Contents

1 Introduction **1**

1.1 Research Motivation 1

1.2 Thesis Organization 2

2 Trellis Based Decoding **4**

2.1 Decoding Algorithm for Turbo Code 4

2.1.1 The MAP Algorithm 4

2.1.2 The Log-MAP Algorithm 8

2.1.3 The Max-Log-MAP Algorithm 10

2.2 Sliding Window Approach 11

3 Turbo Code **13**

3.1 Structure of Turbo code 13

3.1.1 Turbo Encoding 14

3.1.2 Turbo Interleaver 15

3.1.3 Turbo Decoding 17

3.1.4 Error floor effect 19

3.2 Fixed Point Analysis of Turbo Decoder 19

4 The High-Speed MAP Decoder Design **22**

4.1 High-Speed ACS with Retiming Technique 23

4.1.1 Two-dimensional ACS unit 23

4.1.2 The Retiming Approach 25

4.2 Proposed MAP Decoder Architecture 27

4.2.1 Modified Max-Log-MAP Algorithm 27

4.2.2 Compare-Select Operation Circuit 30
4.3 Chip Implementation Result 31

5 The High-Speed Turbo Decoder Design **34**
5.1 The Inter-Block permutation interleaver 34
5.2 Proposed Turbo Decoder Design 36
5.2.1 IBPI with Butterfly Structure 36
5.2.2 SISO decoder 39
5.2.3 Memory unit 42
5.3 Chip Implementation 43

6 Conclusion **48**

List of Figures

1.1 The block diagram of digital communication system 2

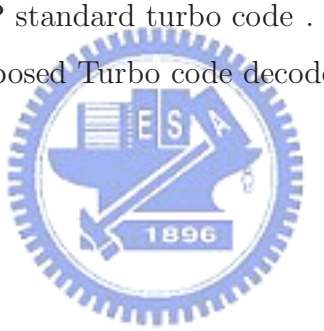
2.1 A (2,1,2) RSC encoder and its state transition diagram 5
2.2 The decoding trellis diagram of the (2,1,2) RSC encoder 6
2.3 The process diagram of sliding window algorithm 11

3.1 The structure of turbo encoder 14
3.2 Trellis termination for component RSC encoder 15
3.3 The graph representation of the interleaver process 16
3.4 The structure of turbo decoding 18
3.5 An example of four-state trellis diagram 20

4.1 A block diagram of MAP decoder 23
4.2 The radix-4 × 4 two-dimensional ACS unit 24
4.3 Retiming of registers 25
4.4 Retiming of adders 26
4.5 The retiming result of conventional radix-2 ACS unit 26
4.6 Comparison of critical path delay for original and retiming ACS 27
4.7 Conventional radix-4 × 4 ACS 28
4.8 Retimed radix-4 × 4 ACS 29
4.9 The decoding process of the modified MAP algorithm 30
4.10 The selection circuit for the max operation in (4.6) 31
4.11 Fixed point simulation of the input symbol 32
4.12 The die micrograph of the MAP decoder chip 33

5.1 Graph representation of the IBP interleaver 35

5.2 Message passing of the IBP intereleaver 35
5.3 Block diagram of Turbo decoder 36
5.4 The butterfly network structure 37
5.5 Graph representation of the IBP process 37
5.6 The overall procedure of the IBP interleaver 38
5.7 The graphical representation of the processing with the two input buffers . 39
5.8 The block diagram of the radix-2x2 Max-Log-MAP decoder 40
5.9 The decoding process for tail-biting trellis 41
5.10 The cyclic shifted input sequence order of the SISO decoder 41
5.11 The final input sequence order of the SISO decoder 41
5.12 The memory unit 42
5.13 The decoding schedule diagram 43
5.14 The block diagram of the proposed IBP turbo decoder 44
5.15 The iteration number simulation 45
5.16 Comparison with 3GPP standard turbo code 46
5.17 Layout view of the proposed Turbo code decoder 47



List of Tables

3.1 Standard specifications for turbo coding 13

4.1 MAP Decoder Specification 31
4.2 Summary of fixed representation in MAP decoder 32
4.3 Summary of the MAP Decoder Chip 33

5.1 Turbo Decoder Specification 45
5.2 Summary of fixed representation in Turbo decoder 45
5.3 Summary of the Turbo decoder chip 46

Chapter 1

Introduction

1.1 Research Motivation

A communication system conveys a information source to a destination through a channel. Fig. 1.1 shows a fundamental block diagram of traditional digital communication system. Generally, the system can be divided into transmitter and receiver via a channel. The main task of transmitter, including source encoder, channel encoder and modulator, is to transform the information into a form that can withstand the effect of noise over the transmission media. And the receiver will reverse the signal transformation by demodulator, channel decoder and source decoder. Since the channel impairments such as noise, interference and distortion may cause the error in the received signal, the channel encoder is incorporated in the system to add certain structural redundancy to the source codeword to minimize the transmission errors. Although these redundant bits may lowered data transmission rate, the channel coding eliminate the effects of noise disturbances and thus improve the performance, relative to an uncoded system.

There are two most frequently used types of channel codes, the block codes and the convolutional codes. The main different between the two of these codes is memory of the encoder. For the block code, the encoded codewords are only depend on the current input message and not on any previous messages. That is, the block code is memoryless. In contrast, the encoding procedure in the convolutional code depends not only on the current input message but also on certain past message blocks.

Turbo code, which belongs to the convolutional code, is one of the most popular

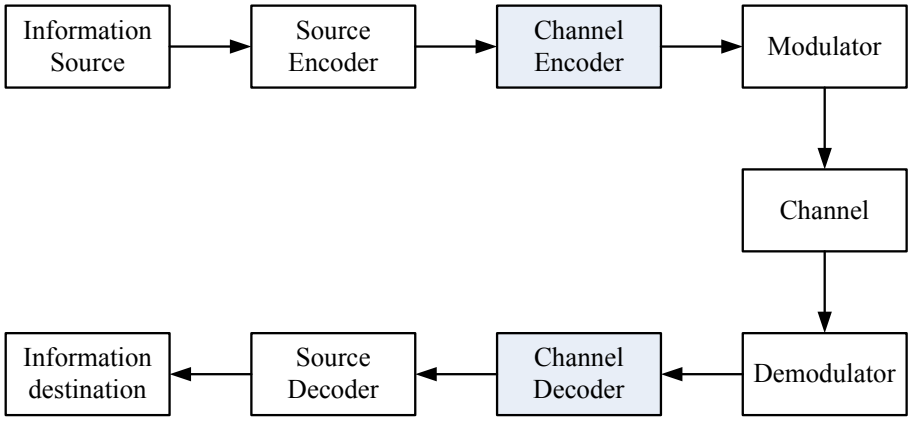


Figure 1.1: The block diagram of digital communication system

coding technique for the modern digital communication systems in the last decade. It is attractive by its excellent error correction ability resulted from soft iterative decoding. The main problem of turbo codes is that a long interleaver is able to support exceptional performance, but also introduces a long decoding latency and huge memory requirement. In the Third Generation Partnership Projects 3GPP [1] and 3GPP2 [2] defined detail standard, they provide maximum data rate of about 2 Mb/s and 3Mb/s, respectively. However, several communication application in the future may demand for a higher speed channel coding scheme. And it may become a obstacle for the turbo code in the hardware implementation.

In this thesis, our work is motivated to design a high-throughput turbo decoder. We attempt to achieve the target from two aspects: First one is to speed up the SISO decoder used in the whole turbo decoder by shortening its critical data path. Second, we employ a new interleaver to reduce the latency caused by the interleaver and propose a practical hardware architecture for the whole turbo decoder. Finally, we will propose a high speed turbo decoder with the modest hardware cost.

1.2 Thesis Organization

This thesis consists of 6 chapter. In chapter 2, several iterative decoding algorithm used for the turbo code and its relative practical techniques will be introduced. In chapter 3, we will review the structure of turbo code including its encoder, interleaver and decoder. Furthermore, the fixed point analysis about turbo code is also involved in this

chapter. Chapter 4 presents a high-speed Max-Log-MAP decoder, which utilize the retiming technique and two-dimensional ACS structure to reduce the critical path delay in the high-radix design under the efficient area cost. And the chip implementation results will also be reported in the end of this chapter. In Chapter 5, we further proposed a complete high-speed turbo decoder. Beside using the proposed Max-Log-MAP decoder as its component decoder, a new interleaver called inter-block permutation interleaver will be introduced and employed in our turbo decoder. Similarly, the implementation result will also be reported. Finally, the conclusion are given in chapter 6.

Chapter 2

Trellis Based Decoding

2.1 Decoding Algorithm for Turbo Code

The turbo code is composed of two soft-input soft-output (SISO) constituent codes that communicate iteratively through an interleaver. And the maximum a posteriori probability (MAP) [3] algorithm and soft-output Viterbi algorithm (SOVA) [4] are commonly employed for the SISO decoders. Unlike the SOVA which exploits maximum likelihood (ML) algorithm to minimize the word error probability, the MAP algorithm minimizes the symbol (or bit) error probability. In this section, we will focus on introducing the turbo decoding based on MAP algorithm, because it has been proved that the MAP algorithm is the optimal decoding method for turbo code while comparing with SOVA [5]. Furthermore, the Log MAP and Max-Log MAP algorithms will also be introduced, which reduce the hardware complexity and are widely used for implementation.

2.1.1 The MAP Algorithm

The MAP decoding algorithm, termed as *BCJR algorithm*, is developed by Bahl, Cocke, Jelinek, and Raviv [3] in 1974. For each transmitted information bit u_t, the MAP algorithm estimates the *a posteriori* probabilities (APP) based on the received code sequence $\mathbf{r}$ over a discrete memoryless channel (DMC). It computes the log-likelihood ratio (LLR)

$$L(u_t) = \frac{P(u_t = +1|\mathbf{r})}{P(u_t = -1|\mathbf{r})} \tag{2.1}$$

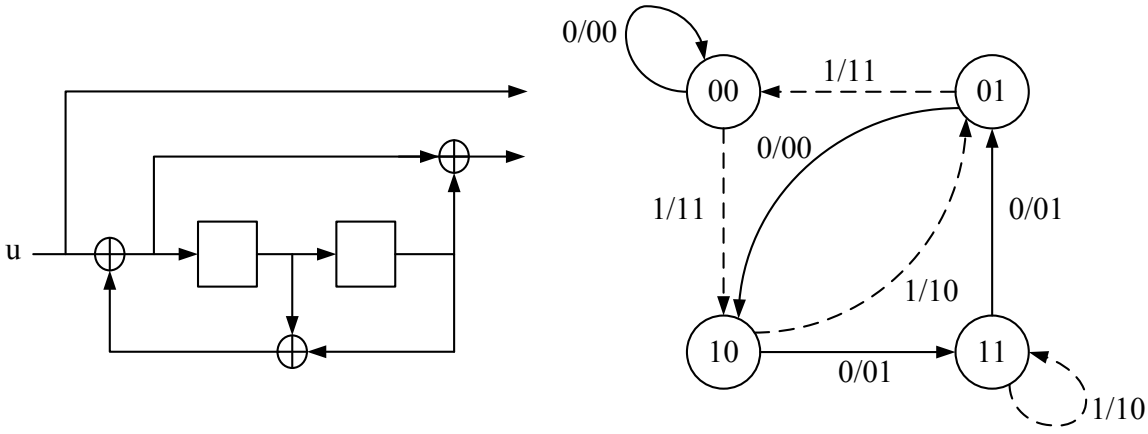


Figure 2.1: A (2,1,2) RSC encoder and its state transition diagram

for $1 \leq t \leq N$, where N is the received sequence length, and compares this value to a zero threshold to determine the hard estimate u_t as

$$u_t = \begin{cases} +1, & \text{if } L(\hat{u}_t) \geq 0 \\ -1, & \text{otherwise} \end{cases} \tag{2.2}$$

As an example, a rate 1/2 memory order 2 RSC encoder and its state transition are illustrated in Fig. 2.1. And its decoding trellis diagram is shown in Fig. 2.2. Note that the solid lines represent the state transitions corresponding to an information bit u_t of -1, while the dotted lines represent the state transitions corresponding to an information bit u_t of $+1$. From Fig. 2.2, the APP's in (2.1) can be computed by the summation of state transition probabilities. Therefore, the equation can be further expressed as

$$\begin{aligned} L(\hat{u}_t) &= \frac{P(u_t = +1|\mathbf{r})}{P(u_t = -1|\mathbf{r})} \\ &= \frac{\sum_{(m',m)\in\mathbf{B}_t^{+1}} P(S_{t-1} = m', S_t = m|\mathbf{r})}{\sum_{(m',m)\in\mathbf{B}_t^{-1}} P(S_{t-1} = m', S_t = m|\mathbf{r})} \\ &= \frac{\sum_{(m',m)\in\mathbf{B}_t^{+1}} P(S_{t-1} = m', S_t = m, \mathbf{r})}{\sum_{(m',m)\in\mathbf{B}_t^{-1}} P(S_{t-1} = m', S_t = m, \mathbf{r})} \end{aligned} \tag{2.3}$$

where $P(S_{t-1} = m', S_t = m, \mathbf{r})$ represents the joint probability for the existing transition from S_{t-1} at time t to S_t at time $t+1$. $\mathbf{B}_t^{+1}$ and $\mathbf{B}_t^{+1}$ is the sets of (m', m), denoted the state transitions which are due to input bit $u_t = +1$ and $u_t = -1$ respectively.

In order to compute joint probability required for calculation of $L(u_t)$ in (2.3), we define the following metrics:

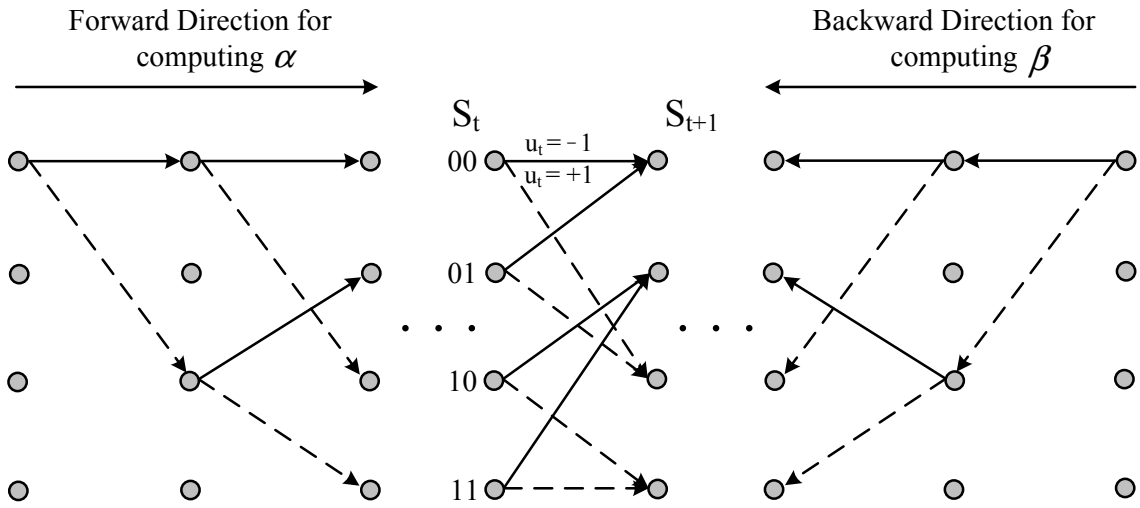


Figure 2.2: The decoding trellis diagram of the (2,1,2) RSC encoder

$$\lambda_t(m', m) = P(S_{t-1} = m', S_t = m, \mathbf{r}) \tag{2.4}$$

$$\alpha_t(m) = P\{S_t = m, \mathbf{r}_0^t\} \tag{2.5}$$

$$\beta_t(m) = P\{\mathbf{r}_{t+1}^{N-1} | S_t = m\} \tag{2.6}$$

$$\gamma_t(m', m) = P\{S_t = m, r_t | S_{t-1} = m'\} \tag{2.7}$$

Since we assume the code sequence after encoding is transmitted through discrete memoryless channel, the joint probability can be expressed as

$$\begin{aligned}
\lambda_t(m', m) &= P(S_{t-1} = m', S_t = m, \mathbf{r}_0^{t-1}, r_t, \mathbf{r}_{t+1}^{N-1}) \\
&= P(\mathbf{r}_{t+1}^{N-1} | S_{t-1} = m', S_t = m, \mathbf{r}_0^{t-1}, r_t) \\
&\quad \cdot P(S_t = m, r_t | S_{t-1} = m', \mathbf{r}_0^{t-1}) \\
&\quad \cdot P(S_{t-1} = m', \mathbf{r}_0^{t-1}) \\
&= P(\mathbf{r}_{t+1}^{N-1} | S_t = m) \cdot P(S_t = m, r_t | S_{t-1} = m') \cdot P(S_{t-1} = m', \mathbf{r}_0^{t-1})
\end{aligned} \tag{2.8}$$

Here $\mathbf{r}_0^{t-1}$ represents the received code sequence from time instance 0 to $t-1$, while $\mathbf{r}_{t+1}^{N-1}$ is from time instance $t+1$ to the end of sequence. Note that the second equation of (2.8) comes from Bayes' rule, and the third equation is due to the Markov process in the state transitions. Therefore, compared with the definition of (2.5), (2.6) and (2.7), the joint probability defined in (2.4) can be rewritten as

$$\lambda_t(m', m) = \alpha_{t-1}(m') \cdot \gamma_t(m', m) \cdot \beta_t(m) \tag{2.9}$$

Now we will derive the equations (2.5), (2.6) and (2.7) as follow:

$$\begin{aligned}
\alpha_t(m) &= P(S_t = m, \mathbf{r}_0^{t-1}) \\
&= \sum_{m' \in \mathbf{S}} P(S_{t-1} = m', S_t = m, \mathbf{r}_0^{t-1}) \\
&= \sum_{m' \in \mathbf{S}} P(S_t = m, r_{t-1} | S_{t-1} = m', \mathbf{r}_0^{t-2}) \cdot P(S_{t-1} = m', \mathbf{r}_0^{t-2}) \\
&= \sum_{m' \in \mathbf{S}} P(S_t = m, r_{t-1} | S_{t-1} = m') \cdot P(S_{t-1} = m', \mathbf{r}_0^{t-2}) \\
&= \sum_{m' \in \mathbf{S}} \alpha_{t-1}(m') \cdot \gamma_t(m', m)
\end{aligned} \tag{2.10}$$

Similarly, we have

$$\begin{aligned}
\beta_t(m) &= P(\mathbf{r}_{t+1}^{N-1} | S_t = m) \\
&= \sum_{m' \in \mathbf{S}} P(S_{t+1} = m', \mathbf{r}_{t+1}^{N-1} | S_t = m) \\
&= \sum_{m' \in \mathbf{S}} P(S_{t+1} = m', r_{t+1}, \mathbf{r}_{t+2}^{N-1}, S_t = m) \;/\; P(S_t = m) \\
&= \sum_{m' \in \mathbf{S}} P(\mathbf{r}_{t+2}^{N-1} | S_{t+1} = m', r_{t+1}, S_t = m) \cdot P(S_{t+1} = m', r_{t+1} | S_t = m) \\
&= \sum_{m' \in \mathbf{S}} P(\mathbf{r}_{t+2}^{N-1} | S_{t+1} = m') \cdot P(S_{t+1} = m', r_{t+1} | S_t = m) \\
&= \sum_{m' \in \mathbf{S}} \gamma_{t+1}(m, m') \cdot \beta_{t+1}(m')
\end{aligned} \tag{2.11}$$

where $\mathbf{S}$ represent the set of all states. Note that the forward metric α in (2.10) and the backward metric β in (2.11) are computed recursively in opposite direction. If the trellis of encoding diverges from zero state at $t = 0$ and converges to zero state at $t = N - 1$, as shown in Fig. 2.2, the following initial conditions are satisfied:

$$\begin{aligned}
\alpha_0(0) = 1, \quad \alpha_0(m) = 0 \quad \text{for } m \neq 0 \\
\beta_N(0) = 1, \quad \beta_N(m) = 0 \quad \text{for } m \neq 0
\end{aligned} \tag{2.12}$$

Furthermore, for any existing transitions from state m' to m, the branch transition prob-

ability $\gamma_t(m', m)$ can be decomposed as

$$\begin{aligned}
\gamma_t(m', m) &= P(S_t = m, r_t | S_{t-1} = m') \\
&= \frac{P(S_{t-1} = m', S_t = m, r_t)}{P(S_{t-1} = m')} \\
&= \frac{P(S_{t-1} = m', S_t = m)}{P(S_{t-1} = m')} \cdot \frac{P(S_{t-1} = m', S_t = m, r_t)}{P(S_{t-1} = m', S_t = m)} \\
&= P(S_t = m | S_{t-1} = m') \cdot P(r_t | S_{t-1} = m', S_t = m) \\
&= P(u_t) \cdot P(r_t | \mathbf{v_t})
\end{aligned} \tag{2.13}$$

where $P(u_k)$ is well-known as a prior probability of u_k and $\mathbf{v_t}$ is the codeword associated with the transition $S_{t-1} = m'$ to $S_t = m$ corresponding to encoder input u_t.

As a summary of the MAP algorithm, with computation of $\gamma_t(m', m)$ in (2.13), we can derive α and β for each state at different time instances. As a result, the joint probability in (2.9) is also available for $t = 0, 1, \cdots, N-1$. And we can calculate the log-likelihood ratio $L(u_t)$ by

$$L(u_t) = \log \frac{\sum_{(m',m) \in \mathbf{B}_t^{+1}} \alpha_{t-1}(m') \cdot \gamma_t(m', m) \cdot \beta_t(m)}{\sum_{(m',m) \in \mathbf{B}_t^{-1}} \alpha_{t-1}(m') \cdot \gamma_t(m', m) \cdot \beta_t(m)} \tag{2.14}$$

2.1.2 The Log-MAP Algorithm

The MAP algorithm requires large memory and a large number of operations involving exponentiation and multiplication. And the hardware realization of MAP decoder will be quite complex and difficult. Therefore, the Log-MAP algorithm is proposed to solve this problem. First, we transfer the branch metrics defined in the MAP algorithm to the logarithmic domain; that is

$$\bar{\gamma}_t(m', m) = \log \gamma_t(m', m) \tag{2.15}$$

Referring to (2.10) and (2.11), the forward path metric $\bar{\alpha}_t$ can be expressed as

$$\begin{aligned}
\bar{\alpha}_t(m) &= \log \alpha_t(m) \\
&= \log \sum_{m' \in \mathbf{S}} e^{\bar{\alpha}_{t-1}(m') + \bar{\gamma}_t(m', m)}
\end{aligned} \tag{2.16}$$

and the backward path metric $\bar{\beta}_t$ can be expressed as

$$\begin{aligned}
\bar{\beta}_t(m) &= \log \beta_t(m) \\
&= \log \sum_{m' \in \mathbf{S}} e^{\bar{\gamma}_{t+1}(m, m') + \bar{\beta}_{t+1}(m')}
\end{aligned} \tag{2.17}$$

Note that the initial conditions of path metrics also have changed, since all computations work with the logarithm domain.

$$\begin{aligned} \bar{\alpha}_0(0) = 0, \quad \bar{\alpha}_0(m) = -\infty \quad \text{for } m \neq 0 \\ \bar{\beta}_N(0) = 0, \quad \bar{\beta}_N(m) = -\infty \quad \text{for } m \neq 0 \end{aligned} \tag{2.18}$$

After substituting (2.15), (2.16) and (2.17), the APP information $L(\hat{u}_t)$ in (2.14) can be rewritten as

$$L(u_t) = \log \frac{\sum_{(m',m)\in \mathbf{B}_t^{+1}} e^{\bar{\alpha}_{t-1}(m')+\bar{\gamma}_t(m',m)+\bar{\beta}_t(m)}}{\sum_{(m',m)\in \mathbf{B}_t^{-1}} e^{\bar{\alpha}_{t-1}(m')+\bar{\gamma}_t(m',m)+\bar{\beta}_t(m)}} \tag{2.19}$$

Considering the following Jocobian algorithm [6]

$$\begin{aligned} \log(e^{\delta_1} + e^{\delta_2}) &= \max(\delta_1, \delta_2) + \log(1 + e^{-|e^{\delta_2} - e^{\delta_1}|}) \\ &= \max(\delta_1, \delta_2) + f_c(|\delta_2 - \delta_1|) \end{aligned} \tag{2.20}$$

where $f_c(\cdot)$ is a correction function and thus the performance can be improved. By a recursive procedure of (2.20), the expression $\log(e^{\delta_1} + e^{\delta_2} + \cdots + e^{\delta_n})$ can be computed exactly, as follows

$$\begin{aligned} \log(e^{\delta_1} + e^{\delta_2} + \cdots + e^{\delta_n}) &= \log(\Delta + e^{\delta_n}), \quad \Delta = e^{\delta_1} + \cdots + e^{\delta_{n-1}} = e^{\delta} \\ &= \max(\log \Delta, \delta_n) + f_c(|\log \Delta - \delta_n|) \\ &= \max(\delta, \delta_n) + f_c(|\delta - \delta_n|) \end{aligned} \tag{2.21}$$

Now we can use (2.20) to represent forward metrics in (2.16) and backward metrics in (2.17) as

$$\bar{\alpha}_t(m) = \max_{m'\in \mathbf{S}}{}^*\{\bar{\alpha}_{t-1}(m') + \bar{\gamma}_t(m',m)\} \tag{2.22}$$

and

$$\bar{\beta}_t(m) = \max_{m'\in \mathbf{S}}{}^*\{\bar{\gamma}_{t+1}(m,m') + \bar{\beta}_{t+1}(m')\} \tag{2.23}$$

where the $\max^*(\cdot)$ operation is defined as

$$\max{}^*(\cdot) = \max(\delta_1, \delta_2) + f_c(|\delta_2 - \delta_1|) \tag{2.24}$$

Therefore, the (2.27) can be expressed as

$$\begin{aligned} L(\hat{u}_t) = &\max_{(m',m)}{}^*\{\bar{\alpha}_{t-1}(m') + \bar{\gamma}_t(m',m) + \bar{\beta}_t(m)\} \\ &- \max_{(m',m)}{}^*\{\bar{\alpha}_{t-1}(m') + \bar{\gamma}_t(m',m) + \bar{\beta}_t(m)\} \end{aligned} \tag{2.25}$$

The performance of the Log-MAP algorithm is equivalent to the performance of the MAP algorithm but the complexity has been reduced considerately.. However, some difficulty for hardware implementation still exists since computing $f_c(\cdot)$ also involves exponentiations and multiplications. For simplified the computation of correction function, it is usually stored in a pre-computed table. And this table is only one dimensional due to the correction only depends on $|\delta_2 - \delta_1|$. Thus, the Log-MAP algorithm can be implemented with max function as well as a lookup table.

2.1.3 The Max-Log-MAP Algorithm

In order to further simplify the complexity, another approximation of MAP algorithm termed Max-Log-MAP algorithm is derived. Now, considering the following approximation formula **max** function

$$\log(e^{\delta_1} + e^{\delta_2} + \cdots + e^{\delta_n}) \approx \max_{i \in \{1,2,\cdot,n\}} \delta_i \tag{2.26}$$

Note that the term $f_c(\cdot)$ is ignored in comparison with (2.21). Then we can simplify the equation (2.19) as follows:

$$\begin{aligned} L(u_t) = & \max_{(m',m)\in \mathbf{B}_t^{+1}} \{\bar{\alpha}_{t-1}(m') + \bar{\gamma}_t(m',m) + \bar{\beta}_t(m)\} \\ & - \max_{(m',m)\in \mathbf{B}_t^{-1}} \{\bar{\alpha}_{t-1}(m') + \bar{\gamma}_t(m',m) + \bar{\beta}_t(m)\} \end{aligned} \tag{2.27}$$

Similarly, the forward recursive and backward recursive metrics in (2.16) and (2.17) can be individually expressed as

$$\bar{\alpha}_t(m) = \max_{m' \in \mathbf{S}} \{\bar{\alpha}_{t-1}(m') + \bar{\gamma}_t(m',m)\} \tag{2.28}$$

and

$$\bar{\beta}_t(m) = \max_{m' \in \mathbf{S}} \{\bar{\gamma}_{t+1}(m,m') + \bar{\beta}_{t+1}(m')\} \tag{2.29}$$

Here we can see that the computations of $\bar{\alpha}$ and $\bar{\beta}$ are reduced to simple add-compare-select operations, which are equivalent to the path metric updating of Viterbi algorithm. Therefore, compared with the MAP algorithm, the Max-Log-MAP algorithm utilizes additions to replace the multiplications and avoids the complicated exponentiations. However, the performance would degrade because of the information loss in (2.26).

2.2 Sliding Window Approach

In the conventional MAP-series decoding algorithm(including MAP algorithm, Max-Log MAP algorithm and Log-MAP algorithm), the LLR computation requires the path metric values generated by the forward and backward processes. Furthermore, since the backward recursive computation initials from the end of decoding trellis, as shown in Fig. 2.2, the decoding process can be started after the entire block message to be received. If the sequence length is large, it will lead to long output latency and huge memory requirement for hardware implementation. For example, the maximum block length of 3GPP2 standard is 20730, which means 20730 metrics should be stored. And it is the main disadvantage of turbo code for real application.

The main problem is that long block length can not divided into several short sub-block immediately, since the unknow initial condition of backward recursive metrics computations will damage the performance of turbo codes. Therefore, the sliding window approach was proposed [7] to overcome it. This algorithm utilizes the fact that the backward metrics can be highly reliable even without the initial condition if the backward recursion goes long enough. Fig. 2.3 shows the process of the sliding window algorithm and will be further illustrated as follows. First, the received codeword sequence is divided into several sub-blocks of length of W. W is called the convergence length, which normally is set to be five times constraint length of component encoder in turbo code to ensure the reliable

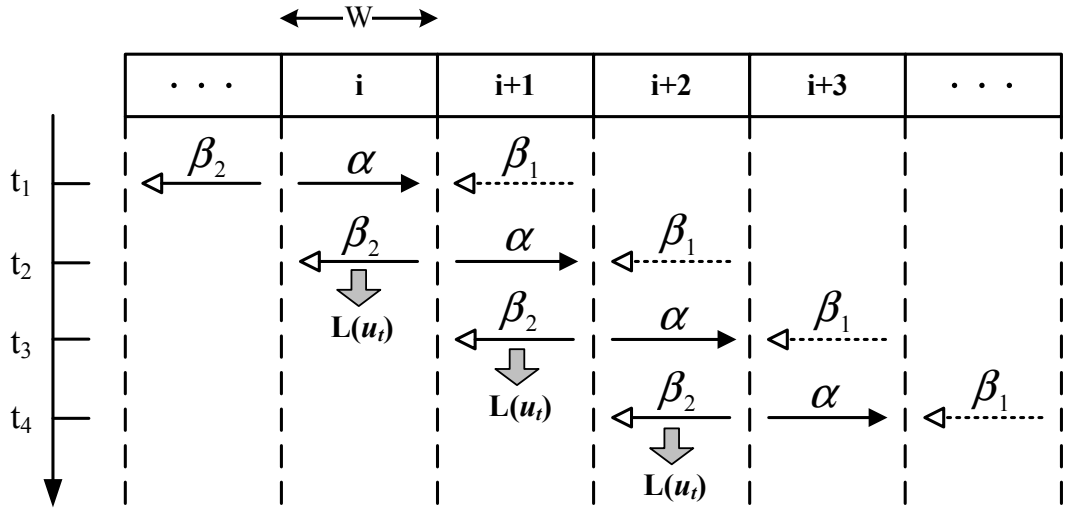


Figure 2.3: The process diagram of sliding window algorithm

initialization. In the sliding window approach, the end of sub-block is the initial of next sub-block whether the forward or backward recursive operation. Thus, the initial metric values are inherited from the last metrics calculated in the previous sub-block. Note that the dummy backward recursion β_1 is employed to establish the initial condition for the true backward recursion β_2. Although the initial condition for the β_1 is unknow except the last sub-block, we utilize the equally likely condition for the β_1 values at time instance $(i+1) \cdot W$:

$$\beta_1(m) = \frac{1}{M}, \quad \text{for all } m \in \mathbf{S} \tag{2.30}$$

where $\mathbf{S}$ represents all possible state and M is equal to the total state number. During the forward recursion α proceeds in the i-th sub-block and stores these values into memory, the dummy backward recursion β_1 is performed in the $i+1$ sub-block concurrently. As soon as the β_1 computation is finished, the initial metrics in the i-th sub-block are available for the β_2 recursion. And $L(\hat{u}_t)$ can be calculated based on the α metrics in the memory, the β_2 metrics in computation, and the corresponding branches metrics in the i-th sub-block.

Chapter 3

Turbo Code

3.1 Structure of Turbo code

Turbo Codes, first introduced by C. Berrou, A. Glavieux and P. Thitimajshima in 1993, are impressive with the near Shannon limit performance. It exploits a similar idea of connecting two RSC (Recursive Systematic Convolutional) codes and separating them by a random interleaver. Moreover, they are concatenated in parallel. The primary reason for using a interleaver is to ensure that, each component codes get independent estimates on the information symbols from the other one at each iteration.

The turbo codes have found applications in several standards listed in Table 3.1 due

Table 3.1: Standard specifications for turbo coding

Standard	**Application**	**Iterative Code**	**Max. Throughput**
DVB-RCS	Digital video broadcast	Parallel conc. of 8-state conv. codes	68 Mb/s (rate 7/8)
IEEE 802.16	Wireless networking (MAN)	Turbo product code	25 Mb/s (rate 5/6)
3GPP UMTS	Wireless cellular	Parallel conc. of 8-state conv. codes	2 Mb/s (rate 1/3)
3GPP2 CDMA2000	Wireless cellular	Parallel conc. of 8-state conv. codes	3.09 Mb/s (rate 1/5)
CCSD	Space telemetry	Parallel conc. of 16-state conv. codes	384 kb/s (rate 1/2)

to its outstanding error correction ability. And in the following, we will describe the structure of turbo coding in detail.

3.1.1 Turbo Encoding

A turbo encoder is formed by two parallel recursive systematic convolutional (RSC) encoders and separated by a turbo interleaver, and the interleaver is a process of rearranging the ordering of a data sequence in a one-to-one deterministic format. Therefore, the information is encoded by the first component encoder, interleaved and encoded by the second one simultaneously. In other words, the same set of information sequence is encoded twice but in a different order. Thus, the turbo codes are also referred to as parallel concatenated convolutional codes (PCCC). A block diagram of a turbo encoder, based on a (2,1,4) RSC code is shown in Fig. 3.1. The generator matrix G(D) for this component RSC code can be written in the so-called systematic form:

$$G(D) = \left[1 \quad \frac{\mathbf{g_1}(D)}{\mathbf{g_0}(D)}\right] = \left[1 \quad \frac{1+D+D^3}{1+D^2}\right] \tag{3.1}$$

where $\mathbf{g_0}(D)$ and $\mathbf{g_1}(D)$ are feedback and feedforward polynomial, respectively

Note that each input bit is encoded as one systematic bit and one parity check bit in a rate 1/2 RSC encoder. However, in order to increase the code rate of turbo code, only the parity check sequence of the second encoder, denoted by $\mathbf{Z'_k}$, is transmitted. Therefore,

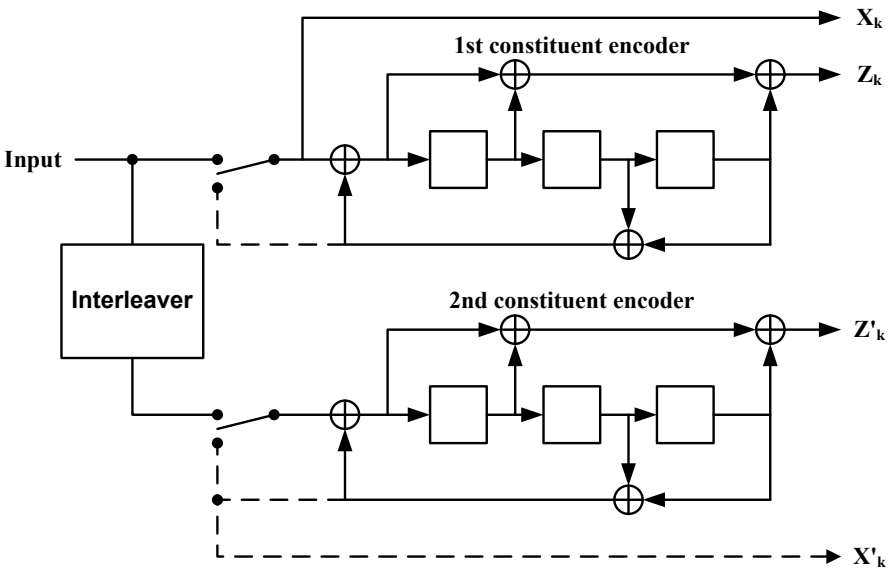


Figure 3.1: The structure of turbo encoder

the overall code rate is 1/3 and output encoded sequence should be $\{\mathbf{X_k}, \mathbf{Z_k}, \mathbf{Z'_k}\}$.

After encoding all input messages, transmission of some tail bits, which forces the encoder to finish encoding one block in the all-zero state, is required. The trellis termination makes sure that the initial state for the next block is the all-zero state. However, this operation would lead to two kinds of overhead. First, an extra amount of bits has to be sent through the encoder, decreasing the whole code rate. Second, extra circuitry in the turbo encoder is introduced. Since the component encoders are recursive, it is impossible to terminate the trellis to all-zero state only by inserting dummy zeros directly.

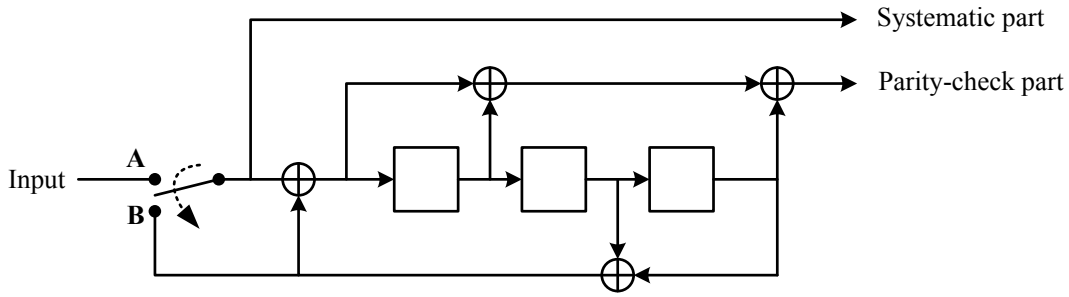


Figure 3.2: Trellis termination for component RSC encoder

To consider the latter problem, a simple solution is provided in Fig. 3.2. A switch in each parallel component encoder is set to position "A" for the first N input symbol and in position "B" for 3 tail bits in our example of Fig. 3.2, which shows a rate 1/2 RSC encoder with memory order 3. This trellis termination will flush all registers with zeros and thus the trellis return to all zero state.

3.1.2 Turbo Interleaver

For the turbo code, the interleaver plays an important role to achieve good performance. Its function is to rearrange the ordering of a data sequence in a one-to-one deterministic format. Fig. 3.3 shows a graph representation of interleaver process, where $\mathbf{u}$ is the input sequence as well as the systematic part of the codeword and $\tilde{\mathbf{u}}$ is the permuted sequence for the input of the second component encoder. Thus we can construct a long block code from small memory component convolutional codes via a interleaver. Moreover, it spreads out the burst errors and further eliminates the correlation of the input of two RSC encoders so that the iterative decoding algorithm based on exchanging

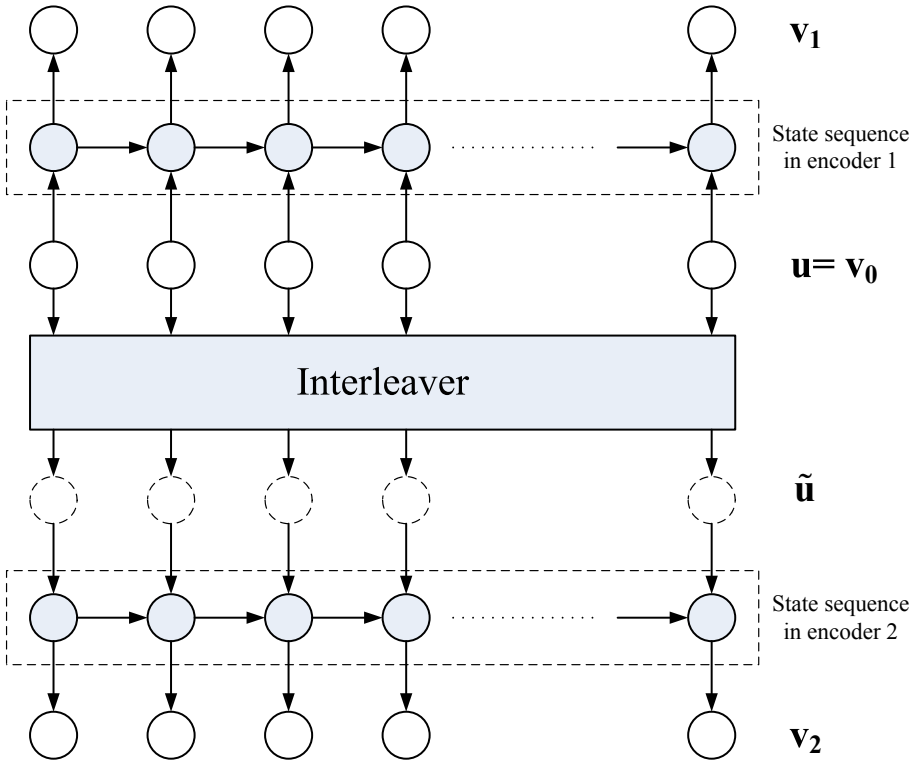


Figure 3.3: The graph representation of the interleaver process

the "un-correlated" information between two component decoders can be applied. Lastly, the interleaver can break low weight codewords to improve the coding gain.

The error performance of the turbo code is determined by the code distance spectrum. In [8], it points out a process of the interleaver called spectral thinning can reduce the error coefficients of low weight codewords. This effect leads to a bit error probability reduction by a factor $1/N$, where N is the interleaver size. And the factor $1/N$ is also referred to the interleaver gain. Under the analysis, the error performance at low SNR's is mainly dominated by the interleaver size. However, the low weight codewords produced by certain low weight input sequences dominate the turbo code performance at high SNR's. Consequently, a interleaver structure is desirable to break these input patterns. In such case, the input sequence to the second encoder ,which is generated by the interleaver, will most likely produce a high weight parity check sequence and further increase the whole turbo codeword weight. As a result, the interleaver size and structure will both affect the turbo code error performance considerately.

3.1.3 Turbo Decoding

The main idea for iterative turbo decoding is to exchange soft information among SISO decoders to calculate *a posteriori* probabilities of each information bit u_t. And the turbo decoding process based on the MAP algorithm will be examine as follows.

For a rate $1/n$ RSC encoder, each codeword consists of one systematic bit $v_t^{(0)}$ and $(n-1)$ parity bit $v_t^{(1)} \sim v_t^{(n)}$. Similarly, the decoder will receive codeword including one systematic symbol $r_t^{(0)}$ and parity symbols $r_t^{(1)} \sim r_t^{(n)}$. Thus, considering the AWGN channel with $2\sigma^2 = N_0/E_s$, the branch metric in (2.13) can be expressed as

$$\gamma_t(m', m) = P(u_t) \cdot P(r_t|\mathbf{v_t}) = P(u_t) \cdot e^{-\frac{\sum_{i=0}^{n-1}(r_t^{(i)} - v_t^{(i)})^2}{2\sigma^2}} \tag{3.2}$$

Note that the expression for $P(r_t|\mathbf{v_t})$ is normalized by multiplying a factor $(\sqrt{2\pi\sigma})^n$. By substituting (3.2), the log-likelihood ratio in the MAP algorithm in (2.14) can be further represented as

$$L(u_t) = \log \frac{\sum_{(m',m)\in \mathbf{B}_t^{+1}} \alpha_{t-1}(m') \cdot P(u_t = +1) \cdot e^{-\frac{\sum_{i=0}^{n-1}(r_t^{(i)} - v_t^{(i)})^2}{2\sigma^2}} \cdot \beta_t(m)}{\sum_{(m',m)\in \mathbf{B}_t^{-1}} \alpha_{t-1}(m') \cdot P(u_t = -1) \cdot e^{-\frac{\sum_{i=0}^{n-1}(r_t^{(i)} - v_t^{(i)})^2}{2\sigma^2}} \cdot \beta_t(m)} \tag{3.3}$$

where $P(u_t = +1)$ and $P(u_t = -1)$ are the *a priori* probabilities corresponding to the information bit 0 and 1, respectively. Now, we rewrite $L(u_t)$ by extracting the common term as

$$\begin{aligned} L(u_t) = \log \frac{P(u_t = +1)}{P(u_t = -1)} &+ \log \frac{e^{-\frac{(r_t^{(0)} - (+1))^2}{2\sigma^2}}}{e^{-\frac{(r_t^{(0)} - (-1))^2}{2\sigma^2}}} \\ &+ \log \frac{\sum_{(m',m)\in \mathbf{B}_t^{+1}} \alpha_{t-1}(m') \cdot e^{-\frac{\sum_{i=1}^{n-1}(r_t^{(i)} - v_t^{(i)})^2}{2\sigma^2}} \cdot \beta_t(m)}{\sum_{(m',m)\in \mathbf{B}_t^{-1}} \alpha_{t-1}(m') \cdot e^{-\frac{\sum_{i=1}^{n-1}(r_t^{(i)} - v_t^{(i)})^2}{2\sigma^2}} \cdot \beta_t(m)} \end{aligned} \tag{3.4}$$

Note that the systematic bits are independent of the state transition (m, m'). That is, $v_t^{(0)}$ in the numerator and denominator of the (3.3) must be $+1$ and -1. And $L(u_t)$ could be further decomposed into

$$L(u_t) = L_a(u_t) + \frac{2}{\sigma^2} r_t^{(0)} + L_e(u_t) \tag{3.5}$$

The term $L_a(u_t)$ is defined as the log-likelihood ratio of the *a priori* probabilities and $L_e(u_t)$ is called the extrinsic information.

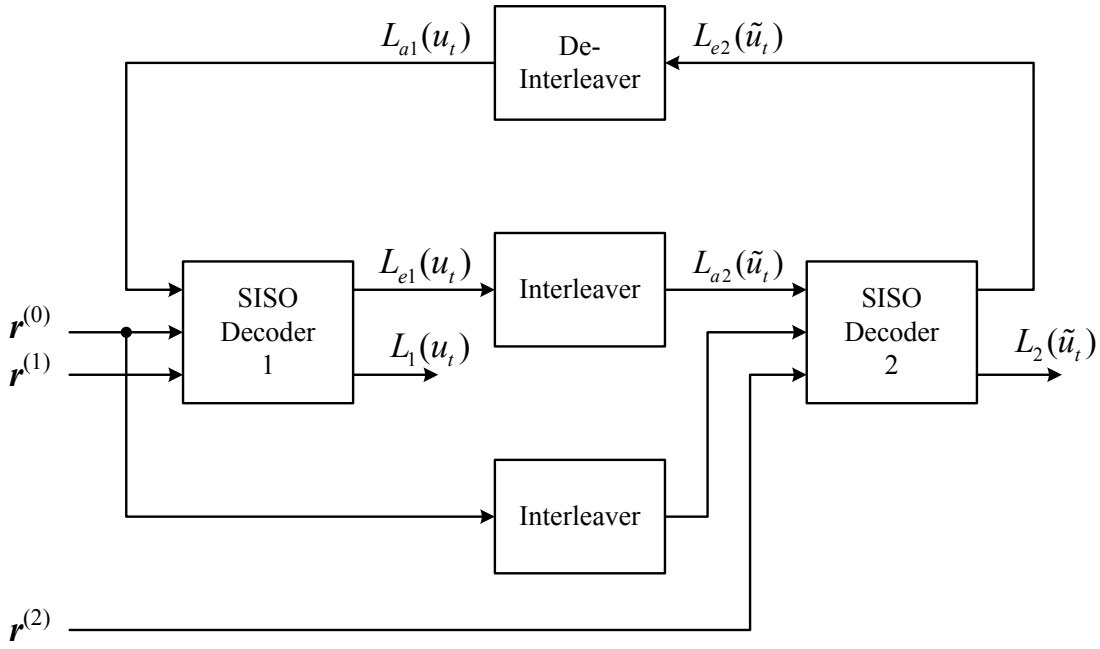


Figure 3.4: The structure of turbo decoding

Fig. 3.4 shows a general structure of turbo decoder, which consists of two component decoders via an interleaver as well as the one in the encoder. In the turbo decoding process, we first set the *a priori* information $L_{a1}(u_t)$ for the first SISO decoder to zero and calculate $L_1(u_t)$ base on the MAP algorithm. And we can get the extrinsic information $L_{e1}(u_t)$ from (3.5)

$$L_{e1}(u_t) = L_1(u_t) - \frac{2}{\sigma^2} r_t^{(0)} - L_{a1}(u_t) \tag{3.6}$$

Here, we can observe that the $L_{e1}(u_t)$ is a function of the redundant information that introduced by the encoder but removes the contribution due to systematic input and *a priori* information from $L_1(u_t)$. Therefore, it can be used as the *a priori* probability for the second decoding stage. Beside the parity sequence $r^{(2)}$, the SISO decoder 2 will also receive $\tilde{r}^{(0)}$ and $L_{a2}(\tilde{u}_t)$ which come from $r^{(0)}$ and $L_{e1}(u_t)$ after permutation, respectively. Under these input information, the SISO decoder 2 can also evaluate the *a posteriori* output $L_1(\tilde{u}_t)$ and the extrinsic information $L_{e2}(\tilde{u}_t)$ by

$$L_{e2}(\tilde{u}_t) = L_2(\tilde{u}_t) - \frac{2}{\sigma^2} \tilde{r}_t^{(0)} - L_{a2}(\tilde{u}_t) \tag{3.7}$$

Similarly, the information $L_{e2}(\tilde{u}_t)$ can be regarded as the *a priori* information $L_{a1}(u_t)$ for SISO decoder 1 after being reordered via the de-interleaver. For the turbo code, the decoding performance can be improved as the number of its decoding iteration increases.

However, the correlation between two component decoder will also be more apparent and further limit the performance improvements. Then the iterative turbo decoding process will stop after a certain number of iterations and makes hard decision using APP information $L_2(\tilde{u}_t)$ through the de-interleaver.

3.1.4 Error floor effect

Although the turbo code provides an excellent performance, the bit-error-rate (BER) will decrease quite slowly at high signal-to-noise ration (SNR). This phenomenon, called "error floor" region, is determined by the minimum free distance of turbo codes which is related to the interleaver. To consider the relation between the minimum free distance and the bit error probability in turbo coding, which can be expressed by

$$P_b \propto Q(\sqrt{2d_{free}R\frac{E_b}{N_0}}) \tag{3.8}$$

where d_{free} is the code minimum free distance, R is the code rate, and E_b/N_0 is the SNR.

3.2 Fixed Point Analysis of Turbo Decoder

Since this quantization will be the trade-off between coding performance and hardware cost, the fix-point analysis should be considered to minimize error performance loss. In general, the bit-width of input information is determined via simulations, and then the range of internal variables can be derived according to the bounded input [9].

Now we define the notation (n_i, n_f) to represent the symbol quantization, where n_i bits are integer part and n_f are floating parts. After quantizing, the maximum absolute value of the input symbols $r_t^{(i)}$ and *a priori* information $L_a(u_t)$, denoted by B_{in} and B_a respectively, are given. Therefore, the maximum difference of branch metrics, which is decided from (3.2), can be derived by

$$\Delta\gamma_t \leq n \times B_{in} + B_a \tag{3.9}$$

Here we assume that the decoder will receive n input symbols in each time instance based on a rate $1/n$ RSC encoder and all the received symbols are equally quantized.

For a RSC encoder with memory order m, the paths merging each state at time instance t originate from all states at time instance $t-m$. As a result, the difference of

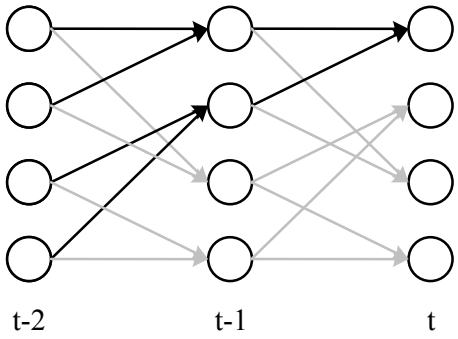


Figure 3.5: An example of four-state trellis diagram

any two path metrics at time instance t is only dependent on the branch metrics from time instance $t-m$ to t. Fig. 3.5 shows an example with $m = 2$. Therefore, we can derive the upper bounds for the difference of the forward path metrics α as

$$\Delta\alpha_t \leq d_m \times B_{in} + m \times B_a \tag{3.10}$$

where the value d_m represents the maximum Hamming distance between any two paths across m trellis sections. Similarly, the backward path metrics β is bounded by

$$\Delta\beta_t \leq d_m \times B_{in} + m \times B_a \tag{3.11}$$

In order to derive the upper bound of the $L(u_t)$, we can extend the equation (2.19) as follows:

$$\begin{aligned} L(u_t) &\leq \log \frac{\sum_{(m',m)\in\mathbf{B}_t^{+1}} e^{\max(\alpha_{t-1})+\max(\gamma_t)+\beta_t(m)}}{\sum_{(m',m)\in\mathbf{B}_t^{-1}} e^{\min(\alpha_{t-1})+\min(\gamma_t)+\beta_t(m)}} \\ &= [\max(\alpha_{t-1}) + \max(\gamma_t)] - [\min(\alpha_{t-1}) + \min(\gamma_t)] \\ &\quad + \left(\log \sum_{(m',m)\in\mathbf{B}_t^{+1}} \beta_t(m)\right) - \left(\log \sum_{(m',m)\in\mathbf{B}_t^{-1}} \beta_t(m)\right) \end{aligned} \tag{3.12}$$

Since each state at time instance t originates from two branches corresponding to the information bits $u_t = +1$ and $u_t = -1$. As a result

$$\log \sum_{(m',m)\in\mathbf{B}_t^{+1}} \beta_t(m) = \log \sum_{(m',m)\in\mathbf{B}_t^{-1}} \beta_t(m) \tag{3.13}$$

And the bound in (3.12) can be further simplified as

$$L(u_t) \leq \Delta\alpha_t + \Delta\gamma_t \tag{3.14}$$

Similarly, the lower bound can be obtained

$$L(u_t) \geq -(\Delta\alpha_t + \Delta\gamma_t) \tag{3.15}$$

As a conclusion, given the bound of the difference of the forward path metric $\Delta\alpha_t$ and the bound of the difference of the branch metric $\Delta\gamma_t$, the magnitude of the output LLR is bounded by

$$|L(u_t)| \leq \Delta\alpha_t + \Delta\gamma_t \tag{3.16}$$

Finally, the bound for the magnitude of the extrinsic information $L_e(u_t)$ can be derived from the (3.6), where

$$Le(u_t) = L(u_t) - \frac{2}{\sigma^2} r_t^{(0)} - L_a(u_t) \tag{3.17}$$

Hence, the bound can be obtained by

$$|Le(u_t)| \leq \left| L(u_t) - \frac{2}{\sigma^2} r_t^{(0)} - L_a(u_t) \right| \tag{3.18}$$

Chapter 4

The High-Speed MAP Decoder Design

As mentioned in Chapter 2, the component decoders for the Turbo codes perform iterative decoding based on maximum a posterior (MAP) probability algorithm. Considering the implementation complexity, the MAP algorithm is approximated to Max-Log-MAP algorithm with less complicated arithmetic. Fig. 4.1 illustrates the block diagram of the proposed MAP decoder, which consists of branch metric unit (BMU), add-compare-select (ACS) unit, log-likelihood-ratio (LLR) unit, and buffers. The first three units calculate the metrics and LLRs, while the buffers store the input symbols and forward path metrics. For a conventional MAP decoder, the computation time is dominated by the ACS operation, in which many studies on decoder architectures have been presented to reduce the critical path delay. Inkyu [10] proposed the double-state technique which enables addition and comparison to execute concurrently. However, this approach would lead to a large overhead because there are forward and backward path metric calculations in MAP algorithm. In [11], although the normalization operation is moved from ACS unit to BMU for higher speed, only radix-2 structure is considered.

In this chapter, we introduce a two-dimensional ACS structure to reduce the complexity of high-radix design and a retimed ACS unit to increase the operating frequency. From the implementation results, the present MAP decoder can facilitate high throughput designs with area efficiency.

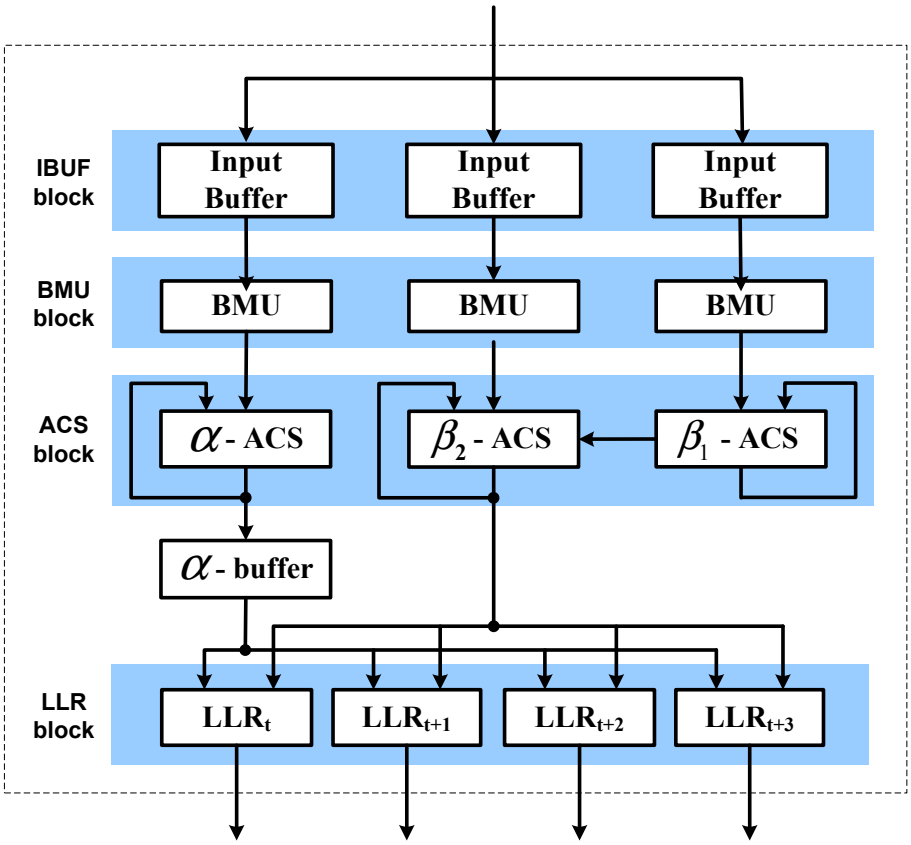


Figure 4.1: A block diagram of MAP decoder

4.1 High-Speed ACS with Retiming Technique

For the Max-Log MAP algorithm implementation, the decoder throughput is limited by the critical path delay of ACS unit due to the recursive computations. Even with the high-radix design, the performance is still dominated by the large critical path because of the exponentially increasing branches. In this section, we refer to a structured two-dimensional ACS unit with retiming technique to speed up high-radix architectures while keeping the least cost increase.

4.1.1 Two-dimensional ACS unit

In general, the radix-16 design can provide a speed-up by 4 as compared to the radix-2 structure. Nevertheless, the complexity and the branch number increase exponentially, resulting in large critical path delay and huge hardware cost. It also limits the feasibility to achieve a high speed ACS unit through the high-radix approach.

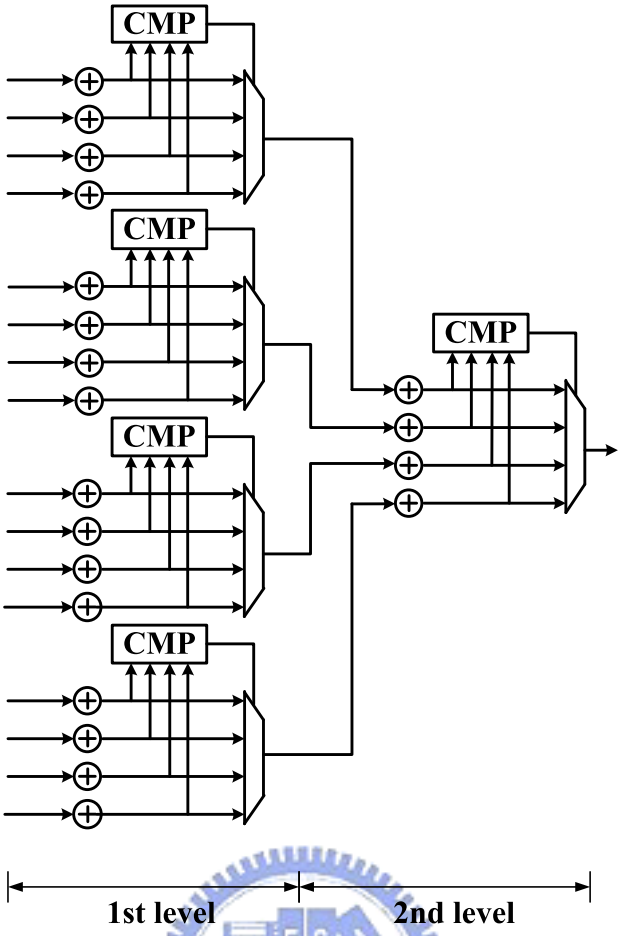


Figure 4.2: The radix-4 × 4 two-dimensional ACS unit

Therefore, we introduce a radix-4 × 4 structured ACS unit to decompose the compare operation among 16 branches into two levels. As shown in Fig. 4.2, the radix-4 × 4 ACS unit, referred to the two-dimensional structure, consists of two consecutive radix-4 ACS units. The survivors among four branches selected from each radix-4 ACS unit in first level will be the four candidates in the second radix-4 ACS unit.

The throughput of radix-4 × 4 structure is equivalent to the radix-16 approach. Additionally, the exponential increase of complexity has been restricted due to the multiple lower-radix ACS units instead of one single high-radix ACS unit. The number of branches between time instances t and $t + 4$ can be reduced from $16 \times N$ to $(4 + 4) \times N$ where N is the state number. However, the critical path of a radix-4 × 4 ACS unit through two levels of ACS units is longer than that of a radix-16 ACS unit.

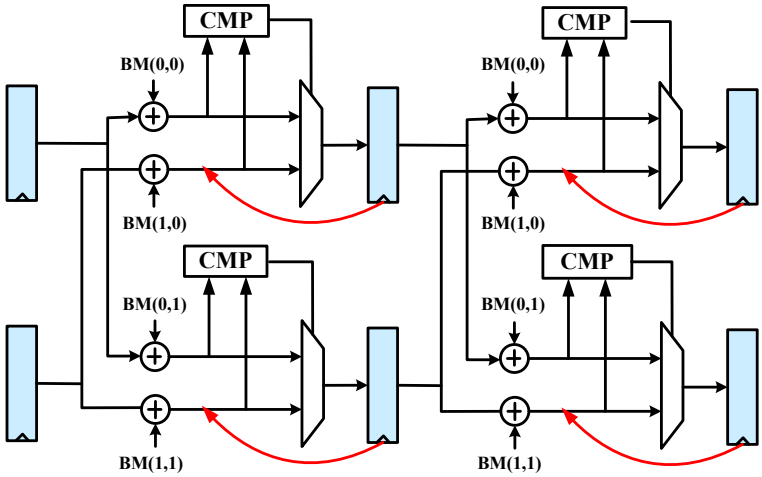


Figure 4.3: Retiming of registers

4.1.2 The Retiming Approach

The speed of the two-dimensional ACS unit can be further enhanced through the retiming approach. In order to illustrate clearly, the radix-2 architecture is employed as example.

In the radix-2 Max-Log-MAP algorithm, the forward and the backward recursions can be expressed by

$$\alpha_t(s_t) = \max_S\{\alpha_{t-1}(s_{t-1}) + \gamma_{t-1}(s_{t-1}, s_t)\} \tag{4.1}$$

$$\beta_{t-1}(s_{t-1}) = \max_S\{\gamma_{t-1}(s_{t-1}, s_t) + \beta_t(s_t)\} \tag{4.2}$$

where t is the time instance, S is the set of transitions from state s_{t-1} to s_t and α, β and γ represent the forward state, backward state and branch metric values, respectively. From (4.1) and (4.2), we find that pipelining is inapplicable because of the data dependency between the state and the branch metrics. And, the retiming approach can be exploited to break this dependency and increase the parallelism.

Fig. 4.3 illustrates the retiming procedure which moves registers from time instance t to the branches between t and $t-1$. Furthermore, the adders are relocated in order to concurrently perform addition and comparison as shown in Fig. 4.4. Note that the metric values being stored and compared are no longer associated with the results after additions.

The retimed radix-2 ACS architecture with parallelism of two is shown in Fig. 4.5. We assume that the delay time of comparators is larger than that of additions, thus, the

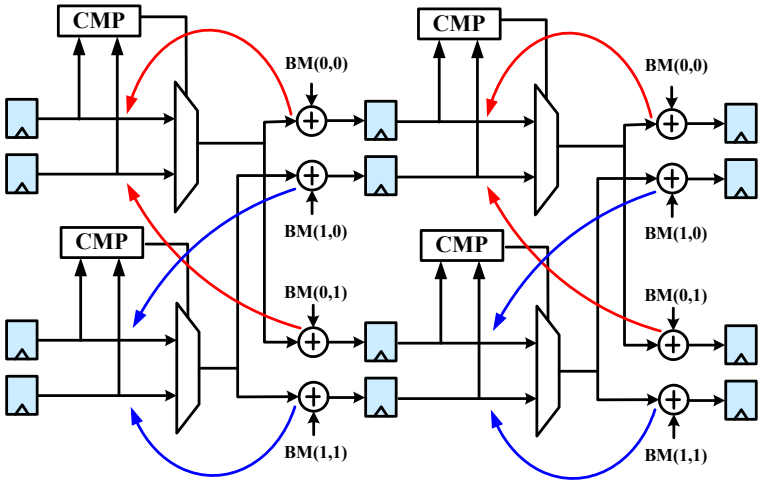


Figure 4.4: Retiming of adders

critical path delay will only be dominated by a single compare operation. Nevertheless, the number of registers, adders and multiplexers are doubled as compared to the original ACS unit in Fig. 4.3.

With combining the retiming technique and the radix-4×4 ACS unit, we can overcome the timing bottleneck in high-radix structure. Fig. 4.6 shows the delay time of three

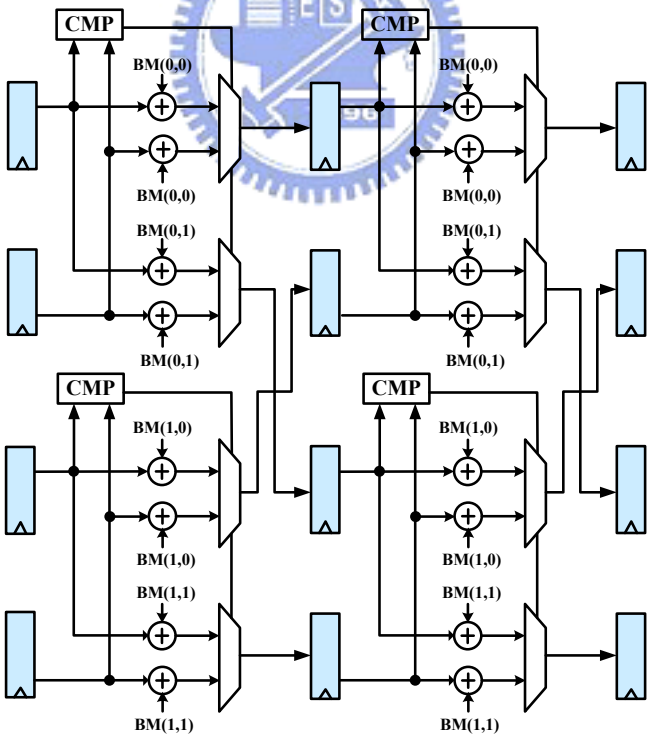


Figure 4.5: The retiming result of conventional radix-2 ACS unit

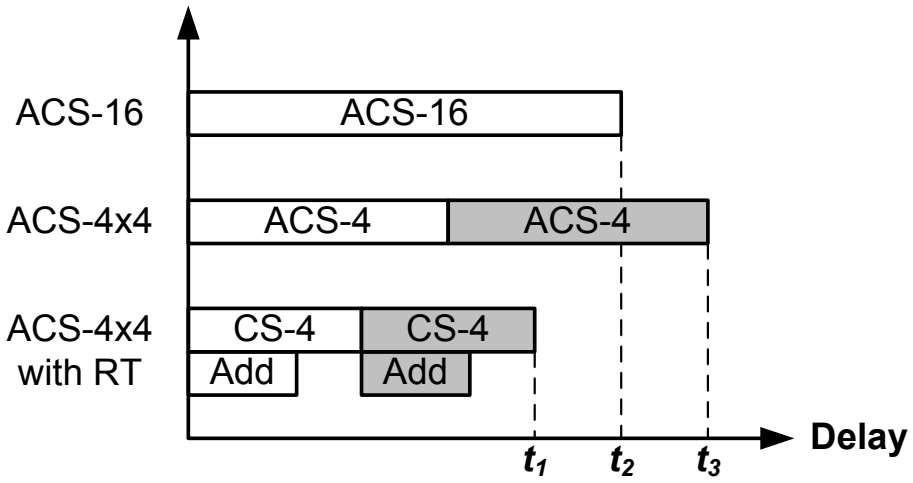


Figure 4.6: Comparison of critical path delay for original and retiming ACS

architectures. It is evident that the retimed ACS has the shortest path delay among all. Since area overhead resulted from retiming is less than the reduction from the two-dimensional ACS unit, the optimization method mentioned above can accomplish not only high-speed but area-efficient solutions based on two-dimensional structure with retiming technique.

4.2 Proposed MAP Decoder Architecture

4.2.1 Modified Max-Log-MAP Algorithm

In this section, we employ the retimed radix-4 $\times$ 4 ACS architecture into the MAP decoder. Since the retiming technique relocates the registers, the log-likelihood ratio (LLR) calculations in the Max-Log MAP algorithm have to be modified. The trellis diagrams in Fig. 4.7 and Fig. 4.8 illustrate the differences after applying retimed two-dimensional ACS unit. First, the critical path of the ACS unit has been reduced because adders and comparators operate in parallel. Note that the dotted blocks in Fig. 4.7 represent the dependency within each pair of add and compare operations and they are broken in Fig. 4.8. Second, the registers are moved to the second level branches of the radix-4 $\times$ 4 ACS unit, leading to different values stored in the registers, instead of the conventional path metrics.

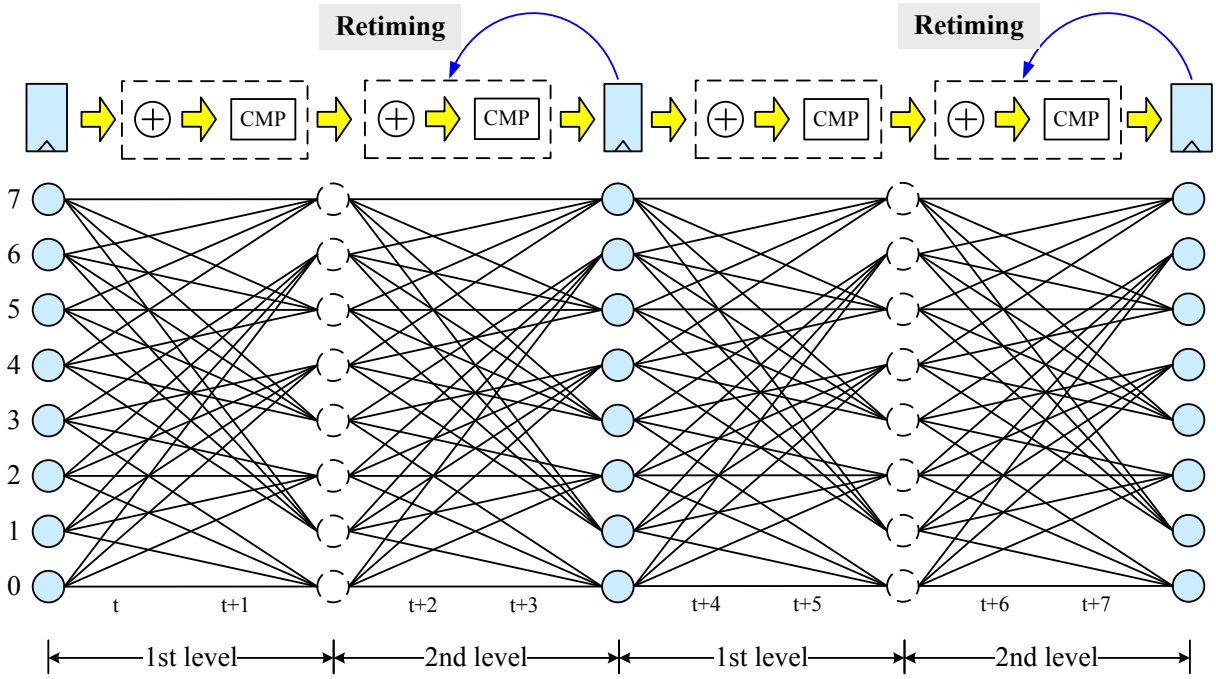


Figure 4.7: Conventional radix-4 × 4 ACS

For the retimed radix-4×4 architecture, we define another forward and backward path metrics, denoted by $\hat{\alpha}$ and $\hat{\beta}$ respectively,

$$\hat{\alpha}_t(s_{t-2}, s_t) = \alpha_{t-2}(s_{t-2}) + \gamma_{t-2}(s_{t-2}, s_{t-1}) + \gamma_{t-1}(s_{t-1}, s_t) \tag{4.3}$$

$$\hat{\beta}_t(s_t, s_{t+2}) = \beta_{t+2}(s_{t+2}) + \gamma_{t+1}(s_{t+1}, s_{t+2}) + \gamma_t(s_t, s_{t+1}) \tag{4.4}$$

For a radix-16 MAP decoder, the log-likelihood (LL) corresponding to the a zero input (bit) at time instant t can be represented by the following equation :

$$\begin{aligned} LL_t^0(s) = \max_{\hat{S}}\{\ & \alpha_t(s_t) + \gamma_t^0(s_t, s_{t+1}) \\ & + \gamma_{t+1}(s_{t+1}, s_{t+2}) + \gamma_{t+2}(s_{t+2}, s_{t+3}) \\ & + \gamma_{t+3}(s_{t+3}, s_{t+4}) + \beta_{t+4}(s_{t+4})\ \} \end{aligned} \tag{4.5}$$

where $\gamma_t^0(s_t, s_{t+1})$ corresponds to the branches when information bit is 0 at time t. $\hat{S}$ is the set of transition combinations emerging from state s_{t+1} to the all possible states at time t+4. Note that the path metrics α and β are only available at time instants t and $t+4$ due to the radix-16 design. And, the LLs are calculated by selecting a different combination of branches [12]. In our proposed design in this chapter, the radix-16 decoder will be composed of two level retimed radix-4 × 4 ACS units, and (4.5) can be also simplified to

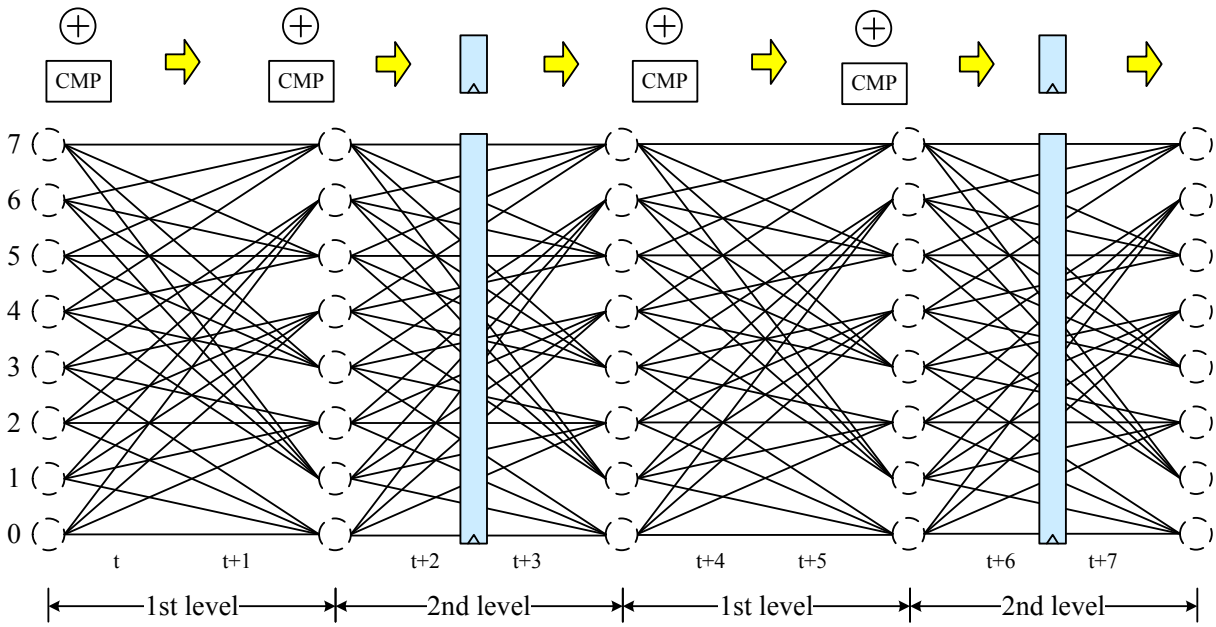


Figure 4.8: Retimed radix-4 × 4 ACS

be

$$LL_t^0(s) = \max_{S(\hat{\alpha}),S(\hat{\beta})} \{\hat{\alpha}_{t+2}(s_t, s_{t+2}) + \hat{\beta}_{t+2}(s_{t+2}, s_{t+4})\} \tag{4.6}$$

Now $S(\hat{\alpha})$ is the set of $\hat{\alpha}_{t+2}$ including the component $\gamma_t^0(s_t, s_{t+1})$, and $S(\hat{\beta})$ denotes the set of all $\hat{\beta}_{t+2}$ originated from s_{t+2}. As a result, the LLR for time t (LLR_t) can be derived by

$$LLR_t = \max_{s\in\bar{S}}\{LL_t^0(s)\} - \max_{s\in\bar{S}}\{LL_t^1(s)\} \tag{4.7}$$

where $\bar{S}$ represents the set of all states. Similarly, we can calculate LLR_{t+1}, LLR_{t+2}, LLR_{t+3} with the same approach. Note that the number of addition has been reduced from (4.6), resulting in less overhead caused by the retiming approach.

Finally, we use Fig. 4.9 as a simple example to illustrate the overall decoding process. Since radix-4 × 4 design is involved, there are four LLR's evaluated at one time instance through the retimed forward recursion $\hat{\alpha}$ and the retimed backward recursion $\hat{\beta}$. In addition, we should pay more attention on the beginning and end of the whole trellis. Since the value stored in the registers is different, the initial condition in (2.18) can be written as

$$\begin{aligned} &\hat{\alpha_0}(0, s') = 0, \quad \hat{\alpha_0}(s, s') = -\infty \quad \text{for } s \neq 0 \\ &\hat{\beta_N}(s, 0) = 0, \quad \hat{\beta_N}(s, s') = -\infty \quad \text{for } s' \neq 0 \end{aligned} \tag{4.8}$$

It indicates that the retimed forward recursions $\hat{\alpha_0}$ corresponding to the branches originating from the zero state and backward recursions $\hat{\beta_N}$ corresponding to the branches entering to the zero state will be set to be the maximum probability. From Fig. 4.9, we can further notice that the computation of the first $\hat{\alpha_0}$ will only consider two branches caused by the information bit u_0 and u_1. Therefore, the initial condition in (4.8) will be bypassed from first level retimed ACS unit to the second level. As a result, a extra bypass circuit will be introduced between two levels of the two-dimensional retimed ACS structure. However, the bypass circuit is composed by the simple logic gate and only one gate level delay will be introduced to the critical data path.

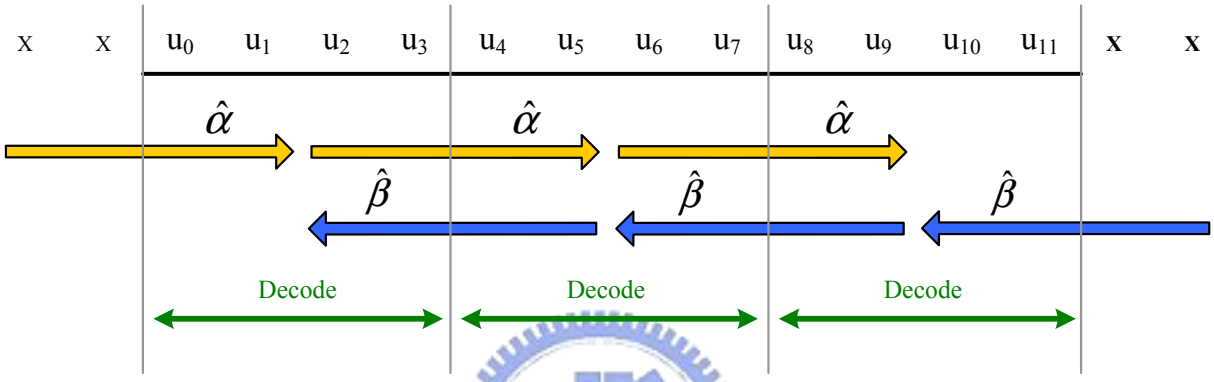


Figure 4.9: The decoding process of the modified MAP algorithm

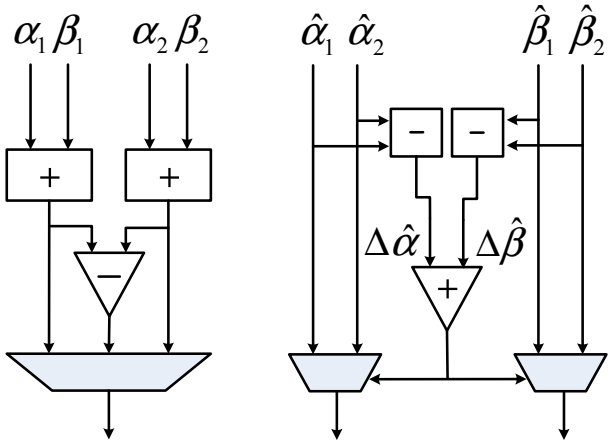
4.2.2 Compare-Select Operation Circuit

We adopt the modulo normalization scheme [13] to avoid path metric overflow. The overhead is the extra bit required for each ACS unit and metric storage.

Since only differences are meaningful in modulo normalization, the operation in (4.6) should be modified. The final LLR calculation in (4.7) is also the difference. Hence the original operation in Fig. 4.10(a) is modified to Fig. 4.10(b) where the α and β with larger $(\alpha + \beta)$ are selected without additions. After these selections, the LLR can be modified to

$$LLR_t = (\hat{\alpha}^0_{max} - \hat{\alpha}^1_{max}) + (\hat{\beta}^0_{max} - \hat{\beta}^1_{max}) \tag{4.9}$$

where $(\hat{\alpha}^0_{max}, \hat{\beta}^0_{max})$ and $(\hat{\alpha}^1_{max}, \hat{\beta}^1_{max})$ both are obtained from all possible states with Fig. 4.10(b). The modification in Fig. 4.10(b) guarantees the correctness of the function and causes no extra critical path delay, but introduces an additional multiplexer.



(a) Original (b) Modified

Figure 4.10: The selection circuit for the max operation in (4.6)

4.3 Chip Implementation Result

As discuss before, the fixed point representation of the internal variable in the MAP decoder is determined from the received symbol quantization. Fig. 4.11 shows the simulation result with the different input symbol quantization under the BPSK modulation. And the primary specifications of the MAP decoder are given in Table 4.1, where the code polynomial is followed 3GPP2 system [2]. We can observe that the quantized format $(3, 3)$ is suitable scheme for our case. Furthermore, the range of the $\Delta\alpha$, $\Delta\beta$, and $\Delta\gamma$ can be derived and we summarize the fixed representations in Table. 4.2.

The chip is fabricated with 1.2V, 0.13μm 1P8M CMOS technology, and the die photo is shown in Fig. 4.12. A delay lock loop (DLL) circuit is applied to generate internal clock whose frequency is four times the external frequency. Because of the large bandwidth, the chip use registers as storage elements instead of SRAM. The total core size is 1.96mm^2, where the DLL contains 0.063mm^2. The MAP decoder has 220K gates, and the chip density is 69.4%. After static timing analysis and post layout simulation, the decoder

Table 4.1: MAP Decoder Specification

code polynomial	$[\ 1\ \ \frac{1+D+D^3}{1+D^2+D^3}\ \ \frac{1+D+D^2+D^3}{1+D^2+D^3}\ ]$
code rate	1/3
sliding window size	20

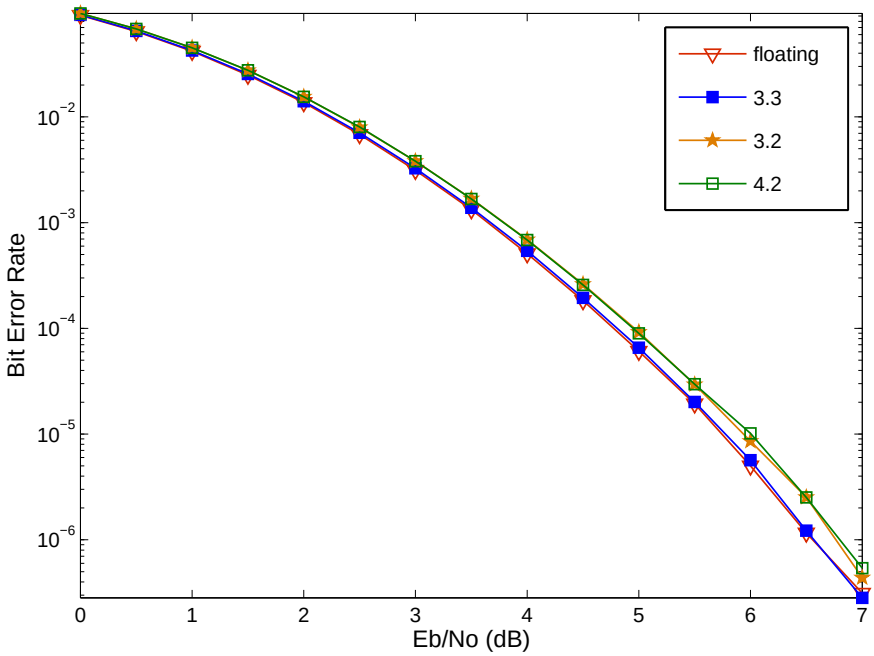


Figure 4.11: Fixed point simulation of the input symbol

Table 4.2: Summary of fixed representation in MAP decoder

quantities	input symbols	α	β	γ
width	6(3.3)	8(6.2)	8(6.2)	8(6.2)

achieves 952MS/s throughput under 1.08V power supply and the worst case corner. The estimation also includes the crosstalk analysis for signal wires that cause coupling noise. The power consumption is evaluated with 1.32V supply and with the switching activities generated at 952MS/s. We list the summary of simulation results and make a comparison with a MAP decoder proposed by Lee [14] in Table 4.3. The chip is still under measurement and there is an obstacle to examine the internal clock generated by the DLL. The measurement result until now is 15 MHz for the external clock rate when the DLL is working. However, if the clock signal bypass the DLL, the chip can work on the 100 MHz which is the maximum frequency that testing board can provide. The possible reasons leading to this problem are discussed as follows: First it the unknow loading of the testing board. Second, if external clock is not stable enough, it may cause the incorrect internal clock through the DLL. Finally, we do not consider the driving ability about the output of the DLL and hence the cad tools may generate a wrong clock tree originating from the DLL.

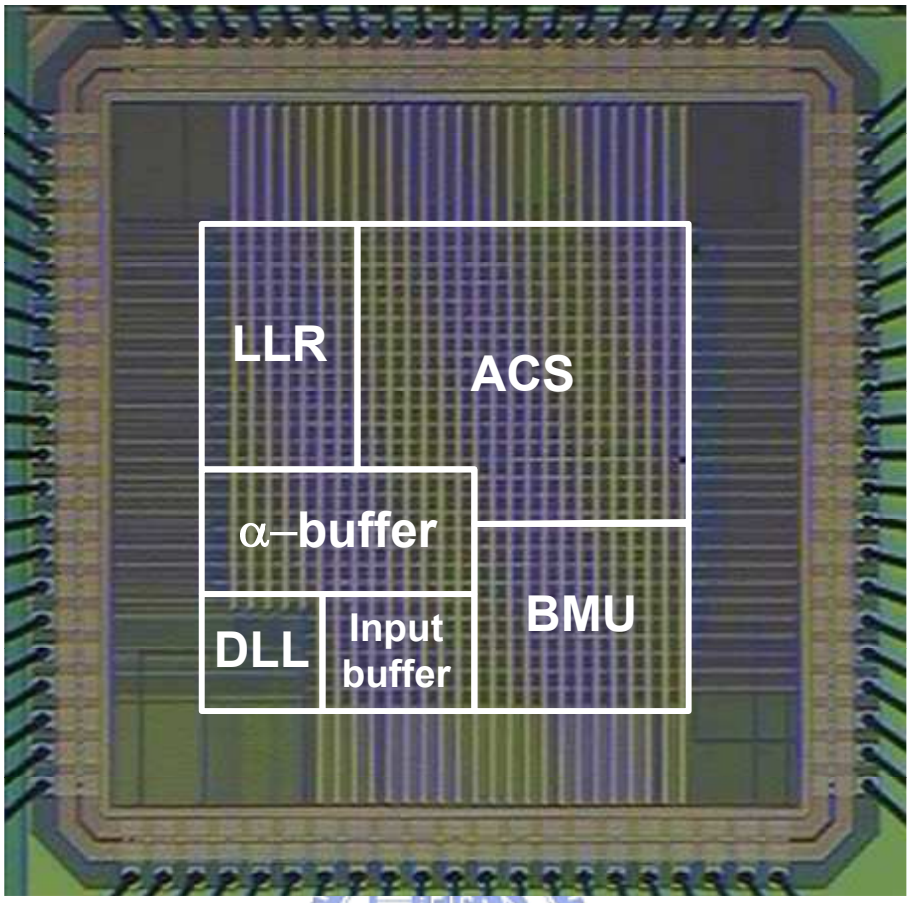


Figure 4.12: The die micrograph of the MAP decoder chip

Table 4.3: Summary of the MAP Decoder Chip

	Proposed	[14]
Technology	0.13μm	0.18μm
Operating Frequency	238MHz	285MHz
Date Rate	952(MS/s)	285(MS/s)
Average Power	528mW (1.32V)	330mW (1.8V)
Core Area	1.96 (mm^2)	8.7 (mm^2)
Gate Count	220K	N/A
Algorithm	Max-Log-MAP	Log-MAP

Chapter 5

The High-Speed Turbo Decoder Design

For a turbo decoder implementation, the whole decoding throughput is not only restricted by the computation time of its component decoder but also the interleaver (data block) size. Moreover, the effect of the latter factor will be critical under a long interleaver size design. It is due to the iterative decoding and interleaver structure, leading to the SISO decoders will not start decoding before the end of the previous iteration. In order to reduce the decoding delay, a shorter interleaver size may be used but at the expense of performance degradation.

In the following section, we will introduce a new interleaver design, termed inter-block permutation (IBP) interleaver [15], which provides a short decoding delay and impressive performance. Then, a high speed turbo decoder implementation, involving the IBP concept and the Max-Log MAP decoders referred in chapter 4, will be proposed. There are further some modification for the IBP interleaver under the hardware considerations.

5.1 The Inter-Block permutation interleaver

The IBP interleaver (IBPI) [15] is composed of two permutations: the first permutation, called intra-block permutation, is performed on the symbol sequence within a data block; while the second one, called inter-block permutation, is to interchange symbols in a block with neighboring blocks. Fig. 5.1 presents an example of the IBP interleaver

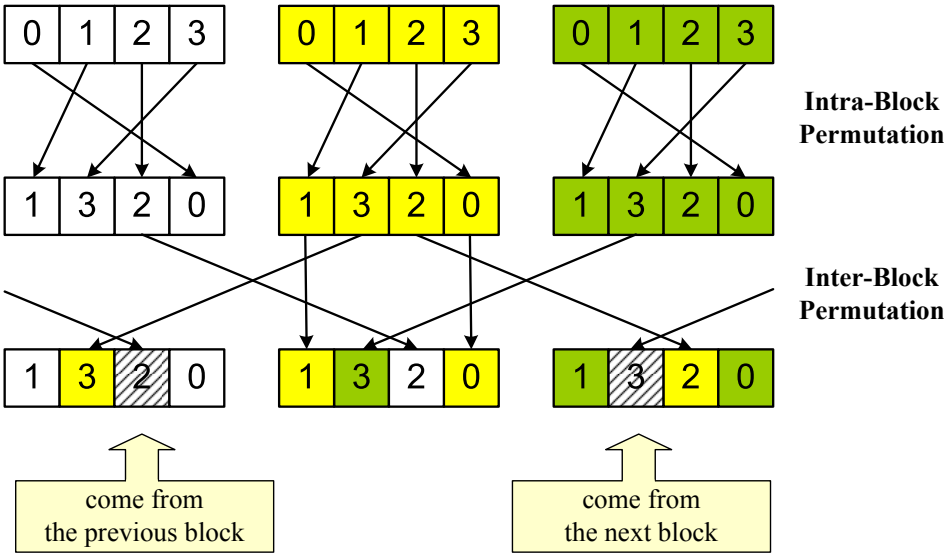


Figure 5.1: Graph representation of the IBP interleaver

process, where the permuted block not only associates with the original block but also the previous one and next one blocks. Note that the intra-block permutation can be built upon any existing block interleaver and hence the inter-block permutation interleaver is named from its characterized second permutation.

Since the IBP interleaver maps the symbols in a block to other blocks, we can consider its behavior as a class of message passing. As shown in Fig. 5.2, each circle is a data block and the colored ones represent message passing between different blocks. We can observe that the range of information transmission is proportional to the iteration number and leading to performance improvement. As a result, if a simple extra inter-block permutation properly designed, the IBP turbo codes will render significant performance gain.

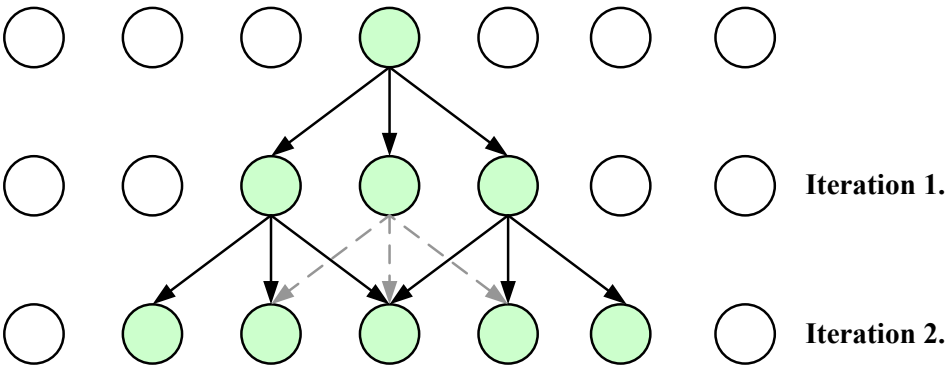


Figure 5.2: Message passing of the IBP intereleaver

5.2 Proposed Turbo Decoder Design

As described in previous section, the inter-block permutation interleaver interchange the information between each blocks to improve the performance. Therefore, we can adopt short block size and decoding several blocks in parallel to reduce the large decoding latency. Fig. 5.3 shows a simple block diagram of the proposed turbo decoder. This architecture consist of three main units: SISO decoders, memory units, and network. Each SISO decoder is based on the Max-Log MAP algorithm and structured as the radix-2×2 retiming ACS, while the memory units are used to store the input and output data corresponding to the SISO decoders. Note that the number of the SISO decoders and memory units is the same, as the power of 2, which is estimated for the trade-off between area and throughput. Finally, the network is a permutation architecture which interchanges the input symbols and extrinsic informations between each data blocks. And it realize the main concept of the IBP algorithm. In the following subsections, we will describe each part of the proposed turbo decoder in detail.

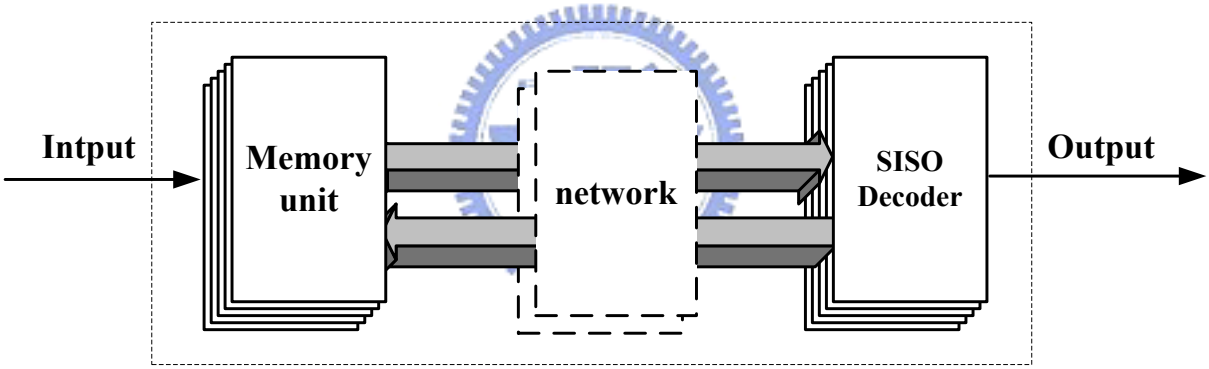


Figure 5.3: Block diagram of Turbo decoder

5.2.1 IBPI with Butterfly Structure

For the IBPI referred to in the section 5.1, the massage passing span, means the range of the information interchange, will be extended more widely as the decoding iteration increasing and further improve the performance. However, the routing congestion problem in the physical layer will be the obstacle for the hardware implementation and the circuit to control the connection between SISO decoders and memory blocks may also be complex.

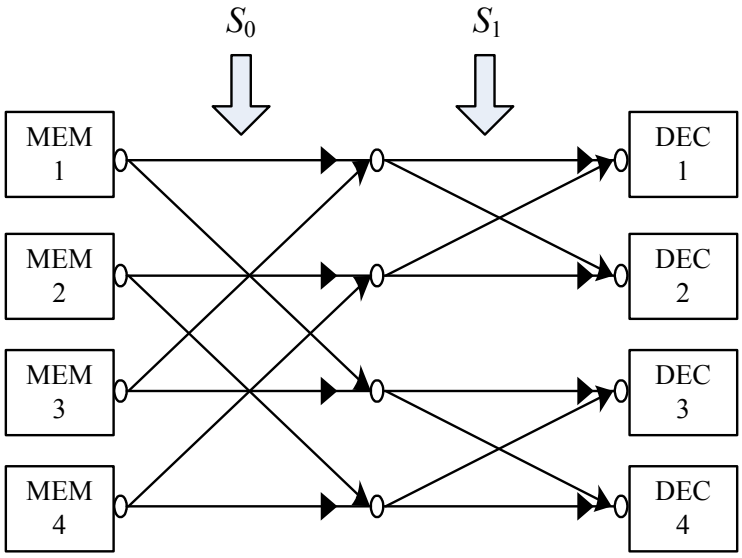


Figure 5.4: The butterfly network structure

Thus, we utilize a network as our inter-block permutation structure to simplify hardware complexity and provide a more feasible hardware implementation solution. A proposed network, termed butterfly structure, is presented in Fig. 5.4. Now considering the number of SISO decoders as well as memory units defined as N, where N is equal to 2^M, the butterfly network well be divided into M level. And each level is composed with N switches controlled by the same single-bit signal. We can see that Fig. 5.4 is a example for $M = 4$ and $N = 2$ and controlled by the signals (s_0, s_1). Consequently, the control logic of the butterfly network is quite simple and this connection may be more acceptable for chip implementation, since each node in the network only associates with two nodes. As mentioned before, the overall IBPI consists of intra-block permutation and inter-block

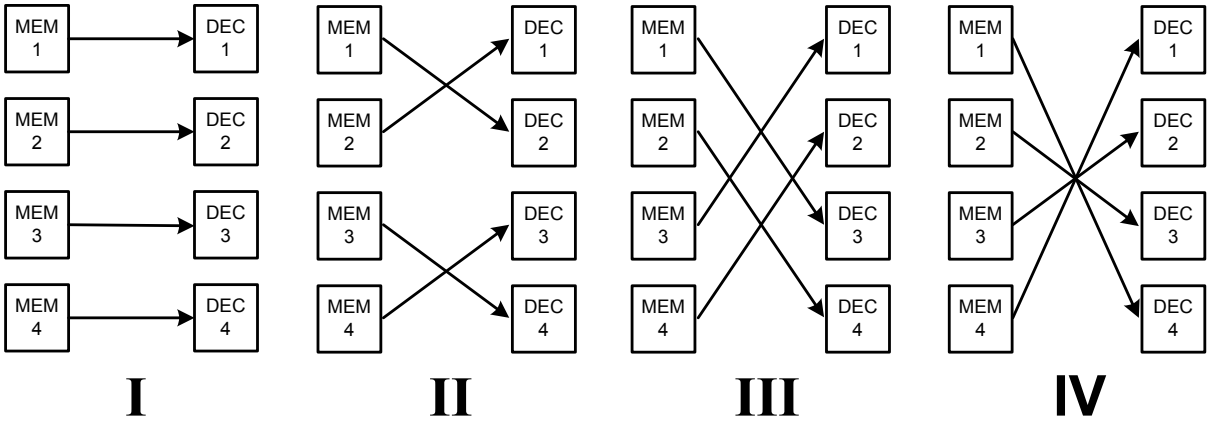


Figure 5.5: Graph representation of the IBP process

permutation. The former can be realized through the addressing of each memory units and the latter is through the proposed butterfly network, which will be illustrated as follows. Fig. 5.5 shows a graph representation of the all kinds of the connections corresponding to the network in Fig. 5.4. Here, we assume the switches pass the data forward with the control signal valued as 0 and turn on with the control signal valued as 1. As a result, there are four types of connection: Type I for $(s_0, s_1) = (0, 0)$, Type II for $(s_0, s_1) = (0, 1)$, Type III for $(s_0, s_1) = (1, 0)$, and Type IV for $(s_0, s_1) = (1, 1)$. And we can use these connections to perform several different inter-block permutation. Finally, using a flow diagram to illustrate the overall procedure of the proposed IBP interleaver architecture. As shown In Fig. 5.6, we have four data block denoted by the A, B, C and D respectively and each block size is equal to 8. The whole IBPI will first perform a intra-block permutation to rearrange the order within each block and in turn a inter-block permutation through the butterfly network referred to the connections in Fig. 5.5.

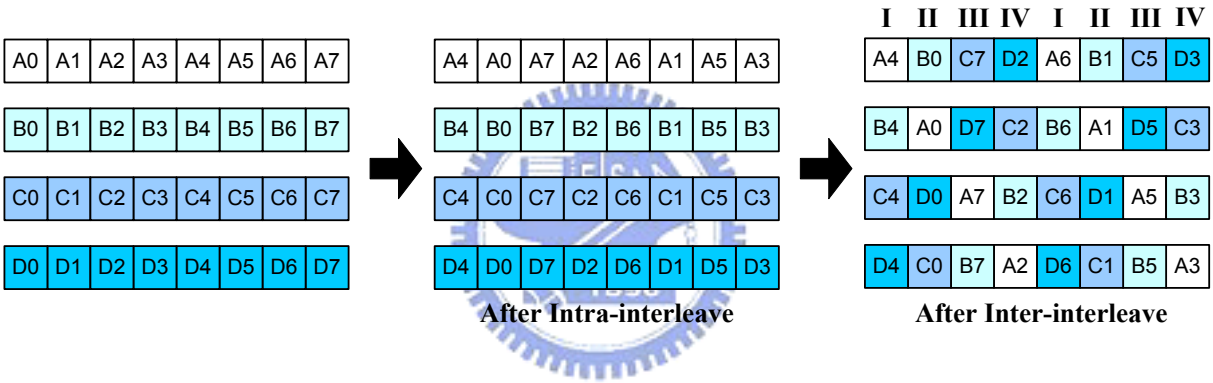


Figure 5.6: The overall procedure of the IBP interleaver

Note that the inter-block permutation in this example is according to a sequence [I II III IV] repeatedly, which indicates the order of which connection type is used. This sequence, called as IBP sequence, will be an input signal in our design, which represents that we can control the inter-block permutation of the interleaver outside the whole turbo codec. Moreover, the IBP sequence can also be a security key for the communication between the encoder the the decoder.

Instead of developing a control mechanism to handle the complicated connections between component decoders and memory units, our proposed IBPI architecture is based on a simple butterfly network and utilize its hardware structure to implement the behavior of inter-block permutation. However, note that in this architecture the message passing

span is not depends on the decoding iteration but the number of SISO decoders (memory units). It represents that fewer component decoders may lead to the performance degradation. Since the turbo decoder in this thesis is considered for the high speed issue, several SISO decoders, up to 16 or 32, will be used for parallel process and provide a large enough message passing span. Therefore, our proposed decoder can achieve high throughput and good performance under modest hardware cost.

5.2.2 SISO decoder

Since several SISO decoders is involved in the inter-block permutation turbo decoder, the Max-Log-MAP decoder with retimed radix-2 $\times$ 2 ACS structure, as described in chapter 4, will be a compromise between throughput and area cost. Moreover, there are some modification introduced to further reduce its complexity. To consider the sliding window approach shown in Fig. 2.3, the backward metrics β evaluation can be started until the required window of data have been stored. However, if we reverse the order of input sequence within a sub-block, the input buffer of the β_1 computation can be saved [16]. Fig. 5.7 is a graphical representation which illustrate the processing of this modified architecture. The black dashed line represents the writing input sequence into the input buffer. Note that at the same time the gray dashed line, means the β_1 computation, can be also executed immediately since the sequence is in the reversed order. Then the α recursion, denoted by the black solid line, is performed on the previous stored data

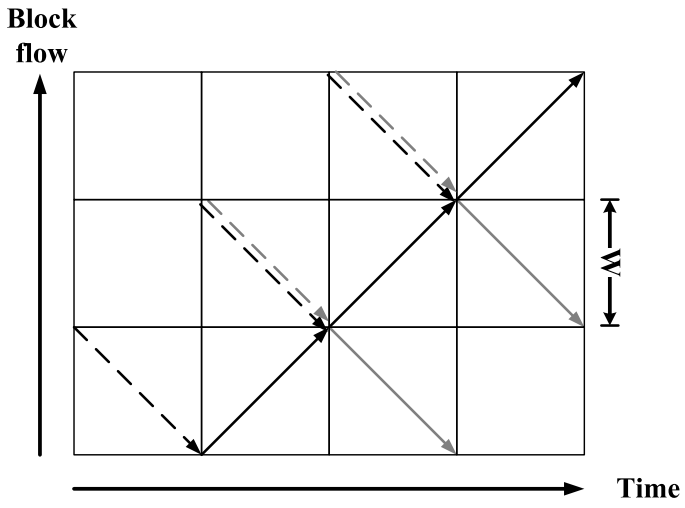


Figure 5.7: The graphical representation of the processing with the two input buffers

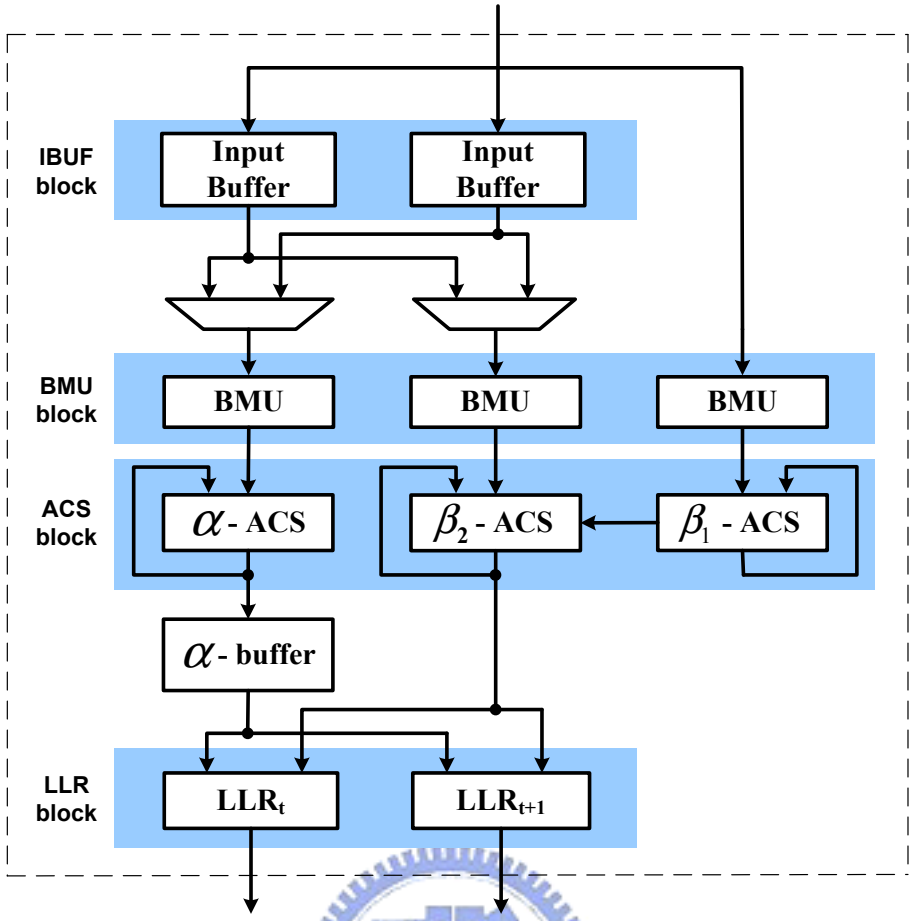


Figure 5.8: The block diagram of the radix-2x2 Max-Log-MAP decoder

and the second window data will be written into the other input buffer. Finally, the first window data will be read for backward metrics β_2 calculation and the third window data will be written into the same input buffer simultaneously. As a result, The block diagram of the modified Max-Log-MAP decoder is shown in Fig. 5.8. Note that it only requires two input buffer, which output the data to the α and β_2 computation units by turns via multiplexers. Furthermore, the radix-2 $\times$ 2 design is included.

As described before, utilizing the dummy tail bits to terminate the decoding trellis may decrease the code rate. Since in our proposed IBP turbo decoder the shorter block length is adopted, the drawback resulted from tail bits will be more serious. Thus, the tail-biting idea [17], which avoids the rate loss without suffering from performance degradation due to the end of the block, is considered. A valid codeword in the tail-biting trellis will cause the encoder to start and end in the same any possible state, instead of zero state only. Therefore, a dummy sub-block, as well as the last sub-block, will be calculated in

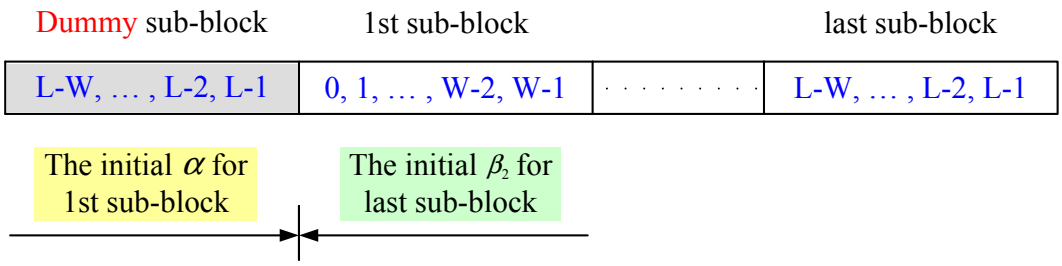


Figure 5.9: The decoding process for tail-biting trellis

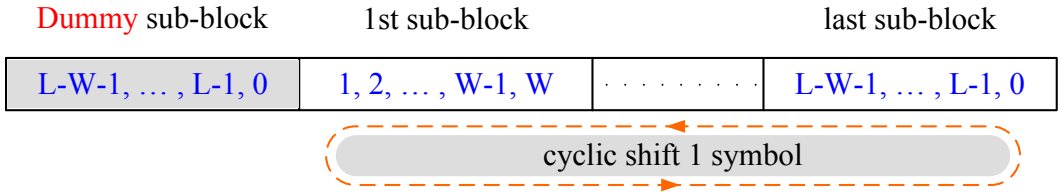


Figure 5.10: The cyclic shifted input sequence order of the SISO decoder

advance to estimate the unknow initial condition of the forward metrics α in the first sub-block. Similarly, the β computation of the first sub-block can also be a estimation for the initial condition of the backward metrics in the last sub-block. The decoding process for the tail-biting trellis mentioned above is shown in Fig. 5.9, where W is defined as sliding window size and L is defined as block length. Note that the estimation is based on the trellis which starts and ends at the same state.

In addition, the tail-biting can also simplify some hardware complexity within the retimed two-dimensional ACS structure and further reduce its critical path delay. In the section 4.2.1, a extra logic should be introduced to bypass the initial condition. However, it may lead to some additional routing difficulty and increase the path delay. In Fig. 5.10, we perform a cyclic shift on the input sequence order to improve this problem. Here the dummy sub-block is still equivalent to the last sub-block. Note that we only shift one symbol, instead of two, since radix-2 $\times$ 2 design is considered. Therefore, the 0th symbol is moved to the end of the dummy sub-block.

As a summary, to associate with the modification described in the beginning of this

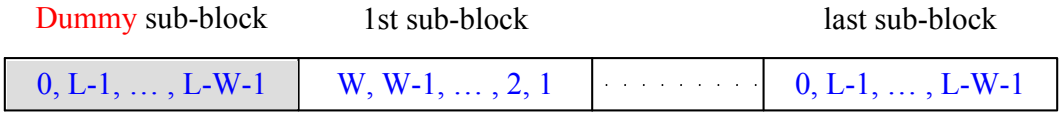


Figure 5.11: The final input sequence order of the SISO decoder

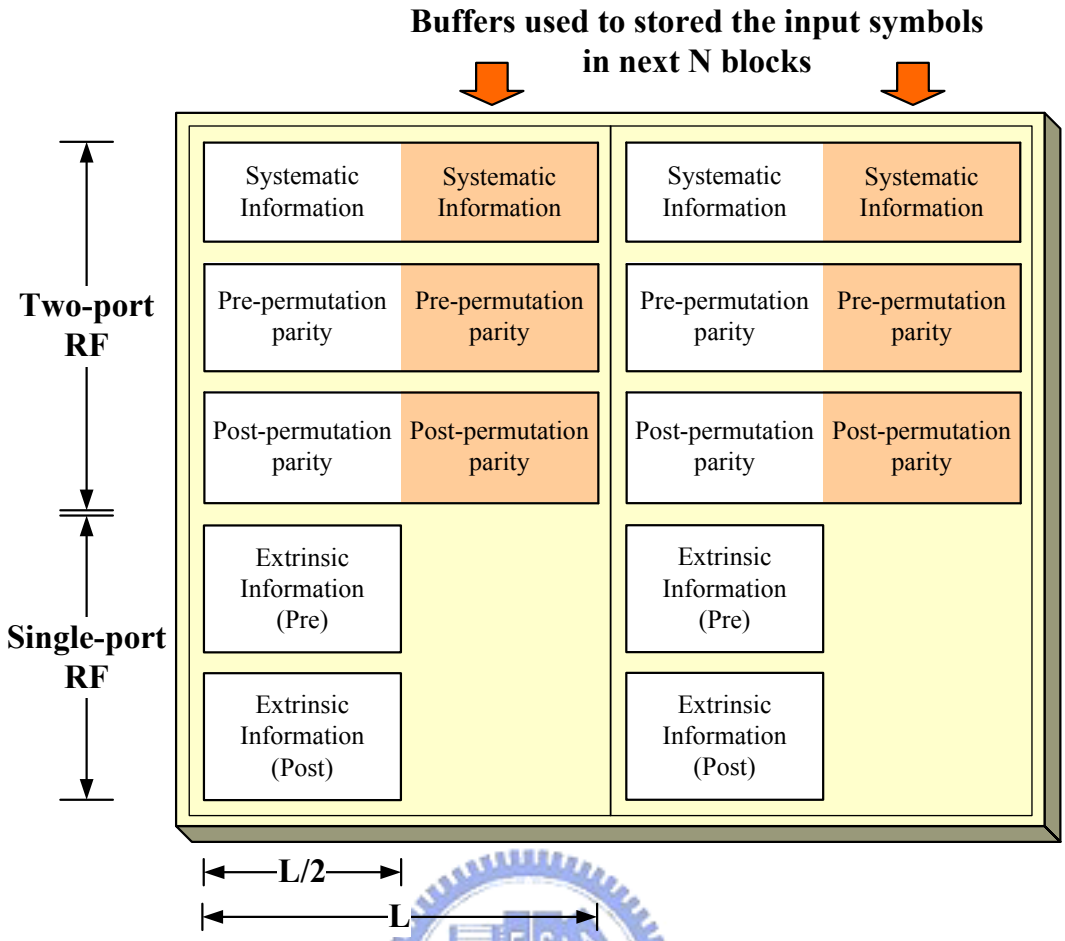


Figure 5.12: The memory unit

subsection, the final input sequence order transmitted from memory units to SISO decoders is represented in the Fig. 5.11. These modification can be achieved easily through the addressing of memory unit and no extra hardware control need to be involved.

5.2.3 Memory unit

Under the area cost consideration, the memory unit is composed of several register files, which store different input and output information for the SISO decoders. In a memory unit, we use three two-port register files (TP-RFs) for storing the encoded codeword, including systematic information, parity check corresponding to the original sequence and parity check corresponding to the interleaved sequence. In addition, there are two single-port register files (SP-RFs) for storing the extrinsic informations obtained from the SISO decoders in the pre-permutation round and post-permutation decoding round respectively.

Since the radix-2×2 Max-Log-MAP decoder is used as the SISO decoder in our design, the data bandwidth of the each memory unit will be doubled. Therefore, each register file referred to before is partitioned into two parts. The final memory unit allocation, as shown in Fig. 5.12, consists of six TP-RFs with depth of L and four SP-RFs with depth of L/2, where L is defined as the data block size. Note that the TP-RFs in all N memory units are also served as buffers to store the input symbols of next N block in advance, which leads to depth of each TP-RF is twice as that of the SP-RF. We can illustrate it using the decoding schedule diagram in Fig. 5.13. During the data of the current N blocks are read out from TR-RFs for decoding, the input symbols of the next N blocks will also be written into TR-RFs simultaneously. Consequently, only the latency at the beginning of the decoding process will be introduced and the decoding latency due to re-fetching the input symbols between every N data blocks can be saved.

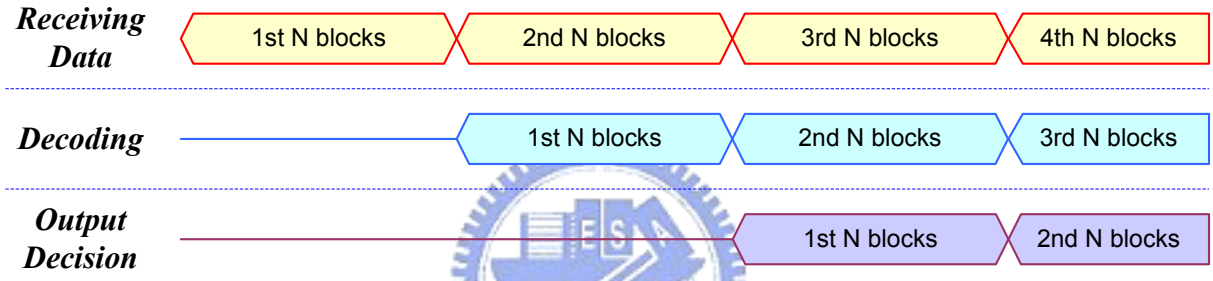


Figure 5.13: The decoding schedule diagram

5.3 Chip Implementation

Based on the architectures described above, we proposed a high-throughput turbo decoder based on the inter-block permutation interleaver. Fig. 5.14 shows the block diagram of our proposed turbo decoder. Consider the trade-off between the throughput and hardware cost, we utilize 32 memory units and 32 SISO decoders based on the retimed radix-2 × 2 ACS structure. In addition, a delay lock loop (DLL) circuit is also applied to generate internal clock whose frequency is four times the external frequency. The primary specifications of the turbo decoder are given in Table 5.1 and the fixed point representation is summarized in 5.2. The code polynomial we adopted is equal to the 3GPP standard and the quite short block length is used. After the simulation as shown

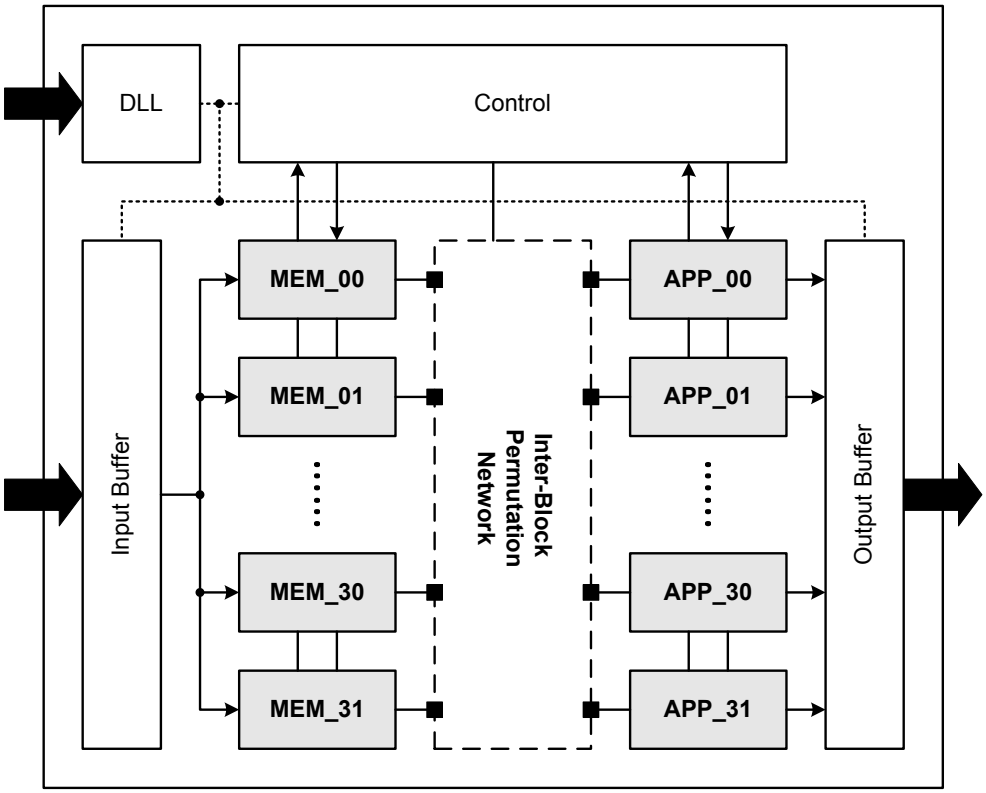


Figure 5.14: The block diagram of the proposed IBP turbo decoder

in Fig. 5.15, a appropriate decoding iteration number is decided for performance and decoding throughput. Note that the scaling factor approach [18] is applied as compensation for the performance loss due to the sub-optimal Max-Log-MAP algorithm. Finally, we compare the performance of the proposed turbo decoder with the 3GPP standard turbo code in Fig. 5.16. Since we interchange information between 32 component decoders, it represents that the message span range is equivalent to the 4096. Therefore, the block length of 3GPP standard turbo code in Fig. 5.16 is 4096 for the fair comparison.

A test chip has been implemented in a 1.2V, 0.13μm 1P8M CMOS technology, and the layout view is shown in Fig. 5.17. The chip size is 25 mm^2 while the core occupies 17.808 mm^2. The total gate count is 2.67M including memory units based on the the RFs and the chip core density is about 91%. After static timing analysis and post layout simulation, the turbo decoder achieves 1.06 Gb/s throughput with 8 iterations under 1.08V power supply and the worst case corner. The estimation also includes the crosstalk analysis for signal wires that cause coupling noise. Table. 5.3 gives the characteristic summary of the test chips.

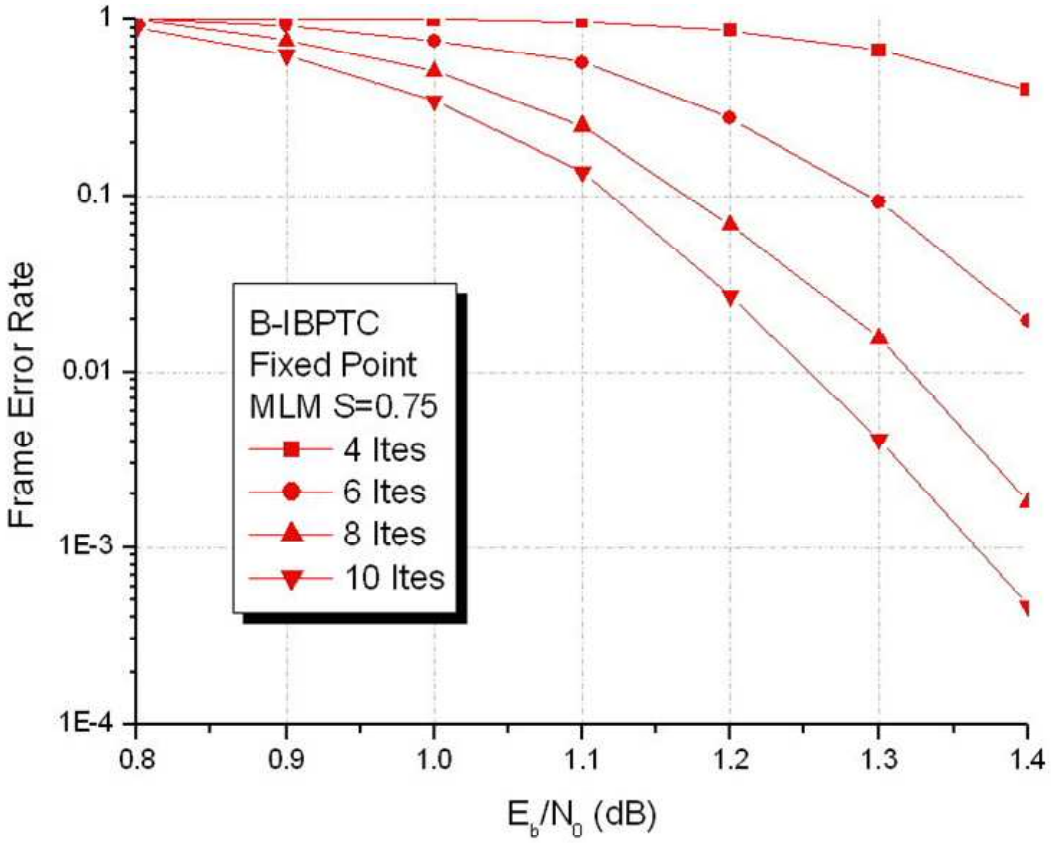


Figure 5.15: The iteration number simulation

Table 5.1: Turbo Decoder Specification

code polynomial	$[\ 1\ \frac{1+D+D^3}{1+D^2+D^3}]$
block size	128
sliding window size	32
iteration number	8
scaling factor	0.75
SISO decoder algorithm	Max-Log-MAP algorithm

Table 5.2: Summary of fixed representation in Turbo decoder

quantities	input symbols	$L_a(u_t)$	α	β	γ
width	6(3.3)	6(4.2)	8(6.2)	8(6.2)	8(6.2)

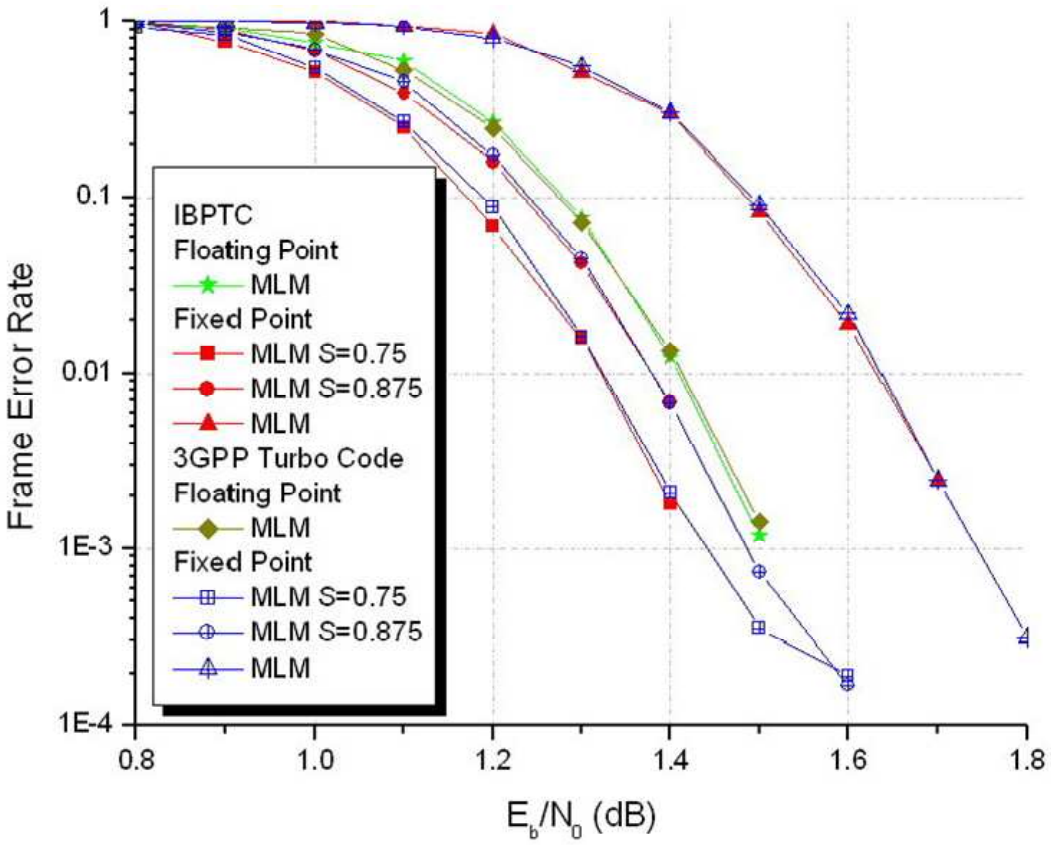


Figure 5.16: Comparison with 3GPP standard turbo code

Table 5.3: Summary of the Turbo decoder chip

Technology		0.13μm 1P8M CMOS
Operating Frequency		265MHz
Date Rate(8 iteration)		1.06(Gb/s)
Gate Count		2.67M
Area	Core Size	17.808mm^2
	Memory	3.328mm^2
	DLL	0.054mm^2
Power consumption		508mW (1.2V)

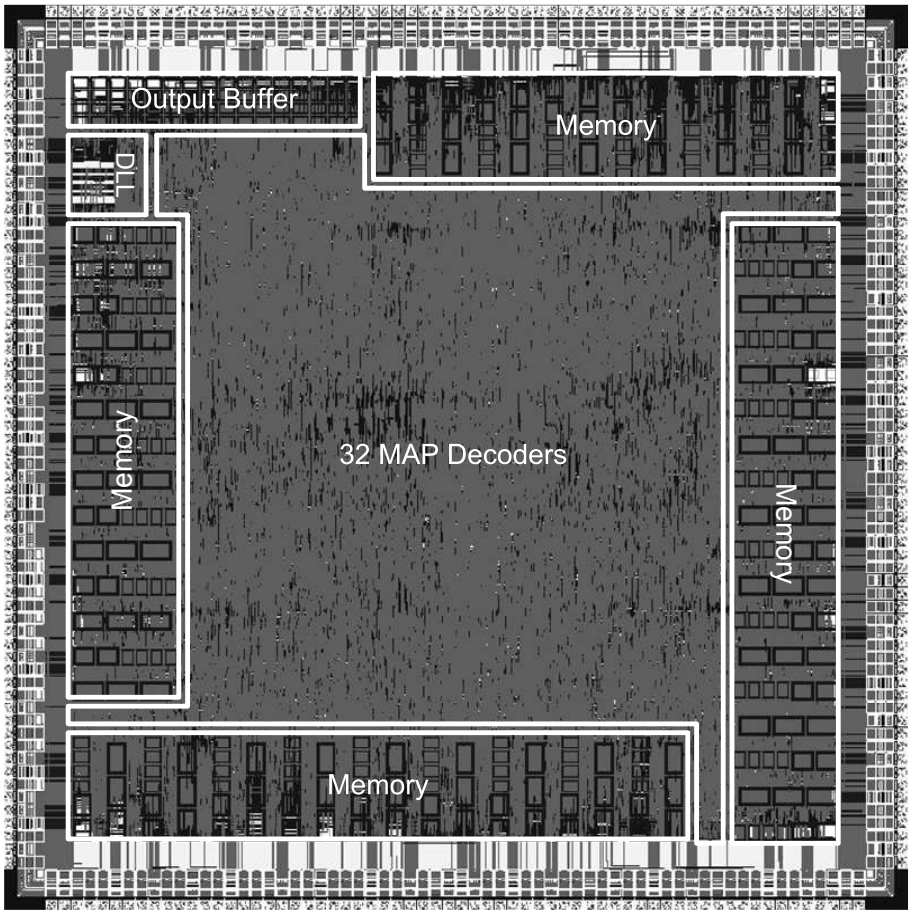


Figure 5.17: Layout view of the proposed Turbo code decoder

Chapter 6

Conclusion

In this thesis, the high throughput and area-efficient MAP decoder is first proposed. The two-dimensional architecture is introduced for high-radix trellis structure, and the retimed ACS units are applied for higher clock frequency, leading to much higher data rate. The hardware overhead is minimized because of the two-dimensional ACS architecture. After chip implementation in 0.13μm 1P8M technology, the 1.96mm^2 core area can support the maximum 952MS/s data rate.

Second, we propose a efficient hardware architecture for implementation of the inter-block permutation turbo decoder and further combine with retimed radix-2 $\times$ 2 Max-Log-MAP decoder as its component decoder. A butterfly structure is proposed as the network to interchange the symbols between each block, which avoid too complex control and reduce the hardware complexity considerably. Since the parallel decoding is feasible under IBP interleaver, several Max-Log-MAP decoder are included to achieve high throughput under modest area cost. Finally, a turbo decoder implementation in 0.13μm 1P8M technology can achieve 1.06 Gb/s throughput with 8 decoding iterations.

Bibliography

[1] *Technical Specification Group Radio Access Network; Multiplexing and channel coding (FDD)*, 3GPP TS 25.212 Std. V3.11.0, 2002.

[2] *Physical Layer Standard for cdma2000 Spread Spectrum Systems*, 3GPP2 Std. C.S0002-C, 2002.

[3] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol," *IEEE Trans. Inform. Theory*, no. IT-20, pp. 284–287, Mar. 1974.

[4] J. Hagenauer and P. Hoeher, "A viterbi algorithm with soft-decision output and its applications," in *IEEE CLOBE-COM*, Dallas, Nov. 1989, pp. 47.1.1–47.1.7.

[5] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal map decoding algorithms operating in the log domain," in *Proc. ICC'95*, Seattle, June 1995.

[6] J. A. Erfanian, S. Pasupathy, and G. Gulak, "Reduced complexity symbol detectors with parallel structures for isi channels," *IEEE Trans. Commun.*, vol. 42, no. 2/3/4, pp. 1261–1271, Feb./Mar./Apr. 1994.

[7] S. A. Barbulescu, "Sliding window and interleaver design," in *IEEE Electronics letters*, vol. 37, no. 21, Oct. 2001, pp. 1299–1300.

[8] L. C. Perez, J. Seghers, D. Costello, and Jr., "A distance spectrum interpretation of turbo codes," *IEEE Trans. Inform. Theory*, vol. 42, no. 6, pp. 1698–1709, Nov. 1996.

[9] Y. Wu, B. D. Woener, and T. K. Blankenship, "Data width requirements in SISO decoding with modulo normalization," *IEEE Trans. Commun.*, vol. 49, no. 11, pp. 1861–1868, Nov. 2001.

[10] I. Lee and J. L. Sonntag, "A new architecture for the fast Viterbi algorithm," *IEEE Trans. Commun.*, vol. 51, no. 10, pp. 1624–1628, Oct. 2003.

[11] J. Han, A. Erdogan, and T. Arslan, "High speed Max-Log-MAP turbo SISO decoder implementation using branch metric normalization," in *IEEE Computer Society Annual Symposium on VLSI*, 2005, pp. 173–178.

[12] M. Bickerstaff, L. Davis, C. Thomas, D. Garrett, and C. Nicol, "A 24Mb/s radix-4 logMAP turbo decoder for 3GPP-HSDPA mobile wireless," in *IEEE Int. Solid-State Circuit Conf. (ISSCC) Dig. Tech. Papers*, 2003, pp. 151–484.

[13] C. Shung, P. Siegel, G. Ungerboeck, and H. Thapar, "VLSI architertures for metric normalization in the Viterbi algorithm," in *Int. Conf. Communications*, vol. 4, Atlanta, CA, Apr. 1990, pp. 1723–1728.

[14] S. Lee, N. Shanbhag, and A. Singer, "A 285-MHz pipelined MAP decoder in 0.18μm CMOS," *IEEE J. Solid-State Circuits*, vol. 40, no. 8, pp. 1718–1725, Aug. 2005.

[15] Y.-X. Zheng and Y. T. Su, "A new interleaver design and its application to turbo codes," in *Proc. VTC2002fall*, vol. 3, Sep. 2002, pp. 1437–1441.

[16] G. Masera, M. Mazza, G. Piccinini, F. Viglione, and M. Zamboni, "Architectural strategies for low-power VLSI turbo decoders," *IEEE Trans. on VLSI Systems*, vol. 10, no. 3, pp. 279–285, June 2002.

[17] C.Weiss, C. Bettstetter, S. Riedel, and D. Costello, "Turbo decoding with tail-biting trellises," in *Proc. IEEE Int. Symp. Signals, Syst., Electron.*, Pisa, Italy, Oct. 1998, pp. 343–348.

[18] J. Vogt and A. Finger, "Improving the max-log-map turbo decoder," *Electronics Letters*, vol. 36, pp. 1937–1939, Nov. 2000.