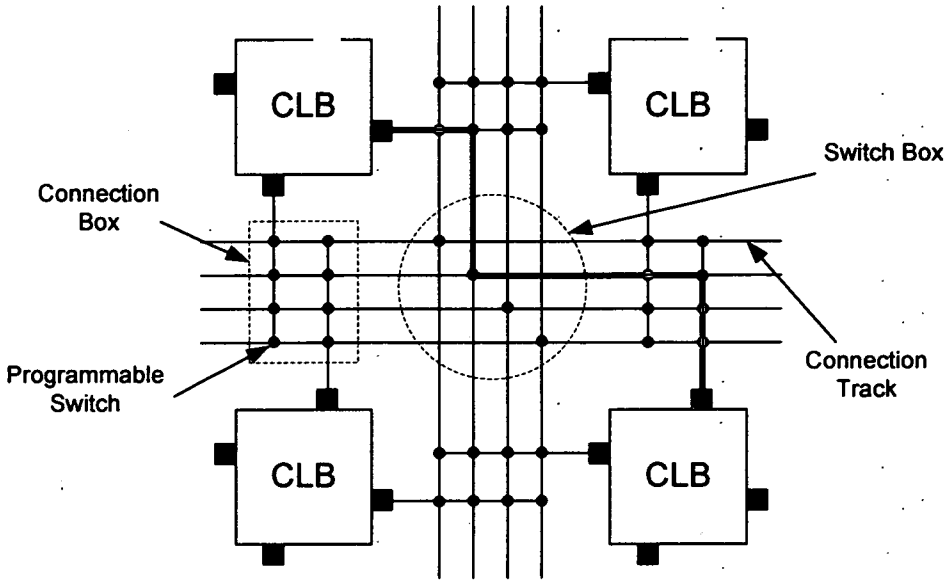


Figure 2.13 Routing resource of FPGA

Since FPGAs are flexible at bit level, a very wide range of applications can be mapped on FPGA. In terms of the Viterbi decoder, [47] [48] exhibited a Viterbi decoder on an FPGA device. However, this high flexibility also produces very high power consumption and area penalty which prohibits the application of FPGAs in the portable wireless communication domain.

2.7.2.2 Coarse-grained Reconfigurable Architecture

The definition of coarse-grained reconfigurable architecture is very wide. The architectures responding to this category are designed by increasing the granularity of their function blocks, thereby improving the computation efficiency and reducing the amount of interconnection resources. Unlike the LUTs in FPGA which are exactly the same with each other, these coarse-grained function blocks can be identical leading to a homogenous coarse-grained architecture, and also can be disparate resulting in a heterogeneous coarse-grained architecture.

2.7.2.2.1 MONTIUM

MONTIUM [52], a coarse-grained reconfigurable architecture, is proposed to execute highly regular, computational intensive DSP kernels in the Chameleon platform [53] which is a dynamically reconfigurable System-on-Chip (SoC) platform targeting 3G/4G wireless communication applications.

A MONTIUM tile consisting of five homogeneous 16-bit ALUs resembles a VLIW-like architecture, which is depicted in Figure 2.14. A single ALU has four 16-bit inputs. Each input has a private input register file that can store up to four operands. Input registers can be written via a flexible interconnection network. Each ALU has two 16-bit outputs, which are connected to the interconnection network. The ALU is entirely combinatorial and consequentially there are no pipeline registers within the ALU. In order to fulfill the high memory bandwidth requirement, 10 local memories (M01 to M10) are placed in parallel in each tile. The data between ALUs and memories are switched through a crossbar network.

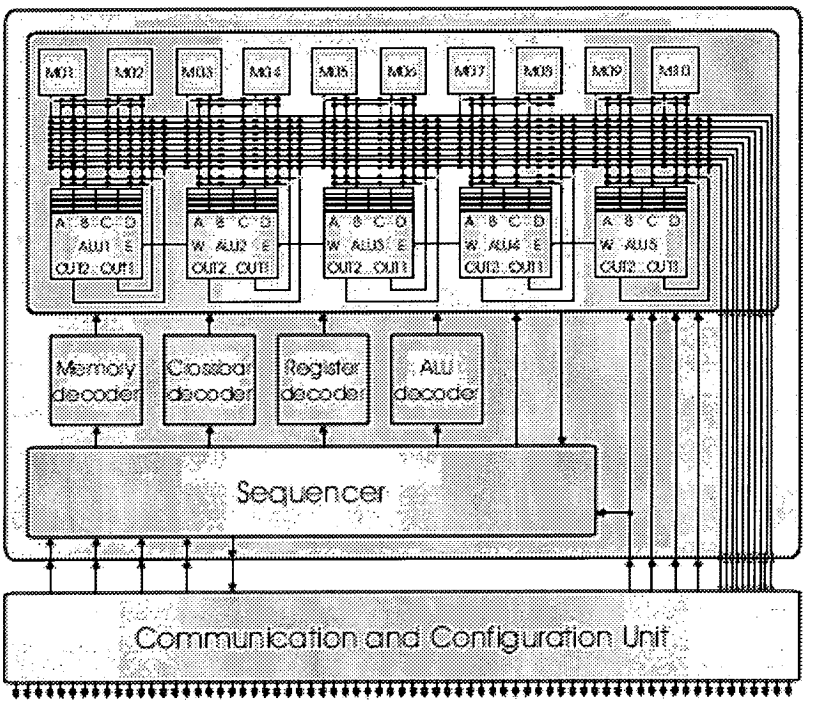


Figure 2.14 Architecture of MONTIUM

Viterbi decoder on MONTIUM

[54] presents implementing a Viterbi decoder on MONTIUM architecture. By adding a compare-select operation to the ALUs of the MONTIUM, 32 butterflies can be processed in 42 clock cycles. Since the path metric values have to be read and written instantaneously, [54] chose to read the input information from two memories and to write the results back to two different memories. The functions of the memory pairs are interchanged in the

consecutive stages of the trellis. In the end, [54] demonstrates a constraint length 7 and code rate 1/2 Viterbi decoder with 2.1 Mbps throughput on a MONTIUM architecture with 0.13μm technology.

2.7.2.2.2 MorphoSys

MorphoSys [55] is a reconfigurable DSP architecture targeted on computational intensive applications, such as wireless communication and image processing. The MorphoSys architecture, which is shown in Figure 2.15, comprises three main unique components: a 32-bits RISC processor called TinyRISC processor, a homogeneous array of 64 16-bits Reconfigurable Cells (RCs) connected by a mesh network, and a special data movement unit called Frame Buffer which can accelerate the data movement between external memory and RC array.

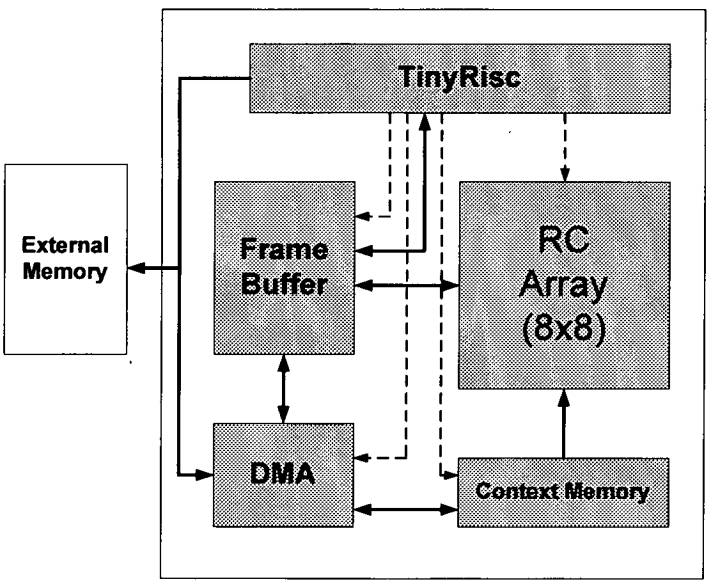


Figure 2.15 Architecture of MorphoSys

TinyRISC is always in charge of controlling the whole architecture as well as executing the sequential part of codes for applications. The 8x8 RC array is used to cover word-level, computation-intensive, intrinsically parallel applications. Each RC consists of functional units for arithmetic and logic operations, local memory to store the results and implement look up tables, input and output modules to form the RC array structure and a fine grain reconfigurable logic block which is a field programmable gate array (FPGA) and used for

implementing some customized functions.

The RC array follows the single instruction multiple data (SIMD) model of computation. All RCs in the same row/column share the same configuration data (context). However, each RC operates on different data.

Viterbi decoder on MorphoSys

In [56], a Viterbi enhanced MorphoSys is described to resolve the bottleneck in the Viterbi decoder. The ALU in each RC is modified by adding an add-compare-select and complex arithmetic units. Every clock cycle, 64 ACS computations can be executed on the 8x8 RC arrays. The trace back operation is performed in TinyRISC. Viterbi-enhanced MorphoSys is synthesized using 0.13 μm technology. The array area, not including main and sequential RISC processors, is 25mm^2. With 330MHz clock frequency, a constraint length 7 and code rate 1/2 Viterbi decoder can achieve 54Mbps.

2.7.2.2.3 Silicon Hive Processor

The configurable parallel-processing architecture designed by Silicon Hive [57] is ultra long instruction-word (ULIW) architecture, whose instruction words stretch up to 768 bits long. The foundation of Silicon Hive's ULIW architecture is a logic block called a processing and storage element (PSE). A PSE is a VLIW-like data path consisting of several interconnection networks (IN), one or more function units (FU), distributed register files (RF) and, optionally, local memory storage (MEM). All resources in a PSE are distributed, leading to lower power and reduced silicon footprint through locality of reference. The functions of PSE are varied and are configured at design time to tailor to a specific application domain.

Besides the data path blocks, a sequencer is also incorporated into the architecture. The sequencer is a simple state machine with a program counter to schedule the instructions in program memory to PSEs. Silicon Hive also provides its own C compiler, HiveCC, to translate high level language on its ULIW architecture.

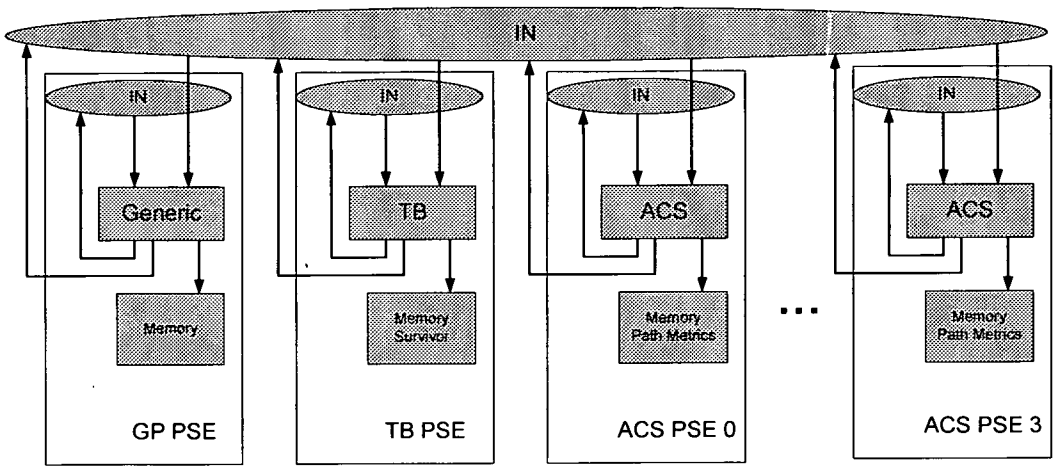


Figure 2.16 Architecture of Silicon Hive processor

Viterbi decoder on Silicon Hive processor

In [58], a multi-standard Viterbi decoding processor based on Silicon Hive's infrastructure is presented. According to [58], this processor consists of four identical PSEs optimized for add-compare-select (ACS) operations, each of them performing 32 butterflies in parallel, one PSE to accelerate traceback processing and one generic PSE which handles typical ALU operations and the synchronization protocols. The processor was fabricated in CMOS 0.13 μm technology resulting in 2.8 mm^2. When incorporated in a DVB-T system, the Viterbi processor was simulated to dissipate 47.7mW at 160MHz and achieve up to 54 Mbps throughput.

2.7.2.2.4 Transport Triggered Architecture (TTA)

TTA architecture [59] was developed to reduce the data path complexity and the underutilization of the register file in the VLIW DSP architecture.

Figure 2.17 depicts the principle of TTA architecture. A TTA processor with 32-bit data path consists of a set of heterogeneous blocks, such as function units (FUs), register units (RUs) and user-defined units (SFUs), which are connected by a crossbar network.

In TTA, FUs are responsible for performing operations on data. FUs receive data from the input sockets and when the operation is completed, the result data can be accessed from the output socket. Each FU contains several trigger and operant registers. The data to be

processed can be written to the operant registers in earlier cycles, but the operation is triggered only when instruction is written to the trigger register.

The instruction fetch unit (IFU) is responsible for fetching instructions from the memory and the load-store unit (LSU) provides an interface to the memory.

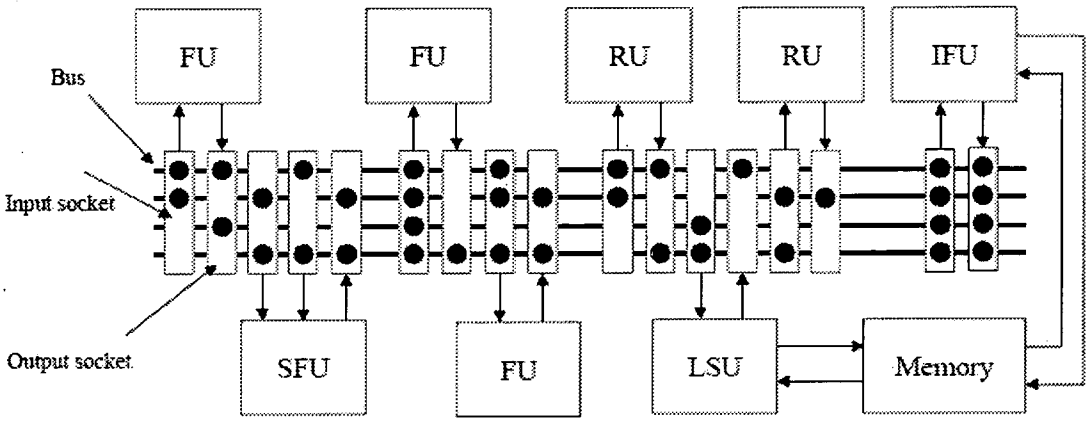


Figure 2.17 Architecture of TTA

Viterbi decoder on TTA

In [60], a constraint length K=9 and code rate 1/2, Viterbi decoder has been manually mapped on tailored TTA architecture. Besides general FUs, four specific SFUs are integrated into the TTA architecture to speed up the Viterbi decoding process.

The branch metric generation SFU gets two branch metrics, which are the negations or copies of the input branch metrics corresponding to the value of loop counter. Both the input operands and outputs are packed into a single 32-bit word.

The ACSU SFU can handle two ACS computations at a time and generate two updated path metrics which are packed into a 32-bit word. In addition, the ACSU SFU also will take two 32-bit operands, shift them, and insert two new survivor bits. The path metrics re-organizer SFU takes charge of reordering and repacking two path metrics in a 32-bit word. The address generation unit SFU generates addresses for loading and storing path metrics. In practice, the read and write base addresses are passed as initial operands which are saved in the internal state of the SFU.

This tailored TTA processor is synthesized with 0.11 μm technology and achieves 0.57 Mbps throughput for a constraint length 9 and code rate 1/2 Viterbi decoder.

2.8 Turbo coding

2.8.1 Turbo encoder

Turbo codes [16], which belong to a set of convolutional codes, provide very high reliability data transmission at very low signal to noise ratio. A classical Turbo encoder, as introduced in [16], comprises two recursive systematic convolutional (*RSC*) component encoders and an interleaver (INT) as depicted in Figure 2.18.

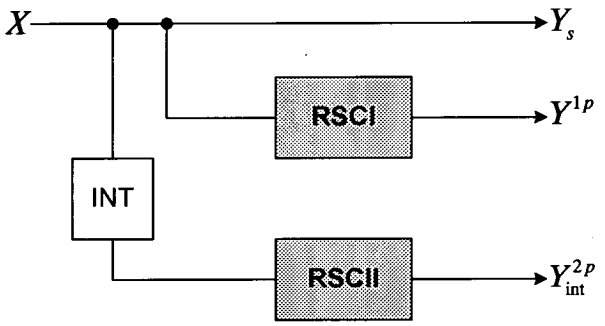


Figure 2.18 Classical Turbo encoder

The *RSC* encoder is a systematic convolutional encoder with feedback. Such an encoder with a two taps shift register is depicted in Figure 2.19. For systematic codes, the information sequence is a part of the codeword, which corresponds to the direct connection from the input to one of the outputs. For each input bit, the encoder generates two codeword bits: the systematic bit and the parity bit.

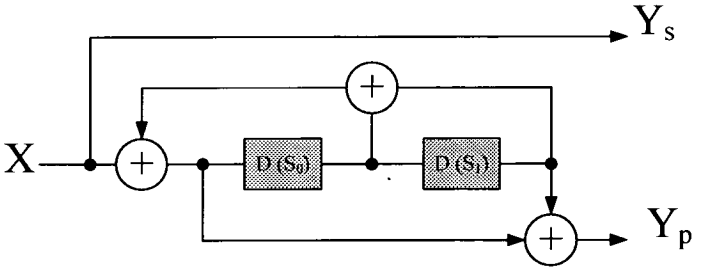


Figure 2.19 Constraint length 3, code rate 1/2 RSC encoder

Although both encoders work on the same frame of information bits, *RSCII* access the information bits in an interleaved order. The coded sequence to be transmitted is consisted of

two parity sequences generated by two *RSCs*, and a systematic sequence which equals X.

As compared with convolutional encoder shown in Figure 2.2, Turbo encoder is a recursive encoder. Since the performance of any binary codes is dominated by their free distance, the recursive component encoders can maximize their effective free distance [14]. In addition, the systematic encoder is somewhat simpler than non-systematic encoder because less hardware is required. On the other hand, non-systematic codes may be subject to catastrophic error propagation [14], which is called catastrophic codes that a finite number of channel errors cause an infinite number of decoding errors [14]. However, systematic codes are always non-catastrophic.

2.8.2 Turbo decoder

[16] not only proposed a new class of channel codes, but it also presented an efficient way of decoding them. Figure 2.20 shows the block diagram of a classical Turbo decoder presented in [16], where *INT* and *DINT* represent interleaver and de-interleaver, respectively. As can be seen, the decoder's main elements are component decoders, interleavers, de-interleavers, and a de-multiplexing unit. In the decoder, a block of received symbols is de-multiplexed into the systematic symbols and the parity symbols. The Turbo decoder is a typical soft-in-soft-out decoder [18]. Unlike Viterbi decoder, each component decoders use these soft symbols to calculate soft outputs, for example, *DECI* consumes the symbols Y^s and Y^{1p} produces the soft outputs Λ^{1e}. On the other hand, *DECII* consumes the symbols Y^s and Y_{int}^{2p} and produces the soft output Λ_{int}^{2}. In addition, each decoder takes into account a priori information, which is an interleaved version of the other decoder's soft outputs. Hence, Turbo decoding process consists of an exchange of information between two decoders which are connected with a feedback loop to support that exchange in an iterative fashion. After a sufficient number of iterations, we finally obtain the estimated decoding outputs from *DECII*.

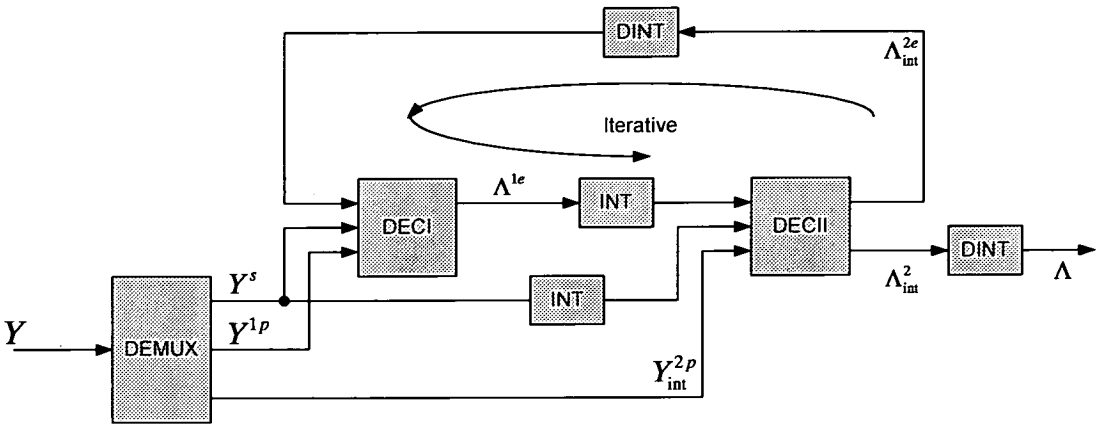


Figure 2.20 Classical Turbo decoder

An in-depth introduction to classical Turbo coding system would be beyond the scope of this thesis. For this part of information, the reader can refer to the literatures [17] [18] [19] [20]. Nevertheless, a modern set of Turbo codes, M-binary circular Turbo codes will be addressed in Chapter 6 and Chapter 7, where the details of encoder and decoder architectures for M-binary circular Turbo codes will be elaborated.

2.9 Conclusion

The basic concepts of wireless communication systems have been introduced in this chapter. Regarding the channel coding system, the Viterbi decoding algorithm is highly dependent on the constraint length, code rate and generator polynomials. According to the concept of 4G, flexible convergence of current wireless standards, a reconfigurable Viterbi decoder with varied parameters is a must. Current implementation approaches for different modules, such as BMU, ACSU and SMU of a Viterbi decoder are elaborated. Furthermore, the concept of reconfigurable architecture is addressed. Different trends of reconfigurable Viterbi decoder design methodology and architecture are described and compared.

Chapter 3:

Reconfigurable Viterbi Decoder Architecture I

Forward error correction (FEC) is a channel coding approach to overcome the variable deterioration during the wireless transmission, whereby the transmitter adds redundant data to the messages, and the receiver can detect and correct errors without the need to ask the transmitter for retransmission. FEC codes, such as Reed-Solomon codes, convolutional codes, Turbo codes and LDPC codes, have played a crucial role in telecommunication scenario. Based on various channel conditions and system requirements, different coding schemes are integrated to different communication standards. For example, in terms of convolutional codes, GSM [7] specifies constraint length 5 and code rate 1/2 convolutional codes, otherwise WLAN [10] and WiMAX [11] require constraint length 7 with code rate 1/2 and 1/3, and 3G [8] demands constraint length 9 and code rate 1/2 or 1/3 convolutional codes.

In the case of convolutional codes, the encoder architecture is very simple, normally consisting of several *XOR* gates and shifters. Both software and hardware approaches can

fulfill the system requirement. However, the decoder architecture is much more complex than encoder. In order to provide the decent performance, such as low power consumption and high throughput, decoder is always implemented by a hardware approach. Moreover, the post-fabrication reconfigurability is crucial for hardware devices in beyond 3G wireless communication system. Therefore, there is a need to investigate a flexible decoder architecture which can support the decoding process of various convolutional codes with different constraint lengths and code rates.

This chapter presents a reconfigurable fabric for Viterbi decoder which can support the decoding process for convolutional codes with constraint lengths from 3 to 9 and code rates 1/2 and 1/3. Totally, there are four salient features of the proposed fabric.

As a reconfigurable Viterbi decoder targeting multiple standards, it not only needs to support different types of convolutional codes, but also has to satisfy the decoding throughput requirement. The decoding throughput varies from standard to standard. For instance, GSM supports up to 270kbps, 3G supports up to 2Mbps, DVB supports up to 20Mpbs, and WiMAX can provide up to 75Mbps. The proposed architecture offers a maximum 150Mpbs decoding throughput, which can fulfill the requirement of all current standards.

Secondly, reconfigurable RAM blocks are integrated into the architecture. Every RAM block is built by synthesizable latches and the size of the RAM can be changed depending on the required application. Compared with custom RAM, the synthesizable reconfigurable RAM is more flexible and can be easily tailored to a particular application.

Thirdly, several power-saving schemes are incorporated into the architecture. For a particular application, the unused blocks will be automatically turned off. Thus, dynamic power consumption of these blocks can be reduced to zero, which results in a significant power saving on the proposed architecture.

Finally, the proposed architecture can be dynamically configured. Its function is controlled by *405 configuration bits*. By means of loading different configuration bits, a Viterbi decoder with different parameters can be mapped on the proposed fabric.

3.1 Reconfigurable Viterbi decoder architecture

The block diagram of the proposed reconfigurable fabric is shown in Figure 3.1. The following sections will break down the whole architecture into several blocks and analyze the design approaches of each block. It can be seen from Figure 3.1 that this fabric consists of three major blocks: branch metric unit (BMU), add-compare-select unit (ACSU), and survivor management unit (SMU).

The following chapter is organized as follows: Section 3.1.1 describes the architecture of the BMU in the proposed reconfigurable Viterbi decoder. ACSU and SMU are addressed in Section 3.1.2 and Section 3.1.3 respectively. Section 3.1.4 describes the LIFOs which are employed to reorder the output message. The configuration memory is presented in Section 3.1.5.

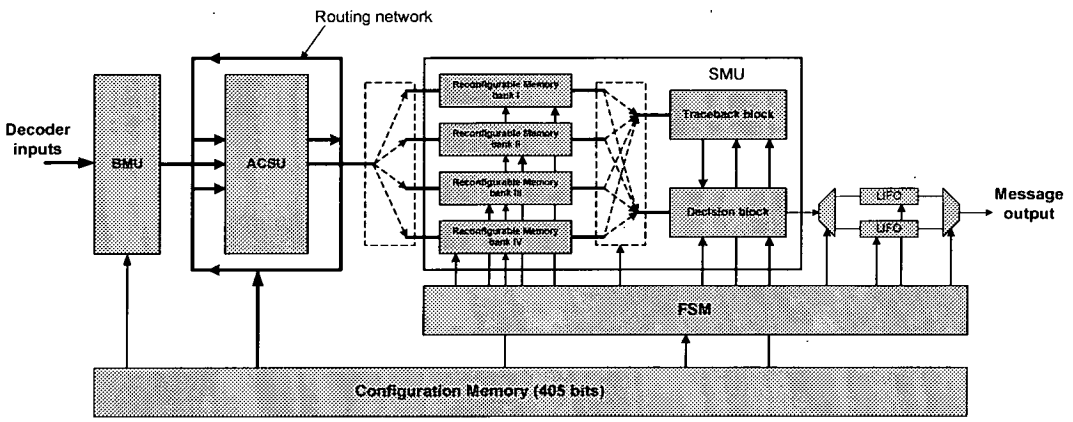


Figure 3.1 Block diagram of reconfigurable Viterbi decoder architecture I

3.1.1 Reconfigurable BMU

BMU calculates the branch metrics which represent the distances between received signals and ideal transmitted signals [6]. According to [6], for a constraint length K, code rate 1/R Viterbi decoder, there are 2^{K-1} states at every trellis stage and two branch metrics going into each state. Thus, totally 2^K branch metrics need to be considered. However, among 2^K branch metrics, only 2^R unique branch metrics have to be computed at every trellis stage. If x, y, z denote quantized received symbols, the branch metric computations for code rate 1/2 and 1/3 Viterbi decoder are shown in Table 3.1.

Table 3.1 Branch Metric Computation

Code Rate	1/2	1/3
Branch Metric Computation	$BM_{00} = x + y$	$BM_{000} = x + y + z$
		$BM_{001} = x + y - z$
	$BM_{01} = x - y$	$BM_{010} = x - y + z$
		$BM_{011} = x - y - z$
	$BM_{10} = -x + y$	$BM_{100} = -x + y + z$
		$BM_{101} = -x + y - z$
	$BM_{11} = -x - y$	$BM_{110} = -x - y + z$
		$BM_{111} = -x - y - z$

It can be seen from Table 3.1 that a half group of branch metrics are antipodal with the others. For instance, the value of BM_{00} is opposite to that of BM_{11}, and BM_{111} is opposite to BM_{000}. Thus, half of the addition/subtraction operations in BMU can be replaced by inverters. On the other hand, the proposed BMU must support both code rate 1/2 and 1/3 branch metric computations. A straightforward way is to provide two individual circuits, one for each code rate. Obviously, it will increase the power and area overhead. However, from Table 3.1, it can be seen that branch metric computation for code rate 1/2 is a part of code rate 1/3 branch metric computation. For example, $BM_{000} = x + y + z = BM_{00} + z$ and $BM_{100} = -x + y + z = BM_{10} + z$. Hence, we can reuse the code rate 1/2 branch metric computation circuits during code rate 1/3 branch metric computation to decrease the power and area overhead.

The proposed BMU which can support both code rate 1/2 and 1/3 is shown in Figure 3.2. The proposed architecture consists of two major blocks. Block ***A*** takes charge of branch metric computation for code rate 1/2. Block ***B*** will be active, if code rate 1/3 Viterbi decoder is required. ***MUX I***, ***II*** and ***III*** are used to configure the data path and are controlled by the same configuration bit. For a code rate 1/2 Viterbi decoder, ***MUX I*** and ***II*** are switched to 0 value input, thus there are no data triggers in block ***B*** and the dynamic power consumption in block ***B*** is zero. ***MUX III*** will output branch metrics for code rate 1/2. In the case of code rate 1/3, both blocks ***A*** and ***B*** are active. ***MUX I*** and ***II*** will transfer the results of block ***A*** as the inputs to block ***B***. Finally, ***MUX III*** selects to output branch metrics for code rate 1/3.

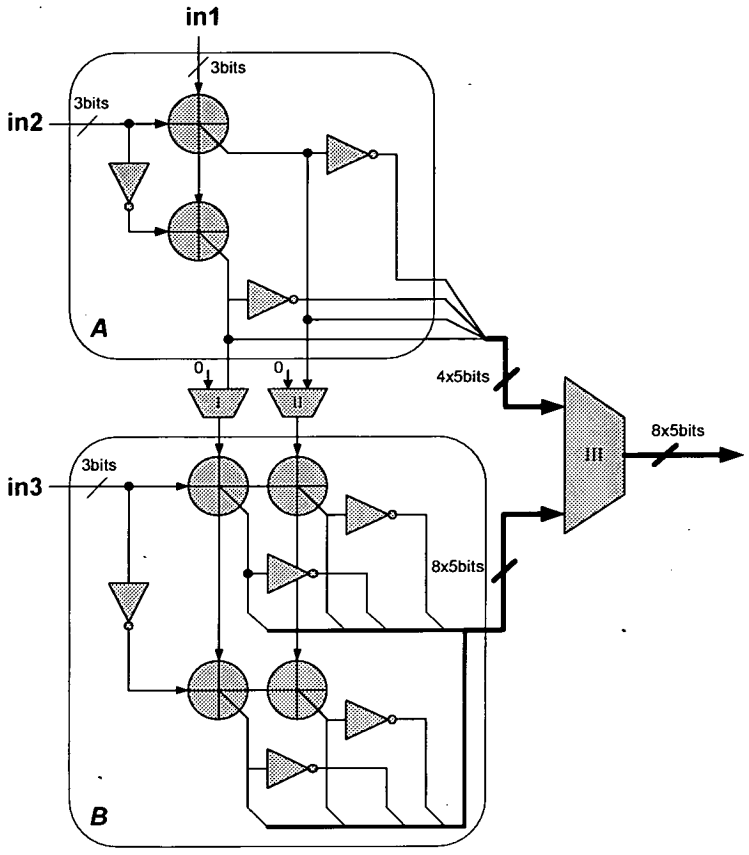


Figure 3.2 BMU for both code rate 1/2 and 1/3

3.1.2 Reconfigurable ACSU

Since our target is a high throughtput Viterbi decoder, a full parallel ACSU architecture with modulo arithmetic normalization is employed in this design. As described in Chapter 2, each butterfly unit will consume two path metrics and two branch metrics, whilst output two updated path metrics and two survivor bits. However, the input and output data path for each butterfly unit is highly dependent upon the constraint length, code rate and generator polynomials. Therefore, an efficient and flexible butterfly unit has been proposed for the reconfigurable fabric, which is shown in Figure 3.3. The following sections will present the details for each part of the reconfigurable butterfly unit.

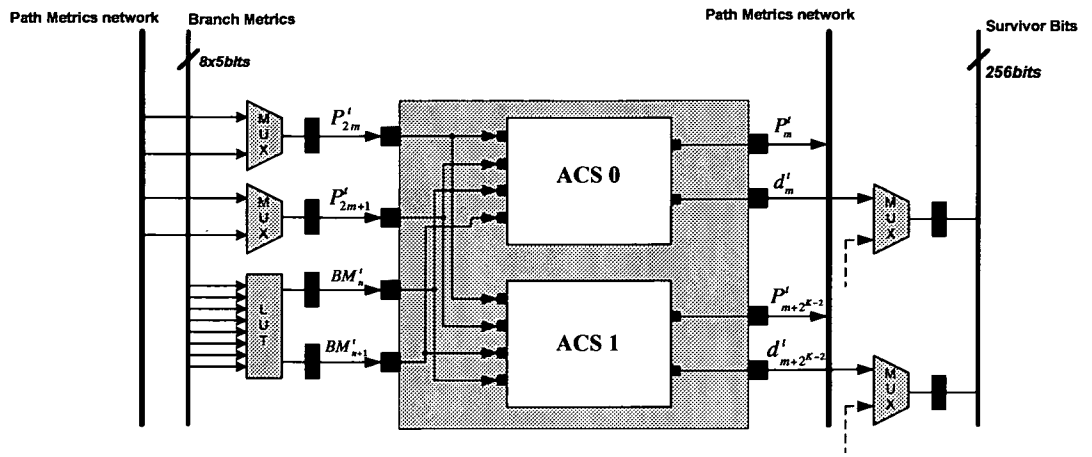


Figure 3.3 Reconfigurable Butterfly Unit

3.1.2.1 Routes for branch metrics

The branch metrics for an individual butterfly unit are irregular and decided by generator polynomials. Due to the symmetric characteristic of generator polynomials [14], two branch metrics accessed by a butterfly unit are antipodal with each other. For instance, if BM_{00} is one of the branch metric inputs, BM_{11} must be the other. Similarly, if BM_{100} is accessed by a dedicated butterfly unit, BM_{011} is also accessed by the same butterfly unit.

Hence, branch metric interface for each butterfly unit can be implemented by a look-up table (LUT). The number of elements in a LUT is decided by the rate of codes. Since the proposed core intends to support both 1/2 and 1/3 code rates, content of the LUT is defined in Table 3.2. According to configuration bits (index), suitable branch metric pairs are selected for an individual butterfly unit.

Table 3.2 Look-up table for branch metric routing

Index	LUT outputs
000	BM_{000} & BM_{111} or BM_{00} & BM_{11}
001	BM_{001} & BM_{110} or BM_{01} & BM_{10}
010	BM_{010} & BM_{101} or BM_{10} & BM_{01}
011	BM_{011} & BM_{100} or BM_{11} & BM_{00}
100	BM_{100} & BM_{011}
101	BM_{101} & BM_{010}
110	BM_{110} & BM_{001}
111	BM_{111} & BM_{000}

3.1.2.2 Routes for path metrics

In general, a butterfly unit will read two path metrics from local registers and write two updated path metrics back. However, the addresses of path metric read and writing are dependent on the constraint length. In the case of constraint length K=3, two path metric inputs of butterfly unit 1 are produced by butterfly unit 0 and butterfly unit 1, respectively. In the case of constraint length K=4, two path metric inputs of butterfly unit 1 are produced by butterfly unit 2 and 3, respectively.

Let us suppose, at time t, path metric inputs of butterfly unit m are $B_m^t\_in_0$ and $B_m^t\_in_1$, and path metric outputs are $B_m^t\_out_0$ and $B_m^t\_out_1$. After observing the trellis structures for all different constraint lengths, we can summarize the rule for the path metric routing network, which is described by Equation 3.1 and Equation 3.2.

$$B_m^t\_in_0 = \begin{cases} B_{2m}^{t-1}\_out_0 & if\ m < 2^{k-3} \\ B_{2m-2^{k-2}}^{t-1}\_out_1 & otherwise \end{cases} \tag{3.1}$$

$$B_m^t\_in_1 = \begin{cases} B_{2m+1}^{t-1}\_out_0 & if\ m < 2^{k-3} \\ B_{2m-2^{k-2}+1}^{t-1}\_out_1 & otherwise \end{cases} \tag{3.2}$$

As can be seen from Equation 3.1 and Equation 3.2, every path metric input for an individual butterfly unit has two potential connections which are decided by constraint length. Thus, two ***MUX*** blocks are inserted into each butterfly unit to select associated path metrics for Viterbi decoders with different constraint lengths.

3.1.2.3 Routes for survivor bits

In addition to two path metrics, a butterfly unit also produces two survivor bits at a time, which will be stored in survivor memory and used by the SMU to generate the decoding sequence. Since the maximum constraint length we considered is 9, the largest word length of survivor memory is 256bits. However, the order of these 256bits is also decided by constraint length. If assuming $surbit_m$ is the m^{th} bit among 256bits, and $B_m^t\_bit_0$ and

$B_m^t\_bit_1$ are the survivor bits generated by butterfly unit m at time t, the survivor bits routing network can be defined by Equation 3.3.

$$surbit_m = \begin{cases} B_m^t\_bit_0 & if\ m < 2^{k-2} \\ B_{m-2^{k-2}}^t\_bit_1 & otherwise \end{cases} \quad (3.3)$$

Similarly with the rule for path metrics, there are two potential positions for each one individual bit, and a ***MUX*** with two inputs can be used to switch the survivor bits for different constraint lengths.

3.1.2.4 Low power strategy

Since a full parallel architecture is considered in our design, the ACSU consists of 128 butterfly units. However, for a particular application, not all of them are active. In the case of constraint length K=7, only 32 butterfly units are used during the decoding process. On the other hand, the ACSU is the main logic module of the Viterbi decoder and power consumption of the ACSU takes the biggest portion in the whole design. Thus, if inactive butterfly units can be powered off, it can achieve a significant power reduction for the whole design.

Clock gating [61] is a helpful approach to limit the power consumption of the unused butterfly units. It can be seen from Figure 3.3 that all data inputs to a butterfly unit are triggered by *flip-flop*. In Figure 3.4, an *AND* gate is used to turn off the clock input of *flip-flop*, thus neither inputs, nor outputs, of butterfly units are toggled, and the dynamic power consumption of the unused butterfly units are down to zero. Since the hardware overhead is very small, the clock gating is very efficient at limiting the power consumption of the unused blocks. Compared with the design without clock gating, nearly 50% of power consumption can be saved in the case of constraint length K=7. The more detailed comparison results are given in Section 3.2.

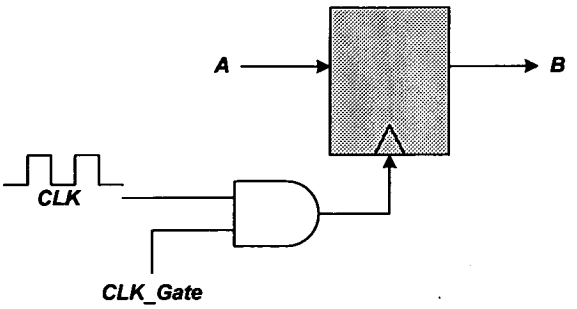


Figure 3.4 Clock gating

3.1.3 Reconfigurable SMU

As described in Chapter 2, the SMU is composed of logic section and memory section. Due to the high throughput and low power consumption requirements of a portal device, sliding window and trackback based SMU is employed in the reconfigurable fabric.

3.1.3.1 Reconfigurable RAM

As described in previous sections, the ACSU generates a survivor vector which consists of 2^{K-1} bits at a time which are stored in memory banks of SMU to rebuild the transmitted message. On the other hand, the size of each memory bank has to be up to five times of the constraint length K in order to ensure the trellis can reach S_{merge}. Thus, the memory size varies depending on the constraint length. Based on upwards aspects, a synthesizable RAM which can be configured to different memory sizes is developed to provide the required flexibility.

RAM is usually delivered from the vendors (such as *UMC* and *TSMC*) as hard-macro. The designer receives an HDL simulation model and a layout abstract view which defines the ports and dimension of the hard-macro from the vendor. However, RAMs provided by vendors are blocked to designer and they cannot be optimized for a particular design. Therefore, we developed our own synthesizable RAMs to accommodate to the reconfigurable architecture.

The proposed synthesizable RAM is described in register transfer level (RTL) by hardware description language (HDL). HDL program can be transferred to hardware gates and

mapped to a specific technology (such as 0.18um) by synthesis tools. The proposed RAM is depicted in Figure 3.5, where the basic element is a latch. The number of words is 64, which provides a trace back length up to 64.

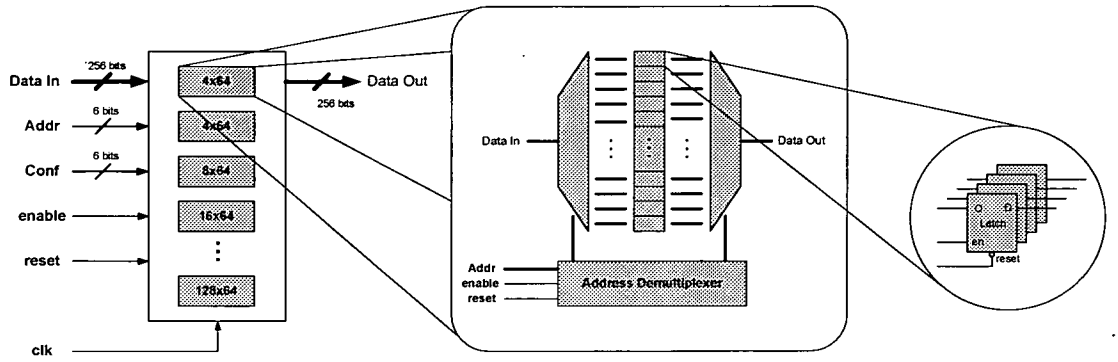


Figure 3.5 Diagram of reconfigurable RAM

In addition, the proposed RAM is composed of seven sub-banks. The effective bits per word vary from 4 to 256bits, which is configured by six configuration bits. The modes of the RAM are tabulated in Table 3.3.

Table 3.3 Modes of the reconfigurable RAM block

Configuration Bits	Size of Memory
000000	4x64 bits
000001	8x64 bits
000010	16x64 bits
000100	32x64 bits
001000	64x64 bits
010000	128x64 bits
100000	256x64 bits

The controller inside RAM can block the data toggles and clock of the unused sub-banks. Therefore, in a particular application, only the required sub-banks will be active. Although it incurs a slight area increase, the power consumption can be dramatically reduced as compared with the design employing a non-reconfigurable RAM for all the applications.

3.1.3.2 Reconfigurable trace back logic block

The trace back operation starts from an arbitrary state (merge block) or from a certain state (decision block). The current state S_n is a bit line index to the survivor bit d_n. Based on Equation 2.8, previous state can be decided by current state and associated survivor bit, where a shifter operation is employed. This recursive operation is continued till all outputs are produced.

However, the size of the shifter is decided by constraint length. Hence, we proposed a reconfigurable shift register to handle the computation which is described in Equation 2.8. The proposed shift register, shown in Figure 3.6, consists of eight flip-flops and seven multiplexers and the length of shift registers can be varied from two to eight. Multiplexers are used to control the data path of flip-flops whose inputs can be either output from the previous flip-flop or zero. Seven configuration bits (C_0 to C_6) are exploited to decide the length of the shift registers according to the constraint length of the targeted Viterbi decoder.

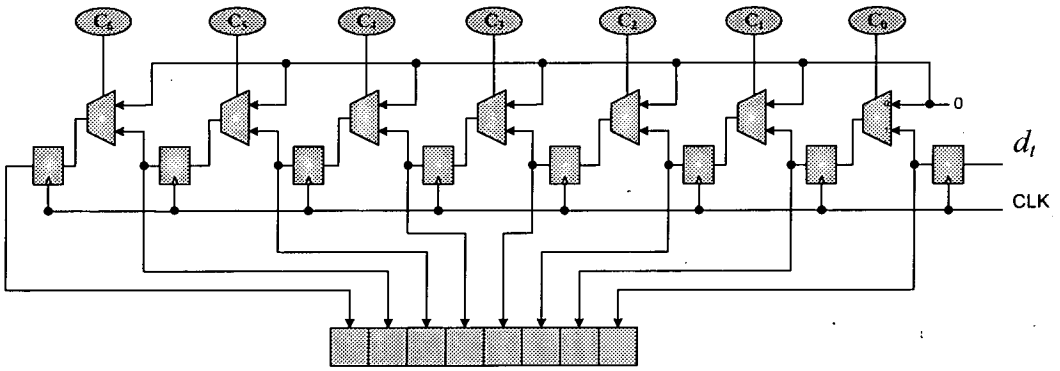


Figure 3.6 Reconfigurable traceback logic block

3.1.4 Decoder output LIFO

The Viterbi decoder based on TBA naturally produces a decode sequence in a reversed order. A bit reordering circuit is required to arrange the decoding output to a correct order. In the proposed architecture, two latch based last-in first-out (LIFO) buffers are exploited to correct the order of output sequence, which is shown in Figure 3.7. The maximum length of each LIFO is the same with memory block in SMU, but the stack size is set by six configuration bits, which indicate the traceback length.

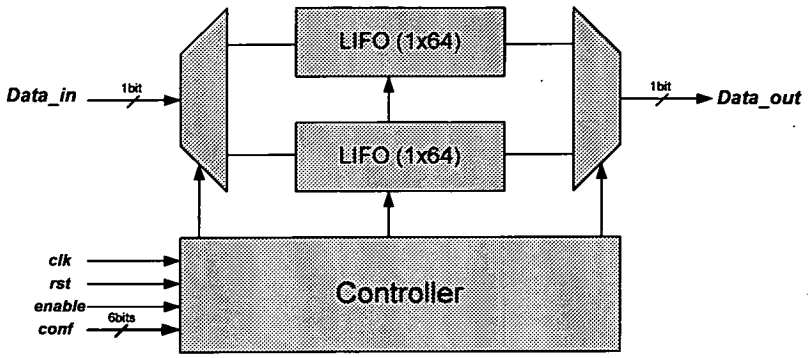


Figure 3.7 Diagram of LIFO

3.1.5 Configuration memory

This memory stores the configurable bits which are used to program the proposed architecture to a particular application. The total configuration bits of the proposed reconfigurable Viterbi decoder are 405 bits, shown in Figure 3.8.

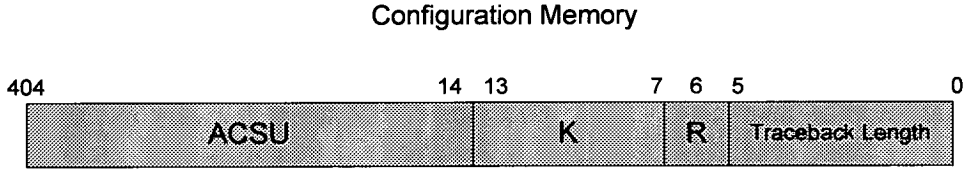


Figure 3.8 Structure of configuration memory

It can be seen that, the first 6bits (5:0) in configuration memory are used to set the traceback length. The following 8bits (13:6) are employed to select different constraint length and code rate. The rest of the bits are employed to configure the routing network for ACSU. Initially, these configuration bits were stored in the system memory. During the run-time, the operation system will take charge of downloading different configuration bits into the configuration memory, thus various Viterbi decoders can be mapped on the proposed architecture. For example, if the system runs at 100 MHz, since every clock cycle only one configuration bit loads into the configuration memory, the proposed fabric requires 4.05μs to reconfigure its function, which allows the core to be configured during the run-time.

3.2 Performance comparisons

3.2.1 Design flow

The basic design flow of the proposed architecture is illustrated in Figure 3.9, which helps in

understanding the results presented in this section. The design is firstly defined at the register transfer level (RTL) by using Verilog hardware description language. The functionally correct design is then synthesized by *Synopsys DesignCompiler* to convert the RTL description into a gate level netlist. The synthesis tool generates a delay file in standard delay format (SDF) for more accurate timing simulation and also provides the area of the design. The functionality of the design is again verified at the gate level using *Cadence SimVision*. In case of any problems, either the RTL code or/and the synthesis timing constraints have to be modified. The power consumption for some designs is calculated at the gate level. The power consumption is estimated by *Synopsys DesignPower*. It uses a gate level netlist along with the net switching activity obtained after gate level simulation to compute the power consumption. The switching activity is generated by defining the whole design as the toggle region in the testbench. The toggling of each net is recorded in the switching activity file called SAIF (Switching activity interchange format).

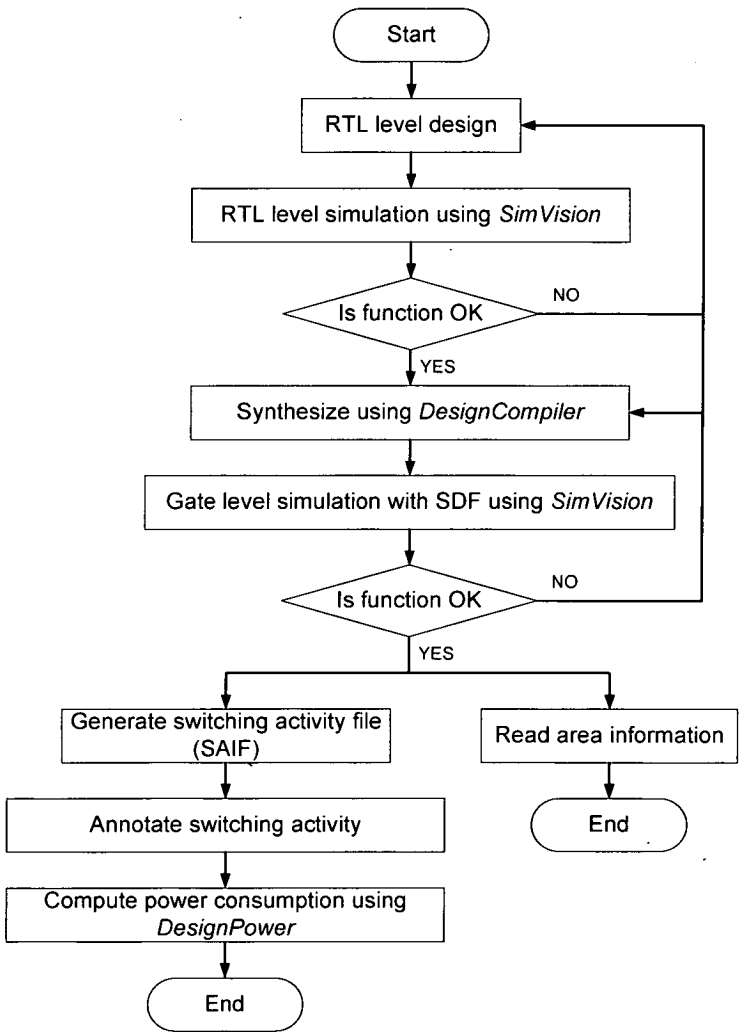


Figure 3.9 Flow chart of the design flow

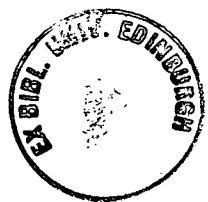
The proposed reconfigurable Viterbi decoder fabric has been implemented on a UMC 0.18μm CMOS technology library following the presented design flow. For comparative results, eight fixed Viterbi decoders (constraint length 3, 5, 7 and 9 with code rate 1/2 and 1/3) are also implemented with the same CMOS technology. All designs were synthesized by *Synopsys DesignCompiler* at a clock frequency of *125 MHz*, which means the core can generate a decoding throughput up to *125 Mbps*.

Table 3.4 Power consumption (mW) of different test cases

	ASIC	Architecture with power saving strategy	Architecture without power saving strategy
K3R1/2	0.39	2.75	17.35
K3R1/3	0.45	2.91	20.70
K5R1/2	1.13	4.09	17.38
K5R1/3	1.46	4.40	20.13
K7R1/2	4.72	9.14	17.76
K7R1/3	4.99	10.38	21.04
K9R1/2	20.56	28.75	23.50
K9R1/3	26.12	33.31	28.08

Table 3.5 Area (μm^2) of different test cases

	ASIC	Architecture with power saving strategy	Architecture without power saving strategy
K3R1/2	48,817	232,396	229,068
K3R1/3	49,903	237,442	234,132
K5R1/2	226,651	534,664	521,404
K5R1/3	235,794	539,711	526,416
K7R1/2	838,758	1,743,738	1,690,752
K7R1/3	842,584	1,748,785	1,695,763
K9R1/2	5,907,020	6,580,033	6,368,140
K9R1/3	5,984,808	6,588,592	6,375,673



3.2.2 Power consumption

The power evaluation was carried out by *Synopsys DesignPower* at a clock frequency of *10MHz*. In order to investigate the power distribution of the core, we implement eight test cases (constraint length 3, 5, 7, 9 with code rate 1/2 and 1/3) on the proposed fabric. The distributions of power consumption for different test cases are illustrated in Figure 3.10. It can be seen that ACSU occupies the biggest portion, around 52.54% averagely. The reason for this is that 2^{K-1} ACS operations are executed in parallel to ensure that the required decoding throughput can be achieved. In addition, four memory banks in SMU consume around 25.52% of total power of the whole core leading to the second biggest power thirsty part. As mentioned in previous sections, the power saving strategies, such as clock gating and memory distribution, have been employed into these two modules, where the dynamic power consumption of the unused blocks is reduced to zero for each particular application.

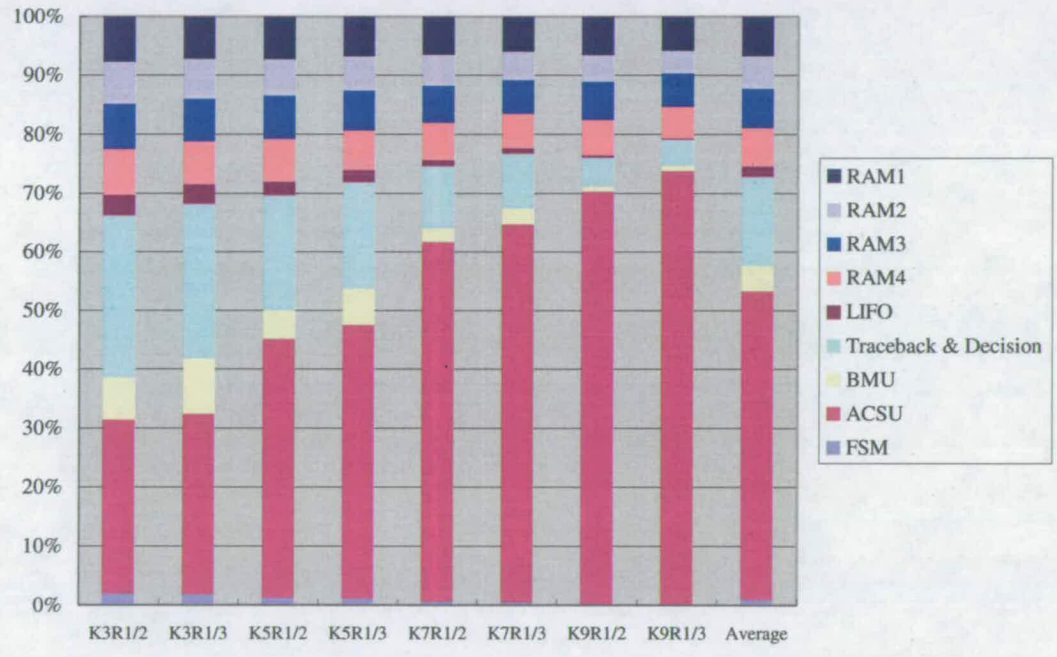


Figure 3.10 Distribution of power consumption

The power consumption comparisons among ASIC, reconfigurable architecture with power saving strategy and without power saving strategy are presented in Figure 3.11. The power values have been normalized so that the ASIC design has unity power consumption. It can

be seen that the proposed fabric consumed around 3.4 times power as compared with ASIC. However, since the unused blocks are turned off which are conducted by clock gating and distributed memory approaches, the proposed architecture with power saving can reduce 79.3% power consumption as compared with the architecture without power saving. Since the power consumption is proportional to the clock frequency, we also can scale the power numbers presented in Table 3.4 by 125/10 to obtain the power performance of the core when running at *125 MHz*.

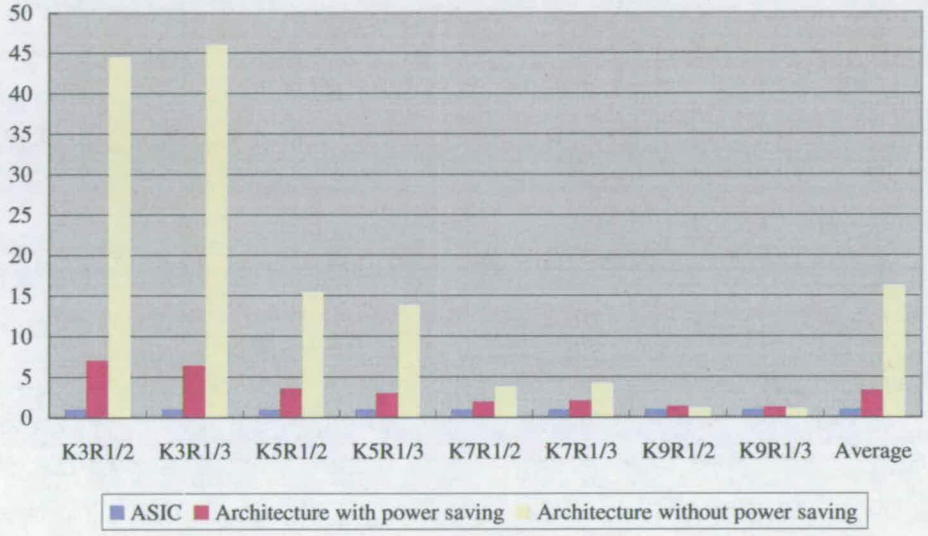


Figure 3.11 Normalized power comparison (ASIC = 1)

3.2.3 Area comparison

The area distributions occupied by different modules of different test cases on the proposed reconfigurable fabric are illustrated in Figure 3.12. Similarly, we also provide the average occupation of each module. As can be seen, memory banks take a big portion of the whole die area, around 62.43% averagely. It can be note that in small cases, such as K3R1/2 and K3R1/3, *Traceback and Decision* module takes the biggest portion of area. With the increasing of constraint length, the area percentage of this module is dramatically reduced, because bigger memory size and more butterfly units are employed.

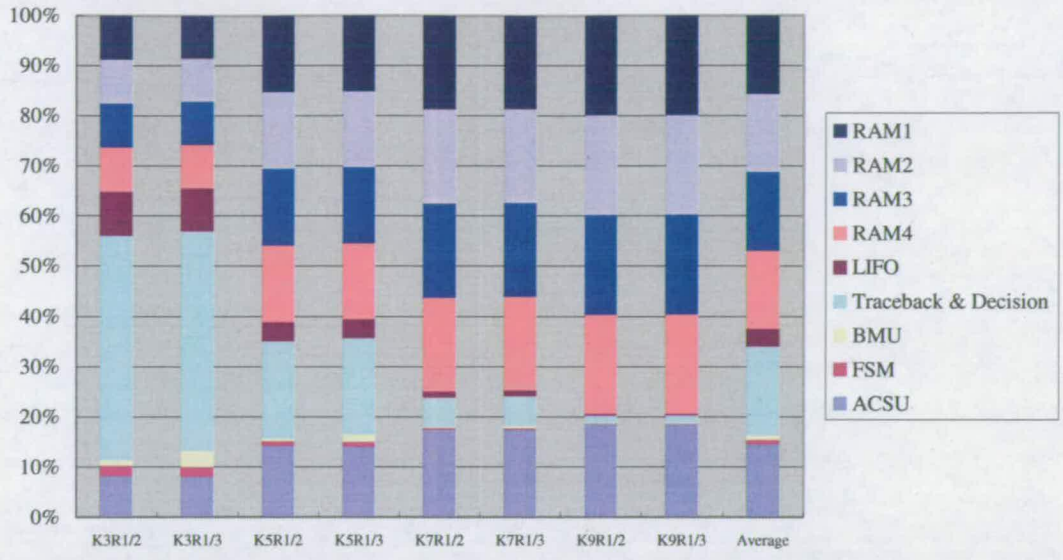


Figure 3.12 Distribution of area

The normalized area comparison is shown in Figure 3.13. It can be seen that the overhead of reconfigurable fabric with a power saving strategy is only 2.2% as compared with the one without power saving strategy. Due to its attractive reduction in terms of power consumption, it is worth paying this area overhead. Compared with ASIC, this fabric will pay 2.7 times area penalty as the price for its flexibility.

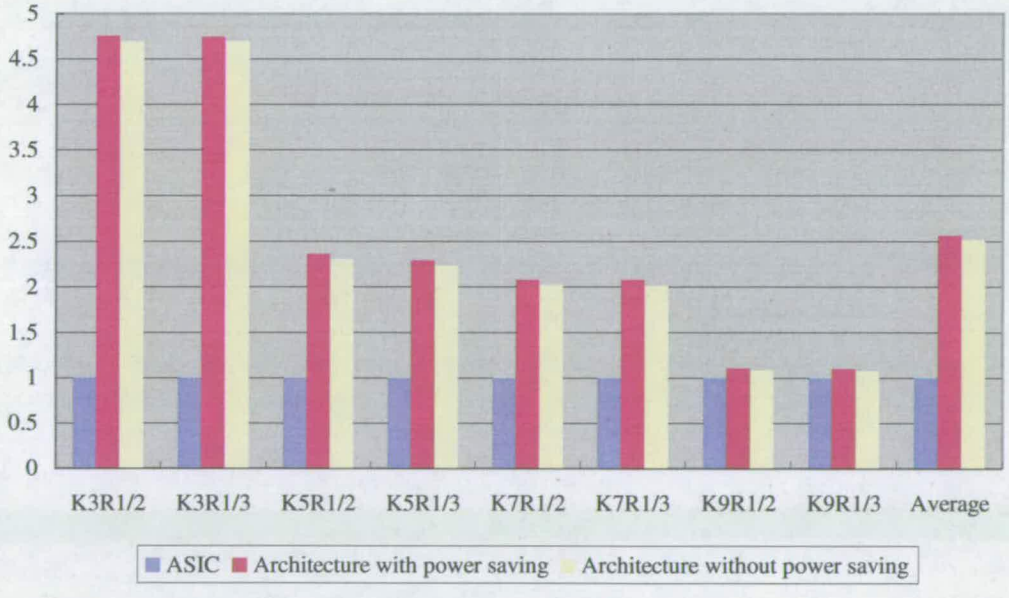


Figure 3.13 Normalized area comparison (ASIC = 1)

3.2.4 Comparison with other state-of-the-art works

The comparison with other state-of-the-art domain specific reconfigurable fabrics in terms of throughput is tabulated in Table 3.6. To our knowledge, our Viterbi fabric is the only published reconfigurable implementation which can support *150Mpbs* decoding throughput, and also provide flexibility and low power consumption.

Table 3.6 Comparison with other state-of-the-art works

	Platform	**Maximum Throughput**	**Maximum Decoding Frequency**
[41]	FPGA	20Mpbs	20MHz
[42]	FPGA	1.578Mpbs—12.625Mpbs	101MHz
[43]	FPGA	82.3Kpbs—333.7kpbs	17.2MHz
Proposed	ASIC 0.18μm	150Mpbs	150MHz

[41] demonstrated a K=3-7 and R=1/3-1/2 Viterbi decoder which was implemented on the FPGA device. This design adopted a full-parallel ACSU to speed up the decoding throughput. However, it only exploited one RAM bank to save the survivor bits leading to a bottleneck for the SMU and the decoding throughput can only achieve *20 Mpbs* on an FPGA device. In addition, it considers the Viterbi decoder based on hardware decision, the decoding performance would be worse than the proposed fabric.

[42] exploited a partial parallel ACSU with 5-level pipelining architecture to support K=7-10 and R=1/2 Viterbi decoder. This fabric employed four butterfly units to carry on ACSU computations for all the applications. This approach can save the die area of the core, but partial parallel ACSU architecture suffered from stringent timing budget. Although this core targeted WLAN application, it only can achieve *12.625 Mbps* decoding throughput which is far behind the *54 Mpbs* requirement of WLAN.

[43] introduced a reconfigurable Viterbi architecture with an adaptive Viterbi algorithm which supports constraint length from 4 to 14. However, the adaptive Viterbi algorithm requires additional hardware for each state and incurs extra computational cost. As reported in [43], the decoding throughput can only achieve hundreds of *Kbps*.

Some published ASIC designs [62] [63] [64] [65] of Viterbi decoder can achieve higher throughput than the proposed fabric by means of bit-level pipeline [62] [63] or higher radix ACS architecture [64] [65]. However, they are constrained to the Viterbi decoder with fixed constraint length and code rate. Compared with these ASIC implementations, the proposed architecture not only can support different constraint length and code rate, but also can provide the demanded throughput for different standards.

The proposed architecture requires more area for its paralleled butterfly units and memory blocks. However, considering its salient throughput performance, it worth to pay this price. In addition, for each specific application, the unused blocks can be automatically powered off, and the power overhead incurred by the parallel architecture has been dramatically reduced.

3.3 Conclusion

In this chapter, the implementation of a domain specific reconfigurable Viterbi decoder fabric is addressed. This fabric can be configured to a constraint length from 3 to 9 and code rate 1/2 or 1/3 Viterbi decoder. It employs a fully parallel ACSU and sliding window schemes to achieve high throughput. In addition, for each specific application, the unused modules such as butterfly units in the ACSU and sub-memory blocks in SMU can be automatically turned off in order to save power consumption.

After the comparison, it can be seen that the proposed fabric can reduce 79.3% power consumption with only 2.2% area overhead as compared with the architecture without power saving strategy. However, this fabric pays 3.4 and 2.7 times more power consumption and area as the price of its flexibility.

Furthermore, our design has compared with other domain specific reconfigurable designs. It can be seen that our design can provide the best throughput performance. Because of all these characteristics, our core is very competitive for use in portable wireless communication scenario.

However, the drawback of this design methodology is obvious. Firstly, since the reconfigurable interconnection network is manually implemented, the design and

verification cost of this fabric is huge. In addition, the fabric is not easily extended to cope with additional functions. For example, if this fabric has to support constraint length 10, the interconnection network needs to be redesigned, which leads to another timing and labor cost task. In order to overcome these drawbacks, in the following chapter, we will present a novel reconfigurable architecture and its associated CAD design flow, which can automatically generate, configure and map Viterbi decoders with different constraint length and code rate.

Chapter 4:

Reconfigurable Viterbi Decoder Architecture II

In the previous chapter, a domain specific reconfigurable Viterbi decoder architecture was described, which is denoted by *reconfigurable architecture I* in this chapter. *Reconfigurable architecture I* provided a good compromise in terms of flexibility, power consumption and throughput. As compared with dedicated hardware, the area and power overheads of *reconfigurable architecture I* are not far away and it can be efficiently and effectively integrated into a high performance wireless communication system. However, this type of reconfigurable architecture is manually implemented, where the significant design/verification cost and difficulties of redesigning are contrary to the basic principles of the underlying reconfigurable architecture, which requires quick time-to-market and low design costs. If a domain specific reconfigurable architecture can be automatically generated for the targeted computation domain, the inefficient design cost can be reduced. This potential requirement pushes us to look for a computer-aided design (CAD) flow to accelerate the generation and verification of a domain specific reconfigurable architecture.

In this chapter, we address a design methodology to automatically generate a reconfigurable architecture which performs a given range of computations. By means of the proposed CAD tools, the design and verification time of a domain specific reconfigurable architecture can be lessened. The rest of the contents are divided in five sections. The architecture of the proposed domain specific reconfigurable fabric is addressed in Section 4.1. Section 4.2 introduces the associated automatic design flow for the proposed domain specific reconfigurable architecture. Section 4.3 explains the implementation procedure for the Viterbi decoder on the proposed architecture with associated design flow. Section 4.4 discusses the experimental results and compares them with other platforms. Section 4.5 draws the conclusions.

4.1 Domain specific reconfigurable architecture overview

The proposed architecture is depicted in Figure 4.1. It can be seen that the reconfigurable fabric on a SoC platform is composed of heterogeneous processing units (PU) and a 2-D programmable interconnection mesh. Each type of PUs is in charge of one specific operation. These domain specific reconfigurable fabrics can be provided as synthesizable soft IP (Intellectual Property) cores to allow the customization of all aspects of the array at design time. This also permits an easy integration of these fabrics into the SoC platform and verification at every stage of the design flow.

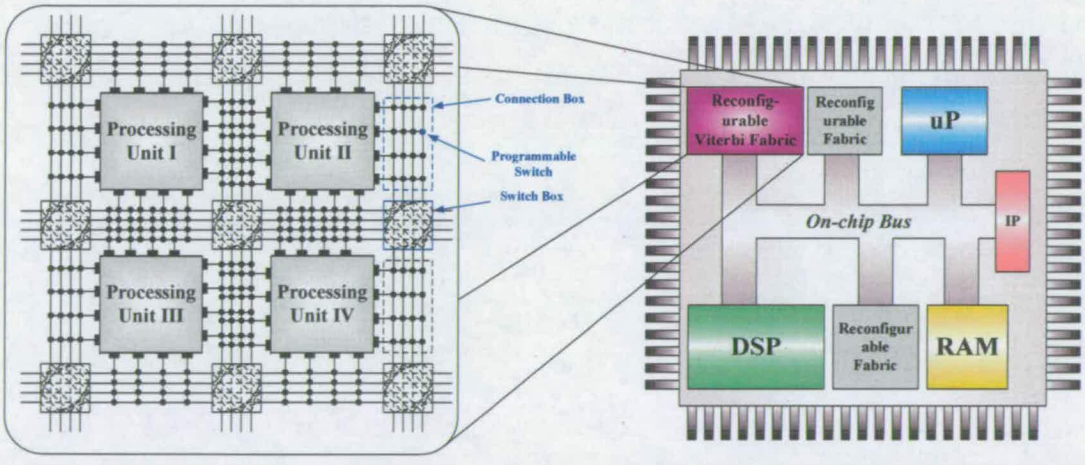


Figure 4.1 Heterogeneous coarse-grain domain specific reconfigurable architecture

4.1.1 Processing unit (PU)

PUs are the main computational elements in the proposed array architecture and they provide the basic operations which are required by the targeted application domain. Individual PUs can talk to each other via the interconnection mesh to form the required data path. As a heterogeneous architecture, the proposed reconfigurable fabric consists of PUs with different granularity, flexibility and functionality. In addition, each PU only performs an individual computation for a set of applications, which potentially reduces the area and power consumption as compared with a homogenous architecture.

Coarse-grained architecture with path widths greater than one bit can dramatically reduce the delay, area, power consumption and configuration time as compared with fine-grained architecture. However, some DSP algorithms do require 1-bit operation, such as the Viterbi decoder, thus a mixed PU which has either 1-bit or multiple bits port is introduced for the proposed domain specific reconfigurable architecture.

A diagram of 2-input-2-output PU is illustrated in Figure 4.2. It can be seen from Figure 4.2 that each I/O of the PU has been organized in four directions: north, south, east and west, where PU can import and export data from four directions. During the configuration period, CAD software can select one of four directions for each I/O based on the number of PUs and capacity of the interconnections to optimize the power and timing performance.

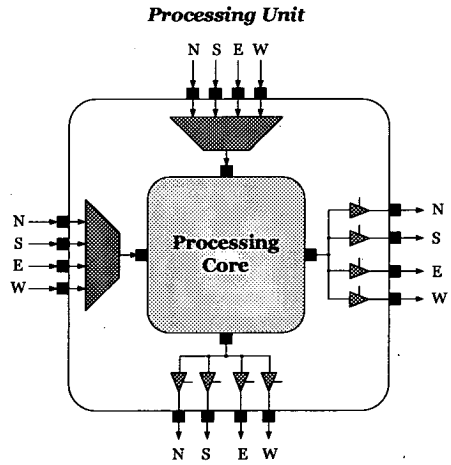


Figure 4.2 Architecture of processing unit

In order to accelerate the design time for different domain specific applications, a pool of PUs can be built. The PU library is capable to provide PUs with different operation, timing,

area and power consumption, which makes it possible to generate architecture for the required functionality and performance in a short time.

4.1.2 Interconnection mesh

Interconnection mesh inside the architecture is utilized to connect PUs together and form large operational circuits. As depicted in Figure 4.1, a two-dimensional (2-D) island-style interconnection mesh is employed by the proposed domain specific reconfigurable architecture.

The conventional fine-grained FPGA [33] [34] exploits 1-bit tracks to form an interconnection mesh. Coarse-grained mesh [49] [51] [52] [55] [57] [58], on the other hand, bonds several 1-bit tracks together in a group to save the hardware cost. Although the coarse-grained mesh is more efficient when groups of signals have the same source and destination, they are not sufficient to route individual signals. In the proposed interconnection mesh, both multiple bits and one bit tracks are employed, which can reduce the number of switches and configuration bits compared to fine-grain 1-bit track FPGAs. In addition, in contrast to coarse-grained architecture, this kind of mixture of fine-grained and coarse-grained mesh can achieve an efficient routing solution for individual signals.

In conventional FPGA, the configuration switches are implemented as pass-transistors, which provide the connection between two tracks [66]. For the sake of generating a synthesizable architecture, the configuration switches are implemented by multiplexers and tri-state buffers in the proposed architecture. However, they would increase area and power consumption when compared with pass-transistors.

There are two types of interconnection devices in the proposed architecture: connection boxes (C-boxes) which are used to transfer data between PUs and data tracks, and switch boxes (S-boxes) which are used to connect data tracks together.

4.1.2.1 C-box architecture

The architecture of C-box is presented in Figure 4.3. The role of C-box is to connect PUs

with data tracks inside the interconnection mesh. During the run time, since no more than one data track from the interconnection mesh will be connected to the input pin of PU, a multiplexer is utilized for the C-boxes of each input pin to reduce the number of the configuration bits because the bits are encoded rather than one bit for each switch.

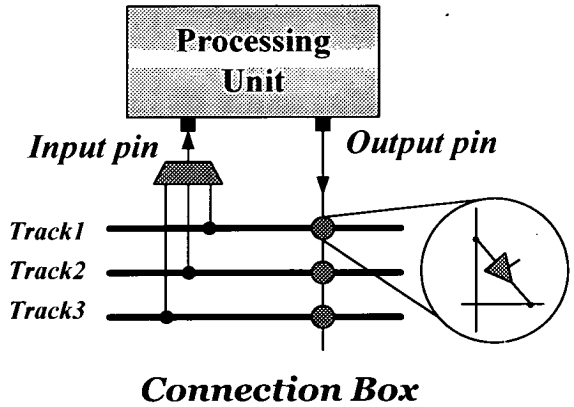


Figure 4.3 Architecture of C-box

Otherwise, the output signal from a PU may be desirable to have the ability to connect more than one data track. To achieve this requirement, a tri-buffer based switcher must be employed for the output pin.

4.1.2.2 S-box architecture

S-box is used to connect the data tracks together. Unlike C-box, each channel inside the switch box has to be bi-directional, in order to increase the routability of the whole architecture. The topology of the S-box can be varied, such as *Disjoint* type [67], *Universal* type [68] and *Wilton* type [69]. However, the investigation of topologies of the S-box for a domain-specific reconfigurable architecture is out of the scope of this thesis. The topology used by the proposed architecture is the *Disjoint* type, which is widely used by Xilinx FPGA families.

For a synthesizable S-box, different fabrication technologies would affect the performance of the whole array architecture in terms of area, power consumption, delay and routability. The tradeoff among different fabrication technologies has been elaborated in [70]. A 6W tri-buffer based S-box is recommended in [70], depicted in Figure 4.4, because of its best tradeoff among all crucial parameters.

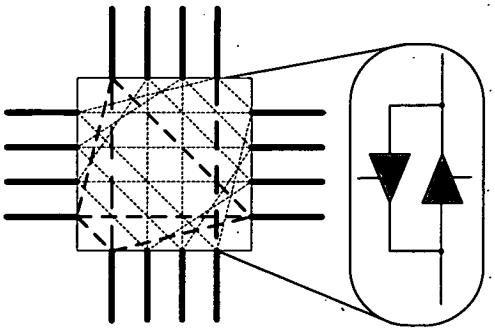


Figure 4.4 Architecture of switch box

4.2 Design tool flow

In order to accelerate the time-to-market, software tool flow has been developed to generate, program and verify the proposed domain specific reconfiguration architecture. The whole software design flow is shown in Figure 4.5, which is divided into three stages, array generation, array programming and array verification. Initially, the prototype [71] of this software design flow is developed by the author's colleague, Dr. Sami Khawam. The author did further modifications on the original prototype to smooth the interface between different tools and support complex test cases, such as the Viterbi decoder.

4.2.1 Array generation

Depending on the requirements of the design, such as application domain, flexibility, area and timing constraints, designers have to identify what kind of PUs are required. Basically, PUs are defined at RTL level by Verilog HDL.

The Array generation tool will read and analyze the Verilog HDL codes for each PU, and with respect to preceding constraints, generate the whole array architecture. Except for the Verilog HDL codes of PUs, the following parameters also have to be provided to the array generation tool. The resulting array is produced as synthesizable RTL description.

- Number of rows and columns
- Position of each type of PUs
- Number of bit-wide and word-wide tracks

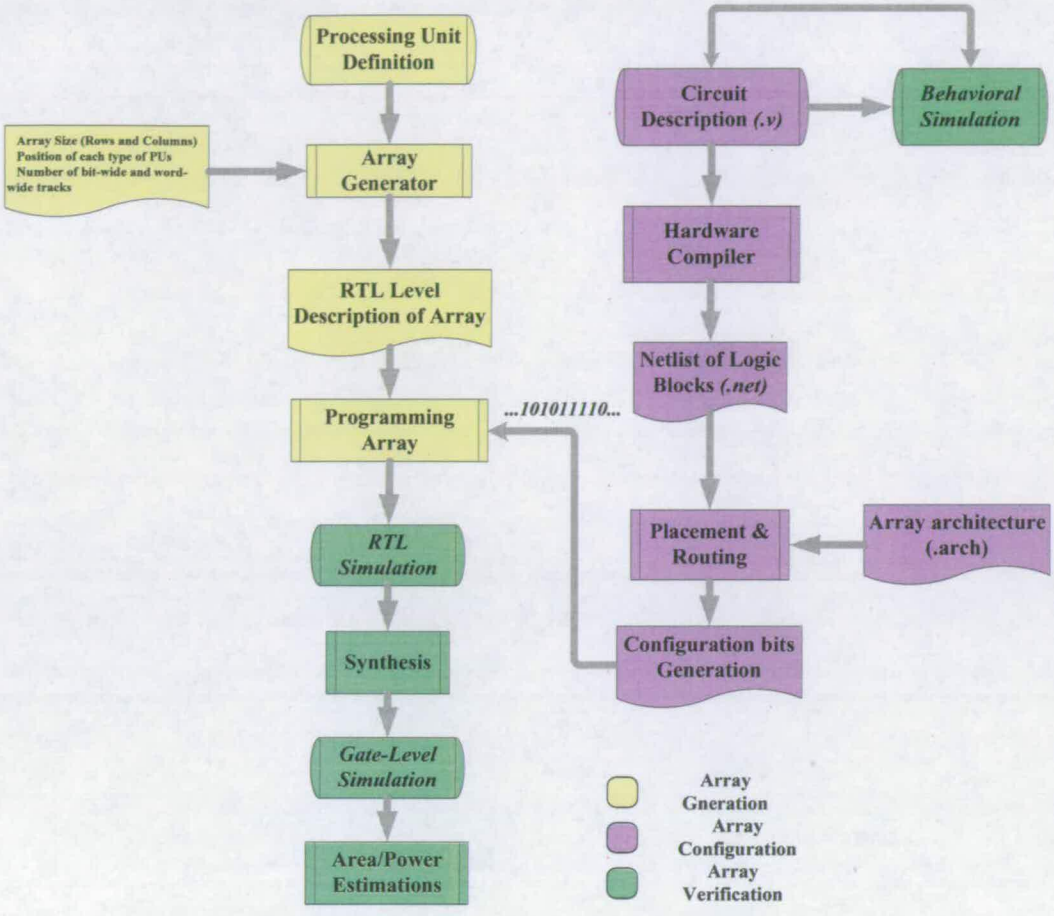


Figure 4.5 Design flow for domain specific reconfigurable architecture

4.2.2 Array programming

The array programming flow is divided into four sub-steps. It starts from the circuit description, such as Verilog. This file is provided by the designer, which described the interconnections of the PUs. Secondly, the hardware compiler can transfer the circuit description into the associated netlist file. The compiler is developed based on the academic synthesis tool *vl2mv* [72], where its output netlist format has been modified in order to be consistent with the place and route tool. The academic FPGA place and route tool, Versatile Place and Route (*VPR*) [73], was evolved to automatically place and route applications described by netlist files on the proposed heterogeneous and coarse-grain array architecture. After circuits mapping, the modified *VPR* also can output the configuration bits-stream which is used to test and verify the function on the array architecture.

4.2.3 Array verification

The array verification can be achieved in three levels: behavior level, RTL level and gate level. Before mapping to array architecture, the circuit description can be the first test in the behavior level to verify and debug the functionality. The RTL level and gate level verification are similar with ASIC design flow. The configuration bits generated by modified VPR can be loaded into the test benches for both RTL level and gate level simulation. Furthermore, based on gate level netlist, power and area estimation of the whole array can be done.

It should be noted that the verification is achieved by using the existing ASIC tools, unlike FPGA devices where new tools need to be utilized. Another advantage of the proposed design flow is that every stage can be integrated into the whole SoC design flow for further accurate simulation and verification.

4.3 Viterbi decoder implementation on the proposed array architecture

The section presents how to develop and map domain specific reconfigurable Viterbi decoders on the proposed array architecture with associated proposed CAD design flow. A full parallel Viterbi decoder with sliding window algorithm is considered as the test bench, thus we can do a deep comparison with previous architecture.

4.3.1 Processing unit

The processing core selection for a Viterbi decoder domain can follow the estimation in Chapter 3. Considering the architecture presented in Figure 3.1, four PUs are developed for the Viterbi decoder domain.

4.3.1.1 Processing core for BMU

Equations for BM computations have been tabulated in Table 3.1. It can be seen that the basic components of BM computations are adders and inverters. Therefore, addition and inversion operations can be extracted to form into an individual processing core (PC) for branch metric computation which can cover branch metric computations for different code rate.

The PC for BMU is illustrated in Figure 4.6. Two adders with an inverter are involved in the BMU PC. One BMU PC can calculate the summation or inversion between two inputs. For example, in order to calculate four branch metrics for code rate 1/2 Viterbi decoder, three BMU PCs have to be employed. If assuming X and Y denote two received signals, the first BMU PC outputs X and –X. Then, the second and third PCs compute the branch metrics, X+Y, X-Y, –X+Y and –X–Y. Each output pin of BMU PC contains a register which can be bypassed or not. This allows BMU PUs to form combinatorial or pipelined circuits to calculate different code rate BMs during one clock cycle.

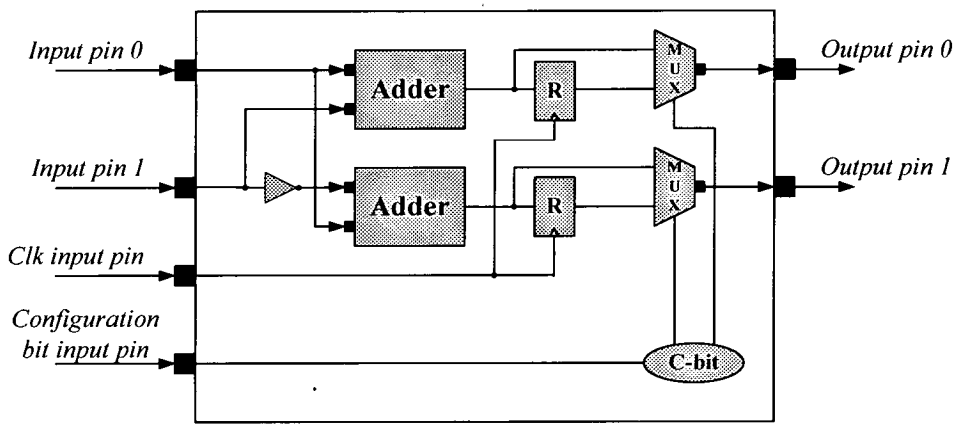


Figure 4.6 Branch metric processing core

4.3.1.2 Processing core for ACSU

As described in Chapter 3, the best architecture of ACSU for a high throughput Viterbi decoder is the parallel architecture with modulo arithmetic normalization. Thus in the proposed array architecture, the butterfly unit with modulo arithmetic normalization is selected as a PC for ACSU, which is shown in Figure 4.7. Since each butterfly would output two word-wise path metrics and two bit-wise survivor bits, mixed type C-boxes with one-bit

track and 8-bit tracks will be provided to ACSU PUs.

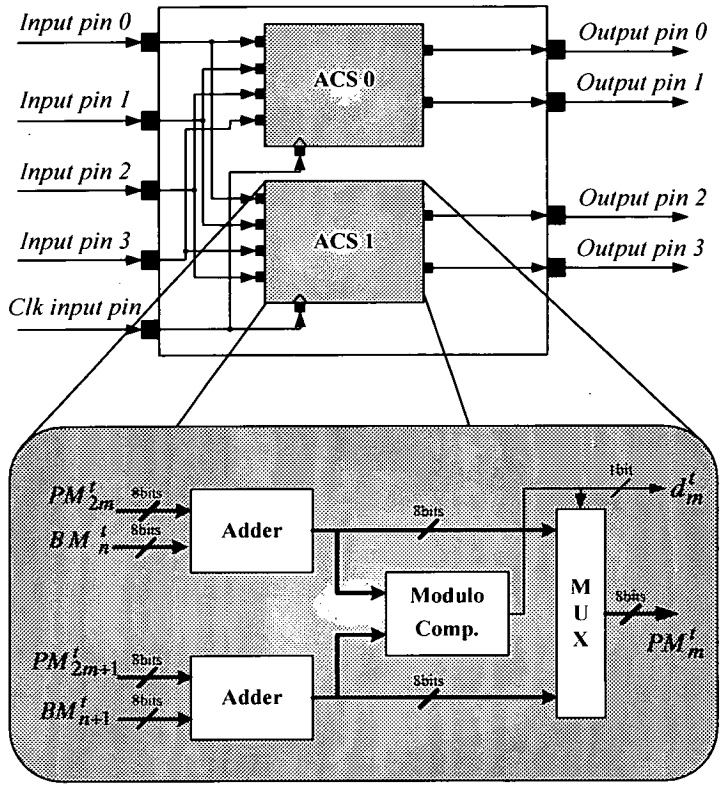


Figure 4.7 ACSU processing core

4.3.1.3 Processing core for RAM

A sliding window based Viterbi decoder requires four individual RAM banks to store the survivor bits generated by ACSU. Suitable RAM PUs have to be developed to efficiently map the sliding window based Viterbi decoder on the array architecture. Since the inputs of RAM PUs are connected with one-bit tracks, illustrated in Figure 4.8, there is a dilemma to select the appropriate word length for RAM PC. On one hand, the large word length of RAM PCs will affect the routability of the whole array architecture, because more tracks have to be involved in the interconnection mesh. On the other, small word length will increase the area of the whole array architecture, because more RAM PUs with associated C-boxes and S-boxes have to be integrated into the array architecture. The tradeoff between number of tracks and number of RAM PUs has to been investigated.

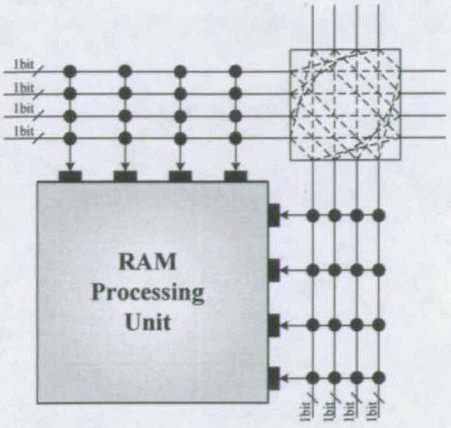


Figure 4.8 RAM processing unit

Constraint length 7 and code rate 1/3 Viterbi decoder is selected as a test bench to investigate the tradeoffs of different word lengths RAM PCs with respect to required number of tracks and number of RAM PUs .

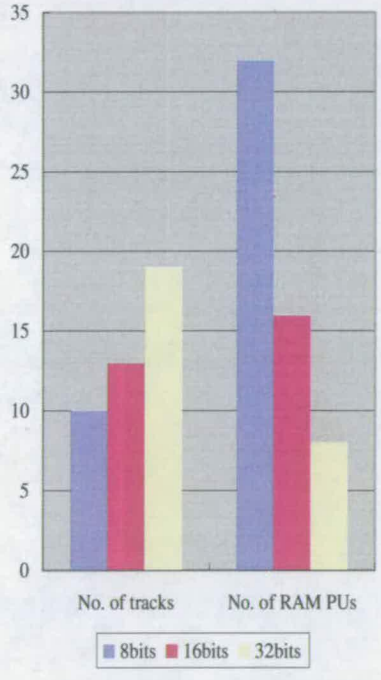


Figure 4.9 Tradeoff between No. of tracks and No. of RAM PUs

It can be seen from Figure 4.9 that, with the augment of RAM PC's word length, the required number of tracks is also increased. On the other hand, increasing the word length of PC, the required number of RAM PUs is decreased. Regarding preceding analysis, a memory block with word length 16bits is employed to a RAM PC to balance the number of

tracks and area of the whole array architecture. The number of words of RAM PC is equal to 5K to fulfill the traceback requirement and the RAM PC is built up by latches.

4.3.1.4 Processing core for traceback and LIFO

With the knowledge from last chapter, it can be seen that the blocks, such as traceback logic and LIFOs, only take a small portion of the whole Viterbi decoder (5% in terms of power consumption and 2% in terms of area). In addition, only one-bit operations are involved in these blocks. Bearing these two reasons in mind, it is not worth employing individual PU for each of these blocks. Thus, only one PU for traceback logic block and LIFOs is introduced to the array architecture and this PU has the same architecture as its counterparts in *reconfigurable architecture I*.

4.3.2 PUs arrangement for the Viterbi decoder domain

The arrangement of PUs in the domain specific reconfigurable Viterbi decoder is illustrated in Figure 4.10. The arrangement is manually achieved according to the tool flow shown in Figure 4.10. As can be seen, four different types of PUs, BMU PU, ACSU PU, RAM PU and Traceback & LIFO PU are involved in the array architecture and the same type of PU is placed in the same column to keep the uniform of the array architecture. The interconnection mesh consists of the C-boxes and S-boxes, described in previous sections, which support both one-bit track and eight-bit track. And totally 13 eight-bit tracks and 13 one-bit tracks are used for the Viterbi decoder application domain.

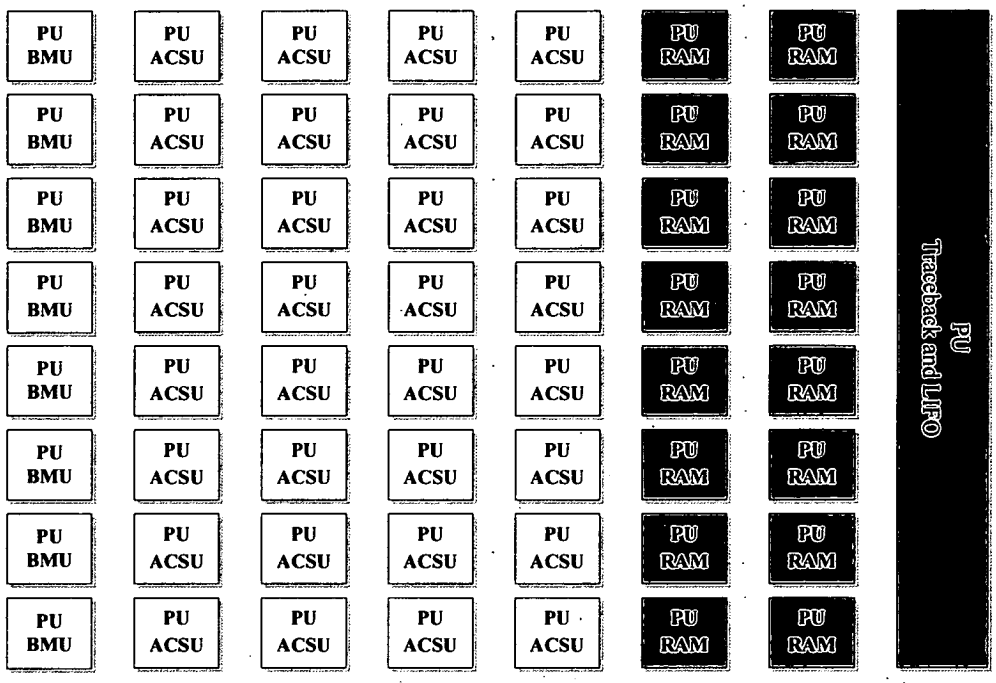


Figure 4.10 Arrangement of PUs

4.4 Performance and comparisons

Six Viterbi decoders, constrain length from 3 to 7, code rate 1/2 and 1/3 have been mapped on the proposed reconfigurable array architecture with associated software design flow. The Viterbi decoders with the same architecture are also mapped on Xilinx Virtex-E (xcv1000e-8) FPGA architecture. Comparison results on four different architectures: ASIC, *reconfigurable architecture I*, proposed array architecture and FPGA are detailed in Table 4.1. Area and power figures of the proposed array architecture are analyzed by *Synopsys Design Compile* and *Design Power*, respectively. The features of FPGA are obtained by Xilinx ISE 7.1 tool kits and Xilinx Xpower. All platforms are targeted on 0.18μm UMC CMOS technology and run at 1.8V and 10MHz.

Table 4.1 Power consumption (mW) comparison

	ASIC	Architecture I	Architecture II	FPGA
K3R1/2	0.39	2.75	4.89	9.11
K3R1/3	0.45	2.91	7.07	11.73
K5R1/2	1.13	4.09	9.93	27.46
K5R1/3	1.46	4.40	15.79	42.58
K7R1/2	4.72	9.14	30.76	150.74
K7R1/3	4.99	10.38	42.65	252.35

Table 4.2 Area (μm^2) comparison

	ASIC	Architecture I	Architecture II	FPGA
K3R1/2	48,817	232,396	970,011	2,872,422
K3R1/3	49,903	237,442	1,176,860	3,014,270
K5R1/2	226,651	534,664	1,886,550	8,714,787
K5R1/3	235,794	539,711	2,367,587	9,610,202
K7R1/2	838,758	1,743,738	6,164,657	31,694,163
K7R1/3	842,584	1,748,785	6,525,821	32,013,321

4.4.1 Area comparison

The normalized area comparison is presented in Figure 4.11, where we can easily study the area gap among different platforms. In order to give an in-depth analysis of the proposed reconfigurable core, rather than block memories, configurable logic block (CLB) based distributed memories are exploited in the FPGA implementations, thus the proposed coarse-grained architecture can be directly compared with fine-grained FPGA architecture. The detailed area occupied by Xilinx Virtex-E FPGA is based on the estimation of the area of a LUT plus a register. According to [74], a CLB in Virtex-E FPGA occupied 35462 μm^2, and a CLB is coomposed of four LUTs in Virtex-E family FPGA.

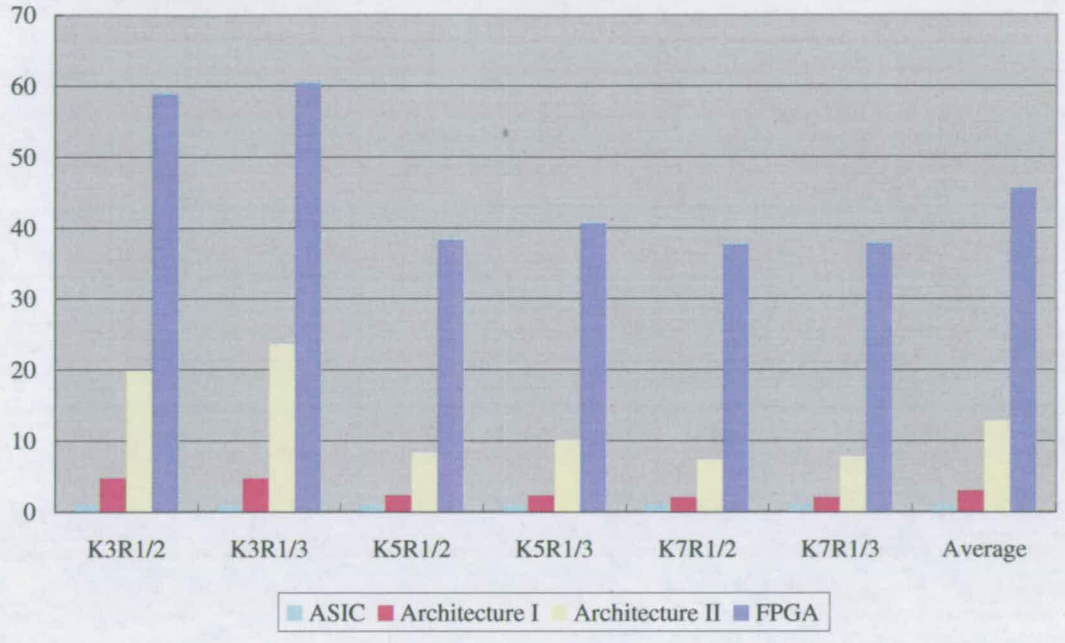


Figure 4.11 Area Comparison of Viterbi decoders on four platforms (ASIC = 1)

It can be seen that the proposed heterogeneous and coarse-grained architecture only cost 28% of the area of fine-grained FPGA averagely. As compared with the *reconfigurable architecture I*, heterogeneous and coarse-grained architecture are around 4.1 times bigger, but it takes an impressive advantage over the *reconfigurable architecture I* in terms of design and testing costs. It also can be seen that the area of heterogeneous and coarse-grained architecture is 12.8 times bigger compared with ASIC and this area overhead is the price must be paid for the flexibility and programmability.

4.4.2 Power consumption comparison

During the power consumption measurement, we only considered the summation of *signals power* and *logic power* [75] of FPGA, since they measure the power consumed by LUTs and routing network in the FPGA. From the normalized power comparison, shown in Figure 4.12, it can be seen that the proposed array architecture consumes 66.1% less power than the Virtex-E, averagely. This is mainly due to heterogeneous processing units and a mixture of coarse-grained and fine-grained interconnection mesh being used in our proposed array architecture. Similarly with *reconfigurable architecture I*, the data triggers of unused PUs,

C-boxes and S-boxes would be kept to zero, thus only leakage power is consumed in the unused blocks. From Figure 4.12, it also can be seen that the power consumption of the proposed heterogeneous and coarse-grained architecture is 2.6 times higher than *reconfigurable architecture I*. This is mainly incurred by the extra switches in C-boxes and S-boxes. In contrast to ASIC, proposed array architecture paid 10.5 times power consumption as the price of its excellent flexibility.

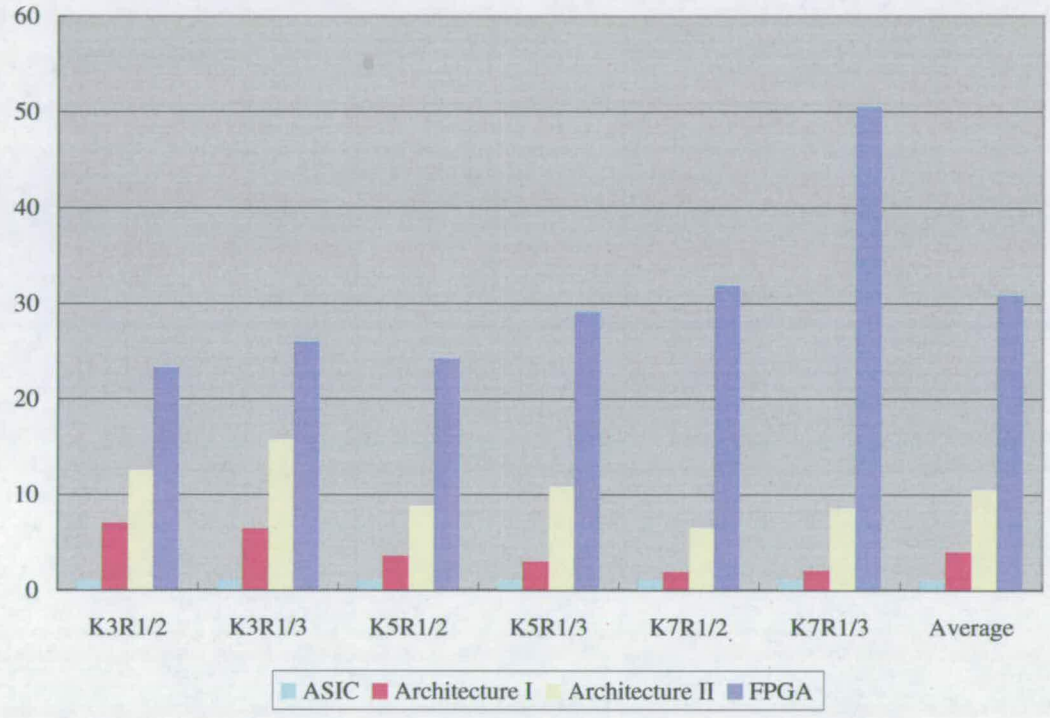


Figure 4.12 Area Comparison of Viterbi decoders on four platforms

4.4.3 Overhead measurement

As compared with ASIC and *reconfigurable architecture I*, the proposed array architecture pays addition area and power penalties for its superior flexibility and programmability. Since C-boxes and S-boxes are the main additional programmable components in the proposed architecture, in the following section, their contributions to the area and power overheads are estimated.

4.4.3.1 Area overhead

Figure 4.13 shows the area distribution inside the array architecture. It can be seen that PUs only occupy 14.5% of the total area averagely. However, C-boxes and S-boxes take a big portion of the area, 53.2% and 32.3%, respectively. It also can be seen that C-boxes require more area than S-boxes, since each port of PU connects to data tracks in four different directions via four C-boxes and the required switches in the C-boxes are obviously more than those in the S-boxes.

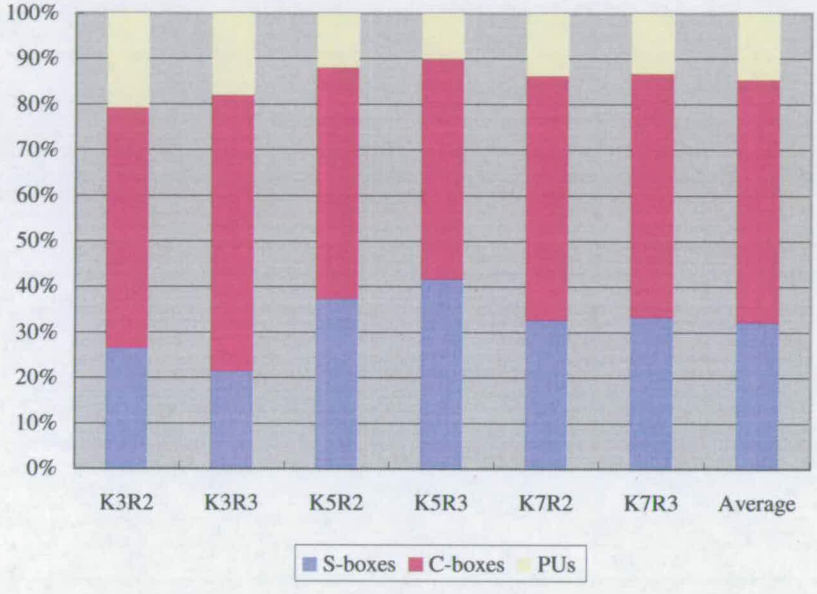


Figure 4.13 Area distribution of the array architecture

4.4.3.2 Power overhead

The power overheads of the proposed array architecture are illustrated in Figure 4.14. In the proposed architecture, the unused blocks can be powered off, thus only leakage power consumptions are exhibited in these blocks. It can be seen from Figure 4.14 that, PUs occupy 20.5% of total power consumption. However, interconnection meshes, C-boxes and S-boxes, consume 51.2% and 28.3% of total power, respectively. The power overhead of the proposed architecture is mainly produced by the switching activities in the interconnection mesh.

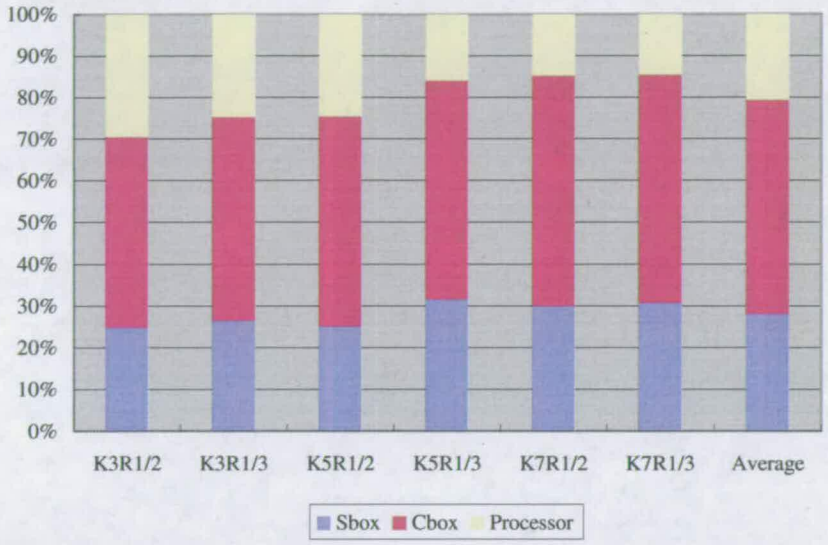


Figure 4.14 Power distribution of the array architecture

The future work would focus on the optimization of interconnection mesh to reduce the area and power overheads. In the proposed architecture, a switcher is implemented by tri-buffers which have high drive strength to guarantee that coarse-grain data tracks can be driven. However, the tri-buffer incurs bigger area and power consumption compared with the pass transistor which is exploited as switcher in general FPGA. In [76], a mixed interconnection mesh with tri-buffers and pass transistors is investigated in order to find the best trade-off for the routing network in an FPGA. In future, this kind of mixed interconnection mesh also can be employed in the proposed array architecture to lessen the area and power overheads.

The aim of the proposed architecture and its associated CAD tools is to reduce the design and verification time of a domain specific reconfigurable fabric. It can be seen that the reconfigurable array architecture can be automatically generated and programmed by the CAD tools, where the user only needs to define the function for each PU. Compared with the *reconfigurable architecture I*, the proposed architecture can achieve significant design and verification time reduction. It also can be seen that the verification is achieved by using the existing ASIC tools, unlike FPGA devices where new tools need to be utilized. With the proposed design flow, every stage of the reconfigurable fabric design can be integrated into the whole SoC design flow for further accurate simulation and verification. Assuming a designer without any experience on hardware implementation, with the author's experience, it is fair to say that implementing the same algorithm on ASIC requiring two months, one month for the *reconfigurable architecture I*, probably three weeks on FPGA, and one week

on the proposed architecture with the proposed software design flow. Significant reduction of design time is obtained by the proposed architecture, and the time-to-market also is improved as compared with ASICs and FPGAs.

4.5 Conclusion

We have presented a novel domain specific reconfigurable architecture and design methodology based on a synthesizable heterogeneous coarse-grained array and 2-D mixed interconnection mesh, which can be provided as a soft-IP core for integration into a SoC platform. The fabric has demonstrated high flexibility as well as good power performance, hence, making it satisfy the requirement for future portable devices. By using software design methodology, this fabric can be automatically generated and programmed to implement the different versions of domain specific applications. Several Viterbi decoders with different constraint lengths and code rates have been implemented on the proposed architecture by the developed software tools. After comparing with commercial FPGAs, the author demonstrated that power consumption is reduced considerably by 66.1%, and only cost 28% area of fine-grained FPGA.

However, there are still some limitations blocking this proposed architecture to be prevailing. During the proposed design flow, the designers must be familiar with hardware design language and carefully partition the required applications to look for the suitable PUs. Thus, the time-to-market time could be still hard to bear. In the following chapter, a superior dynamic reconfigurable architecture, which can be programmed through a high-level language, such as *C*, will be introduced. In addition, an efficient Viterbi decoder design on this novel architecture also will be described.

Chapter 5:

Implementation of the Viterbi Decoder on the Reconfigurable Instruction Cells Array Platform

In Chapter 4, a domain specific reconfigurable architecture with its associated CAD design flow has been described. The proposed architecture employing heterogeneous coarse-grained PUs surrounded by hybrid 2-D interconnection mesh provides an excellent compromise between FPGA and ASIC in terms of flexibility, area and power consumption. However, this proposed architecture has to be implemented by HDL which is similar to the implementation of FPGA. This would require designers to have a deep understanding of the targeted application domain and efficiently partition the application into several smaller function modules, which increases the design time and the time-to-market is postponed.

If a reconfigurable architecture can be programmed by high-level language, such as C and C++, the barriers of implementation can be eliminated and the time-to-market can be further improved. In this chapter, the reconfigurable instruction cell array (RICA) [77] platform which is a dynamic reconfigurable architecture programmed by ANSI C, will be introduced. In addition, Viterbi decoders have been implemented on RICA to test and verify the

performance of RICA. Furthermore, several advanced optimization approaches for the Viterbi decoder on RICA are proposed in this chapter to boost its performance.

This chapter is organized as follows. Section 5.1 will briefly review the RICA architecture and tool flow. Implementation of the Viterbi decoder on a general RICA architecture and its performance are given in Section 5.2 and Section 5.3, respectively. The detailed description of the advanced implementation and optimization of Viterbi decoder on RICA is presented in Section 5.4. The performance of an advanced RICA is presented in Section 5.5. The comparison with other dynamic reconfigurable architectures is presented in Section 5.6 followed by conclusions in Section 5.7.

5.1 RICA architecture and tool flow

RICA is a dynamic reconfigurable computing architecture. The basic concept of RICA is shown in Figure 5.1, where it can be seen that RICA has a straightforward and processor-like design-flow. The hardware-modules inside RICA can execute assembly-like instructions, which are named instruction cells (ICs). ICs in RICA are heterogeneous and each cell is limited to a small number of operations, such as *ADD*, *CONST*, *MUL*, *LOGIC*, *SHIFT* and *REG*. Except for the basic instruction cells, RICA also permits tailoring to a specific application domain by introducing user-defined custom instruction cells to improve the performance.

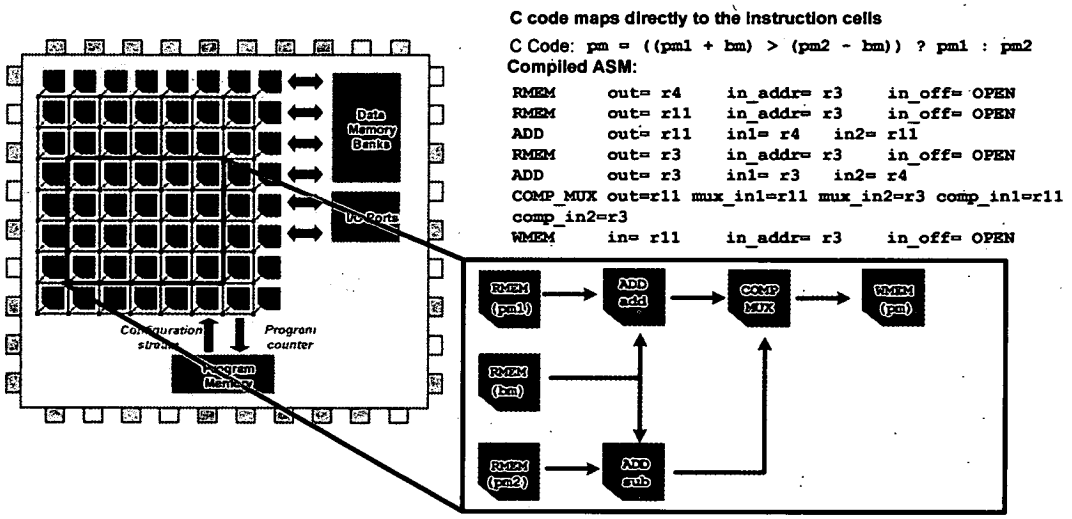


Figure 5.1 Architecture and tool flow of RICA

There are several compilation stages that need to be performed for RICA in order to take the high-level C code and transform it into various output formats for debugging purposes, software emulation or to create the final binary to be loaded in the hardware. The default design methodology for the RICA platform involves three main steps, which are briefly described as follows. More details of RICA tools can be found in [77] [78] [79].

5.1.1 Compiler

This step is performed by a modified GNU C Compiler (GCC), which compiles C/C++ code and transforms it into assembly format. In addition, GCC is adjusted to take into account some limitations of the target RICA architecture.

5.1.2 Scheduler

The scheduling process involves taking the assembly code; constructing the control and dataflow graphs; and breaking the code into steps which maximize resource usage and throughput.

5.1.3 Backend simulator

The simulator is a SystemC cycle-accurate model of RICA architecture. It allows design exploration in terms of core configuration and verification of algorithm code generated to execute on RICA; assists in debugging; and provides detailed timing, activity and core utilization figures.

5.2 Implementation of the Viterbi decoder on RICA

As mentioned in previous chapters, there are three major modules in a Viterbi decoder: BMU, ACSU and SMU. This section will elaborate the implementation issues for each module on the RICA platform.

5.2.1 Branch metric computation

As described in Chapter 3, branch metric computation is a straightforward addition and subtraction process. However, since one half of branch metrics is the invertion of the other half, only two branch metrics must be calculated. Branch metric computation can be denoted as a simple C loop, where loop unrolling can be performed to yield a more balanced use of RICA resources. An example of loop unrolling for BMU is illustrated in Figure 5.2.

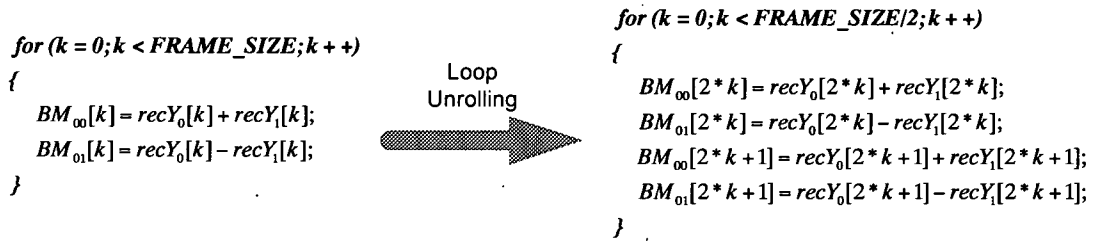


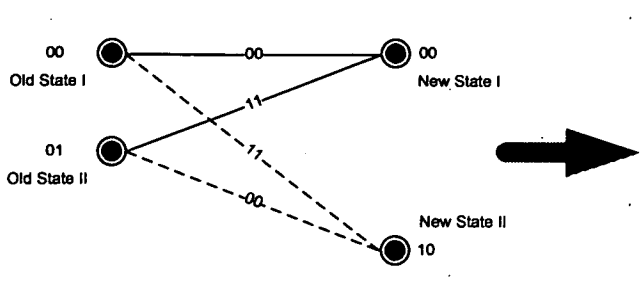
Figure 5.2 Loop unrolling for branch metric computation

The unrolling factor depends on the available hardware resources of the RICA architecture. It can be seen from Figure 5.2 that a BMU loop with unrolling factor 2 requires twice more *WMEM* and *RMEM* cells to fit the whole loop into a single step as compared with the original loop.

5.2.2 Path Metric Computation

Since all the path metrics (2^{K-1}) must be updated at each trellis stage, ACSU consumes most of the calculation time. According to previous chapters, two new states are connected by the same sources and branch metrics on the connection path are complementary with each other. For example, if one branch metric is BM_{00}, the other must be BM_{11}. Thus, two ACS operations for two different path metrics can be paired in a butterfly-like structure to simplify the metric update procedure.

It can be seen from Figure 5.3 that the simplifications of C implementation are exhibited in two ways. Firstly, only one branch metric is needed for each butterfly. It is alternately added and subtracted in each ACS operation. Secondly, the old metric values are the same for both new ones where address manipulation can be minimized.




```
PM1 = old_buf[0] + BM00;
PM2 = old_buf[1] - BM00;
PM3 = old_buf[0] - BM00;
PM4 = old_buf[1] + BM00;
New_buf[0] = (PM1 > PM2) ? PM1 : PM2;
New_buf[2] = (PM3 > PM4) ? PM3 : PM4;
Survivor_bit0 = (PM1 > PM2) ? 0 : 1;
Survivor_bit1 = (PM3 > PM4) ? 0 : 1;
```

Figure 5.3 Butterfly structure with its C implementation

In addition, during the path metric updating procedure, two memory buffers have to be adopted, one for the old path metrics and the other for the new path metrics. Each buffer contains 2^{K-1} bits, each bit reserved by a path metric. At the end of the metric updating, the pointer of two buffers must be swapped, so that the recently updated metrics will become the old path metrics for the next iteration.

Similar to previous reconfigurable architectures, there is a trade-off between throughput and the number of butterfly units which are processed in parallel. The more butterfly units are in parallel, the faster throughput can be achieved. However, more instruction cells are required. The effect of this trade-off will be shown in the following section.

Besides path metrics, two survivor bits which indicate survivor paths outputted by each butterfly unit have to be stored in the memory. Since RICA is a 32bits processor, one word of memory (32bits) is shared by 16 butterflies. However, in order to pack 32 survivor bits into a word, extra *SHIFT* and *LOGIC* cells are needed. Eventually, the survivor bits table in the memory is shown in Table 5.1.

The ***Bit Number*** indicates the bit line of a survivor bit in a 32bits word. It is worth noting that survivor bits are saved in a reversed manner. Survivor bit 0 is in MSB and survivor bit 31 is in LSB. The ***Word Number*** indicates the word line of a survivor bit. If the constraint length K is less than 7, the ***Word Number*** can be ignored, since one word is sufficient to memorize all survivor bits.

Table 5.1 Survivor bits table from different constraint length

		Bit Number in survivor bits table							
		31	**30**	**29**	**28**	**27**	**26**	**25**	**24**
Word Number	**0**	0	1	2	3	4	5	6	7
	……								
	2^{K-6}-1	2^{K-1}-32	2^{K-1}-31	2^{K-1}-30	2^{K-1}-29	2^{K-1}-28	2^{K-1}-27	2^{K-1}-26	2^{K-1}-25

		Bit Number in survivor bits table							
		23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Word Number	**0**	8	9	10	11	12	13	14	15
	……								
	2^{K-6}-1	2^{K-1}-24	2^{K-1}-23	2^{K-1}-22	2^{K-1}-21	2^{K-1}-20	2^{K-1}-19	2^{K-1}-18	2^{K-1}-17

		Bit Number in survivor bits table							
		15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Word Number	**0**	16	17	18	19	20	21	22	23
	……								
	2^{K-6}-1	2^{K-1}-16	2^{K-1}-15	2^{K-1}-14	2^{K-1}-13	2^{K-1}-12	2^{K-1}-11	2^{K-1}-10	2^{K-1}-9

		Bit Number in survivor bits table							
		7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Word Number	**0**	24	25	26	27	28	29	30	31
	……								
	2^{K-6}-1	2^{K-1}-8	2^{K-1}-7	2^{K-1}-6	2^{K-1}-5	2^{K-1}-4	2^{K-1}-3	2^{K-1}-2	2^{K-1}-1

5.2.3 Traceback Operation

Traceback operation is the major logic function in SMU and requires much less computational cost than the path metric updating procedure. The main operation of the traceback operation is to extract the relevant survivor bit from the memory to rebuild the trellis structure. Based on the survivor bits table presented in Table 5.1, extracting the correct bit from the survivor bit table is divided into two steps: searching for ***Word Number*** and ***Bit Number.***

For a constraint length K Viterbi decoder, the ***Word Number*** is determined by masking off the bits between four LSBs and MSB, which is defined by Equation 5.1.

$$Word\ Number = (trace\_state \gg 5) \& (2^{K-6} - 1) \tag{5.1}$$

The ***Word Number*** will be added by the pointer of the current table address to indicate a memory with 32bits survivor bits. For the constrain length K < 7, this part of the computation can be eliminated, since only one word is required by each trellis stage.

Because the survivor bits in the memory are in an inverse manner, for example, the survivor bit of state 0 is saved in MSB, the bit address corresponds to *31 - State Number*. Based on the properties of Table 5.1, the calculation for the ***Bit Number*** is accomplished by Equation 5.2:

$$Bit\ Number = 31 - (\text{trace\_state} \ \&\ 0\text{x}1\text{F}) \tag{5.2}$$

After finding the correct survivor bit, the old *trace_state* value is shifted one bit left and the survivor bit can push into the LSB to form the new *trace_state*, which is denoted by Equation 5.3.

$$trace\_state_{new} = (trace\_state_{old} \ll 1 \,|\, sur\_bit) \tag{5.3}$$

The new *trace_state* is exploited to look for the survivor bit in the next iteration.

5.3 Performance of Viterbi decoder on RICA

To analyze the performance of RICA, three versions of a constraint length 7 and code rate 1/2 Viterbi decoder have been implemented on the RICA platform. In *case I*, only one butterfly unit is performed in an ACSU loop. Otherwise, two and four butterfly units are executed at a time in *case II* and *case III*, respectively. The performances of these three cases are presented in Table 5.2. *Case I* requires least number of cells, but it provides the slowest speed. Although *case III* achieves the highest throughput, it also requires more instruction cells to perform more butterfly units in a step. This is expected due to more butterfly units being processed simultaneously, the faster throughput is achieved.

Table 5.2 Performance of Viterbi decoder on general RICA

	Throughput (Mbps)	**No. of Steps**	**No. of cells**
Case I	0.89	1943	102
Case II	1.39	1047	126
Case III	2.56	599	182

The utilities of each individual cell are presented in Figure 5.4. It can be seen that *ADD*, *CONST* and *REG* are the most popular instruction cells, because they are widely used by loop control, address manipulation and branch operations of C program.

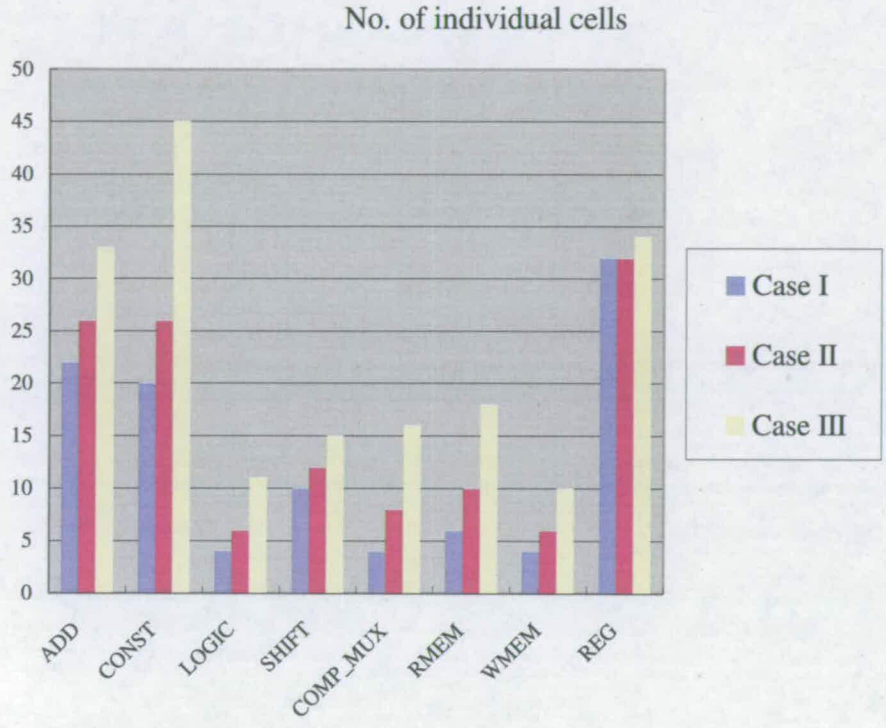


Figure 5.4 Requirement of individual cells for Viterbi decoder

5.4 Advanced implementation and optimization for the Viterbi decoder

At present, wireless communication standards expect to provide fast system throughput. For instance, 3G systems can achieve 2Mbps, DVB-T/H provides round 20Mpbs transmission speed, and WLAN claims higher throughput, up to 54Mbsp. As a reconfigurable architecture exploited in the portable communication scenario, RICA must fulfill the throughput requirement of these high-speed communication standards. In this section, several novel approaches to accelerate Viterbi decoding on RICA architecture are elaborated.

5.4.1 SIMD based RICA

Due to the fact that the required word width for digital communication operations is often much narrower than 32bits, employing 32bits representing smaller data size not only wastes the hardware resource, but also decreases the performance and efficiency of the RICA processor. In terms of the Viterbi decoder, as discussed in Chapter 2, four or five bits representing the soft input symbols can achieve a similar performance as the floating point representation in the case of AWGN channel and BPSK [22].

In addition, due to the inherent symmetry properties of digital communication algorithms, several variables can be aligned together to be processed at the same time. According to these two phenomena, a *Single Instruction Multiple Data* (SIMD) based RICA is proposed, where each instruction cell of the RICA can be split into several narrower word length operations in order to enhance the productivity of the RICA processor. For instance, a 32bits *ADD* cell can be configured to four independent 8bits *ADD* cells or two independent 16bits *ADD* cells, thus with the similar die area, *ADD* operations can be augmented 2 or 4 times. This thesis is the first time to propose using the SIMD technique in the RICA architecture in order to achieve data level parallelism. As compared with the high performance processor designs described in [80] [81], where SIMD based ALUs are integrated into the processor, RICA looks at cell based architecture, which exhibits a lower level application.

SIMD based RICA is very efficient at accelerating the processing of ACSU. It can be seen from Figure 5.5 that the branch metrics and path metrics for two individual butterfly operations are packed together, thus two butterfly operations can be executed at a time. As compared with general RICA architecture, its benefits are obvious. Firstly, half-required instruction cells are eliminated and the utilization of instruction cells is improved. In addition, due to the data packing, the transmission delay incurred by the interconnection mesh is reduced and the timing characteristic of RICA is further enhanced.

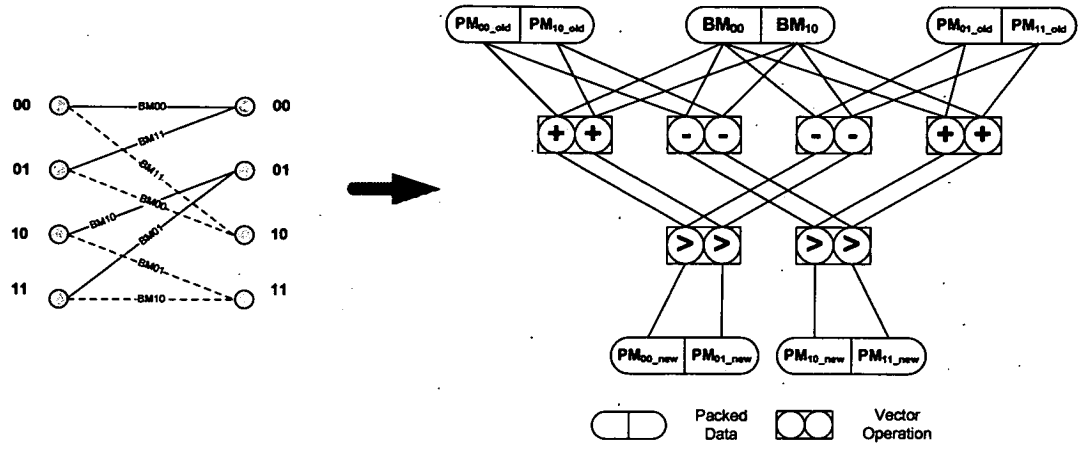


Figure 5.5 Butterfly units on a SIMD based RICA architecture

5.4.2 Custom Function Cells

Since the RICA processor has a very flexible interface with the instruction cells, the custom instruction cells can be easily integrated into the architecture to improve the performance for a specific application domain. In order to tailor the RICA to the high throughput required scenario, several individual custom instruction cells are proposed to accelerate the Viterbi decoding procedure.

5.4.2.1 Modulo comparison

In a Viterbi decoder, path metrics are accumulated till the end of frame. The value of each path metric will increase without limits and incur an arithmetic overflow if it is not periodically normalized. As illustrated in Chapter 2, normalization can be achieved by subtracting the minimum path metric in each time cycle. However, the major drawback of this normalization scheme is the huge clock delay caused by looking for the minimum path metric and subtracting it from all other path metrics. The other approach is modulo arithmetic which is an efficient method of solving system crash caused by path metric overflow. The ideal of modulo normalization is not to avoid overflow, but instead to accommodate overflow into a way that is does not affect the correctness of the results. This is achieved by using two's complement arithmetic with a modified comparator. But this modulo comparator is not suitable for implementation by software. However, because of the flexible interface of RICA architecture, modulo comparison function, which is depicted in

Figure 5.6, is extended to a COMP_MUX cell. By changing the configuration bits, two 15bits modulo comparators can be realized on the extended COMP_MUX cell. In addition, it can be seen from Figure 5.6 that the MSB of each 16bits vector is reserved for the survivor bit which is the output flag from the modulo comparators. Packing survivor bits with path metrics can further reduce function cells and data transmission as compared with general RICA.

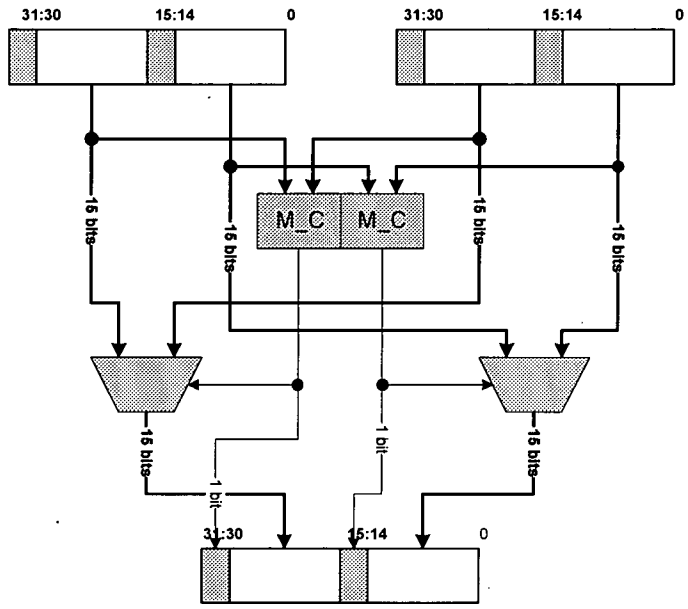


Figure 5.6 Modulo Comparison function for COMP_MUX cell

The performance comparison between a software implementation and a custom cell implementation of a butterfly unit is tabulated in Table 5.3. The comparison between a general *COMP_MUX* cell and a custom *COMP_MUX* cell is also presented in Table 5.3. It can be seen that, as compared with software implementation, the custom cell-based butterfly unit can reduce latency by 79.0% and only requires one instruction cell to perform a butterfly unit. In contrast to general a *COMP_MUX* cell, a custom *COMP_MUX* cell has to pay an extra 16.67% time delay, 44.24% area and 54.68% power consumption as the price of throughput improvement.

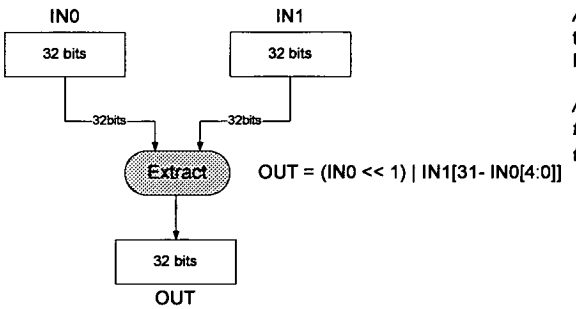
Table 5.3 Performance comparisons for a butterfly unit

	Latency (ns)	Area (um^2)	Power (uW)	No. of cells
Software Implementation	7.0	N.A.	N.A.	2 SHIFT 4 LOGIC 2 COMP_MUX
General COMP_MUX	1.26	2151.73	33.67	N.A
Custom COMP_MUX	1.47	3103.68	52.08	1 COMP_MUX

5.4.2.2 Traceback instruction cell

As described in the previous section, traceback logic ought to look for the ***Word Number*** and ***Bit Number*** to extract the survivor bit from the survivor bit table. However, this procedure results in a long combinatorial circuit consisting of logic, shift and bit level manipulation. In order to speed up the traceback operation, a specific instruction cell is introduced to handle the loosest computation in the hardware. The proposed architecture of this custom cell for traceback is depicted in Figure 5.7.

It can be seen that the output of the traceback instruction cell is equivalent to shifting the *trace_state* variable one bit up and pack the survivor bit at the LSB. ***IN0*** and ***IN1*** represent *trace_state* and a 32bits survivor bit table, respectively. Since the survivor bits in the survivor bit table are in an inverse manner, the ***Bit Number*** corresponds to ***31 – IN0[4:0].*** Moreover, the traceback instruction cell only adopts four LSBs of ***IN0***, thus there is no need to mask off the upper bits via additional LOGIC cells.




```
//Initialization
trace_state = 0;
P = survivor memory address for last state

//traceback operation
for(i=size_frame;i>0;i--)
{
    survivor_vector  = *(P + (trace_state >> 5) & 0x1);
     trace_state = extract (trace_state, survivor_vector);
    output_bit[i] = trace_state & 0x1;
    P = P-2;
}
```

Figure 5.7 Custom cell for traceback operation

The comparison of the traceback operation with and without custom cell is presented in Table 5.4. Compared with general RICA architecture, the latency of the traceback operation with the custom cell can achieve a 19.15% reduction. Moreover, a *SHIFT* cell, *ADD* cell and *LOGIC* cell can be preserved by the adoption of a custom cell.

Table 5.4 Comparison of Traceback with/without custom cell

	Latency (ns)	**No. of cells**
With custom cell	14.35	2 SHIFT 4 ADD 3 LOGIC 1 RMEM 1 WMEM 9 CONST
Without custom cell	17.75	4 SHIFT 5 ADD 4 LOGIC 1 RMEM 1 WMEM 9 CONST

5.4.3 Software pipelining

When a loop is implemented on the RICA architecture, it would be best to be fitted in a single operation step to reduce the jump inside a loop. Taking butterfly operation as an example, the data flow graph of a loop executing one butterfly operation is shown in Figure 5.8. It can be seen that this step consists of two parts, address generator which generates the read and write address for the *RMEM* and *WMEM* cells and computation logic which calculates the butterfly operation. The longest-path in this loop is also highlighted in Figure 5.8. According to a 32bits RICA timing model, the longest-path delay of this loop is 24.1 ns.

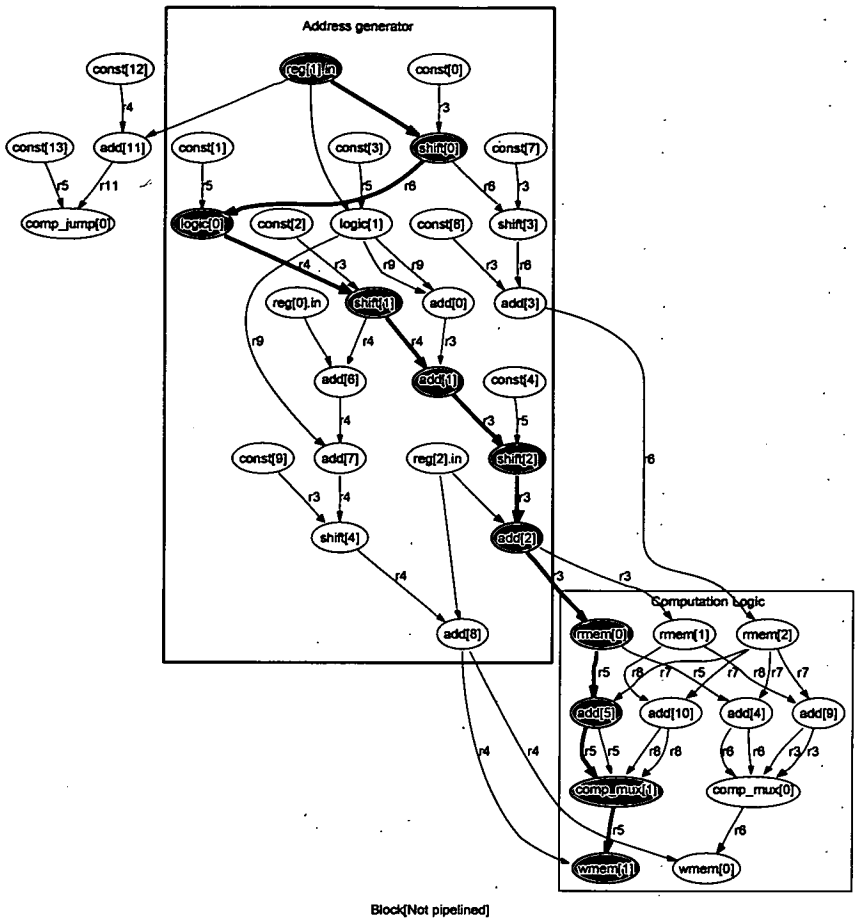


Figure 5.8 Unpipelined Butterfly operation

Software pipelining is an approach to reorganize loops by means of inserting registers into the longest-path. Thus, a combinatorial circuit is split into several portions running at the same time. Since the address generator and computational logic consume roughly the same time, the registers can be inserted between these two parts to reduce the latency by half. The data path of the butterfly operation after software pipelining is depicted in Figure 5.9. Since these two parts are executed in parallel, the total latency after software pipelining can be decreased to 13.9 ns, where the pipelined loop can save 42.32% time delay of the original loop, but additional *REG* cells need to be paid as the price.

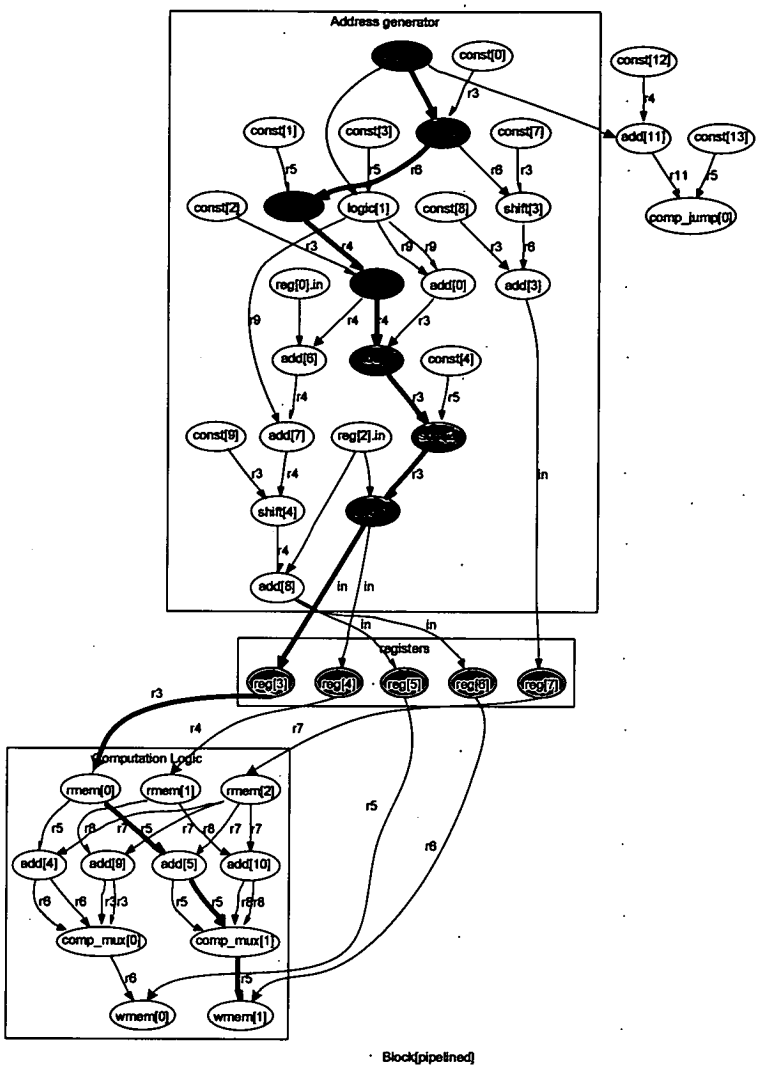


Figure 5.9 Pipelined butterfly operation

5.5 Performance of the Viterbi decoder on an advanced RICA platform

According to preceding advanced implementation approaches, the constraint length 7 and code rate 1/2 Viterbi decoder has been redesigned and optimized on the RICA platform. The performances of the optimized implementations on three test cases have been tabulated in Table 5.5. It can be noted that, in terms of throughput, SIMD based RICA with custom instruction cells can achieve 91%, 74.1% and 59.8% performance gains as compared with general RICA. In terms of number of execution steps, advanced RICA architecture can reduce by 40.2%, 31.8% and 27.7% compared with general RICA. In other words, less

configuration latency and power consumption can be accomplished by the advanced RICA architecture. It also can be seen that the number of required instruction cells is roughly as the same as general RICA architecture.

Table 5.5 Performance of Viterbi decoder on advanced RICA

	Throughput (Mbps)	**Improve**	**No. of steps**	**Improve**	**No. of cells**	**Improve**
Case I optimized	1.70	*91%*	1162	*40.2%*	100	*2%*
Case II optimized	2.42	*74.1%*	714	*31.8%*	131	*-4%*
Case III optimized	4.09	*59.8%*	433	*27.7%*	185	*-1.6%*

The cost of each individual instruction cell is illustrated in Figure 5.10. Apart from *LOGIC* and *SHIFT* cells, other instruction cells are consumed less than those of the general RICA architecture. The reason is that additional *LOGIC* and *SHIFT* cells are employed in packing and shuffling data for SIMD based RICA architecture.

In order to analyze the maximum throughput for the Viterbi decoder on RICA architecture, a full parallel Viterbi decoder, where 32 butterfly units are executed in one single operation step, has been implemented on SIMD based RICA architecture with specific instruction cells. It can be seen from Table 5.6 that, if a full parallel ACSU is implemented on the RICA, 20.9 Mpbs throughput can be obtained. After three-stage software pipelining, a further 2.7 times throughput gain can be accomplished. With a throughput up to 56.4 Mpbs, the advanced RICA architecture can be exploited by current wireless communication devices.

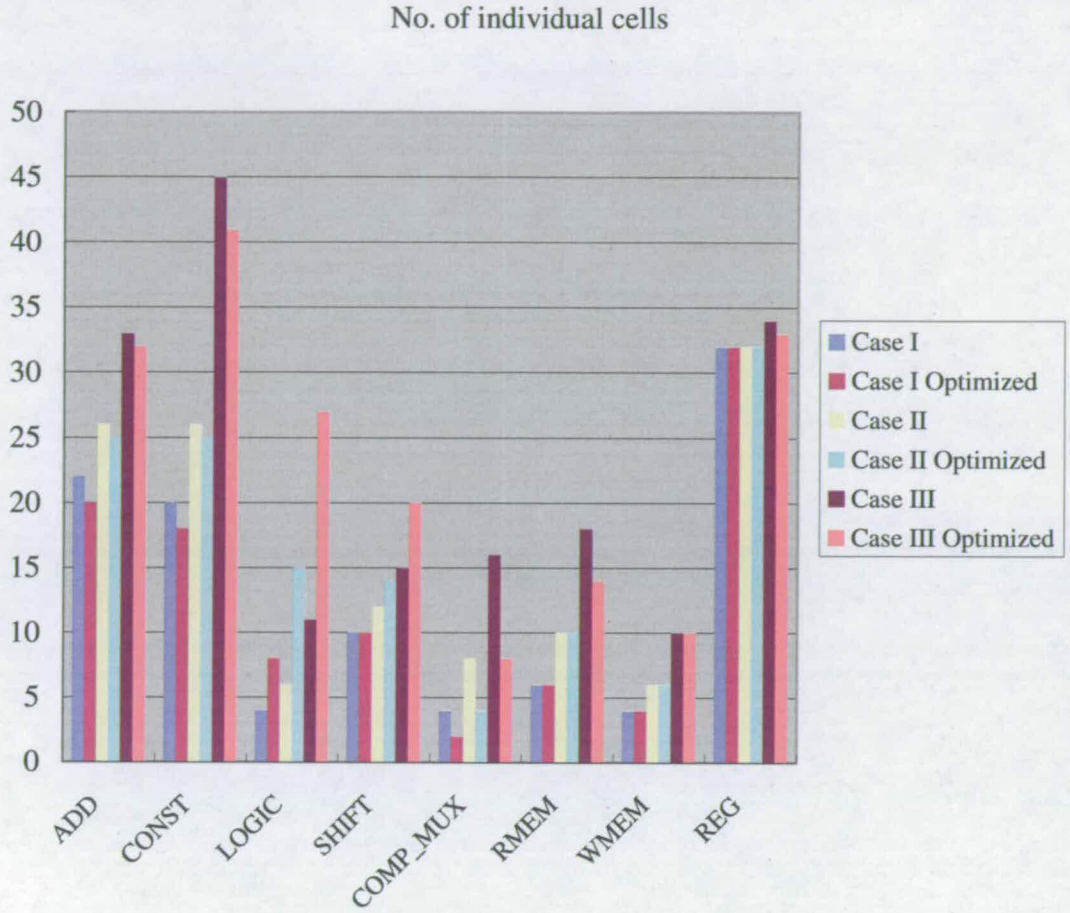


Figure 5.10 Requirements of individual cells for Viterbi decoder

Table 5.6 Ultimate Viterbi decoder on advanced RICA

	Throughput	**Throughput (software pipelining)**	**No. of Steps**	**No. of cells**
Full parallel Viterbi decoder	20.9 Mbps	56.4 Mpbs	120	349

5.6 Conclusion

This chapter has described the implementation of the Viterbi decoder on a novel dynamic reconfigurable architecture, RICA. In addition, several advanced optimization approaches have been proposed to accelerate the throughput of the Viterbi decoding process on the

RICA platform. With the proposed approaches, the throughput of Viterbi decoder can be improved up to 91% as compared with general RICA architecture. Ultimately, 56.4 Mbps Viterbi decoding throughput can be achieved by employing a full parallel ACSU architecture with software pipelining optimization scheme. In contrast to other reconfigurable architectures which are programmed by high-level language, the Viterbi decoder on the RICA platform exhibits the best throughput performance.

Since RICA can be easily programmed, it can dramatically reduce the time-to-market for portal products. In the following chapter, a more complicated channel decoding approach, double binary circular Turbo codes, will be described. In addition, several efficient decoding schemes will be proposed and an efficient decoder design for double binary circular Turbo codes will be demonstrated on RICA architecture.

Chapter 6:

M-binary Circular Turbo Decoder and its Application

Turbo codes [82] were first introduced to the coding community in 1993 to provide higher reliability data transmission at very low signal-to-noise ratio (SNR) as compared with convolutional codes. For the sake of its outstanding performance and competitive implementation complexity, Turbo codes have been specified in numerous communication standards, such as satellite communication, third-generation communication system, DVB-RCS, WiMAX, etc.

In the recent decade, many works have focused on improving Turbo encoder/decoder algorithms and architectures to achieve better BER/FER performance, reduce computation complexity, increase throughput and minimize power consumption of the Turbo codes system. Amongst these works, M-binary circular Turbo codes [84] have stood out and its particular application, double-binary circular Turbo codes, has been standardized by industry applications, such as DVB-RSC and IEEE 802.16d.

However, the computational cost needed by Turbo decoding process is an order of magnitude greater than that of Viterbi decoding [83]. Moreover, M-binary circular Turbo decoding requires M times more computational cost than classical Turbo decoding. Thus, an efficient implementation for M-binary circular Turbo decoders is a big challenge for mobile terminal designers.

For these reasons, the implementation design flow for M-binary circular Turbo decoders is divided in two steps which are presented in Chapter 6 and Chapter 7 respectively. The first step will explore the implementation design space on algorithm level. The issues addressed in this step consist of the mathematical model for the M-binary circular Turbo decoder, the selection of decoding algorithms, their simplification and optimization, and parallel decoding approaches which can achieve high decoding throughput. On algorithm level, the M-binary circular Turbo decoder is investigated by a floating point representation. However, a fixed-point representation is mandatory for most hardware architectures, such as the RICA platform. Thus, step two investigates a suitable quantization scheme for M-binary circular Turbo codes, leading to a bit-true model. In addition, corresponding to this bit-true model, a high throughput double binary circular Turbo decoder on RICA architecture is demonstrated.

6.1 M-binary Circular Turbo encoder

6.1.1 M-binary recursive systematic convolutional encoder

Inside an M-binary circular Turbo encoder, each recursive systematic convolutional (RSC) encoder can manage a vector with m bits rather than 1 bit data at a time [84]. From Figure 6.1, it can be seen that $m+n$ bits codeword are generated by an M-binary RSC encoder at a time, where the first m bits are systematic bits which are a duplication of the input vector, and the following n bits are parity bits which are employed to recover data from transmission noises.

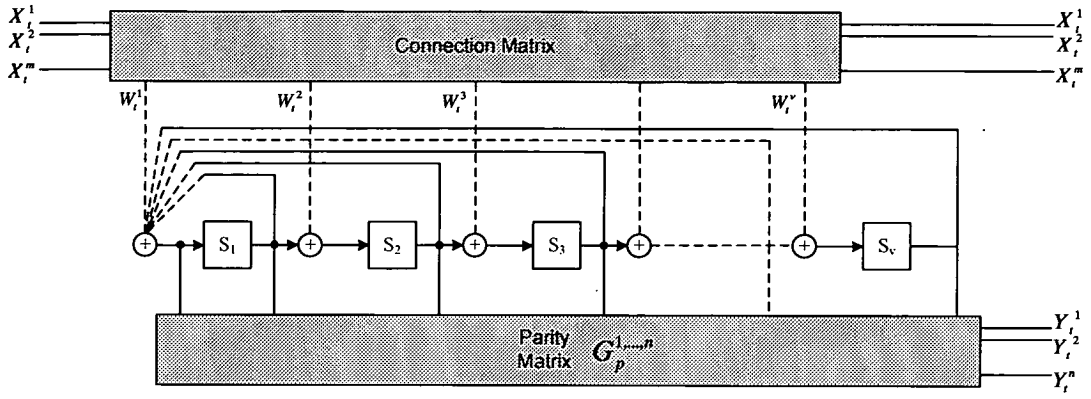


Figure 6.1 M-binary recursive systematic convolutional encoder

The M-binary Turbo codes can offer many advantages over classical single binary Turbo codes, and these advantages have already been elaborated in [85]:

- Increased minimum free distance by introducing 2-D permutation.
- Reduced sensitivity of puncturing as compared with the single-binary Turbo codes.
- Improved performance by reducing the correlation effects between two elementary decoders.
- Reduced decoder latency because *m* bits are processed as a symbol.

6.1.2 Circular Turbo encoder

In a classical Turbo coding system, several tail bits have to be padded to the end of a frame in order to force the trellis starting and ending at the all-zero state. However, transmission of extra tail bits would diminish the transmission bandwidth, especially for a frame with small frame size.

Circular (Tail-biting) encoder [86] is a technique which ensures that, at the end of the encoding process, the final state is constrained to be identical with the initial state, and the merged identical state is called *circular state*. The advantage of circular Turbo codes is obvious. It can reduce the code rate and increase the system transmission bandwidth. However, the computational complexities of both encoder and decoder are augmented.

In addition, the studies of algorithms/architectures for an efficient decoder implementation of M-binary circular Turbo codes are not sufficient in the current literature. These issues

motivate us to build a mathematical model for an M-binary circular decoder in order to investigate different decoding algorithms, their simplification and optimization.

6.2 M-binary Circular Turbo decoder

A typical Turbo decoder, shown in [82], consists of two soft-input soft-output (SISO) decoders, one operating on the actual order, the other on the interleaved order. The interleaver and de-interleaver are employed to reorder the sequence during the iterative process.

There are two prevalent SISO decoding algorithms which are used in Turbo decoder: the maximum-a-posterior (MAP) algorithm [87] and the soft-output Viterbi algorithm (SOVA) [88]. Since the SOVA is more complex than the MAP and its approximation algorithms [89], and also the SOVA algorithm incurs 0.5dB performance degradation as compared with the MAP and its approximation algorithms [87], this thesis only focuses on investigating the MAP algorithms for the M-binary circular Turbo decoder.

6.2.1 MAP algorithm for M-binary Turbo codes

6.2.1.1 Mathematic model for single MAP decoder

As depicted in Figure 6.1, the output from an RSC encoder is a $(m+n)\times N$ size matrix which can be represented by:

$$E_{out} = (E_1, E_2, E_3, ..., E_N) \tag{6.1}$$

where the codeword at time k is represented by $E_k = (X_k^1, X_k^2, X_k^3, ..., X_k^m, Y_k^1, Y_k^2, Y_k^3, ..., Y_k^n)^T$, m denotes the number of systematic bits, n denotes the number of parity bits and N is the frame size.

If binary phase-shift keying (BPSK) is assumed, the modulated codewords are presented as:

$$E' = (E_1', E_2', E_3', ..., E_N') \tag{6.2}$$

Each component $E_k' = (e_k^1, e_k^2, e_k^3, ..., e_k^{m+n})^T$ is a vector containing $m+n$ elements, where

$$\underset{a=1,\ldots,m+n}{e_k^a} = +1/-1.$$

During the transmission, these codewords are corrupted by channel noises, and the noised symbols received by decoder are:

$$D = (D_1, D_2, D_3, \ldots, D_N) \tag{6.3}$$

where $D_k = (d_k^1, d_k^2, d_k^3, \ldots, d_k^{m+n})^T$ is the received symbol at time *k*.

The posteriori probabilities of each possible codeword are indexed by:

$$P\left(x_k = \underset{i=1,\ldots,2^m}{X_i'} \mid D\right) \tag{6.4}$$

where D is received symbol sequence, and $\underset{i=1,\ldots,2^m}{X_i'}$ represent the 2^m different codewords. It can be seen from Equation 6.4 that at time *k*, there are 2^M posteriori possibilities for each codeword. According to the MAP algorithm, all these posteriori possibilities must be calculated and the data pair with the maximum posteriori possibility is selected as the decoder output.

According to *Bayes' rule,* posteriori probabilities can be approximate represented by its joint probabilities [90]:

$$P(x_k = \underset{i=1,\ldots,2^m}{X_i'} \mid D) \approx P(x_k = \underset{i=1,\ldots,2^m}{X_i'} \wedge D) \tag{6.5}$$

Joint probabilities $P(x_k = \underset{i=1,\ldots,2^m}{X_i'} \wedge D)$ can be classically partitioned [90] into three terms. Defining T_k^i is a set of transitions from previous trellis state $S_{k-1} = s'$ to current trellis state $S_k = s$ which are caused by transmitting M-binary vector x_k, the joint probabilities can be rewritten as:

$$\begin{aligned} P(x_k = \underset{i=\mu}{X_i'} \wedge D) &= \sum_{(s',s)\in T_k^\mu} P(D_{t>k} \mid s) \cdot P(\{D_k \wedge s\} \mid s') \cdot P(s' \wedge D_{t<k}) \\ &= \sum_{(s',s)\in T_k^\mu} \beta_{k+1}(s) \cdot \gamma_{k+1}(s',s) \cdot \alpha_k(s') \end{aligned} \tag{6.6}$$

$P(D_{t>k} \mid s)$ represents the probability that, given the trellis is in state *s* at time *k*, the future received vectors will be $D_{t>k}$. $P(s' \wedge D_{t<k})$ denotes the trellis is in state s' at time *k-1* and the received channel sequence up to this point is $D_{t<k}$. $P(\{D_k \wedge s\} \mid s')$ is the probability

that, given the trellis is in state s' at time *k-1*, it moves to s with received vector D_k. They are also nominated as backward metrics $\beta_k(s)$, forward metrics $\alpha_k(s)$ and branch metrics $\gamma_k(s',s)$, respectively.

Referring to *Bayes' rule* again, forward and backward metrics can be recursively deduced by [91]:

$$\begin{aligned}\alpha_k(s) &= \sum_{all\,s'} \gamma_k(s',s)\cdot\alpha_{k-1}(s') \\ \beta_{k-1}(s') &= \sum_{all\,s} \gamma_k(s',s)\cdot\beta_k(s)\end{aligned} \tag{6.7}$$

The main difference between these two recursive computations is that forward metric calculation processes from beginning to the end of the frame, otherwise backward metrics are calculated from the end to the beginning.

Moreover, if a memoryless Gaussian channel and BPSK modulation are assumed, branch metric $\gamma_k(s',s)$ can be denoted by: (The deduction of Equation 6.8 is presented in Appendix A)

$$\gamma_k(s',s) = \exp\left(\sum_{l=1}^{m+n} e_k^l \cdot d_k^l\right)\cdot P(x_k) \tag{6.8}$$

It can be seen from Equation 6.8 that a branch metric includes two items, the first item is an exponent of the correlation between received codeword and expected codeword, which indicates the difference between the received and expected codeword. The other is the a-priori probability of x_k, which indicates the statistical characteristic of the transmitted M-binary vector acquired before the current decoding process. In an iterative Turbo decoder, $P(x_k)$ will be transferred between two MAP decoders to improve the reliability of a Turbo decoder.

6.2.1.2 Mathematic Model for an iterative Turbo decoder

In an iterative decoder, the output from one MAP decoder will be fed to the other as priori probabilities of $x_k(P(x_k))$. During the iterative process, priori probabilities will be more and more reliable, resulting in a dramatic improvement on BER/FER performance of a Turbo decoder.

The exponent term of Equation 6.8 can be split into two items in terms of systematic and parity parts, which is presented as follows:

$$\begin{aligned}\gamma_k(s',s) &= \exp\left(\sum_{l=1}^{m+n} e_k^l \cdot d_k^l\right) \cdot P(x_k) \\ &= \exp\left(\sum_{l=1}^{m} e_k^l \cdot d_k^l\right) \cdot \exp\left(\sum_{l=m+1}^{m+n} e_k^l \cdot d_k^l\right) \cdot P(x_k) \\ &= \exp\left(\sum_{l=1}^{m} e_k^l \cdot d_k^l\right) \cdot \chi_k(s',s) \cdot P(x_k)\end{aligned} \tag{6.9}$$

Thus Equation 6.6 can be rewritten in another format:

$$\begin{aligned}P(x_k = X_i^{'} \wedge D) &= \sum_{(s',s)\in T_k^{\mu}} \beta_{k+1}(s) \cdot \gamma_{k+1}(s',s) \cdot \alpha_k(s') \\ &= \exp\left(\sum_{l=1}^{m} e_{k+1}^l \cdot d_{k+1}^l\right) \cdot P(x_{k+1}) \cdot \sum_{(s',s)\in T_k^{\mu}} \beta_{k+1}(s) \cdot \alpha_k(s') \cdot \chi_{k+1}(s',s)\end{aligned} \tag{6.10}$$

It can be seen that the last term of Equation 6.10 is independent upon the channel effect of current systematic bits and $P(x_k)$. Therefore, it was named as extrinsic probabilities, which is represented as:

$$P_{out}^{ex}\left(x_k = X_i^{'} \mid D\right) = \sum_{(s',s)\in T_k^{\mu}} \beta_{k+1}(s) \cdot \alpha_k(s') \cdot \chi_{k+1}(s',s) \tag{6.11}$$

In an iterative Turbo decoder, extrinsic information of x_k represents the probabilities that are obtained based on the received sequence and priori probabilities excluding the received systematic symbols and priori probability of x_k. In an iterative Turbo decoder, extrinsic probabilities from the current MAP decoder would be provided to the other MAP decoder as the priori probability for x_k [90].

6.2.2 Simplifying the MAP algorithm with the MAX* approximations

Although a MAP algorithm for the M-binary Turbo decoder has been deduced in Section 6.2.1, it is too complex to be implemented by hardware or software methods. The

approximations which can be employed to simplify the computation of a MAP algorithm have been addressed in [87] [92] [93] [94], in the case of the classical Turbo decoder. However, they also can be evolved for an M-binary Turbo decoder.

6.2.2.1 Log-domain MAP

Transferring the computation of MAP algorithm into a logarithmic domain can replace multiplication by addition and also eliminate exponent operations. In a logarithmic domain, these probabilities can be denoted by:

$$
\begin{cases}
L_k^{\alpha}(s) = \ln \alpha_k(s) = \ln\left(\sum_{all s'} \exp(L_k^{\gamma}(s',s) + L_{k-1}^{\alpha}(s')) \right) = \underset{all s'}{MAX^*}\left(L_k^{\gamma}(s',s) + L_{k-1}^{\alpha}(s') \right) \\
L_{k-1}^{\beta}(s) = \ln \beta_{k-1}(s') = \ln\left(\sum_{all s} \exp(L_k^{\gamma}(s',s) + L_k^{\beta}(s)) \right) = \underset{all s}{MAX^*}\left(L_k^{\gamma}(s',s) + L_k^{\beta}(s) \right) \\
L_k^{\gamma}(s';s) = \ln \gamma_k(s',s) = \sum_{l=1}^{m+n} e_k^l \cdot d_k^l + \ln P(x_k) \\
L\left(x_k = \underset{i=\mu}{X_i'} \mid D \right) = \ln\left(\sum_{(s',s)\in T_k^{\mu}} \exp\left(L_{k+1}^{\beta}(s) + L_{k+1}^{\gamma}(s',s) + L_k^{\alpha}(s') \right) \right) \\
\qquad = \underset{(s',s)\in T_k^{\mu}}{MAX^*}\left(L_{k+1}^{\beta}(s) + L_k^{\alpha}(s') + L_{k+1}^{\gamma}(s',s) \right) \\
L_{out}^{ex}\left(x_k = \underset{i=\mu}{X_i'} \mid D \right) = \ln\left(\sum_{(s',s)\in T_k^{\mu}} \exp\left(L_{k+1}^{\beta}(s) + L_k^{\alpha}(s') + L_{k+1}^{\chi}(s',s) \right) \right) \\
\qquad = \underset{(s',s)\in T_k^{\mu}}{MAX^*}\left(L_{k+1}^{\beta}(s) + L_k^{\alpha}(s') + L_{k+1}^{\chi}(s',s) \right)
\end{cases}
\tag{6.12}
$$

where $\underset{i}{MAX^*}(\lambda_i) = \ln\left(\sum_i e^{\lambda_i} \right)$

Since MAX^* is the major function for log-domain MAP algorithm and also it is not easy to be directly implemented, thus there are several approximation schemes focus on reducing the associated implementation complexity of the MAX^* function. In general, these are *Log-MAP* [87], *Constant-Log-MAP* [92], *Linear-Log-MAP* [93], *MAX-Log-MAP* [87] and *Enhanced MAX-Log-MAP* [94]. *Log-MAP, Constant-Log-MAP* and *Linear-Log- MAP* approximations aim to convert the MAX^* function to an addition of a *MAX* and a correction

function.

6.2.2.2 Log-MAP algorithm

In the *Log-MAP* algorithm, applying Jacobian logarithm, the MAX^* function can be expressed by:

$$\begin{aligned} MAX^*(x,y) &= \ln(e^x + e^y) \\ &= \max(x,y) + \ln(1 + e^{-|x-y|}) \\ &= \max(x,y) + f_c(|x-y|) \end{aligned} \tag{6.13}$$

which is an addition between the maximum of the function's two arguments and a nonlinear correction function. In practice, this correction function $f_c(|x-y|)$ might be implemented in a software method with floating-point functions, and might be implemented by a finite lookup table in hardware design [95].

6.2.2.3 Constant-log-MAP algorithm

In the *Constant-Log-MAP* algorithm, MAX^* is approximated to:

$$MAX^*(x,y) = \max(x,y) + \begin{cases} 0 & if\ |x-y| > T \\ C & if\ |x-y| \le T \end{cases} \tag{6.14}$$

Compared with the *Log-MAP* algorithm, the *Constant-Log-MAP* algorithm supersedes the correction function $f_c(|y-x|)$ by a value equal to 0 or a constant C which depends on the absolute difference between x and y. It offers an easier way to implement MAX^* function, but it will suffer more from the performance loss as compared with the *Log-MAP* algorithm.

6.2.2.4 Linear-log-MAP algorithm

The *Linear-Log-MAP* algorithm provides a tradeoff between the *Log-MAP* algorithm and the *Constant-Log-MAP*, which adopts the following linear approximation for a MAX^* function:

$$MAX^*(x,y) = \max(x,y) + \begin{cases} 0 & if\ |x-y| > T \\ a(|x-y| - T) & if\ |x-y| \le T \end{cases} \tag{6.15}$$

Rather than a nonlinear correction function in the *Log-MAP* algorithm, the *Linear-Log-MAP* algorithm exploiting a linear correction function not only reduces the computation

complexity, but also compensates for the performance loss. The comparison of three different correction functions is illustrated in Figure 6.2.

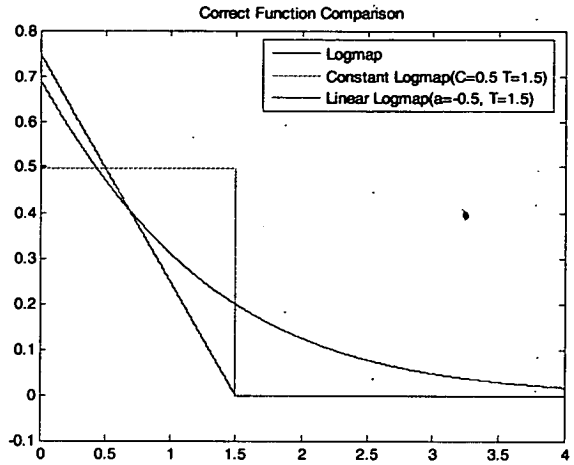


Figure 6.2 Correct functions of different approximation

In the classical binary Turbo MAP decoder, only two arguments need to be considered in MAX^* function. However, in the case of M-binary Turbo codes, each MAX^* function contains 2^m arguments. In addition, a MAX^* function with 2^m arguments can be recursively calculated by MAX^* functions with two arguments:

$$MAX^*\left(x_1, x_2, \cdots x_{2^m}\right) = MAX^*\left(MAX^*\left(x_1, x_2\right), \cdots, MAX^*\left(x_{2^m-1}, x_{2^m}\right)\right) \tag{6.16}$$

6.2.2.5 MAX-Log-MAP Algorithm

On the other hand, *MAX-Log-MAP* and *Enhanced MAX-Log-MAP* approximations eliminate the correction function and directly compute MAX^* by:

$$MAX^*\left(x_1, x_2, \cdots x_i\right) \approx MAX\left(x_1, x_2, \cdots x_i\right) \tag{6.17}$$

In the case of classical Turbo codes, the *MAX-Log-MAP* algorithm can reduce computational cost by half compared with other approximate MAP algorithms with correction functions, but it has to pay 0.5dB performance loss as the price [87]. In addition, *Enhanced Max-Log-MAP* algorithm scales the extrinsic information with a constant coefficient smaller than 1.0, typically around 0.75, to compensate for the performance loss caused by rough

approximation of MAX^*.

6.2.3 Metric Initialization for Circular Turbo Code

It can be seen from Equation 6.7 that both forward and backward metrics are calculated recursively. The initial values of forward and backward metrics can dramatically affect the values of following forward and backward metrics and also the Turbo decoding performance.

In the case of classical Turbo codes, since the tail bits compel trellis starting and ending at the *zero-state*, forward metrics and backward metrics can be easily initialized as:

$$\begin{aligned} &\alpha_0(S_0 = 0) = 1 \\ &\alpha_0(S_0 = s) = 0 \quad \textit{for all } s \neq 0 \\ &\beta_N(S_N = 0) = 1 \\ &\beta_N(S_N = s) = 0 \quad \textit{for all } s \neq 0 \end{aligned} \tag{6.18}$$

This kind of initialization emphasizes that both forward and backward metric computations are expected to start from *zero-state.*

However, the trellis of circular Turbo codes begins and finishes at a *circular state* which is unknown to decoder. Since the initial values might dramatically affect the decoding performance, a special initial value estimation module has to be introduced to the circular Turbo decoder.

[96] [97] suggested that the circular state can be obtained by the decoder through starting the forward and backward some steps ahead of the circular states, called the prologue part. In practice, a prologue of 32 steps is sufficient to converge to the right circular state. However, this scheme imposes on extra decoding latency and computational cost contributed by the forward and backward metrics computations in the prologue part.

Owing to the iterative feature of a Turbo decoder, we proposed two new approaches to initialize forward metrics and backward metrics. The first approach we call feedback initialization. The mechanism of feedback initialization is that before the first iteration, all trellis states will be assumed to be equiprobable, and forward metrics and backward metrics

are set to value one:

$$\alpha_0(S_0 = s) = 1$$
$$\beta_N(S_N = s) = 1 \tag{6.19}$$

During the MAP decoding process, forward and backward metrics are recursively updated according to Equation 6.7. During the decoding process, some initialization errors can be rectified. Thus the final forward and backward metrics ($\alpha_N(S_N = s), \beta_0(S_0 = s)$) represent much more reliable information than initial ones. Since the beginning and ending of the trellis are merged at the same state, the final forward metrics and backward metrics of current iteration can be used to initialize the forward and backward metrics at the next iteration.

In contrast with the prologue approach, more errors are produced by the feedback initialization approach at the first several iterations. However, with the increasing of the number of iterations, the performance can be improved further. Most importantly, the feedback approach eliminates the prologue part in order to reduce the decoding latency and additional computational cost.

The feedback initialization approach produces more errors at the first several iterations than the prologue approach because of the arbitrary assumption that all trellis states are equiprobable at the beginning. Thus, a hybrid initialization approach can be proposed to remove this drawback. In the hybrid approach, in the first iteration, the initial forward and backward metrics are acquired from a prologue part. However, during the following iterative process, the initial values are inherited from previous iterations.

6.3 Application of M-binary circular Turbo codes

Due to the significant advantages of M-binary circular Turbo codes, its special case, double binary circular Turbo codes, have been standardized in DVB and WiMAX specification as one of channel coding approaches. The following parts will analyze the tradeoff between different decoding algorithms and conclude the most suitable decoding algorithm for hardware implementation.

6.3.1 Double binary circular Turbo encoder

When M is equal to 2, at every clock cycle, a vector comprising 2bits is fed into an RSC encoder which outputs two systematic bits and two parity bits. The encoder structure of double binary circular Turbo codes standardized in IEEE 802.16 is depicted in Figure 6.3. It can be seen that the encoder is a parallel concatenation of two identical RSC encoders. The corresponding encoded word consists of two systematic bits and four parity bits. The systematic bits are copies of input bits and parity bits correspond to outputs of two RSC encoders.

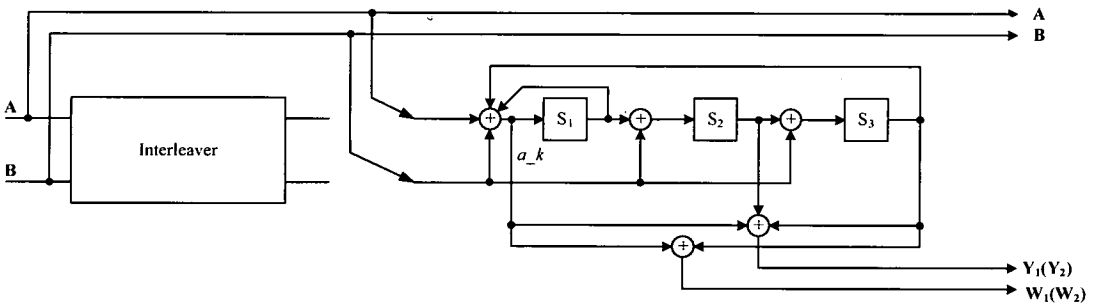


Figure 6.3 Double binary RCS encoder [11]

6.3.2 Double binary circular decoder architecture

The architecture for the double binary Turbo decoder is described in Figure 6.4. It can be seen that at a time, two systematic bits $Y_s(A_k, B_k)$ and two parity bits $Y_p(A_{1k}, B_{1k})$ are fed to MAP decoder I. MAP decoder II will take two interleaved systematic bits and another two parity bits $Y_p(A_{2k}, B_{2k})$. The iterative operations are highlighted in red. The extrinsic information $\ln P_{out}^{ex}(u_k \mid y)$ from a MAP decoder will be passed to the other MAP decoder as the priori information. The dashed line indicates the metric initialisation approach where the final forward and backward metrics of each MAP decoder will be served as the initial metrics in the next iteration. In the end, the hard decision block will select the data pair with the maximum posterior probability as the decoder output $\{A_k, B_k\}$.

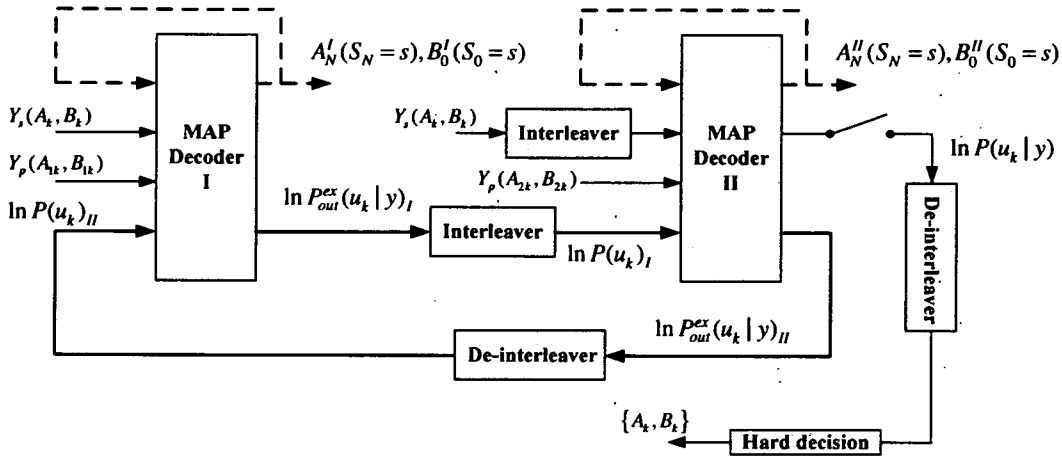


Figure 6.4 Double binary circular Turbo decoder architecture

6.3.3 Decoding performance comparison

In the previous section, several candidates' decoding algorithms for M-binary circular Turbo decoders have been addressed. Since the decoding performance, such as BER and FER, is an important factor in selecting a suitable implemented algorithm, a system model including encoder, decoder and channel model has been built in *C* to estimate the tradeoffs between different decoding algorithms. During the comparisons, unless otherwise stated, system parameters (which are defined by WiMAX specification) selected are as follows:

- 288bits per frame
- Code rate 1/2
- AWGN channel and BPSK modulation

6.3.3.1 Initialization approaches

In order to solve the metric initialization conundrum, three initialization approaches (prologue approach, feedback approach and hybrid approach) are presented in Section 6.2.3. The FER and BER plots for the three initialization approaches are illustrated in Figure 6.5 and Figure 6.6, respectively. It can be clearly seen that all three initialization approaches can provide superior decoding performance as compared with the decoder without metrics initialization.

In addition, after two iterative decoding processes, the prologue approach outperforms the

other two approaches. The performance of the hybrid approach is slightly better than that of the feedback approach in terms of both FER and BER. However, there are no evident differences among these three approaches after four iterations. Since there is no obvious FER or BER gain after five or six iterative decoding process, it can be concluded that the feedback approach can provide the same performance as the prologue and hybrid approaches. On the other hand, according to Section 6.2.3, the feedback approach claims the least computational cost among three initialization approaches. In addition, by removing the prologue procedure, the feedback approach maintains a consistent decoding process, which can facilitate the hardware implementation. Therefore, the feedback approach is the most suitable candidate for a high-speed double binary circular Turbo decoder.

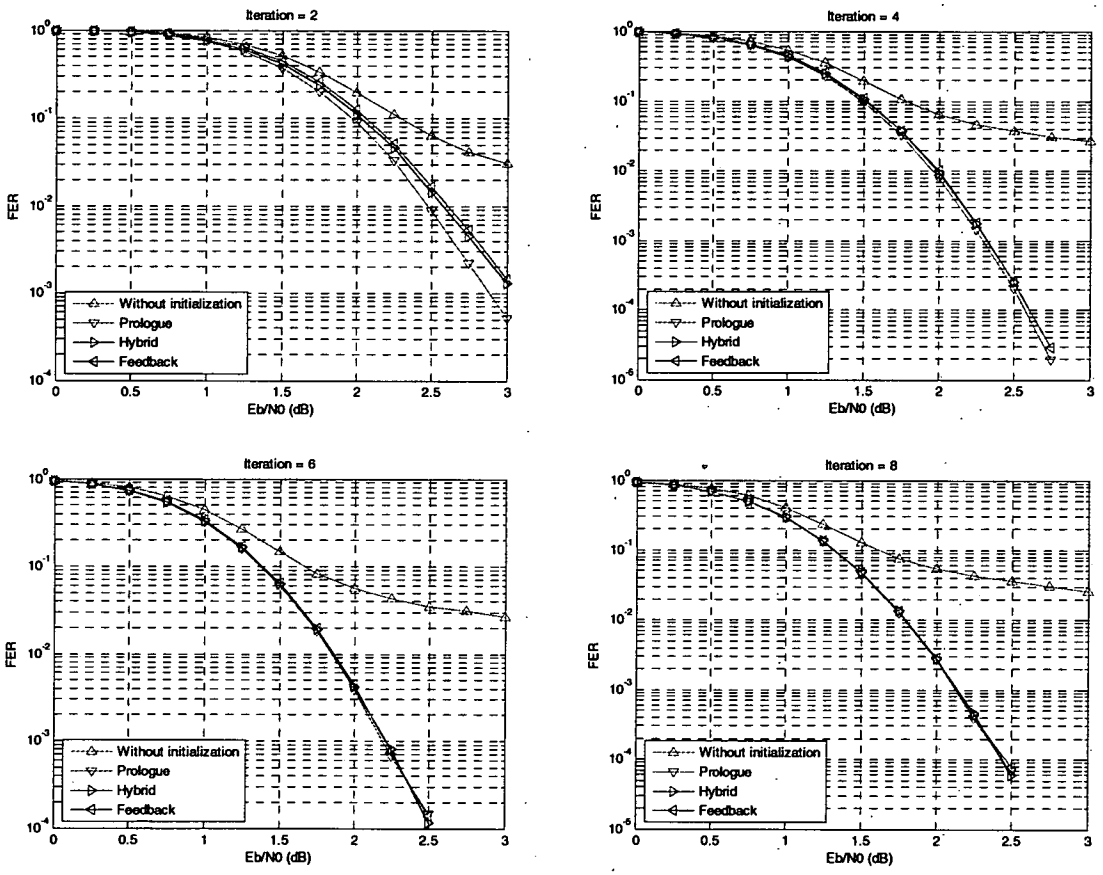


Figure 6.5 FER comparison for different metric initializations

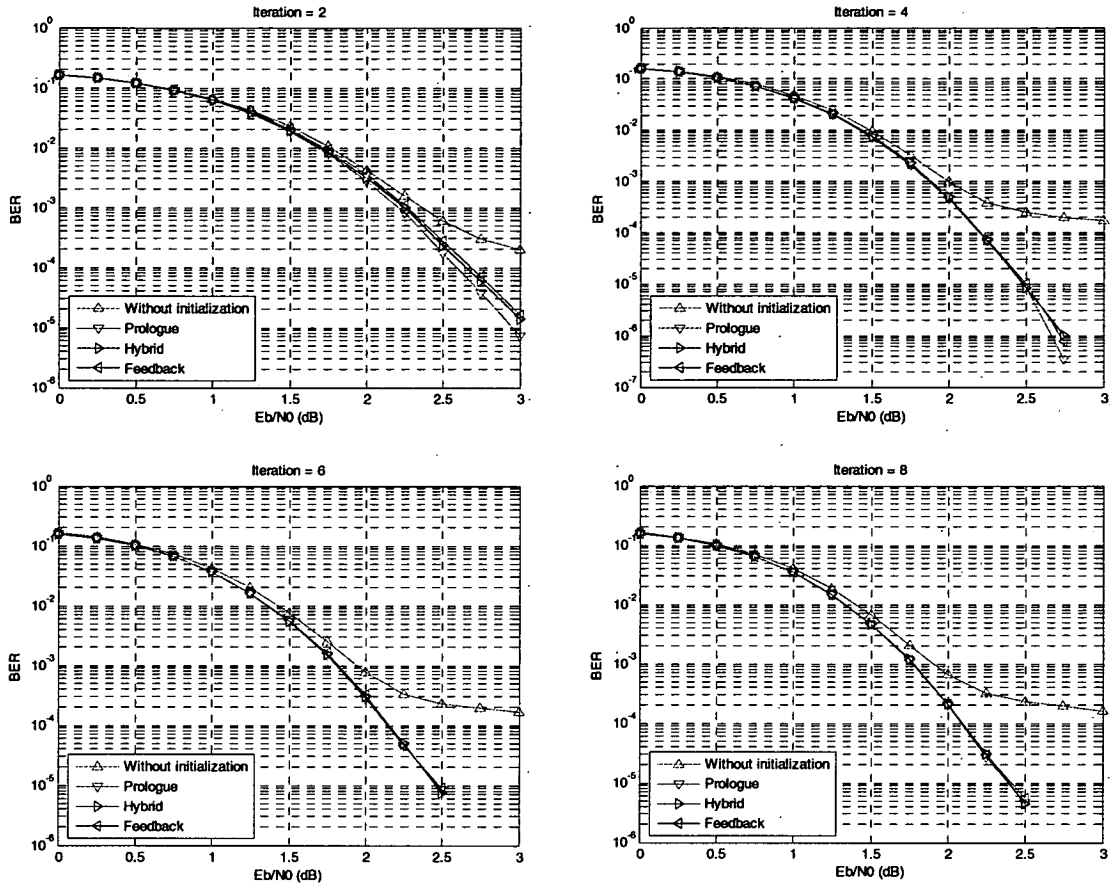


Figure 6.6 BER comparison for different metric initializations

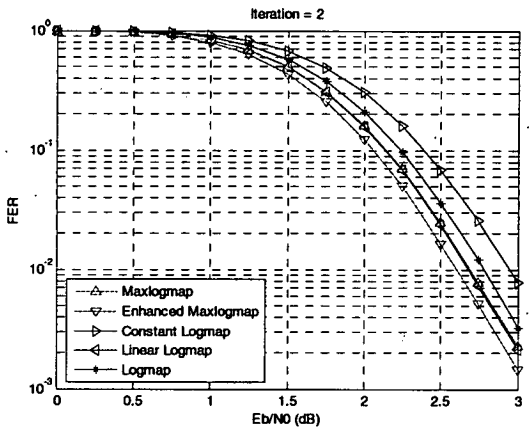
6.3.3.2 MAP decoder algorithms

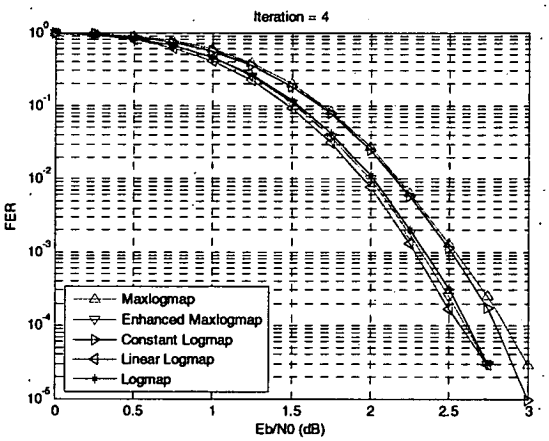
In order to simplify the implementation of a MAP decoder, five approximate MAP algorithms have been illustrated in Section 6.2.2. It can be seen from Equation 6.12 to 6.17 that these approximations are tradeoffs between performance and computational cost. Their FER/BER performances have been investigated in order to find out the best balance point between performance and computational cost.

It can be seen from Figure 6.7 and Figure 6.8 that, although the *MAX-Log-MAP* requires the least computational cost for each iterative process by a means of approximating MAX^* to maximum operation, it suffers from the worst performance degradation. On the other hand, the *Constant-Log-MAP* algorithm approximates correction function to either a constant C or a value of zero, which achieves better performance than *MAX-Log-MAP*, but still incurs

0.2dB performance degradation in terms of FER and 0.1dB performance degradation in terms of BER as compared with *Log-MAP, Linear-Log-MAP* and *Enhanced MAX-Log-MAP*. In addition, after four iterations, *Log-MAP, Linear-Log-MAP* and *Enhanced MAX-Log-MAP* can achieve the same decoding performance. However, from the computational complexity point of view, *Enhanced MAX-Log-MAP* stands out from the three approximations.

On the other hand, the number of iterations is another important parameter that has to be investigated. With the increasing of the number of iterations, the decoding performance is improved and the computational cost is augmented indeed. It can be seen from Figure 6.7 and Figure 6.8, while the number of iterations increases from two to four, the performance improvement caused by the increase of the number of iterations is remarkable. However, the performance gain between four iterations and six iterations is little. Moreover, eight iterations perform the same results compared with six iterations, but it demands 33.33% more computational cost. Thus, it can be concluded that a double binary circular Turbo decoder with five or six iterations is capable of providing a decent decoding performance.





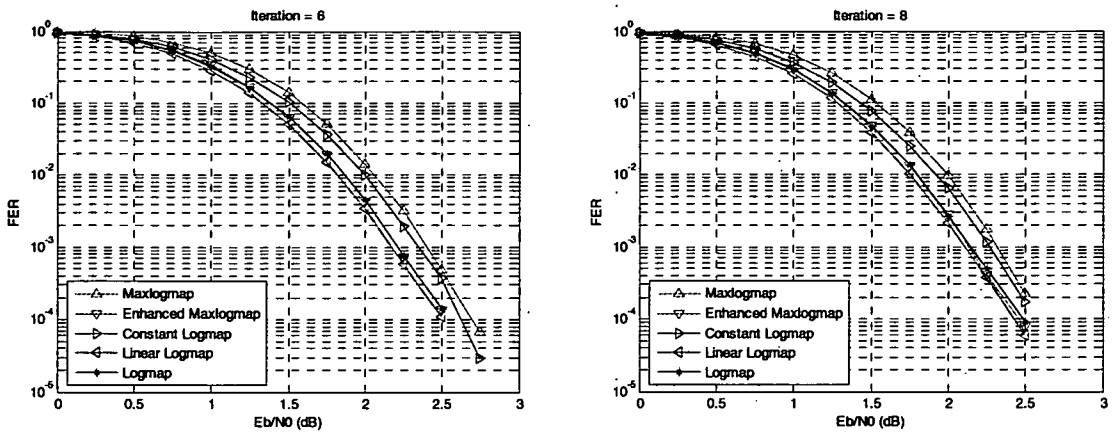


Figure 6.7 FER comparison for different MAP algorithms

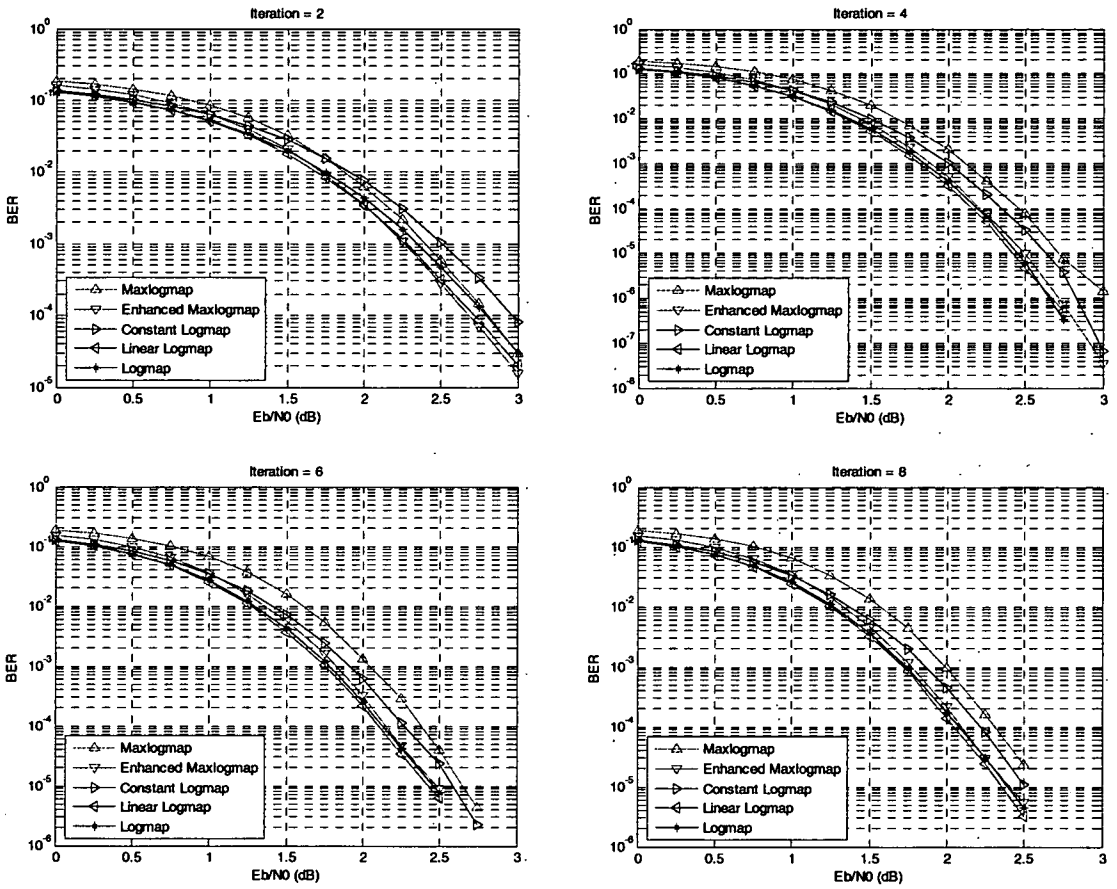


Figure 6.8 BER comparison for different MAP algorithms

6.4 Conclusion

In this chapter, the fundamental principles of the M-binary circular Turbo coding system have been described. We described the mathematic model for the M-binary circular Turbo codes system based on the MAP algorithm. In addition, five approximate MAP algorithms with optimized computational cost have been evolved from classical Turbo decoder to M-binary Turbo decoder. Aiming at the circular Turbo codes, since the trellis starting and ending states are unknown to decoder, two novel metric initialization schemes have been proposed to reduce the computational cost of the traditional prologue initialization approach.

Furthermore, the double binary circular Turbo codes system has been selected as the test bench to demonstrate the system performance under different initialization approaches and MAP algorithms. According to the FER and BER performance, the *Enhanced MAX-Log-MAP* algorithm and feedback initialization approach show the best tradeoff between computational cost and decoding performance. In the following chapter, an efficient hardware implementation of the double binary circular Turbo decoder based on the *Enhanced MAX-Log-MAP* algorithm and feedback initialization approach will be elaborated.

Chapter 7:

mplementation of Double Binary Circular Turbo Decoder on RICA Platform

Double binary circular Turbo codes have become a part of the WiMAX and DVB-RSC system, and also are under discussion for use in future communication standards which demand fast data transmission speed in the range of 10-100Mbps and even above. Compared with the single binary Turbo decoder, the double binary circular Turbo decoder demands around twice as much computational cost, thus how to implement a double binary circular Turbo decoder to fulfill the high-throughput, low-power consumption and flexibility criterion of future communication systems is an attractive topic. Since the RICA architecture has outstanding performance in terms of flexibility, programmability, throughput and power consumption, this chapter will continue to focus on RICA architecture.

In Chapter 6 a system model for double binary circular Turbo decoders was built and the suitable decoder algorithms were analyzed. Based on the findings of Chapter 6, the *Enhanced MAX-Log-MAP* algorithm with feedback initialization approach outperforms other algorithms in terms of performance and computational cost. This chapter will focus on

the RICA implementation of the double binary circular Turbo decoder, such as parallel implementation schemes, quantization schemes, efficient data packing and instruction scheduling on the RICA platform.

This chapter is organized as follows. Section 7.1 makes reasonable analysis of the implementation bottleneck of the proposed algorithm. In order to achieve high throughput, the inherent parallelism of the decoding algorithm has to be explored. Section 7.2 will present the parallel implementation schemes for the double binary circular decoder and also propose a novel approach to reducing the computational cost and speed-up the decoding process. Since directly employing floating-point arithmetic is usually not a proper choice for practical implementations, transformation from floating-point to fixed-point representation becomes mandatory. A suitable quantization scheme for the proposed decoding algorithms is investigated in Section 7.3. The implementation of proposed algorithms with instruction-level and data-level optimization on RICA architecture is illustrated in Section 7.4. The implementation results and conclusion can be accessed in Section 7.5 and Section 7.6, respectively.

7.1 Implementation bottleneck of a MAP decoder

The pseudo codes for an *Enhanced MAX-Log-MAP* decoder with feedback initialization are shown in Figure 7.1. It can be seen that there are four major processing steps of a MAP decoder: branch metric computation, forward metric computation, backward metric computation and extrinsic information computation. The bottlenecks of a MAP decoder are mainly exhibited in two aspects.

According to the estimation and comparison in [89], in the case of 3GPP, a Turbo decoder with *MAX-Log-MAP* demands 2.5 times more computational cost than a constraint length K=7 Viterbi decoder. On the other hand, based on Equation 6.12, it can be seen that a double binary Turbo decoder might double the computational cost of a single binary Turbo decoder employed in 3GPP. Thus, the huge computational cost is one of the major obstacles to achieving an efficient double binary circular Turbo decoder implementation.

MAP Decoder:

{L: length of a frame}

$recA(k \in \{0 \cdots L-1\}), recB(k \in \{0 \cdots L-1\})$: noised systematic bits

$recY(k \in \{0 \cdots L-1\}), recW(k \in \{0 \cdots L-1\})$: noised parity bits

$\Lambda_{in}(k \in \{0 \cdots L-1\})$: priori probabilities

$\alpha(m \in \{0 \cdots 7\}, k \in \{0 \cdots L\})$: forward path metrics

$\alpha_{ini}(m \in \{0 \cdots 7\})$: initial value for forward path metrics

$\beta(m \in \{0 \cdots 7\}, k \in \{0 \cdots L\})$: backward path metrics

$\beta_{ini}(m \in \{0 \cdots 7\})$: initial value for backward path metrics

$\lambda(k \in \{0 \cdots L-1\})$: branch metric

$\Lambda_{out}(k \in \{0 \cdots L-1\})$: extrinsic probabilities

//Branch metrics computation

for k = 0 ***to*** L-1

$\lambda(k) = BMU(recA(k), recB(k), recY(k), recW(k), \Lambda_{in}(k))$

end

//Initialize forward metrics and backward metrics

for m = 0 ***to*** 7

$\alpha(m,0) = \alpha_{ini}(m), \beta(m, L-1) = \beta_{ini}(m)$

end

//Updating forward metrics

for k = 1 ***to*** L

for m = 0 ***to*** 3

$\{\alpha(m,k), \alpha(m+4,k)\} = PMU(\alpha(a,k-1), \alpha(b,k-1), \alpha(c,k-1), \alpha(d,k-1), \lambda(k-1))$

end

end

//Updating backward metrics (in a reversed order)

for k = L-1 ***to*** 0

for m = 0 ***to*** 3

$\{\beta(2*m,k), \beta(2*m+1,k)\} = PMU(\beta(a,k+1), \beta(b,k+1), \beta(c,k+1), \beta(d,k+1), \lambda(k))$

end

end

//Extrinsic information computation

for k = 0 ***to*** L-1

$\Lambda_{out}(k) = softoutput(\alpha(0,k), \ldots, \alpha(7,k), \beta(0,k+1), \ldots \beta(7,k+1), recY(k), recW(k))$

end

//Memorizing the final forward metrics and backward metrics

for m = 0 ***to*** 7

$\alpha_{ini}(m) = \alpha(m,L), \beta_{ini}(m) = \beta(m,0)$

Figure 7.1 Pseudo-code description for a MAP decoder

In addition to computational complexity, a great demand of data transfers is also required by the targeted decoder. For instance, during the branch metric computation step, 16 branch metrics are generated and stored in the memory, which will be loaded by forward and backward metric computations, respectively. At every trellis stage, the updated metrics also have to be stored in the memory, which are required by metric computations at the next stage or served as the initial metrics for the next iteration.

Because of the huge computational cost and data transfer operations, an implementation following the flow presented in Figure 7.1 leads to the unacceptable execution delay. In order to implement a double binary circular Turbo decoder with a high throughput, one feasible way is to exploit the inner parallelism of a MAP decoder. The following section will introduce several prevalent parallel MAP decoder algorithms and also propose a novel parallel MAP decoder algorithm, which not only demands less computational complexity, but also achieves higher throughput and better decoding performance as compared with its counterparts.

7.2 Parallel MAP decoder algorithm

To fulfill significant high throughput required by modern communication standards, the system requirement can not only be achieved by increasing the clock frequency because of the technological constraints. Therefore, exploring the algorithm level parallelism is crucial for an implementation of a high-speed double binary circular Turbo decoder.

7.2.1 Data dependence

From the findings of Chapter 6, the data dependencies between these four steps of a MAP decoder are presented as follows. The first task of a MAP decoder is to compute branch metrics according to the received signals. Forward metrics and backward metrics are obtained by recursive computations. It can be seen from Equation 6.7 that forward metrics $\alpha_k(s)$ only depend on previous metrics $\alpha_{k-1}(s)$ and branch metrics $\gamma_k(s',s)$. Since backward metric are calculated in a reversed order and backward metrics $\beta_{k-1}(s')$ are

deduced by backward metrics $\beta_k(s)$ and branch metrics $\gamma_k(s',s)$. However, there is no data dependency between forward metrics and backward metrics which means backward metric computation can be processed before, in parallel to, or after forward metric computation. On the other hand, extrinsic information can be calculated independently from any other extrinsic information. However, it depends on forward metric computation and backward metric computation finishing first.

7.2.2 Sliding window MAP decoder

If a whole frame can be split into several sub-blocks and each sub-block can be decoded independently, the data dependence inside a MAP decoder can be eliminated and a high throughput Turbo decoder is realized. This kind of approach is called *sliding window* which was originally introduced for a Viterbi decoder [98], and now has been widely adopted by a Turbo decoder. Based on the truncation employed, there are two types of sliding window schemes, *one-side sliding window* and *two-side sliding window* schemes.

7.2.2.1 One-side sliding window

In a *one-side sliding window* scheme [99] [100] [101], either forward metric or backward metric computation is truncated. Figure 7.2 shows an example where backward metric computation is truncated. It can be seen from Figure 7.2 that forward metric computation represented by ①, proceeds continuously across window borders. Otherwise, backward metric and extrinsic information computation, represented by ②, start at the end of each window and stop at the beginning. Since backward metric computation is truncated, in order to ensure that there is no significant performance loss incurred by the truncation, additional guard windows, represented by ③ and providing initial metrics for the truncated backward metric computations, have to be incorporated. Since the extrinsic information computation only depends on the forward metric computation within the same window finishing first, the decoding latency can be reduced. In addition, ①, ② and ③ of different windows can be executed simultaneously, thus the throughput can be further improved.

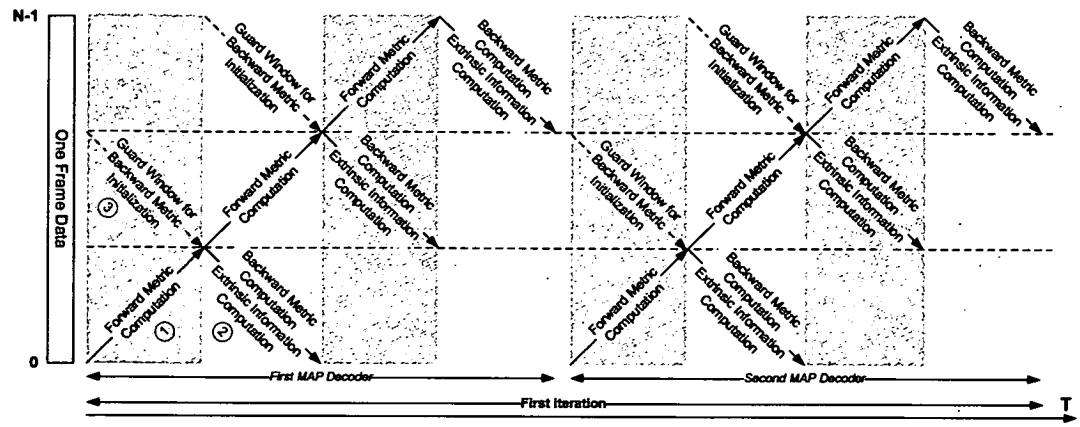


Figure 7.2 One-side guard window scheme

7.2.2.2 Two-side sliding window

In contrast to a *one-side sliding window*, a *two-side sliding window* scheme [102] [103] introduces much more parallelizability to a MAP decoder. In a *two-side sliding window* scheme, which is illustrated in Figure 7.3, both forward and backward metric computations are truncated, thus the operations within a window are totally independent of those within other windows. The decoding processes of different windows can be executed in parallel which results in a distinct reduction on decoding time. The Turbo decoder with the highest throughput is achieved by this scheme [103]. However, two additional guard windows (① and ②) which provide initial metrics for forward and backward metric computations must be added on.

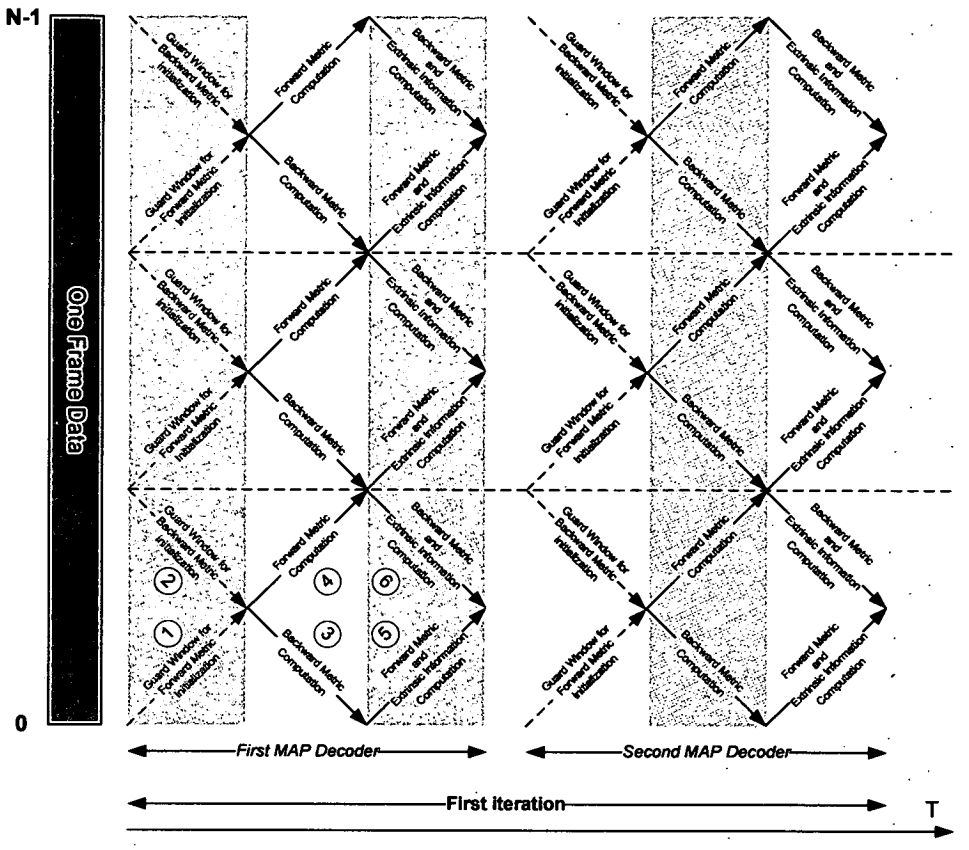


Figure 7.3 Two-side guard window scheme

7.2.2.3 Enhanced two-side sliding window

In the cases of one-side and two-side sliding window schemes, thanks to guard windows, the performance loss caused by trellis truncation can be minimized. However, it has to pay additional computational cost as the price of performance retrieval.

Unlike the Viterbi decoder, the Turbo decoder performs an iterative process. Regarding the simulation provided in Chapter 6, it can be seen that each MAP decoder must be iteratively executed at least five or six times to perform a decent decoding performance. Based on this iterative characteristic of the Turbo decoder, we propose an *enhanced two-side sliding window* which removes guard windows and also provides an attractive performance. Figure 7.4 depicts the proposed *enhanced two-side sliding window* scheme and shows two consecutive iterative operations. Since there is no guard window in an *enhanced two-side sliding window* scheme, initial values for both forward and backward metrics are set to zero in the first iteration. However, at the end of steps ② and ④, the final forward and backward

metrics are stored in memory and will be used by the windows at the second iteration as the initial forward and backward metrics. For example, the red and blue curves show the transmission of backward metrics between the first iteration and the second iteration.

The benefits of the proposed sliding window scheme are exhibited in two aspects. Since it totally removes guard windows, nearly one third of execution time is saved and higher throughput can be achieved as compared with a *two-side sliding window* scheme [102] [103]. Secondly, from the hardware implementation point of view, the power consumption and silicon area consumed by guard windows are also removed, thus life time and area of terminal receiver can be further improved.

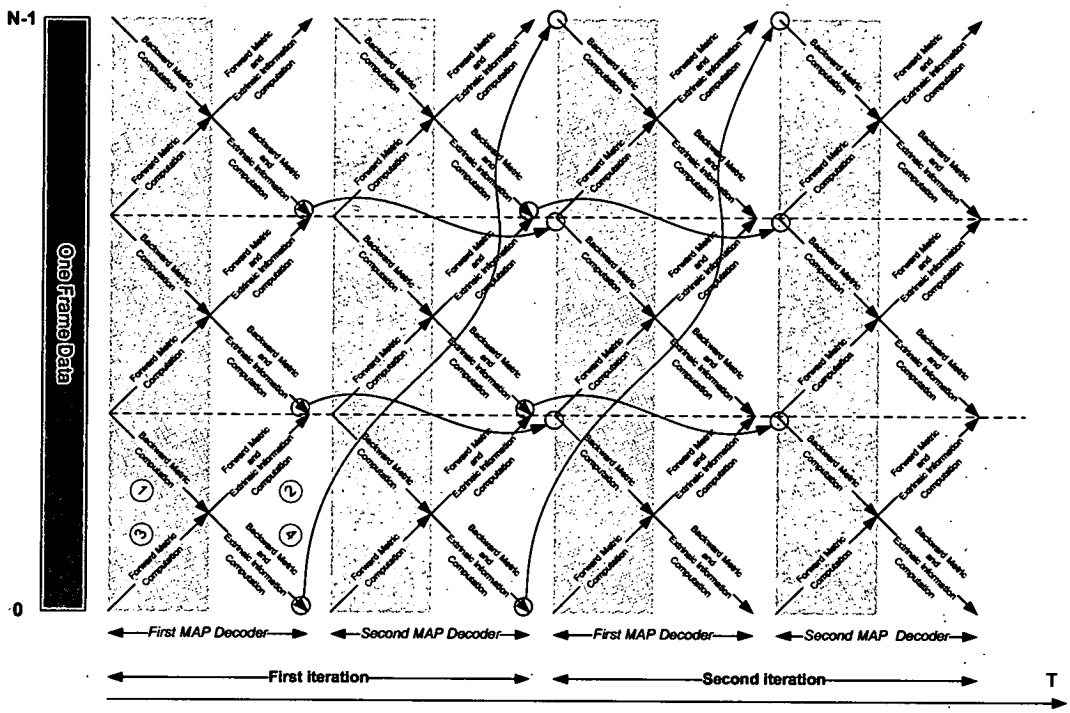


Figure 7.4 Enhanced two-side sliding window scheme

7.2.3 Comparison amongst sliding window schemes

In this section, three sliding window schemes are compared in terms of operational cost, memory requirement and decoding execution time. In addition, the performance comparison in terms of FER and BER are also provided.

7.2.3.1 Computational cost

A. Operational cost

Table 7.1 tabulates the number of operations cost by *one-side sliding window*, *two-side sliding window* and *enhanced two-side sliding window* schemes to decode one-bit data. Since the *enhanced MAX-LOG-MAP* algorithm with feedback initialization approach is assumed, only ADD/SUB and MAX operations are required by each scheme. From Table 7.1, it can see that *two-side sliding window* schemes require more operations than *one-side sliding window* because one more guard window is required by each sub-window. However, the *enhanced two-side sliding window* totally eliminates guard windows, and it needs the least number of operations amongst three sliding window schemes.

Table 7.1 Comparison of operations per decoded bit

Algorithms	No. of operations	
	ADD/SUB	MAX
One-side sliding	127	60.5
Two-side sliding	161	76
Enhanced two-side	93	45

B. Memory requirements

The number of memory operations per decoded bit and total memory requirement to decode a frame with size N is tabulated in Table 7.2. In terms of the number of memory operations, due to the metric calculations of guard windows, *one-side sliding window* and *two-side sliding window* schemes need more memory accessing as compared with *enhanced two-side sliding window* schemes. On the other hand, since the *one-side sliding window* can exploit a ping-pong based memory to buffer forward metrics, it requires the minimum size of memory [101].The *enhanced two-side sliding window* scheme demands more memory than the other two schemes, since it has to store the final metrics at the end of each iterative decoding process.

Table 7.2 Memory operation and requirement comparison (*W* represents the window size, *N* represents frame size (N>2W) and *K* represents the word-length)

Algorithms	No. of memory operations		Total memory requirement
	Load	Store	
One-side sliding	21	6	(16*W+4*N+16)*K
Two-side sliding	25	6	(12*N+16)*K
Enhanced Two-side	17	6	(12*N+16*N/W)*K

C. Decoding execution time

In terms of decoding execution time, Table 7.3 shows that the *two-side sliding window* scheme outperforms the *one-side sliding window* scheme, because it introduces more parallelism to the decoding process. As expected, it can be seen that the *enhanced two-side sliding window* scheme requires least execution time of the three sliding window schemes. In addition, since the decoding time depends on window size *W*, with a smaller window size, higher decoding throughput can be achieved.

Table 7.3 Decoding time comparison (T_1, T_2, T_3 and T_4 represent the execution time of branch metric, forward metric, backward metric and extrinsic information computation, respectively, *W* represents the window size, *N* is the frame size and *Iter* is the number of iterations)

Algorithms	Total decoding time
One-side sliding	$2*Iter*((T_1 + T_2)*W/N + T_1 + T_2 + T_4)$
Two-side sliding	$2*Iter*(3*T_1 + 2*T_2 + T_3 + T_4)*W/2N$
Enhanced Two-side	$2*Iter*(2*T_1 + T_2 + T_3 + T_4)*W/2N$

7.2.3.2 Performance comparison

A. Sliding windows VS consecutive decoding process

Figure 7.5 and Figure 7.6 show the comparison in terms of FER and BER performance between the *enhanced two-side sliding window* scheme and consecutive MAP decoders. From the performance curves of the *enhanced two-side sliding window*, it can be seen that FER/BER performance can be improved by increasing the window size. A better performance can be achieved by a larger window size. When compared with the consecutive MAP decoder, in the case of window size 6, the *enhanced two-side sliding window scheme* with *enhanced MAX-LOG-MAP* algorithm is still superior to consecutive decoding with *MAX-LOG-MAP* algorithm. When the window size is increased to 24, there is no visible

difference between the enhanced *two-side sliding window* scheme and consecutive decoding both with *enhanced MAX-LOG-MAP* algorithm.

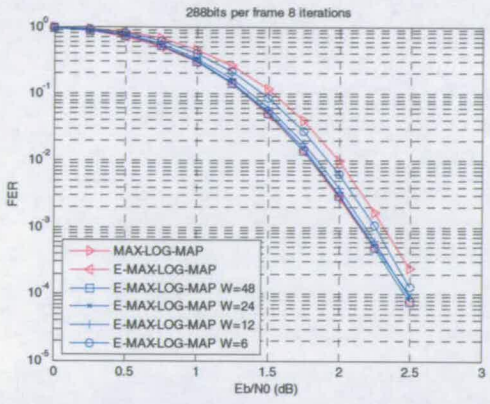


Figure 7.5 FER performance on consecutive and sliding window decoder

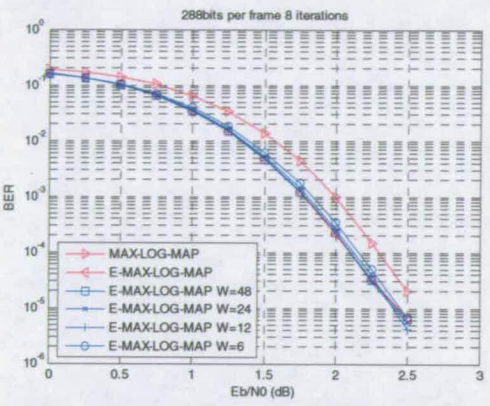


Figure 7.6 BER performance on consecutive and sliding window decoder

B. Enhanced two-side sliding window VS normal two-side sliding window

Since the *enhanced two-side sliding window* scheme removes guard windows, the FER/BER performance comparison between the *enhanced two-side sliding window* and *two-side sliding window* scheme is extremely important, since it has to be proved that removing the guard window will not incur any performance loss. The comparison consists of two parts, *size of sliding window* and *number of iterations*.

1) Size of sliding window

In sliding window schemes, since truncations introduce performance loss, there is a limitation on the size of window. According to [104], a rule of thumb for the minimum length of guard window is 40 in the case of the 3GPP standard. However, there are not any published works which investigated suitable window size for double binary circular Turbo codes.

Figure 7.8 and Figure 7.9 show the performances of the double binary circular Turbo decoder exploiting the *enhanced two-side sliding window* and the *two-side sliding window* with different window size. It can be clearly seen that in the case of window size 6 and 12, the *enhanced two-side sliding window* outperforms the *two-side sliding window*. Only when sliding window size is increased to 24, the *two-side sliding window* scheme can obtain a performance similar to the *enhanced two-side sliding window* scheme. It also can be seen

that the *enhanced two-side sliding window* scheme has the ability to achieve higher throughput than the *two-side sliding window* scheme, because with small sliding window sizes, such as 6 and 12, the *enhanced two-side sliding window* scheme still can provide attractive performance.

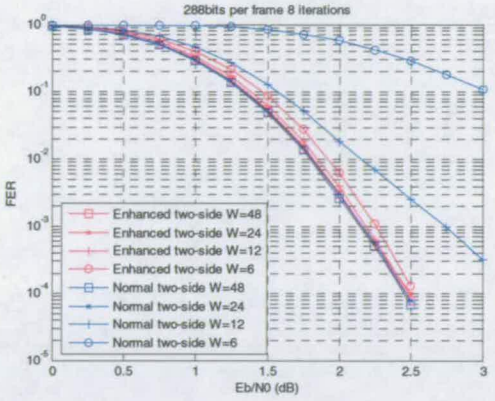


Figure 7.7 FER performance on different sizes of two-side sliding window scheme

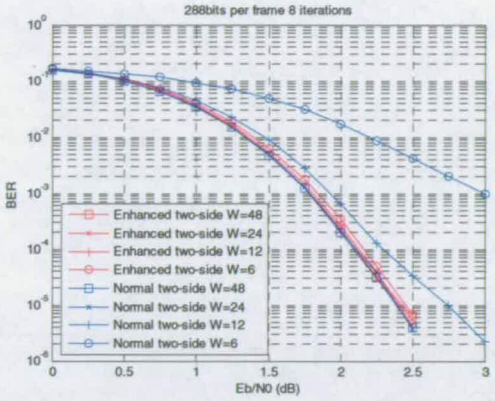


Figure 7.8 BER performance on different sizes of two-side sliding window schemes

2) Number of Iterations

As illustrated in Chapter 6, FER and BER performance are also subject to the number of iterations; with the increase of number of iterations, FER and BER performance can be significantly improved. However, computational cost and power consumption will also be augmented. Since in the *enhanced two-side sliding window* scheme, initial values of each sliding window are provided by previous iterations rather than by guard windows at current iteration, the effect of the number of iteration has to be compared with the *two-side sliding window* scheme.

We have chosen a window size of 24 as the test bench which is the minimum requirement of the *two-side sliding window* scheme. The FER and BER performance are plotted in Figure 7.9 and Figure 7.10, respectively. These plots show that in the case of two and four iterations, the *two-side sliding window* scheme outperforms the *enhanced two-side sliding window* scheme. However, for a higher number of iterations, the curves of the two schemes converge. Since a typical system requires at least five or six iterations, it can be concluded that there is no obvious performance loss when employing the *enhanced two-side sliding window* scheme.

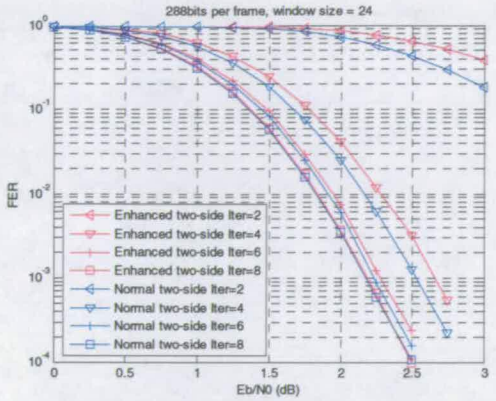


Figure 7.9 FER performance on different number of iterations

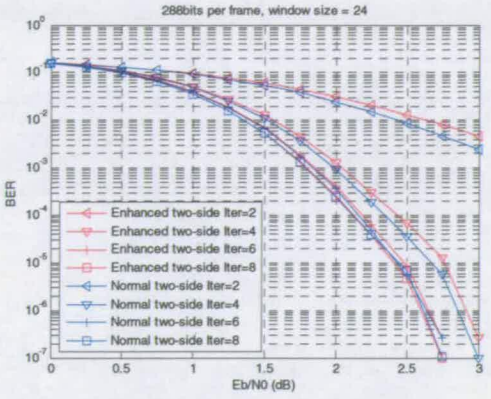


Figure 7.10 BER performance on different number of iterations

7.3 Fixed-point MAP decoder

In previous works, a floating-point representation was employed to model the MAP decoder. However, a floating-point representation is not the best choice for practical implementations, because of its high cost of hardware. Although, a fixed-point implementation is accepted by all architectures, the finite word length will deteriorate the system performance. Although some works [105] [106] [107] have been published regarding a suitable fixed-point implementation of a Turbo decoder, these works are limited to single binary Turbo codes. To our best knowledge, this is the first work that investigates a suitable quantization for the double binary circular Turbo decoder with our proposed sliding window scheme.

The notation *(q, f)* is used to denote a quantization scheme where q represents the total number of bits and f represents the fractional part. In general, the conversion from floating-point to fixed-point representation has to consider three major objectives [108]:

- The dynamic range which is managed by the number of integer bits $(q - f)$ has to be sufficiently large to avoid overflow during the processing.
- In order to achieve appropriate accuracy of the algorithm, enough fractional bits (f) have to be provided.
- The sum of integer and fractional bits (q) must be minimized. In a hardware implementation, this will reduce the area and power consumption of the entire design.

As described in the previous section, a MAP decoder reads four received symbols $Y_k(A,B,Y,W)$. Based on these symbols, branch metrics, forward metrics, backward metrics

and extrinsic information are calculated. Finally, the extrinsic information is passed to the other MAP decoder as priori information. Hence, the following variables of a MAP decoder have to be quantized:

- Input signals
- Internal metrics (branch metrics, forward path metrics and backward path metrics)
- Extrinsic information

7.3.1 Quantization of input signals

The quantization on input signals is crucial to the whole design, because precision of input signals directly determines the system performance and the quantization of extrinsic information and internal metrics both are subject to the input signals.

Since an AWGN channel is assumed, the channel noise is distributed with a *Gaussian* distribution. More than 99% of the channel noise is covered by the dynamic range of *[-3, 3]*. Since a large quantization length will increase area and power of the whole decoder and a small quantization length may incur poor performance, our analysis in this section will be limited to the word lengths of 3, 4 and 5 bits for input signals. FER and BER performance for different quantization schemes, where the first, second and third pairs of numbers represent the quantization schemes for input signals, internal metrics and extrinsic information, respectively, are shown in Figure 7.11 and Figure 7.12, respectively. For example, the first row of numbers *(3:0, 8:0, 5:0)* represents the quantization schemes for input signals, internal metrics and extrinsic information are *(3, 0)*, *(8, 0)* and *(5, 0)*, respectively.

In the case of *3 bits* quantization, the best performance is provided by the *(3, 1)* scheme. Similarly, for *4 bits* quantization, the *(4, 2)* scheme outperforms the *(4, 1)* scheme. Moreover, as the graphs clearly show, there is not much difference between the *(3, 1)* and *(4, 1)* schemes. On the other hand, increasing the word length to *5 bits* with the *(5, 3)* scheme did not provide much improvement compared to the *(4, 2)* scheme. As a result of all these, we can conclude that the best choice is the *(4, 2)* scheme, as it provides a performance close to floating point with the minimum word length.

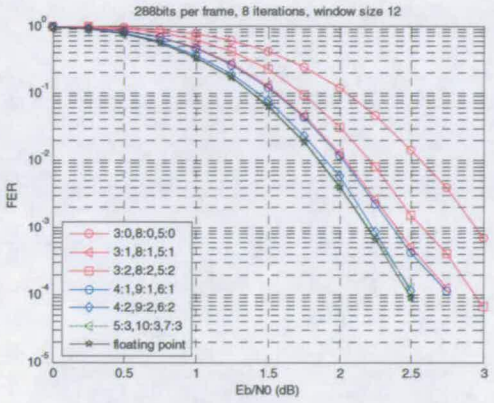


Figure 7.11 FER performance for different quantization schemes on input signals

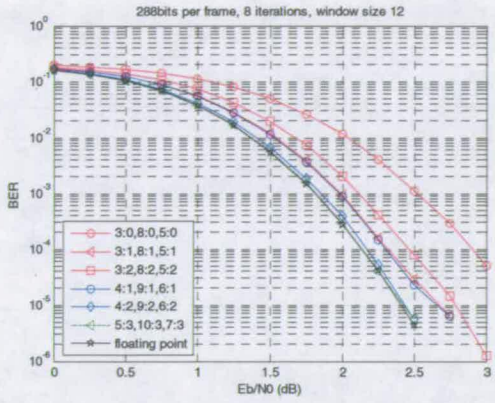


Figure 7.12 BER performance for different quantization schemes on input signals

7.3.2 Quantization of extrinsic information

Extrinsic information which is transferred between two MAP decoders is also critical to the hardware implementation. Since 2 bits are used to represent the fractional part for input signals, the same number of bits for the fractional part could be used for extrinsic information as well. For extrinsic information, various quantization schemes have been tested, such as *(4, 2), (5, 2), (6, 2)* and *(7, 2)*. Their FER and BER performances are plotted in Figure 7.13 and Figure 7.14, respectively. It can be seen that the performance drops dramatically if the dynamic range is less than *3 bits*. But there is no further improvement on both FER and BER performances when the dynamic range is larger than *4 bits*. For example, the *(6, 2)* quantization scheme has a range from *-7.75* to *7.75*, and *-7.75* corresponds to the probability of 4.3e-4 which is already quite a low probability. Further increasing the dynamic range of extrinsic information can only take the priori probability closer to 0, and its effect on the other MAP decoder will be negligible. Hence, quantization scheme *(6, 2)* is suitable for representing the extrinsic information.

7.3.3 Quantization on internal metrics

Since branch metrics, forward metrics and backward metrics are subject to input signals and extrinsic information, as long as quantization schemes for input signals and extrinsic information have been decided, the minimum bit-width requirement for internal metrics can be obtained. If *(4, 2)* and *(6, 2)* schemes are selected for input signals and extrinsic

information, the *(9, 2)* scheme for the internal metrics can give a decent performance but no further reduction on internal metrics can be accepted.

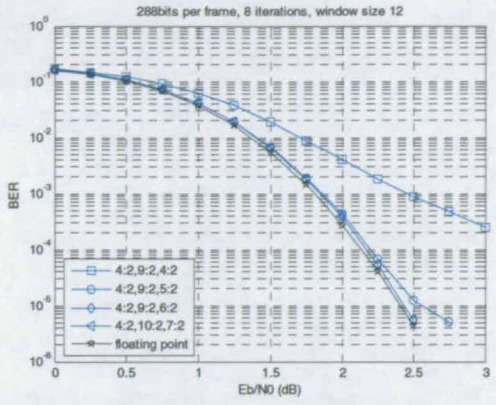


Figure 7.13 FER performance for different quantization on extrinsic information

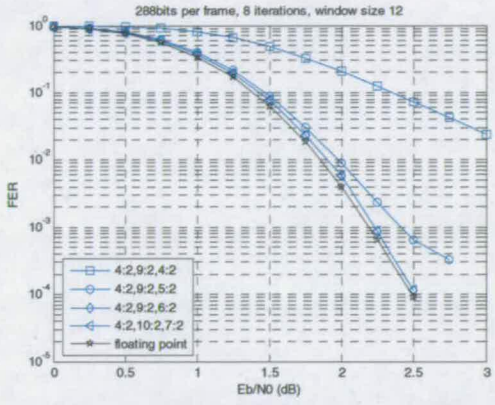


Figure 7.14 BER performance for different quantization on extrinsic information

7.4 Implementation of a MAP decoder on RICA architecture

The detailed implementation issues of a double binary circular Turbo decoder on the dynamical reconfigurable architecture, RICA, will be presented in this section.

7.4.1 Branch metric computation

The first task of a MAP decoder is calculating the branch metrics from its input values, which are related to the respective state transition probabilities. According to the findings of Chapter 6, in the case of double binary Turbo codes, branch metrics can be presented by:

$$L_k^{\gamma}(s',s) = \ln \gamma_k(s',s) = \sum_{l=1}^{4} e_k^l \cdot d_k^l + \ln P(x_k) \tag{7.1}$$

where e_k^l and d_k^l represent the transmitted and received codeword respectively, and $\ln P(x_k)$ is a-priori probabilities which is presented in logarithmic domain.

At a time, a MAP decoder reads two systematic symbols and two parity symbols from the

input buffer. It can be seen that the first term of Equation 7.1 contains 16 different combinations. However, due to the symmetrical characteristic, only eight of them need to be calculated and the other eight values can be obtained by invertion.

Based on the sub-word parallel mechanism on RICA, two branch metric computations can be packed and calculated together. The scheduled branch metric computation is shown in Figure 7.15. A, B, Y, W represent systematic symbols and parity symbols, respectively. Each of them is presented by 8bits. In step 3, the packing function of the *LOGIC* cell would pack and extend two 8bits data into one 32bits data. In addition, the priori probabilities have been already packed and stored in the memory. In step 4, eight 32bits symbols which contain 16 branch metrics are formed by sub-word addition and subtraction operations.

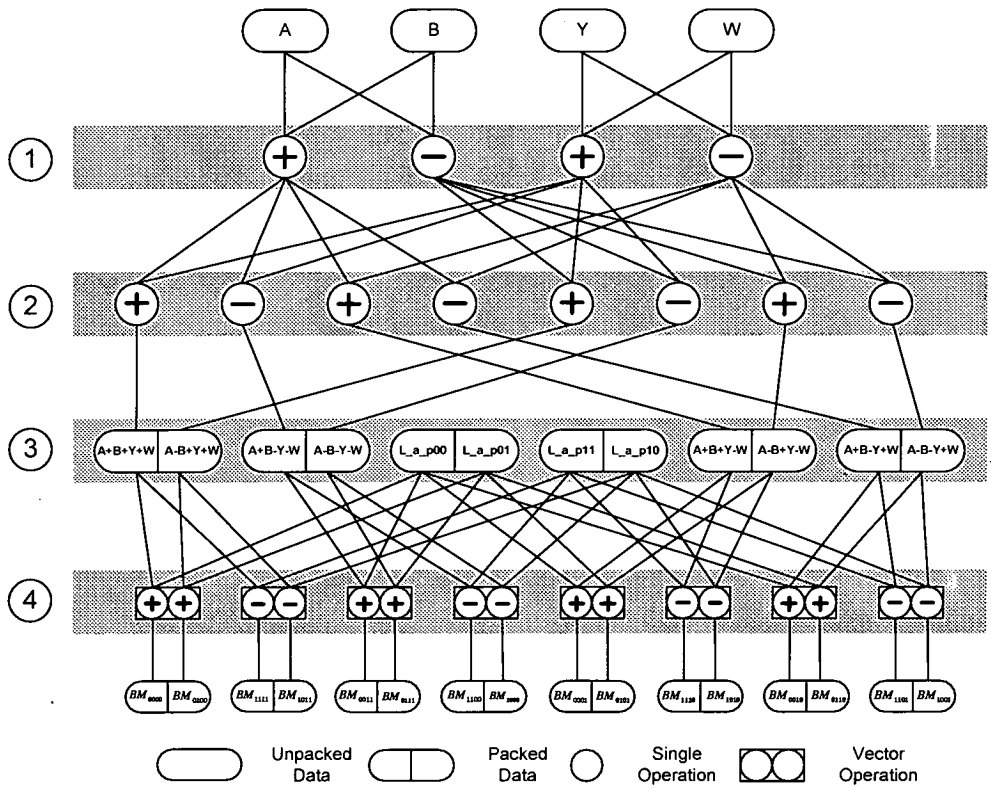
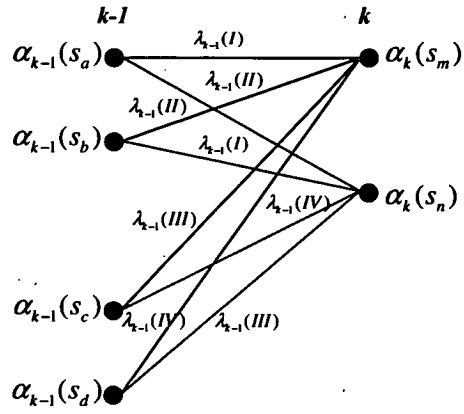


Figure 7.15 Scheduled branch metric computation on RICA

7.4.2 Forward and backward metric computations

Forward and backward metric recursions are the crucial parts of a MAP decoder. The basic block of the recursion is the famed *butterfly* function. A *butterfly* of forward metric

computation, illustrated in Figure 7.16, produces two new metrics through four ancestral metrics and four branch metrics.



$$\alpha_k(s_m) = MAX\left(\alpha_{k-1}(s_a)+\lambda_{k-1}(I),\alpha_{k-1}(s_b)+\lambda_{k-1}(II),\alpha_{k-1}(s_c)+\lambda_{k-1}(III),\alpha_{k-1}(s_d)+\lambda_{k-1}(IV)\right)$$
$$\alpha_k(s_n) = MAX\left(\alpha_{k-1}(s_a)+\lambda_{k-1}(II),\alpha_{k-1}(s_b)+\lambda_{k-1}(I),\alpha_{k-1}(s_c)+\lambda_{k-1}(IV),\alpha_{k-1}(s_d)+\lambda_{k-1}(III)\right)$$

Figure 7.16 Butterfly function for forward metric computation

The trellis diagram of double binary circular Turbo codes is depicted in Figure 7.17. It can be seen that at each trellis stage, eight forward metrics $\alpha_k(s_{0\cdots7})$ have to be calculated and stored into memory. For an efficient implementation on the RICA platform, the trellis can be decomposed to two independent parts. The computations from different parts can be packed in order to reduce the function cells of the RICA platform.

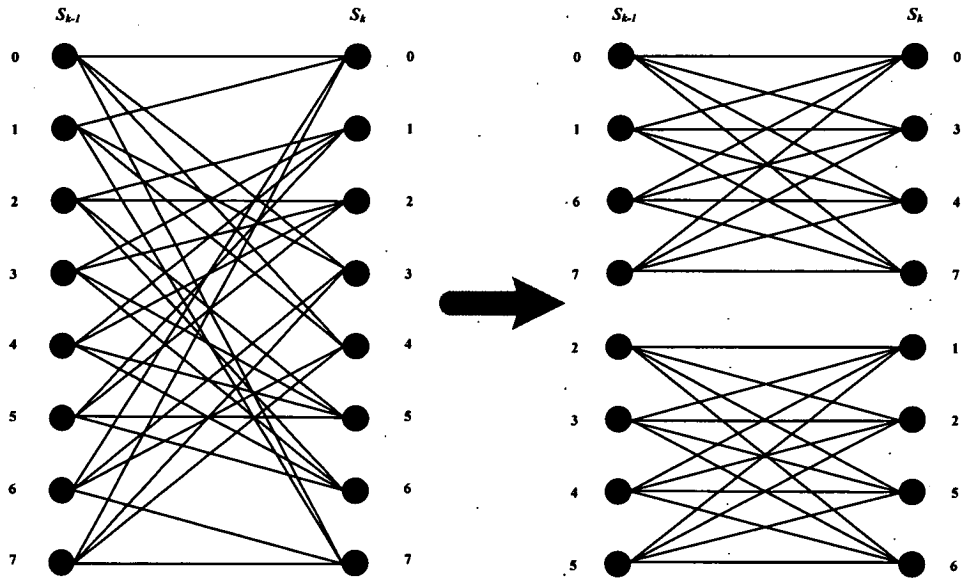


Figure 7.17 Trellis decomposition

In the case of forward recursion, the computations of $\alpha_k(s_0), \alpha_k(s_3), \alpha_k(s_4), \alpha_k(s_7)$ are independent with $\alpha_k(s_1), \alpha_k(s_2), \alpha_k(s_5), \alpha_k(s_6)$. Since each forward metric is represented by 16bits, two butterfly operations from a different trellis block can be packed together on a sub-word parallel RICA architecture. The scheduled forward metric computation on RICA is shown in Figure 7.18.

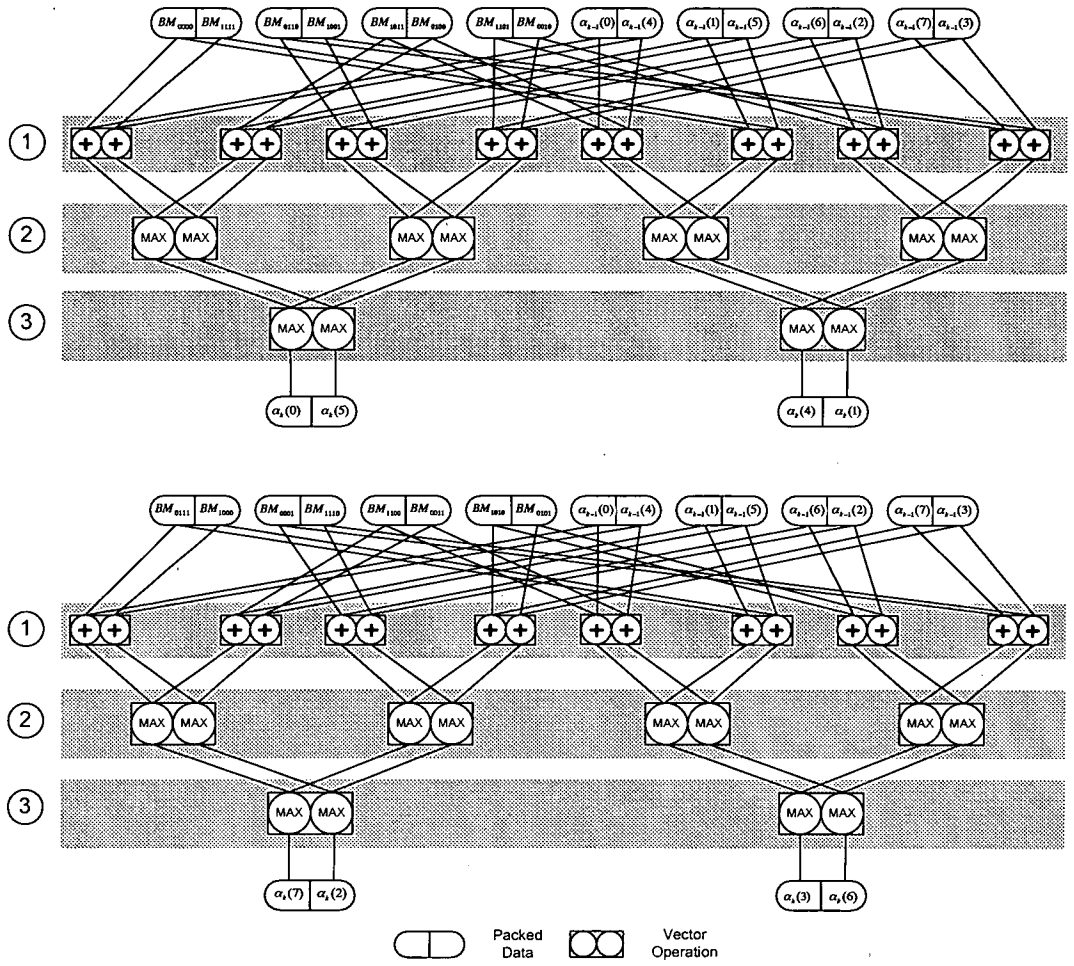


Figure 7.18 Scheduled forward metric computation on RICA

It has to be emphasized that, besides the butterfly unit, a data realignment block is also demanded to reorder the two forward metrics in a 32bits data. In order to facilitate this kind of data aligning, the *LOGIC* cell has been extended to support shuffling the position of the first 16bits and last 16bits between two 32bits data.

On the other hand, since backward metric computation has a similar structure as forward metric computation, the same packing approach can be adopted in backward recursion to reduce the demanding function cells on the RICA platform.

7.4.3 Extrinsic information computation

The calculation of the extrinsic information can also be efficiently implemented on the sub-word parallel RICA platform. According to Chapter 6, in the case of the double binary Turbo decoder, 4 log domain extrinsic probabilities $\left\{L^{ex}_{x_k=00}, L^{ex}_{x_k=01}, L^{ex}_{x_k=10}, L^{ex}_{x_k=11}\right\}$ have to be considered in each MAP decoder. In addition, distributive law $MAX(a+c,b+c) = MAX(a,b)+c$ can be applied to further reduce the computational cost. In the case of $L^{ex}_{x_k=00}$ computation, we can move the addition of parity symbols after maximum operation, which is denoted by Equation 7.2:

$$L^{ex}_{x_k=00} = MAX\begin{Bmatrix} \alpha_{k-1}(0)+\beta_k(0)+Y+W \\ \alpha_{k-1}(2)+\beta_k(1)-Y+W \\ \alpha_{k-1}(5)+\beta_k(2)-Y-W \\ \alpha_{k-1}(7)+\beta_k(3)+Y-W \\ \alpha_{k-1}(1)+\beta_k(4)+Y+W \\ \alpha_{k-1}(3)+\beta_k(5)-Y+W \\ \alpha_{k-1}(4)+\beta_k(6)-Y-W \\ \alpha_{k-1}(6)+\beta_k(7)+Y-W \end{Bmatrix} \tag{7.2}$$

$$= MAX\begin{Bmatrix} MAX\left(\alpha_{k-1}(0)+\beta_k(0), \alpha_{k-1}(1)+\beta_k(4)\right)+Y+W \\ MAX\left(\alpha_{k-1}(7)+\beta_k(3), \alpha_{k-1}(6)+\beta_k(7)\right)+Y-W \\ MAX\left(\alpha_{k-1}(2)+\beta_k(1), \alpha_{k-1}(3)+\beta_k(5)\right)-Y+W \\ MAX\left(\alpha_{k-1}(5)+\beta_k(2), \alpha_{k-1}(4)+\beta_k(6)\right)-Y-W \end{Bmatrix}$$

Since the computations of four extrinsic information are independent with each other, two of them can be packed in a 32bits symbol. The scheduled extrinsic information computation circuits are shown in Figure 7.19. The packing operations for parity symbols in step 2 are ignored.

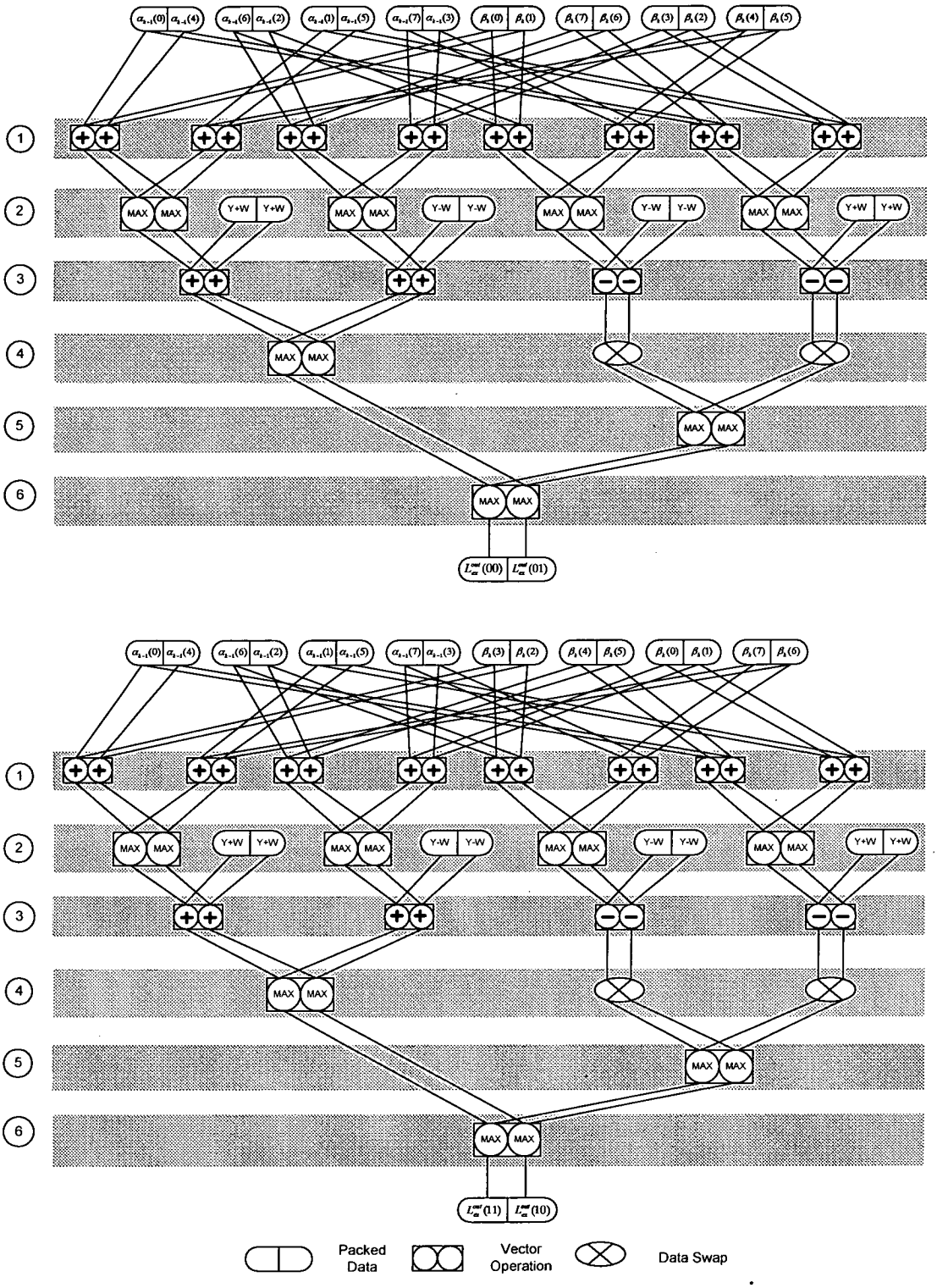


Figure 7.19 Scheduled extrinsic information on RICA platform

7.5 Performance and Results

An *Enhanced MAX-Log-MAP* decoder with feedback initialization has been implemented on RICA architecture. The performance of each sliding window is tabulated in Table 7.4.

Table 7.4 Proposed MAP decoder on advanced RICA

	Throughput	**No. of Steps**	**No. of cells**
One sliding window	46.15 Mbps	31	355

It can be seen from Table 7.4 that, a window-based double binary circular MAP decoder can provide 46.15 Mbps throughput per iteration. After pipelining, this throughput can be up to 46.15*2.5 = 115.38 Mbps per iteration. If five iterations are assumed, the output of a double binary circular Turbo decoder would be 115.38/5 = 23.08 Mbps.

The timing consumption distribution of the implemented MAP decoder is depicted in Figure 7.21. Step ① + Step ③ takes 43.85% of total execution time, otherwise Step ② + Step ④ consumes another 46.84%. While Step ① + Step ③ requires the same computational cost as two guide windows of the *two-side sliding window* scheme, it is fair to conclude that the *enhanced two-side sliding window* scheme can reduce execution time by 30.48% as compared with *two-side sliding window* scheme on RICA architecture.

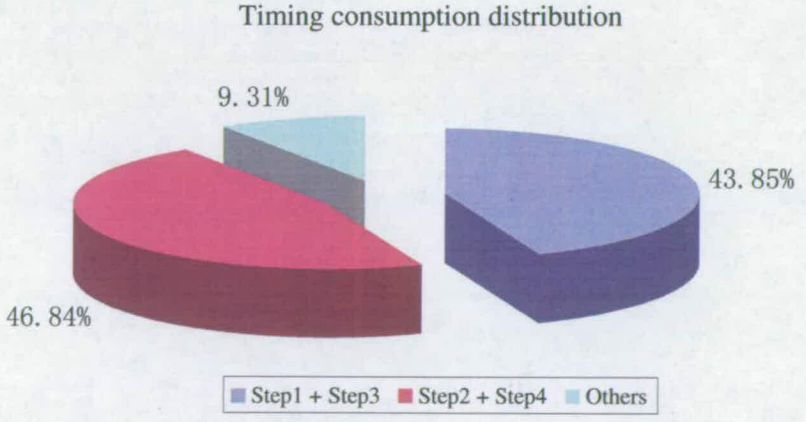


Figure 7.20 Timing consumption distribution for the targeted MAP decoder

As the decoding throughput depends on the number of sliding windows, Table 7.5 summarizes the multiple sliding windows double binary circular Turbo decoder based on proposed algorithms, with five iterative processes. If RICA can provide enough hardware resources, the throughput of the double binary circular Turbo decoder can achieve 184.64 Mbps with eight sliding windows. With our best knowledge, this performance can fulfill the requirement of any current wireless communication standards.

Table 7.5 Decoding throughput with different No. of sliding windows

Number of sliding windows	1	2	4	8
Throughput (Mbps)	23.08	46.16	92.32	184.64

In [109], a high throughput double binary circular Turbo decoder was demonstrated on an application specific instruction set processor (ASIP), Tensilica. By means of integrating a specific instruction set for double binary Turbo decoder into data path of Tensilica Xtensa core, [109] showed a maximum throughput of 201.6 Mbps on a 32 ASIPs Tensilica processor where 16 sliding windows were executed in parallel.

However, [109] did not consider the optimization of implementation at the algorithm level, where the two-side sliding window algorithm was employed. The proposed double binary circular Turbo decoder on RICA platform produces 83.17% throughput gain as compared with the design presented in [109], if the same number of sliding window is exploited.

Because of the lack of the power simulation tool, this thesis can not provide the accurate power performance of the RICA architecture. However, due to the distributed cell based architecture of RICA, it can be assumed that RICA can achieve significant power consumption as compared with general processors and DSPs, which makes RICA more feasible for future portable devices.

7.6 Conclusion

Chapter 6 and Chapter 7 have demonstrated an efficient design for the double binary circular Turbo decoder on the dynamic reconfigurable architecture, RICA. Rather than working at the algorithm level, this chapter investigated efficient implementation approaches to achieve a high throughput design for double binary circular Turbo codes. In Section 7.2, a distinct sliding window scheme was proposed, which not only reduced execution time by 30.48%,

window schemes. Since fixed-point representation is mandatory to hardware implementation, a suitable quantization scheme for the proposed algorithms has been provided by the heuristic method. In Section 7.4, a high throughput double binary circular Turbo decoder has been implemented on RICA architecture by means of instruction and data parallelism. In the end, the throughput of the Turbo decoder can be up to 184.64 Mbps with five iterative processes.

Chapter 8:

Conclusion and Future Works

8.1 Research Summarize

This thesis investigates three underlying reconfigurable architectures for portable devices based on two cases, the Viterbi decoder and the double binary circular Turbo decoder.

Chapter 2 provided a review of the existing literature which was relevant to this thesis.

In Chapter 3, a reconfigurable fabric for the Viterbi decoder was introduced. The design of this architecture was broken down into BMU, ACSU and SMU to support multiple Viterbi decoders from constraint length 3 to 9 and code rate 1/2 and 1/3. This architecture employed fully parallel ACSU and four memory blocks based sliding window scheme to achieve the expected high throughput. Due to the power saving schemes used in the design, for a specific application, the unused parts of BMU, butterfly units of ACSU and sub-memory blocks of SMU were automatically powered off, thus the dynamic power consumption of

these modules was down to zero. This domain specific reconfigurable fabric reduced power consumption by 79.3% with only a 2.2% area overhead as compared with the architecture without power saving strategy. But it paid 3.4 and 2.7 times the power consumption and area penalties for its flexibility.

In Chapter 4, in order to reduce the design cost and time-to-market, a design methodology which can automatically generate a domain specific reconfigurable architecture and map applications on the generated architecture was proposed. By means of associated CAD tools, the design and verification time of a reconfigurable Viterbi decoder were decreased. Six Viterbi decoders with different constraint lengths and code rates have been implemented on the proposed architecture by the developed software tools. In contrast to commercial FPGAs, the proposed architecture demonstrated 66.1% power consumption and 72% area reduction as compared with fine-grained FPGA.

Implementation of a Viterbi decoder on the reconfigurable instruction cell array (RICA) platform, a dynamic reconfigurable architecture programmed by ANSI C, was described in Chapter 5. In order to boost the performance, several advanced optimization approaches, such as sub-word parallel, custom function cells and software pipelining, have been proposed to accelerate the throughput of the Viterbi decoding process on the RICA platform. With the proposed approaches, the throughput of the Viterbi decoder can be improved up to 91% as compared with the general RICA architecture. Ultimately, a Viterbi decoder with the throughput of 56.4 Mbps can be achieved by employing a full parallel ACSU architecture with software pipelining optimization scheme.

Chapter 6 and Chapter 7 demonstrated an efficient double binary circular Turbo decoder design on the RICA platform. A system model for M-binary circular Turbo codes was built in Chapter 6. Based on this model, Chapter 6 explored the design space on algorithm level. According to the FER and BER performance, the *Enhanced MAX-Log-MAP* algorithm and the proposed feedback initialization approach exhibited the best tradeoff between computational cost and decoding performance for double binary circular Turbo codes. Chapter 7 investigated the design space on implementation level and the implementation parameters, such as size of window inside the sliding window scheme, number of iterations and fixed-point representation. In the end, a double binary circular Turbo decoder with scalable throughput (from 23.08 Mbps to 184.64 Mbps) was demonstrated on the RICA platform.

8.2 Specific Findings

This thesis has investigated several reconfigurable architectures targeting for beyond 3G portable devices. The architecture presented in Chapter 3 provided the best performance in terms of low consumption, area and throughput, and it also can be easily integrated with other IPs and a RISC processor resulting in an efficient and effective platform for beyond 3G portable devices. However, its tremendous design and verification cost limit its further applications.

A reconfigurable architecture composed of heterogeneous coarse-grained processing units and a 2-D programmable interconnection mesh was proposed in Chapter 4. As compared with generic fine-grained FPGA, this architecture demonstrated 66.1% power consumption and 28% area reduction. Most importantly, the associated CAD design flow can automatically generate a reconfigurable architecture, and map applications on the targeted architecture, thus the time-to-market and non-recurring engineering cost are lessened. The design methodology presented in Chapter 4 exhibits an attractive potential value for beyond 3G portable devices.

A reconfigurable and extendible architecture, RICA, was introduced in Chapter 5. The hardware modules inside the RICA consist of heterogeneous coarse-grained instruction cells (ICs) which can execute assembly-like instructions. The RICA exhibited better programmability than the previous two architectures, since the associated tools can take the high-level C codes and transform them into the final binary to be loaded into the hardware. As compared with the previous two architectures, the RICA must pay a performance penalty for its programmability. In the case of the Viterbi decoder, the maximum throughput so far is 56.4 Mbps. In addition, the custom defined ICs can be easily extended on the base architecture, which eliminated the need for RISC+IP-based architecture. The approach simplified the whole system architecture and provided the opportunities to remove the bottleneck of software implementation.

Chapters 6 and 7 demonstrated a top-down design approach to implement a considerable complex decoding algorithm, double binary circular Turbo decoder on RICA platform. By means of building a system model for double binary circular Turbo codes, the tradeoffs between computational cost and decoding performance for different decoding algorithms have been comprehensively studied. *Enhanced MAX-Log-MAP* algorithm, feedback

initialization approach and enhanced two-side sliding window stood out from their competitors. Since fixed-point representation is a must for RICA implementation, the suitable quantization schemes for input signals, internal metrics and extrinsic information were investigated based on a bit-true fixed-point model. The findings from both algorithm and implementation level lead to a high performance implementation on the RICA platform.

8.3 Directions for further research

The domain specific reconfigurable architecture presented in Chapter 3 can be treated as a reconfigurable IP core. A platform targeting 4G portable devices can be an integration of an RISC and several reconfigurable IP cores, where each reconfigurable IP core tackles one computational intensive task, such as Viterbi decoder, Turbo decoder and FFT. Future research work will focus on a suitable connection network for this kind of platform which must provide seamless data transmission between reconfigurable IPs and the communication between reconfigurable IPs and the RISC.

As presented in Chapter 4, the routing network of the proposed architecture occupied 85.5% total area and consumed 79.5% total power. Future work will focus on the optimization of the interconnection mesh to reduce the area and power overheads. In future, a mixed interconnection mesh with tri-buffers and pass transistors can be employed in the proposed array architecture to lessen the area and power overheads. In order to reduce the time-to-market, a library of PUs for different applications, such as FFT, FIR, Viterbi decoder and Turbo decoder can be established. According to different system requirements, the CAD tools can select the suitable PUs from the library to balance the power consumption, area and throughput.

It can be seen from Chapter 5 and Chapter 7 that the RICA platform with advanced optimization approaches can significantly improve the performance. In the case of the Viterbi decoder, as compared with general RICA architecture, the throughput of the Viterbi decoder can be improved by up to 91% by means of the advanced optimization approaches. However, these advanced optimization approaches, such as sub-word parallel, custom function cells and software pipelining are manually implemented. In the future, the function of the compiler needs to be enhanced, which can automatically optimize the applications to achieve the expected outcomes. On the other hand, the current backend simulation tool only can provide

executable time of a targeted application. In order to obtain a comprehensive estimation of RICA core, the backend simulation tool needs to be extended to analyze power consumption and area for the targeted application. Thus, at system level design stage, the designer can more efficiently partition the software and hardware sections.

Appendix A

Branch Metric Computation for M-binary circular Turbo decoder

Branch transition probability, also called branch metric, can be denoted by:

$$\gamma_k(s',s) = P(\{D_k \wedge s\} | s') \tag{A.1}$$

Referred to Bayes' rule, the definition of branch metric can be rewritten as:

$$\begin{aligned}\gamma_k(s',s) &= P(\{D_k \wedge s\} | s') \\ &= P(D_k | \{s' \wedge s\}) \cdot P(s | s') \\ &= P(D_k | E_k') \cdot P(x_k)\end{aligned} \tag{A.2}$$

where $P(x_k) = P\left(x_k = \underset{i=1,\ldots,2^m}{X_i'} \right)$ is a-priori probability of vector x_k which can be known before a MAP decoder.

If assuming the transmission channel is a memoryless Gaussian channel and BPSK modulation is adopted, we can rewrite the first term in (A.2) as:

$$\begin{aligned}P(D_k | E_k') &= \prod_{l=1}^{m+n} P(d_k^l | e_k^l) \\ &= \prod_{l=1}^{m+n} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{E_b}{2\sigma^2} (d_k^l - a e_k^l)^2 \right) \\ &= \frac{1}{\left(\sqrt{2\pi}\sigma\right)^{m+n}} \exp\left(-\frac{E_b}{2\sigma^2} \sum_{l=1}^{m+n} (d_k^l - a e_k^l)^2 \right) \\ &= \frac{1}{\left(\sqrt{2\pi}\sigma\right)^{m+n}} \exp\left(-\frac{E_b}{2\sigma^2} \sum_{l=1}^{m+n} (d_k^{l2} - 2a e_k^l d_k^l + a^2 e_k^{l2}) \right)\end{aligned} \tag{A.3}$$

where e_k^l and d_k^l are the individual bits of the transmitted and received codewords E_k' and D_k, respectively. E_b is the transmitted energy per bit, σ^2 is the noise variance and a is the fading amplitude.

In order to reduce the computation complexity, the common portions of (A.3) are eliminated. Thus, (A.3) can be rewritten as:

$$P(D_k \mid E_k^{'}) \approx \exp\left(\sum_{l=1}^{m+n} e_k^l \cdot d_k^l\right) \tag{A.4}$$

Replacing (A.2) by (A.4), the branch metric will be deduced by:

$$\gamma_k(s', s) = \exp\left(\sum_{l=1}^{m+n} e_k^l \cdot d_k^l\right) \cdot P(x_k) \tag{A.5}$$

Appendix B

Publications from this work

- **Cheng Zhan**; Arslan, T.; Erdogan, A.T.; MacDougall, S.; "An Efficient Decoder Scheme for Double Binary Circular Turbo Codes", *2006 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2006. ICASSP 2006 Proceedings. Volume 4, 2006 Page(s):IV-229 - IV-232

- **Cheng Zhan**; Khawam, S.; Arslan, T.; Lindsay, I.; "Architecture and design methodology for synthesizable reconfigurable array targeting wireless system-on-chip applications", *IEEE International SOC Conference*, 2005. Proceedings. 25-28 Sept. 2005 Page(s):93 – 94

- **Cheng Zhan**; Khawam, S.; Arslan, T.; Lindsay, L.; "Efficient implementation of trace-back unit in a reconfigurable Viterbi decoder fabric", *ISCAS 2005. IEEE International Symposium on Circuits and Systems*, 2005, 23-26 May 2005 Page(s):1048 - 1050 Vol. 2

- **Cheng Zhan**; Arslan, T.; Khawam, S.; Lindsay, I.; "A domain specific reconfigurable Viterbi fabric for system-on-chip applications", *Proceedings of the ASP-DAC 2005. Asia and South Pacific Design Automation Conference, 2005.* Volume 2, 18-21 Jan. 2005 Page(s):916 - 919 Vol. 2

- **Cheng Zhan**; Khawam, S.; Arslan, T.; "Domain specific reconfigurable fabric targeting Viterbi algorithm", *IEEE International Conference on Field Programmable Technology, 2004.* Proceedings. 2004 Page(s):363 - 366

References

[1] Neuvo, Y.; "Cellular phones as embedded systems", 2004 IEEE International Solid-State Circuits Conference, 2004, 15-19 Feb. 2004 Page(s):32 - 37 Vol.1

[2] Minoru Etoh, *Next Generation Mobile Systems: 3G & Beyond*, June 2005

[3] Rahim Tafazolli, *Technologies for the Wireless Future: Wireless World Research Forum (WWRF), Volume 2*, June 2006

[4] WiMAX Overview and Freescale Solutions for Basestation Design, Freescale Technology Forum, Orlando, 2005

[5] Cedric Paillard, "Make sure you're ready for 4G", Semiconductor Insights, Jun 21, 2006, Available from: http://www.commsdesign.com/showArticle.jhtml?articleID=189600052

[6] J.G.Proakis, *Digital Communicatin.* McGraw-Hill, Inc. 4th edition, 2000

[7] ETSI, GSM Technical Specification 05.01, v.8.4.0, ETSI, 1999

[8] 3GPP Technical Specification, "Physical Layer General Description", 3GPP document No. 3GPP TS 25.201 V5.0.0 2001-12

[9] 3GPP Technical Specification, "High Speed Downlink Packet Access: Physical Layer Aspects", 3GPP document No. 3GPP TR 25.858 V5.0.0, 2002-03

[10] IEEE Standard, "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications", IEEE Std 802.11a, 1999

[11] IEEE Standard, "Air Interface for Fixed Broadband Wireless Access Systems", IEEE Std 802.16 2004

[12] IEEE Standard, "Air Interface for Fixed and Mobile Broadband Wireless Access Systems", IEEE Std 802.16e 2005

[13] European Standard, "Digital Video Broadcasting (DVB), Framing structure, channel coding and modulation for digital terrestrial television", ETSI EN 300 744 V1.5.1

[14] Bernard Sklar, *Digital Communications: Fundamentals and Applications*, Prentice Hall PTR, 2nd edition, 2001

[15] Nikolaos S. Voros and Konstantinos Masselos, *System Level Design of Reconfigurable Systems-on-Chip*, Springer, 1st edition, 2005

[16] Berrou, C.; Glavieux, A.; Thitimajshima, P.; "Near Shannon limit error-correcting coding and decoding: Turbo-codes", IEEE International Conference on Communications, 1993. ICC 93. Geneva Volume 2, 23-26 May 1993 Page(s):1064 - 1070 vol.2

[17] S.A.Barbulescu and S.S.Pietrobon, "TUTBO CODES: a tutorial on a new class of powerful error correcting coding schemes, Part I: Code Structures and Interleaver Design," Journal of Electrical and Electronics Engineering, Australia, 19(3):129-142, September 1999

[18] S.A.Barbulescu and S.S.Pietrobon, "TURBO CODES: a tutorial on a new class of powerful error correcting coding schemes, Part II: Decoder Design and Performance,"

Journal of Electrical and Electronics Engineering, Australia, 19(3):143-152, September 1999

[19] Berrou, C.; Glavieux, A.; "Near optimum error correcting coding and decoding: turbo-codes," IEEE Transactions on Communications, Volume 44, Issue 10, Oct. 1996 Page(s):1261 – 1271

[20] Benedetto, S.; Montorsi, G.; "Unveiling turbo codes: some results on parallel concatenated coding schemes, " IEEE Transactions on Information Theory, Volume 42, Issue 2, March 1996 Page(s):409 - 428

[21] A.J Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," IEEE Transactions on Information Theory, vol. IT-13, pp.260-269, April 1967

[22] Heller, J.; Jacobs, I.; "Viterbi Decoding for Satellite and Space Communication", IEEE Transactions on Communications, Volume 19, Issue 5, Part 1, Oct 1971 Page(s):835 - 848

[23] Sparso, J.; Jorgensen, H.N.; Paaske, E.; Pedersen, S.; Rubner-Petersen, T.; "An area-efficient topology for VLSI implementation of Viterbi decoders and other shuffle-exchange type structures", *IEEE Journal of Solid-State Circuits,* Volume 26, Issue 2, Feb. 1991 Page(s):90 – 97

[24] Shung, C.B.; Siegel, P.H.; Ungerboeck, G.; Thapar, H.K.; "VLSI architectures for metric normalization in the Viterbi algorithm", *IEEE International Conference on Communications*, 1990, 16-19 April 1990 Page(s):1723 - 1728 vol.4

[25] Hekstra, A.P.; "An alternative to metric rescaling in Viterbi decoders", *IEEE Transactions on Communications*, Volume 37, Issue 11, Nov. 1989 Page(s):1220 – 1222

[26] Black, P.J.; Meng, T.H.; "A 140-Mb/s, 32-state, radix-4 Viterbi decoder", *IEEE Journal of Solid-State Circuits*, Volume 27, Issue 12, Dec. 1992 Page(s):1877 – 1885

[27] Yun-Nan Chang; Suzuki, H.; Parhi, K.K.; "A 2-Mb/s 256-state 10-mW rate-1/3 Viterbi decoder", IEEE Journal of Solid-State Circuits, Volume 35, Issue 6, June 2000 Page(s):826 – 834

[28] Kang, I.; Willson, A.N., Jr; "Low-power Viterbi decoder for CDMA mobile terminals", *IEEE Journal of Solid-State Circuits*, Volume 33, Issue 3, March 1998 Page(s):473 – 482

[29] Y.-N. Chang, 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, p. 826 to 834, June 2000.

[30] Biver, M.; Kaeslin, H.; Tommasini, C.; "In-place updating of path metrics in Viterbi decoders", *IEEE Journal of Solid-State Circuits*, Volume 24, Issue 4, Aug. 1989 Page(s):1158 – 1160

[31] Chien-Ming Wu, Ming-Der Shieh, Chien-Hsing Wu, Ming-Hwa Sheu; "VLSI architecture of extended in-place path metric update for Viterbi decoders", IEEE International Symposium on Circuits and Systems, Volume 4, 6-9 May 2001 Page(s):206 - 209 vol. 4

[32] Chien-Ming Wu; Ming-Der Shieh; Chien-Hsing Wu; Ming-Hwa Sheu; "An efficient approach for in-place scheduling of path metric update in Viterbi decoders", IEEE

International Symposium on Circuits and Systems, Volume 3, 28-31 May 2000 Page(s):61 - 64 vol.3

[33] Ming-Der Shieh; Ming-Hwa Sheu; Chien-Ming Wu; Wann-Shyang Ju; "Efficient management of in-place path metric update and its implementation for Viterbi decoders", IEEE International Symposium on Circuits and Systems, Volume 4, 31 May-3 June 1998 Page(s):449 - 452 vol.4

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

[35] Gang, Y.; Erdogan, A.T.; Arslan, T.; "An Efficient Pre-Traceback Architecture for the Viterbi Decoder Targeting Wireless Communication Applications", IEEE Transactions on Circuits and Systems I, Volume 53, Issue 9, Sept. 2006 Page(s):1918 – 1927

[36] Truong, T.K.; Shih, M.-T.; Reed, I.S.; Satorius, E.H.; "A VLSI design for a trace-back Viterbi decoder", IEEE Transactions on Communications, Volume 40, Issue 3, March 1992 Page(s):616 – 624

[37] Rader, C.; "Memory Management in a Viterbi Decoder", IEEE Transactions on Communications, Volume 29, Issue 9, Sep 1981 Page(s):1399 – 1401

[38] Bustamante, H.A.; Kang, I.; Nguyen, C.; Peile, R.E.; "Stanford Telecom VLSI design of a convolutional decoder", IEEE Military Communications Conference, 1989, 15-18 Oct. 1989 Page(s):171 - 178 vol.1

[39] Feygin, G.; Gulak, P.; "Architectural tradeoffs for survivor sequence memory management in Viterbi decoders", IEEE Transactions on Communications, Volume 41, Issue 3, March 1993 Page(s):425 – 429

[40] Ma, H.; Wolf, J.; "On Tail Biting Convolutional Codes," IEEE Transactions on Communications, Volume 34, Issue 2, Feb 1986 Page(s):104 - 111

[41] Zhu, Y.; Benaissa, M.; "Reconfigurable Viterbi decoding using a new ACS pipelining technique", IEEE International Conference on Application-Specific Systems, Architectures, and Processors, 2003, 24-26 June 2003 Page(s):360 - 368

[42] Chadha, K.; Cavallaro, J.R.; "A reconfigurable Viterbi decoder architecture", Thirty-Fifth Asilomar Conference on Signals, Systems and Computers, Volume 1, 4-7 Nov. 2001 Page(s):66 - 71 vol.1

[43] Tessier, R.; Swaminathan, S.; Ramaswamy, R.; Goeckel, D.; Burleson, W., "A reconfigurable, power-efficient adaptive Viterbi decoder", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Volume 13, Issue 4, April 2005 Page(s):484 – 488

[44] Xilinx FPGA, http://www.xilinx.com/products/silicon_solutions/fpgas/

[45] Altera FPGA, http://www.altera.com/products/devices/dev-index.jsp

[46] Xilinx Virtex-E family document, http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex_e_em/index.htm

[47] Angarita, F.; Perez-Pascual, A.; Sansaloni, T.; Valls, J.; "Efficient mapping on FPGA of a Viterbi decoder for wireless LANs", IEEE Workshop on Signal Processing Systems Design and Implementation, 2-4 Nov. 2005 Page(s):710 – 715

[48] Man Guo; Ahmad, M.O.; Swamy, M.N.S.; Chunyan Wang; "FPGA design and implementation of a low-power systolic array-based adaptive Viterbi decoder", IEEE Transactions on Circuits and Systems I: Regular Papers, Volume 52, Issue 2, Feb. 2005 Page(s):350 – 365

[49] MSC81xx StarCore-Based DSPs, http://www.freescale.com/webapp/sps/site/taxonomy.jsp?nodeId=0127950E5F8594

[50] Freescale, "How to Implement a Viterbi Decoder on the StarCore SC140", Application note, 2000

[51] TI DSP platform, http://focus.ti.com/paramsearch/docs/parametricsearch.tsp?family=dsp&sectionId=2&tabId=25&familyId=44

[52] Paul M. Heysters, Gerard J. M. Smit, Egbert Molenkamp, "A Flexible and Energy-Efficient Coarse-Grained Reconfigurable Architecture for Mobile Systems", The Journal of Supercomputing 26(3): 283-308, 2003

[53] Gerard J. M. Smit, Ties Bos, Paul J. M. Havinga, Sape J. Mullender, Jaap Smit, "Chameleon -Reconfigurability in Hand-Held Multimedia Computers", HUC 340-342, 1999

[54] Gerard K. Rauwerda, Gerard J. M. Smit, Werner Brugger, "Implementing an Adaptive Viterbi Algorithm in Coarse-Grained Reconfigurable Hardware", ERSA 62-70 2005

[55] Singh, H.; Ming-Hau Lee; Guangming Lu; Kurdahi, F.J.; Bagherzadeh, N.; Chaves Filho, E.M.; "MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications", IEEE Transactions on Computers, Volume 49, Issue 5, May 2000 Page(s):465 - 481

[56] Kamalizad, A.; Plettner, R.; Chengzhi Pan; Bagherzadeh, N.; "Fast parallel soft Viterbi decoder mapping on a reconfigurable DSP platform", IEEE International SOC Conference, 2004. Proceedings. 12-15 Sept. 2004 Page(s):3 – 6

[57] T.R. Halfhill, "Silicon Hive Breaks Out", Microprocessor Report December 2003, available at www.siliconhive.com

[58] Marc Quax and Ingolf Held, "Multi-Standard Embedded Processor for Viterbi Decoding", GSPx TV to Mobile May 17-18, 2005

[59] H. Corporaal and M. Arnold, "Using transport triggered architecture for embedded processor design," Integrated Computer-Aided Engineering, vol. 5, no. 1, pp. 19–38, 1998.

[60] Salmela, P.; Jarvinen, T.; Sipila, T.; Takala, J.; "256-state rate 1/2 Viterbi decoder on TTA processor", 16th IEEE International Conference on Application-Specific Systems, Architecture Processors, 2005. ASAP 2005. 23-25 July 2005 Page(s):370 - 375

[61] J.M.Rabaey and M. Pedram, *Low power design methodologies*, Kluwer, 1996

[62] Parhi, K.K.; "An improved pipelined MSB-first add-compare select unit structure for Viterbi decoders," IEEE Transactions on Circuits and Systems I: Regular Papers, Volume 51, Issue 3, March 2004 Page(s):504 – 511

[63] 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-um CMOS Viterbi decoder," in Proc. IEEE Int. Conf. Application-Specific Systems, Architectures, and Processors, 2000, pp. 195–201.

[64] A. Yeung and J. Rabaey, "A 210 Mb/s radix-4 bit-level Viterbi decoder," Proc. IEEE Int. Solid-State Circuits Conf., Feb. 1995, pp. 88–89.

[65] P. J. Black and T. H.-Y. Meng, "A 1-Gb/s, four-state, sliding block Viterbi decoder," IEEE J. Solid-State Circuits, vol. 32, pp. 797–805, June 1997.

[66] Xilinx Inc., "XC4000E and XC4000X Series Field-Programmable Gate Arrays," Data Sheet, 1997.

[67] Rose, J.; Brown, S.; "Flexibility of interconnection structures for field-programmable gate arrays", IEEE Journal of Solid-State Circuits, Volume 26, Issue 3, Page(s):277 - 282-Mar 1991

[68] Chang Y.-W.; Wong. D.; and Wong. C., "Universal switch modules for FPGA design", ACM Transactions on Design Automation of Electronic Systems, vol. 1, pp. 80-101, January 1996.

[69] Wilton S. J. E., Architectures and Algorithms for Field-Programmable Gate Arrayswith Embedded Memory. PhD thesis, University of Toronto, 1997.

[70] Khawam, S.; Arslan, T.; Westall, F.; "Synthesizable reconfigurable array targeting distributed arithmetic for system-on-chip applications", 18th International Parallel and Distributed Processing Symposium, 2004. Proceedings. 26-30 April 2004 Page(s):150

[71] Khawam, S, "Domain-specific and Reconfigurable Instruction Cells based Architectures for Low-Power SoC", PhD thesis, University of Edinburgh, 2006

[72] Cheng S. T., "Compiling Verilog into Automata",Tech. Rep. UCB/ERL M94/37, Electronics Research Lab, Univ. of California, Berkeley, CA 94720, May 1994.

[73] V.Betz, J.Rose and A.Marquardt, "Architecture and CAD for Deep-Submicron FPGAs", Kluwer Academic Publisher, 1999. ISBN 0-7923-8460-1

[74] Ho, C.H.; Leong, P.H.W.; Luk, W.; Wilton, S.E.J.; Lopez-Buedo, S., "Virtual Embedded Blocks: A Methodology for Evaluating Embedded Elements in FPGAs", 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2006. FCCM '06, April 2006 Page(s):35 - 44

[75] Xilinx Inc, "FPGA Xpower tutorial". Available from: http://toolbox.xilinx.com/docsan /xilinx5/help/xpower/xpower.htm

[76] M. Sheng and J. Rose, "Mixing buffers and pass transistors in FPGA routing architectures". ACM/SIGDA International. Symposium on FPGAs, 2001.

[77] S. Khawam, I. Nousias, M.Milward, Y.Ying, T.Arslan, "Reconfigurable Instruction Cell Array", UK Patent Office, UK Patent Application Number 0508589.9, April 2005

[78] Ying Yi, Ioannis Nousias, Mark Milward, Sami Khawam, Tughrul Arslan, Iain Lindsay, "System-level Scheduling on Instruction Cell Based Reconfigurable Systems", 2006 Design Automation and Test in Europe Conference (DATE06), Volume 1, pp. 1-6, 6-10 March 2006, Munich, Germany.

[79] Nousias, I.; Arslan, T., "Wormhole Routing with Virtual Channels using Adaptive Rate Control for Network-on-Chip (NoC)", First NASA/ESA Conference on Adaptive Hardware and Systems2006 (AHS-2006), pp. 420- 423, 15-18 June 2006, Istanbul, Turkey.

[80] Sam Fuller, "Motorola's AltiVec™ Technology", White paper, Freescale Semiconductor,

Inc.

[81] MMX™ Technology, Intel, available from http://www.intel.com/design/intarch/mmx/mmx.htm

[82] Berrou, C.; Glavieux, A.; Thitimajshima, P.; "Near Shannon limit error-correcting coding and decoding: Turbo-codes", IEEE International Conference on Communications, 1993. ICC 93. Geneva Volume 2, 23-26 May 1993 Page(s):1064 - 1070 vol.2

[83] Wu, P.H.-Y., "On the complexity of turbo decoding algorithms", Vehicular Technology Conference, 2001. VTC 2001 Spring. IEEE VTS 53rd Volume 2, 6-9 May 2001 Page(s):1439 - 1443 vol.2

[84] Douillard, C.; Berrou, C.;"Turbo codes with rate-m/(m+1) constituent convolutional codes", IEEE Transactions on Communications, Volume 53, Issue 10, Oct. 2005 Page(s):1630 – 1638

[85] Berrou, C.; Jezequel, M.; Douillard, C.; Kerouedan, S.;"The advantages of non-binary turbo codes", IEEE Information Theory Workshop, 2001. Proceedings. 2001 2-7 Sept. 2001 Page(s):61 – 63

[86] Weiss, C.; Bettstetter, C.; Riedel, S.; Costello, D.J., Jr; "Turbo decoding with tail-biting trellises", International Symposium on Signals, Systems, and Electronics, 1998. ISSSE 98. 1998 29 Sept.-2 Oct. 1998 Page(s):343 – 348

[87] Robertson, P.; Villebrun, E.; Hoeher, P.; "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain", IEEE International Conference on Communications, 1995. Volume 2, 18-22 June 1995 Page(s):1009 - 1013 vol.2

[88] Hagenauer, J.; Hoeher, P.; "A Viterbi algorithm with soft-decision outputs and its applications", IEEE Global Telecommunications Conference, 1989, and Exhibition. 'Communications Technology for the 1990s and Beyond'. GLOBECOM '89. 27-30 Nov. 1989 Page(s):1680 - 1686 vol.3

[89] Vogt, J.; Koors, K.; Finger, A.; Fettweis, G.;" Comparison of different turbo decoder realizations for IMT-2000," Global Telecommunications Conference, 1999. GLOBECOM '99 Volume 5, 1999 Page(s):2704 - 2708 vol.5

[90] Woodard, J.P.; Hanzo, L., "Comparative study of turbo decoding techniques: an overview", IEEE Transactions on Vehicular Technology, Volume 49, Issue 6, Nov. 2000 Page(s):2208 – 2233

[91] William E. Ryan, "A Turbo Code Tutorial", New Mexico State University, Las Cruces, NM 88003.

[92] Classon, B., Blankenship. K., and Desai, V., "Turbo decoding with the constant-log-MAP algorithm," in Proc. Second International Symposium on Turbo Codes and Related Topics, pp. 467-470, Sept. 2000.

[93] Jung-Fu Cheng; Ottosson, T.; "Linearly approximated log-MAP algorithms for turbo decoding" IEEE 51st Vehicular Technology Conference Proceedings, 2000. VTC 2000-Spring Tokyo. 2000 Volume 3, 15-18 May 2000 Page(s):2252 - 2256 vol.3

[94] Vogt, J.; Finger, A.; "Improving the max-log-MAP turbo decoder", Electronics Letters, Volume 36, Issue 23, 9 Nov. 2000 Page(s):1937 - 1939

[95] Montorsi, G.; Benedetto, S.;"Design of fixed-point iterative decoders for concatenated codes with interleavers", IEEE Journal on Selected Areas in Communications, Volume 19, Issue 5, May 2001 Page(s):871 - 882

[96] C. Douillard, M. Jézéquel, C. Berrou, N. Brengarth, J. Tousch and N. Pham, "The Turbo code Standard for DVB-RCS," 2nd International Symposium on Turbo Codes & Related Topics, Brest, France, Sept. 2000, pp. 535 – 538.

[97] Guidelines for the Implementation and Usage of the DVB Interaction Channel for Satellite Distribution Systems (draft TR 101 790 V1.3.1) ,http://www.dvb.org/ technology/

[98] Viterbi, A.J.; "An intuitive justification and a simplified implementation of the MAP decoder for convolutional codes", IEEE Journal on Selected Areas in Communications, Volume 16, Issue 2, Feb. 1998 Page(s):260 – 264

[99] Masera, G.; Piccinini, G.; Roch, M.R.; Zamboni, M.; "VLSI architectures for turbo codes" IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Sept. 1999 Page(s):369 – 379

[100] Chien-Ming Wu; Ming-Der Shieh; Chien-Hsing Wu; "Memory arrangements in turbo decoders using sliding-window BCJR algorithm" IEEE International Symposium on Circuits and Systems, 26-29 May 2002 Page(s):V-557 - V-560 vol.5

[101] Chien-Ming Wu; Ming-Der Shieh; Chien-Hsing Wu; Yin-Tsung Hwang; Jun-Hong Chen;"VLSI architectural design tradeoffs for sliding-window log-MAP decoders" IEEE Transactions on Very Large Scale Integration (VLSI) Systems, April 2005 Page(s):439 – 447

[102] Worm, A.; Lamm, H.; Wehn, N.;"VLSI architectures for high-speed MAP decoders" Fourteenth International Conference on VLSI Design, 3-7 Jan. 2001 Page(s):446 – 453

[103] Worm, A.; Lamm, H.; Wehn, N.; "Design of low-power high-speed maximum a priori decoder architectures" Proceedings of Design, Automation and Test in Europe, 13-16 March 2001 Page(s):258 – 265

[104] Han, J.H.; Erdogan, A.T.; Arslan, T.; "High speed max-log-MAP turbo SISO decoder implementation using branch metric normalization" IEEE Computer Society Annual Symposium on VLSI, 11-12 May 2005 Page(s):173 – 178

[105] Montorsi, G.; Benedetto, S.; "Design of fixed-point iterative decoders for concatenated codes with interleavers", IEEE Journal on Selected Areas in Communications, Volume 19, Issue 5, May 2001 Page(s):871 – 882

[106] Michel, H.; Wehn, N.; "Turbo-decoder quantization for UMTS", IEEE Communications Letters, Volume 5, Issue 2, Feb 2001 Page(s):55 - 57

[107] Gibong Jeong; Dan Hsia; "Optimal quantization for soft-decision turbo decoder", IEEE VTC 50th 1999.

[108] Yates. R. Fixed-point arithmetic: An introduction. Digital Sound Labs, March 2001. Available from http://personal.bellsouth.net/lig/y/a/yatesc/fp.pdf

[109] Muller, O.; Baghdadi, A.; Jezequel, M.; "ASIP-Based Multiprocessor SoC Design for Simple and Double Binary Turbo Decoding", Design, Automation and Test in

Europe, 2006. DATE '06. Proceedings Volume 1, 6-10 March 2006 Page(s):1 - 6