A Memory-Reduced Log-MAP Kernel for Turbo Decoder

Tsung-Han Tsai
Department of Electrical Engineering
National Central University
Jhongli 320, Taiwan, R.O.C.
han@dsp.ee.ncu.edu.tw

Cheng-Hung Lin and An-Yeu Wu
Graduate Institute of Electronic Engineering
National Taiwan University
Taipei 102, Taiwan, R.O.C.
dixson@access.ee.ntu.edu.tw

***Abstract*—Generally, the Log-MAP kernel of the turbo decoding consume large memories in hardware implementation. In this paper, we propose a new Log-MAP kernel to reduce memory usage. The comparison result shows our proposed architecture can reduce the memory size to 26% of the classical architecture. We also simplify the memory data access in this kernel design without extra address generaters. For 3GPP standard, a prototyping chip of the turbo decoder is implemented to verify the proposed memory-reduced Log-MAP kernel in 3.04×3.04mm^2 core area in UMC 0.18um CMOS process.**

I. INTRODUCTION

The turbo codes have been proved that they can get a high coding gain near the Shannon capacity limit [1]. The reduction in *bit error rates* (BER) is achieved at the expense of intensive computations involved in the iterative turbo decoding steps. The iterative turbo decoding is composed of the soft-input soft-output (SISO) decoding algorithms. A powerful SISO algorithm for the turbo decoding is the maximum *a posteriori* (MAP) algorithm. However, the MAP algorithm has been converted into logarithm maximum *a posteriori* (Log-MAP) algorithm with additive form due to the large requirements of multiplication and exponential computations [2].

The memory organization of the SISO algorithms, especially the Log-MAP, is critical because they are highly data dominated. Some previous works have been developed based on the techniques for memory reduction [3], [4], [7]. In this paper, we focus on the Log-MAP kernel of turbo decoder, as shown in Fig. 1, and propose a memory-reduced Log-MAP architecture with low complexity in data access.

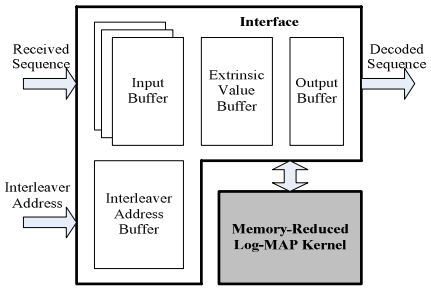

Figure 1. Block diagram of the turbo decoder.

This paper is organized as follows. The basic Log-MAP algorithm is described in Section II. In Section III, we describe the details of the previous and proposed organizations of the Log-MAP algorithm. The analysis and evaluation results are demonstrated in Section IV. Finally, the experiment results and the conclusions are given in Section V and Section VI, respectively.

II. FUNDAMENTALS OF LOG-MAP ALGORITHM

In turbo decoding, the Log-MAP decides binary encoded bit u_k on the sign bit of the *a posteriori log-likelihood ratio* (*LLR*). The arithmetic operations of the Log-MAP are described as follows. The branch metrics are computed as

$$\gamma_k(S_{k-1},S_k)=\frac{1}{2}\left(\Lambda_{in,k}(u_k)x_k^s+L_c y_k^s x_k^s+L_c\sum_{i=1}^{m}y_k^{pi}x_k^{pi}\right) \quad (1)$$

, where $\Lambda_{in,k}$ is *a priori* information, x_k denotes the transmitted codewords, y_k denotes received codewords, m is the number of each parity bit, and $L_c = 4E_S / N_0$. Note that x^s_k is the systematic bit which equals to u_k.

$$\alpha_k(S_k)=\underset{S_{k-1}}{MAX^*}(\gamma_k(S_{k-1},S_k)+\alpha_{k-1}(S_{k-1})), \quad (2)$$

$$\beta_k(S_k)=\underset{S_{k+1}}{MAX^*}(\gamma_{k+1}(S_k,S_{k+1})+\beta_{k+1}(S_{k+1})), \quad (3)$$

, where α_k is the forward recursion state metrics, β_k is the backward recursion state metrics.

The *LLR* is defined as

$$\Lambda_k(u_k)=\underset{S_k,u_k=+1}{MAX^*}(\alpha_k(S_k)+\gamma_{k+1}(S_k,S_{k+1})+\beta_{k+1}(S_{k+1}))-\underset{S_k,u_k=-1}{MAX^*}(\alpha_k(S_k)+\gamma_{k+1}(S_k,S_{k+1})+\beta_{k+1}(S_{k+1})). \quad (4)$$

The sign bit of the *LLR* value decides whether $u_k = +1$ or $u_k = -1$. The *MAX** operation is defined as

$$MAX^*(x,y)=\ln(e^x+e^y)=MAX(x,y)+\ln(1+e^{-(x-y)}). \quad (5)$$

The *MAX** can be implemented by an add-compare-select-offset (ACSO) unit, as shown in Fig 2. Small look-up tables can implement the corrective term $ln(1+e^{-(x-y)})$.

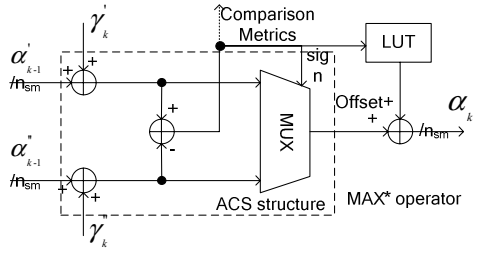

Figure 2. Architecture of the ACSO.

Because of the turbo decoding, there is one value to be iteratively interleaved and fed back to the Log-MAP as *a priori* information. This value called *extrinsic* information is defined as

$$\Lambda_{e,k}(u_k) = \Lambda_k(u_k) - \Lambda_{in,k}(u_k) - L_c y_k^s. \quad (6)$$

III. MEMROY-REDUCED APPROACH

A. Memory Reduction in Algorithms

Conventionally, all branch metrics γ_k^i, $i = 0 \sim 2^{m+1}-1$, are calculated and then stored in the branch metrics memory (BM) until the $\Lambda_k(u_k)$ is calculated. If each branch metric is n_{bm}–bit wide, the bit-length of the BM is $2^{m+1}*n_{bm}$ bits. However, we can further reduce the branch metrics stored in the BM. Table I shows that the $\gamma_k^j = -\gamma_k^{M-j}$, where $j = 0 \sim 2^m-1$ and $M = 2^{m+1}-1$. To reduce the memory, the BM only stores the γ_k^{M-j}, and the other branch metrics can be generated by multiplying the stored γ_k^{M-j} by *-1*. Fig. 3 shows an example of our BM-reduced approach when $m = 1$. The bit-length of the BM can be reduced to 2^m*n_{bm} bits.

TABLE I. THE BRANCH METRICS GENERATION.

Case (1) $m = 1$		Case (2) $m = 2$	
(x^s_k, x^{p1}_k)	γ_k^i	$(x^s_k, x^{p1}_k, x^{p2}_k)$	γ_k^i
(+1, +1)	γ_k^3 *	(+1, +1, +1)	γ_k^7 *
(+1, -1)	γ_k^2 *	(+1, +1, -1)	γ_k^6
(-1, +1)	$\gamma_k^1 = -\gamma_k^2$	(+1, -1, +1)	γ_k^5
(-1, -1)	$\gamma_k^0 = -\gamma_k^3$	(+1, -1, -1)	γ_k^4 *
		(-1, +1, +1)	$\gamma_k^3 = -\gamma_k^4$
		(-1, +1, -1)	$\gamma_k^2 = -\gamma_k^5$
		(-1, -1, +1)	$\gamma_k^1 = -\gamma_k^6$
		(-1, -1, -1)	$\gamma_k^0 = -\gamma_k^7$

Besides, the *a priori* term, $-(\Lambda_{in,k}(u_k) + L_c y_k^s)$ in (6), is also buffered until the $\Lambda_{e,k}(u_k)$ is calculated. It can be further eliminated in our Log-MAP kernel design. Take $m = 1$ for example, (6) can be transferred as follows:

$$\Lambda_{e,k} = \Lambda_k(u_k) - \Lambda_{in,k}(u_k) - L_c y_k^s = \Lambda_k(u_k) - (\gamma_k^2 + \gamma_k^3). \quad (7)$$

The *a priori* term can be easily replaced by the sum of two branch metrics which is marked * in Table I. In our kernel design, we only store two branch metrics used to calculate the $\Lambda_k(u_k)$ in registers instead of buffering the *a priori* term. Fig. 4 shows an example when $m = 1$.

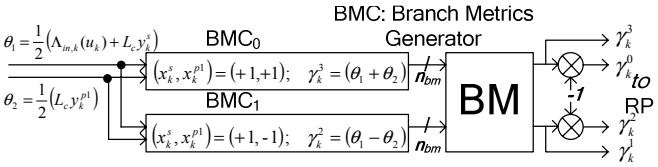

Figure 3. Block diagram of the branch metrics generation.

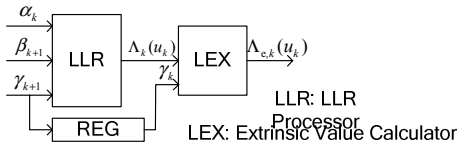

Figure 4. Block diagram of the extrinsic values generation.

In general, the values of the forward (2) and backward (3) recursions are computed in chronologically reverse order. Both forward (A_k) and backward recursion state vector (B_k) are required for computation of the $\Lambda_k(u_k)$, so it is necessary that a large size of state metric memory (SMM) stores the A_k (or B_k) values to compute the $\Lambda_k(u_k)$ until the B_k (or A_k) is generated. Each state vector is composed of 2^υ state metrics (the size of the trellis state) , and each one is n_{sm}-bit wide. The size of SMM for each state vector is $2^\upsilon*n_{sm}$.

B. Classical Architecture

Since the SMM reduction is an important issue in facilitating the hardware implementation, the classical Log-MAP architecture has been proposed to reduce the memory usage [3]. The classical Log-MAP architecture is composed of three recursion processes (RPs) operated in parallel. Two of them are used for the backward recursions (RP_B and ARP_B, Acquisition RP_B), and one is used for the forward recursions (RP_A). Each RP contains 2^υ ACSO units to do MAX^* operations working in parallel so that one recursion can be computed in one cycle.

The scheduling of the classical architecture is shown in Fig. 5. L refers to the acquisition depth by using the sliding window concept (i.e. $L = (4 \sim 6)*(\upsilon + 1)$, where υ denotes the number of delay elements in a convolutional encoder). The ARP_B provides the reliable B_k to the RP_B and does not need any SMM to store these state vectors of length L. However, the A_k is stored in a $L*2^\upsilon*n_{sm}$ size of the SMM in order to compute the $\Lambda_k(u_k)$ values. Note that the $\Lambda_k(u_k)$ is generated in reverse order. The scheduling also illustrates that the decoding latency of this architecture is $4L$ and that the computation cost is 3, which equals the total number of RPs. The branch metrics are buffered in $4L$ symbol time and the *a priori* term is buffered in $3L$ symbol time.

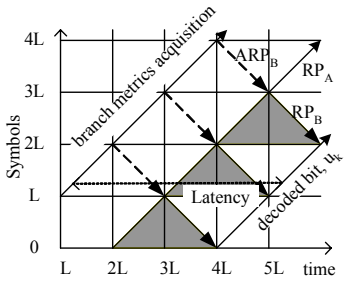

Figure 5. Scheduling of the classical Architecture [3].

C. Traceback Architecture

In this paper, we propose a traceback architecture to reduce more SMM size. In our approach, The B_ks are traced back with the forward recursions. The B_ks of length L are generated in the backward recursion and then traced back in the trace-back recursion when A_ks are generated. The trellis evolution of backward recursion and trace-back recursion is shown in Fig. 6 and the scheduling of this architecture is illustrated in Fig. 7.

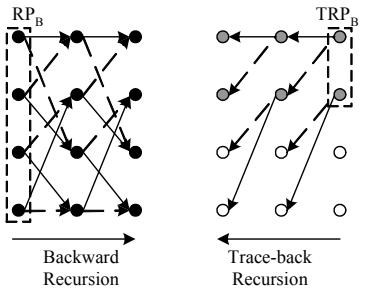

Figure 6. Trellis evolution of the backward and trace-back recursion ($\upsilon = 2$).

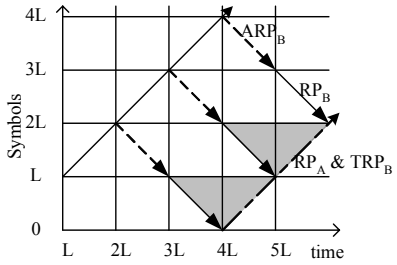

Figure 7. Scheduling of the proposed Architecture.

In the trace-back recursion, the TRP_B (trace-back RP_B) is required to regenerate 2^{υ} backward state metrics with only n_{sm}-bit comparison metrics (*comp*, as shown in Fig. 1.) In Fig. 6, the $2^{\upsilon-1}$ *comp*s can regenerate 2^{υ} backward state metrics so that the TRP_B contains $2^{\upsilon-1}$ TBUs (trace-back unit, as shown in Fig. 8). Thus the total size of the SMM which stores the *comp*s is $L*2^{\upsilon-1}*n_{sm}$. The TRP_B is half the complexity of the RP_B because the TRP_B only contains $2^{\upsilon-1}$ TBUs with similar complexity as the ACSO units. The $\Lambda_k(u_k)$ is generated in natural order.

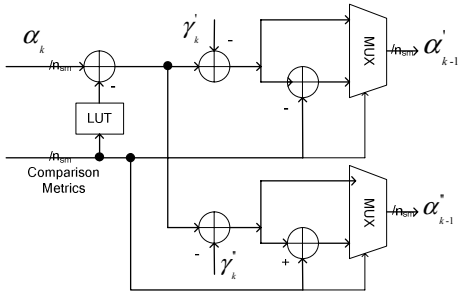

Figure 8. Architecture of the TBU.

IV. Analysis Results

The recomputed architecture (Section V.E in [5]) has been proposed as an efficiently memory-reduced architecture. In order to reduce the SMM size, it uses the simplified ACSO to recompute the B_ks every P cycles with the stored one-bit decision and n_{off}-bit offset values of length L. To describe the overall performance, the different configurations are evaluated in Table II and III. Note that the computation cost in Table II means the total number of RPs and *0.5* RP_B means the either Recompute-RP_B or TRP_B are used. In Table III, the Artisan SRAM Generator in UMC 0.18um Process is used to generate the dual-port SRAMs of the different configurations. The parameters, such as $\upsilon = 3$, $L = 24$, $P = 4$, $n_{off} = 3$ bits and $n_{sm} = 13$ bits, are under the constructions followed 3GPP standard [5]. Note that the state vectors and decision, offset values of the recomputed configurations are stored and loaded with different widths, so two different sizes of SRAM are required. Although the recomputed configurations can reduce the SMM size efficiently, the total physical SRAM areas of the recomputed configuration are larger. However, the SMM size and physical SRAM area of the proposed configuration are less because only one size of SRAM is needed to store the *comps*.

TABLE II. The Performance of The Different Organizations.

Organization		Classical [3]	Recomputed [4]	Proposed
Computation Cost	RP_A	1	1	1
	RP_B	2	2.5	2.5
Additional Address Generator		None	Needed	None
Controllability		Easy	Complex	Easy
Latency		*4L*	*4L*	*4L*

The total memory size of the classical configurations is compared with our approach in Table IV. The AP denotes the *a priori* term buffer whose bit-length is n_{ap} bits. We take $\upsilon = 3$, $m = 1$, $L = 40$, $n_{bm} = 10$ bits, $n_{sm} = 13$ bits and $n_{ap} = 9$ bits to comply with the 3GPP standard. Besides, a more advanced approach is combined our method with the low latency concept [6]. Since ARP_B can be substituted by a look-ahead RP_B ($LARP_B$) in the low latency method, *5L/2* decoding latency can be achieved for improvement. In

addition, the advanced architecture can further reduce the total memory size. The overall architecture of the advanced Log-MAP decoder is shown in Fig. 9.

TABLE III. THE SMM AREA COMPARISON OF THE DIFFERENT ORGANIZATIONS. (ν = 3, L = 24, P = 4, n_{off} = 3 bits and n_{sm}= 13 bits. Generated by Artisan HS-SRAM-DP Dual Port SRAM Generator in UMC 0.18um Process)

Organization	SMM Size	Words x Bits	Total Memory Cells	Single Memory Area (mm^2)	Total Memory Area (mm^2)
Classical [3]	$L*2^{\nu}*n_{sm}$	192x13	2496	0.15	0.15
Recomputed [4]	$(\lceil L/P \rceil+P-1)*2^{\nu}*n_{sm}$	72x13	1704	0.08	0.16
	$L*2^{\nu}*(1+n_{off})$	192x4		0.08	
Proposed	$L*2^{\nu-1}*n_{sm}$	96x13	1248	0.13	0.13

TABLE IV. THE TOTAL MEMORY SIZE OF DIFFERENT ORGANIZATIONS.

Organization	BM	SMM	AP	Total (Kb)	Memory Size Ratio
Classical [3]	$4L*2^{m+1}*n_{bm}$	$L*2^{\nu}*n_{sm}$	$3L*n_{ap}$	11.64	100%
Proposed	$4L*2^{m}*n_{bm}$	$L*2^{\nu-1}*n_{sm}$	0	5.28	45%
Advanced	$5L/2*2^{m}*n_{bm}$	$L/2*2^{\nu-1}*n_{sm}$	0	3.04	26%

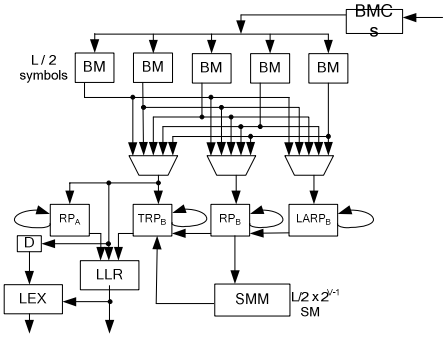

Figure 9. Architecture of the memory-reduced Log-MAP decode.

V. EXPERIMENTAL RESULTS

A fast and quite accurate VLSI implementation approach is obtained by simply using cell-based synthesis tools to compile Verilog HDL cores. In Fig. 10, we implement the 3GPP turbo decoder to verify the advanced memory-reduced Log-MAP kernel. We summarize the characteristics of our design in Table V. The SRAM size of the Log-MAP kernel is 3.04Kb (listed in Table IV) and the total physical area of the SRAMs inside the Log-MAP kernel is almost half of the total physical area of the kernel. The overhead of the TRP_B used to regenerate B_ks is 0.06mm^2, which is less than 4% of the total area of the Log-MAP kernel.

VI. CONCLUSIONS

In this paper, a memory-reduced Log-MAP kernel is proposed. In this kernel, the BM, SMM, and *a priori* term are modified to reduce the memory size. The comparison result shows it can efficiently reduce the memory usage to 26% of the classical architecture. A prototyping chip is implemented to verify the proposed Log-MAP kernel. The proposed approach ensures that the memory size of the memory-reduced Log-MAP kernel is much less than that of the classical architecture, while the kernel is implemented in a system-on-chip for wireless communication applications.

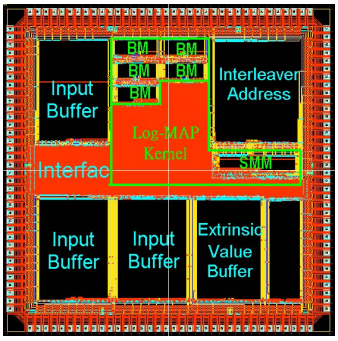

Figure 10. Chip layout of the 3GPP turbo decoder.

TABLE V. CHIP SUMMARY OF THE 3GPP TURBO DECODER.

Technology	UMC 0.18um 1p6m CMOS process
Core Size	3.04x3.04mm^2
Maximum Operating Frequency	145MHz
Maximum Decoding Rate	12Mb/s (6 iterations)
Supply Voltage	1.8V
Total Dual-Port SRAM Size of The Turbo Decoder	0.28Mb
Dual-Port SRAM Size inside The Log-MAP Kernel	3.04Kb
Dual-Port SRAM area inside The Log-MAP Kernel	0.95mm^2
Total Area of The Log-MAP Kernel	1.85mm^2

REFERENCES

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo Codes," in *Proc. ICC'93*, pp. 1064-1070, May 1993.

[2] J. P. Woodard, and L. Hanzo, "Comparative study of turbo decoding techniques: an overview," *IEEE Trans. Vehicular Technology*, vol. 49, pp. 2208 –2233, Nov. 2000.

[3] A. J. Viterbi, "An intuitive justification and simplified implementation of the MAP decoder for convolutional codes," *IEEE J. Selected Area in Commun.*, vol. 16, pp. 260-264, Feb. 1998.

[4] E. Boutillon, W. J. Gross, and P. G. Gulak, "VLSI architectures for the MAP algorithm," *IEEE Trans. on Commun.*, vol. 51, pp. 175-185, Feb. 2003.

[5] "Technical Specification Group Radio Access Network; Multiplexing and Channel Coding (FDD)," 3rd Generation Partnership Project, 3GPP Ts25.212 v5.1.0, 2002.

[6] A. Raghupathy, and K. J. R. Liu, "A transformation for computational latency reduction in turbo-MAP decoding," in *Proc. ISCAS '99,* vol. 4, pp. 402 –405, Jul 1999.

[7] J. Dielissen, and J. Huisken, "State vector reduction for initialization of sliding windows MAP," in *Proc. 2nd Int. Symp. Turbo Codes*, pp. 387-390, Sept. 2000.