

Low Power VLSI Implementation of the Map Decoder for Turbo Codes through Forward Recursive Calculation of Reverse State Metrics

Indrajit Atluri¹ and Tughrul Arslan^{1,2,3}

¹School of Engineering and Electronics, The University of Edinburgh, Edinburgh, UK.

²Institute for System Level Integration, Alba Centre, Livingston, UK.

³Jet Propulsion Laboratory, NASA, Pasadena, California, USA.

Abstract

Memory utilization is a vital issue in the implementation of the MAP algorithm. The MAP decoder operating in the log domain requires either the forward or the backward path metrics to be stored before finally calculating the log-likelihood decisions. This process consumes large amount of memory. In this paper, we present a reduced complexity version of a technique which computes the reverse state metrics in the forward direction and hence reduces the required memory size. A low power VLSI architecture of a Log-MAP decoder employing this technique is presented. We provide results which demonstrate that the proposed design reduces the memory size and hence reduces the power consumption by 35%.

Introduction

The major breakthrough in the field of Error control coding was the invention of Turbo Codes by Berrou *et al* in 1993 (1) which immediately generated considerable interest. MAP algorithm which received little interest in the early days, now gained prominence due to its reduced complexity and minimization of the bit-error rate (BER). Due to its superiority with respect to coding gain over the soft output Viterbi algorithm (SOVA) (2) which produces qualitatively inferior soft outputs, MAP decoder is used in the implementation of Turbo codes. Hence, the MAP algorithm has become an increasingly important building block for future communication systems.

The storage of forward or reverse state metrics for the whole frame before the likelihood decisions could be obtained, in a MAP decoder not using the sliding window technique requires large amount of memory. A MAP algorithm which involves the forward computation of the reverse state metrics (3) has been proposed in order to reduce the memory size considerably. This reduced size comes at the expense of increased computation required for this calculation. In this paper, the authors present a more reduced complexity version of the MAP algorithm using the forward computation of reverse state metrics and implement a MAP decoder using the same technique. The VLSI implementation of both the conventional Log Map decoder and the MAP decoder employing this reduced complex version of the MAP algorithm are discussed. Overall power reduction of up to 35% is obtained as compared to the conventional MAP decoder. A power

profile of the main contributing blocks in the core is also provided.

Algorithms and Implementation

The MAP algorithm provides not only the estimated sequence, but also the probabilities for each bit that has been decoded correctly. Assuming binary codes are to be used, the MAP algorithm gives, for each decoded bit u_k in step k , the probability that this bit was +1 or -1, given the received distorted symbol sequence $y_0^N = (y_0, y_1, y_2, \dots, y_N)$. This is equivalent to finding the likelihood ratio

$$\lambda_k = \frac{P\{u_k = +1 | y_0^N\}}{P\{u_k = -1 | y_0^N\}} \quad (1)$$

where $P\{u_k = i | y_0^N\}$, $i = +1, -1$ is the a posteriori probability (APoP) of u_k .

Computation of $P\{u_k | y_0^N\}$ is done by determining the probability to reach a certain encoder state m after having received k symbols $y_0^{k-1} = (y_0, y_1, y_2, \dots, y_{k-1})$:

$$\alpha_k(m) = P\{m | y_0^{k-1}\} \quad (2)$$

and the probability to get from encoder state m' to the final state in step N with symbols y_{k+1}^N :

$$\beta_{k+1}(m') = P\{y_{k+1}^N | m'\} \quad (3)$$

The probability of the transition $m \rightarrow m'$ using the source symbol u_k , under knowledge of the received symbol y_k , is called γ_k :

$$\gamma_k(m, m', u_k) = P\{m, m', u_k | y_k\} \quad (4)$$

The probabilities $\alpha_k(m)$ and $\beta_{k+1}(m')$ are computed recursively over $\gamma_k(m, m', u_k)$ which are a function of the received symbols and the channel model as below:

$$\alpha_k(m') = \sum_{all\ m} \gamma_k(m, m', u_k) \alpha_{k-1}(m) \quad (5)$$

$$\beta_{k-1}(m) = \sum_{all\ m'} \beta_k(m') \gamma_k(m, m', u_k) \quad (6)$$

Knowing these values for each transition $m \rightarrow m'$, the probability of having sent the symbol u_k in step k is the

sum of all paths using the symbol u_k in step k . With $\phi(u_k)$ being the set of all transitions with symbol u_k , we can write

$$P\{u_k | y_0^N\} = \sum_{(m,m') \in \phi(u_k)} \alpha_k(m) \gamma_k(m, m', u_k) \beta_{k+1}(m') \quad (7)$$

Thus, from the above equations we can conclude that to evaluate the likelihood value of a decoded bit we require many additions and multiplications. The decoding complexity of the MAP algorithm has been reduced by operating it in the log domain. This technique was first used for the ISI channel (4). Taking the negative logarithm of $\alpha_k(m)$, $\beta_{k+1}(m')$, $\gamma_k(m, m', u_k)$ and λ_k values from the MAP algorithm we have:

$$A_k(m) = -\ln \alpha_k(m) \quad (8)$$

$$B_{k+1}(m') = -\ln \beta_{k+1}(m') \quad (9)$$

$$D_k(m, m', u_k) = -\ln \gamma_k(m, m', u_k) \quad (10)$$

$$L_R(u_k) = -\ln \lambda_k \quad (11)$$

Using the above equations, we rewrite (5) & (6) as follows:

$$A_k(m') = -\ln \sum_{all\ m} \exp[A_{k-1}(m) + D_k(m, m', u_k)] \quad (12)$$

$$B_{k-1}(m) = -\ln \sum_{all\ m'} \exp[D_k(m, m', u_k) + B_k(m')] \quad (13)$$

$D_k(m, m', u_k)$ is calculated as follows (5):

from (10) we have $D_k(m, m', u_k) = -\ln \gamma_k(m, m', u_k)$

$$= -\ln \left[\frac{1}{\sqrt{2\pi\sigma^2}} \exp \left[-\frac{1}{2\sigma^2} (y_m - \hat{y}_m)^2 \right] \right] P(u_k) \quad (14)$$

where y_m , \hat{y}_m are the received signal and estimated signal over a Gaussian channel respectively, $P(u_k)$ is the a-priori probability of bit u_k and σ^2 is the noise variance. And finally the log likelihood ratio value is calculated as below:

$$L_R(u_k) = -\ln \frac{\sum_{(m \rightarrow m')=u_k=+1} \exp[A_{k-1}(m) + D_k(m, m', u_k) + B_k(m')]}{\sum_{(m \rightarrow m')=u_k=-1} \exp[A_{k-1}(m) + D_k(m, m', u_k) + B_k(m')]} \quad (15)$$

All the summations in (12), (13) and (15) can be represented in the form of a 'E' function which is defined as follows: $a \text{ E } b = -\ln(e^{-a} + e^{-b})$. This can be rewritten as $\min(a, b) - \ln(1 + e^{-|a-b|})$.

The log-MAP decoder is divided into four major blocks. These are the branch metric calculator (BMC), the forward state metric calculator (FSMC), reverse state metric calculator (RSMC) and the log-likelihood ratio calculator (LLRC). A detailed description of the log-MAP decoder could be found in (6). To increase the speed of the decoder a parallel implementation is adopted in this paper, that is all the SM's are calculated simultaneously. The block diagram of the architecture is shown in Fig. (1).

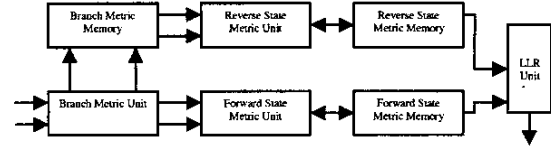


Fig. 1: Block diagram of a MAP decoder.

The forward state metric unit uses the branch metrics obtained directly from the BMU to calculate the forward state metrics, and the backward state metric unit uses the reversed branch metrics from the branch metric storage to calculate the reverse state metrics.

An important part of the state metric calculator is the implementation of adders and the 'E' operand. The FSMC and RSMC are identical except for the direction of recursion. The branch metrics are added to the state metrics (as in Viterbi algorithm) and then the minimum is found between the two obtained competing path metrics. The difference between the two path metrics is stored in a look-up table to obtain the value of the function: $f(|a-b|) = \ln(1 + e^{-|a-b|})$. Thus the add-compare-select and the E-operations are carried out in ACSE (add-compare-select and E-operation) block. For the forward state metrics the state metric value for state 00 is initialized to zero and the remaining states to the maximum possible value (the closest value to infinity). For the reverse state metrics, if the final state is unknown all the initial state metrics are set to zero, otherwise they are initialized in the same way as the forward state metrics.

A rate $\frac{1}{2}$ encoder with code generator $g(D) = [1, 1+D^2]$ in an AWGN channel is taken as an example. The trellis structure for this code generator at time k is shown in Fig. (2). From the conventional MAP algorithm the Reverse State metrics are updated by the following equations:

$$\beta_k(00) = \gamma_k(00) \cdot \beta_{k+1}(00) + \gamma_k(11) \cdot \beta_{k+1}(01) \quad (16)$$

$$\beta_k(01) = \gamma_k(00) \cdot \beta_{k+1}(10) + \gamma_k(11) \cdot \beta_{k+1}(11) \quad (17)$$

$$\beta_k(10) = \gamma_k(01) \cdot \beta_{k+1}(00) + \gamma_k(10) \cdot \beta_{k+1}(01) \quad (18)$$

$$\beta_k(11) = \gamma_k(01) \cdot \beta_{k+1}(10) + \gamma_k(10) \cdot \beta_{k+1}(11) \quad (19)$$

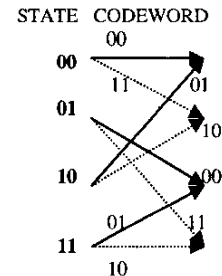


Fig. 2: The trellis structure.

The above equations could be expressed in matrix form as below:

$$\begin{bmatrix} \beta_k(00) \\ \beta_k(10) \end{bmatrix} = \begin{bmatrix} \gamma_k(00) & \gamma_k(11) \\ \gamma_k(01) & \gamma_k(10) \end{bmatrix} \begin{bmatrix} \beta_{k+1}(00) \\ \beta_{k+1}(01) \end{bmatrix} \quad (20)$$

$$\begin{bmatrix} \beta_k(01) \\ \beta_k(11) \end{bmatrix} = \begin{bmatrix} \gamma_k(00) & \gamma_k(11) \\ \gamma_k(01) & \gamma_k(10) \end{bmatrix} \begin{bmatrix} \beta_{k+1}(10) \\ \beta_{k+1}(11) \end{bmatrix} \quad (21)$$

Assuming $\begin{bmatrix} \gamma_k(00) & \gamma_k(11) \\ \gamma_k(01) & \gamma_k(10) \end{bmatrix}$ to be γ , we consider the inverse

of γ as γ^{-1} if it exists, and obtain β_{k+1} from β_k as follows

$$\begin{bmatrix} \beta_{k+1}(00) \\ \beta_{k+1}(01) \end{bmatrix} = \gamma^{-1} \begin{bmatrix} \beta_k(00) \\ \beta_k(10) \end{bmatrix} \quad \& \quad \begin{bmatrix} \beta_{k+1}(10) \\ \beta_{k+1}(11) \end{bmatrix} = \gamma^{-1} \begin{bmatrix} \beta_k(01) \\ \beta_k(11) \end{bmatrix}$$

Now, we have

$$\gamma^{-1} = \frac{1}{(\gamma_k(00)\gamma_k(10) - \gamma_k(01)\gamma_k(11))} \begin{bmatrix} \gamma_k(00) & -\gamma_k(11) \\ -\gamma_k(01) & \gamma_k(10) \end{bmatrix} \quad (22)$$

$$\text{let } \frac{1}{(\gamma_k(00)\gamma_k(10) - \gamma_k(01)\gamma_k(11))} = C$$

Finally getting back to the equation form, we finally have

$$\beta_{k+1}(00) = C \times \{\gamma_k(00)\beta_k(00) - \gamma_k(11)\beta_k(10)\} \quad (23)$$

$$\beta_{k+1}(01) = C \times \{\gamma_k(10)\beta_k(10) - \gamma_k(01)\beta_k(00)\} \quad (24)$$

$$\beta_{k+1}(10) = C \times \{\gamma_k(00)\beta_k(01) - \gamma_k(11)\beta_k(11)\} \quad (25)$$

$$\beta_{k+1}(11) = C \times \{\gamma_k(10)\beta_k(11) - \gamma_k(01)\beta_k(01)\} \quad (26)$$

From the above equations we have the reverse state metric β_{k+1} calculated from the most recent state metric β_k .

New Implementation of the Log-MAP decoder

The calculation of reverse state metrics in a forward direction as discussed in the previous section increases the computational complexity. In this section the authors introduce a reduced complexity version of the same technique explained in the previous section. This reduction in complexity is obtained by operating (23), (24), (25) and (26) in the logarithmic domain. Now applying the negative log to (23) we have

$$\begin{aligned} -\ln|\beta_{k+1}(00)| &= B_{k+1}(00) \\ &= -\ln\{C \times (\gamma_k(00)\beta_k(00) - \gamma_k(11)\beta_k(10))\} \\ &= -\ln|C| - \ln\{(\gamma_k(00)\beta_k(00) - \gamma_k(11)\beta_k(10))\} \\ &= -\ln\{e^{-(D_k(00)+B_k(00))} - e^{-(D_k(11)+B_k(10))}\} - \ln|C| \quad (27) \end{aligned}$$

In the above equation the function $-\ln(e^{-a} - e^{-b})$ (the absolute value ensures that the function is defined) is evaluated in a similar way to the function $-\ln(e^{-a} + e^{-b})$, hence can be rewritten as $= \min(a, b) - \ln(1 - e^{-|a-b|})$.

Now we implement the values of $\ln(1 - e^{-|a-b|})$ in a small look-up table, similar to that of a conventional Log-Map decoder. One problem that arises in implementing this look-up table is the value of the function $\ln(1 - e^{-|a-b|})$ which is invalid when $a=b$. The value of this function is almost zero all the time except when $a=b$. Hence, the value could be neglected. This approximation is taken only for

the calculation of reverse state metrics. The approximation process which is the result of matrix inversion operation, may lead to degradation in the performance of the decoder. In order to ensure the computational stability of this matrix inversion operation, the complete frame is divided into blocks of N_b (≥ 2) bits, storing only the $B_k(m)$ at the beginning of each block during backward recursion, and using (16), (17), (18) and (19) to compute the $B_k(m)$ within each block during forward recursion. Thus, a smaller value of N_b ($=2$) has been suggested for medium to high SNR (Signal to Noise Ratio) values as the MAP decoder converges rapidly. But, for Turbo Codes (employing the MAP decoders) where a low SNR is expected, a larger size of N_b is acceptable.

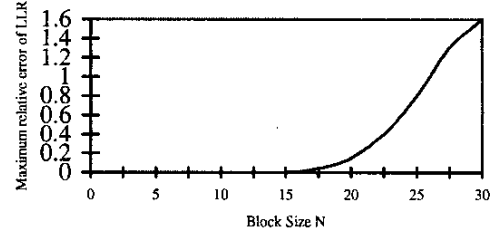


Fig. 3: Variation of maximum relative error of LLR with block size N_b . Results extracted from (3).

The variation of maximum relative error of LLR with block size N_b , is shown in Fig. (3). The relative error is defined as follows:

$$\Delta_r^{N_b} = \frac{|\lambda_k^A(u; O) - \lambda_k^{A, N_b}(u; O)|}{E(|\lambda_k(u; O)|)}$$

where $\lambda_k^A(u; O)$ is the k -th soft output obtained with the conventional method, $\lambda_k^{A, N_b}(u; O)$ is the output obtained with the new method and block size N_b and $E(x)$ is the mean value of x . It has been suggested in that the relative error is negligible when $N_b \leq 5K$, where K ($K=3$ in our implementation) is the constraint length. As a compromise between the decoder performance at low SNR and the reduction in memory size we have chosen N_b to be eight for our implementation.

Now the value of $\ln C$ is calculated in a similar way using the same function as described above. Note that the calculation of $\ln C$ is a new addition into the decoder architecture. Thus, $\ln C$ is given as

$$\begin{aligned} &\ln \left[\frac{1}{\gamma_k(00)\gamma_k(10) - \gamma_k(01)\gamma_k(11)} \right] \\ &= -\ln[\gamma_k(00)\gamma_k(10) - \gamma_k(01)\gamma_k(11)] \\ &= -\ln(e^{-(D_k(00)+D_k(10))} - e^{-(D_k(01)+D_k(11))}) \end{aligned}$$

The above expression is similar to the function $-\ln(e^{-a} - e^{-b})$ discussed previously. This value of $\ln C$ obtained, is

subtracted from the value $-\ln\left(e^{-(D_i(00)+B_i(00))} - e^{-(D_i(01)+B_i(10))}\right)$ in (27) to evaluate the corresponding reverse state metric.

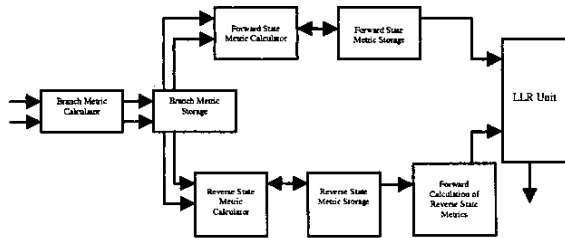


Fig. 4: Functional Block diagram of the new Log-MAP decoder.

The forward calculation of the reverse state metrics is carried out in a separate FRSMC (forward calculation of the reverse state metrics) block. Thus, after conventionally calculating the reverse state metrics in the RSMC and storing only the $B_j(m)$ (reverse state metrics), where $j=1$ or $i.N_b$, $0 < i < N_{blk}$ and $N_{blk} = N/N_b$ we carry out the forward calculation using these values. A block diagram of the new decoder architecture is shown in Fig.(4) below.

Results

The designs of both the Conventional Log-MAP decoder and our new low power Log-MAP decoder have been implemented as a fully synthesizable Verilog models. We have used the 0.18 μ m standard cell library, and the Synopsis tool for synthesis and characterization. Power analysis is performed by feeding the switching activity information obtained from the netlists into Synopsys DesignPower tool. These results are shown in Fig. (5) and (6). In our implementation there is 88% reduction in memory size. This reduction leads to approximately 35% saving in power when compared to the traditional Log-MAP decoder. Power reduction of 17% is also achieved in the RSMC block. From Fig.(6) we can deduce that the branch metric storage consumes extra power when compared to the conventional decoder as the branch metrics are required for the forward calculation of reverse state metrics apart from their role in obtaining the reverse state metrics. For this reason, we have used a dual port RAM instead of a single port RAM as in the normal decoder architecture. The drastic reduction in reverse state metric (RSM) storage is obtained at the expense of introducing an additional block FRSMC which calculates the RSMs in the forward direction.

Conclusions

We have proposed a VLSI architecture for low power implementation of a Log-MAP decoder through the minimization of memory size for the storage of reverse state metrics. This technique proceeds by computing the reverse state metrics in a forward direction in a manner similar to the processing of the forward state metrics. The

technique results in 88% reduction in memory utilization at the expense of 13% increase in branch metric storage. This provides a net power reduction of 35% when compared to the conventional implementation of the Log-MAP decoder.

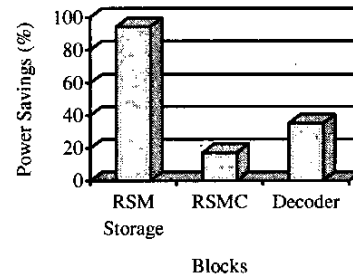


Fig. 5: Power Savings in various blocks.

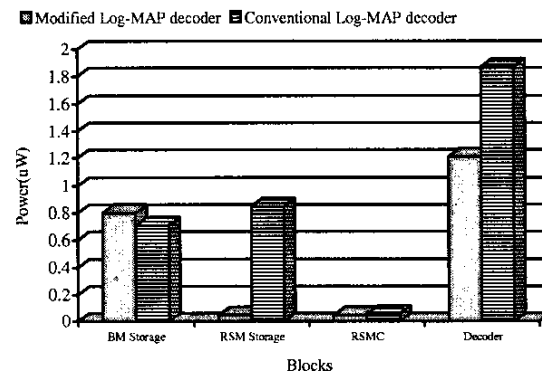


Fig. 6: Block Power Comparisons.

References

- (1) C.Berrou, A.Glavieux, and P.Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-Codes," In *Proc. ICC '93*, Pages 1064-1070, Geneva, Switzerland, May 1993.
- (2) J.Hagenauer and P.Hoehner, "A Viterbi algorithm with soft-decision outputs and its applications," in *Proc. Globecom, Dallas, TX*, Nov. 1989, pp. 1680-1686.
- (3) Yufei Wu, William J.Ebel, and Brian D.Woerner, "Forward computation of backward path metrics for MAP decoder," *IEEE VTC2000*.
- (4) J.Erfanian, S.Pasupathy, and G.Gulak, "Reduced complexity symbol detectors with parallel structures for ISI channels," *IEEE Trans. Commun.*, vol.42, pp.1661-1671, Feb./Mar./Apr. 1994, Part III.
- (5) L.Hanzo, T.H.Liew, B.L.Yeap, "Turbo Coding, Turbo Equalization and space-Time Coding for transmission over fading channels," John Wiley and Sons 2002, pp.543-544.
- (6) S.S.Pietrobon, "Implementation and performance of a Turbo/MAP decoder," *International Journal of Satellite Communications*, Vol.16, no.1, Jan.-Feb.1998. pp.23-46.