1999

RTL design and performance analysis of near-optimum turbo codec

Yi Wang
Lehigh University

Wang, Yi

RTL Design and Performance Analysis of Near-optimum Turbo Codec

January 2000

RTL Design and Performance Analysis of Near-optimum Turbo Codec

by

Yi Wang

A Thesis

Presented to the Graduate and Research Committee

Of Lehigh University

in Candidacy for the Degree of

Master of Science

In

Computer Engineering

Lehigh University

December 1, 1999

This thesis is accepted and approved in partial fulfillment of the requirements for the Master of Science

Dec. 6, 1999

Date

Thesis Advisor

Chairperson of Department

Acknowledgements

I would like to express my gratitude to my advisor Dr. Frank Hielscher for his kind guidance and support during my research work and my master courses study. With his instruction both inside and outside classes, I was highly motivated in this project. This thesis was made possible with his encouragement and help.

I would also like to thank my husband, Zhenyu Zhu, for his continuous support and assistance during the whole process. His knowledge of communication and coding theory helped me with a better understanding of the turbo coding algorithm.

Contents

Abstract

Turbo codes is the most exciting and potentially important development in coding theory in recent years and has opened a whole new way of looking at the problem of constructing good codes and decoding them with low complexity. It is claimed that the codes can achieve near-Shannon-limit error correction performance, which is the theoretical limit, with relatively simple component codes and large interleavers. However, some important details that are necessary to reproduce these results were omitted in the theoretical derivation, as well as those factors related to VLSI implementation.

In this thesis, a pipelined and scalable architecture of the turbo codec ASIC was proposed to maximize the data throughput and minimize the system delay. The theoretical background of turbo codes is discussed, as well as the detailed VLSI implementation with Verilog hardware description language. To make the algorithm feasible for hardware implementation, some modifications and simplifications were made to the original floating-point theory. The new algorithm is then modeled and simulated using Verilog. Based on simulation results, the performance of the fixed-point turbo codec can achieve very high quality with low bit error rate, high throughput and minimal delay. All the parameters and specification of the design were selected to conform to the commercial wireless communication standards. The modular design also make the architecture scalable so that different applications can use different numbers of the basic core to achieve various communication requirements.

Chapter 1

Introduction

Turbo Codes, developed in the early 1990s at ENST (École Nationale Supériere des Télécommunications de Bretagne) in Brest [1] , France, are a class of iterative decoding algorithms that provide error-correction performance near channel capacity, which is the theoretical limit. The data stream is separated into blocks that are encoded twice with a convolutional encoder: once in a traditional fashion and once after the data order has been scrambled with a random interleaver. The data stream is transmitted with the parity bits generated by both encoders. In the receiver the data is decoded using the transmitted data and the uninterleaved parity stream to produce soft-decision correction metrics. The correction metrics and received data are then interleaved and decoded a second time using the transmitted interleaved parity bits. This process is repeated iteratively to provide near optimal decoding performance.

Coding theorists have traditionally attacked the problem of designing good codes by developing codes with a lot of structure [2], which lends itself to feasible decoders, although

coding theory suggests that codes chosen "at random" should perform well if their block size is large enough. The challenge to practical decoders for "almost" random, large codes has not been seriously considered until recently. Perhaps the most exciting and potentially important development in coding theory in recent years has been the dramatic announcement of "turbo codes". The announced performance of these codes was so good that the initial reaction by the coding establishment was met with deep skepticism, but recently researchers around the world have been able to reproduce those results [3,4]. The introduction of turbo codes has opened a whole new way of looking at the problem of constructing good codes and decoding them with low complexity.

Claude Shannon defined the capacity of a channel to be:

$$C = W \log_2(1 + \frac{E_s}{N_0}) \; bits \; per \; \text{sec} ond$$

Where W is the bandwidth of the channel in Hertz, C is the channel capacity in bits per second, and E_s/N_0 is the signal to noise ratio. He also provided the Noisy Channel Coding Theorem: *Consider an additive white Gaussian noise channel with capacity C. There exist error control codes such that information can be transmitted across the channel at rates less than C with arbitrarily low word error rate* [2].

Within 50 years after Shannon's theorem, a gap of 2dB continued to separate the performance of the most advanced error control systems from the theoretical limit. This gap vanished overnight with the advent of turbo coding. It is claimed in [1] that turbo codes achieve near-Shannon-limit error correction performance with relatively simple component codes and large

interleavers. For a bit error probability of 10^{-5}, the performance is approximately 0.5dB away from capacity.

In theory, turbo codes are very high-performance error-correcting codes which can be used in many modern communication systems. However, it is still a new and challenging area for real VLSI implementation. Due to the high computational complexity of the statistical operations, and the comparative long delay related to the codes, it seems formidable to implement turbo codes in hardware.

This thesis aims to develop a hardware version of a turbo codec via an Application Specific Integrated Circuit (ASIC) to verify the performance of the code in a real-time, fixed-point environment. The ASIC design is based on the original theory, with appropriate modification and simplification to optimize the real time operation performance. The Verilog hardware description language was used to model the design, and simulation was done using Modeltech's simulations program (VSIM). The performance of different aspects including quality, delay, data rate, as well as the hardware complexity were analyzed. Based on the final simulation result, the design shows promise in the current wireless communication area. The parameters of the codec were selected based on the current wireless standards, such as that of the global system for mobile communication (GSM). GSM is a globally accepted standard for digital cellular communication. GSM is the name of a standardization group established in 1982 to create a common European mobile telephone standard that would formulate specifications for a pan-European mobile cellular radio system operating at 900 MHz.

1.1. Literature Review

This project was based on the paper "Near Optimum Error Correcting Coding And Decoding: Turbo-Codes" by C. Berrou and Alain Glavieux [1] with regard to later achievements on the area of Turbo Codes, and thus tries to evaluate and position the achieved results with a more realistic "hardware" perspective. Although Turbo Codes are still a very young topic, significant progress has been achieved during the last two years.

The above paper introduces the principle of concatenating two channel coding schemes, with a specific focus on the concatenation of two convolutional codes. A well understandable practical motivation behind the idea of Turbo Codes are the improvements in the performance of channel codes, especially for low signal to noise ratio (SNR).

Such improvements could for instance be achieved by using very long codes and soft decision decoding. In the construction of the long codes it is, due to the low SNR, necessary to take the complete weight spectrum of the code into account and not only the minimum free distance of the code. But long channel codes have serious drawbacks, namely:

1. The coding gain (the difference between the SNR required to achieve a given bit error rate in a coding system and the SNR required to achieved the same BER in an uncoded system) increases only linearly with code memory.
2. Decoder complexity grows exponentially with code memory.

For practical reasons, i.e. cheap and simple implementations, the concatenation of two channel codes was proposed. The first implementations were developed as early as 1993 by the same authors [1].

Hagenauer [2] calls this method of concatenating two channel codes the "turbo principle", because of the similarity with the "turboprop engine", i.e. the cascading of two engines to increase their efficiency. The functionality of turbo codes can be described as:

1. Encoding of the source bit stream with two channel codes;
2. Iterative decoding of the two codes;
3. During each iteration the two decoders take advantage of the a-posteriori probabilities obtained from the previous decoding step in a feed back loop manner;
4. The turbo decoder therefore outputs the Maximum A-Posteriori (MAP) estimation of the received code words (other methods took advantage of extrinsic information).

Hagenauer's paper [4] focuses mainly on how to improve the performance of turbo codes, and especially two methods were discussed:

1. The use of recursive systematic convolutional codes;
2. Interleaving of data between the two encoding steps.

It considers only the parallel concatenation of convolutional codes. Several simulations and a final evaluation of the achieved results conclude their paper.

1.2. Outline of the Thesis

In this thesis, the theoretical algorithm described in the paper "Near Optimum Error Correcting Coding and Decoding: Turbo-Codes" by Claude Berrou and Alain Glavieux, was analyzed, and was then modified to make it feasible for ASIC development.

Chapter 2 analyzes the theoretical aspect of turbo codes. The basic structures of both the encoder and decoder are introduced. The algorithm is analyzed and partitioned into several functional blocks. The detailed system design of encoder, decoder and interleaver are presented with the appropriate modification and simplification.

Chapter 3 gives the hardware implementation of the whole system modeled in Verilog. The entire data path, computational units, timing and control signals are shown at the Register Transfer Level (RTL). The dedicated pipelined and scalable architecture are presented.

Chapter 4 provides the simulation scheme for each module, as well as for the entire system. MATLAB was used to model the additive white Gaussian noise (AWGN). Using the simulation results; several performance criteria were analyzed, including bit error rate, data rate, system delay and hardware complexity. It is shown that the system can achieve the requirements of most of the modern wireless communication specification such as global system for mobile communication (GSM).

Chapter 5 summarizes the hardware turbo codec system and its features. Some promising areas of future exploration are discussed.

Chapter 2

Algorithm and Structure of Turbo Codes

2.1. Structure of Digital Communication Systems

Figure 2.1 shows a one-way system, in which the transmission is strictly in the forward direction, from the transmitter to the receiver. The information source is the signal to be transmitted, such as computer data, images, speech etc. It must be translated into a set of signals optimized for the channel over which we want to send it, and it usually contains redundancy. The source decoder is used to remove the redundant part so as to maximize the information transmission rate. Since a noisy communication channel will introduce perturbations and misinterpretation of the transmitted message at the receiving end, error control strategies must be taken in order to correct errors at the receiving end. This is achieved by the channel encoder and decoder. The channel encoder purposely adds redundancy into the information sequence in a controlled manner, which allows the decoder at the receiver end to detect errors and thus increases transmission reliability. After the channel encoder, the modulator transforms the binary bits into a continuous-time analog waveform for transmission. This waveform is sent over the physical channel.

At the receiver end, the digital demodulator processes the corrupted waveform and produces the estimation of the transmitted data. The output of the demodulator is passed to the channel decoder, which uses the redundancy and the knowledge of the channel code to detect and correct errors added by the physical channel. Finally , the source decoder accepts the decoded bits and attempts to reconstruct the original information source with the knowledge of the source encoding method.

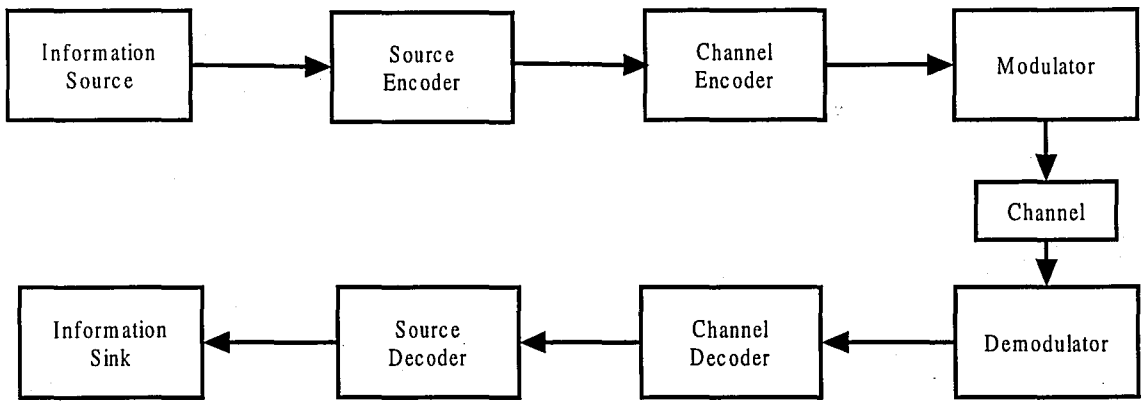

Figure 2.1: Block Diagram of a Communication System.

In this paper, the channel encoding and decoding pair is our concern. We will discuss the algorithm and implementation of turbo codes, the most advanced forward error correction technique.

2.2. Turbo Codes

In contrast to a two-way system which can use ARQ (automatic repeat request) with error detection and retransmission, the error control strategy for a one-way system must be FEC (forward error correction), which automatically corrects errors detected at the receiver. FEC includes block codes, convolutional codes as well as concatenated codes. A major breakthrough in coding systems was made by Berrou et al. in 1993 with the discovery of "turbo codes". Turbo codes outperforms all the previous FEC coding techniques and achieves performance close to the Shannon limit. For a bit error probability of 10^{-5}, it performs about 0.5 dB away from capacity.

Turbo codes implies (i.e. is synonymous with) parallel concatenated convolutional codes (PCCC). It contains two or more recursive convolutional codes, a pseudorandom interleaver and a MAP iterative decoding algorithm.

Turbo codes has several features which lead to its outstanding performance:

1. Soft in Soft out (SISO)

The demodulator in Fig2.1 has two ways of making a decision, soft decision or hard decision. A decision is hard when the demodulator compares the incoming value to a predetermined threshold. A soft decision is when the demodulator gives an indication of the probability of the received sample being a 0 or a 1. In the traditional approach, the demodulator block makes a hard decision of the received symbol and passes it to the error control decoder block. With the new SISO algorithms, additional information is passed from the output of demodulator to the input of the decoder and from the output of one decoder to the input of the next decoder.

2. Interleaver

Turbo codes use an interleaver and encode the same information twice, but in a different order. This allows the decoder to correct the bits in two dimensions and can reduce burst error (the continuous errors occurred in a very short period of time).

3. Iterative decoding

Turbo codes decode in an iterative manner. The soft output decision algorithm provides a real number as an output, which is a measure of the probability of an error in decoding a particular bit. This can also be interpreted as a measure of the reliability of the decoder's hard decision. This extra information is very important and is used for the next stage in an iterative decoding process.

The conventional error correcting codes, like block codes and convolutional codes, can achieve bit error rates (BER) of 10^{-3} at a SNR around 3 dB. The current BER requirement of GSM is 10^{-3} for voice and 10^{-5} for data. At a 3dB noise environment, turbo codes can attain a BER of 10^{-5}, which can satisfy the high quality requirement of GSM. But in order to get the full lossless data transmission, some higher level processing is still needed, like outer error-correcting (e.g. Reed Solomn) codes and feedback handshaking (e.g. ARQ). For error free data transmission, the extra protection is always needed, but these kinds of techniques are very expensive and not efficient. They will reduce the overall data rate and coding gain. Therefore, we should try to avoid using them or use them as little as possible, which is the reason why we need high performance Forward Error Correcting coding (FEC). The more advanced type of FEC will result in less error and will require less complex higher-level processing. The overall efficiency and the system cost highly depends on the FEC. So far, turbo codes is the best FEC which performs close to the theoretical limits.

2.3. Encoder Algorithm

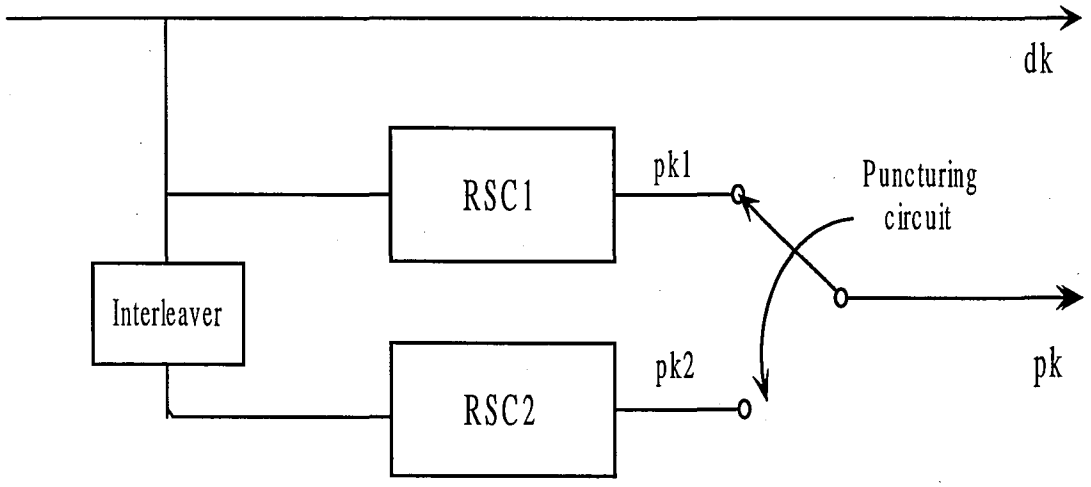

Figure 2.2: Generic Turbo Encoder

Figure 2.2. depicts a standard turbo encoder. In this figure, dk is the incoming data stream, RSC is the recursive systematic coder, and pk is the generated parity signal. As seen in the figure, a turbo encoder consists of two binary rate 1/2 convolutional encoders separated by an N-bit interleaver together with an optional puncturing mechanism (a selector switch). Clearly, without the puncturer, the encoder is rate 1/3, mapping N data bits to 3N code bits. The encoders are configured in a manner reminiscent of classical concatenated codes. Two identical convolutional encoders are arranged in a so-called parallel concatenation. Before describing further details of the turbo encoder in its entirety, we shall first discuss its individual components.

A. The Systematic Convolutional Encoder

Convolutional codes are a family of error correcting codes, and they are of the recursive systematic variety. They add redundant information based on the input data and on the previous state of the encoder. The encoder can be thought of as having memory. Each code has three main features:

- a memory length corresponding to the number of delay elements which we call v.
- a constraint length equal to $(v+1)$.
- a coding rate. This is the ratio between the number of information bits we encode and the number of bits after the encoding (including the redundant bits).

An example of a convolutional encoder is shown in Fig. 2.3.

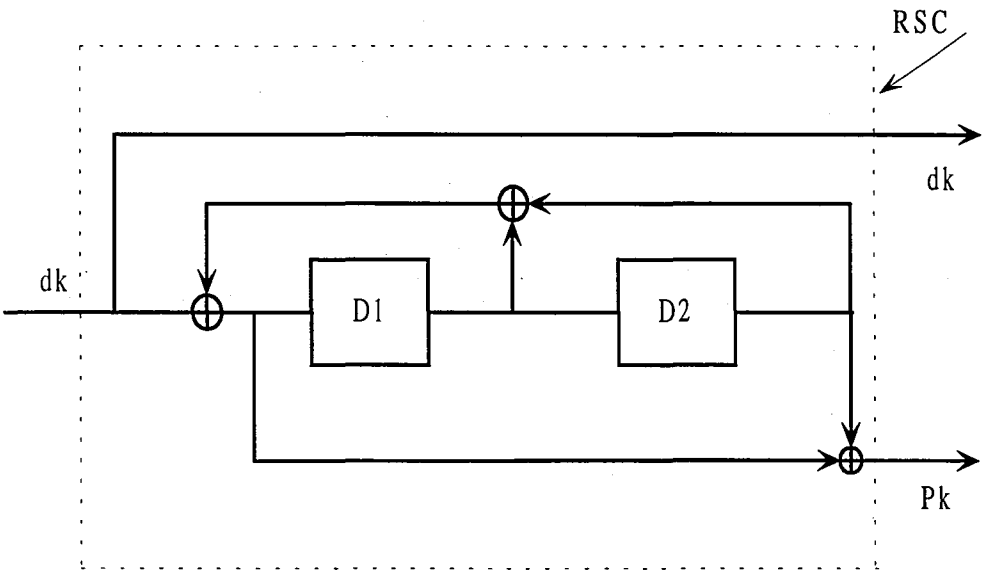

Figure 2.3: 1/2 rate Systematic Convolutional Encoder

In this Figure, D1 and D2 are two storage elements. This encoder has a rate of ½ and a memory length v=2, so the number of encoder states equal to 2v=4. The convolutional code is systematic because the information bits dk are directly written to the output. Turbo codes use

systematic convolutional codes because they perform better than non systematic codes at low signal to noise ratio.

B. Interleaver

The function of the interleaver is to take each incoming block of N data bits and rearrange them in a pseudo-random fashion prior to encoding by the second encoder. The interleaver design is a key factor which determines the good performance of a turbo code. Some interleaver types used in turbo codes are block interleaver and pseudo-random interleaver.

The main purpose of the interleaver is to increase the minimum distance of the turbo codes such that after the correction in one dimension the remaining errors should become correctable error patterns in the second dimension.

The performance of turbo codes using iterative decoding algorithms depends on the structure and length of the interleaver used. The different kinds of interleaver affect the distance property of the resulting turbo code. In the previous literature [6], block interleavers and pseudo-random interleavers were investigated. The pseudo-random interleavers were found to give better performance only for larger interleaver lengths. These larger interleavers can be used in deep-space communications for which the decoding delay is not so important. But for wireless communication, the length of interleaver is limited by the maximum acceptable speech delay. For the smaller interleaver size, the block interleavers perform better. Therefore, in my design, I selected the non-uniform block interleaver proposed by Claude Berrou and Alain Glavieux [1] with a block size of 256.

This design chooses an interleaving procedure in which, for reading, the column index is a function of the line index. Let i and j be the addresses of line and column for writing, and ir and jr the addresses of line and column for reading. For an M*M memory (M being a power of two, here it is 16), i,j,ir,jr have values between 0 and M-1. Nonuniform interleaving may be described by:

$$ir=(M/2+1)(i+j) \bmod M$$

$$\xi=(i+j) \bmod 8$$

$$jr=[P(\xi)(j+1)]-1 \bmod M$$

P(.) is a number, relatively prime with M, which is a function of line address (i+j) mod 8. Note that reading is performed diagonally in order to avoid possible effects of a relation between M and the period of puncturing. A multiplying factor (M/2+1) is used to prevent two neighboring data written on two consecutive lines from remaining neighbors upon reading.

Theoretically, the interleaver is made up of an M*M matrix and bits {dk} are written in row by row and read out following the non-uniform rule given above. This non-uniform reading procedure is able to spread the residual continuous burst error blocks and gives a large free distance to the concatenated code.

For the hardware implementation a 16*16 matrix has been used, and from above, the addresses of line ir and column jr for reading are the following:

$$ir= 9*(i+j) \bmod 8$$

$$\xi= (i+j) \bmod 8$$

$$jr= [P(\xi)(j+1)]-1 \bmod 8$$

with P(0)=17; P(1)=37; P(2)=19; P(3)=29; P(4)=41; P(5)=23; P(6)=13; P(7)=7. These P values are arbitrarily selected but all the values are relative prime with M=16.

C. The Puncturer

The puncturer is used to achieve higher coding rates. In Fig.2.2, the parity bits from the two parallel RSCs can be "punctured" by a multiplexing switch and thus the coding rate is 1/2. The same puncturer will be used in the decoder to dispatch the parity bits into the corresponding MAP (the maximum *a posteriori*) decoders.

2.4. Decoder

When the maximum likelihood decoding algorithm is applied to the encoder trellis structure, the result is the optimum decoding of turbo codes. However, due to the interleaver embedded in the encoder, the turbo code trellis will have an extremely large number of states. This fact makes the maximum likelihood decoding process almost impossible, in practice, for large interleaver sizes. Thus, practically, turbo codes use an iterative decoding approach where the maximum likelihood decoding algorithm is applied to the elementary convolutional codes. This iterative technique is a very efficient way to decode turbo codes and to achieve performance close to the theoretical limits.

2.4.1. Turbo Decoder Structure

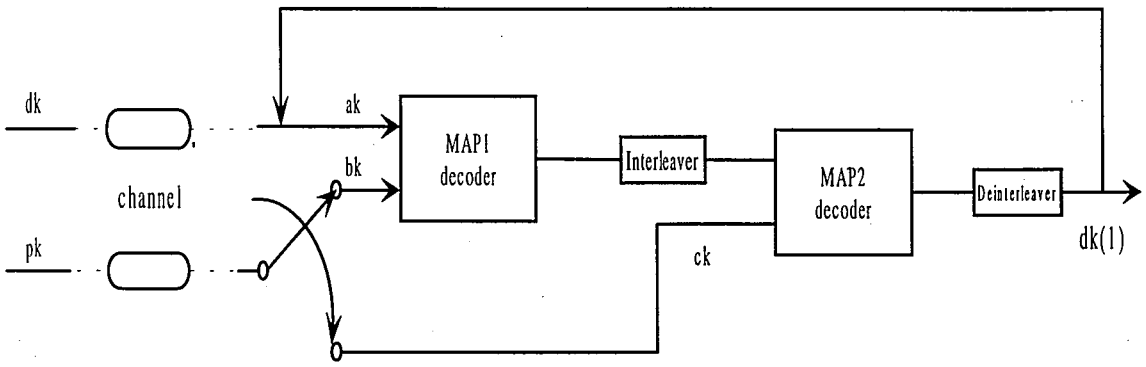

Figure 2.4: Turbo Code Decoder

Figure 2.4 shows the basic structure of the iterative turbo decoder. The output of the turbo encoder are the information sequence dk and punctured parity sequence pk. After the modulation and transmission over the physical channel, the received signals at the turbo decoder are the noisy information sequence ak and the noisy parity sequence pk. Through a mutiplexer switch, pk will be switched to produce the two noisy parity sequences bk and ck corresponding to the two RSC encoder outputs. As shown in the figure, a single iteration is employed using two soft decision MAP decoders. The first MAP decoder provides a soft output which is a measure of the reliability of each decoded bit. From this reliability information, the extrinsic information is produced, which does not depend on the current inputs to the decoder. This extrinsic information, after interleaving, is passed on to the second MAP decoder which uses this information to decode the interleaved bit sequence. The output of the second MAP decoder goes through the deinterleaver and generates the output sequence of the single iteration decoding: dk(1). This soft output can be fed back to the input of the first decoder to be decoded.

The performance of a turbo coding scheme improves as the number of decoder iterations is increased, where each decoding iteration involves two decoding stages. Here, the maximum *a posteriori* (MAP) algorithm is used to implement the decoder.

2.4.2. MAP algorithm

1. Principle

At the input of the MAP decoder we receive the noisy sequence $R_{1,N}=(R_1,R_2,.....,R_N)$, where $R_k=(a_k,b_k)$.

The purpose of the MAP decoding algorithm is to find the most likely dk for a given $R_{1,N}$. This is achieved by the corresponding log-likelihood ratio (LLR) Λ_k defined as follows:

$$\Lambda_k(d_k) = \log(\frac{p_r(d_k = 1 \mid R_{1,N})}{p_r(d_k = 0 \mid R_{1,N})}) \tag{2.1}$$

where the P_r's are the probabilities of the data bits d_k being either a 0 or a 1.

The values of $\Lambda_k(d_k)$ represent a soft output. To make a hard decision, we must decide what was the most likely information bit of the transmission.

$$if\ \Lambda_k(d_k) \geq 0 \rightarrow d_k = 1; \tag{2.2}$$

$$if\ \Lambda_k(d_k) < 0 \rightarrow d_k = 0. \tag{2.3}$$

If S_k is the state of the encoder at the time k, we have

$$P_r(d_k = l \mid R_{l,N}) = \sum_{m=0}^{2^v-1} P_r(d_k = l, S_k = m \mid R_{l,N})$$

(2.4)

If we let $\lambda_{k,i}(m) = P_r(d_k=i, S_k=m \mid R_{1,N})$ and we sum over all possible states of the encoder, we have

$$\Lambda_k = \log\left[\frac{\sum_{m=0}^{2v-1} \lambda_{k,1}(m)}{\sum_{m=0}^{2v-1} \lambda_{k,0}(m)}\right]$$

(2.5)

2. Expression of $\alpha_{k,i}(m)$, $\beta_{k,i}(m)$ and $\delta_I(R_k, m)$

To compute each Λ_k, we must define three very important parameters in the MAP decoding algorithm: $\alpha_{k,i}(m)$, $\beta_{k,i}(m)$ and $\delta_I(R_k, m)$. Each branch of the trellis will have particular values of $\alpha_{k,i}(m)$, $\beta_{k,i}(m)$ and $\delta_I(R_k, m)$.

Defining

$$\alpha_{k,i}(m) = P_r(d_k = i, S_k = m, R_{1,k})$$

(2.6)

and

$$\beta_{k,i}(m) = P_r(d_k = i, S_k = m, R_{1,k})$$

(2.7)

It is shown in [1] (a simplification of the Modie Bahl decoding) that we can express $\lambda_{k,I}$ by

$$\lambda_{k,i} = \frac{\alpha_{k,i}(m)\beta_{k,i}(m)}{P_r(R_{1,N})}$$

(2.8)

Finally, we can express Λ_k by

$$\Lambda_k = \log \frac{\sum_m \alpha_{k,1}(m)\beta_{k,1}(m)}{\sum_m \alpha_{k,0}(m)\beta_{k,0}(m)} \tag{2.9}$$

We now know the theoretical definitions of α and β. The state matrices α and β can be computed recursively as shown later. But first we need to address the expression for $\delta_i(R_k,m)$.

The expression $\delta_i(R_k,m)$ is called the branch metric and depends on the transmission channel. On the trellis, $\delta_i(R_k,m)$ is the branch metric corresponding to the branch of the transitions from time k to (k+1) with an initial encoder state m and an information bit i. The representation of $\delta_i(R_k,m)$ on the trellis is shown in Figure 2.5. All the branches of the trellis between time k and time (k+1) use the noisy data (a_k,b_k) that we received at the inputs of the MAP decoder. Each branch has its own value $\delta_i(R_k,m)$. For an additional white Gaussian Noise (AWGN) channel with mean zero and variance σ^2, $\delta_i(R_k,m)$ can be computed as:

$$\delta_i(R_k,m) = \exp\left(\frac{2}{\sigma^2}(a_k i + b_k P_{k,i}(m))\right) \tag{2.10}$$

The expression of $\alpha_{k,i}(m)$ is called the forward state metric, which represents the state metric for the transition from state m to the next state, at time k and with a transition bit of i. It is shown in [1] that we can compute $\alpha_{k,i}(m)$ with the recursive formula:

$$\alpha_{k,i}(m) = \delta_i(R_k,m)\sum_{j=0}^{1}\alpha_{k-1,j}(s_{b,j}(m)) \tag{2.11}$$

where $\delta i(R_k,m)$ is a branch metric defined as before and $S_{b,j}(m)$ is the state before m with a transition of j. All the α's can be computed from equation (2.11) on the condition that they are initialized correctly.

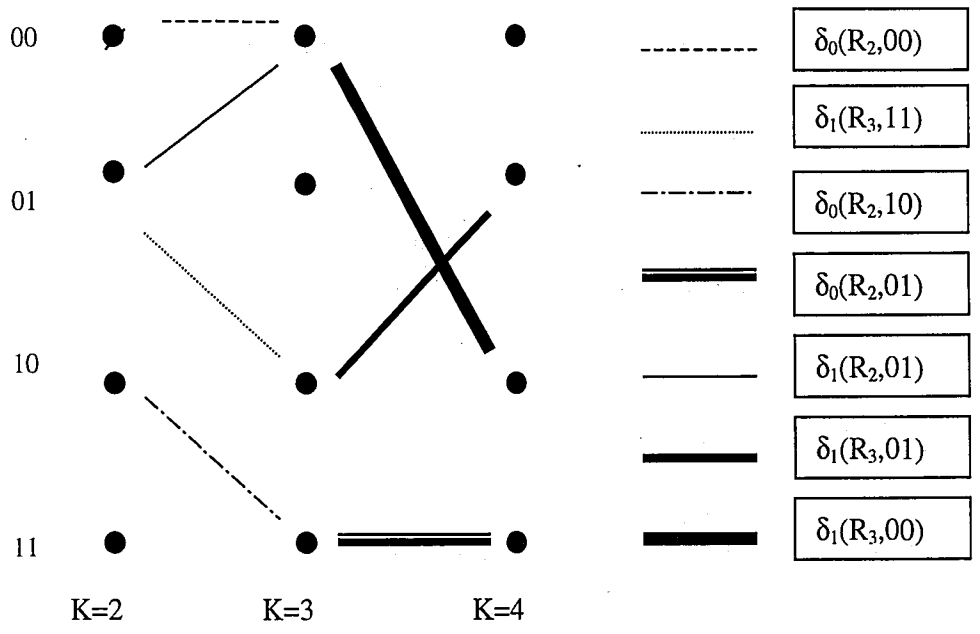

Figure 2.5: Representation of $\delta i(R_k,m)$ on the trellis

The expression for $\beta_{k,i}(m)$ is very similar to the expression for α except we compute it backward in time. This is why the $\beta_{k,i}(m)$ are called the backward state metrics. They represent the value of β for the transition from the state m, at time k and with a transition bit of i. It is shown in [1] that they can be computed by the recursive formula

$$\beta_{k,i}(m) = \sum_{j=0}^{1} \beta_{k+1,j}(s_{f,i}(m))\delta j(R_{k+1}, S_{f,i}(m)) \tag{2.12}$$

where $\delta i(R_{k+1},S_{f,i}(m))$ is a branch metric defined as before and $S_{f,i}(m)$ is the state after m with a transition of j. We notice that two values of β corresponding to the two paths coming into the same node of the trellis are equal. All the β's can be computed from equation (2.12) on the condition that they are initialized correctly.

2.4.3. The Details of the MAP Algorithm

With α, β and δ defined, the steps of the decoding algorithm are :

(1) Starting at time k=1, compute $\delta i(R_k,m)$ for all received symbols and store in an array of size 2^nN (for the 2^n possible code symbols).

(2) Initialize α at time k=0 and compute α from time k=1 to k=N-v-1 and for each state $m \in \{0,\ldots,2v-1\}$ using equation (2.11).

(3) Initialize β at time k=(N+v-1) and compute β for each time k from k=(N+v-2) to k=0 and for each state $m \in \{0,\ldots,2v-1\}$ using equation(2.12).

(4) Having all the values of α, β and δ for all the branches of the trellis and for each time, we can compute Λ_k from time k=0 to k=N+v-1 using equation(2.9).

(5) For each time k, make a hard decision to define each value of dk as in equation(2.1).

We see that, using the equations for Λ_k, α, β and δ, we can find the most likely value of dk. However, looking at the equations, we find that each time we decode one sample (a_k,b_k), a considerable number of complicated computations are required such as division, multiplication, logarithm and exponential.

For example,

- for the calculation of α and β, we need to perform a $(2^v \text{ x} 2)$ multiplication;
- for the calculation of δ, we need to compute n exponentials and 2^n-1 multiplications;
- for the calculation of Λ_k, we need to compute 1 logarithm , 1 division and $2\text{x}2^v$ multiplications;

These complex operations require large computation times and memory and, of greater concern, hardware which is too complex or which may be impossible to implement. Thus, it is

desirable to avoid such complex operations. Next we will illustrate a modified algorithm which is more suitable for hardware implementation and allows better simulations.

2.4.4. Modification of the MAP algorithm

A simplification of the MAP algorithm is developed in [1] to improve the efficiency of software simulations.

A new function E is defined:

$$E(x, y) = -\log(e^{-x} + e^{-y}) \tag{2.13}$$

We define new state metrics $A_{k,i}(m)$, $B_{k,i}(m)$ and new branch metrics $D_j(R_k,m)$ as:

$$\begin{aligned} A_{k,i}(m) &= -\log \alpha_{k,i}(m) \\ B_{k,i}(m) &= -\log \beta_{k,i}(m) \end{aligned} \tag{2.14}$$

Thus we have

$$\begin{aligned} A_{k,i}(m) &= D_i(R_k, m) + \underset{j=0}{\overset{1}{E}} A_{k-1,j}(S_{b,j}(m)) \\ B_{k,i}(m) &= \underset{j=0}{\overset{1}{E}} B_{k+1,j}(S_{f,i}(m)) + D_j(R_{k+1}, S_{f,i}(m)) \end{aligned} \tag{2.15}$$

$$\begin{aligned} \Lambda k &= \log \frac{\sum_m \alpha_{k,1}(m)\beta_{k,1}(m)}{\sum_m \alpha_{k,0}(m)\beta_{k,0}(m)} \\ &= \underset{m=0}{\overset{2v-1}{E}} (A_{k,1}(m) + B_{k,1}(m)) - \underset{m=0}{\overset{2v-1}{E}} (A_{k,0}(m) + B_{k,0}(m)) \end{aligned} \tag{2.16}$$

By looking at equation (2.15) and (2.16), we can see that we have suppressed all the logarithms, exponentials and multiplications and the only remaining operations are additions, subtractions and E functions. In [1], we can see that the E functions can be approximated as min(x,y), and this can be easily achieved by using a comparator in hardware. In chapter 3, we will present the detailed hardware design using this modified MAP algorithm.

Chapter 3

Hardware Design of Turbo Codecs

In the previous chapter, the theory and algorithm of Turbo coding has been explored. In this chapter, we present our implementation of turbo codes. The design has been done in a top-down manner. First, the top-level scheme is designed based on the theoretical algorithm with the appropriate specifications like block length, decoder iteration times, encoder memory length, etc. It is decomposed to smaller functional modules, and then each functional module is modeled and implemented with Verilog codes in a bottom-up procedure.

The first section shows the encoder design. We illustrate the concatenation of the recursive systematic encoder and its state diagram. The second section details the design of the iterative decoder. We use a pipelined structure, with the decoder based on the simplified MAP algorithm. The interleaver design is shown separately in section 3.3 since it is used both in the encoder and the decoder.

3.1. Encoder

As illustrated in 2.3, the turbo encoder uses a recursive systematic encoder (RSC). There are two ways to concatenate two RSC coders, serial or parallel concatenation. In this design, we use parallel concatenation.

With parallel concatenation, the information sample is simultaneously applied to the inputs of each RSC encoder. The encoder generates a parity sequence and the information sequence at the same time. In our design shown in Fig. 3.1, we parallel concatenate two RSC encoders. The input sequence and the interleaved input sequence are fed into the inputs of the two RSCs simultaneously. This is achieved with the control circuit of the interleaver module. We will present the interleaver design in more detail in section 3.3. The input sequence has a block length of 256 bits. A delay module is implemented to ensure that the output information sequence and the output parity sequence are ready at the same clock edge. The puncturer of the encoder is implemented by a mutiplexer. The mutiplexer selects the two outputs of the RSCs, alternately, based on the alternate control signal. In our design, the coding rate of the encoder is 1/2. As seen in the encoder block diagram, the input to the encoder is a string of 256 bits of the input data dk_in, and the system clock. The encoder will output the information sequence dk_out, and the parity sequence pk, and at the same time raise the dready signal.

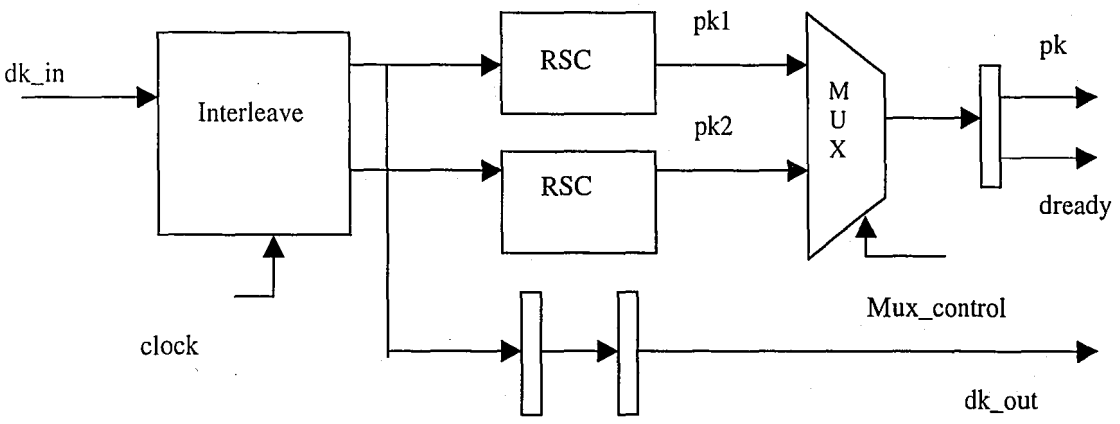

Figure 3.1: Block Diagram of Turbo Encoder

3.1.1. Recursive systematic encoder (RSC)

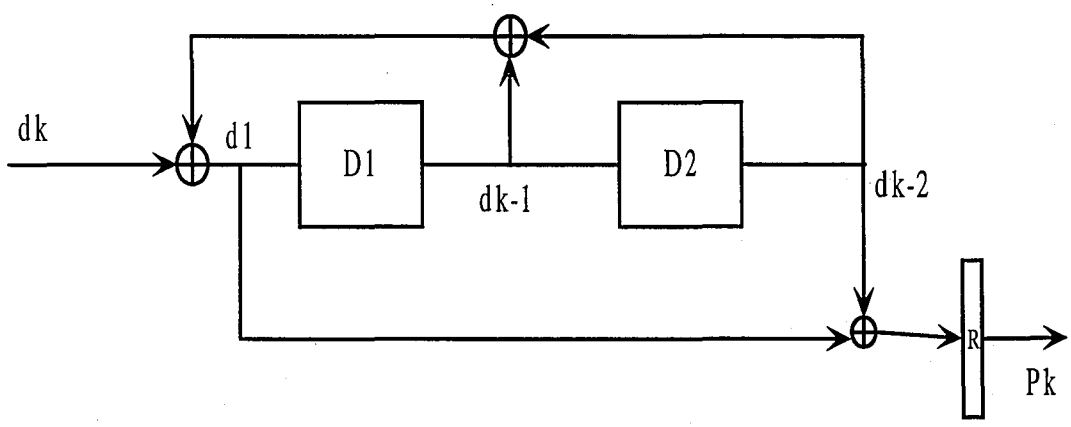

Figure 3.2: Recursive Systematic Coder (RSC)

The recursive systematic encoder used in this design is shown in Fig 3.2. Two serial storage elements D1 and D2 are used. The memory length is 2 and thus the RSC has four states. For this particular RSC, we have

$$d_1 = d_k \oplus d_{k-1} \oplus d_{k-2}$$
$$p_1 = d_1 \oplus d_{k-2}$$

At each positive edge of the clock, d1, d(k-1) and p1 will be latched to d(k-1),d(k-1) and pk. The state diagram of the RSC is shown as follows:

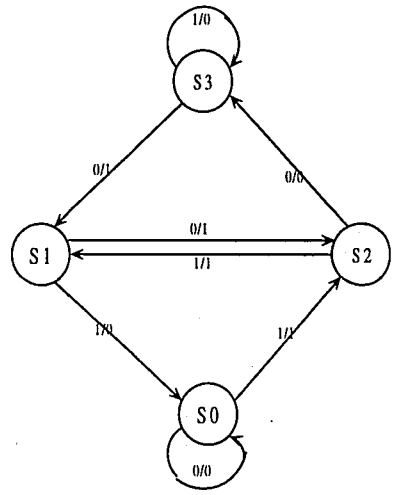

Figure 3.3: State Diagram of Turbo Encoder

3.2. Decoder

We have seen that turbo decoding is an example of iterative decoding. The MAP algorithm was based on the estimation of the probability of a decoded bit given the received sequence. It produces soft outputs which can be used in an iterative process to increase the reliability of the final decision. In this design, the soft input and output of the decoder are mapped onto an eight-bit bus, to represent the signed number ranged from –128 to 127. Dedicated functional modules are designed with structure-level Verilog models to handle the signed operations including addition, subtraction and comparison.

We used a pipelined structure to realize the reiteration and to improve the performance of the decoder output. Each stage of the decoder iteration is a parallel concatenation MAP decoder.

3.2.1. Pipeline Structure

There are two ways to realize the iterative turbo decoder algorithm.

1. To feed back the soft output of the decoder to the input of the decoder

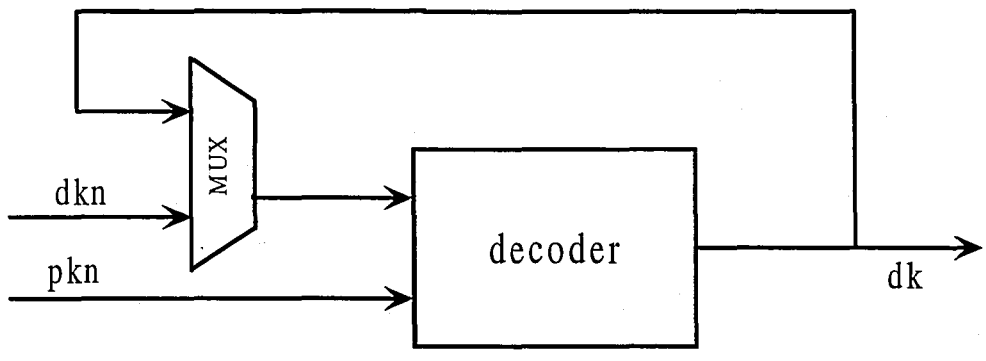

Figure 3.4: Iterative Turbo Decoder

As shown in figure 3.4, by use of the feedback loop, the iteration is realized with one decoder. This design uses less hardware. However, the data rate of the system will be decreased and control circuit will tend to be more complex. For n iterations, this structure will introduce n*2m delay for each input block, where m is the block length. This will reduce the data rate of the system by a factor of n compared with the structure proposed next. Thus, this implementation is not appropriate for real-time wireless communications.

2. To pipeline different stage of decoders

In this design, we are using the pipelined structure [5], shown in figure 3.5, where the same decoder design is used three times. The soft output of the first decoder is fed to the second decoder and the second decoder's output is fed to the third decoder. Thus we achieve three iterations. This serial concatenation structure allows the data to be pipelined and thus the data rate will be improved. The tradeoff is the need for additional hardware.

The noisy parity sequence pkn is fed into each decoder through delay lines. The noisy information sequence dkn is fed into the first decoder. The output of the first decoder is fed into the second decoder with the delayed parity sequence. Finally the soft output of decoder3 will be sent into a comparator, a hard decision will be made and the restored information sequence will be sent to the output. Different iterations can be achieved by cascading various numbers of decoders according to the requirements of specific applications.

In order to control the timing for the data flow, each stage of the decoder has an 'enable' signal and a 'dkready' signal. At the positive edge of the 'enable' signal, each decoder will start to read the input sequence. When each decoder starts to output data, it will raise the

'dkready' line. By connecting the 'dkready' line of each decoder to the 'enable' signal of its following decoder, we can ensure the timing correctness of the system.

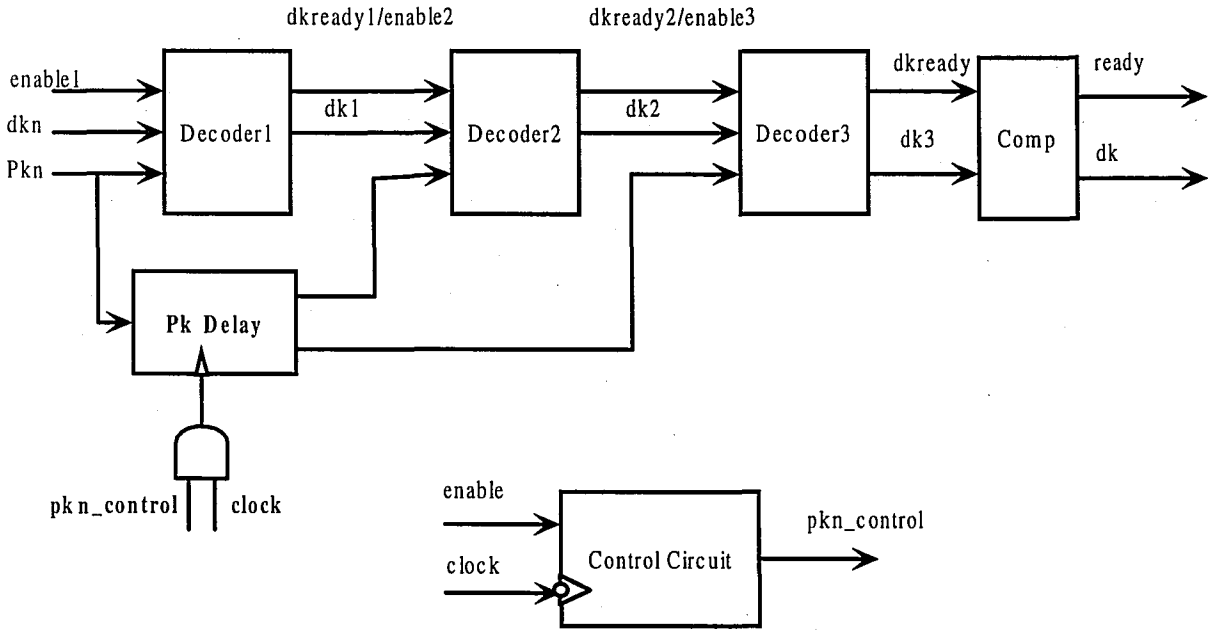

Figure 3.5: Circuit Scheme for Module Decoder

3.2.2. Parallel Concatenated Decoder

The basic scheme for each of the decoder stages is parallel concatenation, since we used a parallel concatenation encoder. The decoder is constructed by concatenating two MAP decoders. In this decoder, the noisy encoded parity sequence is distributed to each corresponding MAP decoder. Each MAP decoder output estimates dk. As the codes are systematic, we can send the output of the first MAP decoder to the input of the second MAP decoder. Since we have interleaved the data between two encoders so as to scatter errors and

improve performance, we need to interleave the data between the two MAP decoders and deinterleave the data to restore the sequence when necessary.

As stated in chapter 2, the module decoder1 is constructed with two MAP decoders, one interleaver and one deinterleaver. The input sequence dkn feed into submodule MAP1 through a buffer. The input parity sequence pkn will go through a punctuator, which is controlled by an alternate signal, and generates two sequences: bdk and cdk, corresponding to the two encoder's parity outputs. The bdk string will feed into MAP1 with the information sequence abk, while cdk will be delayed and then fed into MAP2 with the soft output of MAP1. In order to delay cdk, a 256-bit shift register triggered by ck_control is used.

Additionally, each module has an 'enable' input and a 'dkready' output. The preceding module's 'dkready' signal serves as the next module's 'enable'. At the positive edge of the 'enable' , the module starts to read data, initialize control signals and clear counters in the control circuit.

The implementation of MAP module, interleaver and deinterleaver will be discussed in more detail in the following sections.

3.2.3. MAP Decoder

According to our encoder structure, we design the MAP decoder based on the MAP algorithm discussed in chapter 2. The MAP decoder is to find the most likely dk, given the noisy information and parity sequence (a_k,b_k), while $k \in (1, N)$. N is the block length. In our design, N =256.

1. Compute the branch matrices $\delta i(R_k,m)$

In our encoder, with a memory length v=2, we have 4 states, so that for each time k there are 8 different branch matrices $\delta i(R_k,m)$. In equation (2.10), $P_{k,i}(m)$ is the encoder parity output when the input of the encoder is 'i' and the current state is 'm'. From the encoder's state diagram, the eight-branch metrics $\delta i(R_k,m)$ can be calculated as follows:

	$P_{k,I}(m)$		$\delta i(R_k,m)$	
Present State	i = 0	i = 1	i = 0	i = 1
S0	0	1	1	$Exp((2/\sigma^2) *(a_k+b_k))$
S1	1	0	$Exp((2/\sigma^2) *b_k)$	$Exp((2/\sigma^2) *a_k)$
S2	0	1	1	$Exp((2/\sigma^2) *(a_k+b_k))$
S3	1	0	$Exp((2/\sigma^2) *b_k)$	$Exp((2/\sigma^2) *a_k)$

We notice that there are two equal δ's for each δ. This is due to the fact that with a four-state code, we have four different values (dk,pk) for 8 branch metrics. This fact helps to simplify the design.

For a given encoder structure, all the trellis and state diagrams of the encoder are fixed, so the δ computation can be hardwired to improve the efficiency.

2. Computing the forward state metrics $\alpha_{k,i}(m)$

I. Derivation of $A_{k,i}(m)$

We can see that there are also 8 different branch matrices $\alpha_{k,i}(m)$ for each time k. In equation (2.11), $S_{b,j}(m)$ is the previous state number of the encoder when the encoder's current state number is m and previous input is i. According to the state diagram of our encoder, we find:

$S_{b,0}(0)=0;$ $S_{b,0}(1)=3;$ $S_{b,0}(2)=1;$ $S_{b,0}(3)=2;$

$S_{b,1}(0)=1;$ $S_{b,1}(1)=2;$ $S_{b,i}(2)=0;$ $S_{b,1}(3)=3;$

Substituting the branch metrics into equation (2.11), we can compute the 8 $\alpha_{k,i}(m)$ metrics:

$$\alpha_{k,1}(0) = [\alpha_{k-1,0}(0) + \alpha_{k-1,1}(1)] * \exp(\frac{2}{\sigma^2}(a_k + b_k))$$
$$\alpha_{k,1}(1) = [\alpha_{k-1,0}(3) + \alpha_{k-1,1}(2)] * \exp(\frac{2}{\sigma^2}a_k)$$
$$\alpha_{k,1}(2) = [\alpha_{k-1,0}(1) + \alpha_{k-1,1}(0)] * \exp(\frac{2}{\sigma^2}(a_k + b_k))$$
$$\alpha_{k,1}(3) = [\alpha_{k-1,0}(2) + \alpha_{k-1,1}(3)] * \exp(\frac{2}{\sigma^2}a_k)$$
$$\alpha_{k,0}(0) = [\alpha_{k-1,0}(0) + \alpha_{k-1,1}(1)]$$
$$\alpha_{k,0}(1) = [\alpha_{k-1,0}(3) + \alpha_{k-1,1}(2)] * \exp(\frac{2}{\sigma^2}b_k)$$
$$\alpha_{k,0}(2) = [\alpha_{k-1,0}(1) + \alpha_{k-1,1}(0)]$$
$$\alpha_{k,0}(3) = [\alpha_{k-1,0}(2) + \alpha_{k-1,1}(3)] * \exp(\frac{2}{\sigma^2}b_k) \quad (3.1)$$

We can see that the computation of $\alpha_{k,i}(m)$ includes operations like exponentials and multiplication. Here we applied the simplified MAP algorithm to avoid these operations. We defined the state metrics $A_{k,i}(m)$ by equation(2.14) and $A_{k,i}(m)$ is computed as follows.

$$
\begin{aligned}
A_{k,1}(0) &= E[A_{k-1,0}(0), A_{k-1,1}(1)] - \frac{2}{\sigma^2}(a_k + b_k) \\
A_{k,1}(1) &= E[A_{k-1,0}(3), A_{k-1,1}(2)] - \frac{2}{\sigma^2}a_k \\
A_{k,1}(2) &= E[A_{k-1,0}(1), A_{k-1,1}(0)] - \frac{2}{\sigma^2}(a_k + b_k) \\
A_{k,1}(3) &= E[A_{k-1,0}(2), A_{k-1,1}(3)] - \frac{2}{\sigma^2}a_k \\
A_{k,0}(0) &= E[A_{k-1,0}(0), A_{k-1,1}(1)] \\
A_{k,0}(1) &= E[A_{k-1,0}(3), A_{k-1,1}(2)] - \frac{2}{\sigma^2}b_k \\
A_{k,0}(2) &= E[A_{k-1,0}(1), A_{k-1,1}(0)] \\
A_{k,0}(3) &= E[A_{k-1,0}(2), A_{k-1,1}(3)] - \frac{2}{\sigma^2}b_k
\end{aligned}
\tag{3.2}
$$

II. The Ak module design

Looking at equation (3.2), we find that the complexity of the hardware for computing $A_{k,i}(m)$ has been greatly decreased. It only requires E operations and subtractions. As stated earlier, E operations can be achieved with a comparator. We noticed that $A_{k,i}(m)$ is computed in a recursive formula, so that the current $A_{k,i}(m)$ metrics depends on the previous $A_{k,i}(m)$ metrics. Once we initialize $A_{0,i}(m)$ correctly, we can keep computing $A_{1,i}(m)$ to $A_{255,i}(m)$ (in our design). Based on equation (2.11) and the assumption that the initial state of the encoder is 0, we can initialize $\alpha_{k,i}(m)$ as follows:

$\alpha_{0,0}(0)=0;$ $\alpha_{0,0}(1)=1;$ $\alpha_{0,0}(2)=1;$ $\alpha_{0,0}(3)=1;$

$\alpha_{0,1}(0)=0;$ $\alpha_{0,1}(1)=1;$ $\alpha_{0,1}(2)=1;$ $\alpha_{0,1}(3)=1;$

Since

$$A_{k,i}(m) = -\log \alpha_{k,i}(m)$$

$A_{0,i}(m)$ will be initialized as follows:

$A_{0,0}(0)=-\infty$; $A_{0,0}(1)=0$; $A_{0,0}(2)=0$; $A_{0,0}(3)=0$;

$A_{0,1}(0)= -\infty$; $A_{0,1}(1)=0$; $A_{0,1}(2)=0$; $A_{0,1}(3)=0$;

In our design, $-\infty$ will be replaced with the most negative number.

According to equation (3.2), the input (a_k,b_k) should be scaled by the noise variance and then fed into the Ak module. Since the operations in all decoders are linear and the final result dk is determined by comparison, we find that this scaling factor can be considered outside the MAP module. The design of the Ak module is shown in figure 3.4. Eight registers are used to generate the values $A_{k-1,i}(m)$, which are fed back to the subtractor to compute the next $A_{k,i}(m)$. A dedicated signed subtractor module was designed at the structure-level. The registers will be initialized with $A_{0,i}(m)$ metrics. The inputs x,y,z correspond to a_k, b_k and (a_k+b_k). At each active clock edge, the next $A_{k,i}(m)$ will be produced.

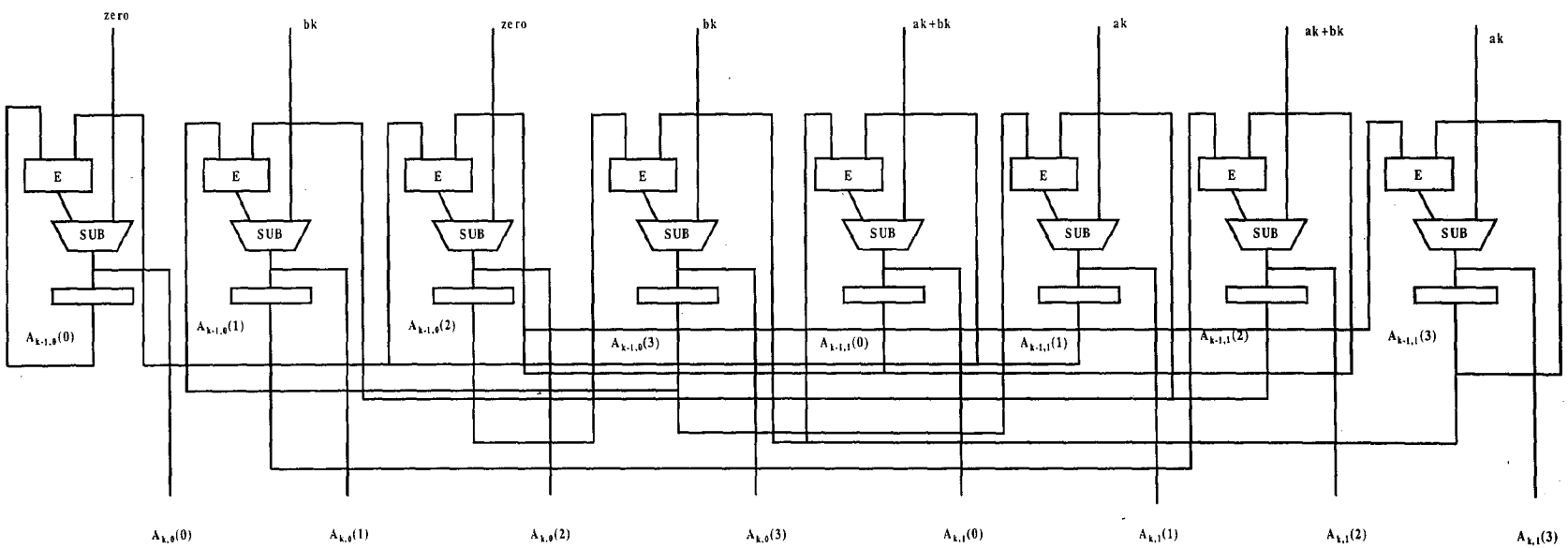

Figure 3.4: Ak module

3. Computing of the backward state metrics $\beta_{k,i}(m)$

I. Derivation of $B_{k,i}(m)$

Like the metrics $\alpha_{k,i}(m)$, there are also 8 different branch matrices $\beta_{k,i}(m)$ for each time k. In equation (2.12), $S_{f,i}(m)$ is the next state number of the encoder when the encoder's current state number is m and the current input is i. According to the state diagram of our encoder, we find:

$S_{f,0}(0)=0;$ $S_{f,0}(1)=2;$ $S_{f,0}(2)=3;$ $S_{f,0}(3)=1;$

$S_{f,1}(0)=2;$ $S_{f,1}(1)=0;$ $S_{f,1}(2)=1;$ $S_{f,1}(3)=3;$

Substituting the branch metrics into equation (2.12), we can compute the 8 $\beta_{k,i}(m)$ metrics as follows:

$$\beta_{k,1}(1)=\beta_{k,0}(0)=\beta_{k+1,0}(0)+\beta_{k+1,1}(0)*\exp(\frac{2}{\sigma^2}(a_{k+1}+b_{k+1}))$$

$$\beta_{k,1}(0)=\beta_{k,0}(1)=\beta_{k+1,0}(2)+\beta_{k+1,1}(2)*\exp(\frac{2}{\sigma^2}(a_{k+1}+b_{k+1}))$$

$$\beta_{k,1}(3)=\beta_{k,0}(2)=\beta_{k+1,0}(3)*\exp(\frac{2}{\sigma^2}b_{k+1})+\beta_{k+1,1}(3)*\exp(\frac{2}{\sigma^2}a_{k+1})$$

$$\beta_{k,1}(2)=\beta_{k,0}(3)=\beta_{k+1,0}(1)*\exp(\frac{2}{\sigma^2}b_{k+1})+\beta_{k+1,1}(1)*\exp(\frac{2}{\sigma^2}a_{k+1}) \quad (3.3)$$

Again we see that the computation of $\beta_{k,i}(m)$ includes the operations such as exponentials and multiplications. We applied the simplified MAP algorithm to avoid these operations. We defined the state metrics $B_{k,i}(m)$ by equation (2.14), where $B_{k,i}(m)$ is computed as follows:

$$B_{k,1}(1) = B_{k,0}(0) = E\{[B_{k+1,0}(0)], [B_{k+1,1}(0) - \frac{2}{\sigma^2}(a_{k+1} + b_{k+1})]\}$$

$$B_{k,1}(0) = B_{k,0}(1) = E\{[B_{k+1,0}(2)], [B_{k+1,1}(2) - \frac{2}{\sigma^2}(a_{k+1} + b_{k+1})]\}$$

$$B_{k,1}(3) = B_{k,0}(2) = E\{[B_{k+1,0}(3) - \frac{2}{\sigma^2}b_{k+1}], [B_{k+1,1}(3) - \frac{2}{\sigma^2}a_{k+1}]\}$$

$$B_{k,1}(2) = B_{k,0}(3) = E\{[B_{k+1,0}(1) - \frac{2}{\sigma^2}b_{k+1}], [B_{k+1,1}(1) - \frac{2}{\sigma^2}a_{k+1}]\} \quad (3.4)$$

II. The Bk module design

Like $A_{k,i}(m)$, the computation of $B_{k,i}(m)$ also involves a recursive formula, except that it should be calculated backward in time. The current $B_{k,i}(m)$ metrics depends on the next $A_{k,i}(m)$ metrics. We should initialize $B_{255,i}(m)$ first and then we can keep computing the values from $B_{254,i}(m)$ to $B_{0,i}(m)$. Based on equation (2.12) and the fact that the state of the encoder will always be reset at the final bit of every block, we initialize $\beta_{k,i}(m)$ as follows:

$\beta_{255,0}(0)=1;$ $\beta_{255,0}(1)=0;$ $\beta_{255,0}(2)=0;$ $\beta_{255,0}(3)=0;$

$\beta_{255,1}(0)=0;$ $\beta_{255,1}(1)=1;$ $\beta_{255,1}(2)=0;$ $\beta_{255,1}(3)=0;$

Since

$$B_{k,i}(m) = -\log \beta_{k,i}(m)$$

$B_{255,i}(m)$ will be initialized as follows:

$B_{255,0}(0)=0;$ $B_{255,0}(1)=-\infty;$ $B_{255,0}(2)=-\infty;$ $B_{255,0}(3)=-\infty;$

$B_{255,1}(0)=-\infty;$ $B_{255,1}(1)=0;$ $B_{255,1}(2)=-\infty;$ $B_{255,1}(3)=-\infty;$

In our design, $-\infty$ will be replaced with the most negative number.

The design of the Bk module is shown in figure 3.5. It is similar to the Ak module. Eight registers are initialized with $B_{255,i}(m)$ metrics. The three inputs x,y,z correspond to a_{k+1} , b_{k+1} and $(a_{k+1}+b_{k+1})$. The input sequence is fed into the Bk module backward in time. Thus at each active clock edge, the $B_{k,i}(m)$ values will be produced.

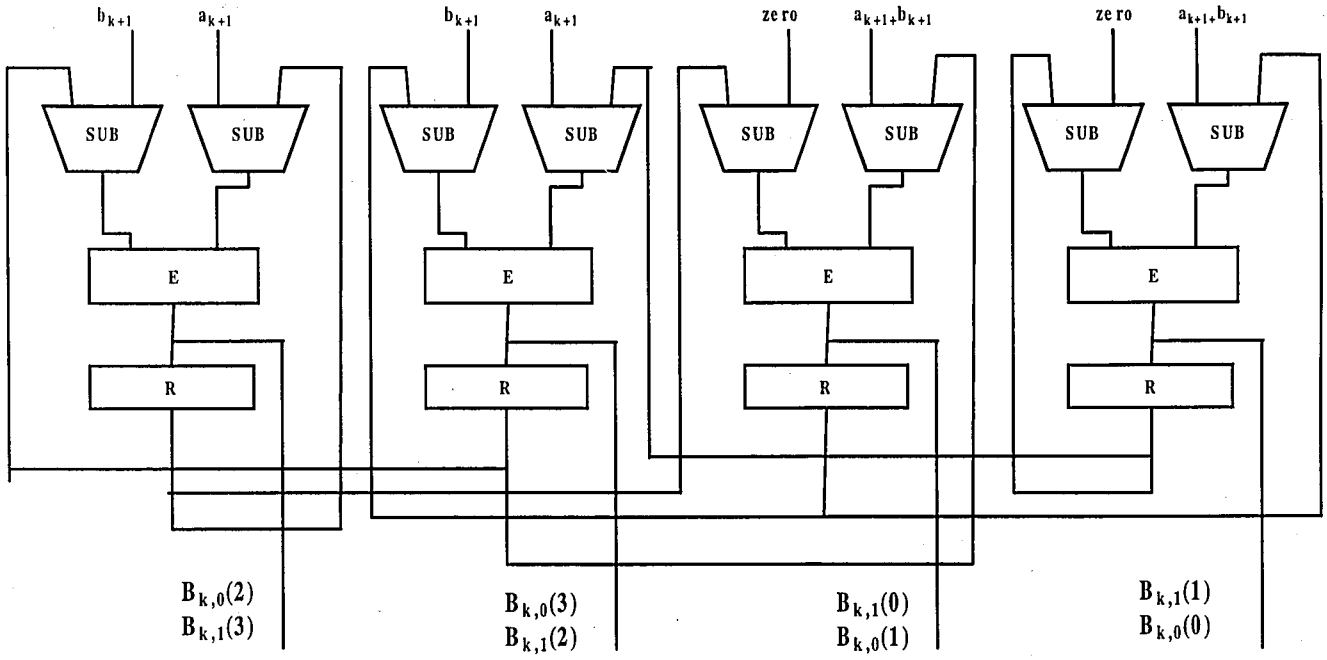

Figure 3.5: Bk Module

4. Design of the Ldk module to compute Λ_k

After we computer the values of $A_{k,i}(m)$ and $B_{k,i}(m)$, we are ready to compute the log-likelihood ratio Λ_k according to equation (2.16). It is computed by the Ldk module. As shown in Figure 3.6, the Ldk module takes eight different Ak metrics and eight different Bk metrics as inputs to produce the soft output of Λ_k.

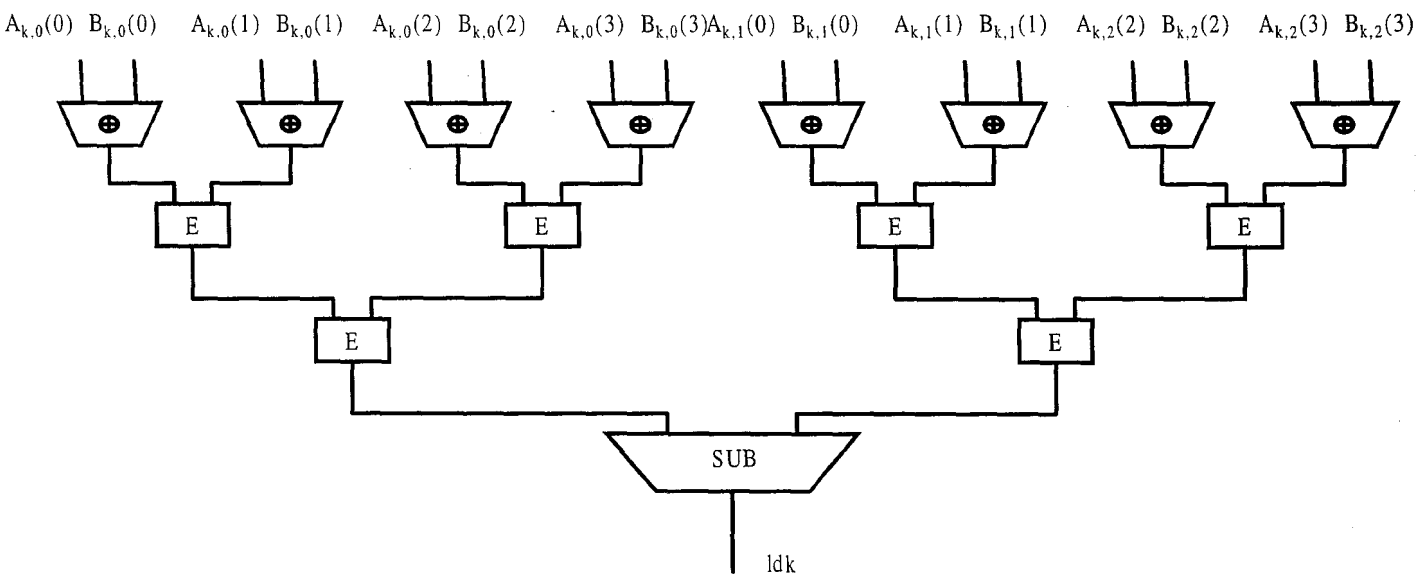

Figure3.6: Ldk Module

5. MAP module

The top MAP module ties together the basic modules. Besides the Ak, Bk, Ldk modules discussed earlier, we designed the shiftAk module and shiftReg module to store and manipulate the input sequence and the intermediate results. The MAP module also generates control signals to control the timing of the data flow.

The block diagram of the MAP module is shown in Figure 4.7. We can see that the input sequence a_k, b_k along with (a_k+b_k) are fed to the Ak module directly to compute the Ak metrics. When signal 'Ak_control' is '1', a new Ak will be generated at each active edge of the clock. Thus after 256 active clock edges, A_0 to A_{255} will be generated.

Since the Bk module will take the reversed input sequence as input and computes B255 first, a shift register is needed to store the input sequence first and then shift out the sequence in the reverse direction. We built the shiftReg module to model the shift register. It can shift in both directions, controlled by the 'sdir' signal. At each positive edge of the 'sr_control' signal, data will be shifted in and out.

Considering that the Ldk module takes the Ak and Bk metrics at the same time k as the inputs, we shall also store the computed Ak metrics in a shift register and output these to the Ldk module until the first Bk metrics (B_{255}) is ready. This is realized with the shiftAk module.

We can see that after 256 positive edges, a_{255}, b_{255} and $a_{255}+b2_{55}$ can be shifted to the Bk module and B_{255} can be generated and fed into the Ldk module. At this point, we can shift out the A255 to the Ldk module. The output of the MAP module is the likelihood dk sequence computed by the Ldk module and buffered with a register. Notice that this dk sequence is in reverse direction, i.e. from dk_{255} to dk_0;

In the MAP module, we build control circuits to generate the control signals 'Ak_control', 'Bk_control', 'sabkdir', 'sadir', 'sra_control', 'dk_control'. The module has an input 'enable'. Data should be fed into the system along with a positive edge of the 'enable' signal. This will tell the system to start computation, to initialize all the control signals and to clear the counter.

At the positive edge of the 'enable' signal,

- set 'Ak_control' and 'sra_control' to 1 ;

  To compute A_0 to A_{255} and store in shift register sra;
- set 'sabkdir' and 'sadir' to 1;

  To store a_k, b_k and (a_k+b_k) into the shift registers;
- set 'Bk_control' and 'dk_control' to 1;

  Not to compute Bk and not to output dk;
- set 'dkready' to 0;
- set 'count'=0;

In the control circuit, the counter will be incremented at the negative edge of the system clock. The control signals will be alternated by the value of the counter.

When 'count'=256,

- change sadir to 0;

  To shift a_k, b_k and (a_k+b_k) in the opposite direction and feed them into the Bk module;
- change 'Bk_control' to 1;

  start to compute Bk;
- change 'Ak_control' to 0;

  Stop computing Ak and start to shift Ak out in the opposite direction (A_{255} out first);
- change 'dk_control' to 1;

  start to output dk;
- set 'dkready'=1;

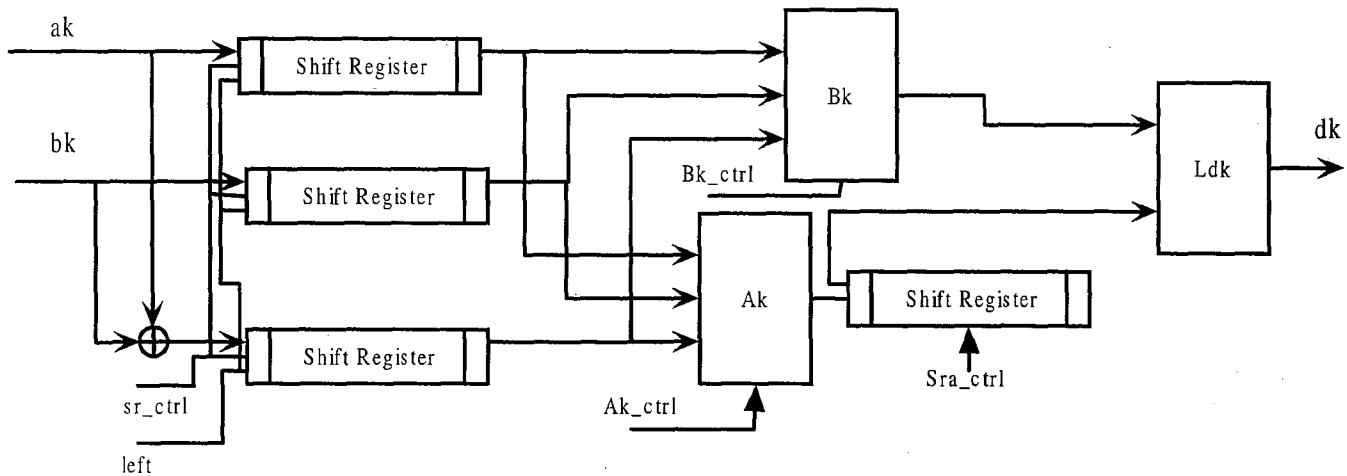

Figure 3.7: MAP Block Diagram

In our design, the control circuit is trigged by the negative edge of the clock. The control signals generated are AND-ed with 'clock' signal and then connected to control the data flow modules. In this way, we can ensure a clean control signal for the devices.

3.3. Interleaver and Deinterleaver

In the section 2.3, the functionality and theory of the interleaver were discussed. In the wireless environment where the delay cannot be too large, a non-uniform block interleaver is used in the design to achieve better performance than the pseudo-random interleaver.

Our design is based on an interleaver size of 256. Based on the 16*16 read-write matrix, the index of the data sequence after the interleaver can be determined. The index pattern is fixed for all the 256 data blocks. It is very easy to implement the interleaver by hardwiring two 256 shift registers as follows:

The input bit sequence is inserted into the upper 256 bit shift register. After all the 256 bits are filled, the contents of the upper shift register will be moved into the bottom 256 bit register based on the given order through the scrambled hard wires between the two registers. A level sensitive control signal 'load' is designed to trigger the download from the upper register to the bottom register, which is obtained by using a counter which can count up to the block length. Once a block of data fills the register, the 'load' signal triggers the download once.

The download only takes a half clock cycle, after which the two shift registers can shift out to form two synchronized bit streams. For the encoder, the two output streams will be encoded by two RSCs. For the decoder, the registers will be 8 bit wide and only the bottom register output will be used to generate the interleaved (deinterleaved) data streams.

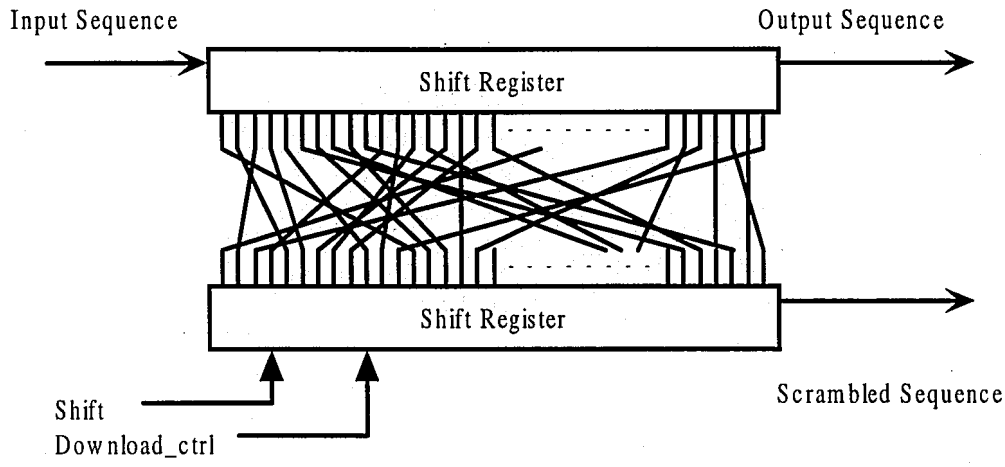

Figure 3.8 Interleaver Structure

After the first data block is interleaved by the interleaver, the following blocks can be inserted into the interleaver continuously without any delay gap. In this manner, the interleaver can be pipelined to maximize the throughput.

Chapter 4

Simulation and Performance Analysis

In this chapter, we are going to present the simulation and performance analysis of the system. The simulation is conducted in a bottom up fashion. Each functional block is first simulated and functionally verified through the Modeltech compiler and simulator. Then the whole system is integrated and a test bench is developed. Section 4.2 illustrates the simulation schematic and noise modeling used for the system simulation. Finally, the system performance is analyzed. Simulation results with different configurations are discussed in section 4.3.

4.1. Submodule Simulation

1. Encoder simulation

There are two submodules in the encoder, the RSC module and the interleaver module. The top module encoder instantiates submodules and implements control logic circuits. The RSC module generates the parity sequence according to the encoder's state diagram. As shown in Figure 4.1, we generate the clock signal and a random input sequence dk, and the module

outputs the parity sequence pk. Based on the encoder's state diagram and input bit value, we compute the output pk to verify the function of the RSC module.

The interleaver module takes the information sequence dk as input and outputs the randomized sequence dk02 and the initial information sequence itself dk01. Since we used a storage element in the design in order to randomize the sequence, the output sequence is delayed by 256 clock cycles. As shown in Figure 4.2, dk01 and dk02 are initially unknown. After 256 clock cycles, the interleaver modules output the original sequence and the interleaved sequence.

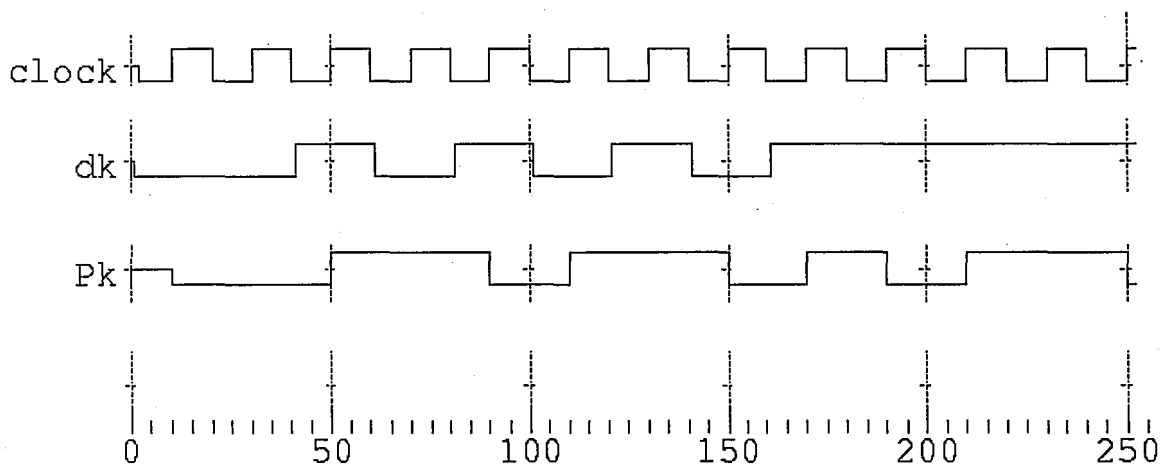

Figure 4.1. Waveform for Module RSC

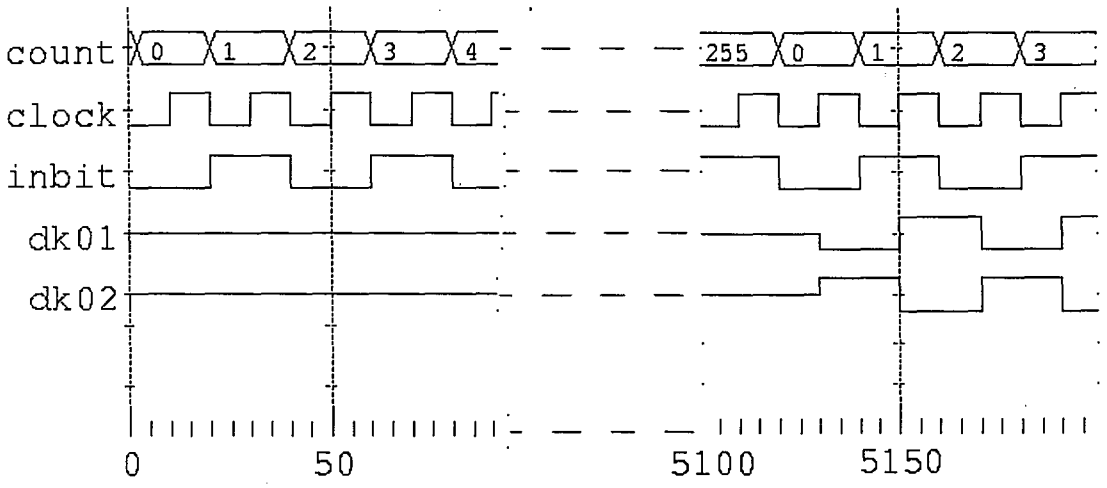

Figure 4.2. Waveform for Module Interlever

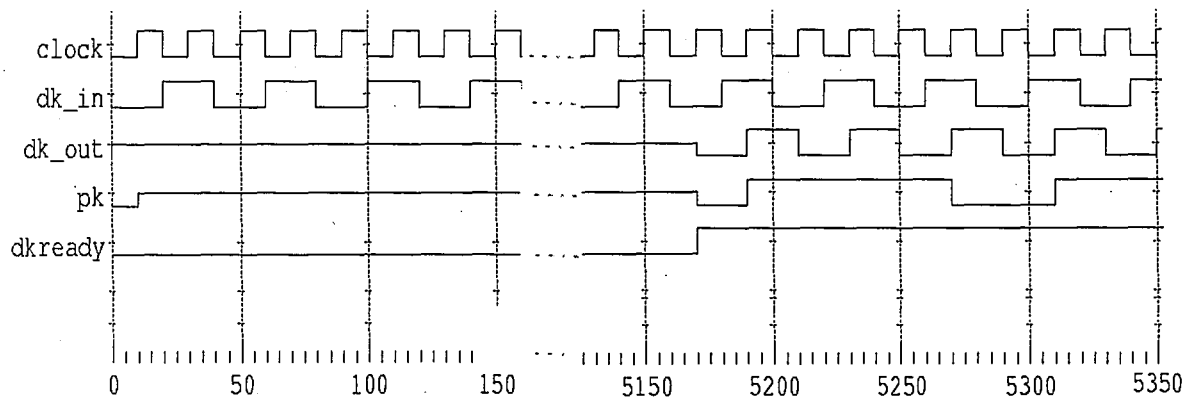

Figure 4.3. Waveform for Module Encoder

Finally, the top module encoder is simulated. We feed in the clock signal and the input sequence dk_in. After a 256 clock period delay, the encoder outputs the information sequence dk_out and the parity sequence pk and also raises the dkready line to '1'. The result is shown in Figure 4.3. The output is exactly as we expected.

2. Decoder Simulation

The simulation of the decoder starts with the Ak and Bk modules. By setting the initial value of eight metrics and feeding in the three inputs x, y and z, the output data string is obtained. Compared with the theoretical result according to equation (3.2), their functions are verified. Further, the MAP module is tested. While raising the 'enable' to be 1, we feed the module with two input sequences. After 258 clock cycles, the output is obtained and 'dkready' is raised to 1. The output waveforms are shown in Figure 4.4, 4.5,4.6. respectively.

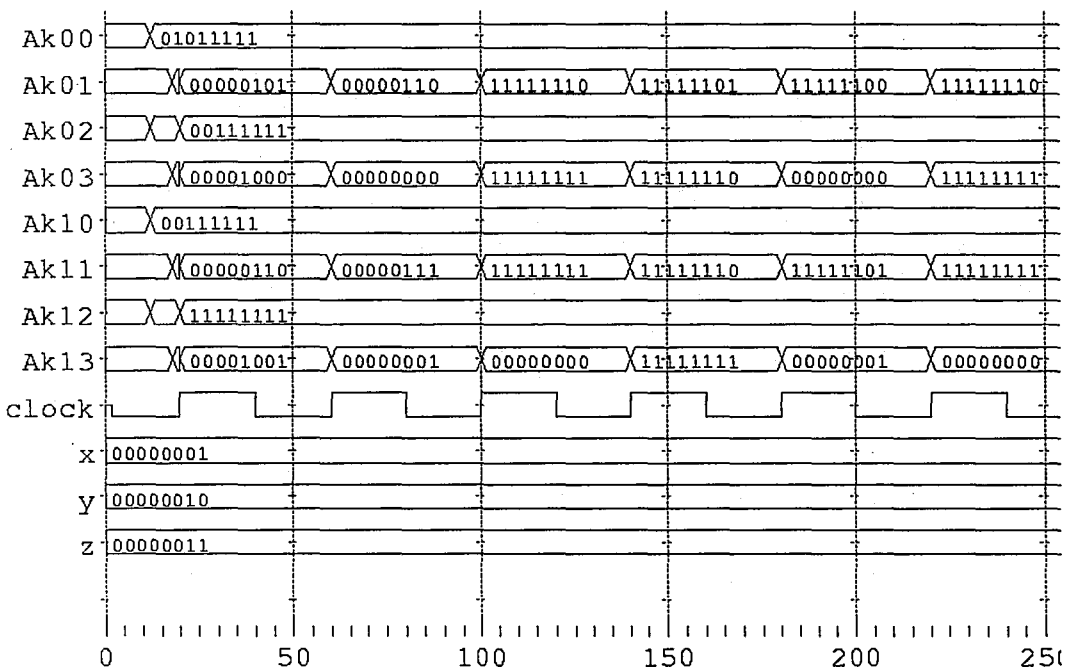

Figure 4.4. Waveform for Module Ak

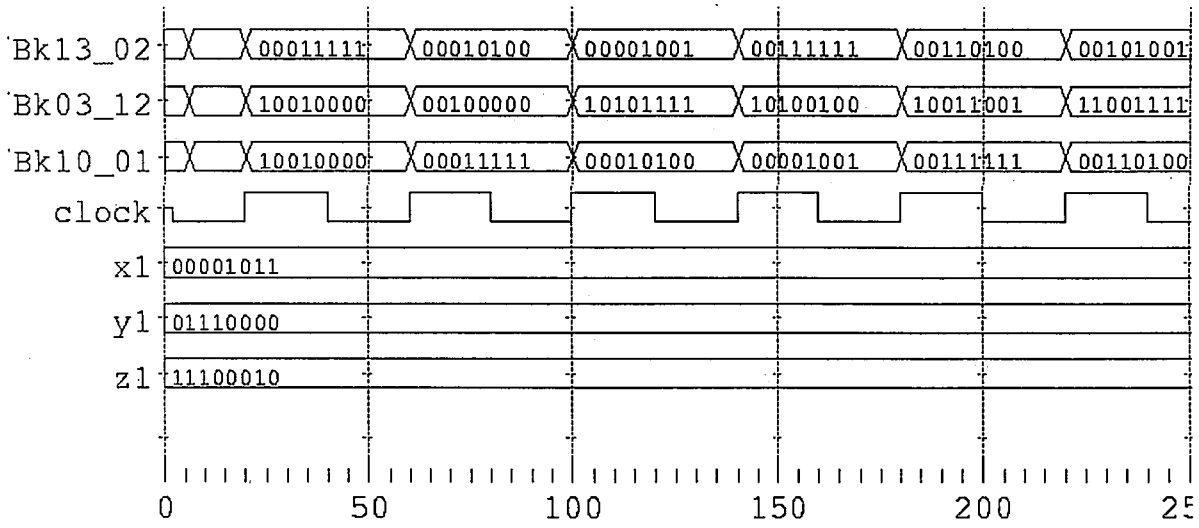

Figure 4.5. Waveform for Module Bk

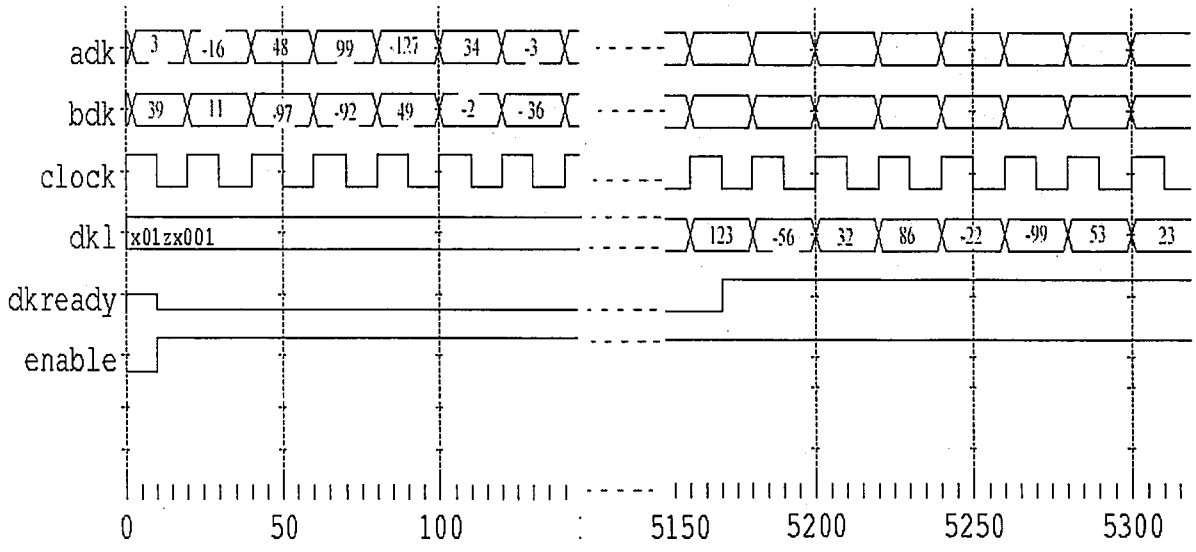

Figure 4.6. Waveform for Module MAP

4.2. System Simulation Scheme

With the functionally verified modules, the entire codec system is integrated together. To determine the working performance of the turbo codec, a system level simulation scheme is constructed which is shown below:

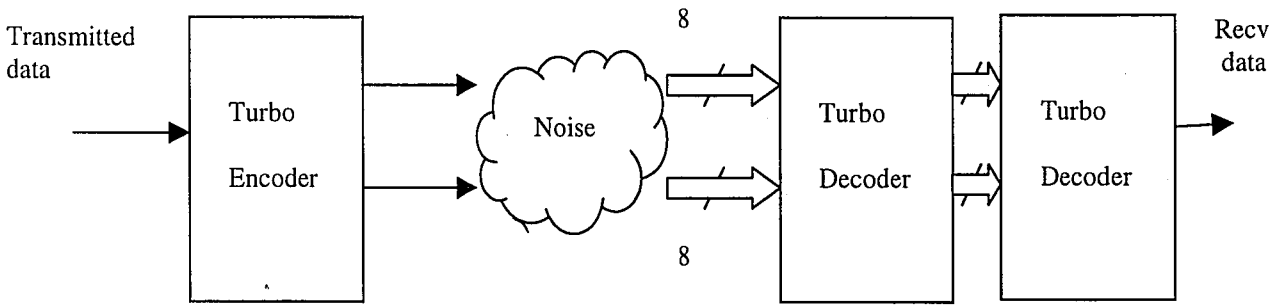

Figure 4.7 System Level Simulation Scheme

A random transmitted data sequence was generated using Verilog's system call $random. The data is grouped into 256-bit long blocks. All the blocks are fed into the encoder continuously without any gap in between. In all the simulations, 1000 blocks are used to generate trustable performance results. Otherwise, there won't be enough errors collected for the reliable performance evaluation. All the generated data blocks are stored in an input file using Verilog system call $fwrite, so that it can be compared with later recovered data from decoder to generate the bit-error rate curve in different noise environments.

The outputs of the encoder are two streams of bits, one of which is the original systematic data and the other is the parity bit sequence from the RSCs after puncturation. The two binary bit streams are collected and stored in the encoder output files using $fwrite.

To model the noisy channel, the MATLAB program was used. In this thesis, as well as in most of the literature, the additive white Gaussian noise (AWGN) is used to simulate a noisy environment. MATLAB provides a very accurate model of AWGN noise and is thus well suited for the task. For the simulation, the encoder output file is read into MATLAB, and the corresponding AWGN noise is added to the transmitted data and the parity sequence to produce the noisy received data sequence for the decoder. The original binary data sequences are converted into floating point noisy data and further quantized into 8-bit fixed point data which can be handled by the decoder. The system is tested for noise in the range from 1dB to 3.5 dB, which simulates realistic wireless communication environments.

The decoder input is the noisy data, which is generated by the MATLAB program, using $readmem system call of Verilog. The recovered data is obtained at the output of the last stage MAP decoder. The decoder can be configured with various numbers of MAP decoders to adjust the system for different requirements of communications. Because the MAP decoder is a Soft In Soft Out (SISO) structure, the data paths among adjacent modules are all 8 bits wide, except for the output of the last decoder, where the final hard decision has to be made to recover the binary data.

After obtaining the recovered data and the original transmitted data, the performance of the codec can be analyzed under different noise environments by altering the noise variation in MATLAB.

4.3. Performance Analysis

With the above system level simulation scheme, the performance of a turbo codes circuit can be simulated for several criteria, i.e. bit-error-rate at given SNRs, system delay, system throughput and system complexity. They reflect the quality, latency, data-rate and hardware cost of a communication system. All these factors are very important for evaluation for the different types of communication applications. There are a number of tradeoffs which have to be made in order to achieve the various requirements of a specific communications system.

1. Bit error rate

The first criterion is the measurement of the bit error rate for different noise environments. This is the most important measurement which decides the quality of the communication in a given channel. The whole Verilog turbo codec system is simulated for different noise levels (generated by using MATLAB). The resulting bit error rate versus signal to noise ratio (BER-SNR) curve is shown in Figure 4.8. The result is also compared with an ideal MAP decoder and a floating point log-MAP decoder. Both the MAP decoder and log-MAP decoder were modeled in C, with the same interleaver size 256, same interleaver structure and three iterations. The channel is simulated as an Additive White Gaussian Noise channel with the noise variance ranging from 1 dB to 3.5 dB. From the BER curve, we find that there is insignificant degradation due to using the simplified log-MAP algorithm. However, the fixed point implementation introduces a more severe degradation around 1 dB. Based on the specifications of the GSM wireless communication standards, the BER of interest are $4*10^{-2}$ for speech and 10^{-5} for data transmission. With the current simple configuration, this is attainable.

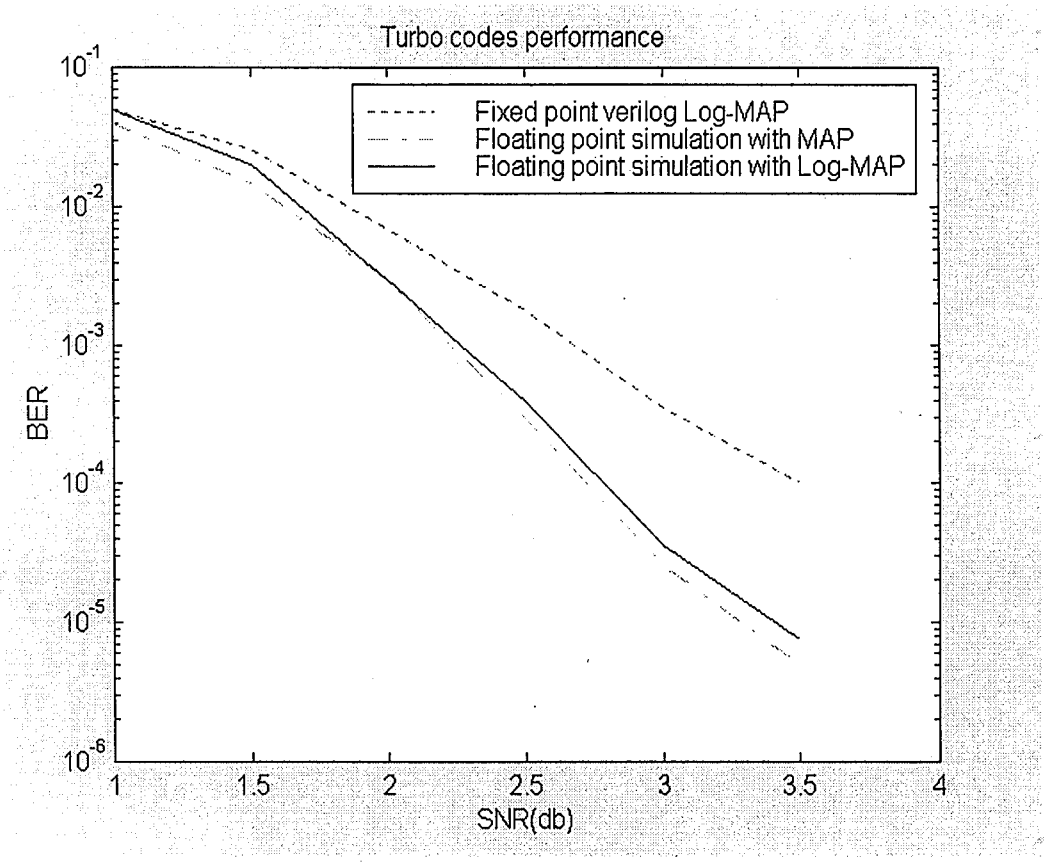

Figure 4.8 BER performance of turbo codes with different implementations

2. System Delay

The second important criterion is the system delay, i.e. the time period from the time when the first input enters the decoder to the time when its corresponding output appears at the output. This measurement decides the delay of the decoder, which can not be too large in wireless communication. With the intensive optimization of decoder logic to obtain the full pipeline structure, the system delay is determined by the number of iterations in the decoder. The more

iterations the decoder involves, the higher the quality and the longer the delay. The tradeoff of the quality and delay have to be made for different kinds of applications. Because the decoder was designed as a modular structure, the whole system is very scalable and adjustable for different applications. Based on different communication requirement, the decoder can be integrated with a different number of iterations by cascading different numbers of one-pass decoder modules, without requiring any more control signals. The adjacent modules can be synchronized by connecting the R-ready signal of the previous module to the enable signal of the following module. This kind of self-timing like control makes the whole system very

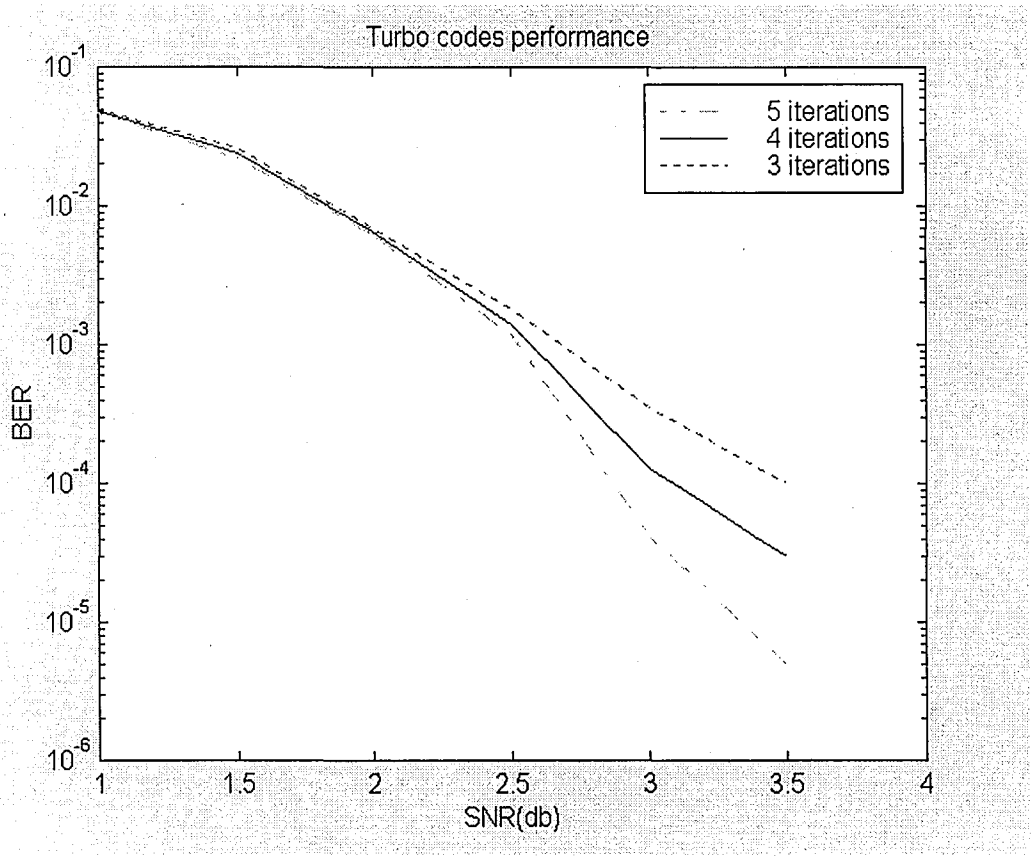

Figure 4.9 BER performance of turbo codes with different number of iterations

scalable. Each module corresponds to 2 data block's delay, i.e. 512 cycles in the current design. If N iterations are used for a system, N*512 cycles of delay will be introduced. The relationship between the performance (bit error rate) and the number of iterations was also tested using the above system-level simulation. The results are shown in Figure 4.9. Based on different BER requirements, the decoder can be integrated with various numbers of MAP modules. There are no specific circuits required to cascade these modules. The only work needed is to connect the outputs of the previous module to the inputs of the following modules and to connect the 'dkready' signal of the previous module to the 'enable' signal of the following module. This modular design makes the structure very scalable.

The GSM standard defines a block size is 189 bits. With the short block size of 256 bits used here, the current design can be applied to the standard.

3. Data rate

The fully pipelined structure makes the decoder operate at the maximum data rate. The throughput of the decoder is fixed and doesn't depend on the decoder iteration number. Due to the fully optimized design with the pipelined structure, there is no gap between data blocks so that the maximum data rate can be achieved. The data can be processed continuously, and the block boundaries are handled by the internal timing circuit. The real data rate will depend on the synthesis target, the clock, and the input data rate. The logic structure of the decoder doesn't impose any limitation on the throughput of the system.

4. Complexity

To get higher data throughput and lower delay, the complexity of the system is a little bit higher with more delay lines. For real-time communication applications environments, a tradeoff has to be made. On the other hand, by simplifying of the original complicated probability algorithm to a system without any multiplication and exponential computation, the current design only requires an adder/subtracter and a comparator. Also, the fixed point design makes the system even simpler.

Chapter 5

Conclusions

In this thesis, a pipelined and scalable turbo codec ASIC was designed to provide a circuit for high quality of communication in a high noise environment. Starting from the original "Turbo codes" paper [1], the algorithm was analyzed and partitioned into several functional blocks. The software simulation was performed using C to verify the theoretical performance of turbo codes. Furthermore, the algorithm was modified and simplified in order to make it feasible to be implemented in hardware. The implementation was modeled with the Verilog hardware description language in a bottom-up manner. Each module was developed and tested to guarantee functional correctness. With these tested modules, the encoder and decoder were integrated to form a whole codec system. The whole system was simulated with Modeltech to verify its error-correcting performance, as well as its system delay, throughput and hardware complexity. Based on the final simulation results, the codec performed very well in spite of some performance degradation compared with the theoretical floating point codec. All the

specifications and parameters of the system conforms to the latest wireless communication standards, like GSM, which has short block size, low bit error rate requirements.

Besides the functional correctness, the codec designed here has following features:

1. Pipelined structure

   The most complicated parts in the codec system are the MAP decoders. Through simplification, all the multiplication and exponential operations were converted into addition and comparison, respectively. But the whole structure of the decoder is recursive and the stages of computation depend on each other, which make the structure hard to integrate. In this design, in order to increase the data rate and take full advantage of the hardware resource, the turbo decoder was partitioned into several stages which enable pipelined processing of the data flow.

2. Scalable module

   The decoder was designed to be modular and scalable in order to provide the user the flexibility to adjust the system complexity based on different kinds of quality requirements. The more decoder modules a design uses, the higher the communication quality of the codec. In some less adverse environments with lower quality requirements, fewer modules may be used in order to keep the complexity and system cost low.

3. Continuous data flow

With the above pipelined design, the data rate of the system depends only on the synthesis technology and the input data rate. The codec itself does not impose any restriction on the throughput. The input data can be fed into the encoder and decoder continuously without any special timing circuits. All the block partitioning was handled inside the codec. Only an enable signal is required to indicate the beginning of the input. When the first data is recovered, a ready signal will be generated by the decoder for the user of the codec. This kind of design frees the codec's user from the need to design complicated timing circuit.

4. Synthesizable implementation

The whole design is synthesis ready for any kinds of target technology. In the Verilog modeling process, only logic equations and structural level statements were used with no single unsynthesizable statement, in order to make sure that the system can be synthesized easily and efficiently. Some dedicated modules were developed to keep the final complexity as low as possible, including signed computation and RSCs.

From the hardware and real application perspective, the above features are essential to most of the modern wireless communication systems.

Future improvements

1. Synthesis

Due to time limitation, the whole design has not been synthesized for any specific technology. The logic is ready for synthesis using any target technology, including FPGA and other ASIC implementation. Further modification of the algorithm may be needed when an efficient synthesis cannot be achieved.

2. Application specific adjustment

The codec is adjustable based on different types of applications. All the following parameters can be adjusted including:

- Interleaver block size (256 bits in the current design)
- Interleaver structure (non-uniform block interleaver in the current design)
- RSC module structure (constraint length is 3 in the current design)
- Coding rate (1/2 in the current design)
- Decoder iteration number (3 in the current design)
- Soft input/ Soft output data width (8 bits in the current design)

For example, in satellite communication, the block size can be longer compared with wireless communication, and thereby a higher quality of communication can be achieved. In some less adverse channels, like Racian channel where a line of sight exists between transmitter and receiver, the coding rate can be increased and less soft data width is needed.

3. More accurate E operation

The E function, i.e. $E(x,y)= - \log (e^{-x} + e^{-y})$ was simplified as minimal of x and y in the current design. From the simulation results, there is a little degradation due to this simplification. If more accuracy is needed and more complexity is allowed, the error introduced by this simplification can be reduced by generating a look-up table to store the E function value. The increased accuracy improves the performance of the decoder at the price of more hardware complexity.

List of References

[1] Berrou C, Glavieux A. "Near optimum error correcting coding and decoding: turbo-codes", *IEEE Transactions on Communications*, vol.44, no.10, Oct. 1996, pp.1261-71. Publisher: IEEE, USA.

[2] J. G. Proakis, "Digital Communications", 3rd ed. Boston, MA: WCB/McGraw-Hill,1995.

[3] Divsalar D, Pollara F. "Turbo codes for PCS applications", *ICC `95 Seattle. Communications - Gateway to Globalization 1995, IEEE International Conference on Communications*, vol.1, 1995, pp.54-9.

[4] Hagenauer J, Offer E, Papke L. "Iterative decoding of binary block and convolutional codes", *IEEE Transactions on Information Theory*, vol.42, no.2, March 1996, pp.429-45.

[5] Hall EK, Wilson SG. "Stream-oriented turbo codes", *VTC '98. 48th IEEE Vehicular Technology Conference. Pathway to Global Wireless Revolution,* vol.1, 1998, pp.71-5.

[6] A. S. Barbulescu and S. S. Pietrobon, " Interleaver design for turbo codes", Electronics Letters, vol. 30, pp. 2107- 2108, 1994.

Vita

Yi Wang was born in Shanghai, China, on November 1, 1972, the daughter of Shanli Lu and Deshu Wang. After completing her work at Nantong No.1 Middle School, Nantong, Jiangsu, China, in 1989, she entered Nantong Institute of Technology in Nantong, Jiangsu, China. She received the degree of Bachelor of Science from Nantong Institute of Technology in July, 1993.During the following years, she was employed as an assistant engineer with Hoechst Celanese in Nantong, Jiangsu, China. In January 1998, she entered the Graduate School of Lehigh University, Bethlehem, Pennsylvania.

END OF TITLE