2004

New VLSI design of a MAP/BCJR decoder.

Leila Sabeti
University of Windsor

New VLSI Design of a MAP/BCJR Decoder

by

Leila Sabeti

A Thesis
Submitted to the Faculty of Graduate Studies and Research through the Department of Electrical and Computer Engineering in Partial Fulfillment of the Requirements for the Degree of Master of Applied Science at The University of Windsor

Windsor, Ontario, Canada
2004

Your file Votre référence
ISBN: 0-494-00161-5
Our file Notre référence
ISBN: 0-494-00161-5

New VLSI Design of a MAP/BCJR Decoder

by

Leila Sabeti

APPROVED BY:

R. Lashkari
IMSE Department

M.A. Sid-Ahmed
ECE Department

M. Ahmadi, Advisor
ECE Department

K.F. Tzepe , Co-advisor
ECE Department

B. Shahrava, Chair of Defense
ECE Department

September 29, 2004

Abstract

Any communication channel suffers from different kinds of noises. By employing forward error correction (FEC) techniques, the reliability of the communication channel can be increased. One of the emerging FEC methods is turbo coding (iterative coding), which employs soft input soft output (SISO) decoding algorithms like maximum a posteriori (MAP) algorithm in its constituent decoders. Owing to their outstanding performances, turbo codes have already been adopted by recent communication systems such as Consultative Committee for Space Data systems (CCSDS), telemetry channel coding and the 3rd Generation Partnership Project. A major difficulty of applying turbo codes in many applications is the decoding complexity of SISO decoders.

Efficient implementations of these decoders can significantly increase the employment of turbo codes in different communication applications. Because Max-Log-MAP algorithm is the best compromise among other algorithms in terms of performance and implementation complexity, our implementation is based on this algorithm.

In this thesis we introduce a design with lower complexity and less than 0.1dB performance loss compare to the best performance observed in Max-Log-MAP algorithm. A parallel and pipeline design of a MAP decoder suitable for ASIC (Application Specific Integrated Circuits) is used to increase the throughput of the chip. The branch metric calculation unit is studied in detail and a new design with lower complexity is proposed. The design is also flexible to communication block sizes, which makes it ideal for variable frame length communication systems. A new even-spaced quantization technique for the proposed MAP decoder is utilized. Normalization techniques are studied and a suitable technique for the Max-Log-MAP decoder is explained. The decoder chip is synthesized and implemented in a 0.18 μ m six-layer metal CMOS technology.

Acknowledgments

I would like to extend my sincere gratitude and appreciation to people who have contributed to the completion of this thesis.

First and foremost, I would like to thank my supervisor, Dr. Majid Ahmadi, for whom I have the utmost respect and admiration. He had a tremendous impact on me academically, socially and personally, and for this I am forever in his debt.

I am also grateful to Dr. Kemal Tepe for his knowledge suggestions and his enthusiasm towards my research. I would also like to thank my committee members, Dr. M. A. Sid-Ahmed and Dr. R. Lashkari for their patience and support.

Additionally, I would like to thank my husband, Shahram Talakoub for his patience, his confidence in my ability and his guidance and assistance toward my research. Finally, I would like to acknowledge my parents for their constant support, encouragement and motivation.

Table of Contents

List of Figures

Chapter 1

Introduction

The purpose of a communication system is to transport information from a source to one or more user destinations via a communication channel. There are two types of communication systems, analog communication systems and digital communication systems. In analogue systems the information varies continuously in both amplitude and time, and is used to modify some characteristic of a sinusoidal carrier wave (e.g. amplitude, phase, frequency), however in digital systems the information is processed so that it can be represented by a sequence of discrete messages.

1.1 Digital Communication Systems

A digital communication system conveys information in digital form from a source to one or more destinations through a communication channel. Figure 1.1 gives the block diagram of a typical digital communication system.

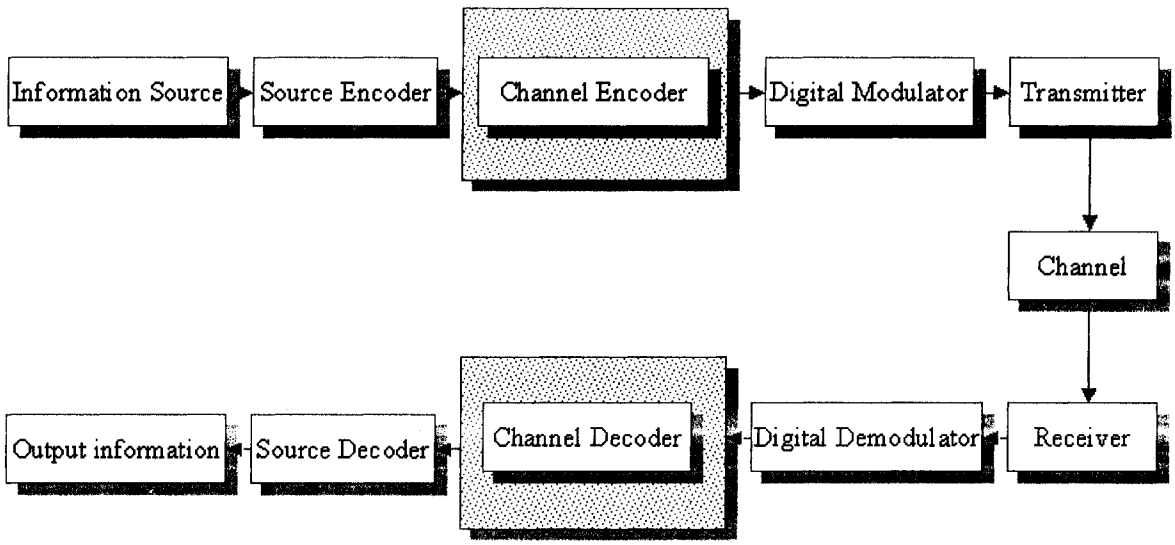


Figure 1.1 A typical digital communication system

1.1.2 Information Source

The input is source signal. It might be a sequence of symbols such as letters from the English or Chinese alphabet, binary symbols from a computer file, etc. Alternatively, the input might be a waveform, such as a voice signal from a microphone, the output of a sensor, a video waveform, or, it might be a sequence of images such as X-rays or photographs.

Whatever the source signal is, we will model it as a sample function of a random process. This is one of the reasons why probability is an essential prerequisite for communication theory. It is not obvious why inputs to communication systems should be modeled as random, and in fact this was not appreciated before Shannon developed information theory in 1948. The study of communication before that time (and well after that time) was based on Fourier analysis, which basically studies the effect of passing sine waves through various kinds of systems and components. This kind of analysis often called Nyquist theory in the context of digital communication.

However, Shannon's view was that if the recipient knows that a sine wave of a given frequency is to be communicated, why not simply regenerate it at the output rather than send it over a long distance? Or, if the recipient knows that a sine wave of unknown frequency is to be communicated, why not simply send the frequency rather than the entire waveform?

The essence of Shannon's viewpoint is that we should focus on the set of possible inputs from the source rather than any particular input. The objective then is to transform each possible input into a transmitted signal in such a way that each possible transmitted signal can be distinguished from the others at the output. A probability measure is needed on this set of possible inputs to distinguish typical inputs from abnormal inputs. It will be explained later how this point of view drives the processing of the inputs as they pass through a communication system.

1.1.3 Source Encoder

The source encoder has the function of converting the input from its original form into a sequence of bits. The simplest source coding techniques involve representing the source signal by a sequence of symbols from some finite alphabet, and then coding the alphabet symbols into fixed-length blocks of bits. For example, letters from the 27-symbol English alphabet (including a space symbol) may be encoded into 5-bit blocks. Or, upper-case letters, lower-case letters, and a great many other special symbols may be converted into 8-bit blocks (bytes) using the 7-bit standard ASCII code.

The most straightforward approach to converting an analog waveform to a bit sequence, called analog to digital (A/D) conversion, is first sampling the source at a sufficiently high rate (called the "Nyquist rate"), and then quantizing it properly for adequate reproduction. For example, in standard voice telephony, the voice waveform is filtered to a bandwidth of less than 4 KHz and sampled 8000 times per second; each sample is then quantized into one of 256 levels and represented by an 8-bit byte. This yields a source coding bit rate of 64 kb/s.

Beyond the basic objective of conversion to bits, the source encoder often has the further objective of transmitting as few bits as possible, subject to the need to reconstruct the input adequately at the output. In this case source encoding is often called data compression. For example, modern speech coders can encode telephone-quality speech at bit rates of the order of 6-16 kb/s rather than 64 kb/s.

1.1.4 Channel

Here, the channel in a generic digital communication system is discussed before considering channel coding.

In general, the channel is that part of the communication medium that is given and not under the control of the designer. Thus, to a source code designer, the channel might be a digital channel with bits as input and output; to a telephone-line modem designer, it might be a 4 KHz voice channel; to a cable modem designer, it also might be a physical coaxial cable of up to a certain length, with certain bandwidth restrictions.

For a channel code designer, the channel is often a physical channel; e.g., a pair of wires, a coaxial cable, or an optical fiber going from the source location to the destination. It also might be the open space between source and destination over which, electromagnetic radiation can carry signals, underwater acoustic channel or storage channel.

If a channel was simply a linear time-invariant system (e.g., a filter), then it could be completely characterized by its impulse response or frequency response. However, the channels that we look at here (and channels in practice) always have an extra ingredient noise. Suppose that there were no noise and a single input voltage level could be communicated exactly. Then, representing that voltage level by its infinite binary expansion, we would in principle be able to transmit an infinite number of binary digits by transmitting a single real number. This is absurd in practice, of course, precisely because noise limits the number of bits that can be reliably distinguished. Again, it was

Shannon in 1948 who, realized that noise provides the fundamental limitation to performance in communication systems.

1.1.5 Noise

The major characteristic of a communication channel in is how the channel distorts the information. We start by listing out some common channel defects:

1. Thermal noise in electronic devices
2. Signal attenuation
3. Amplitude and phase distortion
4. Multipath distortion
5. Finite-bandwidth (low-pass filter) distortion
6. Impulsive noise

Based on a knowledge of these channel defects, we construct the generic channel model. The most common channel model involves a waveform input X(t), an added noise waveform Z(t), and a waveform output Y(t) that is the sum of the input and the noise, Y(t) = X(t) + Z(t), as shown in Figure 1.2.

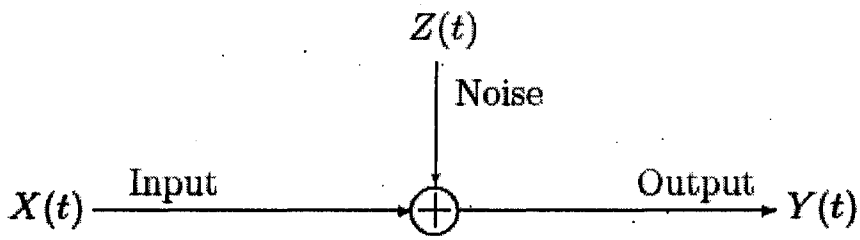


Figure 1.2 A channel model

Each of these waveforms is viewed as a stochastic process. For any channel with input X(t) and output Y(t), we could define the noise to be Z(t) = Y (t) - X(t), which is essential to be statistically independent of the input. The noise Z(t) is often modeled as white

Gaussian noise (in this case it is called Additive White Gaussian Noise), which is the most common and most studied model among the various models.

1.1.6 Channel Coding

The objective of channel coding is to map the signal to a form where reliable communication can be achieved over a noisy channel. This may be achieved by introducing controlled redundancy into the signal. The channel decoder attempts to reconstruct the original encoder input as accurately as possible. The choice of channel coding depends considerably on the channel over which the data will be transmitted. Block codes, convolutional codes, and more recently block turbo codes are examples of channel encoding techniques.

1.1.7 Modulation

Modulation is performed to provide for efficient transmission of the signal over the channel. It operates by keying shifts in the amplitude, frequency, or phase of a sinusoidal carrier wave to the channel encoder output. The detector performs demodulation, attempting to produce a signal that follows the time variations in the channel encoder output. Modulation is required for transmitting over a band-pass channel.

1.1.8 Source Decoder

The source decoder performs the inverse mapping of the source encoder. Usually, it can be realized with low complexity.

1.1.9 Output Information

Output information is the sink of the transmitted information and should be similar to the original information of the source before they are sent trough the channel.

1.2 History of Coding

Unlike analog communication, digital communication has the ability to detect and correct errors produced by the noise of the channel. Forward error correction plays an important role in the system design process, which attempts to balance the tradeoffs of power, bandwidth, and data reliability.

Digital communication is a field in which theoretical ideas have had an unusually powerful impact on actual system design. The basis of the theory was developed in 1948 by Claude Shannon, and is called Information Theory. For the first 25 years or so of its existence, information theory served as a rich source of academic research problems and as a tantalizing suggestion that communication systems could be made more efficient and more reliable by using these approaches. By the mid 1970's, mainstream systems using information theoretic ideas began to be widely implemented for two reasons. First, by that time there were a sizable number of engineers who understood both information theory and communication system development. Second, implementation of the sophisticated algorithms was possible because of the low cost and increasing processing ability of digital hardware.

Shannon's major accomplishments include the development of the noiseless source coding theorem, the rate distortion theorem, and the channel coding theorem [gallager]. In 1948, Shannon published a groundbreaking paper and showed that reliable communication trough a noisy channel is possible. What Shannon showed was fact that more sophisticated coding schemes can achieve arbitrarily low error probabilities without lowering the data rate below a certain data rate that depends on the channel being used, called the channel capacity.

Channel capacity can be calculated from the following equation:

$$C = W \log_2 (1+S/N)$$

where W is the bandwidth in Hz, S is the signal power in watts and N is the total noise power.

Now, after explanation about the digital communication systems there is an overview of different number systems, which is part of any digital system computation.

1.3 Number Systems

In this design, we are working with fixed-point values that can be either positive or negative and one of the underlying considerations that must be made when carrying out digital arithmetic is the number representation. Therefore, in this section different number systems are discussed with their advantages and shortfalls.

1.3.1 Signed Fixed Point Numbers

The natural number system (also known as unsigned integer), although simple to put into practice, limits the potential of the overall design. To realize sophisticated arithmetic computation, the number system of choice must be capable of signed representation. There are several means by which signed representation may be achieved.

The earliest form of signed numbers representation is known as signed magnitude (or sign-and-magnitude), where a sign bit is included as part of the value. Similar to the manual representation of negative numbers, a number will have a numeric value, and a sign bit in front identifying negation. Thus a k-bit system will have a (k-1)-bit magnitude description. Although conceptually signed values will require supplementary circuitry (such as magnitude comparator, or subtractor) for proper addition.
An encoding scheme may also be used to eliminate negative values during computation. Biased representations, for example, will convert everything to positive numbers by adding a fixed bias value. Also referred to as excess-biased encoding, this type of representation is the difficulty in multiplication and division, and the additional computation that is required to unbiased values

Complement formats are the third major representation of signed numbers. A large complementation constant is added to all negative values, satisfying the condition that there is no overlap in the representation of the positive values. In binary systems, 2's complement representation obtained by taking the ones complement (bit-wise negation) and adding 1, is used to describe negative values. The ease of negation and computation using complement formats leads to their attractiveness as a signed digit representation.

1.3.2 Redundant Number Systems

A number system with radix-R maybe fully described using *R* distinct digits. For example the binary system (radix-2), can define any value using the 2-digit set [0, 1]. In general a positional radix-*R* number system representations a k-digit value as a string of digits:

$$(d_{k-1}, d_{k-2}, \ldots d_0)$$

$$\sum_{i=0}^{k-1} d_i R^i$$

A system is referred to as redundant if more than *R* digits are used to define a radix-R representation. Redundant number systems are primarily used in digital systems for arithmetic speed-up techniques. By over defining a system using redundant values, the cost of computation of certain operations may be appreciably reduced. Addition is one such application, where use of redundant representation allows for constant time addition, since the value of the carry bit may be obtained by examining a fixed number of previous bits [pillai].

A simple case of redundant number systems that has already been represented is the carry save representation of a number. By describing a K-bit value using two k-bit numbers in the carry save format, [0, 1, 2], the necessary for carry propagation is alleviated. The description of a value using what are essentially twice as many bits may be clearly justified when considering the array multiplier. By maintaining the partial product

summation in carry save form, the latency of each stage is reduced to only one full adder delay, since carry propagation is postponed until the final stage.

Other redundant number systems have also proven to be favorable alternative to conventional systems. The signed digit number system, first classified in 1961, has demonstrated 33% savings in adders required for multiplication over standard binary notation [parhami].

The disadvantage of using redundant number representation is need for re-conversion back into conventional notation.

1.3.3 Residue Number Systems

In a residue number system representation (RNS), a umber x, is represented by the set of its residues with respect to modulo m:

$$x_i = \mathrm{x} \bmod m_i = \langle \mathrm{x} \rangle_{mi}$$

Since a value is uniquely represented using smaller residues, the mathematical operations that are carried out will inevitably be fast and simple. Addition, multiplication and subtraction are the primary vantage points in residue number systems. This is due to the fact that these functions may be carried out by directly performing the given operation on the smaller residues. Frenking and Parhi [frenking] present an application of RNS arithmetic in public-key cryptography scheme, which inherent use of modular exponentiation and multiplication.

One disadvantage of such number systems is reduction of representation efficiency over binary notation. A k-bit representation, yielding 2^k unique values, may only produce half as many in RNS format. Also, any gains in performance achieved by implementing addition, subtraction and multiplication may be eclipsed by the severe complexity of other mathematical and logical operations.

1.3.4 Logarithmic Number Systems

In pure mathematics, the multiplication and division of logarithms are easily performed by addition and subtraction respectively. Since the hardware implementation of addition/subtraction circuits is substantially more straightforward than that of multipliers and dividers, logarithmic number systems (LNS) may be employed to carry out these operations. As in other non-conventional representations, LNS in only directed towards the enhancement of certain operations, and presents restrictions on most other standard numeric tasks, such as addition and subtraction.

Recently the Double Base Number System (DBNS) has been proposed at yet another class of redundant number representation [dimitrov]. Simple arithmetic operations are made possible by simple geometric interpretation of the orthogonal bases. DBNS provides logarithmic like computation with reduced look-up table dimensions. This representation provides yet another alternative for application specific computation enhancement.

1.3.5 Floating Point Number Systems

In order to achieve the levels of precision demanded by modern systems, it becomes imperative to have a number representation capable of describing real numbers. The limited range and/or precision of fixed-point values alleviated through the use of the floating-point number system. Unlike fixed-point representations where the location of the decimal point is predefined, floating point values allow extremely large or small numbers to be represented with the same high degree of precision by defining a value using a dynamic range.

As defined in IEEE standard for binary floating-point arithmetic [IEEE], a floating-point value is defined as:

$$x = \pm f \times b^e$$

where x is the floating point value f is the fraction of mantissa, b is the base (fixed at 2 for precision) and e in the exponent.

Floating-point numbers have two distinct representations according to the standard, depending on word size. Figure 1.3 outlines the difference in the structure of the words of the 32-bit single precision and the 64-bit double precision formats. The sign (s), exponent (e) and fraction or mantissa (f), form the 32 and 64 bit precision formats. The mantissa is normalized to lie within the set [1, 2), such that the MSB is a 1. In this manner the leading 1 is removed and understood; this is referred to as the "hidden one", saving one bit in representation. The signed integer exponent is biased accordingly, such that the value will always be a positive number; the exponent biased for 127 and for single and 1023 for double precision formats.

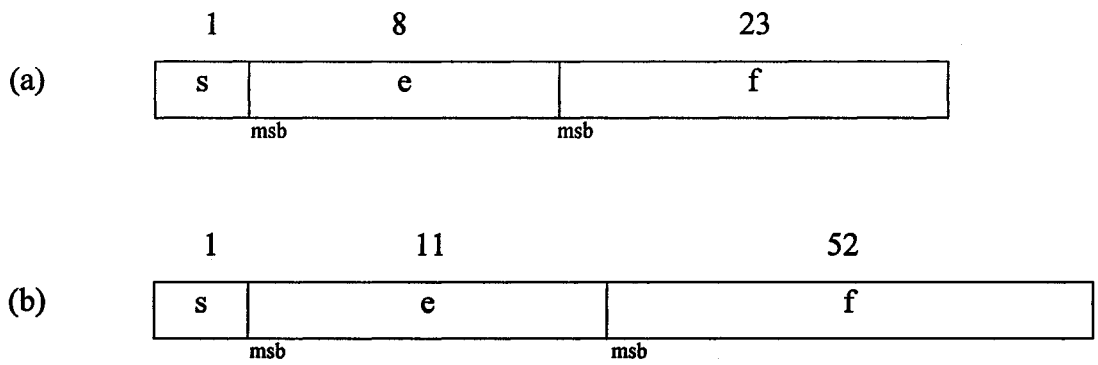


Figure 1.3 IEEE floating-point standard word widths for
(a) single precision (b) double precision

1.4 Thesis Overview

1.4.1 Thesis Highlights

This thesis will present the design and implementation of a new Maximum a Posteriori (MAP) Decoder, which is a Soft-Input Soft-Output (SISO) decoder. The Max-Log-MAP algorithm is found to be the best compromise between performance and

complexity for implementation of this decoder. The architecture is parallel and pipeline and therefore, makes the design work at a high throughput rate. A new quantization is introduced that best fits the performance requirements. Also a new branch metric calculation unit is proposed, which is less complex and smaller in terms of area size.

The RTL code of the design is written in Verilog and the decoder is synthesized and implemented in CMOS 0.18 μ m technology.

The system specifications are as follows:

- Encoder: Recursive Systematic Convolutional (RSC)
- Channel: Additive White Gaussian Noise (AWGN)
- Considered Modulation: Binary Phase Shift Keying (BPSK)
- Block size: 10-1024

1.4.2 Thesis Overview

This thesis has begun with a general overview of the digital communication systems in this chapter. Turbo coding and its algorithms are discussed in chapter 2. System level design including the decoder architecture and the MATLAB simulation results of quantization and normalization techniques are explained in chapter 3. Chapter 4 will focus on RTL simulation and synthesis level of the design, and VLSI implementation steps are presented in chapter 5. The thesis will end with the results, comparisons and conclusions in chapter 6.

Chapter 2

Turbo coding

Although Shannon proved the theoretical limit at which error-free communications could take place using error-correcting codes, all previous coding schemes have fallen far short of this limit. In 1993 a group of French researchers devised a new class of error-correcting codes, which achieved near-Shannon limit performance. Turbo codes were developed in 1993 by C. Berrou, A. Glavieux and P. Thitimajshima [berrou] at the "Ecole Nationale Superieure des Telecommunications de Bretagne" in Brest, France, as an exercise in VLSI design and claimed to achieve performance near Shannon limit in the AWGN channels. The modified BAHL et al. algorithm [bahl] was used for decoding. The original turbo code's performance came within 0.7 dB of Shannon's theoretical limit after 18 decoding iterations. This chapter presents the development of turbo codes and discusses the theoretical background necessary to understand their application.

2.1 Turbo Encoder

The focus of coding theory since Shannon's initial work has been to find a constructive way to place 2^k codewords in an n-dimensional space without overlapping the decoding spheres. The code rate r is defined as the ratio of k, the number of information symbols transmitted per codeword, to n, the total number of symbols

with 7 symbols and has a rate equal to 4/7. This code, the first error correcting code, was able to correct a single error in a block of seven encoded bits. Other attempts to solve the problem presented by coding theory have included block codes (such as Golay, BCH, and Reed-Solomon codes) and convolutional codes, but prior to the early 1990's, no practical techniques achieved the full promise of Shannon's predictions. Turbo coding is a unique approach to the old coding problem. Turbo codes are able to integrate structured codes in a pseudo-random manner, which approximates Shannon's capacity limit; this constitutes a significant increase in power efficiency compared to previous block and convolutional coding schemes. The original turbo code employed two recursive systematic convolutional (RSC) encoders concatenated in parallel and separated by a pseudo-random interleaver.

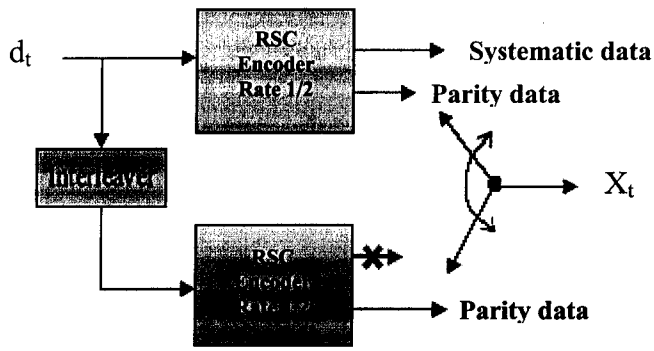


Figure 2.1 Turbo encoder

Each rate 1/2 RSC encoder produces a set of systematic and parity bits. The systematic bits are identical to the input bits; the parity bits are determined from the input bits, the state of the encoder, and the generator matrix. Because transmitting two sets of systematic bits is redundant, the interleaved systematic bits from the second RSC encoder are punctured, or removed, before transmission. The overall rate of the turbo code can be increased from 1/3 to 1/2 by alternately puncturing the parity bits from each of the constituent encoders. As the code rate increases, bandwidth efficiency improves; however, performance is degraded since the decoder has less information to use in making a decision.

The resulting code has a complex structure and appears quite random. This characteristic of the code results in good performance, particularly at low signal-to-noise ratios (SNRs). The overall code, however, is broken down into its constituent parts at decoder.

2.2 RSC Encoder

As previously mentioned, the original turbo code was a parallel-concatenated convolutional code. Concatenated codes can be classified as either parallel-concatenated convolutional codes (PCCCs), in which two encoders operate on the same information bits, or serial concatenated convolutional codes (SCCCs), in which one encoder encodes the output of another encoder. The term "turbo code" is often associated with PCCCs and will be used to refer to PCCCs throughout the rest of this thesis. PCCCs employ two or more recursive systematic convolutional (RSC) encoders joined in parallel by one or more pseudo-random interleavers. Although the encoders need not be identical, they often are, in practice. An example of a RSC encoder is shown in Figure 2.2.

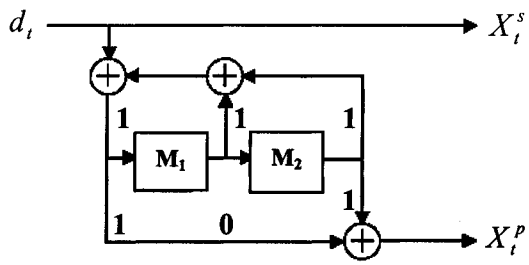


Figure 2.2 RSC Encoder

The data bits d_t are fed into the encoder, which generates a set of systematic and parity bits. There are two memories in this encoder, which are shift registers. The size of the memory defines how many symbols in the input sequence will affect the output sequence at a time. The rate of the convolutional code is the ratio between the number of inputs and the corresponding number of outputs at a time. For example if the rate is R_C=1/2, then each bit of the binary input sequence (d_t) is mapped into two bits in binary output sequence (X_t^s and X_t^p) as shown in Figure 2.2. The modulation used is binary phase shift keying (BPSK), which map 1 to 1 and 0 to –1. The generator sequence defines how the

output will be obtained by the inputs in the memories and the most recent one. The generator is (7, 5) or (111, 101), which is shown in Figure 2.2. Each connection in the Figure 2.2 is shown by 1 and otherwise by 0.

Another parameter is the constraint length of the code, K, which is equal to the number of memories plus one. Figure 2.2 shows a RSC encoder with code rate ½, generators (7, 5) and constraint length 3. Table 1.1 shows how encoder outputs are obtained by the inputs in the memories.

Table 1.1 RSC encoder values

d_t	M1	M2	S	X_t^s	X_t^P
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	0	1	1
1	1	0	0	1	0
1	1	1	1	1	0

The convolutional code's properties are usually presented in graphical form by using one of the three equivalent diagrams: code trellis, state diagram and code tree.

The state diagram of the RSC encoder with inputs and outputs shown in above table is presented in Figure 2.3.

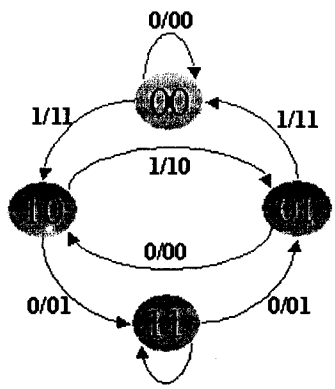


Figure 2.3 State diagram of RSC encoder

Figure 2.3 is a finite state machine that changes its state when an input symbol enters to the machine and produces two output symbols during the transition (since the code rate is ½). Although we obtain the state relationships from the state diagram, we cannot obtain the time information from it. Usually the convolutional encoders are forced to the zero state at the end of the coding, a process called trellis termination, in order to make the decoding process easier. It is done by placing the M zero bits at the end of the information sequence. This causes rate loss. For long blocklengths, the rate loss is acceptable, but for short blocklengths, the termination causes severe degradation in the system performance because, those additional zero bits carry no information. The trellis diagram created by the encoder (7, 5) is shown in Figure 2.4.

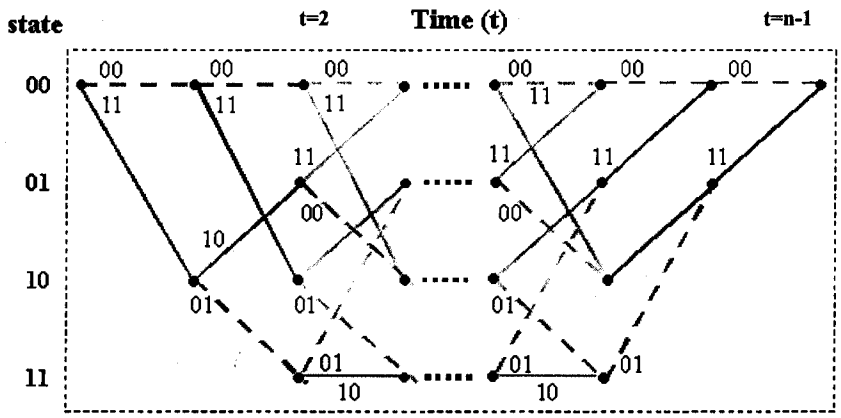


Figure 2.4 Trellis diagram of the RSC decoder

It starts from state 0 and ends again to state 0. State numbers show the values inside the encoder memories.

The middle section of the code trellis created by encoder (7, 5) is shown in Figure 2.5. The dashed lines indicate zero-transition (-1 in BPSK modulation) and the solid lines indicate one-transition branches. The branch labels are also shown in this Figure.

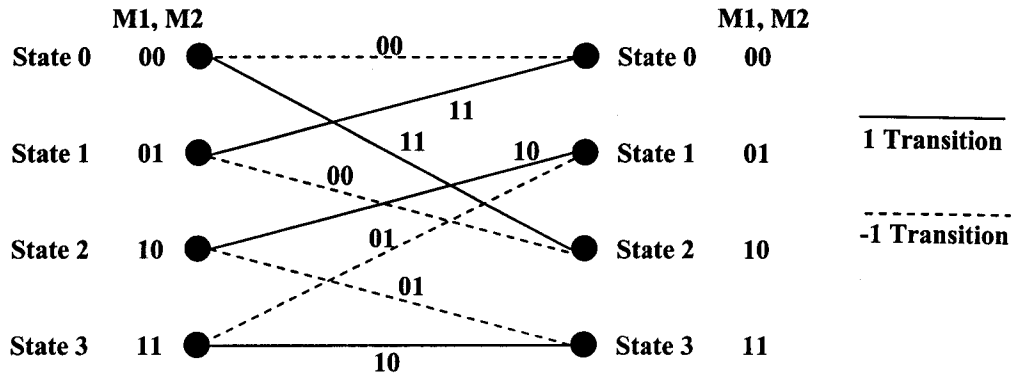


Figure 2.5 A middle section of the trellis diagram

The code tree of the code (7, 5) is shown in Figure [tree]. The output sequence follows the upper branch when zero enters to the encoder and lower branch when one enters to the encoder as an input. The code tree for code (7, 5) starts repeating itself after the third branch because it is a memory 2 code and after the third branch, the first input bit has no affect on the output sequence. The number of possible branches in the code tree doubles itself at each time unit. For an L-bit input sequence, there are 2^L distinct possible paths that the code can follow on the tree.

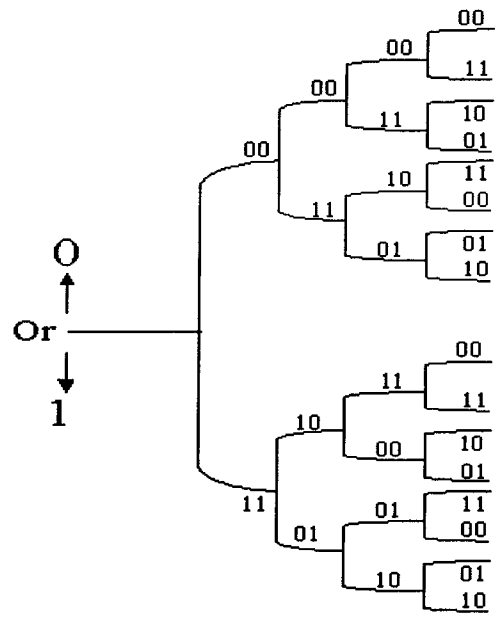


Figure 2.6 Code tree representation

2.3 Interleavers

An interleaver changes the order of the information sequence for the second and other encoders in pseudo-random fashion. Intereavers are widely used to improve code performance for channels, which introduce burst errors, but the place and function of an interleaver in the turbo codes are different from those of previously employed interleavers. The turbo interleaver reorders the uncoded information sequence for the second and other encoders so that the correlation between encoded sequences is arbitrarily low. In this way each generated code can be treated as independent. A good interleaver can significantly increase the performance of turbo coding by increasing the separation between sequences.

2.3.1 Pseudo-Random Interleavers

One of the fundamental components of turbo codes is the pseudo-random interleaver. This design technique generates an overall code, which appears random but is actually composed of two or more structured codes. The random characteristics of the code result in good performance, yet the overall code is easily decoded at the receiver by breaking it down into its structured, constituent components. Choosing a good interleaver design is important for obtaining good turbo code performance, but the most significant parameter relating to the interleaver is its size. As the interleaver size increases, performance improves. There is a tradeoff, however, between performance and latency.

2.3.2 Convolutional Interleavers

There are a number of interleavers to choose from when designing PCCCs. Block interleavers tend to give poor performance because they do not adequately break apart certain input sequences which result in low weight codewords. Convolutional interleaving also results in an interleaving pattern where low weight codewords are likely to degrade performance. Simulation results have shown that convolutional interleaving between constituent RSC encoders yields poor performance for these reasons [refthes14]. A

comparison of block, convolutional, and random interleaving is shown in Figure 2.7. A rate 1/3, constraint-length 4, turbo code with generator matrices $G = (13, 15)_{octal}$ was used to encode 1024-bit data frames. The Log-MAP algorithm was used for decoding, and plots are shown for six decoding iterations. Twenty-five errors were logged at each value of Eb/No. In general, when structure is introduced to the interleaver design, turbo code performance suffers. With a few exceptions, random interleavers provide good performance. Slightly improved performance can be obtained with a spread interleaver. A spread interleaver is a pseudo-random interleaver designed according to an algorithm, which guarantees that a specified distance always separates consecutive input symbols in the output sequence.

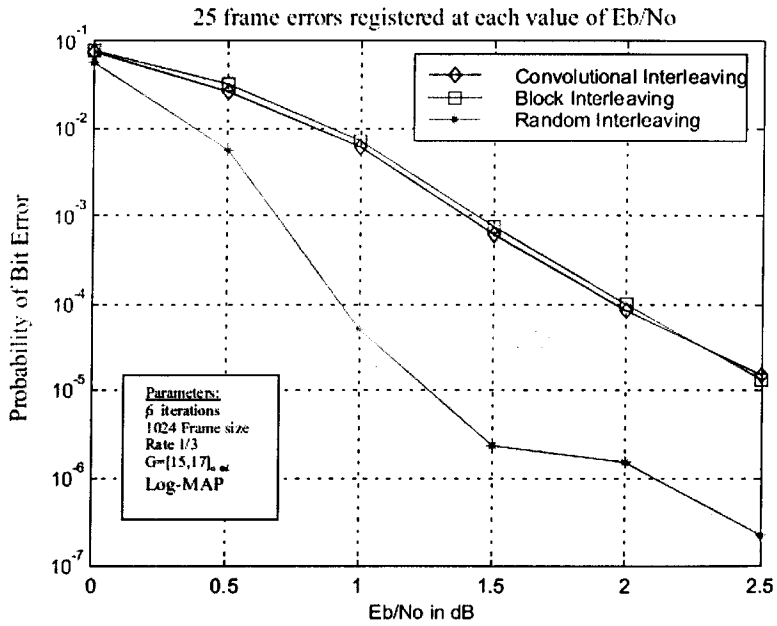


Figure 2.7 Comparison of block, convolutional, and random interleaving

2.4 Turbo Decoder

The whole decoder consists of two constituent decoders, deinterleavers and interleavers. Each decoder operates on the systematic and parity bits (channel outputs) associated with its constituent encoder and produces soft outputs of the original data bits in the form of extrinsic values. Soft output gives the probability of each received bit from the channel to be 0 or 1 (-1 or 1 in BPSK modulation). The decoders then share their

respective soft information in an iterative fashion. The goal of this project is implementation of these two decoders or soft input soft output (SISO) modules and if MAP algorithm runs through these decoders, they are called maximum a posteriori (MAP) decoders. The extrinsic output of each decoder is interleaved (or deinterleaved) and passed to the next decoding module as a priori information. Decoding continues in an iterative fashion for a fixed number of iterations or until a given convergence criteria is met. Because iterative decoding is subject to diminishing returns, the coding gains realized with each additional iteration are less than for the previous iteration. This principle is illustrated in Figure 2.8.

Although turbo codes integrate the two desirable qualities of pseudo-randomness and ease of decoding, the important contribution to communication theory lies in the iterative decoding method used to decode them. This iterative strategy has been employed in other communications areas such as iterative multiuser detection, turbo equalization, and turbo code assisted synchronization with good results.

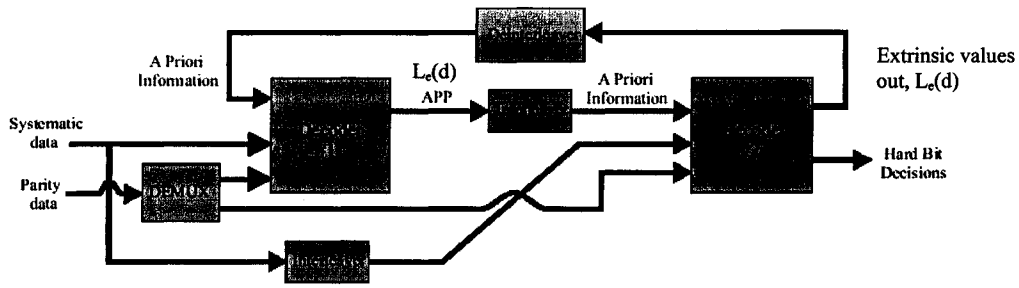


Figure 2.8 Turbo decoder

Turbo decoder is a soft input soft output (SISO) decoder and for a systematic code, it can be shown [turbo] that the soft output of the decoder $L(d)$ is equal to:

$$L(d) = L'(d) + L_e(d)$$

where $L'(d)$ is a priori log likelihood ratio and $L_e(d)$ is extrinsic log likelihood ratio.

$$L'(d) = \log\left[\frac{p(x \mid d = +1)}{p(x \mid d = -1)}\right]$$

$$L_e(d) = \log\left[\frac{p(d = +1)}{p(d = -1)}\right]$$

The soft output or log-likelihood ratio (LLR) provides the probability of the channel output bits and decides if the channel output bits have been originally 1 or –1. The soft decision $L(d)$ is a real number that provides a hard decision as well as the reliability of that decision. The sign of $L(d)$ denotes the hard decision; that is, for a positive value of $L(d)$ decides that the channel output bit has been originally +1 and for negative values decides that the bit has been –1. The magnitude of $L(d)$ denotes the reliability of that decision.

2.5 Turbo Coding Algorithms

Having outlined the iterative decoding process, the specific decoding algorithms used by the SISO modules will now be described. This section highlights two classes of trellis-based algorithms, which are typically used to decode turbo codes. Figure 2.9 lists the two classes of trellis-based algorithms.

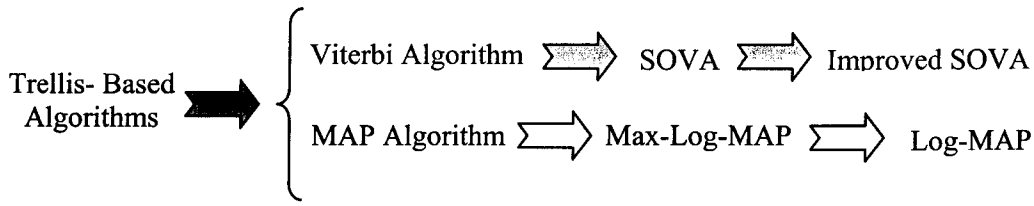


Figure 2.9 Turbo decoding algorithms

In 1967 the VA was presented in [viterbi1] as a practical procedure for maximum-likelihood decoding of convolutional codes. The VA is optimal for estimating the state sequence of a finite-state Markov process observed over a discrete memoryless channel (DMC)[forney]. The VA minimizes the frame error rate by finding the most likely path

through the trellis. The VA is unsuitable for turbo decoding because it is a soft-input, hard-output algorithm. A soft-output VA (SOVA), introduced by Hagenauer and Hoeher in [hagenauer1], is a SISO algorithm, which retains information related to the pruned, competing paths. This information determines the reliability of the bits, which differ from those in the surviving path.

In 1996 the Improved SOVA was developed in [papke] to combat an inherent bias with SOVA. The bias is removed by multiplying the SOVA output by a normalizing constant derived from the estimated mean and variance of the output. A small performance increase mitigates the slight increase in computational complexity.

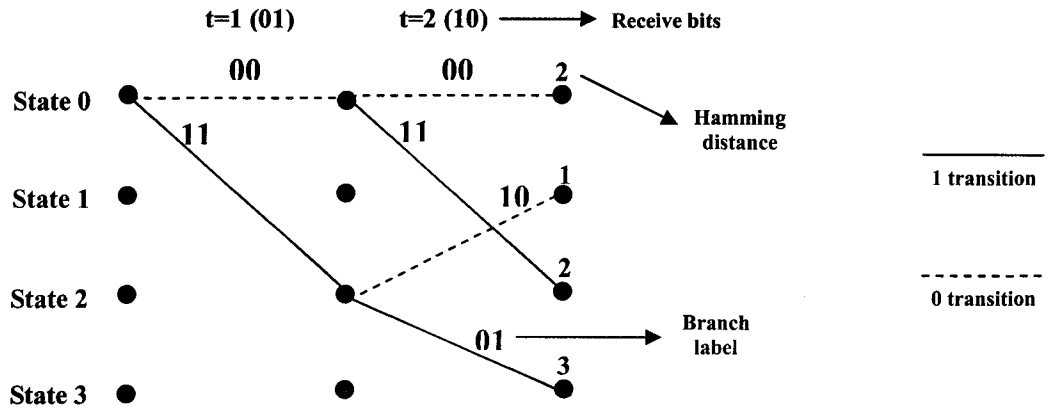
2.5.1 Viterbi Algorithm (VA)

VA starts with a known initial state and the metric of the path originating starting from this state is zero. Then the received sequence at each time instant is compared stage by stage with branch labels of the code and the distances between these two are measured by a metric. These metrics are added to the paths' previous metrics. For the binary symmetric channel (BSC), the Hamming distance[1] between the branch labels and received sequence is a good choice for the metric. When the paths merge at a particular state, the paths with the bigger metrics are dropped and the path with the smaller metric survives for next step. If the merging paths have the same metric, then the surviving path will be chosen randomly among them. The procedure is repeated until the end of the trellis is reached and the path whose metric is the smallest is hopefully the correct path and the bit sequence, which creates this path, is hopefully the error free decoded sequence. Figure 2.10 shows the trellis diagram for the VA algorithm with the generator (7, 5). Suppose that we receive the sequence (01, 10, 11, 00). The algorithm starts decoding from the zero state at time t=1. Figure 2.10(a) shows the paths after decoding for received bits (01) and (10) and each path metric is shown above the paths. The next received sequence is (11). Figure 2.10(b) shows the path's new metrics after decoding for that sequence. The X marked paths are dropped paths. Paths with larger Hamming distances drop and the path with the least distance remains. Figure 2.10(c) shows the

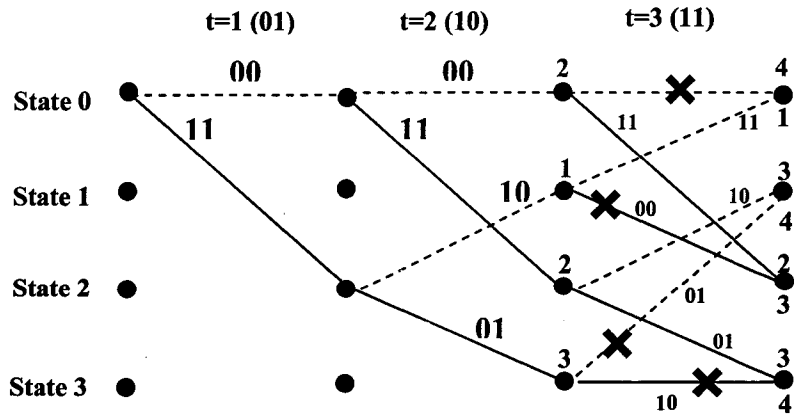
[1] The Hamming distance is the number of bits which differ between two binary strings

paths after decoding the last received sequence (00). At that stage, the entire path's metrics of the state 2 are the same and we drop one of them randomly. Figure 2.10(d) shows the final survivor paths. According to Figure 2.10(d), the most likely decoded sequence will be (1000).

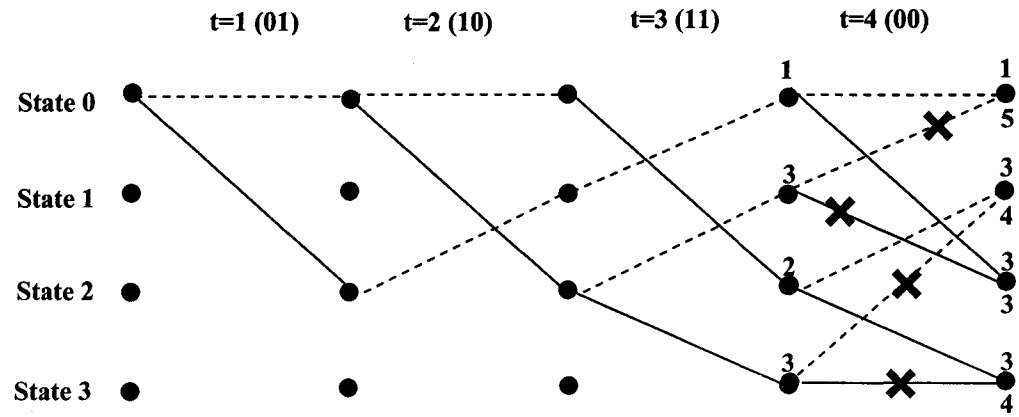
As we can see from Figure 2.10, we have always 4 survivor paths for t>=2. In general, we have always 2^{k-1} survivor paths after t>=k-1 for a constraint length K convolutional code, because only one paths will survive among the paths entering to a state. Due to the exponential increase on the number of survivor paths, 2^{k-1}, with respect to the constraint length, K, the utilization of the VA is practical for relatively short constraint length codes.



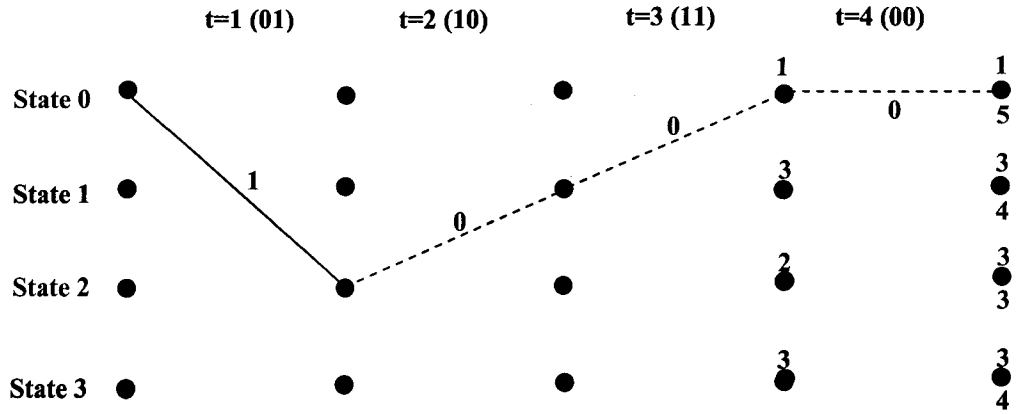
(a)



(b)



(c)



(d)

Figure 2.10 Viterbi algorithm with generator (7, 5) for BSC

2.5.2 MAP/BCJR Algorithm

Based on an algorithm developed by Chang and Hancock for removing inter-symbol interference, the MAP algorithm was introduced in 1974 as an optimal means for estimating the a posteriori probabilities (APPs) for a finite-state Markov process observed over a DMC [bahl].

The MAP algorithm, also known as the BCJR algorithm for the four researchers who developed it, is a forward-backwards recursion algorithm, which minimizes the probability of bit error rate. Therefore, the path that the MAP algorithm traces through the trellis need not be connected, as is the case for the VA. In the 1970's the MAP algorithm fell out of favor for decoding convolutional codes because of its computational complexity in comparison to VA. When SISO decoding of turbo codes became an important issue, the Max-Log-MAP and Log-MAP algorithms were introduced to solve the instability problem; they are now the preferred SISO algorithms used to decode turbo codes. In the following thesis, these algorithms are explained and MAP algorithm is described first.

Transition probabilities and output probabilities in MAP algorithm are represented by the following equations [bahl],

$$p_t(m \mid m') = \Pr\{S_t = m \mid S_{t-1} = m'\}$$
$$q_t(X \mid m', m) = \Pr\{X_t = X \mid S_{t-1} = m'; S_t = m\}$$

where m and m' are transitions and X is the encoder output. The source starts in the initial state $S_t = 0$, and produces an output sequence X_1^t ending in the terminal state $S_t = 0$. X_1^t is the input to a noisy discrete memoryless channel (DMC), whose output is the sequence $Y_1^t = Y_1, Y_2, \ldots Y_t$. The transition probabilities of the channel are defined by R(.|.) so that

$$\Pr\{Y_1^t \mid X_1^t\} = \prod_{j=1}^{t} R(Y_j \mid X_j)$$

The objective of the decoder is to examine Y_1^t and estimate the a priori probability (APP) of the states and transitions of the source or encoder, i.e., the conditional probabilities

$$\Pr\{S_t = m \mid Y_1^\tau\} = \Pr\{S_t = m; Y_1^\tau\} / \Pr\{Y_1^\tau\} \quad (1)$$

and

$$\Pr\{S_{t-1} = m'; S_t = m \mid Y_1^\tau\} = \Pr\{S_{t-1} = m'; S_t = m; Y_1^\tau\} / \Pr\{Y_1^\tau\} \qquad (2)$$

A graphical interpretation of the problem is shown in Figure 2.3. The nodes are the states and the branches represent the transitions having nonzero probabilities. If we index the states with both the time index t and state index m, we get the "trellis" diagram of Figure 2.4. The trellis diagram shows the time progression of the state sequences. For every state sequence S_1^t there is a unique path through the trellis diagram, and vice versa.

If the source is time variant, then we can no longer represent it by a state-transition diagram; however, it is obvious that we can construct a trellis for its state sequences.

Associated with each node in the trellis is the corresponding APP $\Pr\{S_t = m \mid Y_1^\tau\}$ and associated with each branch in the trellis is the corresponding APP $\Pr\{S_{t-1} = m'; S_t = m \mid Y_1^\tau\}$. The objective of the decoder is to examine Y_1^t and compute these APPs.

For ease of exposition, it is simpler to derive the joint probabilities

$$\lambda_t(m) = \Pr\{S_t = m; Y_1^\tau\}$$

and

$$\sigma_t(m', m) = \Pr\{S_{t-1} = m'; S_t = m; Y_1^\tau\}$$

Since, for a given Y_1^τ, $\Pr\{Y_1^\tau\}$ is a constant, we can divide $\lambda_t(m)$ and $\sigma_t(m', m)$ by $\Pr\{Y_1^\tau\}$ $(=\lambda_t(0)$, which is available from the decoder) to obtain the conditional probabilities of (1) and (2). Alternatively, we can normalize $\lambda_t(m)$ and $\sigma_t(m', m)$ to add up to 1 to obtain the same result. We now derive a method for obtaining the probabilities $\lambda_t(m)$ and $\sigma_t(m', m)$. Let us define the probability functions:

$$\alpha_t(m) = \Pr\{S_t = m; Y_1^t\}$$

$$\beta_t(m) = \Pr\{Y_{t+1}^{\tau} \mid S_t = m\}$$

$$\gamma_t(m', m) = \Pr\{S_t = m; Y_t \mid S_{t-1} = m'\}$$

Now

$$\lambda_t(m) = \Pr\{S_t = m; Y_1^t\}.\Pr\{Y_{t+1}^{\tau} \mid S_t = m; Y_1^t\}$$

$$= \alpha_t(m).\Pr\{Y_{t+1}^{\tau} \mid S_t = m\}$$

$$= \alpha_t(m).\beta_t(m) \tag{3}$$

The middle equality follows from the Markov property that if S_t is known, events after time t do not depend on Y_1^t.

Similarly,

$$\sigma_t(m', m) = \Pr\{S_{t-1} = m'; Y_1^{t-1}\}.\Pr\{S_t = m; Y_t \mid S_{t-1} = m'\}.\Pr\{Y_{t+1}^{\tau} \mid S_t = m\}$$

$$= \alpha_{t-1}(m').\gamma_t(m', m).\beta_t(m) \tag{4}$$

Now for $t = 1,2, \ldots, \tau$

$$\alpha_t(m) = \sum_{m'=0}^{M-1} \Pr\{S_{t-1} = m'; S_t = m; Y_1^t\}$$

$$= \sum_{m'} \Pr\{S_{t-1} = m'; Y_1^{t-1}\}.\Pr\{S_t = m; Y_t \mid S_{t-1} = m'\}$$

$$= \sum_{m'} \alpha_{t-1}(m').\gamma_t(m', m) \tag{5}$$

Again, the middle equality follows from the fact that events after time t - 1 are not influenced by Y_1^{t-1} if S_{t-1} is known.

For $t = 0$ we have the boundary conditions

$$\alpha_0(0) = 1, \text{ and } \alpha_0(m) = 0, \qquad \text{for } m \neq 0. \tag{6}$$

Similarly, for t = 1,2,. . ., τ - 1

$$\beta_t(m) = \sum_{m'=0}^{M-1} \Pr\{S_{t+1} = m'; Y_{t+1}^{\tau} \mid S_t = m\}$$

$$= \sum_{m'} \Pr\{S_{t+1} = m'; Y_{t+1} \mid S_t = m\}.\Pr\{Y_{t+2}^{\tau} \mid S_{t+1} = m'\}$$

$$= \sum_{m'} \beta_{t+1}(m').\gamma_{t+1}(m, m') \tag{7}$$

The appropriate boundary conditions are

$$\beta_\tau(0) = 1, \text{ and } \beta_\tau(m) = 0, \qquad \text{for } m \neq 0. \tag{8}$$

Relations (5) and (7) show that $\alpha_t(m)$ and $\beta_\tau(m)$ are recursively obtainable. Now

$$\gamma_t(m', m) = \sum_X \Pr\{S_t = m \mid S_{t-1} = m'\}.\Pr\{X_t = X \mid S_{t-1} = m', S_t = m\}.\Pr\{Y_t \mid X\}$$

$$= \sum_X p_t(m \mid m').q_t(X \mid m', m).R(Y_t \mid X) \tag{9}$$

where the summation in (9) is over all possible output symbols X.

In the equation (9), $p_t(m \mid m')$ is the transition probability and the output X, is a deterministic function of the transition so that, for each transition, there is a 0 - 1 probability distribution $q_t(X \mid m', m)$. For time invariant codes $q_t(.|.)$ is independent of t. If the output sequence is sent over a DMC with symbol transition probabilities $r_t(.|.)$, the derived block transition probabilities are

$$R\ (Y_t \mid X_t) = \prod_{j=1}^{n_0} r(y^{(j)} \mid x_t^{(j)})$$

where $Y_t = (y_t^{(1)}, ..., y_t^{(n_0)})$ is the block received by the receiver at time t. For instance, in a BSC with crossover probability p_c:

$$R\ (Y_t \mid X_t) = (p_c)^d (1-p_c)^{n-d}$$

where d is the Hamming distance between X_t, and Y_t.

2.5.3 Results of the MAP Algorithm

The ultimate result of the MAP algorithm is the logarithm of the ratio of the a posteriori probability (APP) of each information bit being 1 to the APP of it being 0.

$$\Lambda(X_{t+1}) = \log \frac{\sum_{(m',m),X=1} \gamma_{t+1}(m',m).\beta_{t+1}(m).\alpha_t(m')}{\sum_{(m',m),X=-1} \gamma_{t+1}(m',m).\beta_{t+1}(m).\alpha_t(m')} \tag{10}$$

In this design binary phase shift-keying (BPSK) modulation is considered so that, bits with the value of 0 and 1 are mapped to –1 and 1 respectively. Forward node metric at time t is calculated from the computed forward node metric at time *t-1* and the branch metric at time *t*:

$$\alpha_t(m) = \sum_{m'} \alpha_{t-1}(m').\gamma_t(m',m)$$

Backward node metric at time *t* is calculated from the computed forward node metric and the branch metric at the current time *t+1*:

$$\beta_t(m') = \sum_{m} \beta_{t+1}(m).\gamma_{t+1}(m',m)$$

Transition metric or Gamma is the symbol transition probability and for a 1/2 code rate encoder can be computed as:

$$\gamma_t(m',m) = \sum_{x} p_t(m \mid m').q_t(X_t \mid m',m).R(Y_{t_d} \mid X_t).R(Y_{t_p} \mid X_t) \tag{11}$$

where, X_t is the channel input symbol at time t and Y_d and Y_p are channel outputs for Systematic data and Parity data respectively q_t and R are probability functions that mentioned before.

2.5.4 Max-Log-MAP Algorithm[robertson2]

The complexity of the MAP algorithm is high in terms of number of operations, especially number of multiplications. Max-Log-MAP algorithm is the logarithmic approximation of MAP algorithm therefore; we work with the logarithms of the values mentioned in MAP algorithm using the following approximation:

$$\ln(e^{\gamma_1} + \ldots + e^{\gamma_n}) \approx \max_{i \in \{1 \ldots n\}} \gamma_i \tag{12}$$

In AWGN, the maximum likelihood probability function, $R(Y_t \mid X_t)$, is equal to:

$$R(Y_t \mid X_t) = \frac{1}{\sqrt{\pi N_0}} . e^{-\frac{1}{N_0}(Y_t - X_t)^2} \tag{13}$$

By taking the logarithm of equation (11) and substituting for R from the equation (13) we obtain the following equation for branch metric:

$$\ln \gamma_t(m', m) = \frac{2Y_{t_d} X_t}{N_0} + \frac{2Y_{t_p} X_t}{N_0} + \ln AP_t + K \tag{14}$$

where $N_0/2$ is the variance of the noise in Additive White Gaussian Noise (AWGN) channel and K is a constant which can be ignored since it cancels out during the recursive calculations. Equation (14) is a less complex algorithm in comparison to (11); however there are still some multiplications and divisions in (14) that can be removed and changed to an equation with only additions and comparisons without using any approximation. This fact is explained in section 3.2.

Forward and backward node metrics in MAP algorithm can be shown by the following equations:

$$\alpha_t(m) = \frac{\sum_{m'} \alpha_{t-1}(m').\gamma_t(m',m)}{\sum_{m}\sum_{m'} \alpha_{t-1}(m').\gamma_t(m',m)} \tag{15}$$

$$\beta_t(m') = \frac{\sum_{m} \beta_{t+1}(m).\gamma_{t+1}(m',m)}{\sum_{m'}\sum_{m} \beta_{t+1}(m).\gamma_{t+1}(m',m)} \tag{16}$$

By taking the logarithm of α and β in equations (15) and (16) and using approximation (12), we get:

$$\ln \alpha_t(m) = \ln(\sum_{m'} e^{\ln(\alpha_{t-1}(m').\gamma_t(m',m))}) - \ln(\sum_{m}\sum_{m'} e^{\ln(\alpha_{t-1}(m').\gamma_t(m',m))})$$

and

$$\ln \beta_t(m') = \ln(\sum_{m} e^{\ln(\beta_{t+1}(m).\gamma_{t+1}(m',m))}) - \ln(\sum_{m'}\sum_{m} e^{\ln(\beta_{t+1}(m).\gamma_{t+1}(m',m))})$$

to get:

$$\ln \alpha_t(m) = \max_{m'}[\ln \alpha_{t-1}(m') + \ln \gamma_t(m',m)] - \max_{(m',m)}[\ln \alpha_{t-1}(m') + \ln \gamma_t(m',m)]$$

$$\ln \beta_t(m') = \max_{m}[\ln \beta_{t+1}(m') + \ln \gamma_{t+1}(m',m)] - \max_{(m,m')}[\ln \beta_{t+1}(m') + \ln \gamma_{t+1}(m',m)]$$

The second terms are a result of the derivation from (15) and (16) and because these normalization terms will cancel out in (17), omitting them has no effects on the output of the Max-Log-MAP algorithm and the final equation will be:

$$\ln \alpha_t(m) = \max_{m'}[\ln \alpha_{t-1}(m') + \ln \gamma_t(m',m)]$$

$$\ln \beta_t(m') = \max_{m}[\ln \beta_{t+1}(m') + \ln \gamma_{t+1}(m',m)]$$

There are no multiplications exist in the above equation and it is only includes additions and comparisons. Therefore, this is obvious that the above equations are less complex and need smaller area size for implementation.

The Log-likelihood reliability is also computed from branch and node metrics as follow:

$$\ln \Lambda_{t+1} \approx \max_{(m,m'),X=1} [\ln \gamma_{t+1}(m',m) + \ln \beta_{t+1}(m) + \ln \alpha_t(m',m)] - \max_{(m,m'),X=-1} [\ln \gamma_{t+1}(m',m) + \ln \beta_{t+1}(m) + \ln \alpha_t(m',m)]$$

Again by looking at the above equation and compare it to the same equation in MAP algorithm (equation (10)), we can conclude that LLR obtained in Max-Log-MAP algorithm is less complex.

The Max-Log-MAP algorithm has a performance loss of about 0.5dB that can be improved ~0.4dB more by using a scaling factor within the extrinsic information in turbo decoders [vogt1]. This concept will be explained later.

2.5.5 Log-MAP Algorithm [robertson2]

Applying a correction term to the Max-Log-MAP algorithm forms the Log-MAP algorithm:

$$\ln(e^{\delta_1} + e^{\delta_2}) = \max(\delta_1, \delta_2) + \ln(1 + e^{-|\delta_1 - \delta_2|})$$

$$= \max(\delta_1, \delta_2) + f_c(|\delta_1 - \delta_2|)$$

where $f_c(.)$ is a correction function and by considering $\delta = \ln(e^{\delta_1} + ... + e^{\delta_{n-1}})$ we can show that:

$$\ln(e^{\delta_1} + ... + e^{\delta_n})$$

$$= \ln(\Delta + e^{\delta_n}) \text{ with } \Delta = e^{\delta_1} + ... + e^{\delta_{n-1}} = e^{\delta}$$

$$= \max(\ln \Delta, \delta_n) + f_c(|\ln \Delta - \delta_n|)$$

$$= \max(\delta, \delta_n) + f_c(|\delta - \delta_n|)$$

as a result, by correcting the approximation made in Max-Log-MAP at each step, the performance of Log-MAP algorithm will be almost the same as MAP algorithm.

Term "$f_c(|\delta - \delta_n|)$" compensates the approximation error in equation (12). But correction is done using look-up tables in practical implementations and multiple look up tables are required for a wide range of operating signal-to-noise ratios (SNRs), thus it increases the hardware cost [cheng].

2.5.6 Sliding MAP

Sliding window MAP [chass] performs the calculations on the sub block instead of recursive calculation on the whole received block.

The size of the sub block is in the order of the survivor length of the Viterbi algorithm, SMAP reduces the demand on memory. A disadvantage is increase in the complexity, since for each sub block a reliable start vector for the backward recursion has to be provided. That means, instead of one forward and one backward recursion now one forward and two backward recursions are necessary.

2.6 Improving the Max-log-MAP Turbo Decoder

Remember from section 2.4 that:

$$L(d) = L'(d) + L_e(d)$$

where $L(d)$ is a posteriori probability, $L'(d)$ is soft output of the decoder and $L_e(d)$ is extrinsic probability. By applying an additional scaling factor, R, to the extrinsic information the result will be:

$$L(d) = L'(d) + R.L_e(d)$$

The scaling factor R allows us to change the amount of information exchanged between two constituent decoders. Larger R makes the previous decoder outcome dominate the current decoding result, whereas the smaller R makes one decoder less dependent on the other decoder's result. In [Wu1], the scaling factor R=0.6, and in [vogt1], the scaling factor R=0.7 have given the best BER performance for AWGN. A properly selected scaling factor could improve the Max-Log-MAP by ~0.4 dB, yielding a near optimal (Log-MAP) BER performance.

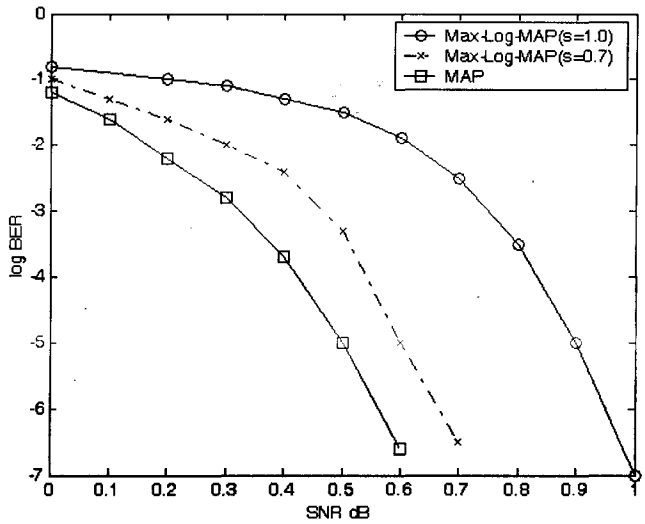


Figure 2.11 Turbo code with different scaling factors, block length 5114 bit, 8iterations, AWGN, rate 1/3 and generators $(13, 15)_{oct}$

The BER improvement is due to the fact that the scaling factor R can effectively mitigate error propagation through iterations. Normally in Max-Log-MAP, when a cluster of extrinsic values of the first decoder are in error, they would dominate the branch metrics in the second decoder, and in turn causes more decoding errors in the second decoder. Because the decoding is iterative, more errors would be produced in the end. However, with the scaling factor the effect of erroneous a-priori values on branch metrics is reduced, allowing the decoder to upset the previous incorrect decoding results.

Figure 2.11 shows the performance of the best-evaluated scaling factor compare to the standard algorithm (s=1.0) for block length 5114 and AWGN channel. For a bit error rate of 10^6 the improvement of the Max-Log-MAP is ~0.3dB and the difference between Max-Log-MAP and MAP is now only 0.1dB.

2.7 Algorithm Comparison

Figure 2.12 shows the comparison between different algorithm's performances [vogt2]. SOVA has the maximum performance loss and Log-MAP and MAP, which are optimum algorithms, have the best performance. The Max-Log-MAP algorithm has a performance loss of about 0.4dB and as it mentioned in previous section, it can be improved up to ~0.3dB more by using a scaling factor within the extrinsic information in turbo decoders.

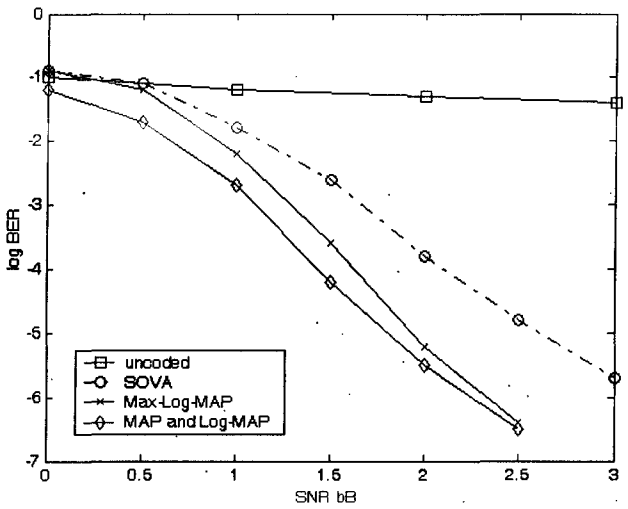


Figure 2.12 Comparison between performances of turbo coding algorithms, block length 668bit, 4 iterations, AWGN, rate 1/3 and generators (7, 5)

Comparison between computational complexity shows that the MAP algorithm is the most complex algorithm and not suitable for implementation. This fact is observable from the MAP algorithm equations. Moreover, for implementation of the additional of the Log-MAP algorithm, multiple look up tables are required for varies signal to noise ratios and, which makes the design larger and increases the cost. Therefore, Max-Log-MAP

algorithm is chosen for the purpose of implementation of MAP decoder, which is the best compromise between performance and complexity.

2.8 System Specification Summary

The proposed decoder specifications are as follows:

- Encoder: Recursive Systematic Convolutional (RSC)
- Channel: Additive White Gaussian Noise (AWGN)
- Modulation: Binary Phase Shift Keying (BPSK)
- Number Of Memories: 2
- Code Rate: 1/2
- Frame size: variable up to 1024 bits

Chapter 3

System Design and Modeling

A major difficulty of applying turbo codes in many applications is the decoding complexity of SISO decoders. Efficient implementations of those decoders can significantly increase employment of turbo codes in different communication applications.

After introducing turbo coding and its related algorithms and choosing Max-Log-MAP algorithm for implementation, we discuss the system design and the proposed architecture for the implementation of MAP decoder. In this chapter an efficient design of a MAP Decoder suitable for ASIC, with lower complexity and smaller area size is introduced. The performance loss of the proposed design is less than 0.1dB in comparison with the best performance observed in Max-Log-MAP algorithm.

At the beginning, quantization and normalization techniques are studied and those that best fit the proposed MAP decoder are described. Furthermore, a new design with lower complexity is introduced for transition probability (γ) calculation unit. The architecture used for this design is parallel and pipeline, which explained in this chapter.

3.1 Max-Log-MAP Decoder Block Diagram

It was mentioned in chapter 2 that four parameters are calculated in Max-Log-MAP algorithm.

1. γ : branch metrics or probability of transitions
2. α and β : Node metrics or probability of states
3. Λ : Log likelihood ratio (LLR)

Figure 3.1 shows the block diagram of the proposed Max-Log-MAP decoder and its units. Inputs to the decoder are the a priori probability, systematic data and parity data. The a priori probability comes from the output of another Max-Log-MAP decoder inside the turbo decoder. Remember that turbo decoder includes two constituent decoders and Figure 3.1 shows one of them. Systematic and parity data inputs on the other hand are channel outputs and distorted by the noise of the channel. In a pipeline process bits received in input go to branch metric calculation unit and afterward to α calculation unit. γ and α values are then computed and saved into memories. Therefore, in forward recursion, γ and α calculation units work in parallel, until one complete frame or block of bits is received. By reaching to the end of each frame, γ and α units stop their execution, backward recursion starts and β and soft output (LLR) calculation units compute their values in parallel. The output is a probability that help us decide for each received bit from the channel to be originally 1 or –1.

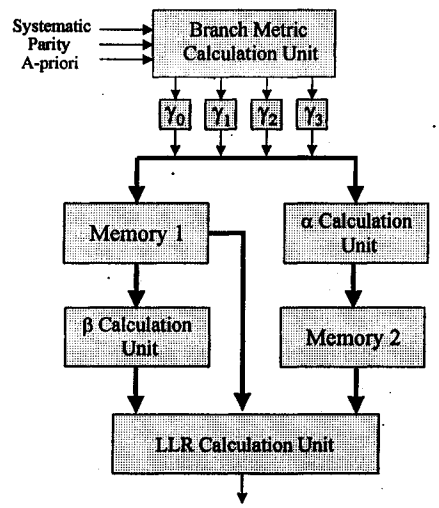


Figure 3.1 Block diagram of the proposed Max-Log-MAP decoder

This design is very fast because of its parallel and pipeline nature and suitable for implementation. The only drawback of parallel and pipeline architecture is the need for using large memories, which in tern makes the design larger. In the proposed architecture, by computation of the LLR in backward recursion and in parallel with β calculation unit, one large memory is omitted and it saves about 30% of the area size and reduces the implementation cost of the chip.

Memories used are 4k, 8bits and there are for different γ or α values for each received bit. Therefore, a 4k memory can store γ or α values for a frame of up to 1024 bits. The number of bits per frame can be increases by using larger memories in other applications; however for a general application such as the current work, 1024 bits per frame is an adequate number.

3.2 New Branch Metric Calculation Unit

Recall from section 2.5.4 that there were still some multiplications and additions in equation (14). The decoder design can be further simplified by considering the insensitivity of Max-Log-MAP algorithm to the AWGN channel variance of the noise (N0/2). In [worm] it was shown that performance of turbo decoding with the Max-Log-MAP decoder, as its constituent decoder was SNR independent; however, turbo-decoding with the Log-MAP decoder theoretically requires SNR estimates. Elimination of the SNR significantly reduces the complexity of the decoder and yields an ASIC with a smaller area and lower cost.

If in Equation (14) we consider N_0=2,

$$\ln \gamma_t(m', m) = \frac{2Y_{t_d} X_t}{\cancel{N_0}\,2} + \frac{2Y_{t_p} X_t}{\cancel{N_0}\,2} + \ln AP_t + K$$

there will be no multiplication (or division) in the calculation of γ values. Since X_{t_p} is equal to either 1 or -1, the four final values for $\ln\gamma$ are:

$$\begin{aligned}
\ln\gamma_{t,00}(m',m) &= (-Y_{t_d} - Y_{t_p}) + \ln AP_t(-1) \\
\ln\gamma_{t,01}(m',m) &= (-Y_{t_d} + Y_{t_p}) + \ln AP_t(-1) \\
\ln\gamma_{t,10}(m',m) &= (+Y_{t_d} - Y_{t_p}) + \ln AP_t(+1) \\
\ln\gamma_{t,11}(m',m) &= (+Y_{t_d} + Y_{t_p}) + \ln AP_t(+1)
\end{aligned} \tag{3.1}$$

For implementation of these four equations, only adders and a small look up table are required. The input to the look up table is the a priori probability[2], and the outputs are $\ln AP_t(+1)$ and $\ln AP_t(-1)$. The following equations clear the relationship between input and outputs of the LUT:

$$APP = \ln\left(\frac{AP_t(+1)}{AP_t(-1)}\right)$$

and because

$$AP_t(+1) + AP_t(-1) = 1$$

we can write

$$APP = \ln\left(\frac{AP_t(+1)}{1 - AP_t(+1)}\right)$$

$$e^{APP} = \frac{AP_t(+1)}{1 - AP_t(+1)}$$

and

$$AP_t(+1) = \frac{e^{APP}}{1 + AP_t(+1)}$$

$$AP_t(-1) = 1 - \frac{e^{APP}}{1 + AP_t(+1)}$$

[2] In turbo decoders usually a posteriori probability is shown with APP, which is the decoder output, but here APP is the name of the input pin of the chip and shows the logarithm of the a priori probability.

as a result, the input to the decoder or LUT is logarithm of the a priori probability (APP) and the outputs of the LUT are:

$$\ln AP_t(+1) = APP - \ln(1+e^{APP})$$

and

$$\ln\ AP_t(1) = -\ln(1+e^{APP})$$

Figure 3.2 shows the design of the branch metric calculation unit. It shows how γ values of equations 3.1 are computed.

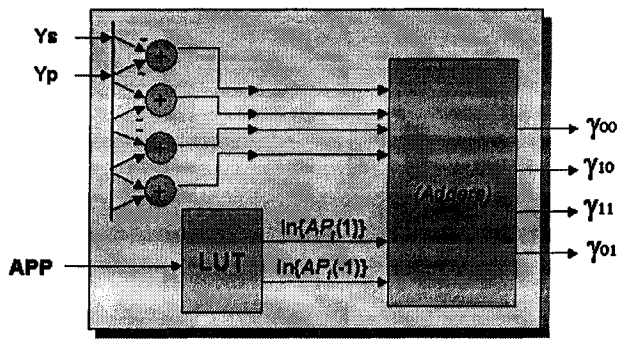


Figure 3.2 Branch metric calculation unit

3.3 Proposed Node Metric Calculation Unit

After explanation about the computation of the first parameter of the Max-Log-MAP algorithm or branch metric, we move to the second parameter or α and explanation about the α calculation unit. Figure 3.3 shows the parallel architecture of the α calculation unit. In this Figure $\bar{\alpha}$ is equal to $\ln\alpha$ and $\bar{\gamma}$ is equal to $\ln\gamma$.

Considering equation

$$\ln\alpha_t(m) = \max_{m'}[\ln\alpha_{t-1}(m') + \ln\gamma_t(m', m)]$$

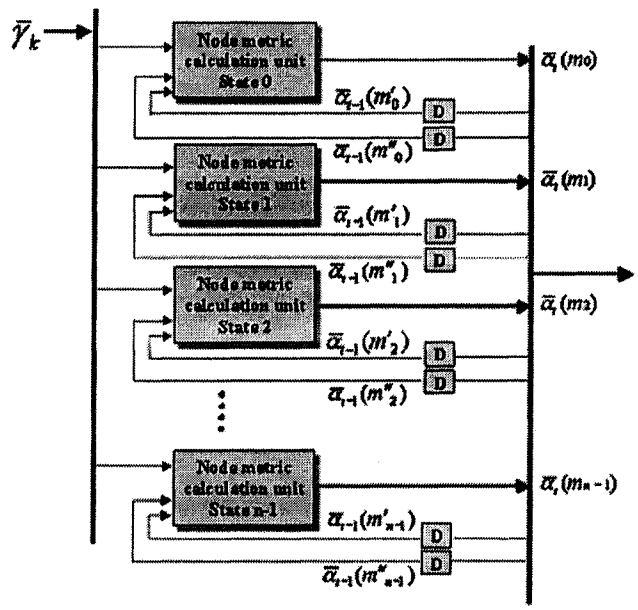


Figure 3.3 Parallel architecture of the α calculation unit

for calculation of α, we can see in forward recursion that, α values are calculated using current branch metrics and previous calculated α values. There can be *n* number of states and for each state there is a node metric calculation unit. According to the above equation node metric calculation units are add, compare and select units. It takes two clock cycles for each α value to be computed. Therefore, this architecture is very fast.

Considering β computation equation shows that the architecture of the β calculation unit is similar to the α calculation unit, except that β values are calculated in backward recursion. Therefore, β values at time *t* are calculated using β and γ values at time *t+1*.

$$\ln \beta_t(m') = \max_m [\ln \beta_{t+1}(m') + \ln \gamma_{t+1}(m', m)]$$

3.4 Log-likelihood Ratio (LLR)

As it mentioned in chapter 2, Log-likelihood ratio is our ultimate result and proposed decoder output. It gives us the probability of each received bit from the channel to be 0 or 1 (–1 or +1 in BPSK modulation).

$$\ln \Lambda_{t+1} \approx \max_{(m,m'),X=1} [\ln \gamma_{t+1}(m',m) + \ln \beta_{t+1}(m) + \ln \alpha_t(m',m)] - \max_{(m,m'),X=-1} [\ln \gamma_{t+1}(m',m) + \ln \beta_{t+1}(m) + \ln \alpha_t(m',m)]$$

in this equation, the part $\ln \gamma_{t+1}(m',m) + \ln \beta_{t+1}(m) + \ln \alpha_t(m',m)$ is called the a posteriori probability. Therefore, in this equation we should compute maximum a posteriori probability for *X=-1* and then subtract it from maximum a posteriori probability for *X=+1*.

3.5 Soft output calculation unit

Figure 3.4 shows the block diagram of the soft output calculation unit. Maximum a posteriori probability is computed from α, β and γ values coming from the data bus. After that, a subtractor subtracts maximum a posteriori probabilities for *X=-1* and *X=+1*. Add, compare units used in this design work in a pipeline process.

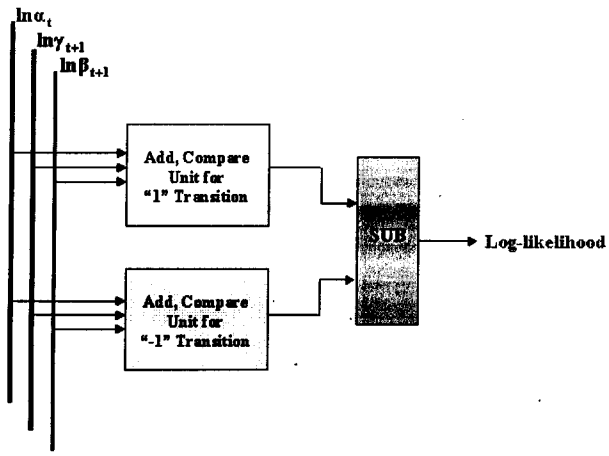


Figure 3.4 Log-likelihood calculation

The pipeline process of the LLR computation is shown in details in Figure 3.5. It takes five clock cycles for each LLR value to be computed. Figure shows the pipeline computation of the a posteriori probability for $X=1$ and $X=-1$. α, β and γ values of the different states enters the unit. The state numbers are for the proposed Max-Log-MAP decoder, which uses RSC encoder with rate 1/2.

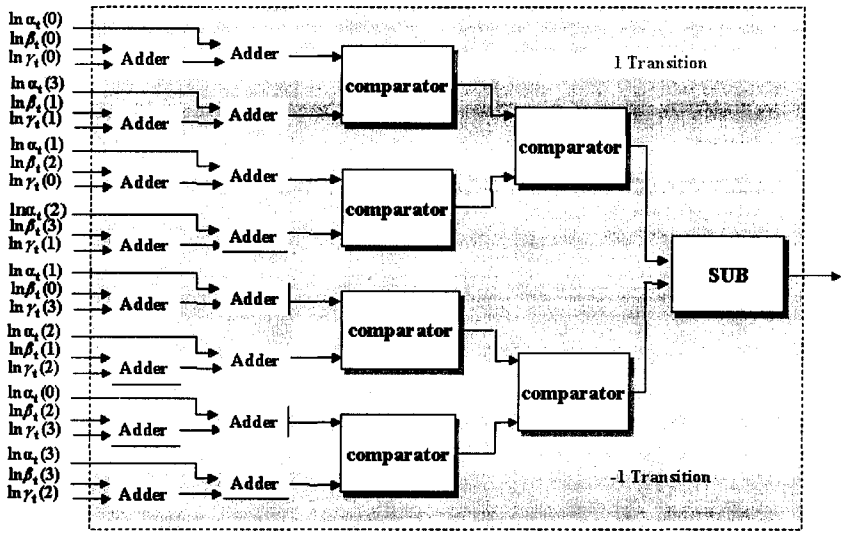


Figure 3.5 Pipeline calculation of LLR for 4 state Max-Log-MAP decoder

As a result, it takes 11 clock cycles for each LLR bit to be computed from the decoder input values.

3.6 Quantization

Because channel outputs are distorted by the noise of the channel, we should consider real values for decoder inputs. These inputs are stochastic values that can include a range from $[-\infty,\infty]$. This fact causes requirement for quantization of the values to a specific number of bits in a proper range. Different quantization techniques have been proposed in [montorsi], [wu1] and [michel2], which use various ranges and number of bits. The following table shows the differences between quantizations techniques used by them.

Table 3.1 comparison between different quantization techniques

References	Extrinsic values (Bits)	Data inputs (Bits)	Internal metrics (Bits)
Wu1	7	5	8
Montorsi, michel2	9	7	10
stralen	-	6	-
robertson	6	4	8
Proposed	8	6	8

First, the proposed quantization technique is explained and then it is compared to other techniques in terms of performance and complexity.

Simulation results show that if we consider data inputs (systematic and parity data) between [-4, +4] and truncate all other possible values, there will be 99% coverage for the channel outputs. This fact is also confirmed in [michel2]. Values between this range are real values. Therefore, in [wu1] only integer values are considered; however in other references fixed-point values are used.

It is experienced in simulation that by considering 6 bits fixed-point values for data inputs (3 bits for integer and 3 bits for decimal), the best comparison between performance and complexity will be obtained. Increasing the number of bits does not have much influence on the performance while; it increases complexity and cost. Also simulation results show the same fact by choosing 8 bits integer values for node metrics and extrinsic or LLR values. Figure 3.6 shows the result of using the proposed quantization. Simulation is done in AWGN and for RSC encoder with 400 bits block length and no iteration in Max-Log-MAP decoder. This result is compared to the best possible answer that Max-log-MAP algorithm can achieve in simulation. Best possible answer is when we do not quantize values in simulations i.e. considering any real values. Figure 3.6 shows that the performance difference is less than 0.1 dB, which is a reasonable result.

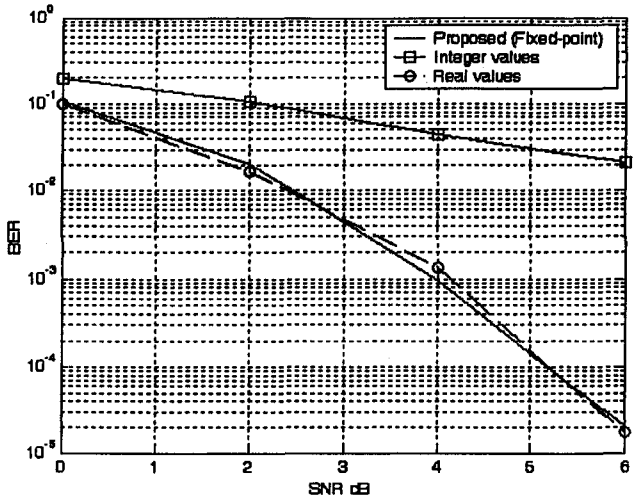


Figure 3.6 Comparison between proposed quantization, using integer values for quantization and no quantization, BCJR decoder, no iterations, RSC encoder, rate 1/2, block length 512, AWGN

In the proposed quantization, the a priori probability values (APP input of the decoder) are also quantized into integer values between [-8, 0]. The reason for this quantization is that after monitoring the a priori probability values by a histogram, it is observed that most of the values are between [-8, 8], and that after these values enter the LUT inside the branch metric calculation unit (Figure 3.2), the outputs of the LUT are real values mostly between [-8, 0].

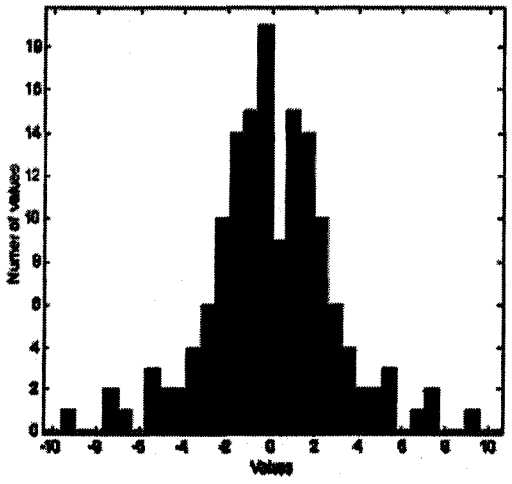


Figure 3.7 Histogram for the a Priori Probability values (LUT input)

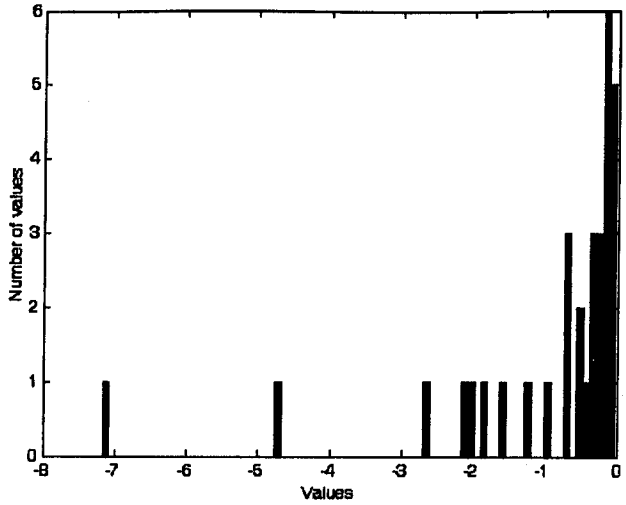


Figure 3.8 Logarithm of the AP(-1) or AP(+1) (LUT output)

Figure 3.8 shows that LUT output values are real values between [-8, 0]. The reason for quantization of the LUT output values into *integer* values is using the result of [worm], which states, "Theoretically, Turbo-decoding with the Max-Log-MAP decoder (when properly defined) is SNR independent." And "properly defined" means quantization of these values into integer values and not real or fixed point.

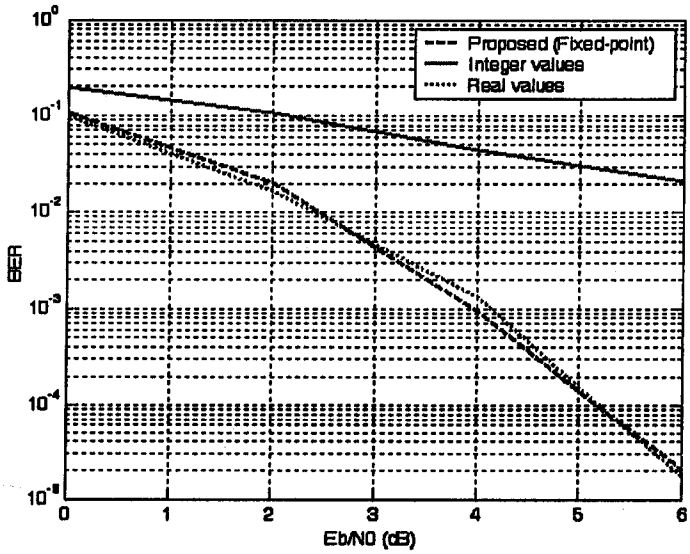


Figure 3.9 Comparison between proposed quantization, using integer values for quantization and the best possible performance (AWGN, 400 bits frame, no iteration, Max-Log-MAP decoder)

Figure 3.9 shows the comparison between the proposed quantization technique, using only integer values for quantization [wu1] and the best possible performance of the decoder in simulation using real values. As this Figure shows, using only integer values for quantization has a very small decrease in complexity, however it causes a high performance loss. This fact prevents us to use this method. Also, Figure 3.9 clearly shows that there is no need for increasing the number of bits, as it increases the complexity without significant improvement on performance.

3.7 Normalization

In forward/backward recursions of the MAP or Max-Log-MAP algorithm, metric values can easily overflow or underflow. When the most significant bits of two binary values, which are being added or subtracted, have the same values (both 1 or both 0) and the same bit of the result has a different value, overflow or underflow has happened. This means that the result will be an incorrect value and therefore, not reliable. This fact can be shown by an example:

-5 → 1011 (2's compliment)
-7 → 1001 (2's compliment)
---
1 0100

Normalizing the values can solve this problem. Several methods of metric normalization have been proposed in [bernard1]. Among these are:

- Reset
- Difference Metric ACS
- Variable shift
- Fixed Shift
- Modulo Normalization

Which are all normalization techniques for Viterbi algorithm.

Reset:

Redundancy is introduced into the channel input sequence in order to force the survivor sequences to merge after some number N of Add-Select-Compare (ASC) recursions for each state.

Difference Metric ACS:

The Viterbi algorithm is reformulated to keep track only of differences between metrics for each pair of states.

Variable shift:

After some fixed number N of recursions, the minimum survivor metric is subtracted from all of the survivor metrics.

Fixed Shift:

When all survivor metrics become negative (or all positive), the survivor metrics are shifted up (or down) by a fixed amount.

Modulo Normalization

Use two's complement representation of branch and survivor metrics and modular arithmetic during ACS operations.

The method used in [Wu1], also adds the complexity without improving the design more than other methods. The most popular methods are subtraction of the node metrics from the max or minimum [bernard1] node metrics at each time.

$$[\ln \alpha_t(m)]_N = \ln \alpha_t(m) - \min(\ln \alpha_t)$$
$$[\ln \alpha_t(m)]_N = \ln \alpha_t(m) - \max(\ln \alpha_t)$$

We have tested different methods of normalization for MAP decoder. The method that gives the best performance is to subtract all of the node metrics at each time, from the

maximum node metric at that specific time. Also values more/less than 128/–128 should be assigned to 128/–128 for 8bits 2's complement integer values. Figure 3.10 shows the comparison between two normalization methods. This Figure shows that by subtracting the max node metric from each node metric at every specific time, the performance in MAP decoders will be improved. In addition to its performance, this method is low complex and suitable for implementation.

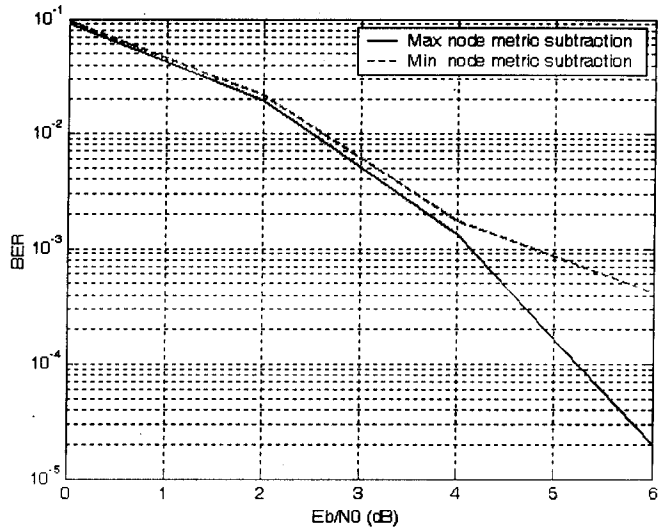


Figure 3.10 Comparison between two normalization methods, block length 400

Chapter 4

RTL Simulation and Synthesis

After system-level modeling and hardware architecture exploration, the design is ready for the next step, which is Register Transfer Level (RTL). Therefore, the RTL code is written in *Verilog HDL*. Code is tested at each step of programming by test benches and simulated to find any possible errors. Test benches examine the functionality of the design and Test vectors located in test benches check the accuracy of the program. Program is also tested, by monitoring the waveforms status of the inputs, outputs and intermediate parts of the design in *Simvision*. After finishing the RTL codes successfully, the design is also synthesized in *Synopsys Design analyzer*. More details of the steps mentioned above are explained in this chapter.

4.1 Verilog History

The Verilog Hardware Description Language, usually just called Verilog, was designed and first implemented by Phil Moorby at Gateway Design Automation in 1984 and 1985. It was first used beginning in 1985 and was extended substantially through 1987.

The implementation was the Verilog simulator sold by Gateway. The first major extension was Verilog-XL, which added a few features and implemented the infamous "XL algorithm", which was a very efficient method for doing gate-level simulation. This occurred in 1986, and marked the beginning of Verilog's growth period. Many leading-edge electronic designers began using Verilog at this time because it was fast at gate level simulation, and had the capabilities to model at higher levels of abstraction. These users began to do full system simulation of their designs, where the actual logic being designed was represented by a netlist and other parts of the system were modelled behaviorally.

In 1988, Synopsys delivered the first logic synthesizer, which used Verilog as an input language. This was a major event, as now the top-down design methodology could actually be used effectively. The design could be done at the "register transfer level", and then Synopsys' Design Compiler could translate that into gates. With this event, the use of Verilog increased dramatically.

Beginning in 1989, the second major trend began to emerge, and that was the use of Verilog-XL for sign-off certification by ASIC vendors. As Verilog became popular with the semiconductor vendors customers, they began to move away from their own, proprietary simulators, and started allowing customers to simulate using Verilog-XL for timing certification. As more ASIC vendors certified Verilog-XL, they requested more features, especially related to timing checks, back annotation, and delay specification. In response, Gateway implemented many new features in the language and the simulator to accommodate this need.

Cadence Design Systems acquired Gateway in December 1989, and continued to market Verilog as both a language and a simulator. At the same time, Synopsys was marketing the top-down design methodology, using Verilog. This was a powerful combination.

All this time, Verilog was a proprietary language. No other vendors were allowed to make a Verilog simulator. In response, the other vendors put their weight behind the VHDL

standardization process, which had been started by the DoD in the early 1980's. They needed a comparable product to sell, and VHDL was the only viable alternative.

By 1990, Cadence recognized that if Verilog remained a closed language, the pressures of standardization would eventually cause the industry to shift to VHDL. Consequently, Cadence organized Open Verilog International (OVI), and in 1991 gave it the documentation for the Verilog Hardware Description Language. This was the event, which "opened" the language. Subsequently, OVI did a considerable amount of work to improve the Language Reference Manual (LRM), clarifying things and making the language specification as vendor-independent as possible.

In 1994, the IEEE 1364 working group was formed to turn the OVI LRM into an IEEE standard. This effort was concluded with a successful ballot in 1995, and Verilog became an IEEE standard in December 1995.

When Cadence gave OVI the LRM, several companies began working on Verilog simulators. In 1992, the first of these were announced, and by 1993 there were several Verilog simulators available from companies other than Cadence. The most successful of these was VCS, the Verilog Compiled Simulator, from Chronologic Simulation. This was a true compiler as opposed to an interpreter, which is what Verilog-XL was. As a result, compile time was substantial, but simulation execution speed was much faster.

Now, Verilog simulators are available for most computers at a variety of prices, and which have a variety of performance characteristics and features. Verilog is more heavily used than ever, and it is growing faster than any other hardware description language. It has truly become the standard hardware description language.

4.2 RTL Coding

The program includes 10 modules. Figure 4.1 shows these modules where "topMAPchp2" is the top-level module and "write_a" and "write_b" are RAM macro

blocks. "topMAPchp2" is an IO wrapper around the chip. The hierarchical levels of the modules are as below:

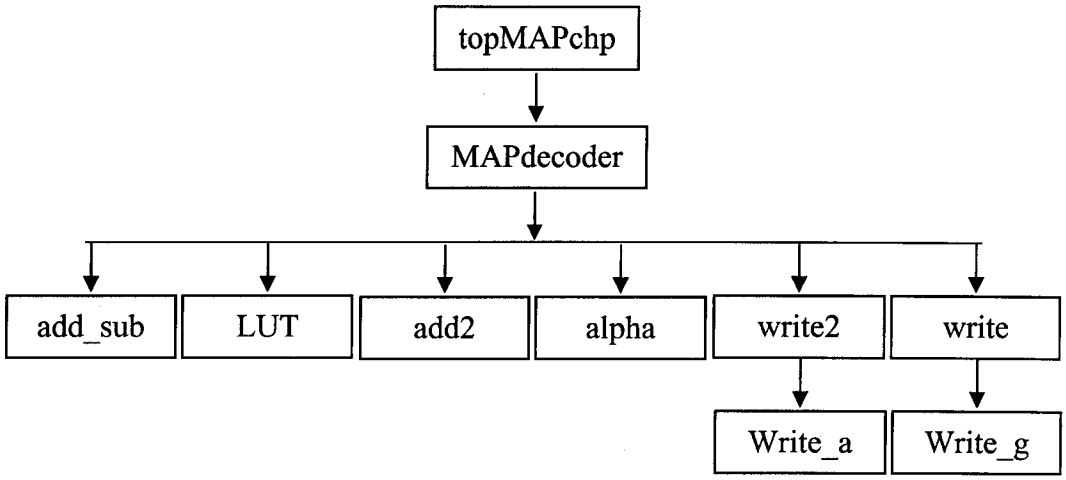


Figure 4.1 hierarchical levels of the modules written in Verilog

Inputs and outputs of the chip are:

Inputs:

- Yp: Parity data.
- Yd: Systematic data.
- App: The a priori probability.
- CLK: System clock.
- Start: Allows the chip to take the data values from the input data bus.
- Reset: Resets the chip.
- Block: Inform the chip about reaching to the end of each block or frame.

Output:

- l: Log-likelihood ratio (LLR)

Modules are briefly explained here. For better understanding of the function of each module look at Figure 3.1 and Figure 3.2.

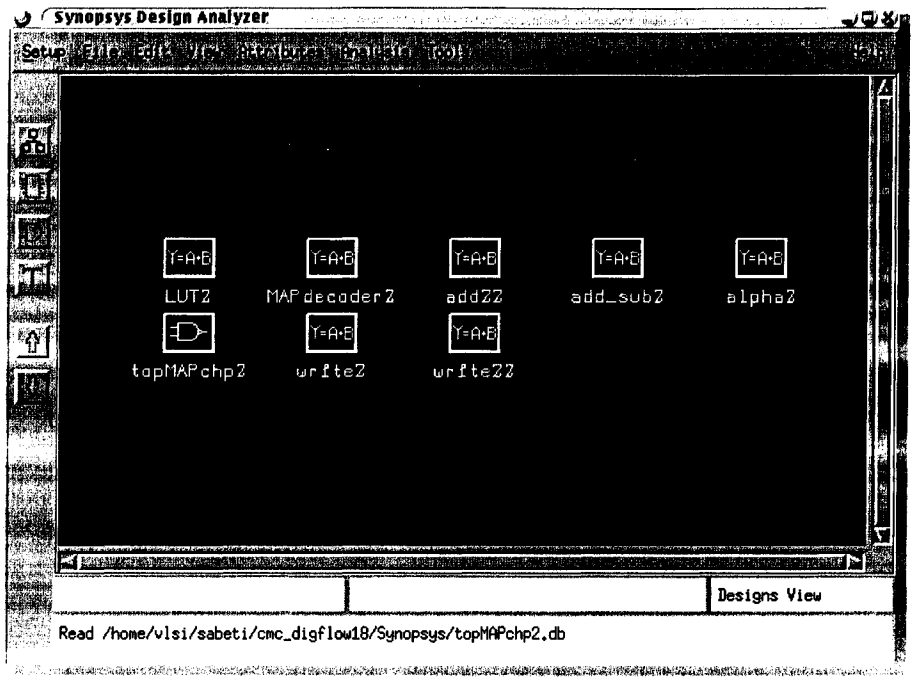


Figure 4.2 Modules in Synopsys, Design analyzer

topMAPchp2: The top module, which is an IO wrapper.

Inputs:

- Yp: Parity data.
- Yd: Systematic data.
- app: The a priori probability.
- CLK: System clock.
- start: Allows the chip to take the data values from the input data bus.
- Reset: Resets the chip.
- block: Inform the chip about reaching to the end of each block or frame.

Output:

- l: Log-likelihood ratio (LLR)

Wires:

- start_top,block_top,Reset_top,CLK_top,Yd_top,Yp_top,app_top,l_top

example of wrapper:

PDIDGZ xYd00 (.C(Yd_top[0]), .PAD(Yd[0])); //for input pin

PDO08CDG xl00 (.PAD(l[0]), .I(l_top[0])); //for output pin

MAPdecoder: Top module of the logical gates.

Inputs:

- Yp, Yd, app, CLK, start, Reset, block

Output:

- l

add_sub: Adds and subtract systemic and parity data inputs.

Inputs:

- busy, start, Yd, Yp;
  busy=0 in forward recursion and busy=1 in backward recursion.

Outputs:

- add1, add2, add3, add4;

Figure 4.3 shows the flowchart of this module.

LUT: Is the look up table for the a priori probability input.

Inputs:

- busy, start, app;

Outputs:

- pr1, prm1
  pr1: Pr(1)
  prm1: Pr(-1)

add2: Adds outputs from "add_sub" and "LUT" modules in an specific order.

Inputs:

- Reset, block, busy, CLK, n, add1, add2, add3, add4, pr1, prm1;

Outputs:

- g1, g2 ,g3, g4; // Four different values of branch metrics at each time

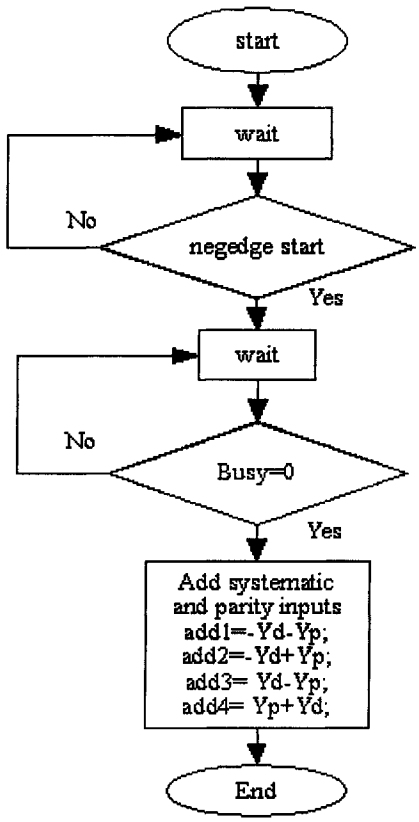


Figure 4.3 *add_sub* module flowchart

alpha: Reieves branch metrics, computes Alpha values and normalizes them in forward recursion.

Inputs:

- Reset, busy, CLK, nb, g1, g2, g3, g4;
- n //when start pulse comes n will be equal to 0.

Outputs:

- a1, a2, a3, a4; //Four different Alpha values
- max_a; //Max Alpha value at each time

write: Writes branch metric values in RAM1, computes Beta values and performs Beta normlization.

Inputs:

- start, block, Reset, CLK, max_a, g1, g2, g3, g4

- gg1, gg2, gg3, gg4,
  Branch metrics read by Beta calculation unit from RAM1

Outputs:

- b1, b2, b3, b4, ADR, D, WE, ME, OE, n, nb, busy
- lastADR
  lastADR, keeps address of the last Alpha or branch metric value stored in RAM

write2: Writes Alpha values in RAM2 and computes Log-likelihood ratio (LLR) and normalize it.

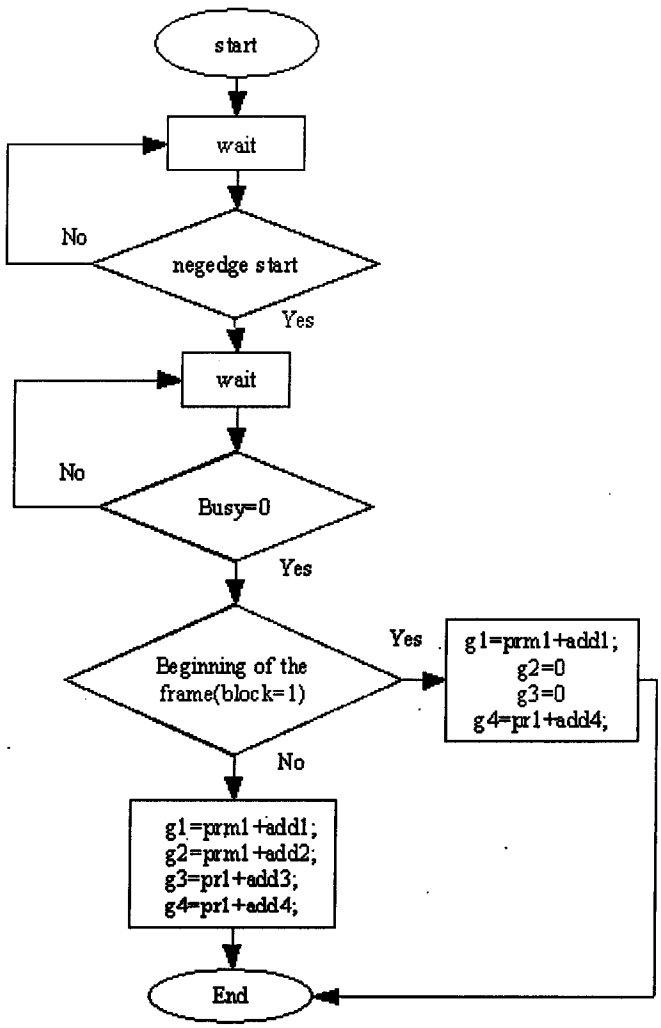


Figure 4.4 *add* module

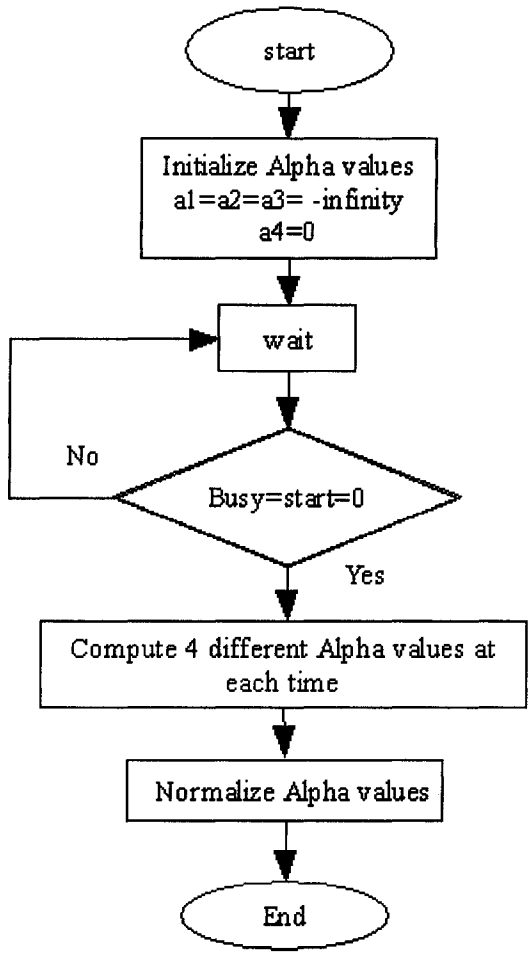


Figure 4.5 *Alpha* module

Inputs:

- n, busy, block, Reset, CLK, a1, a2, a3, a4
- aa1, aa2, aa3, aa4

  Alpha values read by soft output information unit from RAM2
- gg1, gg2, gg3, gg4,

  Branch metrics read by soft output information unit from RAM1
- b1, b2, b3, b4

  Beta calculation unit outputs

Outputs:

- ADR2, D2, WEr1, ME2, OE2, nl, max1, max2

Figure 4.8 and Figure 4.9 shows the algorithm for this module.

write_a: RAM macro block for storage of Alpha values in forward recursion.

write_g: RAM macro block for storage of branch metric values.

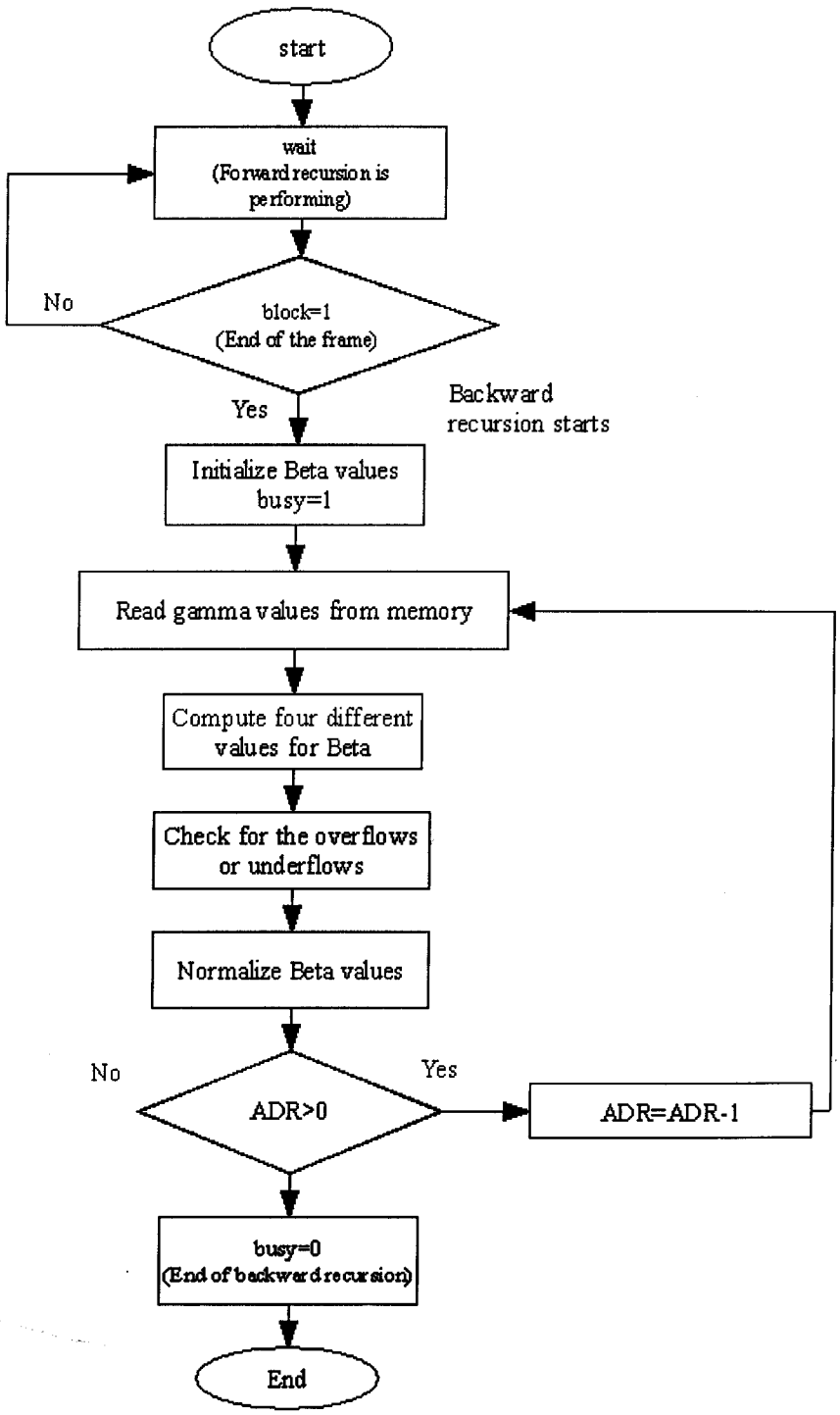


Figure 4.6 *write* module (computation of Beta values)

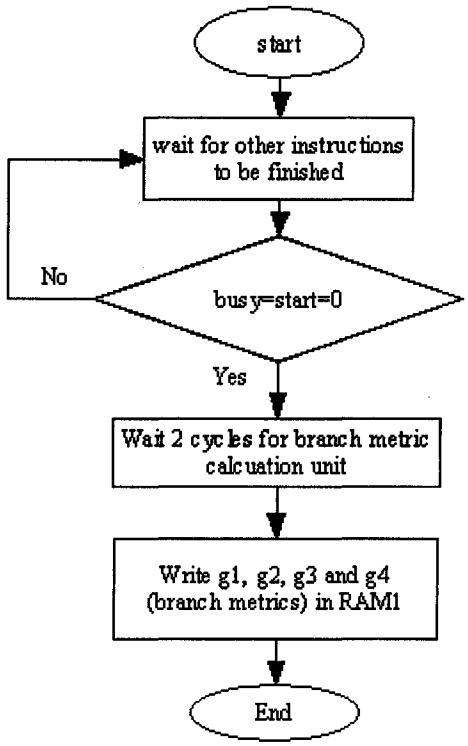


Figure 4.7 *write* module (writing branch metric values into RAM1)

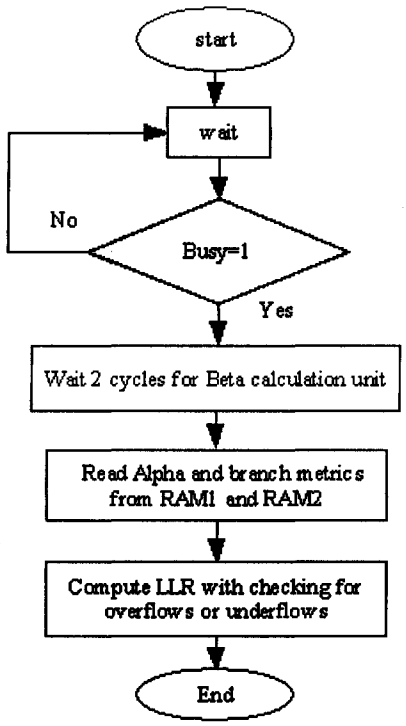


Figure 4.8 *writ2a* module (LLR computation)

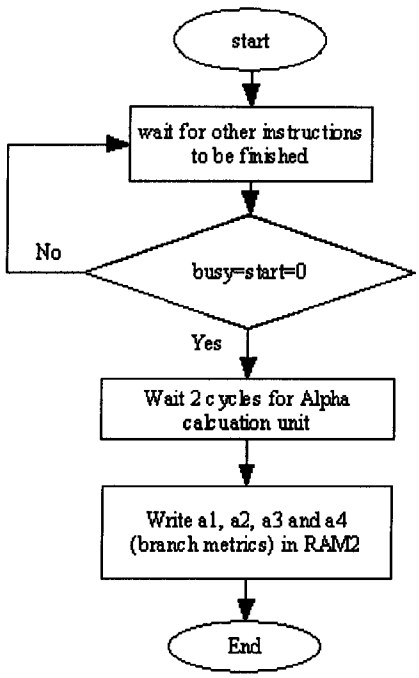


Figure 4.9 *writ2e* module (writing Alpha values into RAM2)

Figure 4.10 also shows an example of the results and status of the waves in *Simvision.* Monitoring the values in this software helps us to verify the accuracy of the program and check its functionality. It is possible to observe the inputs and outputs of each module or unit during integration of the program.

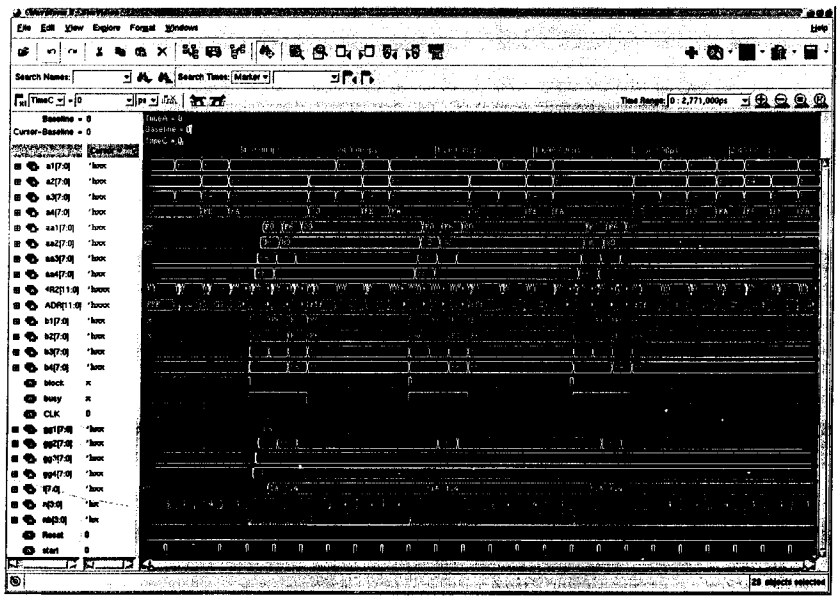


Figure 4.10 Wave forms status in *Simvision*

Next Figure shows a closer view of the waveforms and values of some input/outputs in a specific time.

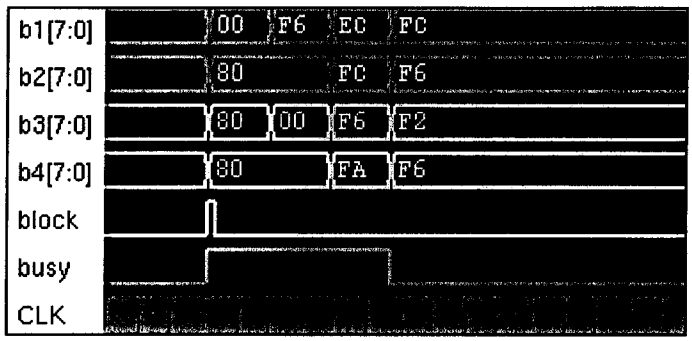


Figure 4.11 Waveform status in *Simvision*

4.3 Synthesis

Synthesis tools change the RTL level of the design into gate level. This job is performed here with *Synopsys Design analyzer*. The steps required for synthesis of a design in *Synopsys Design analyzer* is showed here.

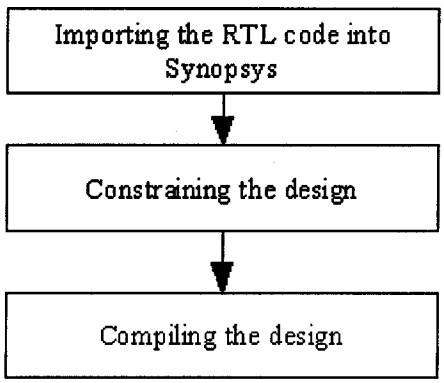


Figure 4.12 Synthesis steps

Fig 4.13 shows the synthesized level of the top module. In this Figure, I/O wrapper can be observed around the main module or *MAPdecoder*.

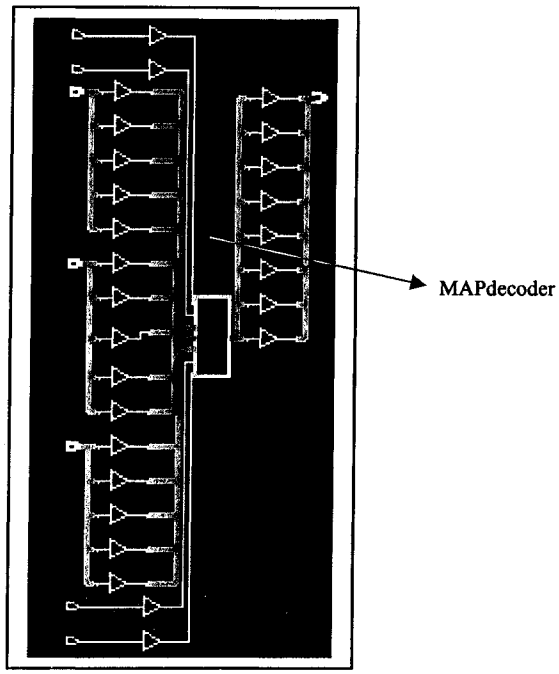


Figure 4.13 shows the synthesis level of the main module or *MAPdecoder.*

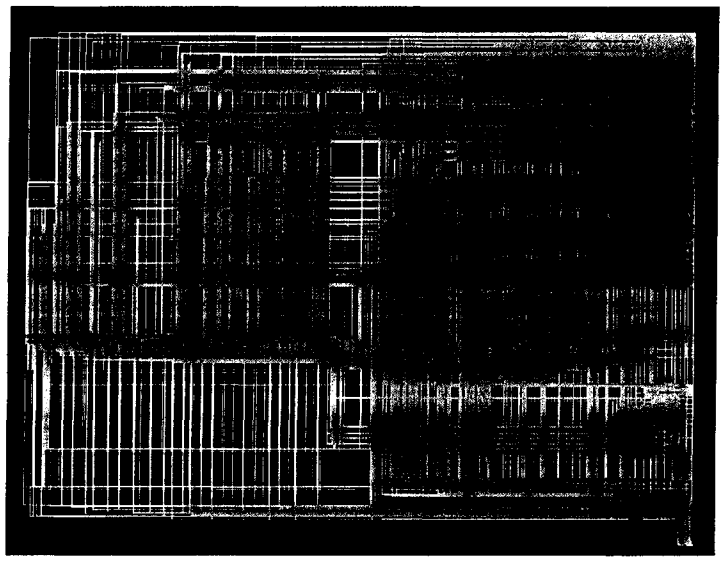


Figure 4.14 Inside the main module after synthesis level

During synthesis steps, different reports should be generated by designer; including area, constraint, timing, power, etc. These reports show that the proposed design has a total cell area of 0.94 mm^2 and consumes total dynamic power of 48.4 mW. Design has already 27

ports that will increase in later steps by adding other ports such as power rings or power cores. Each RAM block used in this design has a total area of 263,685 square microns. The clock period is 7ns, which make the design frequency to be about 143 MHz.

Chapter 5

VLSI Implementation

After system-level modeling and hardware architecture exploration, the design is ready for physical implementation. Most of these steps are done in Cadence First Encounter. Figure 5.1 shows briefly the required steps for digital design, which starts from system design and ends in physical verifications.

System level design, Register Transfer Level (RTL) coding including simulation and synthesis are already explained in previous chapter. Another simulation is also done after synthesis level to ensure the functionality of the design in gate level.
The next steps are physical implementations and include:

- Floorplanning and power planning
- Placement
- Clock Tree Generation
- Routing and timing verification
- Adding filler cells
- Physical verifications
- Adding pins and metal fills

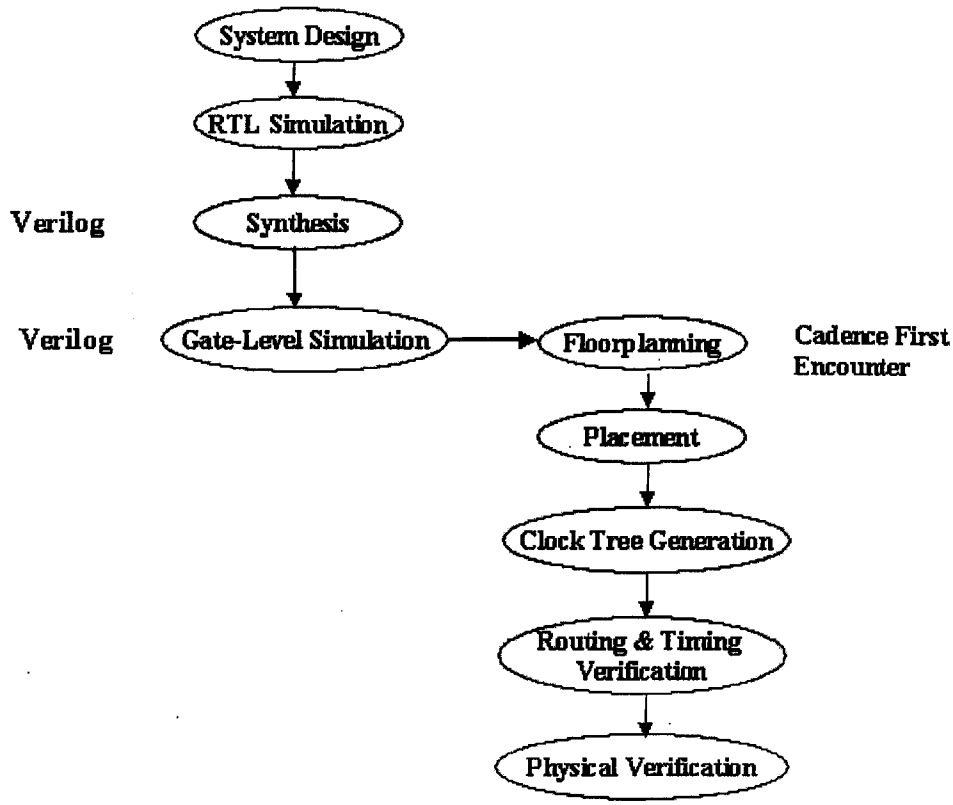


Figure 5.2 Digital design flow

These terms are explained in more details in this section, however it is good to briefly talk about hardware and software trade-offs for advanced 3G channel coding [michel], first.

5.1 Hardware and Software Trade-offs

One of the main drivers for the development of next generation wireless communications systems is the increasing demand for high-rate data services. Today's 2G systems, e.g. GSM, were targeted to voice as primary service. The supported data-services are limited to low rates. Therefore, the computational complexity for the baseband signal processing in the receiver, comprising equalizing, channel decoding and source decoding is relatively low and can be implemented using one state-of-the-art DSP running at 80 to 150MHz [nicol].

The 3G cellular wireless standards comprise advanced equalizing and coding algorithms. Especially Turbo-Codes [nicol, robertson] as channel coding scheme pose great demands on the computing power of communication devices and infrastructure. Along with the higher throughput requirements (384 kbit/s, 2 Mbit/s in UMTS) the resulting computational complexity has raised by at least an order of magnitude compared to 2G. Dedicated hardware solutions for the baseband signal processing fulfill these requirements, but often lack the flexibility to support the various existing or even emerging future communication standards. Therefore efficient implementations on programmable architectures are of great interest.

The actual requirements for the DSPs differ for the various fields of application: For hand-held devices, e.g., meeting the signal processing demands of one data channel while minimizing cost and power consumption is critical. In wireless infrastructure, several data channels have to be processed simultaneously, which exceeds the capabilities of a single signal processor even if a high-end device is used. Hence, the baseband receiver can be implemented either by a set of high-performance DSPs or by a combination of DSPs with dedicated IP cores.

It is argued in [bick] that "the trends in decoding algorithms are moving from standard Viterbi towards more computationally-expensive algorithms like softoutput Viterbi algorithm (SOVA) and maximum a posteriori (MAP) algorithm. The implementation efficiency of these algorithms will become a differentiating factor for next generation wireless communications – particularly for those employing programmable DSP devices." Thus we focus on channel decoding, which is besides equalization the computationally most demanding part within the baseband receiver. The computational complexity of these algorithms is very high, e.g. up to about 6000 MOPS for a Log-MAP decoder for 3G, assuming a data-rate of 2 Mbit/s. Therefore it is necessary to identify the primary bottlenecks in pure software implementations.
Depending on the targeted system environment these bottlenecks are solved by using advanced VLIW DSPs, *application-customized RISC cores* or custom IP blocks. Current state-of-the-art DSPs, from low-cost to high performance, supports already the kernel

operations of the Viterbi algorithm, used in 2G channel decoding, with dedicated instructions. For the MAP algorithm this support is actually lacking. The new approach of customizing and extending a RISC core blurs the borders between hardware and software. In contrast to a heterogeneous RISC/DSP or RISC/IP-block architecture the application specific hardware is coupled to the RISC core by an extension of its base instruction set. Thus application specific performance bottlenecks can be removed.

5.1.1 16-bit Fixed-point DSPs

The core architecture of a classical 16-bit fixed point DSP usually comprises one ALU/MAC unit, a program memory and two data memories, each with separate address and data busses. ALU unit and address generation unit (AGU) operate independently from each other, thus allowing parallel execution of instructions to a limited extent, e.g. ALU operations and memory-register transfers. A member of this architectural class of processors is, e.g., Motorola's DSP 56603. The DSP56603 is a low-cost low-power DSP, which is specially optimized for mobile wireless applications and provides a processing performance of 80 MIPS.

5.1.2 Modern VLIW DSPs

Modern DSP architectures attempt to increase the signal processing performance by exploiting the inherent parallelism of many signal-processing algorithms. This class of DSP architectures provides several independent ALU units along with wide and fast busses to the internal memories. To allow this increased degree of instruction level parallelism, instructions to be executed in parallel are grouped together to so-called very large instruction words (VLIW). Further, the processing units usually support the single instruction/multiple-data approach (SIMD). An example is sub-word parallelism, where several sub-words of a data word are processed with the same operation. A (Max-) Log-MAP implementation for decoding an 8-state Turbo-Code on this class of DSPs should exploit the benefits of the subword parallelism by using 16-bit packed data types. A state-of-the-art architecture is the ST120 from STMicroelectronics. It features two ALU units

and supports three different instruction sets: A 16-bit instruction set for compact microcontroller code, a 32-bit instruction set for higher performance and more complex instructions, and a third one for an increased level of instruction parallelism. In this 4x32-bit Score-boarded Long Instruction Word mode the processor is able to execute four 32-bit instructions in one clock cycle. Following the SIMD approach, the processor supports 2x16-bit data packed into one 32-bit data word.

5.1.3 Application-Customized RISC Cores

Significant attention has recently been drawn towards configurable processor cores such as those offered by ARC [arc] and Tensilica [tensilica]. These are based on classical RISC architectures that can be conFigured in two dimensions: First with respect to the architectural features of the core, e.g. inclusion of fast MAC units, cache sizes and policies, memory size and bus-width. Secondly the instruction set can be extended by user-defined instructions. This approach offers the benefit of removing application specific performance bottlenecks while still maintaining the flexibility of a software implementation, thus blurring the borders between hardware and software. For instance, performance-critical code portions that require multiple instructions on a generic RISC architecture can be compressed into a single, user defined instruction to obtain a significant speed-up. More importantly, this can on system-level eliminate the need for a heterogeneous RISC/DSP or RISC/IP-block architecture, therefore simplifying the architecture, reducing the cost of the system and simplifying the validation of the total system.

Key to efficiently using a configurable processor core is the methodology behind defining and implementing the custom instructions.

ARC's approach uses two independent descriptions that have to be provided to the tools: A C-model that describes the behavior of the new instruction to the instruction set simulator. And second, a synthesizable VHDL model that extends the VHDL description of the processor core. However, it is difficult to validate the consistency of the two

models, therefore creating a potential validation hole. The approach followed by Tensilica tries to tackle this validation problem by a single source model that is used for both extending the software tool chain with the custom instructions as well as for generating the synthesizable HDL of the extended processor. The implementation of the new instructions is done using a special extension language, called *TIE*, which is essentially a hybrid of Verilog and C.

5.2 Floorplanning and power planning

Floorplanning is the process of identifying structures that should be placed close together, and allocating space for them in such a manner as to meet the sometimes conflicting goals of available space (cost of the chip), required performance, and the desire to have everything close to everything else. The effort required for floorplanning depends on the prototyping level of the design. For example, floorplanning is very important if the design is being prepared for timing closure and detailed routing. Floorplanning, in conjunction with placement and trial routing, can be an iterative design process.Floorplanning usually starts by preplacing blocks, modules, and submodules according to the prepared floorplan and pad or pin locations in the chip I/O. All other modules or blocks not in the prepared floorplan are left outside the chip area.

Figure 5.3 shows the design in Cadence First Encounter environment after floorplanning. RAM macro blocks are placed manually and other modules are outside of the chip area to be placed later by software. The pad locations are allocated around the chip core. Also some space is considered between core and I/O for power planning.

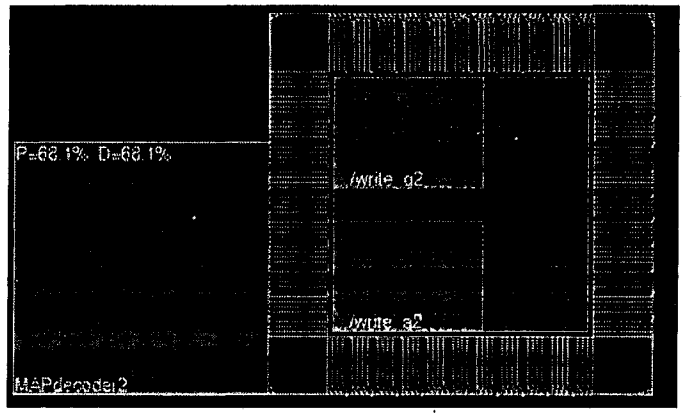


Figure 5.3 Floorplanned design

In power planning, power rings and power stripes are added to the design to connect blocks and cells to the power structures. Power rings includes rings around the chip core and all of the macro blocks. Power rings and power stripes are shown in the Figure.

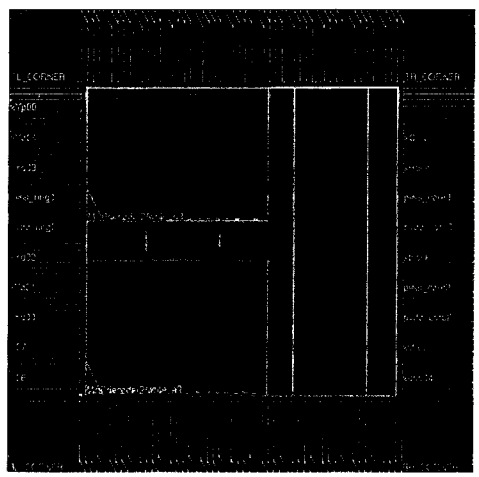


Figure 5.4 Power planned design

5.2.1 Ring power pads

Ring power pads are used to provide power to the I/O cells, and drive signals off-chip. The number of ring power pads are decided based on the rule-of thumb of one pair for every 4-6 output pins. Therefore, two pairs of ring power pads are considered here.

5.2.2 Core power pads

Core power pads provide power to the core of the chip. Core and ring power are often separate to reduce the off-chip noise associated with high-current drive from affecting the core power. For core power, the rule of thumb is 1mA per micron of metal width. Power connections are assumed 60-micron and 60mA is considered for each power connection. It should provide a margin of almost 8x the estimated maximum core requirement, which is 59.66mW. Therefore, 4 pairs of power pads are considered:

4 x 60mA=240mA (4 pairs of power pads provide 240mA)
240mA x 1.8v=432mW
Maximum core power requirement=59.66mW
432/59.66=7.24x (almost 8x more than Maximum core power requirement

As a result, the total number of I/O pins after adding power pads is 39.

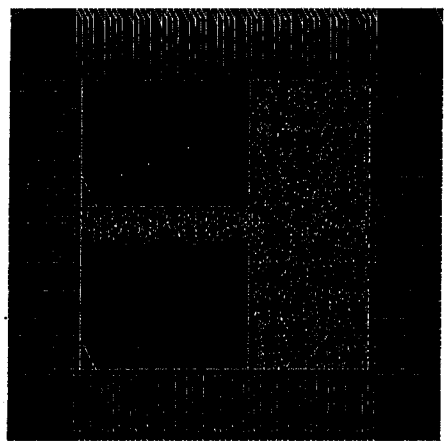


Figure 5.5 Placed design

5.3 Placement

Placement includes placing standard cells and blocks inside the chip core to create a placement that is routable and meets the performance constraints. Timing-driven placement algorithms are iterative algorithms that consider critical path to find the best placement that satisfies both timing and area requirements.

5.4 Clock Tree Generation

Clock Tree Synthesis (CTS) is used to build a buffer tree and balance insertion delays from the clock source to all flip-flops.

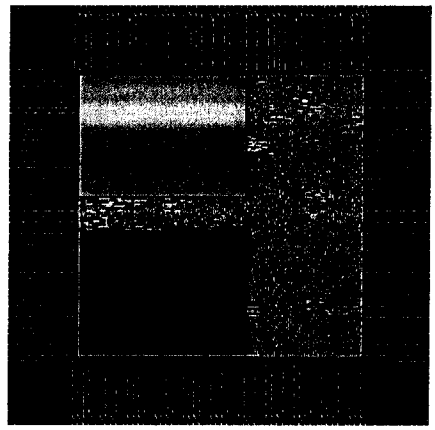


Figure 5.6 Chip after Clock Tree Generation

5.5 Routing

Routing connects standard cells to each other, to blocks and I/O pads considering minimum congestion. Also, routing includes power routing, which routes power and ground to all parts of the chip.

5.5.1 Sroute

SRoute creates pad rings and routes power and ground nets to the following power structures:

- Block pins
- Pad pins
- Standard cell pins
- Unconnected stripes

Srouting in usually performs before other kinds of routing.

5.5.2 Trial routing

The Trial Routing follows the design rules described in the technology file to accurately gauge routing congestion and extract resistance and capacitance (RC) parasitic values. The program routes the metal interconnection to completely connect the design according to the imported netlist, and also incorporates any changes made during placement.

5.5.3 NanoRoute

NanoRoute performs concurrent timing and signal integrity–driven routing and physical optimization, and postroute repair of timing and noise problems in multimillion instance cell, block, or mixed cell and block level designs.

5.5.4 Wroute

Wroute in this software performs global and detailed routing of placed block-and-cell designs using an area-based algorithm.

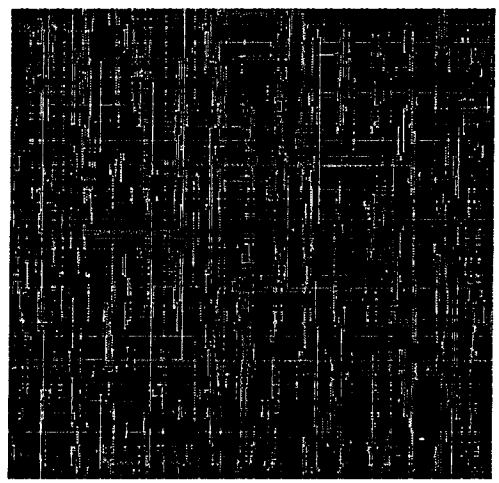


Figure 5.7 Routed design

5.6 Filler cells

The Add Filler program inserts filler cell instances between the gaps of standard cell instances. Filler cells provide decoupling capacitance or complete the power connections in the standard cell rows.

5.7 Metal fill

Metal fill is electrically inactive metal that fills open areas on each metal layer. This serves to minimize variations in the metal thickness, resulting in a more even distribution of the dielectric and improved chip performance. You can insert metal fill areas into a placed and routed design to achieve a metal density within the range required by a specific manufacturing process. Metal density is defined as follows:

$$\text{MetalDensity} = \frac{\text{TotalMetalArea}}{\text{WindowArea}}$$

There are also many implementation details during steps mentioned above and design must be verified to meet all the physical and timing requirements.

Chapter 6

Results and Conclusion

Channel decoding is one of the most computationally intensive and power-consuming tasks and is therefore normally implemented in a dedicated hardware block. A 2-Mb/s 3G turbo decoder with ten turbo iterations must carry out more than 1.2 billion add-compare-select (ACS) operations per second plus the additional overhead of data handling and control. The flexible nature of reconfigurable general purpose DSP devices means that for any specific task they will always consume more power, have less throughput, and be larger than a dedicated hardware solution. For this reason, the high computational load involved in 3G channel decoding makes a software solution on a general purpose DSP less attractive than a dedicated hardware. This fact has led some latest-generation DSPs to include dedicated hardware decoder cores on chip [agrawala]. Others have used additional hardware to implement specialized instruction sets for accelerating ACS computations [olofsson]; however, the flexible nature of the DSP architecture still inherently makes them less attractive than dedicated channel decoder hardware solutions with respect to throughput, power consumption, and area.

6.1 Results and comparison

Other turbo decoders proposed in literature are implemented in 0.35 μm [vogt1, stralen], 0.25 μm [miyauchi] and 0.18 μm [bick1] technologies. There are also other designs such as [Wu, bai, zeng, youyun], which are implemented in FPGAs or [michel1] in DSP processors.

Table 6.1 shows the proposed turbo decoders in literature. Almost all of the designs are implemented in (RTL level) VHDL or Verilog HDL and the results are from the reports obtained from synthesis level. Between these implementations Bicherstaff et. al. have fabricated an ASIC [bicherstaff].

By comparing the results presented in table 6.1, and the proposed design, we conclude that our achieved results are superior. Our throughput will be about 6.5Mb/s in turbo decoders after ten iterations. This result is comparable with [michel1] that has used the same algorithm but less throughput or 0.54Mb/s after 5 iterations. The proposed chip in [michel1] is implemented in a 200MHz DSP processor. Also, we can compare our throughput result of 16.2Mb/s after 4 iterations with 1Mb/s that Vogt et. al have reached.

6.2 Summary of Contributions

With the application of turbo coding to more communication systems, low-complexity implementation of turbo decoder becomes a more popular and challenging topic. The aim of this thesis has been design and implementation of a low-complex MAP decoder with performance that meets the requirements for sound and data transfer. A summary of the contributions in the design and implementation of this decoder is provided in this section.

Table 6.1 Results and comparison

	Rate	Block size	Algorithm	Throughput Mb/s	Frequency MHz	Implementation μ m	Area mm^2
Vogt2	1/3	668	ML-MAP	1(4it)	16	0.35	2.8
Vogt2	1/3	668	SOVA-OU	1.9(4it)	16	0.35	2.5
Stralen	1/2	512	L-MAP	3.5	62.5	0.35	-
Miyauchi	2/3	8920	SL-MAP	-	100	0.25	-
Bicherstaff	1/3	2568	L-MAP	2.5	110.8	0.18	9
Y. Wu	1/2	640	ML-MAP	1	-	FPGA	-
Bai	1/3	3168	L-MAP	0.14(5it)	-	FPGA	-
Zeng	1/3	2000	L-MAP	2	-	FPGA	-
Youyun	1/3	5114	SL-MAP	2.5	30.72	FPGA	-
Michell	1/3	5114	ML-MAP	0.54(5it)	-	ST120	-
Michell	1/3	5114	L-MAP	0.3(5it)	-	Custom RISC	-
Proposed	**1/2**	**512**	**ML-MAP**	**13, 1.6(4it), 1.3(5it)**	**143**	**0.18**	**1.46**

6.2.1 Algorithmic Contribution

The original turbo decoder consists of SISO decoders based on MAP algorithm, which involves a large amount of multiplications, exponentials, and logarithm computations. Implementations of these mathematical operations are usually quite complex in VLSI design. Sub-optimal, but much simpler, varieties of MAP algorithms, Max-Log-MAP and Log-MAP, reduce the computational complexity. These sub-optimal SISO algorithms bring certain level of complexity reductions with some performance degradations.

In this thesis by using the fact that Max-Log-MAP algorithm is not sensitive to the AWGN variance of the noise [worm], we could reduce the complexity of the branch metric calculation and introduce a less complex equation. Using this fact results in omitting the multiplication and division exists in branch metric calculation equation.

6.2.2 Architectural Contribution

At the architectural level, through computation of soft output information unit in backward recursion, we could omit a large RAM block for storage of β values. This design offers two RAM blocks, one for the storing branch metric values and the other for α values. The stored values will be needed for the computation of log-likelihood ratios. Log-likelihood ratio is computed by using saved branch metrics and α values in RAMs and real time β values.

6.2.3 System Level Design Contribution

Because distorted information that decoder receives from the channel can be any real value, they are required to be quantized into a proper number of bits. Therefore, in this thesis the quantization of the decoder inputs and metric values is studied and the best quantization with the minimum possible number of bits and reliable Bit Error Rate (BER) is chosen based on the simulation results. Normalization techniques are also studied and a

suitable technique for the Max-Log-MAP decoder is explained. In addition, the decoder is flexible with respect to Block Size and accepts frames from 10 to 512 bits per block.

6.3 Conclusions

We have demonstrated the new implementation of a MAP decoder suitable for ASIC, which can be used in turbo decoders. Various turbo-coding algorithms were studied and Max-Log-MAP algorithm was selected for this design for being the best compromise between performance and complexity. The branch metric calculation unit is redesigned to reduce its complexity, which in turn reduces overall complexity of the decoder. The complexity reduction is due to exploiting the insensitivity of the MAX-Log-MAP algorithm to AWGN channel variance. The new decoder design only needs two RAM blocks compared to three RAM blocks of previous proposals. The design architecture is parallel and pipeline, which makes the design work fast. The quantization and normalization techniques that can best fit our MAP decoder design are studied and a new quantization is presented based on the simulation results. The decoder has achieved the throughput of 6.5Mb/s after 10 iterations and the core of the chip has an area of $1mm^2$ in a 0.18 μm six-layer metal CMOS technology.

References

[agarwala] S. Agarwala *et al.*, "A 600 MHz VLIW DSP," in *IEEE Int. Solid StateCircuits Conf. Dig. Tech. Papers*, vol. 45, Feb. 2002, pp. 56–57.

[anderson] Anderson, J.B.; Hladik, S.M.; "Tailbiting MAP decoders," IEEE Journal on Selected Areas in Communications, Volume: 16 Issue: 2, pp. 297 –302, Feb. 1998

[arc] ARC International Ltd. http://www.arccores.com

[atluri] Atluri, I.; Arslan, T.; "Low power VLSI implementation of the map decoder for turbo codes through forward recursive calculation of reverse state metrics," *IEEE International [Systems-on-Chip] SOC Conference 2003*, Proceedings, pp. 408- 411, 17-20 Sept. 2003

[bahl] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Transactions on Information Theory*, vol. 20, pp. 284–287, Mar. 1974.

[bai] Chunlong Bai; Jun Jiang; Ping Zhang; "Hardware implementation of Log-MAP turbo decoder for W-CDMA Node B with CRC-aided early stopping," *IEEE 55th Vehicular Technology Conference, 2002, VTC*, Vol. 2, pp. 1016 – 1019, 6-9 May 2002

[bernard] Shung C.B., Siegel P.H., Ungerboeck G., Thapar H.K.; "VLSI architectures for metric normalization in the Viterbi algorithm," *IEEE International Conference on Communications, 1990, ICC 90, SUPERCOMM/ICC '90*, vol. 4, pp. 1723 - 1728, 16-19 April 1990

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

[bick1] M. Bickerstaff et al. "DSP Systems for Next-Generation Mobile Wireless Infrastructure," In *Proc. ICASSP 2000*, pp. 3710–3713, 2000.

[bick2] Bickerstaff, M.; Garrett, D.; Prokop, T.; Thomas, C.; Widdup, B.; Gongyu Zhou; Nicol, C.; Ran-Hong Yan; "A unified turbo/viterbi channel decoder for 3GPP mobile wireless in 0.18 μm CMOS," *2002 IEEE International Solid-State Circuits Conference, 2002, Digest of Technical Papers, ISSCC*, Volume: 1, pp. 124 -451 vol.1, 3-7 Feb. 2002

[boutillon] Boutillon, E.; Gross, W.J.; Gulak, P.G.; "VLSI architectures for the MAP algorithm," IEEE Transactions on Communications, Volume: 51 Issue: 2, pp. 175–185, Feb. 2003

[chaikalis] Chaikalis, C.; Noras, J.M.; "Implementation of an improved reconfigurable SOVA/log-MAP turbo decoder in 3GPP," *Third International Conference on 3G Mobile Communication Technologies, 2002* (Conf. Publ. No. 489), pp. 146 – 150, 8-10 May 2002

[chass] A. Chass and A. Gubeskys, "On Performance/Complexity Analysis and SW Implementation of Turbo-Decoding," In *Proc. 2nd International Symposium on Turbo-Codes & RelatedTopics*, pp. 531–534, Brest, France, September 4–7, 2000.

[cheng] Jung-Fu Cheng; Ottosson, T.; "Linearly approximated log-MAP algorithms for turbo decoding," *Vehicular Technology Conference Proceedings, 2000, VTC 2000*, Spring Tokyo, Volume: 3, pp. 2252 -2256, 15-18 May 2000.

[dawid1] Dawid, H.; Gehnen, G.; Meyr, H.; "Map channel decoding: Algorithm and VLSI architecture," *VLSI Signal Processing, VI, 1993.*, [Workshop on] , pp. 141 –149, 20-22 Oct. 1993

[dawid2] Dawid, H.; Meyr, H.; "Scalable architectures for high speed channel decoding," *Workshop on VLSI Signal Processing, VII, 1994*, pp. 226 –235, 26-28 Oct. 1994

[dawid2] Dawid, H.; Meyr, H.; "Real-time algorithms and VLSI architectures for soft output MAP convolutional decoding," *Sixth IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, 1995, PIMRC'95,* 'Wireless: Merging onto the Information Superhighway', Volume: 1, pp. 193 –197, 27-29 Sept. 1995

[dimitrov] V. S. Dimitrov, G. A. Jullien, W.C. Miller, "Theory and applications of the double base number system", IEEE Transactions on computers, vol. 48 pp. 1098-1106, 1999

[el-Assal] El-Assal, M.; Bayoumi, M.; "A high speed architecture for MAP decoder," *IEEE Workshop on Signal Processing Systems, 2002 (SIPS '02)*, pp. 69 –74, 16-18 Oct. 2002,

[engling] Engling Yeo; Pakzad, P.; Nikolic, B.; Anantharam, V., "VLSI architectures for iterative decoders in magnetic recording channels," IEEE Transactions on Magnetics, Volume: 37 Issue: 2, pp. 748 –755, March 2001

[erfanian] Erfanian, J.A.; Pasupathy, S.; "Low-complexity parallel-structure symbol-by-symbol detection for ISI channels," *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, 1989, Conference Proceeding, pp. 350–353, 1-2 June 1989

[feldman] Feldman, J.; Abou-Faycal, I.; Frigo, M.; "A fast maximum-Liklihood Decoder for Convolutional Codes," *IEEE 56th Vehicular Technology Conference, 2002, Proceedings, VTC 2000*,Vol. 1, pp. 371 -375, 24-28 Sept. 2002

[forney] G. D. Forney, "The Viterbi algorithm," *Proceedings of the IEEE*, vol. 61, pp. 268–278, Mar. 1973.

[frenking] Frenking and Parhi, "Montgomery modular multiplication and exponentiation in the residue number system", *Asilomar Conference on Signals, Systems and Computers*, vol. 2, pp. 1312-1316, 1999

[gallager] Gallager, R.G.; "Claude E. Shannon: a retrospective on his life, work, and impact," *IEEE Transactions on Information Theory,* Volume: 47, Issue: 7, pp. 2681 – 2695, Nov. 2001

[gibong] Gibong Jeong; Dan Hsia; " Optimal quantization for soft-decision turbo decoder," *Vehicular Technology Conference, 1999, VTC 1999 - Fall. IEEE VTS 50th,* Volume: 3, pp. 1620 -1624, 19-22 Sept. 1999

[gross] Gross, W.J.; Gaudet, V.C.; Gulak, P.G.; "Difference metric soft-output detection: architecture and implementation," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, Volume: 48, Issue: 10, pp. 904 – 911, Oct. 2001

[hagenauer1] J. Hagenauer and P. Hoeher, "A Viterbi algorithm with soft-decision outputs and its applications," *Proceedings of the IEEE, GLOBECOM*, pp. 1680–1686, 1989.

[hagenauer2] Hagenauer, J.; Offer, E.; Papke, L.; "Iterative decoding of binary block and convolutional codes," IEEE Transactions on Information Theory, Volume: 42 Issue: 2, pp. 429 –445, March 1996

[halter] S. Halter, M. Oberg, P. M. Chau, and P. H. Siegel, "Reconfigurable Signal Processor for Channel Coding & Decoding in Low SNR Wireless Communications," In *Proc. 1998 Workshopon Signal Processing Systems (SiPS '98)*, pages 260–274, Cambridge, Massachusetts, USA, Oct. 1998.

[heller] Heller, J.; Jacobs, I.; "Viterbi Decoding for Satellite and Space Communication," *IEEE Transactions on Communications*, [legacy, pre - 1988], Volume: 19 Issue: 5, pp. 835 –848, Oct 1971

[hsu] Jah-Ming Hsu; Chin-Liang Wang; "On finite-precision implementation of a decoder for turbo codes," Proceedings of the 1999 IEEE International Symposium on Circuits and Systems, 1999 ISCAS '99, Volume: 4, pp. 423 –426, 30 May-2 June 1999

[hong] Hong, S.; Yi, J.; Stark, W.E.; "VLSI design and implementation of low-complexity adaptive turbo-code encoder and decoder for wireless mobile communication applications," *1998 IEEE Workshop on Signal Processing Systems, SIPS 98*, pp. 233–242, 8-10 Oct. 1998

[IEEE] IEEE Standard for binary floating-Point, IEEE Std. 754-1985

[kienle] Kienle, F.; Kreiselmaier, G.; Wehn, N.; "VLSI-implementation issues of turbo trellis-coded modulation," *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003, Proceedings, (ICASSP '03)*, Volume: 2, 6-10, pp. 633-636, April 2003

[kwon] Taek Won Kwon; Dae Won Kim; Woo Tae Kim; Eon Kyeong Joo; Jun Rim Choi; Pyung Choi; Jun Jin Kong; Sung Han Choi; Won Hee Chung; Ki Won Lee; "A modified two-step SOVA-based turbo decoder for low power and high performance," *TENCON 99. Proceedings of the IEEE Region 10 Conference*, Volume: 1, pp. 297-300, 15-17 Sept. 1999

[lee] Sung-Woo Lee; Won-sub Kim; Jin-Soo Park; "System Implementation in Turbo Code for Digital Video Transmission," *TENCON '02, Proceedings 2002 IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering*, Volume: 2, pp. 1043 -1047 vol.2, 28-31 Oct. 2002

[lopez] Lopez-Vallejo M.; Mujtaba, S.A.; Inkyu Lee; "A low-power architecture for maximum a posteriori decoding," *Conference on Signals, Systems and Computers, 2002.* Conference Record of the Thirty-Sixth Asilomar, Volume: 1, pp. 47 –51, Nov. 3-6, 2002

[luukkanen] Luukkanen, P.; Ping Zhang; "Comparison of optimum and sub-optimum turbo decoding schemes in 3rd generation cdma2000 mobile system," *Wireless Communications and Networking Conference, 1999, WCNC 1999 IEEE*, pp. 437 -441 vol.1, 21-24 Sept. 1999

[mansour] Mansour, M.M.; Shanbhag, N.R.; "VLSI architectures for SISO-APP decoders," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Volume: 11 Issue: 4, pp. 627–650, Aug. 2003

[markman] Markman, I.; Anderson, J.B.; "New estimates for the performance of soft-decision sequential decoders," *IEEE International Conference on Communications, 1994. ICC 94, SUPERCOMM/ICC '94*, Conference Record, Serving Humanity Through Communications, vol.3, pp. 1260-1264, 1-5 May 1994

[masera1] Masera, G.; Mazza, M.; Piccinini, G.; Viglione, F.; Zamboni, M.; "Architectural strategies for low-power VLSI turbo decoders," *IEEE Transactions on*

Very Large Scale Integration (VLSI) Systems, Volume: 10, Issue: 3, pp. 279 –285, June 2002

[masera2] Masera, G.; Piccinini, G.; Roch, M.R.; Zamboni, M.; "VLSI architectures for turbo codes", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Volume: 7 Issue: 3, pp. 369–379, Sept. 1999

[michel1] Michel, H.; Worm, A.; Munch, M.; Wehn, N.; "Hardware/Software Trade-offs for Advanced 3G Channel Coding," *Design, Automation and Test in Europe Conference and Exhibition*, pp. 396 – 401, 4-8 March 2002

[michel2] Michel, H.; Wehn, N.; "Turbo-decoder quantization for UMTS," *Communications Letters, IEEE*, Volume: 5 Issue: 2, pp. 55 –57, Feb 2001.

[miki] Miki, M.H.; Taki, D.; Fujita, G.; Onoye, T.; Shirakawa, I.; Fujiwara, T.; Kasami, T.; "Recursive maximum likelihood decoder for high-speed satellite communication," *Proceedings of the 1999 IEEE International Symposium on Circuits and Systems, 1999. ISCAS '99*, Volume: 4, pp. 572 –575, 30 May-2 June 1999

[miyauchi] Miyauchi, T.; Yamamoto, K.; Yokokawa, T.; Kan, M.; Mizutani, Y.; Hattori, M.; "High-performance programmable SISO decoder VLSI implementation for decoding turbo codes", *IEEE Global Telecommunications Conference, GLOBECOM '01*. Volume: 1, pp. 305 -309, 25-29 Nov. 2001

[montorsi] Montorsi, G.; Benedetto, S.; "Design of fixed-point iterative decoders for concatenated codes with interleavers," *Global Telecommunications Conference, 2000, GLOBECOM '00, IEEE*, Vol. 2, pp. 801 -806, 27 Nov.-1 Dec. 2000

[nicol] I. Verbauwhede and C. Nicol. "Low-Power DSP's for Wireless Communications," In *Proc. ISLPED 2000*, pp. 303–310, Rapallo, Italy, 2000.

[nikolic] J. Nikolic-Popovic. "Implementing a MAP Decoder for cdma2000 Turbo Codes on a TMS320C62x DSP Device," Technical report, Texas Instruments Incorporated, Houston, Texas, USA, May 2000. Application Report SPRA629

[olofsson] A. Olofsson and F. Lange, "A 4.32GOPS 1 W general purpose DSP with an enhanced instruction set for wireless communication," in *IEEE Int. Solid State Circuits Conf. Dig. Tech. Papers*, vol. 45, Feb. 2002, pp. 54–55.

[papke] L. Papke, P. Robertson, and E. Villebrun, "Improved decoding with the SOVA in a parallel concatenated (turbo-code) scheme," *Proceedings, IEEE International Conferenceon Communications*, pp. 102–106, 1996.

[parhami] Behrooz Parhami, "Computer Arithmetic: Algorithms and Hardware Designs", Oxford University Press, New York, 2000.

[park] Goohyun Park; Sukhyon Yoon; Changeon Kang; Daesik Hong; "An implementation method of a turbo-code decoder using a block-wise MAP algorithm," *Vehicular Technology Conference, 2000. IEEE VTS-Fall VTC 2000*, 52nd, Volume: 6, pp. 2956 -2961, 24-28 Sept. 2000

[pietrobon] S. S. Pietrobon, "Implementation and performance of a turbo/MAP decoder," *International Journal of Satellite Communications*, vol. 16, pp. 23–46, 1998

[pillai] R. V. K. Pillai, D. Al-Khalili and A. J. Al-Khalili, "Energy delay analysis of partial product reduction methods for parallel multiplier implementation", *International Symposium on Low Power Electronics and Design*, pp. 201-204, 1996

[robertson1] P. Robertson, P. Hoeher and E. Villebrun, "Optimal and Sub-Optimal Maximum a Posteriori Algorithms Suitable for Turbo Decoding," *ETT*, pp. 119–125, Mar.–Apr. 1997

[robertson2] Robertson, P.; Villebrun, E.; Hoeher, P.; "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," *1995 IEEE International Conference on Communications, ICC 95 Seattle, Gateway to Globalization*, Volume: 2, pp. 1009 -1013 vol.2, 18-22 June 1995

[robertson3] Robertson, P.; "Illuminating the structure of code and decoder of parallel concatenated recursive systematic (turbo) codes," *Global Telecommunications Conference, 1994, GLOBECOM '94*, 'Communications: The Global Bridge'. IEEE, Volume: 3, pp. 1298-1303, 28 Nov.-2 Dec. 1994

[rosa] La Rosa, A.; Passerone, C.; Gregoretti, F.; Lavagno, L.; "Implementation of a UMTS turbo-decoder on a dynamically reconfigurable platform," *Design, Automation and Test in Europe Conference and Exhibition, 2004,* Proceedings, Volume: 2, pp. 1218-1223, 16-20 Feb. 2004

[schurgers] Schurgers, C.; Catthoor, F.; Engels, M.; "Memory optimization of MAP turbo decoder algorithms," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Volume: 9, Issue: 2, pp. 305 – 312, April 2001

[sklar] Sklar, B.; "A primer on turbo code concepts," *Communications Magazine, IEEE*, Volume: 35 Issue: 12, pp. 94 –102, Dec. 1997

[stralen] N. Van Stralen S. Haladik, "Design of Turbo Decoder ASIC," *2nd International Symposium on Turbo Codes & Related Topices*, Brest, France, 2000

[tajima] Tajima, M.; Shibata, K.; Kawasaki, Z.; "On the equivalence between scarce-state-transition viterbi decoding and syndrome decoding of convolutional codes," IEEE *International Symposium on Information Theory*, 2003 Proceedings, pp. 304 –304, June 29 - July 4, 2003

[tensilica] Tensilica Inc. http://www.tensilica.com.

[tepe] K. E. Tepe; "Iterative Decoding Techniques for Correlated Raleigh Fading and Diversity Channels," Report TR 2001-1, February 2001

[thul] Thul, M.J.; Gilbert, F.; Vogt, T.; Kreiselmaier, G.; Wehn, N.; "A scalable system architecture for high-throughput turbo-decoders," *IEEE Workshop on Signal Processing Systems, (SIPS '02)*, pp. 152 – 158, 16-18 Oct. 2002

[tong] Tong, Y.; Yeap, T.-H.; Chouinard, J.-Y.; "VHDL Implementation of a Turbo Decoder With Log-MAP-Based Iterative Decoding," *IEEE Transactions on Instrumentation and Measurement*, Volume: 53 , Issue: 4 , pp. 1268 – 1278, Aug. 2004

[viglione] Viglione, F.; Masera, G.; Piccinini, G.; Ruo Roch, R.; "Zamboni, M.; A 50 Mbit/s iterative turbo-decoder," Design, Automation and Test in Europe Conference and Exhibition 2000, Proceedings, pp. 176 – 180, 27-30 March 2000

[viterbi1] A. J. Viterbi, "Error bounds for convolutional codes and an assymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, vol. 13, pp. 260–269, Apr. 1967.

[viterbi2] 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, pp. 260–264, Feb. 1998

[vogt1] Vogt, J.; Finger, A.; "Improving the max-log-MAP turbo decoder" *Electronics Letters,* Volume: 36 Issue: 23, pp. 1937 –1939, 9 Nov. 2000

[vogt2] Vogt, J.; Koors, K.; Finger, A.; Fettweis, G.; "Comparison of different turbo decoder realizations for IMT-2000," *Global Telecommunications Conference, 1999 GLOBECOM '99*, Volume: 5, pp. 2704 –2708, 1999

[wang1] Zhongfeng Wang; Suzuki, H.; Parhi, K.K.; "VLSI implementation issues of TURBO decoder design for wireless applications," *IEEE Workshop on Signal Processing Systems, 1999, SiPS 99*, pp. 503 – 512, 20-22 Oct. 1999

[wang2] Zhongfeng Wang; Parhi, K.K.; "High performance, high throughput turbo/SOVA decoder design," *IEEE Transactions on Communications*, Volume: 51, Issue: 4, pp. 570 – 579, April 2003

[wang3] 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, Volume: 10, Issue: 6, pp. 902 – 912, Dec. 2002

[wee] Wee-Peng Ang; Garg, H.K.; "A new iterative channel estimator for the log-MAP & max-log-MAP turbo decoder in Rayleigh fading channel," *Global Telecommunications Conference, 2001, GLOBECOM '01, IEEE*, Vol. 6, pp. 3252-3256, 25-29 Nov. 2001

[wei] Lei Wei; "Near Shannon limit decoding for 1% packet error rate," *Global Telecommunications Conference, 1998, GLOBECOM 98*, The Bridge to Global Integration IEEE, Vol. 5, pp. 2818 –2823, 8-12 Nov. 1998

[wiesler] Wiesler, A.; Muller, O.; Jondral, F.; "MAP-algorithm with fixed-point representation for software radios," *Vehicular Technology Conference, 2000. IEEE VTS-Fall VTC 2000.* 52nd, Volume: 6, pp. 2962 –2968, 24-28 Sept. 2000

[worm] Worm, A.; Hoeher, P.; Wehn, N.; "Turbo-decoding without SNR estimation," *Communications Letters, IEEE*, Volume: 4, Issue: 6, pp. 193 - 195, June 2000.

[wu1] Wu, P.H.-Y.; Pisuk, S.M.; "Implementation of a low complexity, low power, integer-based turbo decoder," *Global Telecommunications Conference, 2001, GLOBECOM '01, IEEE*, Vol. 2, pp. 946 –951, 25-29 Nov. 2001

[wu2] Wu, P.H.-Y.; "On the complexity of turbo decoding algorithms," *Vehicular Technology Conference, 2001, VTC 2001,* IEEE VTS 53rd, Vol. 2, pp. 1439 -1443, 6-9 May 2001

[youyun] Xu Youyun; Li Zongwang; Ruan Ming; Luo Hanwen; Song Wentao; "VLSI design and implementation of WCDMA channel decoder," *Canadian Conference on Electrical and Computer Engineering, 2001*, Volume: 1, pp. 241 – 245, 13-16 May 2001

[zeng] Xiao-Jun Zeng; Zhi-Liang Hong; "Design and implementation of a turbo decoder for 3G W-CDMA systems," *IEEE Transactions on Consumer Electronics*, Volume: 48, Issue: 2, pp. 284 – 291, May 2002

Appendix A

List of Abbreviations

3GPP Third Generation Partnership Project
AM Amplitude Modulation
AMPS Advanced Mobile Phone System
API Application Programming Interface
APP A Posteriori Probability
ASIC Application-specific Integrated Circuit
AWGN Additive White Gaussian Noise
BCJR Bahl, Cocke, Jelinek, and Raviv (algorithm)
BER Bit Error Rate
BSC Binary Symmetric Channel
CB Citizens Band
CCM Configurable Computing Machine
CDMA Code-Division Multiple-Access
CEPT Conference Europeenne des Postes et Telecommunications
CLB Configuration Logic Block
DMC Discrete Memoryless Channel
DS Direct-Sequence (spread-spectrum)
DSP Digital Signal Processing
ETACS European Total Access Cellular System
FIFO First-In, First-Out
FM Frequency Modulation
FPGA Field-Programmable Gate Array
FSM Finite State Machine
GSM Global System for Mobile communications
HDL Hardware Descriptive Language
IMT 2000 International Mobile Telecommunications by the year 2000

IOB Input-Output Blocks
IP Intellectual Property
ITU International Telecommunications Union
JHDL Java-based Hardware Descriptive Language
IFU Interconnect Functional Unit
LC Logic Cell
LUT Look-Up Table
LLR Log-Likelihood Ratio
LVS Layout Versus Schematic
MAP Maximum A Posteriori (algorithm)
NAMPS Narrowband AMPS
NMT Nordic Mobile Telephone
PCCC Parallel Concatenated Convolutional Code
PCS Personal Communication Systems
PE Processing Element
PSTN Public Switched Telephone Network
QOS Quality of Service
RMTS Radio Telephone Mobile System
RSC Recursive Systematic Convolutional (code)
RTR Run-Time Reconfiguration
SCCC Serial Concatenated Convolutional Code
SDR Sign Difference Ratio
SISO Soft-Input, Soft-Output
SNR Signal-To-Noise ratio
SOVA Soft-Output Viterbi Algorithm
SRAM Synchronous Random Access Memory
TDMA Time-Division Multiple-Access
TTD Time-Division Multiple-Access
VA Viterbi Algorithm
VHDL VLSI Hardware Descriptive Language
VLSI Very Large Scale Integrated circuits
W-CDMA Wideband CDMA

Appendix B

Verilog HDL Code for the MAP Decoder

```
`timescale 1ns/1ps
`include "RAM1.v"
`include "RAM2.v"

module topMAPchp2(start,block,Reset,CLK,Yd,Yp,app,l);

 input start,block,Reset;
 input CLK;
 input [4:0] Yd,Yp;
 input [4:0] app;
 output [7:0] l;
 wire start_top,block_top,Reset_top,CLK_top;
 wire [4:0] Yd_top,Yp_top,app_top;
 wire [7:0] l_top;

 MAPdecoder2
MAPdecoder2(start_top,block_top,Reset_top,CLK_top,Yd_top,Yp_top,app_top
,l_top);


  PDIDGZ xstart ( .C(start_top), .PAD(start) );
  PDIDGZ xblock ( .C(block_top), .PAD(block) );
  PDIDGZ xReset ( .C(Reset_top), .PAD(Reset) );
  PDIDGZ xCLK ( .C(CLK_top), .PAD(CLK) );
  PDIDGZ xYd00 ( .C(Yd_top[0]), .PAD(Yd[0]) );
  PDIDGZ xYd01 ( .C(Yd_top[1]), .PAD(Yd[1]) );
  PDIDGZ xYd02 ( .C(Yd_top[2]), .PAD(Yd[2]) );
  PDIDGZ xYd03 ( .C(Yd_top[3]), .PAD(Yd[3]) );
```

```
  PDIDGZ xYd04 ( .C(Yd_top[4]), .PAD(Yd[4]) );
  PDIDGZ xYp00 ( .C(Yp_top[0]), .PAD(Yp[0]) );
  PDIDGZ xYp01 ( .C(Yp_top[1]), .PAD(Yp[1]) );
  PDIDGZ xYp02 ( .C(Yp_top[2]), .PAD(Yp[2]) );
  PDIDGZ xYp03 ( .C(Yp_top[3]), .PAD(Yp[3]) );
  PDIDGZ xYp04 ( .C(Yp_top[4]), .PAD(Yp[4]) );
  PDIDGZ xapp00 ( .C(app_top[0]), .PAD(app[0]) );
  PDIDGZ xapp01 ( .C(app_top[1]), .PAD(app[1]) );
  PDIDGZ xapp02 ( .C(app_top[2]), .PAD(app[2]) );
  PDIDGZ xapp03 ( .C(app_top[3]), .PAD(app[3]) );
  PDIDGZ xapp04 ( .C(app_top[4]), .PAD(app[4]) );

  PDO08CDG xl00 ( .PAD(l[0]), .I(l_top[0]) );
  PDO08CDG xl01 ( .PAD(l[1]), .I(l_top[1]) );
  PDO08CDG xl02 ( .PAD(l[2]), .I(l_top[2]) );
  PDO08CDG xl03 ( .PAD(l[3]), .I(l_top[3]) );
  PDO08CDG xl04 ( .PAD(l[4]), .I(l_top[4]) );
  PDO08CDG xl05 ( .PAD(l[5]), .I(l_top[5]) );
  PDO08CDG xl06 ( .PAD(l[6]), .I(l_top[6]) );
  PDO08CDG xl07 ( .PAD(l[7]), .I(l_top[7]) );


endmodule

module MAPdecoder2(start,block,Reset,CLK,Yd,Yp,app,l);

 input start,block,Reset;
 input CLK;
 input [4:0] Yd,Yp;
 input [4:0] app;
 output [7:0] l;

 wire [31:0] Q,Q2;
 wire [7:0] g1,g2,g3,g4;
 wire [7:0] add1,add2,add3,add4;
 wire [7:0] pr1,prm1;
 wire OE,OE2, ME,ME2,WE,WEr1,busy;
 wire [9:0] ADR,ADR2,lastADR;
 wire [31:0] D,D2;
 wire [7:0] a1,a2,a3,a4;
 wire [3:0] n,nb,nl;
 wire [7:0] y1,y2,y3,y4,y5,y6,y7,y8,b1,b2,b3,b4,max1,max2,max_a;
 reg [31:0] gg1,gg2,gg3,gg4,aa1,aa2,aa3,aa4;
 reg [7:0] l;

        //-------------------------------------------------------------
-------------------------------------------------
       add_sub2 add_sub2(busy,start,Yd,Yp,add1,add2,add3,add4);
       LUT2 LUT2(busy,start,app,pr1,prm1);
       add22
add22(Reset,block,n,busy,CLK,add1,add2,add3,add4,pr1,prm1,g1,g2,g3,g4);

       alpha2 alpha2(nb,busy,n,Reset,CLK,g1,g2,g3,g4,a1,a2,a3,a4,max_a);
       write22
write22(n,busy,block,Reset,CLK,a1,a2,a3,a4,aa1,aa2,aa3,aa4,gg1,gg2,gg3,
gg4,b1,b2,b3,b4,ADR2,D2,WEr1,ME2,OE2,nl,max1,max2);//Write alpha
```

```
      hdss1_1024x32cm8 write_a2( Q2, ADR2,  D2, WEr1, OE2, ME2,  CLK);
//##### RAM2 #####//

      write2
write2(start,block,Reset,CLK,max_a,gg1,gg2,gg3,gg4,g1,g2,g3,g4,b1,b2,b3
,b4,ADR, D, WE,ME,OE,n,nb,busy,lastADR);//Write Gamma
      hdss1_1024x32cm8 write_g2( Q, ADR,  D, WE, OE, ME,  CLK); //#####
RAM1 #####//
      //-----------------------------------------------------------------
----------------------------------------------
       always@(negedge CLK)   //Reading from RAM1 (Gammas) for Beta and
log-liklihood calculation
       begin
                      if(busy==1)
                   begin
                       case(nb)
                        1:  gg4=Q;
                        2: begin if(ADR==lastADR-1)  gg3=0;else gg3=Q;
end //End of the block goes back to state 0
                        3: begin if(ADR==lastADR-2)  gg2=0;else gg2=Q;
end //End of the block goes back to state 0
                        4:  gg1=Q;
                     endcase
                   end
       end//always

       //-----------------------------------------------------------------
----------------------------------------------
       always@(negedge CLK)   //Reading from RAM2 (alphas) for log-
liklihood calculation
       begin
                      if(busy==1)
                   begin
                       case(nl)
                        1:  aa4=Q2;
                        2:  aa3=Q2;
                        3:  aa2=Q2;
                        4:  aa1=Q2;
                     endcase
                   end
       end//always

       //-----------------------------------------------------------------
----------------------------------------------
       always@(negedge CLK)
       begin

                  if(nl==7 && busy==1)
                  begin
                      l<=max1-max2;
                      //nl=0;
                    end
       end//always

      //-----------------------------------------------------------------
-----------------------------------------
//    initial begin
```

```
//    $shm_open("decoder3.db");                    //open a database
file
//
      $shm_probe(CLK,start,Reset,busy,n,b1,b2,b3,b4,a1,a2,a3,a4,aa1,aa2
,aa3,aa4,gg1,gg2,gg3,gg4,1,block,nb,ADR,ADR2);
//        end

endmodule
      //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  MODULEs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//

module add_sub2 (busy,start,Yd,Yp,add1,add2,add3,add4);

 input busy,start;
 input [4:0] Yd,Yp;
 output [7:0] add1,add2,add3,add4;
 reg [7:0] add1,add2,add3,add4;


    always@(negedge start)
    begin
      if(busy==0)
        begin
         add1=-Yd-Yp;
       add2=-Yd+Yp;
       add3=Yd-Yp;
       add4=Yp+Yd;
      end
    end
endmodule


//****************************************
module LUT2(busy,start,app,pr1,prm1);

 input busy,start;
 input [4:0] app;
 output [7:0] pr1,prm1;
 reg [7:0] pr1,prm1;

       always@(negedge start)
       begin
           if(busy==0)
           begin
               case(app)
               5'h 18: begin pr1=8'h F8; prm1=0    ;end      //F8 & 18-
>2's Complement of -8
             5'h 19: begin pr1=8'h F9; prm1=0    ;end      //F9 & 19-
>2's Complement of -7
             5'h 1A: begin pr1=8'h FA; prm1=0    ;end      //FA & 1A-
>2's Complement of -6
             5'h 1B: begin pr1=8'h FB; prm1=0    ;end      //FB & 1B-
>2's Complement of -5
             5'h 1C: begin pr1=8'h FC; prm1=0    ;end      //FC & 1C-
>2's Complement of -4
             5'h 1D: begin pr1=8'h FD; prm1=0    ;end      //FD & 1D-
>2's Complement of -3
```

```
            5'h 1E: begin pr1=8'h FE; prm1=0    ;end      //FE & 1E-
>2's Complement of -2
            5'h 1F: begin pr1=8'h FF; prm1=0    ;end      //FF & 1F-
>2's Complement of -1
                 0: begin pr1=0 ;     prm1=0    ;end
                 1: begin pr1=0 ;    prm1=8'h FF;end
                 2: begin pr1=0 ;    prm1=8'h FE;end
                 3: begin pr1=0 ;    prm1=8'h FD;end
                 4: begin pr1=0 ;    prm1=8'h FC;end
                 5: begin pr1=0 ;    prm1=8'h FB;end
                 6: begin pr1=0 ;    prm1=8'h FA;end
                 7: begin pr1=0 ;    prm1=8'h F9;end
                 8: begin pr1=0 ;    prm1=8'h F8;end
               default:begin pr1=0; prm1=0;end
              endcase
          end//if
        end
endmodule

//*********************************************************************
*******
module
add22(Reset,block,n,busy,CLK,add1,add2,add3,add4,pr1,prm1,g1,g2,g3,g4);

 input Reset,block,busy,CLK;
 input [3:0]n;
 input [7:0] add1,add2,add3,add4,pr1,prm1;
 output [7:0] g1,g2,g3,g4;
 reg [7:0] g1,g2,g3,g4;
 reg gstart;

       always@(negedge CLK)//(pr1 or prm1 or add1 or add2 or add3 or
add4)
       begin
          if(Reset) gstart=0;
            else if(busy==0)
           begin
                   g1=prm1+add1;
                 if(gstart==0) g2=0; else g2=prm1+add2; //if n=0 or
Start then g2=0(begining of the block is from state 0)
             if(gstart==0) g3=0; else g3=pr1+add3;
             g4=pr1+add4;
             if(n==4) gstart=1;
           end
           if(block) gstart=0;
       end
endmodule

//*********************************************************************
************************************

module alpha2(nb,busy,n,Reset,CLK,g1,g2,g3,g4,a1,a2,a3,a4,max_a);

input busy,Reset,CLK;
input [7:0]g1,g2,g3,g4;
input [3:0]n,nb;
output [7:0]a1,a2,a3,a4,max_a;
```

```
reg [7:0]a11,a12,a13,a14,a1,a2,a3,a4,max_a;
reg [7:0]x1,x2,x3,x4,x5,x6,x7,x8;
reg [7:0] temp1,temp2,tmpa1,tmpa2,tmpa3,tmpa4;
reg [1:0] flag;

always@(negedge CLK)
begin
    if(Reset)
       begin          //Initializing Alphas,a=0 =>ln(a)=-infinity(-128)
          a1=0;
            a2=8'h80;//-128;
          a3=8'h80;//-128;
          a4=8'h80;//-128;
          a11=0;
            a12=8'h80;//-128;
          a13=8'h80;//-128;
          a14=8'h80;//-128;
       end

    else if (nb==7)
      begin
          a1=0;
            a2=8'h80;//-128;
          a3=8'h80;//-128;
          a4=8'h80;//-128;
          a11=0;
            a12=8'h80;//-128;
          a13=8'h80;//-128;
          a14=8'h80;//-128;
          flag=0;
      end


    else if(n==0 && busy==0)  //Start
      begin
      x1=g1+a11;
      x2=g4+a12;
      x3=g3+a13;
      x4=g2+a14;
      x5=g4+a11;
      x6=g1+a12;
      x7=g2+a13;
      x8=g3+a14;

      if (g1[7]==1'b1 && a11[7]==1'b1 && x1[7]==1'b0)  x1=8'h80; else
if(g1[7]==1'b0 && a11[7]==1'b0 && x1[7]==1'b1) x1=127;
      if (g4[7]==1'b1 && a12[7]==1'b1 && x2[7]==1'b0)  x2=8'h80; else
if(g4[7]==1'b0 && a12[7]==1'b0 && x2[7]==1'b1) x2=127;
      if (g3[7]==1'b1 && a13[7]==1'b1 && x3[7]==1'b0)  x3=8'h80; else
if(g3[7]==1'b0 && a13[7]==1'b0 && x3[7]==1'b1) x3=127;
      if (g2[7]==1'b1 && a14[7]==1'b1 && x4[7]==1'b0)  x4=8'h80; else
if(g2[7]==1'b0 && a14[7]==1'b0 && x4[7]==1'b1) x4=127;
      if (g4[7]==1'b1 && a11[7]==1'b1 && x5[7]==1'b0)  x5=8'h80; else
if(g4[7]==1'b0 && a11[7]==1'b0 && x5[7]==1'b1) x5=127;
      if (g1[7]==1'b1 && a12[7]==1'b1 && x6[7]==1'b0)  x6=8'h80; else
if(g1[7]==1'b0 && a12[7]==1'b0 && x6[7]==1'b1) x6=127;
```

```
      if (g2[7]==1'b1 && a13[7]==1'b1 && x7[7]==1'b0)  x7=8'h80; else
if(g2[7]==1'b0 && a13[7]==1'b0 && x7[7]==1'b1) x7=127;
      if (g3[7]==1'b1 && a14[7]==1'b1 && x8[7]==1'b0)  x8=8'h80; else
if(g3[7]==1'b0 && a14[7]==1'b0 && x8[7]==1'b1) x8=127;

        if (x1[7]==1'b0 && x2[7]==1'b1)      a11=x1;
//a1=MAX(g1+a1 and g4+a2)
        else if (x1[7]==1'b1 && x2[7]==1'b0) a11=x2;
        else if (x1[7]==x2[7] && x1>= x2)    a11=x1;
          else if (x1[7]==x2[7] && x1<= x2)    a11=x2;

        if (x3[7]==1'b0 && x4[7]==1'b1)      a12=x3;
//a2=MAX(g3+a3 and g2+a4)
        else if (x3[7]==1'b1 && x4[7]==1'b0) a12=x4;
        else if (x3[7]==x4[7] && x3>= x4)    a12=x3;
        else if (x3[7]==x4[7] && x3<= x4)    a12=x4;

         if (x5[7]==1'b0 && x6[7]==1'b1)     a13=x5;
//a3=MAX(g4+a1 and g1+a2)
        else if (x5[7]==1'b1 && x6[7]==1'b0) a13=x6;
        else if (x5[7]==x6[7] && x5>= x6)    a13=x5;
        else if (x5[7]==x6[7] && x5<= x6)    a13=x6;

         if (x7[7]==1'b0 && x8[7]==1'b1)     a14=x7;
//a4=MAX(g2+a3 and g3+a4)
        else if (x7[7]==1'b1 && x8[7]==1'b0) a14=x8;
        else if (x7[7]==x8[7] && x7>= x8)    a14=x7;
        else if (x7[7]==x8[7] && x7<= x8)    a14=x8;

        //--------------------------- Finding Maximum alpha -----------
---------------//

        if (a11[7]==1'b0 && a12[7]==1'b1)             temp1=a11;
//a1 positive,a2 negative
        else if (a11[7]==1'b1 && a12[7]==1'b0)        temp1=a12;
        else if (a11[7]==a12[7] && a11>= a12)         temp1=a11;
          else if (a11[7]==a12[7] && a11<= a12)         temp1=a12;

        if (a13[7]==1'b0 && a14[7]==1'b1)             temp2=a13;
//a3 positive,a4 negative
        else if (a13[7]==1'b1 && a14[7]==1'b0)        temp2=a14;
        else if (a13[7]==a14[7] && a13>= a14)         temp2=a13;
          else if (a13[7]==a14[7] && a13<= a14)         temp2=a14;

        if (temp1[7]==1'b0 && temp2[7]==1'b1)         max_a=temp1;
        else if (temp1[7]==1'b1 && temp2[7]==1'b0)    max_a=temp2;
        else if (temp1[7]==temp2[7] && temp1>= temp2) max_a=temp1;
          else if (temp1[7]==temp2[7] && temp1<= temp2) max_a=temp2;

         //------------ alpha normalization --------------//
         max_a=-max_a;

        tmpa1=a11+max_a;
        tmpa2=a12+max_a;
        tmpa3=a13+max_a;
        tmpa4=a14+max_a;
```

```
      if (a11[7]==1'b1 && max_a[7]==1'b1 && tmpa1[7]==1'b0)
tmpa1=8'h80;//-128;
      else if(a11[7]==1'b0 && max_a[7]==1'b0 && tmpa1[7]==1'b1)
tmpa1=127;

      if (a12[7]==1'b1 && max_a[7]==1'b1 && tmpa2[7]==1'b0)
tmpa2=8'h80;//-128;
      else if(a12[7]==1'b0 && max_a[7]==1'b0 && tmpa2[7]==1'b1)
tmpa2=127;

      if (a13[7]==1'b1 && max_a[7]==1'b1 && tmpa3[7]==1'b0)
tmpa3=8'h80;//-128;
      else if(a13[7]==1'b0 && max_a[7]==1'b0 && tmpa3[7]==1'b1)
tmpa3=127;

      if (a14[7]==1'b1 && max_a[7]==1'b1 && tmpa4[7]==1'b0)
tmpa4=8'h80;//-128;
      else if(a14[7]==1'b0 && max_a[7]==1'b0 && tmpa4[7]==1'b1)
tmpa4=127;

      a1=tmpa1;
      a2=tmpa2;
      a3=tmpa3;
      a4=tmpa4;
       end
 end
endmodule
//**************************************************************************
**************************************

module
write22(n,busy,block,Reset,CLK,a1,a2,a3,a4,aa1,aa2,aa3,aa4,gg1,gg2,gg3,
gg4,b1,b2,b3,b4,ADR2,D2,WEr1,ME2,OE2,nl,max1,max2);

input busy,block,Reset,CLK;
input [7:0] a1,a2,a3,a4;
input [3:0]n;
input [31:0] aa1,aa2,aa3,aa4,gg1,gg2,gg3,gg4;
input [7:0] b1,b2,b3,b4;
output[7:0] max1,max2;
output [9:0] ADR2;
output [31:0] D2;
output OE2,WEr1,ME2;
output [3:0]nl;
reg WEr1,OE2,ME2;
reg [9:0] ADR2;
reg [31:0] D2;
reg [3:0]nn2;
reg [3:0]nl;
reg [7:0]cmp1,cmp2,cmp3,cmp4,max1,max2;
reg [7:0]
z1,z2,z3,z4,z5,z6,z7,z8,i1,i2,i3,i4,i5,i6,i7,i8,j1,j2,j3,j4,j5,j6,j7,j8
;

always@(negedge CLK)
  begin
        if(Reset)
```

```
        begin
          nn2=4;
          ADR2=10'hfff;//-1;
          OE2 = 1'b0;
          ME2 = 1'b1;
        end//Reset

//++++++++++++++++++++++ Reading Alpha and Log-liklihood calculation
++++++++++++++++++++//

             else if (block)
                  begin
                   WEr1=1'b0;
                   nl=0;
                   ADR2=ADR2-3;
                  end

            else if((busy==1) && (ADR2>0) && (nl>=0) && (nl<4))
                  begin
                   ADR2=ADR2-1;
                   OE2 = 1'b1;
                   nl=nl+1;
                    end

               else if (nl==4 && busy==1) nl=nl+1;

          else if (nl==5 && busy==1)
               begin

//MAX1(a2+b1+g4 , a3+b2+g3 , a1+b3+g4 , a4+b4+g3) - MAX0(a1+b1+g1 ,
a4+b2+g2 , a2+b3+g1 , a3+b4+g2)

               nl=6;

        i1=aa2+b1;
        i2=aa3+b2;
        i3=aa1+b3;
        i4=aa4+b4;
        i5=aa1+b1;
        i6=aa4+b2;
        i7=aa2+b3;
        i8=aa3+b4;

if(aa2[7]==1'b1 && b1[7]==1'b1 && i1[7]==1'b0)  i1=8'h80; else
if(aa2[7]==1'b0 && b1[7]==1'b0 && i1[7]==1'b1) i1=127;
if(aa3[7]==1'b1 && b2[7]==1'b1 && i2[7]==1'b0)  i2=8'h80; else
if(aa3[7]==1'b0 && b2[7]==1'b0 && i2[7]==1'b1) i2=127;
if(aa1[7]==1'b1 && b3[7]==1'b1 && i3[7]==1'b0)  i3=8'h80; else
if(aa1[7]==1'b0 && b3[7]==1'b0 && i3[7]==1'b1) i3=127;
if(aa4[7]==1'b1 && b4[7]==1'b1 && i4[7]==1'b0)  i4=8'h80; else
if(aa4[7]==1'b0 && b4[7]==1'b0 && i4[7]==1'b1) i4=127;
if(aa1[7]==1'b1 && b1[7]==1'b1 && i5[7]==1'b0)  i5=8'h80; else
if(aa1[7]==1'b0 && b1[7]==1'b0 && i5[7]==1'b1) i5=127;
if(aa4[7]==1'b1 && b2[7]==1'b1 && i6[7]==1'b0)  i6=8'h80; else
if(aa4[7]==1'b0 && b2[7]==1'b0 && i6[7]==1'b1) i6=127;
if(aa2[7]==1'b1 && b3[7]==1'b1 && i7[7]==1'b0)  i7=8'h80; else
if(aa2[7]==1'b0 && b3[7]==1'b0 && i7[7]==1'b1) i7=127;
```

```
if(aa3[7]==1'b1 && b4[7]==1'b1 && i8[7]==1'b0)  i8=8'h80; else
if(aa3[7]==1'b0 && b4[7]==1'b0 && i8[7]==1'b1) i8=127;

       z1=i1+gg4;  //z1=aa2+b1+gg4;
       z2=i2+gg3;  //z2=aa3+b2+gg3;
       z3=i3+gg4;  //z3=aa1+b3+gg4;
       z4=i4+gg3;  //z4=aa4+b4+gg3;
       z5=i5+gg1;  //z5=aa1+b1+gg1;
       z6=i6+gg2;  //z6=aa4+b2+gg2;
       z7=i7+gg1;  //z7=aa2+b3+gg1;
       z8=i8+gg2;  //z8=aa3+b4+gg2;

if(i1[7]==1'b1 && gg4[7]==1'b1 && z1[7]==1'b0)  z1=8'h80; else
if(i1[7]==1'b0 && gg4[7]==1'b0 && z1[7]==1'b1) z1=127;
if(i2[7]==1'b1 && gg3[7]==1'b1 && z2[7]==1'b0)  z2=8'h80; else
if(i2[7]==1'b0 && gg3[7]==1'b0 && z2[7]==1'b1) z2=127;
if(i3[7]==1'b1 && gg4[7]==1'b1 && z3[7]==1'b0)  z3=8'h80; else
if(i3[7]==1'b0 && gg4[7]==1'b0 && z3[7]==1'b1) z3=127;
if(i4[7]==1'b1 && gg3[7]==1'b1 && z4[7]==1'b0)  z4=8'h80; else
if(i4[7]==1'b0 && gg3[7]==1'b0 && z4[7]==1'b1) z4=127;
if(i5[7]==1'b1 && gg1[7]==1'b1 && z5[7]==1'b0)  z5=8'h80; else
if(i5[7]==1'b0 && gg1[7]==1'b0 && z5[7]==1'b1) z5=127;
if(i6[7]==1'b1 && gg2[7]==1'b1 && z6[7]==1'b0)  z6=8'h80; else
if(i6[7]==1'b0 && gg2[7]==1'b0 && z6[7]==1'b1) z6=127;
if(i7[7]==1'b1 && gg1[7]==1'b1 && z7[7]==1'b0)  z7=8'h80; else
if(i7[7]==1'b0 && gg1[7]==1'b0 && z7[7]==1'b1) z7=127;
if(i8[7]==1'b1 && gg2[7]==1'b1 && z8[7]==1'b0)  z8=8'h80; else
if(i8[7]==1'b0 && gg2[7]==1'b0 && z8[7]==1'b1) z8=127;


                    if (z1[7]==1'b0 && z2[7]==1'b1) cmp1=z1;
               else if (z1[7]==1'b1 && z2[7]==1'b0) cmp1=z2;
               else if (z1[7]==z2[7] && z1>= z2)    cmp1=z1;
               else if (z1[7]==z2[7] && z1<= z2)    cmp1=z2;

                    if (z3[7]==1'b0 && z4[7]==1'b1) cmp2=z3;
               else if (z3[7]==1'b1 && z4[7]==1'b0) cmp2=z4;
               else if (z3[7]==z4[7] && z3>= z4)    cmp2=z3;
               else if (z3[7]==z4[7] && z3<= z4)    cmp2=z4;

                    if (z5[7]==1'b0 && z6[7]==1'b1) cmp3=z5;
               else if (z5[7]==1'b1 && z6[7]==1'b0) cmp3=z6;
               else if (z5[7]==z6[7] && z5>= z6)    cmp3=z5;
               else if (z5[7]==z6[7] && z5<= z6)    cmp3=z6;

                    if (z7[7]==1'b0 && z8[7]==1'b1) cmp4=z7;
               else if (z7[7]==1'b1 && z8[7]==1'b0) cmp4=z8;
               else if (z7[7]==z8[7] && z7>= z8)    cmp4=z7;
               else if (z7[7]==z8[7] && z7<= z8)    cmp4=z8;
               end

     else if(nl==6 && busy==1)
          begin
                nl=7;
            if (cmp1[7]==1'b0 && cmp2[7]==1'b1)          max1=cmp1;
//com1=positive com2=negative
```

```
              else if (cmp1[7]==1'b1 && cmp2[7]==1'b0)      max1=cmp2;
//com1=negative com2=positive
              else if (cmp1[7]==cmp2[7] && cmp1>= cmp2)     max1=cmp1;
              else if (cmp1[7]==cmp2[7] && cmp1<= cmp2)     max1=cmp2;

              if (cmp3[7]==1'b0 && cmp4[7]==1'b1)           max2=cmp3;
//com3=positive  com4=negative
              else if (cmp3[7]==1'b1 && cmp4[7]==1'b0)      max2=cmp4;
//com3=negative  com4=positive
              else if (cmp3[7]==cmp4[7] && cmp3>= cmp4)     max2=cmp3;
              else if (cmp3[7]==cmp4[7] && cmp3<= cmp4)     max2=cmp4;
           end

                        else if(nl==7 && busy==1)
                        begin
                         // l<=max1-max2;    //in main body
                            nl=0;
                        end

//++++++++++++++++++++++++++++++++++ writing alpha
++++++++++++++++++++++++++++++++//
         else if(nn2>=4 && nn2<8 && busy==0)
                 begin
                ADR2 = ADR2+1;
                nn2=nn2+1;
                   case (nn2)
                   5: begin D2 = a1; WEr1 = 1'b1; end
                   6: begin D2 = a2; WEr1 = 1'b1; end
                   7: begin D2 = a3; WEr1 = 1'b1; end
                   8: begin D2 = a4; WEr1 = 1'b1; end
                   //default: D2 = a1;
                   endcase

                 end

         else if(  nn2<3 || nn2==4'b1111 & busy==0)  //n2=-1 or 0,1,2
              begin
               ADR2 = ADR2+1;
              nn2=nn2+1;
                   case (nn2)
                   0: begin D2 = a1; WEr1 = 1'b1; end
                   1: begin D2 = a2; WEr1 = 1'b1; end
                   2: begin D2 = a3; WEr1 = 1'b1; end
                   3: begin D2 = a4; WEr1 = 1'b1; end
                   default: D2 = a1;
                   endcase
             end//if n<3

        else if (n==0 && busy==0) nn2=4'hf;//-1;      //when start
signal is received

       else if (ADR2==0) begin ADR2=ADR2-1; nn2=4; end
//??????????????????????????????


end//always
endmodule
```

```
//*********************************************************************
*******
module
write2(start,block,Reset,CLK,max_a,gg1,gg2,gg3,gg4,g1,g2,g3,g4,b1,b2,b3
,b4,ADR,D,WE,ME,OE,n,nb,busy,lastADR);

input start,block,Reset,CLK;
input [7:0] max_a,g1,g2,g3,g4;
input[31:0] gg1,gg2,gg3,gg4;
output [7:0] b1,b2,b3,b4;
output [9:0] ADR,lastADR;
output [31:0] D;
output WE,OE,ME,busy;
output [3:0] n,nb;
reg WE,OE,ME;
reg [9:0] ADR,lastADR;
reg [31:0] D;
reg [3:0] n,m,nb;
reg busy;
reg [7:0] y1,y2,y3,y4,y5,y6,y7,y8,b1,b2,b3,b4;
reg [7:0] tmpb1,tmpb2,tmpb3,tmpb4;
reg check2,nstart;

always@(negedge CLK)
begin

   if(Reset)  begin
   ADR=10'hfff;//-1;
   OE = 1'b0;
   ME = 1'b1;
   busy=0;
   check2=0;
   nstart=0;
               end

  //+++++++++++++++++++++++++++++++++++++ Beta Calculation
++++++++++++++++++++++++++++++++
  //          ------------------------------  Busy=1  ----------------
---------------        //

          else if (block)
          begin
             WE=1'b0;
             b4=8'h80;//-128;
             b3=8'h80;//-128;
             b2=8'h80;//-128;
             b1=0;
             busy=1'b1;//At the end of the block busy=1 and Beta
calculation is started
             nb=0;
             lastADR=ADR;
             ADR=ADR+1;
            end


             else if((busy==1) && (ADR>0) && (nb>=0) && (nb<4))
```

```
                begin
                  ADR=ADR-1;    //Reading gammas
                  OE = 1'b1;
                  nb=nb+1;
                end

        else if (nb<=6 && busy==1) nb=nb+1;          //make gamma to
be read synchron with alpha and beta
          else if((nb==7) && (busy==1))
        begin
           y1=gg1+b1;
           y2=gg4+b2;
           y3=gg3+b3;
           y4=gg2+b4;
           y5=gg4+b1;
           y6=gg1+b2;
           y7=gg2+b3;
           y8=gg3+b4;
                                  //-------- Check for overflow or
underflow ---------//

            if (gg1[7]==1'b1 && b1[7]==1'b1 && y1[7]==1'b0)  y1=8'h80;
else if(gg1[7]==1'b0 && b1[7]==1'b0 && y1[7]==1'b1) y1=127;
            if (gg4[7]==1'b1 && b2[7]==1'b1 && y2[7]==1'b0)  y2=8'h80;
else if(gg4[7]==1'b0 && b2[7]==1'b0 && y2[7]==1'b1) y2=127;
            if (gg3[7]==1'b1 && b3[7]==1'b1 && y3[7]==1'b0)  y3=8'h80;
else if(gg3[7]==1'b0 && b3[7]==1'b0 && y3[7]==1'b1) y3=127;
            if (gg2[7]==1'b1 && b4[7]==1'b1 && y4[7]==1'b0)  y4=8'h80;
else if(gg2[7]==1'b0 && b4[7]==1'b0 && y4[7]==1'b1) y4=127;
            if (gg4[7]==1'b1 && b1[7]==1'b1 && y5[7]==1'b0)  y5=8'h80;
else if(gg4[7]==1'b0 && b1[7]==1'b0 && y5[7]==1'b1) y5=127;
            if (gg1[7]==1'b1 && b2[7]==1'b1 && y6[7]==1'b0)  y6=8'h80;
else if(gg1[7]==1'b0 && b2[7]==1'b0 && y6[7]==1'b1) y6=127;
            if (gg2[7]==1'b1 && b3[7]==1'b1 && y7[7]==1'b0)  y7=8'h80;
else if(gg2[7]==1'b0 && b3[7]==1'b0 && y7[7]==1'b1) y7=127;
            if (gg3[7]==1'b1 && b4[7]==1'b1 && y8[7]==1'b0)  y8=8'h80;
else if(gg3[7]==1'b0 && b4[7]==1'b0 && y8[7]==1'b1) y8=127;

              if (y1[7]==1'b0 && y2[7]==1'b1)      b1=y1;
//b1=MAX(g1+b1 and g4+b2)
            else if (y1[7]==1'b1 && y2[7]==1'b0) b1=y2;
            else if (y1[7]==y2[7] && y1>= y2)    b1=y1;
              else if (y1[7]==y2[7] && y1<= y2)    b1=y2;

            if (y3[7]==1'b0 && y4[7]==1'b1)      b2=y3;
//b2=MAX(g3+b3 and g2+b4)
            else if (y3[7]==1'b1 && y4[7]==1'b0) b2=y4;
            else if (y3[7]==y4[7] && y3>= y4)    b2=y3;
              else if (y3[7]==y4[7] && y3<= y4)    b2=y4;

            if (y5[7]==1'b0 && y6[7]==1'b1)      b3=y5;
//b3=MAX(g4+b1 and g1+b2)
            else if (y5[7]==1'b1 && y6[7]==1'b0) b3=y6;
            else if (y5[7]==y6[7] && y5>= y6)    b3=y5;
              else if (y5[7]==y6[7] && y5<= y6)    b3=y6;
```

```
              if (y7[7]==1'b0 && y8[7]==1'b1)      b4=y7;
//b4=MAX(g2+b3 and g3+b4)
              else if (y7[7]==1'b1 && y8[7]==1'b0) b4=y8;
              else if (y7[7]==y8[7] && y7>= y8)    b4=y7;
                else if (y7[7]==y8[7] && y7<= y8)    b4=y8;


                          //---------------------- beta normalization
-------------------------//
        tmpb1=b1+max_a;      // tmpb1=b1-max_a;    max_a=-max_a
        tmpb2=b2+max_a;      // tmpb2=b2-max_a;    max_a=-max_a
        tmpb3=b3+max_a;      // tmpb2=b2-max_a;    max_a=-max_a
        tmpb4=b4+max_a;      // tmpb2=b2-max_a;    max_a=-max_a

      if (b1[7]==1'b1 && max_a[7]==1'b1 && tmpb1[7]==1'b0)
tmpb1=8'h80;//-128;
      else if(b1[7]==1'b0 && max_a[7]==1'b0 && tmpb1[7]==1'b1)
tmpb1=127;

      if (b2[7]==1'b1 && max_a[7]==1'b1 && tmpb2[7]==1'b0)
tmpb2=8'h80;//-128;
      else if(b2[7]==1'b0 && max_a[7]==1'b0 && tmpb2[7]==1'b1)
tmpb2=127;

      if (b3[7]==1'b1 && max_a[7]==1'b1 && tmpb3[7]==1'b0)
tmpb3=8'h80;//-128;
      else if(b3[7]==1'b0 && max_a[7]==1'b0 && tmpb3[7]==1'b1)
tmpb3=127;

      if (b4[7]==1'b1 && max_a[7]==1'b1 && tmpb4[7]==1'b0)
tmpb4=8'h80;//-128;
      else if(b4[7]==1'b0 && max_a[7]==1'b0 && tmpb4[7]==1'b1)
tmpb4=127;

      b1=tmpb1;
      b2=tmpb2;
      b3=tmpb3;
      b4=tmpb4;
                nb=0;
              if(ADR==0) begin busy=0; ADR=ADR-1; end //At ADR=0 Beta
computation stops and busy=0 and gamma and alpha units start working
again
            end


 //++++++++++++++++++++++++++++++++++++++ writing Gammas in RAM1
++++++++++++++++++++++++++++++++++++++++
              else if(m==0 && busy==0)  //Start
              begin //Start
                    if((n[3]==0) && (n>3)  && (busy==0))   //Waiting
for Start
                      begin
                      end

                    else
                       begin
                        n=n+1;
```

```
                        case (n)
                        0:;
                        1: begin D = g1; WE = 1'b1;ADR=ADR+1;  end
                        2: begin D = g2; WE = 1'b1;ADR=ADR+1;
if(ADR==1) D=0; end  //begining of the block is from state 0
                        3: begin D = g3; WE = 1'b1;ADR=ADR+1;
if(ADR==2) D=0; end  //begining of the block is from state 0
                        4: begin D = g4; WE = 1'b1;ADR=ADR+1;  end
                        default: D = g1;
                        endcase
                        end//else
               end//else

                  if(start && busy==0)
                  begin
                     m=0;
                     n=4'hf;//-1;
                    end
end//always
endmodule
```

Appendix C

MATLAB Code for the MAP Decoder

This is the main function of the Max-Log-MAP decoder with proposed quantization:

```
function L_all = logmapo(rec_s,g,L_a,ind_dec)

%       g: code generator for the component RSC code, in binary matrix form.
%       L_a: a priori info. for the current decoder,
%            scrambled version of extrinsic Inftyo. of the previous decoder.
%       ind_dec: index of decoder. Either 1 or 2.
%
%
% Output: L_all: log-likelihood ratio of the symbols. Complete information.

% Total number of bits: Inftyo. + tail

L_total = length(rec_s)/2;
[n,K] = size(g);
m = K - 1;
nstates = 2^m;        % number of states in the trellis
```

```
% Set up the trellis
[next_out, next_state, last_out, last_state] = trellis(g);

Infty = 128;

% Initialization of Alpha
Alpha(1,1) = 0;
Alpha(1,2:nstates) = -Infty*ones(1,nstates-1);

% Initialization of Beta
if ind_dec==1
  Beta(L_total,1) = 0;
  Beta(L_total,2:nstates) = -Infty*ones(1,nstates-1);
elseif ind_dec==2
  Beta(L_total,1:nstates) = zeros(1,nstates);
else
  fprintf('ind_dec is limited to 1 and 2!\n');
end

% Trace forward, compute Alpha
for k = 2:L_total+1
   for state2 = 1:nstates
     gamma = -Infty*ones(1,nstates);
     gamma(last_state(state2,1)) =int8( (-rec_s(2*k-3)+rec_s(2*k-2)*last_out(state2,2))....
        -log(1+exp(L_a(k-1))) );
     gamma(last_state(state2,2)) =int8( (rec_s(2*k-3)+rec_s(2*k-2)*last_out(state2,4))....
        +L_a(k-1)-log(1+exp(L_a(k-1))) );

     if(max(gamma+Alpha(k-1,:))<-Infty)%sum(exp(gamma+Alpha(k-1,:)))<-Infty)
       Alpha(k,state2)=-Infty;
    elseif(max(gamma+Alpha(k-1,:))>Infty)
       Alpha(k,state2)=Infty;
      else
       Alpha(k,state2) =int8(  max(gamma+Alpha(k-1,:))  );%log( sum( exp( gamma+Alpha(k-1,:) ) ) );
      end
     end
     %end
```

```
  tempmax(k) =  max(Alpha(k,:))  ;
  Alpha(k,:) = Alpha(k,:)- tempmax(k);
      if(max(gamma+Alpha(k-1,:))<-Infty)%sum(exp(gamma+Alpha(k-1,:)))<-Infty)
      Alpha(k,state2)=-Infty;
   elseif(max(gamma+Alpha(k-1,:))>Infty)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Alpha(k,state2)=Infty;
   end

end

% Trace backward, compute Beta
for k = L_total-1:-1:1
  for state1 = 1:nstates
    gamma = -Infty*ones(1,nstates);
    gamma(next_state(state1,1)) =int8( (-rec_s(2*k+1)+rec_s(2*k+2)*next_out(state1,2))....
         -log(1+exp(L_a(k+1)))  );
    gamma(next_state(state1,2)) =int8( (rec_s(2*k+1)+rec_s(2*k+2)*next_out(state1,4))....
         +L_a(k+1)-log(1+exp(L_a(k+1)))  );
    if(max(gamma+Beta(k+1,:))<-Infty)
      Beta(k,state1)=-Infty;
   elseif (max(gamma+Beta(k+1,:))>Infty)
      Beta(k,state1)=Infty;
    else
      Beta(k,state1) =int8(  max(gamma+Beta(k+1,:))  ); %log(sum(exp(gamma+Beta(k+1,:))));
    end
  end
  %end
  Beta(k,:) =  Beta(k,:)-tempmax(k+1);
     if(max(gamma+Beta(k+1,:))<-Infty)
      Beta(k,state1)=-Infty;
   elseif (max(gamma+Beta(k+1,:))>Infty)
      Beta(k,state1)=Infty;
   end
```

```
end

% Compute the soft output, log-likelihood ratio of symbols in the frame

for k = 1:L_total
 for state2 = 1:nstates
   gamma0 = (-rec_s(2*k-1)+rec_s(2*k)*last_out(state2,2))....
       -log(1+exp(L_a(k)));
   gamma1 = (rec_s(2*k-1)+rec_s(2*k)*last_out(state2,4))...
       +L_a(k)-log(1+exp(L_a(k)));
   temp0(state2) = (gamma0 + Alpha(k,last_state(state2,1)) + Beta(k,state2));
   temp1(state2) = (gamma1 + Alpha(k,last_state(state2,2)) + Beta(k,state2));
 end
 L_all(k) = max(temp1)-max(temp0);%log(sum(temp1)) - log(sum(temp0));
end
```

Appendix D

Simulation Reports

Synopsys Reports:

```
Information: Updating design information... (UID-85)

****************************************
Report : area
Design : topMAPchp2
Version: 2003.06
Date   : Fri Aug 20 13:52:39 2004
****************************************

Library(s) Used:

    hdss1_1024x32cm8_lib (File:
/CMC/kits/cmosp18/VGdir/SRAM/hdss1_1024x32cm8/syn/hdss1_1024x32cm8_tc.d
b)
    vst_n18_sc_tsm_c4_typ (File:
/CMC/kits/cmosp18/synopsys/2002/syn/vst_n18_sc_tsm_c4_typ.db)
    tpz973gtc (File: /CMC/kits/cmosp18/synopsys/2002/syn/tpz973gtc.db)

Number of ports:              27
Number of nets:               54
Number of cells:              28
Number of references:          3

Combinational area:       365109.375000
Noncombinational area:    578619.937500
Net Interconnect area:      undefined  (Wire load has zero net area)

Total cell area:          943754.312500
```

```
Total area:                 undefined

Information: This design contains black box (unknown) components. (RPT-
8)
1
design_analyzer> report_constraints

****************************************
Report : constraint
Design : topMAPchp2
Version: 2003.06
Date   : Fri Aug 20 13:52:41 2004
****************************************


                                                  Weighted
    Group (max_delay/setup)       Cost     Weight     Cost
    ---------------------------------------------------
    CLK                           0.00      1.00      0.00
    default                       0.00      1.00      0.00
    ---------------------------------------------------
    max_delay/setup                                   0.00

                             Total Neg  Critical
    Group (critical_range)     Slack    Endpoints   Cost
    ---------------------------------------------------
    CLK                           0.00         0      0.00
    default                       0.00         0      0.00
    ---------------------------------------------------
    critical_range                                    0.00

                                                  Weighted
    Group (min_delay/hold)        Cost     Weight     Cost
    ---------------------------------------------------
    CLK (no fix_hold)             0.00      1.00      0.00
    default                       0.00      1.00      0.00
    ---------------------------------------------------
    min_delay/hold                                    0.00


    Constraint                                        Cost
    ---------------------------------------------------
    multiport_net                                     0.00 (MET)
    max_transition                                    0.00 (MET)
    max_fanout                                        0.00 (MET)
    max_capacitance                                   0.00 (MET)
    max_delay/setup                                   0.00 (MET)
    critical_range                                    0.00 (MET)


1
design_analyzer> report_power
Performing power analysis through design. (low effort)
****************************************
Report : power
      -analysis_effort low
Design : topMAPchp2
```

```
Version: 2003.06
Date   : Fri Aug 20 13:57:12 2004
****************************************


Library(s) Used:

    hdss1_1024x32cm8_lib (File:
/CMC/kits/cmosp18/VGdir/SRAM/hdss1_1024x32cm8/syn/hdss1_1024x32cm8_tc.d
b)
    vst_n18_sc_tsm_c4_typ (File:
/CMC/kits/cmosp18/synopsys/2002/syn/vst_n18_sc_tsm_c4_typ.db)
    tpz973gtc (File: /CMC/kits/cmosp18/synopsys/2002/syn/tpz973gtc.db)


Operating Conditions: NCCOM   Library: tpz973gtc
Wire Load Model Mode: segmented

Design        Wire Load Model            Library
------------------------------------------------
topMAPchp2             TSMC128K_Conservative
                                         tpz973gtc
MAPdecoder2            TSMC64K_Conservative
                                         tpz973gtc


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  =  38.0972 mW   (79%)
  Net Switching Power  =  10.3057 mW   (21%)
                         ---------
Total Dynamic Power    =  48.4029 mW  (100%)

Cell Leakage Power     =  10.6374 uW

1
design_analyzer> report_timing -path full -delay max -max_paths 1 -
nworst 1

****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : topMAPchp2
Version: 2003.06
Date   : Fri Aug 20 13:57:19 2004
****************************************

Operating Conditions: NCCOM   Library: tpz973gtc
```

```
Wire Load Model Mode: segmented

  Startpoint: MAPdecoder2/alpha2/a14_reg_3_
              (falling edge-triggered flip-flop clocked by CLK)
  Endpoint: MAPdecoder2/alpha2/a4_reg_7_
            (falling edge-triggered flip-flop clocked by CLK)
  Path Group: CLK
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  topMAPchp2         TSMC128K_Conservative tpz973gtc
  MAPdecoder2        TSMC64K_Conservative  tpz973gtc

  Point                                                    Incr
Path
  ------------------------------------------------------------------------
-----
  clock CLK (fall edge)                                    4.00
4.00
  clock network delay (ideal)                              0.00
4.00
  MAPdecoder2/alpha2/a14_reg_3_/CKB (DFERSNB2)             0.00
4.00 f
  MAPdecoder2/alpha2/a14_reg_3_/Q (DFERSNB2)               0.41
4.41 r
  MAPdecoder2/U0_1_368/Z (NAN2D1)                          0.08
4.49 f
  MAPdecoder2/U1_4_0_441/Z (OAI21D1)                       0.26
4.75 r
  MAPdecoder2/U1_4_1_641/Z (AOI21D1)                       0.07
4.82 f
  MAPdecoder2/U848/Z (OA21D1)                              0.13
4.95 f
  MAPdecoder2/U0_5_741/Z (EXNOR2D2)                        0.28
5.23 r
  MAPdecoder2/U272/Z (INVD2)                               0.04
5.28 f
  MAPdecoder2/U2041/Z (NAN3D4)                             0.16
5.43 r
  MAPdecoder2/U2095/Z (NAN2D2)                             0.04
5.47 f
  MAPdecoder2/U34/Z (NAN2D2)                               0.10
5.57 r
  MAPdecoder2/U0_3_536/Z (INVD1)                           0.06
5.63 f
  MAPdecoder2/U0_2_536/Z (NOR2D1)                          0.16
5.79 r
  MAPdecoder2/U1_1_0_58/Z (OAI21D1)                        0.06
5.85 f
  MAPdecoder2/U1_1_1_78/Z (AOI21D1)                        0.16
6.01 r
  MAPdecoder2/U1_1_2_78/Z (OAI21D1)                        0.06
6.07 f
  MAPdecoder2/U0_128/Z (AO21D2)                            0.17
6.24 f
```

```
  MAPdecoder2/U300/Z (NAN2D2)                          0.11
6.35 r
  MAPdecoder2/U2196/Z (NAN2D4)                         0.04
6.39 f
  MAPdecoder2/U1962/Z (NAN2D4)                         0.12
6.51 r
  MAPdecoder2/U2187/Z (INVD4)                          0.03
6.54 f
  MAPdecoder2/U1964/Z (NAN2D4)                         0.09
6.63 r
  MAPdecoder2/U2168/Z (INVD7)                          0.05
6.68 f
  MAPdecoder2/U2067/Z (NAN2M1D2)                       0.09
6.77 r
  MAPdecoder2/U1886/Z (NAN2D4)                         0.07
6.84 f
  MAPdecoder2/U0_3_340/Z (INVD1)                       0.12
6.95 r
  MAPdecoder2/U0_2_340/Z (NOR2D2)                      0.04
6.99 f
  MAPdecoder2/U1_2_0_340/Z (NOR2D1)                    0.14
7.14 r
  MAPdecoder2/U1_1_1_312/Z (AOI21D1)                   0.08
7.22 f
  MAPdecoder2/U1_1_2_712/Z (OAI21D1)                   0.15
7.37 r
  MAPdecoder2/U0_1212/Z (AO21D2)                       0.12
7.49 r
  MAPdecoder2/U1973/Z (INVD4)                          0.04
7.53 f
  MAPdecoder2/U2191/Z (NAN2D4)                         0.06
7.59 r
  MAPdecoder2/U2139/Z (NAN2M1D2)                       0.05
7.64 f
  MAPdecoder2/U1974/Z (NAN2D4)                         0.11
7.75 r
  MAPdecoder2/U1722/Z (INVD7)                          0.04
7.79 f
  MAPdecoder2/U2028/Z (NAN2D4)                         0.13
7.92 r
  MAPdecoder2/U1686/Z (INVD7)                          0.04
7.95 f
  MAPdecoder2/U1921/Z (NOR2D4)                         0.07
8.02 r
  MAPdecoder2/U2029/Z (INVD4)                          0.06
8.08 f
  MAPdecoder2/U2050/Z (NAN2D4)                         0.08
8.16 r
  MAPdecoder2/U105/Z (NAN3D2)                          0.10
8.26 f
  MAPdecoder2/U0_3_041/Z (INVD1)                       0.12
8.38 r
  MAPdecoder2/U0_1_041/Z (NAN2D1)                      0.05
8.42 f
  MAPdecoder2/U1_1_0_113/Z (OAI21D1)                   0.15
8.58 r
```

```
  MAPdecoder2/U1_1_1_313/Z (AOI21D1)                    0.07
8.65 f
  MAPdecoder2/U1_1_2_713/Z (OAI21D1)                    0.16
8.81 r
  MAPdecoder2/U0_1213/Z (AO21D2)                        0.12
8.93 r
  MAPdecoder2/U2054/Z (NAN3D4)                          0.06
8.99 f
  MAPdecoder2/U1781/Z (NAN3D2)                          0.12
9.11 r
  MAPdecoder2/U1780/Z (BUFDA)                           0.09
9.20 r
  MAPdecoder2/U1113/Z (OR2D2)                           0.08
9.29 r
  MAPdecoder2/U1896/Z (NAN3M1D2)                        0.07
9.35 f
  MAPdecoder2/U1_5_0_228/Z (NOR2D2)                     0.08
9.44 r
  MAPdecoder2/U1_4_1_228/Z (AOI21D2)                    0.17
9.61 f
  MAPdecoder2/U1_2_2_228/Z (INVD0)                      0.11
9.72 r
  MAPdecoder2/U1_2_3_228/Z (INVD2)                      0.03
9.75 f
  MAPdecoder2/U0_5_328/Z (EXOR2D4)                      0.34
10.09 f
  MAPdecoder2/U0_1_357/Z (NAN2D1)                       0.21
10.30 r
  MAPdecoder2/U1745/Z (OAI21D1)                         0.07
10.37 f
  MAPdecoder2/U1_4_1_632/Z (AOI21D1)                    0.16
10.53 r
  MAPdecoder2/U1494/Z (OAI21D1)                         0.06
10.60 f
  MAPdecoder2/U498/Z (EXNOR2D2)                         0.30
10.89 r
  MAPdecoder2/U1260/Z (OR2D0)                           0.20
11.09 r
  MAPdecoder2/U2181/Z (NAN2D4)                          0.03
11.12 f
  MAPdecoder2/U727/Z (NAN2D1)                           0.07
11.19 r
  MAPdecoder2/U726/Z (NAN2D1)                           0.05
11.24 f
  MAPdecoder2/alpha2/a4_reg_7_/D (DFERSNB1)             0.00
11.24 f
  data arrival time
11.24

  clock CLK (fall edge)                                12.00
12.00
  clock network delay (ideal)                           0.00
12.00
  clock uncertainty                                    -0.50
11.50
  MAPdecoder2/alpha2/a4_reg_7_/CKB (DFERSNB1)           0.00
11.50 f
```

```
  library setup time                                        -0.26
11.24
  data required time
11.24
  ---------------------------------------------------------------------
-----
  data required time
11.24
  data arrival time                                                   -
11.24
  ---------------------------------------------------------------------
-----
  slack (MET)      0.00
```

VITA AUCTORIS

NAME	Leila Sabeti
PLACE OF BIRTH	Tehran, Iran
YEAR OF BIRTH	1977
EDUCATION	Narges High School, Tehran 1990-1995
	Shahid Beheshti (Melli) University, Tehran, Iran 1995-2000 B.Sc.
	University of Windsor, Windsor, Ontario 2003-2004 M.Sc.
AWARDS	2004 University of Windsor Visa Differential Fee Bursary