A Low-Power Architecture for Maximum a Posteriori Decoding *

Marisa López-Vallejo
ETSI Telecomunicación (UPM)
Ciudad Universitaria s/n
28040 Madrid, Spain
marisa@die.upm.es

Syed Aon Mujtaba
Wireless Systems Research
Agere Systems
New Jersey, USA
mujtaba@agere.com

Inkyu Lee
Dept. of Communications Eng.
Korea University
Seoul, Korea
inkyu@korea.ac.kr

Abstract

In this paper we present a novel architecture for soft-input soft-output (SISO) Maximum a Posteriori (MAP) decoding. The architecture leverages an ASIP (Application Specific Instruction-Set Processor) structure, where the datapath has been designed to achieve high-speed performance and low power dissipation. Salient features of this architecture include: (a) delayed renormalization of the α metrics with register by-passing to reduce latency, (b) use of register files to minimize power dissipation, and (c) microprogrammed control to achieve flexibility. The resulting architecture for the SISO-MAP decoder achieves a maximum throughput of 10.9 Msymbols/second, operating at 142MHz and dissipating 21mW in the datapath.

1 Introduction

Turbo codes have gained a lot of interest in the channel coding area because they approach the Shannon limit. It has been shown that turbo codes achieve outstanding bit error rate performance with a relatively simple iterative decoding algorithm. Due to its excellent performance, the third generation of wireless mobile system defined in the UMTS standard has adopted turbo codes as the channel coding scheme.

The basic element of a turbo decoder is the Soft Input Soft Output (SISO) decoding module, which provides $soft\ output$ or reliability values for the transmitted information sequence. The soft values are typically expressed as log likelihood ratios (LLR). The Maximum a Posteriori (MAP) algorithm [1] - also known as the BCJR algorithm - provides a symbol-by-symbol estimation of the soft values. Compared to the Viterbi algorithm, the BCJR algorithm requires 4 times more computation and consequently dissipates more power. Several techniques have been presented that address these issues [2-6]. Most implementations of the MAP algorithm are based on the (max)log-MAP algorithm employing the sliding window technique. For instance, [10] tries to reduce power dissipation by using parallel sliding windows, but results in an inflexible solution. The problem of reducing memory accesses is considered in [7]. However, the final architecture is not described in detail. [5] presents an optimized Add-Compare-Select Module to achieve lower power dissipation. In [2] voltage scaling is proposed to reduce power consumption, even though it is well known that this technique degrades performance. Nevertheless, this implementation of the original BCJR algorithm requires large memories, with the corresponding increase in area and power dissipation.

In this work, we present a flexible architecture for MAP decoding that can accomodate different encoding schemes, and achieve low power operation with high performance. The structure of the paper is as follows. Next, the basics of the MAP decoding algorithm are introduced. Section 3 describes the algorithmic to architecture mapping. Section 4 presents details of the main contributions in the datapath. Finally, synthesis results are presented and some conclusions are drawn.

2 Turbo-decoding using the MAP algorithm

The high computational complexity of the MAP algorithm [1] makes its implementation expensive and power hungry, which is why most implementations perform a simplified version of the algorithm. The two most common simplifications are: the Log-MAP and max-log-MAP algorithms [8]. Both approaches are based on performing the algorithm in the logarithmic domain, where multiplications become additions, and additions become maximizations using the Jacobian algorithm: $[ln(e^a + e^b) \approx max(a, b)]$. The max-log MAP algorithm uses the Jacobian approximation as is. However, this approximation becomes less accurate as a approaches b. For such instances, a correction term based on the difference $|a - b|$ is added to the $max(a, b)$ term. The correction term is typically stored in a one-dimensional lookup table [6]. The resulting algorithm approaches the log MAP algorithm, which strictly

*This work is partly funded by MCYT project TIC2000-0583-C02

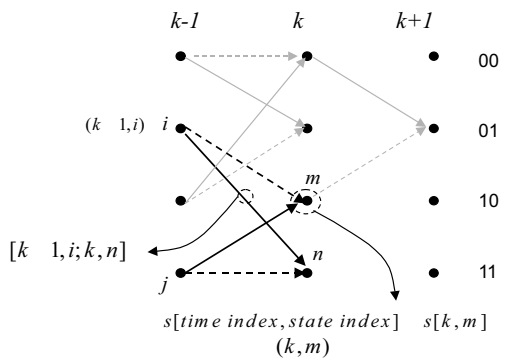

Figure 1: Notations used in this work.

speaking does not approximate the addition operations.

Figure 1 depicts the main variables involved in the MAP decoding algorithm. For a given transition in the trellis, $s[k,i]$ is the starting state at time stage k, $s[k+1,i]$ is the corresponding ending state at time stage $k+1$, $\alpha(k,m)$ is the forward-recursion path metric, $\beta(k-1,m)$ is the backward-recursion path metric and $\gamma[k-1,i;k,m]$ is the branch (or state transition) metric. N is the code sequence length, u_k is the information bit at time k, and y_k is the noisy received symbol through an AWGN channel. For the Max-log-MAP algorithm, the expressions to compute the LLR ratio $[L(u_k)]$ are presented in the pseudo-code in figure 2. The equations presented in the psudeo-code form the basis of the architecture described in this paper.

3 Algorithmic to Architecture Mapping

To implement the max-log-MAP algorithm targeting programmability and low power consumption we have chosen a VLIW Application Specific Instruction-Set Processor (ASIP) architecture [3]. The functional units of this processor have been designed to achieve fast decoding required for the UMTS standard. Since the datapath control is microprogrammed, our proposed architecture provides a flexible solution, and can accomodate: (a) variable number of trellis states, (b) variable number of time stages (i.e. block length), (c) sliding window processing [9], (d) block window processing [4], and (e) different generator polynomials at the turbo encoder.

A significant source of power dissipation in a MAP decoder is the memory accesses [7]. To this end, our proposed architecture makes good use of several register files that locally store the metrics that are $consumed\ on\ the\ fly$. This structure is shown in figure 3. Only those values (variables) that are used infrequently and with significant delay are stored in the main memories. Moreover, the architecture has been designed to perform all possible computations once a given set of input metrics have been

```
for k=1 to k < N {   /* forward recursion */
 for(i=0 to s < 2^b) {
  for(s[k,m]) { /*starting state is s[k-1,i])*/
    γ^e[k-1,i;k,m]=½(L_c y^p_k x^p_k)
    γ[k-1,i;k,m]≈½u_k(L_e(u_k)+L_c y^s_k)+γ^e[k-1,i;k,m]
  }
 }
 for(m=0 to m<2^M-1) /* all states at time k */
  α(k,m)=max_|∀s[k-1,i]{α(k-1,i)+ γ[k-1,i;k,m]} -
         max_|(∀s[k-1,i],∀m{α(k-1,i)+ γ[k-1,i;k,m]}
}

for k=N to k>0 /* backward recursion */
  for (m=0 to m < 2^M-1)
    β(k-1,m) = max_|∀s[k,i]{(β(k,m)+ γ[k-1,i;k,m]} -
    max_|∀s[k-1,i],∀m{α(k-1,i)+ γ[k-1,i;k,m]}

for (s=0 to s < 2^m ) {
   L(u_k)= L_C y^s_k+L_e(u_k)+
         max_|S+{α(k-1,i)+γ^e[k-1,i;k,m]+ β(k,m)} -
         max_|S-{(α(k-1,i)+γ^e[k-1,i; k,m]+ β(k,m)}
         = L_C y^s_k+ L^e(uk)+ LLR^+ - LLR^-
     }
}
```

Figure 2: Pseudo-code of Max-Log-MAP Algorithm.

loaded. This can be accomplished by taking advantage of the butterfly computation, which reduces the memory access by 50%. (See the butterfly structure outlined on figure 1, formed by states $s[k-1,i]$, $s[k-1,j]$, $s[k,m]$ and $s[k,n]$).

Computation of α metrics is the most involved due to the normalization requirement (for a given time stage, the maximization term is subtracted from all α and β metrics). We have pipelined the computation of alpha and its normalization by delaying the normalization operation by one cycle in order to be able to complete the whole computation (see section 4.2). Normalization term is common for all the trellis states at a given time stage, and is also used by the beta computation unit. Next section provides details on the implementation of the basic computation units of this ASIP.

4 Hardware Architecture Details

Figure 3 shows the main architecture of the decoder. While performing the implementation of this decoder the datapath was the key element of the design. The design of the functional units that compute the decoding metrics has been customized. As a result, the datapath that is described next exhibits excellent figures in terms of performance and power consumption.

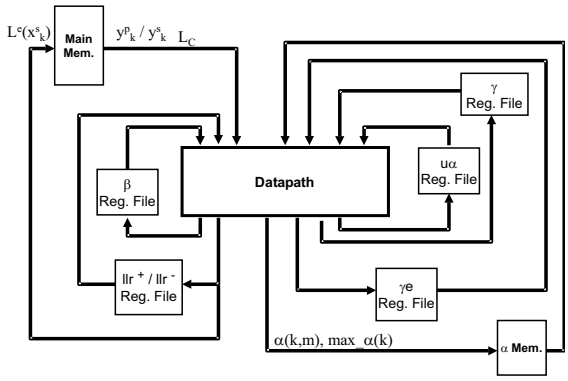

Figure 3: Basic architecture.

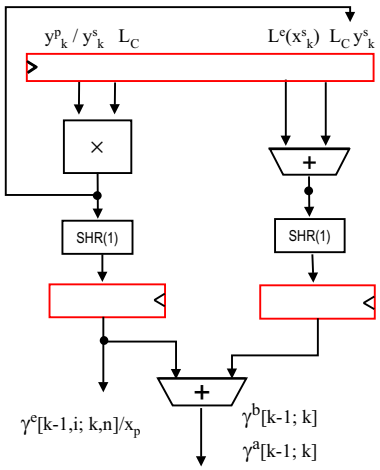

Figure 4: Gamma unit of the datapath.

4.1 Gamma computation

The γ functional unit shown in figure 4 computes the branch metrics, γ and γ^e. The equations that compute these values using the max-log simplification are the following:

$$\gamma^e[k-1,i;k,m] = \frac{1}{2}(L_c y_k^p x_k^p)$$
$$\gamma[k-1,i;k,m] \approx \frac{1}{2}u_k(L^e(u_k) + L_c y_k^s) + \gamma^e[k-1,i;k,m] \quad (1)$$

It is important to note that all branch metrics γ^e have the same absolute value, and that the parity signal, x_p, is the variable that makes the decision of this sign. We have stored these values in intermediate registers to speed-up performance and save power consumption. At any given time stage, there are a total of sixteen branches, of which only four are unique values. Those four values are obtained as a combination of $\pm|\gamma^e| \pm B_k$, with $B_k = 1/2(L_e(u_k) + L_c y_k^s)$. We can redefine γ with $\gamma^a = |\gamma^e| + B_k$ and $\gamma^b = |\gamma^e| - B_k$, which are the two single absolute values this metric can assume. If we just store these two values there are significant savings in terms of:

1. Storage: only two values are stored per time stage. These values are stored in the γ refister file.

2. Power consumption: only two values are loaded, instead of four, and these load operations are performed only once per time stage. Power is reduced by a factor of 16.

The multiplier is reused to compute all multiplications needed for gamma ($L_c y_k^s$ and $L_c y_k^p$), requiring for this purpose a bypass.

4.2 Alpha and Beta computation

α and β metrics must be computed recursively. Both have a common normalization term[1], called in the following $max\_\hat{\alpha}(k)$, which is the second maximization operation in the algorithm pseudo-code ($max_{|\forall s[k-1,i],\forall m}\{\alpha_{k-1}(i)+\gamma_{k-1}(i,m)\}$). Since the normalization term is based on α and γ values, we will calculate $max\_\hat{\alpha}(k)$ while α is being computed and the resulting value is stored in main memory, that is to be used in the backward iteration by the β computation unit.

The value α_k cannot be obtained until $max\_\hat{\alpha}(k)$ has been computed for all the previous $\{\alpha_{k-1} + \gamma_k\}$ values. Thus, the latency of the α computation is extremely long, and hence we have pipelined the computation of α. This pipeline is based on the pre-computation of un-normalized α values ($\hat{\alpha}$ in the following) in the first pipeline phase, and the normalization is performed in the second phase. This can be carried out because finding the maximum implies comparing values and we can compare two numbers before or after subtracting a common term. The following equations describe the foundations of this pipelining:

$$\begin{aligned}
\alpha_k(m) &= max_{|\forall s[k-1,i]}\{\alpha_{k-1}(i) + \gamma_{k-1}(i,m)\} \\
&\quad - max_{|\forall s[k-1,i],\forall m}\{\alpha_{k-1}(i) + \gamma_{k-1}(i,m)\} \\
&= \hat{\alpha}_k(m) - max_\hat{\alpha}_k \\
\hat{\alpha}_k(m) &= max_{|\forall s[k-1,i]}\{\alpha_{k-1}(i) + \gamma_{k-1}(i,m)\} \\
&= max_{|\forall s[k-1,i]}\{\hat{\alpha}_{k-1}(i) - max_\hat{\alpha}_{k-1}(i) \\
&\quad + \gamma_{k-1}(i,m)\} \\
&= max_{|\forall s[k-1,i]}\{\hat{\alpha}(k-1,i) + \gamma_{k-1}(i,m)\} \\
&\quad - max_\hat{\alpha}(k-1) \\
max_\hat{\alpha}_k &= max_{|\forall s[k-1,i],\forall m}\{\alpha_{k-1}(i) + \gamma_{k-1}(i,m)\}
\end{aligned} \quad (2)$$

[1]This normalization is needed for stability reasons.

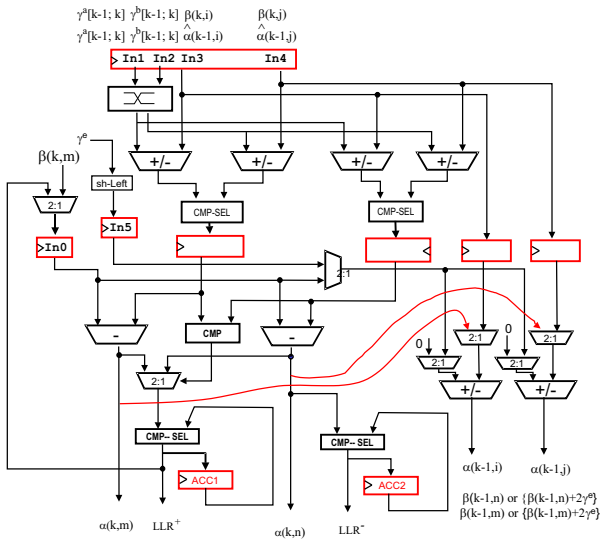

Figure 5: α-β-LLR unit of the datapath.

As these equations show, we can compute α in the first pipeline phase using the un-normalized values computed one time stage ahead ($\hat{\alpha}_k(m)$), and normalize them in the second phase once the normalization term is calculated.

The final implementation of this unit computes two un-normalized metrics $\hat{\alpha}_k(m)$, $\hat{\alpha}_k(n)$ and two normalized metrics, $\alpha_{k-1}(i)$, $\alpha_{k-1}(j)$ every clock cycle, requiring four clock cycles to compute all alpha metrics at a given time step k. We take advantage of having loaded $\alpha_{k-1}(i)$, and $\alpha_{k-1}(j)$, because in the butterfly these metrics are shared by both ending states $s[k,m]$ and $s[k,n]$. Consequently, only two more γ values are required to acheive this speedup of 2 in computation.

Once we have the normalization terms stored in memory, the β computation follows:

$$\beta_{k-1}(m) = max_{|\forall s[k,i]}\{\beta_k(m) + \gamma_{k-1}(i,m)\} - max\_\hat{\alpha}_k \quad (3)$$

4.3 Final LLR computation

For the LLR computation, the operations that must be performed are similar to the γ ones. Again the computation is based on maximization operations which are implemented by add-compare-select structures. The log-likelihood value of the information bit u_k at time k is:

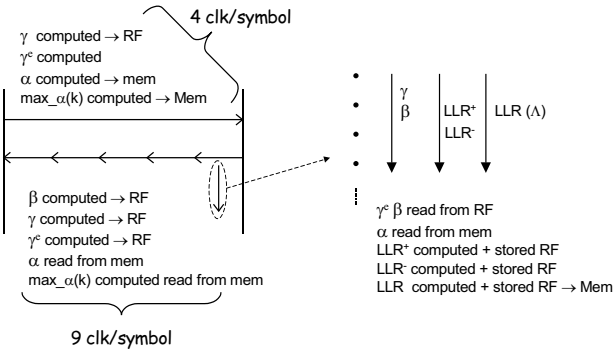

Figure 6: Data and control flow on the MAP architecture.

$$\begin{aligned} L(u_k) &= L_C y_k^s + L^e(u_k) \\ &\quad + max_{|S+}\{\alpha(k-1,i) + \gamma^e[k-1,i;k,m] + \beta(k,m)\} \\ &\quad - max_{|S-}\{\alpha(k-1,i) + \gamma^e[k-1,i;k,m] + \beta(k,m)\} \\ &= L_C y_k^s + L^e(u_k) + LLR^+ - LLR^- \end{aligned} \quad (4)$$

We will reuse the α datapath operators to compute the LLR, and we will simultaneously compute LLR^+ and LLR^-. There is a combination of α and β inputs (those associated to a butterfly) that reduces the number of metric loads and consequently power.

Figure 5 depicts the main structure of the alpha-beta-LLR unit of the datapath. As it can be seen, pipeline registers have been carefully added to balance the delays of the combinational logic.

4.4 Control

Figure 6 shows the control flow that is required for the complete MAP decoding algorithm. The number of clock cycles requried to perform the various computations assume an 8-state trellis, as specified in the UMTS standard. The control of the MAP decoder is microcoded.

The forward recursion computes α and γ metrics concurrently. γ computation must be performed a stage ahead to feed the α computation. The backward recursion computes β and γ metrics for every symbol in parallel, after which the LLR value is calculated.

For their use in the β computation the γ values are not stored in memory because it is more expensive in power terms to access them instead of recomputing them. Actually, the γ unit would be idle if this storage is performed, while no performance degradation would be observed.

Unit	clk freq (MHz)	Area (eq. gates) (eq. gates)	power Cons. (mW)
γ	67	8382	0.887
$\alpha - \beta$	142	10279	20.046
Datapath	142	18561	20.933

Table 1: Synthesis results of the datapath.

5 Synthesis results

The SISO decoder has been synthesized using Agere Systems library LV160C[2]. Table 1 shows the main figures related to the datapath of the ASIP.

The maximum delay of the datapath is related to the alpha unit critical path. In the two stages present there are two adder/subtracter/comparator and some multiplexers. The delay related to this path is 7.03 ns, which provides the final performance of 142 MHz. Nevertheless, the most critical element is the multiplier of the gamma unit, which imposes a path delay of 14.83 ns. This problem can be solved using a pipelined multiplier. The final performance of MAP decoder is 10.9 Msymbols/second, operating at 142 MHz.

6 Conclusions and Future Work

A dedicated architecture for the computation of the max-log-MAP algorithm has been described. The main characteristics of the architecture include: (a) the use of delayed renormalization to reduce the latency in computing α, (b) reduce computation latency by comparing α values before normalization, (c) extensive use of the butterfly computation concept and efficient design of the datapath functional units, and (d) register files have been widely used to store temporary metrics in order to save power consumption.

Future work includes the evaluation of the use of a look-up table to correct the maximization error required by the Log-MAP algorithm.

References

[1] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate. *IEEE Trans. on Information Theory*, pages 284–287, March 1974.

[2] S. Hong and W. E. Stark. Design and Implementation of a Low Complexity VLSI Turbo-Code Decoder for Low Energy Mobile Wireless Comunications. *Journal of VLSI Signal Processing Systems*, (24):43–57, 2000.

[3] Margarida F. Jacome and Gustavo de Veciana. Design Challenges for New Application-Specific Processors. *Design&Test of computers*, 17(2):40–50, Apr-Jun 2000.

[2] Formerly Lucent Technologies library LV160C (1.5V, CMOS).

[4] I. Lee, M. López-Vallejo, and S. A. Mujtaba. Block processing technique for low power turbo decoder design. In *Vehicular Technology Conference*, pages 1025–1029, Birmingham, AL, 2002.

[5] G. Masera, G. Piccinini, M. Ruo Roch, and M. Zamboni. "VLSI architectures for turbo codes," *IEEE Trans. on VLSI Systems*, pages 369–379, 1999.

[6] P. Robertson and P. Hoeher. Optimal and Sub-Optimal Maximum a Posteriori Algorithms Suitable for Turbo Decoding. *European Trans. on Telecommunication*, (8):119–125, Mar-Apr 1997.

[7] C. Schurgers, F. Catthoor, and M. Engels. Energy efficient data transfer and storage organization for a map turbo decoder module. In *ISLPED'99*, pages 76–81. IEEE, 1999.

[8] M.C. Valenti. *Iterative Detection and Decoding for Wireless Communications*. PhD thesis, Virginia Polytechnic Institute and State University, July 1999.

[9] A. J. Viterbi. An Intuitive Justification and a Simplified Implementation of the Map Decoder for Convolutional Codes. *IEEE Jnl. Selected Areas in Comms*, (2):260–264, 1998.

[10] A. Worm, H. Lamm, and N. Wehn. A high Speed MAP Architecture with Optimized Memory Size and Power Consumption. In *Proc. IEEE Workshop of Signal Processing*, pages 265–274, 2000.