As turbo decoding is a highly memory-intensive algorithm consuming large power, a major issue to be solved in practical implementation is to reduce power consumption. This paper presents an efficient reverse calculation method to lower the power consumption by reducing the number of memory accesses required in turbo decoding. The reverse calculation method is proposed for the Max-log-MAP algorithm, and it is combined with a scaling technique to achieve a new decoding algorithm, called hybrid log-MAP, that results in a similar BER performance to the log-MAP algorithm. The periodic storing method is a variation of the sliding window scheme. A practical reverse calculation method was proposed for the log-MAP algorithm along with its implementation in [10]. However, the implementation is relatively complex and requires a large logic overhead.

This paper proposes an efficient reverse calculation method to reduce not only memory power consumption but also memory size. The proposed reverse calculation method is based on the Max-log-MAP algorithm, and it is combined with a scaling technique to achieve a new decoding algorithm called hybrid log-MAP. Compared to the loser storing method that always accesses the memory to decide the position and the value of the loser stored during forward recursion, the proposed reverse calculation method stores and accesses the loser selectively only when the loser cannot be recovered by the reverse calculation. Therefore, the proposed hybrid log-MAP algorithm can further reduce power consumption by reducing the number of memory accesses as well as the memory size.

The rest of this paper is organized as follows. In Sect. 2, conventional turbo decoding algorithms are described. A new reverse calculation method is proposed for Max-log-MAP algorithm in Sect. 3. A hybrid log-MAP algorithm that combines Max-log-MAP and log-MAP algorithms is proposed in Sect. 4. Experimental results are presented in Sect. 5.
The soft outputs of both SISO decoders are not improved further in performance. Then the last stage makes hard decisions by examining the log-likelihood ratio (LLR).

Turbo coding requires SISO decoders to generate extrinsic information and LLR. Either the maximum a posteriori (MAP) algorithm [11] or the soft output Viterbi algorithm (SOVA) [12] can be used for SISO decoding. As MAP-based turbo decoders provide much better performance than SOVA-based turbo decoders, we focus on the MAP-based algorithm in this work. In practical implementations, Max-log-MAP and log-MAP algorithms are commonly employed due to the high complexity of the MAP algorithm [13], [14].

2.2 Max-log-MAP Algorithm

The Max-log-MAP decoder decides whether an information bit \( u_k = 0 \) or 1 depending on the log-likelihood ratio \( L_R(u_k) \) which is defined below.

\[
L_R(u_k) = \max_U \left[ \alpha_{k-1}(s_{k-1}) + \gamma_k(s_{k-1} \rightarrow s_k) + \beta_k(s_k) \right] - \max_U \left[ \alpha_{k-1}(s_{k-1}) + \gamma_k(s_{k-1} \rightarrow s_k) + \beta_k(s_k) \right]
\]

(1)

where \( s_k \) is an encoder state at time \( k \), \( U^1 \) is the set of all possible state transitions \( (s_{k-1} \rightarrow s_k) \) corresponding to the case of \( u_k = 1 \), and \( U^0 \) is similarly defined. If \( L_R(u_k) \) is greater than or equal to zero, \( u_k \) is decided to 1. Otherwise, \( u_k \) is decided to 0. In Eq. (1), \( \alpha_{k-1}(s_{k-1}) \), \( \beta_k(s_k) \), and \( \gamma_k(s_{k-1} \rightarrow s_k) \) are the forward, backward, and branch metrics, respectively.

The metrics are calculated as

\[
\alpha_k(s_k) = \max_{s_{k-1}} \left[ \alpha_{k-1}(s_{k-1}) + \gamma_k(s_{k-1} \rightarrow s_k) \right]
\]

(2)

\[
\beta_{k-1}(s_{k-1}) = \max_{s_k} \left[ \beta_k(s_k) + \gamma_k(s_{k-1} \rightarrow s_k) \right]
\]

(3)

\[
\gamma_k(s_{k-1} \rightarrow s_k) = \frac{1}{2} x_k^1 \left( L_C y_k^1 + L_C y_k^2 \right) + \frac{L_C}{2} x_k^1 y_k^1
\]

(4)

Eq. (4) is derived with assuming that the code is transmitted through an AWGN channel with a noise variance of \( \sigma^2 \), where \( x_k \) and \( y_k \) represent the transmitted codeword associated with this transition, and the codeword received from the channel, respectively, \( L_C(u_k) \) is the extrinsic information received from the other SISO decoder, and \( L_C = 2/\sigma^2 \).

The extrinsic information to be passed to the companion decoder, \( L_{e, \text{OUT}}(u_k) \) is calculated as

\[
L_{e, \text{OUT}}(u_k) = L_R(u_k) - L_C y_k^1 - L_{e, \text{IN}}(u_k),
\]

(5)

where \( L_{e, \text{IN}}(u_k) \) denotes the extrinsic information received from the companion decoder. In the case of the second SISO decoder in Fig. 2, \( L_{e, \text{OUT}}(u_k) = L_{2e} \) and \( L_{e, \text{IN}}(u_k) = L_{1e} \).

2.3 log-MAP Algorithm

As some approximations are used in the max functions of Eqs. (1), (2), and (3), there is performance degradation in the Max-log-MAP algorithm compared to the MAP algorithm. It can be improved by using the \( \max^* \) function [15], [16].
which is defined as
\[
\max (x, y) = \max (x, y) + \ln (1 + e^{-|x-y|})
\] (6)

Given the \(\max^*\) function, we can rewrite Eqs. (1), (2), and (3) as
\[
\alpha_k(s_k) = \max^*_S [\alpha_{k-1}(s_{k-1}) + \gamma_k(s_{k-1} \rightarrow s_k)]
\] (7)
\[
\beta_{k-1}(s_{k-1}) = \max^*_S [\beta_k(s_k) + \gamma_k(s_{k-1} \rightarrow s_k)]
\] (8)
\[
L_R(u_k) = \max^*_U [\alpha_{k-1}(s_{k-1}) + \gamma_k(s_{k-1} \rightarrow s_k) + \beta_k(s_k)]
\] (9)

The performance of the log-MAP algorithm is identical to that of the MAP algorithm. As computing \(\ln(\cdot)\) in Eq.(6) is complex, a look-up table is usually used to simplify the computation.

3. Proposed Reverse Calculation for Max-log-MAP Decoding

As shown in the previous section, the calculation direction of the forward metrics is different from that of the backward metrics. The forward metrics are initialized at the starting point and calculated in the forward direction. On the other hand, the backward metrics initialized at the ending point progress in the backward direction. Since the calculation directions of the two metrics are different, it is impossible to calculate both metrics simultaneously. All the forward (backward) metrics have to be calculated first and saved in a memory. Then, the saved forward (backward) metrics are loaded from the memory in order to compute \(L_R(u_k)\) with the calculated backward (forward) metrics. Due to the memory accesses, a turbo decoder suffers from larger power consumption. If we can calculate the forward (or backward) metrics in the reverse direction, there is no need to access the memory any more, saving power consumption. Assuming that forward metrics are calculated prior to backward metrics, we will present in this paper an efficient method to reversely calculate the forward metrics.

In a binary system, two branch transitions appear as a butterfly pair if the first and the last shift register are connected both in the feedback and feed-forward polynomials, and good RSC encoders always satisfy this condition [8]. Since the W-CDMA turbo encoder has such a butterfly structure as shown in Fig. 1, the trellis diagram can be grouped into butterfly pairs. Figure 3(a) shows a trellis section, where the dotted line means the transition by a systematic bit of 0, and the solid line represents the transition induced by a systematic bit of 1. A re-arranged version of the trellis section is drawn in Fig. 3(b) to clearly show four butterfly pairs.

Let us consider one butterfly pair shown in Fig. 4, \((\alpha^0_k, \alpha^4_k, \alpha^0_{k+1}, \alpha^4_{k+1})\). Even though there are four branches in a butterfly pair, there exist only two distinguishable branch metric values for a butterfly pair [8]. The branch metric values, \(\gamma_p\) and \(\gamma_c\), are always different except the case of

\[
\gamma_p = \gamma_c = 0.
\]

In the Max-log-MAP decoding, \(\alpha^0_{k+1}\) and \(\alpha^4_{k+1}\) are calculated from \(\alpha^0_k\) and \(\alpha^4_k\) as
\[
\begin{align*}
\alpha^1_{k+1} &= \max (\alpha^4_k + \gamma_p, \alpha^0_k + \gamma_c) \\
\alpha^0_{k+1} &= \max (\alpha^4_k + \gamma_c, \alpha^0_k + \gamma_p)
\end{align*}
\] (10)

After calculating \(\alpha^4_{k+1}\) and \(\alpha^1_{k+1}\), \(\alpha^0_k\) and \(\alpha^4_k\) are stored into the memory. To calculate \(L_R(u_k)\), they are loaded later from the memory. If \(\alpha^0_k\) and \(\alpha^4_k\) can be calculated from \(\alpha^0_{k+1}\) and \(\alpha^4_{k+1}\) in the same direction as backward metrics, the memory access can be substituted by calculations. The rest of this section describes how to calculate the forward metrics in the backward direction.

From Fig. 4 and Eq. (10), the decision of \(\alpha^0_{k+1}\) and \(\alpha^4_{k+1}\) can be categorized into 4 cases as shown in Fig. 5.

1) Case 1
The values of \(\alpha^0_{k+1}\) and \(\alpha^4_{k+1}\) are determined by \(\alpha^4_k\) as follows.
\[
\begin{align*}
\alpha^1_{k+1} &= \alpha^4_k + \gamma_p \\
\alpha^0_{k+1} &= \alpha^4_k + \gamma_c
\end{align*}
\] (11)
Equation (11) can be rewritten as
\[ a_k^4 = a_k^1 - \gamma_p \]
\[ a_k^4 = a_k^0 - \gamma_c \]  
(12)

From Eq. (12), we can see that if \( a_{k+1}^1 \) and \( a_{k+1}^4 \) come from \( a_k^4, a_k^1 - \gamma_p \) and \( a_k^0 - \gamma_c \) are equal to the value of \( a_k^4 \), as shown below.
\[ a_{k+1}^4 - \gamma_p = a_k^0 - \gamma_c = a_k^4 \]  
(13)

2) Case 2
The values of \( a_{k+1}^0 \) and \( a_{k+1}^1 \) are determined by \( a_k^0 \). This case is very similar to case 1, and we can see that if \( a_{k+1}^0 \) and \( a_{k+1}^1 \) come from \( a_k^0, a_k^1 - \gamma_c \) and \( a_k^0 - \gamma_p \) to the value of \( a_k^0 \).
\[ a_{k+1}^1 - \gamma_c = a_{k+1}^0 - \gamma_p = a_k^0 \]  
(14)

3) Case 3
The values of \( a_{k+1}^0 \) and \( a_{k+1}^1 \) come from \( a_k^0 \) and \( a_k^0 \) as follows.
\[ a_{k+1}^0 = a_k^0 + \gamma_c \]
\[ a_{k+1}^1 = a_k^0 + \gamma_p \]  
(15)

Equation (15) can be rewritten as
\[ a_k^0 = a_{k+1}^0 + \gamma_p \]
\[ a_k^0 = a_{k+1}^0 + \gamma_c \]  
(16)

From Eqs. (10) and (15), the followings are derived.
\[ a_{k+1}^0 > a_k^0 + \gamma_c \]
\[ a_{k+1}^0 > a_k^0 + \gamma_c \]  
(17)

From Eqs. (16) and (17), the following expressions come out.
\[ a_{k+1}^1 - \gamma_c > a_{k+1}^0 - \gamma_p = a_k^0 \]
\[ a_{k+1}^0 - \gamma_c > a_{k+1}^1 - \gamma_p = a_k^0 \]  
(18)

From Eq. (18), we can see that between two paths connected to each \( a_k \), the smaller one is the value of \( a_k \).

4) Case 4
The values of \( a_{k+1}^0 \) and \( a_{k+1}^1 \) come from \( a_k^4 \) and \( a_k^0 \). This case is analogous to case 3, and we can derive the following expressions.
\[ a_{k+1}^1 - \gamma_p > a_{k+1}^0 - \gamma_c = a_k^4 \]
\[ a_{k+1}^0 - \gamma_p > a_{k+1}^1 - \gamma_c = a_k^0 \]  
(19)

From Eq. (19), the smaller \( a_{k+1}^1 - \gamma \) is the value of \( a_k \) as in case 3.

3.1 Strategy for Reverse Calculation

Among the four cases, case 1 and case 2 can be satisfied simultaneously if and only if \( \gamma_p = \gamma_c = 0 \). To avoid such a situation, although the situation is extremely rare, a zero-valued branch metric is replaced by a value of minimal non-zero magnitude. The replacement makes the four cases unique and experimental results show that the performance influence induced by the replacement is ignorable. Looking at the four above cases, we can calculate the forward metrics in the reverse direction, as is summarized below.

- When we calculate \( a_{k+1} \) from \( a_k \), the only \( a_k \) value which never wins in the max competitions of Eq. (10) is saved into the memory. If two values for a max function are equal, the loser in the other max function is saved in the memory. Later when we need \( a_k \), we calculate \( a_k \) from \( a_{k+1} \) as described below.
  - First, we calculate two \( a_{k+1} - \gamma \) values on the paths connected to a state of \( a_k \), as shown in Fig. 6.
  - If these values are equal, then they are the value of \( a_k \) as indicated in Eqs. (13) and (14). And the value of \( a_k \) of the other state in the butterfly pair has to be loaded from the memory. The latter \( a_k \) must be in the memory because it was the loser in the max competition.
  - If these values are different in both the states of a butterfly pair, each state \( a_k \) takes the smaller value of two \( a_{k+1} - \gamma \) values, as shown in Eqs. (18) and (19). Since in this case the two \( a_k \)'s can be reversely calculated from the two \( a_{k+1} \)'s belonging to the butterfly pair, there is no need to store any \( a_k \)'s into the memory.

This strategy stores, in the worst case, only one \( a_k \) in the memory for a butterfly pair, instead of two \( a_k \)'s required in the conventional turbo decoding. Therefore the required memory size is reduced to a half of the original memory size, and the number of memory accesses is reduced drastically. Compared to the loser storing method [9] which always accesses the memory to read the position and the value of the loser, the proposed strategy requires memory accesses only when the loser cannot be recovered by the reverse calculation, reducing the number of memory accesses significantly than the loser storing method.

4. Hybrid log-MAP Algorithm

The hardware complexity of the Max-log-MAP algorithm is lower than that of the log-MAP algorithm, and the former algorithm is more tolerant to the channel estimation error. However, if the channel noise is properly estimated, the log-MAP algorithm outperforms the Max-log-MAP algorithm. The log-MAP algorithm gives exactly the same performance as the MAP algorithm, but the Max-log-MAP algorithm gives a slight degradation in performance com-
pared to the MAP algorithm. If we can have reliable channel estimation and do not take into account the higher complexity, the log-MAP algorithm will be the best choice. In the log-MAP algorithm, all the max functions used in the Max-log-MAP algorithm are replaced with max* functions defined in Eq. (6).

Due to the correction factor of the max* function, the reverse calculation of state metrics in the log-MAP algorithm is much more difficult than the Max-log-MAP algorithm. In [10], a reverse calculation method is applied to the log-MAP algorithm, but the logic complexity is increased by 40%.

In this section, we propose a new decoding algorithm called hybrid log-MAP, which gives almost the same performance as the conventional log-MAP algorithm. The reverse calculation method proposed for the Max-log-MAP decoding is also applied to the hybrid log-MAP algorithm.

The reverse calculation method for the Max-log-MAP decoding can be applied to the log-MAP decoding if all the max* functions used for the forward metric calculation in the log-MAP algorithm are replaced with the max functions. This replacement makes the forward metric calculation equal in both of the log-MAP and Max-log-MAP algorithms. The replaced max function leads to performance degradation in the log-MAP algorithm. Since the backward metric calculation and \( L_R(u_k) \) calculation are still based on the max* function even if the forward metric calculation is replaced with the max function, the performance of the hybrid log-MAP decoding will be placed between the conventional log-MAP and Max-log-MAP decodings.

In addition, more performance improvement is expected in the proposed hybrid log-MAP decoding if it is associated with the scaling technique [17], [18]. Recent works proved that scaling the extrinsic information improves the performance of Max-log-MAP algorithm by more than 0.2 dB. As the hybrid log-MAP decoding shares some characteristics with the Max-log-MAP decoding because of the replaced max function in the forward metric calculation, the extrinsic information scaling can be employed to improve the performance of the hybrid log-MAP decoding.

5. Experimental Results

Based on the proposed hybrid log-MAP decoding, a turbo decoder of which specification is presented in Table 1 was described in Verilog-HDL and synthesized using a 0.18 \( \mu \)m standard-cell library and compiled SRAM memories. In Table 1, \((q,f)\) denotes a quantization scheme that uses \(q\) bits in total and \(f\) bits to represent the fractional part. As the fractional part of the branch metric is represented by 2 bits, zero-valued branch metrics are replaced with 0.25, the value of minimal non-zero magnitude, in the proposed reverse calculation.

The proposed decoder employs the sliding window technique [19] to reduce the size of memory required. The timing diagram for the sliding window technique is shown in Fig. 7. The window size is set to 32, which is 8 times of the constraint length of W-CDMA turbo code. The window size of 32 gives almost the same performance as the ideal case associated with an infinite window.

Since at each updating step branch metrics are added to the current metrics and the largest value is selected as a new metric for each state, the state metric values increase as their calculations are progressed according to the calculation direction. To avoid arithmetic overflow, a normalization scheme is required, leading to additional hardware and latency. However, the normalization can be avoided by using the 2’s complement arithmetic and setting its bit width to \(k\) to take into account the upper bound of metric difference [19].

\[
k = \lceil \log_2 \Delta_{\text{MAX}} \rceil + 1
\]

In Eq. (20), \(\Delta_{\text{MAX}}\) is the upper bound of metric difference between two path metrics of a butterfly. The value of \(\Delta_{\text{MAX}}\) for W-CDMA turbo code is given to 256 in [20]. Hence, the bit width of \(k = 9\) is sufficient for the internal SISO operation.

Table 2 shows the memory specification of the proposed decoder, comparing with the conventional decoder and the loser storing method [9]. Since \(L_c(u_k)\) and \(L_c y_k^l\) are always used together as shown in Eqs. (4) and (5), we
save \((L_{c,k} + L_{c,y}^e) / 2\) and \(L_{c,y}^e / 2\) in the branch memory, instead of separate instances, \(L_{c,k}\) and \(L_{c,y}^e\). Hence, the word size of branch memory is determined to 10 bits: 6 bits for \((L_{c,k} + L_{c,y}^e) / 2\) and 4 bits for \(L_{c,y}^e / 2\). Since there are 4 processes as shown in Fig. 7, the whole branch memory is partitioned into 4 memory banks to make every process access data at the same time without causing conflicts, one for storing the inputs coming from the channel and three for dummy beta, beta, and alpha calculations. As usual, the alpha memory comprises 2 banks, one for storing the value during the alpha calculation and the other for reading the value for the \(L_D(u_k)\) calculation, as shown in Fig. 7. In this case, the word size of each bank is set to 72 bits to store 8 state metrics. To allow each butterfly pair individual access to a memory, we use a separate bank for each butterfly pair. Totally, the alpha memory comprises 8 banks of word size 9 bits in the proposed SISO decoder. Employing the proposed reverse calculation scheme reduces the state memory size by 50%. Therefore, the total memory size is reduced by 39.2% as shown in Table 2.

Figure 8 shows the overall architecture of the proposed SISO decoder. The branch metric input stream is fed into several units for parallel processing as shown in Fig. 7. As beta metrics are calculated in the opposite direction, the dummy beta unit is needed to decide the initial values of beta metrics, and its architecture is identical to the beta unit.

Figure 9 compares the numbers of alpha metrics loaded from the memory in the proposed hybrid log-MAP decoding and in the loser storing method [9], where the number is normalized by the number of alpha metrics loaded in the conventional log-MAP decoding. The loser storing method reduces the memory accesses by 50% because only one of two metrics in a butterfly is loaded, while the proposed reverse calculation scheme reduces the memory accesses by about 80% because there is no need to access the memory if both the metrics in a butterfly can be reversely calculated. As the memory access is tightly related to the power consumption, the more reduction of the memory accesses leads to the less power consumption.

The power consumption of the proposed SISO decoder is presented in Table 3, compared to the conventional decoder and the loser storing method. The power consumption of the state metric memory is reduced by 75%, and the total power consumption is reduced by 34.4%. With a higher SNR and a larger number of iterations, the proposed decoder will consume less power because more alpha metrics can be calculated reversely, while the conventional decoder consumes a fixed power. In the case of the loser storing method, more power consumption in the state memory is required due to more frequent memory accesses as shown in Fig. 9. As the SISO logic complexity of the hybrid log-MAP decoder is comparable to that of the loser storing method, the less total power consumption is expected due to the less power consumption in the state memory.

Table 4 shows that the proposed decoder is designed with 21820 gates, while the conventional decoder requires
18004 gates when they are synthesized under the constraint of the same operation frequency of 200 MHz. The logic complexity of the proposed decoder is increased by 21.2%. The critical path delay is 4.74 ns. As a result, the proposed decoder can operate at approximately 200 MHz, which is enough to meet the W-CDMA standard specification of 2 Mbps.

The BER performances of the Max-log-MAP, conventional log-MAP and hybrid log-MAP algorithms are presented in Fig. 10. The extrinsic information scaling improves the performance of the hybrid log-MAP decoding. Simulating on various scaling factors, we found that the hybrid log-MAP algorithm gives the best performance when the scaling factor is close to 0.9. Considering finite precision, we set the scaling factor to 0.875 as presented in Table 1. The hybrid log-MAP with the scaling factor shows almost the same performance as that of the conventional log-MAP algorithm. In the worst case, the performance of the proposed decoder associated with scaling is only 0.05 dB lower than the conventional log-MAP decoder. Without scaling, the performance of the hybrid log-MAP algorithm is 0.08 dB worse than the conventional log-MAP algorithm and 0.27 dB better than the Max-log-MAP algorithm for a BER of $10^{-5}$.

### 6. Conclusion

This paper has presented a new efficient reverse calculation method to reduce power consumption by minimizing memory accesses required in turbo decoding. As the turbo code has butterfly structures in its trellis diagram, the reverse calculation is possible and simple especially for the Max-log-MAP decoding. First, we presented an efficient reverse calculation method for the Max-log-MAP decoding. Then, to apply the same reverse calculation to the log-MAP algorithm, we modified the log-MAP algorithm by replacing the max* operations needed in the forward metric calculation with the max operation and by scaling the extrinsic information. The hybrid log-MAP algorithm shows almost the same performance as the log-MAP algorithm.

Experimental results show that about 80% forward metrics can be reversely calculated in the W-CDMA standard. By employing the reverse calculation method in the hybrid log-MAP turbo decoder, power consumption and memory size are reduced by approximately 35% and 40%, respectively, compared to the conventional log-MAP turbo decoder. Although the proposed method is simulated and implemented for the W-CDMA standard, it can also be applied to other applications.

### Acknowledgments

This work was supported by Institute of Information Technology Assessment through the ITRC and by the Korea Science and Engineering Foundation through MICROS center and by IC Design Education Center (IDEC).

### References


[13] H.-M. Choi received the B.S. and M.S. degrees in electronics engineering from Korea Advanced Institute of Science and Technology (KAIST) in 2003 and in 2005, respectively. Her research interests include VLSI design for communication and SoC.

Ji-Hoon Kim received the B.S. degree in electronics engineering from Korea Advanced Institute of Science and Technology (KAIST) in 2004. Currently, he is a M.S. student in the Department of Electrical Engineering and Computer Science at KAIST. His research interests include VLSI design for communication and low power circuit design.

In-Cheol Park received the B.S. degree in electronics engineering from Seoul National University, in 1986, and the M.S. and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST) in 1988, 1992, respectively. Since June 1996, he has been an Assistant Professor and now a Professor in the Department of Electrical Engineering and Computer Science at KAIST. Prior to joining KAIST, he was with IBM T.J. Watson Research Center, Yorktown, NY, from May 1995 to May 1996, where he researched high-speed circuit design. His current research interests include computer-aided design algorithm for high-level synthesis and very large scale integration architectures for general-purpose microprocessors. Dr. Park received the best paper award at the ICCD in 1999 and the best design award at the ASP-DAC in 1997.