A Memory-Efficient Block-wise MAP Decoder Architecture

Sik Kim, Sun-Young Hwang, and Moon Jun Kang

Next generation mobile communication system, such as IMT-2000, adopts Turbo codes due to their powerful error correction capability. This paper presents a block-wise maximum a posteriori (MAP) Turbo decoding structure with a low memory requirement. During this research, it has been observed that the training size and block size determine the amount of required memory and bit-error rate (BER) performance of the block-wise MAP decoder, and that comparable BER performance can be obtained with much shorter blocks when the training size is sufficient. Based on this observation, a new decoding structure is proposed and presented in this paper. The proposed block-wise decoder employs a decoding scheme for reducing the memory requirement by setting the training size to be N times the block size. The memory requirement for storing the branch and state metrics can be reduced 30% to 45%, and synthesis results show that the overall memory area can be reduced by 5.27% to 7.29%, when compared to previous MAP decoders. The decoder throughput can be maintained in the proposed scheme without degrading the BER performance.

Keywords: MAP decoder, training length, memory requirement.

Manuscript received Aug. 11, 2003; revised Mar. 18, 2004.

This research was supported by the MIC (Ministry of Information and Communication), Korea, under the ITRC (Information Technology Research Center) support program supervised by IITA (Institute of Information Technology Assessment).

Sik Kim (phone: +82 31 209 4263, email: s90.kim@samsung.com) and Sun-Young Hwang (email: hwang@ccs.sogang.ac.kr) are with the Electronic Engineering Department, Sogang University, Seoul, Korea.

Moon Jun Kang (email: malas@eecad.sogang.ac.kr) is with the Pinetron Co. Ltd., Seoul, Korea.

I. Introduction

Since Shannon announced the channel limit theorem in 1948, numerous channel codes for various applications have been developed [1]. Even though most of them failed to reach the Shannon limit, the Turbo code, proposed by Berrou in 1993, shows error correcting performance close to the Shannon limit [2]. However, the Turbo code has been used in a limited number of applications where real-time processing is not required, such as satellite communications, due to its hardware complexity and decoding delay. Since then, a great deal of research effort has been taken to improve the performance of the Turbo code. As a result, the Turbo code has been adopted in the IMT-2000 system for high data rate transmission.

A Turbo encoder has a simple architecture, generating parity information together with input data by two constituent encoders. Each of the two encoders encodes the input data and scrambles input data through an interleaver. A Turbo decoder performs iterative decoding to enhance the error correction capability. A Turbo decoder consists of two soft input/soft output (SISO) decoders, each of which takes a-priori information computed from the other SISO decoder as an input for iterative decoding [3]. Two SISO decoding algorithms have been popularly used: soft output Viterbi algorithm (SOVA) and maximum a posteriori (MAP) algorithm. Even though the decoder realizing the MAP algorithm is four times as complex as that for the SOVA, it shows that the BER performance is twice as high as that of SOVA [4]. Due to recent progress in fabrication and circuit design technology, the BER performance becomes more of a concern than hardware complexity. Accordingly, the MAP algorithm is preferred to the SOVA. A log-MAP decoding algorithm has been proposed to reduce computation complexity of the MAP algorithm, and the

block-wise MAP decoding algorithm has been proposed to reduce memory usage, which is known to be proportional to the frame length [5], [6].

The block-wise MAP decoding algorithm requires less memory than the original MAP decoding algorithm. However, it requires a training process determining the initial backward state metric to avoid the BER performance degradation caused by backward state metric discontinuity [5]. In the MAP algorithm, training size as well as block size affects the BER performance. The previous block-wise MAP decoding algorithm uses the same training size as the block size for higher hardware utilization [6]. In this research, it has been found that the BER performance could be maintained with shorter blocks when the training size is sufficiently large enough. This paper proposes an efficient decoding scheme where the block size is set to 1/n of the training size. For the efficient implementation of the proposed scheme, a pipelined architecture is also proposed.

II. Turbo Code and MAP Algorithm

This section presents a brief description on the structure of the Turbo code and the MAP algorithm.

Figure 1 shows an overall structure of the Turbo encoder having a 1/2 coding ratio. It consists of two encoders, one of which encodes the input bit sequence while the other encodes the bit sequence obtained by interleaving the input bit sequence. Input bit sequence D_k is directly output together with encoded bit sequence Y_k. At time k, encoder 1 produces encoded output sequence Y_k^1 using input sequence D_k, and encoder 2 produces output sequence Y_k^2 by encoding D_i, which is obtained by interleaving the input bit sequence D_k. To construct a half-code rate, Y_k could be generated using a puncturing method between Y_k^1 and Y_k^2 [2].

Constituent encoders employ the recursive systematic convolutional code. The outputs of the systematic convolutional code consist of the original input data and encoded bit sequence Y_k. It is well known that the BER performance of the systematic convolutional code is better than that of the non-systematic convolutional code in low signal-to-noise ratio (SNR) channel environments [2]. The Turbo code also shows a remarkable BER performance because the recursive systematic convolutional code has an infinite impulse response [7].

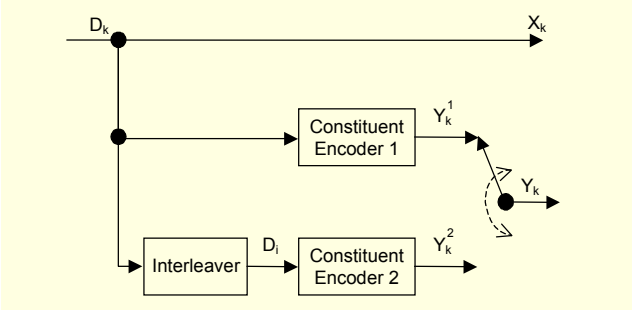

Fig. 1. The overall structure of the Turbo encoder.

Figure 2 shows the overall structure of the Turbo decoder. The two constituent SISO decoders (DECs) perform the backward process of the two encoders in the Turbo encoder. The interleaver and de-interleaver reassemble the information bit sequence, and the hard decision block determines and generates the decoded bit sequence. This architecture has a recursive structure for iterative decoding [1], [3], [4]. Decoder input bit sequences x_k and y_k, consisting of y^1_k and y^2_k, may include channel error. SISO DEC1 uses x_k, y^1_k and a priori information $L_{a1}(d_k)$ as input. In the initial decoding, $L_{a1}(d_k)$ is set to '0' because there exists no a posteriori probability value. After that, $L_{a1}(d_k)$ is supplied from extrinsic information of the SISO DEC2 output. Then, SISO DEC1 output $L_{e1}(d_k)$ is reassembled by interleaver and fed to SISO DEC2 together with y^2_k. After SISO DEC2 output $L_2(d_n)$ is reassembled to form $L_2(d_k)$ by the de-interleaver, the hard decision block generates $\hat{d}_k$. SISO DEC2 output $Le_2(d_n)$ is reassembled by de-interleaver and fed to SISO DEC1 for iterative decoding.

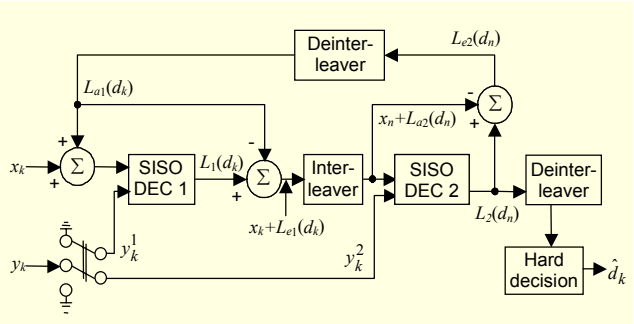

Fig. 2. The overall structure of the Turbo decoder.

1. MAP Decoding Algorithm

Bahl proposed the MAP algorithm [8]. Though the design complexity of MAP decoder is four times as complex as that of the SOVA decoder, researches have been performed to implement the MAP decoder because the BER performance of the MAP algorithm is twice as high as that of SOVA. The input sequence consisting of information bits x_k and parity bits y_k may include additive white Gaussian noise at time k. The MAP decoder output, the log-likelihood ratio of information bits d_k, can be derived from (1) and (2) using Bahl's MAP decoding algorithm,

$$\alpha_k^m = \sum_{j=0}^{1} \alpha_{k-1}^{b(j,m)} \delta_{k-1}^{j,b(j,m)}$$
$$\beta_k^m = \sum_{j=0}^{1} \beta_{k+1}^{f(j,m)} \delta_k^{j,m} \quad , \qquad (1)$$
$$\delta_k^{i,m} = \pi_k^i \exp\left(\frac{x_k u_k^i + y_k v_k^{i,m}}{\sigma^2} \right)$$

where $\delta_k^{i,m}$ represents the branch metrics and α_k^m, β_k^m represent the forward and backward state metrics, respectively.

$$L(d_k) = \log \frac{\sum_m \alpha_k^m \delta_k^{1,m} \beta_{k+1}^{f(1,m)}}{\sum_m \alpha_k^m \delta_k^{0,m} \beta_{k+1}^{f(0,m)}} \qquad (2)$$

The decoding process is as follows: Branch metric $\delta_k^{i,m}$ and for ward state metrics α_k^m are calculated from the input frame. Then, the backward state metrics β_k^m and the log-likelihood ratio of d_k (output of MAP decoder) are computed by using $\delta_k^{i,m}$ and α_k^m.

2. MAP Decoder Implementation

In most MAP decoders, the log-MAP decoding algorithm has been used to reduce the computation complexity of the MAP decoding algorithm by computing the metrics in the log domain [9]. This algorithm requires memory proportional to the frame size for storing the branch metrics and the forward state metrics [3], [7], [10]. The block-wise MAP decoding algorithm has been proposed to reduce the memory requirement in the log-MAP decoding algorithm. It has been shown that the block-wise log-MAP decoding algorithm can be implemented in a smaller area. Figure 3 shows the block-wise MAP decoding scheme. In this scheme, a data frame is divided into M sub-blocks with size L, and each block is fed to the MAP decoder one by one. In this way, memory requirement is reduced [5], [6].

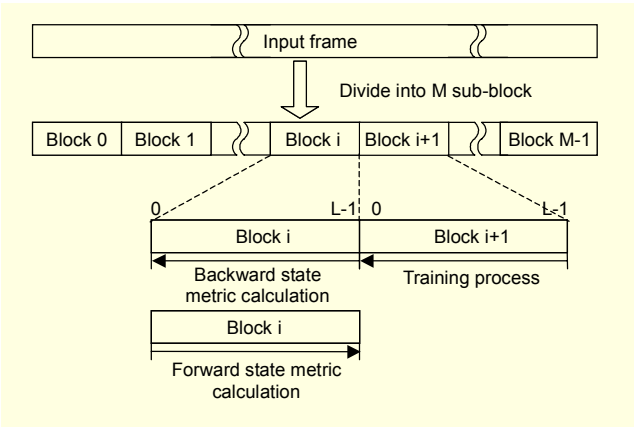

Fig. 3. The block-wise MAP decoding process.

In the block-wise MAP decoding algorithm, forward and backward state metrics need to be computed for each block. When the block MAP decoding is processed forward from block 0 to block M-1, the initial forward state metric of each block can be obtained from the last forward state metric of the previous block. The initial backward state metrics cannot be obtained from the previous block, unlike the initial forward state metrics, because the calculation sequence of backward state metrics is opposite to that of the block processing. This induces the reliability problem in the backward state metrics, which in turn degrades the BER performance. To avoid the BER performance degradation, a training process is introduced. The training process determines the reliable initial backward state metrics by calculating the backward state metrics in the next block without storing their values in the memory. For successive decoding, a parallel invocation of backward state metric calculation process and the training process is required using two processors [5], [6].

Figure 4 shows the timing diagram of the decoding process for a sequence of data blocks. The numbers within the arrows represent the block numbers. This figure shows the process of calculating the branch metrics and the forward and backward state metrics. For example, the branch metrics for data block #1 is calculated at time frame 1. The branch metrics for data block #2 and the forward state metrics for data block #0 are calculated at time frame 2. At the same time, the training process, marked as a shadowed arrow, is performed for data block #1 to get an initial metric for the backward state metrics of data block #0.

For continuous decoding, the memory capacity for storing the branch metric for four blocks is required. In this scheme, memory usage can be significantly reduced. However, it requires several state metric processors to compute several state metrics, simultaneously. To overcome this problem, a pipelined architecture for calculating state metrics is presented in this paper [6].

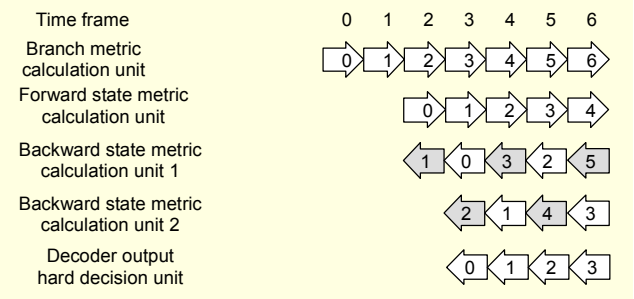

Fig. 4. Timing diagram of the decoding process of block-wise MAP decoder.

3. Block-wise MAP Decoder Parameters

The block size and the training size of the block-wise MAP

decoder determine the amount of the required memory and BER performance. Figure 5(a) shows the BER performance versus the training size under variable SNRs. As the training size increases, the BER becomes smaller, getting closer to log-MAP values. As the training size decreases, the BER becomes larger, resulting in poor performance. This is due to the fact that reliable initial backward state metrics cannot be obtained from shorter training sizes. Figure 5(b) shows the BER performance versus training size under several channel conditions where the SNR = 0.6, 0.8, 1.4, 1.8 and 2.2 dB. Note that the BER performance does not change with training sizes larger than 16 (at SNR = 2.2 dB) and larger than 10 (at SNR = 1.0 dB). This means that, when the training size is sufficiently large, the BER performance is not affected. The right-most log index of Fig. 5(b) shows the limit of the BER of the block-wise MAP decoding algorithm. From these observations, training size of five times the constraint size is considered to be sufficient.

The training process determines the initial backward state metrics. When the block size is large enough, the backward state metrics become reliable as the decoding process continues regardless of the initial error. When the training size is not large enough, the initial backward state metrics are not reliable and the BER performance is degraded.

Figure 6(a) shows the BER performance with various values of the block size and training size. When the training size is set to 4, the BER performance heavily depends on the block size. When the training size is 16, the BER performance shows little differences with different values of block size. In other words, with sufficient training size, the BER performance is not affected by the block size. Figure 6(b) shows the BER differences when the block sizes are set to 4 and 16 with different values of the training size. When the training size is below 16, the difference of the BER performance is remarkable. In other words, when the training size is sufficient, the BER performance is not affected by the block size.

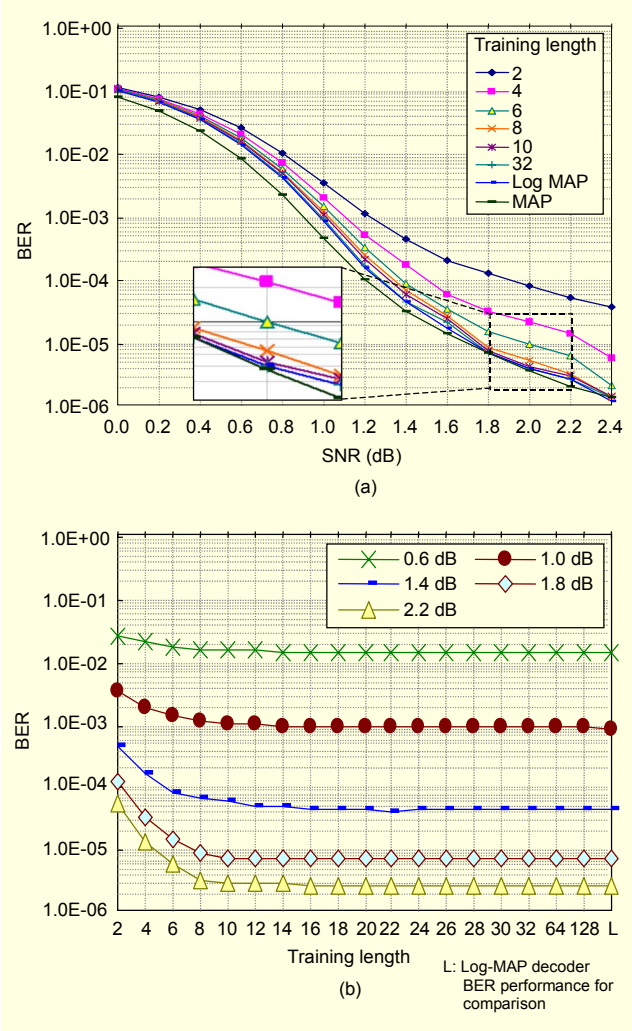

Fig. 5. Decoder performance of the block-wise MAP decoding algorithm. (constraint length = 3): (a) BER versus training size with SNRs and (b) BER versus training with the channel.

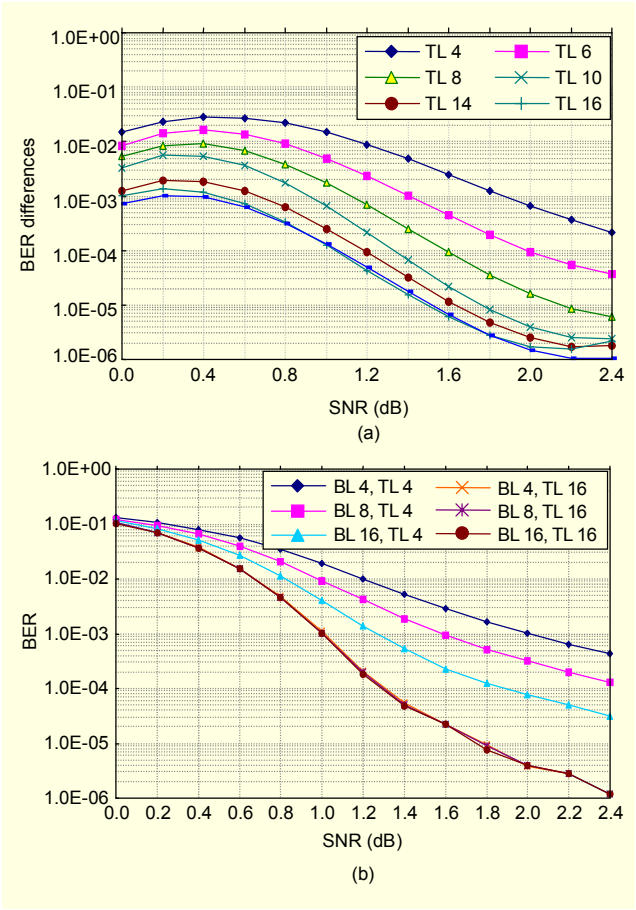

Fig. 6. Decoder performance of the block-wise MAP decoding algorithm (constraint length = 3): (a) BER versus variable block size and training size and (b) BER versus block size with training sizes set to 4 and 16.

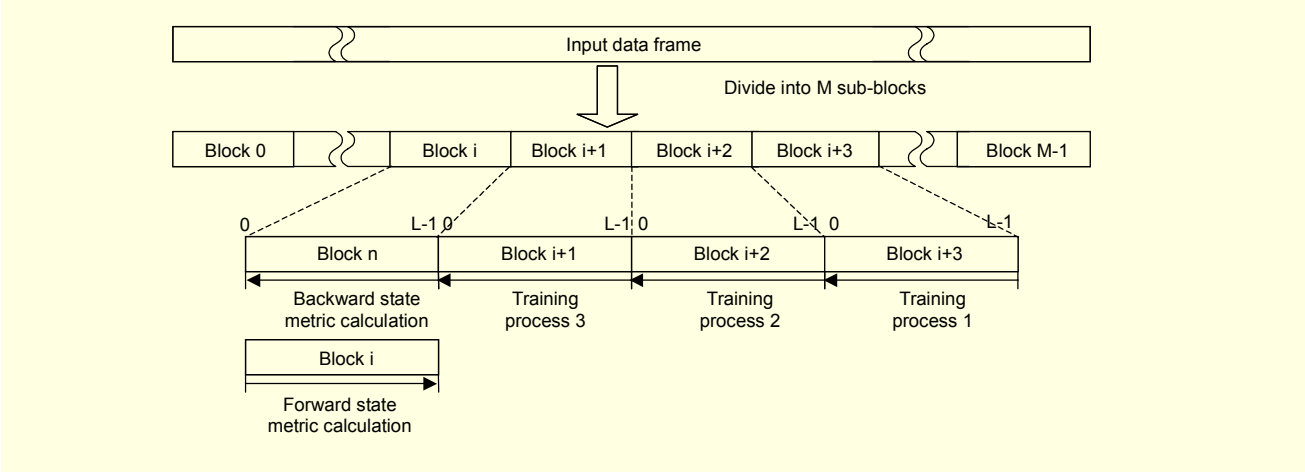

Fig. 7. Proposed block-wise MAP decoding scheme where the block size is set to 1/μ of the training size.

III. Proposed Block-wise MAP Decoder

1. Proposed Block-wise MAP Decoding Algorithm

Previous block-wise MAP decoders have the training size equal to the block size. However, when the training size is large enough, high BER performance can be achieved even with a small block size. A small block size implies less memory requirement. This paper proposes a new scheme where the block size is only a fraction of the training size. Figure 7 shows the proposed block-wise MAP decoding scheme where the block size is set to 1/μ of the training size. The training blocks are divided into μ sub-blocks, and the proposed decoding scheme performs the training process sequentially using μ backward state metric processors. In this case, μ +2 state metric processors are required for calculating the forward and backward state metrics.

Figure 8 shows the sequence of computation of all the metrics and the log-likelihood ratio in the proposed scheme with μ =3. The state metric processor of the block-wise MAP decoder consists of one forward state metric processor and four backward state metric processors. When the backward state metrics is computed, three backward state metric processors perform the training process to calculate the initial backward state metrics to be used at the next time frame. That is, the initial backward state metrics is obtained through three continuous training processes. The backward state metric calculating process and three training processes use the initial values obtained from the training process for the next blocks. There is no need to store the backward state metric in the training process, which makes it possible to have a continuous decoding process.

The proposed scheme requires memory for storing the forward state metric and the branch metrics. However, the memory requirement in the proposed scheme is reduced when compared to the previous scheme due to the employment of the reduced block size.

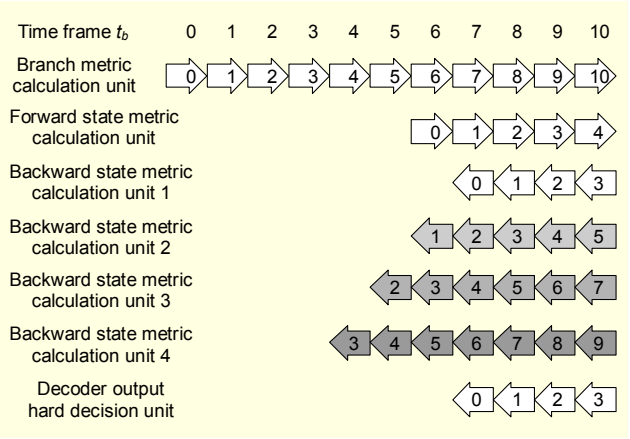

Fig. 8. Timing diagram of the proposed decoding scheme where the block size is set to 1/3 of the training size.

2. Proposed Pipelined Architecture

The block-wise MAP decoding algorithm can induce a larger latency for obtaining state metrics. To reduce the latency, another backward state metric processor is required for the training. Moreover, a conventional pipelined structure cannot be applied to the state metric processor due to the recursion structure of the state metric processor [6]. The proposed block-wise MAP decoding algorithm requires μ+2 state metric processors for the calculation of state metrics in parallel. We propose a (μ+2)-stage pipelined metric processor architecture for the block-wise MAP decoder for maximal hardware utilization [10]. We first examined the delay of a state metric processor and selected points where the delay for each stage can be evenly distributed. The

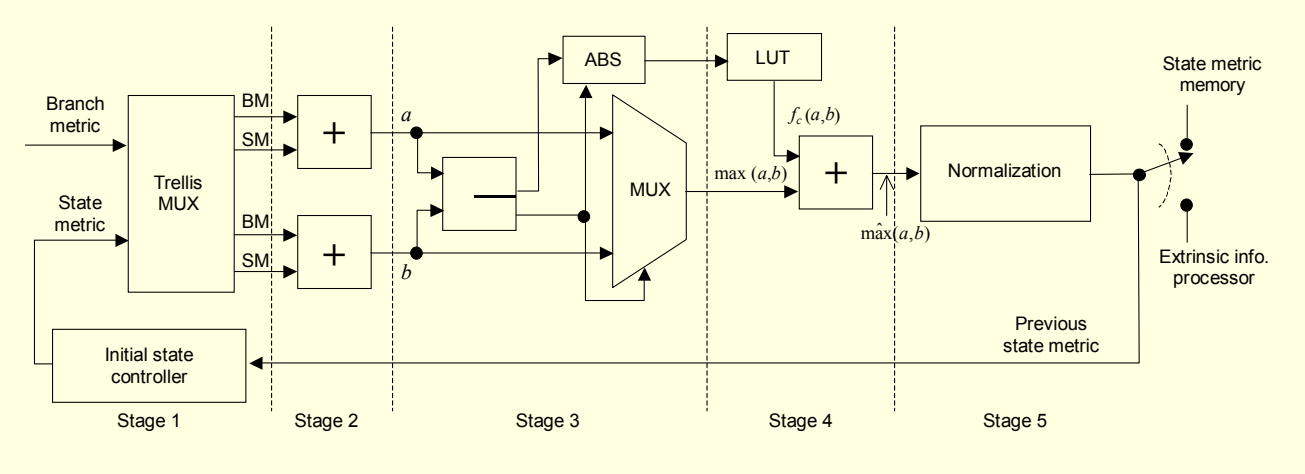

Fig. 9. The proposed metric processor structure with μ=3.

proposed architecture assigns a different metric calculation to each stage, as shown in Fig. 8. The proposed state metric processor performs a time-shared operation to ensure the proper metric calculation.

Figure 9 shows the proposed pipelined state metric processor architecture, which consists of three adders, one subtracter, one comparator, one selection MUX, one trellis MUX, and glue logics. Since the add-compare select operations are more complex than trellis MUX or the normalization block, it is divided into three pipeline stages in the proposed architecture, where μ = 3. Hence, the proposed pipelined architecture has five stages operating in a pipelined fashion.

IV. Experimental Results

The proposed decoder has been implemented in C language running on UNIX. The proposed block-wise MAP decoding algorithm has been simulated with the training size set to 24. The performance has been measured for various block size values. The block size is set to 1/ μ of the training size. Table 1 shows the memory requirement to store the branch and state metrics. The memory requirement for the branch and state metrics with μ = 2, 3, and 4 is reduced by 30%, 40%, and 45%, respectively, when compared to the previous block-wise log-MAP decoder [4].

Table 2 represents a synthesized memory area for various MAP decoders. Hynix 0.35 μm technology library has been used for the synthesis. Due to the large memory requirement for interleaving and de-interleaving, the overall memory area is reduced by 5.27%, 6.62%, and 7.29% with μ = 2, 3, and 4, respectively.

Table 1. Memory requirement for state and branch metrics.

μ	Memory requirement			Percent reduction
	State metric	Branch metric	Total	
Previous work [4]	1,536	6,144	7680.0	-
2	768	4,608	5376.0	30 %
3	512	4,096	4608.0	40 %
4	384	3,840	4224.0	45 %

Table 2. Synthesized memory area in gate count.

μ	Memory area*	Area reduction	
		Gate count	%
Previous work [4]	135,627.4	-	-
2	128,474.8	7,152.6	5.27%
3	126,650.6	8,976.8	6.62%
4	125,739.2	9,888.2	7.29%

* A 2-input NAND gate is counted as 1 gate.

V. Conclusion

Based on the simulation results, which show that a satisfactory BER performance can be achieved when the training size is sufficient, this paper proposes a new decoding structure employing the block size that is set to a fraction of the training size. In the proposed MAP decoder structure, the memory area has been reduced by 5.27%, 6.62%, 7.29% with μ = 2, 3, and 4, respectively, when compared to previous architecture where the training size is the same as the block size. Still, it has a comparable BER performance.

References

[1] C. Shannon, "A Mathematical Theory of Information," *Bell System Technical J.*, vol. 27, July 1948, pp. 379-423.

[2] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes(1)," *Proc. ICC'93*, Geneva, Switzerland, May 1993, pp. 1064-1070.

[3] J.P. Woodard and L. Hanzo, "Comparative Study of Turbo Decoding Techniques: An Overview," *IEEE Trans. on Vehicular Technology*, vol. 49, no. 6, Nov. 2000, pp. 2208-2238.

[4] S. Barbulescu and S. Pietrobon, "Turbo Codes: A Tutorial on a New Class of Powerful Error Correcting Coding Schemes, Part 2: Decoder Design and Performance," *IEEE J. of Electrical and Electronics Eng.*, vol. 19, no. 3, Sept. 1999, pp. 143-152.

[5] A. Viterbi, "An Intuitive Justification and a Simplified Implementation of the MAP Decoder for Convolutional Codes," *IEEE J. on Selected Areas in Comm.*, vol. 16, no. 2, Feb. 1998, pp. 260-264.

[6] G. Park, S. Yoon, I. Jin, and C. Kang, "A Block-wise MAP Decoder Using a Probability Ratio for Branch Metric," *Proc. VTC'99*, Amsterdam, Netherlands, Sept. 1999, pp. 1610-1614.

[7] S. Donilar and D. Divsalar, "Weight Distributions for Turbo Codes Using Random and Non-Random Interleaving," *JPL TDA Progress Report*, 42-122, 1995, pp. 56-65.

[8] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *Proc. IEEE Int. Symp. Inform. Theory*, Asilomar, CA, May 1972, pp. 90.

[9] P. Robertson, E. Villebrun, and P. Hoeher, "A Comparison of Optimal and Sub-optimal MAP Decoding Algorithms Operating in the Log Domain," *Proc. ICC'95*, Seattle, Washington, June 1995, pp. 1009-1013.

[10] Z. Wang, H. Suzuki, and K. Parhi, "VLSI Implementation Issues of Turbo Decoder Design for Wireless Applications," *Proc. IEEE Workshop on Signal Processing Systems*, Taipei, Taiwan, Oct. 1999, pp. 503-512.

Sik Kim received the BS, MS and PhD degree in electronic engineering from Sogang University, Seoul, Korea, in 1994, 1996 and 2003, respectively. In 2003, he joined System LSI Division of Samsung Electronics Corporation as a senior engineer. His current research interest includes HW/SW codesign, C-based design methodology and system level low power design for SoC design.

Sun-Young Hwang received the BS degree in electronic engineering from Seoul National University in 1976, the MS degree from KAIS (Korea Advanced Institute of Science), Seoul, Korea, in 1978, and the PhD degree in electrical engineering from Stanford University, California, CA, in 1986. Since 1986, he has been with the Center for Integrated Systems at Stanford University as a research associate, working on design of a high-level synthesis and simulation system. In 1978, he joined Samsung Semiconductor Inc., Korea, where he designed several CMOS VLSI chips and managed MOS VLSI design section. In 1986 and 1987, he held a consulting position at Palo Alto Research Center of Fairchild Semiconductor Corporation. Since 1989, he has been with the Department of Electronic Engineering at Sogang University, Seoul, Korea, and is currently a Professor. His current research interests include SoC design and SoC design methodology, DSP/embedded systems design.

Moon Jun Kang received the BS and MS degree in electronic engineering from Sogang University, Seoul, Korea, in 2000 and 2002 respectively. In 2002, he joined Pinetron Corporation as a Senior Engineer. His current research interest includes embedded system design, image processing VLSI system design, and system level low power design for SoC design.