# VLSI Design and Implementation of High-Throughput Turbo Decoder for Wireless Communication Systems

A

Thesis Submitted

in Partial Fulfillment of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

By

Rahul Shrestha

Under the Supervision of Prof. Roy Paily


Department of Electronics and Electrical Engineering

Indian Institute of Technology Guwahati

April, 2014

# Certificate

This is to certify that the thesis entitled "**VLSI Design and Implementation of High-Throughput Turbo Decoder for Wireless Communication Systems**" submitted by Rahul Shrestha, a Research Scholar in the Department of Electronics and Electrical Engineering, Indian Institute of Technology Guwahati, for the award of the degree of **DOCTOR OF PHILOSOPHY**, is a record of an original research work carried out by him under my supervision and guidance. The thesis has fulfilled all the requirements as per the regulations of the Institute. The results embodied in this thesis have not been submitted to any other University or Institute for the award of any degree or diploma.

Signed: _____

**Supervisor: Prof. Roy Paily.**
**Department of Electronics and Electrical Engineering,**
**Indian Institute of Technology Guwahati,**
**Guwahati-781039, Assam, India.**

Date: _____

*Dedicated*
*to God, my Parent, Dillip Sir & my Soulmate Neha . . .*

# Acknowledgements

My heart is filled with immense pleasure as I have the privilege to thank everyone and everything those have endorsed my confidence to work harder towards the fulfillment of my thesis work. First and foremost, I am extremely thankful to Prof. Roy Paily for allowing me to work under his supervision. His selfless guidance, patiently effort and moral support have made me more passionate towards my research and escorted my vision into wider dimension. He has inculcated quality of researcher in me by giving ample amount of freedom towards my research work. I would like to express my heart-full thank to my Guru Dr. Dillip Kumar Sar who has made me understand the importance of education and the moral values of life. I am immensely thankful to Prof. Anil Mahanta, Prof. Anup Kumar Gogoi, Dr. A. Rajesh, Dr. Shaik Rafi Ahamed and Dr. Amit Acharyya for their invaluable guidance and concern towards my research work. I owe my gratitude to the reviewers of IEEE Transactions on Circuits & Systems: Regular Paper I and IET Communication for their valuable suggestions and comments those have escorted our work towards wider perspective.

I take this opportunity to thank Chief Scientist Raj Singh from IC Design Group CEERI, Dr. T. Laxminidhi and Dr. Ramesh Kini M from NIT Karnataka, Dr. Bharadwaj Amrutur from IISc Bangalore and Prof. B. Venkataramani from NIT Trichi for organizing excellent training program on FPGA and SoC design flow under Special Manpower Development Programme II, Government of India. I would like to thank my seniors Mr. Kuntal Deka and Mr. Om Prakash Singh for making me understand the concepts of error-correcting codes. Similarly, I owe my gratitude to my seniors Mr. Sanyasi Rao, Mr. Diptaman Hazarika and Mr. Naveen Sudha for clearing the basic concepts of VLSI design and helping in circuit simulation as well as layout designs. It was a great opportunity to work with my enthusiastic colleagues Mr. Gaurav Saxena, Mr. Suyog Jagtap, Mr. Sunil Joshi and Mr. Sudhanshu Bhagel as they have always inspired me to work hard towards my goal and I am extremely thankful to them.

This work was carried out using the resources like SYNOPSYS and CADENCE tools from Special Manpower Development Programme II project sponsored by Department of Information Technology, under Government of India, at Indian Institute of Technology Guwahati. Thereby, I would like to thank the Government and the Institution for allowing me to make extensive use of these resources, as they have greatly helped in our work. I am so thankful to the organizing committee of VLSI Design Conference 2011, which was held in IIT Madras, for awarding me with fellowship to attain all the tutorials and paper-presentations as they have inspired me during the initial phase of my PhD. This work would not have been possible without the support of staff members from Department of EEE, Central Library, Academic Section, R&D Section, Student Affairs and Finance Section of IIT Guwahati. I really appreciate their patience and thank them all for the support. I also acknowledge Ministry of Human Resource Development, Government of India for providing the scholarship.

I am extremely thankful to my parent and wife Neha for their unconditional support and love during good as well as bad periods of my PhD. It gives me immense pleasure to thank my sister Sumnima, Aunty, Chacha and Chachi for their profound support, love and care. I take this

opportunity to thank my grandmother Mrs. Saraswati Rai under whose love, care and support I have grown up to become what I am today. I am extremely thankful to my best friend Mr. Gaurav Saxena who has always supported me unconditionally and has shown rays of hopes even during the worst phase of my PhD. I sincerely thank my colleagues Dhrubojyoti, Vinay, Pawan, Debojit, Fedric, Nagesh Sir and Ratul Sir from VLSI design and communication labs for their support as well as keeping the surrounding enjoyable and informative. I would like to profoundly thank the wonderful IIT-Guwahati campus for providing calm and nature-friendly environment that has always made me think positively as well as regain my momentum in work. I would like to thank Tirupati Balaji Temple in Guwahati for providing wonderful and peaceful place to pray God and get his blessings. I am also thankful to sports and indoor gym facilities of IIT Guwahati which has enable me to maintain healthy life style and overcome my frustrations. I acknowledge all the tea-stalls and their associated staffs, of Core-I/II/III/IV of Academic Complex, for giving me the fuel to work. I thank all the hostel canteens and messes for providing us food at anytime. Last but not the least, I am so thankful to Barak hostel management team for maintaining wonderful environment to stay and relax after day and night of hectic work.

Signed:

**Rahul Shrestha**

# Abstract

Each evolution of wireless communication system demands ever increasing growth in the rate of data transmission with no sign of pause. The demand of higher data-rate, exhibited by increasing users of mobile wireless services, has been on an exponential trajectory. To meet such requirement of data-rate, wireless industry has already specified to further augment data rates up to 3 Gbps milestone for next generation wireless communication systems. Thus, each of the communication blocks involved in a physical layer of wireless communication system must support such higher data-rates. Turbo codes are widely employed in wireless communication systems to achieve reliable information transfer and they deliver near optimal error-rate performance; however, the inherent iterative-decoding process restricts turbo decoder to attain higher data-rate or throughput. Thereby, this work explores enhancement of throughput and energy-efficiency of turbo decoder using optimization in architectural and algorithmic level.

We have carried out performance analysis of turbo code in the DVB-SH wireless communication standard under various conditions. Achievable throughputs of turbo decoder are also estimated under different channel environments. Comparative study of the reported simplified MAP algorithms from algorithmic and architectural aspects is discussed. Based on this study, suitable high-speed algorithm with optimum error-rate performance has been chosen for an ASIC implementation of radix-2 non-parallel turbo decoder in 130 nm CMOS technology node. From the algorithmic perspective, memory reduction techniques for parallel turbo decoder are also presented in this work.

A new technique of un-grouped MAP decoding that resulted in a deep-pipelined MAP-decoder architecture is introduced in this thesis. We have also suggested an architecture of ACS (add compare select) unit that incorporates state-metric normalization technique and it bears shortest critical path delay. By using these high-speed MAP decoders, high-throughput and energy-efficient parallel turbo decoder is designed and it is compliant to 3GPP-LTE and LTE-A wireless communication standards. It has been implemented in 90 nm CMOS technology node and can attain throughput beyond 3 Gbps. Finally, suggested turbo decoder design is implemented on FPGA and tested in a communication environment using a logic analyzer.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

In the field of communication, wireless communication has always been the most vibrant area as it often confronts profound challenges. Such as offering high-speed data transmission over wireless networks, delivering high-definition audio and video, improving voice quality, and expanding broadband data services. Evolution of such wireless communication technologies from second generation (2G) to till-date third generation (3G) has seen a surge in the rate of data transmission and it has been predicted to reach beyond 3 Gbps for the next generation wireless communication standards. Thereby, each communication block associated with the physical layer of wireless communication system must process data at this rate.

Channel decoder is an integral part of wireless communication system and is responsible for reliable data communication. A channel decoder which employs turbo codes for error-correction delivers excellent bit-error-rate performance and it has made this code widely accepted by various wireless communication standards [2]. Peak data-rates of 3G and 4G wireless communication standards which include turbo codes for error correction are shown in Fig. 1.1. It can be observed that the 3GPP-LTE (third generation partnership project - long term evolution) wireless standard has highest peak data-rate among 3G standards [76]. Similarly, as per the specification of ITUR (international telecommunication union radiocommunication-sector) for 4G technology, 3GPP LTE-Advanced supports a peak data-rate of 1 Gbps [77]. On the other hand, the inherent iterative process of decoding restricts turbo decoder to process data at higher data-rate. A great

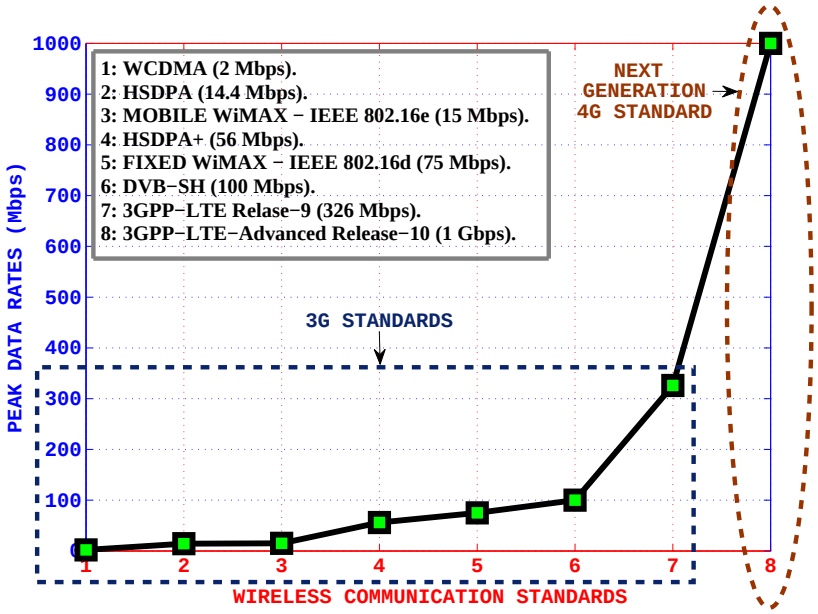


FIGURE 1.1: Ever increasing peak data rates of various wireless communication standards which include turbo code as their error-correcting codes.

deal of work is going on the design of higher data-rate or throughput turbo decoders and their implementations have achieved throughputs up to 2.2 Gbps [49, 71, 74, 79, 80]. However, wireless industry has already targeted milestone throughput beyond 3 Gbps for the next generation wireless communication standards [75]. Thereby, our research goal lies on the design of efficient turbo decoder which can support such higher throughputs for the future wireless communication systems.

## 1.1 Background

Coding research was enlightened with a landmark contribution on the reliable communication over noisy transmission channel by Claude Shannon in 1948 [1]. Theme of this pioneering work was that, for a transmission rate less than the capacity of channel, error introduced by noisy channel can be mitigated to any desired level, using a proper encoding technique for information, without losing the rate of information. Research in this field exploded from the era of 1980s and 1990s when there were novel theoretical developments and they revolutionized the coding methods which have had profound practical impact in wireless mobile, satellite and space communications. Some of the outstanding developments include application of binary convolutional and block codes, devising of practical soft decoding method and exploration of soft-input-soft-output iterative decoding techniques for convolutional and block codes. Enormous research ensued after the remarkable work of Shannnon to construct specific codes with excellent

error-correcting capabilities and their efficient decoding algorithms. A random like code with an efficient iterative decoding technique was invented in the year 1993 and was termed as turbo code [2]. It has exceptionally good error-correcting capability which can deliver near-optimal error-rate performance within 1 dB of Shannon limit. Berrou et al. have pioneered in its development; and an inherent feature of turbo code is the concatenation of constituent codes using pseudo-random interleaver [2–4]. Each of these constituent codes is employed with MAP (maximum a-posteriori probability) or SISO (soft input soft output) decoder which can iteratively process input soft-values such that the output from one decoder is transferred to other one and vice versa, until the final soft-values are obtained. Major influential-resources on random-like codes and iterative decoding are contributed by Batill et al. [5–8]. Similarly, an excellent theoretical justification on the near-optimal error-rate performance of turbo code is provided by Benedetto et al. in [9, 10]. Interestingly, multiple types of turbo codes are reported in the literature such as serial-concatenation, self-concatenation and hybrid-parallel-&-serial-concatenation turbo codes [11–13]. Additionally, various design aspects of turbo coding are comprehensively covered in the reports of Divsalar et al., from Jet Propulsion Laboratory, specifically addressing turbo codes for deep-space communications [14–17].

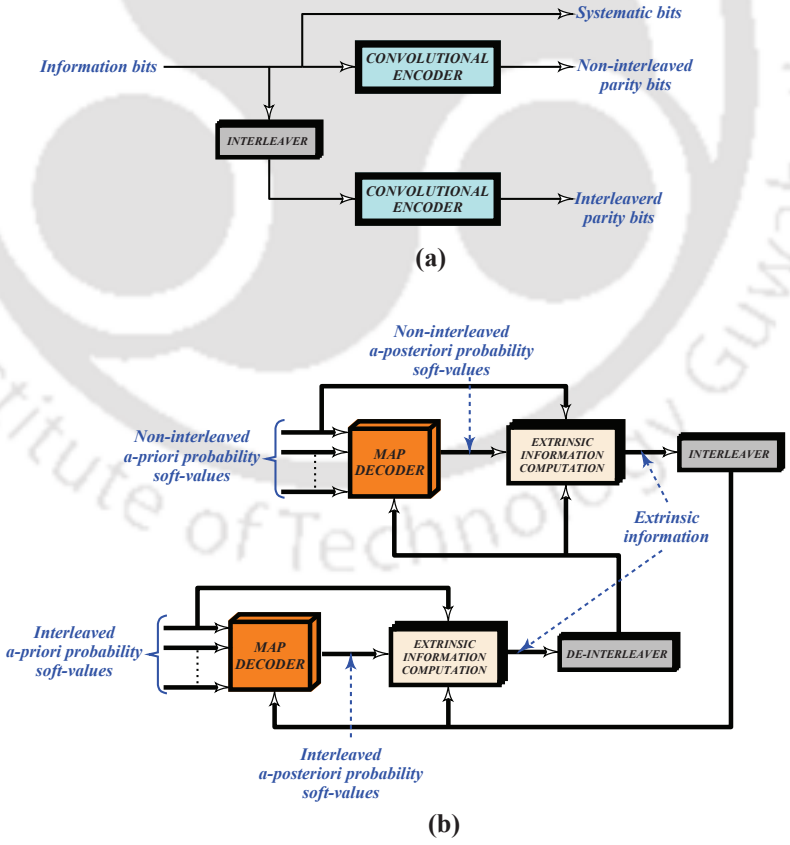


FIGURE 1.2: Basic block diagrams of (a) turbo encoder (b) turbo decoder.

A parallel concatenation of convolutional encoders via pseudo-random interleaver for turbo coding the information bits, which need to be transmitted, is shown in Fig. 1.2 (a). It generates sequences of systematic bits as well as non-interleaved and interleaved parity bits. On the other side, Fig. 1.2 (b) shows a basic block diagram of turbo decoder which is an integration of constituent MAP decoders with pseudo-random interleaver and de-interleaver. The soft-demodulated values of transmitted bits are referred as a-priori probability values and are fed to constituent MAP decoders, as shown in Fig. 1.2 (b). Such MAP decoders are fundamentally based on BCJR (Bahl Cocke Jelinek Raviv) algorithm that works on the principle of trellis graph [18], and it processes a-priori probabilities of systematic and parity bits to produce a-posteriori probability values of the transmitted information bits. Thereafter, the extrinsic information is computed using a-posteriori probability values from the MAP decoder, interleaved/non-interleaved a-priori probability values, and interleaved/de-interleaved extrinsic information from another MAP decoder. Such extrinsic information values are shuffled between these MAP decoders and are iteratively processed along with a-priori probability values to produce error-free a-posteriori probabilities of the transmitted bits.

## 1.2 Design Perspective

From the design aspect of turbo decoder, throughput has been a key issue in the designer's mind because conventional architecture of turbo decoder cannot achieve throughput that is higher than 100 Mbps [65–68, 81]. In the year 2002, R. Dobkin et al. proposed a novel concept of parallel architecture for turbo decoder that can achieve higher throughput [48]. Such architecture processes soft-demodulated a-priori probability values in parallel using stack of multiple MAP decoders. Various contributions on this topic have been reported and are being adopted by latest wireless communication standards [49–52, 70, 71, 74, 79, 80]. With the shrinking CMOS technology node in the semiconductor industry (as predicted by Moore's Law [105]), such complex parallel-turbo decoder occupies nominal silicon area and consumes considerable amount of power. Apart from scaling-up the number of MAP decoders for higher throughput, the achievable throughput ($\Theta_T$) also depends on the clock frequency ($F$) and the number of decoding iterations ($\rho$) as

$$\Theta_T \propto F \quad and \quad \Theta_T \propto 1/\rho. \tag{1.1}$$

Number of decoding iterations remains unaltered as it affects the error-rate performance of turbo decoder. However, there is a provision to enhance decoder-throughput by improving operating

clock frequency. VLSI design and implementation part of our work includes this aspect of turbo-decoder design. Two fundamental metrics are affected by such design methodology: dynamic power dissipated ($\mathbf{P}_{Dyn}$) and silicon area occupied ($\Lambda$). Dependency of clock frequency on the dynamic power consumption is

$$\mathbf{P}_{Dyn} \propto \alpha \times F \times \mathbf{C} \times \mathbf{V}_{DD}^2 \tag{1.2}$$

where $\alpha$ is an activity factor, $\mathbf{C}$ represents overall load capacitance and $\mathbf{V}_{DD}$ is a supply voltage. Low power technique has been incorporated while implementing the parallel turbo decoder-architecture in this work. Similarly, large design-area issue can be resolved to some extent by scaling down the CMOS technology node from channel-length $\ell_{org}$ to shorter channel-length of $\ell_{scal}$. Thereby, scaled silicon area of decoder-architecture ($\Lambda_{scal}$) with respect to the original area ($\Lambda_{org}$) is given as

$$\Lambda_{scal} \approx \Lambda_{org}/\Delta^2 \tag{1.3}$$

where $\Delta$=$\ell_{org}/\ell_{scal}$ is a scaling factor. For example, assuming a decoder area of 10 mm$^2$ at a technology node of 1000 nm, Fig. 1.3 shows the variation of scaled silicon-area consumed by decoder from 1000 to 22 nm technology nodes.

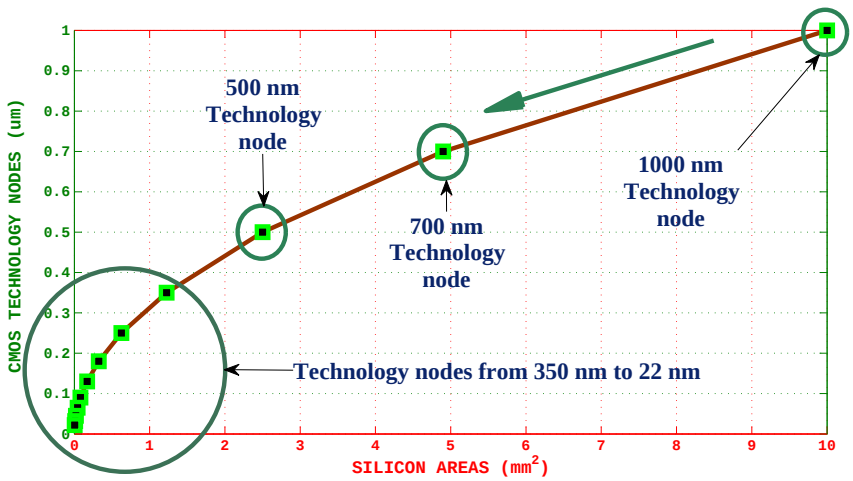


FIGURE 1.3: Variation of silicon areas with the technology scaling from 1000 nm to 22 nm CMOS processes.

## 1.3 Contributions

This thesis explores performance analysis of turbo code in the physical layer of wireless communication system. Consecutively, a comprehensive study of simplified MAP algorithms is carried out to design high-speed non-parallel turbo decoder. Then, we have designed and implemented parallel turbo decoder using the proposed MAP decoder for high-throughput application. Brief descriptions of these contributions are as follows.

- Bit-error-rate performance-analysis of turbo code in the physical layer of DVB-SH (digital video broadcasting - satellite-services to handhelds) wireless communication standard has been carried out. Such analysis has been performed for diverse design parameters which provided adequate information for the design of efficient turbo-decoder architecture that is compliant to wireless communication standards. Similarly, analysis of turbo-decoder throughputs which can be achieved at various decoding iterations under different channel conditions are carried out.

- Conventional BCJR algorithm of MAP decoder is inappropriate for practical implementations, thereby; various simplified versions of this algorithm have been reported. Hence, we have presented comparative study of these simplified MAP algorithms in terms of error-rate performances and digital-architectures. Then, an algorithm with nominal error-rate performance and best operating clock frequency has been chosen for an implementation of radix-2 non-parallel turbo decoder. Additionally, memory reduction techniques are introduced for MAP decoder which can be used in parallel turbo decoder to improve its hardware efficiency.

- We have proposed a new architecture for MAP decoder based on an un-grouped backward recursion technique. Such decoder has a dual-clock architecture which is synchronized to avoid the timing violations. Proposed technique allows digital-architecture of MAP decoder to be deeply pipelined and thus improves operating clock frequency, this eventually elevates achievable throughput of turbo decoder. Additionally, a new state-metric normalization technique has been introduced in this work and it also focuses on shortening the critical path delay. Implementation of parallel turbo decoder with 8 and 64 such MAP decoders are carried out and their error-rate performances are analyzed based on various design metrics. Thereby, this implemented turbo decoder can achieve a throughput higher than 3 Gbps.

- Finally, the testing of hardware-prototypes of MAP and parallel turbo decoders are carried out using FPGA (field programmable gate array). A software model of communication

system has been designed and the error-rate performances of decoders are recorded. The fixed point quantized soft-values from this model are stored using on-board memories of FPGA. Thereafter, these soft-values are fetched and fed to decoder's hardware-prototype. FPGA board has been interfaced with logic analyzer to visualize the outputs from decoders. Finally, these outputs are compared with the simulated outputs of software model of communication system. Comparative error-rate curves are plotted with noted values from software simulations and those values obtained from the hardware implementations.

## 1.4 Organization of the Thesis

The works presented in this thesis are organized as follows. Chapter 2 includes error-rate performance analysis of turbo code and throughput estimation of turbo decoder for DVB-SH wireless communication standard. Algorithmic and architectural comparative-analysis of simplified MAP algorithms and an implementation of non-parallel turbo decoder are presented in chapter 3. Additionally, it contains memory reduction techniques for the parallel turbo decoder architecture. Chapter 4 presents the design and implementation of high-throughput parallel turbo decoder using high-speed as well as deeply pipelined MAP decoders along with interconnecting networks and pseudo-random interleavers. In chapter 5, the hardware-prototypes of MAP and parallel turbo decoders are tested in a simulated communication environment. Finally, conclusion and future direction of this work are included in chapter 6.

# Chapter 2

# Performance and Throughput Analysis of Turbo Decoder for the Physical Layer of DVB-SH Standard

## 2.1 Introduction

EXPERTS associated with the field of satellite and terrestrial communication have succeeded to conceive a hybrid system that is able to operate over both the satellite and terrestrial platforms to serve the hand-held devices. This novel hybrid system has been termed as DVB-SH and is a part of ETSI (European telecommunications standards institute) [19]. DVB-SH standard provides an efficient way of carrying multimedia services over satellite and terrestrial networks at the frequencies below 3 GHz to the mobile and fixed terminals. The significant up-gradations in the physical layer of DVB-SH standard are incorporation of turbo encoder and flexible channel interleaver by replacing Reed Solomon block encoder and Forney interleaver respectively [20]. Turbo code delivers exceptional coding performance which is bounded by various factors and are

well established in the literature [21–26]. The analysis of impacts on the performance of turbo code by such factors is essential and some contributions have been made in the literature [26, 27]. However, these contributions are not compliant to any of the wireless communication standards. In contrast, the turbo code is widely used by 3G and 4G wireless communication standards like DVB-SH, 3GPP-LTE, LTE-A and WiMAX (worldwide interoperability for microwave access). Simultaneously, turbo code has entered the field of practical implementation [28, 29]; thereby, the comprehensive study on additional parameters which can affect the coding performance is essential for providing adequate information to the designers. From the implementation perspective, these additional factors are sliding window size, different MAP algorithms, various modulation schemes, system throughput and maximum frequency of operation. In summary, the system level performance analysis of turbo code compliant to a recent wireless communication standard and impact on its coding performance by such factors are still lacking in the literature.

In this chapter, performance analysis of turbo code using the system level model of physical layer for the DVB-SH wireless communication standard has been carried out. Comprehensive analysis on the coding performance of turbo code for AWGN (additive white Gaussian noise) and frequency selective fading channels with different modulation schemes compliant to DVB-SH standard are presented. The effect of decoding iteration and sliding window size on the coding performance of turbo code for AWGN as well as fading channel environments are investigated. Subsequently, the magnitudes of these parameters for an optimum coding performance of turbo code are obtained. In addition, optimization and dependency of system throughput on the decoding iteration and sliding window size for various processor speeds are presented. Such an analysis is carried out for different architectural configurations of turbo decoder to meet the throughput requirement, as per the specification of 3G wireless communication standard. Coding performance and running time comparison of various MAP algorithm based turbo decoders which are compliant to DVB-SH standard are investigated in AWGN and fading channels. The choice of suitable MAP algorithm for a specific application is also discussed in brief. Finally, the significance of code rate in algorithmic and architectural design of turbo decoders followed by its coding performance for various code rates are presented for DVB-SH standard. To the best of our knowledge, there is no such contribution in literature where the detailed performance analysis of turbo code compliant with DVB-SH standard is presented. This will definitely provide a sufficient knowledge for practical and real time implementation of DVB-SH physical layer as well as turbo decoders compliant with wireless communication standards. This chapter is further organized as follows. Section-2.2 includes the analysis of system level architecture of DVB-SH physical layer. Simulations for the coding performances of turbo code under various conditions and throughput analysis are presented in section-2.3. Finally, this chapter is summarized in section-2.4.

## 2.2 Overview of DVB-SH Physical Layer

This section presents an overview of communication blocks involved in the physical layer of *'spectrum efficient'* SH-A (satellite handheld A) mode of DVB-SH communication standard, as shown in Fig. 2.1.

### 2.2.1 Transmitter

Transmitter of the DVB-SH physical layer consists of 'turbo encoding & QPSK/QAM modulation' and 'OFDM (orthogonal frequency division multiplexing) framing & transmission' blocks, as shown in Fig. 2.1. The DVB-SH frame of 12282 bits from 'transmitter data link layer' is fed to PCCC (parallel concatenated convolutional code) 'turbo encoder' that consist of two convolutional encoders and a turbo interleaver [26]. The transfer function for 'turbo encoder', compliant with DVB-SH standard, is given as

$$S(D) = \left(1, \frac{1 + D + D^2}{1 + D^2 + D^3}, \frac{1 + D + D^2 + D^3}{1 + D^2 + D^3}\right). \tag{2.1}$$

ARP (almost regular permutation) turbo interleaver [31] for the block length of 12282 bits is used and the mathematical expression for interleaved address $\Pi(i)$ is given as

$$\Pi(i) = (P_0 \times i + Q_0) \; mod \; N_{dvb}. \tag{2.2}$$

In the above expression, $N_{dvb}$=12282 bits, $P_0$=6125, $Q_0$=1225 and $i$={1, 2, 3, 4 ..... $N_{dvb}$}. At the transmitter, 'puncturing unit' processes the turbo encoded bits to achieve different code rates of 1/5, 2/9, 1/4, 2/7, 1/3, 2/5 and 1/2 for an efficient utilization of channel bandwidth [32]. The punctured data is bit interleaved for different code rates compliant to DVB-SH standard [19]. In order to perform the mapping optimization on the DVB-T (digital-video-broadcasting-terrestrial) frame purpose, the 'rate adaptation unit' is used for puncturing the bit interleaved block. After the rate adaptation process, the bit interleaved block is fed to a 'convolutional interleaver' which mitigates the burst error incurred by long term fading of mobile satellite channel that may immensely degrade the quality of service [30]. The 'bit demux' unit maps input bit stream for $M$-ary modulation schemes. DVB-SH in a SH-A mode of operation, incorporates modulation schemes such as QPSK (quadrature phase shift keying) and 16-QAM (quadrature amplitude modulation), thereby 'bit demux' unit maps the input bit stream into n=$\log_2 M$=2 for QPSK ($\because$ $M$=4) and n=4 for 16-QAM ($\because$ $M$=16), as shown in Fig. 2.1. The 'OFDM framing

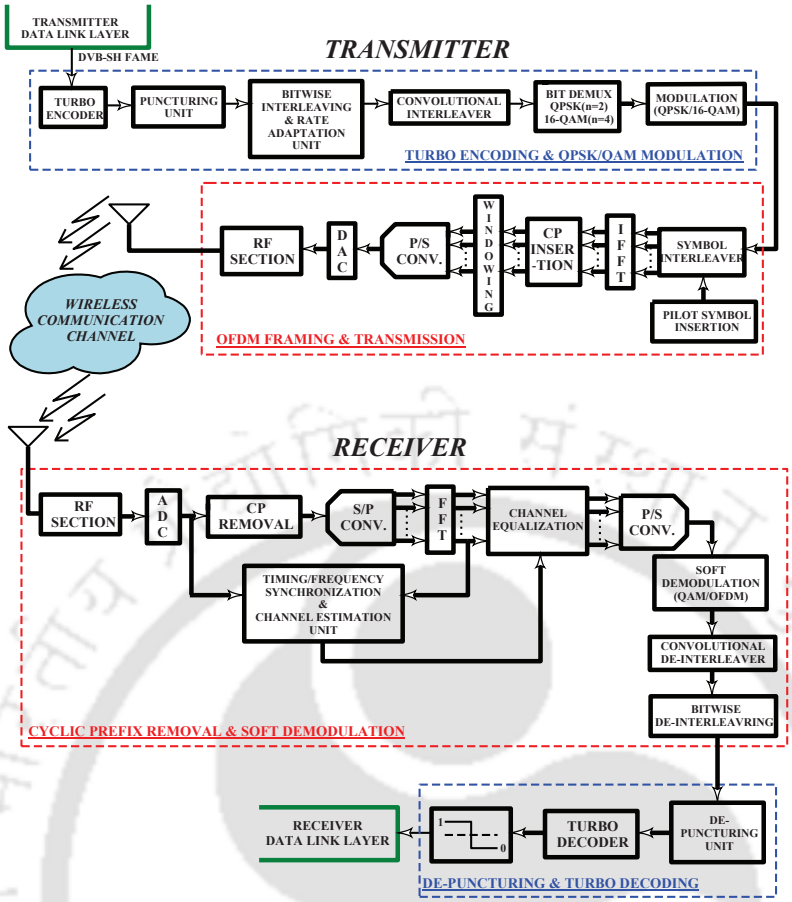


FIGURE 2.1: System level architecture for the physical layer of DVB-SH-A wireless communication standard.

& transmission' block performs IFFT (inverse fast Fourier transform), DAC (digital to analog conversion) and RF (radio frequency) transmission. Various IFFT sizes of 1K, 2K, 4K and 8K for OFDM multi carrier system are supported by DVB-SH standard depending on the bandwidth utilization [19]. The 'symbol interleaver' unit is fed with QPSK or 16-QAM modulated symbols and is used for mapping these modulated symbols with pilot symbols for different IFFT sizes. 'Symbol interleaver' unit incorporates pilot symbols with the modulated symbols to produce $N_f$ parallel symbols, where $N_f$ is the size of IFFT. Cyclic prefix is concatenated and windowed into different OFDM frames. The OFDM frames are fed to 'parallel to serial conversion' unit, then transformed to analog signals using DAC and finally, transmitted via RF transmitting antenna.

### 2.2.2 Receiver

In this work, we have simulated the physical layer model of DVB-SH standard in frequency selective fading environment. The faded analog signals from the channel are received at the antenna of 'RF receiver' unit and Gaussian noise is added to these analog signals, as shown in

Fig. 2.1. These faded plus noisy analog signals are converted into discrete values using ADC (analog to digital converter) and fed to the receiver base-band system. Timing recovery and channel estimation are being performed to estimate the frequency response of faded channel that can be used for channel equalization process to mitigate the effects of ISI (inter symbol interference). The CP (cyclic prefix) from each of the OFDM symbol is removed by 'CP removal' unit and then the serial stream of OFDM symbols are converted into parallel stream by 'serial to parallel conversion' unit in 'cyclic prefix removal & soft demodulation' block, as shown in Fig. 2.1. $N_f$-point FFT is performed for parallel symbols to extract the transmitted symbols which are modulated using multiple sub-carriers. In the 'channel equalization' block, Fourier transformed frequency domain symbols are equalized using the estimated frequency response of channel to mitigate the effect of ISI. Finally, the ISI free symbols are parallel to serial converted and soft demodulated using QPSK or 16-QAM demodulation scheme. The soft demodulation process generates *LLR* (logarithmic likelihood ratio) of a-priori probabilities for the transmitted bits. These *LLR* values are time and bit de-interleaved to produce an input bit stream for de-puncturing unit. The 'de-puncturing & turbo decoding' block constitutes turbo decoder as an error-correcting channel-decoder followed by de-puncturer unit. De-punctured *LLR* values of a-priori probabilities of the transmitted bits are fed to turbo decoder which is subjected to an iterative decoding process to generate the final *LLR* values of a-posteriori probabilities. Turbo decoder comprises of SISO (soft input soft output) units based on MAP algorithm, interleaver and de-interleaver [21]. Decoded a-posteriori probability *LLR* values of the transmitted bits $U_k$ can be computed using the received a-priori probability *LLR* values of systematic and parity bits as well as logarithmic a-priori extrinsic information generated in every iteration of the decoding process [2], and is given as

$$LLR_k = \ln\left(\frac{\sum\limits_{(s',s)=U_k=+1} \widehat{\alpha}_{k-1}(s') \times \widehat{\gamma}_k(s',s) \times \widehat{\beta}_k(s)}{\sum\limits_{(s',s)=U_k=-1} \widehat{\alpha}_{k-1}(s') \times \widehat{\gamma}_k(s',s) \times \widehat{\beta}_k(s)}\right), \tag{2.3}$$

where, $\widehat{\alpha}_k(s)$, $\widehat{\beta}_k(s)$ and $\widehat{\gamma}_k(s)$ are forward-state, backward-state and branch metrics, respectively, of each state $s$ at $k^{th}$ trellis stage. Finally, the turbo decoded *LLR* values are fed to the hard decision unit, which produces a sequence of 12282 bits for every DVB-SH frame. These decoded frames are passed to the upper data link layer of receiver side.

## 2.3 Performance and Throughput Analysis

This section of the chapter presents BER (bit error rate) performance analysis of turbo decoder compliant with DVB-SH communication standard. Simulations are carried out using the physical layer model of DVB-SH standard, as shown in Fig. 2.1. The BER performance analyses are carried out for various significant parameters those are crucial for designing efficient architecture of turbo decoder. In addition, throughput analyses for various configurations of the turbo decoder architecture, in order to meet the specification of 3G wireless communication standard, are presented in this section. Tradeoff between the throughputs, maximum operating frequencies, sliding window sizes and decoder iterations are also investigated. These simulation results impart significant information for understanding the turbo decoder performance in wireless communication standard and the process of selecting adequate design values for near-optimal BER performance.

### 2.3.1 Performance Analysis of Turbo Decoder in AWGN and Frequency Selective Fading Channels

For the DVB-SH standard in SH-A mode of operation, multi-carrier OFDM is associated with QPSK or 16-QAM modulation-schemes for each of the sub-carriers. Therefore, simulations are carried out for both the modulation-schemes with 1K point FFT and IFFT ($N_f$=1K) at receiver and transmitter sides respectively. An OFDM symbol consists of 534 QPSK or 16-QAM modulated symbols, 466 pilot symbols and 466 symbols of cyclic prefix. Pilot symbols are null values those are placed in the beginning and between 534 modulated symbols at the time of feeding 'IFFT' unit, as shown in Fig. 2.2. Additionally, 466 symbols of cyclic prefix are concatenated with Fourier transformed symbols, resulting in an OFDM symbol of 1466 symbols. Code rates of 1/2 and 1/3 are fixed for the simulations in AWGN and frequency selective fading channels, respectively, and eight iterations are performed while turbo decoding. In this simulation, an OFDM frame comprising of 12 and 23 OFDM symbols are used for 16-QAM and QPSK modulation-schemes respectively. For multi-path fading channel [27], simulations are carried out with the standard frequency-selective fading ITUR channel model [33]. The PDF (power delay profile) of this channel model is shown in Table 2.1. Fig. 2.3 shows the coding performance of turbo decoder for AWGN channel. It shows that the coding gain of turbo decoder for QPSK modulation, with respect to the performance of turbo decoder for 16-QAM, is 2.3 dB at a BER of $10^{-4}$. Additionally, the turbo coded QPSK modulation reaches a BER of $10^{-3}$ earlier than the un-coded QPSK by 3.2 dB. Similarly, at a BER of $10^{-2}$, turbo coded 16-QAM has a coding gain of 2.8 dB in comparison with the un-coded 16-QAM performance. On the other

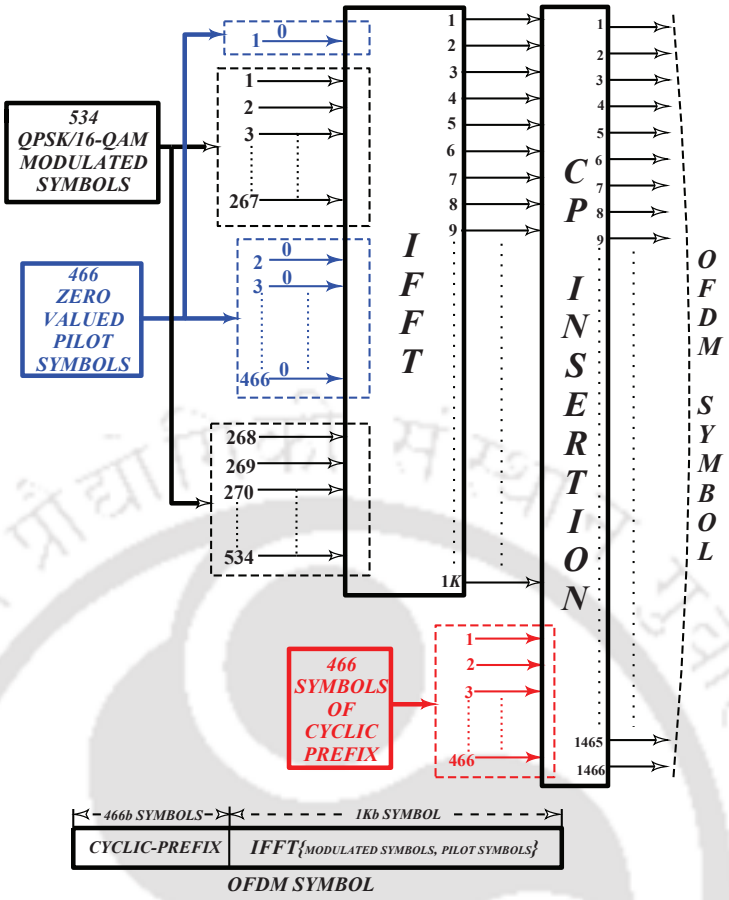


FIGURE 2.2: Organization of an OFDM symbol at the transmitter-side using 1K-IFFT, where QPSK/16-QAM modulated symbols are concatenated with pilot-symbols and cyclic-prefix.

side, BER performance of turbo code in ITUR fading channel model shows a coding gain of 6 dB at a BER of $10^{-4}$, for QPSK modulation in comparison with 16-QAM, as shown in Fig. 2.4. In AWGN and fading channel environments, OFDM with QPSK modulation has better coding performance than 16-QAM. However, rate of data transmission in case of 16-QAM is better than QPSK modulation because each of the 16-QAM symbol carries four bits of data per symbol and is double the value of QPSK modulation. The channel capacity of 2D (two dimensional) AWGN channel is derived by Shannon's limit theorem [1] and is given as

$$C = \log_2\{1 + r_c \times E_b/N_0\} \tag{2.4}$$

where $r_c$ is code rate and $E_b/N_0$ is signal-energy-per-bit to noise ratio. This is an ideal assumption which is valid for continuous and normally distributed inputs to the channel. However, such inputs for the channel do not exist in the practical communication-system. For such system of communication in which the $M$-ary modulation techniques such as BPSK (binary phase shift

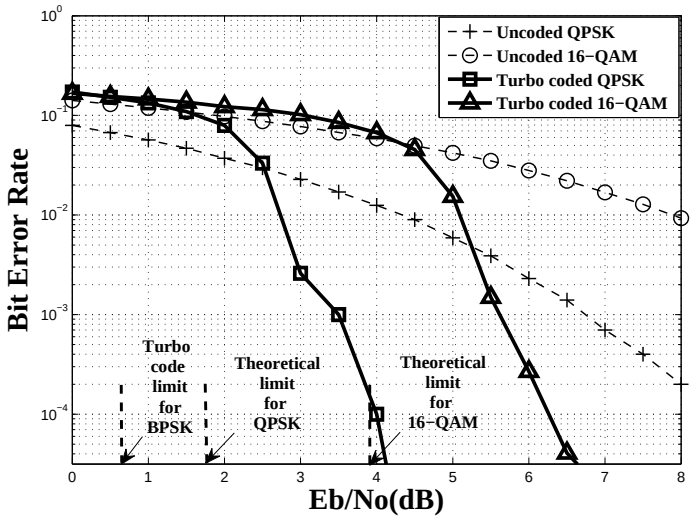


FIGURE 2.3: Coding performances of turbo code for DVB-SH-A standard in AWGN channel for a code rate of 1/2. The $E_b/N_0$ values, corresponding to a BER of $10^{-4}$ on the dashed vertical lines, represent their minimum theoretical limits.

TABLE 2.1: Power delay profile of ITUR (Vehicular A) model [33]

| Taps | Average power (dB) | Relative delay (nS) |
|---|---|---|
| 1 | 0.0 | 0 |
| 2 | -1.0 | 310 |
| 3 | -9.0 | 710 |
| 4 | -10.0 | 1090 |
| 5 | -15.0 | 1730 |
| 6 | -20.0 | 2510 |

keying)/ QPSK/ 16-QAM/ 64-QAM are used, the channel inputs are constrained to take on a finite set of values. Thereby, assuming 2D signal set and received vector, a constellation-constraint channel capacity is given as [34]

$$C = \log_2(M) + \frac{1}{M}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\sum_{i=1}^{M}\left[p(y_1,y_2|c_i)\times\log_2\left(\frac{p(y_1,y_2|c_i)}{\sum_{k=1}^{M}p(y_1,y_2|c_k)}\right)\right]dy_1\cdot dy_2 \qquad (2.5)$$

where $(y_1,y_2)$ and $(x_1,x_2)$ are arbitrary 2D received and transmitted points respectively; $c_i{=}(x_{1i},x_{2i})$ is $i^{th}$ symbol in the discrete set of $M$ input symbols. Subsequently, the conditional probability $p(y_1,y_2|c_i)$ can be expressed as [34]

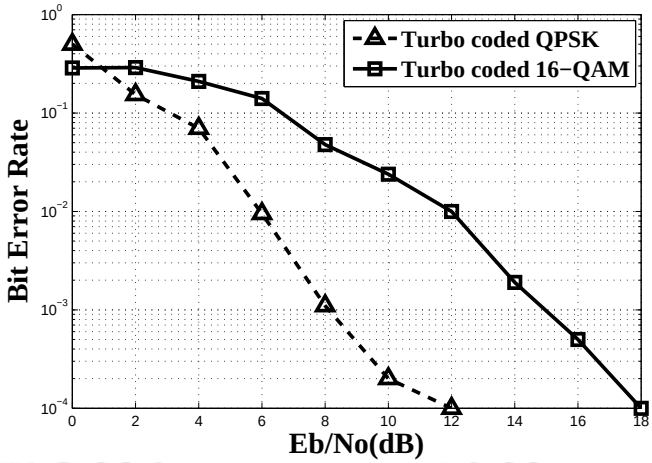


FIGURE 2.4: Coding performances of turbo code for DVB-SH-A standard in ITUR fading channel for a code rate of 1/3.

$$p(y_1, y_2|c_i = (x_{1i}, x_{2i})) = \frac{1}{2\pi \times \sigma_n^2} \exp\left[\frac{-1}{2 \times \sigma_n^2}\{(y_1 - x_{1i})^2 + (y_2 - x_{2i})^2\}\right] \tag{2.6}$$

where $\sigma_n^2$ is the noise variance. Based on this constellation-constraint channel capacity, a minimum theoretical value of $E_b/N_0$ required for the coded communication system with a code rate to achieve error-free communication can be determined. There is no close form expression of such minimum theoretical value of $E_b/N_0$ for QPSK and 16-QAM modulation schemes in AWGN channel environment. However, it can be evaluated numerically for various code rates [34, 35] and the same method has been followed in this chapter. In this subsection, theoretical limits of minimum $E_b/N_0$ values for a code rate of 1/2 to achieve an error-probability of $10^{-4}$ is numerically computed for QPSK and 16-QAM in AWGN channel environment as shown in Fig. 2.3. It shows that the minimum $E_b/N_0$ values for QPSK and 16-QAM for a code rate of 1/2 are 1.8 dB and 3.9 dB respectively. At a BER of $10^{-4}$, turbo code in AWGN environment for QPSK and 16-QAM modulations perform 2.2 dB and 2.4 dB away from their respective minimum theoretical limits. Performance of turbo code at a BER of $10^{-4}$ has $E_b/N_0$ value of 0.7 dB for BPSK modulation in AWGN channel [3] and has coding gains of 3.3 dB and 5.5 dB in comparison with the performances of turbo code for QPSK and 16-QAM, respectively, as shown in Fig. 2.3. The $E_b/N_0$ values, corresponding to a BER of $10^{-4}$ on the dashed vertical lines, represent their minimum theoretical limits.

### 2.3.2 Performance Analysis of Turbo Decoder for Different Decoding Iterations

Turbo decoding is an iterative process, in which extrinsic information are processed continuously by SISO units (or MAP decoders) in every iteration, to deliver near-optimal BER performance [2]. In this subsection, BER-performance analysis has been carried out for turbo code, which is

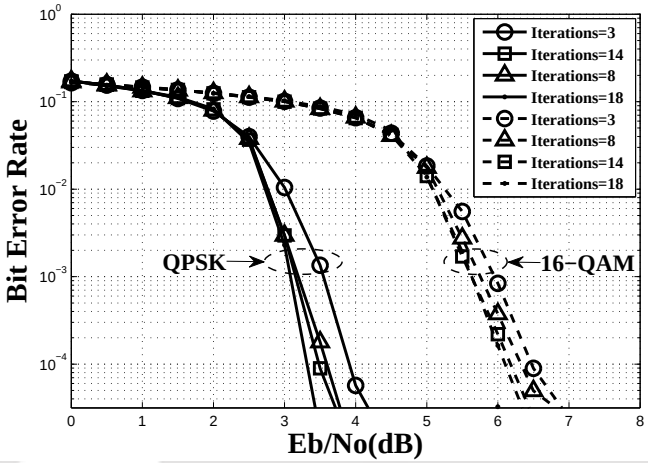


FIGURE 2.5: Coding performances of turbo code for different iterations in AWGN channel for a code rate of 1/2.

specifically used in DVB-SH wireless communication standard, for various decoding iterations in AWGN as well as fading-channel environments. This analysis provides optimum values of decoding iterations to be performed under different channel conditions. Thereby, it avoids redundant decoding iterations which have no significance in the BER performance of turbo code, thus improves system throughput and reduces power consumption from implementation perspective. The turbo decoder used in our simulations is based on max-log-MAP approximation [21]. The transmitted information-bits are turbo encoded with a code rate of 1/3 and each of the sub-carriers in OFDM is modulated using QPSK or 16-QAM modulation scheme. As shown in Fig. 2.5, for both QPSK and 16-QAM schemes, the coding performances delivered by turbo decoder in AWGN channel for 8, 14 and 18 iterations are identical at a BER of $10^{-2}$. However, at a BER of $10^{-4}$, a coding gain of less than 0.5 dB is seen for 18 iterations. Thereby, the turbo decoder has adequate coding performance even with 8 decoding iterations in AWGN channel. Coding performance of turbo decoder with QPSK modulation for various iterations in frequency selective ITUR fading-channel model for a code rate of 1/2 has been shown in Fig. 2.6. Unlike AWGN channel, the turbo decoder has a coding gain of 3 dB between 8 and 18 decoding iterations in fading channel environment, at a BER of $10^{-4}$. Thereby, the optimum

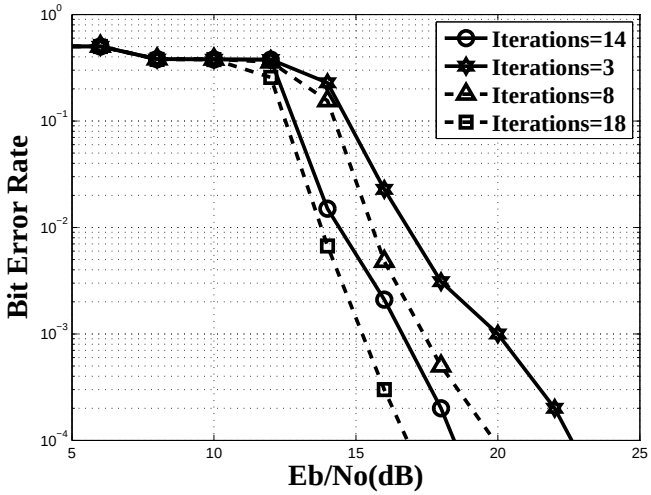


FIGURE 2.6: Coding performances of turbo code for different iterations in fading channel for a code rate of 1/2.

coding performance of turbo decoder for AWGN and fading channels can be achieved for 8 and 18 iterations, respectively, in the DVB-SH wireless communication system.

### 2.3.3 Performance Analysis of Turbo Decoder for Different Sliding Window Sizes

The SISO unit based on MAP algorithm is an integral part of turbo decoder [21]. Conventional MAP algorithm is based on trellis structure, and it involves forward and backward recursions of all trellis-stages during the decoding process. Length of trellis structure is proportional to block length which is specified by wireless communication standard. Since the turbo block length of DVB-SH standard is 12282 bits, the trellis length is huge and turbo decoder has to compute as well as store forward-state, backward-state and branch metrics for each of the trellis-stages. Thereby, large memory and excessive decoding delay are required to successively estimate a-posteriori-probability *LLR* values of the transmitted bits [22]. Sliding-window technique based MAP algorithm that mitigates such shortcoming has already been reported [36]. Sliding window technique segregates entire trellis length into different windows, where each of these windows includes fixed number of trellis stages. Forward-state and branch metrics are computed in conventional manner; whereas, computation of backward state metrics begins with the estimated values of these metrics from the successive sliding window. Accuracy of estimated backward state metric improves with the increase in sliding window size; this implies that the coding performance of turbo decoder is proportional to sliding window size. Analogous to the dependency of system throughput with decoding iterations, sliding window size is also inversely proportional

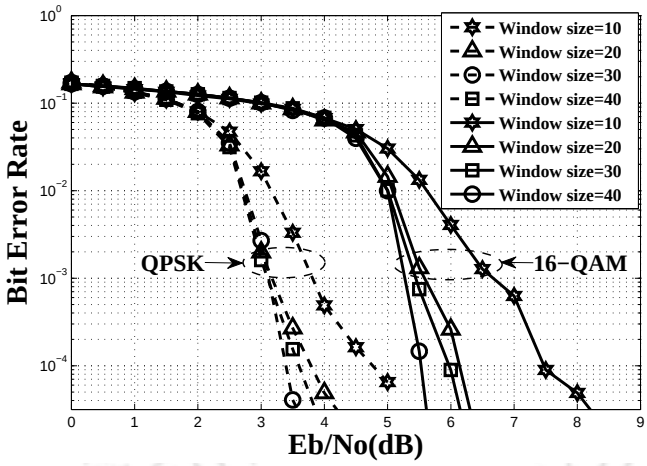


FIGURE 2.7: Coding performances of turbo code for different sliding window sizes in AWGN channel for a code rate of 1/2.

to the system throughput. Therefore, it is of major concern to determine the optimum value of sliding window size with which the turbo decoder can deliver near-optimal BER performance and achieves adequate system-throughput. Thereby, we have carried out BER-performance analysis of turbo decoder which is compliant with DVB-SH standard for different sliding window sizes in both AWGN as well as fading channel environments. Fig. 2.7 shows that the coding performance

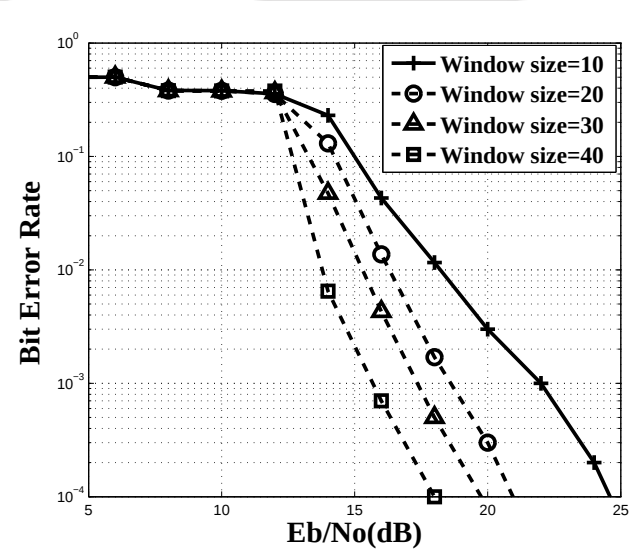


FIGURE 2.8: Coding performances of turbo code for different sliding window sizes in fading channel for a code rate of 1/2.

of turbo decoder with sliding window sizes of 20, 30 and 40 have similar BER performances at a code rate of 1/2 using QPSK as well as 16-QAM modulation-schemes. Unlike the coding performances of turbo decoder for these sliding window sizes, turbo decoder with a sliding window size of 10 has degraded coding performance of at least 1.5 dB at a BER of $10^{-4}$ in both the cases

of QPSK and 16-QAM modulation-schemes. Hence, it is suitable to design a turbo decoder of sliding window sizes 20 or 30 or 40 for an optimum BER performance in AWGN channel environment. The coding performance of turbo decoder for different sliding window sizes with QPSK modulation scheme for a code rate of 1/2 in frequency selective fading channel is shown in Fig. 2.8. It shows that the turbo decoder with sliding window size of 40 has larger coding gains of 2 dB, 3 dB and 7 dB with respect to the decoder with sliding window sizes of 30, 20 and 10, respectively, in the frequency selective ITUR fading channel. Thereby, it is necessary to design efficient turbo decoder with a sliding window size of 40 for such fading channel.

### 2.3.4 System-Throughput Analysis for Different Architectural Configurations of Turbo Decoder

As mentioned earlier, the coding performance of turbo code is proportional to number of iterations those are performed while turbo decoding [2]. Unlike coding performance, system throughput is inversely proportional to decoding iterations because it increases latency for decoding transmitted bits, and it simultaneously boasts power consumed by decoder [37]. In addition, the system latency is also proportional to sliding window size and is key factor which affects system throughput. It is essential to understand the tradeoffs among system throughput, number of decoding iterations and sliding window sizes for designing a high-level architecture of turbo decoder that is suitable for physical layer of wireless communication standard. Specifically, DVB-SH is 3G wireless communication standard and supports system throughput in the range of 100-300 Mbps [19]. In order to achieve such throughput with an optimum coding performance, there are various turbo decoder configurations. Conventional turbo decoder configuration is a non-parallel radix-2 architecture [28] which has considerably less throughput as compared to the specification of 3G communication standards. The state-of-the-art configuration of turbo decoder has radix-4 with parallel architecture [29] to meet these 3G-specifications. Mathematical expression for the system throughput $\theta$ is given as [37]

$$\theta = \frac{N \times f_{max} \times P}{2 \times I(\widehat{N} + L_{siso} \times P)} \tag{2.7}$$

where $N$ represents turbo block length for DVB-SH standard ($N$=12282 bits), $P$ represents number of SISO units, $L_{siso}$ represents latency of SISO unit ($L_{siso} = 2 \times SW$) such that $SW$ denotes sliding window size, $I$ represents number of decoding iterations, $\widehat{N}$ is equal to $N$ for binary turbo decoder and is $N/2$ for duo-binary turbo decoder ($\widehat{N}$=$N$ in this work). In addition, $f_{max}$ represents maximum operating frequency of turbo decoder. Fig. 2.9 shows the plot of system throughputs of radix-2 turbo decoder configuration as a function of different decoding-iterations

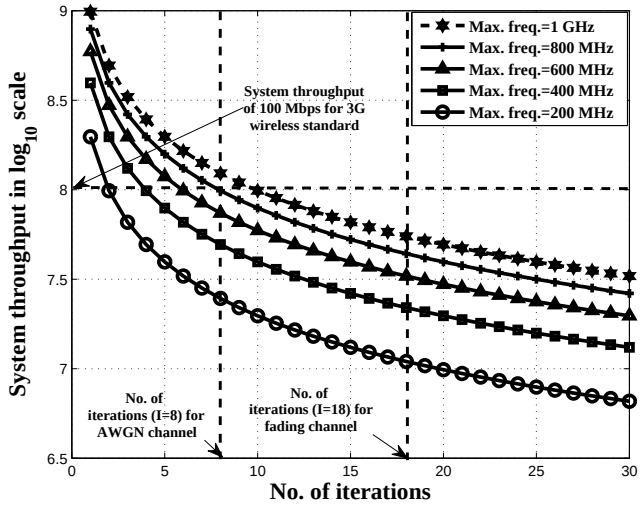


FIGURE 2.9: Plots for the system throughputs versus number of iterations at different frequencies for turbo decoder with radix-2 configuration. Intersecting points of two vertical dash lines with the plots indicate system throughputs (along y-axis) which can be achieved with the iterations (along x-axis) of 8 and 18 for AWGN and fading channels respectively.

for various frequencies of operation. Here, the value of $P$ is taken as 2 because it is non-parallel configuration. Since the optimum coding performance is achieved for a sliding window size of 40 as mentioned previously, the value of $L_{siso} = 2{\times}SW{=}80$. In case of AWGN and fading channels, previous subsection has shown that optimum coding performance can be achieved with the decoding iterations of 8 and 18 respectively. Fig. 2.9 shows that the throughput of 100 Mbps for 8 iterations can be achieved at the frequencies of 800 MHz and 1GHz operating frequencies for AWGN channel environment. However, 100 Mbps throughput with 18 iterations for the fading channel is not achievable at any of these frequencies. Thereby, it is necessary to realize radix-4 parallel-configuration of turbo decoder to achieve specified throughput of 3G wireless communication standard. A parallel radix-4 architecture [29] of turbo decoder is configured with multiple SISO units in parallel and hence value of $P$ is greater than two for the computation of throughout ($\theta$). Subsequently, two trellis stages are processed in each clock cycle; therefore, the throughput of radix-4 configuration is twice the throughput achieved by radix-2 architecture ($\theta_{rad-4} = 2 \times \theta_{rad-2}$). Fig. 2.10 shows the plots of system throughputs for radix-4 parallel-configurations of turbo decoder for $P$=4, $P$=8, $P$=12 and $P$=16. For the configurations $P$=16 and $P$=12, the throughputs are greater than 100 Mbps for all the given frequencies of operation. Thereby, turbo decoder configured with 16 or 12 parallel SISOs can be used for DVB-SH standard. For $P$=8, turbo decoder has adequate throughput for all the frequencies in AWGN channel environment. However, this decoder cannot achieve required throughput at operating frequency of 200 MHz in fading channel. On the other hand, $P$=4 parallel-configured turbo

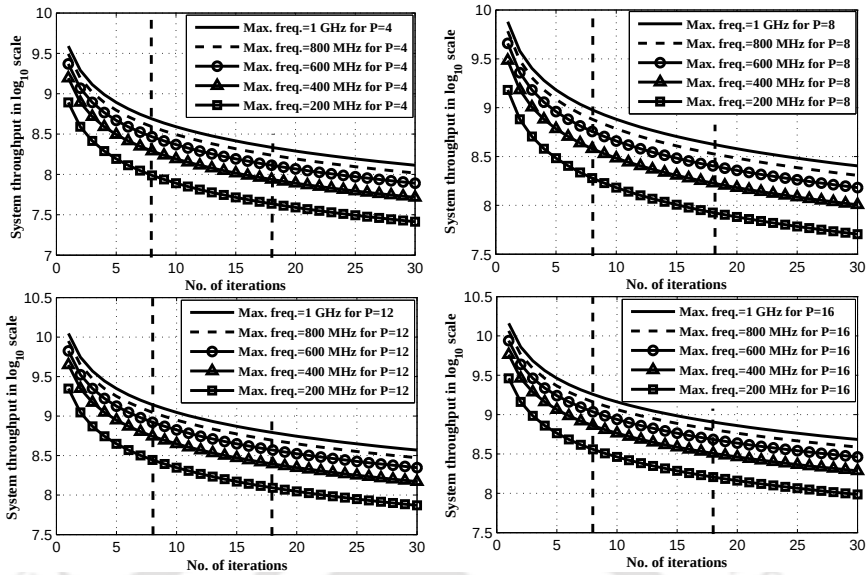


FIGURE 2.10: Plots of the system throughputs versus number of iterations at different frequencies for turbo decoders with radix-4-parallel configurations.

decoder meets throughput requirement for AWGN channel at all the frequencies and it fails to achieve required throughput at 200 MHz and 400 MHz for the fading channel environment.

### 2.3.5 Performance Analysis of Turbo Decoder for Different MAP Algorithms

Conventional MAP algorithm involves complex mathematical operations, such as exponential, division and multiplication [18]. Logarithmic transformation of such algorithm has been suggested in the literature to overcome such complex computations and has made its implementation simpler [21, 38]. Logarithmic MAP algorithm simplifies the computation of state metric for a given state in each of the trellis stage using state metrics and branch metrics of the previous states. Let the logarithmic forms of state metrics for previous states be $A1'$ and $A2'$, and their respective branch metrics be $Y1$ and $Y2$. Thereby, state metric $A$ of the present state can be computed using max-log-MAP algorithm as [21]

$$A = max(A1', A2'). \tag{2.8}$$

Similarly, the computation of state metrics for log-MAP [21] and MAP algorithm based on Maclaurin series expansion [38] can be computed as

$$A = max(A1', A2') + \ln\left(1 + e^{-|A1'-A2'|}\right) \ and \tag{2.9}$$

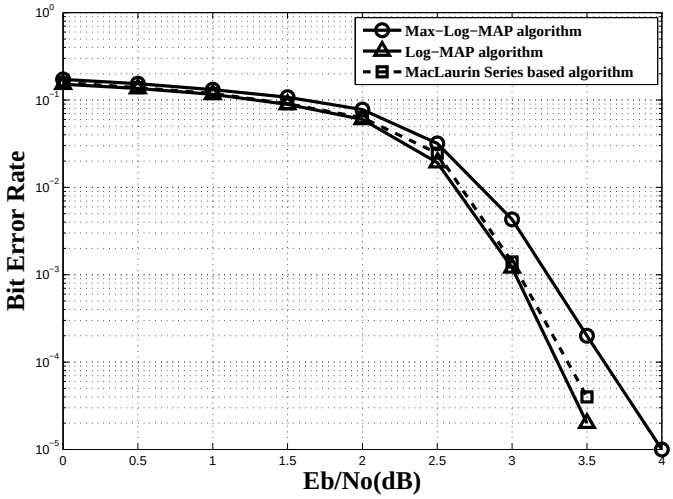


FIGURE 2.11: Coding performances of turbo code for different logarithmic MAP algorithms in AWGN channel for a code rate of 1/2.

$$A = max(A1', A2') + max\left(0, \ln(2) - 0.5|A1' - A2'|\right) \tag{2.10}$$

respectively. In this subsection, coding performances of turbo code for DVB-SH standard with these logarithmic MAP algorithms are presented. The simulations are carried out using OFDM in which each subcarrier is QPSK modulated and the transmitted bits are turbo encoded with a code rate of 1/2 for AWGN and fading channels. Fig. 2.11 shows the coding performance of

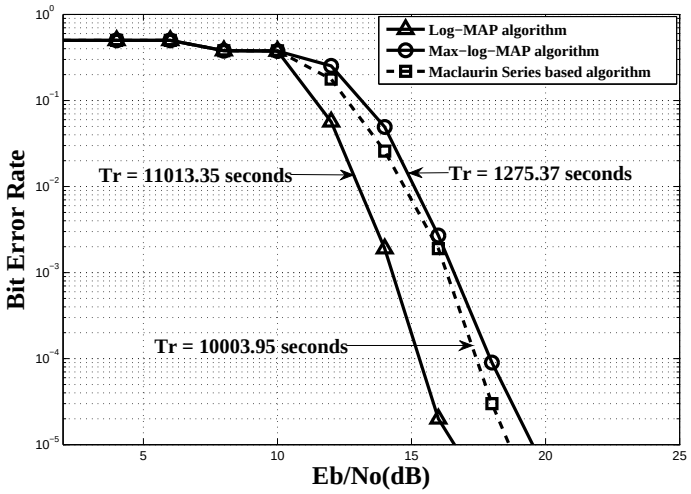


FIGURE 2.12: Coding performances of turbo code for different logarithmic MAP algorithms with the CPU running time (Tr) in fading channel for a code rate of 1/2.

various logarithmic MAP algorithms in AWGN channel environment. Log-MAP algorithm has the best BER performance with coding gains of approximately 0.3 dB and 0.1 dB in comparison with max-log-MAP and Maclaurin series based MAP algorithms, respectively, at a BER of

$10^{-4}$. Hence, for AWGN channel, it appears that the Maclaurin series approximation is very attractive (may be even preferred) design alternative to log-MAP, since it gives almost the same performance for only a fraction of the complexity. Moreover, Maclaurin series approximation delivers better performance than max-log-MAP approximation, as shown in Fig. 2.11. Similarly, coding performance of these logarithmic algorithms is also carried out for frequency selective fading channels, as shown in Fig. 2.12. In addition, the running time for each of these algorithms in a 64-bit CPU (central processing unit) is also presented. Fig. 2.12 shows that the log-MAP algorithm, at a BER of $10^{-5}$, has coding gains of 2 dB and 3 dB in comparison with Maclaurin series based MAP and max-log-MAP algorithms, respectively, in the fading channel environment. However, the log-MAP approximation has largest CPU running time of 11013.35 seconds in comparison with Maclaurin series and max-log-MAP approximations. The CPU running-time values of Maclaurin and max-log-MAP approximations are 10003.95 seconds and 1275.37 seconds, respectively, as shown in Fig. 2.12. Therefore, for a specific application, suitable logarithmic algorithm which provides satisfactory performance can be chosen.

### 2.3.6 Performance Analysis of Turbo Decoder for Different Code Rates

Code rate is a significant parameter in the design of turbo decoder from algorithmic as well as architectural perspectives. From an algorithmic aspect, code rate is proportional to error-rate performance of turbo code as it delivers better performance with smaller value of code-rate; since, there is more number of parity bits for such lower code-rate values. In an architectural domain, code rates are responsible for the design of encoder, puncturing and de-puncturing units in the communication system. DVB-SH wireless communication standard supports various code rates of 1/2, 1/3, 2/5, 1/4, 1/5, 2/7 and 2/9, and these code rates are possible to realize with puncturing unit [32]. The architectures of turbo encoder and puncturing unit compliant with DVB-SH standard are shown in Fig. 2.13. The input bit stream to turbo encoder is represented by $U_k$ and the encoded bit pattern [$X$, $Y_0$, $Y_1$, $X'$, $Y_0'$, $Y_1'$] is fed to the puncturing unit. The puncturing pattern for encoded bit stream is taken from DVB-SH standard implementation guidelines [19]. Finally, the punctured output is represented as $U_p$, as shown in Fig. 2.13. The coding performance of turbo code is inversely proportional to the value of code rate, as discussed earlier in this section. Transmission takes place with different code rates depending on channel condition; for example, code rate below 1/3 or 2/7 are not very suitable for the pure terrestrial environment. BER performances of turbo code are analyzed for various code rates using OFDM with QPSK modulation in AWGN channel environment where the BER plot of minimum code rate has the best performance, as shown in Fig. 2.14. On applying the numerical methods as mentioned in section-2.3.1, theoretical limits of minimum $E_b/N_0$ values for all the code rates

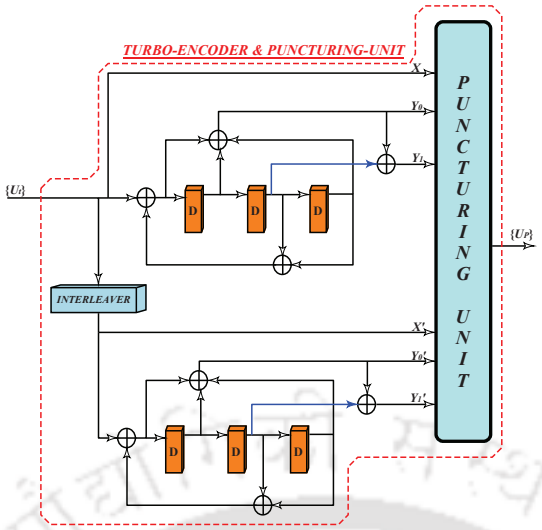


FIGURE 2.13: Architectures of turbo-encoder and puncturing-unit compliant to DVB-SH wireless communications standard [19].

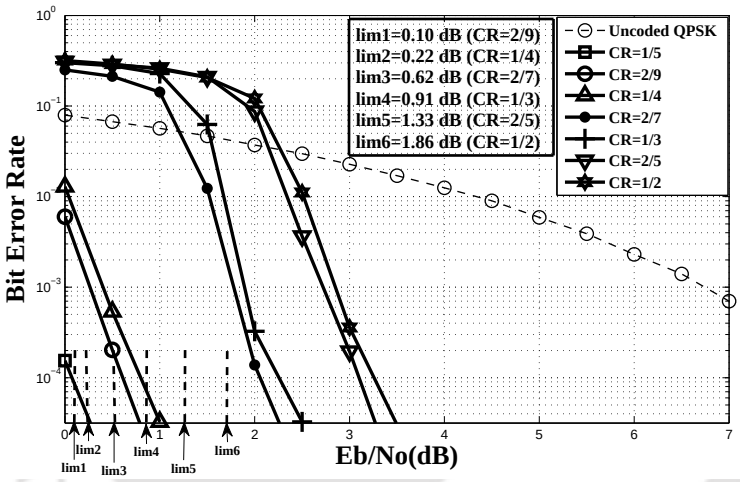


FIGURE 2.14: Coding performances of turbo code for different code rates in AWGN channel. The $E_b/N_0$ values, corresponding to a BER of $10^{-4}$ on the dashed vertical lines, represent their minimum theoretical limits.

of DVB-SH standard to achieve the least error-probability of $10^{-4}$ are computed for QPSK modulation in AWGN channel environment. Fig. 2.14 indicates these minimum values for all the code rates except for a code rate of 1/5 in which case the minimum $E_b/N_0$ is -0.425 dB. At a BER of $10^{-4}$, these minimum values increase with the code rate; for example, minimum $E_b/N_0$ values for the code rates 1/2 and 2/5 are 1.86 dB and 1.33 dB, respectively, as shown in Fig. 2.14. The theoretical limits of minimum $E_b/N_0$ values for a particular BER of $10^{-4}$ are indicated by lim1, lim2, lim3, lim4, lim5 and lim6 on the vertical dashed lines for various code rates.

## 2.4 Summary

In this chapter, coding performances of turbo decoder which is compliant to the physical layer of DVB-SH wireless communication standard for AWGN and frequency selective fading channels were presented. The modulation of transmitted bits was carried out with OFDM technique, incorporating 1K-FFT where each subcarrier was modulated using QPSK or 16-QAM modulation-scheme. Performance analysis of turbo decoder for various decoding iterations of 3, 8, 14 and 18 as well as the sliding window sizes of 10, 20, 30 and 40 were investigated for both the channel-environments. Subsequently, discussion on the values of these design metrics to achieve near-optimal error-rate performance was discussed. The optimization of system throughput for turbo decoder based on the decoding iteration and sliding window size for various processor speeds ranging from 200 MHz to 1 GHz were carried out. Such an analysis was presented for non-parallel radix-2 as well as parallel radix-4 configuration of turbo decoder to meet the system throughput specification of 3G wireless communication standard ranging from 100 Mbps to 300 Mbps. The coding performance of turbo decoder based on max-log-MAP, log-MAP and Maclaurin series based algorithms were studied for both the channel conditions. Simultaneously, the running time for each of these algorithms in a 64 bit processor was presented for comparison. Finally, the coding performances of turbo decoder for various code rates of 1/5, 2/9, 1/4, 2/7, 1/3, 2/5 and 1/2 were carried out. The presented work is specific to DVB-SH standard; however, it has derived a framework for designing an efficient turbo decoder and its dependency on various design metrics for any wireless communication standard.

# Chapter 3

# Comparative Study of MAP Algorithms and Design Exploration of Turbo Decoder

## 3.1 Introduction

Motivation behind the work presented in this chapter is to study VLSI design aspects of turbo decoder for high-speed application, specifically based on various simplified MAP algorithms. As we have already mentioned earlier, high-speed data processing and energy saving are the major concerns, while designing architectures for the present era of advance wireless communication systems. In the digital baseband of recent wireless communication standards such as LTE-A, DVB-SH, 3GPP-LTE, WCDMA (wideband code division multiple access), Mobile-WiMAX, HSDPA (high speed downlink packet access) [19, 28, 39], turbo decoders are being extensively used to deliver excellent BER performances [40]. In such turbo decoder, SISO unit has significant impact on error-rate performance as well as speed of data processing and energy consumption. However, mathematically complex MAP algorithm for this unit has adverse effect on the VLSI implementation-process of turbo decoders. Over the years, contributions which are

intended to simplify such complex algorithm have been reported in the literature [38, 41–46]. On the other side, comparative study of such simplified MAP algorithms is an essential procedure to decide on an algorithm with near-optimal BER performance; specially, targeting high speed application is rather important in the present-day scenario. In addition, analysis of hardware requirement as a function of various design metrics is necessary to perform for a cost effective VLSI implementation. With such motivations, this work presents optimized architectures for the approximations of simplified MAP algorithms based on MSE (Maclaurin series expansion) [38] and reduced forms of PWLA (piece wise linear approximation) [46]. Analysis of the critical path delay for each of these architectures is also carried out. Subsequently, BER performances of turbo code using these simplified MAP algorithms based on MSE and PWLA are compared. Thereafter, an algorithm with shortest critical path delay and near-optimal BER performance is chosen for an implementation of turbo decoder for high speed application. Architecture for SISO unit of turbo decoder based on the chosen simplified MAP algorithm is presented. Brief discussion is carried out for the QPP (quadratic permutation polynomial) interleaver, which has been used in the VLSI-implementation of turbo decoder [31]. In addition, a quantitative model for the memory requirement of SISO unit as a function of various design metrics such as sliding window size, number of trellis states and data width of internal metrics is developed. Eventually, a radix-2 non-parallel architecture of turbo decoder is designed by integrating a SISO unit with QPP interleaver and its ASIC (application specific integrated circuit) implementation in 130 nm CMOS technology node is carried out. Moreover, achievable throughput by various configurations of turbo decoder and its application for suitable wireless communication standards has been discussed. Then, the implementation result of turbo decoder architecture is compared with the reported works.

However, such comparison showed that the conventional turbo decoder with non-parallel architecture is incapable of achieving higher data-rates over 300 Mbps and 1 Gbps, as per the specifications of 3G and 4G wireless communication standards respectively [29, 47]. On the other hand, turbo decoder with parallel architecture of multiple SISO units can achieve such data-rates [48]. Recently, various contributions have been reported for the design of such parallel turbo decoders [29, 49–52]. Fig. 3.1 shows a conventional turbo decoder with parallel architecture for higher data-rate applications. Soft-demodulated input soft-values of received bits, at the receiver-side of communication system, are stored in the stack of memories where each of them is represented as *MEM* along the input-side of decoder. As shown in Fig. 3.1, outputs of *MEMs* are linked with multiple SISO units via *ICNWs* (inter connecting networks) which route the input soft-values from *MEMs*, either sequentially or pseudo-randomly based on the interleaved addresses, to their respective SISO units. Extrinsic-information produced by these SISO units are

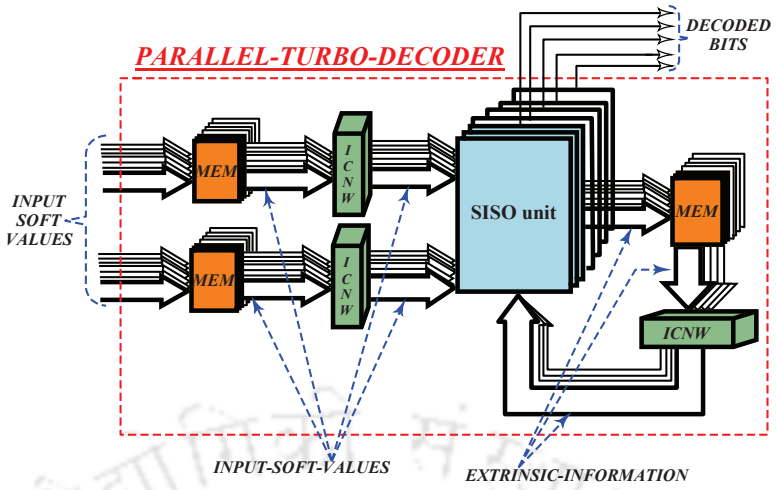


FIGURE 3.1: A conventional parallel-architecture of turbo decoder which iteratively processes input-soft-values to produce decoded-bits.

stored in *MEMs* after processing the input soft-values; finally, the extrinsic-information outputs from these *MEMs* are fed back to SISO units via *ICNW*. These information are used as a-priori-probabilities in the iterative process of decoding, as shown in Fig. 3.1. Though the parallel turbo decoder can achieve higher data-rate, it demands huge amount of hardware resources. Thereby, next objective of our work presented in this chapter is to scale-down the hardware requirements of parallel turbo decoders by reducing the memory required for storing forward state metrics and branch metrics in each of the SISO units of parallel turbo decoder. There are some reported works in the literature with similar motivation [53–56]. A memory reduction technique based on metric compression using non-uniform quantization and Walsh-Hadamard transform has been presented in [56]. Another approach is based on low power trace back of MAP based duo-binary turbo decoders [55]. Reduction of branch-metrics memory and scheduling the back-trace of MAP algorithm are performed in [54] and [53] respectively. Our contributions in the design of memory-reduced architecture for parallel turbo decoder are as follows.

- A new method of estimating the values of backward state metrics which initiate the back-trace in MAP algorithm is presented. Furthermore, a branch-metric reformulation technique is provided to reduce the memory requirement.
- Architecture of SISO unit based on the suggested techniques is presented. Scheduling of this new SISO unit and a comparative analysis of memory consumption by proposed and conventional parallel turbo decoders are carried out.
- Simulations for the BER performances of MAP and parallel turbo decoders are accomplished. An overall hardware saving of the proposed turbo decoder with parallel architecture has been estimated.

This chapter has been further organized as follow. Section-3.2 presents brief discussion on the reported simplified MAP algorithms and their architectural as well as BER performance comparisons. Turbo decoder architecture and its integral parts, such as SISO unit based on a simplified MAP algorithm and QPP interleaver are presented in section-3.3. Design procedure for the VLSI implementation of suggested turbo decoder as well as its applications and comparison with the reported works are included in section-3.4. In section-3.5, discussion on the mathematical background of BCJR algorithm, the suggested RSWMAP (reduced sliding window maximum a-posteriori probability) algorithm and branch-metric reformulation technique are carried out. Section-3.6 presents architectural and scheduling details of the SISO unit architecture. In section-3.7, BER performance evaluation of the SISO-unit and parallel-turbo-decoder, and implementation trade-off are presented. Finally, this chapter is summarized in section-3.8.

## 3.2 Comparative Study

In this section, comparative analysis of architectures and BER performances based on the simplified MAP algorithms are carried out. Additionally, architecture of MAP algorithm based turbo decoder, which is best suited for high-speed applications and exhibits near-optimal BER performance, has been discussed for VLSI implementation.

### 3.2.1 Overview of Simplified MAP Algorithms

Conventional logarithmic MAP algorithm uses Jacobian logarithm for computing forward/backward state metrics and *LLR* values of a-posteriori probabilities [21]. According to Jacobian logarithm, mathematical equation involving logarithmic and exponential functions can be approximated as

$$\ln(e^{\psi_1} + e^{\psi_2}) = \widehat{\max}(\psi_1, \psi_2) = \max(\psi_1, \psi_2) + \ln(1 + e^{-|\triangle|}) \tag{3.1}$$

where $\psi_1$ and $\psi_2$ are arbitrary variables and $\triangle$ is a difference value of $\psi_1 - \psi_2$. In the MAP algorithm, forward state metric for $k^{th}$ trellis stage at a given state $s_0$, can be computed as

$$\alpha_k(s_0) = \widehat{\max}\left[\{\alpha_{k-1}(s_0') + \gamma_k(s_0', s_0)\}, \{\alpha_{k-1}(s_1') + \gamma_k(s_1', s_0)\}\right] \tag{3.2}$$

where $\alpha_{k-1}(s'_0)$ and $\alpha_{k-1}(s'_1)$ are the forward state metrics for $s'_0$ and $s'_1$ states, respectively, at $(k\text{-}1)^{th}$ trellis stage. Similarly, $\gamma_k(s'_0, s_0)$ and $\gamma_k(s'_1, s_0)$ are the branch metrics associated with state transitions $s'_0$-to-$s_0$ and $s'_1$-to-$s_0$ respectively. In general, for a code length of $n$ and state transition $s'_x$-to-$s_y$ from $(k\text{-}1)^{th}$ to $k^{th}$ trellis stages, branch metric expression is given as

$$\gamma_k(s'_x, s_y) = \frac{U_k \cdot L(U_k)}{2} + \frac{Lc}{2}\{x \cdot X + x_{p1} \cdot X_{p1} + x_{p2} \cdot X_{p2} + ..... + x_{pn} \cdot X_{p(n-1)}\} \tag{3.3}$$

where $x$ and $x_{pi}$ $\forall$ $i$={1, 2, 3, 4 ..... $n$-1} are systematic and parity bits, respectively, such that $x$∈{+1,-1} and $x_{pi}$∈{+1,-1}. Similarly, $X$ and $X_{pi}$ $\forall$ $i$={1, 2, 3, 4 ..... $n$-1} are the received soft-values of systematic and parity bits respectively. $L(U_k)$ is a-priori-probability information and $Lc$ is channel reliability measure, which is proportional to fading amplitude as well as noise variance [21]. Similar to (3.2), expression of backward state metric for $k^{th}$ trellis stage at a given state $s_0$ can be expressed as

$$\beta_k(s_0) = \widehat{\max}\left[\{\beta_{k+1}(s''_0) + \gamma_k(s''_0, s_0)\}, \{\beta_{k+1}(s''_1) + \gamma_k(s''_1, s_0)\}\right] \tag{3.4}$$

where $s''_0$ and $s''_1$ are the states at $(k+1)^{th}$ trellis stage. MAP algorithm uses forward-state, backward-state and branch metrics of $(k\text{-}1)^{th}$, $k^{th}$ and all the state-transitions from $(k\text{-}1)^{th}$ to $k^{th}$ trellis stages, respectively, to compute a-posteriori $LLR$ value at $k^{th}$ trellis stage and is given as

$$LLR \approx \widehat{\max}_{(s',s)\Rightarrow U_k=1}\left[\alpha_{k-1}(s') + \gamma_k(s', s) + \beta_k(s)\right] - \widehat{\max}_{(s',s)\Rightarrow U_k=0}\left[\alpha_{k-1}(s') + \gamma_k(s', s) + \beta_k(s)\right] \tag{3.5}$$

where $\widehat{\max}_{(s',s)\Rightarrow U_k=1/0}[\cdot]$ is a function which obtains $\widehat{\max}$ value among the sums of forward-state, backward-state and branch metrics for each of the state transitions of the transmitted bit $U_k$ equals 1 or 0. In the simplified MAP algorithm, mathematical representation of the correction-factor, that is given as $\ln(1 + e^{-|\triangle|})$ in (3.1), is approximated with an expression which is implementation friendly. Such simplified versions of MAP algorithm are well established in the literature and are summarized in Table 3.1. A recently proposed simplified MAP algorithm based on PWLA has shown promising results in terms of BER performance and from VLSI-implementation perspective [46, 58]. The number of terms (denoted by $r$) involved in PWLA of $\widehat{\max}(\Psi_1, \Psi_2)$ is proportional to the BER performance and these approximations for $r$=3 and $r$=4 are shown in Table 3.1. From the literature [46, 58], it has been shown that the simplified MAP algorithm based on PWLA with $r$=4 has a performance

TABLE 3.1: Simplified MAP algorithms of various reported works.

| *Works* | *Approximation for* $\widehat{\max}$, $\widehat{\max}(\Psi_1,\Psi_2)=\max(\Psi_1,\Psi_2)+\ln(1+e^{-\lvert\Delta\rvert})$ *and* $\Delta=(\Psi_1-\Psi_2)$ |
|---|---|
| [21] | $\max(\Psi_1,\Psi_2)$ |
| [41] | $\begin{cases}\max(\Psi_1,\Psi_2)+3/8, & \text{if } \lvert\Delta\rvert<2\\ \max(\Psi_1,\Psi_2), & \text{otherwise}\end{cases}$ |
| [44] | $\max(\Psi_1,\Psi_2)+\max[\{\ln(2)-\lvert\Delta\rvert/4\},0]$ |
| [45] | $\begin{cases}\max(\Psi_1,\Psi_2)+(-\lvert\Delta\rvert/2+0.7), & \text{if } \lvert\Delta\rvert=[0,0.5)\\ \max(\Psi_1,\Psi_2)+(-\lvert\Delta\rvert/8+0.375), & \text{if } \lvert\Delta\rvert=[1.6,2.2)\\ \max(\Psi_1,\Psi_2)+(-\lvert\Delta\rvert/16+0.2375), & \text{if } \lvert\Delta\rvert=[2.2,3.2)\\ \max(\Psi_1,\Psi_2)+(-\lvert\Delta\rvert/4+0.575), & \text{if } \lvert\Delta\rvert=[0.5,1.6)\\ \max(\Psi_1,\Psi_2)+(-\lvert\Delta\rvert/32+0.1375), & \text{if } \lvert\Delta\rvert=[3.2,4.4)\\ \max(\Psi_1,\Psi_2), & \text{if } \lvert\Delta\rvert=[4.4,+\infty)\end{cases}$ |
| [43] | $\max(\Psi_1,\Psi_2)+\max(5/8-\lvert\Delta\rvert/4,0)$ |
| [42] | $\begin{cases}\max(\Psi_1,\Psi_2)+\{\ln(2)+\lvert\Delta\rvert/2\}, & \text{if } \lvert\Delta\rvert<2\times\ln(2)\\ \max(\Psi_1,\Psi_2), & \text{otherwise}\end{cases}$ |
| [57] | $\max(\Psi_1,\Psi_2)+\{\ln(2)\times 2^{-\lvert\Delta\rvert}\}$ |
| [38] | $\max(\Psi_1,\Psi_2)+\max\{0,(\ln 2-0.5\times\lvert\Delta\rvert)\}$ |
| [46] | $\begin{cases}\max\{\Psi_1,0.5\times(\Psi_1+\Psi_2+1),\Psi_2\}, & \text{for } r^{\dagger}=3\\ \max\{\Psi_1,\varphi_1(\Psi_1,\Psi_2)^{\ddagger},\varphi_2(\Psi_1,\Psi_2)^{\S}\}, & \text{for } r^{\dagger}=4\end{cases}$ |

‡: $\varphi_1(\Psi_1,\Psi_2)=0.271\times\Psi_1+0.729\times\Psi_2+0.584$;

§: $\varphi_2(\Psi_1,\Psi_2)=0.729\times\Psi_1+0.271\times\Psi_2+0.584$;

†: $r$=No. of terms for PWL approximation.

degradation of only 0.03 dB in comparison with the conventional log-MAP (logarithmic-MAP) algorithm from (3.1). Subsequently, it delivers identical BER performance with respect to simplified MAP algorithms existing in literature [38, 41–45, 57]. Approximation of $\widehat{\max}$ for PWLA based simplified MAP algorithm for $r$=3 and $r$=4, as shown in Table 3.1, are further reduced to more simplified approximations. Thereby, these approximations for $r$=3 and $r$=4 are represented as $\widehat{\max}(\psi_1,\psi_2)\approx\max_{red1}=\max\{\max(\Psi_1,\Psi_2),(\Psi_1+\Psi_2+1)/2\}$ and $\widehat{\max}(\psi_1,\psi_2)\approx\max_{red2}=\max[\max(\Psi_1,\Psi_2),\{0.25\times(\Psi_1+\Psi_2)+0.5+0.5\times\max(\Psi_1,\Psi_2)\}]$ respectively [58]. Furthermore, an approximation of $\max_{red2}$ for $r$=4 is reduced as $\widehat{\max}(\psi_1,\psi_2)\approx\max_{red3}=\max(\Psi_1,\Psi_2)+\max\{0,(0.5\mp0.25\times\Delta)\}$ [58]. These approximations result in low implementation-complexity as compared to other simplified MAP algorithms [46, 58]. Similarly, MSE based simplified MAP algorithm [38] could be another candidate from the perspective of BER performance and implementation complexity.

### 3.2.2 Comparative Analysis of Architectures

In this subsection, architectures for $\widehat{\max}(\Psi_1,\Psi_2)$ will be analyzed for PWLA and MSE based simplified MAP algorithms. For such algorithm based on MSE [38], $\widehat{\max}(\Psi_1,\Psi_2)$ is approximated

as $\widehat{\max}(\Psi_1, \Psi_2) \approx \max_{mac} = \max(\Psi_1, \Psi_2) + \max\{0, (\ln 2 - 0.5 \times |\Delta|)\}$, as shown in Table 3.1. Fig. 3.2 (a) shows an architecture for $\max_{mac}$ expression, where $C_1$ is an output of CMP (comparison unit) which determines a maximum value between $\Psi_1$ and $\Psi_2$. In ABS (absolute-value unit), $\Delta$ and its two's complement values are fed to the multiplexer that selects an absolute value using a sign-bit or msb (most significant bit) of $\Delta$. Then, this absolute value is shifted by one bit-position to right (indicated as >>i=1) to obtain $C_2$ value. Finally, the value of $C_3 = \max\{0, (\ln 2 - 0.5 \times |\Delta|)$ is added with $C_1$ to realize $\max_{mac}$ value for MSE based simplified MAP algorithm, as shown in Fig. 3.2 (a). For PWLA based simplified MAP algorithm, architectures

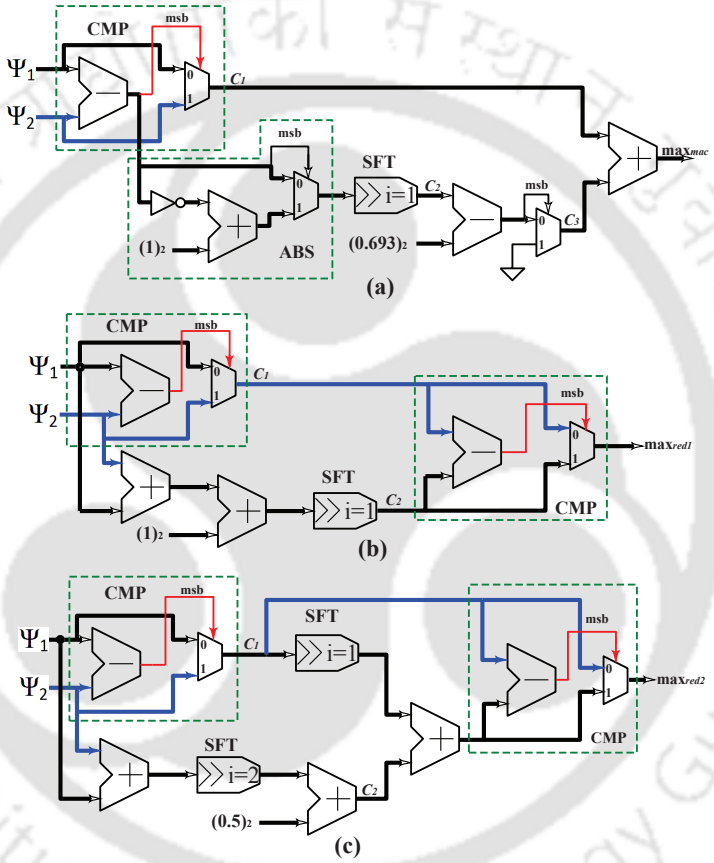


FIGURE 3.2: Logic-level architectures for $\widehat{\max}(\Psi_1, \Psi_2)$ approximation using MSE and PWLA based simplified MAP algorithms: (a) $\max_{mac}$ (b) $\max_{red1}$ (c) $\max_{red2}$; where $(0.5)_2$, $(0.693)_2$ and $(1)_2$ are the 2's complement binary representations of decimal numbers 0.5, 0.693 and 1 respectively.

for reduced $\widehat{\max}(\Psi_1, \Psi_2)$ expressions ($\max_{red1}$, $\max_{red2}$ and $\max_{red3}$, as discussed in section-3.2.1) with approximations $r$=3 and $r$=4 are analyzed. Fig. 3.2 (b) shows an architecture that realizes $\max_{red1} \approx \widehat{\max}(\psi_1, \psi_2)$ for an approximation of $r$=3. Here, $C_1$ is an output of CMP unit and $C_2$ holds the shifted value of $(\Psi_1 + \Psi_2 + 1)$. Finally, these values are fed to CMP unit to obtain the value of $\max_{red1}$, as shown in Fig. 3.2 (b). Similarly, an architecture which maps

reduced expression $\max_{red2}$ for an approximation $r$=4 is shown in Fig. 3.2 (c). Comparator-output $C_1$ is shifted and added with the value $C_2 = 0.25 \times (\Psi_1 + \Psi_2) + 0.5$. Thereafter, this sum and compared $C_1$ values are fed to CMP unit to compute the value of $\max_{red2}$. Fig. 3.3 shows

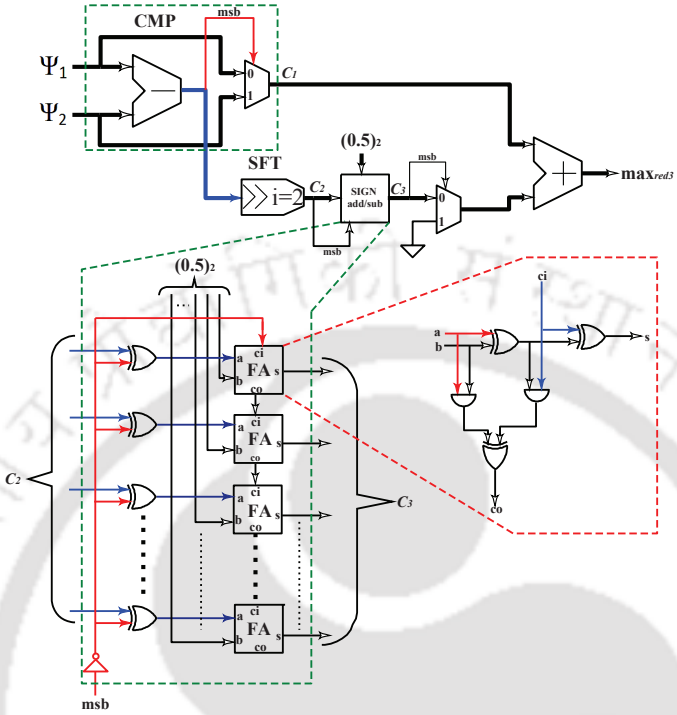


FIGURE 3.3: Logic-level architecture for an approximation $\max_{red3}$ using PWLA based simplified MAP algorithm.

an architecture to compute further reduced expression $\max_{red3}$ for $r$=4. Here, the value of $\Delta$ is shifted right by two bit-positions to generate $C_2$ which is fed to SIGN-add/sub unit along with its sign-bit or msb. SIGN-add/sub unit adds or subtracts the binary value $(0.5)_2$ with shifted $C_2$ value depending on its sign. As shown in Fig. 3.3, an internal architecture of SIGN-add/sub unit is enclosed by dash lines in which each bit of $C_2$ is XORed with negated msb and are fed to stack of one-bit FAs (full adders). These FAs add XORed bits with the bits of binary value $(0.5)_2$ to produce the value of $C_3 = 0.5 \mp 0.25 \times \Delta$, where the value of ci (carry-in) of first FA is a negated value of msb. Finally, the value of $C_3$ is compared with zero and added with $C_1$ to produce $\max_{red3}$, as illustrated in Fig. 3.3. Table 3.2 shows the list of critical path delays derived for the architectures presented in Fig. 3.2 and Fig. 3.3. Assuming the data-width of $n_d$; $\partial_{add}$, $\partial_{sub}$, $\partial_{mux}$ and $\partial_{sft}$ represent the delays imposed by $n_d$-bits adder, subtractor, multiplexer and shifter respectively. Subsequently, $\partial_{xor}$ and $\partial_{not}$ are the single-gate delays of XOR and NOT gates respectively. From Table 3.2, it can be seen that the architectures of $\max_{red1}$ and $\max_{red3}$ have the smallest critical path delays and the latter differs from the former only by an XOR-gate

TABLE 3.2: Critical path delays of the architectures for $\widehat{\max}(\Psi_1, \Psi_2)$ approximation using simplified MAP algorithms.

| Approximations | Critical path delays |
|---|---|
| $\max_{mac}$ | $\tau_{mac} = 2 \times (\partial_{sub} + \partial_{add} + \partial_{mux}) + \partial_{sft} + \partial_{not}$ |
| $\max_{red1}$ | $\tau_{red1} = 2 \times \partial_{add} + \partial_{sub} + \partial_{sft} + \partial_{mux}$ |
| $\max_{red2}$ | $\tau_{red2} = 3 \times \partial_{add} + \partial_{sub} + \partial_{sft} + \partial_{mux}$ |
| $\max_{red3}$ | $\tau_{red3} = 2 \times \partial_{add} + \partial_{sub} + \partial_{sft} + \partial_{mux} + \partial_{xor}$ |

delay. Thereby, it may be concluded that both these architectures are suitable for high speed implementations of turbo decoder.

### 3.2.3 Performance Analysis

In this subsection, comparative analysis on BER performances of turbo code using the simplified MAP algorithms based on PWLA ($\max_{red1}$, $\max_{red2}$ and $\max_{red3}$) and MSE-approximation ($\max_{mac}$) are carried out. Subsequently, these error-rate performances are compared with conventional log-MAP and max-log-MAP algorithms. Simulations are carried out with BPSK (binary phase shift keying) modulation in AWGN-channel environment. Standard parameters used in this process are block length ($N$) of 6144 bits, convolutional-encoder with a transfer function of $\{1, (1 + D + D^3) / (1 + D^2 + D^3)\}$ and a code rate of 1/3. Iterative turbo decoding of 5.5 iterations has been carried out and QPP interleaver is used for scrambling the data while decoding [31]. Fig. 3.4 (a) shows that the turbo code based on $\max_{mac}$ approximation performs ≈0.125 dB better than conventional max-log-MAP and ≈0.1 dB inferior to log-MAP algorithm at a BER of $10^{-4}$. On the other hand, BER performance of turbo code based on simplified MAP algorithm with PWLA of $\max_{red3}$ performs only ≈0.03 dB inferior to conventional log-MAP algorithm, as shown in Fig. 3.4 (b). Since $\max_{red2}$ and $\max_{red3}$ are the approximations for same simplified MAP algorithm based on PWLA of $r$=4 [46], BER performances of turbo code using these approximations are similar, as shown in Fig. 3.4 (c). However, Fig. 3.4 (c) shows that the turbo code based on $\max_{red1}$ approximation for $r$=3 performs inferiorly by ≈0.07 dB at a BER of $10^{-4}$ with respect to $\max_{red2}$ and $\max_{red3}$ approximations. Max-log-MAP algorithm with extrinsic scaling (s=0.7) [59] performs better than the conventional max-log-MAP algorithm and inferior to $\max_{red3}$ approximations based simplified MAP algorithm, as shown in Fig. 3.4 (d). Performance analysis has shown that the PWLA based simplified MAP algorithms ($\max_{red2}$ and $\max_{red3}$) for $r$=4 performed very close to conventional log-MAP algorithm, when compared to other algorithms. However, Table 3.2 shows that the architecture for $\max_{red3}$ has shorter critical path delay in comparison with $\max_{red2}$. Though an architecture of $\max_{red1}$ has shorter critical

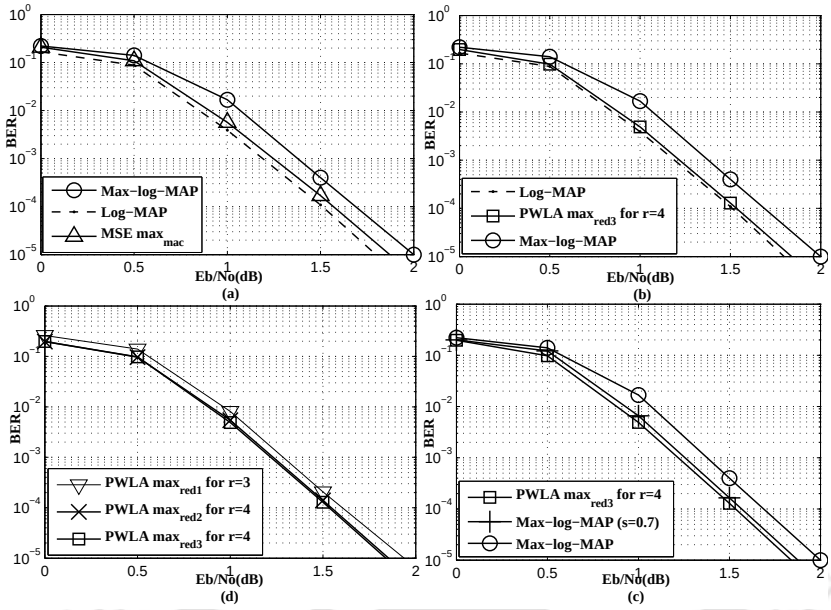


FIGURE 3.4: Performance comparison of turbo code based on simplified MAP algorithms for 5.5 decoding-iterations.

path by one gate delay in comparison with $\max_{red3}$, BER performance of $\max_{red3}$ is better than $\max_{red2}$ approximation. Thereby, simplified MAP algorithm based on $\max_{red3}$ approximation, which delivers near-optimal BER performance and has an architecture suitable for high-speed application, is chosen for an implementation of turbo decoder.

## 3.3 Turbo Decoder Architecture

This section discusses architectural aspects of various sub-blocks of turbo decoder, as well as integration of these sub-blocks to conceive turbo-decoder architecture for implementation.

### 3.3.1 SISO Architecture

This work presents radix-2 SISO-architecture for eight trellis-states ($S_N$=8) and sliding window size of 23 ($M$=23). Fig. 3.5 shows such architecture that comprises of BMC (branch metrics computation) unit, BMR (branch metrics routing) unit, FSMC (forward state metrics computation) unit, BSMC (backward state metrics computation) unit, DBSMC (dummy backward state metrics computation) unit and LCU ($LLR$ computation unit). Inputs $X$ and $X_{p1}$ are the received soft values of systematic and parity bits. In general, the total number of systematic and parity bits (denoted by $\omega$) for each transmitted bit decides number of parent-branch metrics which is equal to $2^{\omega}$. Since, single parity bit is generated by encoder for a systematic bit, the

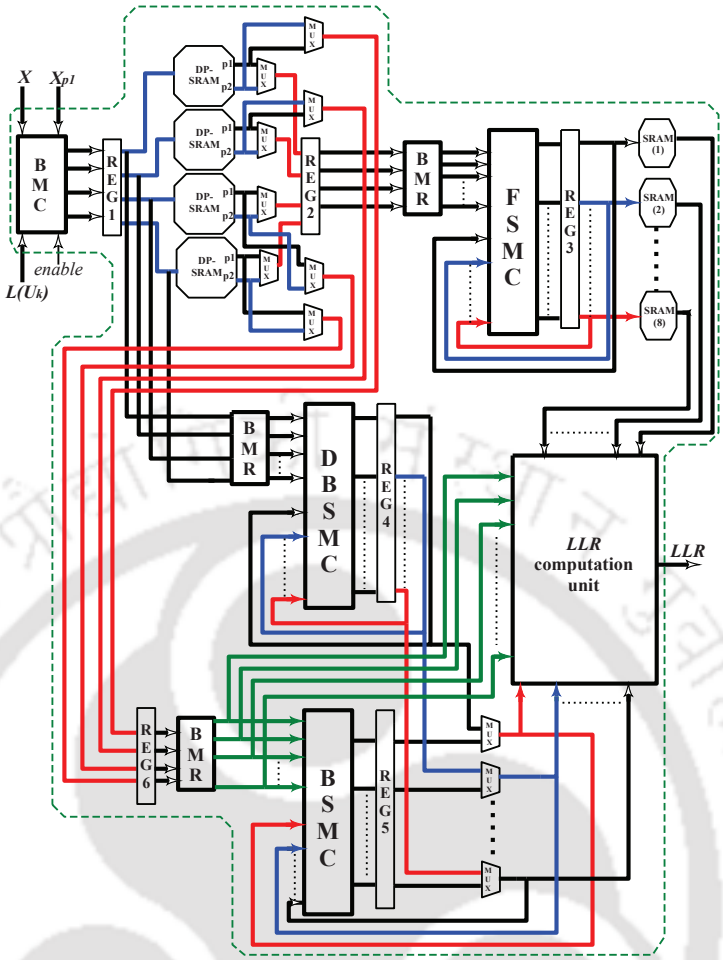


FIGURE 3.5: High-level architecture of SISO unit which is an integration of various sub-blocks like BMC, BMR, FSMC, BSMC, DBSMC, LCU, DP-SRAMs and SRAMs.

value of $\omega$ is two for this SISO unit and it corresponds to four parent branch metrics. Referring (3.3), the parent branch metric equations for the SISO unit are given as

$$\begin{aligned}
\gamma_k(s_a, s_b) &= -L(U_k)/2 - X - X_{p1} = -L(U_k)/2 - (X + X_{p1}),\\
\gamma_k(s_c, s_d) &= -L(U_k)/2 - X + X_{p1} = -L(U_k)/2 - (X - X_{p1}),\\
\gamma_k(s_e, s_f) &= L(U_k)/2 + X - X_{p1} = L(U_k)/2 + (X - X_{p1})\ and\\
\gamma_k(s_g, s_h) &= L(U_k)/2 + X + X_{p1} = L(U_k)/2 + (X + X_{p1})
\end{aligned} \tag{3.6}$$

where the value of $Lc$ in (3.3) is two, which is sufficient to deliver an optimum BER performance. Thereby, BMC unit computes these parent branch metrics, as shown in Fig. 3.6 (c), and is combinational circuit with adders, subtractors, shifter with a critical path delay of $\tau_{bmc} = \partial_{sub} + \partial_{add} + \partial_{sft} + \partial_{not}$. For all state transitions in trellis stage, radix-$r$ architecture of SISO unit has $r \times S_N$ branch metrics. Thereby, radix-2 architecture of SISO unit presented in this work has 16 branch metrics ($r \times S_N$=2×8). The BMR unit routes these four parent branch metrics into

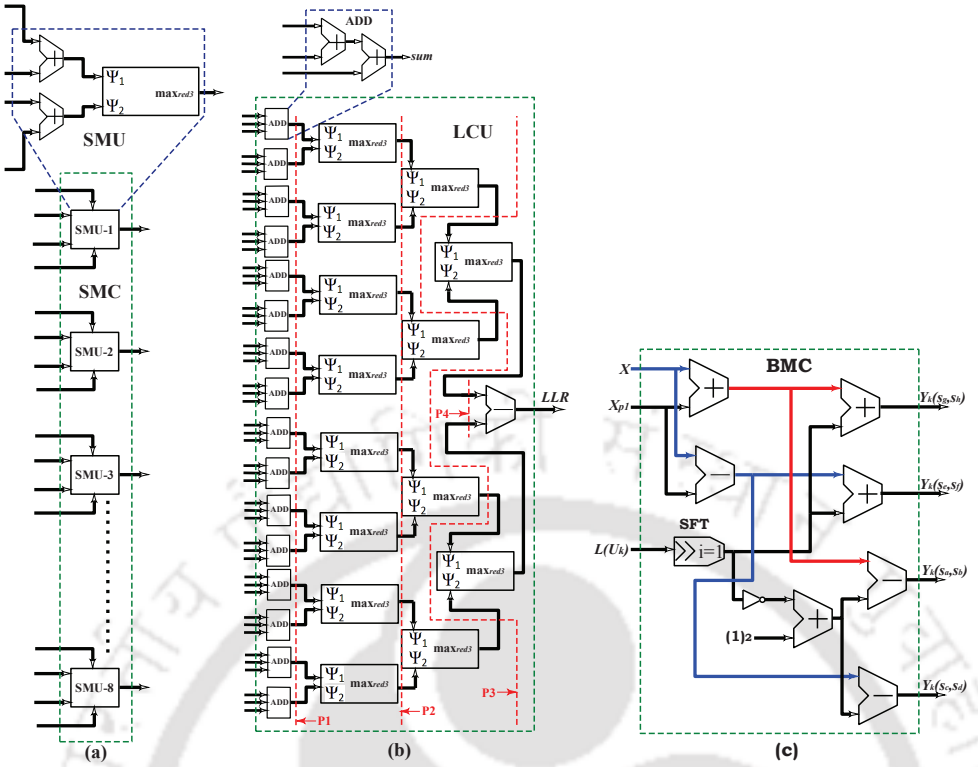


FIGURE 3.6: Logic-level architectures of (a) SMC (state metric computation) unit (b) LCU (LLR-computation-unit) (c) BMC (branch metric computation) unit.

TABLE 3.3: Hardware resources consumed by various sub-blocks of SISO unit.

| Sub-blocks | Adders | Subtractors | Multiplexers | Shifters | Registers |
|---|---|---|---|---|---|
| FSMC unit | 32 | 8 | 16 | 8 | None |
| BSMC unit | 32 | 8 | 16 | 8 | None |
| DBSMC unit | 32 | 8 | 16 | 8 | None |
| LCU | 60 | 15 | 28 | 14 | 30 |
| BMC unit | 3 | 3 | None | 1 | None |
| Additional units | None | None | 16 | None | 36 |
| Total elements | 159 | 42 | 92 | 39 | 66 |

16 branch metrics for various state transitions in the trellis stage. As shown in Fig. 3.6 (a), SMC (state metric computation) unit is a stack of $S_N$ SMUs (state metric units) based on simplified MAP algorithm ($\max_{red3}$) that is chosen in section-3.2. SMU computes forward or backward state metrics using $\max_{red3}$ architecture from Fig. 3.3. As shown in Fig. 3.6 (a), $(\Psi_1, \Psi_2) = \{\alpha_{k-1}(s'_0) + \gamma_k(s'_0, s_0), \alpha_{k-1}(s'_1) + \gamma_k(s'_1, s_0)\}$ and $(\Psi_1, \Psi_2) = \{\beta_{k+1}(s''_0) + \gamma_k(s''_0, s_0), \beta_{k+1}(s''_1) + \gamma_k(s''_1, s_0)\}$ for forward and backward state metric computations respectively. Inputs for SMC unit are 16 branch metrics for all state transitions and 8 state metrics of $(k\text{-}1)^{th}$ trellis stage.

SMC unit is used as FSMC and BSMC units for computing forward and backward state metrics of each trellis stage respectively. Subsequently, it is used as DBSMC unit for the estimation of initial backward state metrics for each sliding window, as shown in Fig. 3.5. LCU computes *LLR* value of $k^{th}$ trellis stage, as given in (3.5). In Fig. 3.6 (b), ADD sub-blocks are used for adding forward-state, backward-state and branch metrics for all state transitions of trellis stage. The maximum value among these added results of state transitions for transmitted bits of $U_k$=1 and $U_k$=0 are computed using $max_{red3}$ architecture. Finally, these two maximum values are subtracted to produce a-posteriori-probability *LLR* value for each of the trellis stage, as shown in Fig. 3.6 (b). Vertical dashed lines denoted by P1, P2, P3 and P4 are the portions of LCU architecture where registers are incorporated to pipeline this unit into three stages. Thereby, LCU starts delivering the *LLR* values after three clock cycles of delay. Table 3.3 summarizes the number basic elements like adders, subtractors, multiplexers, registers and shifters those are required by various sub-blocks of SISO unit presented in this work. It also accounts for additional multiplexers and registers used in SISO unit, as shown in Fig. 3.5.

### 3.3.2 SISO Scheduling

Soft-values ($X$ and $X_{p1}$) are sequentially fed to SISO unit in every clock cycle, and these values are used for the computation of branch metrics for each trellis stage. For the first SW1 (sliding window) time slot ($T_{SW1}$), BMC unit computes four parent branch metrics for each trellis stage in SW1 and these buffered parent branch metrics (using REG1) are stored in DP-SRAMs (dual port static - random access memories), as shown in Fig. 3.5. In $T_{SW2}$, parent branch metrics for SW2 are computed and stored in DP-SRAMs. Simultaneously, previously stored parent branch metrics of SW1 are fetched through *p1* ports of DP-SRAMs and are fed to BMR unit before FSMC unit via REG2, as shown in Fig. 3.5. Rest of the branch metrics for each trellis stage of SW1 are derived from BMR unit and are fed to FSMC unit. Subsequently, FSMC unit computes eight forward state metrics for each trellis stage of SW1 and stores them in eight different SRAMs, as shown in Fig. 3.5. On the other hand, parent branch metrics of SW2 are directly fed to BMR unit before DBSMC unit, which is used for dummy-back-trace. During this process, a backward trace of trellis stages in SW2 takes place to compute the initial values of backward state metrics, which are used for starting actual back-trace of SW1. In $T_{SW3}$, parent branch metrics for SW3 are computed by BMC unit and stored in DP-SRAMs. The parent branch metrics of SW1 fetched through *p1* ports of DP-SRAMs are fed to BMR unit, which is located before BSMC unit, via REG6. Initial value of backward state metric computed by DBSMC unit is fed to BSMC unit via multiplexer, as shown in Fig. 3.5. Thereby, using branch metrics computed using BMR unit and initial backward state metrics, BSMC unit starts actual back-trace for computing all

the backward state metrics of SW1 and are fed to LCU via multiplexers. Simultaneously, all the forward state metrics of SW1 are fetched from SRAMs and the branch metrics from BMR unit, which is before BSMC unit, are fed to LCU. These forward, backward and branch metrics are utilized by LCU to compute a-posteriori-probability *LLR* values for all the trellis stages of SW1, as shown in Fig. 3.5. Simultaneously, parent branch metrics of SW2 are fetched through *p2* ports of DP-SRAMs and are fed to FSMC unit to compute forward state metrics for SW2. This process continues and the *LLRs* for all trellis stages can be sequentially computed by SISO unit after two sliding windows. However, LCU is pipelined architecture that imposes additional delay ($\partial_{pipe}$) in the computation of *LLR* values. Therefore, decoding delay ($\partial_d$) is given as $\partial_d = 2\times \mathrm{T}_{SW} + \partial_{pipe}$.

### 3.3.3 Analysis of Memory Requirement

In general, SISO unit needs to store parent branch metrics and forward state metrics for all the trellis stages of two-sliding windows and one-sliding window, respectively, for computing the *LLR* values. In Fig. 3.5, there are four DP-SRAMs to store these parent branch metrics. Each DP-SRAM is of the size $2\times M\times \mathrm{n}_{pbm}$ bits where $\mathrm{n}_{pbm}$ denotes the data-width in bits for two's complement representation of parent branch metric. Thereby, memory required to store all the parent branch metrics is $2^{\omega}\times 2\times M\times \mathrm{n}_{pbm}$ bits. Similarly, eight single-port SRAMs are used for storing all the forward state metrics, as shown in Fig. 3.5. Memory required for this purpose is $\mathrm{S}_N\times M\times \mathrm{n}_{fsm}$ bits where $\mathrm{n}_{fsm}$ is data-width of forward state metric. Thereby, the total memory required by SISO unit to store parent-branch and forward-state metrics, for $S_N$ trellis states and $M$ sliding window size, is

$$\Pi_{mem} = M\times \{2^{\omega+1}\times n_{pbm} + n_{fsm}\times S_N\}\ bits. \tag{3.7}$$

This expression shows that the sliding window size and data-widths of metrics have profound influence on the memory requirement. For an optimum BER performance, sliding window size must be five to seven times the value of $K_r$ (constraint length) [60]. Based on encoder transfer function presented in section-3.2, the value of $K_r$ is three; thereby, a sliding window size of 23 has been used in this work. Similarly, internal data-width of parent-branch and forward-state metrics influence memory requirement as well as the BER performance of turbo decoder [24]. Thereby, two's complement fixed-point representations of forward and parent-branch metrics are $\mathrm{n}_{fsm}$=($\mathrm{n}_b$=9,$\mathrm{n}_p$=4) and $\mathrm{n}_{pbm}$=($\mathrm{n}_b$=7,$\mathrm{n}_p$=3), respectively, where the total number of bits is represented by $\mathrm{n}_b$, and $\mathrm{n}_p$ is the number of bits for fractional precision. Since the memories are

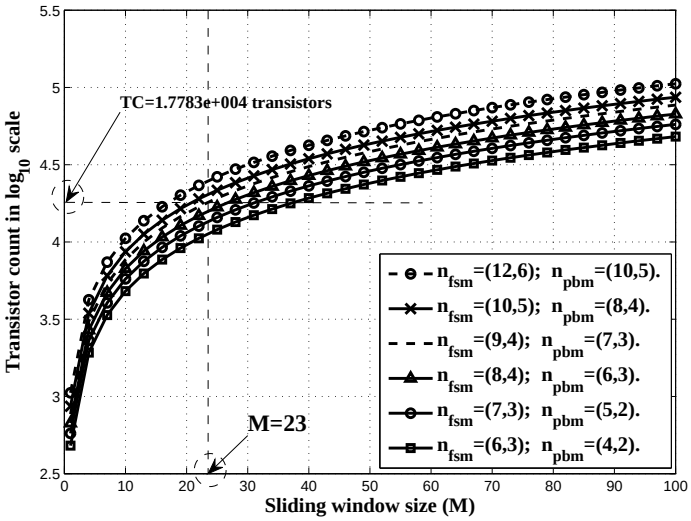


FIGURE 3.7: Transistor count required by memories in SISO unit for various sliding window sizes and data-widths of internal metrics.

DP-SRAM and SRAM, six CMOS transistors are required to store a bit in SRAM [61]. Thereby, an expression (3.7) for memory consumed by SISO unit in terms of TC (transistor count) is given as

$$TC = 6 \times M \times \left(2^{\omega+1} \times n_{pbm} + n_{fsm} \times S_N\right) \ transistors. \tag{3.8}$$

Fig. 3.7 shows the plots of such TCs in logarithmic scale ($\log_{10}$ scale) with respect to increasing sliding window sizes for different values of $n_{fsm}$ and $n_{pbm}$. In Fig. 3.7, intersection of horizontal and vertical dashed lines shows that for $M$=23 as well as $n_{fsm}$=(9, 4) and $n_{pbm}$=(7, 3), the memory required by SISO unit for branch and forward state metrics consumes 17783 ($10^{4.25}$) CMOS transistors. As the sliding window size increases from 10 to 30 for data-widths of $n_{fsm}$=(12, 6) and $n_{pbm}$=(10, 5), the SISO unit requires approximately 21623 ($10^{4.5}-10^{4.0}$) additional CMOS transistors ($\approx$ 68% more), as shown in Fig. 3.7. For a sliding window size of 23, on increasing the data widths from $n_{fsm}$=(9, 4) to $n_{fsm}$=(12, 6) and $n_{fsm}$=(7, 3) to $n_{fsm}$=(10, 5), the SISO-unit memory uses approximately 5931 ($10^{4.375}-10^{4.25}$) additional transistors. This approach can be used for determining the number of transistors required for any arbitrary values of sliding window sizes and data-widths.

### 3.3.4 Interleaver Design

Interleaver is an essential part and is also responsible for an excellent BER performance of turbo code. Interleaver architectures are well studied in literature [31, 62]; and the recent wireless communication standards like 3GPP-LTE and WiMAX have incorporated QPP and ARP interleavers respectively. In this work, contention free QPP interleaver architecture is used in the turbo decoder design [31]. Mathematical equation for the interleaved address is given as $I(i)=(\psi_1 \times i + \psi_2 \times i^2)\ mod\ N$ where $N$ represents a turbo block length, $I(i)$ is an interleaved address for each sequential address $i$ (such that $0<i$), $\psi_1$ is a value which is relatively prime to $N$ and $\psi_2$ is a prime factor of $N$. However, an equation for $I(i)$ can be recursively computed as $I(i+1)=I(i)+G(i)$ where $G(i)=(\psi_1+\psi_2+2\times\psi_2\times i)\ mod\ N$, similarly, $G(i)$ is recursively calculated as $G(i+1)=G(i)+(2\times\psi_2\ mod\ N)$. This recursive architecture of QPP interleaver has a simplified design and it can be easily used in the parallel architecture of turbo decoder to achieve higher throughput [31]. Subsequently, QPP interleaver can be configured to calculate interleaved addresses for any value of $N$. For example, 3GPP-LTE wireless standard uses 188 different values of $N$, ranging from 40 bits to 6144 bits. Thereby, QPP interleaver can be configured to produce contention-free interleaved addresses for any of these $N$ values by changing the values of $\psi_1$ and $\psi_2$ in the expression for $I(i)$.

### 3.3.5 Decoder Architecture

Architecture of turbo decoder that uses SISO unit based on simplified MAP algorithm and QPP interleaver is shown in Fig. 3.8. It has been designed for a code-rate of 1/3, $N$ of 6144 bits and with a transfer function of encoder based on the specification of 3GGP-LTE wireless communication standard, as discussed in section-3.2. Incoming soft values from soft-demodulator are S/P (serial-to-parallel) converted into three soft values of $X$, $X_{p1}$ and $X_{p2}$. These values are stored in three different memories, as indicated by INP-MEM in Fig. 3.8. Soft values are quantized as ($n_b$, $n_p$)=(7, 3) and the size of each memory is $N\times n_b$ bits. Fig. 3.8 shows the AGU (address generation unit) incorporated with sequential and QPP interleaved address generators. As illustrated by Fig. 3.8, a multiplexed *memory-address*, which can be sequential or pseudo-random in nature, from the AGUs is fed to all memories used in the turbo decoder. After storing these soft values, systematic flow of turbo decoding is described as follows.

- Initially, the soft-values $X$ and $X_{p1}$ are fetched sequentially from INP-MEM using the addresses generated by AGU and are fed to SISO unit. This unit processes these values to generate all $LLR_k$ values for $k$={1, 2, 3 ..... $N$}. Simultaneously, the extrinsic information

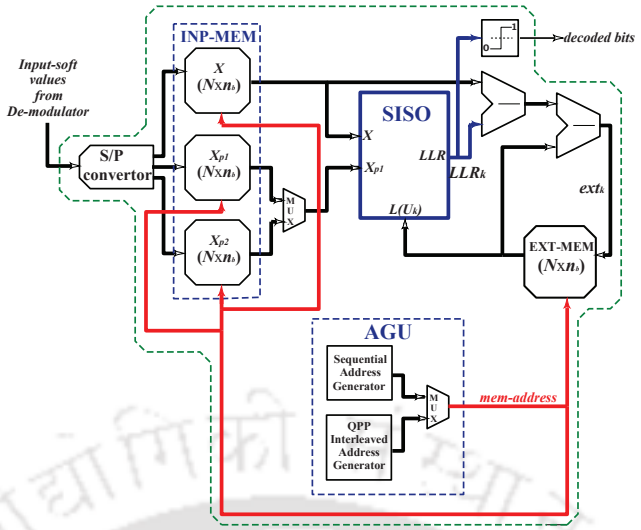


FIGURE 3.8: High-level architecture of turbo decoder which incorporates SISO unit using the simplified MAP algorithm based on PWLA ($\max_{red3}$) and QPP interleaver.

is computed by subtracting the soft value $X$ and a-priori-probability value $L(U_k)$ with $LLR_k$ values. Mathematical expression for extrinsic information is given as $ext_k = \{LLR_k - X - L(U_k)\}$ where $L(U_k)$ has null value for the first-half iteration. Subsequently, these $ext_k$ values are sequentially stored in memory using sequential address generator of AGU, as shown in Fig. 3.8.

- In the second half iteration, soft-values $X$ and $X_{p2}$ are fetched pseudo-randomly and sequentially, respectively, from INP-MEM and are fed to SISO unit for the computation of $LLR_k$. Simultaneously, stored extrinsic information values are fetched pseudo-randomly from EXT-MEM using interleaved addresses produced by QPP address generator of AGU and these values are fed to SISO unit as $L(U_k)$. Extrinsic information is computed analogously as that of the first-half iteration except the soft-values $X$ and $ext_k$ are fetched pseudo-randomly using AGU and are given as $ext_k = \{LLR_k - \pi(\mathrm{X}) - \pi(ext_k)\}$ where $\pi(\cdot)$ represents an interleaving function. Such extrinsic information are stored pseudo-randomly in the memory (denoted by EXT-MEM), as shown in Fig. 3.8, and this completes one iteration of turbo decoding.

- In the third half iteration, extrinsic information are fetched sequentially from the memory for de-interleaving process and are fed at $L(U_k)$ port of SISO unit. Rest of the operations is same as first-half iteration and this iterative process continues for fixed number of decoding iterations. Finally, $LLR_k$ values are fed to hard-decision unit for generating error-free hard *decoded bits*, as shown in Fig. 3.8.

## 3.4 VLSI Implementation, Application and Comparison

In this section, implementation of suggested turbo-decoder architecture has been carried out and the results are compared with reported works.

### 3.4.1 VLSI Implementation

***Front-end design procedure***: Turbo-decoder architecture presented in this chapter is coded with Verilog HDL (hardware descriptive language) and its functional verification with the test-vectors of input soft-values has been carried out using *SYNOPSYS*-verilog-compiler-simulator tool [63]. Such functionally verified HDL-code of turbo decoder is synthesized with the standard-cell libraries of 130 nm CMOS technology node, using *SYNOPSYS*-design-compiler tool, by setting various timing constraints [63]. Such synthesis-process generates gate-level netlist for turbo-decoder design. Then, STA (static timing analysis) of this netlist under worst and best corner cases are carried out for checking setup and hold time violations respectively. At this stage, all the setup-time violations are fixed; however, few hold-time violations are unresolved. Nevertheless, handful of such hold-time violations will be mitigated during back-end design flow. Thereafter, this STA-verified netlist is subjected to post-synthesis simulation using the same test vectors of input soft-values and its outputs are verified with the earlier results of functional verification.

***Back-end design procedure***: In this design, we have used five metal layers; and the IO (input output) pads along with the corner pads are set in their appropriate positions around the core-area where standard cells of the design are placed. Power/ground rings and stripes are set for standard cells on the core area. Then, CTS (clock tree synthesis) is carried out and an optimum tree structure is set for the clock network. In order to fix the hold time violations, additional buffers are placed along the violated paths. On performing STA thereafter, hold-time violations are fixed and the timing closure is achieved at maximum operating clock frequency of 303 MHz. Special routing of the design is performed to interconnect all the standard cells among each other. Core and IO filler cells are added in the design to maintain the continuity and to fill the gaps between the standard cells. Then the layout is verified for geometry, connectivity, antenna effects and metal density. Finally, STA of the layout is carried out to check the timing closure. Thereafter, the netlist of layout is extracted and subjected to post-layout simulation along with the RC extracted values and the test vectors of soft values. Subsequently, the post-layout simulated output is matched with functionally verified output. It is to noted that the

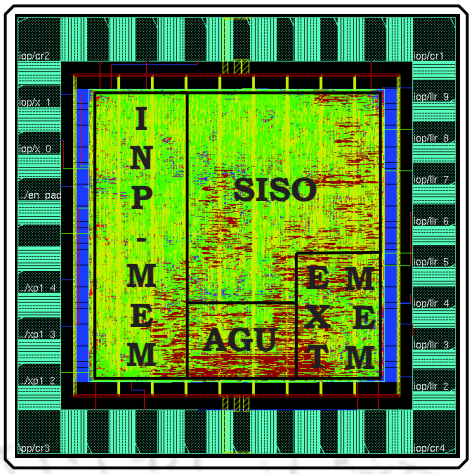


FIGURE 3.9: Chip-layout of turbo decoder implemented at 130 nm CMOS technology node.

TABLE 3.4: Design metric values obtained by implementing the turbo decoder at 130 nm CMOS technology node.

| Design metrics | Values obtained |
|---|---|
| Level of logics | 34 levels |
| Hierarchical cell count | 4172 standard-cells |
| Combinational area | 0.83 $mm^2$ |
| Non-combinational area | 1.34 $mm^2$ |
| Design core area | 2.2 $mm^2$ |
| Critical path delay | 2.01 nS |
| Maximum clock frequency | 303 MHz |
| Leakage power @ 303 MHz clock frequency | 512.7 $\mu$W |
| Dynamic power @ 303 MHz clock frequency | 41.87 mW |
| Total Power consumption @ 303 MHz clock frequency | 42.38 $mW$ |

back-end design in this work has been carried out using CADence-SOC-Encounter and CADence-Virtuoso tools [64]. Fig. 3.9 shows a final chip-layout of turbo decoder architecture with various sub-blocks. It has 29 IO pads and four corner pads around the core area of this layout. Since the data-width ($n_b$) for each of the soft values is seven bits, there are 21 input pads, assigned for $X$, $X_{p1}$ and $X_{p2}$. Similarly, two input pads are used for clock and enable signals, and one output pad is assigned to deliver decoded bits from turbo decoder. There are two power pads for the supply voltage of 1.2 V and one power pad for the supply voltage of 3.3 V. These voltages of 1.2 V and 3.3 V are used as supplies for standard-cells of core and digital-programmable IO pads respectively. The remaining two IO pads are ground pins for the chip. Power rings are placed around the core area and the power strips are vertically oriented on it. Placed and routed cells of the design

core are shown in Fig. 3.9. Design metrics such as core area, power consumption and maximum operating clock frequency of an implemented turbo decoder design at 130 nm technology node are presented in Table 3.4. The decoder-architecture has a core area of 2.2 $mm^2$ and it can be operated with a maximum clock frequency of 303 MHz. Implementation of turbo decoder with 34 levels of logic consumes 4172 standard cells. In order to estimate the power consumption, power analyzer tool generates a forward SAIF (switching activity interchange format) file. This file contains the information regarding switching activity of design and is processed with test-vector to produce a backward annotated SAIF file. Finally, backward annotated SAIF file is read using power analyzer tool to compute the power consumption of an implemented design. Thereby, total dynamic power of 41.87 mW and static leakage power of 512.7 $\mu$W are consumed by the implemented design at 303 MHz.

### 3.4.2 Possible Applications

As discussed earlier, turbo decoders are used in the physical-layer design of various wireless communication standards. Thereby, implemented turbo decoder must support data rates of these standards such that the input soft values are processed at specified rate. Throughput achieved by turbo decoder decides such processing speed and its applicability in the wireless communication system. Achievable throughput of conventional turbo decoder in bps (bits per second) is given as [37]

$$\theta_T = \frac{N \times f_{siso} \times P \times b}{2 \times I \times (N + \partial_d \times P)} \tag{3.9}$$

as discussed in the earlier chapter. The turbo-block length values are $N$=6144 bits for 3GPP-LTE/LTE-A standard and $N$=12282 bits for DVB-SH standard. Maximum operating clock frequency of SISO unit ($f_{siso}$) achieved in this work is 303 MHz. Suggested turbo decoder is designed for 5.5 iterations ($I$=5.5) and sliding window size of 23 ($M$=23). Similarly, $\partial_d$ represents the decoding delay of our design, as discussed in section-3.3.2. A single SISO unit ($P$=1) has been used in this design of turbo decoder and this radix-2 decoder-architecture processes only $b$=1 bit per clock cycle. By substituting these values in (3.9), achievable throughput of an implemented turbo decoder is approximately 28 Mbps. This decoder has radix-2 configuration, however, its throughput can be increased by using radix-4 configuration, where two bits are processed in every clock cycle (i.e. $b = 2$ bits/cycle). Similarly, multiple SISO units can be used to meet the higher throughput specifications. Fig. 3.10 shows the estimation of achievable throughputs at various operating frequencies for different configurations of turbo decoder architecture. Vertical

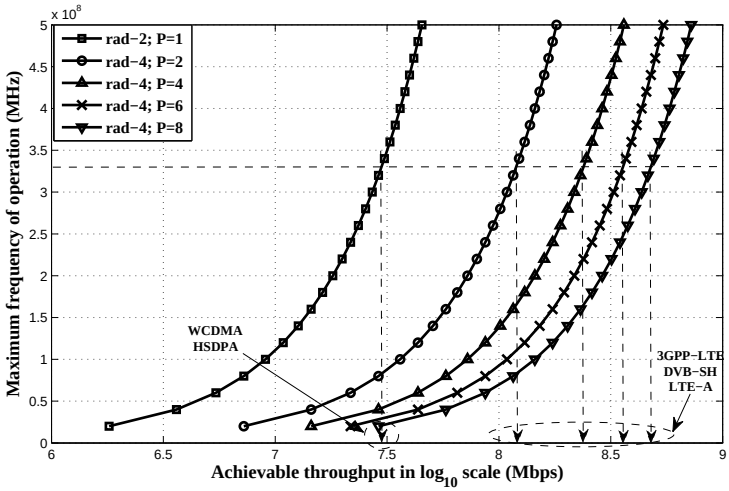


FIGURE 3.10: Plots of achievable throughputs with respect to operating clock frequencies for various configurations of turbo decoder.

dash lines with arrows indicate the throughput achieved by these configurations at a frequency of 303 MHz, which is indicated by a horizontal dashed line. Turbo decoder presented in this work with radix-2 (*rad-2*) configuration and single SISO unit has achieved a throughput $\approx$ 28 Mbps at 303 MHz and is suitable for wireless communication standards such as WCDMA and HSDPA which require throughput greater than 2 Mbps and 14.5 Mbps respectively. On the other hand, using radix-4 configuration (*rad-4*) with parallel SISO units for turbo decoder architecture, throughputs of approximately 110 Mbps, 220 Mbps, 320 Mbps and 425 Mbps at a clock frequency of 303 MHz can be achieved for $P = 2$, 4, 6 and 8, respectively, as shown in Fig. 3.10. Thereby, such configurations are applicable for wireless standards like 3GPP-LTE, DVB-SH and LTE-A, which require greater throughput than 100 Mbps. On the other side, energy efficiency is also an important measure of the decoder-architecture and its implementation. It estimates the amount of energy consumed to decode a hard bit in every decoding-iteration. It is proportional to throughput achieved, power consumed and iterations performed by the turbo decoder. Thereby, energy efficiency can be computed as

$$\eta_{energy} = \left(\frac{\rho}{\Theta_T \times I}\right) nJ/b/iterations \tag{3.10}$$

where $\rho$ represents the power consumption of decoder at an operating clock frequency. For the turbo decoding of 5.5 iterations, the suggested turbo-decoder implementation consumes 42.38 mW of power at 303 MHz with energy efficiency of 0.28 nJ/b/iterations.

### 3.4.3 Comparison of Results

The turbo decoder implemented in this work targets a radix-2 non-parallel architecture which has achieved throughput and energy efficiency, as shown in Table 3.5. Benkeser et al. [28] have proposed radix-2 non-parallel architecture for turbo decoder and have incorporated max-log-MAP algorithm in their design. It has achieved a throughput of 20.2 Mbps, which is suitable for HSDPA wireless communication standard, as presented in Table 3.5. An implementation of suggested turbo-decoder architecture, which is designed using PWLA based simplified MAP algorithm, has achieved better throughput and is energy-efficient than the reported turbo decoder design of [28]. However, their design [28] occupies lesser silicon-area as compared to the turbo decoder implemented in this work. Another implementation work in 180 nm CMOS technology node by Bickerstaff et al. [65] is a radix-4 non-parallel architecture in which a conventional log-MAP algorithm is used. We have achieved better throughput than the decoder presented in [65]. Silicon-area occupied and power consumed by this reported design [65] are higher because of the larger technology node and higher supply voltage of 1.8 V respectively. Radix-2 non-parallel architecture from [66] by Bickerstaff et al. uses log-MAP algorithm with programmable log-sum correction table. There is no justification for comparing area occupancy and power consumption because the implementation in [66] is for both Viterbi and turbo decoders on a single chip, fabricated at 180 nm technology node. However, Table 3.5 shows that the throughput of turbo-decoder architecture presented in this work is better than the throughput of turbo decoder in [66]. Myoung et al. [67] presented radix-2 non-parallel turbo decoder architecture for multiple 3G standards and it uses log-MAP algorithm. The throughput of their architecture is lower compared to presented decoder architecture. The turbo decoder implemented in this work is also compared with other parallel turbo decoders in Table 3.5. The radix-4 parallel architecture of turbo decoder proposed by Kim et al. [68] utilizes eight SISO units in parallel to achieve a throughput of 100 Mbps using max-log-MAP algorithm. The realization of parallel architecture increases the design area to 10.7 $mm^2$. Another parallel radix-4 architecture implemented by Maurizio et al. [37] uses four SISO units in parallel. It uses max-log-MAP algorithm and the throughput achieved is more compared to the turbo decoder presented in this work. Zhongfeng et al. [69] has designed a low complexity parallel architecture achieving a throughput of 55 Mbps. May et al. [47] proposed a parallel turbo decoder architecture which is based on radix-4 configuration with max-log-MAP algorithm and has achieved better throughput of 150 Mbps, than the proposed architecture. The radix-4 max-log-MAP parallel architecture designed by Studer et al. [29] achieved a throughput of 390.6 Mbps which is also much better than the proposed turbo decoder architecture. Similarly, this work is compared with other turbo decoders with parallel [49, 52, 70] and double-binary [55, 71, 72] architectures, as shown in Table 3.5. It

TABLE 3.5: Comparison for the implementation of turbo decoder with the related works

| | **Prop.**♠ | [28]♣ | [65]♣ | [66]♣ | [67]♣ | [68]♣ | [37]¶ | [69]¶ | [47]¶ | [29]♣ | [52]♣ | [49]♠ | [72]♣ | [70]♣ | [55]♠ | [71]♠ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Tech. (nm) | **130** | 130 | 180 | 180 | 250 | 130 | 130 | 130 | 65 | 130 | 90 | 90 | 130 | 65 | 130 | 90 |
| Supply (V) | **1.2** | 1.2 | 1.8 | 1.8 | 2.5 | 1.2 | 1.2 | 1.2 | 1.1 | 1.2 | 1.0 | 0.9 | 1.2 | 1.2 | 1.2 | 1.1 |
| Throughput (Mbps) | **28** | 20.2 | 24 | 2 | 5.5 | 100 | 90 | 55 | 150 | 390.6 | 130 | 1400 | 50 | 108 | 115.4 | 186 |
| Area ($mm^2$) | **2.2** | 1.2 | 14.5 | 9 | 8.9 | 10.7 | NA | NA | 2.10 | 3.57 | 2.1 | 9.61 | 2.24 | 0.66 | 6.66 | 3.38 |
| Power (mW) | **42.38** | 61.5 | 1450 | 306 | NA | 153 | NA | NA | 300 | 789 | 219 | 1356 | NA | 90.9 | 197.3 | 148 |
| Clk Freq. (MHz) | **303** | 246 | 145 | 111 | 135 | 250 | 200 | NA | 300 | 302 | 275 | 175 | 200 | 270 | 100 | 152 |
| Iter. | **5.5** | 5.5 | 6 | 10 | 6 | 8 | 8 | 6 | 6.5 | 5.5 | 8 | 8 | 8 | 8 | 4 | 8 |
| $\eta_{energy}$ (nJ/b/iter.) | **0.28** | 0.54 | 15.3 | 10.1 | 6.9 | 0.61 | NA | NA | 0.31 | 0.37 | 0.21 | 121 | NA | 0.17 | 0.43 | 0.13 |
| Radix | **2** | 2 | 4 | 2 | 2 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 2 | 4 | 4 |
| Architecture | **NP**† | NP | NP | NP | NP | P§=8 | P=4 | P=2 | PM✠ | P=8 | P=8 | P=32 | DB‡ | P=4 | DB | DB |

†: NP = Non-parallel architecture; ‡: DB = Double-binary architecture; §: P = Parallel architecture; ✠: PM = Pipelined XMAP architecture.
♣: On-chip measured results; ♠: Post-layout simulated results; ¶: Post-synthesis simulated results.

may be safely concluded from Table 3.5 that the design presented in this work has achieved better performance among radix-2 and radix-4 non-parallel architecture implementations. However, the parallel architectures are better than the proposed non-parallel-radix-2 architecture in terms of throughput.

## 3.5 Memory-Reduced MAP Decoding for Parallel Turbo Decoders

Based on comparative analysis from previous section, it may be concluded that the throughput more than 100 Mbps can be achieved by parallel turbo decoders. However, such decoders occupy large silicon-area due to multiple SISO units involved in their designs [48]. Thereby, this work presents memory-reduced technique for these SISO units and this eventually reduces the silicon-area consumed by such parallel turbo decoders. This section begins with brief discussion on conventional BCJR algorithm [18]. Thereafter, memory-savior MAP algorithm with new backward state-metric estimation has been presented and is referred as RSWMAP algorithm in this work. Subsequently, mathematical reformulation of branch metric equation has been carried out for further memory saving.

### 3.5.1 Theoretical Background

Conventional BCJR algorithm determines probability (denoted by $\mathrm{P}(U_k|y)$) that the transmitted bit $U_k$ is 1/0, provided the sequence of corrupted soft values $y$ are received [18]. This is equivalent to a-posteriori-probability $LLR_k$ value which is obtained by logarithmic transformation of such probability-ratio [73], and is given as $LLR_k$=ln$\{P(U_k$=1$|y)/P(U_k$=0$|y)\}$. The sign of $LLR_k$ indicates whether the transmitted bit is 1/0, and its magnitude indicates the likelihood of determining a correct value of the transmitted bit. If $(s',s)\rightarrow U_k$=1 and $(s',s)\rightarrow U_k$=0 represent the sets of state transitions for the transmitted bit $U_k$=1 and $U_k$=0, respectively, then $LLR_k$ can be expressed as [21]

$$LLR_k = \ln\left(\frac{\sum\limits_{(s',s)\rightarrow U_k=1} P(s',s,y)}{\sum\limits_{(s',s)\rightarrow U_k=0} P(s',s,y)}\right). \tag{3.11}$$

The entire received sequence $y$ can be partitioned into three sub-parts: $y_{i<k-1}$, $y_k$ and $y_{i>k+1}$. Such that $y_k$ represents a part of $y$, received at an instant $k$, and the other two parts of $y$

received before and after this instant are $y_{i<k-1}$ and $y_{i>k+1}$ respectively. Thereby, the probability $P(s', s, y)$ from (3.11) can be expressed using these sub-parts of $y$ as

$$P(s', s, y) = P(s', s, y_{i<k-1}, y_k, y_{i>k+1}). \tag{3.12}$$

Applying Bayes' rule and assuming that the channel is memory-less and discrete, an expression for $P(s', s, y)$ from (3.12) can be rewritten as

$$\begin{aligned} P(s', s, y) &= P(y_{i>k+1}|s) \times P\{(y_k, s)|s'\} \times P(y_{i<k-1}|s) \\ &= \widehat{\beta}_k(s) \times \widehat{\gamma}_k(s', s) \times \widehat{\alpha}_{k-1}(s') \end{aligned} \tag{3.13}$$

where $\widehat{\alpha}_{k-1}(s')$, $\widehat{\beta}_k(s)$ and $\widehat{\gamma}_k(s', s)$ represent forward-state-metric, backward-state-metric and branch-metric respectively. They are used in the computation of a-posteriori-probability $LLR$ values for successive trellis stages. From (3.13) and (3.11), expression for $LLR_k$ is given as

$$LLR_k = \ln\left(\frac{\sum\limits_{(s',s)\to U_k=1} \widehat{\alpha}_{k-1}(s') \times \widehat{\gamma}_k(s', s) \times \widehat{\beta}_k(s)}{\sum\limits_{(s',s)\to U_k=0} \widehat{\alpha}_{k-1}(s') \times \widehat{\gamma}_k(s', s) \times \widehat{\beta}_k(s)}\right). \tag{3.14}$$

### 3.5.2 RSWMAP Algorithm

In the conventional BCJR algorithm [18], computations of forward-state, backward-state and branch metrics for entire trellis stages result in huge memory requirement and impose large decoding delay. Unlike such conventional decoding technique, the SWBCJR (sliding window Bahl Cocke Jelinek Raviv) algorithm segments entire trellis structure into number of sliding windows and each window covers $M$ trellis stages that is referred as sliding window size [36]. This value of $M$ affects memory requirement, decoding delay and error-rate performance of the turbo decoder. Similarly, initialization of backward-state metrics while backward tracing the trellis stages is an important factor that is responsible for error-rate performance. RSWMAP algorithm suggested in this work focuses on the estimation of backward-state metric values that initiates back-trace and aims to deliver better error-rate performance. On the other hand, forward-state metrics and a-posteriori-probability $LLR$ values are computed with conventional methods in this algorithm. Major steps involved in the RSWMAP algorithm are presented as follow.

***Initialization***: Assuming that the encoder is reset, the forward state metrics are initialized as $\widehat{\alpha}_{k=0}(s_i)$=1 $\forall$ $i$=0 and $\widehat{\alpha}_{k=0}(s_i)$=0 $\forall$ $i\neq$0.

***Forward recursion***: During this process, the forward state metric of each states for successive trellis stages are computed as

$$\widehat{\alpha}_k(s) = \sum_{all\ \ s'} \widehat{\alpha}_{k-1}(s') \times \widehat{\gamma}_k(s', s), \tag{3.15}$$

where $\widehat{\gamma}_k(s', s)$ is a branch metric, which is mathematically expressed as

$$\widehat{\gamma}_k(s', s) = \exp(U_k \times L(U_k)/2) \times \exp\left(\frac{Lc}{2} \sum_{l=1}^{n} y_{kl} \times x_{kl}\right) \tag{3.16}$$

where $x_{kl}$ and $y_{kl}$ are transmitted bit and its demodulated soft value respectively.

***Backward-recursion and estimation of backward state metrics***: If $S_N$ represents total number of states in each trellis stage then for $k>M$, the backward state metrics are initialized as $\widehat{\beta}_k(s_j)=1/S_N\ \forall\ j \in S_N$ and during the backward recursion, backward state metrics for successive trellis stages are computed from instant (*k-1*) to (*k-M*) as

$$\widehat{\beta}_k(s) = \sum_{all\ \ s} \widehat{\beta}_{k+1}(s) \times \widehat{\gamma}_{k+1}(s', s). \tag{3.17}$$

In this work, we have suggested a new method of initializing backward state metrics, which starts the backward recursion in MAP decoding. For a block length of $N$, consider a trellis structure that defines relationship among present, past and future trellis states at an instant $k$. This relation is expressed by a-posteriori transition-probability from (3.13), in which the backward state metric is represented as

$$\widehat{\beta}_k(s) = P(y_{i>k+1}|s). \tag{3.18}$$

It represents a probability that the received sequence after an instant $k+1$ is $y_{i>k+1}$ at $s$ trellis state. At $k=M$ and from (3.18), the initial value of backward state metric which starts the backward recursion can be expressed as

$$\widehat{\beta}_M(s) = P(y_{i>M+1}|s) = \sum_{all\ \ s''} P\{(y_{i>M+1}, s'')|s\} \tag{3.19}$$

where $s''$ represents a set of trellis states at $k$=$M$+1 and they are associated with the transitions to state $s$, during the backward recursion. Probability equation from (3.19) can be further expressed as

$$\begin{aligned}\widehat{\beta}_M(s) &= \sum_{all\ \ s''} P\{(y_{i=M}, y_{i>M}, s'')|s\} = \sum_{all\ \ s''} P\left[\{(y_{i>M}), (y_{i=M}, s'')\}|s\right] \\ &= \sum_{all\ \ s''} P\{(y_{i>M})|(y_{i=M}, s'', s)\} \times P\{(y_{i=M}, s'')|s\}\end{aligned} \tag{3.20}$$

based on the Bayes' rule, which states that $P[(X,Y)|Z] = P[X|(Y,Z)] \times P(Y|Z)$. Applying conditions of discrete memoryless channel in (3.20), the mathematical expression for $\widehat{\beta}_M(s)$ is given as

$$\widehat{\beta}_M(s) = \sum_{all\ \ s''} P(y_{i>M}|s) \times P\{(y_{i=M}, s'')|s\}. \tag{3.21}$$

Referring (3.13), the probabilities $P(y_{i>M}|s)$ and $P\{(y_{i=M}, s'')|s\}$ from (3.21) can be expressed as $\widehat{\beta}_{M+1}(s'')$ and $\widehat{\gamma}_M(s'', s)$ respectively. Finally, the value of estimated backward state metric is

$$\widehat{\beta}_M(s) = \sum_{all\ \ s''} \widehat{\beta}_{M+1}(s'') \times \widehat{\gamma}_M(s'', s) \tag{3.22}$$

where $\widehat{\beta}_{M+1}(s'')$ represents the probability of encoder-state at an instant $k$=$M$+1 provided that the received sequence is $y_{i>M+1}$. This expression can be replaced with the value $1/S_N$ which is a probability that the encoder can attain one of the $S_N$ states. Thereby, an expression for $\widehat{\beta}_M(s)$ in (3.22) can be computed as

$$\widehat{\beta}_M(s) = \frac{1}{S_N} \sum_{all\ \ s''} \widehat{\gamma}_M(s'', s). \tag{3.23}$$

***Computation of a-posteriori-probability LLR value***: At an instant $k$-$M$, the probability $P_{k-M}(s', s, y)$=$\widehat{\alpha}_{k-M-1}(s') \times \widehat{\gamma}_{k-M}(s', s) \times \widehat{\beta}_{k-M}(s)$ is computed. Finally, the value of $LLR_k$ at ($k$-$M$) is obtained as

$$LLR_k = \ln\left(\frac{\sum\limits_{(s',s)\to U_k=1} \widehat{\alpha}_{k-M-1}(s') \times \widehat{\gamma}_{k-M}(s', s) \times \widehat{\beta}_{k-M}(s)}{\sum\limits_{(s',s)\to U_k=0} \widehat{\alpha}_{k-M-1}(s') \times \widehat{\gamma}_{k-M}(s', s) \times \widehat{\beta}_{k-M}(s)}\right). \tag{3.24}$$

### 3.5.3 Mathematical Reformulation of Branch Metric Equations

Mathematical reformulation in this work can reduce memory-requirement of storing branch metrics in SISO unit. Applying Jacobian logarithm for $LLR_k$ expression in (3.14), it can be expressed as

$$\begin{aligned} LLR_k = \max_{(s',s)\to U_k=1}\{\alpha_{k-1}(s') + \gamma_k(s',s) + \beta_k(s)\} \\ - \max_{(s',s)\to U_k=0}\{\alpha_{k-1}(s') + \gamma_k(s',s) + \beta_k(s)\} \end{aligned} \tag{3.25}$$

where $\alpha_{k-1}(s') = \ln\{\widehat{\alpha}_{k-1}(s')\}$, $\beta_t(s') = \ln\{\widehat{\beta}_k(s)\}$ and $\gamma_k(s',s) = \ln\{\widehat{\gamma}_k(s',s)\}$ [21]. By substituting $\widehat{\gamma}_k(s',s)$ from (3.16) for $\gamma_k(s',s)$, the branch metric is represented as

$$\gamma_k(s',s) = \frac{1}{2} \times U_k \times L\left(U_k\right) + \frac{Lc}{2}\sum_{l=1}^{n} y_{kl} \times x_{kl}. \tag{3.26}$$

Considering a trellis structure with $\{1,(1+\mathrm{D}+\mathrm{D}^3)/(1+\mathrm{D}^2+\mathrm{D}^3)\}$ encoder transfer-function for $n$=2, the branch-metric expression from (3.26) can be expressed as

$$\gamma_k(s',s) = \frac{1}{2} \times U_k \times L(U_k) + \frac{Lc}{2}(x_{k1} \times y_{k1} + x_{k2} \times y_{k2}) \tag{3.27}$$

where $x_{k1}$ and $x_{k2}$ are systematic and parity bits, respectively, such that $x_{k1}\in\{+1,\text{-}1\}$ and $x_{k2}\in\{+1,\text{-}1\}$. Similarly, $y_{k1}$ and $y_{k2}$ are their respective soft values. The number of parent branch metrics are proportional to the value of $n$, such that $2^n$ parent branch metrics are required for each trellis stage and are given as

$$\begin{aligned} &\gamma_k(s'_0,s_0) = -\tfrac{1}{2} \times L\left(U_k\right) + \tfrac{Lc}{2}(-y_{k1} - y_{k2}), \\ &\gamma_k(s'_0,s_4) = \tfrac{1}{2} \times L\left(U_k\right) + \tfrac{Lc}{2}(y_{k1} - y_{k2}), \\ &\gamma_k(s'_4,s_2) = -\tfrac{1}{2} \times L\left(U_k\right) + \tfrac{Lc}{2}(-y_{k1} + y_{k2}) \ \ and \\ &\gamma_k(s'_4,s_6) = \tfrac{1}{2} \times L\left(U_k\right) + \tfrac{Lc}{2}(y_{k1} + y_{k2}). \end{aligned} \tag{3.28}$$

Among these parent branch metrics, $\gamma_k(s'_0,s_0)$ and $\gamma_k(s'_4,s_2)$ can be expressed using $\gamma_k(s'_4,s_6)$ and $\gamma_k(s'_0,s_4)$, respectively, as given below

$$\begin{aligned} \gamma_k(s'_0,s_0) &= -\left[\tfrac{1}{2} \times L\left(U_k\right) + \tfrac{Lc}{2}(y_{k1} + y_{k2})\right] = -\gamma_k(s'_4,s_6). \\ \gamma_k(s'_4,s_2) &= -\left[\tfrac{1}{2} \times L\left(U_k\right) + \tfrac{Lc}{2}(y_{k1} - y_{k2})\right] = -\gamma_k(s'_0,s_4). \end{aligned} \tag{3.29}$$

Reformulating the parent branch metric expression of $\gamma_k(s'_0, s_0)$ from (3.28), the value $L(U_k)$=$-Lc(y_{k1}+y_{k2})-2\times\gamma_k(s'_0, s_0)$, which is substituted in the second branch metric expression of $\gamma_k(s'_4, s_2)$ from (3.29) and it simplifies to

$$\begin{aligned} \gamma_k(s'_4, s_2) &= \gamma_k(s'_0, s_0) + Lc \times y_{k2} = -\gamma_k(s'_0, s_4) \\ \Rightarrow \gamma_k(s'_4, s_2) &= -\gamma_k(s'_4, s_6) + Lc \times y_{k2} = -\gamma_k(s'_0, s_4), \end{aligned} \tag{3.30}$$

since $\gamma_k(s'_0, s_0)$=$-\gamma_k(s'_4, s_6)$ from (3.29). Referring the reformulated equations for parent branch metrics from (3.29) and (3.30), a parent branch metric $\gamma_k(s'_4, s_6)$ needs to be computed as well as stored, for each of the trellis stages, and the rest can be derived as

$$\begin{aligned} &\gamma_k(s'_0, s_0) = -\gamma_k(s'_4, s_6), \\ &\gamma_k(s'_0, s_4) = \gamma_k(s'_4, s_6) - Lc \times y_{k2}, \ \textit{and} \\ &\gamma_k(s'_4, s_2) = -\gamma_k(s'_4, s_6) + Lc \times y_{k2}. \end{aligned} \tag{3.31}$$

For the practical implementation of SISO unit based on conventional SWBCJR algorithm, it has to store $2^n$ parent branch-metrics of each trellis stage for at least two sliding windows [60]. Thereby, if $n_\gamma$ represents the quantization of branch metric then the SISO-unit architecture must accommodate memory to store $2\times M\times 2^n\times n_\gamma$ bits for parent branch metrics. Unlike the conventional method, the SISO unit, which is based on branch metric reformulation of this work, needs to store $2\times M\times n_\gamma$ bits for the branch metrics. For example, if $M$=32 is used in the design of MAP decoder for $n$=2 and $n_\gamma$=8 then the decoder with branch-metric-reformulation can achieve 75% reduction in the memory requirement as compared to conventional SWBCJR algorithm. The overall saving of hardware resources due to reduced-memory for forward/backward-state-metrics and branch-metrics is referred as SBMS (state branch memory saving) in this work. Fig. 3.11 shows the percentages of SBMSs achieved by proposed and reported works. *Arch-1* presented in [53] has achieved a saving due to reduced memory required for forward state metrics, and its SBMS is 50%. Similarly, *Arch-2* designed in [54] has achieved SBMS of 26%. Low-power and reduced-memory design proposed in [55] has shown SBMSs of 24.9% and 19.6% for radix-2 (*Arch3a*) and radix-4 (*Arch3b*) architectures respectively. State-metric-compression based architecture *Arch4* [56] has SBMS of 50%, as shown in Fig. 3.11. Thus, the memory-reduced architecture presented in this work has shown better SBMS in comparison with the reported works.

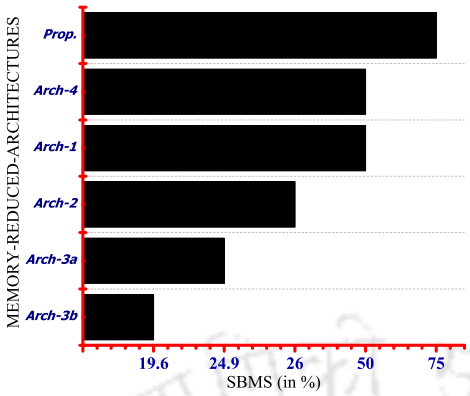


FIGURE 3.11: Comparison for the SBMSs (state branch memory savings) of proposed and reported SISO units w.r.t conventional SISO unit:
*Arch-1* [53], *Arch-2* [54], *Arch-3a* [55], *Arch-3b* [55] and *Arch-4* [56].

## 3.6 Architecture and Scheduling of SISO Unit

In this section, architecture and scheduling of SISO unit based on RSWMAP algorithm and branch-metric reformulation are presented.

### 3.6.1 Architecture

Fig. 3.12 shows SISO-unit architecture based on suggested memory-reduced techniques. Input soft values ($y_{k1}$ and $y_{k2}$) and $L(U_k)$ a-priori-information are fed to this decoder. These values are processed by BMC (branch metrics computation) sub module that computes $\gamma_k(s'_4, s_6)$ value, which is used for computing other parent branch metrics from (3.31) and the corresponding architecture of this sub module is shown in Fig. 3.13 (a). Its output is routed to three separate memories: *MEM1*, *MEM2* and *MEM3* via de-multiplexer, as shown in Fig. 3.12. Each of them stores $M \times n_\gamma$ bits for the branch metrics $\gamma_k(s'_4, s_6)$ $\forall$ $1 \leq k \leq M$. Fig. 3.12 shows that the outputs from these memory-units are multiplexed and are fed to BMR (branch metric router) sub module with architecture, as shown in Fig. 3.13 (b). It computes rest of the parent branch metrics $\gamma_k(s'_0, s_0)$, $\gamma_k(s'_0, s_4)$ and $\gamma_k(s'_4, s_2)$ from (3.31). An expression $\widehat{\beta}_M(s)$ is an estimated backward state metric which is derived in (3.23) and its logarithmic transformation is

$$\begin{aligned} \beta_M(s) = \ln\{\widehat{\beta}_M(s)\} &= \ln\left(\tfrac{1}{S_N} \sum_{all\ s''} \widehat{\gamma}_M(s'', s)\right) \\ &= \ln(1/S_N) + \max\{\gamma_k(s''_1, s), \gamma_k(s''_2, s)\} \end{aligned} \tag{3.32}$$

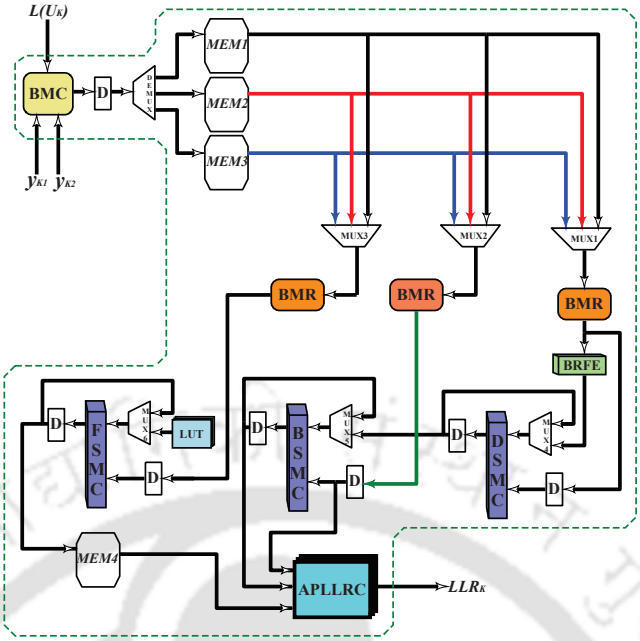


FIGURE 3.12: High-level architecture of SISO unit based on RSWMAP algorithm and reformulation of branch metric equation.

where $s'' \in \{s_1'', s_2''\}$ for radix-2 SISO-unit architecture. Such values are computed for all $S_N$ states to initiate the backward recursion. Fig. 3.13 (c) shows an architecture of BRFE (backward recursion factor estimator) sub module, which computes these values of estimated backward recursion factors $\beta_M(s_i)\ \forall\ i \in S_N$. For BRFE sub module, branch metrics at the input-side are fed to comparators and they determine the maximum values those are added with a constant value of $\ln(1/S_N)$ from LUT (look up table). In Fig. 3.12, the estimated backward state metrics from BRFE sub module are fed to DSMC (dummy state metric computation) sub module, which is used in the dummy-backward-recursion process of MAP decoding. It is a SMC unit that comprises of $S_N$ ACS (add compare select) units and computes backward state metrics for all states of the trellis stage [22]. DSMC sub module is fed with the branch metrics from BMR sub module and its own feedback outputs those are multiplexed with estimated backward state metrics from BRFE sub module, as shown in Fig. 3.12. Outputs from DSMC sub module is consecutively fed to BSMC sub module, which is also a SMC unit. It computes backward state metrics, using branch metrics and dummy backward state metrics obtained from BMR and DSMC sub modules, respectively, for successive trellis stages during backward recursion. Another sub module with feedback architecture is termed as FSMC that computes forward-state metrics for $S_N$ states during forward recursion, as shown in Fig. 3.12. In this process, the forward state metrics of first trellis stage must be initialized as $\alpha_{k=0}(s_i)=0\ \forall\ i=0$ and $\alpha_{k=0}(s_i)=$-1 $\forall$ $i \neq 0$. The computed forward-state metrics from FSMC sub module are stored in *MEM4* memory that can store $M \times S_N \times n_\alpha$ bits where $n_\alpha$ is the quantization of forward state metric. Finally,

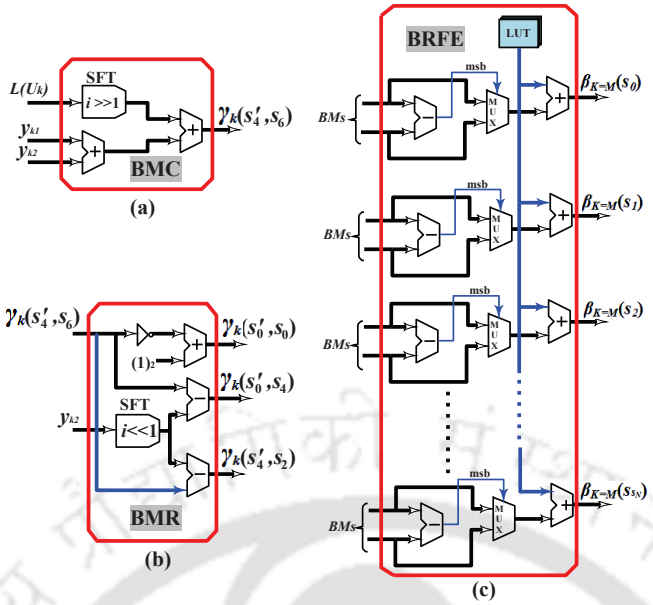


FIGURE 3.13: Logic-level architectures of (a) BMC (branch metrics computation) sub module (b) BMR (branch metric router) sub module (c) BRFE (backward recursion factor estimator) sub module. Here *BMs* indicates branch metrics.

branch metrics obtained from BMR sub module, backward state metrics computed by BSMC sub module and forward state metrics those are fetched from *MEM4* are fed to APLLRC (a-posteriori logarithmic likelihood ratio computation) sub module. It determines sum of $\alpha_{k-1}(s')$, $\beta_k(s)$ and $\gamma_k(s', s)$ for all the state transitions, then obtains maximum-values separately among these sums for the transitions $(s',s) \rightarrow U_k=1$ and $(s',s) \rightarrow U_k=0$. These maximum values are subtracted to obtain the value of $LLR_k$, as expressed in (3.25).

### 3.6.2 Scheduling

Scheduling for the decoding-process of SISO unit has been illustrated using timing-chart in this work, as shown in Fig. 3.14. Total time required for forward/backward recursion of the entire sliding window is denoted by $T_{SW}$. Forward, dummy-backward, backward recursions and the computation of $LLR_k$ at successive time-slots of various sliding windows while traversing the trellis stages are schematically illustrated in this timing-chart. Referring timing-chart and SISO-architecture from Fig. 3.14 and Fig. 3.12, respectively, systematic procedure of MAP decoding is explained as follows.

- In the time-slot $1 \leq t \leq T_{SW}$, branch metrics of $M$ trellis stages for the first-sliding-window are computed by BMC sub module and are stored in *MEM1*.

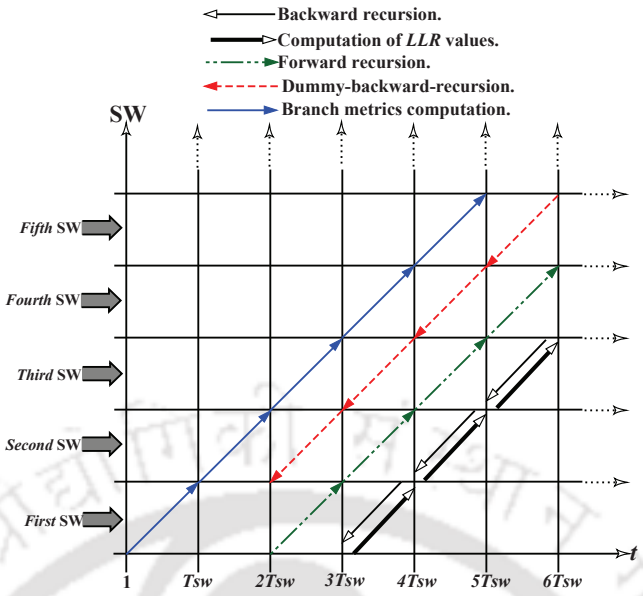


FIGURE 3.14: Timing-chart that illustrates scheduling of MAP decoding based on the suggested memory-reduced techniques.

- In the time-slot $T_{SW}<t\leq 2T_{SW}$, branch metrics of second-sliding-window are computed by BMC sub module and are stored in *MEM2*.

- In the time-slot $2T_{SW}<t\leq 3T_{SW}$, forward state metrics of $S_N$ states for $M$ trellis stages of first-sliding-window are computed by FSMC sub module, using the branch metrics fetched from *MEM1* as well as routed by BMR sub module. These forward state metrics are stored in *MEM4*. Simultaneously, BMC sub module computes branch metrics for third-sliding-window and stores them in *MEM3*. Using the branch metrics which are fetched from *MEM3* for the trellis stage $k$=2$M$, BRFE sub module estimates the backward state metric which is fed to DSMC sub module to start a dummy-backward-recursion for the first-sliding-window.

- In the time-slot $3T_{SW}<t\leq 4T_{SW}$, BSMC sub module is fed with backward state metrics estimated by DSMC sub module, and this BSMC sub module starts actual backward recursion to compute backward state metrics, which are fed to ALLRC sub module, for the first-sliding-window. Simultaneously, forward state metrics for first-sliding-window are fetched from *MEM4*, and are also fed to ALLRC sub module, along with the branch metrics of first-sliding-window from *MEM1*. Thereby, ALLRC sub module computes the values of $LLR_k$ $\forall$ $0\leq k\leq M$-1 using these values of backward state metrics, forward state metrics and branch metrics. Branch metrics for the fourth-sliding-window are computed and then stored in *MEM1*. Subsequently, estimation of backward state metrics and dummy-backward-recursion are performed for the second-sliding-window.

- In the time-slot $4T_{SW}<t\leq 5T_{SW}$, backward state metrics for second-sliding-window are determined during the actual backward recursion by BSMC sub module, using the branch metrics from *MEM2*, and these computed backward state metrics are fed to ALLRC. It computes $LLR_k \; \forall \; M\leq k\leq 2M$-1 using these backward state metrics, as well as forward state metrics and branch metrics of second-sliding-window from *MEM4* and *MEM2* respectively. Computation of forward state metrics and dummy-backward-recursion with backward state metric estimation for third-sliding-window are carried out. In addition, the branch metrics for fifth-sliding-window are computed by BMC sub module and stored in *MEM2*.
- This process of decoding successively continues until all the $N$ values of $LLR_k$ are obtained by SISO unit.

### 3.6.3 Comparative Analysis of Memory Requirement

Scheduling illustrated in the timing-chart of Fig. 3.14 has indicated that SISO-unit must store parent branch metrics $\gamma_k(s'_4, s_6)$ for three sliding windows. This implies that the memories *MEM1*, *MEM2* and *MEM3* for branch metrics have to store $3\times M\times n_\gamma$ bits. Similarly, forward state metrics of $M$ trellis stages where each stage has $S_N$ states are needed to be stored by *MEM4*. Such memory for forward state metrics must store $S_N\times M\times n_\alpha$ bits. Thereby, the total memory required by suggested SISO-unit architecture is

$$MEM_{siso} = M \times (3\times n_\gamma + S_N\times n_\alpha) \; bits. \tag{3.33}$$

For a SISO unit based on conventional SWBCJR algorithm [60], the memory required for forward state metrics is same as that of the suggested SISO unit. On the other side, such conventional SISO unit has to store $2^n$ parent branch metrics for each trellis stage, thereby, a total of $M\times(2\times 2^n\times n_\gamma + S_N\times n_\alpha)$ bits are necessary to be stored. Similarly, the conventional BCJR algorithm based SISO unit [18] needs to store forward state metrics, backward state metrics and parent branch metrics for the entire $N$ trellis stages. Hence, the memory required by such MAP decoder is $N\times(S_N\times n_\alpha + S_N\times n_\beta + 2^n\times n_\gamma)$ bits, where $n_\beta$ is the quantization of backward state metric. As we know that the turbo decoder with parallel architecture includes multiple SISO units. Such turbo decoder needs to store soft values for systematic and parity bits as well as the values for $N$ extrinsic information, since they are used in the iterative process of turbo decoding, as illustrated in Fig. 3.1. Table 3.6 shows the comparative analysis of memory required by parallel turbo decoders. It shows that the memory required by soft values and extrinsic information of the turbo decoder is $N\times(n\times n_\varphi + n_\varepsilon)$ bits, which remains constant for all the

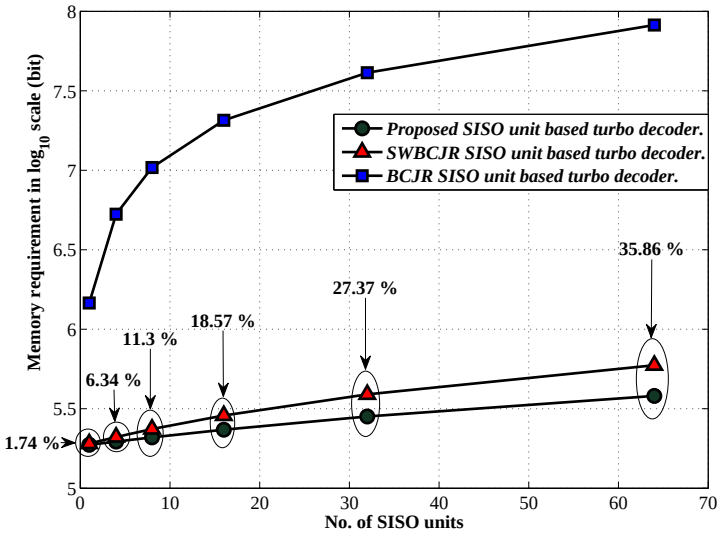


FIGURE 3.15: Memory required by parallel turbo decoder architectures using branch-metric reformulation, SWBCJR and BCJR algorithms based SISO units. The plot is shown for the values $N$=6144, $n$=3, $M$=32, $S_N$=8 and the quantization of ($n_\varepsilon$, $n_\varphi$, $n_\gamma$, $n_\alpha$, $n_\beta$)=(9, 7, 8, 9, 9, 8) bits.

TABLE 3.6: Comparison of the memory consumed by parallel turbo decoder based on different MAP algorithms

| **MAP algorithms** | Required memory by turbo decoder (bit) |
|---|---|
| **Proposed** | $N\times(n\times n_\varphi + n_\varepsilon) + P\times M\times(3\times n_\gamma + S_N\times n_\alpha)$. |
| **SWBCJR [60]** | $N\times(n\times n_\varphi + n_\varepsilon) + P\times M\times(2\times 2^n\times n_\gamma + S_N\times n_\alpha)$. |
| **BCJR [18]** | $N\times\{n\times n_\varphi + n_\varepsilon + P\times(S_N\times n_\alpha + S_N\times n_\beta + 2^n\times n_\gamma)\}$. |

$n_\varphi$: quantization of input soft-values of systematic and parity bits;
$n_\varepsilon$: quantization of extrinsic information;
$P$: total number of SISO units used in the parallel architecture of turbo decoder.

parallel architectures of turbo decoder. In order to evaluate the memory saving in parallel turbo decoder using SISO units based on the branch-metric reformulation, Fig. 3.15 shows the plots of memory consumed by turbo decoder for $P$= 1, 4, 8, 16, 32 and 64 number of SISO units in parallel. The proposed SISO unit based design of turbo decoder requires the least number of bits to be stored, as compared to SWBCJR and BCJR algorithm based decoders. The percentages of improvements achieved by the parallel turbo decoder for different values of $P$ are shown in Fig. 3.15. For a turbo decoder with parallel architecture of $P$=64, an improvement of 35.86% is obtained in comparison with the SWBCJR algorithm based turbo decoder.

## 3.7 Performance Analysis, Implementation Trade-offs and Comparison

In this section, BER performance analysis of SISO-units and parallel-turbo-decoders based on the suggested RSWMAP algorithm are carried out. From an implementation perspective, estimation of overall hardware saving achieved by parallel turbo decoders based on RSWMAP algorithm and branch-metrics reformulation are presented.

### 3.7.1 BER Performance

Fig. 3.16 shows the BER performance of SISO units with transfer function of $\{1, (1+D+D^3)/(1+D^2+D^3)\}$ and a code-rate of 1/2 in AWGN-channel environment using BPSK-modulation scheme. This performance analysis is carried out for the SISO units based on RSWMAP, SWBCJR and BCJR algorithms with $M$=32. Fig. 3.16 shows that the SISO unit with RSWMAP algorithm performs better than the conventional SWBCJR algorithm based SISO unit by 1.28 dB at a BER of $10^{-5}$. However, it has degraded performance of 0.21 dB, compared to BCJR algorithm based SISO

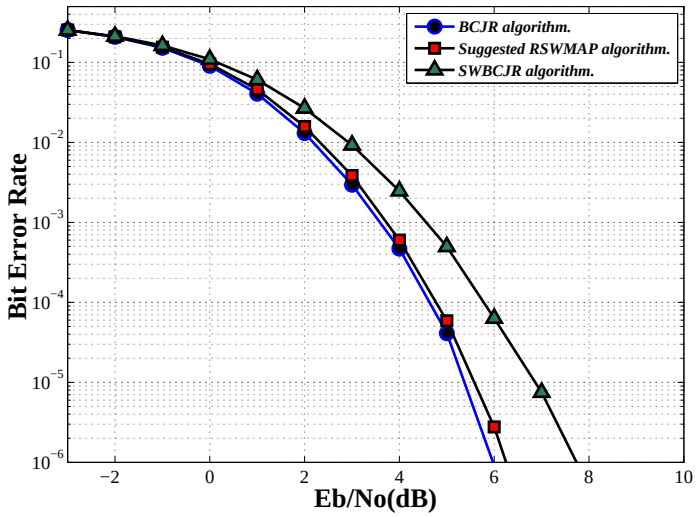


FIGURE 3.16: BER performance of SISO units based on different MAP algorithms for a code-rate of 1/2 and sliding window size of 32.

unit, at a BER of $10^{-5}$. Similarly, the BER performance of parallel turbo decoder, in AWGN channel-environment with BPSK modulation, for six decoding iterations is shown in Fig. 3.17. It shows that the BER performance of parallel turbo decoder based on RSWMAP algorithm for $M$=24 has a coding gain of 0.4 dB at a BER of $10^{-4}$ in comparison with the decoder based on SWBCJR algorithm for the same value of $M$=24. Subsequently, Fig. 3.17 shows that the

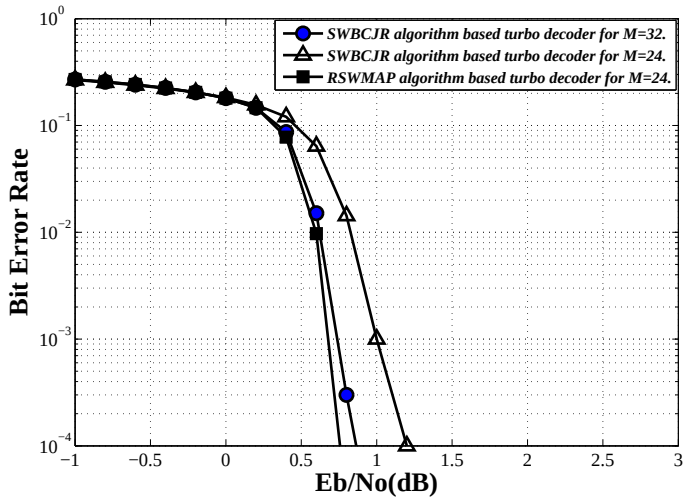


FIGURE 3.17: BER performance of parallel turbo decoders with $P$=64, based on different MAP algorithms for a code-rate of 1/3 and six decoding iterations.

SWBCJR algorithm based turbo decoder with $M$=32 has a similar BER performance as that of the RSWMAP algorithm based turbo decoder with $M$=24.

### 3.7.2 Implementation Trade-offs

Comparative study of BER performances has shown that the parallel turbo decoder based on RSWMAP algorithm achieves an adequate BER performance with smaller value of $M$ in comparison with the SWBCJR algorithm based parallel turbo decoder. A reduced sliding window size would require lesser memory for storing branch-metrics and forward-state-metrics. The branch-metric reformulation as well as the RSWMAP algorithm contribute to memory saving in SISO unit. From the implementation perspective, overall savings of hardware resources due to reduced-memory architecture of parallel turbo decoder, which uses SISO units based on branch-metric reformulation and RSWMAP algorithm, is presented here. Recently, the VLSI implementations of parallel turbo decoders with $P$=8 [52], $P$=16 [50], $P$=32 [51] and $P$=64 [74] have been reported for higher data-rate applications. Thereby, the hardware savings of parallel turbo decoders are analyzed up to $P$=64 parallel configuration. Such savings are accounted in terms of CMOS transistor count and the comparison is carried out with the parallel turbo decoder based on SWBCJR algorithm. Assuming that the memory used in parallel turbo decoder is SRAM (static random access memory), it requires six CMOS transistors to store each bit, as mentioned earlier [61]. Referring the expressions from Table 3.6, the parallel decoders based on proposed and conventional-SWBCJR algorithm consume $6\times\{N\times(n\times n_{\varphi}+n_{\varepsilon})+P\times M\times(3\times n_{\gamma}+S_N\times n_{\alpha})\}$ transistors and $6\times\{N\times(n\times n_{\varphi}+n_{\varepsilon})+P\times M\times(2^{n+1}\times n_{\gamma}+S_N\times n_{\alpha})\}$ transistors respectively. Fig.

3.18 shows the overall hardware savings in terms of CMOS transistor count for various parallel configuration of the decoder. From the previous BER analysis, it has been observed that the parallel turbo decoder based on RSWMAP algorithm can deliver optimum BER performance for $M$=24 rather than $M$=32 which is required by SWBCJR algorithm based decoder. Thereby,

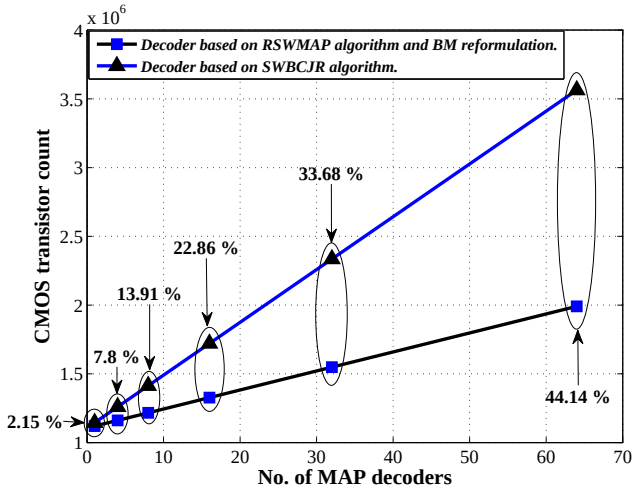


FIGURE 3.18: Hardware savings in terms of CMOS transistor counts for parallel turbo decoders based on the proposed and the SWBCJR algorithm based SISO units.

Fig. 3.18 shows the CMOS transistors consumed by turbo decoders based on suggested SISO unit for $M$=24 and SWBCJR algorithm based SISO unit for $M$=32. The percentage of hardware saving for different values of $P$ are shown in Fig. 3.18, and a maximum of 44.14% hardware resources are saved, due to the reduction of memory in parallel turbo decoder, for $P$=64.

## 3.8 Summary

This chapter presented architectural aspect and comparative BER-performance study of simplified MAP algorithms based on MSE [38] and PWLA [46]. It was observed that the algorithm based on reduced PWLA of $r$=4 delivered optimal BER performance and had lower critical-path delay that was suitable for high speed applications. Thereafter, SISO-unit architecture was designed for a sliding window size of 23 using such PWLA based simplified MAP algorithm. Subsequently, quantitative analysis of memory required by SISO unit in terms of bits as well as CMOS transistors consumed for various sliding window sizes, number of trellis states and data width of internal metrics were carried out. This quantitative model estimated that the memory required by proposed SISO unit consumed 17783 CMOS transistors. Non-parallel turbo-decoder architecture that incorporated suggested SISO unit and QPP interleaver was implemented at 130 nm CMOS technology node. It occupied a core area of 2.2 mm$^2$ and consumed 42.38 mW

of power at 303 MHz clock frequency. Subsequently, achievable throughput was estimated to be 28 Mbps with an energy efficiency of 0.28 nJ/b/iterations and it was suitable for WCDMA and HSDPA wireless communication standards. Analysis of achievable throughput for various configuration of turbo decoder architecture was also carried out. Finally, implemented turbo decoder was compared with the reported works and was able to achieve throughput that is better than those achieved by radix-2 and radix-4 non-parallel turbo decoders.

We have also suggested a method of estimating backward state metrics to initiate backward recursion for successive sliding windows during the MAP-decoding process. Consecutively, mathematical reformulation of branch-metric equations was performed, and this enabled SISO unit to store only single branch-metric for each trellis stage. Based on these methods, architecture and scheduling of a SISO unit was presented. Thereafter, comparative study on BER performance of parallel turbo decoders based on proposed and conventional methods were carried out, and the former had a coding gain of 0.4 dB at a BER of $10^{-4}$. The parallel turbo decoder with proposed SISO units has resulted in better coding performance and reduced-memory design. Finally, an overall hardware saving of this decoder was analyzed in terms of CMOS-transistor count and it has shown 44.14% saving in case of parallel turbo decoder with 64 SISO units.

# Chapter 4

# High-Throughput Turbo Decoder with Parallel Architecture for LTE Wireless Communication Standards

## 4.1 Introduction

With the advent of powerful smart phones and tablets, multimedia-wireless-communication has become an integral part of human life. In the year 2012, approximately 700 million such gadgets were estimated to be sold worldwide and there has been a huge demand of profound data rate by customers of mobile wireless services [75]. Such statistics clearly project greater challenges to deploy new standards that support higher data rates. With these motivations, 3GPP defined an air interface 3GPP-LTE Release-8, and then evolved it to 3GPP-LTE Release-9 in the year 2009, that has increased the spectrum efficiency more than 100 times and supported a peak data rate of 326.4 Mbps [76]. Since 2009, due to the scarcity of available contiguous spectrum allocated to wireless operators, carrier aggregation has been incorporated by 3GPP-LTE-Advanced to

achieve a peak data rate of 1 Gbps, as specified by ITUR for IMT-A (international mobile telecommunications advanced), which is also referred as 4G [78]. Thereby, signaling specification of a new air interface 3GPP-LTE-Advanced Release-10 was completed in 2011 which supported 1 Gbps downlink and 500 Mbps uplink peak data rates with up to 100 MHz bandwidth [77]. Based on the enhanced use of multi-antenna techniques and support for relay nodes, future LTE-Advanced Releases are expected to support peak data rate up to 3 Gbps milestone [75].

For reliable and error-free communications in these standards, turbo codes have been extensively used because of their inherently excellent near Shannon limit BER performances [1]. Iterative processing of soft values by MAP decoder [1] and pseudo-random scrambling/de-scrambling of extrinsic information by interleaver/de-interleaver are the essences for an excellent BER performance of turbo code. As we have discussed earlier, such iterative process has adverse effect that defers turbo decoders to achieve higher throughput benchmarks specified by recent wireless communication standards. Extensive research on the parallel turbo decoders has shown promising capabilities of achieving such benchmarks [48]. Thereby, maximum achievable throughput of such decoder with $P$ radix-$2^{\omega}$ MAP-decoders for a block length of $N$ and a sliding window size of $M$ is given as [49]

$$\Theta_T = \frac{P \times \omega \times F}{2 \times \rho} \times \frac{Z \times M/\omega}{(Z+2) \times M/\omega + \partial_{map} + \partial_{ext} + \partial_{dec}} \tag{4.1}$$

where $Z{=}N/M$, $F$ is maximum operating clock frequency, $\rho$ represents number of iterations, $\partial_{map}$ is pipeline delay for accessing data from memories to MAP decoders, $\partial_{ext}$ is pipeline delay for writing extrinsic information to memories and $\partial_{dec}$ is decoding delay of MAP decoder. This expression suggests that the achievable throughput of parallel turbo decoder has dominant dependencies on number of MAP decoders, clock frequency and number of iterations. Valuable contributions have been reported to improve these factors. Implementation of parallel turbo decoder which uses retimed and unified radix-$2^2$ MAP decoders for Mobile WiMAX and 3GPP-LTE standards has been presented in [68]. Similarly, parallel turbo decoder architecture with contention-free interleaver is designed for higher throughput applications in [50]. Reconfigurable and parallel architecture of turbo decoder with novel multistage interconnecting networks is implemented for 3GPP-LTE standard in [52]. Recently, a peak data rate of 3GPP-LTE standard has been achieved by parallel turbo decoder implemented in [29]. Processing schedule for parallel turbo decoder has been proposed to achieve 100% operating efficiency in [49]. High-throughput parallel turbo decoder suggested in [74] is based on algebraic-geometric properties of QPP interleaver. Architecture incorporating 16 × MAP decoders with an optimized state-metric

[1] Soft-decoding in SISO unit is based on MAP algorithm, thereby; SISO-unit will be refereed as MAP decoder throughout this chapter.

initialization scheme for low decoder latency and high throughput is presented in [79]. Another contribution of [80] includes very high throughput parallel turbo decoder for LTE-Advanced base station applications. Hybrid-decoder architecture for turbo as well as LDPC (low density parity check) codes compliant to multiple wireless communication standards has been proposed in [81].

Primary motive of this work is to conceive parallel turbo-decoder architecture for high throughput applications. We have focused on an improvement of maximum clock frequency ($F$) and this eventually improves an achievable throughput of parallel turbo decoder from (4.1). Works with similar motivations have been reported in the literature [82, 83] and [84]. So far, no work has reported parallel turbo decoder that can achieve higher throughput beyond 3 Gbps milestone targeted for the future releases of 3GPP-LTE-Advanced. The contributions of our work presented in this chapter are summarized as follows:

- We propose a modified MAP-decoder architecture based on a new un-grouped backward recursion scheme for the sliding window technique of LBCJR (logarithmic-Bahl-Cocke-Jelinek-Raviv) algorithm and a new state metric normalization technique. The suggested techniques have made provisions for retiming and deep-pipelining in the architectures of SMCU (state-metric-computation-unit) and MAP decoder, respectively, to speed up the decoding process.
- As a proof of concept, an implementation in 90 nm CMOS technology is carried out for the parallel turbo decoder with 8 $\times$ radix-2 MAP-decoders which are integrated with memories via pipelined interconnecting networks based on contention-free QPP interleavers. It is capable of decoding 188 different block lengths ranging from 40 to 6144 with a code-rate of 1/3 and achieves more than the peak data rate of 3GPP-LTE. We have also carried out synthesis-study and post-layout simulation of parallel turbo decoder with 64 $\times$ radix-2 MAP decoders that can achieve milestone throughput of 3GPP-LTE-Advanced.
- Subsequently, the fixed point simulation for BER performance analysis of parallel turbo decoder is carried out for various iterations, quantization and code rates.
- Finally, the key characteristics of parallel turbo decoder presented in this work are compared with the reported contributions from literature.

The remainder of this chapter is organized as follows. In section-4.2, brief discussion on transceiver-design for wireless communication and mathematical background of LBCJR algorithm as well as its sliding window technique are presented. Section-4.3 presents detail explanation of the modified sliding window approach and the state metric normalization technique. In section-4.4, VLSI

design as well as scheduling of high-speed MAP decoder architecture and discussion on parallel turbo decoder architecture are carried out. Section-4.5 includes BER performance evaluation of the turbo decoders, implementation details and comparison with the reported works. Finally, this chapter is summarized in section-4.6.

## 4.2 Theoretical Background

Basic transmitter and receiver schematic representations of the wireless communication device that is used for 3GPP-LTE/LTE-Advanced standards are shown in Fig. 4.1. Major functional blocks are segregated as digital-baseband module, analog-RF module and MIMO (multiple inputs multiple outputs) antennas. In digital-baseband module of the transmitter, sequence of information bits $U_k$ $\forall$ $k = \{1, 2, 3 \ldots\ldots N\}$ are processed by various sub-modules and are fed to the channel encoder. For each information bit of sequence $U_k$, a systematic bit $x_{sk}$ as well as

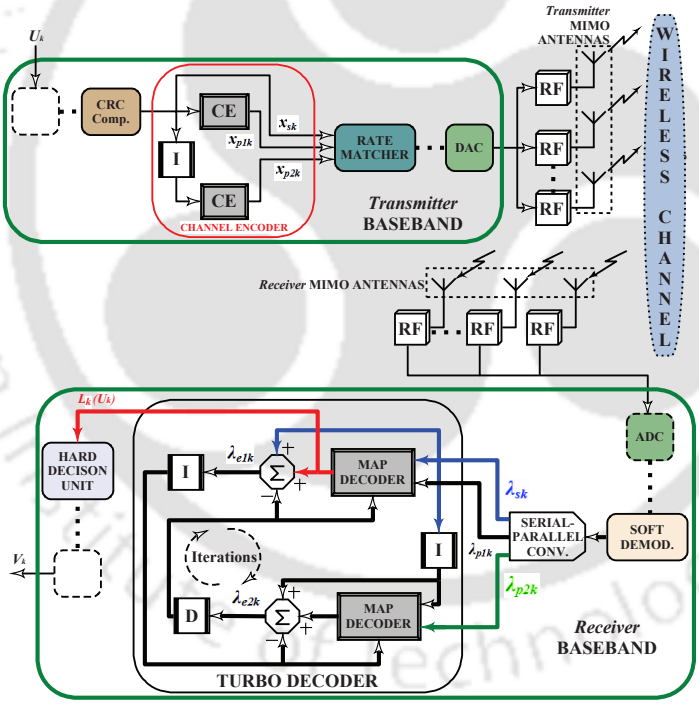


FIGURE 4.1: Basic block diagram of transmitter and receiver used for 3GPP-LTE/LTE-Advanced wireless communication standards.

parity bits $x_{p1k}$ and $x_{p2k}$ are generated by channel encoder using CEs (convolutional encoders) and I (QPP interleaver). These encoded bits are further processed by remaining submodules; finally, the output-digital data from baseband are converted into quadrature and in-phase analog signals by DAC. Analog signals fed to multiple analog-RF modules are up-converted to an RF frequency, amplified, band-passed and transmitted via MIMO antennas, which transform

RF signals into electromagnetic waves for transmission through wireless channel, as shown in Fig. 4.1. At the receiver, RF-signals provided by multiple antennas to analog-RF modules are band-pass filtered to extract signals of desired band. Then, they are low-noise-amplified and down-converted into baseband signals. Subsequently, these signals are sampled by ADC of the digital-baseband module where various sub-modules process such samples and are fed to soft-demodulator. It generates a-priori *LLR* values $\lambda_{sk}$, $\lambda_{p1k}$ and $\lambda_{p2k}$ for the transmitted systematic and parity bits, respectively, and are fed to turbo decoder via serial-parallel converter. We have already discussed in our earlier chapters that the turbo decoder works on graph-based approach in which MAP decoder uses BCJR algorithm to process input a-priori *LLRs* and then determines a-posteriori *LLR* values for the transmitted bits. As shown in Fig. 4.1, extrinsic information values are computed as $\lambda_{e1k} = \{\lambda_{sk} + L_{1k}(U_k) - \lambda^d_{e2k}\}$ and $\lambda_{e12k} = \{\lambda^i_{sk} + L_{2k}(U_k) - \lambda^i_{e1k}\}$ where $L_{1k}(U_k)$ and $L_{2k}(U_k)$ are a-posteriori *LLRs* from MAP decoders; $\lambda^d_{e2k}$ and $\lambda^i_{e1k}$ are de-interleaved and interleaved values of extrinsic information respectively. These extrinsic information values are iteratively processed by MAP decoders for maximum error control. Finally, a-posteriori *LLR* values those are generated by turbo decoder are processed by rest of the baseband sub-modules and sequence of decoded bits $V_k$ is obtained, as shown in Fig. 4.1.

Conventional BCJR algorithm performs mathematically-complex computations to deliver near-optimal error-rate performance albeit at the cost of huge memory and computationally-intense VLSI architecture that results in large decoding delay [18]. Thereby, logarithmic transformation of such miscellaneous mathematical equations of BCJR algorithm have scaled down the computational complexity and simplified implementation aspects of decoder architecture and this transformation is referred as LBCJR algorithm [21]. Furthermore, huge memory requirement and large decoding delay can be controlled with sliding window technique [36], as discussed earlier. It is a trellis-graph based decoding process in which $N$ stages are used for determining a-posteriori *LLRs* $L_k(U_k)$ $\forall$ $k$ = {1, 2, 3 .... $N$} and each stage comprises of $N_s$ trellis states. LBCJR algorithm traverses forward and backward of this graph to compute forward $\alpha_k(s_i)$ as well as backward $\beta_k(s_i)$ state metrics, respectively, for each trellis state such that $k \in N$ and $i \in N_s$. For states $s_0$ and $s_1$ from Fig. 4.2(a), forward and backward state metrics during their respective traces are computed as

$$\begin{aligned} \alpha_k(s_0) &= \widehat{max}\{\alpha_{k-1}(s'_0) + \gamma_k(s'_0, s_0), \alpha_{k-1}(s'_1) + \gamma_k(s'_1, s_1)\}, \\ \beta_k(s_1) &= \widehat{max}\{\beta_{k+1}(s''_0) + \gamma_k(s''_0, s_1), \beta_{k+1}(s''_1) + \gamma_k(s''_1, s_1)\}, \end{aligned} \tag{4.2}$$

respectively, where $\widehat{max}$ is a logarithmic approximation which simplifies mathematical computations of BCJR algorithm, as discussed in Chapter 3. Similarly, for an arbitrary state transition

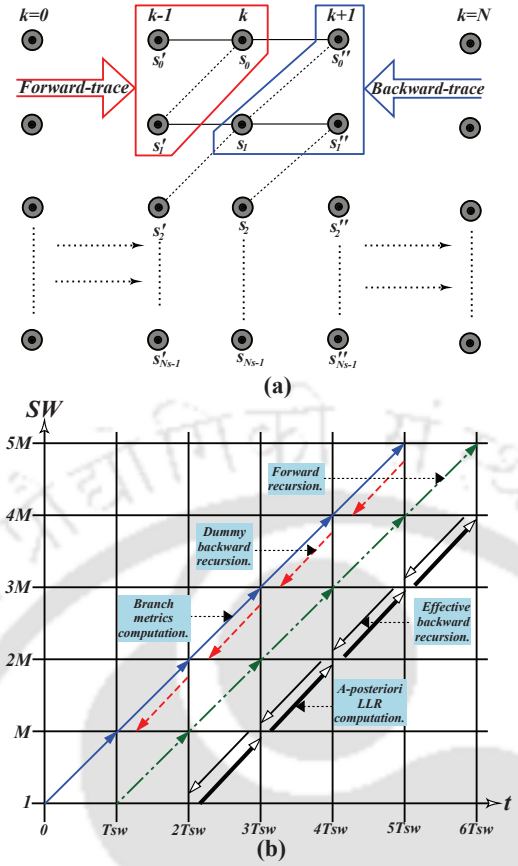


FIGURE 4.2: (a) Trellis graph with $N$ stages and $N_s$ trellis states. (b) Scheduling of sliding window technique for LBCJR algorithm, where x-axis and y-axis represent time and sliding-windows (SWs) respectively.

from $s'_i$ to $s_j$ such that $(i, j) \in N_s$, $\gamma_k(s'_i, s_j)$ is a branch metric which uses a-priori $LLRs$ for the computation and is expressed as

$$\gamma_k(s'_i, s_j) = 1/2 \cdot [U_k \cdot L_{uk} + Lc \cdot \{x_{sk} \cdot \lambda_{sk} + x_{p1k} \cdot \lambda_{p1k} + x_{p2k} \cdot \lambda_{p2k}\}], \tag{4.3}$$

such that $Lc \approx 2$ when the value of fading amplitude is $a$=1 [21]. A-posteriori $LLR$ value of a trellis stage is computed after the computation of all state and branch metrics. Assuming that $\delta$ represents trellis transition where $s^{st}(\delta)$ and $s^{en}(\delta)$ corresponds to start and end states, the a-posteriori $LLR$ value for $k^{th}$ trellis stage is computed as [21]

$$L_k(U_k) = \widehat{max}_{\delta:(s',s)\Rightarrow U_k=1}\{f(\delta)\} - \widehat{max}_{\delta:(s',s)\Rightarrow U_k=0}\{f(\delta)\} \tag{4.4}$$

where the function $f(\delta)$ is expressed as

$$f(\delta) = \alpha_{k-1}\{s^{st}(\delta)\} + \gamma_k(\delta) + \beta_k\{s^{en}(\delta)\}. \tag{4.5}$$

Additionally, $\delta : (s', s) \Rightarrow U_k$=0/1 indicates set of all trellis transitions when the information bit is $U_k$=0/1. Fig. 4.2(b) shows time-scheduling for sliding window technique of LBCJR (SW-LBCJR) algorithm for various operations those are carried out in successive sliding windows (SWs) [60]. In the first time-slot $T_{sw}$, branch metrics of the first SW (SW1) are computed. Subsequently, branch metrics for SW2 as well as dummy-backward-recursion that estimates boundary backward state metrics for SW1 are accomplished in the time-interval $T_{sw} < t \leq 2T_{sw}$. Similarly, effective-backward-recursion for SW1 is initiated during the interval $2T_{sw} < t \leq 3T_{sw}$ where the computation of a-posteriori *LLRs* for SW1 begins simultaneously and other operations such as dummy-backward and forward recursions run in parallel during this interval. Moreover, such process is carried out successively for all the SWs, as shown in Fig. 4.2(b). Thereby, conventional SW-LBCJR algorithm has a decoding delay of $2T_{sw}$ and it needs to store branch metrics for two SWs as well as forward state metrics for one SW [60].

## 4.3 Proposed Techniques

This section presents modified sliding window approach and state metric normalization technique for LBCJR algorithm.

### 4.3.1 A Modified Sliding Window Approach

This work presents an un-grouped backward recursion technique for LBCJR algorithm. Unlike conventional SW-LBCJR algorithm, the suggested approach performs backward recursion for each trellis stage independently for the computation of backward state metrics. For a sliding window size of $M$, such an un-grouped backward recursion for $k^{th}$ stage begins from $(k+M\text{-}1)^{th}$ trellis stage. Each of these backward recursions is initiated with logarithmic-equiprobable values assigned to all backward state metrics of $(k+M\text{-}1)^{th}$ trellis stage as

$$\beta_{k+M-1}(s_j) = \ln(1/N_s) \; \forall \; j \in N_s. \tag{4.6}$$

Simultaneously, the branch metrics are computed for successive trellis stages and are used for determining state metric values using (4.2). After computing $N_s$ backward state metrics of $k^{th}$ trellis stage using un-grouped backward recursion, all the forward state metrics of $(k\text{-}1)^{th}$ trellis

stage are computed. It is to be noted that the forward recursion starts with initialization at $k$=0 such that

$$\alpha_{k=0}(s_{i=0}) = 0 \; and \; \alpha_{k=0}(s_i) = -\infty, \; i \neq 0. \tag{4.7}$$

Thereafter, a-posteriori *LLR* value of $k^{th}$ trellis stage is computed using the branch metrics of all state transitions, as well as forward and backward state metrics from $(k\text{-}1)^{th}$ and $k^{th}$ trellis stages, respectively, as given in (4.4). Paralleling such un-grouped backward recursions for successive trellis stages to compute their a-posteriori *LLRs* using LBCJR algorithm is a primary concern of our approach. For the sake of clarity, we have used handful of new notations while explaining this approach for LBCJR algorithm. For example, $\mathbf{B}_k$ and $\mathbf{A}_k$ represent sets of $N_s$ backward and forward state metrics of $k^{th}$ trellis stage, respectively, and they are given as $\mathbf{B}_k = \{\beta_k(s_i) \mid i \in \mathbf{N}^0, 0 \leq i < N_s\}$ and $\mathbf{A}_k = \{\alpha_k(s_i) \mid i \in \mathbf{N}^0, 0 \leq i < N_s\}$ where $\mathbf{N}^0$ is a set of natural numbers including zero. Similarly, a set of all branch metrics, associated with the transitions from $(k\text{-}1)^{th}$ to $k^{th}$ trellis stages, is denoted by $\Gamma_k$ which is expressed as $\Gamma_k$=$\{\gamma_k(\chi) \mid \chi$ is a set of all state transitions$\}$. Multiple un-grouped backward recursions are involved in this approach; thereby, we have denoted $\mathbf{B}_k$ for different un-grouped backward recursions as $\{\mathbf{B}_k\}^u$ such that $u \in \boldsymbol{U}$ and $\boldsymbol{U}$ is a set of all un-grouped backward recursions for each time instant. Fig. 4.3 illustrates the un-grouped backward recursions for a value of

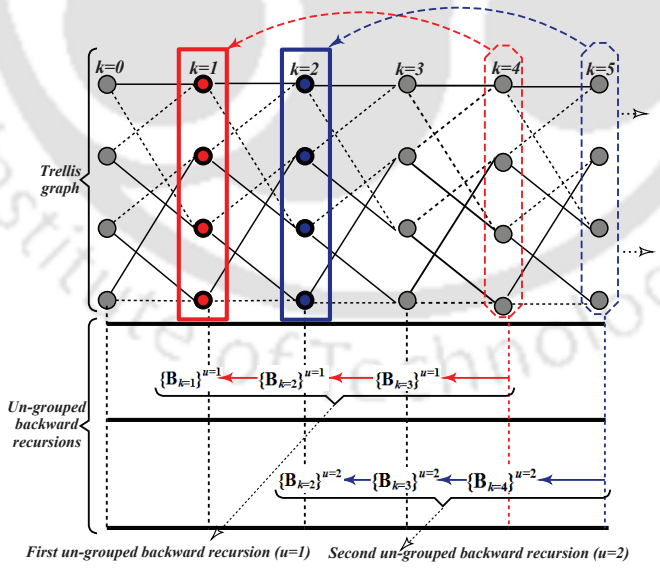


FIGURE 4.3: Illustration of un-grouped backward recursions in four-state trellis graph, with $M$=4, for trellis stages $k$=1 and $k$=2.

$M$=4 and the computation of backward state metrics for $k$=1 and $k$=2 trellis stages. First un-grouped backward recursion (denoted by $u$=1) starts with the computation of $\{\mathbf{B}_{k=3}\}^{u=1}$

using the initialized backward state metrics from $k$=4 trellis stage. Thereafter, $\{\mathbf{B}_{k=2}\}^{u=1}$ is computed using $\{\mathbf{B}_{k=3}\}^{u=1}$; finally, an effective set of backward state metric $\{\mathbf{B}_{k=1}\}^{u=1}$, which is then used in the computation of a-posteriori *LLR* for $k$=1 trellis stage, is obtained using the value of $\{\mathbf{B}_{k=2}\}^{u=1}$. Similarly, such successive process of second un-grouped backward recursion ($u$=2) is carried out to compute an effective-set of $\{\mathbf{B}_{k=2}\}^{u=2}$ for $k$=2 trellis stage, as shown in Fig. 4.3. In the suggested approach, time-scheduling of various operations to be performed for the computation of successive a-posteriori *LLRs* is schematically presented in Fig. 4.4. This

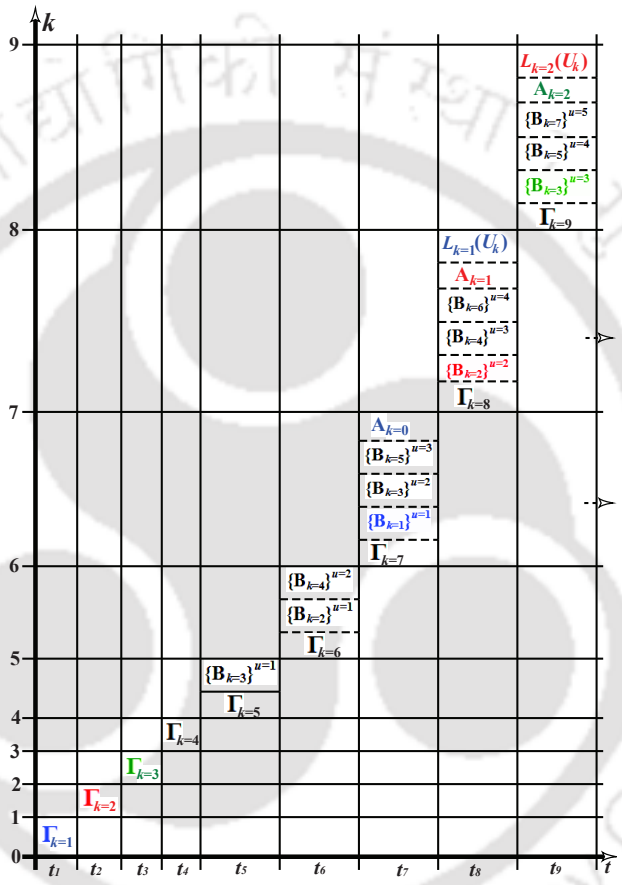


FIGURE 4.4: Scheduling of the modified sliding window approach for LBCJR algorithm based on un-grouped backward recursion technique for $M$=4.

scheduling is illustrated for $M$=4, where the trellis stages and time intervals are plotted along y-axis and x-axis respectively. As the time progresses, a set of branch metrics (denoted by $\Gamma_k$) is computed in each time interval. Thereby, $\Gamma_k$ $\forall$ $1 \leq k \leq 9$ are successively computed from the time interval $t_1$ to $t_9$, as shown in Fig. 4.4. Similarly, un-grouped backward recursions begin from $t_4^{th}$ time interval because branch metrics required for these recursions are available from this interval onwards. As illustrated in Fig. 4.4, operations performed from this interval onwards are systematically explained as follows.

$t_5$**:** A first un-grouped backward recursion ($u$=1) begins with the computation of $\{\mathbf{B}_{k=3}\}^{u=1}$ which uses initialized backward state metrics from $k$=4 trellis stage. Since this backward recursion is performed to compute an effective-set of backward state metrics for $k$=1, it is initiated from $k$+$M$-1=4 trellis stage.

$t_6$**:** A consecutive-set $\{\mathbf{B}_{k=2}\}^{u=1}$ is computed for the continuation of first un-grouped backward recursion. Simultaneously, a second un-grouped backward recursion starts from the initialized trellis stage $k$=5, with the computation of a new-set $\{\mathbf{B}_{k=4}\}^{u=2}$.

$t_7$**:** First un-grouped backward recursion ends in this interval with the computation of effective-set $\{\mathbf{B}_{k=1}\}^{u=1}$ for $k$=1 trellis stage. In Parallel, second un-grouped backward recursion continues with the computation of consecutive-set $\{\mathbf{B}_{k=3}\}^{u=2}$. Similarly, a new-set $\{\mathbf{B}_{k=5}\}^{u=3}$ is computed and it marks a start of third un-grouped backward recursion. Initialization of all the forward state metrics of set $\mathbf{A}_{k=0}$ is also carried out, as given in (4.7).

$t_8$**:** An effective-set $\{\mathbf{B}_{k=2}\}^{u=2}$ is obtained with the termination of second un-grouped backward recursion and a consecutive-set $\{\mathbf{B}_{k=4}\}^{u=3}$ is computed for an ongoing third un-grouped backward recursion. At the same time, fourth un-grouped backward recursion begins with the computation of a new-set $\{\mathbf{B}_{k=6}\}^{u=4}$. Using an initialized set $\mathbf{A}_{k=0}$, a set of forward state metrics $\mathbf{A}_{k=1}$ is determined. A-posteriori *LLR* value $L_{k=1}(U_k)$ of the trellis stage $k$=1 is computed using forward, backward and branch metrics from the sets $\mathbf{A}_{k=0}$, $\{\mathbf{B}_{k=1}\}^{u=1}$ and $\Gamma_{k=1}$ respectively.

$t_9$**:** From this interval onwards, similar pattern of operations are carried out in each time-interval where an un-grouped backward recursion is terminated with the calculation of an effective-set, a consecutive-set is obtained to continue an incomplete un-grouped backward recursion and a new-set is determined using the initialized values of backward state metrics to start an un-grouped backward recursion. Simultaneously, sets of forward state metrics and a-posteriori *LLRs* for successive trellis stages are obtained from $t_9$ time interval onwards.

Decoding delay $\partial_{dec}$ for the computation of a-posteriori *LLRs* for $M$=4 is a sum of seven time-intervals ($\partial_{dec}$=$\Sigma_{j=1}^{7} t_j$), as shown in Fig. 4.4. Thereby, it can be concluded that the decoding delay of this approach is $\partial_{dec}$=$(2 \times T_{sw}) - 1$. It can be seen that from $t_7$ time-interval onwards, three $\{\mathbf{B}_k\}^u$ sets are simultaneously computed in each interval. Thereby, in general, this approach requires $M$-1 units to accomplish such parallel task. However, implementation aspects of the MAP decoder based on this approach is discussed in section-4.4.

### 4.3.2 A State Metric Normalization Technique

Magnitudes of forward and backward state metrics grow as recursions proceed in the trellis graph. Overflow may occur without normalization, if the data widths of these metrics are finite. There are two commonly used state metric normalization techniques: subtractive and modulo normalization techniques [24]. In the subtractive normalization technique, normalized forward and backward state metrics for $k^{th}$ trellis stage are computed as

$$\begin{aligned} \alpha_k(s_i)^* &= \left[\alpha_k(s_i) - \max_{j:0\leq j<N_s}\{\alpha_{k-1}(s_j)\}\right], i \in N_s \ \ and \\ \beta_k(s_i)^* &= \left[\beta_k(s_i) - \max_{j:0\leq j<N_s}\{\beta_{k+1}(s_j)\}\right], i \in N_s \end{aligned} \tag{4.8}$$

respectively [24]. On the other side, two's complement arithmetic based modulo normalization technique works with a principle that the path selection process during forward/backward recursion depends on bounded values of path metric difference [85]. The normalization technique suggested in our work is focused to achieve high-speed performance of turbo decoder from an implementation perspective. Assume that the states $s'_x$ and $s'_y$ at ($k$-1)$^{th}$ stage as well as $s''_x$ and $s''_y$ states at ($k$+1)$^{th}$ stage are associated with $s_x$ state at $k^{th}$ stage in trellis graph. Thereby, normalization of a forward state metric for $s_x$ state at $k^{th}$ trellis stage is carried out as

$$\alpha_k(s_x)^* = \max\left[\{\Gamma^{p1}_{k'} - \alpha_{k-1}(s'_i)\}, \{\Gamma^{p2}_{k'} - \alpha_{k-1}(s'_i)\}\right], i \in N_s \tag{4.9}$$

where $\Gamma^{p1}_{k'}$ and $\Gamma^{p2}_{k'}$ are path metrics for the transitions from $s'_x$ and $s'_y$ to $s_x$, respectively, and are expressed as $\Gamma^{p1}_{k'}$=$\{\alpha_{k-1}(s'_x) + \gamma_k(s'_x, s_x)\}$ and $\Gamma^{p2}_{k'}$=$\{\alpha_{k-1}(s'_y) + \gamma_k(s'_y, s_x)\}$. The normalizing factor $\alpha_{k-1}(s'_i)$ from (4.9) is one of the previously computed forward state metrics of $N_s$ states from ($k$-1)$^{th}$ trellis stage. Similarly, a backward state metric at $k^{th}$ trellis stage can be normalized as

$$\beta_k(s_x)^* = \max\left[\{\Gamma^{p1}_{k''} - \beta_{k+1}(s''_j)\}, \{\Gamma^{p2}_{k''} - \beta_{k+1}(s''_j)\}\right], j \in N_s \tag{4.10}$$

where $\Gamma^{p1}_{k''}$=$\{\beta_{k+1}(s''_x) + \gamma_k(s''_x, s_x)\}$ and $\Gamma^{p2}_{k''}$=$\{\beta_{k+1}(s''_y) + \gamma_k(s''_y, s_y)\}$ are the path metrics. Similarly, the normalizing factor is $\beta_{k+1}(s''_j)$ from a state among $N_s$ trellis states at ($k$+1)$^{th}$ stage. It is to be noted that such normalizing factors $\alpha_{k-1}(s'_i)$ and $\beta_{k+1}(s''_j)$ can be used for computing all $N_s$ normalized forward and backward state metrics, respectively, at $k^{th}$ trellis stage.

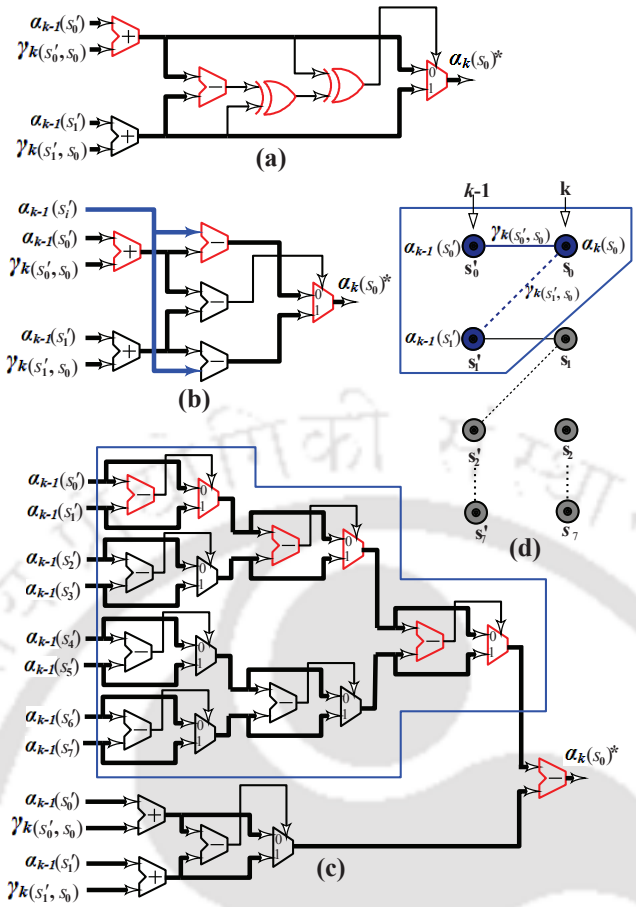


FIGURE 4.5: (a) An ACSU for modulo normalization technique [28] (b) An ACSU for suggested normalization technique (c) An ACSU for subtractive normalization technique [24] (d) Part of a trellis graph with $N_s$=8 showing $(k\text{-}1)^{th}$ and $k^{th}$ trellis stages and metrics involved in the computation of forward state metric at $s_0$ trellis state.

From the implementation perspective, an ACSU (add compare select unit) is used for computing such normalized state metric in the MAP decoder and it requires $N_s$ ACSUs to compute all the forward/backward state metrics for each trellis stage. Fig. 4.5 shows the comparison of ACSU architectures based on suggested approach, modulo and subtractive normalization techniques. These ACSUs can be used for computing a normalized forward state metric at $s_0$ state of a trellis graph with $N_s$=8 states, as shown in Fig. 4.5(d). An ACSU design that is used in our work, based on (4.9) is shown in Fig. 4.5(b). In this architecture, path metrics are subtracted with a normalizing factor $\alpha_{k-1}(s'_i)$ using subtractors along second stage and then multiplexed to obtain a normalized forward state metric $\alpha_k(s_0)^*$. Similarly, the state-of-the-art architecture of ACSU for modulo normalization technique is presented in Fig. 4.5(a) and it achieves normalized forward state metric value with controlled overflow using two two-input-XOR gates [24]. However, an ACSU for subtractive normalization technique requires additional comparator circuit to obtain a value of $\max_{j:0\leq j<N_s}\{\alpha_{k-1}(s_j)\}$ from (4.8), as shown in Fig. 4.5(c), and it includes comparator circuit for $N_s$=8 trellis states. Thereafter, a maximum value obtained is subtracted with

TABLE 4.1: Comparison of SMCUs for different state metric normalization techniques

| **Design metrics** | This work | [28]‡ | [24]♮ |
|---|---|---|---|
| **Technology (nm)** | 90 | 90 | 90 |
| **Supply voltage (V)** | 0.9 | 0.9 | 0.9 |
| **Design area ($\mu m^2$)** | 14531 | 13656 | 17693 |
| **Power (mW) @ 100 MHz** | 1.88 | 1.84 | 2.0 |
| **Maximum clock frequency (MHz)** | 306.75 | 239.81 | 120.34 |

‡: SMCU based on modulo normalization technique.

♮: SMCU based on subtractive normalization technique.

the state metric to compute its normalized value. These architectures of ACSUs are presented for max-log-MAP LBCJR algorithm for high-speed applications [21]. However, its degradation in BER performance, as compared to Log-MAP LBCJR algorithm, may be avoided by using an extrinsic scaling process [57]. Critical paths of ACSUs based on suggested approach, modulo and subtractive normalization techniques are highlighted in Fig. 4.5(a-c) and are quantified as

$$\begin{aligned} \urcorner_{new} &= \tau_{add} + \tau_{sub} + \tau_{mux}, \\ \urcorner_{mod} &= \tau_{add} + \tau_{sub} + \tau_{mux} + \tau_{xor} + \tau_{xor}, \ \ and \\ \urcorner_{sub} &= \tau_{sub} + \tau_{sub} + \tau_{sub} + \tau_{sub} + \tau_{mux} + \tau_{mux} + \tau_{mux} \end{aligned} \tag{4.11}$$

respectively, where $\tau_{add}$, $\tau_{sub}$, $\tau_{mux}$ and $\tau_{xor}$ are the delays imposed by an adder, a subtractor, a multiplexer and an XOR gate respectively. In this work, stack of $N_s$ ACSUs for computing all the forward/backward state metrics are collectively referred as SMCU. We have performed a post-layout simulation study, in 90 nm CMOS process, of SMCUs with $N_s$=8 based on these state metric normalization techniques and their key characteristics obtained are presented in Table 4.1. Subsequently, design-synthesis and static-timing-analysis are performed under worst corner case with a supply of 0.9 V at $125^0$C operating temperature. It can be seen that SMCU based on the suggested approach have 21.82% and 60.77% better operating clock frequencies than the SMCUs based on modulo and subtractive normalization techniques respectively. Subsequently, SMCU used in this work consumes 17.87% lesser silicon-area than SMCU based on subtractive normalization technique. However, it has area overhead of 6.02% in comparison with modulo normalization based SMCU. Total power consumed at 100 MHz clock frequency by this SMCU is 6% lesser and 2.13% more than subtractive and modulo normalization techniques, respectively, as shown in Table 4.1. Among these implementations, suggested approach for the state metric normalization technique has shown better operating clock frequency at the expenses of nominal

degradations, in terms of area and power consumed, as compared to modulo normalization technique.

## 4.4 Decoder Architectures and Scheduling

This section presents MAP-decoder architecture and its scheduling based on the proposed techniques. We have further discussed design and implementation-trade-offs of high-speed MAP-decoder architecture. Then, parallel turbo-decoder architecture and interleaver used in this work are presented.

### 4.4.1 MAP Decoder Architecture and Scheduling

Proposed decoder architecture for LBCJR algorithm based on un-grouped backward recursion technique is presented in Fig. 4.6. It includes five major sub blocks: BMCU (branch metric computation unit), ALCU (a-posteriori *LLR* computation unit), RE (registers), LUT (look up table) and SMCU that uses suggested state metric normalization technique to compute state metric values. The BMCU processes $n$ a-priori *LLRs* of systematic and parity bits ($\lambda_{sk}$, $\lambda_{p1k}$

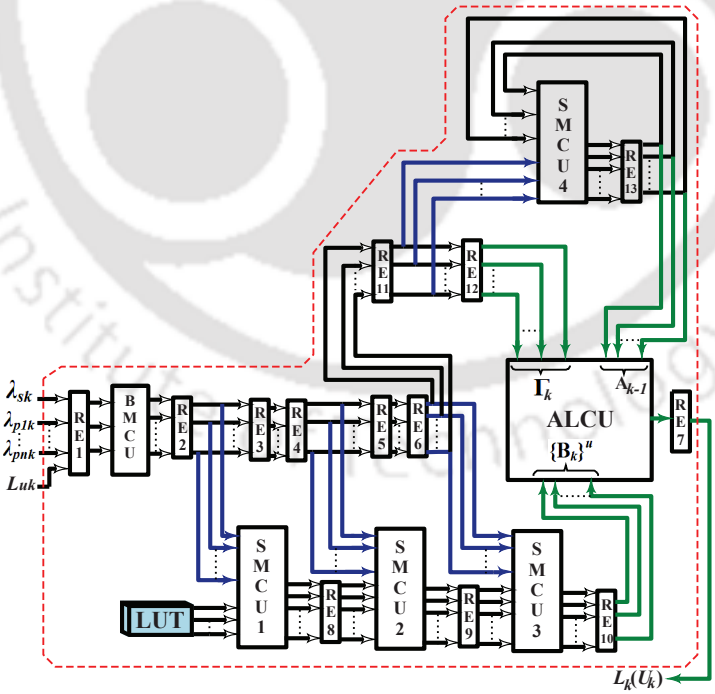


FIGURE 4.6: High-level architecture of the proposed MAP decoder, based on modified sliding window technique, for $M$=4.

..... $\lambda_{pnk}$), where $n$ is a code-length, to successively compute all branch metrics in each of

the sets $\Gamma_k$ $\forall$ $1 \leq k \leq N$. A-posteriori *LLR* value for $k^{th}$ trellis stage is computed by ALCU using the sets of state and branch metrics, as shown in Fig. 4.6. Sub block RE is a bank of registers

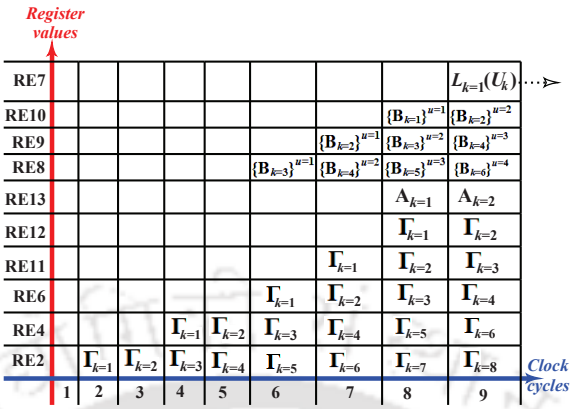


FIGURE 4.7: Launched values of state and branch metric sets as well as a-posteriori *LLRs* by different registers of MAP decoder in successive clock cycles.

used for data-buffering in the MAP decoder. Subsequently, LUT stores logarithmic equiprobable values, as given in (4.6), for backward state metrics of $(k+M\text{-}1)^{th}$ trellis stage and it initiates un-group backward recursion for $k^{th}$ trellis stage. As discussed earlier, SMCU is used for computing $N_s$ forward or backward state metrics for each trellis stage. Based on the time-scheduling that is illustrated in Fig. 4.4, we have presented architecture of MAP decoder for $M$=4 in Fig. 4.6. Thereby, three ($M$-1) SMCUs are used for un-grouped backward recursions in this decoder architecture and are denoted as SMCU1, SMCU2 and SMCU3. Similarly, forward state metrics for successive trellis stages are computed by SMCU4. For better understanding of the decoding process, a graphical representation of data launched by different registers in the decoder architecture for successive clock cycles are illustrated in Fig. 4.7.

In this decoder architecture, input a-priori *LLRs* as well as a-priori information $L_{uk}$ for the successive trellis stages are sequentially buffered through RE1 and then processed by BMCU, which computes all the branch metrics of these stages, as shown in Fig. 4.6. These branch metric values are buffered through series of registers and are fed to SMCUs, those are assigned for backward recursion, as well as SMCU4 and ALCU for forward recursion and *LLR* computation respectively. In fifth clock cycle, branch metrics of $\Gamma_{k=4}$ set are launch from RE2 and are used by SMCU1 along with initial values of backward state metrics from LUT to compute backward state metrics of $\{\mathbf{B}_{k=3}\}^{u=1}$ for the first un-grouped backward recursion and then stored in RE8, as shown in Fig. 4.7. These stored values of RE8 are launched in sixth clock cycle and are fed to SMCU2 along with a branch metric set $\Gamma_{k=3}$ from RE4 to compute a set $\{\mathbf{B}_{k=2}\}^{u=1}$, which is stored in RE9. In the same clock cycle, $\{\mathbf{B}_{k=4}\}^{u=2}$ for second un-grouped backward recursion are computed by SMCU1 using $\Gamma_{k=5}$ launched from RE2 and are stored in RE8. Both these sets

of backward state metrics are launched by RE8 and RE9 in seventh clock cycle, as illustrated in Fig. 4.7. It can be observed that the similar pattern of computations for branch and state metrics are carried out for successive trellis stages, as shown in Fig. 4.7. Branch metric sets from RE11 are used by SMCU4 to compute sets of forward-state metrics $\mathbf{A}_k$ for successive trellis stages. Fig. 4.6 and Fig. 4.7 shows that the sets of forward state, backward state and branch metrics are fed to ALCU via RE13, RE10 and RE12, respectively. Thereby, a-posteriori *LLRs* are successively generated by ALCU from ninth clock cycle onwards, for the value of $M$=4, as shown in Fig. 4.7. From an implementation perspective, decoding delay $\partial_{dec}$ of this MAP decoder is $2\times M$ clock cycles.

### 4.4.2 Retimed and Deep-pipelined Decoder Architecture

In the suggested MAP decoder architecture, SMCU4 with buffered feedback paths is used in forward recursion and impose critical path delay of $\urcorner_{new}$ from (4.11), as discussed in section-4.3. On the other hand, architecture of SMCU4 can be retimed to shorten the critical path delay of this decoder. For the trellis-graph of $N_s$=4, retimed data-flow-graph of SMCU, with buffered feedback paths, that computes forward state metrics of successive trellis stages is shown in Fig. 4.8(a). It has four ACSUs based on suggested state metric normalization technique and they compute forward state metrics using $\alpha_{k-1}(s'_1)$ normalizing factor. However, this retimed data-flow-graph based architecture has to operate with clock (*clk2*) at twice the frequency of clock (*clk1*) with which the branch metrics are fed, as shown in Fig. 4.8(b). Otherwise, it may miss the successive forward state metrics from ($k$-1)$^{th}$ stage to compute state metrics for $k^{th}$ trellis stage. It can be seen that the critical path of this SMCU has a subtractor-delay only; thereby, this retimed-unit can be operated at much higher clock frequency $f_{clk2}$. However, remaining units of MAP decoder such as BMCU, ALCU and SMCUs, those are used for un-grouped backward recursions, must operate at a clock frequency of $f_{clk1}$=$f_{clk2}$/2. Fortunately, all these units in our decoder are feed-forward digital architectures those are suitable for deep-pipelining. In general, BMCU and ALCU are combinational designs and can be pipelined with ease. An advantage of the suggested MAP decoder architecture is that, SMCUs involved in the backward recursion can also be pipelined which increases an actual data-processing frequency ($f_{clk1}$) at which the branch metrics are fed to retimed SMCU that is already operating at much higher clock frequency. However, such retimed SMCU is not suitable for conventional MAP decoders because the SMCUs, for backward recursion in these designs, have feedback architectures and they cannot be pipelined to enhance the data-processing frequency; though, the retimed SMCU for forward recursion are operating at higher clock frequency [28, 29].

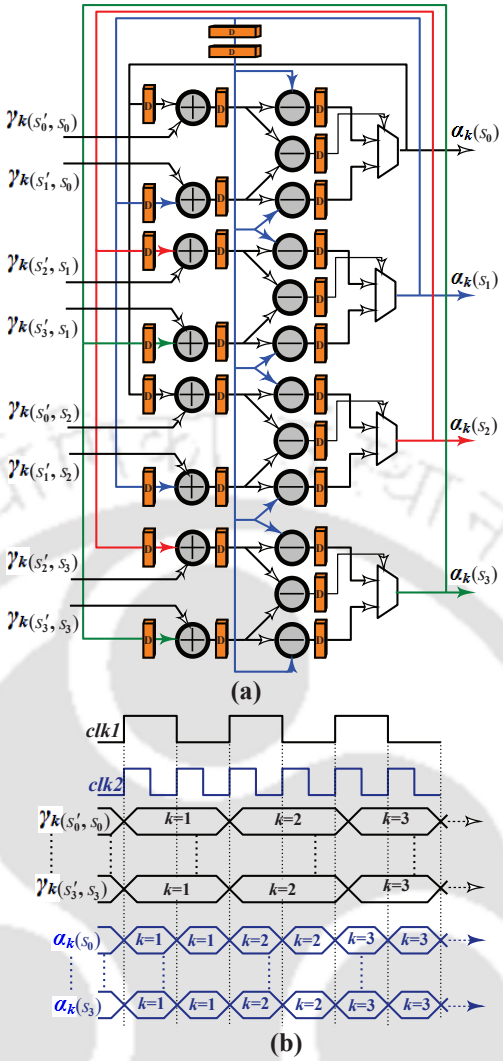


FIGURE 4.8: (a) Data-flow-graph of retimed SMCU for computing $N_s$=4 forward state metrics. (b) Timing diagram for the operation of retimed SMCU with *clk1* and *clk2*.

1) ***High-speed MAP decoder architecture***: In this work, we have presented architecture of MAP decoder for turbo decoding, as per the specifications of 3GPP-LTE/LTE-Advanced [77]. It has been designed for eight-states convolutional encoder with a transfer function of $\{1, (1+D+D^3)/(1+D^2+D^3)\}$, basic block diagram of turbo encoder/decoder can be referred from Fig. 4.1. For $N_s$=8 trellis graph which is devised based on this transfer function, four parent branch metrics are required in each trellis stage to compute state metrics as well as a-posteriori *LLR* value. Based on (4.3), these four branch metrics are given as

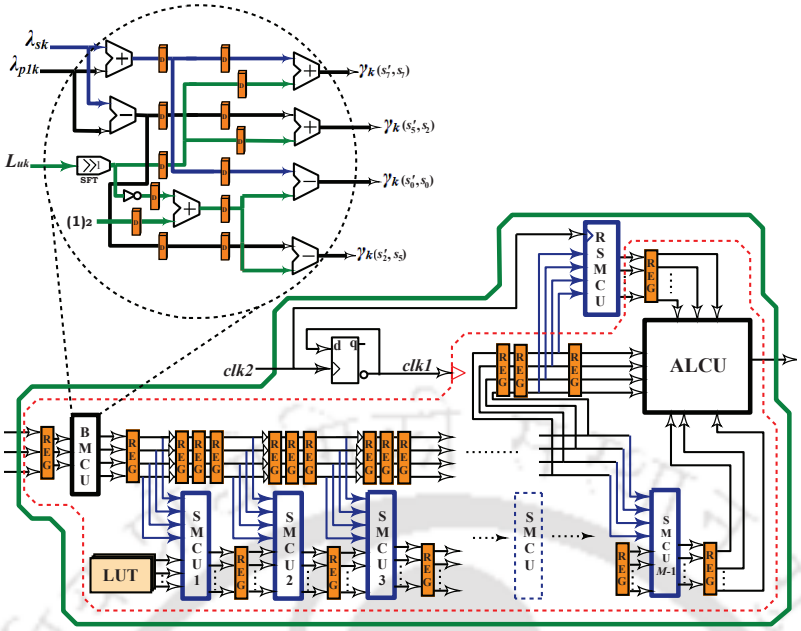


FIGURE 4.9: Deep-pipelined and retimed architecture of MAP decoder for $M$ sliding window size. Clock distribution network and pipelined BMCU are also shown.

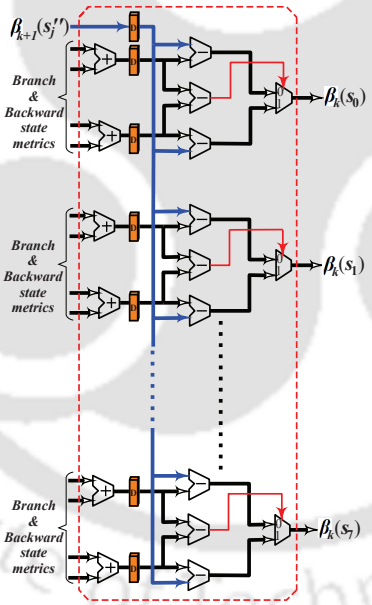


FIGURE 4.10: A feed-forward architecture of pipelined SMCU that can be used for un-grouped backward recursions in the suggest decoder architecture.

$$
\begin{aligned}
&\bullet\ \gamma_k(s'_0, s_0) = -L_{uk}/2 - (\lambda_{sk} + \lambda_{p1k}),\\
&\bullet\ \gamma_k(s'_2, s_5) = -L_{uk}/2 - (\lambda_{sk} - \lambda_{p1k}),\\
&\bullet\ \gamma_k(s'_5, s_2) = L_{uk}/2 + (\lambda_{sk} - \lambda_{p1k})\ \textit{and}\\
&\bullet\ \gamma_k(s'_7, s_7) = L_{uk}/2 + (\lambda_{sk} + \lambda_{p1k})
\end{aligned}
\tag{4.12}
$$

where the channel reliability measure has a value of $Lc$=2 in (4.3). The BMCU architecture that computes these parent branch metrics is shown in Fig. 4.9. One-bit shifter realizes the divided value by two and an inverted value is added with a decimal equivalent of one $(1)_2$ to produce a two's complement value of a fixed-point number. Additionally, this architecture is pipelined with two stages of register delays along its forward paths. Collectively, eight ACSUs are stacked in the feed-forward pipelined-architecture of SMCU which can be used for un-grouped backward recursion, as shown in Fig. 4.10. It computes $\beta_k(s_0)$ to $\beta_k(s_7)$ values for $N_s$=8 trellis states and are normalized with the value of $\beta_{k+1}(s''_j)$. As we have already discussed in chapter 3, ALCU is simple feed-forward architecture of adders, subtractors and comparators. These adders are used for computing path metric values, as given in (4.5), comparators determine maximum path metric values and are subtracted to produce a-posteriori *LLRs.* Additionally, six stages of register delays are used to pipeline ALCU in this work. These individually pipelined units are included in the MAP decoder design to make it a deep-pipelined architecture, as shown in Fig. 4.9. A retimed architecture of SMCU based on the data-flow-graph of Fig. 4.8 has been used as a RSMCU (retimed state metric computation unit) for determining the values of $N_s$ forward state metrics for the successive trellis stages. Incorporating all the pipelined feed-forward units in the MAP decoder of Fig. 4.9, both SMCUs and ALCU has a subtractor and a multiplexer in their critical paths, where as BMCU has a subtractor along this path. Thereby, the critical path delay among all these units is sum of subtractor and multiplexer delays, $\urcorner_{clk1} = \tau_{sub} + \tau_{mux}$ which decides the data-processing clock frequency of $f_{clk1}$ and it is also proportional to the decoder throughput. On the other hand, a subtractor delay $\tau_{sub}$ fixes the retimed clock frequency $f_{clk2}$ for RSMCU. Fig. 4.9 shows the clock distribution of MAP decoder in which *clk2* signal for RSMCU is frequency divided, using a flip-flop, to generate *clk1* signal which is then fed to feed-forward units. Since each of the feed-forward SMCUs are single-stage pipelined with register delays, one additional stage of register bank is required to buffer branch metrics for each SMCU, as shown in Fig. 4.9. Thereby, the decoding delay of this MAP decoder is given as

$$\partial_{dec} = (\eta_{smcu} + 1) \times (2 \times M) + (\eta_{bmcu} + 1) + (\eta_{alcu} + 1)\ clock\ cycles \tag{4.13}$$

where $\eta_{smcu}$, $\eta_{bmcu}$ and $\eta_{alcu}$ are the number of pipelined stages in SMCU, BMCU and ALCU respectively. Subsequently, respective clock cycle delays imposed by these units are ($\eta_{smcu}$+1), ($\eta_{bmcu}$+1) and ($\eta_{alcu}$+1) in the above expression.

2) ***Multi-clock domain design***: In this multi-clock decoder architecture, it is essential to synchronize the signals crossing between clock domains. Fig. 4.11(a) shows two clock domains of high-speed MAP decoder architecture: DPU (deep pipelined unit) and RSMCU. The DPU

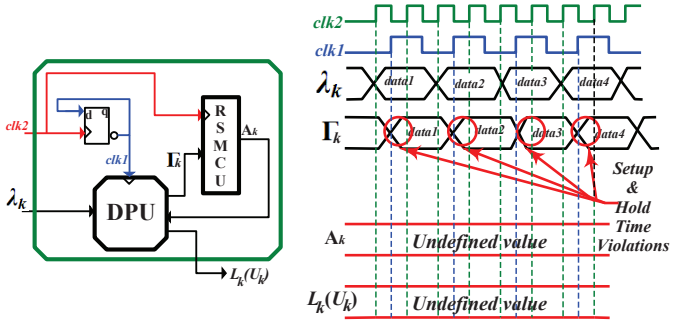


FIGURE 4.11: Architectural representation and timing diagram of dual-clock design of high-speed MAP decoder.

includes all the feed-forward units and is operated with clock *clk1*, and RSMCU runs with clock *clk2* which is at twice the clock frequency of *clk1*. In this design, set of branch metrics ($\Gamma_k$) and set of forward state metrics ($\mathbf{A}_k$) are the signals crossing from lower-to-higher and higher-to-lower clock-frequency domains respectively. Timing diagram illustrated in Fig. 4.11 shows that the input a-priori *LLRs* {denoted by $\lambda_k$=($\lambda_{sk}$, $\lambda_{p1k}$, $\lambda_{p2k}$)} are fed to DPU, synchronously at half the clock frequency of *clk2* signal. Since *clk1* is a *generated-clock-signal* from *clk2*, it is initiated after some delay with respect to *clk2*. Thereby, $\Gamma_k$ signals crossing from *clk1* to *clk2* domain violates setup and hold time criteria of *clk2* signal, as indicated in the timing diagram of Fig. 4.11. Thereby, RSMCU and DPU generate undefined-values of $\mathbf{A}_k$ and a-posteriori *LLRs* respectively.

A promising solution to mitigate this problem is to include two-stage synchronizers along the signal-paths crossing between clock domains [86]. Two-stage-synchronizer is basically two flip-flops connected in series and it samples an asynchronous signal to generate a version of the signal that posses transitions, synchronized to the local clock. We have included such synchronizers along the paths of $\Gamma_k$ with *clk2* signal and $\mathbf{A}_k$ with *clk1* signal to generate synchronous signals $\Gamma_k^s$ and $\mathbf{A}_k^s$, respectively, as shown in Fig. 4.12. Timing diagram shows that the first data

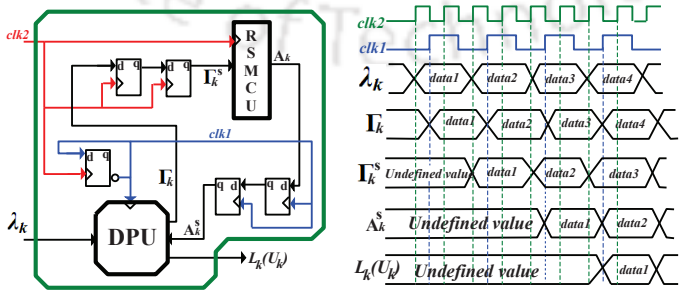


FIGURE 4.12: Dual-clock high-speed MAP decoder with two-stage-synchronizers along clock-domain-crossing paths and its timing diagram.

(*data1*) of $\Gamma_k$ is sampled by second positive edge of *clk2* signal and the synchronizer generates

TABLE 4.2: Comparison of different MAP decoders for area-consumption and processing-speed

| **Parameters** | This work† | [89]† | [90]✠ | [84]✠ |
|---|---|---|---|---|
| **Technology (nm)** | 130 | 130 | 130 | 180 |
| **Supply voltage (V)** | 1.2 | 1.2 | 1.2 | 1.8 |
| **Design area ($mm^2$)** | 2.12 | 1.28 | 1.96 | 8.7(4.54♮) |
| **Clock frequency (MHz)** | 526 | 125 | 238 | 285 |
| **Sliding window size** | 32 | 24 | 20 | 16◇ |
| **Number of trellis states** | 8 | 8 | 8 | 8 |

†: Post-layout simulation; ✠: On-chip measured; ◇: Warm-up length.
♮: Normalization area factor = $(130\text{ nm}/180\text{ nm})^2$=0.52.

$\Gamma_k^s$ signal in the next positive edge which satisfies timing requirements of *clk2* signal. Output signal $\mathbf{A}_k$ from RSMCU at higher clock frequency are synchronized to lower frequency using a similar synchronizer which operates with *clk1* signal, as shown in Fig. 4.12; thus, a-posteriori *LLRs* are synchronously generated with *clk1* signal.

3) ***Implementation trade-offs***: Deep-pipelined high-level architecture of MAP decoder based on LBCJR algorithm using the proposed techniques has achieved lower critical path delay and is suitable for high-speed applications. However, as "*there is no such thing as a free lunch*", the affected design-metric is its large silicon-area. This decoder needs $M$-1 SMCUs for un-grouped backward recursions; whereas, conventional MAP decoders require two backward recursion SMCUs for computing dummy and effective backward state metrics [28]. Basically, value of $M$ must be five to seven times the constraint length of convolutional encoder to achieve near-optimal error-rate performance [60]. Since the convolutional encoder has a value of $K_r$=4 in this work, we have considered $M$=32 for our design. On the other side, memories required by conventional decoder [28] to store branch and forward-state metrics are excluded in this work. Thereby, it is important to find out which is more expensive in terms of hardware efficiency: $M$-1 SMCUs for un-grouped backward recursions, or two SMCUs for backward recursion plus memories for branch and state metrics**?** For the sake of fair comparison among the suggested and traditional decoder architectures, we have implemented this design in 130 nm CMOS process with the supply of 1.2 V and the key characteristics are presented in Table 4.2. An architecture of MAP decoder presented in [90] is based on retimed radix-4×4 two-dimensional ACSU. By relocating adders and retiming the architecture of parallel radix-2 ACSUs, for concurrent operation, the critical path of this architecture includes two adders and a multiplexer. Thereby, the suggested MAP decoder operates at a higher clock frequency by 54.75% but with an area overhead of 7.55%, in comparison with the reported work of [90]. Scalable radix-4 MAP decoder architecture has been proposed in [89]. It has conventional ACSU with radix-4 architecture which includes two adders

and two multiplexers along its critical path. Comparatively, the MAP decoder presented in this paper operates with 76.23% better clock frequency than the reported work of [89] and has an area overhead of 39.62%, as shown in Table 4.2. Another MAP decoder based on block-interleaved pipelining technique is presented in [84]. It has radix-2 architecture for ACSU which is pipelined to achieve a critical path delay that is equal to the sum of two adders and multiplexer delays. Thereby, the suggested decoder-architecture has shorter critical path delay as compared to the work of [84]. Irrespective of different CMOS technology nodes, the normalized design-area of the suggested decoder is approximately 2× lesser than the reported work of [84].

### 4.4.3 Parallel Turbo Decoder Architecture

With an objective of designing a high-throughput parallel turbo decoder that meets the benchmark data-rate of 3GPP specification [77], we have used a stack of MAP decoders with multiple memories and ICNWs (inter connecting networks). Parallel turbo decoding is a promising solution for achieving higher decoder-throughput as it simultaneously processes $N/P$ input a-priori $LLRs$ at each time instant that reduces decoding time of every half-iteration [48]. For 188 different block lengths of 3GPP-LTE/LTE-Advanced, any one of the $P \in \{1, 2, 4, 8, 32, 64\}$ can be used for the parallel configuration of turbo decoder [77]. In this work, a parallel configuration of $P$=8 has been used for a code-rate of 1/3, as shown in Fig. 4.13. It can be seen that input a-priori $LLRs$ $\lambda_{sk}$, $\lambda_{p1k}$ and $\lambda_{p2k}$ are serial-to-parallel channeled into banks of memories. Single bank comprises of eight memories (MEM1 to MEM8) where each stores $N/P$ a-priori $LLRs$. For seven-bit quantized values of a-priori $LLRs$ and a maximum value of $N$=6144, these banks store 126 kB of data. These stored a-priori $LLR$ values are fetched in each half-iteration and are fed to the stack of 8 × MAP decoders. As shown in Fig. 4.13, memory-bank for $\lambda_{sk}$ is connected with 8 × MAP decoders via ICNW. Multiplexed $LLR$ values from memory-banks of $\lambda_{p1k}$ and $\lambda_{p2k}$ are also fed to these MAP decoders. It is to be noted that the ICNW is used for an interleaving phase of turbo decoding. It processes contention free addresses generated by dedicated AGUs and then route these data-outputs from memories to correct MAP decoders, and avoids the risk of memory-collision [31]. In this work, we have used an area-efficient ICNW which is based on the master-slave Batcher network [29]. In addition, this ICNW has been pipelined to maintain the optimized critical path delay of MAP decoder. Fig. 4.13 shows the ICNW used in this work with nine pipelined stages. The AGUs in ICNW generates the contention free pseudo-random addresses of QPP interleaver based on the equation which is given as [52]

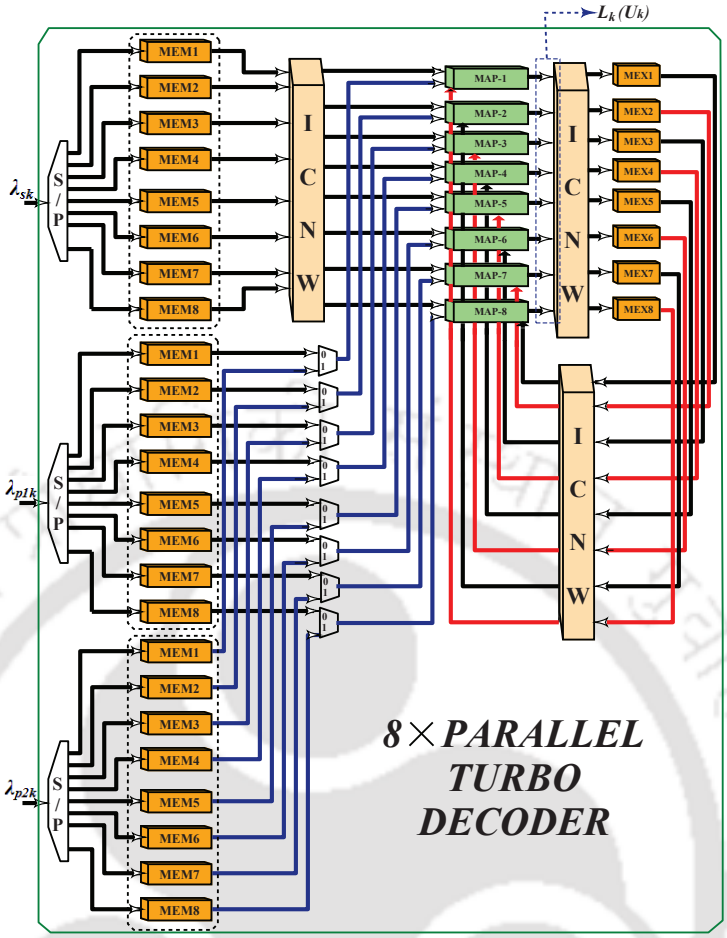


FIGURE 4.13: Parallel turbo decoder architecture with 8 × MAP decoders.

$$\Pi(i) = \{(f_1 \times s \times K) + (f_2 \times s^2 \times K^2) + (2 \times f_2 \times s \times K \times i) + (f_1 \times i) + (f_2 \times i^2)\} \ mod \ N \tag{4.14}$$

where $i$={1, 2, 3 ..... $K$}, $K$=$\lceil N/P \rceil$, and $s$={0, 1, 2, 3, 4, 5, 6, 7} for AGU0 to AGU7 respectively. Similarly, $f_1$ and $f_2$ are the interleaving factors and these values are determined by the turbo block length of 3GPP standards [77]. Addresses generated by AGUs are fed to the network of master-circuits (denoted by 'M') that generates select signals for the network of slave-circuits (denoted by 'S'), as shown in Fig. 4.14. Data-outputs from the memory-bank are fed to slave network and are routed to 8 × MAP decoders. Stack of MAP decoders and the memories (MEX1 to MEX8) for storing the extrinsic information are linked by ICNW. For the eight-bits quantized extrinsic information, 48 kB of memory is used in the decoder architecture. During the first half-iteration, the input a-priori *LLR* values $\lambda_{sk}$ and $\lambda_{p1k}$ are sequentially fetched from memory-banks and are fed to 8 × MAP decoders. Then, the extrinsic information produced

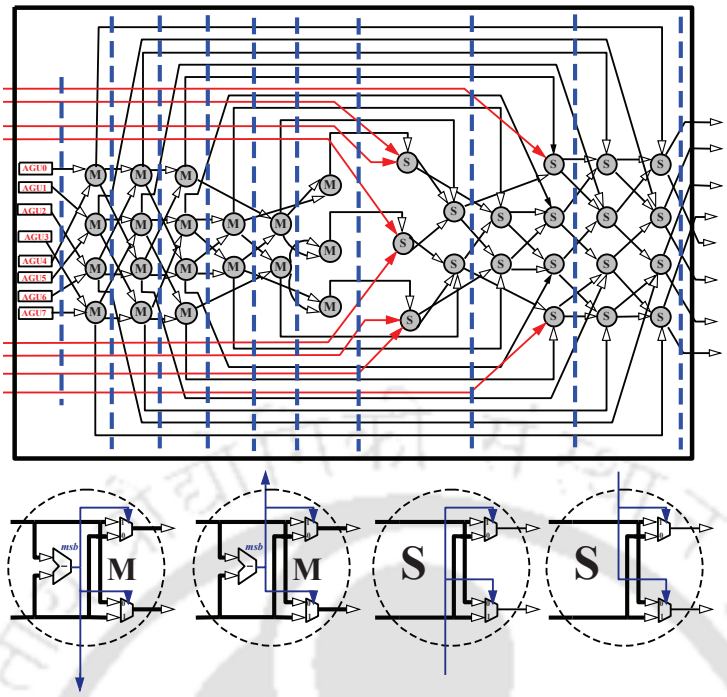


FIGURE 4.14: Pipelined ICNW (inter-connecting-network) based on Batcher network (vertical dashed lines indicate the orientation of register delays for pipelining).

by these MAP decoders is stored sequentially. Thereafter, these values are fetched and pseudo-randomly routed to MAP decoders using ICNW and are used as a-priori-probability values for the second half-iteration. Simultaneously, $\lambda_{sk}$ soft values are fed pseudo-randomly via ICNW and the multiplexed $\lambda_{p2k}$ values are fed to the MAP decoders to generate a-posteriori *LLRs* $L_k(U_k)$ and this completes a full-iteration of the parallel turbo decoding. Similarly, further iterations can be carried out by generating the extrinsic information and repeating the above procedure.

## 4.5 Performance Analysis, VLSI Implementation and Comparison of Parallel Turbo Decoder

To achieve near-optimal error-rate performance, a-priori *LLR* values, state and branch metrics are quantized for the simulation that evaluates BER performances delivered by fixed-point models of parallel turbo decoders. Fig. 4.15 shows the error-rate performances of parallel turbo decoders with $P$=8 for low effective code-rate of 1/3 at 5.5 and 8 full-iterations. For these magnitudes of design metrics, value of $M$=32 is required to deliver an optimum BER performance. It can be seen that the turbo decoder with quantized values of 7, 9 and 8 bits for input a-priori *LLRs*, state and branch metrics ($n_{bi}$, $n_{bs}$, $n_{br}$), respectively, can achieve a low BER of $10^{-6}$ at 0.6 dB, while decoding for 8 full-iterations. Turbo decoder with such quantization can perform 0.5 dB better

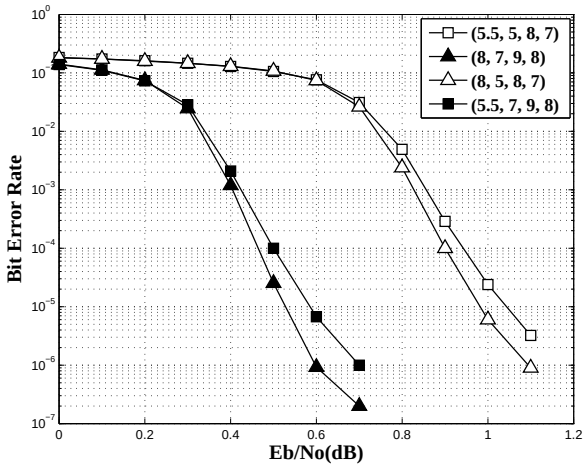


FIGURE 4.15: BER performance in AWGN channel using BPSK modulation for a low effective code-rate of 1/3, $N$=6144 ($f_1$=263, $f_2$=480), $M$=32, $P$=8 and $\omega$=1. The legend format is (Iterations, No. of bits for input a-priori LLR values, No. of bits for state metrics, No. of bits for branch metrics).

than the decoder with $(n_{bi}, n_{bs}, n_{br}) = (5, 8, 7)$ bits of quantized values for 8 full-iterations, as shown in Fig. 4.15. Similarly, BER simulations of turbo decoders with quantization (7, 9, 8) bits are performed at a high effective code-rate of 0.95 for different iterations, as shown in Fig. 4.16. It shows that an iterative decoding of parallel turbo decoder with 12 full-iterations can

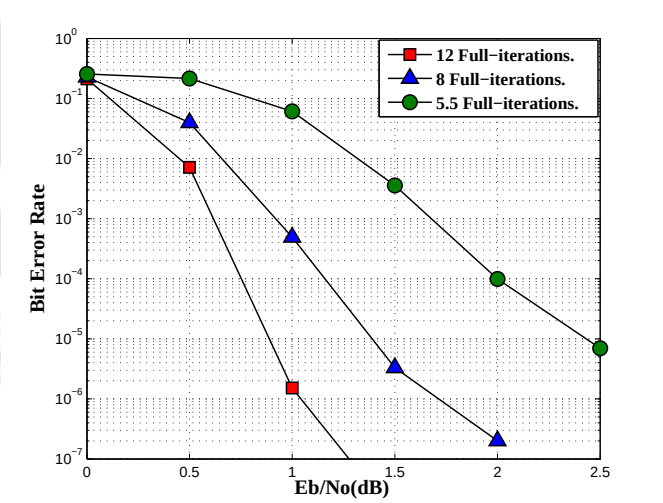


FIGURE 4.16: BER performance in AWGN channel using BPSK modulation for a high effective code-rate of 0.95, $N$=6144 ($f_1$=263, $f_2$=480), $M$=32, $P$=8 and quantization of (7, 9, 8).

perform 0.6 dB better than the decoder with 8 full-iterations at a BER of $10^{-6}$. Similarly with 5.5 full-iterations, this parallel turbo decoder has BER of $10^{-5}$ at an $E_b/N_0$ value of 2.5 dB. In this work, we have confined our simulations within two extreme corners of the code-rates: low effective code rate of 1/3 and high effective code rate of 0.95. It is to be noted that for modern

system, the full range of code-rates between these corners must be supported [74]. On the other hand, BER performance of turbo decoder degrades as parallelism further increases because the sub block length ($N/P$) becomes shorter. Based on the simulation carried out for fixed-point model of turbo decoder, the value of $M$ must be approximately $N/P$ for such highly parallel decoder-design to achieve near-optimal BER performance, while decoding for 8 full-iterations. Thereby, we have chosen the values of $M$=96 for our parallel turbo decoder model, with the configuration $P$=64, for near-optimal BER performance.

In this work, comprehensive study on VLSI implementations in 90 nm CMOS process of parallel turbo decoders with configurations $P$=8 and $P$=64 are carried out. Parallel turbo decoder architecture, with $P$=8, that uses the suggested MAP decoder design has been implemented in 90 nm CMOS process. Based on the simulations for BER performances of turbo decoders, quantized values are decided and a sliding window size of $M$=32 has been considered. It can process 188 different block lengths, as per the specifications of 3GPP-LTE/LTE-Advanced, ranging from 40 to 6144 which decide the magnitudes of interleaving factors $f_1$ and $f_2$ for the AGUs of ICNW [77]. Additionally, it has a provision of decoding at 5.5 as well as 8 full-iterations. For this design, functional simulations, timing analysis and synthesis have been carried on with Verilog-Compiler-Simulator, Prime-Time and Design-Compiler tools, respectively, from Synopsys [2]. Subsequently, place-&-route and layout verifications are accomplished with CADence-SOC-Encounter and CADence-Virtuoso tools [2] respectively [91]. Presence of high-speed MAP decoders and pipelined ICNWs in the parallel turbo decoder has made it possible to achieve timing closure at a clock frequency of 625 MHz. In these dual-clock domain MAP decoders, timing closures at 625 MHz and 1250 MHz have been achieved by deep-pipelined feed-forward units and a RSMCU respectively. With the value of $M$=32 and pipelined-stages of ($\eta_{smcu}$, $\eta_{bmcu}$, $\eta_{aplcu}$)=(1, 2, 6), decoding delay of $\partial_{dec}$ = 138 clock cycles from (4.13) and pipeline delay of $\partial_{map} = \partial_{ext}$ = 9 clock cycles are imposed by MAP decoders and ICNW respectively. Thereby, throughputs achieved by an implemented parallel turbo decoder with $P$=8 are 301.69 Mbps and 438.83 Mbps for 8 and 5.5 full-iterations, respectively from (4.1), for a low effective code-rate of 1/3. However, an achievable throughput is 201.13 Mbps for a high effective code-rate of 0.95, while decoding for 12 full-iterations to achieve near-optimal BER performance. In the suggested MAP decoder architecture, data is directly extracted between the registers and SMCUs rather being fetched from the memories, as it is performed in the conventional sliding window technique for LBCJR algorithm [60], and this may increase the power consumption. To reduce such dynamic power dissipation of our design, fine grain clock gating technique has been used in

[2]Frontend and backend design-procedures, using Synopsys and CADence EDA-tools respectively, carried out for design and implementation of the suggested decoder architecture in this work, at 90 nm CMOS technology node, have been systematically presented in Appendix A.

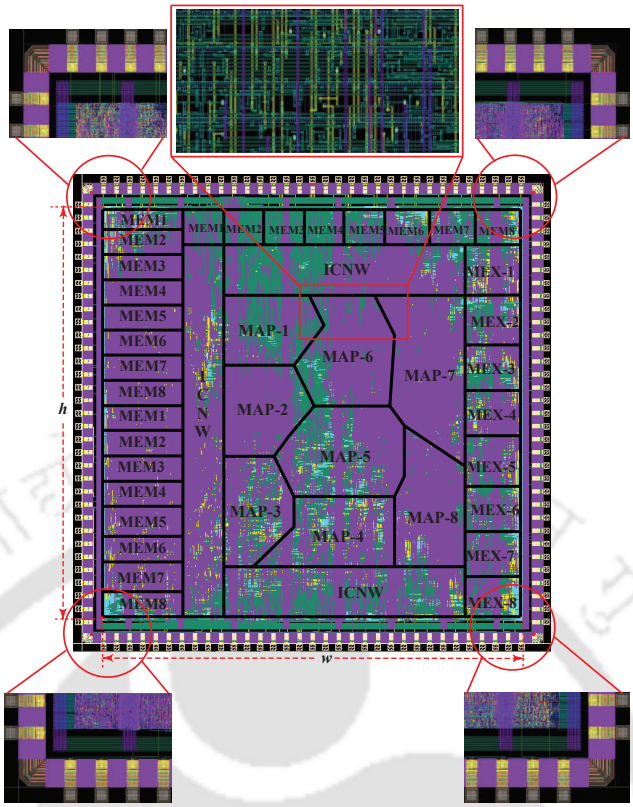


FIGURE 4.17: Metal-filled layout of the prototyping chip for 8 × parallel turbo decoder with a core dimension of $(h \times w) = (2517.2\ \mu\text{m} \times 2441.7\ \mu\text{m})$.

which enable condition is incorporated with the register-transfer-level code of this design and it is automatically translated into clock gating logic by the synthesis tool [87, 88]. The total power (dynamic plus leakage powers) consumed while decoding a block length of 6144 for 8 iterations is 272.04 mW. At the same time, this design requires extra SMCUs as well as registers and it has resulted in an area overhead which can be mitigated to some extent by scaling down the CMOS process node. Fig. 4.17 shows the chip-layout of parallel turbo decoder constructed using six metal layers and integrated with programmable digital input-output pads as well as bonded pads. It has a core area of 6.1 mm$^2$ with the utilization of 86.9% and a gate count of 694 k. Similarly, we have carried out the synthesis-study as well as post-layout simulation for parallel turbo decoder with $P$=64 in 90 nm CMOS process and the layout of this implemented decoder is shown in Fig. 4.18. As discussed earlier, the value of $M$=96 has been chosen for this design and it has increased achievable throughput as well as area overhead. In order to maintain the clock frequency of 625 MHz with increased parallelism, the ICNW is more complex and it imposes pipelined delay of 19 clock cycles. Similarly, deep-pipelined decoding delay ($\partial_{dec}$) has increased to 394 clock cycles using (4.13). Based on (4.1), this decoder with $P$=64 can achieve throughputs of 3.3 Gbps and 2.3 Gbps for 5.5 and 8 full-iterations respectively. However, it requires a core-area of 19.75 mm$^2$ and consumes total power of 1450.5 mW.

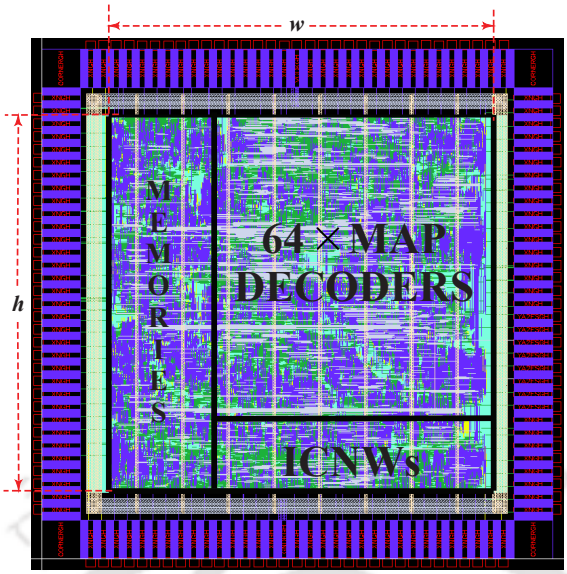


FIGURE 4.18: Chip layout of 64 × parallel turbo decoder with a core dimension of $(h \times w) = (4521.2\ \mu\text{m} \times 4370.1\ \mu\text{m})$.

Table 4.3 summarizes the key characteristics of implemented decoders presented in this work and compares them with the state of the art parallel turbo decoders of [29, 49, 52, 68, 74, 79–81]. These reported works include on-chip measured and post-layout simulated results in 65 nm, 90 nm and 130 nm CMOS processes. Normalized area occupation and energy efficiency have been included in Table 4.3 for fair comparison. Among the contributions in 65 nm CMOS process, the post-layout simulation of parallel turbo decoder with $P$=32 from [80] has shown an excellent achievable throughput. Comparatively, the suggested parallel turbo decoder design in this work with $P$=64 has 29% better throughput than the throughput reported in [80]. Parallel turbo decoder with $P$=64 in this work have normalized-area overheads of 19.4% and 25.2% compared to the works from [74] with $P$=64 and [80] with $P$=32 respectively. Similarly, the post-layout simulation of our design with $P$=8, in 90 nm CMOS process, have 57% better throughput and 65.6% area overhead in comparison with the on-chip measured results of [52]. On the other hand, the parallel turbo decoder with $P$=64 of this work has 38.4% better throughput as compared to the work [49] which is post-layout simulated in 90 nm CMOS process. In between the parallel turbo decoders with $P$=8 presented in this work and on-chip measured results of [29], we have achieved 11.2% better throughput while decoding for 5.5 full-iterations. Parallel turbo decoders implemented in this work are energy efficient, since they have achieved energy efficiencies of 0.11 nJ/bit/iterations and 0.079 nJ/bit/iterations for 8 full-iterations with the configuration $P$=8 and $P$=64 respectively.

TABLE 4.3: Key characteristics comparison of parallel turbo decoder implementations

| **Design metrics** | Proposed♣ | Proposed♣ | [74]♣ | [79]✠ | [80]♣ | [81]♣ | [52]✠ | [49]♣ | [29]✠ | [68]✠ |
|---|---|---|---|---|---|---|---|---|---|---|
| **Technology (nm)** | 90 | 90 | 65 | 65 | 65 | 90 | 90 | 90 | 130 | 130 |
| **Voltage (V)** | 1.0 | 1.0 | 0.9 | 1.2 | 1.1 | – | 1.0 | 0.9 | 1.2 | 1.2 |
| **Max. block length** | 6144¶ | 6144◇ | 6144♭ | 6144◇ | 6144◇ | 2400♯ | 6144¶ | 4096¶ | 6144¶ | 6144¥ |
| **Parallel MAP-cores** | 8 | 64 | 64 | 16 | 32 | 35 PEs | 8 | 32 | 8 | 8 |
| **MAP architecture** | radix-2 | radix-2 | radix-2 | radix-4 | radix-4 | radix-2 | radix-2† | radix-$2^4$ | radix-$2^2$ | radix-$2^2$ |
| **Sliding window size** | 32 | 96 | 64 | 14-30 | 192 | 20 | 32 | 32 | 30 | – |
| **Core area ($mm^2$)** | 6.1(3.17£) | 19.75(10.3£) | 8.3 | 2.49 | 7.7 | 4.87 | 2.1 | 9.61 | 3.57 (1.785♮) | 10.7 (5.35♮) |
| **Gate count** | 694 k | 5304 k | 5.8 M | 1574 k | – | – | 602 k | 2833 k | 553 k | 11000 k |
| **Frequency (MHz)** | 625 | 625 | 400 | 410 | 450 | 200 | 275 | 175 | 302 | 250 |
| **Throughput (Mbps)** | 301.69 (438.83§) | 2274 (3307§) | 1280 | 1013 | 2150 | 292 | 130 | 1400 | 390.6§ | 186 |
| **Max. no. of iterations** | 8 | 8 | 6 | 5.5 | 6 | 8 | 8 | 8 | 5.5 | 8 |
| **Power (mW)** | 272.04 | 1450.5 | 845 | 966 | – | 183.2 | 219 | 1356 | 788.9 | – |
| **Ener. eff. (nJ/bit/iter.)** | 0.11 | 0.079 | 0.11 | 0.17 | – | 0.078 | 0.21 | 0.12 | 0.37 (0.12‡) | 0.61 (0.20‡) |
| **($n_{bi}$, $n_{bs}$) (bit)** | (7, 9) | (7, 9) | (6 ,10) | (–, –) | (–, 11) | (6, 2) | (6, 9) | (5, 8) | (5, 10) | (–, –) |
| **($n_{br}$, $n_{lr}$) (bit)** | (8, 10) | (8, 10) | (10 ,8) | (–, –) | (9, 10) | (6, 4) | (10, 12) | (8, –) | (–, –) | (–, –) |

‡: Normalization energy factor ($NEF$) = (1.0 V/1.2 V)$^2$×(90 nm/130 nm)$^2$=0.3; ♮: Normalization area factor = (90 nm/130 nm)$^2$=0.5; £: Normalization area factor = (65 nm/90 nm)$^2$=0.52.

♣: Post-layout simulation results; ✠: On chip measured results; §: Throughput achieved at 5.5 iterations; †: Reconfigurable parallel turbo decoder architecture.

$n_{bi}$: No. of bits for input a-priori LLR values; $n_{bs}$: No. of bits for state metrics; $n_{br}$: No. of bits for branch metrics; $n_{lr}$: No. of bits for a-posteriori-logarithmic-likelihood-ratio.

¶: Supports 3GPP-LTE standard; ◇: Supports 3GPP-LTE-Advanced standard; ♭: Supports 3GPP-LTE-Advanced & WiMAX standards; ¥: Supports 3GPP-LTE & WiMAX standards; ♯: Supports WiMAX IEEE 802.16e, WiMAX IEEE 802.11n, DVB-RCS, HomePlug-AV, CMMB, DTMB & 3GPP-LTE standards.

## 4.6 Summary

Higher data rates requirement of such latest communication systems has motivated our work towards the design of high-throughput parallel turbo decoders. This chapter focuses on the VLSI design aspect of high-speed MAP decoders which are the intrinsic building-blocks of parallel turbo decoders. For the LBCJR algorithm used in MAP decoders, we have presented an un-grouped backward recursion technique for the computation of backward state metrics. Unlike the conventional decoder architectures, MAP decoder based on this technique was extensively pipelined and retimed to achieve higher clock frequency. Additionally, the state metric normalization technique employed in the suggested design of ACSU has achieved a reduced critical path delay. We have designed and implemented turbo decoders, operating with 8 and 64 parallel MAP decoders, in 90 nm CMOS process. VLSI Implementation of 8 $\times$ parallel turbo-decoder has achieved a maximum throughput 439 Mbps with 0.11 nJ/bit/iteration energy-efficiency. Similarly, 64 $\times$ parallel turbo-decoder has achieved a maximum throughput of 3.3 Gbps with an energy-efficiency of 0.079 nJ/bit/iteration. These high-throughput decoders meet peak data-rates of 3GPP-LTE and LTE-Advanced standards.

# Chapter 5

# Hardware Testing of MAP and Turbo Decoders

## 5.1 Introduction

PROTOTYPING and hardware testing of high-density complex-digital designs on FPGAs prior to fabrication have reduced the risk of chip failure. Flexibility in FPGA design allows setting the values of various design metrics and implement digital-architectures numerous times, until the desired result is obtained [92]. For the proof of concept on real hardware, we have used such FPGAs for testing the proposed MAP and turbo decoders. On the other hand, systematic procedure of building wireless-communication test-environment is an essential step for the verification of such hardware prototypes. However, hardware implementation of entire communication system consumes huge amount of time and is expensive procedure. Nevertheless, significant blocks of such communication system can be implemented on real-hardware (FPGAs/ASIC) and rest can be designed on software platform. Thereby, integrating such software test-environment of the communication system with decoder hardware-prototype can verify its functionality. It is essential to compare the decoder BER-performance that is obtained from simulation in software platform with the performance of hardware-implemented decoder. An overview of testing

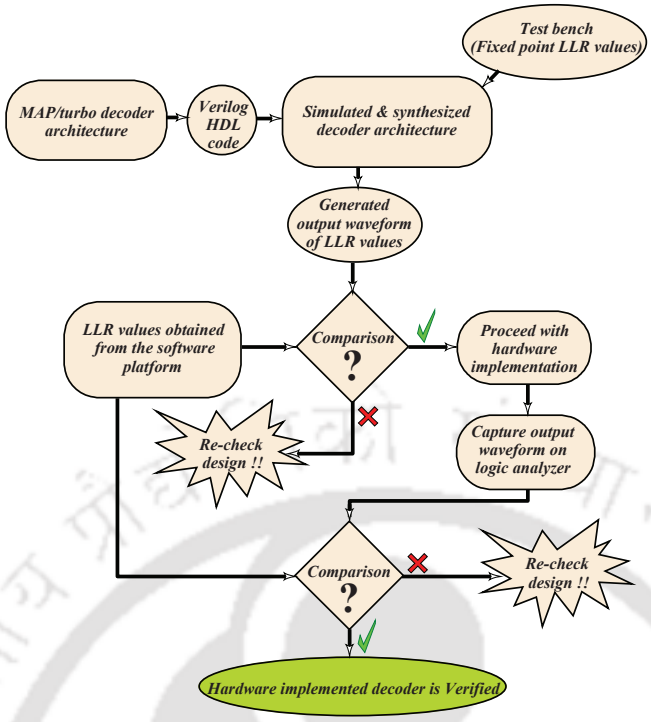


FIGURE 5.1: Schematic-overview of basic procedure for testing the hardware prototype of the proposed decoder.

procedure followed in this work has been illustrated in Fig. 5.1. It shows that the fixed-point decoder architecture that is coded with verilog HDL [93, 94] is simulated and synthesized after setting the magnitudes of various design metrics [95]. Quantized fixed-point a-priori $LLR$ values are fed to this decoder architecture via test-bench and the decoded a-posteriori $LLR$ values are obtained as waveform. Similar a-posteriori $LLR$ values obtained from the software model of communication system are compared with the displayed a-posteriori $LLR$ values. If these values match then we proceed with the hardware implementation of the decoder architecture on FPGA; otherwise, debug the verilog HDL code or redesign the decoder architecture. The test vectors of a-priori $LLR$ values are stored using on-board memories and are fed to the hardware implemented decoder. Decoded a-posteriori $LLR$ values are then captured using logic analyzer and are compared with the $LLR$ values of software model, as shown in Fig. 5.1. If there is mismatch then the design must be rechecked at every stage for debugging. Contributions of this chapter are listed as below.

- We have designed software model of a communication system which serves as test-environment for MAP and turbo decoders. Such model has been designed using MATLAB tool where the input test-vectors of a-priori $LLR$ values and output a-posteriori $LLR$ values are saved for the verification.

- The proposed MAP decoder architecture is simulated, synthesized using Xilinx ISE design suite 10.1 and implemented on Xilinx Virtex-II pro board. Output a-posteriori *LLR* values are captured on a virtual logic analyzer using Xilinx Chipscope pro analyzer [96, 97].
- Finally, the parallel turbo decoder architecture is implemented on ALTERA Cyclone-V SoC hardware board and the outputs are displayed on a logic analyzer (Hewlett Packard: model no. 54620A).

The remainder of this chapter has been organized as follows. Section-5.2 presents a software model of communication system that is used for testing MAP and turbo decoders, additionally, their BER performances are evaluated. Hardware implementation, testing and performance analysis of MAP and turbo decoders are included in section-5.3 and section-5.4 respectively. Eventually, section-5.5 summarizes this chapter.

## 5.2 Software Model

In this section, software model of communication system for testing MAP as well as turbo decoder is presented and it also includes BER performances analysis of these decoders.

### 5.2.1 Communication System

Suggested decoder architectures are tested in communication-system model that includes AWGN-channel environment and BPSK modulation scheme. Fig. 5.2 shows transmitter and receiver blocks of such model for verifying functionality and BER performance of the hardware-implemented decoders. At the transmitter side, randomly generated sequence of bits ($U_k$) are encoded using the convolutional encoder with a transfer function of $\{1, (1+D+D^3)/(1+D^2+D^3)\}$. It has constraint length of four and eight trellis states for each trellis stage. Sequence of encoded bits ($U_{con}$) is punctured to achieve a code-rates of 1/2 and 1/3 for MAP and turbo decoders respectively. Puncturer can produce a sequence of bits ($U_{pun}$) for any code-rates depending on the puncturing pattern employed [98, 99]. Sequence $U_{pun}$ is bit-interleaved using bit-wise interleaving unit to reduce the effect of noisy channel and the generated interleaved sequence is $U_{bi}$. BPSK modulation has been carried out for modulating the sequence $U_{bi}$ to produce sequence of modulated signals $S_{bpsk}$. It is then subjected to AWGN channel environment, where the white Gaussian noise $S_{noise}$ is added with the modulated signal. The received noisy sequence of $r$= ($S_{bpsk}$ + $S_{noise}$) is the output of AWGN channel at receiver side. Soft-demodulator is fed with this noisy sequence $r$ and it produces the soft values of a-priori-probability $V_{dem}$. Soft bit-wise

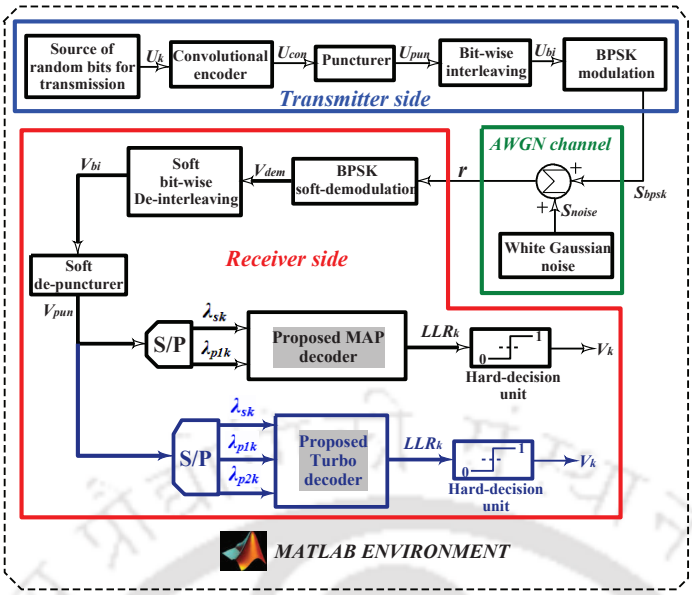


FIGURE 5.2: Software model of communication system for testing the MAP/turbo decoder in MATLAB environment.

de-interleaving and de-puncturing are carried out to generate the sequence of soft-values $V_{bi}$ and $V_{pun}$ respectively. The sequence of soft values $V_{pun}$ is S/P (serial-to-parallel) converted to $\lambda_{sk}$ and $\lambda_{p1k}$ soft values corresponding to systematic and parity bits, respectively, for MAP decoder. On the other hand, for the code-rate 1/3, $V_{pun}$ is S/P converted into $\lambda_{sk}$, $\lambda_{p1k}$ and $\lambda_{p2k}$ soft values for turbo decoder. These soft-values are fed to the MAP/turbo decoder which processes them to compute $LLR_k$ values, as shown in Fig. 5.2. Finally, the $LLR_k$ values are made to pass through hard-decision unit to generate the sequence of decoded bits $V_k \; \forall \; k$={1, 2, 3 ..... $N$}.

In order to extract the fixed point test-vectors of a-priori $LLR$ values for the hardware verification of decoders, these real values of $\lambda_{sk}$, $\lambda_{p1k}$ and $\lambda_{p2k}$ must be quantized and saturated consecutively. We assume that each of the real valued a-priori $LLRs$ is represented by an integer $Z_k$ which needs the total number of $n_B$ bits. Thereby, the fixed point representation of real valued $\lambda_{sk}$ is denoted as $Z_k = F(\lambda_{sk}) = (n_B, \; n_P)$ where $n_P$ is the fractional precision of $\lambda_{sk}$. Quantization process fixes the number of bits required for fractional precision based on the magnitude of real valued a-priori $LLRs$. The operation performed during this quantization process is $Y_k = \lfloor 2^{n_P} \times \lambda_{sk} + 0.5 \rfloor$. For example, if the real valued $\lambda_{sk}$ is 4.53212 then for two different precision $n_P = 2$ and 3, the integer outputs of quantization process are $Y_k = 18$ and 41 respectively. The final quantized value of $\lambda_{sk}$ is obtained by saturation process. For the saturation process, if the input $Y_k$ is positive then final quantized output $Z_k = min(Y_k, \; 2^{n_B-1} - 1)$ else if the value of $Y_k$ is negative then $Z_k = max(Y_k, \; -2^{n_B-1})$. Assuming the total number of bits required is $n_B = 6$, for two values of $Y_k$ obtained in previous example, the quantized value are $Z_k = min(18, \; 2^5 - 1) = 18$ and $Z_k = min(41, \; 2^5 - 1) = 31$. Table 5.1 shows the fixed point

TABLE 5.1: Fixed point representation of real value using quantization and saturation processes

| $\lambda_{sk}$ | $(n_B, n_P)$ | $Y_k$ | $Z_k$ | Binary | Fixed point representation |
|---|---|---|---|---|---|
| 4.53212 | (6, 3) | 41 | 31 | 011.111 | 3.875 |
| 4.53212 | (6, 2) | 18 | 18 | 0100.10 | 4.5 |

representation of real number with same number of total bits but with different precision. Thus, quantization and saturation processes are required for fixed point representation of real valued a-priori *LLRs* ($\lambda_{sk}$, $\lambda_{p1k}$ and $\lambda_{p2k}$). In this work, we have selected the values of ($n_B$, $n_P$) as (5, 2) bits and (7, 3) bits to represent the fixed-point test vectors of input a-priori *LLR* values for MAP and turbo decoders respectively.

### 5.2.2 BER Performance Evaluation

Software model of communication system is simulated with MAP and turbo decoders for BER performance evaluation in MATLAB environment. These simulations are carried out with real-valued input soft values of a-priori *LLRs*. Approximately $10^7$ bits are pseudo-randomly generated, transmitted and received; after the decoding process, the decoded bits $V_k$ are compared with transmitted bits $U_k$ to compute the BERs for various $E_b/N_0$ values, as shown in Fig 5.3. It indicates that the coded communication system with MAP and turbo decoder can attain a BER of $10^{-5}$ at the $E_b/N_0$ values of 5.5 dB and 0.8 dB respectively. Such plots of BER performances serve as benchmark curves which are used for verifying the BER values; those are obtained from the hardware models of decoders.

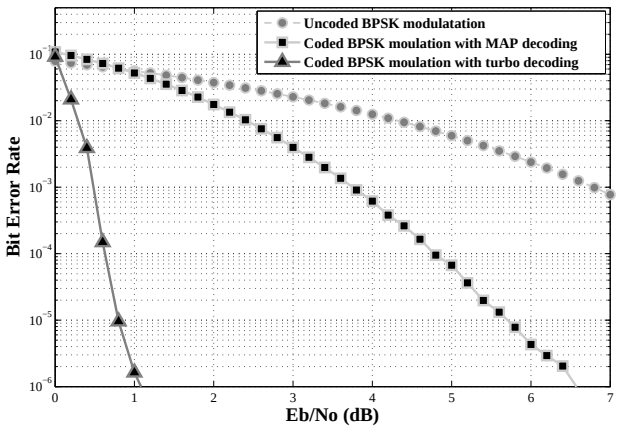


FIGURE 5.3: BER performances of MAP decoder for a code rate of 1/2 and turbo decoder for a code rate of 1/3 with 8 decoding iterations.

## 5.3 FPGA Implementation and Verification of MAP Decoder

This section presents hardware-implementation and testing procedure for the proposed MAP decoder.

### 5.3.1 Implementation

Proposed MAP decoder architecture from Chapter 4 is coded using verilog HDL for simulation and synthesis using Xilinx ISE 10.1 design suite to verify its functionality. For this purpose, quantized soft-values of a-priori *LLRs*, which are denoted by x=$\digamma(\lambda_{sk})$ and xp1=$\digamma(\lambda_{p1k})$ with ($n_B$, $n_P$)=(5, 2) bits, are incorporated as test vectors in the test-bench. Thereafter, the synthesized verilog HDL code of MAP decoder is simulated with this test-bench and the decoded a-posteriori *LLR* values are verified with the quantized a-posteriori *LLR* values obtained from the MATLAB simulation of the software communication model. Fig. 5.4 shows the GUI (graphical user interface) of inputs and simulated output of MAP decoder in Xilinx ISE 10.1 environment. A-posteriori *LLR* value (denoted by llr with 11 bits, as shown in GUI) represents the probability of transmitted bit to be '0' or '1'; for example, the first five a-posteriori *LLR* values {61, 75, 61, -93 and -41} shown in Fig. 5.4 indicate that the transmitted bits are {1, 1, 1, 0 and 0}. These values match with the simulated outputs of the software communication model and it proves correct functionality of MAP decoder. Thereby, it indicates that synthesized netlist of the design is ready for further processing. Generated design-netlist has been placed, routed and checked for the timing violations. Thereafter, the post-routed simulation of MAP decoder is carried out with same set of test-bench and the output is verified with simulated results from MATLAB. Table 5.2 summarizes timing report and hardware consumed by MAP decoder using various families, devices and packages of FPGA. Hardware consumption of this decoder-design has been accounted by number of slices and LUTs used from the available resources of board.

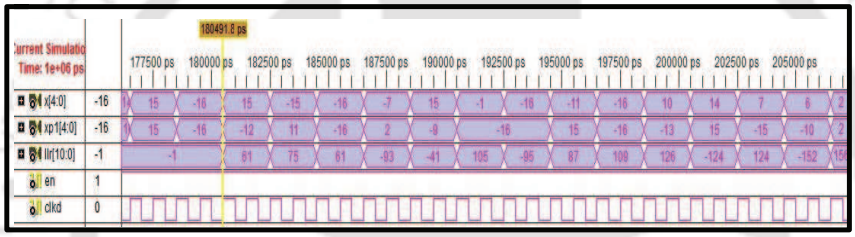


FIGURE 5.4: Snapshot of the GUI that includes inputs and simulated output of MAP decoder in Xilinx ISE 10.1 simulation environment.

TABLE 5.2: Hardware consumption and timing report of the MAP decoder

| Family | Virtex-II-pro | Virtex-IV | Virtex-V |
|---|---|---|---|
| Device | XC2VP30 | XC4VLX15 | XC5VLX30 |
| Package | FF896 | SF363 | FF324 |
| No. of slices | 5998/13696 | 5995/6144 | 9130/19200 |
| No. of slice flip-flops | 9308/27392 | 9303/12288 | 9925/19200 |
| No. of LUTs | 9880/27392 | 9880/12288 | 8491/10564 |
| Max. freq. of operation (MHz) | 288 | 314 | 411 |
| Max. input delay (nS) | 3.6 | 4.2 | 0.9 |
| Max. output delay (nS) | 3.3 | 3.8 | 2.8 |

The maximum clock frequencies, input and output delays of the implemented decoder are also listed in Table 5.2.

### 5.3.2 Testing

In order to test this hardware prototype of MAP decoder, fixed-point quantized a-priori *LLR* soft-values x and xp1 are stored using on-board RAM (random access memory). Fig. 5.5 shows MAP decoder integrated with such memories and it is referred as IMD (integrated MAP decoder) core in this chapter. These memories are denoted as RAMX and RAMXP for x and xp1 respectively. Each of these RAMs stores 12282 soft values, where each soft value is represented by 5 bits, and consumes approximately 60 kb of memory. A triggering input signal (*en*) is fed to all units and it starts the decoding process. A shifted *en_agu* signal enables the AGU which generates sequential addresses (*addr*) from 0 to 12281, and these addresses are used for fetching the soft-values from memories and are fed to MAP decoder, as shown in Fig. 5.5. Flip-flops are used for dividing the clock frequency as well as delaying the enable signal to reset AGU. Enable signals *en_map* and *en_acacs* are used for triggering MAP decoder which processes the soft-values to generate decoded a-posteriori *LLR* values. It is essential to monitor these *LLR* values processed by the MAP decoder which is implemented on FPGA board. Thereby, such values can be monitored using the multi-channeled logic analyzers. ChipScope Pro tools from Xilinx [96] has an ability to integrate the logic analyzer cores with target-design that is dumped on FPGA board and carry out the design testing. In this section, similar methodology has been adopted to verify hardware prototype of MAP decoder. We have incorporated ILA (integrated logic analyzer) and ICON (integrated controller) cores for the purpose of testing such FPGA-hardware prototype of MAP decoder [100]. Cores generated by Xilinx ChipScope Pro tool make use of JTAG (joint test action group) boundary scan port, which is mounted on Xilinx FPGA board to communicate

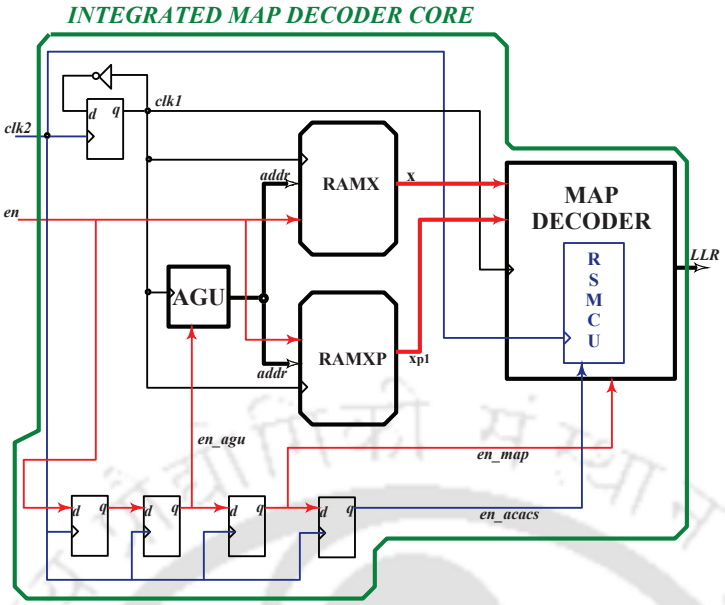


FIGURE 5.5: FPGA on-board integration of suggested MAP decoder-design with memories containing the fixed point soft values x and xp1.

with host computer using JTAG parallel or USB (universal serial bus) downloadable cable. ICON cores are used for setting up communication paths between JTAG boundary scan port and ILA cores of FPGA board. Such ILA core is a customizable logic analyzer core that can be used to visualize input/output signals of implemented design on FPGA using the monitor of host computer. Successive procedure of integrating ILA and ICON cores with the hardware prototype of IMD core are:

***Step-1*:** The CORE generator tool from Xilinx ChipScope Pro is used for creating ILA and ICON cores for IMD core based on its number of input and output signals. Specifications like the number of triggering signals to be monitored and the magnitude of sampling depth are set in this process. Netlists of these ILA and ICON cores can be conveniently integrated with the targeted IMD core.

***Step-2*:** The CORE inserter tool from Xilinx ChipScope Pro automatically integrates these generated netlist of the ILA as well as ICON cores with the netlist of IMD core. At the same time, UCF (user constraint file) is also created for the design.

***Step-3*:** Then, the design is mapped, placed and routed along with the cores using Xilinx ISE 10.1 design suite and such consecutive processes can integrate these cores with the design netlist of IMD core. Subsequently, the configuration file (.bit format) is created for the IMD core which is integrated with ILA and ICON cores.

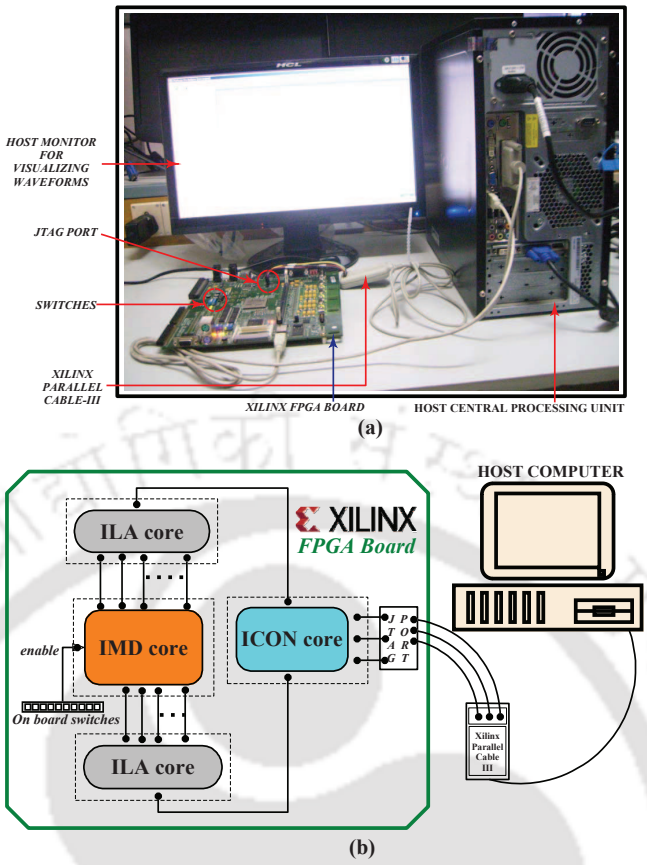


FIGURE 5.6: (a) An actual test setup for the implemented MAP decoder on FPGA board with the host computer. (b) Detail schematic showing the integration of ILA and ICON cores with the IMD core on FPGA board.

Fig. 5.6 (a) shows the setup for hardware testing of the MAP decoder using a Virtex-II-pro (XC2VP30-FF896) FPGA. The JTAG port of FPGA board is connected to CPU (central processing unit) of the host computer via Xilinx parallel cable-III connector. FPGA board is powered up and ChipScope Pro Analyzer tool enables the host computer to detect the FPGA board. Configuration file containing the integrated netlist of IMD core with ILA and ICON cores is dumped on FPGA board. Fig. 5.6 (b) schematically shows the interconnection of ILA and ICON cores with IMD core, on-board switches and JTAG port. These ICON cores transfer signals captured by ILA cores to host-computer CPU via JTAG ports using the Xilinx parallel cable-III. One of the board switches is used as an enable signal that is interfaced with IMD core via UCF file. On setting this *enable* signal high, the input a-priori *LLR* values are sequentially fetched from the memories and are fed to MAP decoder. Then, GUI of ILA core is displayed on the monitor of host computer and has trigger-setup as well as waveform options. By setting up triggering conditions, the signal waveform that shows input and output values of MAP decoding process are displayed on the host-computer monitor, as shown in Fig. 5.7. Output waveforms of a-posteriori *LLR* values are compared with the simulated output waveform from Fig. 5.4 and is found that

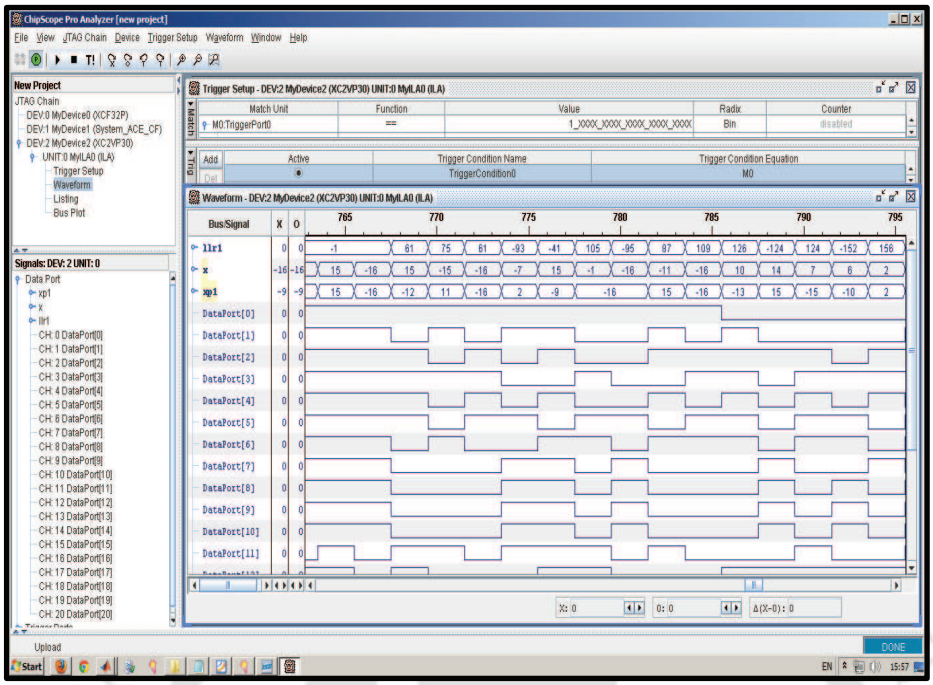


FIGURE 5.7: Output waveform of the MAP decoder implemented on the FPGA board using the integrated logic analyzer of the Xilinx ChipScope Pro Analyzer tool.

these waveforms have same values of a-posteriori *LLRs*. Thereby, the hardware prototype of MAP decoder is working as desired and thus verified.

### 5.3.3 Performance Evaluation

For a given $E_b/N_0$ value, 12282 fixed point a-priori *LLR* soft-values from MATLAB simulation environment are stored in RAMX and RAMXP, thereafter, on triggering *enable* signal, these soft-values are fetched from RAMs and are fed to MAP decoder. The decoded bits $V_k$ $\forall$ $k$={1, 2, 3 ..... 12282} are obtained by inverting the *msb* of a-posteriori *LLR* values and are stored in the built-in RAM of FPGA, in order to compare with the transmitted bits $U_k$. Subsequently, the error is computed by XOR-ing and then summing the sequences $U_k$ and $V_k$ $\forall$ $k$={1, 2, 3 ..... 12282}. This process is repeated for approximately 82 times such that the BER is computed for nearly $10^6$ bits for each $E_b/N_0$ value. The process of computing a BER value for a given $E_b/N_0$ is summarized as follows.

***Initialization***: error = 0; N = 12282; $N_T = 10^6$.

- *for* $i = 1$ to $\lceil N_T/N \rceil$
- sum = 0.
- *for* $k = 1$ to N

TABLE 5.3: BER values at different $E_b/N_0$ values for the implemented MAP decoder.

| $E_b/N_0$ (dB) | BER | $E_b/N_0$ (dB) | BER | $E_b/N_0$ (dB) | BER | $E_b/N_0$ (dB) | BER |
|---|---|---|---|---|---|---|---|
| 0 | 0.1083 | 1.8 | 0.0227 | 3.6 | 0.0014 | 5.4 | 0.0 |
| 0.2 | 0.0959 | 2.0 | 0.0175 | 3.8 | 0.0009 | 5.6 | 0.0 |
| 0.4 | 0.0837 | 2.2 | 0.0135 | 4.0 | 0.0006 | 5.8 | 0.0 |
| 0.6 | 0.0726 | 2.4 | 0.0103 | 4.2 | 0.0004 | 6.0 | 0.0 |
| 0.8 | 0.0618 | 2.6 | 0.0076 | 4.4 | 0.0003 | 6.2 | 0.0 |
| 1.0 | 0.0523 | 2.8 | 0.0056 | 4.6 | 0.0002 | 6.4 | 0.0 |
| 1.2 | 0.0434 | 3.0 | 0.0040 | 4.8 | 0.0001 | 6.6 | 0.0 |
| 1.4 | 0.0355 | 3.2 | 0.0028 | 5.0 | 0.0001 | 6.8 | 0.0 |
| 1.6 | 0.0285 | 3.4 | 0.0020 | 5.2 | 0.0000 | 7.0 | 0.0 |

- x= $U_k \oplus V_k$.

- sum = sum + x.

- *end*

- error = error + sum.

-*end*

- $BER$ = error/(N $\times$ $N_T$).

In this way, the BER values are computed for various $E_b/N_0$ values and are listed in Table 5.3. Fig. 5.8 shows the BER curves plotted using the logarithmic values of BERs in Table 5.3 with respect to $E_b/N_0$ values. In addition, BER curve of simulated MAP algorithm is shown for comparison. The MAP decoder implemented on FPGA has achieved a BER of $10^{-4}$ at an $E_b/N_0$ value of 4.75 dB. However, it has a coding loss of approximately 0.2 dB in comparison with BER performance of simulated MAP algorithm. Such degradation in its performance is due to fixed-point implementation of MAP decoder as compared to simulated values in which the precision used for representing a number is very high. BER performance of implemented MAP decoder can be improved by increasing number of bits for the fixed point representation. However, this process results in larger design area, higher power dissipation and longer critical path delay. Slight degradation in BER performance can be compromised for high speed, low power and area efficient applications from implementation perspective.

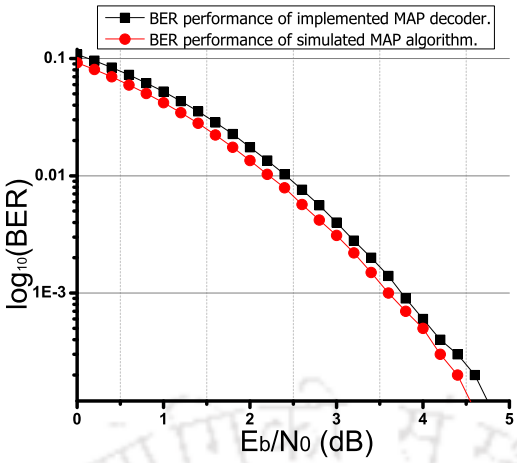


FIGURE 5.8: Comparison of the BER performances of the implemented MAP decoder on FPGA and simulated results from MATLAB environment.

## 5.4 Implementation, Testing and Performance Evaluation of Turbo Decoder

This section presents an implementation of parallel turbo decoder architecture which includes stack of suggested MAP decoders for high-speed application. On-board hardware prototype of such turbo decoder is verified and its BER performance has been evaluated in this work. We have carried out an implementation of parallel turbo decoder with 8 × MAP decoders and QPP interleavers, as presented in chapter 4. Since the turbo decoder is compliant to 3GPP-LTE and LTE-Advanced wireless communication standards, a maximum turbo block length of 6144 bits and a code-rate of 1/3 have been considered. Additionally, this decoder can be operated at 8 as well as 5.5 decoding iterations and the quantization of fixed point input a-priori *LLR* values is $(n_B, n_P)$=(7, 3) bits. The test setup of communication system that is used for testing the decoder hardware-prototype has already been illustrated in Fig. 5.2. An architecture of 8 × parallel turbo decoder is coded in verilog HDL and is analyzed as well as synthesized using ALTERA Quartus II tool [101]. Output waveform of decoded a-posteriori *LLRs* for 8 and 5.5 iterations are compared with *LLR* values obtained from MATLAB simulation of the communication system, as shown in Fig. 5.2. We proceed with the hardware prototyping of our design if these values match, else the designed is rechecked for bugs. Alike the process followed for MAP decoder prototyping, the quantized soft-values of a-priori *LLRs* $\lambda_{sk}$, $\lambda_{p1k}$ and $\lambda_{p2k}$ are stored using on-board memories. Each of these memories has to store 6144 soft-values of 7 bits each and they are fetched while turbo decoding. Detail information regarding memory segregation and their connection with 8 × MAP decoders via inter-connecting networks are comprehensively discussed in chapter 4.

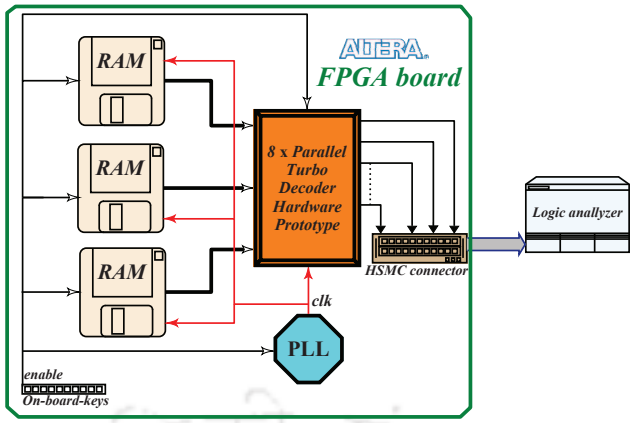


FIGURE 5.9: Schematic of test-plan for the hardware prototype of parallel turbo decoder using FPGA and logic analyzer.

The targeted ALTERA-FPGA board (Cyclone V SoC 5CSXFC6D6F31C8ES device) has been built on TSMC (Taiwan semiconductor manufacturing company) in 28 nm low-power (28L) process [102]. The input a-priori *LLRs* with (7, 3) bits quantization are stored separately using on-board RAMs, as shown in Fig. 5.9. On-board fractional PLL (phase lock loop) are used for generating the clock for RAMs and hardware prototype of parallel turbo decoder. Data-outputs from these memories are fed as inputs to the decoder prototype which processes these test vectors to generate output a-posteriori *LLR* values. These outputs from the board are interfaced with logic analyzer via 160 pins HSMC (high speed mezzanine card) which has a data transfer speed of 3.125 Gbps. Fig. 5.10 shows the practical setup for testing implemented

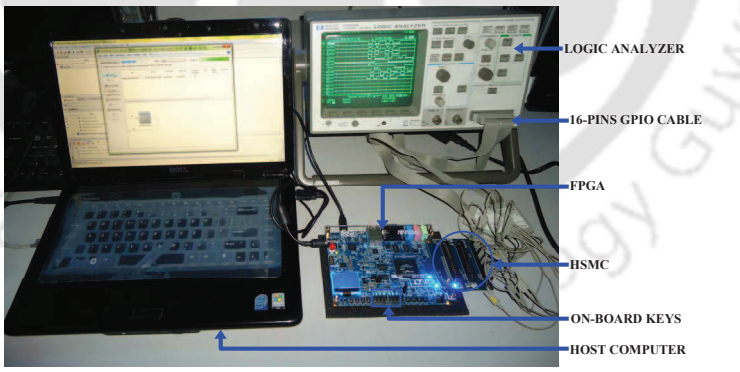


FIGURE 5.10: Actual test setup for the hardware testing of channel decoder using FPGA and logic analyzer in our lab.

hardware on FPGA board. By triggering enable signal high using the on-board keys, the test vectors are fetched from RAMs and are fed to decoder which processes them at a clock frequency of 800 MHz. The 11 bits output *LLR* soft-value of channel decoder is connected to 16-channel logic analyzer (HEWLETT PACKARD, model no. 54620A) via HSMC using GPIO (general

purpose input output) connector. Thereby, the output is displayed using 11 channels (indicated as CH00−CH10) on the logic analyzer screen, as shown in Fig. 5.11.

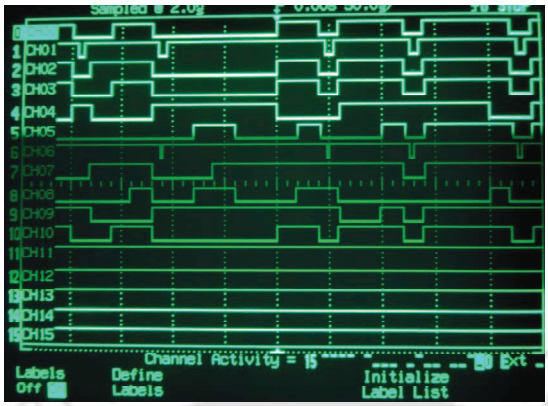


FIGURE 5.11: Output a-posteriori *LLR* soft-values from the parallel turbo decoder displayed using 11 channels (CH00-CH10) on a logic analyzer screen.

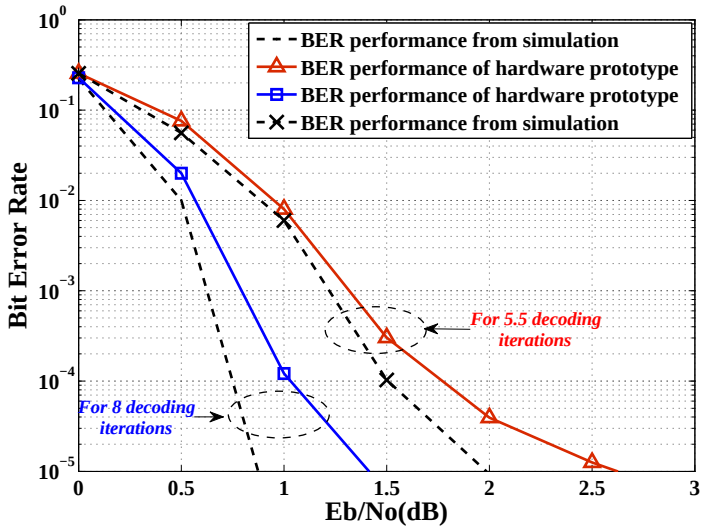


FIGURE 5.12: Comparison of BER performances delivered by hardware prototypes of turbo decoder with simulated BER performance.

Sequence of sign-bits from the output *LLR* soft-values can be considered as decoded bits $V_k$. In this work, for each $E_b/N_0$ value, $10^8$ such decoded bits from the implemented decoder are stored in on-board RAM. These stored values from FPGA are transferred to the host computer via Ethernet-port and then saved as a file (.txt file). Matrix of the transmitted information bits $U_k$ from MATLAB environment is compared with these saved decoded values from hardware to compute a BER at this particular $E_b/N_0$ value and such procedure is carried out for all the $E_b/N_0$ values, as discussed in section-5.3.3. We have computed such BERs for $E_b/N_0$ values ranging from 0 to 3 dB with an interval of 0.5 dB and have achieved reliable BERs upto $10^{-5}$, as shown

in Fig. 5.12. It shows that the hardware prototype of turbo decoder with 8 and 5.5 decoding iterations deliver BER of $10^{-5}$ at 1.4 and 2.6 dB respectively. Fig. 5.12 shows degradations of 0.52 and 0.64 dB when the hardware prototype of turbo decoder is decoding at 8 and 5.5 iterations, respectively, in comparison with the simulated BER performance of decoder. The deviation observed between simulation, which is based on very high precision number system, and the hardware prototype is mainly due to the 'fixed point' type decoder architecture.

## 5.5 Summary

In this chapter, we have presented detail illustrations on the testing of hardware prototypes which are designed for the proposed MAP and turbo decoder architectures. Test setup for communication system in MATLAB software platform was designed for testing the decoder prototypes. Subsequently, the BER performances of MAP and turbo decoders were carried out, in this MATLAB environment, with BPSK modulation and under AWGN channel condition. The MAP decoder architecture was implemented on various families of FPGA and the post place-&-route report was presented. It showed that the design implemented on Virtex-II-pro, Virtex-IV and Virtex-V FPGA boards could be operated at maximum operating frequencies of 288 MHz, 314 MHz and 411 MHz respectively. Subsequently, test vectors generated from software platform of the communication system were stored in RAM and are fed to MAP decoder design. Thereafter, the Xilinx ChipScope Pro tool was used for an integration of on-board decoder design with ILA cores, using ICON cores via Xilinx JTAG parallel cable III. Thereby, the output waveform generated by MAP decoder implemented on FPGA was compared with the simulated waveform and then design verification was accomplished. The comparative plots of BER performances showed that the hardware prototype of MAP decoder has a degradation of 0.2 dB at a BER of $10^{-4}$ in comparison with the simulated BER performance of MAP algorithm from MATLAB environment.

The suggested parallel turbo decoder with 8 $\times$ MAP decoders was simulated, synthesized and then implemented on ALTERA-FPGA board (Cyclone V SoC 5CSXFC6D6F31C8ES device). The input a-priori *LLR* soft-values were stored using on-board memories and were fed to the decoder which could operate at an operating frequency of 800 MHz. As discussed in chapter 4, the high-speed parallel turbo decoder could operate at a maximum clock frequency of 625 MHz at 90 nm CMOS technology node but the same high-speed turbo decoder can operate at a clock frequency of 800 MHz in this FPGA, since the Cyclone V SoC ALTERA FPGA board is designed with 28 nm CMOS process. In order to capture the output waveform of 11 bits a-posteriori *LLR* value, the FPGA board was interfaced with a logic analyzer

via HSMC which transfers data at a maximum rate of 3.125 Gbps. The values displayed on logic analyzer screen were verified with the simulated results from MATLAB environment. Thereafter, the BER plots of hardware prototype of parallel turbo decoder was presented and compared with the simulated BER curve of turbo decoder. It showed that the implemented turbo decoder had a degradation of 0.6 dB in comparison with the simulated BER value at $10^{-4}$ for 8 decoding iterations.

# Chapter 6

# Conclusion

## 6.1 Thesis Conclusion

High-throughput and energy-efficient design of turbo decoder is an important object of interest in the wireless industry at present. These are two serious bottlenecks of present-day turbo-decoder architectures which might be obsolete from the next generation wireless communication standards unless such issues are resolved. Thereby, this thesis has adapted progressive methodology of solving such recent challenges. In this work, we have studied the behavior of turbo code in a wireless communication environment and analyzed the performance under various conditions. A comparative study of existing turbo-decoder architectures was carried out. Finally, a high-throughput and energy-efficient parallel turbo decoder for the future wireless communication systems was conceived.

This work presented behavioral study of turbo code using the physical layer of DVB-SH standard. Software models of various communication blocks in baseband and RF-section of both transmitter and receiver sides of the DVB-SH physical layer were designed. Thereafter, simulations were carried out for BER performance analysis of turbo code in AWGN and frequency selective ITU-R fading channel environments. OFDM modulation scheme with 1K FFT was used where each sub-carrier was modulated with QPSK and 16-QAM. Similarly, BER performances of

turbo code were analyzed for different decoding iterations, sliding window sizes, MAP algorithms and code-rates. Estimation of turbo-decoder throughput for various processor speeds, decoding iterations and parallel configurations were also presented in this work.

MAP decoder is the core engine of turbo decoder and various simplified MAP algorithms have been reported for it. Thereby, we have carried out comparative study of these algorithms from BER-performance and architectural perspectives. It was observed that the PWLA based algorithm resulted in a shortest critical path delay with nominal degradation in BER performance as compared to ideal MAP algorithm. Based on this PWLA simplified MAP algorithm, we presented a design of non-parallel radix-2 turbo decoder which was then ASIC implemented in 130 nm CMOS technology node. Implementation results revealed that it could achieve a throughput of 28 Mbps with an energy-efficiency of 0.28 nJ/bit/iterations and this throughput-value was highest among the reported values of non-parallel turbo-decoder implementations. Thereafter, this work presented a memory reduced technique, which we have referred as RSWMAP algorithm, and it has made parallel turbo decoder to consume 50 % lesser memory as compared to the reported works.

With the goal of conceiving high-throughput architecture of parallel turbo decoder, we have proposed a new un-grouped backward recursion based sliding window technique for MAP decoding. Subsequently, a new method of state-metric normalization was introduced and it

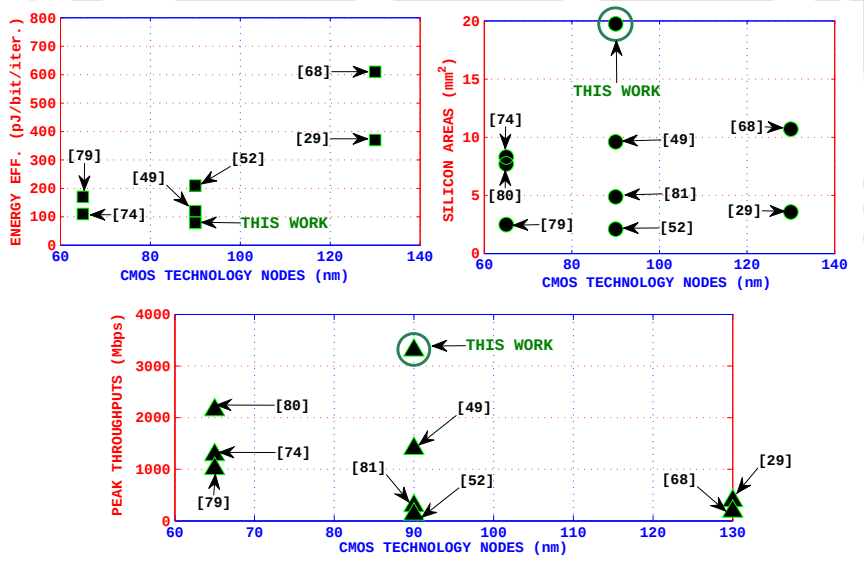


FIGURE 6.1: Comparative plots of energy efficiency, area consumed and throughput achieved by the proposed and the state-of-the-art works.

has reduced the critical path delay by approximately 22 % in comparison with the state-of-the-art normalization techniques. Multi-clocked high-speed MAP decoders, which were deeply

pipelined, have been incorporated in the parallel turbo decoder architecture to achieve throughputs of 3.31 Gbps and 2.27 Gbps at decoding iterations of 5.5 and 8 respectively. Highly-parallel turbo decoders with 8 and 64 MAP decoders, were implemented in 90 nm CMOS technology node, and have achieved best energy efficiencies of 0.11 and 0.079 nJ/bit/iteration respectively. Comparative plots of throughput achieved, design area and energy efficiency of the proposed and reported parallel-turbo decoders are illustrated in Fig. 6.1. In comparison with the state-of-the-art works, this work has better throughput and energy efficiency; however, it has some area overheads. Finally, the hardware prototype of such parallel turbo decoder, using ALTERA-FPGA board (Cyclone V SoC 5CSXFC6D6F31C8ES device), was tested in a communication environment and the outputs were verified on a logic analyzer.

## 6.2 Future Directions

Another linear error-correcting code which is termed as LDPC code has exceptionally good error-rate performance and the formulation of this code was an original work of Robert G. Gallager [103]. Although, this idea was coined in the year 1963, its practical importance was rediscovered by Yu Kou et al., in the year 2001 [104]. LDPC codes have already been adopted by various wireless communication standards like ETSI DVB-S, IEEE 802.11n and IEEE 802.16e [106, 107] and such code is an alternative option for the next generation wireless communication systems. Thereby, our future work includes design and implementation of high-throughput LDPC decoder that is suitable for evolving next generation wireless communication standards. On the other side, there is a strong resemblance between the characteristics of turbo and LDPC decoding algorithms, since, they are iterative processes, works on a graph-based representation and both are routinely implemented in logarithmic form. The next direction of our future work is to conceive a reconfigurable high-throughput turbo-LDPC decoder for multi-standard applications.

# Appendix A

# Design Flow from RTL to GDSII using Synopsys and CADence EDA-Tools

In this appendix, we have presented various steps involved in frontend as well as backend procedures of RTL (register transfer level) to GDSII (graphic database system for information interchange) design flow. This RTL-GDSII flow is presented for 90 nm CMOS process.

## A.1 Frontend Design Flow

In our work, we have used Synopsys tools for the frontend design-procedure. Red-Hat-Linux (version 5.0) operating system has been used and the commands <***csh***> and <***source synopsys.cshrc***> are consecutively executed to invoke Synopsys tool. Comprehensive discussion on step-by-step procedure of the frontend design-flow is presented as follows.

1) ***Logical and Functional Verification***: In this process of design, functionality as well as logics of the digital architectures, which are application specific, are simulated and verified using Synopsys-VCS (verilog compiler and simulator) tool [108]. We have used Verilog-HDL (hardware

descriptive language) to develop codes for the digital-designs. The working directory for this process contains verilog-HDL codes (in *.v* format) for an application specific digital-design and its test-bench. Thereafter, in the working-directory command prompt, we can use <***vcs -Mupdate -RI design_filename.v testbench_filename.v +v2k***> command for simulating these codes to open GUI (graphical user interface) to observe test waveforms, as shown in Fig. A.1, only if there are no syntax errors in Verilog-HDL code of the design. This process is repetitively carried out, until the output waveforms display expected values of designed architecture.

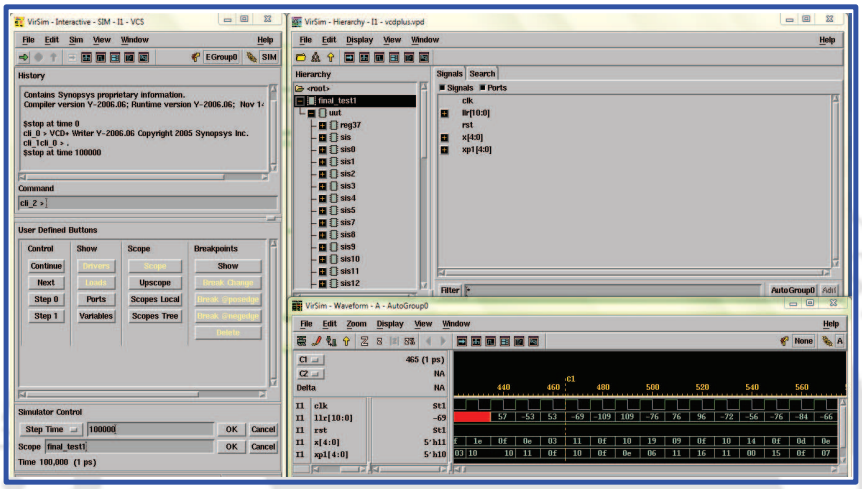

FIGURE A.1: GUI invoked by Synopsys-VCS tool for logical and functional verification of the digital design.

2) ***Design Synthesis***: In this process, logically and functionally verified verilog-HDL codes are synthesized to generate a design-netlist, using the *Faraday* standard-cell-libraries of 90 nm CMOS process, those are provided by UMC (united microelectronics corporation) semiconductor-foundry. For this purpose of design-synthesis, we have used Synopsys-DC (design compiler) tool which is a script based powerful software [109–113]. Prior to the synthesis-process, working directory must contain some of the important folders for systematic-flow, for example: *libs*, *DC_srcipt*, *nets*, *reports*, *sdc* and *src*. The *libs* folder contains standard-cell-libraries of different process corners for synthesis-process:

- *fsd0a_a_generic_core_ss0p9v125c.db* for the worst corner-case,
- *fsd0a_a_generic_core_tt1v25c.db* for the typical corner-case and
- *fsd0a_a_generic_core_ff1p1vm40c.db* for the best corner-case;

files like *standard.sldb*, *dw_foundation.sldb* and *fsd0a_a_generic_core.sdb* are also included in this folder. *DC_srcipt* folder contains TCL (tool command language) scripted files which are used for setting various timing constraints for the design, like clock period (clock frequency), clock

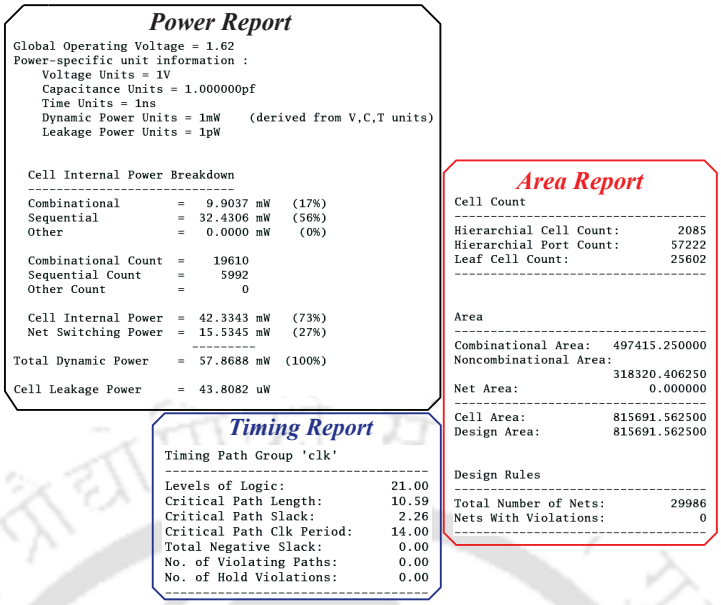


FIGURE A.2: Snapshots of power, area and timing reports generated by Synopsys-DC tool on synthesizing the HDL codes of designs.

pulse-width (clock duty-cycle), input delay, output delay, clock latency, clock uncertainty for setup as well as hold delays, clock transition-time and clock load. Additionally, these scripts are design for instructing Synopsys-DC tool to set a wire-load-model and a standard-cell-library for the synthesis of verilog-HDL code. They also define the magnitude of compiling-effort for area and power, while synthesizing a design. After the synthesis process, final netlist (in *.v* format) as well as synthesis-reports (in *.rpt* format), which include power, area and timing information, are written in *nets* and *reports* folders respectively. Similarly, information regarding input-delays and output-delays for input-ports and output-ports, respectively, with respect to clock signals are written in a file with *.sdc* (synopsys design constraint) extension and such file is used in the backend design-process. *src* folder contains verilog-HDL codes of the designs for synthesis. One of the crucial step is to include *.synopsys_dc.setup* file in the working directory because it sets an environment for the Synopsys-DC tool to run. In order to invoke Synopsys-DC tool from the working directory command-prompt, we can use <***dc_shell-xg-t***> command, which invokes Synopsys-DC tool where we can run our final TCL script for synthesis, using a command <***source working_directory_name/final_script.tcl***>. Finally, the generated netlist are checked and its reports are analysis. Snapshots for some portions of the reports generated by Synopsys-DC tool are shown in Fig. A.2.

3) ***Post Synthesis Simulation***: Basically, this is an essential step to verify the functionality of design-netlist, that is generated by Synopsys-DC tool. A file (named as *fsd0a_a_generic_core_21.v*)

containing verilog-HDL description of each standard-cells, in the 90 nm CMOS process standard-cell-library, must be included in the working directory for post synthesis simulation. Thereby, the working directory must contain design-netlist, test-bench and a verilog-HDL description file of standard-cells. We can use Synopsys-VCS tool for the simulation with a command <***vcs -Mupdate -RI design_netlist.v testbench_filename.v fsd0a_a_generic_core_21.v +v2k***>, to observe the output waveform and then verify with logically simulated outputs, as shown in Fig. A.1.

4) ***Static Timing Analysis***: A question arises in our mind: we have already accomplished timing analysis as well as verified slacks for all the paths in our design during the synthesis-process of Synopsys-DC tool, now, why do we need to perform static timing analysis for the same design? Such an analysis is essential to build a design that is free from timing-violations, as this process performs comprehensive timing analysis for all the possible paths between flip-flop to flip-flip including combinational logic in between, inputs to flip-flops, flip-flops to outputs and direct-paths from inputs to outputs, as shown in Fig. A.3. Unlike such analysis, the Synopsys-DC tool can check timing violations and computes slacks only for those paths lying between flip-flop to flip-flip across the combinational logic. We have used Synopsys-PT (prime time) tool to perform such static timing analysis for the design-netlist [114–117]. The standard-cell-libraries

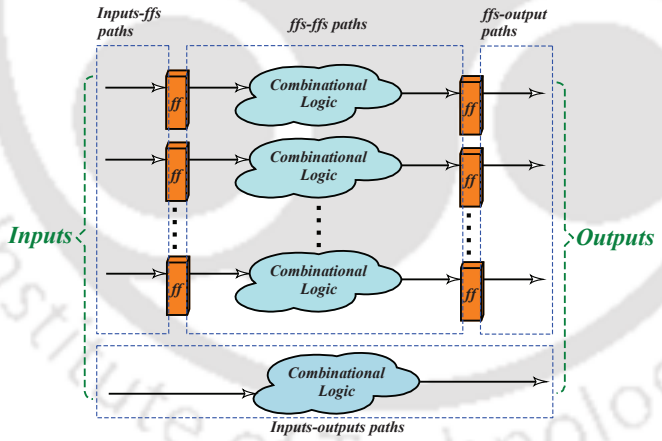


FIGURE A.3: All the possible paths of digital-design architecture; these paths are static-timing-analyzed by Synopsys-PT tool.

for worst and best corner-cases are used for checking setup and hold time-violations respectively. At this stage of design-process, all the setup-time violations must be mitigated, nevertheless, few hold-time violations may exist. Such hold-time violated paths can be corrected by adding buffers to these paths and is possible during the backend design. The working directory for such timing-analysis must include a TCL script which sets the standard-cell-libraries for analysis, decides a maximum number of paths for analysis as well as contains additional commands for

timing verification of various paths, as discussed earlier. In order to invoke Synopsys-PT tool, we must use <***pt_shell***> command, then run TCL script for timing analysis, with the same command that is used in Synopsys-DC tool. After the timing specifications of the design-netlist are met, it is termed as a ***golden-netlist*** which is ready for the backend design-process.

## A.2 Backend Design Flow

In this section, we present a detail description of backend design-process using CADence tools. Systematic procedure for this design process is presented as follows.

1) ***Integration of Design-netlist with Pads***: In this process, ***golden-netlist*** is integrated with various pads like, programmable digital-input/output pads, corner pads, power pads and ground pads. On the other hand, analog input/output pads are also used, if there are analog designs to be integrated on a same SOC (system on chip). Additionally, we require R(right)-cut and L(left)-cut cells for segregating analog and digital power domains. An interfacing code (in *.v* format) is used for instantiating netlist of digital-design, submodule for defining pads and LEF (library exchange format) files for analog-designs as well as hard-macros. Another file (with *.io* extension) is created for an orientation of pads around the core-area of chip. Snapshot of such file and four different directions of the chip, with corner-pad orientations, are shown in Fig. A.4.

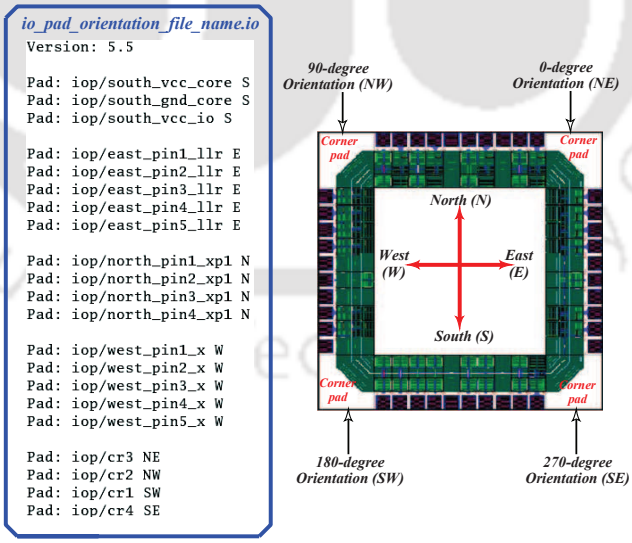


FIGURE A.4: Snapshot of *.io* file for the orientation of pads along various directions of chip-layout and the degree of orientation for corner-pads.

2) ***Essential Files for Backend Design***: Various files with *.LEF* extension, termed as LEF files, are the key requirements for backend design. In general, LEF file contains specifications for

the physical layout of integrated circuits. Semiconductor-foundry provides these standard LEF files for various metal layers. We have used six metal layers for backend design in this work. A LEF file called header-file (*header6m024_V55.lef*) contains information regarding the physical layouts of all the metal-layers (metal1-metal6) as well as vias, those are used in design-layout. These information include metal-layer width, pitch, spacing, offsets, area, capacitance etc. Layout information for all the core-standard-cells and the pads, for six metal-layers, are included in the LEF files *fsd0a_a_generic_core.lef* and *fod0a_b25_t33_generic_io.6m024.lef* respectively. Additionally, LEF files for antenna-cells, which mitigates antenna-effect in the design (these are diodes which drains current), are *FSD0A_A_GENERIC_CORE_ANT_V55.6m024.lef* and *FOD0A_B25_T33_GENERIC_IO_ANT_V55.7m124.lef* for core-standard-cells and pads respectively. If there are any analog design or hard macros (for eg: SRAM hard-macro) then there LEF files must be included along with the LEF files for analog pads and its antenna diodes (such as *fod0a_b33_t33_analogesd_io.6m024.lef* and *FOD0A_B33_T33_ANALOGESD_IO_ANT_V55.7m124.lef*).

Similarly, timing library files (in *.lib* format) for various corner cases are need for core-standard-cells and pads, they are listed as follows.

- *fsd0a_a_generic_core_ff1p1vm40c.lib* best-corner-case for core,
- *fsd0a_a_generic_core_ss0p9v125c.lib* worst-corner-case for core,
- *fsd0a_a_generic_core_tt1v25c.lib* typical-corner-case for core,
- *fod0a_b25_t33_generic_io_ff1p1vm40c.lib* best-corner-case for pads,
- *fod0a_b25_t33_generic_io_ss0p9v125c.lib* worst-corner-case for pads, and
- *fod0a_b25_t33_generic_io_tt1v25c.lib* typical-corner-case for pads.

The Synopsys-design-constraint file (in *.sdc* format), which is generated by Synopsys-DC tool, is also used in the backend design. In summary, the files required for starting a backend design-process are

- integration code (in *.v* format),
- pad orientation code (in *.io* format),
- LEF-files (with *.lef* extension),
- Timing library files (with *.lib* extension) and

- SDC file (with *.sdc* extension).

3) ***Backend Design-flow using CADence-SOC-Encounter Tool***: On executing commands <***csh***> and <***source cadence.cshrc***>, consecutively, CADence tool is invoked. In the command prompt of working directory, which contains all the required files, CADence-SOC encounter tool can be invoked using a command <***encounter***> [118–121]. In the GUI invoked by this

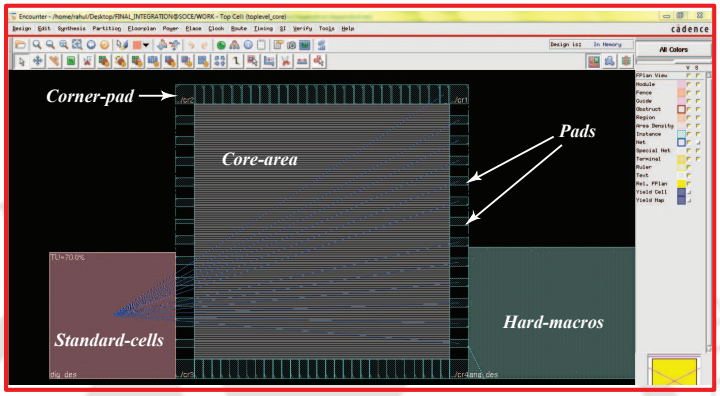


FIGURE A.5: GUI of SOC-Encounter after importing standard-cells, hard-macros and pads. It also shows the connections of standard-cells with pads.

tool, we can import all the files using an option ***Design/Import Design*** from GUI, and then save this configuration in a file (with *.conf* extension). On doing this, all the pads along with standard-cells as well as hard-macros are instantiated, as shown in Fig. A.5. Thereafter, we need to floor-plan the design using an option ***Floorplan/Specify Floorplan*** from GUI. Using this option, various design-metrics such as core-area, die-area and distance between core and pad-boundary are fixed. These values must be set in such a way that the core-utilization must be between 75% to 85%. Macros are dragged and dropped on the core-area, then the halo-ring is placed around this macro using an option ***Floorplan/Edit Floorplan/Edit Halo*** from GUI. Such halo-ring prevents standard-cells from reaching the macros. Thereafter, the next step is to set VCC and GND pins as global-nets and tie them to high and low values respectively. It can be done via ***Floorplan/ Connect Global Net*** option from GUI. Power-ring around the core area is placed using ***Power/ Power Planning/ Add Rings*** option. Here, we can set the metal width for these rings, odd and even numbered metals are used for horizontal and vertical directions, respectively; for example, metal-5 for horizontal-direction and metal-6 for vertical-direction. Similarly, the power-strips on core-area can be placed using an option ***Power/ Power Planning/ Add Stripes***. Then, the core-standard-cells are placed on unoccupied space of the core-area and this is done using an option ***Place/ Standard Cells and Blocks/*** from GUI. Here, ***Run Full Placement*** option is selected and then the placement-process is triggered. Fig.

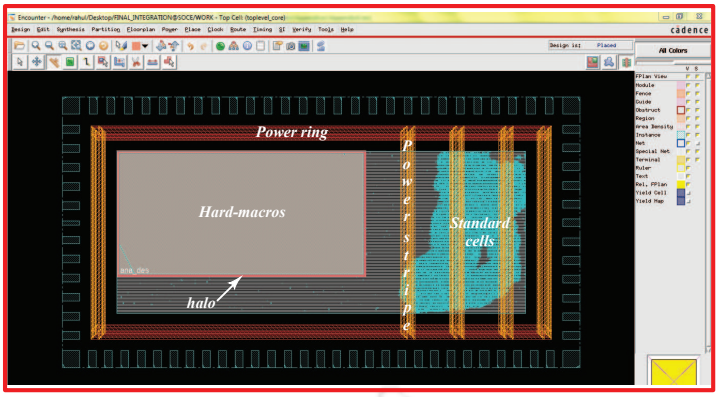


FIGURE A.6: GUI of SOC-Encounter after placing standard-cells and hard-macros with halo on the core-area. Power planning for the chip-layout shows the power rings and stripes.

A.6 shows the complete-layout of placed standard-cells as well as macros along with power rings and stripes.

***Hold-time violated report with negative slacks after STA***

```
------------------------------------------------------------
          timeDesign Summary
------------------------------------------------------------

+--------------------+---------+---------+---------+---------+---------+---------+
|     Hold mode      |   all   | reg2reg | in2reg  | reg2out | in2out  | clkgate |
+--------------------+---------+---------+---------+---------+---------+---------+
|           WNS (ns):| -0.380  | -0.380  |  2.653  |  0.839  |   N/A   |   N/A   |
|           TNS (ns):|-316.060 |-316.060 |  0.000  |  0.000  |   N/A   |   N/A   |
|    Violating Paths:|  1358   |  1358   |    0    |    0    |   N/A   |   N/A   |
|         All Paths:|  3031   |  3019   |   11    |    1    |   N/A   |   N/A   |
+--------------------+---------+---------+---------+---------+---------+---------+

Density: 26.851%
Routing Overflow: 0.00% H and 0.00% V
------------------------------------------------------------
Reported timing to dir timingReports
Total CPU time: 12.14 sec
Total Real time: 42.0 sec
```

**(a)**

***Timing report after the optimazation of hold-time violation***

```
------------------------------------------------------------
          timeDesign Summary
------------------------------------------------------------

+--------------------+---------+---------+---------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg | in2reg  | reg2out | in2out  | clkgate |
+--------------------+---------+---------+---------+---------+---------+---------+
|           WNS (ns):|  2.397  |  2.397  | 10.335  |  8.105  |   N/A   |   N/A   |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |  0.000  |   N/A   |   N/A   |
|    Violating Paths:|    0    |    0    |    0    |    0    |   N/A   |   N/A   |
|         All Paths:|  3031   |  3019   |   11    |    1    |   N/A   |   N/A   |
+--------------------+---------+---------+---------+---------+---------+---------+

+--------------------+---------------------+---------+
|                    |        Real         |  Total  |
|        DRVs        +---------+-----------+---------|
|                    |   Nr    | Worst Vio |   Nr    |
+--------------------+---------+-----------+---------+
|   max_cap          |    5    |   -0.164  |    5    |
|   max_tran         |   166   |   -2.141  |   166   |
|   max_fanout       |    0    |      0    |    0    |
+--------------------+---------+-----------+---------+

Density: 26.851%
Routing Overflow: 0.00% H and 0.00% V
------------------------------------------------------------
Reported timing to dir timingReports
Total CPU time: 13.24 sec
Total Real time: 47.0 sec
```

**(b)**

FIGURE A.7: Timing reports of (a) static timing analysis (b) timing optimization.

Now, an important design-process called CTS (clock tree synthesis) is carried out. This can be initiated using an option ***Clock/Design Clock***, where all the buffers and delays are selected using ***General Specification*** icon from the GUI. After the clock-tree has been designed,

we need to carry out STA (static timing analysis). Firstly, the min-max timing-libraries are set using an option ***Timing/ Analysis Condition/ Specify Operating Condition*** from the GUI. Then, the analysis mode (hold / setup) is set using an option ***Timing/ Analysis***

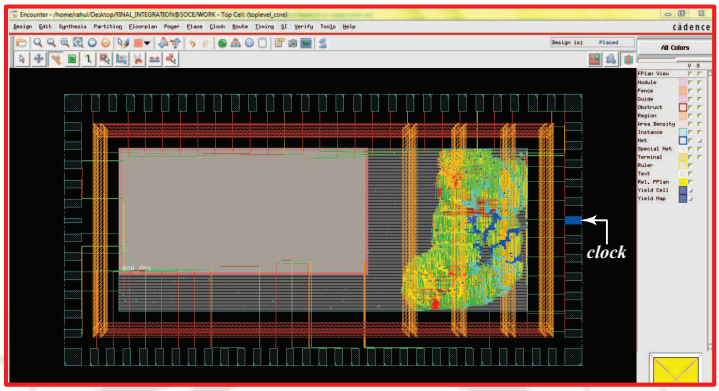


FIGURE A.8: Chip-layout obtained after clock tree synthesis.

***Condition/ Specify Analysis Mode***. Eventually, STA is initiated via ***Timing/ Analysis Timing*** option, where ***Post-CTS*** is selected for hold/setup time violations. Usually, there are no setup times violations at this stage, however, hold time violations may exist (which is indicated by negative slack in the timing report as shown in Fig. A.7 (a)). In order to mitigate this hold-violations, ***Timing/Optimize*** option is selected to open a GUI where ***Post-CTS*** optimization for hold-time violation is initiated. This process has to be carried out, iteratively, until the hold violations are removed from the design and produce result as shown in Fig. A.7 (b). Thereby, Fig. A.8 shows the clock-tree-synthesized layout of the design. Each of the standard-cells and macros need supply as well as ground connections. Power routing connects power-rings and stripes with power and ground pads. Thereby, the standard-cells are provided with supply and ground via these rings as well as stripes. Such power routing is accomplished using an option

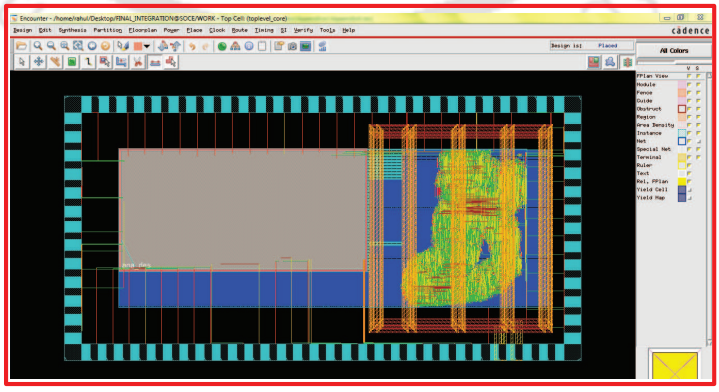

FIGURE A.9: Final chip-layout obtained from SOC-Encounter tool.

***Route/Special Route***, where the power routing is initiated with default setting. Signal-routing

among the standard-cells on the core-area, as specified in the design-netlist is carried out using ***Route/NanoRoute/Route*** option. Thereafter, STA and optimization of the routed layout is again performed in a similar way as earlier, but, the entire analysis is carried out by selecting ***Post-Route*** option. Finally, core-filler-cells are placed along the empty spaces on the core-area by selecting ***Place/Filler/Add Filler***, we have to add all the filler cells available. Similarly, IO-filler-cells are also included to maintain the continuity in pad-ring via ***Place/Filler/Add IO Filler*** option from GUI. It is to be noted that the option ***Fill Any Gap*** must be selected while adding these IO-filler-cells to completely cover the gaps. Finally, the layout is verified for DRC errors, process-antenna, metal-density and connectivity by using ***Verify*** option from GUI. Finally, verified-layout of the design is shown in Fig. A.9. Then, the verified layout is saved as a file (with *.gds* extension) via ***Design/Save/GDS***, additionally, a file which is termed as *streamout.map* is also saved.

4) ***Integration of Bond-pads using CADence-Virtuoso Tool***: In this work, we have used Virtuoso tool from CADence to integrate the layout of design as well as digital/analog input-output pads with the bond-pads [122, 123]. Basically, the entire layout generated by

*streamout.map file generated by CADence-SOC-Encounter tool*

```
NAME        metal3/NET          51          0
NAME        metal3/SPNET        52          0
NAME        metal3/PIN          53          0
NAME        metal3/LEFPIN       54          0
via3        LEFPIN      55          0
via3        FILL        56          0
via3        FILLOPC     57          0
via3        VIA         58          0
via3        VIAFILL     59          0
via3        VIAFILLOPC          60          0
metal4      NET         61          0
metal4      SPNET       62          0
metal4      PIN         63          0
metal4      LEFPIN      64          0
metal4      FILL        65          0
metal4      FILLOPC     66          0
metal4      VIA         67          0
metal4      VIAFILL     68          0
metal4      VIAFILLOPC          69          0
metal4      LEFOBS      70          0
NAME        metal4/NET          71          0
NAME        metal4/SPNET        72          0
NAME        metal4/PIN          73          0
NAME        metal4/LEFPIN       74          0
NAME        COMP        75          0
COMP        ALL         76          0
DIEAREA     ALL         77          0
```

*streamout.map file edited for CADence-Virtuoso tool*

```
NAME        label       51          0
NAME        label       52          0
NAME        label       53          0
NAME        label       54          0
VI3         drawing     55          0
VI3         drawing     56          0
VI3         drawing     57          0
VI3         drawing     58          0
VI3         drawing     59          0
VI3         drawing     60          0
ME4         drawing     61          0
ME4         drawing     62          0
ME4         drawing     63          0
ME4         drawing     64          0
ME4         drawing     65          0
ME4         drawing     66          0
ME4         drawing     67          0
ME4         drawing     68          0
ME4         drawing     69          0
ME4         drawing     70          0
NAME        label       71          0
NAME        label       72          0
NAME        label       73          0
NAME        label       74          0
NAME        label       75          0
prBoundary  label       76          0
prBoundary  drawing     77          0
```

FIGURE A.10: Generated and edited *streamout.map* files of CADence-SOC-encounter and CADence-Virtuoso tools respectively.

CADence-SOC-encounter tool is imported in Virtuoso layout editor where the bond-pad layouts are instantiated and then integrated with the design layout. After the cadence tool is invoked, as discussed earlier, we may use <***icfb***> command to begin with virtuoso layout editor. First of all, the mapping file generated by CADence-SOC-encounter tool must be edited such that they are compatible with Virtuoso tool. This is a significant file because it contains metal-layer and vias information regarding the mapping from encounter to virtuoso tool. Fig. A.10 shows snapshots for the part of ***streamout.map*** file generated by CADence-SOC-encounter tool and the edited version of the same file for CADence-Virtuoso tool. On the other hand, the LEF-files used for

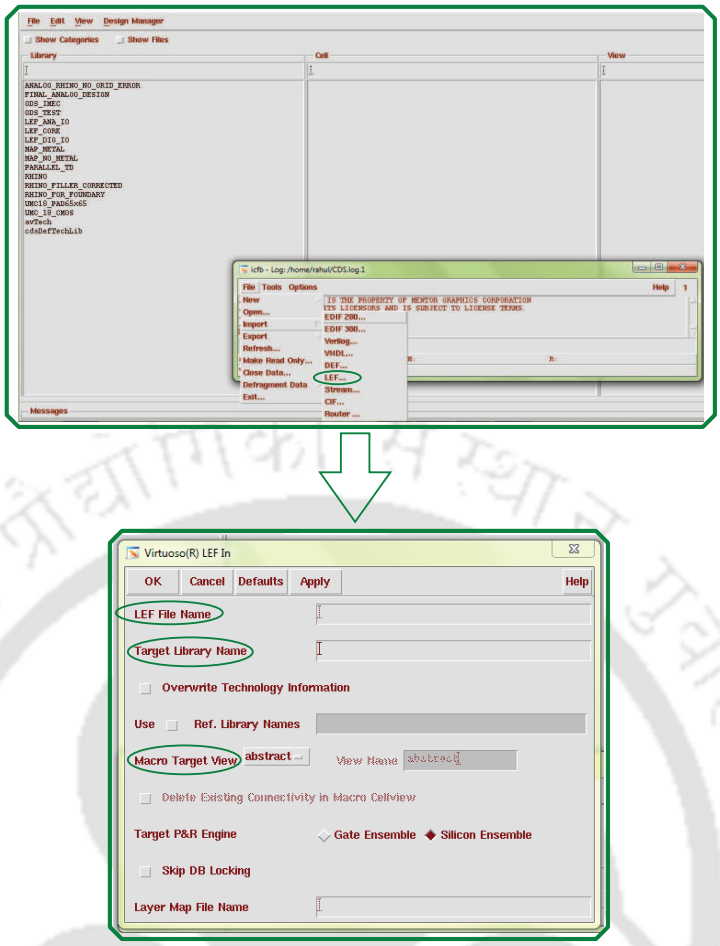


FIGURE A.11: GUI from CADence-Virtuoso tool for importing LEF files.

core, digital and analog pads (*fsd0a_a_generic_core.lef*, *fod0a_b25_t33_generic_io.6m024.lef* and *fod0a_b33_t33_analogesd_io.6m024.lef* respectively) must be imported in the CADence-Virtuoso tool. On doing this, layout of each standard cells as well as pads are created in this tool as per the number of metal layers used. Fig. A.11 shows the GUI which enables designers to enter any arbitrary file name in the box *LEF File Name* as well as the name of the LEF file along with the path for its location must be enter in ***Target library Name*** box. Similarly, the ***Macro Target view*** must be changed to ***Layout*** from ***Abstract***. After importing the LEF files it is necessary to check the layout of each standard cell. However, at this stage, the physical view of these layouts are not shown as it will be only visible after they are metal filled by the foundry. Thereby, such standard cell layout without physical view is shown in Fig. A.12.

Now, the gds file (with *.gds* extension) generated by CADence-SOC-encounter tool must be streamed into CADence-Virtuoso tool. It can be streamed-in using a ***stream*** option from the GUI shown in Fig. A.11. Thereafter, the GUI for stream-in (with heading '**Virtuoso ® Stream In**') appears, as shown in Fig. A.13. In this GUI, the gds file must be browsed and then

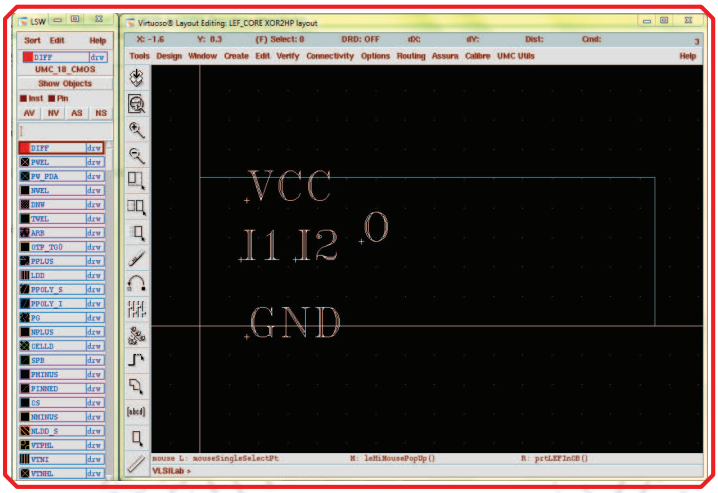


FIGURE A.12: Layout of two-input XOR-gate standard cell without a physical view after importing the LEF files in CADence-Virtuoso tool after importing the LEF files.

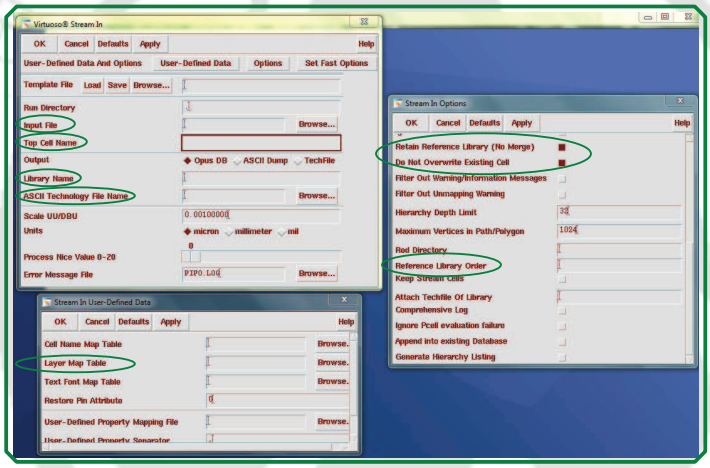


FIGURE A.13: GUI from CADence-Virtuoso tool for importing gds file generated by CADence-SOC-Encounter tool.

instantiated in the option ***Input File***; name of the top module, from an interfacing code for design netlist and pad, must be enter in the blank space of ***Top Cell Name*** option in GUI. The ***Library Name*** must be filled with any arbitrary name which will entitle the file containing the design-layout. Similarly, the technology file (with *.tf* extension) specific to a CMOS technology node is instantiated in the option ***ASCII Technology File Name***. As shown in Fig. A.13, ***User-Defined Data*** option has to be selected to instantiate an edited ***streamout.map*** file for CADence-Virtuoso tool. This can be accomplished by browsing and selecting such file via ***Layer MAP Table*** option of GUI (with a heading '**Stream In User-Defined Data**'). Thereafter, using an ***option*** icon from '**Virtuoso ® Stream In**' GUI, we open '**Stream In Options**' GUI where ***Retain Reference Library (No Merge)*** and ***Do Not Overwrite Existing Cell*** must be selected, as shown in Fig. A.13. Similarly, in the blank space of ***Reference Library Order***

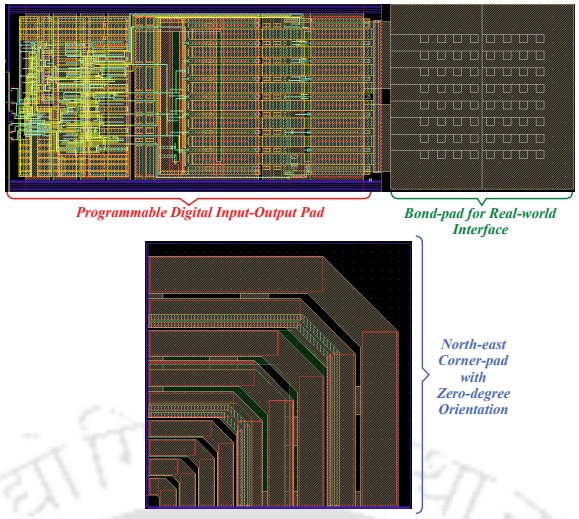


FIGURE A.14: Layouts of various pads displayed on CADence-Virtuoso layout editor.

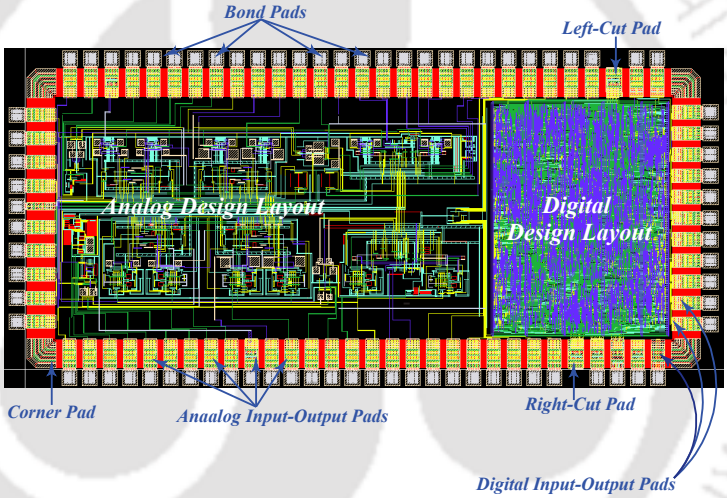


FIGURE A.15: Final layout of integrated-chip with digital and analog designs (mixed signal) for fabrication.

option, names of technology file as well as LEF files of standard cells and pads are included in the same order. On setting theses configurations and then executing this process-step, the layout of design which is integrated with input-output pads is created. On the same Virtuoso layout editor, we must instantiate the layout of bond-pads which is shown in Fig. A.14. Eventually, these pads are integrated with the design-layout and are check for DRC (design rule check) rules as well as LVS (layout versus schematic) match [124]. On the other hand, the netlist of this final layout is extracted and are subjected for post-layout simulation using *Nanosim* tool. After all these verifications, the final layout of design is shown in Fig. A.15 and the gds file is streamed out for this layout. Finally, we send this gds file to foundry for fabrication and start thinking of a test plans for the fabricated-chip.

# Abbreviations

## *A*

**ASIC** : **A**pplication **S**pecific **I**ntegrated **C**ircuit

**AWGN** : **A**dditive **W**hite **G**aussian **N**oise

**ADC** : **A**nalog to **D**igital **C**onverter

**ABS** : **A**bsolute-value unit

**ARP** : **A**lmost **R**egular **P**ermutation

**AGU** : **A**ddress **G**eneration **U**nit

**ACS** : **A**dd **C**ompare **S**elect

**APLLRC** : **A**-posteriori **L**ogarithmic **L**ikelihood **R**atio **C**omputation

**ALCU** : **A**-posteriori ***LLR*** **C**omputation **U**nit

**ACSU** : **A**dd **C**ompare **S**elect **U**nit

## *B*

**BCJR** : **B**ahl **C**ocke **J**elinek **R**aviv

**BER** : **B**it **E**rror **R**ate

**BPSK** : **B**inary **P**hase **S**hift **K**eying

**BMC** : **B**ranch **M**etrics **C**omputation

**BMR** : **B**ranch **M**etrics **R**outing

**BSMC** : **B**ackward **S**tate **M**etrics **C**omputation

**BRFE** : **B**ackward **R**ecursion **F**actor **E**stimator

**BMCU** : **B**ranch **M**etrics **C**omputation **U**nit

## *C*

**CMOS** : **C**omplementary **M**etal **O**xide **S**emiconductor

**CP** : **C**yclic **P**refix

**CMP** : **C**omparison-unit

**CTS** : **C**lock **T**ree **S**ynthesis

**CEs** : **C**onvolutional **E**ncoders

**CPU** : **C**entral **P**rocessing **U**nit

## *D*

**DVB-SH** : **D**igital **V**ideo **B**roadcasting - **S**atellite-services to **H**andhelds

**DVB-T** : **D**igital **V**ideo **B**roadcasting - **T**errestrial

**DAC** : **D**igital to **A**nalog **C**onverter

**DBSMC** : **D**ummy **B**ackward **S**tate **M**etrics **C**omputation

**DP-SRAMs** : **D**ual **P**ort **S**tatic - **R**andom **A**ccess **M**emories

**DSMC** : **D**ummy **S**tate **M**etrics **C**omputation

**DPU** : **D**eep **P**ipelined **U**nit

## *E*

**ETSI** : **E**uropean **T**elecommunications **S**tandards **I**nstitute

## *F*

**FPGA** : **F**ield **P**rogrammable **G**ate **A**rray

**FFT** : **F**ast **F**ourier **T**ransform

**FAs** : **F**ull **A**dders

**FSMC** : **F**orward **S**tate **M**etrics **C**omputation

## *G*

**GUI** : **G**raphical **U**ser **I**nterface

**GPIO** : **G**eneral **P**urpose **I**nput **O**utput

**GDS** : **G**raphic **D**atabase **S**ystem

## *H*

**HSMC** : **H**igh **S**peed **M**ezzanine **C**ard

**HDL** : **H**ardware **D**escriptive **L**anguage

**HSDPA** : **H**igh **S**peed **D**ownlink **P**acket **A**ccess

## *I*

**ITUR** : **I**nternational **T**elecommunication **U**nion **R**adiocommunication-sector

**IMT-A** : **I**nternational **M**obile **T**elecommunications - **A**dvanced

**IFFT** : **I**nverse **F**ast **F**ourier **T**ransform

**ISI** : **I**nter **S**ymbol **I**nterference

**IO** : **I**nput **O**utput

**ILA** : **I**ntegrated **L**ogic **A**nalyzer

**IMD** : **I**ntegrated **M**AP **D**ecoder

**ICON** : **I**ntegrated **C**ontroller

***ICNW*** : **I**nter **C**onnecting **N**etwork

## *J*

**JTAG** : **J**oint **T**est **A**ction **G**roup

## *L*

**LDPC** : **L**ow **D**ensity **P**arity **C**heck

**LUT** : **L**ook **U**p **T**able

**LBCJR** : **L**ogarithmic **B**ahl **C**ocke **J**elinek **R**aviv

**LEF** : **L**ibrary **E**xchange **F**ormat

**LCU** : ***LLR*** **C**omputation **U**nit

***LLR*** : **L**ogarithmic **L**ikelihood **R**atio

**LTE** : **L**ong **T**erm **E**volution

## *M*

**MAP** : **M**aximum **A**-posteriori **P**robability

**MSE** : **M**aclaurin **S**eries **E**xpansion

**msb** : **M**ost **S**ignificant **B**it

**MIMO** : **M**ultiple **I**nputs **M**ultiple **O**utputs

## *O*

**OFDM** : **O**rthogonal **F**requency **D**ivision **M**ultiplexing

## *P*

**PCCC** : **P**arallel **C**oncatenated **C**onvolutional **C**ode

**PDF** : **P**ower **D**elay **P**rofile

**PWLA** : **P**iece **W**ise **L**iner **A**pproximation

**PLL** : **P**hase **L**ock **L**oop

## *Q*

**QPSK** : **Q**uadrature **P**hase **S**hift **K**eying

**QAM** : **Q**uadrature **A**mplitute **M**odulation

**QPP** : **Q**uadratic **P**ermutation **P**olynomial

## *R*

**RF** : **R**adio **F**requency

**RSWMAP** : **R**educed **S**liding **W**indow **M**aximum **A**-posteriori **P**robability

**RSMCU** : **R**etimed **S**tate **M**etrics **C**omputation **U**nit

**RTL** : **R**egister **T**ransfer **L**evel

## *S*

**SISO** : **S**oft **I**nput **S**oft **O**utput

**SWs** : **S**liding **W**indows

**STA** : **S**tatic **T**iming **A**nalysis

**SAIF** : **S**witching **A**ctivity **I**nterchange **F**ormat

**SWBCJR** : **S**liding **W**indow **B**ahl **C**ocke **J**elinek **R**aviv

**SMC** : **S**tate **M**etrics **C**omputation

**SBMSs** : **S**tate **B**ranch **M**emory **S**avings

**SMCU** : **S**tate **M**etrics **C**omputation **U**nit

## *T*

**TCs** : **T**ransistor **C**ounts

**TSMC** : **T**aiwan **S**emiconductor **M**anufacturing **C**ompany

**TCL** : **T**ool **C**ommand **L**anguage

## *U*

**USB** : **U**niversal **S**erial **B**us

**UCF** : **U**ser **C**onstraint **F**ile

**UMC** : **U**nited **M**icroelectronics **C**orporation

## *V*

**VLSI** : **V**ery **L**arge **S**cale **I**ntegration

## *W*

**WiMAX** : **W**orldwide **I**nteroperability for **M**icrowave **A**ccess

**WCDMA** : **W**ideband **C**ode **D**ivision **M**ultiple **A**ccess

**3GPP** : **T**hird **G**eneration **P**artnership **P**roject

**2G** : **S**econd **G**eneration

**3G** : **T**hird **G**eneration

**4G** : **F**ourth **G**eneration

# Symbols

| | |
|---|---|
| $\Theta_T$ | Throughput of decoder |
| $\rho$ | Number of decoding iterations |
| $f$ | Operating clock frequency |
| $E_b/N_0$ | Signal-energy-per-bit to noise ratio |
| $\sigma_n^2$ | Noise variance |
| $Lc$ | Channel reliability measure |
| $M$ | Sliding window size |
| $K_r$ | Constraint length |
| $S_N$ or $N_s$ | Total number of states in each trellis stage |
| $T_{SW}$ | Total time required for tracing an entire sliding window |
| $P$ | Total number of MAP decoders used in a parallel turbo decoder |
| $LLR_k$ or $L_k(U_k)$ | A-posteriori logarithmic likelihood ratio |
| $L(U_k)$ or $L_{uk}$ | A-priori information |
| $\alpha_k(s)$ | Forward state metric |
| $\beta_k(s)$ | Backward state metric |
| $\gamma_k(s',s)$ | Branch metric |
| $a$ | Fading amplitude |
| $\mathbf{B}_k$ | Set of $S_N/N_s$ backward metrics |
| $\mathbf{A}_k$ | Set of $S_N/N_s$ forward metrics |
| $\mathbf{N}^0$ | Set of natural numbers including zero |
| $\Gamma_k$ | Set of all branch metrics |
| $\boldsymbol{U}$ | Set of all un-grouped backward recursions |

# Bibliography


[1] C. E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, vol. 27, pp. 379-423 (Part-1); pp. 623-656 (Part-2), 1948.

[2] C. Berrou, A. Glavieux and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes," *Proceedings of International Conference on Communication*, pp. 1064-1070, 1993.

[3] C. Berrou and A. Glavieux, "Near Optimum Error Correcting Coding and Decoding: Turbo-Codes," *IEEE Transactions on Communications*, vol. 44, pp. 1261-1271, 1996.

[4] C. Berrou and A. Glavieux, "Reflections on the Prize Paper: Near Optimum Error Correcting Coding and Decoding: Turbo-Codes," *IEEE Transactions on Information Theory*, vol. 48, no. 2, pp. 24-31, 1998.

[5] J. Hagenauer and P. Hoeher, "A Viterbi Algorithm with Soft-Decision Outputs and Its Applications," *Proceedings of IEEE Global Communications Conference (GLOBECOM)*, pp. 1680-1686, 1989.

[6] J. H. Lodge, P. Hoeher and J. Hagenauer, "The Decoding of Multidimensional Codes Using Separable MAP Filters," *Proceedings of 16$^{th}$ Biennial Symposium on Communications*, pp. 343-346, 1992.

[7] G. Battail, "Building Long Codes by Combination of Simple Ones, Thanks to Weighted-Output Decoding," *Proceedings of URSI International Symposium on Signal, Systems and Electronics*, pp. 634-637, 1989.

[8] G. Battail, M. Decouvelaere and P. Godlewski, "Replication Decoding," *IEEE Transactions on Information Theory*, vol. IT-25, no. 3, pp. 332-345, 1979.

[9] S. Benedetto and G. Montorsi, "Unveiling Turbo Codes: Some Results on Parallel Concatenated Coding," *IEEE Transactions on Information Theory*, vol. IT-42, pp. 409-428, 1996.

[10] S. Benedetto and G. Montorsi, "Design of Parallel Concatenated Convolutional Codes," *IEEE Transactions on Communications*, vol. COM-44, pp. 591-600, 1996.

[11] D. Divsalar and F. Pollara, "Serial and Hybrid Concatenated Codes with Applications," *Proceedings of 1st International Symposium on Turbo Codes*, pp. 80-87, 1997.

[12] S. Benedetto, D. Divsalar, G. Montorsi and F. Pollara, "Analysis, Design and Iterative Decoding of Double Serially Concatenated Codes with Interleavers," *IEEE Journal on Selected Areas in Communications*, vol. SAC-42, pp. 231-244, 1998.

[13] S. Benedetto, D. Divsalar, G. Montorsi and F. Pollara, "Serial Concatenation of Interleaved Codes: Performance Analysis, Design and Iterative Decoding," *IEEE Transactions on Information Theory*, vol. IT-44, pp. 909-926, 1998.

[14] D. Divsalar and F. Pollara, "Multiple Turbo Codes for Deep-Space Communications," *TDA Progress Report, Jet Propulsion Laboratory (California)*, pp. 42-121, 1995.

[15] D. Divsalar and F. Pollara, "On the Design of Turbo Codes," *TDA Progress Report, Jet Propulsion Laboratory (California)*, pp. 42-123, 1995.

[16] S. Benedetto, D. Divsalar, G. Montorsi and F. Pollara, "A Soft-Input Soft-Output Maximum a Posteriori (MAP) Module to Decode Parallel and Serial Concatenated Codes," *TDA Progress Report, Jet Propulsion Laboratory (California)*, pp. 42-127, 1996.

[17] S. Dolinar, D. Divsalar and F. Pollara, "Code Performance As a Function of Block Size," *TMO Progress Report, Jet Propulsion Laboratory (California)*, pp. 42-133, 1998.

[18] L. Bahl, J. Cocke, F. Jelinek and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Transactions on Information Theory*, vol. 20, pp. 284-287, 1974.

[19] "ETSI EN 302 583 V1.1.0, Digital Video Broadcasting (DVB); Implementation Guidelines for Satellite Services to Handheld Devices (SH) Below 3GHz," European Telecommunications Standards Institute (ETSI), Tech. Rep., 2008.

[20] G. Faria, T. Kurner, B. Lehembre and P. Unger, "Satellite digital broadcast services to handheld DVB-SH: The complementary ground component," *International Journals of Satellite Communication*, vol. 27, pp. 241-274, 2009.

[21] J. P. Woodard and L. Hanzo, "Comparative Study of Turbo Decoding Techniques: an overview," *IEEE Transactions on Vehicular Technology*, vol. 49, pp. 2208-2233, 2000.

[22] G. Masera, G. Piccinini, M. R. Roch and M. Zamboni, "VLSI Architectures for Turbo Codes," *IEEE Transactions on Very Large Scale Integrated (VLSI) Systems*, vol. 7, pp. 369-379, 1999.

[23] H. Michel, A. Worm and N. Wehn, "Influence of Quantization on the Bit-Error Performance of Turbo-Decoders," *Proceedings of IEEE Vehicular Technology Conference*, vol. 1, pp. 581-585, 2000.

[24] Y. Wu, B. D. Woerner and T. K. Blankenship, "Data Width Requirements in SISO Decoding with Modulo Normalization," *IEEE Transactions on Communications*, vol. 49, pp. 1861-1868, 2001.

[25] S. Vafi and T. Wysocki, "Weight Distribution of Turbo Codes with Convolutional Interleavers," *IET Communications*, vol. vol-1, pp. 71-78, 2007.

[26] A. Bhise and P. D. Vyavahare, "Performance Enhancement of Modified Turbo Codes with Two-Stage Interleavers," *IET Communications*, vol. 5, pp. 1336-1342, 2011.

[27] M. R. D. Rodrigues, I. Chatzigeorgiou, I. J. Wassell and R. Carrasco, "Performance Analysis of Turbo Codes in Quasi-Static Fading Channels," *IET Communications*, vol. 2, pp. 449-461, 2008.

[28] C. Benkeser, A. Burg, T. Cupaiuolo and Q. Huang, "Design and Optimization of an HSDPA Turbo Decoder ASIC," *IEEE Journal of Solid-State Circuits*, vol. 44, pp. 98-106, 2009.

[29] C. Studer, C. Benkeser, S. Belfanti and Q. Huang, "Design and Implementation of a Parallel Turbo-Decoder ASIC for 3GPP-LTE," *IEEE Journal of Solid-State Circuits*, vol. 46, pp. 8-17, 2011.

[30] S. Vafi and T. Wysocki, "Performance of convolutional interleavers with different spacing parameters in turbo codes," *Proceedings of Australian Communication Theory Workshop*, pp. 8-12, 2005.

[31] Y. Sun, Y. Zhu, M. Goel and J. R. Cavallaro, "Configurable and Scalable High Throughput Turbo Decoder Architecture for Multiple 4G Wireless Standards," *International Conference on Application-Specific System, Architecture and Processors*, pp. 209-214, 2008.

[32] M. A. Kousa and A. H. Mugaibel, "Puncturing Effects on Turbo Codes," *IEE Proceedings - Communication*, vol. 149, pp. 132-138, 2002.

[33] "Recommendation (1997) ITU-R M.1225. Guidelines for Evaluation of Radio Transmission Technologies for IMT-2000," 1997.

[34] J. Hou, P. H. Siegel and Laurence B. Milstein, "Performance Analysis and Code Optimization of Low Density Parity-Check Codes on Rayleigh Fading Channel," *IEEE Journals on Selected Areas in Communications*, vol. 19, pp. 924-934, 2001.

[35] S. Lin and D. J. Costello, Jr., "Error Control Coding," *Pearson Prentice Hall*, 2004.

[36] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Soft-Output Decoding Algorithms in Iterative Decoding of Turbo Codes," *JPL TDA Progress Rep.*, Rep. 42-124, 1996.

[37] M. Martina, M. Nicola and G. Masera, "A Flexible UMT-WiMax Turbo Decoder Architecture," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 55, pp. 369-373, 2008.

[38] S. Talakoub, L. Sabeti, B. Shahrrava and M. Ahmadi, "An Improved Max-Log-MAP Algorithm for Turbo Decoding and Turbo Equalization," *IEEE Transactions on Instrumentation and Measurement*, vol. 56, pp. 1058-1063, 2007.

[39] "(3GPP TS 36.212 version 10.0.0 Release 10)," *LTE: Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding*, 2011.

[40] C. Berrou, A. Glavieux and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes," *Proceedings of IEEE International Conference on Communications*, pp. 1064-1070, May-1993.

[41] W. J. Gross and P. G. Gulak, "Simplified MAP Algorithm Suitable for Implementation of Turbo Decoders," *Electronics Letters*, vol. 34, pp. 1577-1578, 1998.

[42] B. Classon, K. Blankenship and V. Desai, "Channel Coding for 4G Systems with Adaptive Modulation and Coding," *IEEE Wireless Communications*, vol. 9, pp. 8-13, April-2002.

[43] M. M. Mansour and N. R. Shanbhag, "A 640-Mb/s 2048-Bit Programmable LDPC Decoder Chip," *IEEE Journal of Solid-State Circuits*, vol. 41, pp. 684-698, 2006.

[44] J. Cheng and T. Ottosson, "Linearly Approximated Log-MAP Algorithms for Turbo Decoding," *Proceedings of IEEE Vehicular Technology Conference (VTC)*, vol. 3, pp. 2252-2256, 2000.

[45] X. Hu, E. Eleftheriou, D. Arnold and A. Dholakia, "Efficient Implementation of the Sum-Product Algorithm for Decoding LDPC Codes," *Proceedings of IEEE Global Telecommunication Conference*, vol. 2, pp. 1036-1036E, 2001.

[46] S. Papaharalabos, P. T. Mathiopoulos, G. Masera and M. Martina, "On Optimal and Near-Optimal Turbo Decoding Using Generalized max* Operator," *IEEE Communications Letters*, vol. 13, pp. 522-524, 2009.

[47] M. May, T. Ilnseher, N. Wehn and W. Raab, "A 150 Mbit/s 3GPP-LTE Turbo Code Decoder," *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1420-1425, 2010.

[48] R. Dobkin, M. Peleg and R. Ginosar, "Parallel VLSI Architecture for MAP Turbo Decoder," *Proceedings of IEEE International Symposium on Personal, Indoor Mobile Radio Communications*, pp. 15-18, 2002.

[49] C-C. Wong and H-C. Chang, "High-Efficiency Processing Schedule for Parallel Turbo Decoders Using QPP Interleaver," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, no. 6, pp. 1412-1420, June-2011.

[50] C-C. Wong, M-W. Lai, C-C. Lin, H-C. Chang and C-Y. Lee, "Turbo Decoder Using Contention-Free Interleaver and Parallel Architecture," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 2, pp. 422-432, February-2010.

[51] S. M. Karim and I. Chakrabarti, "High Throughput Turbo Decoder Using Pipelined Parallel Architecture and Collision Free Interleaver," *IET Communications*, vol. 6, pp. 1416-1424, 2012.

[52] C-C. Wong and H-C. Chang, "Reconfigurable Turbo Decoder With Parallel Architecture for 3GPP LTE System," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 57, pp. 566-570, July-2010.

[53] T-H. Tsai and C-H. Lin, "A New Memory-Reduced Architecture Design for Log-MAP Algorithm in Turbo Decoding," *IEEE 6th CAS Symposium on Emerging Technologies: Mobile and Wireless Communications*, vol. 2, pp. 607-610, 2004.

[54] T-H. Tsai, C-H. Lin and A-Y. Wu, "A Memory-Reduced Log-MAP Kernel for Turbo Decoder," *IEEE International Symposium on Circuits and Systems, ISCAS.*, vol. 2, pp. 1032-1035, 2005.

[55] C-H. Lin, C-Y. Chen, T-H. Tsai and A-Y. Wu, "Low-Power Memory-Reduced Traceback MAP Decoding for Double-Binary Convolutional Turbo Decoder," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, pp. 1005-1016, 2009.

[56] M. Martina and G. Masera, "State Metric Compression Techniques for Turbo Decoder Architectures," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, pp. 1119-1128, 2011.

[57] H. Wang, H. Yang and D. Yang, "Improved Log-MAP Decoding Algorithm for Turbo-Like Codes," *IEEE Communication Letters*, vol. 10, no. 3, pp. 186-188, 2006.

[58] M. Martina, G. Masera, S. Papaharalabos, P. Takis Mathiopoulos and F. Gioulekas, "On Practical Implementation and Generalizations of max* Operator for Turbo and LDPC Decoders," *IEEE Transactions on Instrumentation and Measurement*, vol. 61, no. 4, pp. 888-895, 2012.

[59] J. Vogt and A.Finger, "Improving the Max-Log-MAP Turbo Decoder," *Electronics Letters*, vol. 36, pp. 1937-1939, 2000.

[60] Z. Wang, Z. Chi and K. K. Parhi, "Area-Efficient High-Speed Decoding Scheme for Turbo Decoders," *IEEE Transactions on Very Large Scale Integrated (VLSI) Systems*, vol. 10, pp. 902-912, 2002.

[61] N. H. E. Weste and D. Harris, "CMOS VLSI Design: A Circuits and Systems Perspective," *Reading, MA: Pearson-Addison Welsley*, $3^{rd}$ International edition, 2005.

[62] S. Lee, C. Wang and W. Sheen, "Architecture Design of QPP Interleaver for Parallel Turbo Decoding," *Proceedings of IEEE Vehicular Technology Conference (VTC)*, pp. 1-5, 2010.

[63] H. Bhatnagar, "Advanced ASIC Chip Synthesis - Using Synopsys Design Compiler, Physical Compiler, and PrimeTime," *Kluwer Academic Publishers*, $2^{nd}$ Edition, 2002.

[64] M. Keating, "The Simple Art of SOC Design - Closing the Gap between RTL and ESL," *Springer Publishers*, 2011.

[65] M. Bickerstaff, L. Davis, C. Thomas, D. Garrett and C. Nicol, "A 24Mb/s Radix-4 Log MAP Turbo Decoder for 3GPP-HSDPA Mobile Wireless," *Proceedings of IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 1, pp. 150-484, 2003.

[66] M. A. Bickerstaff, D. Garrett, T. Prokop and C. Thomas, "A Unified Turbo/Viterbi Channel Decoder for 3GPP Mobile Wireless in 0.18 um CMOS," *IEEE Journal of Solid-State Circuits*, vol. 37, pp. 1555-1564, 2002.

[67] Myoung-Cheol and I. Park, "SIMD Processor-Based Turbo Decoder Supporting Multiple Third-Generation Wireless Standards," *IEEE Transactions on Very Large Scale Integrated (VLSI) Systems*, vol. 15, pp. 801-810, 2007.

[68] J. Kim and I. Park, "A Unified Parallel Radix-4 Turbo Decoder for Mobile WiMAX and 3GPP-LTE," *Proceedings of IEEE Custom Integrated Circuits Conference (CICC)*, pp. 487-490, 2009.

[69] Z. Wang, Y. Tang and Y. Wang, "Low Hardware Complexity Parallel Turbo Decoder Architecture," *Proceedings of IEEE International Symposium of Circuits and Systems (ISCAS)*, vol. 2, pp. 53-56, 2003.

[70] C. Cheng, Y. Tsai, L. Chen and A. P. Chandrakasan, "A 0.077 to 0.168 nJ/bit/iteration Scalable 3GPP LTE Turbo Decoder with an Adaptive Sub-Block Parallel Scheme and an Embedded DVFS Engine," *Proceedings of IEEE Custom Integrated Circuit Conference (CICC)*, pp. 1-4, 2010.

[71] C. Lin, C. Chen, E. Chang and A. Wu, "Reconfigurable Parallel Turbo Decoder Design for Multiple High-Mobility 4G Systems," *Journals of Signal Processing Systems (Springer US)*, pp. 1-14, 2013.

[72] J-H. Kim and I-C. Park, "A 50Mbps Double-Binary Turbo Decoder for WiMAX based on Bit-Level Extrinsic Information Exchange," *Proceedings of IEEE Asian Solid-State Circuits Conference (ASSCC)*, pp. 305-308, 2008.

[73] L. Hanzo, T. H. Liew and B. L. Yeap, "Turbo Coding, Turbo Equalisation and Space-Time Coding for Transmission over Fading Channels," *England: John Wiley & Sons*, June. 2003.

[74] Y. Sun and J. R. Cavallaro, "Efficient Hardware Implementation of a Highly-Parallel 3GPP LTE/LTE-Advance Turbo Decoder," *INTEGRATION, the VLSI Journal*, vol. 44, pp. 305-315, 2011.

[75] D. Talbot, "A Banner Year for Mobile Devices," *MIT Technology Review, COMMUNICATION NEWS*, December-2012.

[76] "3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)," *Multiplexing and Channel Coding (Release 9) 3GPP Organizational Partners TS 36.212*, Rev. 8.3.0, May 2008.

[77] "3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA)," *Multiplexing and Channel Coding (Release 10) 3GPP Organizational Partners TS 36.212*, Rev. 10.0.0, 2011.

[78] P. Bhat, S. Nagata, L. Campoy, I. Berberana, T. Derham, G. Liu, X. Shen, P. Zong and J. Yang, "LTE-Advanced: An Operator Perspective," *IEEE Communications Magazine*, vol. 50, no. 2, pp. 104-114, 2012.

[79] S. Belfanti, C. Roth, M. Gautschi, C. Benkeser and Q. Huang, "A 1Gbps LTE-Advanced Turbo-Decoder ASIC in 65nm CMOS," *IEEE Symposium on VLSI Circuits (VLSIC)*, pp. C284-C285, 2013.

[80] T. Ilnseher, F. Kienle, C. Weis and N. Wehn, "A 2.15 GBit/s Turbo Code Decoder for LTE Advanced Base Station Applications," *International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, pp. 21-25, 2012.

[81] C. Condo, M. Martina and G. Masera, "VLSI Implementation of a Multi-Mode Turbo/LDPC Decoder Architecture," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 6, pp. 1441-1454, 2013.

[82] H. Dawid and H. Meyr, "Real-time Algorithms and VLSI Architectures for Soft Output MAP Convolutional decoding," *Sixth IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, vol. 1, pp. 193-197, 1995.

[83] D. Wang and H. Kobayashi, "Matrix Approach for Fast Implementations of Logarithmic MAP Decoding of Turbo Codes," *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, vol. 1, pp. 115-118, 2001.

[84] S. Lee, N. R. Shanbhag and A. C. Singer, "A 285-MHz Pipelined MAP Decoder in 0.18-um CMOS," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 8, pp. 1718-1725, 2005.

[85] A. P. Hekstra, "An Alternative to Metric Rescaling in Viterbi Decoders," *IEEE Transactions on Communications*, vol. 37, pp. 1220-1222, November 1989.

[86] M. J. S. Smith, "Application-Specific Integrated Circuits," *Pearson Education (Singapore)*, Seventh Indian Reprint, 2003.

[87] N. Baneerjee, K. Roy, H. Mahmoodi and S. Bhunia, "Low Power Synthesis of Dynamic Logic Circuits Using Fine-Grained Clock Gating," *IEEE Proceedings of Design, Automation and Test in Europe (DATE '06)*, vol. 1, pp. 1-6, March 2006.

[88] H. Li, S. Bhunia, Y. Chen, K. Roy and T. N. Vijaykumar, "DCG: Deterministic Clock-Gating for Low-Power Microprocessor Design," *IEEE Transactions on Very Large Scale Integrated (VLSI) Systems*, vol. 12, pp. 245-254 , March 2004.

[89] C. Lin, C. Chen and A. Wu, "Area-Efficient Scalable MAP Processor Design for High-Throughput Multistandard Convolutional Turbo Decoding," *IEEE Transactions on Very Large Scale Integrated (VLSI) Systems*, vol. 19, no. 2, pp. 305-318, 2011.

[90] C. Tang, C. Wong, C. Chen, C. Lin and H. Chang, "A 952MS/s Max-Log MAP Decoder Chip using Radix-4 x 4 ACS Architecture," *IEEE Asian Solid-State Circuits Conference (ASSCC)*, pp. 79-82, 2006.

[91] A. Pulimeno, M. Graziano and G. Piccinini, "UDSM trends comparison: From technology roadmap to UltraSparc Niagara2," *IEEE Transactions on Very Large Scale Integrated (VLSI) Systems*, vol. 20, no. 7, pp. 1341-1346, 2012.

[92] Z. Navabi, "Digital Design and Implementation with Field Programmable Devices," *Springer*, 2005.

[93] W. F. Lee, "Verilog Coding for Logic Synthesis," *A JOHN WILEY & SONS, INC., PUBLICATION*, 2003.

[94] S. Palnitkar, "Verilog HDL: A Guide to Digital Design and Synthesis," *Prentice Hall PTR, Second Edition*, February 2003.

[95] "Constraints Guide," *UG625 (v. 13.2)*, July 2011.

[96] "ChipScope Pro 10.1 Serial I/O Toolkit User Guide," *UG213 (v10.1)*, March 2008.

[97] "ChipScope ILA Tools Tutorial (for ChipScope ILA Software v4.2i)," *UG044 / PN 0401957 (v4.2.2)*, July 2003.

[98] O. F. Acikel and W. E. Ryan, "Punctured Turbo-Codes for BPSK/QPSK channels," *IEEE Transactions on Communications*, vol. 47, no. 9, pp. 1315-1323, 1999.

[99] C. Hall, "Performance Analysis and Design of Punctured Turbo Codes," *Doctoral thesis: University of Cambridge, Departemt of Engineering*, 2006.

[100] K. Arshak, E. Jafer and C. Ibala, "Testing FPGA based Digital System using XILINX ChipScope$^{TM}$ Logic Analyzer," *29$^{th}$ International Spring Seminar on Electronics Technology (ISSE '06)*, pp. 355-360, 2006.

[101] "Quartus II Hanbook Version 13.1, Volume 1: Design and Synthesis," *ALTERA Corporation*, November 2013.

[102] "Cyclone V Device Handbook, Volume 1: Device Interfaces and Integration (Version 2013.11.12)," *ALTERA Corporation*, November 2013.

[103] R. G. Gallager, "Low-Density Parity-Check Codes," *Doctoral thesis: Massachusetts Institute of Technology*, 1963.

[104] Y. Kou, S. Lin and M. P. C. Fossorier, "Low-Density Parity-Check Codes Based on Finite Geometries: A Rediscovery and New Results," *IEEE Transactions on Information Theory*, vol. 47, no. 7, pp. 2711-2736, November 2001.

[105] G Moore, "Cramming More Components on Integrated Circuits," *Electonics Magazine*, vol. 38, no. 8, April 1965.

[106] IEEE 802.16e, "LDPC Coding for OFDMA PHY," *IEEE Doc. C802-16e-05/066r3*, January 2005.

[107] IEEE 802.11n, "Structured LDPC Codes as an Advanced Coding Scheme for 802.11n," *IEEE Doc. 802.11-04/88r0*, August 2004.

[108] "VCS® MX Document Navigator 2005.06," *SYNOPSYS*, 2005.

[109] "Design Compiler Command-Line Interface Guide, Version Y-2006.06," *SYNOPSYS*, 2006.

[110] "Design Compiler Reference Manual: Constraints and Timing, Version Y-2006.06," *SYNOPSYS*, 2006.

[111] "Design Compiler Reference Manual: Optimization and Timing Analysis, Version Y-2006.06," *SYNOPSYS*, 2006.

[112] "Design Compiler Reference Manual: Register Retiming, Version Y-2006.06," *SYNOPSYS*, 2006.

[113] "Design Compiler User Guide, Version Y-2006.06," *SYNOPSYS*, 2006.

[114] "PrimeTime User Guide: Fundamentals, Version Z-2006.12," *SYNOPSYS*, 2006.

[115] "PrimeTime User Guide: Advanced Timing Analysis, Version Z-2006.12," *SYNOPSYS*, 2006.

[116] "PrimeTime Turotial, Version Z-2006.12," *SYNOPSYS*, 2006.

[117] "PrimeTime Modelling User Guide, Version Z-2006.12," *SYNOPSYS*, 2006.

[118] "SoC Encounter RTL-to-GDSII System: Full-Chip Implementation in a Single System," *CADENCE Online Support*, www.cadence.com.

[119] P. McCrorie, R. Fish and R. Goering, "Solutions for Mixed-Signal SoC Implementation," *CADENCE Design Systems, Incorporation.*

[120] J. Rodriques, "Physical Placement with Cadence SoC-Encounter 7.1," *Lunds University: Department of Electrical and Information Technology (Sweden)*, November 2008.

[121] T. W. Tseng, "Training Course of SoC Encounter," *Advanced Reliable Systems (ARES) Lab (Taiwan).*

[122] "Virtuoso Layout Suite GXL: Rapid Layout Implementation," *CADENCE Online Support.*

[123] E. Naviasky and M. Nizic, "Mixed-Signal Design Chanllenges and Requirements," *CADENCE Design Systems, Incorporation*, www.cadence.com.

[124] "Assura Physical Verification User Guide," *CADENCE: Product Version 4.1 USR2_HF2*, January 2011.

# List of Publications

## Refereed Journal Publications

1. Rahul Shrestha and Roy Paily, "High-Throughput Turbo Decoder with Parallel Architecture for LTE Wireless Communication Standards," *IEEE Transactions on Circuits and Systems I: Regular Papers*, (In Press).

2. Rahul Shrestha and Roy Paily, "Performance and Throughput Analysis of Turbo Decoder for the Physical Layer of Digital Video Broadcasting - Satellite-services to Handhelds (DVB-SH) Standard," *Journals of IET Communications*, Volume: 7, Issue: 12, pp. 1211-1220, 2013.

3. Rahul Shrestha and Roy Paily, "Design and Implementation of a Linear Feedback Shift Register Interleaver for Turbo Decoding," *Springer Berlin/Heidelberg Lecture Notes in Computer Science*, Volume: 7373, pp. 30-39, 2012.

## Manuscripts Submitted

1. Rahul Shrestha and Roy Paily, "Comparative Study of Simplified MAP Algorithms and Implementation of Efficient Turbo Decoder," *Journal of Signal Processing Systems - Springer*, (Submitted in May-2013).

2. Rahul Shrestha and Roy Paily, "Memory-Reduced Maximum-A-Posteriori-Probability Decoding for High-Throughput Parallel Turbo Decoders," *International Journal of Electronics and Communications - Elsevier*, (Submitted in November-2013).

## Refereed Conference Publications

1. Rahul Shrestha and Roy Paily, "Hardware Implementation of Max-Log-MAP Algorithm Based on Maclaurin Series for Turbo Decoder," *IEEE International Conference on Communications and Signal Processing (ICCSP)*, pp. 509-511, 2011.

2. Rahul Shrestha and Roy Paily, "Design and Data Width Requirement for Fixed Point Turbo Decoders Based on Modified MAP algorithm," *IEEE International Conference on Signal Processing and Communications (SPCOM)*, pp. 1-5, 2012.

3. Rahul Shrestha and Roy Paily, "Design and Implementation of a High Speed MAP Decoder Architecture for Turbo Decoding," *26th IEEE International Conference on VLSI Design and the 12th International Conference on Embedded Systems (VLSID)*, pp. 86-91, 2013.

4. Rahul Shrestha and Roy Paily, "A Novel State Metric Normalization Technique for High-Throughput Maximum-a-Posteriori-Probability Decoder,"
*IEEE International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 903-907, 2013.

5. Rahul Shrestha and Roy Paily, "System Level Hardware Testing of a High Speed MAP Decoder Implemented on FPGA," *IEEE International Conference on Signal Processing, Computing and Control (ISPCC)*, pp. 1-6, 2013.

6. Vijaya Kumar K, Rahul Shrestha and Roy Paily, "Design and Implementation of Multi-Rate LDPC Decoder for IEEE 802.16e Wireless Standard," *IEEE International Conference on Green Computing, Communication and Electrical Engineering (ICGCCEE)*, (Accepted in February-2014).

## Award

- Winner of Design Contest on "*27th International Conference on VLSI Design and the 13th International Conference on Embedded Systems*", held at IIT Bombay, January, 2014.
*Design Paper*: "Hardware Implementation and Testing of LMAPP Decoder for High-Throughput Applications".

# Curriculum Vitae of Author

In the year 2004, Rahul Shrestha joined B. M. S. College of Engineering affiliated under Visvesvaraya Technological University. He has been awarded with the Bachelor of Engineering degree in Telecommunication Engineering from this university in the year 2008. He joined Indian Institute of Technology Guwahati for PhD program in 2009 under the supervision of Prof. Roy Paily in the Department of Electronics and Electrical Engineering. His research interests include VLSI design and implementation of high-speed digital architectures for wireless communication applications. Specifically, he has been working with channel codes from algorithmic as well as architectural aspects.