Domain Specific High Performance Reconfigurable Architecture for a Communication Platform

Imran Ahmed

Thesis submitted for the degree of Philosophy.
The University of Edinburgh.
May 2007

DECLARATION OF ORIGINALITY

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

Imran Ahmed

ACKNOWLEDGEMENT

Praise be to GOD ALMIGHTY, my creator and sustainer, who made me capable enough to complete this thesis.

I am also thankful to:

My supervisor, Prof. Tughrul Arslan for his knowledge, expert guidance and patience whilst assisting me through out my research. His technical acumen, precise suggestions and timely discussions are whole heartily appreciated.

I wish to express my indebtness to Dr. Ahmet Erdogen for his helpful suggestions and lively discussions during the research work. I am grateful to Dr. Sami Khawam for his technical guidance in reconfigurable area.

I would also like to thank all the staff at Institute for System Level Integration for providing me with their support and excellent research facilities.

My whole family, especially my wife Rukia and my daughter Sara in whose presence I always found the life in its best colours. Thanks to my nephew Changez and niece Isra for their prayers.

Very special thanks to my parents and their prayers which have supported me throughout my life. This thesis is dedicated to my wife for her untiring love and support.

PUBLICATIONS

1. I Ahmed, T Arslan, "Reconfiguration requirement for convolutional forward error correction decoding for 3G and Beyond", journal submitted to IEEE VLSI transaction.

2. I Ahmed, T Arslan, "A reconfigurable viterbi traceback for implementation on Turbo Decoding Array", in IEEE International SOC Conference, 2006. Publication date: Sept. 2006 Page(s) 107-108.

3. I Ahmed, T Arslan, "A reconfigurable viterbi decoder for a communication platform", in IEEE International FPL Conference, 2006. Publication date: Aug 2006 on pages 1-6.

4. I Ahmed, T Arslan, "VLSI Design of Multi Standard Turbo Decoder for 3G and Beyond", in 12th International IEEE ASP-DAC Conference Jan 23-26, 2007.

5. I Ahmed, T Arslan, "A Low Energy VLSI Design of Random Block Interleaver for 3GPP Turbo Decoding", in IEEE international Symposium on circuits and systems ISCAS 2006 publication date 21-24 May 2006.

6. I. Ahmed, T. Arslan, "Improved Memory Strategy for Log Map turbo decoders," in SOC Conference, 2005. Proceedings IEEE international pages 103-104, Sept 25-28 2005.

7. I Ahmed, T Arslan, "Efficient implementation of Mobile Video Computations on Domain Specific Reconfigurable Arrays" in Proceedings IEEE Design, Automation and Test in Europe (DATE) conference in 2004. Publication date: 16-20 Feb. 2004 on pages1833.

8. I Ahmed, T Arslan, "Video transmission through domain specific reconfigurable architectures over short distance wireless medium utilizing Bluetooth IEEE 802.15.1/spl trade/standard" in Proceedings IEEE international SOC Conference 2004, publication date: 12-15 Sep 2004 on pages 7-10.

9. I Ahmed, T Arslan et al "Domain Specific Reconfigurable Architecture of Turbo Decoder Optimized for Short Distance Wireless Communication", in Proceedings 19th IEEE International Parallel and Distributed Processing 2005 publication date: 04-08 April 2005 on pages 166b.

ABSTRACT

Reconfiguration in an Integrated Circuit (IC) design has become increasingly important in the recent years. Some of the driving factors behind this trend are reduction in transistor size, ever changing standards, very high IC mask costs and short time to market. The programmable hardware design however suffers from performance degradation due to the added flexibility contrary to the end user demand for very high speed and low power electronics. Domain specific reconfigurable architectures provide a powerful solution to the problem by carefully tailoring the domain of the reconfiguration for the increased performance. This research work focussed on investigating such low power reconfigurable VLSI architectures for forward error correction (FEC) to be deployed in a unified communication platform. The viterbi and turbo decoding are very well known techniques for FEC decoding and are essential components in many current and up coming standards such as WCDMA, WLAN, GSM, CDMA2000, ADSL and 3GPP. This thesis presents a reconfigurable unified implementation with a unified state machine control for combined turbo-viterbi decoder array. The amount of flexibility in the reconfigurable design is carefully tailored to meet the performance constraints imposed by these standards. Work on reconfigurable viterbi decoder provided the new novel reconfigurable trace back methodology, new segmentation and memory management techniques along with an open trellis structure that can support multiple standards. The work on reconfigurable turbo array generated novel implementation techniques for low power input metrics management and reconfiguration, low power branch metrics generation, a new matrix normalization scheme and a completely flexible open trellis low power reconfigurable design. Turbo decoder design is combined with a novel low power implementation methodology for 3GPP internal interleaver. The interleaver implementation gives significant reduction in storage requirement for interleaved patterns and hence much improved power performance.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS AND ABBREVIATIONS

ACS	Add compare select
ADC	Analog to digital converter
ADSL	Asymmetric digital subscriber line
ALM	Adaptive logic module
ASIC	Application specific integrated circuit
CLB	Configurable logic blocks
CDMA	Code division multiple access
CMOS	Complementary metal oxide semiconductor technology
CBox	Connection box
DCT	Discrete cosine transform
DSP	Digital signal processing
DP	Data ports
FEC	Forward error correction
FFT	Fast Fourier transform
FIR	Finite impulse response filter
FPGA	Field programmable gate array
FP	Forward processor
GSM	Global system for mobile communications
FSM	Finite state machine
FSM RAM	Forward state metrics random access memory
I/O	Input-output
IFU	Interconnected functional units
LAB	Logic Array Block
LB	Logic block
LC	Logic cell
LE	Logic element
LLR	Loglikelihood ratio
LSB	Least significant bit
LUT	Look-up-table
MAC	Multiply accumulate
MIMD	Multiple instruction multiple data
MSB	Most significant bit
OFDM	Orthogonal frequency division multiplexing
PGAs	Programmable gate arrays
RAM	Random access memory
RP	Reverse processor
SBOX	Switch box
SRAM	Static random access memory
RD	Reconfigurable design
ROM	Read only memory
RSC	Recursive systematic convolutional
RTL	Register transfer language
SAIF	Switching activity interchange format
SDF	Standard delay format
SIMD	Single instruction multiple data

SISO	Soft input soft output
SNR	Signal to noise ratio
SOC	System on chip
VLIW	Very large instruction word
VLSI	Very large scale integration
WCDMA	Wide band carrier division multiple access
WLAN	Wireless local area network
WL	Window length
WiMAX	Worldwide interoperability for microwave access
WRT	with respect to

Chapter 1

INTRODUCTION

1.1 MOTIVATION

VLSI design engineers have to balance many conflicting requirements for example; the design should not only be energy and area efficient but also flexible and high speed. Current devices are expected to combine a large amount of functionality together for example, a mobile phone now has 3G, WLAN, personal digital assistant, camera and many other services all integrated in one device. New standards like WiMAX (IEEE 802.16) and Wi-Fi (IEEE 802.11) allow VOIP and data services at a metropolitan scale with coverage over an area of several miles. This can potentially allow the user to remain connected for communication within an entire metropolitan area using VOIP with the scope of handoff to WCDMA Cell, satellite system or another WiMAX network in other areas.

With all the advances in fabrication technologies ASIC mask cost still remains far above the ground and the time to market (TTM) is shrinking worst than ever before. For example the 3G cellular technologies are already looking obsolete well before establishing itself in the market in a serious way. The non-recurring engineering (NRE) costs impel the designer to think of unorthodox ASIC solutions. Adaptation to ever evolving standards and the constraints that are imposed by computationally intensive applications as in wireless communications demand a clever business model. However from VLSI implementation point of view, finding the optimal reconfigurable architecture for a given problem is very difficult as increasing the level of flexibility degrades the area, power and speed constraints adversely affecting the performance. The contradictory requirement of high speed and flexibility combined with low area and energy can not be satisfied by conventional instruction set processors and non flexible ASICs. Reconfigurable

ASIC (hardware) therefore provides an interesting implementation option. FPGAs implementations cost 10X-100X times in area and speed as compared to ASIC [ALB94]. However the granularity of the reconfigurable blocks and interconnects flexibility is required to be adjusted to meet all the performance constraints imposed by the application.

Therefore the goal of the current research is to distil flexibility such that it meets the required power-area budgets for a desired level of performance. The thesis is motivated by the desire to reduce the power, area and improve the speed while maintaining the flexibility in the domain of convolution based forward error correction.

1.2 CONTRIBUTION

The main contribution of this work is a reusable architecture that can be exploited to implement domain-specific, programmable processors for convolution based forward error correcting algorithms. The work has also produced an architecture template that relies on a heterogeneous network of processing elements, optimized for a given domain of algorithms that can be reconfigured at run time. To verify the effectiveness of this architecture, FEC reconfigurable processing elements were designed, synthesized, and evaluated. Measured results which are presented in subsequent chapters demonstrate the effectiveness of this architecture.

The single most valuable contribution of this research is the unified turbo-viterbi decoder array that can be used for multiple standards [AHM07] Construction of such array requires identification of areas where flexibility should be introduced in Viterbi and Turbo decoder blocks in order to make the overall VLSI design achieve the performance constraints as imposed by different standards shown in table 1 below:

Standard	**Codes (Cyclic Redundancy Code-crc, Convolutional – conv & turbo)**	**Rates**	**States**	**Block size**	**Throughput**
GSM	crc + conv	½	16	33-876	12Kbps
IS-54	crc + conv	½	32	1-512	9.6kbps
IS-95	Con	½,1/3	64	1-512	19.2kbps
PDC	crc + conv	9/17	16	1-512	14.4kbps
UMTS	con Turbo	½, 1/3 ½,1/3	256 8	1-504 40-5114	32kbps 2Mbps
CDMA2000	con Turbo	½-1/6 ½-1/5	256 8	1-744 378-20736	38kbps 2Mbps
EDGE	Con	½,1/3	64	39-870	5-62kbps
W-CDMA (Japan)	crc+con crc+turbo	1/3 1/3	256 4	1-504, 40-5114	384 kbps 2 Mbps
IEEE802.11	Con	½ ¾,	64 or 256	1-4095	6-54mbps
IEEE802.16	con turbo	7/8-1/2 ¾-1/2	64 8	1-2040 1-648	24 Mbps 24 Mbps

Table 1.1. Convolutional decoder in various standards

Our work on reconfigurable viterbi decoder produced a novel implementation scheme for reconfigurable trace back processing and a new memory management and segmentation technique for multiple standards [AHM06]. With the reconfigurable aspects of viterbi decoder design a new and novel dynamic reconfiguration switching methodology is proposed that allows a very fast context switch between different standards [AHM106]. The work also proposed and implemented a reconfigurable state machine control and open trellis reconfigurable architecture for viterbi decoding.

The work in this thesis also identified the commonality of algorithm between viterbi and turbo and shown implementation level details for choice of common hardware blocks for a unified array. The novel implementation also demonstrates commonality of control features by using a single finite state machine for both Viterbi and Turbo decoding. This avoids the use of microprocessor based control

for these arrays. A novel turbo decoding VLSI implementation suitable for the underlying reconfigurable system [AHM05, AHM205] is also proposed. The work on reconfigurable turbo decoder showed an efficient low power input memory management and branch metric calculation scheme. A new open trellis structure for reconfigurable turbo decoding was implemented and the cost of reconfiguration is measured to justify the design decisions.

A new approach for implementing a complex S random interleaver as defined in 3GPP specification [3GPP99] is also presented. Due to the complexity of the interleaver algorithm, earlier implementations used a straight forward approach of storing the entire interleaved addresses in SRAM [MAS99]. This results in interleaver memories consuming major portion of the area and power. The novel implementation technique proposed in the thesis overcomes this bottleneck and provides a new technique for hardware pruning [AHM105].

Experiments have also been performed on a domain specific reconfigurable methodology that automatically connects the domain specific blocks through a reconfigurable interconnect. [AHM104, AHM204].

1.3 STRUCTURE

The structure of this thesis is as follows:

- Chapter 2 presents review of the research work in the area of reconfigurable techniques both by commercial and academic research.
- Chapter 3 describes the reconfigurable methodologies used by various domain specific architectures.
- Chapter 4 presents an overview of turbo decoding and explains the reconfigurable VLSI design detailing each individual component.
- Chapter 5 describes the viterbi decoding along with reconfigurable VLSI design, the trace back methodology, memory management for multiple standards and the role of each viterbi decoding component in the unified array.

- Chapter 6 presents the low power interleaver solution to 3GPP interleaving algorithm. Chapter also describes the hardware technique for bit pruning.
- Chapter 7 gives the summary and the conclusions drawn from the work.

1.4 SUMMARY

Domain specific reconfigurable design is a crucial research area and its benefits have been used to generate high performance reconfigurable design for a communication baseband forward error correction scheme. Multi standard reconfigurable baseband data paths are very attractive for all portable devices in the context of current and future standards. This thesis presents techniques to introduce flexibility with reduced power consumption in the convolution forward error correction blocks to be used in a common communication platform. The next chapter describes the existing architectures in literature and the performance improvement techniques used in domain specific blocks.

Chapter 2

EVOLUTION OF RECONFIGURABLE ARCHITECTURES IN CONVENTIONAL FPGAs

2.1 INTRODUCTION

This chapter summarizes the different implementations of turbo and viterbi decoders reported in the literature. A case is presented for adopting a reconfigurable approach for these decoders. To solve the reconfiguration problem the chapter also describes reconfiguration techniques used in existing conventional FPGA architectures. Reconfigurable architectures have evolved from FPGAs and currently, there are a large class of FPGAs available commercially. Altera and Xilinx are the major contributors in commercial FPGA design and therefore have the biggest market share. The chapter explains the evolution in commercial reconfigurable logic block and routing architectures. Examples are quoted to trace these improvements with architectures present in previous and current generations of commercial reconfigurable devices.

2.2 STATE OF THE ART IN VITERBI DECODER IMPLEMENTATIONS

In Viterbi decoding there are two main techniques used for decoding the bits: the register exchange (RE) and the traceback (TB) [WIC95]. The RE technique is usually used for trellises with only a small number of states, whereas the TB technique is used for trellises with a large number of states. There are several high performance architectures reported in literature using the traceback technique. In [LIN89] a layered approach that combines the stages of the trellis into one stage has been proposed, and further developed by using radix-4 architectures in [BLA92]. The implementation in

[BLA92] is for fixed 32 state (K=6) decoder achieves throughputs of 70Mb/s for a radix 4 iteration in 1.2 μm CMOS process. However, this parallel implementation consumes 1.8W of power which makes it less useful for mobile platforms and because of its fixed architecture is not a good choice in reconfigurable scenarios. A similar radix 4 implementation in [RAB95] achieves a higher throughput of 210 Mb/sec but at the cost of higher power consumption of 3W. A similar radix 4, 32 state (fixed) implementation in 130nm process in [BRU04] achieves a through put of 2.8 Gb/s with a power consumption of 2.23W.

The error correcting capability of an error correcting code is proportional to the constraint length (states) of decoder [HEL88]. Therefore, the new standards like 3GPP [3GPP99] define Viterbi decoders with large constraint lengths for example, there are 256 states in 3GPP decoder. The above mentioned decoders therefore, also have a reduced error correction capability in addition to the higher power consumption.

The latency and through put issues with block based designs [LIN89] have been addressed and improved in sliding block (window) architecture in [BLA97] with a four state decoder. Bit-serial approaches and operation reformulations have been proposed in [BLA97], [TSU99]. The minimum transition scheme in [HWA96] attempts to reduce the paths being traced back and in [LIN00] the speed of traceback was increased by saving the decisions in a permutation network. Another algebraic solution for low power high speed decoding have been proposed in [FET91] by using a semi-ring topology. There are also a few architectures that attempt to combine the traceback and register exchange methodologies together [JUN96], [BLA93].

All of the above mentioned design approaches were developed for low constraint length (K=3 to K=7). The decoder in [CHA00] is designed for CDMA standard with 256 states and have the maximum achievable throughput of 2 Mb/sec. In addition to the limited throughput the decoder also has an input/output memory architecture which is fixed only for CDMA decoder. The decoder is therefore suitable for a single standard but can not be used in multiple standards with its fixed trellis structure and

RAM design. Another CDMA implementation [KAN98] uses state serial approach in which few processing elements are shared for 256 states. This approach is in contrast to the previous high speed state parallel approaches in which every state at a given stage in the trellis corresponds to one processing unit. The architecture in [KAN98] is very low power however can only support voice traffic with data rates of 14.4 Kb/sec. Many other state-sequential architectures have been proposed [SHU93], [FRE86], [GUL88], [FEY93], [CHA89], [CHA92], [DAN95], [BLA92], but most have not had their efficiency validated in the VLSI hardware domain.

It has been shown in [KAN98] that viterbi decoder account for more than one-third of power consumption in CDMA mobile terminals. The importance of power consumption is even more critical in a multiple terminal receiver in the context of software defined radio. In convolutional decoder implementation on Field Programmable Gate Arrays (FPGAs) power consumption is a major concern compared to ASICs and other custom chips. FPGAs have long routing tracks with significant parastic capacitance, and dissipate a significant amount of power at high frequencies. Previous work has presented point solution for power consumption in FPGAs [GEO99], [KUM02], [RAB01]. Numerous CAD algorithms for FPGAs are also proposed that focus primarily on reducing switching acitivity to achieve lower power [WAN98], [MAK01], [ROY99], [SIN02], [WOL03]. Other Implementations of Viterbi algorithm on FPGAs [ZHU07], [HAO06], [ABD06], [TUN06], [LUC06], [IRF05], [PET05], [ANG05], [QIN04], [REV04], [SHA04], [LIN04], [ZHU03], [HEN02], [FAB01], [PAN99], [JAN97], [YEH96], [WAN93], [KEL93] have not had their efficiency validated in power domains.

2.3 STATE OF THE ART IN TURBO DECODING

Turbo code with its excellent error correcting performance has revolutionized the communication engineering. After the successful revelation in the year 1993 [BER93], turbo code has been praised and crowned widely. It became one of the core technology for today's cutting edge products in industries for example, high density magnetic and

optical storage, wire or landline communication systems (Asymmetric Digital Subscriber Line – ADSL) [ELE04], optical fibre networks and wireless communication [HON01], [MCP99], [MCP02], [SON00]. In wireless communications, the application of turbo code principle can further be categorized under the following sections:

2.3.1 TURBO RECEIVER TECHNIQUES FOR CODED MIMO OFDM SYSTEMS

Turbo coding is also a crucial component in broadband wireless access communications. MIMO transmissions can achieve gains in both the information rate increase due to virtual multiple air-links, and diversity gain. In this area of research the turbo techniques are applied in conjunction with multiple-input-multiple-output (MIMO), space time coding, and orthogonal-frequency division multiplexing (OFDM). In this context and for small block size there are many solutions to error control coding such as orthogonal space-time block codes [TAR99], linear dispersion codes [HAS02], threaded algebraic space-time codes [DAM03] and lattice space-time trellis codes [GAM04]. But for moderate to large block sizes the coding schemes are based on turbo codes with bit-interleaved coded modulation (BICM) [LIU01], [WAN02], [TEN04], [ZEH92] and [IMA97, LAM04]. In his Ph.D. thesis Tujkovic presented a design method for constituent recursive space-time trellis codes and parallel concatenated space-time turbo coded modulation [TUJ03]. This space-time coding framework integrated code concatenation into a random-like space time coding approach [TUJ00].

2.3.2 TURBO CODES FOR DIGITAL VIDEO BROADCASTING

In another segment of research the turbo code concept is applied for digital video broadcasting (DVB) which is European Telecommunications Standards Institute (ETSI) standard for digital television services called as DVB-S [ETS94]. Internet over DVB-S can potentially be a competitor against cable modem and DSL technology with an additionally requirement on uplink for DVB-S. DVB project has adopted

turbo codes for the satellite return channel in its DVB-RCS (Return Channel via Satellite) standard [ETS00]. The DVB-RCS turbo code was optimized for short frame sizes (12 to 216 bytes) and return link support data rates from 144 kbps to 2Mbps. One problem with small frame sizes is that the trellis termination imposes a non negligible reduction in code rate and therefore DVB-RCS uses tail biting [HMA86] circular recursive convolutional encoding [BER99]. These techniques encode in such a way that the ending state matches the starting state in the trellis. DVB-RCS also uses duo binary encoders defined over Gallis Field (4) [JEZ99] which results in reduction in trellis states to half and therefore requires half as much memory. Given the fact that the DVB-RCS is a published standard, there are a few codec manufacturers that provide ASIC or core FPGA solutions such as:

- TC1000 turbo encoder/decoder, from TurboConcept [WEB06]
- S2000 from iCoding [WEB07]
- ECC3110 from 'Efficient Channel Coding Incorportation' [WEB08]

2.3.3 TURBO CODES ON SATELLITE COMMUNICATIONS

Turbo codes were selected as a decoding scheme in CCSDS (Consultative Committee for Space Data Systems) recommendations [CCS03], used worldwide by international space agencies (NASA, ESA, RSA etc.). The reason was the significant improvement in terms of power efficiency assured by turbo codes over the old concatenated schemes of the standard with coding gains larger than 2dBs. The structure of CCDS turbo encoder consists of two equal binary, linear, systematic, recursive convolutional encoders with rate ¼ and 16 states. European Space Agency (ESA) investigated the performance of high rate punctured turbo codes for [CCS03] inclusion. Punctured CCSDS turbo codes are obtained by simply puncturing the output of the CCSDS encoder [ANC00], [ANC01]. With puncturing the performance of rate 3/4 codes is very good for large data frame lengths and is competitive with other solutions like LDPC and serial turbo codes. The performance with higher code rate however, is not

so good because puncturing causes a significant reduction of the minimum distance [GAR01]. Therefore this solution is currently being abandoned in [CCS03].

A survey of some of the most popular commercial developments based on serial and parallel concatenation of turbo codes is attempted below:

2.3.3.1 ITERATIVE CONNECTIONS 'S-TEC™'

Iterative Connections [WEB09] decoders are designed for satellite communications providing performance less than 1dB from channel capacity and BER performance of $1/10^{-10}$ with the option of secure communication mode. The S – TEC™ family of serial concatenated convolutional codes was launched in 2003 and was integrated in the Datum Systems line of modems [WEB10]. Their new version, Premier 5 satellite modem can provide data rates up to 5 Mbit/s in QPSK mode with coding rates ½, ¾ and 7/8 and is the most power efficient satellite modem currently available in market.

2.3.3.2 TRELLISWARE 'FLEXICODES'

TrellisWare Technologies [WEB11] launched serial concatenated mode turbo like codes in early 2004 called FlexiCodes. It consists of 4-state outer rate ½ convolutional code, followed by a single parity check (SPC) code and an inner rate 1 convolutional code and therefore, three elementary encoders combined in a single concatenated mode. The outer decoder produces estimates which, after interleaving, are fed to the SPC decoder; in turn, the SPC output is used by the inner decoder. The feedback loop starts from the inner decoder which passes extrinsic information to the SPC decoder; after interleaving, the bit estimates produces by the SPC decoder are input to the outer decoder. This family of codes performs very close to capacity for a wide range of coding rates and modulation.

2.3.3.3 SMALL WORLD COMMUNICATIONS and iCODING

Small world communications [WEB12] provide turbo decoders in BIT/MCS format for downloading in to Xilinx Virtex and Spartan FPGAs or as a EDIF (VHDL) core.

These high speed decoders have some programmability in the design. The S1000 is a 15 Mbits/s iterative decoder for Xilinx or Altera FPGAs designed by iCoding [WEB13]. A speed up to 45Mbit/s is predicted for ASIC mapping of the design. S4000 is their high speed version that can run at 200 Mbit/s in Virtex II or 100 Mbit/s in Virtex E.

2.3.3.4 STMicroelectronics

STMicroelectronics [WEB14] chipset is compliant with DVB-S and DIRECTV specifications and uses QPSK, 8PSK and 16QAM modulations. Due to the turbo decoding the chip allows 50% increased throughput or a reduction in more than 33% in the dish size. STMicroelectronics is one of the largest suppliers of set-top boxes in US with the STV0499 8PSK turbo codec. DirectTV [WEB15], EchoStar Communications [WEB16] and Voom [WEB17] are the other satellite TV providers taking advantage of the new turbo-like technologies.

2.3.3.5 BROADCOM

Broadcom [WEB18] has two turbo decoding chipsets BCM4500 and BCM3348. BCM4500 is an integrated digital receiver that supports BPSK, QPSK and 8PSK in conjunction with turbo codes for up to 30 Mbaud. BCM3348 is the TurboQAM single chip cable modem using advanced TDMA and synchronous CDMA with MIPS, 200Mhz communication processor. It supports 4/16/32/64/128/256/512/1024 QAM FEC decoding and 10/100 Ethernet MAC and USB interface.

2.3.4 BLOCK TURBO AND TURBO PRODUCT CODES

Block turbo codes [PYN97], [LOD93], [PYN94], [HAG96] form a sub-class of turbo codes. Different than regular turbo codes it is typically formed via linear block codes. Therefore they can be processed with algebraic decoders and have low complexity implementation. They are suitable for higher codes rates (greater than 0.75) and therefore for systems that require high spectral efficiency. The performance

improvement in QPSK modulation is not as good as for 16 QAM [PYN95] and they are less sensitive to input quantization when quantized to only 4 bits compared to convolutional based codes. Results from [PYN94] and [PYN95] conclude that for spectral efficiencies greater than 4/bits/s/Hz, block turbo coded QAM systems outperform convolutional coded QAM. A comparison is made in [BEN296] between parallel concatenated block codes (PCBC) using systematic cyclic codes and serial concatenated block codes (SCBC) using Hamming and BCH codes. The SCBCs perform better than PCBCs, but significantly worse than the equivalent structures based on convolutional codes.

Examples of commercial implementation of block turbo codes (for example turbo product code - TPC) can be seen by the decoders developed by companies such as Comtech, Radyne, Paradise, Advantech, iDirect and ViaSat. Very high speed TPC ASIC decoders are available from Comtech AHA Corporation [WEB19]. The family consists of AHA4501 (36 Mbit/s), AHA4522, AHA4540 (155 Mbit/s) and AHA4541(311 Mbit/s) codecs. DMD20 satellite modem is developed by Radyne ComStream [WEB20]. The modem can operate from BPSK to 16QAM upto 20Mbit/sec data rates and offers L-Band interfaces. Paradise datacom [WEB21], P300 series of modems with TPC provides up to eight voice/fax ports, IP Bridge/Router for speeds up to 2Mbit/s. Advantech Microwave Technologies [WEB22] has developed their AMT-70 satellite modem with a range from 8kbit/s to 140 Mbit/s, BPSK to 16 QAM, 70/140 MHz or L-band version with an enhanced TPC option. The iDirect [WEB23] technology also produces TPC based modems based on DVB standard. ViaSat [WEB24] have three products based on turbo-like codes: Linkstar, Surfbeam and WildBlue. Linkstar is a two-way bandwidth-on-demand broadband VSAT system using the DVB standard. Surfbeam and WildBlue are based on satellite-enabled version of the Data Over Cable Service Interface Specifications (DOCSIS1.1) standard.

2.4 VLSI FOR TURBO DECODING

There are number of aspects of turbo decoding that makes their VLSI implementation non-trivial: first of all the algorithms that are used to implement the decoding (for example BCJR algorithm [BAH74]) are of great complexity, coupled with the iterative decoding principle, which makes very difficult the accomplishment of throughput, latency, power constraints as imposed by various standards. Moreover turbo decoders include large RAM memories that need to be organized and managed properly. The best solution for each application can only be selected by carefully exploring the space of design alternatives.

The problem of latency and throughput are traditionally addressed by an approximated version of the original algorithm [BAH74], largely known as sliding window BCJR algorithm [BEN396], [BAR96], [PIE96]. The lowest cost solution is to limit the window overlap to the backward recursion, obtaining the starting metrics for a given window from the last metrics of the previous one. This solution, known as single flow structure introduced in [VIT98]. Another powerful solution is given by DFG [PAR99], applied to various formulations of the sliding window BCJR [DIW95], [SCH99], [WOR00], [MAN03]. DFG methodology can also be applied to the study of parallel BCJR, where more windows are processed in parallel rather than serially [WOR00], [VIG00], [ZHO02]. Another complete and clear study on the many possible alternatives in the SISO internal organization has been published by [MAN03], where expressions for the optimization of decoding delay and metric memories in single flow and parallel SISO is formally derived.

Another approach proposed in [YUF00] is based on the idea of evaluating backward state metrics in the forward direction. The possible advantage by this approach is the elimination of the path metric memory in the BCJR architecture which reduces SISO complexity and energy consumption. Some implementations of this concept are proposed by Prof. T. Arslan in [ATL03] and in [KWA03] reporting improvements between 15% and 35%: however this approach seems to be critical for two main reasons: first of all the inversion of the original reverse metric calculation (and the equations in BCJR corresponding to this operation) poses problems of singularity and

computational complexity; moreover the numerical precision in reverse computation has an heavy impact on the BER performance of the code.

The use of radix-4 computation structures has also been explored [BIC03]: similar to viterbi decoders [BAL92]. In this technique trellis is compressed in time, compressing two trellis steps in a single one; this doubles the throughput for a given clock frequency with respect to radix-2 version. The disadvantage however is that the technique also doubles the edges for each state, so requiring the implementation of radix-4 log-MAP or max-log-MAP units with tree organization [MAS99], [BIC03].

The biggest challenge in implementation of Turbo decoder architecture especially in the domain of wireless communications is the power consumption. Turbo decoder tends to have large power consumption than other decoders [WOR99] because of three main reasons:

- The hardware complexity is larger than for other decoders such as Viterbi decoders.
- The decoding process is iterative and in order to achieve the throughput and decrease the effects of iterations, clock frequency must be kept high.
- Large memories are included in the decoder architecture required as Input/Output buffers needed to support iterations and the interleaver memories.

The most important technique at algorithmic level to reduce the effect of iterations on power consumption is by a concept known as stopping criteria [LEU01], [SCH199], [ZHO99]. In [LEU01] the circuit is shut down after the desired BER performance is achieved. As the energy dissipation tends to grow linearly with iteration, the percentage of power saved with this approach is roughly equal to the average reduction achieved in the number of iterations, which can be as large as 75%. The iteration number can also be controlled by means of decision-aided stop criterion [SCH199]. A threshold can be set (on the basis of target BER) and compared against the output log-likelihood ratios (LLRs). If LLRs magnitudes of all bits in a block are

under the threshold, decisions are not considered reliable enough and new iterations are scheduled and vice versa. This technique gives percentage of energy reduction as high as 50%. The work in [ZHO99] evaluates the number of 1's accumulated from the output of decoded block at each iteration which is a necessary condition for having identical decoded bits from current and previous iteration is that the two accumulated values are equal. This technique is simpler however less precise.

A wide range of trade-offs in turbo decoder architectures have been studied in [SCH99], [SCH01] and several architectural parameters have been introduced, with the aim of finding a storage organization efficient from the energy point of view. It is proved in [SCH01] that optimal choice of these parameters is strongly dependent on the specific turbo code and on the technology models used.

Another technique proposed in [LEU01] called "as slow as possible" algorithm which adapts the supply voltage to the instantaneous workload. The algorithm is based on estimations of the energy and delay associated to the decoding of a given data block. Data flow transformations are also applied to reduce both the storage of state metrics and number of memory transfers. While this idea does not provide any area benefit, it is quite effective in reducing the energy consumption. In [MAN03] the delay and energy benefits deriving from the adoption of some degree of parallelism in the decoding architecture are shown for a particular case of double flow structure. This architecture can be viewed as a particular case of more general parametric description based on the DFG and shows 25% reduction in dissipated energy with respect to single flow architecture.

In [BIC03] a 180nm CMOS turbo decoder for 3GPP-HSDPA (High Speed Downlink Packet Access) is presented. The power dissipation is reduced combining architectural techniques, such as clamping of extrinsic information to save memory, and iteration control by means of efficient stopping criteria. In [BOG03] power performance of 1.45nJ/bit/iteration is achieved by means of several algorithm and architecture level

optimization: reported data refer to a UMTS-like decoder, implemented in 180nm CMOS technology and supporting 75 Mbps.

Several commercial implementations have been developed recently for decoders compliant with the 3GPP standard, both in the form of proprietary cores and soft cores. Some examples of available IP cores are given in [WEB01], [WEB02], [WEB03], [WEB04], [WEB05]. VLSI design figures such as complexity, power dissipation and clock frequency are sometime available for these hard and soft cores. [WEB02] is a turbo decoder IP core available from Xilinx and designed for 3GPP [3GPP99] standard achieves the throughput of 6.5 M bits/sec for a 12 K block size and 11 iterations. [WEB03] is Altera TC1000 DVB compliant block turbo decoder IP with payload bit rate of 4Mbit/sec. In [WEB05] a full 3GPP [3GPP99] standard turbo decoder is implemented as a drop in module for Virtex – E and Virtex II FPGAs. This iCoding S3000 module [WEB05] achieves data rates of 7 Mbits/s. With these IPs as well, there are no power figures provided.

Implementations of turbo decoder on FPGAs have the advantage of flexibility (through device reconfiguration) and the availability of large amount of internal resources that can be exploited to achieve higher processing capabilities. The programming process for an FPGA consists in uploading of a bit stream containing the information for the internal configuration of logic blocks, interconnects and memories. For most devices the configuration process takes a long time and implies that the hardware previously mapped to the FPGA is stopped; these two main difficulties are overcome in recent devices that support partial and dynamic reconfiguration. Another major limitation to the adoption of FPGA platforms for wireless communication comes from the high power dissipation of FPGA devices, both dynamic and static. Inspite of these problems, the very short development time connected with the use of FPGAs, there is an impressive growth in the development of FPGA based system. A number of implementations addressed various turbo decoders achieving medium throughput figures [SHA03], [XIA02], [STE01], [WEB05]. No power figures were quoted for implementations in [SHA03], [XIA02], [STE01], [WEB05].

The review done in above sections presents various implementation options for turbo decoding in academia and commercial. Due to parallel architecture inherent in turbo decoding high throughput architectures are possible exploiting the parallelism available in FPGAs but these implementations suffer from very high power consumption. Block turbo codes (and TPC) codec, because of its ease of implementation offer an attractive high speed low power implementation option which is exploited in variety of deep space application products. TPC codec has worst BER performance compared to convolutional turbo codec and are not used in 2^{nd} and 3^{rd} generation wireless mobile standards.

In the current communication environments where different standard coexist, some new specifications for decoders in the modern platform emerge. Processing speed latency, energy consumption and cost constraints do not remain the only constraints on the design. A rapidly increasing role is now placed by two additional features, namely scalability and flexibility. Scalability is the capability of the platform to adapt to different choices for system level parameters, such as for example the mother convolutional codes or the size of the processed block; throughput, latency and power consumption typically change with these parameters and additional implementation complexity is paid to support scalability, however the decoder architecture is not changed or reconfigured when adapting to a different set of parameters. On the other hand the term flexibility is used to indicate the possibility to update an implementation platform in order to support a completely different decoder that does not simply require a change in some parameters: as an example a turbo decoder that can be reconfigured to perform Viterbi decoding.

Aim of current research is to investigate scalable and flexible implementation aspects of convolutional decoding in the context of a software defined radio (SDR). A SDR device uses reconfigurable hardware that may be programmed over-the-air to operate under different wireless standards. For example, an SDR transceiver in a wireless laptop computer or PDA may be configured by different software loads to operate in

an IEEE-802.11x wireless network, a CDMA2000 wireless network, an OFDM/OFDMA wireless network, a GSM wireless network, or other types of networks. Many of these wireless standards require the use of turbo decoders or other decoders that are based on maximum a-posteriori probability (MAP) decoders. However, conventional decoders have significant drawbacks with respect to SDR applications. Turbo decoders and other types of decoders are optimized for decoding under only one or two specific standards. Conventional designs use different MAP decoders to support each standard separately. For example, a MAP decoder calculates three values: alpha (.alpha.), beta (.beta.), and lambda (.gamma.). Normally, three distinct hardware blocks are used to calculate these values. This increases power consumption and uses a large amount of die space. If an SDR device is required to support many wireless standards, more than one decoder must be implemented in the SDR device. This leads to a complex transceiver design that makes inefficient use of chip space and has high power dissipation. This also increases development cost and time-to-market (TTM). Additionally, some of the newer wireless standards operate at relatively high data rates (e.g., WiBro, HSPDA, and the like). A decoder that is optimized in terms of speed and power consumption for a low data rate standard is unlikely to be optimized in terms of speed and power consumption for a high data rate standard, and vice versa. Thus, conventional decoder designs are not suitable for use in SDR applications.

Next sections describe the reconfigurable architecture and techniques in current reconfigurable devices with an aim to assess the suitability of these techniques for our reconfigurable research.

2.5 RECONFIGURATION IN FPGAs

The origins of reconfigurable computing date back to 1960s by the concepts proposed by Gerald Estrin [EST]. The first FPGA (field programmable gate array) was introduced in [CAR86]. The reconfigurable design (RD) world has seen many changes and evolved both in hardware and software. Some of these improvements are discussed as under:

2.5.1 EVOLUTION IN LOGIC BLOCK ARCHITECTURE

Historically the LBs can be seen to be evolved in the following order:

- LBs build on NAND gates.
- LBs build on mulitplexers.
- SRAM (LUT) based LBs.
- LBs build on clusters of LUTs.
- Mixture of LUTs and application specific coarse grained LBs for example multipliers and dedicated processors.

The first example of NAND gates based logic blocks is Cross point FPGA from Cross point solutions [MAP92]. It uses a single transistor pair (NAND) in the logic module. The transistors are connected in row and can be isolated by turning off the pair of transistor between the gates. In addition Cross point had RAM logic tile to implement memory or any other logic as LUT. A similar two input NAND block from Plessey FPGAs is shown in figure 2.1 [PLE89]. Latch can be made permanently transparent (using configuration RAM) if it is not required.

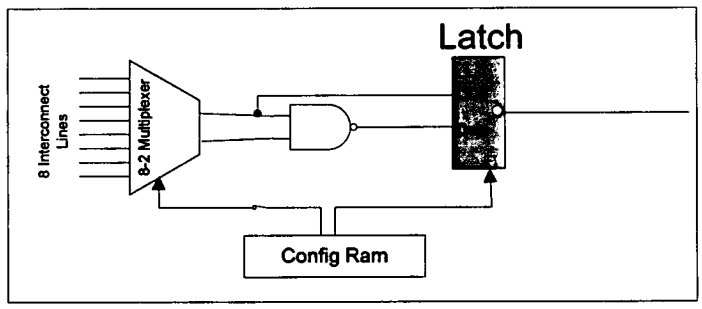

Figure 2.1 Plessey LB

Another example of LB based on NAND gates is the Toshiba FPGA developed in 1991 [MUR91]. This FPGA has two input NAND gate and the input to the NAND gates is provided by either of the multiplexers. The multiplexers are connected to six local lines connected to adjacent cells and three long range connection lines. Latch output can be bypassed and inverted and non inverted outputs can be selected. Shift

registers can be implemented by using the master and slave cells. This is shown in figure 2.2.

The second class of LBs based on multiplexers relied on the ability of the multiplexer to implement different logic functions by connecting each of its inputs to a constant or to a signal.

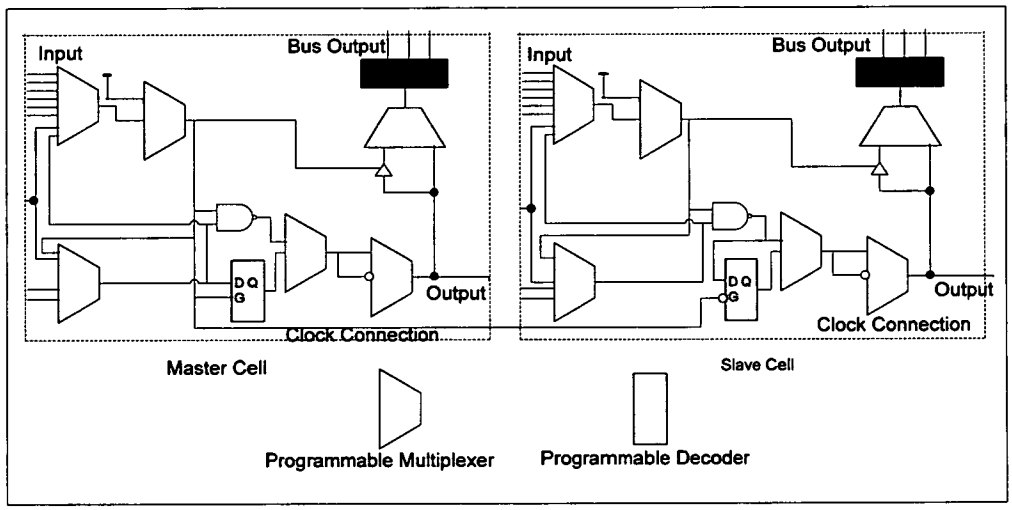

Figure 2.2 Toshiba LB

The Actel [GAR89, AHR90] blocks shown in figure 2.3 can implement 702 and 766 logic functions respectively by connecting the multiplexer inputs to I/O or input signals.

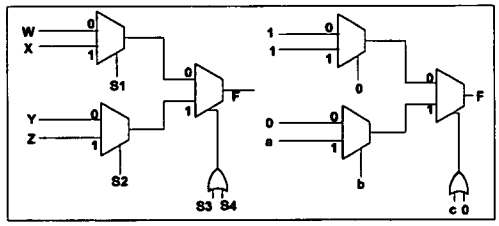

Figure 2.3 Actel LB based on multiplexers

A similar CLB by Quick logic [BIR99] is shown in figure 2.4. In this block alternate input to 'And' gate was inverted which eliminates the need of a separate inverter

circuit. Multiplexer based LBs are disadvantaged due to large number of inputs, demanding more routing resources. However they provide better flexibility for a relatively small number of transistors.

The above set of fine grained CLBs exhibit poor performance in terms of area, power and delay/speed [SAT92]. The work shown in [SAT92] showed LUT based clusters gave best delay performance compared to multiplexer-based gates, NAND gates and the wide input AND-OR gates. One of the conclusions was that the connection delays often exceed the delay of the LB and hence is one of the fundamental limitations on FPGA speed. Increasing the functionality in a LB decreases these blocks in the critical path improving performance.

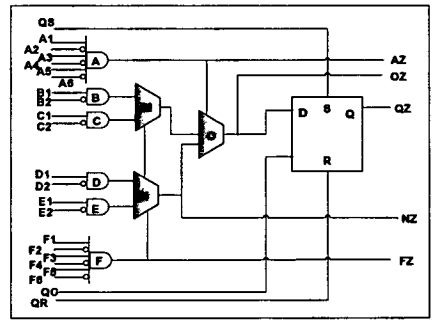

Figure 2.4 Multiplexer based block by quick logic

In the next category of SRAM based LUT, the logic corresponding to K inputs is stored in a 2^K x 1 SRAM block. The address lines of SRAMs function as input and the output of the SRAM provides the value of the logic function.

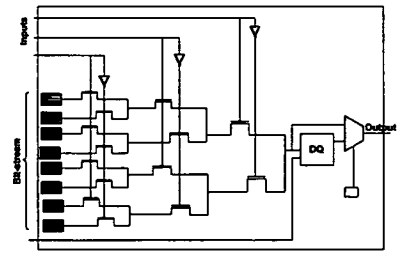

Figure 2.5 A 3 input LUT.

The biggest advantage in this case is the flexibility as a K-input LUT can perform any function of K inputs. The disadvantage is the large size of CLB due to 2^K memory cells. Since the majority of the flexibility provided by LUT is not exploited in logic synthesis, a very large LUT will be underutilized. Figure 2.5 illustrates a three input LUT. The size of LUT determines the delay, area and power characteristics of a reconfigurable design. The work in [ANN1] and [ROS90] showed that LUT size of 4 is the most area efficient in a non clustered context.

After the non clustered approach, the LUTs started appearing in clusters. The main advantage with the clustered approach is that the routing resources are reduced and more functionality is provided within the blocks. It was demonstrated in [KOU91] that using a LUT size of 5 and 6 gave the best delay performance. In [AGR99] it is shown that two – three input LUTs have more advantages in terms of area and speed. [KAP99] suggested that using a heterogeneous mixture of LUT sizes of 2 and 3 was equivalent in area efficiency to LUT size of 4. The same problem was also addressed in [HIL91], [ROS89] and [ROS90]. In a more clustered approach, the recent work by [ELI04] has shown that LUT size of 4-6 and cluster size of 3-10 provides the best area-delay product for an FPGA. The results in the paper are confirmed by majority of the industrial implementations of FPGAs as can be observed by looking at the input size of LUTs in table 2.1.

Device	**Year**	**Logic Block, Cluster size**
Xilinx XC2000	1985	1xLUT-4 Input
Xilinx XC3000	1987	1xLUT-5 Input
Xilinx XC4000	1990	1xLUT-3 Input 2xLUT-4 Input
Xilinx Virtex	1998	4xLUT-4 Input
Xilinx Virtex-II	2000	8xLUT-4 Input
Xilinx Virtex-II Pro	2001	8xLUT-4 Input
Xilinx Virtex-4	2004	8xLUT-4 Input

Altera Flex 8000	1992	8xLUT-4 Input
Altera Flex 10K	1995	8xLUT-4 Input
Altera Apex 20k	1998	10Xlut-4 Input
Altera Apex-II	2001	10Xlut-4 Input
Altera Stratix	2002	10xLUT-4 Input
Altera Stratix II	2004	24xLUT-3 Input, 16xLUT-4 Input

Table 2.1. Cluster size in commercial FPGAs

The first appearance of cluster based LUTs can be seen after Xilinx XC3000 series. In Xilinx 4000 series two different sized (two four inputs LUTs feeding in to 3 input LUT) LUTs were used giving CLB a heterogeneous flavour.

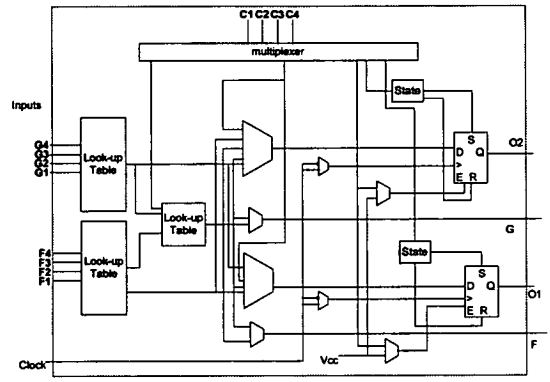

Figure 2.6 Cluster of LUTs in Xilinx 4000

The next series called Xilinx-Virtex provided the first flavour towards high performance by the use of dedicated blocks. It contained dedicated block RAMs every 12 CLB columns. The Xilinx Virtex series can be seen as an example of the last category of the LBs containing cluster of LUTs and several heterogeneous blocks. Similar to Xilinx, Altera reconfigurable devices also have clustered LUTs approach along with several coarse grained heterogeneous blocks. Altera's basic logic block signature can be traced back from its earlier flex 10K device. The details are shown in figure 2.7.

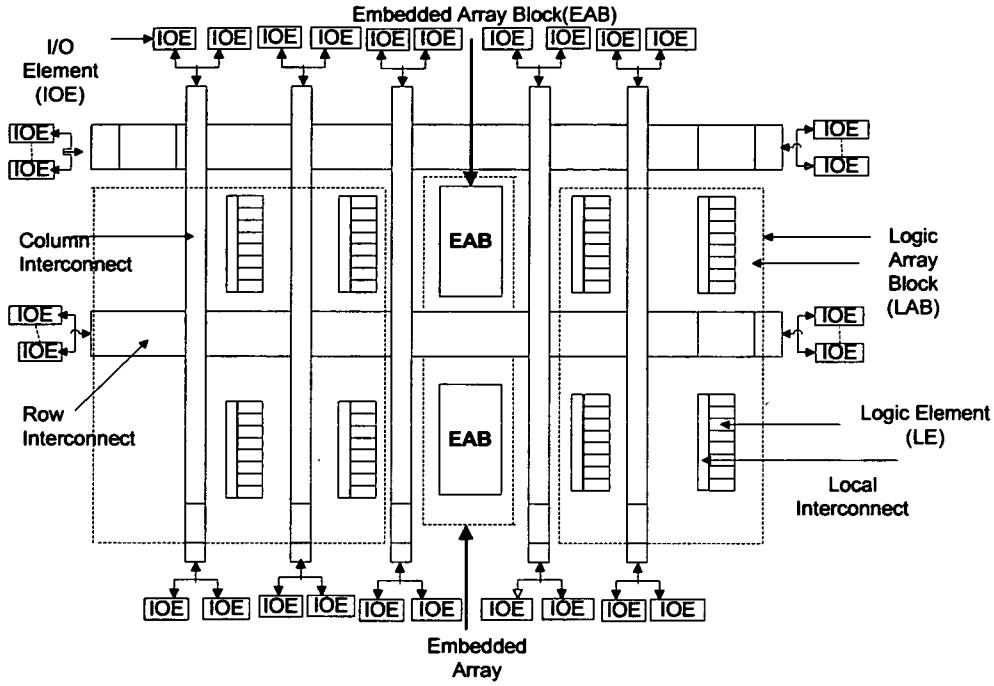

Figure 2.7 Flex 10K device by Altera

The major difference from Xilinx devices is the avoidance of distributed RAMs to gain more predictable timing. The saving is especially higher for bigger size Xilinx distributed RAMs The main logic element is 4 Input LUT but unlike Xilinx, Altera LUT has separate drive capabilities for individual components giving it better utilization as shown in figure 2.8.

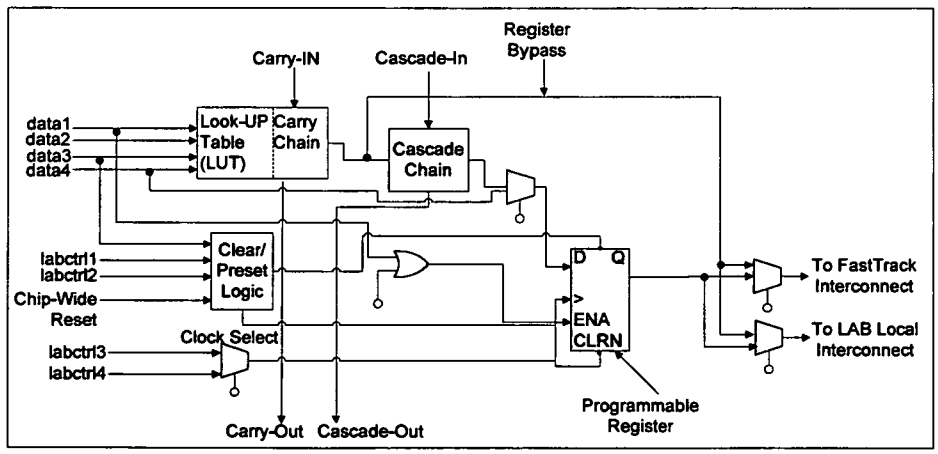

Figure 2.8 Altera Flex 10K logic element

Flex 10K also had faster interconnect topology by utilizing carry-in and cascade-in chains. Its logic array block (LAB) is formed by combining LEs and each LAB contains about 96 usable logic gates. The embedded array consists of a series of EABs (embedded array blocks). The main approach in Stratix follows from its predecessors i.e., packing of multiple independent functional blocks in a single ALM. Altera Stratix II improved on Altera's earlier design by adding more heterogeneous blocks for example, DSP blocks, FIR/IIR filters, FFT functions, DCT and correlators. It has three different classes of memory each defined for a different mode of operation hence moving towards restricting the domains of reconfigurability and gaining on performance. Another important example in this category of LBs is the HardCopy II devices (figure 2.9). HardCopy II device is a fixed non configurable device which removes all configurable resources and replaces them with direct metal connections. Hardcopy II devices consist of an array of HCells manufactured in 90nm process technology which has functionally equivalent architectural features as Stratix II. Only the HCells needed to implement the design are assembled together. The unused area of the HCell logic fabric is powered down.

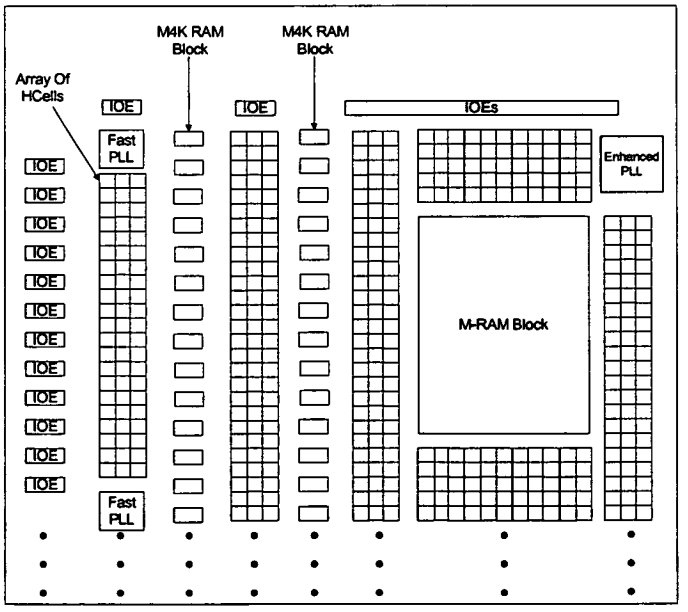

Figure 2.9 HCell placement in HardCopy II

2.5.2 ROUTING IN COMMERCIAL FPGAs

The routing is used in a logic cluster to determine where the inputs come from and where the outputs will go. It also determines how the signal propagates through the logic elements themselves. There are two types of routing available one external to the logic clusters and one internal. We will restrict the discussion to external or global routing as internal routing has already been explained along with CLB/LE discussions in pervious sections. This chapter explains the routing schemes in commercial FPGAs and in chapter 3 the routing architectures are explained for domain specific reconfigurable architectures. In commercial FPGAs generally there are three different switch types used for routing: multiplexers pass transistors and tri-state buffers. There is also some form of fixed routing (non-programmable) which is used for fast carry propagation or fast shift registers. The overall routing architectures can be categorized as island, hierarchal, cellular and row architectures. Note that the current families of the commercial FPGAs have architectures that are more complex than what is described here, but these general categories can still be identified with in the above mentioned FPGAs.

2.5.2.1 XILINX ROUTING ARCHITECTURE

Xilinx uses island-style routing in which the logic structures are placed in two dimensional array surrounded by segmented horizontal and vertical routing channels.

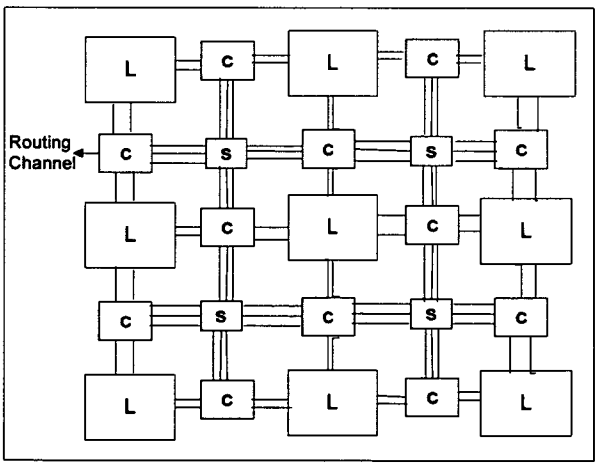

Figure 2.10 Island Style Routing

Each cluster connects to the routing through connection boxes (C-boxes) and C-boxes are connected by Switch boxes (S-boxes). S-boxes also provide change in direction from horizontal and vertical tracks and hence provide means of connecting one segment to another.

Newer Xilinx routing architectures provide segments in several lengths for example long lines, Hex lines, double lines and direct lines.

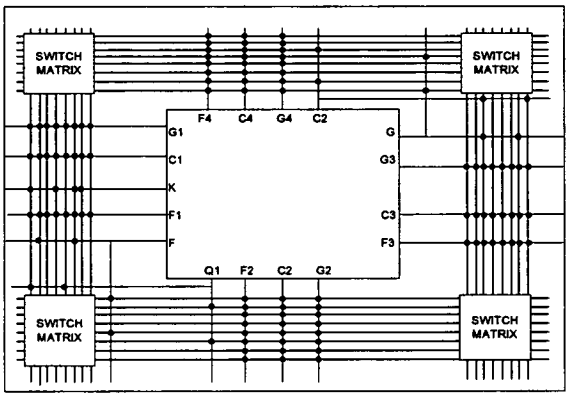

Figure 2.11 single length connections

[HUNG90] has described the interconnect structure of Xilinx XC 4000 series of devices. It consists of single length, double length and long lines. The single-length connects adjacent CLBs via a switch box, where a switch box can connect for example CLB on the left to the top, bottom or right. Single length interconnects are used to connect the signals in localized area. This is shown in the figure 2.11.

Note that the CLB's clock (K) input can be driven from one-half of the adjacent single length lines as shown in figure 2.12. The double length lines runs past two CLBs before entering a switch Matrix and provide connections for intermediate length point-to-point interconnection. Double length lines are grouped in pairs, and the switch boxes are so placed that one member of the pair has switch box nearby alternating CLBs.

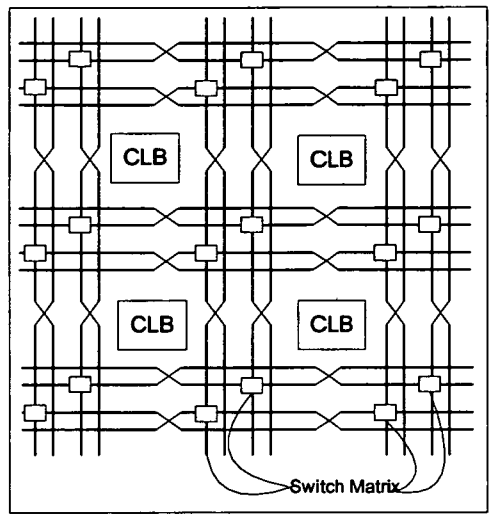

Figure 2.12 Double length connections

The inputs in the CLB can be driven from any adjacent double length line and each CLB output can drive nearby double-length line in both the vertical and horizontal planes. Long lines are a grid of metal interconnect segment that run the entire length or width of the array and are intended for high fan out, time critical signal nets. Xilinx routing architectures are summarized in figure 2.13. Hex lines route signals to every third or sixth CLB in all four directions. Double lines route signals to every first or second CLB. Direct lines connect signal to neighbouring blocks including diagonal neighbours.

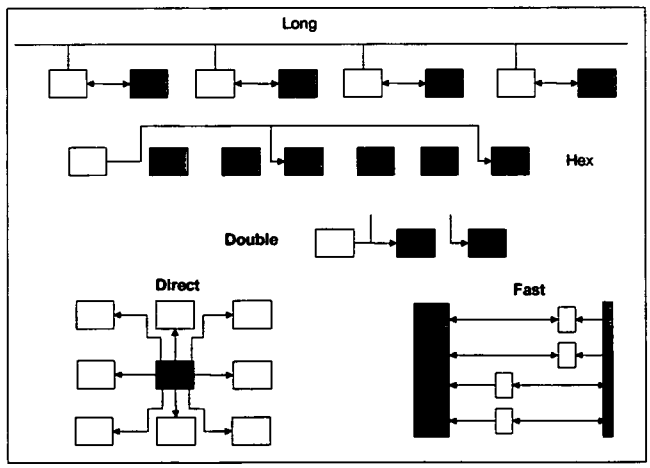

Figure 2.13 Xilinx interconnect topologies

Long lines are a grid of metal interconnect segment that run the entire length or width of the array and are intended for high fan out, time critical signal nets. Xilinx routing architectures are summarized in figure 2.17. Hex lines route signals to every third or sixth CLB in all four directions. Double lines route signals to every first or second CLB. Direct lines connect signal to neighbouring blocks including diagonal neighbours.

2.5.2.2 ROUTING IN ALTERA FPGAs

Altera FPGAs use staggered interconnects topology which demonstrates different levels of routing. It comprises of row and column interconnects that span fixed distances giving routing of different speed and lengths. The software (for example Quartus II compiler) automatically places critical design paths on faster interconnects to improve design performance. For example, Altera (Stratix) routing consists of a local interconnect, direct link, R4, R24, C4 and C16 interconnects. R4 interconnects are used for fast connections in a four LAB region. Figure 2.14 shows R4 interconnect of primary LAB and its left and right neighbours.

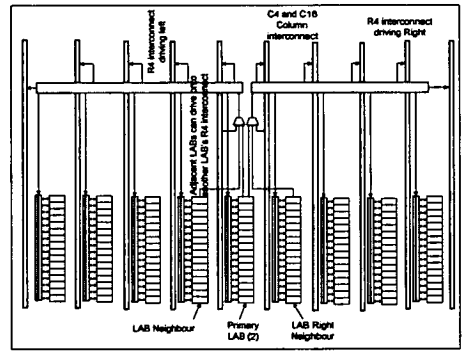

Figure 2.14 R4 interconnect

In this scheme for R4 interconnects that drive to the right, the primary LAB and right neighbour can drive on to the R4 interconnect.

For R4 interconnects that drive to the left, the primary LAB and its left neighbour can drive on to the interconnect. R4 interconnects can drive other R4, R24, C4 and C16 interconnects to extend the range. R24 provide long row connections as it spans 24

LABs and it drives row (R4, R24) interconnects every fourth LAB. It can also drive C4 and C16 interconnects. Similarly column interconnects provide connections vertically where C4 interconnects spans 4 LABs or memory blocks (M512, M4K) and C16 spans 16 LABs. C16 is the fastest resource for column connection between LABs, tri matrix memory and DSP blocks. This is shown in figure 2.15.

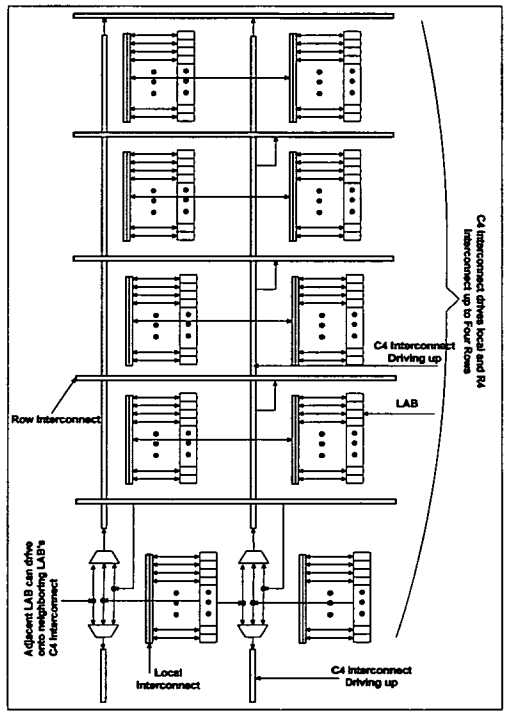

Figure 2.15 C4 interconnect

2.4.3 CELLULAR ROUTING

There is another kind of routing called the cellular routing in which the logic cells themselves are designed so they could be used as a part of the routing network between logic elements. The logic clusters for cellular routing are usually very fine grained and have single logic element in them. However the delays in the combinational paths get significant for circuits requiring longer routes in routing. An example structure is Cell Matrix [DUR01].

2.5.3 ROW ROUTING

A Row type routing is usually found in one-time programmable FPGAs for example many Actel Antifuse FPGAs ACT-I which consisted of vertical routing and some long wires.

2.6 SUMMARY

This chapter provided an overview of academic and commercial implementation of Viterbi and Turbo decoders. Some decoder algorithms are less complex to implement and achieve high data rates and low power consumption. Other algorithms provide excellent error correction performance but at the cost of high power consumption and lower speed. Each wireless standards define these algorithms as per the error correction capability desired. We have identified the state of the art in these implementations such that the results can be used to design a unified decoder structure that can target a large variety of communication standards. The chapter also introduced the evolution of reconfigurable architecture elements citing examples in commercial programmable devices. The configurable logic elements are covered in detail and an overview of different routing architectures is also provided. It has been shown that reconfigurable architectures tend to improve performance by restricting the application domain and by techniques that reduce the cost of routing. The next chapter covers another class of reconfiguration called domain specific reconfigurable architectures which have reconfiguration in even more restricted sense however giving better performance in a particular application domain.

Chapter 3

RECONFIGURATION TECHNIQUES AND ARCHITECTURES FOR DOMAIN SPECIFIC PLATFORMS

3.1 INTRODUCTION

The approach for domain specific reconfigurable design is based on the observation that algorithms within a given domain of signal processing have in common a set of dominant kernels that are responsible for a large fraction of total execution time and energy. By executing these dominant kernels on dedicated, optimized processing elements significant energy savings can potentially be achieved. This yields processing elements that are domain-specific to a particular problem. The domain specific reconfigurable architectures can be classified on the basis of several criterion for example, reconfiguration model (static or dynamic), arrangement of logic blocks (cross bar, mesh or linear arrays), granularity (data path width), computation model (VLIW, MIMD, SIMD or single processor) and type of application domain (DSP, General Purpose, Video etc). One of the most successful applications for reconfigurable computing is in the field of real-time digital signal processing [TOD05]. Wireless is in many cases the driver for DSP processing on reconfigurable logic as a wireless baseband receiver consists of many DSP components. The authors of [VIL98], [HAR01], [ABWEB], [HAR21], [COM99] and [TOD05] presented a comprehensive survey of available reconfigurable computing platforms in the academic and commercial. In this section of dissertation the focus is only on architectures (or techniques) in literature that can directly or indirectly be useful in the baseband processing of a reconfigurable communication receiver. This chapter describes several suitable reconfigurable computation models citing examples from the literature and some existing reconfigurable architectures in the wireless domain.

3.2 DOMAIN SPECIFIC RECONFIGURABLE CORES FOR WIRELESS COMMUNICATION

Based upon all of the published work, there are very few architectures designed specifically for wireless communications and some of the information on these architectures is either incomplete or not disclosed because of being proprietary items. On the basis of published information the most known and relevant architectures are presented below with an aim to cite the mappings for convolutional forward error correction decoding.

3.2.1 CHAMELEON SYSTEMS - MONTIUM RECONFIGURABLE ARCHITECTURE

Recore Systems [WEB25] Montium architecture is an extension of Chameleon Systems SoC template developed earlier at university of Twente, Netherlands [CHWEB], [SMI04], [PAU04]. Montium predecessor Chameleon had CS2000 family of multi-protocol multi-application reconfigurable platforms designed for telecommunication and data communication. Chameleon CS2000 can be considered as a general solution for the wireless application. However, it was not meant to be an implementation solution for the baseband processing of the handheld terminals. The CS2000 family's very sophisticated and in-homogenous array makes an IP-based mapping difficult. The CS2000 family incorporates a 32-bit RISC core as a host, licensed from ARC UK, with full memory controller, PCI controller and a reconfigurable array. The reconfigurable array sizes come in 6, 9, and 12 tiles. The tile consists of seven 32-bit processing elements (each containing an 8 word instruction memory), four local memories of 128(deep) x 32(wide) bits, control logic and two 16x24-bit multipliers. Every three tiles are grouped as a slice which can be configured independently and also includes 8k Bytes of local memory. The 32 bit processing element can also operate in SIMD fashion on four 8 bit data streams or two 16 bit data streams. It is programmed with eight user-definable instructions stored in the instruction memory. The instruction configures the input output routing, shifting, masking, register enables, memory read and write, flag generation and the operation of

processing unit. Dynamic configuration is supported and can be accomplished in one cycle. Their CS1200 family aims at initial markets in communication infrastructure with application areas as wireless base stations, fixed point wireless local loop (WLL), smart antennas, voice over IP (VoIP), very high speed digital subscriber loop (DSL) etc.

Montium architecture designed for a 16-bit digital signal processing domain provides improvement to Chameleon system. The architecture resembles a VLIW processor - with some similarity to architectures explained in section 3.5 but with an optimized control structure. The VLIW instruction scheduling is done at compile time on to Montium coarse grained processing element called tile processor (TP). The TP consists of five identical ALUs each having its own local memory. There are ten local memories for the five ALUs. A sequencer selects the instructions that are stored in the decoder of TP [PAU06]. Each local SRAM is 16-bit wide and 512 locations deep constituting 8Kbit storage capacity per local memory. These local RAMs can also be used as a LUT to perform functions such as sine or divisions similar to FPGA based LUTs explained in chapter 2. Each ALU has four 16 bit inputs and two 16 bit outputs. Each input has a register file that can store up to four operands and can be written by various sources through a flexible interconnect. The ALUs can directly communicate with other ALUs horizontally i.e., in West-East direction etc.

The flexibility of MONTIUM and granularity of its ALU was decided after mapping various FFT, DCT, FIR, VITERBI, TURBO, BLUETOOTH baseband algorithms [WEB26]. The compile time mapping process starts from description of algorithm in C++ or Matlab. The translation from description to architecture is done by programming in Recore's propriety Montium configuration description language (CDL). The CDL programs are then compiled for ALU mappings by using Montium Synsation compiler. The software tool chain with Montium also contains a cycle accurate Montium graphical simulator called Montium Simsation simulator.

Montium multiprocessor system consisting of multiple TPs is connected by a network on chip communication (NOC) fabric. Each TP has communication and configuration unit (CCU) which implements the network interface for communication between NOC and TP. The routers in the NOC connect to other routers creating a heterogeneous NOC of various hardware modules. There is also an AMBA-Advanced High-Performance Bus (AHB) bridge that connects a reconfigurable fabric to embedded processors, high performance peripherals, DMA controller, on chip memory and various other interfaces.

In [EPR01] turbo/viterbi channel decoder mapping on Montium architecture is described however, the implementation level details for the TP or the turbo decoding such as interleaving, modulation type, SISO arrangement and concatenation (serial / parallel) and fixed point considerations are not disclosed. Similarly Viterbi mappings on the array are accompanied by synthesis implementation results without any implementation level details. Results for the power consumption in Montium are not provided directly however, in [PAU07[the power consumption for one TP memory arrangement is calculated. This calculation is based on FFT mapping and is shown that 0.6 mw/ MHz is consumed by one TP. The overall power consumption in the fabric will depend on the speed and number of TPs required for the mapping. The mappings in [EPR01] show successful achievement of 3GPP and DAB data rates however the important power consumption details are not provided.

The main advantage in VLIW structures like Montium is that the compiler decides what can be executed in parallel and there is no need for the hardware to check dependencies or decide on scheduling as these are resolved at compile time. However as indicated in [PAU04] there are a number of software challenges:

- Firstly, building good compilers for VLIW structures is non-trivial as a best re-arrangement of code for long instruction packing may always be not possible.
- Programs will tend to grow bigger as it may not always be enough instructions that can be done in parallel to fill all the available slots in the instruction,

which in turn means bigger code due to this wasted space. Memory band width always has important consequence on speed and power as it determines the size of the caches.

- It is not possible at compile time to identify all possible sources of pipeline stalls and their durations. A stall on cache miss, on one processor may create dynamic data dependency in other execution units which can result in write hazards (write after write hazard). Stalling all the parallel pipelines to avoid this problem results in poor performance.
- Since compilers now need to know more details about architecture and length of the pipeline etc, changing a hardware component in turn means loss of binary compatability and redesign of compiler.

The solution of these problems for example, may require sufficient hazard resolution hardware to deal with the dependencies that dynamically occur during execution, such that complete processor does not stall when one component does. Therefore, there are numerous hardware and software bottlenecks that limit the performance of VLIW based Montium architecture.

MorphICs [MOWEB] also announced its own version of reconfigurable chips targeting the next generation of wireless applications; it never disclosed any information about the inner design of its solution.

Some domain specific cores have been developed around coarse grain general purpose architectures. Carl et. al [CAR04] developed an OFDM receiver based on RaPiD coarse grain reconfigurable architectures. RaPiD is based on linear array with no register file or crossbar interconnects. Therefore, the OFDM core has inherent limitations of the underlined architecture. RaPiD also shares many of the features of VLIW processor architectures and has similar limitations as discussed for Montium but with some improvements from a typical VLIW solution. For example, the RaPiD instruction format and decode logic is configured for each application according to the dynamic control required. This reduces the instruction width as same instruction field

can be used to control many different parts of the data path. It was stated in [CAR04] that 32-bit instructions have been sufficient to map a wide range of algorithms to a datapath with over 100 functional units and memories.

There were no mappings of error correction decoders found on RaPiD, however work in [CAR04] details mappings for some common wireless baseband components. For example: searching blocks, synchronizing blocks and FFT blocks required in a 4 channel MIMO OFDM receiver. It was concluded in [CAR04] that implementation on RaPiD for searching and tracking algorithms has 16 times the performance of the DSP implementation, at about three times the cost. When compared to ASIC, the RaPiD mapping has about half the performance at about three times the cost. When compared to FPGA, RaPiD has three times the performance at about $1/8^{th}$ of the cost. RaPiD is designed for regular data paths like those found in digital signal processing, graphics and communications. Applications with highly irregular computations and complex addressing patterns or the applications that sparingly reuse data and cannot have fine-grained parallelism will not map well on the architecture. Rapid architecture is explained below:

3.2.2 RAPID ARCHITECTURE FOR OFDM WIRELESS RECEIVER MAPPINGS

Rapid (Reconfigurable pipelined datapath) [EBE96, CRO99, DAR98] is an architecture based on linear arrays. Its functional units (FU) are arranged in a serial sequence (having a common data width generally between 8 to 32 bits wide) but with no register file, or crossbar interconnect. Data is streamed in directly from external memory managed by a stream manager. The programmed controllers decode the operations in to parallel FUs of the data path of RAPID. Data and intermediate results are stored locally in registers and small RAMs, close to their destination FUs. The architecture is shown in figure 3.1.

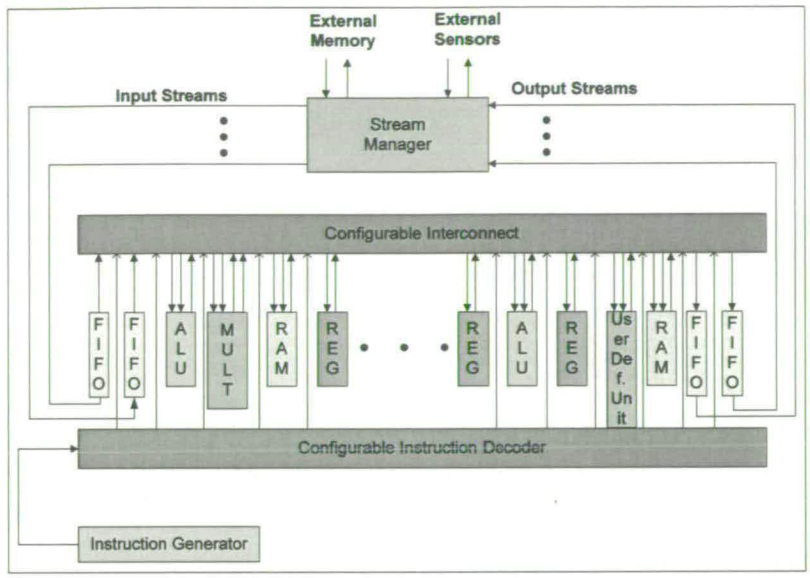

Figure 3.1 Block diagram of RaPiD

Figure 3.1 shows FUs of RAPID. The selection of FUs is chosen based on the application domain for which the device will be used.

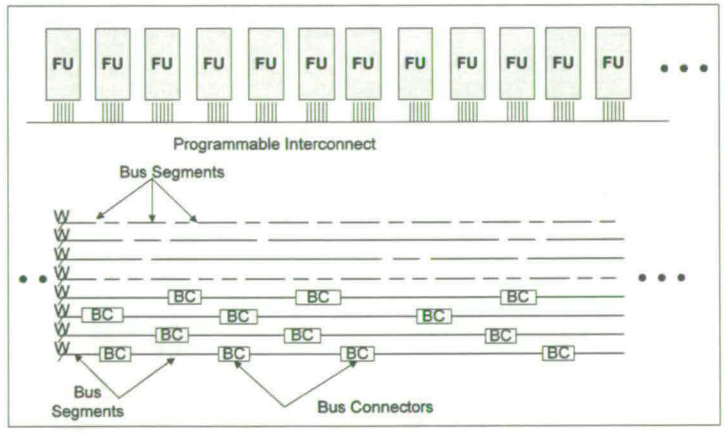

Figure 3.2 interconnects for RAPID

The FUs range in complexity from a simple general purpose register to booth-multiplier with a configurable shifter or a viterbi decoder. The interconnect scheme is created by a linear arrangement of segmented word-based buses. The segments are connected by bus connectors as shown in figure 3.2.

All buses have same width which matches the data width operated on by the FUs. Bus connector can drive left, drive right, or be disconnected by using the tri-state buffers as shown in figure 3.3. The bus connector can also be configured to provide up to three register delays.

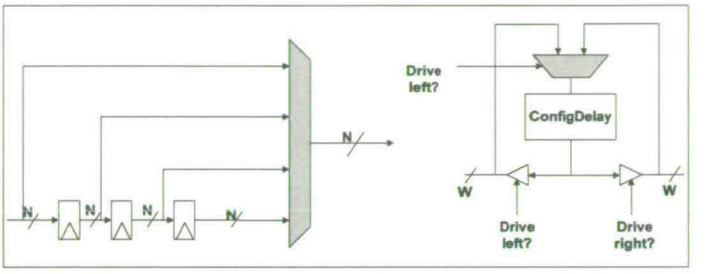

Figure 3.3 Bus connector with configurable delay and BC

An input to a functional unit coming from the tracks is controlled by the multiplexer as shown in figure 3.4.

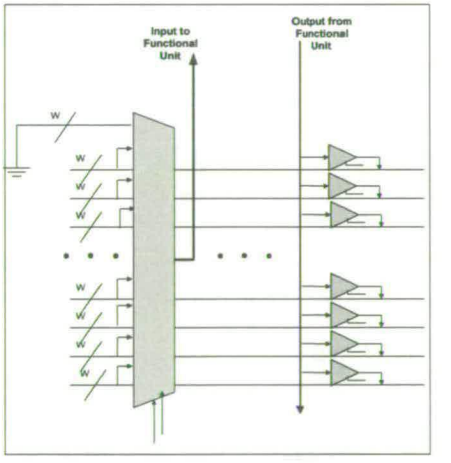

Figure 3.4 Connection of tracks to FUs of RAPID

FUs drive the buses through tristate buffers and each FU can drive an arbitrary number of buses. The multiplexer control bits can be changed dynamically allowing some dynamic reconfiguration. Each track can be driven by any bit of the instruction word. These bits then run in parallel to the datapath and potentially through logic blocks in

order to produce the required soft control signals. Therefore the configuration of the dynamically changeable resources is controlled cycle by cycle using the "instruction generator" and "configurable instruction decoder". There is a stream generator which essentially is a memory interface with an address control and FIFO for each input and output stream. It provides the architecture with streams of data from external memory or other resources and receives the output streams from the RAPID architecture and writes the data to the external devices.

Another method for providing enhanced performance for communication based algorithms is by the use of application-specific instruction set extensions. By creating application specific extensions to an instruction set, the critical portions of an application's dataflow graph (DFG) can be accelerated by mapping them to custom functional units. Though not as effective as ASICs, instructions set extensions improve performance and reduce energy consumption of processor. Instruction set extensions also maintain a degree of system programmability, which enables them to be utilized with more flexibility. Next section describes such architectures

3.2.3 APPLICATION SPECIFIC INSTRUCTION SET PROCESSOR (ASIP) BASED COMMUNICATION DESIGNS

In order to keep the design turnaround times short, there is a general trend in ASIPs (used in communication domain) to avoid complexity of a complete processor design. There are existing processors that allow specialization to their instruction set or data path for example; Xtensa from Tensilica [WEB27], Stretch S5 from Stretch [WEB28], ARCtangent from ARC [WEB29], and LISATek products from CoWare [WEB30]. There are architectures found in literature that use these processors for a unified communication based band mappings.

In [ROS04] UMTS based turbo decoder is mapped on to XiRisc ASIP developed by university of Bologna [ISE95]. The architecture uses 5 stage RISC pipeline tightly integrated with FPGA like reconfigurable array. The design inherits some of the disadvantages of the RISC architecture as the performance suffers due to the load store

pipeline structure and the internal memory bandwidth. The optimizing compiler sees the reconfigurable array as one of the various function units of ALU facilitating the job for compiler. It is demonstrated in [ROS04] that 8 XiRisc processors are required to achieve 3GPP [3GPP99] data rates which makes this solution very impractical for handheld wireless receivers..

In [GIL03] another ASIP based on the Tensilica XTENSA [WEB27] platform targeting the channel decoding domain is presented. This architecture allows new instructions to be added at design time. The hardware for the new instructions within the processor pipeline is synthesized with an ASIC-like flow; hence, the processor cannot be reconfigured after the fabrication. Selection of the new instructions is performed manually by using a simulator and a profiler. When the Xtensa processor is synthesized, a dedicated development tool set is also generated that supports the newly added instruction as function intrinsics. The structure has some similarity to XiRisc architecture but has only four stage RISC pipeline. It suffers from similar 'load-store' drawbacks of the RISC architecture however, the ASIC implementation of data path allows higher turbo decoder throughput (1.4Mbp/s per iteration - 0.2 Mbps for 6 iterations) than XiRisc. The design only allows compile time reconfiguration where as XiRisc allowed dynamic reconfiguration and inspite of the ASIC flow the data rates achieved by XTENSA are still less than that required for 3GPP UMTS [3GPP99] standard.

Another architecture [VOG06] implements viterbi and turbo decoding on ASIP synthesized at 65nm CMOS standard cell libraries. A throughput of 20Mbps (at 5 turbo iterations) is achievable but at a very high clock frequency of 400MHz. Power figures are not quoted for the design which is an important parameter for its domain of interest i.e., battery powered wireless receivers. The design uses 16 parallel ACS blocks at 400 MHz which owing to its size is a very power aggressive design approach. The ASIP is based on framework provided by LISATek from CoWare Inc [WEB30]. It is a SIMD execution model with 11 stages of pipeline and 24 bit wide instruction. The SIMD multiprocessor execution model also suffers from a

complicated compiler, complex instruction scheduling and packing mechanism and runtime data dependency problems that can cause hazards in the processor pipeline. The architecture is claimed to be reconfigurable for DVB and CDMA however only convolution based architecture is described and block based decoding required for DVB are not disclosed.

3.2.4 VITURBO

Viturbo [CAV03] is an architecture developed at university of RICE for a unified turbo and viterbi decoding. The aim of this design was to achieve high data rates suitable for WLAN and 3G standards and allow reconfigurability between these two standards. Its reconfigurable design is based on a full parallel decoding approach for all possible constraint lengths resulting in architecture with 256 parallel states. The array consists of 256 adder-compare-select blocks with the additional control and RAMs circuitry. Reconfigurable interconnect is realized by the use of a large array of multiplexer banks. The design supports Viterbi decoder for constraint lengths 3 to 9 and code rates 1/2 and 1/3. Rate ¼, 1/5 which are also required in 3GPP are not supported. Throughput rates up to 60.5Mbps for Viterbi decoding and 3.54Mbps for Turbo decoding were achieved on Xilinx Virtex II (XC2V2000) FPGA. Power simulations were also performed using Xilinx Xpower simulator. SOVA decoding algorithm were chosen for Turbo decoding which is inferior in BER performance compared to Max Log Map or Log Map decoders. This fully parallel scheme results in larger area and very high power consumption. For example, 3GPP [3GPP99] Viterbi decoder mapping on the array consumes 1.42 Watts of power. The other disadvantage is that the mappings that use only part of the array will have power wasted in the unused circuit. The design requires over 190k logic gates and about 327k bits of memory. The size and the power consumption make the design unsuitable for handheld battery powered wireless receivers.

The research in this thesis investigates speed and power efficient reconfigurable VLSI design for a convolution FEC decoder that can target a large variety of wireless communication standards. Next sections describe some other MIMD, SIMD, VLIW

reconfigurable DSP architectures that may be useful in our reconfigurable design space exploration.

3.3 ARCHITECTURES BASED ON LINEAR ARRAYS

The majority of the VLSI architectures employ a pipelined data path to gain timing advantages. The concept used in pipeline acceleration is to reconfigure pipelines or parts of pipelines onto a reconfigurable architecture. The reconfiguration allows one stage of the pipeline path to be configured in every cycle, while concurrently executing all other stages. The reconfiguration is usually done at run time (dynamic) with an aim to keep the time for reconfiguration as short as possible. This works well for linear pipelines without forks. If there are forks in the pipeline, which would require a two dimensional realization, additional routing resources are needed, which are normally provided by longer lines spanning the whole or part of the array. The linear structure of processing elements allows direct mapping of pipelines with the inherent problem for forks. Unlike most mesh-based architectures (section 3.5), the resources are not evenly distributed as the architecture extends in only one direction.

Piperench and Rapid are the two example architectures in literature with a linear array structure. While Rapid uses a mostly static configuration model, PipeRench relies highly on fast partial dynamic pipeline reconfiguration as well as run time scheduling of both configuration streams and data streams. The architecture is explained here as an illustrative example of dynamic reconfiguration.

PipeRench [GOL02, GOL99] is a coarse grain reconfigurable architecture developed to speed up pipelined applications. The architecture is organised in stripes of pipeline stages. Each strip has 16 processing elements and 8 entry register file. The reconfigurable components allow the configuration of one pipeline stage in every cycle, while executing all other stripes in parallel. It aims to adapt the concept of virtual memory to reconfigurable hardware, resembling a virtual hardware thus implementing a time multiplexing of the physical computation resources. Figure 3.5 shows stripes arrangement in a ring structure.

The ring is formed by connecting the last stripe to the first. R0 register along with the details of PEs is shown in figure 3.6. An application pipeline is mapped on to the PipeRench, and the physical hardware is kept transparent to the application. The state of the over-written virtual stripes is preserved by writing the value into the RO state store memory. The state will be restored when that virtual stripe is returned to the fabric. The processing elements consist of ALUs implemented as 3 input LUTs (8 LUTs/PE), barrel shifter, carry chain circuitry, zero detection circuitry etc.

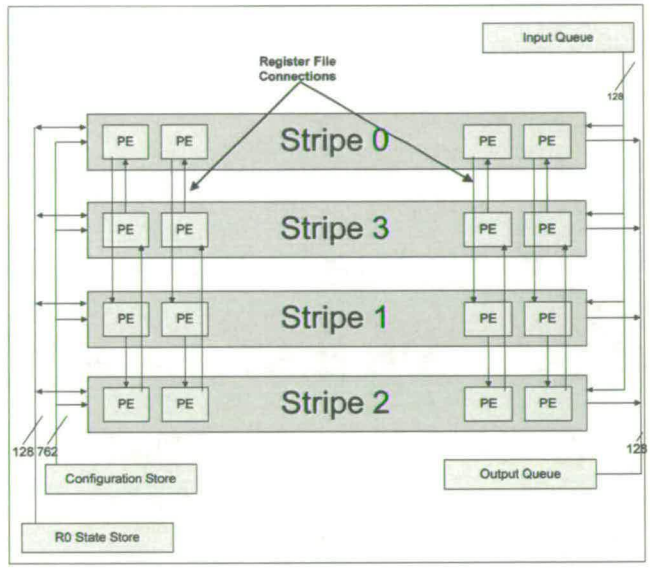

Figure 3.5 Stripes of Piperench architecture

The PEs within a stripe are interconnected through local interconnection network. PEs can access operands from the registered outputs of the previous stripe, as well as registered or unregistered outputs of the other PEs in the same stripe. There are 42 bits required to configure a PE while 672 bits are needed for an entire chip. The output from the PE can be written to any register in the register file. Unused registers are filled by the value from the corresponding register in the previous stripe. The PE output can also connect to horizontal interconnect line, which goes to other PEs in the stripe. This output can be programmed to be connected to the outputs of the previous stripe's register file or the current stripe's RO. When the virtual hardware is larger than

the available stripes, physical stripes will eventually be reconfigured with new virtual stripes.

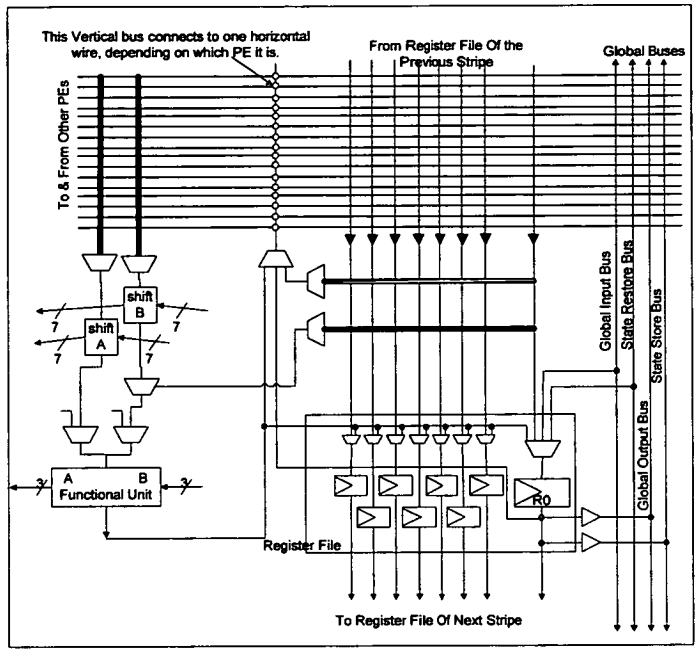

Figure 3.6 Processing elements in piperench.

3.4 WORMHOLE RECONFIGURATION MODEL

In Wormhole runtime reconfiguration model, the data streams to be processed carry a header with configuration information. This header holds the configuration data for both the routing and the functionality of all processing elements the data stream encounters on its way. The main architecture in literature under this classification is Colt [BIT96], [ANN02]. The Colt system is mainly targeted for DSP applications which are implemented by configuring pipelines or part of pipelines onto the architecture. The pipelines are then used to process data streams. Figure 3.7 shows the functional unit of Colt system.

By directly connecting functional units and guaranteeing that only one operand exists on an arc at a given time, the implemented data flow graph is reduced to a set of interacting pipelines. There are four main subsystems in the Colt: Six data ports (DPs),

the crossbar switch, the integer multiplier and the mesh. The mesh is further subdivided into Interconnected Functional Units (IFUs) which is the main computational facility. The FU has 16-bit left and right input registers, each of which can load an operand from any of the four nearest neighbour connections or from any of the four skip bus segments connected to the IFU.

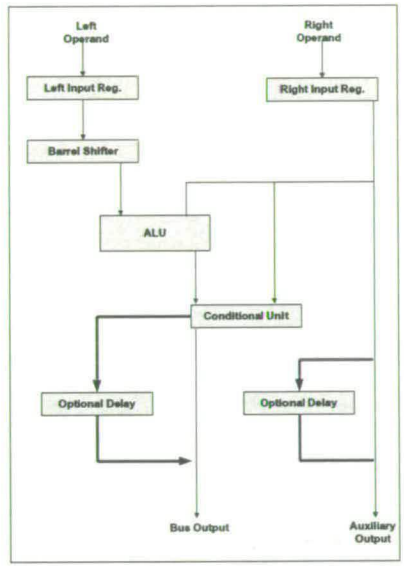

Figure 3.7 Colt functional unit

The conditional unit in figure 3.7 selects left or right path and the output of the conditional unit passes through an optional output delay before being released to the rest of the chip on the four nearest neighbour connections. There is also a carry chain for add, subtract and negation. The IFUs at each side are connected by unidirectional nearest neighbour links in each direction with one outgoing and one incoming port.

There is also a skip bus running between the IFUs which provide segmented connections between IFUs of the rows. In the top row, each IFU has two inputs from cross bar and in the bottom row IFU has single output going to the crossbar. At the left and right edges the two nearest neighbour links and the skip bus are connected to the opposite edges forming a torus structure. The dedicated integer multiplier has two 16 bit inputs and two 16 bit outputs for the high and low word of the 32 bit result in two clock periods. Since the multiplier inputs and outputs are directly connected to the

cross bar the results can be quickly routed to any part of the chip for further processing. The six data ports are bidirectional, each 20 bits wide with 16 bits for data and four bits for stream flow control.

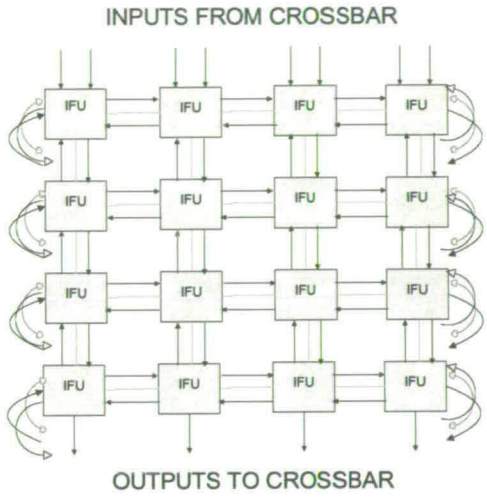

Figure 3.8 Colt IFU interconnection

3.5 VLIW EXECUTION BASED RECONFIGURATION MODEL

Multiprocessor based reconfigurable architectures have multiple instances of ALUs distributed in different placement strategies. There are three execution models for multiprocessor arrays:

- Multiple instruction multiple data (MIMD) model for example Chess array explained in section 3.6.
- Single instruction multiple data (SIMD) for example Morphosys architecture explained in section 3.8.
- Very large instruction word (VLIW) model for example Paddi architectures.

Chess, Morphosys and Paddi are examples of multiprocessor arrays. This section explains Paddi architectures as an example of VLIW execution model and crossbar based routing.

A full crossbar router allows the most flexible communication topology between the processing elements. The routing task thus becomes a simple operation. However, the implementation cost of a full cross bar is very high and therefore usually reduced complexity cross bars are used as in Paddi I.

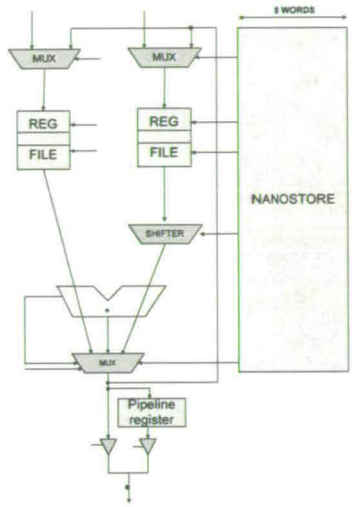

Figure 3.9 EXU of Paddi I

The PADDI (Programmable Arithmetic Device for Digital Signal Processing) family of reconfigurable architectures was developed to address the problem of rapid prototyping for computation intensive DSP data paths. Paddi I [DEV90, CHE92] is a multiprocessor VLIW based architecture in which each execution unit (EXU) operates a four stage pipeline consisting of Fetch – Decode- Execute- Output cycles. The EXU (configurable between 16 or 32 bit wide operations) supports addition, subtraction, saturation, comparison, maximum-minimum and arithmetic right shift operations. Each EXU has an SRAM-based nano store which is configured serially at set-up time. At run time external sequencer broadcasts a 3b global address to each nanostore which is locally decoded in to a 53 bit instruction word. A 3 bit address is able to specify 8 x 53 or one 424 bits very long instruction word. The EXU architecture is explained in figure 3.9. PADDI-2 [YEU93] has a similar architecture but now with Booth multiplier in EXUs (called nanoprocessor in Paddi 2). The small local program at each nanoprocessor implements a node or a cluster of few nodes of the data flow graph.

The arcs of the data flow graph are implemented by a flexible interconnect network that can be configured by programming SRAM cells controlling switches in the interconnect network to create point-to-point links between the nanoprocessors. Computational activities are coordinated by a distributed data-driven control strategy in which nanoprocessor computations are synchronized by passing data and control tokens. Each nanoprocessor has input FIFOs that capture incoming tokens from the communication network. Pleiades is a low power version of this family and consists of additional control processor. It comprises a general-purpose core surrounded by a heterogeneous array of autonomous special-purpose satellite processors. All computation and communication activities are coordinated via a distributed data-driven control mechanism. The dominant, energy-intensive computational kernels of a given DSP algorithm are implemented on the satellite processors as a set of independent, concurrent threads of computation. Examples of satellite processors are Memories, Address generators, PGAs, MAC, ACS and DCTs. Figure 3.10 shows the Pleiades architecture.

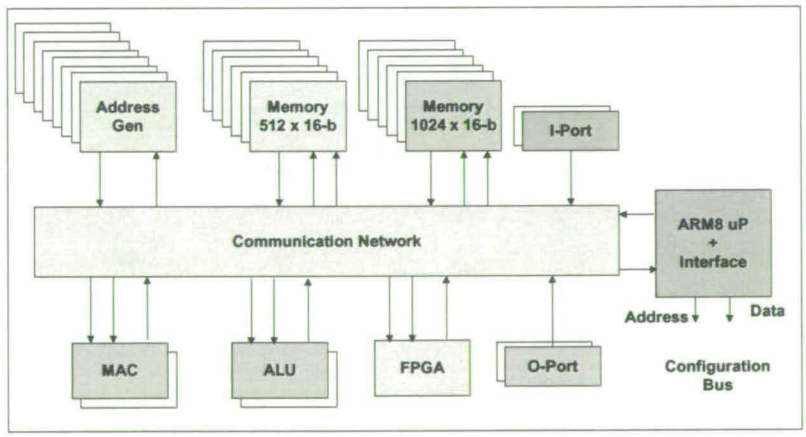

Figure 3.10 Pleiades architecture

The interconnect topology is a generalized mesh [ZHA99] in which the clusters internally use tightly connected generalized mesh architecture and inter-cluster switch boxes allow communication between clusters using the next higher level of communication network. Configuration is loaded into the configuration store registers

by the control processor through a wide configuration bus at a rate of 32 bits per cycle and there is overlap for configuration and kernel execution. This is accomplished by using multiple configuration contexts in multiple configuration store registers. There is also a distributed control mechanism as compared to one large global controller

3.6 MIMD RECONFIGURATION MODEL

The idea of a MIMD execution model based reconfiguration is to provide a highly parallel computing architecture composed of several repeated and interconnected tiles. The tiles comprise of computation facilities (like ALUs) and memory. The best example for such arrangement is the RAW architecture developed in MIT [MIC02, WAI97]. It. is a RISC multi processor architecture and one of the most coarse grained architectures. It is two dimensional array of microprocessor (32-bit MIPS) tiles with 32-bit pipelined floating point unit, local instruction and data caches, controller, register file of 32 general purpose and 16 floating point registers and program counter. It also contains several routers and wiring channels to support static (determined at compile-time) and dynamic (wormhole routing for data forwarding) networks. The prototype chip features 16 tiles arranged in 4 by 4 array. All architectural details are disclosed to the compilation framework. The processors however lack the support for dynamic instruction issuing or caching or register renaming. Due to the lack of these features, the execution model uses statically scheduled instruction streams generated by the compiler. Thus development software has to resolve all the dynamic issues.

The CHESS architecture [MAR99] developed by Hewlett Packard Laboratories is another example of MIMD architecture. The architecture consists of ALUs and switchboxes. These components are arranged in a chessboard-like pattern as shown in figure 3.11. The ALUs feature two inputs and one output (4 bits registered/unregistered) as well as one single-bit input and one output for carry. The instruction set features 16 operations, including add and subtract, nine logical operations, two multiplex operations and three tests using the carry bit as condition output. Carry signals can be routed through the general routing fabric but also have dedicated high speed local routing paths to their north and east neighbours. It is

possible to connect the data output of an ALU to the configuration input of another one. Thus, the functionality of an ALU can be changed in a limited way on a cycle-per-cycle basis during runtime by configuration data generated inside the array.

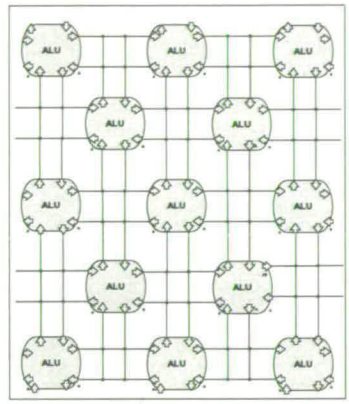

Figure 3.11 Chess board placement pattern in Chess Array

There is no facility for partial reconfiguration from outside of array as is possible in Paddi or Colt architectures. The ALUs and all routing resources are four bits wide. This is shown in figure 3.12.

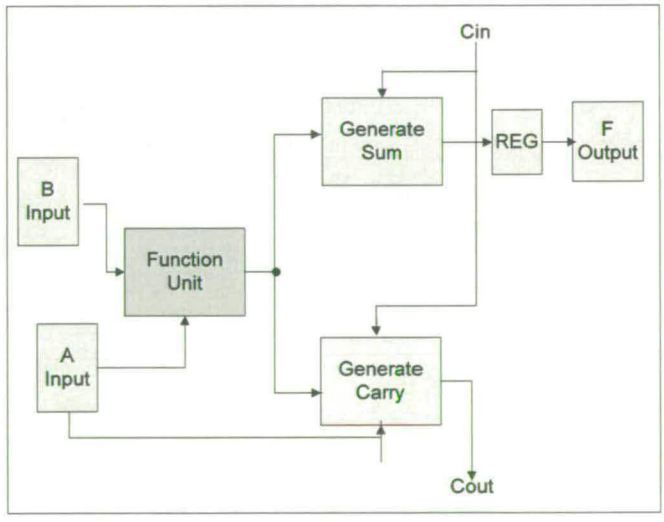

Figure 3.12 Functional unit in Chess

3.7 UNI PROCESSOR RECONFIGURATION MODEL

Domain specific reconfigurable architectures described so far are all coarse grained. The Garp architecture [HAU97] is a fine grained architecture that uses the concept of reconfigurable hardware being used as a slave of a standard MIPS processor. The reconfigurable logic blocks in the array resemble Xilinx 4000 FPGA series. By including a microprocessor, the Garp is targeted for ordinary processing environments, with the reconfigurable array being only activated for acceleration of specific loops or subroutines. The granularity of the processing elements is two bits with clusters of several such processing elements connected across rows. The instruction set of the MIPS-Processor has been extended with instructions to configure and control the array. The array works on a clock counter updated by the processor which determines the number of clock cycles the computation in the array should last. The counter decrements with the array clock cycle. When the counter is zero, updates of the state in the array are stopped. The Garp reconfigurable array consists of entities called blocks. There are at least 32 rows and 24 columns of blocks. One block on each row is known as control block. Figure 3.13 shows the architecture of Garp array.

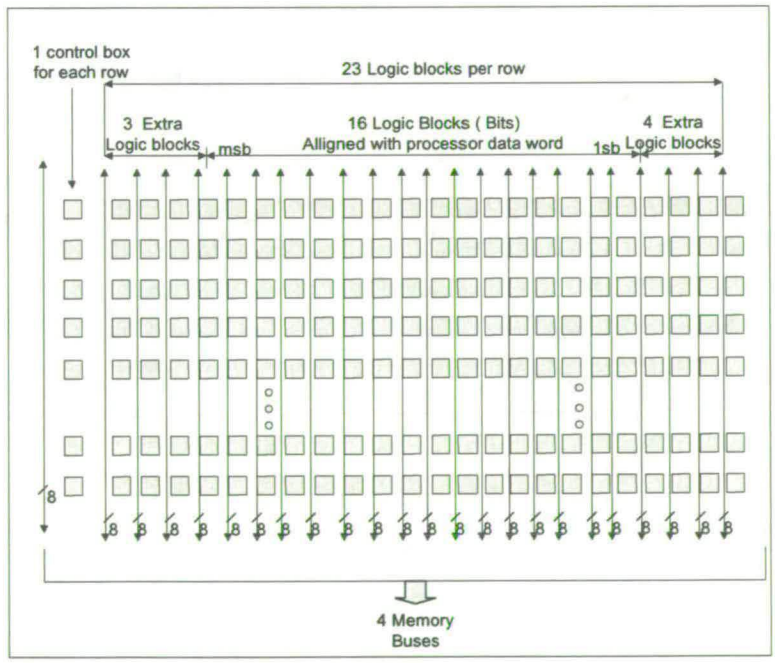

Figure 3.13 Garp architecture.

The 16 consecutive blocks in a row provide operations on 32-bit quantities. In total there are 23 logic blocks per row. The basic unit of reconfiguration is one row, which can be seen to as a kind of reconfigurable ALU, being formed from relatively fine-grained blocks. There is one control block for each row which is used for interfacing tasks like interrupting the processor or for initiating memory accesses. Memory accesses can be initiated by the reconfigurable array, but the data connection to memory is restricted to the central 16 columns with 16 logic blocks of each row.

3.8 SIMD RECONFIGURATION MODEL

A SIMD execution based reconfiguration model consists of a processor and parallel execution units. The processor issues the instructions in a SIMD manner. Morphosys is an example of SIMD reconfiguration model as it consists of a SIMD processor tightly coupled with a reconfigurable array. It is also targeted at highly regular applications with inherent data-parallelism.

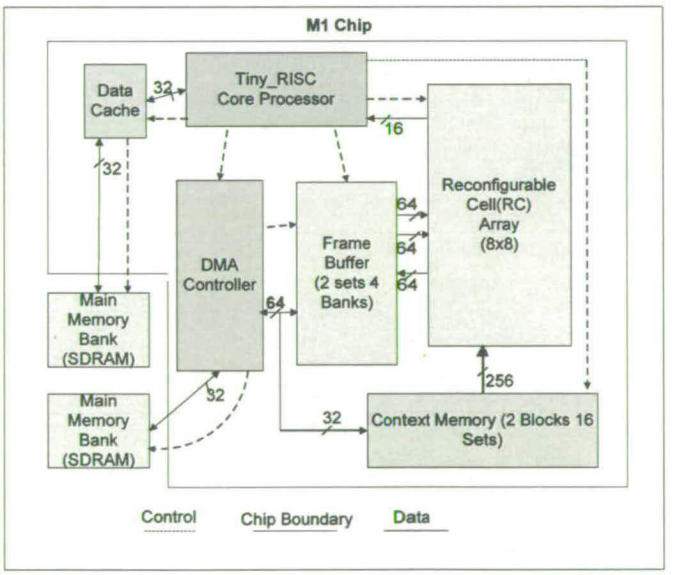

Figure 3.14 Morphosys block diagram

The Morphosys architecture comprises a core processor (Tiny RISC), a frame buffer, a DMA controller, a context memory, and a reconfigurable component organized in

SIMD fashion as an array of 8 by 8 reconfigurable cells. The core processor has an extended instruction set for manipulation of the DMA controller and the reconfigurable array as shown in figure 3.14.

The programmable reconfigurable cell (RC) array of MorphoSys comprises an 8 by 8 array of identical processing elements. The array is divided into four quadrants of 4 by 4 cells each. A RC features a 16-bit datapath, comprising a 16x12 Multiplier, a shift unit, two input multiplexers, a register file with four 16 bit registers and a 32 bit context register for storing the configuration word. The multiplier can perform the standard arithmetic and logical operations as well as a multiply-accumulate operation in a single cycle. The multiplier has four inputs, two over the input multiplexers, one from the output register, and one connected to the context register to load a 12 bit operand contained in the configuration word. The two input multiplexers select one of several inputs for the Multiplier based on the control bits from the context word in the RC context register. These inputs include the outputs of the four nearest neighbour RCs, outputs of other RCs in the same row or column (with the quadrant), horizontal and vertical express lines, FB data bus and RC register file. This is shown in figure 3.15.

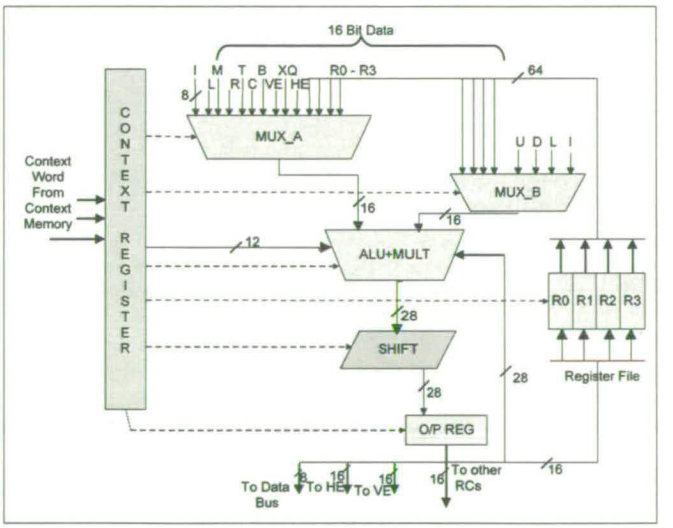

Figure 3.15 Reconfigurable cell in Morphosys

The RC array interconnect consist of three hierarchical levels. In the first level, all cells are connected to their four nearest neighbours in a 2D mesh. The second level of connectivity is at the quadrant level (4x4 group of RCs). The RC array has four quadrants and within each quadrant, each cell can access the output of any other cell in its row and column. The third layer of interconnect consists of buses at a global level spanning the whole array and allowing transfer of data from a cell in a row or column of a quadrant to any other cell in the same row or column in the adjacent quadrant.

3.9 SUMMARY

This chapter introduced domain specific processor for configurable computations. Conventional programmable architectures shown in chapter 2 are concluded to be far less energy efficient than custom, application-specific devices. The inefficiency is due to the manner in which flexibility is achieved in conventional processors. Computations are performed on functional units that are designed for a much larger domain of applications. This makes the functional units large and complex, and their granularity is not always well-matched to the data types and the computations required by target algorithms. Similarly data operands are also stored in large centralized memory structures. This basic weakness afflicted all of the architectures that were discussed in chapter 2. It was then shown in this chapter that architectures that target a smaller set of applications can be more efficient than general-purpose devices and must be pursued. Domain-specific architectures can be particularly efficient, as they provide the architect with the opportunity to match architectural parameters to the properties of the target domain of algorithms. The next chapter proposes a reconfigurable solution for turbo decoding keeping in view the lessons learnt from the architectures in chapter 2 and chapter 3.

Chapter 4

RECONFIGURABLE TURBO DECODING

4.1 INTRODUCTION

In this chapter the solution to the problem of reconfigurable turbo decoding is presented. A two fold reconfiguration context is designed, one is the reconfiguration of turbo decoding array itself as it is used in various wireless communication standards. Secondly, reconfiguration to adapt a turbo decoder for a common communication platform consisting of unified turbo-viterbi decoding components. For the first context, reconfiguration provides flexibility between different generator polynomials, constraint lengths, rates and frame sizes. In the second context reconfiguration addresses issues like state metrics normalization, sharing of input/output RAMs etc. For both contexts, a reconfigurable state machine control for individual and unified turbo decoding components is provided. A low power technique for caching two Window lengths on input matrices that reduces the read excess to larger input RAMs is also developed. Similarly branch metrics storage is avoided by performing more recalculations.

The rest of the chapter is organized as follows. In section 4.2 and 4.3 the mathematical basis of the algorithm is briefly described. Section 4.4 describes the modifications done to the algorithm in literature to make it more suitable for hardware implementation. Section 4.5 gives the iterative BER improvement in turbo decoding. Section 4.6 to 4.11 presents different components of the reconfigurable VLSI design and the domains of reconfiguration. At the end comparisons of our results with the state of art and the contributions are discussed.

4.2 TURBO DECODING – THE INVENTION

Claude Berrou, Alain Glavieux and Patrick Adde seminal contribution in [BER93] reported excellent coding gains approaching Shannonian predictions [SHA48]. This gave an insightful concept in to the error correcting power of turbo codes. The work in [BER93] was based on literature of G. Battail in 1987 [BAT87] and J. Hagenauer and P. Hoecher in 1989 [HAG89]. These scientists started their work on the Soft-Output Viterbi Algorithm which stimulated them to consider cascading/concatenating coding techniques for large asymptotic gains. Punya Thititmajshima, later joined the group in 1989 as a Ph. D student with his work started showing in literature in 1995 [THI95]. The beginning of SOVA was then replaced by Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm [BAH74] at the end of discovery. The main patent filed for Turbo decoding is 1992 [PAT1].

4.3 THE MAXIMUM-A-POSTERIORI (MAP) ALGORITHM – MATHEMATICAL DESCRIPTION

The MAP algorithm [BAH74] examines every possible path through the convolutional encoder trellis and therefore initially seemed to be unfeasibly complex for the majority of applications. Hence it was not widely used before the discovery of Turbo Codes. A brief description of the Log-MAP algorithm is given below. Throughout the thesis binary codes are assumed.

The MAP algorithm gives, for each decoded bit u_k, the probability that this bit was +1 or -1, given the received symbol sequence y_o.This is equivalent to finding the aposteriori log likelihood ratio

$$L(u_K \mid y_o) = \ln \frac{P(u_k = +1|)}{P(u_K = -1)} \qquad (1)$$

Using Bayes' rule which gives the joint probability of a and b, $P(a \wedge b)$, in terms of the conditional probability of *a given b* '$P(a \mid b)$' as

$$P(a \wedge b) = P(a \mid b).P(b) \qquad (2)$$

and also using its consequence

$$P(\{a \wedge b\} \mid c) = P(a \mid \{b \wedge c\}).P(b \mid c) \quad (3)$$

Bayes' rule allows us to rewrite the equation (1) as:

$$L(u_K \mid y_o) = \ln \frac{P(u_k = +1 \wedge y_o)}{P(u_K = -1 \wedge y_o)} \quad (4)$$

Let us consider the convolution encoder with K=3, for which the possible transitions are shown in the figure 4.1

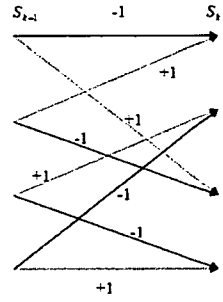

Figure 4.1: Possible transitions in the trellis corresponding to Constraint length K=3

For this code there are four encoder states, and since we consider a binary code, therefore, in each encoder state S_{K-1} there are two possible transitions. These transitions depend whether the input value is 1 or 0 (-1). The next state 'S_K' is connected to previous state S_{K-1} by a unique trellis structure which varies from standard to standard. The transition associated with the input bit of -1 is shown as continuous line and that associated with the input bit of +1 is shown as a broken line. It can be seen from Figure 4.1 that if the previous state S_{k-1} and the present state S_k are known, then the value of the input bit u_k, which caused the transition between these two states, will be known. The probability that $u_k = +1$ is equal to the probability that the transition from the previous state S_{k-1} to the present state S_k is one of the set of four possible transitions that can occur when $u_k = +1$. Since these transitions are mutually exclusive we can rewrite the equation (4) as:

$$L(u_k \mid y_o) = \ln \frac{\sum_{(s`,s)=>u_k=+1} P(S_{k-1} = s` \wedge S_k = s \wedge y_o)}{\sum_{(s`,s)=>u_k=-1} P(S_{k-1} = s` \wedge S_k = s \wedge y_o)} \quad (5)$$

where (s`, s) $\Rightarrow u_k = +1$ is the set of transitions from the previous state $S_{k-1} = s`$ to the present state $S_k = s$ that can occur if the input bit $u_k = +1$, and similarly for (s`,s) $\Rightarrow u_k = -1$. For simplicity we represent $P(S_{k-1} = s` \wedge S_k = s \wedge y_o)$ as $P(s` \wedge s \wedge y_o)$.

Rewriting $P(s` \wedge s \wedge y_o)$ as $P(s` \wedge s \wedge y_0^{j<k} \wedge y_0^{k} \wedge y_0^{j>k})$ where y_0^{k} is the received codeword associated with the present transition at stage k of trellis. $y_0^{j<k}$ is the received sequence prior to the present transition and $y_0^{j>k}$ is the received sequence after the present transition. Using Bayes' rule from equation (2) and the fact that if we assume that the channel is memory less, then the future received sequence $y_0^{j<k}$ will depend only on the present state s and not on the previous state s` or the present and previous received channel sequences y_0^{k} and $y_0^{j<k}$, therefore :

$$P(s` \wedge s \wedge y_o) = P(y_0^{j>k} \mid \{s` \wedge s \wedge y_0^{j<k} \wedge y_0^{k}\}).P(s` \wedge s \wedge y_0^{j<k} \wedge y_0^{k}) \quad (6)$$

$$P(s` \wedge s \wedge y) = P(y_0^{j<k} \mid s).P(s` \wedge s \wedge y_0^{j<k} \wedge y_0^{k}) \quad (7)$$

Again using Bayes' rule we expand equation (7) as follows:

$$\begin{aligned} P(s` \wedge s \wedge y_0) &= P(y_0^{j<k} \mid s).P(\{y_0^{k} \wedge s\} \mid \{s` \wedge y_0^{j<k}\}).P(s` \wedge y_0^{j<k}) \\ &= P(y_0^{j<k} \mid s).P(\{y_0^{k} \wedge s\} \mid s`).P(s` \wedge y_0^{j<k}) \\ &= \beta_k(s).\gamma_k(s`,s).\alpha_{k-1}(s`) \quad (8) \end{aligned}$$

where: $\alpha_{k-1}(s`) = P(S_{k-1} = s` \wedge y_0^{j<k})$ (9)

Equation (9) explains that the trellis is in state s` at time k-1 and the received channel sequence up to this point is $y_0^{j<k}$.

Similarly

$$\beta_k(s) = P(y_0^{j>k} \mid S_k = s) \quad (10)$$

and

$$\gamma_k(s`,s) = P(y_0^k \wedge S_k = s \mid S_{k-1} = s`) \tag{11}$$

The MAP algorithm finds $\alpha_k(s)$ and $\beta_k(s)$ for all states s through the trellis, i.e., for all stages k=0,1,...N-1 and $\gamma_k(s`,s)$ for all possible transitions from state $S_{k-1}=s`$ to state $S_k=s$.

4.3.1 FORWARD RECURSIVE CALCULATION OF THE $\alpha_k(s)$ VALUES

As shown above we have $\alpha_{k-1}(s`)$ as

$$\alpha_k(s) = P(S_k = s \wedge y_0^{j<k+1})$$

$$= P(s \wedge y_0^{j<k} \wedge y_0^k)$$

$$= \sum_{all->s`} P(s \wedge s` \wedge y_0^{j<k} \wedge y_0^k) \tag{12}$$

using Bayes' rule again and the assumption that the channel is memory less we obtain the following equations:

$$= \sum_{all->s`} P(\{s \wedge y_0^k\} \mid \{s` \wedge y_0^{j<k}\}.P(s` \wedge y_0^{j<k})$$

$$= \sum_{all->s`} \gamma_k(s`,s).\alpha_{k-1}(s`) \tag{13}$$

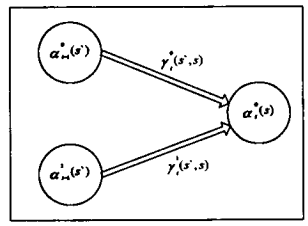

Figure 4.2: Calculating alpha probability

Thus, with the $\gamma_k(s`,s)$ values, the $\alpha_k(s)$ values can be calculated recursively as shown in figure 4.2.

4.3.2 BACKWARD RECURSIVE CALCULATION OF THE $\beta_k(s)$ VALUES:

Similar to the above derivation, the $\beta_k(s)$ values can be evaluated. Again from equation (10) we can write $\beta_{k-1}(s`)$ as:

$$\beta_{k-1}(s`) = P(y_0^{j>k-1} \mid S_{k-1} = s') \tag{14}$$

By splitting a single probability into the sum of joint probabilities and using the derivation from Baye's rule as well as the assumption that the channel is memory less we have:

$$= \sum_{all=>s} P(\{y_0^{j>k-1} \wedge s\} \mid s')$$

$$= \sum_{all=>s} P(\{y_0^{k} \wedge y_0^{j>k} \wedge s\} \mid s') \tag{15}$$

From equation (3)

$$= \sum_{all=>s} P(\{y_0^{j>k} \mid \{s` \wedge s \wedge y_0^{k}\}.P\{y_0^{k} \wedge s\} \mid s')$$

$$= \sum_{all=>s} P(y_0^{j>k} \mid s).P\{y_0^{k} \wedge s\} \mid s')$$

$$= \sum_{all=>s} \beta_k(s).\gamma_k(s`,s). \tag{16}$$

Calculation of β probabilities is shown by figure 4.3:

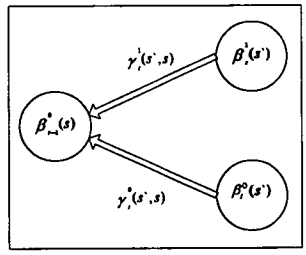

Figure 4.3: Calculating beta probability

4.3.2 CALCULATION OF THE $\gamma_k(s`,s)$ VALUES

From equation (11) we have $\gamma_k(s`,s) = P(\{y_0^k \wedge s\} \mid s`)$ which is rewritten according to equation (3) which => $P(\{a \wedge b\} \mid c) = P(a \mid \{b \wedge c\}).P(b \mid c)$, hence

$$\gamma_k(s`,s) = P(y_0^k \mid \{s` \wedge s\}).P(s \mid s`)$$

$$\gamma_k(s`,s) = P(y_0^k \mid \{s` \wedge s\}).P(u_k) \tag{17}$$

where u_k is the input bit necessary to cause the transition from state S_{k-1}=s` to $S_k$=s and $P(u_k)$ is the a-priori probability of this bit. This first term in the equation $P(y_0^k \mid \{s` \wedge s\})$, is equivalent to $P(y_k \mid x_k)$, where x_k is the transmitted codeword associated with the transition from one state S_{k-1} = s` to S_k = s.

Thus we can write

$$P(y_0^k \mid \{s` \wedge s\}) \equiv P(y_0^k \mid x_0^k) = \prod_{l=1}^{n} P(y_{kl} \mid x_{kl}) \tag{18}$$

where x_{kl} and y_{kl} are the individual bits in transmitted and received code words between x_0^k and y_0^k, and n is the number of bits in each code word x_0^k and y_0^k.

Assuming that the transmitted bits x_{kl} have been transmitted over a Gaussian channel using BPSK, so that the transmitted symbols are +1 or -1 we have.

$$P(y_{kl} \mid x_{xl}) = \frac{1}{\sqrt{2\prod\sigma}} \exp(-\frac{E_b}{2\sigma^2}(y_{kl} - ax_{kl})^2)$$

where E_b is the transmitted energy per bit, σ^2 is the noise variance and a is the fading amplitude (a=1 for non-fading AWGN channels).

$$P(y_0^k \mid \{s` \wedge s\}) = \prod_{l=1}^{n} \frac{1}{\sqrt{2\prod\sigma}} \exp(-\frac{E_b}{2\sigma^2}(y_{kl} - ax_{kl})^2)$$

$$P(y_0^k \mid \{s` \wedge s\}) = \frac{1}{(\sqrt{2\pi\sigma})^n} \exp(-\frac{E_b}{2\sigma^2} \sum_{l=1}^{n} (y_{kl} - ax_{kl})^2)$$

$$P(y_0^k \mid \{s` \wedge s\}) = \frac{1}{(\sqrt{2\pi\sigma})^n} \exp(-\frac{E_b}{2\sigma^2} \sum_{l=1}^{n} (y^2{}_{kl} + a^2 x_{kl}^2 - 2ax_{kl}y_{kl}))$$

$$P(y_0^k \mid \{s` \wedge s\}) = C_1.C_2.\exp(\frac{E_b}{2\sigma^2} 2a \sum_{l=1}^{n} x_{kl}y_{kl}) \tag{19}$$

$$C_1 = \frac{1}{(\sqrt{2\pi\sigma})^n}\exp(-\frac{E_b}{2\sigma^2}\sum_{l=1}^{n} y^2{}_{kl})$$

depends only on the channel SNR and on the magnitude of the received sequence.

$$C_2 = \exp(-\frac{E_b}{2\sigma^2}a^2\sum_{l=1}^{n} x_{kl}^2)$$

$$C_2 = \exp(-\frac{E_b}{2\sigma^2}a^2 n)$$

depends only on the channel SNR and fading amplitude. Hence we can write for $\gamma_k(s`,s)$:

$$\gamma_k(s`,s) = P(u_k).P(y_0^k \mid \{s` \wedge s\})$$

We also know that

$$P(u_k) = (\frac{e^{-L(u_k)/2}}{1+e^{-L(u_k)}}).e^{(u_k L(u_k)/2)}$$

$$P(u_k) = C_3.e^{(u_k L(u_k)/2)}$$

$$C_3 = (\frac{e^{-L(u_k)/2}}{1+e^{-L(u_k)}})$$

C_3 depends only on the LLR $L(u_k)$ and not on the whether u_k is +1 or -1.

Therefore equation (19) becomes:

$$\gamma_k(s`,s) = C.e^{(u_k L(u_k)/2)}.\exp(\frac{E_b}{2\sigma^2}2a\sum_{l=1}^{n} x_{kl}y_{kl})$$

$$\gamma_k(s`,s) = C.e^{(u_k L(u_k)/2)}.\exp(\frac{L_c}{2}\sum_{l=1}^{n} x_{kl}y_{kl}) \qquad (20)$$

where $C=C_1$. C_2 C_3 does not depend on the sign of the bit u_k or the transmitted code word x_k and so is constant over the summations in the numerator and denominator in equation (5) and cancels out. Hence from equation (5) and (8):

$$L(u_k \mid y_o) = \ln\frac{\sum_{(s`,s)=>u_k=+1} P(S_{k-1} = s` \wedge S_k = s \wedge y_o)}{\sum_{(s`,s)=>u_k=-1} P(S_{k-1} = s` \wedge S_k = s \wedge y_o)}$$

$$L(u_k \mid y_o) = \ln\frac{\sum_{(s`,s)=>u_k=+1} \alpha_{k-1}(s`).\gamma_k(s`,s).\beta_k(s)}{\sum_{(s`,s)=>u_k=-1} \alpha_{k-1}(s`).\gamma_k(s`,s).\beta_k(s)} \qquad (21)$$

The decoding bits are estimated based on this value of $L(u_k \mid y_0)$.

When $L(u_k \mid y_0)>0$, $u_k = +1$ and

$L(u_k \mid y_0) < 0$, $u_k = -1$. and when

$L(u_k \mid y_0) = 0$, one cannot be sure about the value of u_k.

MAP algorithm gives exceptional BER results at relatively low E_b/N_0, however, the MAP algorithm is extremely difficult to implement in hardware as it contains mathematical operations such as logarithms and divisions. A modification of the MAP algorithm is the Log – MAP algorithm [ROB89]. As the name suggests the log-MAP algorithm is an implementation of the MAP algorithm in the logarithm domain.

4.4 LOG-MAP DECODING ALGORITHM:

Taking the logarithm of equations (13), (16), (20) and (21)

$$A_k(s) = \ln(\alpha_k(s)) \tag{22}$$

$$B_k(s) = \ln(\beta_k(s)) \tag{23}$$

$$\Gamma_k(s`,s) = \ln(\gamma_k(s`,s)) \tag{24}$$

The Jacobian logarithm can be used to obtain a formula for both (22) and (23) that can be easily implemented in hardware

$$\text{Max}^*(A_k^0, A_k^1) = \ln(e^{A_k^0} + e^{A_k^1})$$

where

$$\text{Max}^*(A_k^0, A_k^1) = \text{Max}(A_k^0, A_k^1) + \ln(1+e^{-|A_k^0 - A_k^1|})$$

We can rewrite equation (22) using equation (13) as

$$A_k(s) = \ln\left(\sum_{all->s`} \gamma_k(s`,s).\alpha_{k-1}(s`)\right)$$

$$A_k(s) = \ln\left(\sum_{all->s`} \exp[\Gamma_k(s`,s) + A_{k-1}(s`)]\right)$$

$$A_k(s) = \max^*_{s`}([\Gamma_k(s`,s) + A_{k-1}(s`)]) \tag{25}$$

Equation (16) can be rewritten as:

$$B_{k-1}(s`) = \max^*_s([\Gamma_k(s`,s) + B_k(s)])$$

and equation (20) becomes

$$\gamma_k(s`,s) = \ln\{C.e^{(u_k L(u_k)/2)}.\exp(\frac{L_c}{2}\sum_{l=1}^{n} x_{kl}y_{kl})\}$$

$$\gamma_k(s`,s) = C`+\frac{1}{2}.(u_k L(u_k)) + \frac{L_c}{2}\sum_{l=1}^{n} x_{kl}y_{kl} \qquad (26)$$

Similarly equation (21) becomes:

$$L(u_k \mid y_o) = \ln \frac{\sum_{(s`,s)=>u_k=+1} \exp(A_{k-1}(s`)+\Gamma_k(s`,s)+\beta_k(s))}{\sum_{(s`,s)=>u_k=-1} \exp(A_{k-1}(s`)+\Gamma_k(s`,s)+\beta_k(s))}$$

$$L(u_k \mid y_o) = \max^*_{(s`,s)=>u_k=+1}(A_{k-1}(s`)+\Gamma_k(s`,s)+\beta_k(s)) - \max^*_{(s`,s)=>u_k=-1}(A_{k-1}(s`)+\Gamma_k(s`,s)+\beta_k(s))$$

The add, compare, select, offset component is at the heart of all of the max* computations, it is shown graphically in figure 4.4 [BOU03].

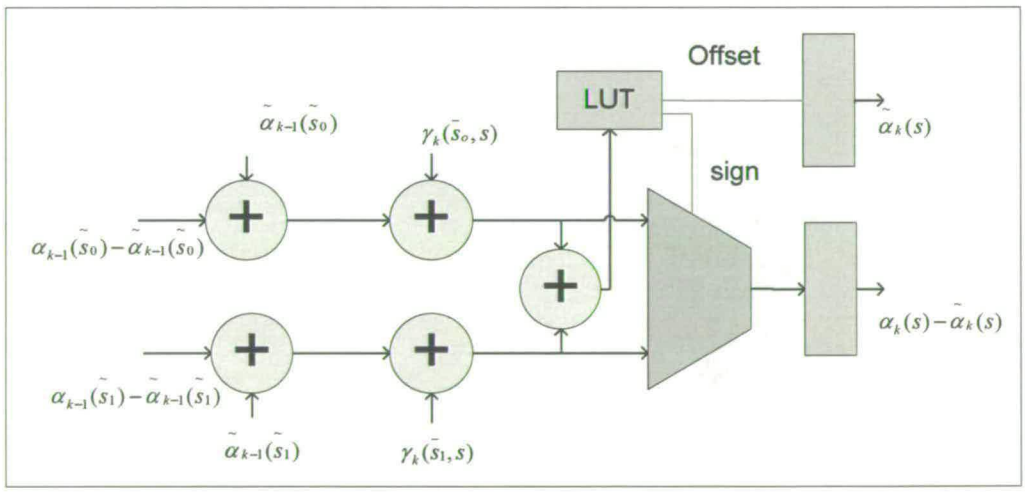

Figure 4.4. ACS block

The log-MAP algorithm can be further simplified by excluding the offset $\ln(1+e^{-|A_k^0 - A_k^1|})$ in above equations, resulting in an implementation called the Max-log-MAP algorithm [ERF94]

4.5 THE TURBO CONCEPT

Traditional Turbo codes use recursive systematic convolutional (RSC) encoders and soft-in, soft-out (SISO) decoders. The information is encoded twice, with an interleaver between the two encoders. A top level diagram of a typical Turbo encoder is shown in figure 4.5.

The interleaver decorrelate the input to the second encoder to make the encoded data sequences approximately statistically independent of each other. Each RSC encoder produces the parity information. The two parity sequences from the corresponding encoders can be punctured before being transmitted along with the original sequence to the decoder. This puncturing of parity information allows a wide range of coding rates to be realised and often half the parity information from each encoder is sent along with the original data sequence.

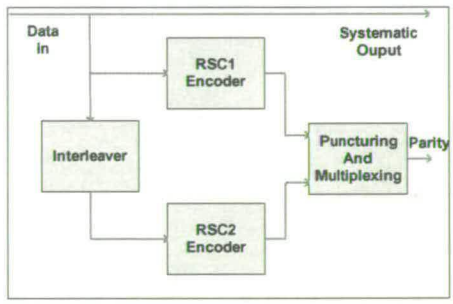

Figure 4.5 Turbo RSC encoders connected by interleaver

A Turbo decoder contains two RSC encoders, the number of SISO decoders always match the number of RSC encoders present in the corresponding Turbo encoder. A SISO decoder reads in a soft data value and produces a soft data value at its output. Soft data is represented by an m-bit symbol. The magnitude of the symbol presents the confidence of the decoder producing the correct output. The multiple encoder and interleaver arrangement generates multiple views of the source data. Each view has its parity and systematic information. After passing through a channel each set of data are input to a SISO decoder. The SISO decoders operate iteratively. In the first iteration the first SISO decoder provides a soft output giving an estimation of the original data sequence based on the soft channel inputs alone. It also provides an extrinsic output which is used by the second RSC decoder (after interleaving) as

a priori information. The sharing of data between the decoders is what gives Turbo codes such a low BER. Top level diagram of turbo decoder is shown in figure 4.6. All data into SISO2 from SISO1 must be interleaved and all data into SISO1 from SISO2 must be de-interleaved. L_1^e and L_2^e are the data values shared between the decoders and are known as extrinsic data. The extrinsic information related to a bit u_k is the information provided by a decoder based on the received sequence and on the a-priori information, but excluding the received systematic bit and the a priori information related to bit u_k. A posteriori information related to a bit is the information that the decoder generates by taking into account all available sources of information concerning u_K. The apriori information related to a bit is information known before decoding commences, from source other than the received sequence or the code constraints. Data is shared between SISO decoders for a pre-determined number of iterations. Upon completing these iterations the turbo decoder produces a soft estimate of the decoded bits.

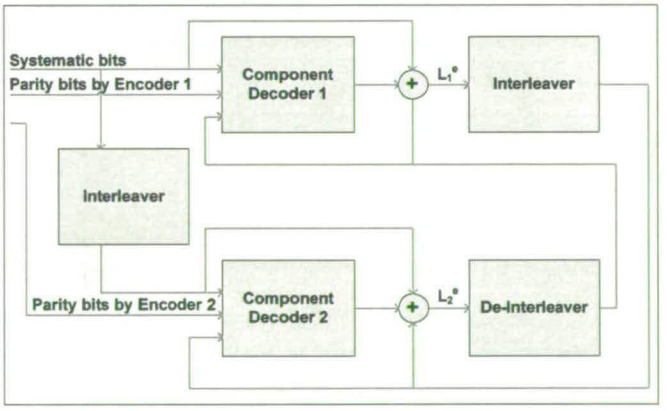

Figure 4.6.Block diagram of turbo decoder

4.6 RECONFIGURABLE VLSI DESIGN TARGETING MULTIPLE STANDARDS

The reconfigurable work on turbo decoder adds to the overall contribution of the thesis i.e., to provide Forward Error Correction (FEC) structure for a common communication platform designed for ubiquitous networks. The platform provides access to any network available to the user for example WLAN, 3G or GSM etc. In

the first phase of research we have looked at the investigation of accurate flexibility in order to meet the performance constraints as imposed by various standards. In the second phase a high speed, power and area efficient unified turbo-viterbi architecture is developed. This is in contrast to designing separate and optimized fixed decoders for all possible standards and switching between one of them. The implemented architecture is capable of Turbo and Viterbi decoding for a wide range of code parameters varying from constraint length 3 to 9, any generator polynomial and different rates. This reconfigurable design gives huge savings in area, energy and speed as opposed to the other option of implementing these decoders separately of each of the standards.

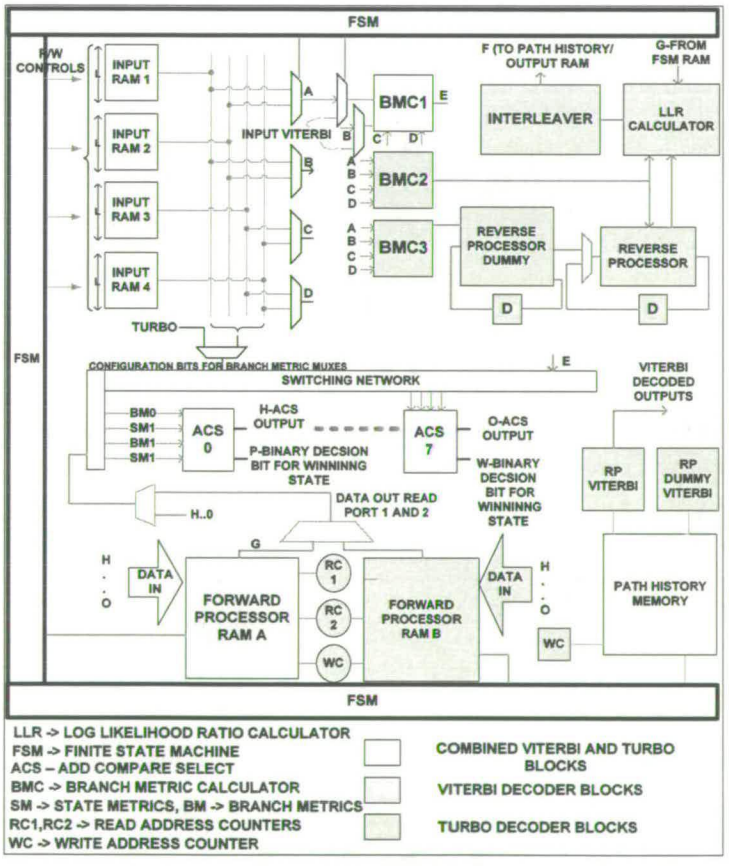

Figure 4.7 Block diagram of unified array

We have also presented a unified control structure for controlling the algorithmic flow in different standards. The control of viterbi components is provided by this reconfigurable finite state machine which avoids the use of separate microprocessor to control these blocks. Figure 4.7 represents the overall block diagram of the array and the individual components of this diagram are explained in two chapters. This chapter explains the individual turbo decoder components of the array individually and explains the design flow along with the reasons for the choice of these components and the flexibility. Chapter 5 explains the viterbi components of the block diagram shown in figure 4.7.

4.7 RECONFIGURABLE DOMAIN

Turbo decoders (example encoders shown in figure 4.8 and 4.9) share various common units; however, the number of units used and the connections of these units are highly variable and change upon any change in coding parameters. For instance, the path metrics computed from one Add Compare Select (ACS) unit might need to be routed back to any of a number of ACS units that need the updated path metrics for their computation. Similarly with the change of constraint length or coding rate or generator polynomial, the data routes are altered and the operations being performed inside a unit are changed accordingly. The other constraint on the design is that the architecture should support both viterbi and turbo decoding. The component RSC encoder for the UMTS standard is shown in figure 4.8. As each UMTS RSC encoder outputs only one parity stream, the encoder has a standard rate of 1/3. A CDMA2000 component RSC encoder is shown in figure 4.9. The CDMA2000 RSC encoder has two parity stream outputs and can therefore have a code rate of ½, 1/3, ¼ or 1/5.

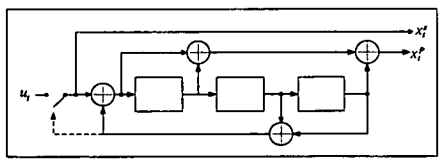

Figure 4.8: UMTS component encoder

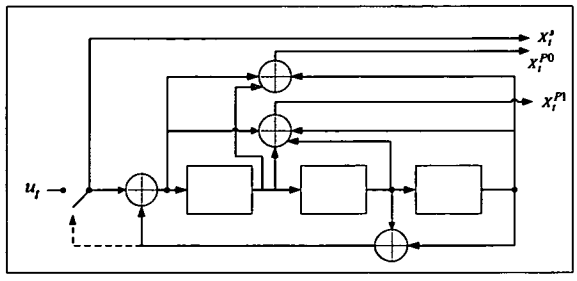

Figure 4.9: CDMA2000 component encoder

The puncturing patterns of CDMA 2000 standard are shown in table 4.1. In the table a 0 represents the bit that is deleted and a non-zero number given in table 4.2 shows how many times the symbol in question is transmitted. For example, if data is being transmitted at a code rate of ½ the output of the encoder will be $X_t^s, X_t^{P0}, X_t^s, X_t^{`P0}$.

	Code Rate			
Output	½	1/3	¼	1/5
X_t^s	11	11	11	11
X_t^{P0}	10	11	11	11
X_t^{P1}	00	00	10	11
$X_t^{`s}$	00	00	00	00
$X_t^{`P0}$	01	11	01	11
$X_t^{`P1}$	00	00	11	11

Table 4.1 Table showing the corresponding bit to be transmitted for different rates in 3GPP [3GPP99]

When the trellis termination is initialized the puncturing pattern is altered so that $X_t^{`s}$ is transmitted, certain bits are repeated so that the code rate during trellis termination matches the 6/R; where R is the rate and each encoder clocked three times with switch in the lower position.

	Code Rate			
Output	½	1/3	¼	1/5
X_t^s	111 000	222 000	222 000	333 000
X_t^{P0}	111 000	111 000	111 000	111 000
X_t^{P1}	000 000	000 000	111 000	111 000
$X_t^{'s}$	000 111	000 222	000 222	000 333
$X_t^{'P0}$	000 111	000 111	000 111	000 111
$X_t^{'P1}$	000 000	000 000	000 111	000 111

Table 4.2 Puncturing patterns for different rates as defined in [3GPP99]

As shown by the above two examples Turbo decoding as defined in different standards have numerous similarities and differences. In multi-standard turbo decoder ACS operation is quite similar, even though the branch metrics feeding the ACS are computed differently, and the trellis structures are different requiring a different feed back from each of the ACS units. Another important issue is the reconfiguration between different constraint lengths and rates in addition to different decoding techniques i.e., Turbo and Viterbi decoding. The power budget of the overall array is also required to be kept minimum in order to keep the design efficient for all of the target standards. The design first phase is the finite precision analysis and is explained below.

4.8 FINITE PRECISION ANALYSIS

The quantization scheme of the array is very important as it determines the size of the storage and the logic blocks in the array. A large word length costs a lot of hardware and hence more area and power consumption however a small word length may result in very poor BER performance. The trade off between hardware complexity and decoding performance has to be reached especially for low power portable wireless applications. Quantization schemes are presented by notation nQf where n is the total number of bits and f is the fractional part. Bit Error Rate (BER)

results with floating point precision in matlab is presented in figure 4.10 below. BER improves by increasing the number of iterations. These improvements are more when iterations are increased initially for example, from iteration 1 to 2 and decreases subsequently for example from iteration 5 to 6. Very little improvement in BER is achieved beyond 6 iterations and therefore the results are presented for a maximum of 6 iterations.

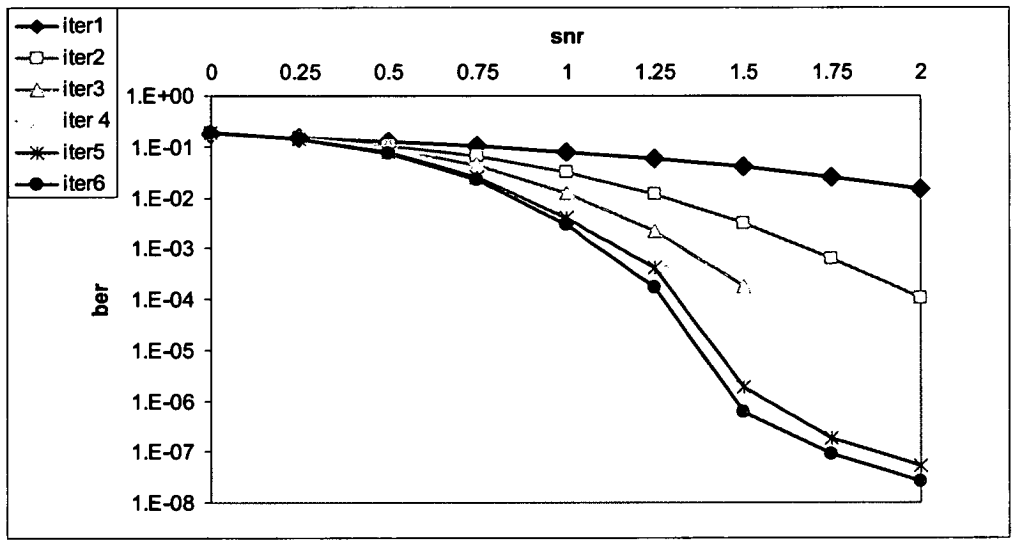

Figure 4.10: Max Log Map BER analysis with floating point precision

Quantization analysis of received bits was done for 3Q1, 3Q2, 4Q2, 4Q1, 4Q3, 5Q2 and 5Q3 schemes. Experiments were performed on a cluster of interconnected unix stations able to perform 1000 parallel matlab executions. The size of the frame was chosen as 1000, and for lower SNR values 1000 parallel executions were performed (total of 10^6) soft input values. For higher SNR, when the error rates drops significantly therefore parallel executions were increased to 1.5 x 10^6 parallel executing frames of 1k to acquire least 200 error samples for averaging. 4Q2 was selected as the best scheme as difference in E_b/N_0 for 5Q2 is ignorable within a wide range of BER, however the difference in 3Q1 is quite significant (>0.3 dB in some range of SNR). This is shown in figure 4.11 below:

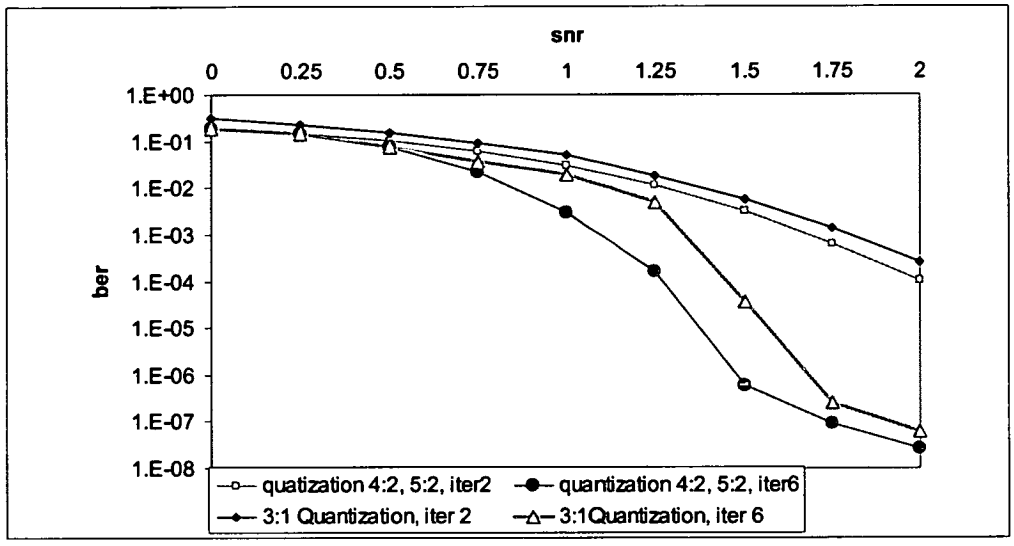

Figure 4.11: Fixed point analysis for input matrics for quantizatios 3:1, 4:2 and 5:2 with 2 and 6 iterations of Max Log Map

Similarly for the extrinsic information the investigated schemes were 5Q2, 5Q1, 6Q2 and 6Q1. 6Q2 turned out to be the optimal choice as shown in figure 4.12.

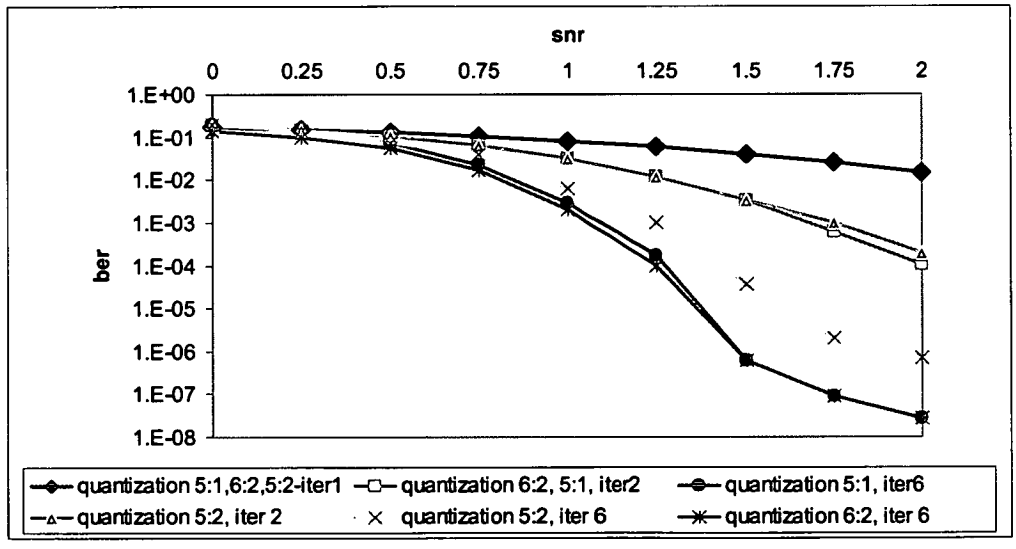

Figure 4.12: Fixed point analysis for extrinsic (apriori) input for quantizatios 5:1, 5:2 and 6:2 with 1, 2 and 6 iterations of Max Log Map

From the input metrics 7 bits are used for branch metrics and 9 bit were established to be sufficient to represent forward and reverse metric calculations. Using this

quantization it was proved for Max Log Map the total quantization loss compared with the floating point precision is no more than 0.1dB.

4.9 SLIDING WINDOW

Both viterbi and turbo decoders use forward and reverse state metrics processing. To improve the latency typically windowed versions of the algorithm are employed for VLSI implementations, largely known as sliding window BCJR algorithm [BEN96]. The basic effect is that the equations will be applied separately to portions (window lengths - WLs) of the global block of data. In its simplistic form the algorithm uses two reverse processors Reverse Processor Dummy B2 and Reverse Processor B1 in parallel with on forward processor FP (shown by ACS0-ACS7 in figure 4.7). These forward recursion unit FP and backward recursion units (B1 and B2) are identical except for the direction of recursion. B2 can start cold in any state (initializing each state as equi – probable) but after a few iterations (equal to WL) the state metrics are as reliable as if the process had been started at the final known correct node of trellis. B2 initializes the start state of B1. The output received is after a delay of 2 WLs. The sliding window technique is therefore a way of reducing the amount of memory required by the Turbo decoder. If the sliding window technique was not used the number of FSM storage would be directly proportional to the block size of the packet being decoded. With the sliding window the number of metrics that need to be stored reduces to the sliding window size. The implementation for sliding window is explained in the subsequent sections.

4.10 DESIGN APPROACH

The algorithm is first implemented in matlab using floating point precision. Then fixed point simulations are carried out using fixed point tool box in matlab. We call these hardwired simulations as the algorithm is written in exactly the same way as will be done in hardware. The hardwired simulations are used to measure the effect of fixed word length on the BER performance. The design methodology for matlab simulations is shown in the figure 4.13 below.

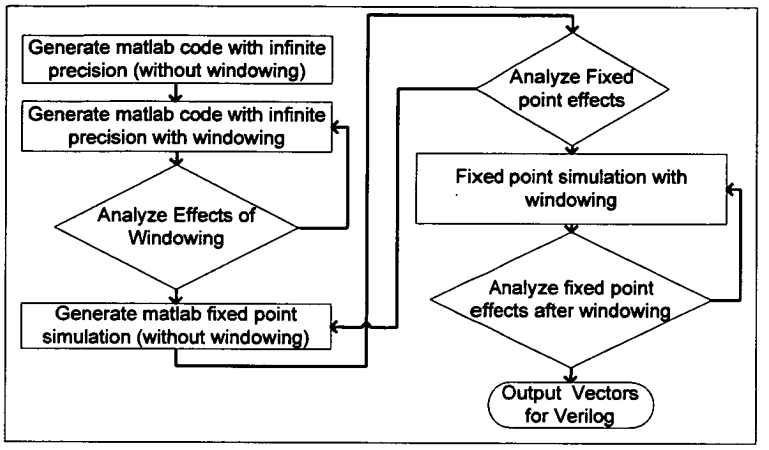

Figure 4.13: Matlab design methodology

As shown in figure 4.13, the matlab design methodology involves translating the algorithm from floating point precision to fixed point simulations. After this analysis the effects of different window lengths are anlysed on BER as the size of window directly determines the size of hardware.

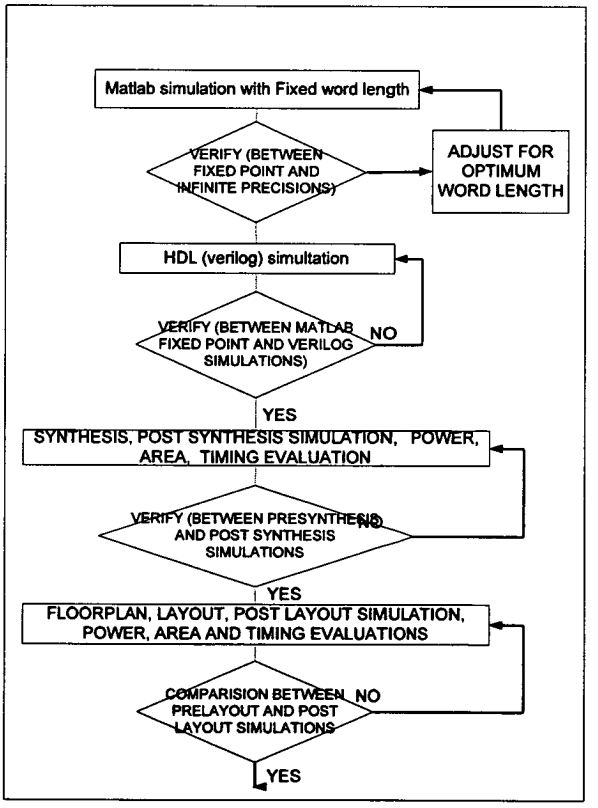

Figure 4.14: ASIC design flow for Max Log Map Implementation

The matlab hardwired simulations generate vectors for the HDL (verilog) and the ASIC design flow used for the design is shown in figure 4.14. The ASIC design flow involves the pre-synthesis analysis and comparison with fixed point simulations in matlab. Design is synthesized on UMC 180nm process technologies to evaluate the area and timing results. Synopsys Prime Time is used for timing evaluations and timing closure. Synopsys prime power is used for power evaluations by noting the toggling activity and back annotating it in to the circuit. Post synthesis verification is done comparing the results with pre synthesis values. Finally, layout and post layout verification is performed using Silicon Ensemble from Cadence.

4.11 VLSI IMPLEMENTATION OF THE TURBO ALGORITHM

The block diagram of turbo decoder is shown in figure 4.7. It can be seen that turbo decoding consists of the following components.

- Input RAMS.
- State Machine Controller.
- Branch Metrics Computation.
- Forward State Metrics and Reverse State Metrics Computations.
- Saturation Hardware.
- Reconfigurable Interconnect for Multiple Turbo Mappings.
- Log Likelihood Ratio Calculator Block.
- Output RAMs.
- Interleaver.

4.11.1 INPUT RAMS

Input RAMs store input metrics for two window lengths (WLs). A novel implementation methodology is adopted to reuse the same input RAMs for both turbo and viterbi with a simple change in finite state machine (FSM) control. The size of the input RAMs is selected such that in viterbi mode these input RAMs can be reused to store the Branch Metrics configuration bits [AHM106]. This is further explained in chapter 5.

The interconnection of state machine controlling the input RAMs in turbo mode is shown in figure 4.15. The read and write control for input RAMs (ramX1n_X2n_A etc) is provided by state machine (shown as FSM). The state machine also controls the multiplexers that provide input data to Branch Metrics Calculators (shown as BMC1-BMC3). Using this design approach change over between viterbi and turbo algorithm is simply done by changing the read/write and the multiplexer controls. The state machine control in Turbo mode is explained in subsequent sections.

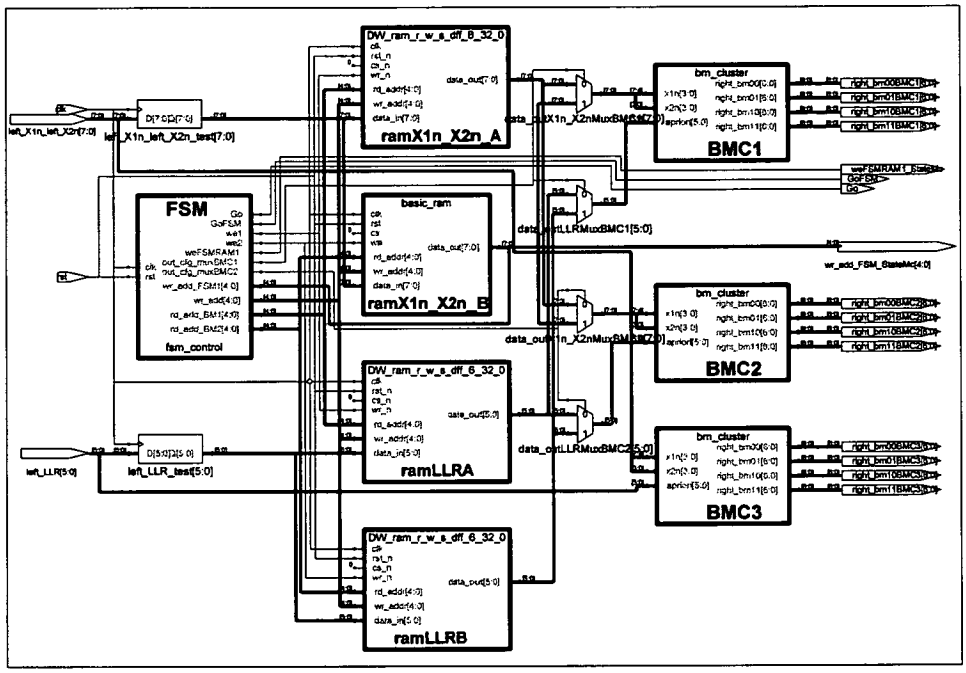

Figure 4.15 FSM control for input RAMs and branch metric calculators

The input RAMs cache in two window lengths of the input metrics which reduces the read access to the larger input RAM saving energy. There are two window lengths required to be saved as per the decoding strategy. We have used Synopsys Designware Write-After-Read (WAR) RAMs to implement two memory architecture compared with three memory architecture proposed in [MAS99]. The proposed two memory architecture employs two memory banks, each storing one window length of the input metrics. We have shown in [AHM106] that in viterbi mode the write and read operation is applied without wasting any clock cycles

resulting in dynamic context switch for multi standard mappings and continuous decoding in turbo mode. The RAMs can be read and written in either forward or reverse direction controlled by state machine.

4.11.2. STATE MACHINE CONTROL AND SCHEDULING ALGORITHM

The turbo decoder state machine controls the read and write operations of input RAMs, Forward State Metric RAMs and the multiplexers. These multiplexers connect the inputs to BMC blocks as shown in figure 4.16. The read/write address is provided by binary up/down counter selected by the state machine. State machine control is responsible for providing the proposed scheduling algorithm. Figure 4.16 shows the read and write controls for input and FSM RAMs. The addresses are generated by forward and reverse counters either one of which can be selected input RAMs. The select signal of this multiplexer (shown as MUX in the figure) is also provided by state machine. The inputs to branch metrics blocks are also provided through multiplexers which are also controlled by the state machine. State machine also provides the read and write control signals for all the RAMs in the deisgn. The counters have a dynamic terminal count flag and the terminal count is kept flexible. This allows different window lengths to be controlled by the same state machine. This controlling scheme is further explained in subsequent sections.

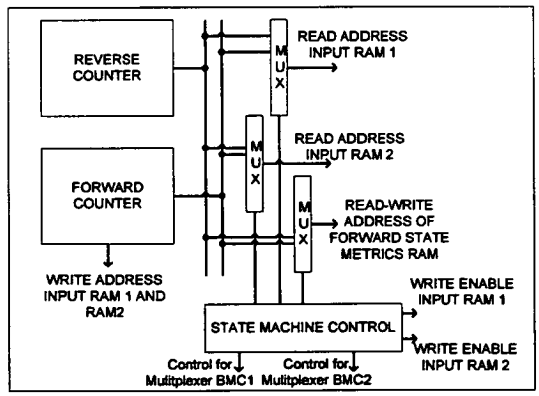

Figure 4.16 VLSI design of State Machine

4.11.2.1 TIME SLOT 0-L (FIGURE 4.17a)

Input metrics corresponding to first window length 0-L are written in Input RAM1. The last metric is saved in first memory location and first metric in last memory location as shown in figure 4.17a.

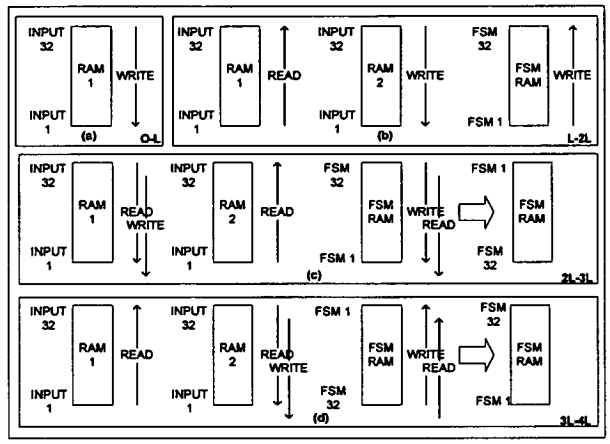

Figure 4.17. Read/Write FSM Control for RAMs

4.11.2.2 TIME SLOT L-2L (FIGURE 4.17b)

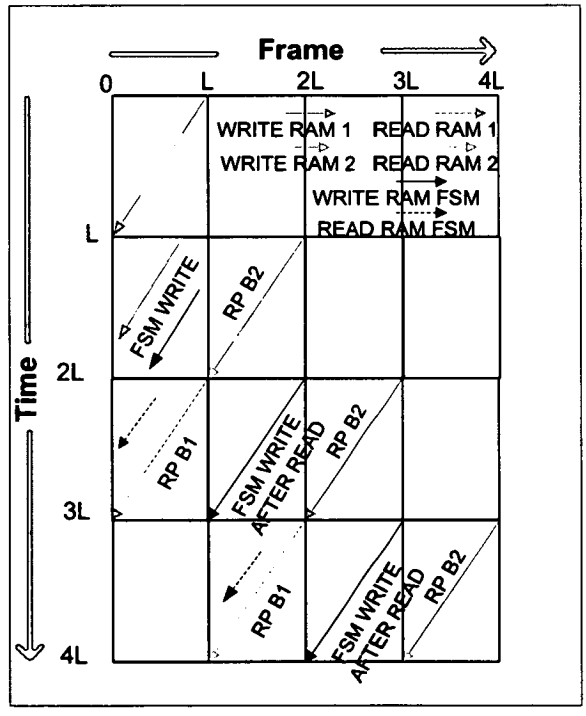

Figure 4.18 Scheduling diagram for max log map implementation

Input metrics corresponding to second WL (L-2L) are written in RAM2. Reverse Processor Beta (RP2) uses these input values to calculate reverse state metrics. Forward processor calculates forward state metrics by reading the RAM1 in reverse direction as shown in figure 4.17b. Calculated forward state metrics are saved in forward state metrics RAMs (FSM RAMs) in the reverse direction i.e., last state metric in first memory location and the first metric in last memory location.

4.11.2.3 TIME SLOT 2L-3L (FIGURE 4.17c)

After the latency of the two WLs, the log likelihood ratio (LLR) values corresponding to WL '0-L' are calculated. LLR calculates the decoded bits by reading FSM RAMs in forward direction as shown in figure 4.17c. Reverse Processor (RP1) is initialized by RP2. RAM1 is read in forward direction to provide input metrics (corresponding to WL 0-L) for reverse processor 1 (RP1) calculations. FP calculation is now performed on WL L-2L, which is done by reading the RAM2 in reverse direction as shown in fig 4.17c. Calculated FSM values are saved in FSM RAM (Write-After-Read). Since FSM RAM was read in forward direction the write will also be performed in forward direction and first FSM value is saved in first memory location and last FSM value saved in last memory location. RAM1 is read for RP2 calculations (corresponding to frame 2L-3L). The input metrics (for frame 2L-3L) are written on RAM1 after the old values are read by RP1. This is shown by solid red arrow in figure 4.18.

4.11.2.4 TIME SLOT 3L-4L (FIGURE 4.17d)

This slot provides the LLR decoded outputs for second WL L-2L. LLR calculator calculates the decoded bits by reading FSM RAMs in reverse direction as shown in figure 4.17d. RAM2 is read in forward direction for RP1 calculation (for window length L-2L). RAM1 is read in reverse direction for FP calculation (for WL, 2L-3L) as shown in fig 4.17d. Calculated forward state metric values are saved in FSM RAM after the read operation. Since FSM RAM was read in reverse direction the write will also be performed in reverse direction and last FSM is saved in first memory location and first FSM saved in last.

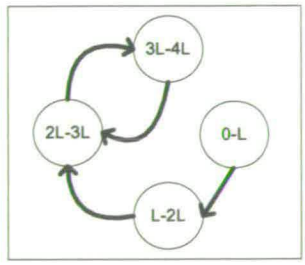

Figure 4.19 FSM control for Max log map implementation

RAM2 is read to calculate RP2 values (corresponding to frame 3L-4L). The cycle repeats after this where time slot 4L-5L is similar to time slot 2L-3L and time slot 5L-6L is similar to time slot 3L-4L. This is shown by the state diagram in figure 4.19.

4.11.3 BRANCH METRICS CALCULATOR (BMC)

Our approach to implementing branch metrics improves on the earlier proposed schemes by eliminating the branch metrics storage [MAS02, BEN96, PIE96, VIT98, ZHO02, YUF00]. These traditional implementation techniques are summarized in figure 4.20.

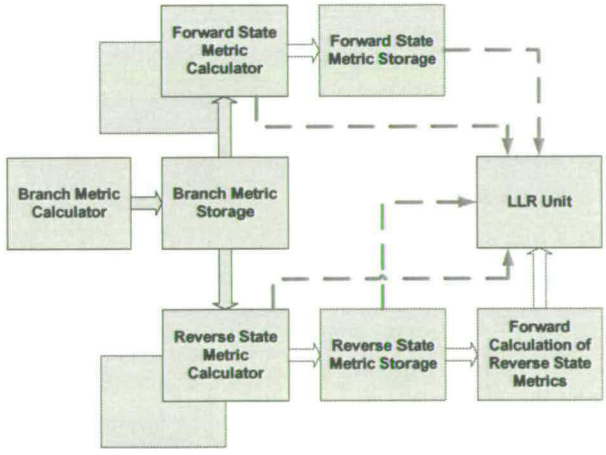

Figure 4.20 Implementation possibilities for max log map

The results shown in [MAS02] for two memory SISO without interleaver partitioning SISO branch metric RAMs (π RAMs) consume 40.2% power for an interleaver length of 5120. The branch metrics storage requirements increase many fold for viterbi

decoding where the number of states are much higher than turbo decoding. Therefore recalculating branch metrics as compared to storage approach improve results in both turbo and viterbi decoding.

BMC1 and BMC2 are connected to input RAMS 1 and 2 by using multiplexers which are controlled by finite state machine (FSM) as per the scheduling algorithm explained in section 4.11.2 and table 4.3. BMC2 which provides branch metrics for RP2 is directly connected to input metrics.

Time Slot	BMC1(FP)	BMC3(RP1)
0-L	RAM2	RAM1
L-2L	RAM1	RAM2
2L-3L	RAM2	RAM1
3L-4L	RAM1	RAM2
4L-5L	RAM2	RAM1

Table 4.3. Input RAMs connections to BMC blocks

Branch metrics are required for each state and stage of the trellis. These are computed by Euclidian distance of the soft input metrics and are given in section 4.4 by equation (26). Our implementation uses one BMC for each forward or reverse processor. Each BMC calculates all the possible branch metrics for a given decoder. It must be noted that the number of branch metrics is equal to 2^n, where n is the denominator in the code rate formula (k/n) - k is the number of input bits and n is the number of output bits. Hence for rate ½, the number of branch metrics will be 4 and 8 for rate 1/3. Similarly if constraint length of the trellis is K, the number of possible states are 2^{K-1} (256 states for 3GPP viterbi). Each ACS node (corresponding to each state) has two branches entering the node and hence each ACS unit needs a specific pair of branch metrics. This selection will differ depending upon the constraint length, coding type, rate, generator polynomial and index of ACS unit in question. For a completely flexible trellis decoder each branch of ACS node should have the ability to select any possible branch metric. For example for a rate ½ encoder, each branch is connected by 4x1 multiplexers that

can select any 1 out of 4 possible branch metrics. Since our design uses 8 ACS units and there are two branches entering each ACS node. Each branch has 8x1 multiplexer (for rate 1/3) therefore there are sixteen 8x1 multiplexer for each stage of turbo decoder trellis. There are 48 configuration bits required to configure the branch metric interconnection. These configuration bits are changed only when a decoder having different parameters is mapped on to the same array. Viterbi decoder requires much larger configuration bits however uses the same ACS unit and multiplexer. This will be explained in next chapter.

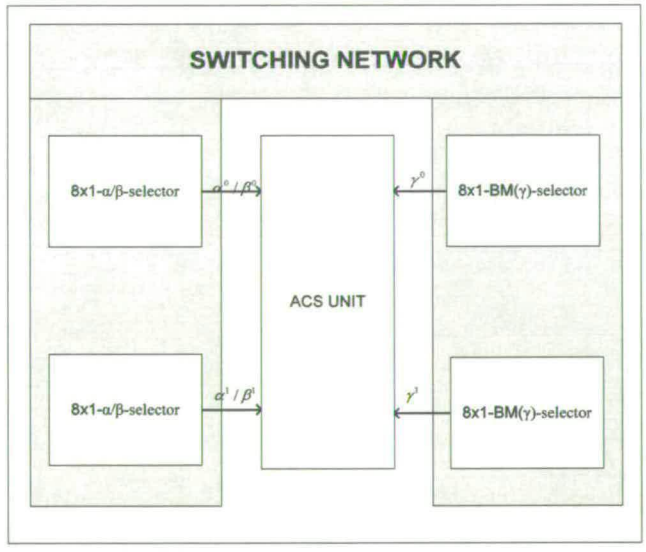

Figure 4.21: Flexible branch metric and state metric connection of ACS unit

4.11.4 FORWARD AND REVERSE PROCESSOR CALCULATION

The main kernel of the Turbo-Viterbi algorithm is ADD-COMPARE-SELECT (ACS) operation which is preformed by each FP, RP1 and RP2 blocks. Since the ACS operation is the only recursive part of the algorithm, the achievable data rate of a VLSI implementation is determined by the computation time of the ACS recursion. Given the branch metrics, the ACS unit calculates the survivor path metrics according to the ACS recursion. These survivor path metrics are again fed back to the ACS units for further processing on next clock cycle (for next stage of trellis). The feedback connection of survivor state to the ACS input is constant for a particular mapped algorithm however changes for decoders with different trellises. Therefore for a fully

flexible trellis decoder, the survivor ACS feedback connections are kept completely flexible by using 8x1 multiplexers for each ACS input. This is shown by figure 4.21. The ACS computation is performed at every state and stage of the trellis. Therefore as we advance in time across different stages of trellis, the same ACS units are reused for processing at each stage. Figure 4.22 shows the architectural diagram of ACS unit which consists of signed two's complement adders (shown as addlower and addupper) for adding branch metrics and forward state metrics. The maximum of the competing paths is seleted (shown by MinMax block) and the adjusted for saturation in sat_block.

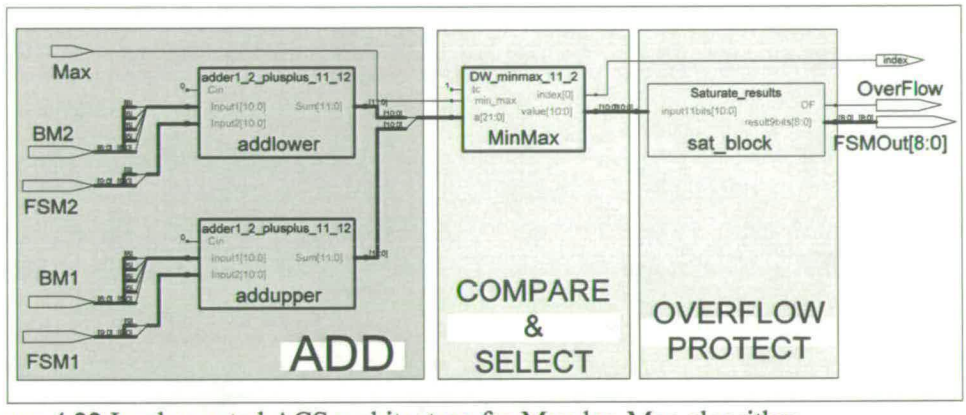

Figure 4.22 Implemented ACS architecture for Max log Map algorithm

FP, RP1 and RP2 have a similar design which consists of 8 parallel ACS blocks and hence 8 states can be processed in parallel. Any decoder that requires greater than 8 states for processing is processed in state serial methodology where 8 states are processed for every clock cycle. For example ADSL (generator matrix [1,17octal/15octal]) and 3GPP turbo mappings on the array work in fully parallel schemes. Fully parallel architectures assign one ACS for each state to meet the performance constraints on speed and latency. In Viterbi mode however since the number of states (N) are higher (256 states for 3GPP) therefore P(P=8) ACS units are used to process N (up to 256) states[AHM106]. Similarly the CCSDS (Consultative Committee for Space Data Systems) turbo decoder family has 16 states which will be decoded 8 states at a time is a similar fashion as GSM Viterbi mappings on Viterbi array as was explained in our work in [AHM106].

4.11.5 NORMALIZATION / SATURATION

State metrics (FSMs and RSMs) are accumulated within a block as they are recursively computed for sliding window ACS computations. Since there is finite number of bits used for storing path metrics, there is a need to normalize the path metrics in order to avoid overflow. The normalization methodology however has to be adjusted for both state serial and state parallel reconfigurable mappings on the array. Therefore a new normalization scheme is adapted to support the mappings that do not use state parallel architectures. These include all viterbi mappings and turbo mappings on the array which have greater than 8 states (for example decoders for CCSDS telemetry operations).

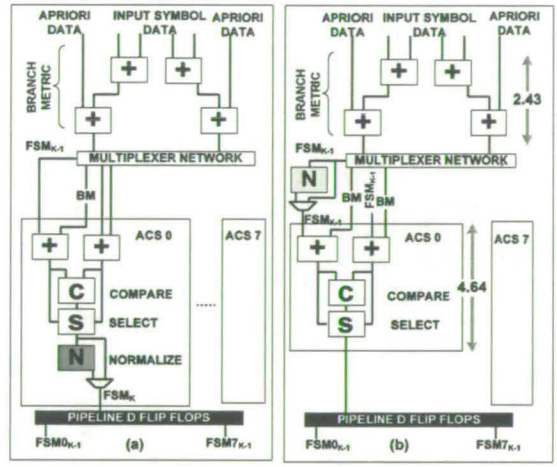

Figure 4.23: Normalization scheme with BM, FSM units for max log map

In this normalization scheme we check at each time instant if any state metric is greater than 2^{q-2}, then a fixed value 2^{q-2} is subtracted from all state metrics. This is shown by normalization (N) block shown in figure 4.20. The block comprises of a subtractor that subtracts a fixed value (2^{q-2}) from state metrics and a multiplexer that selects the subtracted value if the normalization has to be employed. The multiplexer select signal can be provided by any ACS block and in case of state serial architecture mappings (states >8) the select signal is provided after all the states are processed.

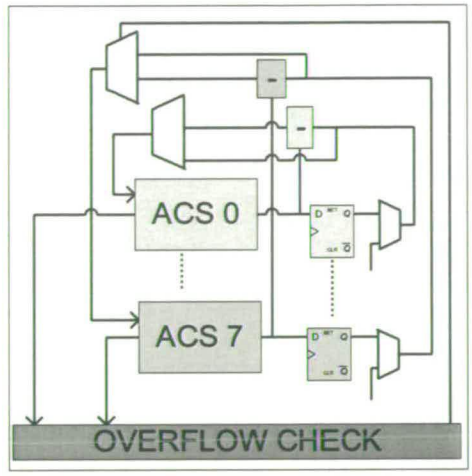

Figure 4.24 New saturation check scheme for max log map

In figure 4.23a the normalized FSMs were saved in the FSM RAM, this new scheme the normalization is applied after reading the state metrics from FSM RAM. The critical path delay of Branch Metric and State Metrics component is shown in figure 4.23 with blue arrows. Note that this adjustment keeps the critical path still exactly the same, however now the same Processor blocks can be used for decoders with states greater than 8. The 8 parallel ACS blocks are shown in figure 4.24.

4.11.6 LOG LIKELIHOOD RATIO (LLR) CALCULATION

As shown in figure 4.25 the LLR block requires the values of forward, backward state metrics and branch metrics. It consists of two identical blocks (block A) calculating the LLR of bit 0 and bit1 respectively. The maximum calculated value of LLR1 and LLR0 is subtracted to find the final LLR output value. The sign of a posteriori value gives the value of decoded bit 1 or 0. LLR block is used in turbo mode only and is disabled in viterbi mode. The LLR block is pipelined to reduce the critical path delay. The position of the pipeline registers is shown by dotted line. Insertion of this pipeline reduces LLR components bottleneck on critical path (delay in ns shown in blue arrows). However ACS still remains in critical path and cannot be further pipelined due to the recursive nature of mapped algorithms. BM, FSM and RSM and LLR blocks shown in figure 4.22 represent the branch metrics, forward state metrics, reverse state metrics and log likelihood values.

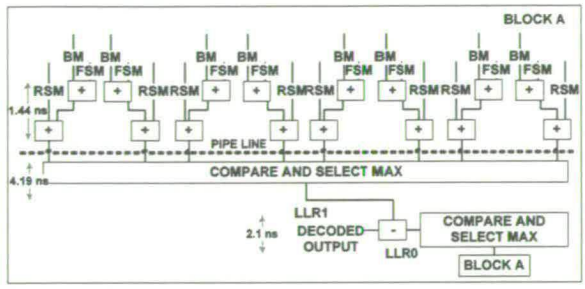

Figure 4.25 LLR Computation Unit of max log map decoder

4.11.7 RECONFIGURABLE INTERCONNECT

The reconfiguration topology for viterbi mappings were explained in chapter 5. A fully flexible trellis processing is used for Turbo decoding as well. This allows mappings of decoder with any generator polynomials. Each branch metrics and FSM connection to ACS block is done through a multiplexer. For example for Rate ½, there are four possible branch metrics that can be connected to each BM branch of ACS block. Similarly for 8 states, there will be 8 possible ACS values that can be fed back to each FSM branch of ACS (refer figure 4.18 for these connections). These flexible connections are provided through multiplexer network as shown in figure 4.7. The multiplexer network is therefore a multiplexer bank providing 4x1 and 8x1 multiplexer connections for each BM and FSM branch of ACS operation of Forward and Reverse processors. Viterbi blocks in the array are shown in white in figure 4.7 and these are clocked down by using an active clocking gating strategy throughout the chip.

4.12 COMPARISON OF RESULTS AND CONTRIBUTION

The design is synthesized using Synopsys Design Compiler for 0.18 microns CMOS UMC cell library and the chip layout is done on Silicon Ensemble. Post layout power figures are taken from Synopsys Design Power by capturing the toggle activity of each node and then back annotating this in the circuit. Synopsys designware SRAMs were used for Forward Processor RAMs. Virtual Silicon 2K x 8 synchronous (separate read and write port) macro RAMs were used for Output/Path history memory consuming 110 uW/MHz/Port.

The overall results are summarized in table 4.4

Technology	UMC 0.18 microns standard cell CMOS
Supported code rates	½,1/3 Turbo, ½,1/3,1/4,1/5 Viterbi
Representation	Signed fixed point
Constraint length	Max 4 in parallel mode and max 9 (256 states) for state sequential (8 states at a time)
Generator Polynomial	Flexible for both turbo and viterbi
Survivor (Trace back length)	Up to 6 times the constraint length
Decision level	4 bit soft decision
A posteriori estimation	6 bit soft decision.
Supply Voltage	1.8V, 1.1 V (for 90nm)
Interleaver	Memory less-supports frames up to 5114 (3GPP)
Max Operating Frequency	84 MHz
Max Throughput Turbo @ 6 iterations (180nm)	41 Mbits / sec / iteration
Latency	2 window lengths
Total area (180nm)	1.67mm^2 (without output RAMs) 2.88mm^2 (with output RAMs)
Total Average Power @ 20 MHz(Turbo mode) 180nm	78.54 mW
Total Average Power @ 100 Mhz (Turbo mode) with 90nm	52 mW

Table 4.4 Results turbo decoder

The components of turbo decoder array contributing in the critical path delay (without LLR pipeline) are: Input Rams, BMC and LLR. The total delay with these components in the critical path will be 15.01 ns. The insertion of LLR pipeline changes the critical path and the components in the path are: Input Rams, BMC and FP.

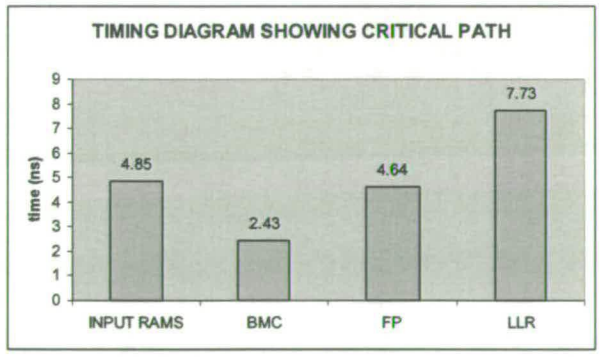

Figure 4.26. Timing diagram for turbo decoder

The overall path delay is now reduced to 11.92 ns. As explained above FP calculations are recursive and cannot be further pipelined, and hence the delay of FP determines the max speed of the design which with the clock period of 4.64 ns. This can be further improved if CLA adders are used instead of Full Adders in FP blocks. The critical path and the delay of the individual components is shown in figure 4.26.

The overall area and power results of individual components can be compared in figure 4.27.

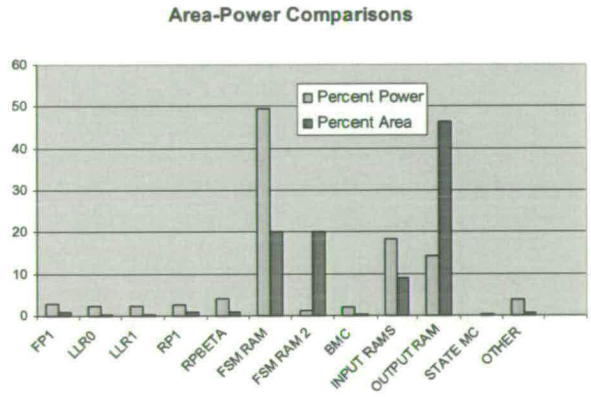

Figure 4.27. Area-Power distributions of individual components of the designed turbo decoder

For 180nm, 82% of the total average power of 78.54 mW is due to net switching and 18% is the cell internal power. The cell leakage power is 221.4798 μW. For 90nm process technology the results are shown in figure 4.28. The results provided in figure 4.28 on 90nm process technologies show that Cell Internal Power also contributes significantly along with the net switching power and the design should be optimized for both.

Input/Output RAMs, FSM RAMs and Input Scratch Pad RAMs when combined together occupy 95.6 % of the overall area and 83.4% of the power.

Figure 4.29 gives the overall area results with 90nm process showing total area, cell count and timing figures. The over all area is reduced to 0.84 mm^2 with a critical path of 9.8 ns (100MHz).

```
Global Operating Voltage = 1.1
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000ff
    Time Units = 1ns
    Dynamic Power Units = 1uW    (derived from V,C,T units)
    Leakage Power Units = 1uW


  Cell Internal Power  =  31.1673 mW   (59%)
  Net Switching Power  =  21.7252 mW   (41%)
                         ---------
Total Dynamic Power    =  52.8925 mW  (100%)

Cell Leakage Power     = 112.7234 uW
```

Figure 4.28. Ratio of the net switching power and cell internal power in 90nm CMOS process

```
Timing Path Group 'clk'
  -----------------------------------
  Levels of Logic:              56.00
  Critical Path Length:          9.80
  Critical Path Slack:           0.01
  Critical Path Clk Period:     10.00
  Total Negative Slack:          0.00
  No. of Violating Paths:        0.00
  -----------------------------------

  Cell Count
  -----------------------------------
  Hierarchial Cell Count:         389
  Hierarchial Port Count:       17768
  Leaf Cell Count:              23952
  -----------------------------------

  Area
  -----------------------------------
  Combinational Area:    79205.951585
  Noncombinational Area:
                        429117.518607
  Net Area:             340051.718750
  -----------------------------------
  Cell Area:            508323.468750
  Design Area:          848375.181088
```

Figure 4.29. Critical path delay, Area and total cell count in 90nm CMOS process

The turbo decoder design can be compared in the following broad reconfigurable categories:

- ASIP (application specific instruction set processor).

- Implementations on general purpose processors.
- Implementation on processors with Viterbi/turbo decoders as co processors.
- Implementations on FPGAs.
- Implementation on ASICs.

4.12.1 ASIP

We have provided a detailed review of ASIPs in chapter 2 section 3.2.3. In ASIPs, flexibility is provided by the use of embedded processors specifically targeted to the decoding application. ASIP being software controlled is broader in domains of reconfigurability and hence more flexible than our design. The increase in flexibility, however degrades power, area and speed. For example in [ROS04] there would be 8 XiRisc processors needed in order to achieve a throughput of 2Mbps. The processors are also required to run in parallel on successive blocks of data.

4.12.2 GENERAL PURPOSE PROCESSORS

General purpose processors are much more flexible than ASIP and our proposed array. However turbo decoder implementation on these results in a much reduced throughput. Since the throughput falls below the data rates as required by the standards, this favours the case for designing ASIC for Turbo decoding either as a coprocessor or as separate reconfigurable array. Table 4.5 below lists some published Turbo decoder implementations on general purpose processors and also lists the maximum throughput possible. Throughput results are worse than ASIP. However flexibility in general purpose processors is much higher.

Processor	Clock Speed	Throughput possible	Ref
Motorola 56603 DSP	Not quoted	48.6 kbps/iteration	[WOR 02]
ST120	200MHz	540 kbps/iteration	[WOR02]
Intel Pentium III	933 MHz	262 kbps/iteration	[VAL01]
DSP SP-5 SIMD	Not quoted	227kbps/iteration	[HAR01]

Table 4.5. Performance of turbo decoders of different architectures in literature

4.12.3 GENERAL PURPOSE RECONFIGURABLE LOGIC (FPGAs)

Dedicated implementations on general reconfigurable logic for example FPGAs can achieve higher throughput. However, it consumes higher power than ASIC implementation and our proposed implementation. For example, in [SHA03] implementation on Xilinx Vitex XCV300E (almost 50% resource utilization) consumes 695mW(25 MHz) for 1Mbps.

4.12.4 ASIC

A more exact ASIC comparison of our IP can be made with the work in [MAR02], where we have achieved similar area and power figures, however the reported array is limited to unified Viterbi and Turbo decoding for 3GPP where data rates of the order of 2Mbps are required. Also the Viterbi decoder is mainly aimed at low data rate (12Kbps) voice channels. The ACS processing is done by 4 Viterbi/log-MAP butterfly unit, which is sufficient for 3G data rates, but cannot support the extremely high data rates demanded by systems like WLAN. Therefore such a system might be useful for a certain specific standards like 3G, but is hardly useful when multiple standards/systems are in question. Our design is more flexible as it can target multiple standards both in viterbi and turbo mode.

4.12.5 TURBO / VITERBI CO PROCESSOR ACCELERATORS

Texas instrument (TI) latest TMS320C6414 fixed point DSP processor [TMS05] uses Viterbi and Turbo decoder coprocessors as two independent operating blocks. The parameters in the blocks can be adjusted to provide flexibility. The exact implementation details for this commercial processor are not disclosed. However it uses the processor bus and input data is quantized to 8 bits (8N3), which gives some indication into the size of blocks used for decoding [TMS04]. The flexibility in the case of turbo decoding is exactly similar to our design. However, our proposed design differs as it provides reconfiguration between viterbi and turbo as one unified design. This results in better reuse of hardware blocks.

Our design is also more optimized for word lengths. The biggest disadvantage in [TMS04] is the power figures of this design. The power though depends on type of application mapped however is in 1000s of mW for any mapped application

[TMS104]. The individual power and area figures of the design are presented in figure 4.22. In turbo mode the decoder consumes 78.5 mW occupying 2.824 mm^2.

4.13 CONCLUSION

In this chapter the subject of Turbo codes is introduced along with the mathematical description of the various turbo decoding algorithms, the use of turbo decoders in various standards and its implementation. The implementation has focused on reconfigurable aspects of the design in the most efficient way. A comparison is done with the existing reconfigurable designs. The chapter has shown that design has both more processing power than general purpose processors, DSPs and FPGAs and more flexible than ASICs. The flexibility is carefully designed to keep it with in the power, area and timing budgets of the mapped standards. Being able to reconfigure the design means that new standards or upgrades to the standards can be implemented remotely potentially saving the network provider money. This avoids any re spun on the design avoiding the large non-recurring engineering costs in the process.

The reconfigurable VLSI implementation was also covered with the design flow of algorithm from floating point precision in matlab to layout. The chapter highlighted the performance gains of such design and revealed the results which also verified the commercial relevance of the current design compared to the state of the art.

Chapter 5

RECONFIGURABLE VITERBI DECODING

5.1 INTRODUCTION

In this chapter we choose to solve the problem of reconfigurable viterbi decoding in the context of a common communication platform consisting of unified turbo-viterbi decoding components. The viterbi algorithm describes a very well known technique for decoding convolutional codes and is an essential part of WCDMA, WLAN, GSM, CDMA2000, ADSL and many other standards. Our goal under this segment of research is to design a viterbi decoder that allows reconfiguration between all of the above mentioned standards. In addition, the design should also achieve maximum resource allocation and performance by reusing components within turbo decoder base array. The reconfiguration within viterbi decoding also provides flexibility to reconfigure the array for different trellis types, constraint lengths, rates, generator polynomials and frame sizes. This makes the decoder ideal for a unified multi-standard telecommunication platform where each standard can map its own parameters on to the viterbi array. In addition a novel dynamic reconfiguration technique is also proposed in order to achieve faster context switching between different mappings. The reconfigurable fabric is implemented as a subset of turbo decoder array on a 180 nm UMC process technology.

There has been prior research in the field of convolution codes and viterbi decoding in general but little work has been done on unified turbo-viterbi design for all the chosen standards in the current research. There is an increasing demand of high speed and low energy in these standards which has to be met by any reconfigurable design that targets them. It was shown with results that the implemented design gives much better results in terms of power, area, speed and timing compared to

existing reconfigurable DSP or FPGA based implementations. The proposed architecture is capable of supporting data rate requirement for 802.11a WLAN and can switch between constraint lengths 3-9. The design also allows different generator polynomials and hence can have mappings of different trellis types. Another novelty in the implementation is to constraint the Viterbi design such that design space is restricted to reuse as much as possible the components of the existing turbo decoder array. We have solved the problem of reconfigurable viterbi communication hardware design in three parts. First, understanding and implementing the algorithm in fixed point format in matlab and evaluating the results (Hardwired Simulation). Secondly identifying the commonality of algorithms in the standards in question and introducing the desired flexibility to satisfy the imposed performance constraints. The identification stage is followed by evaluation stage for BER performance analysis for the chosen word length. The last stage is the reconfigurable VLSI implementation, comparison with hardwired simulation and evaluation of results power, area and speed results.

The rest of this chapter is organized as follows. In section 5.2, convolution coding and Viterbi decoding is briefly described. Section 5.3 describes the reconfigurable domain where we look at specific codes that are used in different standards. In section 5.3 the hardwired simulations of viterbi algorithm are shown. Section 5.4 describes the new reconfigurable viterbi design and the details of the individual components. In section 5.5, simulation results and performance measures for this implementation are provided and compared with the state of the art.

5.2 VITERBI ALGORITHM

Viterbi algorithm was invented to overcome several fundamental drawbacks in Block Codes. First of all block codes are frame oriented which introduces intolerable latency into the decoding system. A Block code also requires frame synchronization and due to hard decisions, the coding gain is also limited. A convolution code [ELI55] improves on block codes by passing the information

sequence to be transmitted through a linear finite-state shift register. Figure 5.1 shows a typical rate 1/2 convolution decoder for cdma 2000.

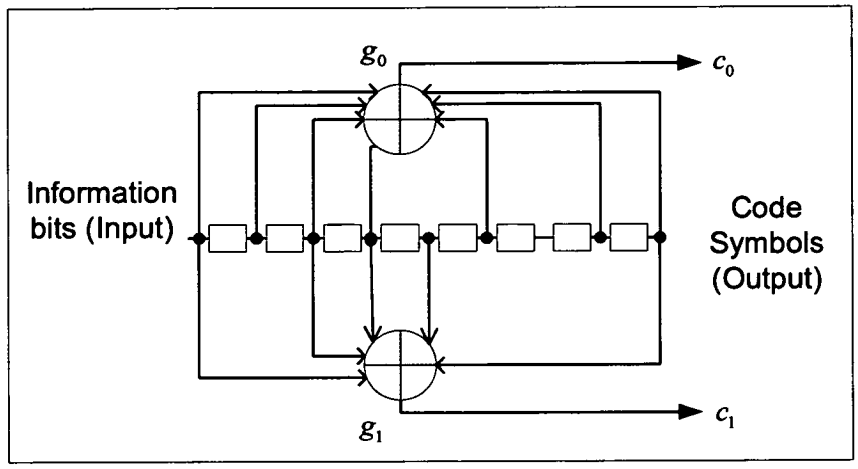

Figure 5.1 K=9, Rate ½ Convolutional Encoder for CDMA 2000

The rate of this encoder is determined by the ratio of the input to output bits. For example in the figure above the encoder outputs two bits for every input bit. In general, an encoder with k inputs and n outputs is said to have rate k/n. The binary data is fed into a series of shift registers (memory elements) and output is taken from the generators (g_0 and g_1). This pattern of taps of the shift register determines the generator polynomial of the encoder. The generator polynomial in figure 5.1 shall be g_0 equals 753(octal) and g_1 equals 561 (octal). The constraint length parameter K (capital) denotes the "length" of the convolutional encoder (n) i.e., how many bits are available to feed the generators (XOR gates) that produce the output symbols. K is always equal to the number of memory elements (shift registers) plus one. For 3GPP encoder trellis shown in figure 5.1, K is 9. The contents of the shift register determine the state of the encoder. Therefore the number of states of the encoder will be 2^{K-1}. The behaviour of the encoder for an input sequence can also be viewed as finite state machines represented by state diagrams, graphs or trellises. The state diagram can be expanded into a trellis diagram which explicitly shows state transitions at each time instant. Any

convolution code can thus be uniquely defined in terms of its rate, constraint length and generator polynomial.

The Viterbi decoding algorithm [VIT67, FOR73, VIT79] is the maximum likelihood decoding algorithm for convolution codes. It finds the most-likely noiseless state transition sequence of symbols in a state diagram, given a sequence of symbols that are corrupted by noise such as additive white Gaussian noise.

5.3 MATHEMATICAL DESCRIPTION

For coherence of presentation the viterbi algorithm is briefly summarized here. The interested readers are referred to [FOR73]. For simplicity all notations are kept exactly the same as described in [FOR73]. Given a sequence 'z' of observations of a discrete time finite state Markov process in memoryless noise, viterbi algorithm finds the state sequence 'x', for which the a-posteriori probability P(x | z) is maximum. This is called MAP (maximum a-posteriori probability) rule, which minimizes the error probability in detecting the whole sequence of message. The maximum likelihood ML decoder selects by definition, the estimate that maximizes P(z |x). In general they can be related by Bayes' rule

$$\Pr(A \mid B) = \frac{\Pr(A \cap B)}{\Pr(A)} \tag{5.1}$$

$$\Pr(B \mid A) = \frac{\Pr(A \cap B)}{\Pr(A)} \tag{5.2}$$

Rearranging and combining equation (5.1) and equation (5.2)

$\Pr(A|B)\Pr(B) = \Pr(A \cap B) = \Pr(B|A)\Pr(A)$

$$P(x,z) = P(x \mid z)P(z) = P(x)P(z|x). \tag{5.3}$$

The process is Markov because the probability $P(x_{k+1} \mid x_0, x_1, \ldots, x_k) = P(x_{k+1} \mid x_k)$ Let the transition ξ_k at time k be defined as the pair of states (x_{k+1}, x_k). The input sequence $u = (u_0, u_1, \ldots)$, where each u_k can take on one of the finite number of values say 'm' (m=2, for values of 0,1). There is a noise-free signal sequence y, in which each y_k is some deterministic function of present and the 'v' previous states.

$$y_k = f(u_k, \ldots, u_{k-v})$$

The observed sequence z is the output of the memoryless channel whose input is y. The channel is memoryless in the sense that the noise effecting one bit in the received word z_k is independent of the noise process affecting all other bits. The process described above can be modelled by a shift register of length v with inputs u_k. The number of states is thus $|X| = m^v$. For example; In figure 5.1 there are 2^8=256 states. The total number of transitions = m^{v+1} (for example there are 2^9 = 512 transitions for each k in figure 5.1).

Also note that viterbi algorithm finds the transition sequence ξ for which $P(\xi|z)$ is maximum (since x has one to one relationship with ξ), which is same as finding the most probable input sequence u, since u has one to one relationship with x. MAP sequence estimation problem is formally identical to the problem of finding the shortest route through a certain graph as every possible state sequence x there corresponds a unique path through the trellis.

Since the process is observed in memoryless noise; that is, there is a sequence of observations z_k in which z_k depends probabilistically only on the transition (or x_k) at time k (where the process runs from 0 to time K-1)

$$P(z|x) = \prod_{k=0}^{K-1} P(z_k \mid x_k). \tag{5.4}$$

For higher rates (bits within the blocks) we add the notion of superscripts.

$$P(z|x) = \prod_{k=0}^{K-1} \prod_{j=0}^{n-1} P(z_k^j \mid x_k^j). \tag{5.5}$$

Equation (5.5) is called the likelihood function for x. Since logarithms are monotonically increasing, the estimate that maximizes P(z|x) is also the estimate that maximizes log P(z|x). By taking the logarithm of each side of the equation, the log likelihood equation is obtained i.e.,

$$\log P(z|x) = \sum_{k=0}^{K-1} \sum_{j=0}^{n-1} \log P(z_k^j \mid x_k^j). \tag{5.6}$$

In implementations of the Viterbi decoder, the summands in equation 5.6 are usually converted to a more easily manipulated form called "bit metrics" which can

be denoted as a function $M(z_k^j / x_k^j)$. The path metric for code word x is then computed as follows

$$M(z/x) = \sum_{k=0}^{K-1} \sum_{j=0}^{n-1} M(z_k^j / x_k^j). \tag{5.7}$$

Also, the path metric can be expressed as the sum of branch metrics where the k^{th} branch metric is defined as the sum of the bit metrics for the k^{th} block of z given x,

$$M(z_k \mid x_k) = \sum_{j=0}^{n-1} M(z_k^j \mid x_k^j). \tag{5.8}$$

The kth partial path metric is thus obtained by summing the branch metrics for the first k branches that the path traverses,

$$M^k(z \mid x) = \sum_{i=0}^{k-1} M(z_i \mid x_i).$$

Since there are 2^m branches entering each node, we have to choose the "best" partial path metric among the metrics for all entering paths. The path with the best metric is the survivor, while the other path/paths are discarded. The best partial path metric might be either the largest or smallest, depending on how the bit metric calculation is done. The distance to be measured can be either Hamming distance or Euclidean distance. As explained above the survivor path for a given state is the path which has the minimum distance from the received sequence of noisy symbols. Using Euclidean distance compared to Hamming distance makes VA optimal [LOU95]. This calculation is explained below.

5.3.1 EUCLIDEAN METRIC COMPUTATION

For soft decision viterbi decoders the squared Euclidean distance is used as a metric to measure the distance between the received and the actual symbols. This is explained with an example below.

Consider a rate ½ Viterbi Encoder which generates 2 bits (-1-1, -1+1, +1-1, +1+1) for every input bit 1 or 0. Each received symbol y_k may be represented in vector form as $\vec{y}_k = \{r_0, r_1\}$, where r_0 and r_1 are soft decision values, whose magnitudes

determine the reliability of the received vector $\vec{y}_k$. Every symbol in the transmitted code alphabet may likewise be represented by the vector $\vec{x}_k = \{\pm 1, \pm 1\}$. The computation of the Euclidean distance metric is:

Distance = $|y_k - x_k|^2$

$$= |y_k|^2 - 2(y_k . x_k) + |x_k|^2 \quad (5.9)$$

The energy of the symbol $\vec{x}_k = \{\pm 1, \pm 1\}$ may be computed as

$$|x_k|^2 = (\pm 1)^2 + (\pm 1)^2 = 2 \quad (5.10)$$

This energy of all symbols in equation (5.9) is constant at normalized value of 2. For this example case, there are two paths that merge at each node of the trellis. Selecting the survivor is thus equivalent to comparing two distance equations (equation 5.9) D_0 and D_1 for these paths. Let the transmitted code alphabet for these two paths of the trellis be x_k^0 and x_k^1.

Therefore:

$$D_0 = |y_k|^2 - 2(y_k . x_k^0) + |x_k^0|^2 \quad (5.11)$$

and

$$D_1 = |y_k|^2 - 2(y_k . x_k^1) + |x_k^1|^2 \quad (5.12)$$

We know that Viterbi algorithm selects the minimum of the distance between the competing paths.

But, from equation 5.10 we know that the energy of x_k is constant (equal to normalized value of 2) and therefore will be constant in minimum calculation of equation 5.11 and equation (5.12). Similarly the energy of y_k is the same in both the cases. This reduces the comparison operation to a minima function between the middle two dot product terms as is shown here under:

$$\min((-2(y_k . x_k^0)), (-2(y_k . x_k^1))) \quad (5.13)$$

Since a min operator on negative numbers may be interpreted as an equivalent max operator on positive numbers. Therefore equation (5.13) becomes:

$$\max((y_k . x_k^0), (y_k . x_k^1)) \quad (5.14)$$

Each dot product term may be expanded as

$=\max((\{r_0, r_1\}.\{\pm1, \pm1\}), (\{r_0, r_1\}.\{\pm1, \pm1\}))$

$=\max(\pm r_0 \pm r_1, \pm r_0 \pm r_1)$ (5.15)

where the signs of each term depend on the transmitted symbol for the branch being compared. Thus the squared Euclidean metric distance calculation to compute the branch metric may be performed with a simple add/subtract and comparison operation. The path metrics at stage k+1 of the connected node of the trellis is calculated by adding the survivor in equation 5.15 with the calculation of equation 5.14 for stage k+1.

5.4 RECONFIGURABLE VITERBI DOMAIN

The major contribution of the work on viterbi decoder is the reconfigurable viterbi VLSI implementation that can target multiple standards and can be reconfigured to decode a range of convolutionally coded data. The architecture can support up to 256 states, trellises with different generator polynomials and rates. The design is mapped onto the turbo decoder array components and these combined together provide forward error correction for a reconfigurable communication platform. The flexibility of the reported design is carefully tailored to reduce the granularity to restricted domains as compared to general purpose fine grained gate arrays. This flexibility trade off provides the desired improved performance in terms of speed, area and power. A Viterbi decoder is an important subsystem of any wireless communication receiver. Each standard, however, defines different encoding parameters for Viterbi Forward Error Correction. The examples of these variations are explained below:

5.4.1 GSM/GPRS

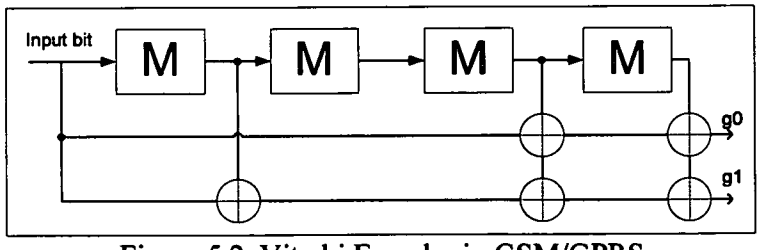

Figure 5.2. Viterbi Encoder in GSM/GPRS

The Viterbi Encoder defined in 2nd generation GSM and GPRS standards is shown figure 5.2. It is rate ½, K = 5 encoder with generator polynomials g0 and g1 as 23, 33 respectively.

5.4.2 3GPP2 (WCDMA, CDMA-2000)

The 3GPP2 (Table 2.1.3.1.5-1 and Table 2.1.3.1.5-2 in [3GPP99]) standard defines convolution codes for channels with Spreading Rate 1 and Spreading Rate 3. All convolution codes have a constraint length K of 9. The generator function for rate ½ is g0 equals 753 (octal) and g1 equals 561 (octal). The symbol c1 is output first and the code symbol c1 is output last. The state of the convolution encoder upon initialization is the all-zero state. The encoder for this code is illustrated in figure 5.1. The generator function for rate 1/3 are g0 equals 557 (octal), g1 equals 663(octal) and g2 equals 711 (octal). The generator function for rate ¼ are g0 equals 765 (octal), g1 equals 671(octal), g2 equals 513(octal) and g3 equals 473 (octal).

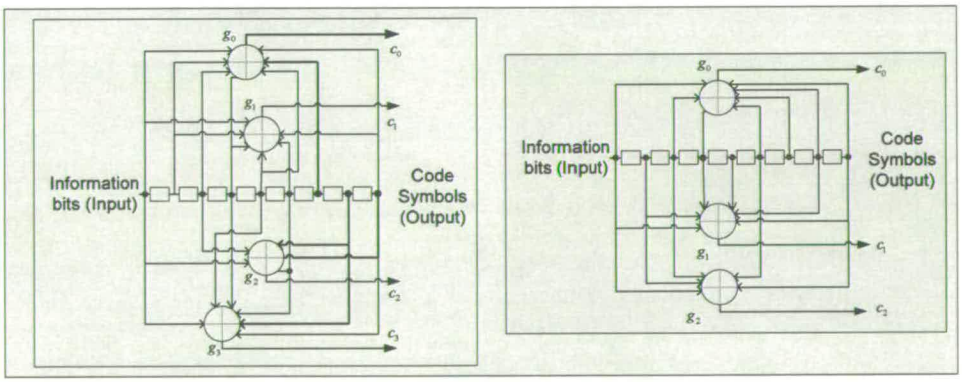

Figure 5.3 Rate ½ an 1/3 Convolution encoders for 3GPP [3GPP99]

5.4.3 WLAN 802.11a AND METROPOLITAN AREA NETWORK IEEE 802.16

These standards use rate ½ constraint length K=7, binary convolution code with the generator polynomials for c0 as 171(octal) and for c1 as 133(octal). Puncturing the rate ½ allows higher rates of 2/3, ¾, 5/6, and 7/8 for IEEE 802.16 and 1/2, 2/3, or ¾

for IEEE 802.11a. Puncturing is the procedure for omitting some of the encoded bits in the transmitter (thus reducing the number of transmitted bits and increasing the coding rate) and inserting a dummy "zero" metric into the convolution decoder on the receive side in place of the omitted bits. The puncturing patterns are defined in the corresponding standards.

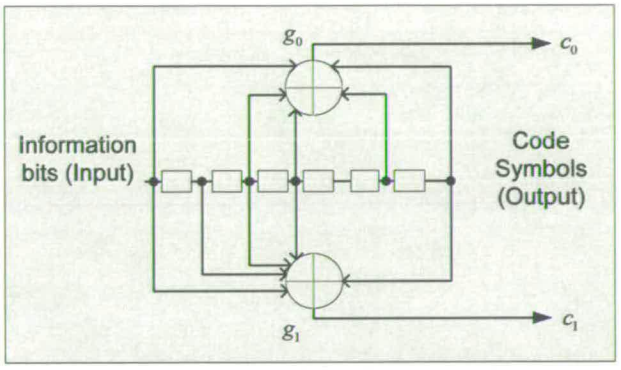

Figure 5.4 Rate ½ Convolution encoder for WLAN and IEEE 802.15

As shown by above examples Viterbi encoders defined in different standards have a number of differences and similarities. The VLSI design of Viterbi Decoder exploits this commonality for a unified Viterbi array for communication platforms.

5.5 HARDWIRED SIMULATIONS

The focus of these simulations is to build a modular and flexible simulation model that can be used for a variety of advanced communication systems. The algorithm is implemented first in floating point model. This is followed by a migration to the fixed point equivalent model. The fixed point model is emulation of hardware in matlab and provides a quick path for evaluating the BER performance for justification of the fixed point decisions. The model bridges the gap between system design and hardware implementation. The hardwired simulation model also provides verification environment at a higher level of abstraction and is used to verify the code written in HDL.

Figure 5.5 below gives the BER results for AWGN channel rate ½ viterbi decoder of GSM, GPRS and 3GPP wireless standards. It was observed that 4 level

quantization in GSM, GPRS decoder is 0.5-1 dBs inferior to 8 level quantization at various SNRs. Increasing the quantization to 16 levels (4bits) allows further improvement of 0.25 dBs with the results closest to floating point precision. The trace back length in all the simulations was kept at 6 times the constraint length of the code.

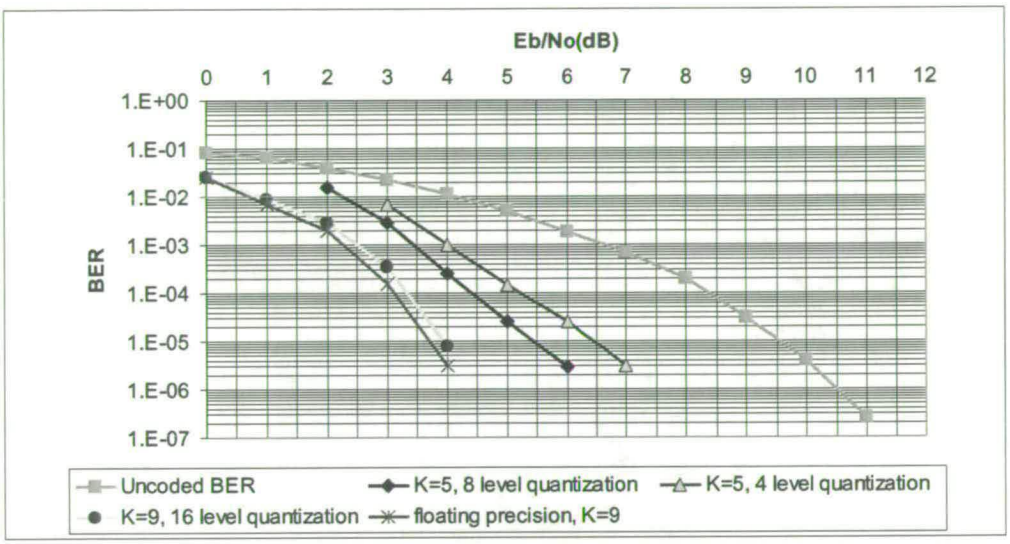

Figure 5.5: Fixed point analysis for rate ½ Viterbi decoding in AWGN channel for GSM, GPRS and 3GPP

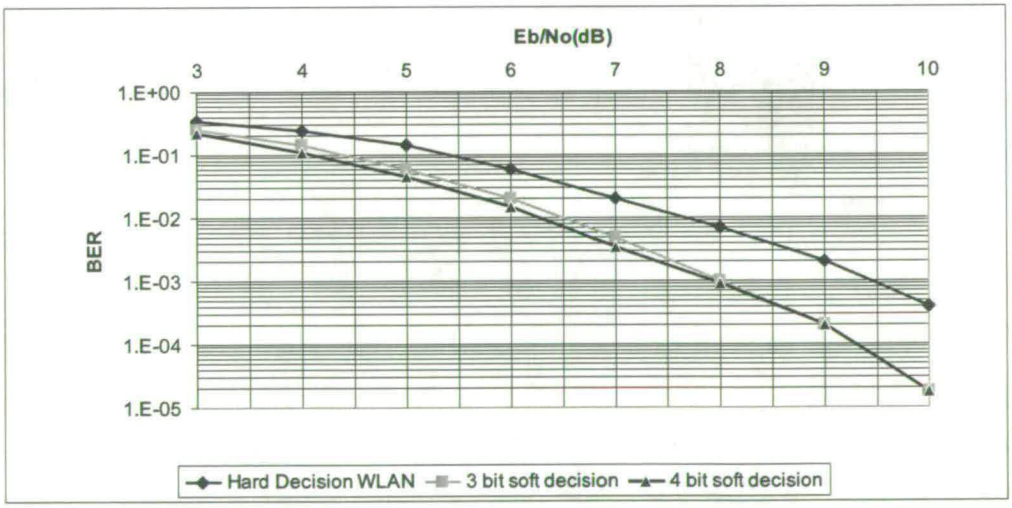

Figure 5.6: BER Results for rate ½ soft decision viterbi decoder in multipath channel for WLAN 802.11a and 802.16.

Figure 5.6 shows the hardwired fixed point simulations for WLAN (802.11a) and Metropolitan area network 802.16. The results are in multipath channel with BPSK modulation and rate ½ viterbi hard and soft decision decoding. 3 bit soft decoding gives up to 1.7 dB improvement in BER compared to hard decision decoding. 16 level soft decoding improve the gains by another 0.2 dB.

A quantization level greater than 4 bits does not provide any significant improvement in BER. Therefore an input quantization of 4 bits (as in Turbo decoder) provides the best compromise between performance and hardware in a unified turbo-viterbi architecture.

The fixed point model is written in Matlab [MAT01] using fixed point tool box and subsequently in the VLSI design all word level quantizations are kept exactly the same as Matlab design. These Matlab simulations provide model by model test vectors for VLSI implementation at various levels of hierarchy as was shown in section 4.10 earlier. The matlab pseudo code is shown on next page.

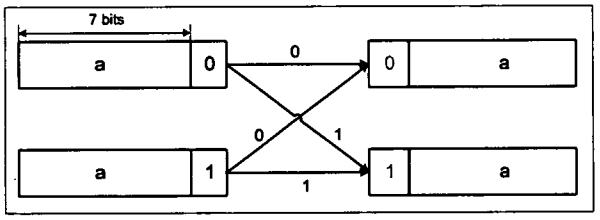

Figure 5.7 Next and Previous state calculation for all trellises

5.6 VLSI IMPLEMENTATION OF THE VITERBI ALGORITHM

The block diagram of the overall array is shown in figure 4.7 with Viterbi decoder components highlighted in yellow and white colour. The overall Viterbi decoder consists of the following major components.

- ACS Blocks.
- Path Metrics Memory.
- Path History Memory.

- Reconfigurable write address generator block.
- State machine.
- Reconfigurable Trace Back Processing blocks.
- Input RAMs.

5.6.1 ACS BLOCK

ACS block is computationally one of the most intensive units in a Viterbi decoder. This block uses the accumulated metrics (called forward state metrics FSMs) and the current branch metrics to compute the survivor path metrics at each node in the trellis. The basic functions performed by this ACS block are add, compare and select.

Formulate Random Input Frame (lines 1-7).
Perform the convolution encoding as per the Trellis (poly2trellis function: line 9).
Separate Systematic and Parity bits (lines 13-21).
Model the Channel for SNR by adding noise to the encoded frame (lines 33-42).
Define the fixed point format for Branch Metrics, FSMs, RSMs etc (lines 48-68).
Initialize the first state with Maximum FSM Metric for first stage of Trellis (k=0) (line 75),
Initialize the start of Dummy Reverse Processor (line 79) and Reverse Processor beta (line 80),
Initialize total windows (line 81).
Calculate the Branch Metrics, Forward State Metrics in the forward recursion as under:
***Loop Outer:** Iteration for input message frame*
***Loop Middle:** Iteration for all states in each stage (k) for trellis. Since there are 8 parallel ACS blocks in hardware that process 8 states, therefore the no of state count = total number of states/8*
***Loop Inner:** Iteration for processing 8 states as per hardware*
Calculate for each state at stage 'k' of the trellis the states at stage 'k-1' that connect to this state. (Using function find_connected_states line 89)
Save configuration bits for Branch metrics multiplexers in hardware (line 95).
Calculate Branch Metrics for the connected states at stage k from stage k-1 of trellis (lines 98-144)
Calculate Forward State Metrics for these connected states i.e. FSM0 and FSM1 (lines 153-154)
Find the maximum of the two FSMs (line 160).
Corresponding to the winning FSM save 1or 0 (0 if upper branch was the winning state: lines 170-174).
Update contents of path memory (lines 175-179)
Return Loop Inner.
Return Loop Middle.
Model Saturation logic as per hardware design i.e., Subtract a fixed value from all FSMs if any FSM crosses a threshold: lines 182-186
Return Loop Outer.
Perform traceback operation for Reverse Processor Dummy lines 211-212 as under:
Calculate the start address for traceback - line 209 (for 3GPP this traceback starts after 2nd window length for which the address calculation is equal to (window length 54) X 2 X 256(total states) /8(8 states are processed in parallel)
Read the memory content of Path History memory corresponding to this address and Multiplexer address– Line 213-214.
Calculate the next trace back address using function calculate_next_address - Line 215 and the figure 3.5 shown below
Actual Traceback starts from the start state calculated by the earlier dummy processor and actual message bits are decoded. (lines 220 – 230).

Matlab Pseudo code for Max log map

This was shown mathematically in equation 5.15 and the pseudo code in section 5.5. Two ACS units combine to formulate one butterfly element as shown in figure 5.8 where state metrics are shown as SM and branch metrics as BM. One trellis stage of a constraint length K coder is made up of 2^{K-2} butterfly elements and requires 2^{K-1} ACS operations. The entire viterbi decoder can be based on one such ACS unit resulting in a structure called state serial architectures [INY98, GLI87]. This is inherently low cost architecture however suffers from very slow speed rendering it less useful for high speed applications. Similarly a separate ACS circuit can be dedicated to every node in the trellis stage, resulting in fully parallel and very high speed architecture [CHU89]. This however is at the cost of very high energy consumption and the second disadvantage in using fully parallel scheme in the poor resource utilization for reconfiguration.

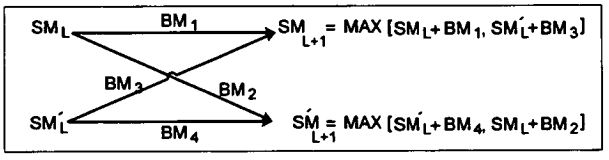

Figure 5.8. A simple butterfly operation for SM calculation

The proposed architecture reuses the available 8 Add-Compare-Select (ACS) blocks in turbo decoding array. The concept of reusing these blocks results in a higher speed architecture than traditionally used state serial architectures. In comparison to fully parallel architectures an intermediate solution is presented where better energy efficiency can be achieved by Processing N states (up to 256) by P ACS (P=8) [CHU89] in contrast to assigning one ACS for each state. This results in 100% utilization of all ACS blocks for any mapped standard on the array. The state machine control makes it possible to use the array as a co-processor. We have also shown a reconfigurable trace back approach for survivor memory management and segmentation, a reconfigurable state metrics memory management methodology, a reconfigurable write address generation and an open trellis and dynamic reconfiguration process. For coherence of representation the viterbi algorithm (VA) is summarized with a 2 state trellis diagram. For each stage

'L' of trellis, branch metrics (BM) are computed using the soft input symbols. State metrics (SMs) for stage L+1 are updated using the SMs for stage 'L' of trellis as shown in figure 5.8.

The BMs are computed by calculating the Euclidian distance of the soft input metrics as shown in section 5.3.1. The input metrics are represented in 4Q2 signed two's complement format similar to the turbo decoding array. The BM block for turbo decoder and viterbi decoder arrays is therefore very similar reutilizing the word length already available in Turbo decoding array.

The error probability of convolution codes decreases exponentially with the constraint length [SHU93] and therefore standards like 3GPP [3GPP99] use large trellises. For brevity the design is explained with the worst case 3GPP example which has 256 states (constraint length=9).

For this large constraint length (K=9), the 256 (2^{K-1}) states are represented by a smaller de Bruijn graph of 2^M states [SHU93]. There are 8 ACS blocks already available in the turbo decoder array [AHM205] and therefore reutilizing these states M=3, processing 8 states/clock cycle for Viterbi. This is shown in the figure 5.9.

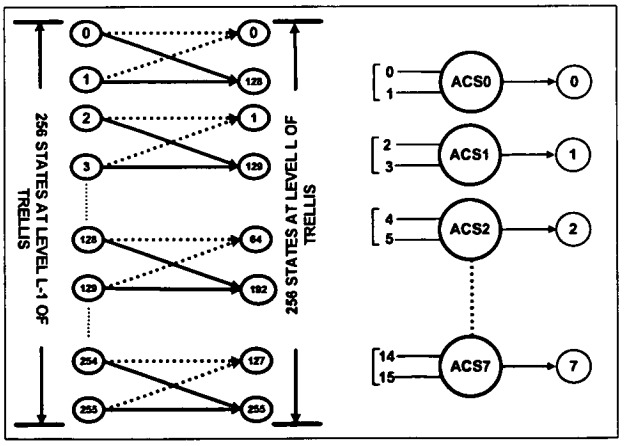

Figure 5.9 256 states 3GPP trellis for generator polynomial 753,561

As shown in Fig 5.9 each ACS unit requires two states at trellis stage 'L-1' to calculate the winning state. This requires saving all the winning states at trellis stage 'L-1' as these are subsequently required at stage 'L'. This contrasts with turbo decoder where forward state metrics were required to be saved for the complete window length. The FSM RAM reutilization for the unified array is explained in section 5.6.2. The Add-Compare-Select operation of viterbi is similar (as explained in section 4.11.4) however in viterbi decoder the ACS units provide additional data bits for Path History RAMs as explained in section 5.6.3. The 8 ACS blocks were used in state parallel sequence in Turbo mode, where as in the viterbi mode this methodology is changed since the number of states are much higher. For example reutilizing these 8 ACS blocks for Viterbi there are 32 clock cycles required to process 256 states in 3GPP [3GPP99]. This requires a modification in the way the saturation logic works since the decision to saturate the results can only be achieved when all the states are processed. This is saturation logic control is shown in figure 5.10 and figure 4.17b.

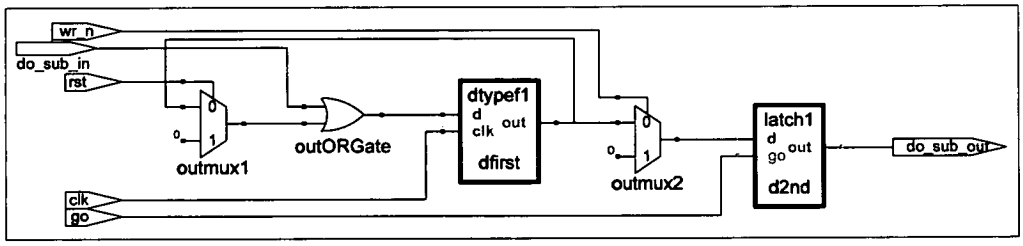

Figure 5.10 Modification in saturation circuit for Viterbi decoding

5.6.2 PATH METRICS (PM) MEMORY

There are 256 winning states at each stage 'L' of the trellis which are saved in SRAMs to be used at stage 'L+1'. Each stage (L) of trellis requires $2^{L-1}/2^{M}$ clock cycles to calculate all wining states. Dual read ports have been used to access two path metrics required for the computation of the next path metric. Size of these RAMs (called FSM RAMs in the context of Turbo) is carefully selected to map both constraint length K=4 trellises in Turbo and up to K=9 constraint length trellises in Viterbi. To avoid the read-write conflict two alternate RAM banks have

been used for read and write operation. The update sequence is shown by the pattern shown in figure 5.11.

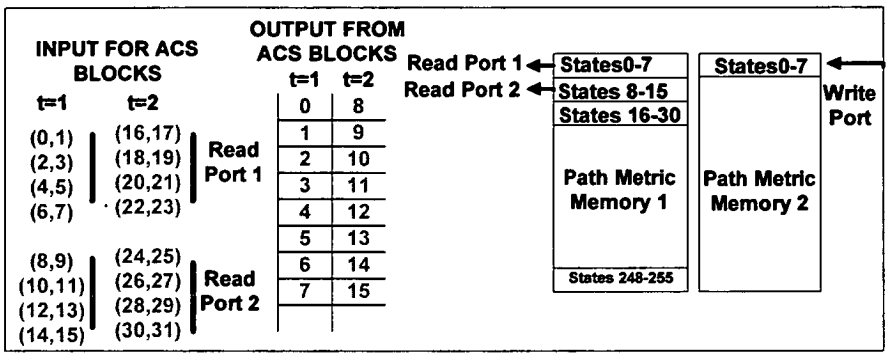

Figure 5.11: PM Memory read and write operations

At time t=1 'Read Port 1' read states 0-7 (for ACS 0-3) and 'Read Port 2' read states 8-15 (for ACS 4-7) from PM Memory 1. The ACS blocks update the path metrics and these are saved on PM Memory 2 as shown in blue. The process continues for trellis stage 'L' and the PM RAM2 is completely updated in 32 clock cycles. These values are then read in trellis stage 'L+1' and now PM Memory 1 will be used for writing the updated metrics. For lower constraint lengths (for e.g. GSM, K=5, 16 states) PM RAMs are updated in just two clock cycles.

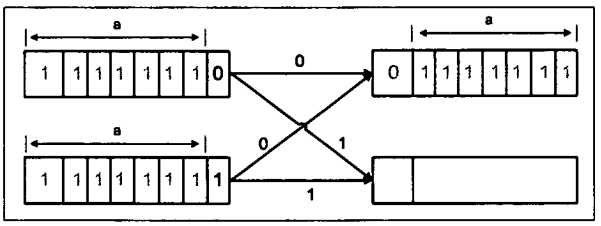

Figure 5.12. Example showing previous state calculation

The input/output state connections for the ACS blocks are explained in figure 5.7. For example if the state at stage 'L' of trellis is 0_1111111(figure 5.12); it implies that the decoded bit (shown in red) is 0 and the two possible states connected to this winning state at stage 'L-1' of trellis are 1111111_0 and 1111111_1. The previous

winning state decision is provided by the path history memory. Path Metrics memory is mapped on to Forward State Metrics RAM [AHM205] already available in turbo decoding array.

The read and write address of PM RAMs is provided by flexible counters in Finite State Machine as shown in figure 5.13 where separate read counters are provided for FSM RAM 1 and FSM RAM2. In turbo mode the read and the write is in the same direction as was shown in figure 4.17 earlier. However in Viterbi mode the read operation as specified in figure 5.11 is performed by two counters for read port 1 and read port 2 for the PM RAMs. Multiplexers A and B are used to switch the counters for turbo mode as was shown and explained in section 4.11.2. Read counter 1 and 2 also have dynamic count_to and count_from flags which are used to provide the PM RAMs controls for multiple viterbi standards. The value of count_to (c.t) and count_from (c.f) is shown for different standards in table 5.1 below:

s/no	Port Name	Counter	Viterbi-3GPP (256 states)		Viterbi-WLAN & IEEE 802.16 64 states		Viterbi GSM/GPRS 16 states	
			c.t	c.f	c.t	c.f	c.t	c.f
1.	Read Port 1 FSM RAM 1	5 bit up counter increment of 2	30	0	7	0	0	0
2.	Read Port 2 FSM RAM 2	5 bit up counter increment of 2.	31	1	6	1	1	1
3.	Write Port FSM RAM1 and RAM2.	5 bit up counter	0	31	0	7	0	1

Table 5.1 Viterbi state machine counter values for different standards

State machine also provides the read and write control signals to all the RAMs. Data read from the PM RAMs is passed on to ACS units. State machine control provides select signals for multiplexer C which determines the PM RAM used to read data. Multiplexer D provides change of control from Viterbi mode to Turbo mode as in this mode PM RAMs are not used to provide data to ACS blocks.

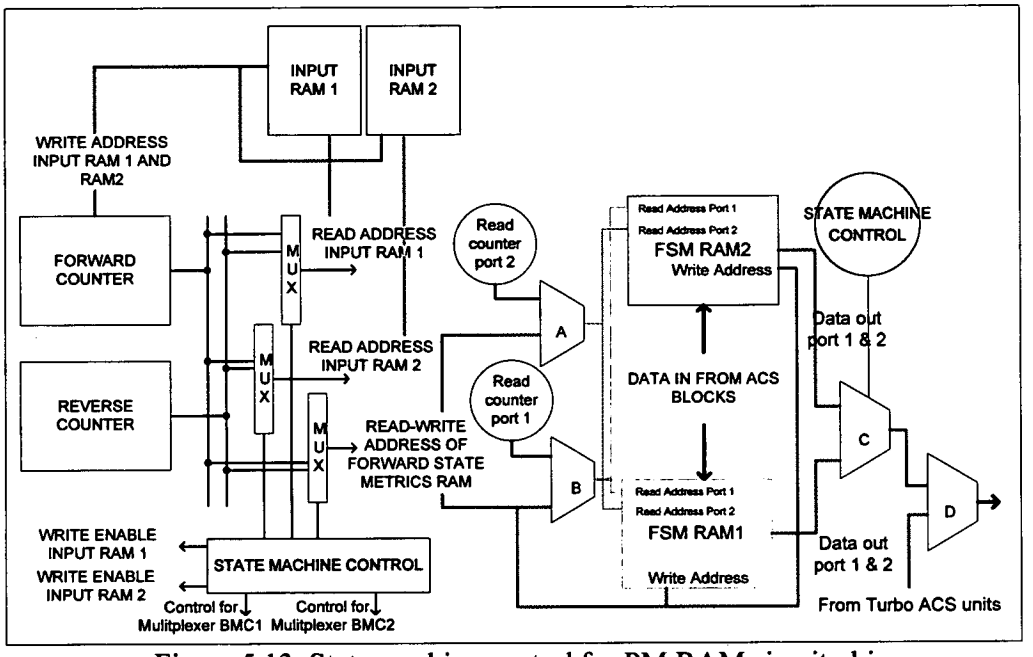

Figure 5.13. State machine control for PM RAMs in viterbi

5.6.3 PATH HISTORY (PH) MEMORY

Path History Memory is used in VA to find the survivor path. The contents of this memory are updated on each stage 'L' of the trellis which allows reconstructing the survivor path. Each ACS unit output one bit for the survivor state. 8 decision bits per clock cycle (given by 8 ACS units) are saved in the PH RAM. The total size of the PH RAM is given by:

$$\text{Size of PH RAM} = (2^{L-1}/2^{M}) \times WL \times \text{Total windows} \text{ -------}$$

--- where window length is 6 times the constraint length.

For constraint length 9, size =32 x 54 x 4 = 6912x8.

There is 8K (2Kx4) of output RAM available in turbo decoding array [AHM205]. This is reused for PH Memory and is shown in the figure 5.14.

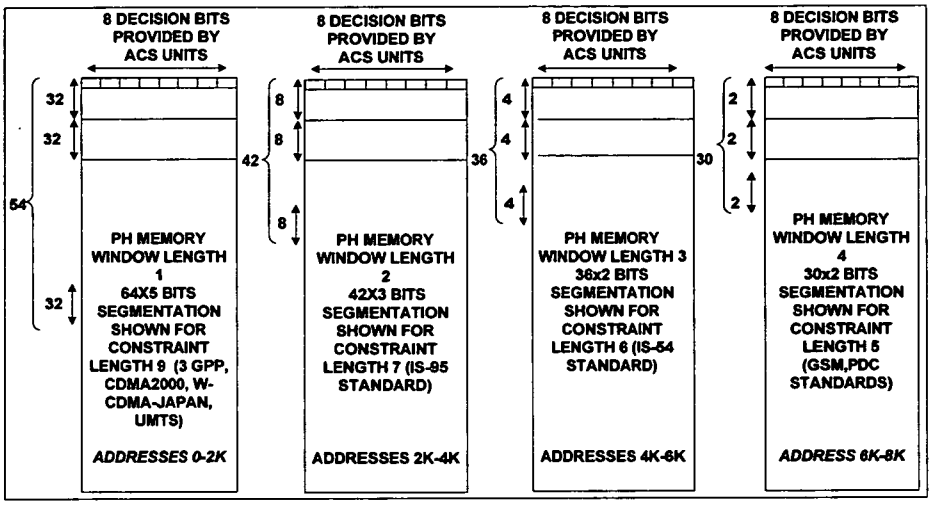

Figure 5.14: 2Kx4 identical path history RAMs in viterbi decoder, Segmentation and mappings shown for different standards

Each block of 2K RAM is used to store one window length (WL) of decision bits. The RAM is segmented by the total states of the trellis. For example, for 256 states there are 256 decision bits (one for each state). Therefore the RAM is segmented in to 32x8 segments. Figure 5.14 shows mappings and segmentation of the PH memory for different standards. However the used area of each block of 2K RAM for a particular mapped standard will exactly be the same. This is enumerated in table 5.2 below

Standard	**Constraint length**	**Memory utilization for each 2K RAM=WL x segment size.**
W-CDMA(Japan) CDMA 2000 UMTS	9	54 x 32 =1728 X 8 bits
GSM,PDC	5	30 x 2 = 60 X 8 bits
IS-95, IEEE 802.16	7	42 x 8 = 336 X 8 bits
IS-54	6	36 x 4 = 144 X 8 bits

Table5.2. Memory utilization for different standards in viterbi decoder

PH memory is read by trace back processors (section 5.6.6) as they calculate the survivor path in the reverse traversing of the trellis. The write and read control is provided by the state machine. This is explained in the section 5.6.5. Address generation is made reconfigurable as explained in section 5.6.4.

5.6.4 RECONFIGURABLE WRITE ADDRESS GENERATOR

The write address generation is controlled by Finite State Machine using a 5-bit counter driving the 6-bit counter. The 6 bit counter will be incremented once the 5 bit counter reaches the terminal (maximum) count.

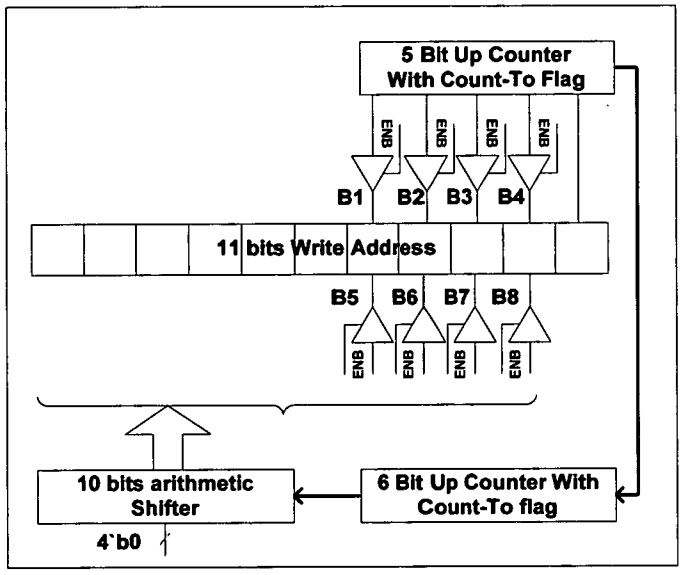

Figure 5.15: Reconfigurable write address generator for viterbi decoder

Both these counters have dynamic 'count_to' flags which provide the terminal count. The terminal count on the 6 bit counter provides means of introducing reconfigurable window length and terminal count setting on 5 bit counter adjusts the segment length for multiple standards. As shown in figure 5.12 a technique of segment by segment address writing on PH memory has been used where each segment corresponds to one trellis stage 'L' (256 states for 3GPP). 5 bit counter controls the segment address. The segment address however is different for different standards and the maximum value is 5 bits for 3GPP. 6 bit up counter is

concatenated with 4 zero bits on the MSB side and is then left shifted depending on the mapped standard. The output of the arithmetic shifter drives the lower end of write address through tri state buffers. 5 bit up counter drives the other side of address generator through tri state buffers (B1-B4) as shown in the figure 5.15.

No	B1	B2	B3	B4	B5	B6	B7	B8	SH
1	On	On	On	On	Off	Off	Off	Off	4
2	Off	Off	Off	Off	On	On	On	On	0
3	Off	Off	On	On	On	On	Off	Off	2
4	Off	Off	Off	On	On	On	On	Off	1

Table 5.3. Configuration bits for tristate buffers in viterbi decoder

Table 5.3 defines configuration bits for the tri state buffers shown in figure 5.15. Rows in the table correspond to constraint lengths 9,5,7,6 respectively. The constraint lengths are for different standards as defined in table 5.2. Last column 'SH' in table 5.3 defines the number of left shifts for the arithmetic shifter. For example if 6 bit up counter gives [C5_C4_C3_C2_C1_C0] and 10 bits arithmetic shifter input is [0_0_0_0_C5_C4_C3_C2_C1_C0] then if SH is 1 (left shift by 1) the output from arithmetic shifter will be [0_0_0_C5_C4_C3_C2_C1_C0_0]. This output will drive the lower end of write address as shown in figure 5.15. Using table 5.2 and configuration switch values corresponding to SH 1 (line No 4 table 5.3), the write address will be

0	0	C6	C5	C4	C3	C2	C1	C0	U1	U0

[U4_U3_U2_U1_U0] are the five bit count values of upper counter (connected with B1-B4). This is the write address for IS54 standard, similarly for 3GPP the write address as shown by switches in line No 1 table 1, will be

C5	C4	C3	C2	C1	C0	U4	U3	U2	U1	U0

Both these counters have count_to flags and for IS54 count_to flag of five bit counter will be set to 4, whereas count_to flag of 6 bit counter will be set to 36. For 3GPP the count_to flags of 5 bit and 6 bit counters will be 32 and 54 respectively.

5.6.5 STATE MACHINE CONTROL OF PATH HISTORY MEMORY

Viterbi state machine control is very similar to turbo decoding. The only difference between the two is that the control is for read and write of Path history memory as opposed to input memories in turbo decoding. Both Turbo and Viterbi decoders use forward and reverse state metrics processing. There are windowed versions of the algorithm to improve the latency and in its simplistic form it uses two reverse processors (B1 and B2) in parallel with one forward processor (FP1). Reverse processor can start cold in any state (initializing each state as equi - probable), but after few iterations (equal to window length WL) the state metrics are as reliable as if the process had been started at the final node of trellis. Let B2 be the dummy reverse processor that starts from state 0 and after reverse traversing the trellis for a WL, provides the start state for the actual reverse processor B1.

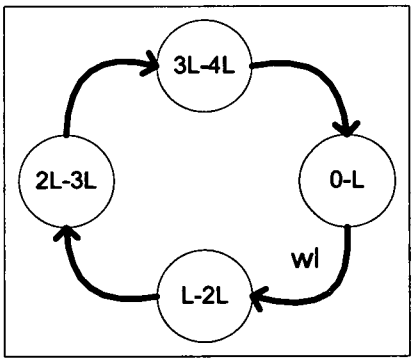

Figure 5.16. State machine in viterbi showing 4 operating states

Figure 5.16 shows the four basic states of VA with start state as '0-L'. The detailed scheduling diagram is presented in figure 5.17. Vertical axis presents time and horizontal axis presents trellis length. Table 5.4 is linked to Figure 5.17 for its explanation. Moreover a similar scheduling diagram for turbo decoding was presented in section 4.11.2 and our earlier work [AHM205]. Table 5.4 shows the

working of FP, B1 and B2 as they write and read PH memories. After 4 WLs (0-L to 3L-4L) the cycle repeats. First decoded bits are output continuously after latency of 3 WLs from time 3L-4L.

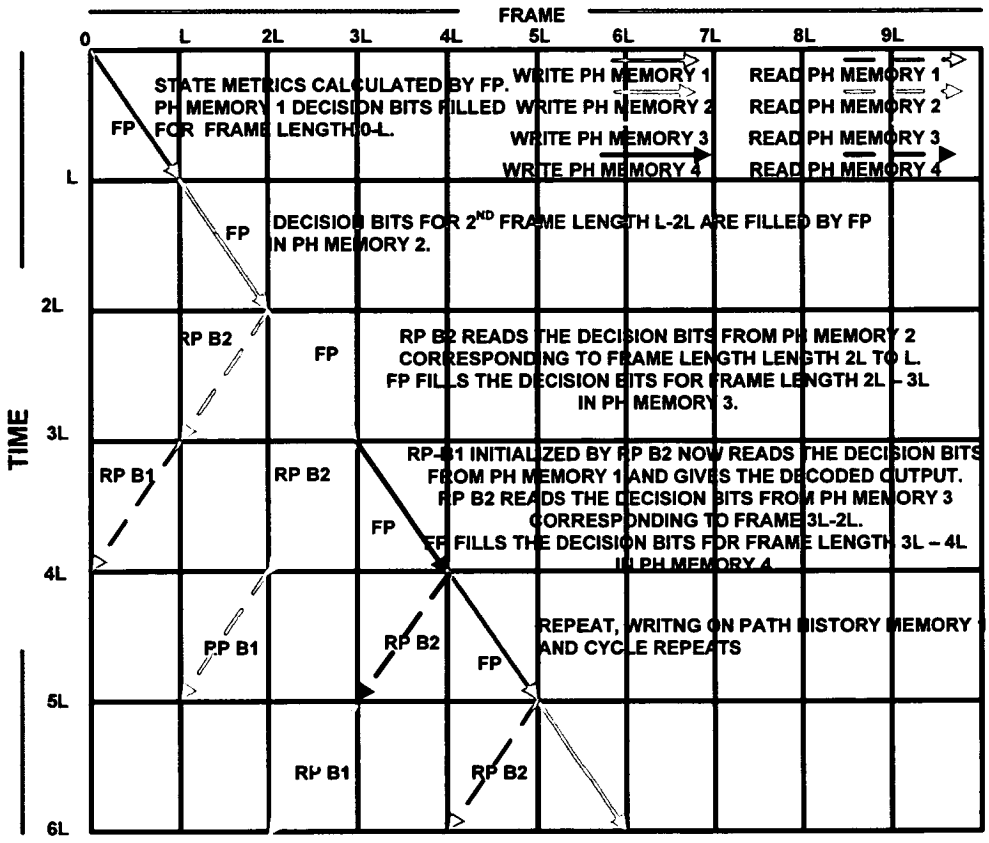

Figure 5.17: Scheduling diagram for viterbi decoding

Time	PH Mem1	PH Mem2	PH Mem3	PH Mem4
0-L	Write FP	Read B1	NO OP	Read B2
L-2L	Read B2	Write FP	Read B1	NO OP
2L-3L	NO OP	Read B2	Write FP	Read B1
3L-4L	Read B1	NO OP	Read B2	Write FP
4L-5L	Write FP	Read B1	NO OP	Read B2

Table 5.4. Read and writes on PH memories by FP,B1 and B2 in viterbi decoder.

5.6.5.1 RECONFIGURABLE ASPECTS OF STATE MACHINE

The state machine control needs to provide flexibility to control not only a switch over between viterbi and turbo decoding but also amongst different standards within viterbi decoding. The state machine controls are explained in subsequent sections along with the explanation of individual components of the array. The controls can be classified into the following categories:

- Read and Write Controls for Input RAMs.
- Controls for generating read and write addresses for Input RAMs.
- Control signals for Branch Metrics Multiplexers.
- Adjustment of scheduling and windowing as per the mapped standard.
- Control signals for disabling the unused blocks.
- Read and Write control signals for Path Metrics RAMs (called FSM RAMs in Turbo)
- Address generation for PM RAMs.
- Read and Write Controls for all the ports of PM RAMs.
- Control signal for multiplexers controlling data outputs of PM RAMs.
- Read and Write Controls for Path History (output RAMs in Turbo decoding).

5.6.6 RECONFIGURABLE TRACE BACK PROCESSING

As shown in section 5.6.5 there are two reverse processors B1 and B2 (dummy) working in parallel. To get the survivor path, either register-exchange or trace back structures can be used [RAD81]. Since the trace-back is efficient for larger constraint lengths and low power applications, trace back processing is selected. In trace back processing the previous trellis path stage S_{L-1} is given by the current path state S_L according to the following update.

$$S_{L-1} = [S_L << 1, D]$$

this corresponds to a left shift of the current state introducing the value of surviving bit D in the vacant position as shown by D1-D7 in figure 5.18. Survivor bit is selected by multiplexer M1 from the data bus of PH Memories. The select control

to this multiplexer is provided by D1, D2, D3 outputs. 6 bit Down Counter and arithmetic Shifter arrangement is also shown which works exactly the same as was explained in section 5.6.4. The only difference is that the counter is initialized with the last address of PH Memory and counts down by 1. The decrement in count by one provides a jump of one segment length. Reverse processor B1 and B2 share the same counter and shifter.

Reconfigurable trace back processing is explained with examples of 3GPP and GSM as was done in section 5.6.4.

	B1	B2	B3	B4	B5	B6	B7	B8
GSM	Off	Off	Off	off	on	on	On	On
3GPP	On	On	On	on	off	off	Off	Off

Table 5.5 Tri state buffer controls for reconfigurable trace back processing in viterbi decoder

Let U4_U3_U2_U1 are outputs of buffers B1-B2_B3_B4 and C5_C4_C3_C2_C1_C0 are the outputs of the 6 bit down counter. Output of the arithmetic shifter without any shift is 0_0_0_0_C6_C5_C4_C3_C2_C1_C0.

No	B9	B10	B11	Sh	Output Shifter
GSM	off	Off	Off	0	0_0_0_0_C5_C4_C3_C2_C1_C0
3GPP	on	On	On	4	C5_C4_C3_C2_C1_C0_0_0_0_0

Table 5.6. Arithmetic shifter outputs and buffer controls in viterbi decoder

The contents of the read register for 3GPP using the controls in table 4 and table 5 are:

C5	C4	C3	C2	C1	C0	U4	U3	U2	U1	U0

Similarly for GSM the read register contents are:

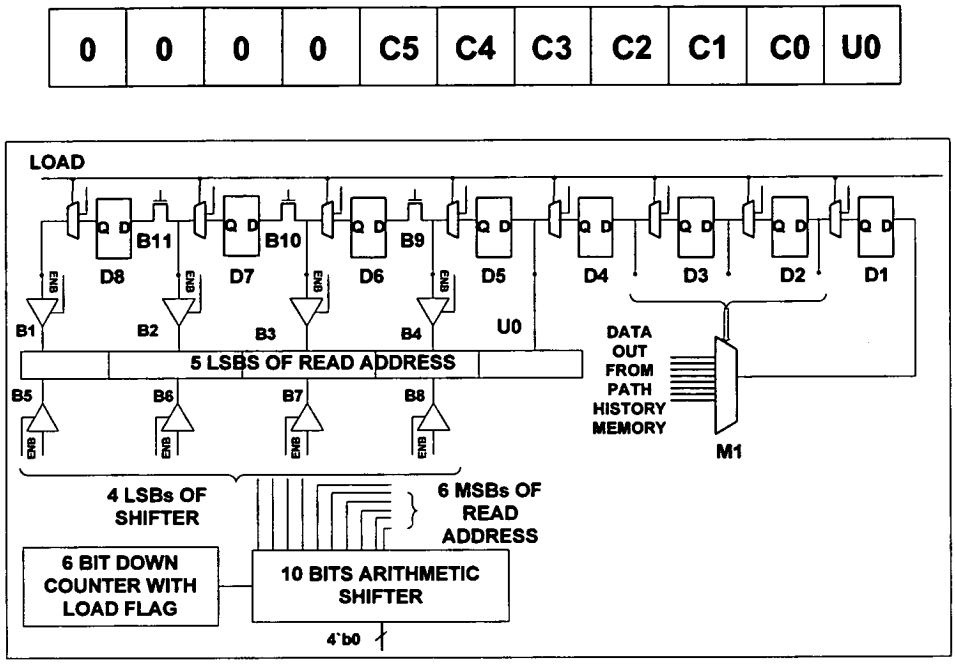

Figure 5.18: Trace back processing for reconfigurable viterbi decoding

5.6.7 OPEN TRELLIS AND DYNAMIC RECONFIGURATION

The input connections to all ACS units are made flexible by reconfigurable logic as shown in figure 4.7. BM configurations are saved in Input RAMs 1- 4. Each RAM block is 32x8 bits, and has asynchronous read and synchronous write ports. The RAMs are filled in 32 clock cycles (for 3GPP [3GPP99] trellis). It is worth noting that during the write operation on the RAMs simultaneous read is also performed. This provides dynamic switch over for different trellis types. These configuration bits provide the appropriate BMs for the ACS units. ACS units also need the previous state metric values which are read from Forward Processor RAMs as explained earlier. After first segment write operation on the RAMs is completed and then only read operation is performed for the subsequent segments. The size of the input RAMs is carefully selected to store all the branch metric configuration bits and the two window lengths of input metrics in case of Turbo decoding. The read write controls for the input RAMs is provided by state machine. These

controls change with the mapped standards and also with turbo and viterbi mappings.

5.7 RESULTS

The Viterbi mappings for various standards are first coded in matlab with floating point precision. The precision is then changed to fixed point as per the hardware design. The matlab code is changed to simulate the hardware, and therefore it provides the test vectors that can be directly used in the RTL design. Matlab fixed point simulations are also used to compare the results with floating point precisions. The RTL design after pre-synthesis simulations and verification is synthesized using Synopsys Design Compiler for 180 nm CMOS UMC cell library and the chip layout is done on Silicon ensemble. Post layout power figures are taken from Synopsys Design Power. Synopsys designware SRAMs were used for Forward Processor RAMs. Virtual Silicon 2K x 8 synchronous (separate read and write port) macro RAMs were used for Path history memory consuming 110 uW/MHz/Port. Figure 5.19 gives the cell count and overall area of the array with 180nm process.

```
****************************************
Report : area
Design : oneFPtwoRPs
Version: X-2005.09-SP1
Date   : Thu Jun  1 01:50:25 2006
****************************************

Library(s) Used:

   umcl18u250t2_wc (File: /home/SLIg/umc0.18/UMCL18U250D2_2.4/
design_compiler/umcl18u250t2_wc.db)

Number of ports:          310
Number of nets:         10028
Number of cells:         8225
Number of references:     212

Combinational area:   1148921.000000
Noncombinational area:  529297.750000
Net Interconnect area:    undefined  (Wire load has zero net area)

Total cell area:      1678616.125000
```

Figure 5.19. Area of array without output RAMs

The array uses 1.67 mm^2 of area as shown in figure 5.19. The overall area and the cost of reconfigurable viterbi components is shown in figure 5.20. Area utilization of reconfigurable switching is 97089.6 um^2 which is only 3.4% of the overall area. The power consumption of this switching fabric is just 2.5% of the overall power consumption. The implementation of viterbi components on turbo decoder array increased the area of turbo decoder array by 20%. This overhead is much less than implementing viterbi and turbo decoder arrays separately. The unused blocks are disabled throughout the chip which results in no significant increase in power consumption of the overall array. The area and power figures are given in table 5.7. Similar power figures are achieved compared to the unified VLSI design [MAR02], however our design is made much more flexible than [MAR02]. 288 configuration bits are required to configure the array. At 20 MHz configuration clock speed it takes 14.4 µsec to completely reconfigure the array.

The power figures are quoted at 20MHz clock speeds.

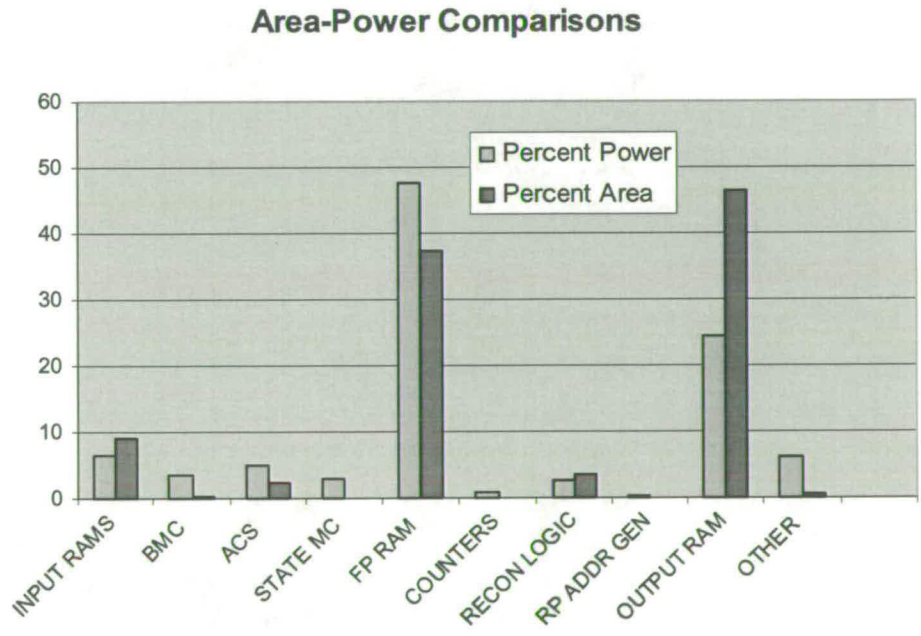

Figure 5.20: Area and Power results of individual components of the viterbi design

Technology	0.18	microns

	standard cell CMOS
Code rate (flexible)	1/2, 1/3, 1/4,1/5
Constraint length (flexible)	Up to 9 (256 states)
Generator polynomial	Flexible
Survivor path length (flexible)	Maximum 54
Decision level	4 bit soft decision
ACS units	8
Power supply	1.8V
Operating frequency	20.0 MHz
Total Power in mW	69.961 mW
Total Area	2.82 mm^2

Table 5.7. Results for viterbi decoder

The percentage of switching power and internal power in viterbi components is shown in figure 5.21 below:

```
Library(s) Used:
    umcl18u250t2_wc (File: /home/SLIg/umc0.18/UMCL18U250D2_2.4/
design_compiler/umcl18u250t2_wc.db)
Operating Conditions: WORST   Library: umcl18u250t2_typ
Global Operating Voltage = 1.8
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1pW

  Cell Internal Power  =  12.5929 mW   (18%)
  Net Switching Power  =  57.3680 mW   (82%)
                          ---------
Total Dynamic Power    =  69.961 mW  (100%)

Cell Leakage Power     = 175.3585 uW
```

Figure 5.21: Net switching power and cell internal power in 180nm UMC CMOS process for viterbi components of the array

5.6.1 COMPARISON

The reconfigurable viterbi decoder design is compared with Texas Instrument's viterbi decoder [HOC00]. This is also a flexible decoder with variable constraint length and code rate. The maximum data rate achievable by the decoder is 2.5 Mbps which is slower than the data rates provided by our design (10.5 Mbps at maximum clock frequency). The decoder in [HOC00] also used a coprocessor within programmable DSP which makes it limited to a particular processor type.
The work proposed by [CHA01] presented a hard input reconfigurable viterbi decoding using single bit Hamming distances for WLAN and 3G. Our proposed design achieves a similar data rate however also has turbo decoding components and a soft decision viterbi decoder. The design in [CHA01] hence has limited flexibility and the single bit hamming distance calculations makes it difficult to use in practical scenarios.

The work proposed in [KEL93] is a foldable scheduling scheme for reconfigurable viterbi decoder. Turbo decoding is not available with this decoder, however the proposed architecture has a utilization of 100% for almost all configurations. The area power results are not provided and the implementation is also done on a set of FPGA boards rather than ASIC.

5.8 CONCLUSION

A fully flexible viterbi decoder for reconfigurable platforms has been designed. The decoder consumes 69mw at 20MHz occupying 2.824 mm^2 area. The array can be mapped onto various communication standards and hence can be used as an IP in reconfigurable platforms. It is shown with results that the cost of reconfiguration in our chosen domain is negligible and a careful reconfigurable design can give results very close to the state of the art ASIC designs but with much increased flexibility.

Chapter 6

LOW POWER INTERLEAVER

In this chapter, a novel implementation methodology to implement 3GPP interleaver is proposed. Interleaving is the key factor in the excellent performance of turbo codes. The novel implementation methodology discussed in this chapter results in 3GPP interleaver design with a significant reduction in SRAM area. Interleaver SRAMs are the major contributor to the area and power of the turbo decoder and hence reducing SRAMs results in significant area and energy savings.

6.1 INTRODUCTION

Numerous interleaver design algorithms have been proposed so far, many of them are purely heuristic, other employ optimization techniques. A large class of interleaver designs are based on spreading which will be presented in subsequent sections. Due to the large number of proposals and due to the limited space in this thesis, we can present only the most significant algorithms in the following sections.

6.1.1 RECTANGULAR INTERLEAVERS

Rectangular interleavers, often simply referred to as "block interleavers", have been used for the transmission over fading channels for a long time [CON87, ESA89]. This interleaver can be represented as a rectangular array of boxes, where each box contains a single bit or a tuple of bits. The boxes of x X y rectangular interleaver are arranged in x rows and y columns. In its simplest form it is implemented as memory in which data is written row-wise and read column-wise. For example, data is written row-wise as shown in Table 6.1.

D1	D2	D3	D4
D5	D6	D7	D8
D9	D10	D11	D12
D13	D14	D15	D16
D17	D18	D19	D20
D21	D22	D23	D24

Table 6.1 Writing data D1-D24 row by row in the memory in the case of a rectangular interleaver

The interleaving process in row-column interleaver consists of reading data column-wise shown in table 6.2.

D1	D5	D9	D13	D17	D21	D2	D6	D10	D14	D18	D22	D3	D7	D11

Table 6.2. Reading data column-wise from memory in rectangular interleaver

It is shown in [PSC96] that the utilization of rectangular interleavers in Turbo codes entails a major problem. With these interleavers, there is a large number of first component input words containing two weight 2 error patterns, which are permuted to similar second component input words containing two weight-2 error patterns. These EP's associated parity weights are low and they form a woven error pattern which produces a low codeword weight. The multiplicity of these low-weight code words lies in the same order as the interleaver length K. [BH00a] shows that Turbo codes with rectangular interleavers can be interpreted as a special case of convolutional codes. Moreover, [BH00a] and [GMB00, GMB01] showed that the Turbo codes exhibit a particularly low trellis complexity for rectangular interleavers. There are many constructions possible for the block interleavers, some popular constructions are given below:

6.1.2 HELICAL INTERLEAVER

A "helical" interleaver writes data row-wise as in Table 6.1 but reads data diagonal-wise as shown below and in Table 6.1 in different colour patterns.

D21	D18	D15	D12	D5	D22	D19	D16	D9	D6	D3	D23	D20	D13	...

Table 6.3 Diagonal interleaver read operation

6.1.3 ODD-EVEN INTERLEAVER

It was shown in [BAR94] that "odd-even" interleaver gives significant improvements when used in a turbo encoder design. Let us assume that we have a random sequence of binary data input to a rate one half systematic encoder and we produce the coded bits but only store the odd coded bits as in table 6.4.

D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15
X1	_	X3	_	X5	_	X7	_	X9	_	X11	_	X12	_	X13

Table 6.4 Table showing odd position bits for Odd-Even interleaver

Now we store row wise the sequence of data D1-D15 say in a block interleaver with an odd number of rows and odd number of columns. The encoding is done after reading column wise and the even positions of the coded bits are stored as shown in table 6.5 below:

D_A	D_B	D_C	D_D	D_E	D_F	D_G	D_H	D_I	D_J	D_K	D_L	D_M	D_N	D_O
_	Y_B	_	Y_D	_	Y_F	_	Y_H	_	Y_J	_	Y_L	_	Y_M	_

Table 6.5 Table showing the even position bits for odd even interleaver

The data which is actually sent through the channel is multiplex data from table 6.4 and 6.5 as shown in table 6.6 below:

D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15
X1	Y_B	X3	Y_D	X5	Y_F	X7	Y_H	X9	Y_J	X11	Y_L	X12	Y_M	X13

Table 6.6 The output to channel from odd-even interleaver

The advantage with this scheme is that the coded power is uniformly distributed as each of the information bits has their own coded bit associated with it.

6.1.4 SIMILE INTERLEAVER

In Simile interleaver [BAR95] the whole block is divided in n+1 sequences, where n is the number of delay elements in the encoder. For a simple four state encoder, n = 2, the corresponding 3 sequences become:

$Sequence^0 = \{dk \mid k \bmod (n + 1) = 0\}$

$Sequence^1 = \{dk \mid k \bmod (n + 1) = 1\}$

$Sequence^2 = \{dk \mid k \bmod (n + 1) = 2\}$

The property of the Simile interleaver is that after encoding both sequences of information bits (original and the interleaved), the state of both encoders stays the same. Therefore same tail bits can be appended to the information bits, which drive both encoders to the zero state. The final encoder state is independent of the order of the bits in normal or interleaved pattern as long as they follow the same sequence i.e., the index for the interleaved output follow the same pattern as for the non interleaved sequence. As an example in [BAR95], to generate a simile odd-even block helical interleaver, the interleaver depth is chosen as an even number which is multiple of (v+1) and the second dimension is chosen to be prime. For a four state RSC encoder it is shown as:

X1	X2	X3	X4	X5	X6
X7	X8	X9	X10	X11	X12
X13	X14	X15	X16	X17	X18
X19	X20	X21	X22	X23	X24
X25	X26	X27	X28	X29	X30

Table 6.7 Row by Row entry for Simile odd-even block helical interleaver

Part of the interleaved sequence is shown in table 6.8 below:

	X25	X20	X15	X10	X5	X30	X19	X14	X9	X4	X29	X24	X13	X8	X3	..
J MOD 3	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	..
	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	X12	X13	X14	X15	..

Table 6.8 The output of a simile odd-even block helical interleaver

Comparison of row 1 and row 3 in table 6.8 shows both interleaved and non interleaved bits follow the same sequence.

6.1.5 FRAME INTERLEAVER

Unlike simile interleaver frame interleaver does not uses tails to drive both encoders to the same state. Instead it puts more constraints on the interleaver design [BER96]. The N information bits to be interleaved are stored twice in a memory of size 2N. The addresses are selected such that their subsequent reading is time-separated by number of periods which is multiple of constraint length L. This arrangement also ensures that if the encoder started in state zero, it will end in state zero without the need of any tail bits.

6.1.6 PSEUDO-RANDOM INTERLEAVER

In this category of interleaver the interleaved patterns are generated by pseudo-random algorithm. This algorithm can either be implemented in hardware as a pseudo-random generator or the interleaved patterns corresponding to the complete block size can be stored in a lookup table. Pseudo-random interleaving results depend on the block size and are varying for small and large block lengths. The criterion for choosing between them is done on computer simulations rather than analytically.

6.1.7 S-TYPE INTERLEAVERS

In an "S-random" interleaver, each randomly selected integer is compared to 'S' previously selected integers and the current integer is selected only if it is at distance larger than S from any of the S pervious selections. The process is repeated for all N addresses. The interleaver gain is larger for larger 'S' values. It was shown in [BER96] that if an interleaver of size N is randomly selected, the probability that a particular weight-2 (i.e. the sequence in which the number of ones are 2) data sequence will be permuted by the interleaver into another sequence of same form is roughly 2/N for large N. The probability is larger for smaller N. The weight 2 data sequence is an important factor in the design of the component codes.

S-random interleavers prove to be an effective methodology to avoid weight-2 data sequences.

6.1.8 UNIFORM INTERLEAVERS

Uniform interleaver was explained in [BEN196] as average of all possible interleavers with an aim to derive the performance bounds for turbo codes. Consider a sequence made of w ones and k-w zeros. A uniform interleaver of length k is a probabilistic device which maps this sequence to all distinct $\binom{k}{w}$ permutations with equal probability $1/\binom{k}{w}$. The method gives upper bounds on the error probability which are quite accurate at high values of SNRs but not so accurate at lower SNRs. This technique allows the analysis of a turbo code as if it were made of two independent elementary codes, due to the uniform distribution produced by the interleaver.

6.1.10 CONVOLUTIONAL INTERLEAVERS

Convolutional interleavers were introduced in ([RAM70], [FOR71]. A typical convolutional interleaver consists of an input and output switch, which cyclically connects L shift registers, each register being delayed in time by B bits. The k-th shift register thus introduces a (L – k)B bit delay. The end-to-end interleaver-deinterleaver delay is (L – 1)LB. This type of interleaver also introduces some zeros at the beginning and end of the interleaved data block due to the delay associated with each shift register

6.1.11 CODE MATCHED INTERLEAVER

Code matched interleaver is optimised for a specific code and starts with an S-random interleaver that can break as many of the short length input patterns as possible. Code matched interleavers aim to eliminate the first spectral lines of the turbo code distance spectrum which are generated by low weight input sequences [FEN99]. The most significant low weight input patterns are then identified. The

interleaver search is done in an iterative way to eliminate the cases where the interleaver outputs form an error pattern of the weight targeted to be eliminated.

6.1.12 CHAOTIC INTERLEAVER

The basic advantage of chaotic interleaver is its reduced complexity and increased level of security in the communication system without penalty in performance [ZHA01]. This interleaver uses chaotic type of mapping, e.g., the Logistic mapping, which is used to produce chaotic data. This procedure is used to order the elements in each column vector, followed by ordering of the column vectors themselves. The interleaver mapping start is based on an initial state which can be considered as a key in a secure communication link. The decoder can't produce any meaningful output unless the key is known. As compared to pseudo-random interleavers a small improvement in performance of turbo codes was shown by Chaotic interleavers in [ZHA01].

6.1.13 NON-BLOCK INTERLEAVERS

The main advantage of non block interleavers is their reduced delay. They were proposed in [PIE96] as self synchronizing turbo codes using non block interleavers. The proposed schemes do not require pre ambles to detect the beginning of the block. These are useful in application using long messages like stream-oriented applications [HAL01] as there is no requirement of data-framing as in other block interleavers.

6.1.14 THE BEST INTERLEAVER

Interleavers are designed for specific system requirements. There is therefore no universal formula that can be used. It was shown in [BAR94] that for short block sizes and low Eb/No, an odd-even interleaver outperforms a pseudo-random interleaver [BAR94] and vice-versa at higher Eb/No. For larger block sizes, an S-type interleaver outperforms a pseudo-random interleaver [BEN196, BEN296]. With the larger size of the interleaver the extrinsic information passing between the two constituent decoder becomes more uncorrelated. Therefore the process allows

for two "independent" criteria to estimate the soft value of the same bit. The second important design parameter is the degree of "randomness" with which the interleaver and the deinterleaver spread the bursts of errors from one decoder output to the next decoder input. From this point of view the ideal interleaver is a random interleaver.

In [KHA97], a method to optimise the interleaver structure using the "Hungarian method" is presented. The goal is to break all the weight-2 sequences such that at least one of the outputs avoids a terminating zero-phase sequence. In [DAN98] a canonical form of the interleaver engine with minimal delay is defined as a finite state permuter (FSP). Two algorithms are developed for a systematic iterative construction of interleavers with a complexity that is polynomial with the interleaver size. Each transposition vector has associated with it a cost function, the algorithms aiming at the minimisation of this cost function. The complexity of the algorithm depends on the length of the error patterns which are taken into consideration.

Recent work has shown that the tails can be dispensed with, without any significant loss in performance for higher interleaver sizes. From all the above examples of interleaver design it is clear that the role of the interleaver is to allow the decoders to make uncorrelated estimates of the soft values of the same information bit. The less "correlated" the two estimates are, the better the convergence of the iterative decoding algorithm.

6.2 THE 3G INTERLEAVER

The turbo code internal interleaver in 3GPP turbo decoding is defined by a complex algorithm for the generation of interleaved addresses. It is a pseudorandom block interleaver consists of two steps: mother interleaver generation and pruning [3GPP99]. The input sequence is first written row by row in a rectangular matrix. The number of rows and columns is a function of the information block size which can vary between 40 and 5114 bits. There are 207 mother interleaver sets that can

be selected depending on the information block length. Once a particular set is selected, some bits are pruned in order to adjust the size back to the original information block size.

There are tables with primitive roots used for inter/intra row permutations. After performing permutations on rows and columns, the sequence is read out from the interleaver matrix column wise. These permutations are based on a set of prime numbers. The straightforward method of implementing the address interleaving is to generate all the possible addresses corresponding to a particular frame length, and store this interleaver address table in a memory. The maximum block size defined in 3GPP is 5114 and hence the memory required for interleaver is 5114 by 13 bits. Thus interleaver/Deinterleaver RAMs are major contributors to the area, power of turbo decoder [MAS99, PEN03].

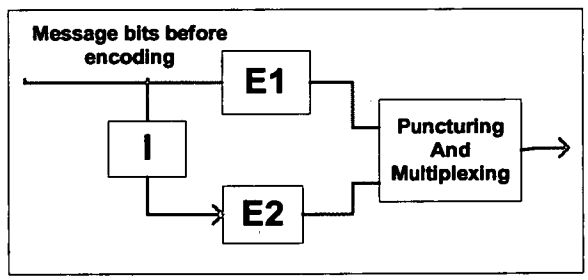

Figure 6.1. PCCC (Parallel Concatenated Convolutional) Encoder

The novelty of our design comes from implementing a low power solution that target the above mentioned bottleneck. Contrary to existing approaches [MAS99, PEN03] a dedicated address interleaving data path is implemented that generates the addresses in real time. The real time address computation avoids the use of pre-computed address storage which greatly reduces the load on the processor and gives significant improvements in area and power. ASIC synthesis results on 0.18 μm CMOS UMC technology demonstrate the efficiency of the proposed VLSI interleaver architecture. Figure 6.2 shows a parallel concatenated convolution (PCCC) turbo consisting of two SISO (soft input soft output) decoders connected

through an interleaver - deinterleaver structure [3GPP99]. The component decoders shown in figure 6.2 are individually matched to the corresponding encoders shown in figure 6.1. These constituent decoders work in an iterative way and the decoding process is stopped when the desired reliability is achieved.

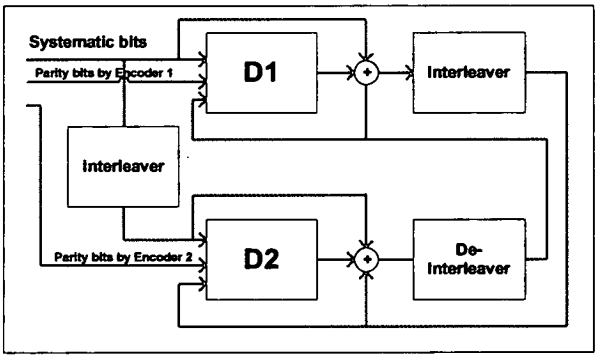

Figure 6.2. PCCC (Parallel Concatenated Convolutional) Decoder

6.3 OVERVIEW OF ALGORITHM

The 3GPP algorithm maps an input sequence of length K (40 – 5114) to an interleaved sequence of the same length (for full details refer [3GPP99]). The algorithm is first implemented in matlab where it translated from its definition of two dimensional rectangular interleaver matrix in to one dimensional space. The translation makes the algorithm easy to implement for one dimensional SRAM based VLSI designs. The hardware design follows from the fixed point simulation in matlab. The matlab code is written exactly replicating hardware in a process called hard wired simulations. The simulations generate the test vectors that can be used for presynthesis verification of HDL code. The VLSI design of the algorithm follows the design cycle as shown in the figure 6.3. The two dimensional interleaver matrix is generated by the following pseudo code:

6.3.1 PSEUDO CODE

The algorithm is explained in two phases. For ease of understanding the notations used in the explanation are kept exactly the same as are used in the standard [3GPP99].

6.3.1.1 PHASE1. PREPARATORY PHASE

Step 1:

Calculate the number of rows defined by length of K (bits input to decoder). It can have a value of 5, 10 or 20.

Step 2:

Length of K also determines the value of a prime number 'p'. It has a fixed value of 53 for range of K between 481 and 530 and is computed from a lookup table for other values of K [3GPP99].

Step 3:

'K' and 'p' determine the number of columns of the interleaver matrix. Corresponding to the number of rows and columns calculated above the bit sequence is entered in the RxC interleaver matrix row wise. There will be some values of K with number of bits less than the total number of rows and columns of the interleaver matrix i.e., RxC >K. These empty values in the rectangular matrix are padded by dummy ones or zeros in a process called pruning. If the R is the total number of rows and C is the total number of columns, then the RxC matrix will be filled starting with bit x_1 in column 1 of row 1 and the last bit x_{RC} (message or pruned dummy bit) in column C of row R as shown below:

$$Input = \begin{bmatrix} x_1 & x_2 & .. & x_C \\ x_{(R2)(C1)} & x_{(R2)(C2)} & .. & x_{(R2)(C)} \\ \vdots & \vdots & \vdots & \vdots \\ x_{(R)(C1)} & x_{(R)(C2)} & .. & x_{(R)(C)} \end{bmatrix}$$

Step 4:

The value of 'p' calculated in step 2 is used to read a primitive root 'v' from table in [3GPP99]. The value of 'p' and 'v' is required to calculate the Base Sequence (called s(j) in [3GPP99]). This base sequence is required to permute bits column wise. The calculation of the base sequence for intra-row permutation is shown below by a pseudo code:

```
For each P
Loop for j=1 to p-2
s(j) = v x s(j-1) mod p
End Loop
```

Step 5:

The prime sequence q_i is constructed using 'p'. The index 'i' is equal to the total number of rows of the interleaver matrix as shown by the following matlab code:

```
q(1) = 1;
for index6=2:R
        prime1 =find(gcd(prime_nos,p-1)==1);% gcd =>greatest common divisor
        prime2=prime_nos(prime1);
        prime3=find (prime2 >6 & prime2>q(1,index6-1));
        prime4=prime2(prime3);
        q(index6)=min(prime4);
end
```

'q_i' is used for permutation calculation of both rows and columns as shown in subsequent steps.

Step 6:

r(i) parameter which is used in intra row (column wise) permutation is calculated by using the using the prime sequence 'q_i' and T(i) patterns defined in table 3 in [3GPP99]. The matlab code is shown below:

```
for index7=1:R  %R is the total number of rows
   r(T(index7)+1)=q(index7);% r(.) is the permuted row index
end
```

Interrow (permutation between different rows of interleaver matrix) is performed by using the interrow permutation table defined in [3GPP99].

6.3.1.2 PHASE 2. CALCULATION PHASE

Step 1. IntraRow (Column-wise) permutation:

The column wise random permutation pattern is different for each row of the interleaver matrix. Columns are permuted according to two operations, first is the multiplication of r(i) (Step 6) with column count (0 to p-2). The modulation operation with the result of this multiplication and p (calculated in step 3) provides the index to base sequence calculated in step 4. The value of base sequence is used as index to permuted column as shown below:

```
for index8=1:R  % R is the total number of rows
for index9 =1:p-1  % p is approximately equal to the number of columns
   Permuted_Matrix_Index(index8,index9)=s(mod((index9-1)*r(index8),p-1)+1);
end
end
```

Step 2. Inter Row (row-wise) permutation

After performing the inter column permutation the inter row permutation of the interleaver matrix is performed by using fixed patterns defined in the standard [3GPP99]. This is explained by the following matlab code:

```
for index10=1:R
  Permuted_Matrix_Rows_changed(index10,:)=Permuted_Matrix(T(index10)+1,:);
end
```

Step 3. Data read with pruning

Finally, data is read column by column from the interleaver matrix and dummy bits are removed. The two dimensional interleaver matrix matlab algorithmic translation precedes by a code in matlab which replaces the interleaver matrix to one dimension suitable for hardware implementation. The complete design flow is shown in figure 6.3.

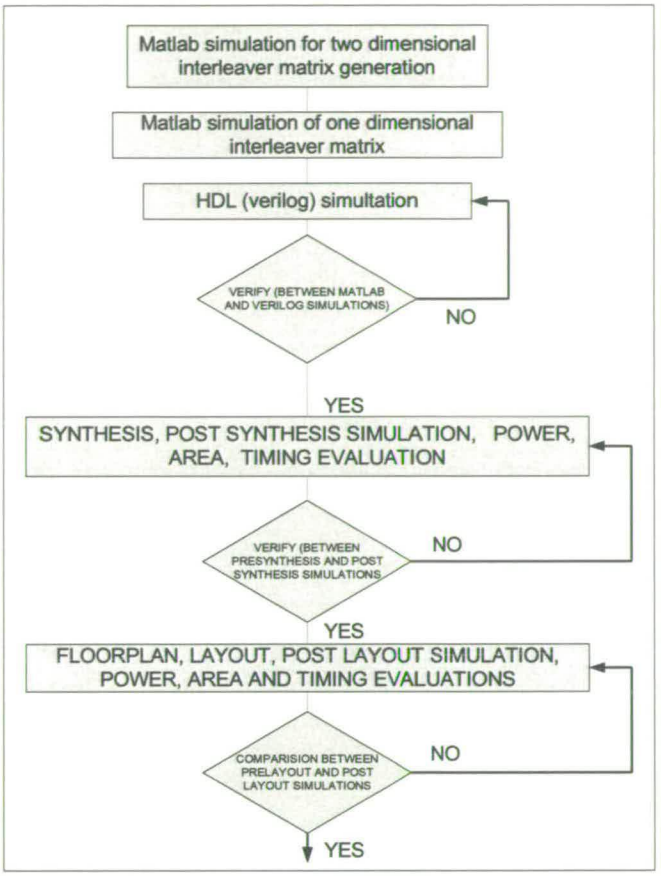

Figure 6.3. Design flow for interleaver implementation

Fig 6.4 shows the interleaver spread using this algorithm for K = 5114.

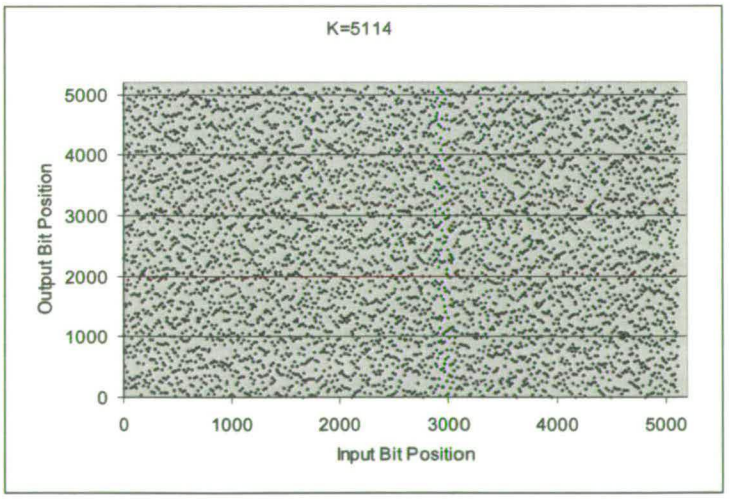

Figure 6.4. Interleaver spread for K=5114.

6.3.1.3 TRANSLATING THE INTERLEAVER MATRIX IN ONE DIMENSION

The dynamic interleaved address calculation requires the translation of interleaver matrix rows and columns into one unified address. A novel implementation methodology is presented that adjusts the 3GPP interleaver algorithm for efficient VLSI implementation. The traditional implementation methodologies for saving the entire interleaved address space can be avoided by just adding a column offset to the calculated address. This offset is calculated as under:

Column offset = (T(1,indexR) x C);

Where C is the total number of columns of the interleaved matrix, indexR is the row index and T(1,indexR) generates the interleaved row index. The calculation of interleaved addresses in one dimension is shown by the following matlab code:

```
%column by column calculations
if (C==p)
for indexC = 1:p-1
for indexR = 1:R
s_mult_factor(indexR,indexC) = mod((indexC-1)*r(indexR),p-1);
s_factor(indexR,indexC) = s(mod((indexC-1)*r(indexR),p-1)+1);
xing_factor(indexR,indexC)= (T(1,indexR)*C);
Corrected_addr(indexR,indexC)=s(mod((indexC-1) * r(indexR),p-
1)+1)+T(1,indexR)*C;
end
end
% logic for last column address if required
```

The other novelty in the implementation comes from the design of dynamic pruning logic which is discussed in section 6.5.

6.4 VLSI ARCHITECTURE

Our implementation approach reduces the size of the RAMs required to save the interleaved addresses. This is achieved by storing three pre-calculated values

instead of the entire sequence of interleaved addresses. The approach reduces the size of the RAM significantly and hence area and power. There is significant improvement in latency on frame changes as instead of changing the entire permuted addresses only a maximum of 256 base sequences are required to be changed and stored.

The architecture follows the same flow as was explained in section 6.3. In the preparatory phase the SRAMs are filled in with pre computed values that are required in the dynamic interleaved address calculation. These values differ for each frame length and hence RAMs are pre-populated whenever frame length changes. The three pre-calculated values stored in SRAMs are base sequence, ordered prime sequence and intra row permutation patterns as explained below:

6.4.1 BASE SEQUENCE

The base sequence calculation was shown in step 4 of section 6.3.1.1. All base sequences 's(j)' are pre computed for all 'j' between 0 and p-1. The size of the base sequence is approximately equal to the number of columns of the interleaver matrix.

6.4.2 ORDERED PRIME SEQUENCE

The prime sequence 'q_i' calculation was shown in step 5 of section 6.3.1.1 where 'i' is equal to the number of rows of the interleaver matrix.

6.4.3 INTER ROW PERMUTATION PATTERNS

Inter row permutation patterns 'T_i' are required in calculation of column offset as shown in section 6.3.1.3. These patterns as defined in [3GPP99] are also saved in SRAM. The individual components in the block diagram are explained below.

6.4.4 DUAL PORT SRAM FOR PARAMETER 'r(j)'

This RAM has synchronous single port write and asynchronous dual read ports and is used to store the permuted prime integers r(j). The values of r(j) change for

different frame lengths hence, it is pre-populated for each frame length. To compensate for pruning there are two data paths used in the design. The matlab calculation of this parameter was shown in step 6 of section 6.3.1.1 which depends on the number of rows of interleaver matrix. The maximum number of rows for 3GPP corresponds to a maximum frame length of 5114. The number of rows for this frame length is 20 which is the maximum value of index 'j' in the parameter 'r(j)'. The maximum permuted prime integer value corresponding to any 'r(j)' is therefore 79. Hence size of the RAM is selected as 20x8 bits. Matlab simulation calculates all the r(j) values and writes it in a file which is read by HDL test bench to be used in the preparation phase to pre populate this dual port RAM. r(j) and r(j+1) values are read from corresponding read ports – where j is the row number. The read counter starts from the first address to the last address (equal to the total number of rows) and r(j) values are read from the RAM. The read counter connected to the second read port reads the next address 'r(j+1)' which is required for pruning logic as explained in section 6.5.

6.4.5 8X8 MULTIPLIERS

As shown in matlab code of section 6.3.1.3 the first step in the dynamic address calculation is determining 's_mult_factor', a part of which requires multiplication of parameter 'r(j)' (section 6.4.4) with the total number of columns of the interleaver matrix. Another matlab simulation is performed to determine the maximum number of columns of the interleaver matrix for all frames. The maximum number of columns for all frame sizes is found out to be 255 which requires 8 bits. A 8x8 multiplier will be required to perform the multiplication of r(j) with the total number of columns in the interleaver matrix. It was shown in section 6.4.4 that r(j) values are saved in SRAM. There are two 8x8 Multipliers for Data path 1 and Data path 2 respectively and perform the multiplication of r(j) (or r(j+1)) with the number of columns. These multipliers are in the timing critical data path. To achieve the desired frequency they are implemented in parallel multiplication schemes (radix-2 booth algorithm).

6.4.6 MODULUS CALCULATOR

Modulus calculator performs the modulus operation for the result calculated in section 6.4.5 and the number of columns. Modulus operation is performed using Synopsys design ware library modulus component and result from the modulus operation form the address of SRAM as shown below in section 6.4.7.

6.4.7 DUAL PORT SRAM FOR PARAMETER 's(j)'

The parameter 's(j)' is calculated using the equation as described in [3GPP99] and shown earlier in Step 4 of section 6.3.1.1. It is repeated here for coherence of presentation:

$$s(j)=(v \times s(j-1)) \bmod p,\ j=1,2,..\ (p-1) \text{ and } s(0)=1 \qquad (1)$$

s(j) constitutes the base sequence for intra row permutation and varies with the total number of rows corresponding to different values of K. Therefore for each K this SRAM is pre-populated as was shown in the preparatory phase (refer section 6.3.1.1). The maximum elements of s(j) for all K is 256 which is equal to the maximum number of columns of interleaver matrix (for K=5114). This SRAM is also dual read port for the corresponding two data paths.

The result read from this SRAM corresponds to the mathematical operation defined in [3GPP99] as

$$Ui(j) = s((j \times ri) \bmod (p-1)) \qquad (2)$$

The interleaved address is constituted when the data is read out from the interleaver matrix column wise. Therefore, for interleaved address calculation equation (2) is also calculated and implemented column wise. The final address is computed by adding an offset to equation (2) as explained in section 6.3.1.3. This offset address is the inter row permutation transformed from two dimensions to one dimension as defined below.

$$\text{Interleaved address} = s(j) + T[\text{Row\_Number}] * \text{Columns}$$

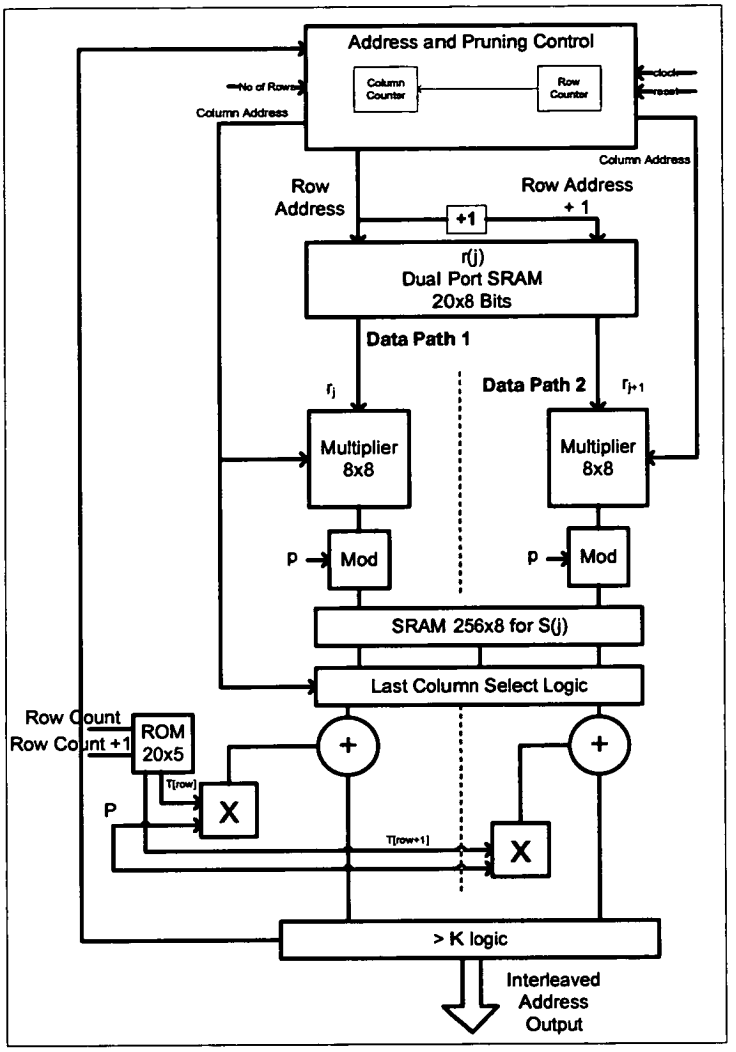

Figure 6.5 Block Diagram of the interleaver implementation

6.4.8 INTER ROW PERMUTATION

The inter row permutation is based on a small table represented in [3GPP99] by T[Row_Number]. SRAM (20 x 5) is used to save this parameter. The maximum size of this memory corresponds to the frame size of K=5114 having 20 rows. These 20 possible permutations can be represented by 5 bits; hence the size of the ROM is 20x5. This exactly corresponds to C-fold decimation of the input sequence with the appropriate phase s(j) [WEL04].

6.5 PRUNING

The algorithm in [3GPP99] calculates the number of rows and columns of interleaver matrix for each block size. The address pruning occurs when the block size is not equal to "Rows x Columns" of the interleaver matrix. Therefore, dummy bits are added to make them equal. Pruning is a major problem for real time address computation as these dummy bits have to be pruned away from the final interleaved addresses. Matlab simulations were performed to calculate the maximum number of pruned bits for all frames K between 41 and 5114. The message size with maximum number of pruned bit is calculated as K=2281. It is also observed through simulations that inter row permutation patterns ensure that the two unused interleaved addresses are never consecutively placed when reading the interleaver matrix column wise. This is shown in figure 6.6 below.

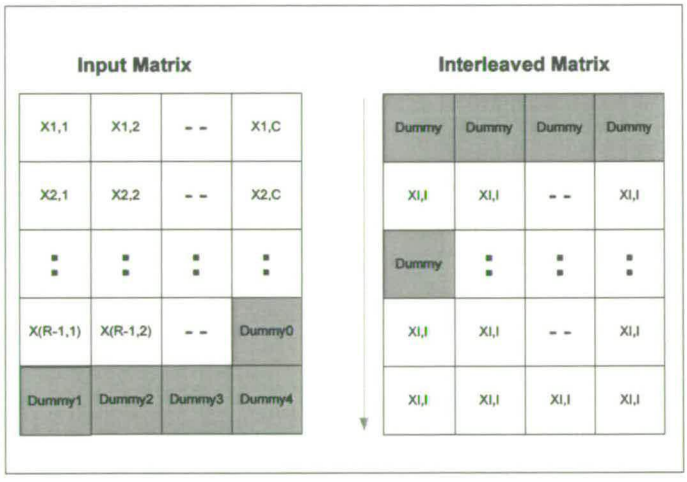

Figure 6.6. Dummy bits position in interleaving

The observation leads us to conclude that whenever there is a pruned (dummy) bit in the interleaved matrix the next bit will always be a valid interleaved bit. Therefore there are two parallel data paths designed to overcome the effects of pruning. Data Path 2 is always calculating the next interleaved address from the one which is calculated from data path1. The dynamic interleaved address

calculation is performed column wise and the pruned address is checked by the following pseudo code

```
If (column wise calculated interleaved address > K)
    Address = dummy (pruned)
    Skip bit and get the next calculated address
    else
    perform the calculation as regular
```

Therefore if Data path1 address corresponds to the address of dummy bits the logic selects the data path 2 value i.e., the next interleaved address.

6.5.1 PRUNING CONTROL LOGIC

The implementation control flow is by use of counters controlling the read addresses of SRAMs. The counters (shown in next section) increment the read address by one and count to the terminal address which in turn depends on the frame size K. Each time dummy bit is detected and Data Path 2 is selected the row counter in the pruning logic increases the count by 2 instead of normal increments of 1. The next input addresses will skip this dummy pruned address and the mechanism will effectively remove all the pruned dummy addresses.

6.5.2 ADDRESS CONTROLLER

The column count is also provided by a binary up counter. The columns counter increments when the row counter reaches its terminal (maximum) count. The maximum number of rows for any frame size is 20 and therefore row counter is 5 bits. The row counter feeds the 8 bits column counter. The 8 bits of column counter corresponds to the maximum number of columns (equal to 256) of interleaved matrix. Whenever dummy bit is detected and data path two is selected it is fed back to row counter. In order to offset the dummy address the row counter will in turn skip one address. The counter arrangement is shown by figure 6.7 below:

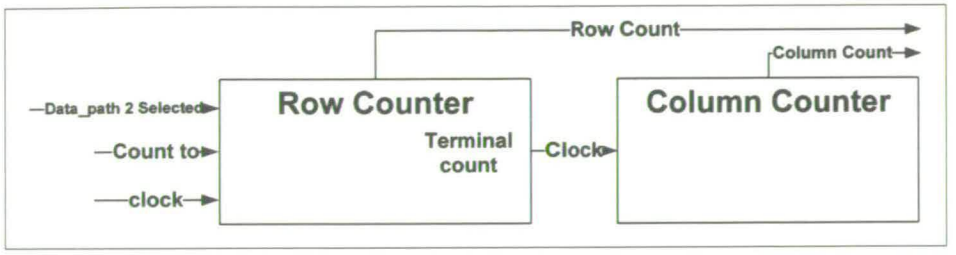

Figure 6.7. Address Generation and Pruning control Logic

6.6 RESULTS

The design is synthesized using Synopsys Design Compiler for 0.18 microns CMOS UMC cell library and chip layout done on Silicon ensemble. Post layout power figures are taken from Synopsys Design Power by inputting the toggling activity for the maximum frame size K =5114. Results are compared with a typical 5114x13 Synopsys design ware SRAM which would otherwise have been used if implementation had adopted the look up table based design. The results are also compared with some 3GPP turbo decoder implementations. The implementation in [MAS99] uses 12 K words memory for interleaver and [PEN03] uses 27 K (total) memory. Similarly any design that implements the interleaver as memory will benefit from this presented approach. Table 6.9 below shows the overall synthesized area and post layout power figures for 21.7 MHz.

Technology	**0.18 microns standard cell CMOS**
Operating frequency	21.73 MHz
Total Power in mW	36.06
Total Area	771925.88 um^2

Table 6.9 Results of implemented interleaver

Figure 6.8 shows the critical timing components in data path. Comparing the design with 5114x13 SRAM (the traditional approach [MAS99, PEN034]), the area of 5114x13 SRAM is 30083782 um^2, which is 38.9 % more than our proposed scheme.

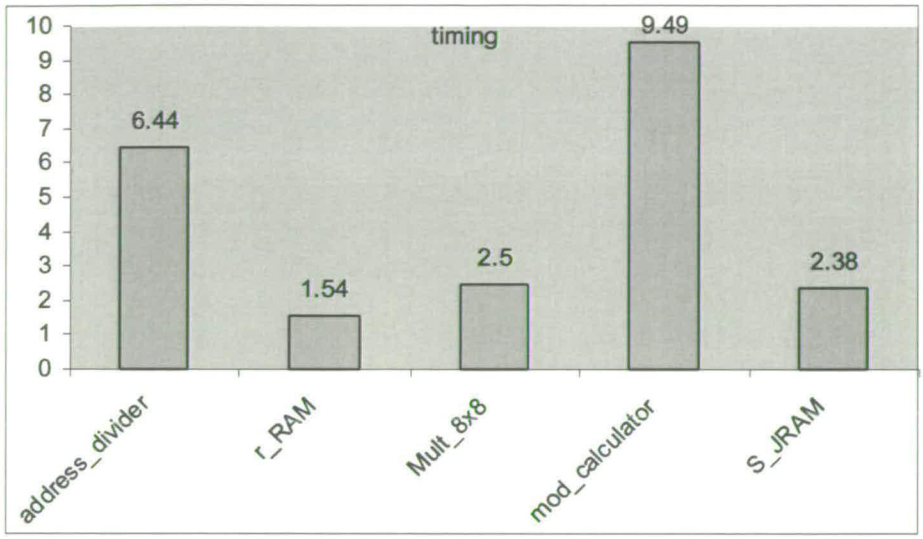

Figure 6.8. Synopsys Prime time critical path components shown (nano seconds)

Figure 6.9 below shows the area and power results of the individual components of the design. The largest contributor to the area and power is the largest RAM S(j) (256x8) in the design.

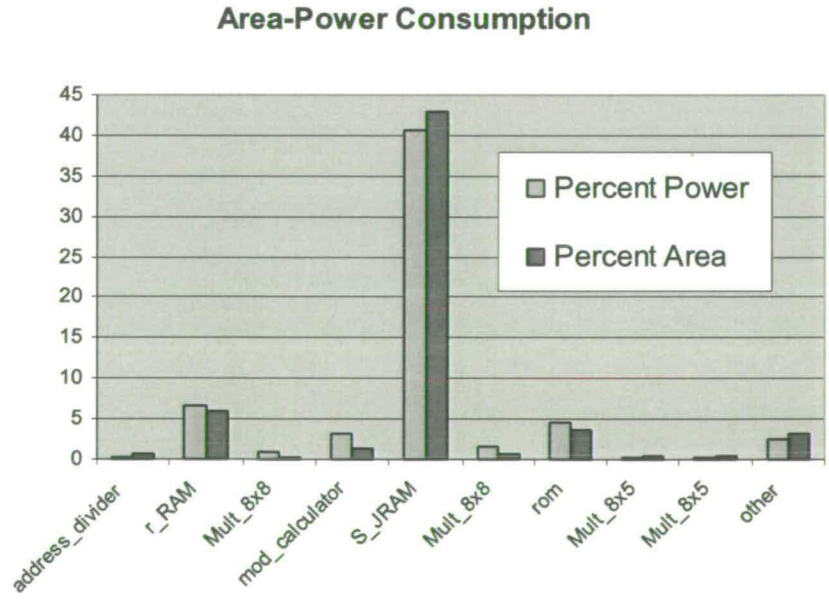

Figure 6.9 Area and Power results of individual components of the design

There is 30% improvement in power as compared to the reference SRAM design.

6.7 CONCLUSION

In this chapter the requirement of efficient implementation for 3GPP [3GPP99] interleaving algorithm is introduced. The allowable frame lengths in 3GPP [3GPP99] standard is quite large (40-5114) and the traditional approach of storing the entire interleaved address space will result in interleaver consuming large power and area. The current implementation has addressed the power aspects of such designs in the most efficient way. A comparison with the existing turbo decoder designs is provided justifying the requirement of low energy solution for the interleaver. It was shown that it is more cost effective to store just the permutation patterns and not the entire interleaved addresses. The design flow was shown from the concept to Silicon layout. The novelty of this approach is in the efficient implementation of 3GPP interleaving algorithm providing an alternative to VLSI design engineers which is efficient in both area and power. The proposed architecture replaces the 5K – 13 bit SRAM with the hardware which is 38.9 % and 30% more efficient in area and power respectively.

Chapter 7

SUMMARY AND CONCLUSIONS

7.1 INTRODUCTION

The aim of this thesis is to investigate an efficient reconfigurable architecture for convolutional forward error correction. The key blocks investigated for reconfigurable performance evaluations are the viterbi and turbo decoders. The reconfiguration is designed with an aim to improve not only the power and timing but also to maintain maximum flexibility for the given domain. The unified array is implemented in 180nm CMOS process technology. There is also a novel low power implementation proposed for 3GPP S-Random block interleaving which is used alongside turbo decoding array.

This chapter is organised into four sections. The first section summarises the content of the thesis and identifies the contributions. The second section draws conclusions from the work presented in this thesis. Final remarks are described in the third section and the last section outlines areas for future investigation.

7.2 SUMMARY OF THESIS

This thesis investigates flexibility in high performance convolutional FEC systems for a common multi-standard communication platform. The traditional reconfigurable architectures suffer from relatively poor performance owing to their high flexibility. By reducing the redundant flexibility in the traditional approaches a flexible, low power and high speed FEC solution is introduced that meets the performance constraints imposed by these standards.

Chapter 2 provides an overview of commercially available reconfigurable architectures and some of the existing concepts in literature for the design of

reconfigurable logic elements and their interconnects. This review highlighted the importance of defining the domains of reconfigurability with restricted freedom so that performance can be improved. This chapter also showed the logic elements and the interconnect evolution with time and availability of such commercial devices that introduce flexibility in a very well defined domain. This chapter also linked these improvements with examples from commercial FPGAs. This chapter showed reconfigurable techniques and methodologies used in current state of the art commercial devices with aim to extort this information for any high performance reconfigurable design.

In chapter 3 the reconfiguration focus shifted towards domain specific architectures in the literature. The useful domain specific reconfiguration techniques used in these architectures are explained in detail to use the information for current domain specific design. This chapter demonstrated that the reconfigurable functional units can be defined large and complex; however their granularity should be well-matched to the data types and the computations required by target algorithms. The chapter again reemphasised that the architectures that target a smaller set of applications can be more efficient than general-purpose devices and must be pursued.

Chapter 4 looked into turbo decoding, the critical path delays and the typical power consumptions. The chapter also looks at the similarities and differences in the design of decoders as used in various communication standards. The information is used to introduce flexibility in the key locations of the decoder data path. The cost of reconfiguration is measured and tailored to achieve the performance imposed by these standards. An efficient control strategy is proposed in hardware that avoids the use of microprocessors to control the array and hence makes the array possible to be used as a standalone unit. The control is also designed for reconfiguration between not only multiple standards within turbo but also for unified turbo and viterbi decoding. An efficient low power technique proposed by caching the two window lengths of input metrics which reduces the read accesses for the larger input RAMs. The proposed reconfigurable methodology for input RAMs allows them to be reutilized for storage

of configuration bits in viterbi. A technique to avoid branch metric storage is also proposed. Finite precision analysis for turbo decoder under varying reconfiguration requirement is performed which emulates the hardware design in matlab for efficient BER analysis. A new matrix normalization scheme is also proposed that makes the turbo array compatible with viterbi decoding. The implemented reconfiguration topology keeps individual decoding components like forward state metrics, reverse state metrics and branch metric blocks completely flexible. The reconfigurable design also keeps the decoding trellis flexible for different rates, generator polynomials, constraint lengths and frame sizes.

Chapter 5 provided various novel concepts in reconfigurable viterbi decoder design. A new reconfigurable path history memory management and segmentation technique for multiple standards is proposed. This is combined with a reconfigurable write address generation mechanism designed in hardware. A novel trace back approach is also proposed that provides the multi standard reconfigurability with a low power implementation. A novel technique for reading and writing path history memories is also implemented that adjusts efficiently in a multi standard environment. The reconfigurable work on viterbi also introduces a novel mechanism for controlling and storing the configuration bits that reduces the time to reconfigure the array. Like the turbo decoding array the viterbi components also provide an open trellis arrangement which makes the trellis flexible for different constraint lengths, rates, generator polynomials and frame sizes. The control of the viterbi components is provided by a flexible reconfigurable finite state machine that avoids the use of separate microprocessor for controlling the array.

Chapter 6 proposed a novel low power implementation methodology for 3GPP interleaver. The proposed technique avoids the use of SRAMs for storing the entire interleaved sequences. This results in big savings in area and energy. The work in this chapter introduces a dynamic interleaver address calculation scheme and an effective mechanism for address pruning. The overall implementation has much improved energy than the traditional SRAM implementations of the 3GPP interleaver.

7.3 SUMMARY OF ACHIEVEMENTS

The main achievements of this work are the following:

- Development of a unified reconfigurable viterbi-turbo decoding array for a large number of communication standards. The Construction of such array requires identification of areas where flexibility should be introduced to make the overall array achieve the performance constraints as imposed by different standards.
- An open trellis implementation of both turbo and viterbi decoders is presented individually and in the unified approach.
- The viterbi implementation has novel path history segmentation and management approach combined with a reconfigurable trace back and write address processors for multiple standards. The reconfigurable aspects of viterbi decoder are carefully designed for a very fast context switch between different standards.
- Both viterbi and turbo processing is controlled by a reconfigurable finite state machine which is configurable not only for the control of individual standard mappings but also in the unified approach. The system flexibility is carefully tailored to give the best performance results in area, power and speed.
- The work on reconfigurable turbo decoder showed an efficient low power input memory management and branch metric calculation scheme. A new open trellis structure for reconfigurable turbo decoding for multiple standards is also proposed.
- A new approach for implementing S-Random interleaver as defined in 3GPP specification is also proposed. The novel implementation produces a much reduced memory implementation for interleaving and a new technique for hardware pruning.

7.4 FINAL REMARKS

This work represents a step forward in the area of high performance reconfiguration for convolutional forward error correction. The results of arrays synthesised in 180nm

process technology show that the reconfigurable array safely meets the performance constraints imposed by the target communication standards.

7.5 FUTURE WORK

This thesis has tried to provide a through investigation into the research proposal of reconfigurable FEC decoder suitable for a SDR communication platform. However, a number of additional issues can be explored which might further add to knowledge gained from the research presented. The additional issues that can form extension to this work are highlighted as follows:

- The template for a unified baseband processor can be further extended to include other base band blocks such as FIR filters, FFT and iFFT components and MIMO detection components to produce an overall reconfigurable baseband receiver.
- The work can also be combined with contention free parallel interleavers for example, the recently proposed Quadratic Permutation Polynomial – QPP by Takeshita [TAK06] and description given earlier in [COS04]. Our research can be extended in the new Long Term Evaluation (LTE) [3GPPr8] proposal for 3GPP to design a higher speed decoder using the maximum contention free property of interleavers in [COS04]. The performance of this new interleaver is shown to be better than S-Random interleavers in [MOT06] with added benefit of contention free access for fast decoding. [3GPPr8] describes the parameters of such interleavers required in 3GPP LTE.
- The research can be extended for reconfigurable design space exploration for forward error correction in very high performance and low power consumption for example; Gb/sec throughput short distance wireless applications like Ultra Wide Band UWB specification by ECMA [UWB05].
- Parallel placement of turbo decoder blocks in our reconfigurable template can produce very high throughputs however for Gb/sec throughputs proposed in the above standards [UWB05] but an investigation is required to achieve low power consumption. Since the size of the array of parallel turbo decoding (for

Gb/s rates) will become prohibitively large an investigation to design more efficient reconfiguration is required.

- Joint source and channel coding can achieve better results, and therefore reconfiguration approach can be extended in this direction to combine source coding reconfigurable architecture with modulation.
- Static power consumption becomes an important parameter for lower process geometries (<90nm) and techniques needs to be investigated to reduce static power consumption as well as dynamic power consumption.
- Efficient stopping criteria in turbo codes need to be combined with techniques such as dynamic voltage scaling to design a variable speed decoder which can control the iterations more interactively.
- A further investigation can also look to produce a power efficient template for wireless as well as non battery powered turbo application as in optical/ magnetic storage and fibre optics.
- There is further scope to experiment on suitable VLIW, SIMD or MIMD processor and the integration of the array either as a dedicated co processor or ALU of the processor data path.

REFERENCES

[3GPP99] Third Generation Partnership Project, Technical Specification Group-Radio Access Network, Multiplexing and channel coding (FDD), 3G TS 25.212 v3.3.0

[3GPPr8] www.3GPP.org, Technical Specification Group Radio Access Network: Multiplexing and channel coding Release 8, 3GPP TS 36.212

[ABWEB] R. Abielmona, "Alphabetical List of Reconfigurable Computing Architectures", http://www.site.uottawa.ca/~rabielmo/personal/rc.html

[ABD06] Abdul-Shakoor et al., "A high performance soft decision Viterbi decoder for WLAN and broadband applications" in Canadian Conference on Electrical and Computer Engineering, May 2006, pp 2468-2471

[ADR05] Adrian Cosoroaba and Frederic Rivoallon, "Achieving Higher System performance with the Virtex-5 Family of FPGAs", White Paper: Virtex-5 family of FPGAs www.xilinx.com

[AGR99] O. Agrawal, et. al., "An innovative, segmented high performance FPGA family with variable-grain-architecture and wide-gating functions," in Proc. IEEE Field Programmable Gate Arrays (FPGA), Monterey, CA, 1999, pp. 17–26

[AHM07] I Ahmed, T Arslan, "Reconfiguration requirement for convolutional forward error correction decoding for 3G and Beyond", journal submitted to IEEE VLSI transaction" journal submitted to IEEE VLSI transaction

[AHM06] I Ahmed, T Arslan, "A reconfigurable viterbi trace back for implementation on Turbo Decoding Array", in IEEE International SOC Conference, 2006. Publication date: Sept. 2006 Page(s) 107-108

[AHM106] I Ahmed, T Arslan, "A reconfigurable viterbi decoder for a communication platform" in IEEE International FPL Conference, 2006. Publication date: Aug 2006 on pages 1-6

[AHM05] I Ahmed, T Arslan, "VLSI Design of Multi Standard Turbo Decoder for 3G and Beyond" in 12th International IEEE ASP-DAC Conference Jan 23-26, 2007

[AHM105] I Ahmed, T Arslan, "A Low Energy VLSI Design of Random Block Interleaver for 3GPP Turbo Decoding", in IEEE international Symposium on circuits and systems ISCAS 2006 publication date 21-24 May 2006

[AHM205] I. Ahmed, T. Arslan, "Improved Memory Strategy for Log Map turbo decoders,". SOC Conference, 2005. Proceedings IEEE international pages 103-104, Sept 25-28 2005

[AHM104] I Ahmed, T Arslan, "Efficient implementation of Mobile Video Computations on Domain Specific Reconfigurable Arrays" in Proceedings IEEE Design, Automation and Test in Europe (DATE) conference in 2004. Publication date: 16-20 Feb. 2004 on pages1833

[AHM204] I Ahmed, T Arslan, "Video transmission through domain specific reconfigurable architectures over short distance wireless medium utilizing Bluetooth IEEE 802.15.1/spl trade/standard" in Proceedings IEEE international SOC Conference 2004, publication date: 12-15 Sep 2004 on pages 7-10

[AHR90] M. Ahrens, et. al., "An FPGA family optimized for high densities and reduced routing delay," in Proc. 1990 CICC, M pp. 31.5.1-31.5.4, May 1990

[ALB94] O.T. Albaharna, P. Y. K. Cheung, and T. J. Clarke, Area and Time limitations of FPGA-based virtual hardware", Proceedings of the IEEE Interntational Conference on Computer Design, October 1994, pp. 184-189

[ALT01] http: //www.altera.com/ products / devices /stratix2/features/dsp/perf/st2-dsp_performance.html

[ANC00] University of Ancona (2000) Study of Bandwidth-Efficient Coding Schemes for Near-Earth Applications. ESA/ESOC Contract No. 14128/00/D/SW – Final Report

[ANC01] University of Ancona (2001) Highly Efficient Channel Codes for High Data Rate Missions. ESA/ESOC Contract No. 15048/01/D/HK(SC) – Final Report

[ANG05] F. Angarita et al., "Efficient Mapping on FPGA of a Viterbi Decoder for Wireless LANs" in IEEE workshop on Signal Processing Systems Design and Implementation 2-4 Nov. 2005 pp. 710-715

[ANN1] "FPGA area vs. cell granularity—PLA cells," in Proc. Custom Integrated Circuits Conf., May 1992, pp. 4.3.1–4.3.4

[ANN02] "Wormhole Run-Time Reconfiguration: Conceptualization and VLSI Design of a High Performance Computing System"

[ATL03] I. Atluri, T. Arslan, "Low power VLSI implementation of the MAP decoder for turbo codes through forward recursive calculation of reverse state metrics" IEEE Int. SOC Conf. 17-20 Sept. 2003, pp. 408-411

[BAH74] L. R. Bahl, J. Cocke, F. Jelinek and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," IEEE Trans. Inform. Theory, IT-20, pp. 248-287, Mar.1974

[BAR94] S. A. Barbulescu and S. S. Pietrobon, "Interleaver design for turbo codes", Electronics Letters, Vol 30, No 25, Dec. 1994

[BAR95] S. A. Barbulescu and S. S. Pietrobon, "Terminating the trellis of turbo codes in the same state", Electronics Letters, Vol. 31, No. 1, pp.22-23, Jan. 1995

[BAR96] S. A. Barbulescu et al , " Iterative decoding of turbo codes and other concatenated codes" PhD dissertation, university South Australia, pp. 23-24, 1996

[BAT87] G. Battail, "Ponderation des symoles decodes par l'algorithme e Viterbi", Ann. Telecommun., Fr., 42, N 1-2, pp. 31-38, Jan. 1987

[BEN96] S. Benedetto et al, "Soft-output decoding algorithms for continuous decoding of parallel concatenated convolutional codes", Proceedings of ICC'96, Dallas, Texas, June 1996

[BER99] C. Berrou et al, "Multiple parallel concatenation of circular recursive convolutional (CRSC) codes" Annals of Telecommunication, pp 166-172, Mar-Apr. 1999

[BLA92] P. J. Black and T. H.-Y. Meng, "A unified approach to the Viterbi algorithm state metric update for shift register processes," in Proc.IEEE Int. Conf. Acoustics, Speech and Signal Processing, Mar. 1992, pp. 629–632

[BH00a] Macro Breiling and Lajos Hanzo, "The Super-Trellis Structure of Turbo Codes", IEEE Trans. Inform. Theory, pages 2212-2228, 2000

[BEN196] S. Benedetto and G. Montorsi, "Unveiling turbo codes: some results on parallel concatenated coding schemes", IEEE Transactions on Information Theory, Vol. 42, No. 2, pp.409-428, March 1996

[BEN296] S. Benedetto et al, "Serial concatenation of interleaved codes: performance analysis, design and iterative decoding", JPL TDA Progress Report 42-126, Aug 1996

[BEN396] S. Benedetto et al, "Soft input Soft output MAP module to decode parallel and serial concatenated codes" in TDA Progr. Rep. 42-127, Jet Propulsion Lab, Pasadena, CA, pp. 1-20, 1996

[BER93] C. Berrou, A. Glavieux and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes". Proc ICC, Geneva, Switzerland, 1993, pp. 1064-1070

[BER96] C. Berrou, S. Evans and G. Battail, "Turbo block codes", Proceedings of Seminar on Turbo Coding, Lund, Sweden, pp.1-7, Aug. 1996

[BIC03] M. Bickerstaff et al, "A 24 Mb/s radix-4 LogMap turbo decoder for 3GPP-HSDPA mobile wireless" International Solid State Conference, ISSCC, Feb 11, 2003, Session 8, paper 8.5

[BIR99] J. Birkner, A. Chan, H. T.Chua, A. Chao, K. Gordon, B. Kleinman, P. Kolze, and R. Wong, " A very high-speed field programmable gate array using metal-to-metal antifuse programmable elements," in New Hardware Product Introduction at CICC' 99

[BIT96] R. A. Bittner et al.: Colt: An Experiment in Wormhole Run-time Reconfiguration; SPIE Photonics East '96, Boston, MA, USA, Nov. 1996

[BLA92] P. Black et al, " A 140 mb/s 32-state radix-4 viterbi decoder", IEEE Journal of Solid-State Circuits, vol. 27, no. 12, pp. 1877-1885, December 1992

[BLA93] P. Black and T.Y. Meng, "Hybrid survivor path architectures for viterbi decoders", Proc. ICASSP, pp. 433-436, 1993

[BLA97] P. Black et al, "A 1-gb/s, four-state, sliding block viterbi decoder", IEEE Journal of Solid-State Circuits, vol. 32, no. 6 pp. 797-805, June 1997

[BOG03] B. Bougard et al, "A Low-Power High Speed Parallel Concatenated Turbo-decoding Architecture" 3rd Int. Symp. on Turbo Codes and Related Topics, Brest, France, 2003, pp. 511-514

[BOU03] E. Boutillon, W. J. Gross, G. Gulak, "VLSI architectures for the MAP algorithm". IEEE Transactions on Communications, Vol. 51, No. 2, pp. 175-185, February 2003

[BRU04] Bruels, N et al, "A 2.8Gb/s, 32-state, radix-4 Viterbi decoder add-compare-select unit", in IEEE Symposium on VLSI Circuits, 17-19 June 2004 pages 170-173

[CAR86] W. Carter et al., "A user programmable reconfigurable gate array," in Proc. CICC, May 1986, pp. 233-235

[CAR04] Carl Ebeling et al, "Implementing an OFDM receiver on the RaPiD Reconfigurable Architecture" IEEE Trans. On Computers, Nov 2004, Vol 53, Issue 11 pp. 1436 - 1448

[CAV03] Cavallaro J. R., Vaya M "Viturbo: a reconifigurable architecture for Viterbi and turbo decoding" IEEE ICASSP 2003, 6-10 April 2003, Vol 2, pages: II-497-500

[CCS03] Consultative Committee for Space Data Systems (2003) TM Synchronization and Channel Coding. Blue Book 131.0-B-1

[CHA89] C.-Y. Chang and K. Yao, "Systolic array processing of the Viterbi algorithm," IEEE Trans. Inform. Theory, vol. 35, pp. 76–86, Jan. 1989

[CHA92] P. M. Chau and K. J. Stephen, "Scaling and folding the Viterbi algorithm trellis," in VLSI Signal Processing V, pp. 479–489, Oct. 1992

[CHA00] Y. N. Chang et al, "A 2-Mb/s 256-state 10mW rate-1/3 Viterbi decoder", IEEE Journal of Solid-State Circuits, vol. 35, no. 6, pp. 826-834, June 2000

[CHA01] K. Chadha and J. Cavallaro, "A reconfigurable Viterbi Decoder Architecture" 35th Asilomar Conference on Signals, Systems and Computers, 2001, vol 1, pp. 66-71

[CHE92] D. C. Chen et. al., "A Reconfigurable Multiprocessor IC for Rapid Prototyping of Algorithmic-Specific High Speed DSP Data Paths" IEEE J. Solid-State Circuits, Vol 27. No. 12, Dec 1992

[CHU89] C. Y Chung and K. Yao, "Systolic array processing of the Viterbi algorithm," IEEE Trans. Inform. Theory, vol. 35, no1, pp. 76-86, Jan 1989

[CHWEB] Chameleon Systems Corp. website http://www.chameleonsystems.com/

[COM99] K. Compton, and S. Hauck, "Configurable Computing: A Survey of Systems and Software", Northwest University, Dept. of ECE, Technical Report, 1999

[CON87] Consultative Committee for Space Data Systems (CCSDS), "Recommendations for Space Data Systems Standard: Telemetry Channel Coding, Blue Book Issue 2, CCSDS 101.0-B2 edition, Jan 1987

[COS04] Costello, D. J., et al "Contention-free interleavers" In IEEE proceedings ISIT 2004, 27 June – 2 July 2004, pp 54

[CRO99] D. C. Cronquist et al., "Architecture design of reconfigurable pipelined data paths", Proceedings of the 20th Anniversary conference on advanced research in VLSI, March 1999, pp. 23-40

[DAM03] M. O. Damen, et al, "Linear threaded algebraic space-time constellations." IEEE Trans. Inform. Theory, pp 2372-2388, Oct. 2003.

[DAN95] F. Daneshgaran and K. Yao, "The iterative collapse algorithm: A novel approach for the design of long constraint length Viterbi decoders—Part I and Part II," IEEE Trans. Commun., vol. 43, pp. 1409–1418, 1419–1428, Feb.–Apr. 1995

[DAN98] F. Daneshgaran and M. Monin, "Iterative interleaver growth algorithms of polynomial complexity for turbo codes" , IEEE International Symposium on Information Theory, Boston, USA, Aug. 1998

[DAR98] Darren C. Cronquist et al., "Specifying and compiling applications for RAPID", IEEE Symposium on FPGAs for Custom Computing machines April 1998, pp. 116-125

[DER05] Derek Curd, "Power Consumption in 65 nm FPGAs" White paper Virtex-5 family of FPGAs, www.xilinx.com

[DEV90] Devereaux C. Chen, "Programmable Arithmetic Devices for High Speed Digital Signal Processing" VLSI Signal Processing IV, IEEE Press 1990

[DIW95] H. Diwid et al, "Real-time algorithms and VLSI architectures for soft output MAP convolutional decoding" Proc. Personal, Indoor, and Mobile Radio Communications, PIMRC'95, vol. 1, 1995, pp. 193-197

[DUR01] Lisa Durbec and Nick Macias, The cell Matrix: An architecture for nanocomputing, Nano technology (2001), 217-230

[EBE96] C. Ebeling et al., "Rapid – reconfigurable pipelined datapath, Field-Programmable Logic: Smart Applications, New Paradigms, and Compilers". 6th international Workshop on Field-Programmable Logic and Applications, Sep 1996, pp 126-135

[ELE04] E. Eleftheriou et al, "Application of capacity-approaching coding techniques to digital subscriber lines" IEEE Communications Magazine, vol. 42, No. 4, pp. 88-94, April 2004

[ELI04] Elias Ahmed and Jonathan Rose, " The Effect of LUT and Cluster Size on Deep-Submicron FPGA Performance and Density" IEEE Trans on VLSI systems, vol 12, No 3, March 2004

[ELI55] P. Elias. "Coding for Noisy Channels". IRE Conv. Record, 4:37-47, 1955

[ERF94] J. Erfanian et. al., "Reduced complexity symbol detectors with parallel structures for ISI channels," IEEE Transactions on Communications, vol. 42, pp. 1661-1671, 1994

[ESA89] European Space Agency (ESA). PSS-04-103, "Telemetry Channel Coding Standard, Sept. 1989"

[EST] G. Estrin, B. Bussel, R. Turn, and J. Bibb, "Parallel processing in a restructurable computer system" IEEE Transactions on Electronic Computers1

[ETS94] European Telecommunications Standards Institute, "Digital broadcasting system for television, sound and data services" ETS 200 421, 1994

[ETS00] European Telecommunications Standards Institute, "Digital video broadcasting (DVB); interaction channel for satellite distribution systems" ETSI EN 301 790 V1.2.2 (2000-12), 2000.

[FAB01] Fabian Luis Vargas et. al., "A FPGA-based Viterbi Algorithm implementation for speech recognition systems" in IEEE ICASSP conference, 7-11 May 2001, pp. 1217-1220

[FEN99] W. Feng, J. Yuan and B. Vucetic, "A code matched interleaver design for turbo codes", Proceedings International Symposium on Personal, Indoor and Mobile radioCcommunications (PIMRC), Osaka, Japan, pp. 578-582, Sep. 1999

[FET91] G. Fettweis et al, "Feedforward architectures for parallel viterbi decoding", Journal of VLSI Signal Processing, vol 3, pp. 105-119, 1991

[FEY93] G. Feygin, P. Gulak, and P. Chow, "A multiprocessor architecture for Viterbi decoders with linear speedup," IEEE Trans. Signal Processing, vol. 41, pp. 2907–2917, Sept. 1993

[EPR01] Gerard K. Rauwerda, "Reconfigurable Turbo/Viterbi Channel Decoder in the Coarse-Grained Montium Architecture", University of Twente, department EEMCS, eprints.eemcs.utwente.nl

[FOR71] G. D. Forney Jr., "Burst-correcting codes for the classic bursty channel", IEEE Transactions on Communications, Vol. 19, No. 5. pp.772-781, Oct. 1971

[FOR73] J. Forney, "The Viterbi Algorithm", Proceeding of the IEEE, vol. 61, pp. 268-78, March 1973

[FRE86] N. J. P. Frenette et al., "Implementation of a Viterbi processor for a digital communications system with a time-dispersive channel," IEEE J. Select. Areas Commun., vol. SAC-4, pp. 160–167, Jan. 1986

[GAM04] H. El Gamal, et al ,"Lattice coding and decoding achieve the optimal diversity-vs-multiplexing tradeoff of MIMO channels." IEEE Trans. Inform. Theory, 2004

[GAR89] A. El Garnal, et al., " An architecture for electrically configurable gate arrays," IEEE JSSC, vol. 124, No. 2, pp. 394-398, Apr. 1989

[GAR01] Garello R, Benedetto S, Pierleoni P (2001), "Computing the free distance of turbo codes and serially concatenated codes with interleavers" IEEE J. on Selected Areas on Communications 19, pp. 800-812

[GEO99] V. George, H. Zhang, J. Rabaey, "The Design of a Low Energy FPGA," International Symposium on Low Power Electronics and Design, pp. 188-193, 1999

[GIL03] F. Gilbert et al, "Communication centric architectures for turbo-decoding on embedded multiprocessors," in Proc. Design, Automation, Test Eur. Conf., Mar. 2003, pp. 356–361.

[GLI87] S. C. Glinski, et. al., "A processor for graph search algorithms" in Proc. ISSCC`87, New York, 1987, pp. 162-163

[GMB00] Roberto Garello, Guido Montorsi, Sergio Benedetto, and Giovanni Cancellieri, "Interleaver properties and their applications to the Trellis Complexity Analysis of Turbo Codes" IEEE Trans. Commun., 49(5):793-807, May 2001

[GMB01] Roberto Garello et. al., "Interleaver Properties and their Applications to the Trellis Complexity Analysis of Turbo Codes" IEEE Trans. Commun., 49(5):793-807, May 2001

[GOL02] S.C. Goldstein, et al., "PipeRench: a reconfigurable architecture and compiler", IEEE Computer Apr 2000, Volume 33, Issue 4 pages 70-77], [Herman Schmit et al., Piperench: A virtualized programmable datapath in 0.18 micron technology, Proceedings of the IEEE Customer Integrated Circuits Conference, May 2002, pp. 63-66

[GOL99] S. C. Goldstein et al., "PipeRench: A Coprocessor for Streaming Multimedia Acceleration", Proc ISCA' 99, Atlanta, May 2-4, 1999

[GUL88] P. G. Gulak and T. Kailath, "Locally connected VLSI architectures for the Viterbi algorithm," IEEE J. Select. Areas Commun., vol. 6, pp. 527–537, Apr. 1988

[HAE06] S. Haene et. al., "FPGA Implementation of Viterbi Decoders for MIMO-BICM" IN 39TH Asilomar conference on Signals, Systems and Computers, October 28 – November 1, 2005 Page(s): 734 -738

[HAG89] J. Hagenauer and P. Hoecher, " A Viterbi algorithm with soft-decision outputs and its applications", Proc. Of Globecom '89, Dallas, Texas, pp. 47.11-47.17, Nov. 1989

[HAG96] Hagenauer J et al, "Iterative decoding of binary block and convolutional codes" IEEE Transactions on Information Theory, 1996, vol. 42, 429-445

[HAL01] E. K. Hall and S. G. Wilson, "Stream Oriented Turbo Codes", IEEE Transactions on Information Theory, Vol.47, No. 5, July 2001

[HAO06] Hao Yang et. al., "Design and Implementation of a high-speed and area-efficient viterbi decoder" in 8th International Conference on Solid-State and Integrated Circuit Technology, 2006, pp. 2108-2110

[HAR01] Hartenstein R., "Coarse Grain Reconfigurable Architectures" in Design Automation Conference, 2001. Proceedings of the ASP-DAC 2001. Asia and South Pacific Publication Date: 30 Jan.-2 Feb. 2001 Page(s): 564 – 569

[HAR21] Hartenstein R., " A Decade of Reconfigurable Computing: A Visionary Retrospective", Proc. DATE 2001 Conf., Munich, Germany, pp. 642, Mar., 2001

[HAR98] R. Hartenstein "Using The KressArray for Reconfigurable Computing," Conf. on Configurable: Technology and Applications, Boston, Nov. 1998

[HAR01] J. Harrison, "Implementation of a 3GPP turbo decoder on a programmable DSP core", Commun. Design Conf., San Jose, CA, Oct. 2001

[HAS02] B. Hassibi et al, "High-rate codes that are linear in space and time", IEEE trans. Inform. Theory, pp 1804-1824, 2002

[HAU97] J. Hauser and J. Wawrzynek: Garp: A MIPS Processor with a Reconfigurable Coprocessor; Proc. IEEE FCCM'97, Napa, April 16-18, 1997

[HEL88] Heller, J et al ,"Viterbi Decoding for Satellite and Space Communication" in IEEE Transactions on Communications, Vol 19 Oct 1971 Pages 835-848

[HEN02] R. Henning "Low-Power approach for decoding convolutional codes with adaptive Viterbi algorithm approximations" in International Symposium on Low Power Electronics and Design, 2002. pp. 68-71

[HIL91] D. Hill and N.-S. Woo, "The Benefits of flexibility in look-up table FPGAs," in Proc. Oxford Int. Workshop FPGAs, W. Moore and W. Luk, Eds., Abingdon, Oxfordshire, , UK, 1991, pp. 127–136

[HOC00] D. Hocevar, A. Gatherer, "Architecture selection for a low power flexible Viterbi Decoder" IEEE international conference on 3rd Generation Wireless Communications, 2000, vol. 5, pp. 2257-2264

[HON01] Hong Wei Song et al, "Iterative decoding for partial response(PR), equalized, magneto-optical (MO) data storage channels" JSAC, vol. 19, no. 4, pp. 774-782, Apr. 2001.

[HUNG 90] Hung-Cheng Hsieh et al, "Third-generation architecture boosts speed and density of field-programmable gate arrays" in IEEE 1990 custom integrated circuits conference

[HWA96] D. I. Oh and S. Y. Hwang, "Design of a viterbi decoder with low power using minimum-transition traceback scheme", IEE Electronics Letters, vol. 32, pp. 2198-2199, November 2006

[IMA77] H. Imai et al, "A new multilevel coding method using error correcting codes", IEEE Trans. Inform. Theory, 371-377, May 1977

[INY98] Inyup Kang, et. al., "Low-Power Viterbi Decoder for CDMA Mobile Terminals," IEEE J. Solid-State Circuits, vol. 33, no. 3, pp. 473–482, Mar. 1998

[IRF05] Muhammad Irfan et. al., "Design and Implementation of Viterbi Encoding and Decoding Algorithm on FPGA" in 17th International Conference on Microelectronics, 13-15 Dec 2005, pp 234-239

[ISE95] C. Iseli et al, "Spyder: A SURE (Superscalar and Reconfigurable) processor" J. Supercomput., vol. 9, no. 3, pp. 231-252, 1995.

[JAN97] Jang-Hyun Park et al, "Performance Test of Viterbi Decoder for Wideband CDMA System" in IEEE ASP-DAC Conference, 28-31 Jan. 1997. pp. 19-23

[JUN96] S. J. Jung et al, "A new survivor memory management method in viterbi decoders: trace-delete method and its implementation", Proc. ICASSP, pp.3284-3286, 1996

[KAN98] I. Kang et al, "Low-power viterbi decoder for cdma mobile terminals", IEEE Journal of Solid-State Circuits, vol 33, no 3, pp. 473-482, March 1998

[KAP99] S. Kaptanoglu, G. Bakker, A. Kundu, and I. Corneillet, "A new high density and very low cost reprogrammable FPGA architecture," in IEEE Field Programmable Gate Arrays, Monterey, CA, 1999, pp. 3–12

[KEL93] P. H. Kelly and P. M. Chau, "A flexible constraint length, foldable Viterbi Decoder" IEEE Global Telecommunications Conference GLOBECOM, 1993, pp. 631-635

[KHA07] A. K. Khandani, "Design of the turbo code interleaver using Hungarian method" , Electronics Letters, 1997

[KOU91] J. Kouloheris and A. El Gamal, "FPGA Performance Versus Cell Granularity," in Proc. Custom Integrated Circuits Conf., May 1991, pp.6.2.1 – 6.2.4

[KRE95] R. Kress et al, "A Datapath Synthesis System for the Reconfigurable Datapath Architecture" ASP-DAC 95, Chiba, Japan, Aug 29-Sep 1, 1995

[KUM02] R. Kumar, C.P. Ravikumar, "Leakage Power Estimation for Deep Submicron Circuits in an ASIC Design Environment," Proceedings of the 15th International Conference on VLSI Design (VLSID'02), January 2002

[KWA03] J. Kwak et al, "Reverse tracing of forward state metric in log-MAP and max-log-MAP decoders" Int. Symp. on Circuits and Systems, 25-28 May 2003

[LAM04] L. H.J. Lampe et al, "Multilevel coding for multiple-antenna transmission", IEEE Trans. Wireless Commun., pp 203-208, 2004

[LEU01] O. Y. Leung et al, "Reducing power consumption of turbo decoder using adaptive iteration with variable supply voltage" IEEE Trans. on VLSI Systems, Vol. 9, No. 1, Feb 2001, pp. 34-40

[LIN89] H. D. Lin et al, "Algorithms and architectures for concurrent viterbi decoding", Proc. IEEE International Conference on Communications, pp. 836-840, June 1989

[LIN00] M. B. Lin, "New path history management circuits for viterbi decoders," IEEE Transactions on Communications, vol. 48, no. 10, pp. 1605-1608, October 2000

[LIN04] Lingyan Sun et al., "A High-Throughput, Field Programmable Gate Array Implementation of Soft Output Viterbi Algorithm for Magnetic Recording" in IEEE Transactions on Magnetics Vol 40, Issue 4 July 2004, pp. 3081-3083

[LIU01] Y. Liu et al, "Full rate space-time turbo codes", IEEE J. Select. Areas in Commun., pp 969-980, 2001

[LOD93] Lodge J et al, "Separable MAP filters for the decoding of product and concatenated codes" IEEE ICC '93, Geneva, Switzerland 1740-1745

[LOU95] H. Lou, "Implementing the Viterbi Algorithm", IEEE Signal Processing Magazine, pp. 42-52, Sep. 1995

[LUC06] Lucia Bissi et al., "A Multi-Standard Reconfigurable Viterbi Decoder using Embedded FPGA blocks" In 9th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools, 2006, pp: 146-154

[MAK01] H. Li, W-K. Mak, S. Katkoori, "LUT-Based FPGA Technology Mapping for Power Minimization with Optimal Depth," IEEE Computer Society Workshop on VLSI, pp.123-128, 2001

[MAN03] M. Mansour et al, "VLSI Architectures for SISO-APP Decoders", IEEE trans. On VLSI Systems, Vol. 11, No. 4, August 2003

[MAP92] D. Marple and L. Cooke, " An MPGA compatible FPGA architecture," in FPGA 92, ACM First Int. Workshop on Field-Programmable Gate Arrays, pp. 39-44, Feb. 1992

[MAR99] A. Marshall et al. "A Reconfigurable Arithmetic Array for Multimedia Applications; Proc. ACWSIGiA FPGA'99, Monterey, Feb. 21-23, 1999"

[MAR02] Mark A. Bickerstaff et. al., " A Unified Turbo/Viterbi Channel Decoder for 3GPP mobile wireless in 0.18-μm CMOS ," IEEE J. Solid-State Circuits, vol. 37, no. 11, pp. 1555–1564, Nov. 2002

[MAS99] Guido Masera et al, "VLSI architectures for Turbo Codes" IEEE transactions on very large scale integration (VLSI) Systems, Vol 7, No. 3, September 1999

[MAS02] Masera, G. et al, " Architectural strategies for low-power VLSI turbo decoders", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, June 2002, pages 279 - 285 Volume: 10 , Issue: 3

[MAT01] www.mathworks.com

[MCP99] L. McPheters et al, "Concatenated codes and iterative(turbo) decoding for PRML optical recording channels" in Optical Data Storage Conference, 1999, pp. 342-343.

[MCP02] L. McPheters, "Turbo-coded optical recording for channels with dvd minimum mark size" IEEE Trans. on Magnetics, vol. 38m, no. 1, pp. 298-302, Jan. 2002.

[MIC02] Michael Bedford Taylor et al., "The RAW microprocessor: A computational fabric for software circuits and general-purpose programs", IEEE Micro 22 (2002), no. 2, 25-35

[MOT06] Motorola, "Performance of contention free interleavers for LTE turbo codes", 3GPP TSG RAN WG1 47bis, Tech Rep. R1-070055

[MOWEB] MorphICs website. http://www.morphics.com/

[PAT1] Patent US 870,483: US, 5406,570 ML Decoding-*Inventor*: C. Berrou and P. Adde, April 22

[PAT2] G. D. Forney, "Interleavers", U.S. Patent 3 652 998, Codex Corporation, Watertown, MA

[PAG98] K. Page and P. Chau, "Improved architectures for the add-compare-select operation in long constraint length viterbi decoding", IEEE Journal of Solid-State Cirucits, vol. 33, pp. 151-155, January 1998

[PAR99] K. K. Parhi et al, "VLSI signal processing systems: design and implementation" John Wiley and Sons, 1999

[PAN99] Bupesh Pandita et al., "Design and Implementation of a Viterbi Decoder Using FPGAs" in 12th International Conference on VLSI design, 7-10 Jan 1999, pp. 611-614

[PAU04] Paul M. Heysters, "Coarse-Grained Reconfigurable Processors – Flexibility meets Efficiency" PhD thesis, University of Twente, Enschede, The Netherlands, September 2004

[PAU06] Paul M. Heysters, "Coarse-Grained Reconfigurable Computing for Power Aware Applications" ERSA 2006, Las Vegas Nevada, June 26-29 2006

[PEN03] Pen-Hsin Chen et al. " Dual – Mode Convolutional / SOVA based turbo decoder VLSI design for wireless communication systems" 2003 IEEE International solid state circuits conference ISSCC. Pages 369-372

[PET05] Mihail Petrov et al., "A State-Serial Viterbi Decoder Architecture for Digital Radio on FPGA" in IEEE International Conference on Field-Programmable Technology, 11-14 Dec. 2005, pp 323-324

[PIE96] S. S. Pietrobon, "Efficient Implementation of Continuous MAP Decoders and a Synchronisation Technique for Turbo Decoders", Proc. Int. Symp. Inform. Theory Appl., Victoria , Canada, pp. 586-589, Sep 1996

[PLE89] Plessey Semiconductor ERA60100 preliminary data sheet, Swindon, England, 1989

[PSC96] Lance Perez, Jan Seghers and Daniel Costello, "A Distance Spectrum Interpretation of Turbo Codes", IEEE Trans. Inform. Theory, 42(6):1698-1709, 1996

[PYN94] Pyndiah R et al, "Near optimum decoding of product codes" IEEE GLOBECOM'94, New Orleans, USA, pp 339:343

[PYN95] Pyndiah R et al, "Performance of block turbo coded 16-QAM and 64-QAM modulations" GLOBECOM' 95, Singapore, 1039-1043

[PYN97] Pyndiah R, "Iterative decoding of product codes" Proceedings International Symposium on Turbo Codes and Related Topics, Best, France, 1997, pp 71-79

[QIN04] Qin Xiang-Ju et. al., "An Adaptive Viterbi Decoder Based on FPGA Dynamic Reconfiguration Technology" in IEEE International conference on Field-Programmable Technology, 2004, on pages 315-318

[RAB01] T. Tuan, J. Rabaey, "Reconfigurable Fabric for Low-Energy Protocol Processing," ICASSP 2001

[RAB95] A. Yeung and J. Rabaey," A 210mb/s radix-4 bit-level pipelined viterbi decoder", Digest of Technical Papers, Proc. International Solid-State Circuits Conference, February 1995

[RAD81] C.M. Rader, " Memory management in a Viterbi decoder," IEEE Trans. Commun., vol. COM-29, pp. 1399-1401, Sept. 1981

[RAM70] J. L. Ramsey, "Realization of optimum interleavers", IEEE Transactions on Information Theory, Vol. 16, No. 3, pp.338-345, May 1970

[REV04] J.S.Reeve et al., "A FPGA Implementation of a Parallel Viterbi Decoder for Block Cyclic and Convolution Codes" in IEEE conference on Communications, 20-24 June 2004. pp. 2596-2599

[ROB89] P. Robertson, E. Villebrun, and P. Hoher, " A Comparison of Optimal and Sub-Optimal MAP Decoding Algorithms Operating in the Log Domain," in proceedings of the International Conference on Communications, Seatle USA, pp. 1680-1686, 1989

[ROS04] A. La Rosa, et al, " Implementation of a UMTS turbo-decoder on a dynamcially reconfigurable platform", Design, Automation and Test in Europe, Volume 2, 16-20 Feb. 2004 pp. 1218-1223 Vol 2

[ROS89] J. Rose, R. J. Francis, P. Chow, and D. Lewis, "The effect of Logic Block complexity on area of programmable arrays," in Proc. Custom Integrated Circuits Conf., May 1989, pp. 5.3.1–5.3.5

[ROS90] J. Rose, R. J. Francis, D. Lewis, and P. Chow, "Architecture of field-programmable gate arrays: The effect of logic functionality on area efficiency," IEEE J. Solid-State Circuits, pp. 1217–1225, Oct. 1990

[ROY99] K. Roy, "Power-Dissipation Driven FPGA Place and Route Under Timing Constraints," IEEE Transactions on Circuits and Systems, vol. 46, no. 5, pp. 634-637, May 1999

[SAT92] Satwant Singh, et al, "The Effect of LB Architecture on FPGA Performance" in IEEE Journal of Solid State Circuits, vol 27, No 3, March 1992

[SCH99] C.Schurgers et al, "Energy efficient data transfer and storage organization for a MAP turbo decoder module" Proc. Of Low Power Electronics and Design Symposium, 16-17 Aug. 1999, pp. 76-81

[SCH199] C. Schurgers et al, "Adaptive Turbo Decoding for Indoor Wireless Communication" IEEE Wireless Communications and Networking Conference (WCNC), 21-24 September 1999, pp. 1498-1502

[SCH01] C. Schurgers et al, "Memory Optimization of MAP Turbo Decoder Algorithms", IEEE Trans. on VLSI Systems, Vol. 9, No. 2, April 2001, pp. 305-312

[SHA48] C. E. Shannon, "A mathematical theory of Communication, Bell System Technical Journal, Vol. 27, July 1948, pp. 379-423 and October 1948, pp. 623-656

[SHA03] S. Sharm et al, "A simplified and efficient implementation of FPGA-based turbo decoder" Proceedings of the 2003 IEEE Intl. Conf. on Perf. ,Computing and Communications", 9-11 April 2003 pp. 207-213

[SHA04] A. R. Abdul Shakoor et al., "High Speed Viterbi Decoder for W-LAN and Broadband Applications" in IEEE Northeast workshop on circuits and systems, 20-23 June 2004, pp. 25-28

[SHU93] C. B. Shung et. al., " Area-efficient architectures for the Viterbi algorithm – Part I: Theory," IEEE Trans. Commun., vol. 41, pp 2907-2917, Sep 1993

[SIN02] A. Singh, M. Marek-Sadowska, "Efficient Circuit Clustering for Area and Power Reduction in FPGAs," Proceedings of International Symposium on Field-Programmable Gate Arrays, February 2002

[SMI04] Smit, G. J. M. et al, "Lessons learned from designing the MONTIUM – a reconfigurable processing tile" in IEEE SOC proceedings, 16-18 Nov. 2004, pp. 29-32

[SON00] H. Song et al, "Turbo decoding for optical storage" in ICC 2000, vol. 1, New Orleans, LA, 2000, pp. 104-108

[STE01] J. Steensma, "FPGA implementation of a 3GPP turbo codec" Thirty-Fifth Asilomar Conference on Signals, Systems and Computers, Vol: 1, 4-7 Nov 2001, pp. 61-65 vol. 1

[TAK06] Oscar Y. Takeshita, "On Maximum Contention-Free Interleavers and Permutation Polynomials Over Integer Rings" IEEE Trans. On Information theory, Mar 2006, pp 1249-1253

[TAR99] V. Tarokh et al, "Space-time block codes from orthogonal designs" IEEE Trans. Inform. Theory, pp 1456-1467 July 1999

[TEN04] S. Ten Brink et al, "Design of low-density parity-check codes for modulation and detection" IEEE Trans. Commun., pp 670-678, Apr 2004

[THI95] Punya Thitimajshima, " Recursive Systematic Convolutional codes and application to parallel concatenation," IEEE globecom '95, Singapore, pp. 2267-2272

[TMS04] TMS320C64x DSP Turbo-Decoder Coprocessor (TCP) Reference Guide, Sep 2004 www.ti.com

[TMS05] TMS320C6414, databook, revised May 2005, www.ti.com

[TMS104] TMS320C6414T/15T/16T Power Consumption Summary, application report spraa45-august 2004, www.ti.com

[TSU99] C. Y. Tsui et al, "Low power acs unit design for the viterbi decoder", Proc. IEEE International Symposium on Circuits and Systems, pp. 137-140, 1999

[TOD05] Todman et al, "Reconfigurable computing: architectures and design methods", IEE Proceedings in Computer and Digital Techniques, Vol 152 on pages 193-207, Mar 2005

[TUJ00] D. Tujkovic, "Recursive space-time trellis codes for turbo coded modulation" in Proc. IEEE Global Telecommun. Conf., vol 2, pp 1010-1015 Nov 27 – Decl 2000

[TUJ03] D. Tujkovic, "Space-time turbo coded modulation for wireless communication systems" Ph.D. dissertation, University of Oulu, Oulu, Finland 2003.

[TUN06] Tung. H.T et al., "Implementation and Comparison of Various Viterbi Detectors for EPRML Systems" in international Conference on Communications, Circuits and Systems, 25-28 June 2006, pp 2309-2313

[UWB05] ECMA UWB Specification "High Rate Ultra Wideband PHY and MAC standard" 1st ed. Vol. ECMA-368, ECMA, ED., 2005

[VAL01] M. C. Valenti and J. Sun, "The UMTS turbo code and efficient decoder implementation suitable for software-defined radios", Int. J. Wireless Inform. Networks, vol. 8, no. 4, pp. 203-216, 2001

[VIG00] F. Vigilone et al, "A 50 Mbit/s Iterative Turbo-Decoder", Proc of DATE 2000 Conference, pp. 176-180, March 2000

[VIL98] J. Villasenor, " The flexibility of configurable computing", IEEE Signal Processing Magazine, Vol. 15, No. 5, Sep. 1998, pp. 67-84

[VIT67] A. J. Viterbi, "Error Bounds for Convolution Codes and an Asymptotically Optimum Decoding Algorithm", IEEE transactions on Information Theory, IT-13: 260-269, April 1967

[VIT79] A. J. Viterbi and J. K. Omura, "Principle of Digital Communication and Coding". McGraw-Hill Book Company, 1979

[VIT98] Viterbi, A.J. "An intuitive justification and a simplified implementation of the MAP decoder for convolutional codes", IEEE Journal on Selected Areas in Communications, Volume: 16 , Issue: 2 , Feb. 1998 Pages:260 – 264

[VOG06] Vogt T, Wehn N, "A Reconfigurable Application Specific Instruction Set Processor for Viterbi and Log-MAP Decoding" IEEE SIPS 2006, Oct 2006, pp 142 – 147.

[WAN93] Wang Kaiming et al., "Viterbi Hardware Implementation for GSM" in IEEE conference on Computer, Communication, Control and Power Engineering, 19-21 Oct. 1993, pp. 120-122

[WAN98] J-M. Hwang, F-Y. Chiang, T-T. Hwang, "A Re-Engineering Approach to Low Power FPGA Design Using SPFD," Proceedings of Design Automation Conference, pp. 722-725, 1998

[WAN02] X. Wang et al , "LDPC-based space-time coded OFDM systems over correlated fading channels" IEEE Trans. Commun., 74-88, Jan 2002.

[WEB01] www.latticesim.com/products

[WEB02] www.xilinx.com/products/logicore/alliance

[WEB03] www.altera.com/products/ip

[WEB04] www.us.design-reuse.com/sip

[WEB05] www.icoding.com/products.htm

[WEB06] www.turboconcept.com

[WEB07] www.icoding.com

[WEB08] www.eccincorp.com

[WEB09] www.iterativeconnections.com

[WEB10] www.datumsystems.com

[WEB11] www.trellisware.com

[WEB12] www.sworld.com.au

[WEB13] www.icoding.com

[WEB14] www.st.com

[WEB15] www.directv.com

[WEB16] www.dishnetwork.com

[WEB17] www.voom.com

[WEB18] www.broadcom.com

[WEB19] www.aha.com

[WEB20] www.radynecomstream.com

[WEB21] www.pardisedata.com

[WEB22] www.advantech.ca

[WEB23] www.idirect.net

[WEB24] www.viasat.com

[WEB25] www.recoresystems.com

[WEB26] www.chameleon.ctit.utwente.nl

[WEI97] E. Waingold, M. Taylor, et al., "Baring it all to software: raw machines, IEEE Computer (1997), 86-93

[WEL04] Welling A. "Two Stage Interleaving Network Analysis to design area and energy efficient 3GPP compliant receiver architectures" IEEE Workshop on Signal Processing Systems, 2004.Page(s):65 – 70

[WIC95] S. B. Wicker, "Error Control Systems for Digital Communication and Storage", Prentice Hall, 1995

[WOL03] F. G. Wolff, M.J. Knieser, D.J. Weyer, C.A. Papachristou, "High-Level Low Power FPGA Design Methodology," National Aerospace and Electronics Conference (NAECON), pp.554-559

[WOR99] A. P.Worthen et al, "Performance optimization of VLSI transceiver for low-energy communications systems" Military Communication Conference, MILCOM 1999, Vol. 2, 31 Oct. – 3 Nov. 1999, pp. 1434-1438

[WOR00] A. Worm et al, "A High Speed MAP architecture with optimized memory size and power consumption" Proc. IEEE Workshop Signal processing Systems, SiPS 2000, pp. 265-274

[WOR02] h. Michel, A. Worm et al, "Hardware/Software trade-offs for advanced 3G channel coding", in Proc. Design, Automation, Test Eur, Conf., Mar 2002, pp. 396-401

[XIA02] Xiao-Jun Zeng et al, "Design and implementation of a turbo decoder for 3G W-CDMA systems" Consumer Electronics, IEEE Transactions, Vol 48, Issue: 2, May 2002 pp. 284-291

[YEH96] David Yeh et al., "RACER: A Reconfigurable Constraint-Length 14 Viterbi Decoder" in IEEE Symposium on FPGAs for Custom Computing Machines, 17-19 April 1996, pp: 60:69

[YEU93] A. K. W. Yeung, J. M. Rabaey, " A Reconfigurable Data-driven Multiprocessor Architecture for Rapid Prototyping of High Throughput DSP Algorithms" Proc. HICSS-26, Kauai, Hawaii, Jan. 1993

[YUF00] Yufei Wu et al, "Forward computation of backward path metrics for MAP decoder," IEEE VTC2000

[ZEH92] E. Zehavi, "8-PSK trellis codes for a Rayleigh channel" IEEE Trans. Commun., pp 873-884, May 1992

[ZHA99] H. Zhang, M. Wan, V. George, and J. Rabaey, "Interconnect Architecture Exploration for Low-Energy Reconfigurable Single-Chip DSPs," Proceedings of the IEEE Computer Society Workshop on VLSI '99, pp. 2-8, 1999

[ZHA01] H. Zhang, L. Wang and J. Yu, "A chaotic interleaver used in turbo codes", submitted to IEEE Transactions on Circuit & Systems, March 2001

[ZHO99] Zhongfeng Wang et. al., "VLSI Implementation issues of Turbo Decoder Design for Wireless Applications", IEEE workshop on Signal Processing Systems, 1999, pp. 503-512

[ZHO02] Zhongfeng Wang; Zhipei Chi; Parhi, K.K, "Area-efficient high-speed decoding schemes for turbo decoders, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Dec. 2002 Pages:902 – 912

[ZHU03] Zhu Y et al., "Reconfigurable Viterbi Decoding Using a New ACS Pipelining Technique" IEEE International Conference on Application-Specific Systems, Architectures, and Processors, 24-26 June 2003, pp 360-368

[ZHU07] Zhuo Xu et al., "Implementation of Folded Sliding Block Viterbi Decoders for MB-OFDM UWB Communication System" in IEEE International Symposium on Circuits and Systems, 27-30 May 2007, pp 2574-2577