Transactions Papers

VLSI Architectures for the MAP Algorithm

Emmanuel Boutillon, Warren J. Gross, *Student Member, IEEE*, and P. Glenn Gulak, *Senior Member, IEEE*

***Abstract*—This paper presents several techniques for the very large-scale integration (VLSI) implementation of the *maximum a posteriori* (MAP) algorithm. In general, knowledge about the implementation of the Viterbi algorithm can be applied to the MAP algorithm. Bounds are derived for the dynamic range of the state metrics which enable the designer to optimize the word length. The computational kernel of the algorithm is the Add-$\mathrm{MAX}^\star$ operation, which is the Add-Compare-Select operation of the Viterbi algorithm with an added offset. We show that the critical path of the algorithm can be reduced if the Add-$\mathrm{MAX}^\star$ operation is reordered into an Offset-Add-Compare-Select operation by adjusting the location of registers. A general scheduling for the MAP algorithm is presented which gives the tradeoffs between computational complexity, latency, and memory size. Some of these architectures eliminate the need for RAM blocks with unusual form factors or can replace the RAM with registers. These architectures are suited to VLSI implementation of turbo decoders.**

***Index Terms*—Forward–backward algorithm, MAP estimation, turbo codes, very large-scale integration (VLSI), Viterbi decoding.**

I. Introduction

In recent years, there has been considerable interest in soft-output decoding algorithms; algorithms that provide a measure of reliability for each bit that they decode. The most promising application of soft-output decoding algorithms are probably turbo codes and related concatenated coding techniques [1]. Decoders for these codes consist of several concatenated soft-output decoders, each of which decodes part of the overall code and then passes "soft" reliability information to the other decoders. The component soft-output algorithm prescribed in the original turbo code paper [1] is usually known as the maximum *a posteriori* (MAP), forward–backward (FB), or Bahl–Cocke–Jelinek–Raviv (BCJR) algorithm [2], [3]. This algorithm, originally described in the late 1960's, was generally overlooked in favor of the less complex Viterbi algorithm [4], [5], moreover, applications taking advantage of soft-output information were not evident. In this paper, we describe techniques for implementing the MAP algorithm that are suitable for very large-scale integration (VLSI) implementation.

The main idea in this paper can be summarized as extending well-known techniques used in implementing the Viterbi algorithm to the MAP algorithm. The MAP algorithm can be thought of as two Viterbi-like algorithms running in opposite directions over the data, albeit with a slightly different computational kernel.

This paper is structured in the following way. Section II is a brief description of the MAP algorithm in the logarithmic domain. Section III studies the problem of internal representation of the state metrics for a fixed-point implementation. Section IV focuses on efficient architectures to realize a forward (or backward) recursion. The log-likelihood ratio (LLR) calculation is also briefly described. Section V proposes several schedules for the forward and backward recursions. As the computations of the forward and the backward recursions are symmetrical in time (i.e., identical in terms of hardware computation), only the forward recursion is described in Sections III and IV.

II. MAP Algorithm

A. Description of the Algorithm

The MAP algorithm is derived in [3] and [6] to which the reader is referred to for a detailed description. The original derivation of the MAP algorithm was in the probability domain. The output of the algorithm is a sequence of decoded bits along with their reliabilities. This "soft" reliability information is generally described by the *a posteriori* probability (APP) $P(u|y)$. For an estimate of bit u $(-1/+1)$ having received symbol y, we define the optimum soft output as

$$L(u) = \ln \frac{P(u=+1|y)}{P(u=-1|y)} \tag{1}$$

which is called the log-likelihood ratio (LLR). The LLR is a convenient measure, since it encapsulates both soft and hard bit information in one number. The sign of the number corresponds to the hard decision while the magnitude gives a reliability estimate. The original formulation of the MAP algorithm requires multiplication and exponentiation to calculate the required probabilities.

In this paper, we consider the MAP algorithm in the logarithmic domain as described in [7]. The MAP algorithm, in its native form, is challenging to implement because of the exponentiation and multiplication. If the algorithm is implemented in

Paper approved by R. D. Wesel, the Editor for Coding and Communication Theory of the IEEE Communications Society. Manuscript received September 1, 1999; revised July 13, 2001 and July 2, 2002. This paper was presented in part at the 5'eme Workshop AAA sur l'Adequation Algorithme Architecture, INRIA Rocquencourt, France, January 26–28, 2000.

E. Boutillon is with L.E.S.T.E.R, Université de Bretagne Sud, 56325 Lorient Cedex, France (e-mail: Emmanuel.Boutillon@univ-ubs.fr).

W. J. Gross and P. G. Gulak are with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 3G4, Canada (e-mail: wjgross@eecg.utoronto.ca; gulak@eecg.utoronto.ca).

Digital Object Identifier 10.1109/TCOMM.2003.809247

the logarithmic domain like the Viterbi algorithm, then the multiplications become additions and the exponentials disappear. Addition is transformed according to the rule described in [8]. Following [9], the additions are replaced using the Jacobi logarithm

$$\begin{aligned}\mathrm{MAX}^{\star}(x,y) &= \ln(e^{x}+e^{y}) \\ &= \mathrm{MAX}(x,y)+\ln(1+e^{-|x-y|}) \end{aligned} \tag{2}$$

which is called the $\mathrm{MAX}^{\star}$ operation, to denote that it is essentially a maximum operator adjusted by a correction factor. The second term, a function of the single variable $x-y$, can be precalculated and stored in a small lookup table (LUT) [9]. The computational kernel of the MAP algorithm is the Add–$\mathrm{MAX}^{\star}$ operation, which is analogous, in terms of computation, to the Add–Compare–Select (ACS) operation in the Viterbi algorithm adjusted by an offset known as a correction factor. In what follows, we will refer to this kernel as ACSO (Add–Compare–Select–Offset).

The algorithm is based on the same trellis as the Viterbi algorithm. The algorithm is performed on a block of N received symbols which corresponds to a trellis with a finite number of stages N. We will choose the transmitted bit u_k from the set of $\{-1,+1\}$. Upon receiving the symbol $\mathbf{y}_k$ from the additive white Gaussian noise (AWGN) channel with noise variance σ^2, we calculate the branch metrics of the transition from state s' to state s as

$$G_k(s',s)=\frac{-1}{2\sigma^2}\|\mathbf{y}_k-\mathbf{c}_k(s',s)\|^2 \tag{3}$$

where $\mathbf{c}_k(s',s)$ is the expected symbol along the branch from state s' to state s. The multiplication by $-1/2\sigma^2$ can be done with either a multiplier or an LUT. Note that in the case of a turbo decoder which uses several iterations of the MAP algorithm, the multiplication by $-1/2\sigma^2$ need only be done at the input to the first MAP algorithm [6].

The algorithm consists of three steps.

• Forward Recursion. The forward state metrics are recursively calculated and stored as

$$A_k(s)=\mathrm{MAX}^{\star}_{s'}(A_{k-1}(s')+G_k(s',s)),\quad k=1,\ldots,N-1. \tag{4}$$

The recursion is initialized by forcing the starting state to state 0 and setting

$$\begin{aligned}A_0(0) &= 0 \\ A_0(s) &= -\infty, \qquad s\neq 0. \end{aligned} \tag{5}$$

• Backward Recursion. The backward state metrics are recursively calculated and stored as

$$B_{k-1}(s')=\mathrm{MAX}^{\star}_{s}(B_k(s)+G_k(s',s)),\quad k=N,\ldots,2. \tag{6}$$

The recursion is initialized by forcing the ending state to state 0 and setting

$$\begin{aligned}B_N(0) &= 0 \\ B_N(s) &= -\infty, \qquad s\neq 0. \end{aligned} \tag{7}$$

The trellis termination condition requires the entire block to be received before the backward recursion can begin.

• Soft-Output Calculation. The soft output, which is called the LLR, for each symbol at time k is calculated as

$$\begin{aligned}L(u_k) = &\mathop{\mathrm{MAX}^{\star}}_{\substack{(s',s)\\ u_k=+1}}(A_{k-1}(s')+G_k(s',s)+B_k(s)) \\ &-\mathop{\mathrm{MAX}^{\star}}_{\substack{(s',s)\\ u_k=-1}}(A_{k-1}(s')+G_k(s',s)+B_k(s)) \end{aligned} \tag{8}$$

where the first term is over all branches with input label $+1$, and the second term is over all branches with input label -1.

The MAP algorithm, as described, requires the entire message to be stored before decoding can start. If the blocks of data are large, or the received stream continuous, this restriction can be too stringent; "on-the-fly" decoding using a sliding-window technique has to be used. Similar to the Viterbi algorithm, we can start the backward recursion from the "all-zero vector" B^0 (i.e., all the components of B^0 are equal to zero) with data $\{y_k\}$, k from n down to $n-L$. L iterations of the backward recursion allows us to reach a very good approximation of $g+B_{n-L}$ (where g is a positive additive factor) [10], [11]. This additive coefficient does not affect the value of the LLR. In the following, we will consider that after L cycles of backward recursion, the resulting state metric vector is the correct one. This property can be used in a hardware realization to start the effective decoding of the bits before the end of the message. The parameter L is called the convergence length. For on-the-fly decoding of nonsystematic convolutional codes as discussed in [10] and [11], five to ten times the constraint length was found to lead only to marginal signal-to-noise ratio (SNR) losses. For turbo decoders, due to the iterative structure of the computation, an increased value of L might be required to avoid an error floor. A value of $L=64$ is reported in [12] for a recursive systematic code with a constraint length of five. In practice, the final value of L has to be determined via system simulation and analysis of the particular decoding system at hand.

B. Upper Bounds for $\mathrm{MAX}^{\star}$

All the following upper bounds are derived from the definition of $\mathrm{MAX}^{\star}$ in (2):

$$\mathrm{MAX}^{\star}(x,y)\geq \mathrm{MAX}(x,y). \tag{9}$$

For practical implementation, one can notice that, due to the finite precision of the hardware implementation, the function $\ln(1+e^{-|x-y|})$ gives a zero result as soon as $|x-y|$ is large enough. For example, if the values are coded in fixed precision with three binary places (a quantum of 0.125), then $|x-y|>2.5\Rightarrow \ln(1+e^{-|x-y|})<0.079$, thus it will be rounded to 0. In that case, the computation of the offset of the $\mathrm{MAX}^{\star}$ operator can be performed with two pieces of information: a Boolean Z (for zero) that indicates if $|x-y|$ is above or equal to the first power of two greater than 2.5, i.e., four. If Z is true, then the offset is equal to 0. If not, its exact value is computed with the five least significant bits of $|x-y|$. The maximum number is $\ln(2)$, which will be quantized to 0.75, i.e., the width of the

LUT is three bits for our example. An LUT is the most straightforward way to perform this operation [9], [15]. In the general case, there is a positive value δ such that

$$\mathrm{MAX}^\star(x,y) = \mathrm{MAX}(x,y), \qquad \text{if } |x-y| > 2^\delta. \tag{10}$$

We also have

$$\mathrm{MAX}^\star(x,y) \leq \mathrm{MAX}(x,y) + \ln(2) \tag{11}$$

with equality if $x = y$ (note that quantization of the LUT is also discussed in [14]).

III. Precision of State Metrics

The precision of the state metrics is an important issue for VLSI implementation. The number of bits n_{sm} used to code the state metrics determines both the hardware complexity and the speed of the hardware. This motivates the need for techniques which minimize the number of bits required without modifying the behavior of the algorithm.

The same problem has been intensively studied for the Viterbi algorithm ([16], [17]) and solutions using rescaling or modulo $2^{n_{sm}}$ arithmetic are widely used [18], [19]. These techniques are based on the fact that, at every instant k, the dynamic range $\Delta(k)$ of a state metric (i.e., the difference between the state metrics with the highest and lowest values), is bounded by $\Delta_{\max}$.

The forward recursion in the MAP algorithm is slightly different than the Viterbi algorithm since:

1) the outputs of the recursion are the state metrics themselves and not the decisions of the ACS;
2) the Add–MAX* is an ACS operation with an added offset (ACSO).

These differences lead us to question whether the well-known implementation techniques for the Viterbi algorithm are also applicable to the MAP algorithm. The first part of this section shows that the LLR result is independent of a global shift of all of the state metrics of the forward and backward recursions. The bounds on the dynamic range of the state metric are then given.

A. Rescaling the State Metrics

Let us first show that the MAX* operator is shift invariant, that is, it still produces a valid result if both of its arguments have a common constant added to them. Let x, y, and z be real numbers. From the definition of MAX*, it follows that:

$$\begin{aligned}&\mathrm{MAX}^\star(x+z, y+z)\\ &\quad = \mathrm{MAX}(x+z, y+z) + \ln(1 + e^{-|(x+z)-(y+z)|}).\end{aligned} \tag{12}$$

Thus

$$\begin{aligned}\mathrm{MAX}^\star(x+z, y+z) =& \mathrm{MAX}(x,y)\\ &+ z + \ln(1+e^{-|x-y|})\\ =& \mathrm{MAX}^\star(x,y) + z.\end{aligned} \tag{13}$$

According to (13), the MAX* operator is linear. Thus, a global shift of z for all $A_{k-1}(s')$ values (or $B_k(s)$) would not change the value of $L(u_k)$, since the contribution of z, when put outside the two MAX* operators, is cancelled. Thus, it is the differences between the state metrics and not their absolute values that are important. Rescaling of the state metrics can be performed.

B. Approximate Bound on the Dynamic Range of the State Metrics

Let us define l as the minimum number such that, for all $k > 0$, there is a path through the trellis between every state at time k and $k+l$. Let us define M as the maximum absolute value of the branch metric. Then, for all $k \geq l$, a rough bound on the dynamic range of the state metrics is

$$\Delta(k) \leq (2M + \ln(2))l. \tag{14}$$

Proof: Let $sm_{\max}(k)$ and $sm_{\min}(k)$ be, respectively, the maximum and minimum value of the state metric at time k. Then, according to the definition of l, there is a path of length l in the trellis between every state s at time k and the state s_m with value $sm_{\max}(k-l)$ at time $k-l$. Since at every step, the maximum state metric (with, eventually, a positive correction factor) is taken, in the worst case, among the path between s_m and s, the state metric can decrease by M at each step. Thus, $sm_{\min}(k)$ is at least equal to or greater than $sm_{\max}(k-l) - Ml$. Similarly, the maximum increase at each stage of the state metric among this path is $M + \ln(2)$ ($\ln(2)$ is the maximum value of the correction factor added at each stage). Thus, $sm_{\max}(k)$ is lower than, or equal to, $sm_{\max}(k-l) + Ml + \ln(2)l$. Grouping the upper bound of $sm_{\min}(k)$ and the lower bound of $sm_{\max}(k)$ leads to (14). ■

Note that in the case of a trellis corresponding to a shift register of length ν, l is equal to ν.

C. Finer Bound on the Dynamic Range of the State Metrics for a Convolutional Decoder

A more precise bound can be obtained in the case of a convolutional decoder using the intrinsic properties of the encoder. All the following developments are based on a previous work based on the Viterbi algorithm [17]. Note that this problem has already been independently addressed by Montorsi *et al.* [20], where they extend through intensive simulation, without any formal proof, the result obtained in [21] for the case of Viterbi algorithm.

1) Exact Bound on $\Delta(k)$: The lower bound of Δ can be obtained using the Perron–Frobenius theorem [25]. Let us work in the probability domain and let us assume that the branch probabilities $\gamma_k(s',s)$

$$\gamma_k(s',s) = \exp(G_k(s',s)) = \exp\left(\frac{-1}{2\sigma^2}\|\mathbf{y}_k - \mathbf{c}_k(s',s)\|^2\right) \tag{15}$$

are normalized so that

$$\sum_{s',s} \gamma_k(s',s) = 1. \tag{16}$$

In a real system, the $\mathbf{y}_k$ are bounded (by the analog–digital conversion) and the standard deviation of the noise σ is a nonzero value, thus, according to (15) and (16), we have the relation

$$0 < \gamma_k(s',s) < 1, \qquad \text{for all } s, s'. \tag{17}$$

Let us first assume that the all-zero path is sent in the channel and that all the received symbols have the highest possible reliability. The forward recursion is performed on the received symbols. Let us study, in this case, the ratio of state probabilities between the state with the highest probability and the state with the lower probability when the forward recursion is performed. Note that this ratio, in the log domain, is associated with the maximum difference between state metrics, i.e., the dynamic range of the state metric.

The initial state vector $\alpha_0 = (\alpha_0(s) = 1/P)_{0\le s<P}$ is the uniformly distributed vector of length P, where P is the number of states of the trellis.

Since by hypothesis all the branch metrics are independent of time, we can express the forward recursion in an algebraic form using $P \times P$ transition matrix Γ

$$\alpha_{k+1} = \alpha_k \Gamma. \tag{18}$$

By recursion, we have

$$\alpha_k = \alpha_0 \Gamma^k. \tag{19}$$

By construction, Γ is a positive irreducible matrix (the coefficients are positive, and Γ only performs a modification of the probability distribution of the state metric vector). Thus, according to the Perron–Frobenius theorem [25], Γ can be expressed, in the basis of eigenvectors $(V_i)_{i=0\ldots P-1}$, by a diagonal matrix $\mathrm{Diag}(\lambda_i)_{i=0\ldots P-1}$ with the two properties:

1) V_0, the Perron eigenvector of Γ, is the only eigenvector of Γ that has all of its components positive;
2) λ_0, the Perron eigenvalue associated with V_0 is positive and $|\lambda_i| \le \lambda_0$ for all $0 < i < P$.

Since, in the trellis, all the states at time k are connected to the states at time $k+\nu$, we deduce that all the coefficients of Γ^ν are strictly positive. Using the Perron–Frobenius theorem for Γ^ν gives an extra property: the Perron eigenvalue of Γ^ν is strictly greater than its other eigenvalues. From this property, we deduce that this property is also true for Γ, i.e., $|\lambda_i| < \lambda_0$ for all $0 < i < P$.

Let $(a_i)_{i=0\ldots P-1}$ be the decomposition of α_0 in the basis $(V_i)_{i=0\ldots P-1}$. The vector α_k can be expressed as

$$\alpha_k = \sum_{i=0}^{P-1} a_i \cdot \lambda_i^k \cdot V_i = \lambda_0^k \cdot \left(a_0 \cdot V_0 + \sum_{i=1}^{P-1} a_i \cdot \mu_i^k \cdot V_i \right) \tag{20}$$

with $|\mu_i| = |\lambda_i/\lambda_0| < 1$, for $i = 1 \ldots P-1$.

Let us call $\mathrm{MAX}(X)$ (respectively, $\mathrm{MIN}(X)$) the maximum (minimum) coordinate value of vector X, and $\delta(X)$ the ratio $\mathrm{MAX}(X)/\mathrm{MIN}(X)$.

Conjecture: For all k, $1 \le \delta(\alpha_k) \le \delta(V_0)$.

Proof: First, $\delta(\alpha_0) = 1$, since $\mathrm{MAX}(\alpha_0) = \mathrm{MIN}(\alpha_0) = 1/P$. Second, using (20), we have

$$\lim_{k\to\infty} \alpha_k \cdot \lambda_0^{-k} = a_0 \cdot V_0 \tag{21}$$

and thus

$$\lim_{k\to\infty} \delta(\alpha_k) = \delta(V_0). \tag{22}$$

Finally, we justify the monotonic increasing of $\delta(\alpha_k)$ (which achieves the proof) by an intuitive argument. $\delta(\alpha_k)$ is the likelihood ratio between the state that has the highest probability (state 0, by construction) and the state with the lowest probability. Since every new incoming branch metric confirms state 0, $\delta(\alpha_k)$ is an increasing function of k.

Using the same type of argument, if one, or more, of the first k received signals do not have the highest reliability, the resulting ratio $\delta'(\alpha_k)$ will be smaller than $\delta(\alpha_k)$.

Since the code is linear, the result obtained for the all-zero sequence is true for all sequences of bits. Thus, the logarithm of the ratio $\delta(V_0)$ gives the maximum differences Δ of the state metric.

2) Exact Bound in Finite Precision: The exact maximum difference Δ obtained with a fixed precision architecture is obtained from (19) starting from the all-zero vector until the system reaches stationarity, i.e., if all state metrics increase by the same constant value at each iteration, $\Delta(k)$ is then equal to Δ.

Note that this algorithm is a generalization of the algorithm proposed in [22] for the case of Viterbi decoder.

3) Simplification of the Computation of the Branch Metrics for a Convolutional Decoder: For a rate $1/r$ convolutional code, $\mathbf{c}_k(s', s)$ is an r-dimensional vector with elements $\{-1, +1\}$ (or $\{-1, 0, +1\}$ in the case of a punctured code where 0 is used for a punctured bit). Using (13), the computation of the branch metrics can be extended and simplified

$$G_k(s', s) = \frac{-1}{2\sigma^2}\left(\|\mathbf{y}_k\|^2 + \|\mathbf{c}_k(s', s)\|^2\right) + \frac{1}{\sigma^2}(\mathbf{y}_k \cdot \mathbf{c}_k(s', s)). \tag{23}$$

The first terms are common to all branch metrics, thus, they can be dropped. The last terms can be decomposed on the r dimensions of the vector. Thus, the modified branch metrics $G'_k(s', s)$ are

$$G'_k(s', s) = \frac{1}{\sigma^2} \cdot \sum_{0\le i<r} y_k^i \cdot \mathbf{c}_k^i(s', s), \qquad \mathbf{c}_k^i(s', s) \in \{-1, 0, 1\} \tag{24}$$

where $\mathbf{c}_k^i(s', s)$ takes the value of zero for a punctured code symbol. This expression can be used to find the exact bound of $\Delta(k)$.

4) Example: As an example, let us consider a recursive systematic encoder with generator polynomials (7, 5). Moreover, let us assume that the modified branch metrics $G'_k(s', s)$ are coded using 128 levels, from -15.75 up to 15.75 (the inputs y_k^i/σ^2 are coded between -7.875 up to 7.875, with a step size of 0.125). We assume that the all-zero path is received with the maximum reliability. The resulting state transition diagram (with values of modified branch metrics) is given in Fig. 1. Table I shows the evolution of the state metrics for the first eight iterations of the forward recursion.

As shown in Table I, the value of $\Delta(k)$ does not increase after seven iterations. The limit value $\Delta' = 47.250$ is the maximum value of the state metric dynamic range obtained for our example. The approximate bound of (14) gives, for this example, $\Delta < 64.375$. The bound obtained by the above method is much more precise and can lead to more efficient hardware realizations, since the precision of the state metrics is reduced.

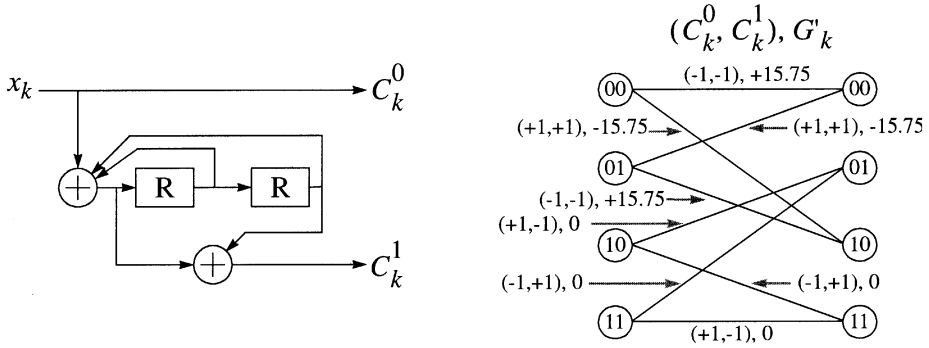

Fig. 1. State transition of a systematic recursive encoder with polynomials (7,5) and modified branch metric when, for all k, $(y_k^0/\sigma^2, y_k^1/\sigma^2) = (-7.875, -7,875)$.

TABLE I
VARIATION OF STATE METRICS AFTER t STAGES ON THE ALL-ZERO PATH

	$t=0$	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$
0	0.000	15.75	31.500	47.250	63.000	78.75	94.500	110.250	126.000
1	0.000	0.750	15.750	16.875	31.500	33.125	48.000	63.000	78.750
2	0.000	15.75	16.500	31.50	32.875	48.000	63.000	78.750	94.500
3	0.000	0.750	15.750	16.875	31.500	33.125	48.000	63.000	78.750
$\Delta(k)$	0.000	15	15.75	30.725	31.500	45.625	46.500	47.250	47.250

Note that the initial state vector is important (the all-zero vector). In the case where the initial state is known (state 0, for example), using an initial state that gives the highest probability possible for state zero and the lowest probability for all the other states can lead to some transitory $\Delta(k)$ values greater than Δ'. The natural solution to avoid this problem is to use the obtained eigenvector (vector (47.250, 0, 15.750, 0) in this example). For turbo-decoder applications, the method can also be used, taking into account the extrinsic information as the initial state.

IV. ARCHITECTURE FOR THE FORWARD AND BACKWARD RECURSIONS

This section is divided into two parts. The first part is a review of the architecture usually used to compute the forward state metrics [9]. The second part is an analysis of the position of the register for the recursion loop in order to increase the speed of the architecture.

A. Computation of the Forward State Metrics: ACSO Unit

The architecture of the processing unit that computes a new value of $A_k(s)$ is shown in Fig. 2. The structure consists of the well-known ACS unit used for the Viterbi algorithm (grey area in Fig. 2) and some extra hardware to generate the "offset" corresponding to the correction factor $\ln(1+e^{-|x-y|})$ of (2).

As said in Section II, the offset is generated directly with a LUT that contains the precalculated result of $\ln(1+e^{-|x-y|})$. Then, the offset is added to the result of the ACS operation to generate the final value of $A_k(s)$. In the following, we will call this processor unit an ACSO unit.

B. Architecture for the Forward State Metric Recursion

The natural way to perform the forward state metric recursion is to place a register at the output of the ACSO unit, in order to keep the value of $A_k(s)$ for the next iteration. This architecture

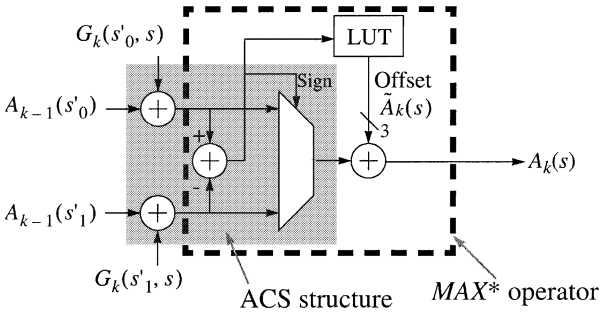

Fig. 2. Architecture of an ACSO.

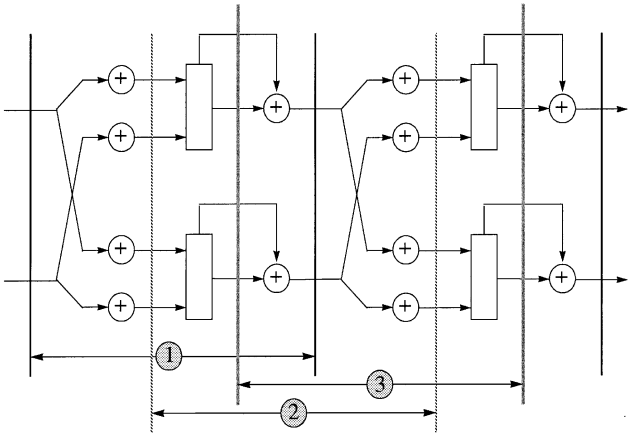

Fig. 3. Three different positions of the register in the data flow of the forward (or backward) algorithm leading to three types of ACSO recursion architectures.

is the same as the one used for the Viterbi algorithm, and all the literature on the speed-area tradeoffs for the ACS recursion can be reused for the ACSO computation. Nevertheless, there is another position for the register which reduces the critical path of the recursion loop. Fig. 3 shows two steps of a two-state trellis.

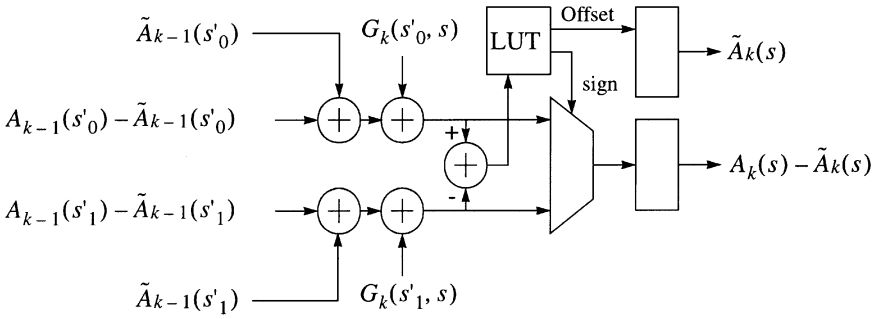

Fig. 4. Architecture of an OACS unit.

Three different positions of the recursion loop register are shown. The first position is the classical one. It leads to an ACSO unit. The second position leads to a compare-select-offset-add (CSOA) unit, while the third position leads to an offset-add-compare-select (OACS) unit. The last one, the OACS unit shown in Fig. 4, has a smaller critical path compared with the ACSO unit. Briefly, in the case of a ACSO unit, the critical path is composed of the propagation of the carry (t_c) in the first adder, the propagation of one full adder ($t_{\rm fa}$) for the comparison (as soon as a result of the sum is available, it can be used for the comparison), the time of the LUT access ($t_{\rm LUT}$) and the multiplexer ($t_{\rm mux}$), and then, once more, the time of the propagation of the carry in the offset addition. For the OACS unit, the critical path is only composed of the propagation of the carry in the first adder (the addition of the offset), the propagation of one full adder for the addition of the branch metric, another propagation of one full adder for the comparison, and then, the maximum of the LUT access and the multiplexer. Thus, the critical path is decreased from

$$t_{\rm acso} = n_{sm} \cdot t_c + t_{\rm fa} + \mathrm{MAX}(t_{\rm LUT}, t_{\rm mux}) + n_{sm} \cdot t_c \quad (25)$$

to

$$t_{\rm oacs} = n_{sm} \cdot t_c + 2 \cdot t_{\rm fa} + \mathrm{MAX}(t_{\rm LUT}, t_{\rm mux}). \quad (26)$$

The decrease of the critical path is paid for by an additional register needed to store the offset value between two iterations. The area-speed tradeoff is determined by the specification of the application. As mentioned by one of the paper's reviewers, a Carry–Save–Adder (CSA) architecture can also be efficiently used in this case [23].

The last step of the MAP algorithm is the computation of the LLR value of the decoded bit. Parallel architectures for the LLR computation can be derived directly from (8). The first stage is composed of $2^{\nu+2}$ adders. The second stage is composed of two 2^{ν} operand $\mathrm{MAX}^{\star}$ operators. Finally, the last operation is the subtraction. A classical tree architecture can be used for the hardware realization of the $2^{\nu} - 1$ operand $\mathrm{MAX}^{\star}$ operators.

V. General Architecture

Each element of the MAP architecture has now been described. The last part of our survey on VLSI architectures for the MAP algorithm is the overall organization of the computation. Briefly speaking, the generation of the LLR values requires both A_k and B_k values, which are generated in chronologically reverse order. The first implication is that, somehow, memory is needed to store a given type of vector (say, A_k), until the corresponding vector (B_k) is generated. Each state metric vector is composed of 2^{ν} state metrics (the size of the trellis), each one n_{sm} bits wide. The total number of bits for each vector is large ($2^{\nu} n_{sm}$) and thus, the reduction of the number of state metrics is an important issue for minimizing the implementation area.

The first part of this section describes the architecture of a high-speed VLSI circuit for the forward algorithm. Then, through different steps, we propose several organizations of computation that reduce the number of vectors that need to be stored by up to a factor of eight. Note that several authors have separately achieved similar results. This point will be discussed in the last section.

A. Classical Solutions [($n_A = 1, n_B = 2, M_A$) and ($n_A = 1, n_B = 2, M_B$)] Architecture

The first real-time MAP VLSI architectures in the literature are described in [11], [13], and [24]. The architecture of [11] and [13] is based on three recursion units (RUs), two used for the backward recursion (RU_{B1} and RU_{B2}), and one forward unit (RU_A). Each RU contains 2^{ν} $\mathrm{MAX}^{\star}$ operators working in parallel so that one recursion can be performed in one clock cycle. The two backward RUs play a role similar to the two trace-back units in the Viterbi decoder of [26].

Let us use the same graphical representation as in [11], [27], and [28] to explain the organization of the computation. In Fig. 5, the horizontal axis represents time, with units of a symbol period. The vertical axis represents the received symbol. Thus, the curve ($x = y$) shows that, at time $t = k$, the symbol $\{y_k\}$ becomes available. Let us describe how the L symbols $\{y_k\}_{L \le k < 2L}$ are decoded (segment I of Fig. 5). From $t = 3L$ to $4L - 1$, RU_{B1} performs L recursions, starting from y_{3L-1} down to y_{2L} (segment II of Fig. 5). This process is initialized with the all-zero state vector B^0, but after L iterations, as noted in [11], the convergence is reached and B_{2L} is then obtained. During those same L cycles, RU_A generates the vectors $\{A_k\}_{L \le k < 2L}$ (segment III of Fig. 5). The vectors $\{A_k\}_{L \le k < 2L}$ are stored in the state vector memory (SVM) until they are needed for the LLR computation (grey area of Fig. 5). Then, between $t = 4L$ and $5L - 1$, RU_{B2} starts from state B_{2L} to compute B_{2L-1} down to B_L (segment IV of Fig. 5). At each cycle, the vector A_k corresponding to the computed B_k is extracted from the memory in order to compute $L(u_k)$. Finally, between $t = 5L$ and $6L - 1$, the data are

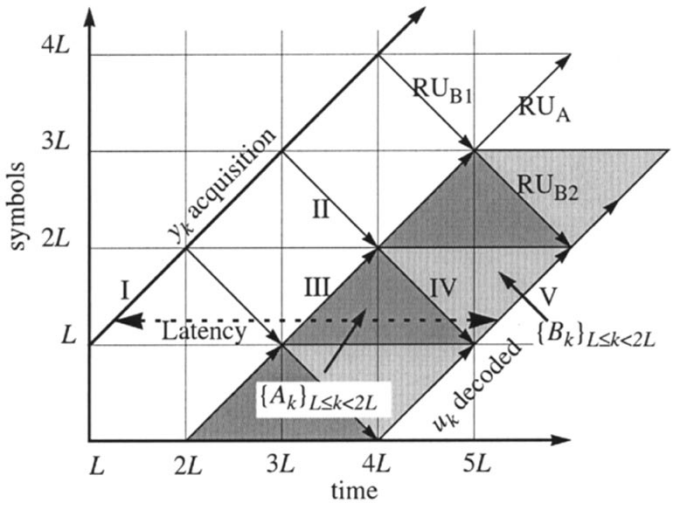

Fig. 5. Graphical representation of a real-time MAP architecture.

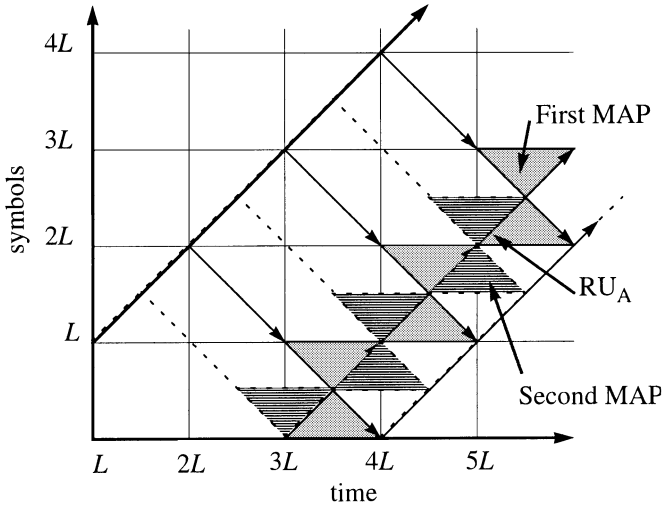

Fig. 6. Graphical representation of the $(n_A = 1, n_B = 2, M_{(A+B)/2})$ architecture.

reordered (segment V of Fig. 5) using a memory for reversing the LLR (light grey area of Fig. 5). The same process is then reiterated every L cycles, as shown in Fig. 5.

In the case where the MAP unit is being used in a turbo decoder, the reordering can be done implicitly by the interleaver. Moreover, the *a priori* information to be subtracted [1] can be reversed in time in order to be directly subtracted after generation of the LLR value (segment IV of Fig. 5). Note that the role of the memories is to reverse the order of the state vectors. Reordering of the state metrics can be done with a single RAM and an up/down counter during L clock cycles. The L incoming data are stored at addresses $0, \ldots, L-1$. In the next L cycles, the counter counts down and the state vectors are retrieved from $L-1, \ldots, 0$ and at the same time, the new incoming state vectors are stored in the same RAM block (from addresses $L-1$ down to 0). Only one read/write access is done at the same location every clock cycle. This avoids the need for multiport memories.

This graphical representation gives some useful information about the architecture. For example, the values of:

1) the decoding latency: $D = 4L$ (horizontal distance between the array "acquisition" and "decoded bit");
2) the number of A vectors to be stored: $M = L$ (maximum vertical size of the grey area);
3) the "computational cost" of the architecture, i.e., the total number of forward and backward iterations performed for each received data: $P = 3$ (the number of arrows of RU cut by a vertical line).

Note that to perform the recursions, branch metrics have to be available. This can easily be done using three RAMs of size L that contain the branch metrics of the three last received blocks of size L. Note that the RAM can simply store the received symbols. In that case, branch metrics are computed on the fly every time they are needed. Since the amount of RAM needed to store branch metric information is small compared with the amount of RAM needed to store the state metric, evaluation of branch metric computation will be omitted in the rest of the paper.

In what follows, this architecture is referred to as $(n_A = 1, n_B = 2, M_A)$, where n_A and n_B are, respectively, the number of RUs used for the forward and backward recursions. M_A (for the memory of state metric A) indicates that the A vectors are stored for the LLR bit computation. Note that in this architecture, the forward recursion is performed $2L$ cycles after the initial reception of data.

With the $(n_A = 1, n_B = 2, M_A)$ architecture, L state vectors have to be stored. The length of convergence L is relatively small (a few times the constraint length ν) but the size of the state vector is very large. In fact, a state vector is composed of 2^ν state metrics, each state metric is n_{sm} bits wide, i.e., $2^\nu n_{sm}$ bits per state metric vector. The resulting memory is very narrow, and thus, not well suited for a realization with a single RAM block, but it can be easily implemented by connecting several small RAM blocks in parallel.

The architecture $(n_A = 1, n_B = 2, M_B)$ is reported in [10]. It is equivalent to the former one, except that the forward recursion is performed $4L$ cycles after the reception of the data, instead of $2L$ cycles (segment V of Fig. 5 instead of segment III). In this scheme, the B vectors generated by RU_{B2} are stored until the computation of the corresponding A vectors by RU_A (light grey of Fig. 5). Then, the LLR values are computed in the natural order.

Other architectures have been developed. Each presents different tradeoffs between computational power, memory size, and memory bandwidth. Their graphical representations are given below.

B. $(n_A = 1, n_B = 2, M_{(A+B)/2})$ Architecture

In this architecture, the forward recursion is performed $3L$ cycles after the reception of the data (see Fig. 6). Thus, $L/2$ A vectors and $L/2$ B vectors have to be stored. The total number of state vectors to be stored is still L. Moreover, with this solution, L bits have to be decoded in the last $L/2$ clocks cycles of an iteration, thus, two APP units have to be used. This scheme becomes valuable when two independent MAP decoders work in parallel. Since two MAP algorithms are performed in parallel, it is possible to share the L memory words between the two MAP algorithms by an appropriate interleaving of the two operations,

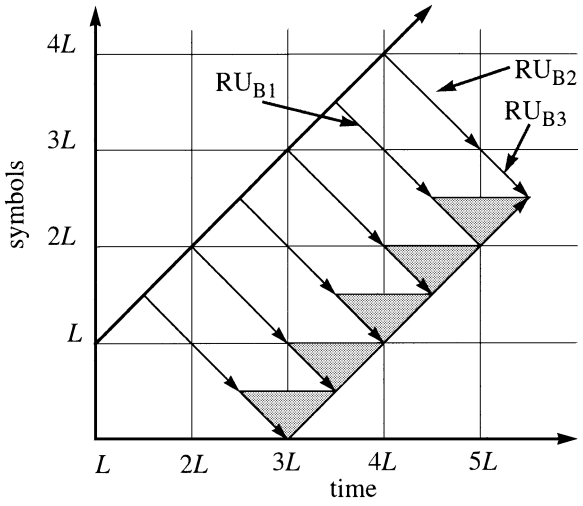

Fig. 7. Graphical representation of the $(n_A = 1, n_B = 3, M_B)$ architecture.

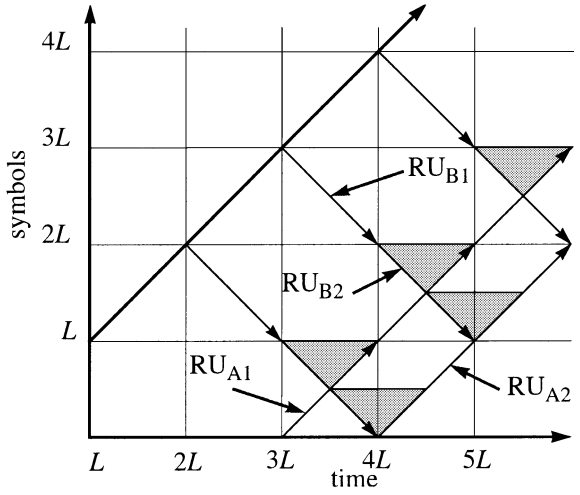

Fig. 8. Graphical representation of the $(n_A = 2, n_B = 2, M_B)$ architecture.

as shown in Fig. 6. In this figure, the second iteration is represented with dotted lines and the corresponding vector memory with a striped region. This scheme can be used in a pipeline of decoders to simultaneously decode two information streams. With this interleaving, the amount of memory for state metrics corresponding to each MAP is divided by two. Thus, the final area will be smaller than the simple juxtaposition of two "classical" MAP algorithms. With this solution, two read and two write accesses are needed at each symbol cycle. Those accesses can be shared harmoniously with two RAMs of size $L/2$ with a read and write access at the same address for each of the two RAMs.

The MAP architecture can use more than two RUs for the backward recursion and/or more than one RU for the forward recursion. The following sections describe some interesting solutions.

C. $(n_A = 1, n_B = 3, M_B)$ Architecture

An additional backward unit leads to the schedule of Fig. 7. A new backward recursion is started every $L/2$ cycles on a length of $L+L/2$ symbols. The first L steps are used to achieve convergence, then the last $L/2$ steps generate $L/2$ vectors B. The new latency is now $3L$, and the amount of memory needed to store the B vectors is only $L/2$. Two observations are worth noting:

1) the reduction of the latency and the memory size is paid for by a new backward unit;
2) a solution of type $(n_A = 2, n_B = 2, M_{(A+B)/2})$ can also be used.

D. $(n_A = 2, n_B = 2, M_B)$ Architecture

The addition of an extra forward unit can also decrease the SVM by a factor of two, as shown in Fig. 8. This scheme has the same number of processing units $(n_A + n_B = 4)$ and the same state metric memory size $M = L/2$ as the $(n_A = 1, n_B = 3, M_B)$ architecture, but its latency is $4L$ compared with $3L$ for the architecture of the previous section. However, the second recursion unit RU_{A2} can be simplified, since it only copies, with a time shift of L cycles, the computation of RU_{A1}. Thus, there exists a tradeoff between computational complexity and memory.

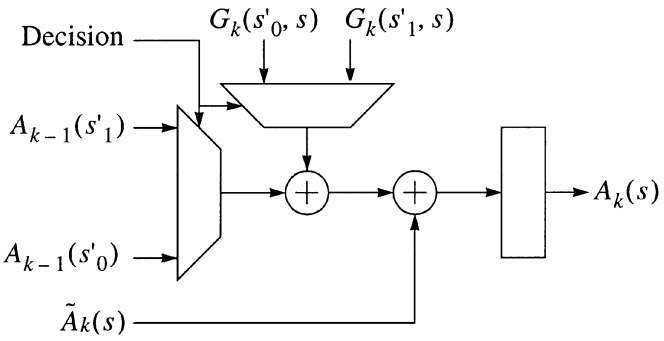

Fig. 9. Simplified ACSO unit.

By storing, during L cycles, each decision and offset value generated by RU_{A1}, the complexity of RU_{A2} is almost divided by two (see Fig. 9).

This method is very similar to the method used for the soft-output Viterbi algorithm [29].

Note that once more, an $(n_A = 2, n_B = 2, M_{(A+B)/2})$ method can be used.

E. $(n_A = 2, n_B = 2, M_A, Pt_B)$ Architecture

This type of architecture is a generalization of the idea described above: instead of memorizing a large number of A (or B) vectors, they are recomputed when they are needed. For this, the context (i.e., the state metrics) of an iteration process is saved in a pointer. This pointer is used later to recompute, with a delay, the series state metric. Such a process is given in Fig. 10.

In this scheme, the state metrics of RU_{B2} are saved every $L/4$ cycles (small circles in Fig. 10). Those four state metrics are used as a seed, or pointer, to start the third backward process (RU_{B3}, in Fig. 10) of length $L/4$. The third backward recursion is synchronized with the forward recursion in order to minimize the size of the B vector to be stored. In practice, only three seeds are needed, since RU_{B2} and RU_{B3} process the same data during the last quarter of a segment of L cycles. With this method, the latency is still $4L$, but the number of state metrics to store is now $3 + L/4$. With such a small number of vectors, the use of registers instead of RAM can be used to store

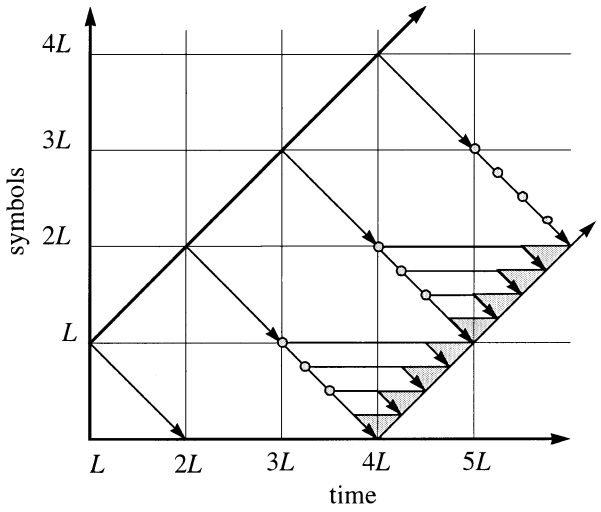

Fig. 10. Graphical representation of the ($n_A = 1, n_B = 3, M_B, Pt_B$) architecture.

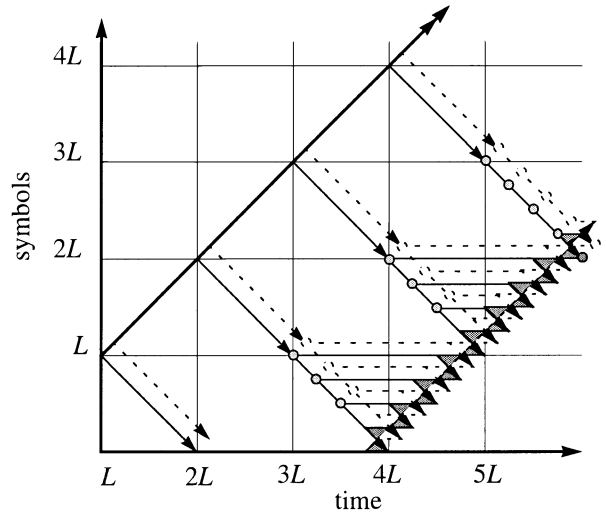

Fig. 11. Graphical representation of the ($n_A = 1, n_B = 3, M_{(A+B)/2}, Pt_B$) architecture.

the state metrics. This avoids the use of a RAM with an unusual aspect ratio and a consequent negative impact on performance. This scheme becomes particularly attractive if two independent MAP algorithms are implemented in a single chip, since an ($n_A = 2, n_B = 2, M_{(A+B)/2}, Pt_B$) architecture can be used to share the $6+L/4$ vectors of the two MAP algorithms (see Fig. 11). As with the ($n_A = 1, n_B = 2, M_{(A+B)/2}$) architecture, this scheme can be used in a pipeline of decoders to simultaneously decode two information streams.

This scheme is particularly efficient because it avoids the use of a RAM for storing the state metrics.

F. Generalization of the Architecture

Many combinations of the above architectures can be realized, each one with its own advantages and disadvantages. In the above examples, the ratio p between the hardware clock and the symbol clock is one. Other architectures can be uncovered by loosening this restriction. For example, if this ratio p is two (i.e., two clock cycles for each received symbol), the speed of the RU is doubled. Thus, an architecture such as ($p = 2, n_A = 1, n_B = 1, M_A, Pt_{1A}$) can be used (see Fig. 12) to obtain an SVM of size $L/2$.

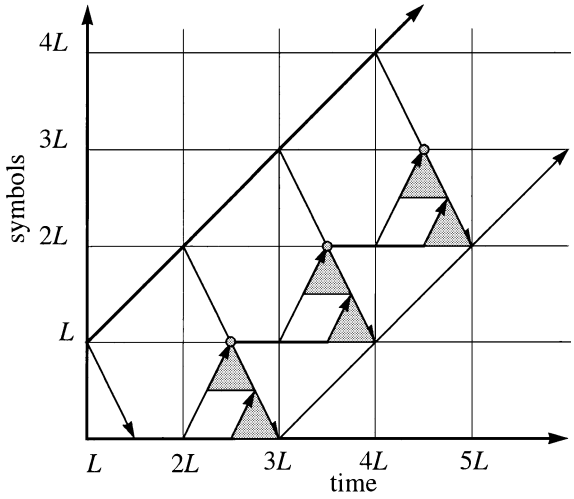

Fig. 12. Graphical representation of the ($p = 2, n_A = 1, n_B = 1, M_A, Pt_{1A}$) architecture.

G. Summary of the Different Configurations

In Table II, different configurations are evaluated in order to help the designer of a system. Note that n is a generalization factor and that 0.5 (in columns n_A and n_B) denotes the simplified ACSO unit of Fig. 9. We can see that in the case of two MAP algorithms implemented together in the same circuit, it is possible to decrease the number of vectors from L to $L/8+3$. This reduction allows the realization of this memory using only registers.

Note that the final choice of a solution among the different proposed alternatives will be made by the designer. The designer's objective is to optimize area and/or power dissipation of the design while respecting application requirements (decoding latency, performance). The complexity of the MAP algorithm depends on the application (continuous stream or small blocks, simple or duo-binary encoder [30], [31], number of encoder states, etc.). The consequence is that the merit of the proposed solution can vary with the application and no general rules can be found. In practice, a fast and quite accurate complexity estimation can be obtained in terms of gate count and memory cells by simply using a field-programmable gate array synthesis tool to compile a VHDL or Verilog algorithm description.

H. Similar Works in This Area

Since the first submission of this paper, much work has been independently published on this topic. In this final subsection, we give a brief overview of these fundamental works.

The architecture of Sections V-B–D has also been proposed by Schurgers *et al.* In [32] and [33], the authors give a very detailed analysis of the tradeoffs between complexity, power dissipation, and throughput. Moreover, they propose a very interesting architecture of double flow structures, where for example, two processes of type ($n_A = 1, n_B = 2, M_A$) and ($n_B = 1, n_A = 2, M_A$) are performed in parallel on a data block of size N, the first one, in natural order, from data 0

TABLE II
PERFORMANCE OF THE DIFFERENT ARCHITECTURES

n_A	n_B	Memory organization	Recursion Units	Memory	Latency	Notes
1	2	M_A or M_B	3	L	$4L$	Figure 5
1	2	$M_{(A+B)/2}$ [1]	3	$L/2$	$4L$	Figure 6
1.5	2	M_B	3.5	$L/2$	$4L$	Figure 8
1	2.5	M_B, Pt_B	3.5	$L/4+4$	$4L$	Figure 10
1	2.5	M_A, Pt_B	3.5	$L/p+p-1$	$4L$	p = number of pointers
1	2.5	$M_{(A+B)/2}$, Pt_B [1]	3.5	$L/8+3$	$4L$	Figure 11
1	3	M_B	4	$L/2$	$3L$	Figure 7
$1+0.5n$	2	M_A or M_B	$3+0.5n$	L/n	$4L$	n = number of half RUB
1	$2+n$	M_A or M_B	$3+n$	$L/(n+1)$	$2L+L/(n+1)$	n = number of RUB

to $N/2$, the second, in reverse order, from data N down to $N/2$. Moreover, Worm *et al.* [34] extend the architecture of Sections V-A and -B for a massively parallel architecture where several processes are done in parallel. With this massive parallelism, very high throughput (up to 4 Gbit/s) can be achieved.

The pointer idea described in Section V-E has been proposed independently by Dingninou *et al.* in the case of a turbo decoder in [35] and [36]. In this "sliding window next iteration initialization" method, the pointer generated by the backward recursion at iteration k is used to initialize the backward recursion at iteration $k+1$. As a result, no further backward convergence process is needed and area and memory are saved at the cost of a slight degradation of the decoder performance. Note that Dielissen *et al.* have improved this method by an efficient encoding of the pointer [37].

Finally, an example of an architecture using a ratio of two between clock frequency and symbol frequency (see Section V-F) is partially used in [38].

VI. CONCLUSION

We have presented a survey of techniques for VLSI implementation of the MAP algorithm. As a general conclusion, the well-known results from the Viterbi algorithm literature can be applied to the MAP algorithm. The computational kernel of the MAP algorithm is very similar to that of the ACS of the Viterbi algorithm with an added offset. The analysis shows that it is better to add the offset first and then do the ACS operation in order to reduce the critical path of the circuit (OACS). A general architecture for the MAP algorithm was developed which exposes some interesting tradeoffs for VLSI implementation. Most importantly, we have presented architectures which eliminate the need for RAMs with a narrow aspect ratio and possibly allow the RAM to be replaced with registers. An architecture which shares a memory bank between two MAP decoders enables efficient implementation of turbo decoders.

ACKNOWLEDGMENT

The authors would like to thank F. Kschischang and O. Pourquier for their help on the Perron–Frobenius theorem.

REFERENCES

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo codes," in *Proc. IEEE Int. Conf. Communications (ICC'93)*, May 1993, pp. 1064–1070.

[2] R. W. Chang and J. C. Hancock, "On receiver structures for channels having memory," *IEEE Trans. Inform. Theory*, vol. IT-12, pp. 463–468, Oct. 1966.

[3] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 284–287, Mar. 1974.

[4] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inform. Theory*, vol. IT-13, pp. 260–269, Apr. 1967.

[5] G. D. Forney, Jr., "The Viterbi algorithm," *Proc. IEEE*, vol. 61, pp. 268–278, Mar. 1973.

[6] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 429–445, Mar. 1996.

[7] A. J. Viterbi, "An intuitive justification and a simplified implementation of the MAP decoder for convolutional codes," *IEEE J. Select. Areas Commun.*, vol. 16, pp. 260–264, Feb. 1998.

[8] N. G. Kingsbury and P. J. W. Rayner, "Digital filtering using logarithmic arithmetic," *Electron. Lett.*, vol. 7, no. 2, pp. 56–58, Jan. 1971.

[9] J. A. Erfanian and S. Pasupathy, "Low-complexity parallel-structure symbol-by-symbol detection for ISI channels," in *Proc. IEEE Pacific Rim Conf. Communications, Computers and Signal Processing*, June 1–2, 1989, pp. 350–353.

[10] H. Dawid, *Algorithms and VLSI Architecture for Soft Output Maximum a Posteriori Convolutional Decoding* (in German). Aachen, Germany: Shaker, 1996, p. 72.

[11] H. Dawid and H. Meyr, "Real-time algorithms and VLSI architectures for soft output MAP convolutional decoding," in *Proc. Personal, Indoor and Mobile Radio Communications, PIMRC'95*, vol. 1, 1995, pp. 193–197.

[12] S. S. Pietrobon, "Efficient implementation of continuous MAP decoders and a new synchronization technique for turbo decoders," in *Proc. Int. Symp. Information Theory and Its Applications*, Victoria, BC, Canada, Sept. 1996, pp. 586–589.

[13] S. S. Pietrobon and S. A. Barbulescu, "A simplification of the modified Bahl algorithm for systematic convolutional codes," in *Proc. Int. Symp. Information Theory and Its Applications*, Sydney, Australia, Nov. 1994, pp. 1073–1077.

[14] S. S. Pietrobon, "Implementation and performance of a turbo/MAP decoder," *Int. J. Satellite Commun.*, vol. 16, pp. 23–46, Jan.-Feb. 1998.

[15] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," in *Proc. IEEE Int. Conf. Communications (ICC '95)*, 1995, pp. 1009–1013.

[16] C. B. Shung, P. H. Siegel, G. Ungerboeck, and H. K. Thapar, "VLSI architectures for metric normalization in the Viterbi algorithm," in *Proc. IEEE Int. Conf. Communications (ICC '90)*, vol. 4, Atlanta, GA, Apr. 16–19, 1990, pp. 1723–1728.

[17] P. Tortelier and D. Duponteil, "Dynamique des métriques dans l'algorithme de Viterbi," *Annales des Télécommun.*, vol. 45, no. 7-8, pp. 377–383, 1990.

[18] G. Masera, G. Piccinini, M. R. Roch, and M. Zamboni, "VLSI architectures for turbo codes," *IEEE Trans. VLSI Syst.*, vol. 7, pp. 369–379, Sept. 1999.

[19] A. Worm, H. Michel, F. Gilbert, G. Kreiselmaier, M. Thul, and N. Wehn, "Advanced implementation issues of turbo decoders," in *Proc. 2nd Int. Symp. on Turbo Codes*, Brest, France, Sept. 2000, pp. 351–354.

[20] G. Montorsi and S. Benedetto, "Design of fixed-point iterative decoders for concatenated codes with interleavers," *IEEE J. Select. Areas Commun.*, vol. 19, pp. 871–882, May 2001.

[21] A. P. Hekstra, "An alternative to metric rescaling in Viterbi decoders," *IEEE Trans. Commun.*, vol. 37, pp. 1220–1222, Nov. 1989.
[22] P. H. Siegel, C. B. Shung, T. D. Howell, and H. K. Thapar, "Exact bounds for Viterbi detector path metric differences," in *Proc. Int. Conf. Acoustics, Speech, and Signal Processing*, vol. 2, 1991, pp. 1093–1096.
[23] G. Fettweis and H. Meyr, "Parallel Viterbi algorithm implementation: Breaking the ACS bottleneck," *IEEE Trans. Commun.*, vol. 37, pp. 785–790, Aug. 1989.
[24] H. Dawid, G. Gehnen, and H. Meyr, "Map channel decoding: Algorithm and VLSI architecture," *VLSI Signal Processing VI*, pp. 141–149, 1993.
[25] F. R. Gantmacher, *Matrix Theory*. New York: Chelsea, 1960, vol. II.
[26] *20 Mbps convolutional encoder Viterbi decoder STEL-2020*: Stanford Telecom, 1989.
[27] E. Boutillon and N. Demassieux, "A generalized precompiling scheme for surviving path memory management in Viterbi decoders," in *Proc. ISCAS'93*, vol. 3, New Orleans, LA, May 1993, pp. 1579–1582.
[28] E. Boutillon, "Architecture et implantation VLSI de techniques de modulations codées performantes adaptées au canal de Rayleigh," Ph.D. dissertation, ENST, Paris, France, 1995.
[29] J. Hagenauer and P. Hoeher, "A Viterbi algorithm with soft-decision outputs and its applications," in *Proc. IEEE Globecom Conf.*, Nov. 1989, pp. 1680–1686.
[30] C. Douillard, M. Jézéquel, C. Berrou, N. Bengarth, J. Tousch, and N. Pham, "The turbo code standard for DVB-RCS," in *Proc. 2nd Int. Symp. on Turbo Codes*, Brest, France, Sept. 2000, pp. 535–538.
[31] C. Berrou and M. Jézéquel, "Nonbinary convolutional codes for turbo coding," *Electron. Lett.*, vol. 35, no. 1, pp. 39–40, Jan. 1999.
[32] C. Schurgers, F. Catthoor, and M. Engels, "Energy efficient data transfer and storage organization for a MAP turbo decoder module," in *Proc. 1999 Int. Symp. Low Power Electronics and Design*, San Diego, CA, Aug. 1999, pp. 76–81.
[33] ——, "Memory optimization of MAP turbo decoder algorithms," *IEEE Trans. VLSI Syst.*, vol. 9, pp. 305–312, Apr. 2001.
[34] A. Worm, H. Lamm, and N. Wehn, "VLSI architectures for high-speed MAP decoders," in *Proc. 14th Int. Conf. VLSI Design*, 2001, pp. 446–453.
[35] A. Dingninou, "Implémentation de turbo code pour trame courtes," Ph.D. dissertation, Univ. de Bretagne Occidentale, Bretagne, France, 2001.
[36] A. Dingninou, F. Rafaoui, and C. Berrou, "Organization de la mémoire dans un turbo décodeur utilisant l'algorithme SUB-MAP," in *Proc. Gretsi*, Gretsi, France, Sept. 1999, pp. 71–74.
[37] J. Dielissen and J. Huisken, "State vector reduction for initialization of sliding windows MAP," in *Proc. 2nd Int. Symp. Turbo Codes*, Brest, France, Sept. 2000, pp. 387–390.
[38] A. Raghupathy and K. J. R. Liu, "VLSI implementation considerations for turbo decoding using a low-latency log-MAP," in *Proc. IEEE Int. Conf. Consumer Electronics, ICCE*, June 1999, pp. 182–183.

Emmanuel Boutillon received the engineering degree in 1990 and the Ph.D. degree in 1995, both from the Ecole Nationale Supérieure des Télécommunications (ENST), Paris, France.

He joined ENST in 1992, where he conducted research in the field of VLSI for communications. In 1998, he spent a sabbatical year at the University of Toronto, Toronto, ON, Canada, where he worked on algorithms and architectures for MAP and LDPC decoding. Since 2000, he is a Professor at the University of South Britany, Lorient, France. His current research interests are on the interactions between algorithms and architectures in the field of wireless communication.

Warren J. Gross (S'92) was born in Montreal, QC, Canada, in 1972. He received the B.A.Sc. degree in electrical engineering from the University of Waterloo, Waterloo, ON, Canada, in 1996 and the M.A.Sc. degree in 1999 from the University of Toronto, Toronto, ON, Canada, where he is currently working toward the Ph.D. degree.

From 1993 to 1996, he worked in the area of space-based machine vision at Neptec Design Group, Ottawa, ON, Canada. His research interests are in the areas of VLSI architectures for digital communications algorithms and digital signal processing, coding theory, and computer architecture.

Mr. Gross received the Natural Sciences and Engineering Research Council of Canada postgraduate scholarship, the Walter Sumner fellowship and the Government of Ontario/Ricoh Canada Graduate Scholarship in Science and Technology.

P. Glenn Gulak (S'82–M'83–SM'96) received the Ph.D. degree from the University of Manitoba, Winnipeg, MB, Canada.

From 1985 to 1988, he was a Research Associate with the Information Systems Laboratory and the Computer Systems Laboratory, Stanford University, Stanford, CA. Currently, he is a Professor with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada, and holds the L. Lau Chair in Electrical and Computer Engineering. His research interests are in the areas of memory design, circuits, algorithms, and VLSI architectures for digital communications.

Dr. Gulak received a Natural Sciences and Engineering Research Council of Canada Postgraduate Scholarship and several teaching awards for undergraduate courses taught in both the Department of Computer Science and the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada. He served as the Technical Program Chair for ISSCC 2001. He is a registered professional engineer in the province of Ontario.