A High-Throughput Maximum *a Posteriori* Probability Detector

Ruwan Ratnayake, *Member, IEEE*, Aleksandar Kavčić, and Gu-Yeon Wei

***Abstract*—This paper presents a maximum *a posteriori* probability (MAP) detector, based on a forward-only algorithm that can achieve high throughputs. The MAP algorithm is optimal in terms of bit error rate (BER) performance and, with Turbo processing, can approach performance close to the channel capacity limit. The implementation benefits from optimizations performed at both algorithm and circuit level. The proposed detector utilizes a deep-pipelined architecture implemented in skew-tolerant domino and experimentally measured results verify the detector can achieve throughputs greater than 750 Mb/s while consuming 2.4 W. The 16-state EEPR4 channel detector is implemented in a 0.13 μm CMOS technology and has a core area of 7.1 mm^2.**

***Index Terms*—Architectures, BCJR algorithm, forward-only algorithm, iterative processing, maximum *a posteriori* probability algorithm, turbo, VLSI.**

I. Introduction

HIGH-SPEED detectors that combat inter-symbol interference (ISI) are of interest for a variety of communications applications such as magnetic recording systems. Typically, high-speed detectors use less computationally-intensive algorithms such as the Viterbi algorithm [1]–[4]. Unfortunately, such algorithms generate hard outputs, which constrain the receiver to non-iterative processing. On the other hand, iterative processing utilizing MAP detection is likely to be used in the next generation of systems due to the bit error rate (BER) performance benefits. This paper describes the implementation details and experimental results of a 750 Mb/s MAP detector that employs a modified near-optimal MAP algorithm.

In magnetic storage systems, areal density in the magnetic medium has increased exponentially within the last few decades and this trend is predicted to continue. The increase in density adversely results in increased ISI and reduced signal-to-noise ratio (SNR). Sophisticated signal processing methods can be utilized to extract the data in such deteriorated signal environments. One method that has shown remarkable performance is iterative or turbo detection [5]–[8]. A block diagram of turbo processing is shown in Fig. 1. Iterative detection requires soft outputs, which carry more information compared to hard outputs. Algorithms such as the soft-output Viterbi algorithm (SOVA) have been considered for iterative detection [9], [10], but SOVA is a suboptimal algorithm in terms of BER performance [11]. In contrast, MAP detection based algorithms offer optimal BER performance. A MAP detector concatenated with a low-density parity-check (LDPC) code decoder is seen as a strong candidate for future high-density magnetic storage systems. Due to high computational complexity, there has been a dearth of VLSI implementations for MAP detectors that target high-speed applications.

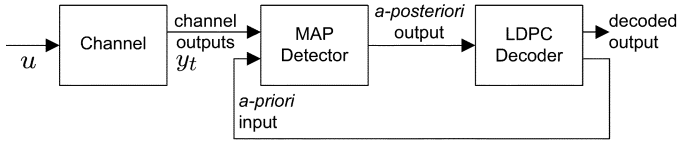


Fig. 1. Block diagram of iterative detection.

This paper presents the design, implementation, and experimental verification of a MAP detector that can perform at very high throughputs. The detector targets a 16-state extended enhanced partial response class-4 (EEPR4) channel that models high areal densities. The implementation benefits from optimizations performed at several levels of system design. First, we chose to implement a forward-only algorithm that has several advantages over the traditional MAP algorithm. One key feature of this forward-only algorithm is its inherent pipelined structure, which we exploit to increase throughput. Second, we leverage techniques to increase throughput at the circuit level. The design uses three-phase, skew-tolerant, dual-rail domino logic. In addition to reducing gate delay, skew-tolerant domino obviates dedicated latches required for traditional pipelining schemes. Removing these latches significantly reduces hardware overhead, eliminates latch delay, but retains time borrowing via overlapped clocks. In order to further increase throughput, computations that impose worst-case bottlenecks, constraining throughput, are addressed at the algorithmic level. We propose a modification to the algorithm that minimally affects BER performance, verified by system-level BER simulations, but speeds up the critical path.

The well-known algorithm by Bahl, Cocke, Jelinek and Raviv (BCJR), which is traditionally considered for most MAP detection/decoding applications, requires forward and backward computations [12]. This is in contrast to the Viterbi or SOVA algorithms, which allow computations to be performed only in the forward direction. Once the input stream is fed into a Viterbi or SOVA detector, the outputs are generated after a fixed latency and retain the same order. On the other hand, the *a posteriori* probability outputs (APPs) of the BCJR algorithm can only be evaluated after both forward and backward metrics have been

Manuscript received December 15, 2007; revised February 23, 2008. Published July 23, 2008 (projected). This work was supported in part by Agere Systems.

R. Ratnayake and G.-Y. Wei are with the School of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138 USA (e-mail: ratnayak@fas.harvard.edu; guyeon@eecs.harvard.edu).

A. Kavčić is with the Department of Electrical Engineering, University of Hawaii, Honolulu, HI 96822 USA (e-mail: kavcic@hawaii.edu).

Digital Object Identifier 10.1109/JSSC.2008.925404

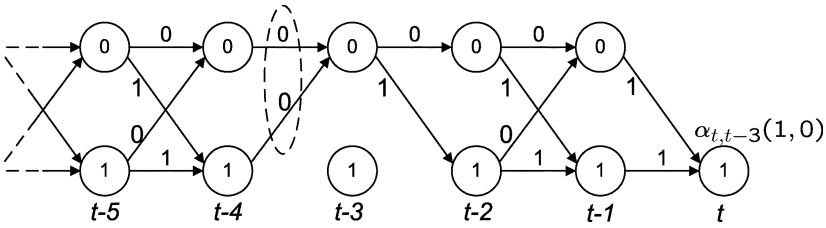


Fig. 2. The trellis paths that contribute to the soft survivor for state 1 at time t, where each path has a branch with input 0 at time $t-3$. The soft survivor is denoted by $\alpha_{t,t-3}(1,0)$. The input corresponding to each branch is shown adjacent to the branch.

TABLE I
COMPARISON OF ALGORITHMS

Algorithm	FOMAP	BCJR	Viterbi
Updates	parallel	sequential	parallel
Latency	fixed (L)	variable ($2L$ to $4L$)	fixed (L)
Order of outputs	ordered $[1, 2, ..., L, L+1, ...]$	permuted $[L, L-1, ..., 1, 2L, ...]$	ordered $[1, 2, ..., L, L+1, ...]$
Control	simple	complicated	simple
Buffering chan. outputs	not required	required	not required

L: Length of the window.

computed. Consequently, the outgoing symbols appear in a permuted order relative to the incoming symbols.

To overcome the complexities inherent to BCJR, we explore a recently developed algorithm that performs MAP with computations only in the forward direction [13]. We call this algorithm forward-only MAP (FOMAP). The FOMAP algorithm has similarities to both Viterbi and BCJR algorithms. The FOMAP algorithm keeps soft survivors (probabilities), which are saved in a fixed-length sliding-window survivor memory. A prominent feature of FOMAP is its ability to update all of the (soft) survivors in parallel, similar to the Viterbi structure, where the (hard) survivors are also updated in parallel. Moreover, the FOMAP algorithm is a sum-of-products algorithm that, with soft survivors, generates APPs. This is in contrast to SOVA, which only computes an approximation of the APPs.

One of the key drawbacks of traditional BCJR is its backward computations, which only allows for sequential state metric updates. After receiving a symbol the BCJR algorithm takes up to four times the latency—pertaining to the window length—to compute corresponding APPs. In contrast, the FOMAP algorithm can perform parallel updates to generate APPs after a fixed latency equal to the latency of a single window length and generate ordered outputs. Latency and ordering of outputs are similar to the Viterbi algorithm. By retaining key attractive features from both Viterbi and MAP algorithms, namely parallel survivor updating and the ability to compute APPs, the FOMAP algorithm can be implemented as a deep pipelined structure to offer superior performance in terms of BER, throughput, and latency. Comparisons of FOMAP to the BCJR and Viterbi algorithms are summarized in Table I.

This paper is organized into several sections as follows. First, a brief overview of the FOMAP algorithm is provided. Then, Section III describes three schemes that improve the throughput of the MAP detector at the algorithm level. Afterwards, Section IV introduces the proposed detector architecture and describes circuit-level design mechanisms employed to increase the throughput. Section V presents measurement results of a test-chip prototype that verifies high-throughput performance. Finally, Section VI concludes the paper.

II. FORWARD ONLY MAP ALGORITHM

FOMAP is a path-partitioning algorithm that computes APPs by processing probabilities of paths. For brevity, we only present a basic introduction to FOMAP with the aim of giving the reader necessary information to understand the proposed architecture. We defer the reader to [13] for detailed discussions of the FOMAP algorithm. A soft survivor in the FOMAP algorithm, denoted by $\alpha_{t,i}(s,u)$, is defined to be the sum of the APPs of all paths that terminate at state s at time t that include a branch at time i with input u. In essence, it is the joint probability of S_t and U_i conditioned upon the received vector Y_1^t, i.e., $\alpha_{t,i}(s,u) = \Pr(S_t = s, U_i = u | Y_1^t)$, where S_t and U_i are random variables denoting state at t and input symbol at i, respectively. Y_1^t is the sequence of received symbols up to time t i.e., $Y_1^t = \{y_1, \ldots, y_t\}$. Fig. 2 clarifies this concept with a trellis diagram. The trellis shown consists of two states (0 and 1) and binary inputs. The figure shows all paths that contribute to $\alpha_{t,t-3}(1,0)$—the soft survivor for state 1 at time t, where each path contains a branch at time $t-3$ that corresponds to the input symbol 0.

The soft survivors of FOMAP are computed recursively. We consider a sliding window version of the algorithm for implementation. The FOMAP algorithm computes the soft survivors pertaining to the latest input symbol, based on the latest received symbol y_t and the soft survivors from the previous iteration. We call this operation *extend*, which implies extending the sliding window by one step. This step is similar to the computation of state metrics in the forward recursion of the BCJR algorithm. Simultaneously, all the soft survivors pertaining to previous input symbols are updated in parallel. We call this step *update*. This step is similar to the update operation of hard survivors in the Viterbi algorithm. Finally, by combining (collecting) all of the appropriate soft survivors at the back of the window, the FOMAP algorithm computes APPs for each input symbol. This step is called *collect*. To explain this in graphical form, a data flow diagram of the three operations is shown in Fig. 3. The horizontal axis going from left to right depicts the trellis index. The vertical axis corresponds to the clock cycles where one trellis step is processed per cycle. The window consists of an array of registers that contains all of the soft survivors

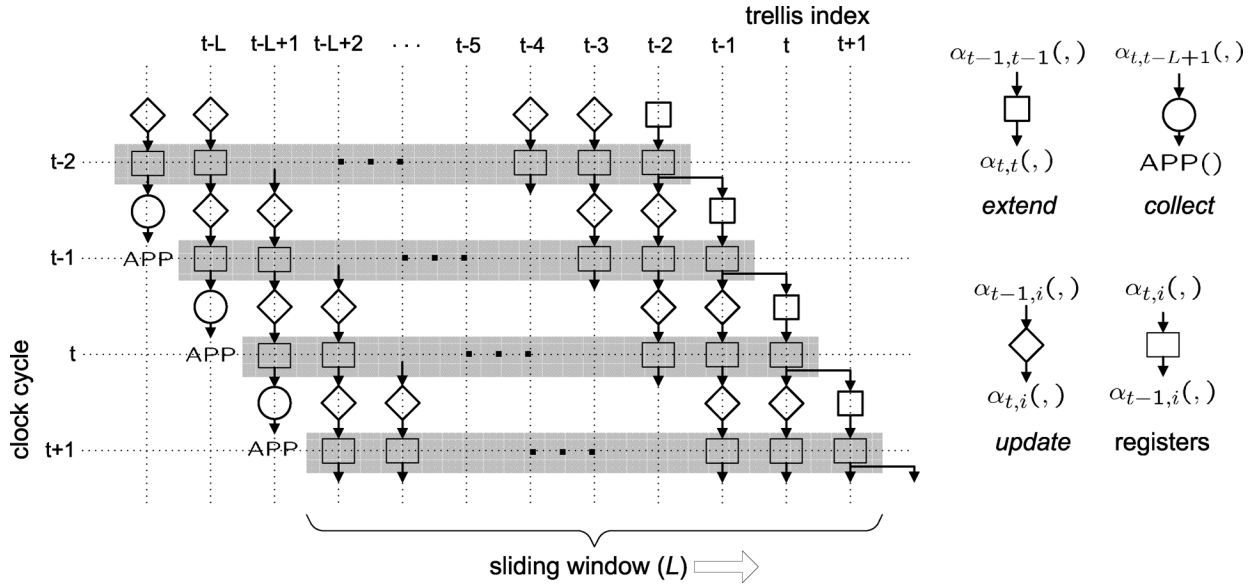


Fig. 3. Data flow of sliding window FOMAP algorithm.

at a given time instance. As the figure shows, at each time instance the window extends and slides to the right by one step. At the same time, all contents of the window are updated by incorporating the latest received symbol. The APPs are driven out the back of the window with a latency of L cycles. Assuming all soft survivors within the window for previous iteration are available, i.e., $\alpha_{t-1,i}(s,u)$ for $t-L \le i \le t-1$, the three steps of FOMAP are evaluated as follows:

A. Extend

The *extend* operation, first computes state metrics for a new time instance based on soft survivors from the previous iteration:

$$\begin{aligned}\hat{\alpha}_{t-1}(s) &= \Pr(S_{t-1}=s|Y_1^{t-1}) \\ &= \sum_{u\in\mathcal{U}} \Pr(S_{t-1}=s, U_{t-1}=u|Y_1^{t-1}) \\ &= \sum_{u\in\mathcal{U}} \alpha_{t-1,t-1}(s,u). \end{aligned} \tag{1}$$

Then it evaluates the soft survivors for the latest input symbol based on these state metrics and the latest received symbol

$$\begin{aligned}&\alpha_{t,t}(s,u) \\ &= \Pr(S_t=s, U_t=u|Y_1^t) \\ &= \rho_1 \sum_{s'\in\mathcal{S},\, l(s',s)=u} \Pr(S_{t-1}=s|Y_1^{t-1}) \Pr(S_t=s, y_t|S_{t-1}=s') \\ &= \rho_1 \sum_{s'\in\mathcal{S},\, l(s',s)=u} \hat{\alpha}_{t-1}(s')\gamma_t(s',s). \end{aligned} \tag{2}$$

Here, $\mathcal{S}$ and $\mathcal{U}$ are sets of all possible states and inputs, respectively, and s and u are elements of these two sets. ρ_1 is a scaling factor that normalize the soft survivor across s and u. $\gamma_t(s',s) = \Pr(S_t=s, y_t|S_{t-1}=s')$ is the branch metric (based on the latest received symbol y_t) connecting states s' and s at time t; e.g., if the system noise is Gaussian, then $\gamma_t(s',s)$ is simply a Gaussian probability density function conditioned on the state transition (s',s). $l(s',s)$ is a function that indicates the input symbol u corresponding to a branch connecting state s' and s; e.g., in the system defined by the trellis shown in Fig. 2, $l(1,0)=0$ and $l(0,1)=1$. Since $\alpha_{t,t}$ depends on $\alpha_{t-1,t-1}$, this operation contains a feedback loop, which can limit throughput. Section III shows how the extend step can be simplified to alleviate this throughput bottleneck.

B. Update

The *update* operation updates the soft survivors for the remaining length of the window, i.e., $t-L+1 \le i \le t-1$.

$$\begin{aligned}&\alpha_{t,i}(s,u) \\ &= \Pr(S_t=s, U_i=u|Y_1^t) \\ &= \rho_2 \sum_{s'\in\mathcal{S}} \Pr(S_{t-1}=s', U_i=u|Y_1^{t-1}) \Pr(S_t=s, y_t|S_{t-1}=s') \\ &= \rho_2 \sum_{s'\in\mathcal{S}} \alpha_{t-1,i}(s',u)\gamma_t(s',s) \end{aligned} \tag{3}$$

where ρ_2 is a scaling factor. All of the soft survivors are updated in parallel across the length of the window. Thus, within one cycle the latest received symbol is incorporated into all of the soft survivors. The data flow for this operation occurs in a feed-forward manner from the front of the window towards the back. Hence, this operation is amenable to pipelining.

C. Collect

Since soft survivors are joint APPs corresponding to states s and inputs u, summing up (or collecting) all of the survivors at the back of the window for a given input across the states gives the APP corresponding to the input.

$$\Pr(U_{t-L+1}=u|Y_1^t) = \mathrm{APP}(u) = \sum_{s\in\mathcal{S}} \alpha_{t,t-L+1}(s,u). \tag{4}$$

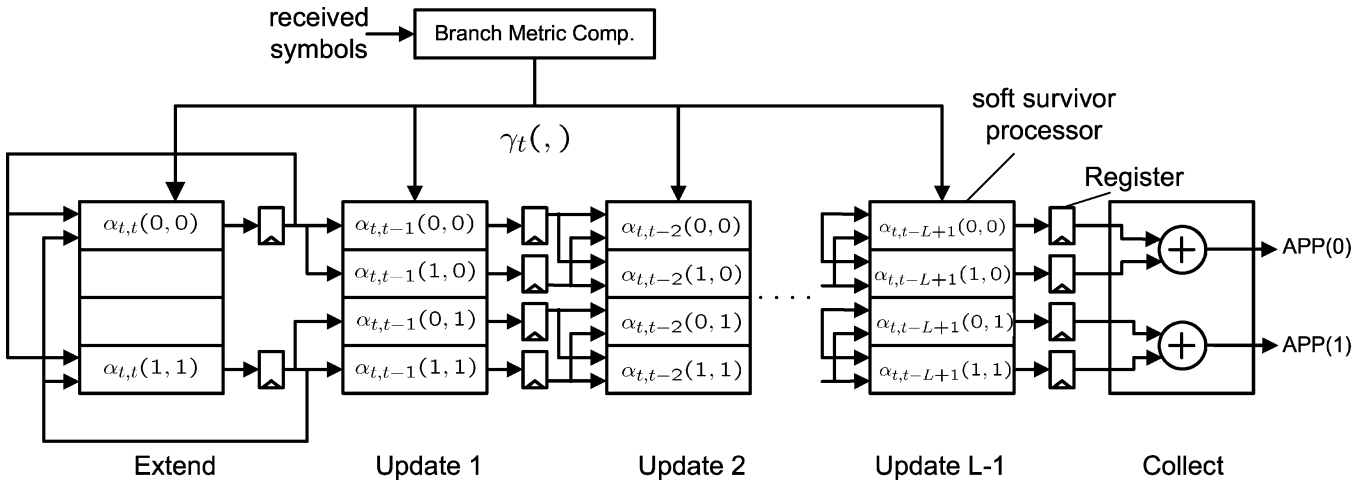


Fig. 4. Block diagram of FOMAP architecture for a two-state system defined by trellis in Fig. 2. Soft survivors at, $\alpha_{t,i}(s,u)$ for each index $i, t-L+1 \le i \le t$ are shown.

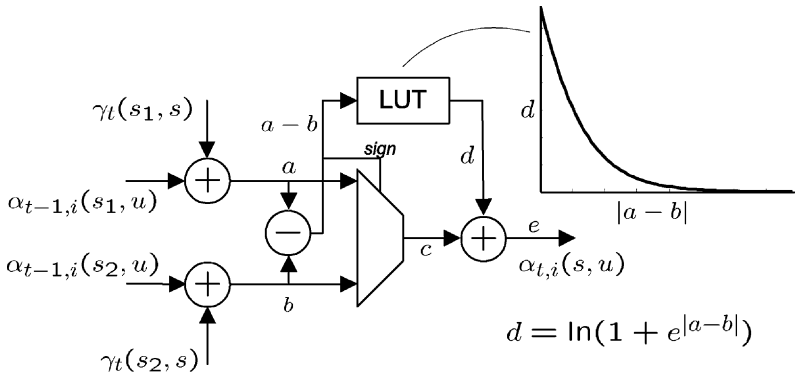


Fig. 5. FOMAP ACSLA soft survivor computation unit.

A block diagram of FOMAP for the simple trellis system with two states, considered in Fig. 2, is shown in Fig. 4. Each soft survivor α is computed by a dedicated processing unit. Each *update* column contains a number of processing units equal to the number of states times the number of inputs. For this example, there are four soft survivors per trellis index and, hence, four processing units per column. Certain soft survivors have zero probability and can be ignored, e.g., $\alpha_{t,t}(0,1)$ and $\alpha_{t,t}(1,0)$.

As shown by (1), (2), and (3), *extend* and *update* operations are essentially sum-product operations. Since computing products is expensive in terms of required computational power and complexity, the algorithm is implemented in the log domain. Thus, a product is mapped to an addition. A summation is mapped to a special operation which is denoted by $\boxplus$. This special addition $\boxplus$ is defined as $a \boxplus b = \ln(e^a + e^b) = \max(a,b) + \ln(1+e^{-|a-b|})$. The correction term, $\ln(1+e^{-|a-b|})$, can be implemented as a look-up table (LUT) to simplify implementation. The bottleneck of a FOMAP algorithm is this sum-product computation in *extend* and *update*, which is implemented by a concatenation of arithmetic functions: addition, comparison, selection, LUT, and another addition. We call this unit illustrated in Fig. 5, the add/compare/select/LUT/add (ACSLA) unit.

III. Increasing Throughput

Applications such as magnetic storage systems require very high throughputs. In this section, we investigate three schemes that can increase the maximum achievable throughput of the FOMAP detector. First, we can speed up the ACSLA unit by reordering the operations. Second, we can consider a higher order radix system by processing multiple received symbols within a single clock cycle, which results in an increase in the delay of the critical path, but also increases throughput. Finally, we propose a slight modification to the algorithm that only minimally affects BER, but significantly reduces delay for the throughput bottleneck. A comparison of these techniques show that the third method yields the best tradeoffs between throughput and hardware cost. We implement the detector based on this method.

A. Compare Select Add LUT Add

Lee *et al.* show that the critical path delay in the Viterbi algorithm can be decreased by reorganizing the add/compare/select (ACS) unit to a compare/select/add (CSA) unit [4]. Similarly, the ACSLA unit for a FOMAP algorithm can be reorganized such that addition and comparison are performed in parallel. We

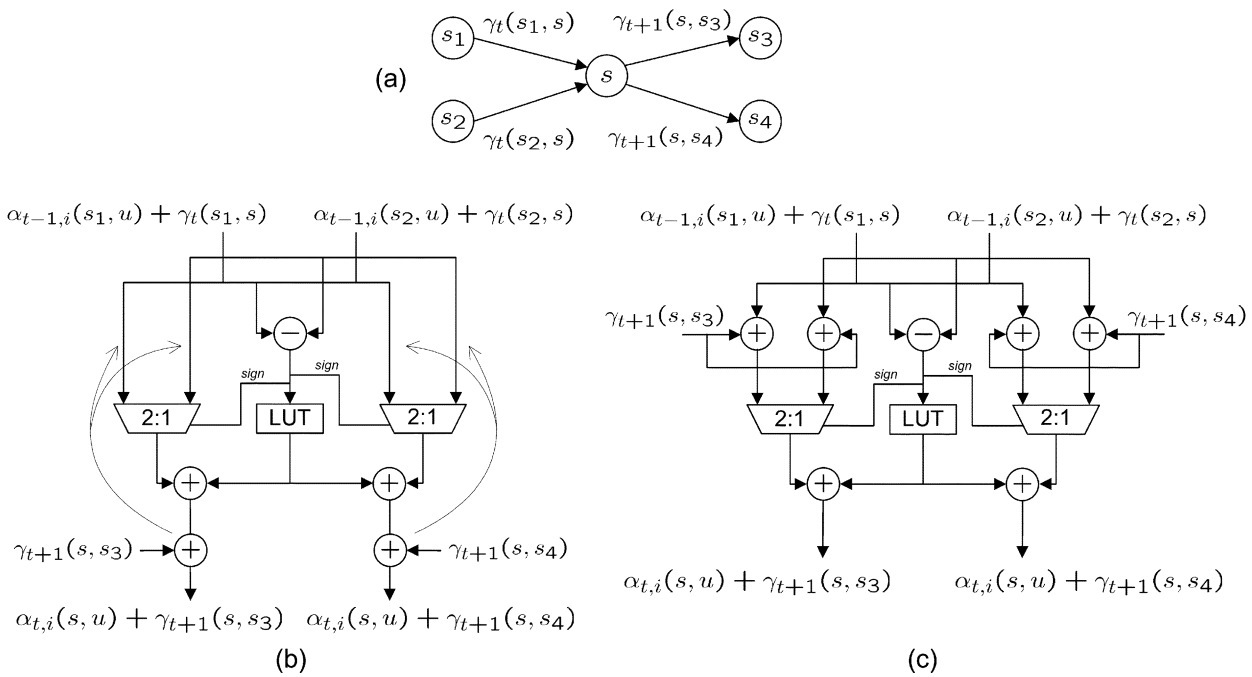


Fig. 6. (a) State transitions. (b) Compare/select/LUT/add/add computation unit. (c) Reorganized compare/select/add/LUT/add (CSALA) computation unit.

explain this in terms of state transitions shown in Fig. 6(a). The soft survivors in log domain are given by ACSLA operation:

$$\alpha_{t,i}(s,u) = \{\alpha_{t-1,i}(s_1,u)+\gamma_t(s_1,s)\} \boxplus \{\alpha_{t-1,i}(s_2,u)+\gamma_t(s_2,s)\}. \quad (5)$$

Adding $\gamma_{t+1}(s,s'')$, $s'' \in \{s_3,s_4\}$ to both sides of the equality yields

$$\{\alpha_{t,i}(s,u)+\gamma_{t+1}(s,s'')\} = \{\alpha_{t-1,i}(s_1,u)+\gamma_t(s_1,s)\} \boxplus \{\alpha_{t-1,i}(s_2,u)+\gamma_t(s_2,s)\}+\gamma_{t+1}(s,s''). \quad (6)$$

Thus, if soft survivors $\{\alpha_{t-1,i}(s',u)+\gamma_t(s',s)\}$ — taken as single entities — for $s' \in \{s_1,s_2\}$ previous states are already known, then soft survivors $\{\alpha_{t,i}(s,u)+\gamma_{t+1}(s,s'')\}$ for $s'' \in \{s_3,s_4\}$ next states can be obtained recursively. This is illustrated in Fig. 6(b). This does not reduce the delay for the sum-product operation but simply reorganize the add/compare/select/LUT/add into compare/select/LUT/add/add operation. However, the final addition in (6) can be performed in parallel with comparison as shown in Fig. 6(c). This is referred to as a compare/select/add/LUT/add (CSALA) unit. If both *extend* and *update* operations are based on CSALA, then the delay in the critical path of the system is reduced by the delay of one addition operation.

B. Radix4-Sum2

Another approach to increase throughput is to concatenate and process multiple trellis steps within a single clock cycle. We consider a radix-4 system where two input symbols are considered within one cycle. This is achieved by combining two successive radix-2 trellis steps into a single radix-4 trellis step. Fig. 7(a) shows two consecutive trellis steps of a radix-2 system. A radix-4 system combines these two trellis steps into one step as shown in Fig. 7(b). The trellis index axis in radix-2 is compressed by half since radix-4 ignores the odd trellis steps $(\ldots t-3, t-1, t+1\ldots)$ of the radix-2 system. The aim is that computational time taken for the state updates with radix-4 would be less than twice the time taken for state updates in a radix-2 system.

This method has been exploited to increase throughput in Viterbi systems [14]. However, this scheme is not very attractive for MAP detectors since four paths must be combined together instead of merely obtaining the maximum of four paths as in the Viterbi algorithm. Combining four paths requires a three-ACSLA-unit tree with two levels and the delay through the critical path is only one addition less than twice the delay for ACSLA.

Combining four paths in a radix-4 system is described by

$$a \boxplus b \boxplus c \boxplus d = \max(a,b,c,d)+\Delta \quad (7)$$

where the correction term Δ is not as simple an expression for implementation. However, if the two most significant paths are combined instead of all four paths, then the computation becomes simpler. Assume m_1 and m_2 are the largest two values among the four paths. Then the soft survivor update step becomes

$$m_1 \boxplus m_2 = \max(m_1,m_2)+\ln(1+e^{-|m_1-m_2|}). \quad (8)$$

This computation is similar to a radix-2 update. This new scheme is shown in Fig. 7 and is called radix4-sum2 since it is based on radix-4, but only effectively combines two paths. There are $\binom{4}{2}=6$ possible comparison pairs drawn from four values. *Logic1* and *Logic2* compute the maximum value and the difference between the maximum and the next largest value based on the six comparisons, respectively.

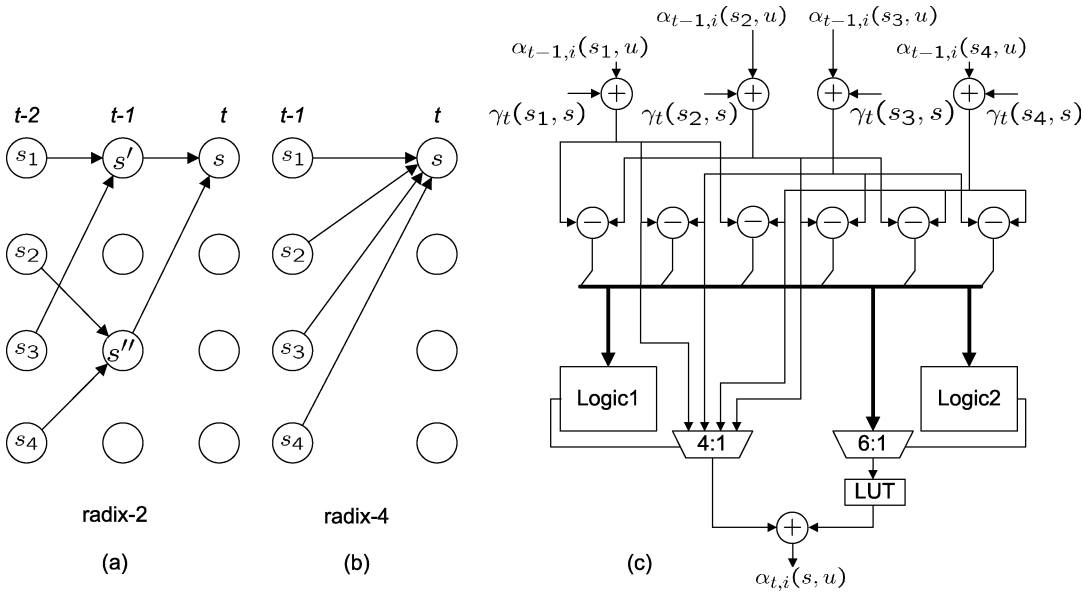


Fig. 7. (a) Radix-2 trellis. (b) Radix-4 trellis. $\gamma_t(s_i, s), i \in \{1, 2, 3, 4\}$ are branch metrics from state s_i to state s in radix-4 system. (c) Radix4-sum2 computation unit.

The delay of this new scheme is only moderately larger than the radix-2 system (due to the logic operation), but since two symbols are decoded within one cycle the throughput is increased compared to ACSLA. Moreover, the effect of combining two paths instead of four paths is shown to have only a marginal impact on BER performance, which is verified by simulation results presented in the next section.

C. CSA-Extend and Deep Pipelined Update

The two methods discussed so far, CSALA and radix4-sum2, consider reducing the critical path delay in the sum-product computation. We propose yet another method to increase the throughput that exploits the architecture of the FOMAP as shown in Fig. 4. This method gives the best throughput and hardware cost tradeoff. The FOMAP detector is implemented based on this method. As explained previously, the *extend* operation has a single cycle feedback loop that cannot be pipelined and limits detector throughput. In order to reduce its delay, computation for *extend* can be simplified by ignoring the correction term, which effectively reduces this operation to add/compare/select (ACS). We further reduce the delay for *extend* by reordering the ACS operation to a CSA operation. Thus, the critical path delay for *extend* is simply an addition and a selection. This simplification has minimal impact on overall BER performance, again verified by simulation results presented in the next section.

It is important to note that the *update* operation must not be simplified, but remain a full sum-product operation. Otherwise, the algorithm would reduce to a suboptimal algorithm. If the correction term in the *update* operation were to be ignored, then instead of combining the probabilities of paths, it chooses the path with the maximum probability. In essence, the algorithm is no longer a log-MAP algorithm but is equivalent to max-log-MAP algorithm [11] with much larger BER performance loss. In order to effectively increase the throughput of the system as a whole, the delay for the ACSLA unit in the *update* operation need not be comparable to the delay of the modified *extend* operation. Since data flow of the *update* operation does not have a loop-back path, its amenable to pipelining. Hence, we break the ACSLA unit of the *update* operation into two pipeline stages where the delay for each pipe stage is comparable to the delay of a CSA operation. This scheme is further discussed in Section IV. We call this method CSA-*extend* deep pipelined-*update*.

D. Comparison of the Schemes

Table II presents a comparison of these three methods in terms of the approximate number of computational units and storage devices. Delay of the sequential operation through the critical path is also shown for each method. The normalized throughput (CSALA normalized to 1) is based on schematic simulations using a 0.13 μm CMOS technology.

Even though the CSALA and radix4-sum2 methods improve throughput compared to a ACSLA-based FOMAP, they require significantly more computational units compared to other methods. Among the three FOMAP methods, CSA-*extend*-deep pipelined-*update* requires less hardware (ignoring the latches needed since synchronization is achieved by multi-phase clocks in the implementation, see Section IV) and enables higher throughput.

For comparison purposes, a high-throughput version of BCJR algorithm is included in the table. In order to increase the throughout this BCJR has two sliding windows that run in parallel. Detailed description of this scheme is given in the Appendix. Comparing the hardware requirements given in the table, it is evident that FOMAP base on CSA-*extend* deep pipelined-*update* is computationally expensive compared to BCJR. On the other hand, BCJR requires more storage and has lower throughput. Moreover, BCJR needs complex control mechanisms to control the forward, backward computational

TABLE II
COMPARISON OF THROUGHPUT INCREMENT SCHEMES

Method	FOMAP			BCJR
	CSALA	Radix4-Sum2	CSA-*extend* Deep-pipelined *up.*	
No. Adders	*ext.* : $0.5n_sn_u7$ *up.* : $(L-1)n_sn_u7$ *col.* : $(n_s-1)n_u2$ (5548)	*ext.* & *up.* : $0.5Ln_sn_u^211$ *col.* : $(n_s-1)n_u^22$ (8860)	*ext.* : $0.5n_sn_u5$ *up.* : $(L-1)n_sn_u4$ *col.* : $(n_s-1)n_u2$ (3212)	α : $2n_s4$ β : $2n_s4$ path concat.: $2n_sn_u2$ path col.: $2n_u(n_s-1)2$ (512)
No. MUXs	*ext.* : $0.5n_sn_u2$ *up.* : $(L-1)n_sn_u2$ *col.* : $(n_s-1)n_u$ (1598)	*ext.* & *up.* : $0.5Ln_sn_u^22$ *col.* : $(n_s-1)n_u^2$ (1660)	*ext.* : $0.5n_sn_u3$ *up.* : $(L-1)n_sn_u$ *col.* : $(n_s-1)n_u$ (846)	α : $2n_s$ β : $2n_s$ path col.: $2n_u(n_s-1)$ (128)
No. LUTs	*ext.* : $0.5n_sn_u$ *up.* : $(L-1)n_sn_u$ *col.* : $(n_s-1)n_u$ (814)	*ext.* & *up.* : $0.5Ln_sn_u^2$ *col.* : $(n_s-1)n_u^2$ (860)	*ext.* : $0.5n_sn_u$ *up.* : $(L-1)n_sn_u$ *col.* : $(n_s-1)n_u$ (814)	α : $2n_s$ β : $2n_s$ path col.: $2n_u(n_s-1)$ (128)
No. Storage elements[‡]	*ext.* : $0.5n_sn_u2$ *up.* : $(L-1)n_sn_u2$ *col.* : $(n_s-1)n_u$ (1598)	*ext.* & *up.* : $0.5Ln_sn_u^2$ *col.* : $(n_s-1)n_u^2$ (860)	*ext.* : $0.5n_sn_u3$ *up.* : $(L-1)n_sn_u3$[†] *col.* : $(n_s-1)n_u$ (2382)	BM : $11Ln_B$ α : $12Ln_s$ (8100)
Critical path delay	$2D_A+D_{LUT}$	$3D_A+D_{Logic}$ $+D_{LUT}+D_{MUX}$	D_A+D_{MUX}	$3D_A+D_{LUT}$
Normalized Throughput	1	1.1	1.7	1.1

n_s: No. of states, n_u: No. of inputs, L: Length of the training segment, n_B: No. of different branch metrics.

α: forward recursions, β: backward recursions

path concat.: path concatenation ($\alpha\gamma\beta$), path col. : path collection, BM : branch metrices.

All *col.* operations are assumed to be compare/sel/LUT/add units.

Convention - e.g. *ext.*: $0.5n_sn_u7$ indicates $0.5n_sn_u$ *ext.* units which has 7 adders within this unit.

[‡]: 7b-Latches for FOMAP for synchronization and 7b-registers/memory cells for BCJR for buffering.

[†]: Assumes buffer the values at a, b, e see Fig. 5.

(.): Values are for EEPR4 channel $n_s=16, n_u=2, L=25, n_B=12$.

D_A: Delay of a adder, D_{LUT}: Delay of LUT, D_{MUX}: Delay of a MUX, D_{Logic}: Delay for logic.

units and to combine their results appropriately. Further, forward and backward recursions and APP computations in the BCJR algorithm all process different channel outputs at a given instance, requiring multiple memory accesses per cycle.

Hardware requirements for both FOMAP and BCJR algorithms are governed by their respective window lengths. The storage requirements for BCJR is a linear function of its window length. Similarly the number of computational units required for FOMAP algorithms is proportional to its window length. For a given memory order, the FOMAP window length is about six times the constraint length of the channel or the encoder. This is equal to the length of the training segment (L) for forward or backward recursions of the BCJR algorithm. Typically BCJR window length is considered to be a multiple of L. A larger window length and having multiple windows sliding concurrently increase the throughput of the BCJR system with a cost of increase in storage capacity and latency.

Storage requirements for the BCJR algorithm can be implemented using area efficient multi-ported SRAM cells. Thus, for a given memory order and comparable window lengths a reasonable conjecture is that BCJR algorithm would be more area and power efficient. On the other hand FOMAP algorithm has significantly less input to output latency which is important for time critical applications. Another advantage of the FOMAP algorithm is its ability to increase throughput by reducing the sum-product operation throughput bottleneck. Simplifying sum-product operation in BCJR algorithm will lead to max-log-MAP with compromised BER performance. However, FOMAP algorithm can be modified to increase the throughput with minimal effect on BER performance as explained in the next section.

E. BER Performance

We now investigate the impact the throughput-enhancing methods described above have on BER performance. The radix4-sum2 method has a simplification in its sum-product operation. It only sums up two leading contenders among four paths. The CSA-*extend* deep pipelined-*update* method has a simplification in the *extend* operation. It ignores the correction term in the *extend* operation. We examine the effects of these simplifications on BER performance. Fig. 8 presents simulated BER results for the proposed methods assuming an EEPR4 channel. The BER for the original FOMAP algorithm is also shown for comparison.

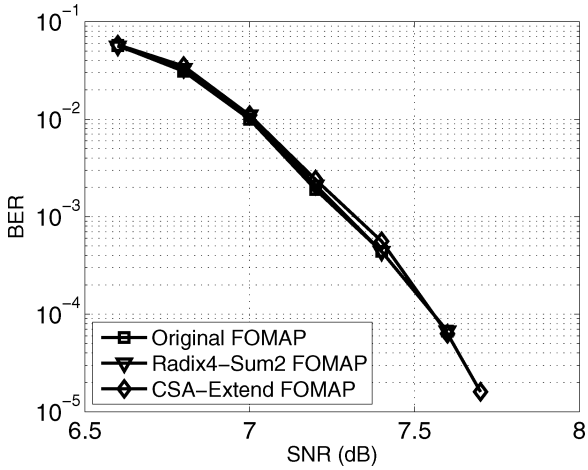


Fig. 8. BER performance of FOMAP for original algorithm, radix-4-sum and CSA-*extend* methods. EEPR4 channel.

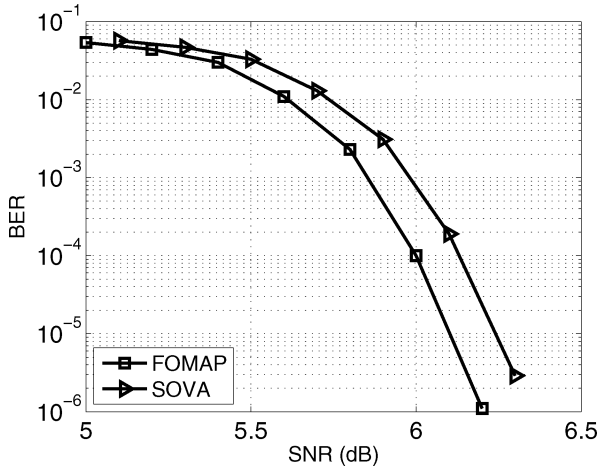


Fig. 9. BER performance for FOMAP and SOVA based for PR4 channel.

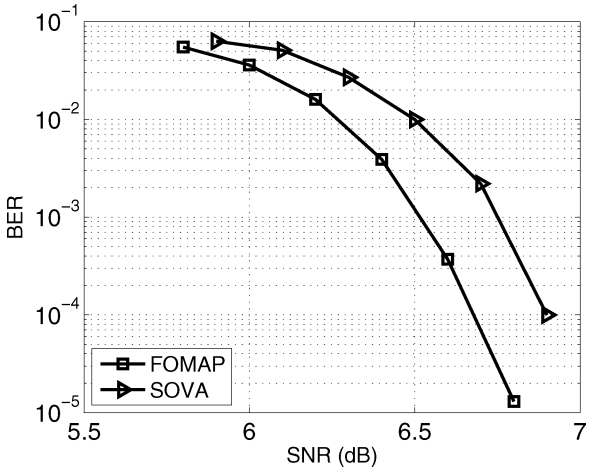


Fig. 10. BER performance for FOMAP and SOVA based for EPR4 channel.

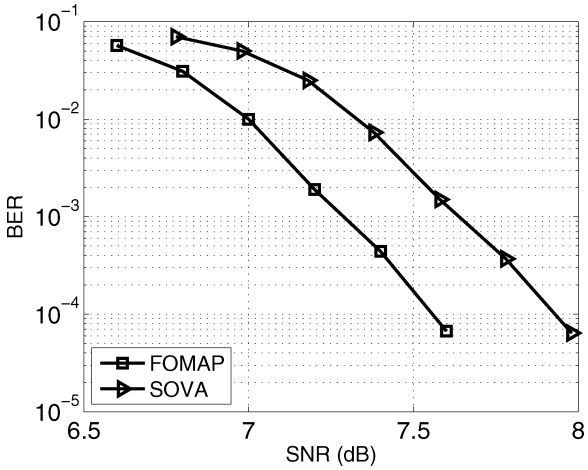


Fig. 11. BER performance for FOMAP and SOVA based for EEPR4 channel.

The simulation system consists of the FOMAP detector concatenated with a LDPC code decoder as shown in Fig. 1. The system has finite resolution where the FOMAP detector takes in 6-bit channel outputs and 6-bit *a priori* probabilities. The soft survivors are capped to 7 bits. The frame length of 5120 bits with code rate 0.8 and window length $L = 25$ are considered. The BER results are for 20 LDPC iterations, where for every five LDPC iterations, the FOMAP detector performs one iteration. As the results indicate, neither the simplification in radix4-sum2 method nor simplification in CSA-*extend* deep pipelined-*update* method has any significant affect on BER performance.[1]

Further, we investigate the BER performance gain of FOMAP over SOVA, for different partial response channel models for this fixed-point iterative system. The window length (trace-back path length) for SOVA is also 25 trellis steps. Figs. 9–11 show the performance for PR4, EPR4, and EEPR4 channels, respectively. The performance gain of FOMAP over SOVA is about 0.15 dB for PR4, 0.25 dB for EPR4, and 0.35 dB for EEPR4 at a BER of 10^{-4}. These results suggest that the BER performance gain of FOMAP over SOVA increases with increasing ISI.[2]

IV. CSA-*Extend* Deep Pipelined-*Update* FOMAP Implementation

The FOMAP algorithm was implemented using the CSA-*Extend* Deep Pipelined-*Update* architecture. The detector targets EEPR4 channel, which consists of 16 states. The detector takes 6-bit quantized channel symbols from an analog-to-digital converter (A/D) at the receiver input and 6-bit quantized *a priori* log-likelihood ratios from previous iterations to generate 8-bit *a posteriori* log-likelihood ratios for the next iteration. The window length of the detector is set to 25, which balances hardware requirements versus performance. Branch metrics are 6-bit values while state metrics and soft survivors are 7-bit binary values.

The FOMAP architecture for EEPR4 is an expanded version of the block diagram shown in Fig. 4. The detector has 32 survivor-updating processors per column corresponding to the 16

[1]Fig. 11 shows the BER performance gap between SOVA and FOMAP algorithms. This indicates that the LDPC decoder does not mask the performance difference resulting from different detection methods.

[2]The essence of partial response channel for magnetic storage systems is to accept ISI in a controlled manner. The channel is a linear combination of responses of known individual symbol transitions. The channel is known *a priori* and hence the ISI affect can be anticipated and accommodated in the detection process. PR4, EPR4, and EEPR4 consider the ISI affects of 3, 4, and 5 adjacent symbols, respectively, where higher number of symbols reflects the higher densities of the magnetic medium.

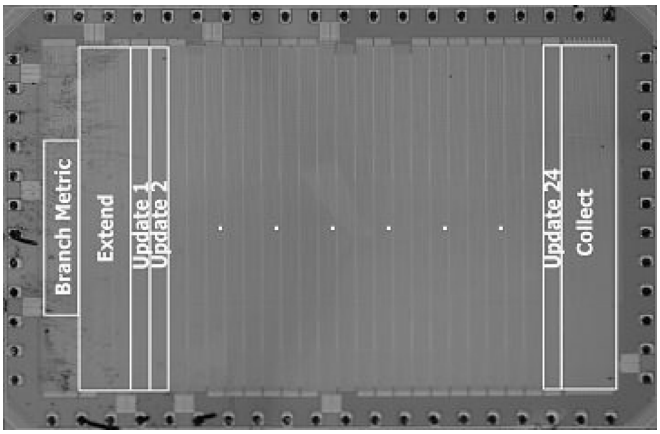


Fig. 12. Die microphotograph and floor plan overlay.

states and two input levels. There are a total of 32×25, 7-bit processors. Fig. 12 presents a die micrograph of the test chip, implemented in 0.13 μm CMOS, with the detector's floor plan superimposed onto the same figure. *Extend*, *update* and *collect* blocks are laid out as columns. The branch metric unit shown on the left computes the relevant branch metrics for each received symbol utilizing a look-up table.

The detector relies on high-speed, dual-rail domino logic to minimize gate delays and maximize throughput. Keepers inserted into each domino stage prevent discharge due to leakage and allow low-speed testing. The domino gates operate with respect to three equally spaced, overlapping clocks (ϕ_1, ϕ_2, ϕ_3), shown in Fig. 13(b), which make the circuit skew tolerant and facilitate time borrowing [15]. Soft survivors are temporarily stored in domino stages, obviating explicit latches, alleviating latch delay, and significantly reducing hardware. Otherwise, the prohibitively large number of latches that would be required (2382, 7-bit latches) makes such memory-intensive detector designs impractical to implement.

Fig. 13(c) shows the ACSLA unit used in the update operation, divided into multiple pipeline stages. Simple Manchester carry chain adders are used for the 7-bit additions. Logic for propagate, generate, and the carry chain evaluates during ϕ_1 and domino XOR gates that compute the sum evaluate on ϕ_2. The MUXs and LUT evaluate during ϕ_3. To prevent overflow, normalization circuitry is needed at the end of each *update* operation. The *update* operations are pipelined into two pipe stages with respect to ϕ_1 as illustrated in the figure.

One of the challenges associated with this architecture is the large number of wires connecting each column. Propagating the signals between *update* columns (each column consisting of 32 different soft survivors and each survivor represented by 7 bits) requires 224 wires. Generally, dual-rail domino circuits need monotonically rising signals and monotonically rising complements of the signals. Hence, propagating the complementary signals would double the required number of wires. This wiring complexity significantly increases the area overheard of the layout and reduces the area utilization.

An alternative approach to reduce the required number of wires is to generate the complementary signal locally by utilizing a complementary signal generator (CSG) [16]. The circuit

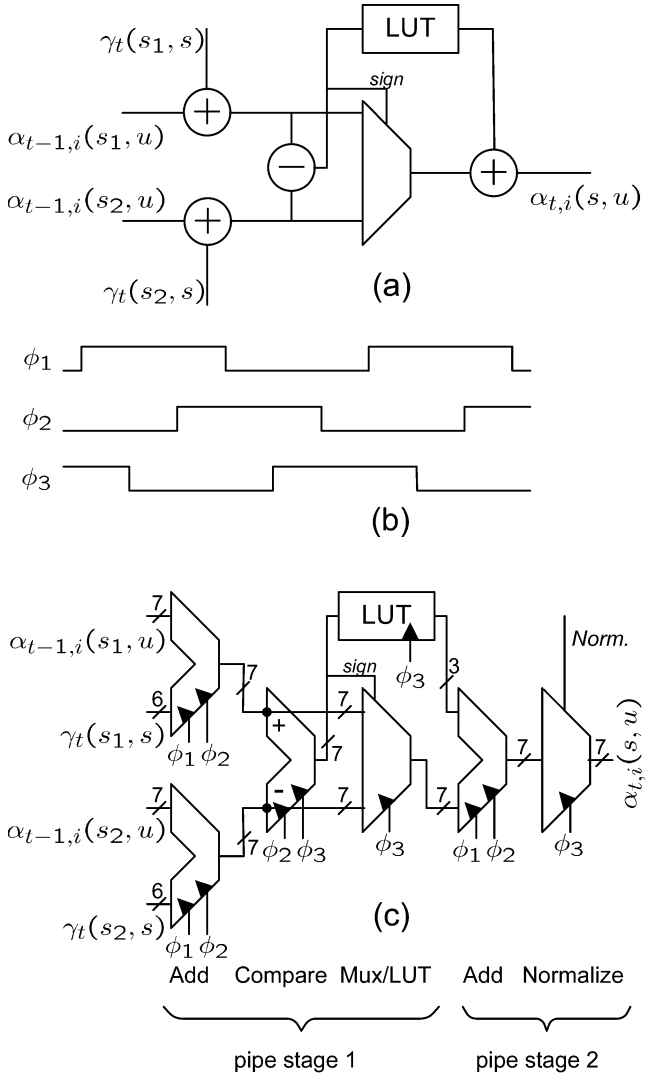


Fig. 13. *Update* processor. (a) ACSLA unit. (b) Overlapping clocks. (c) Deep-pipelined ACSLA circuit. The pipe stages respect to ϕ_1 are also shown.

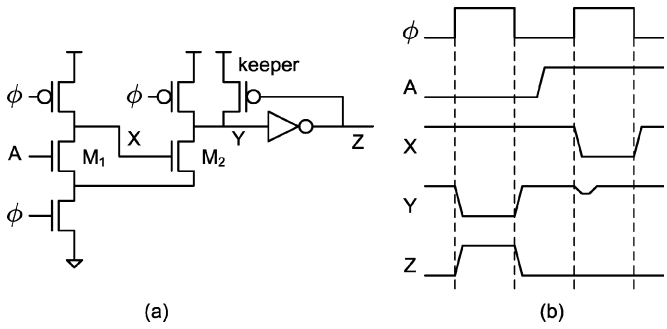


Fig. 14. (a) Complementary signal generator. M_1 is significantly stronger than M_2. (b) Input, intermediate and output signals.

schematic of the CSG is shown in Fig. 14(a). The CSG generates monotonically rising complementary signals that are suitable inputs for domino logic. Fig. 14(b) illustrates the waveform diagram where a monotonically rising input signal A results in monotonically rising complementary signal Z. CGS alleviates the need to propagate the complementary signal and only the true signals connect between the update columns.

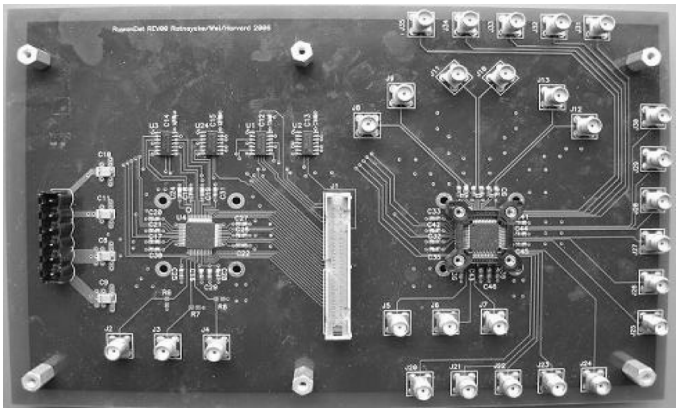


Fig. 15. Photograph of the detector test board. The chip under test on the left operates at low frequencies, and the chip on the right operates at high frequencies.

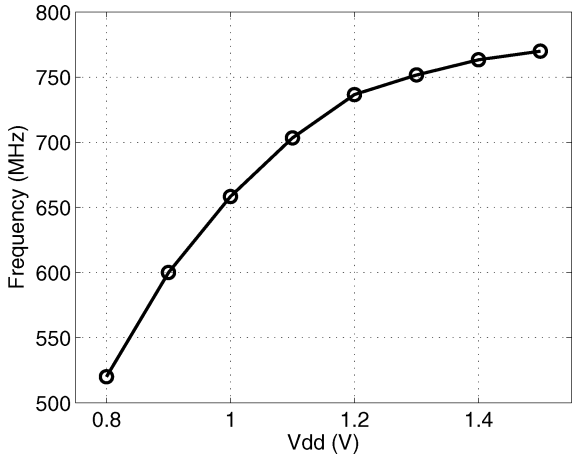


Fig. 16. Experimentally measured throughput.

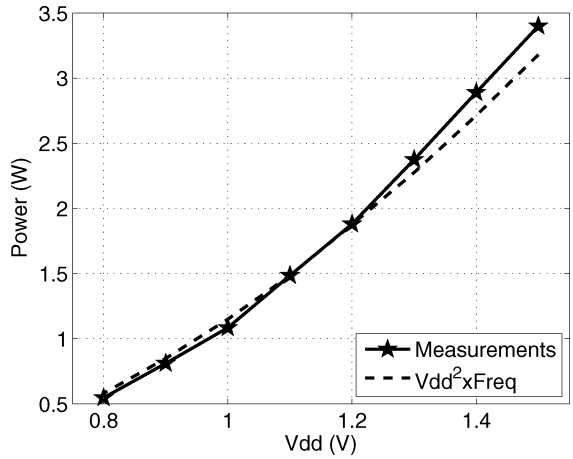


Fig. 17. Experimentally measured power dissipation and corresponding $V_{\rm dd}^2 f$ curve.

The input signal to the CSG must be stable before the clock signal goes high. This imposes a hard edge on the input signal wherever this generator is used. Thus, the signals propagating between *update* columns are constrained by a hard edge and, hence, the last domino gate in the *update* computation cannot borrow time in a subsequent stage. However, all other domino gates in the pipeline can borrow time across stages within an *update* column.

V. Measurement Results

The test-chip prototype was implemented in a 0.13 μm CMOS logic process and was experimentally verified in two stages. First, intermediate values generated by *extend*, *update* and *collect* blocks were tested for correctness while operating at low frequencies. The inputs at each time instant, namely 6-bit channel symbols and 6-bit *a priori* log-likelihood ratios were fed to the detector through a scan chain and outputs at each intermediate stage were compared with the expected vectors. Then, the detector was tested for maximum achievable throughput. Given the difficulty of externally feeding two sets of 6-bit inputs at high speeds, pseudo-random inputs generated by an on-chip linear feedback shift register were used to feed the detector and the corresponding outputs were verified. Lastly, in order to alleviate output buffer speed constraints, the outputs operate at half rate. The test fixture for the two-stage testing is shown in Fig. 15, with two test chips mounted on the board. The left and right halves are designed for low- and high-frequency testing, respectively.

Fig. 16 presents the experimentally measured average frequency plotted with respect to supply voltage. The averages were taken across three prototype chips. For each supply voltage, clock frequency is increased until the outputs become invalid. As shown in the figure, the slope of the frequency curve decreases with increasing supply voltage and flattens out. There are several reasons for this behavior. First, it is well known that the speed of logic gates does not scale linearly with supply voltage. Second, wire delays significantly impact the maximum achievable frequency. As described earlier, the detector requires a large number of long wires in order to connect blocks across the *update* columns. As supply voltages increase, the domino gates and inverter repeaters speed up, but wire delays remain relatively constant. Thus, the fraction of the delay due to propagation through the wires increases with increasing frequency, reducing the slope of the frequency curve. On average, the detectors achieve clock speeds greater than 750 MHz which corresponds to maximum throughput rates over 750 Mb/s.

TABLE III
Summary of Chip Characteristics

Max throughput	$>$ 750Mb/s
Power dissipation	2.4W (at 750MHz, 1.3V)
Transistor count	2.054M (w/o I/O buffers)
Die Dimensions	2633x3793μm
Die Area	9.9mm^2
Core Area	7.1mm^2
Technology	0.13μm CMOS, 7 metal

Fig. 17 shows the experimentally measured average power consumption plotted with respect to supply voltage. The measurements were obtained at ambient temperature of approximately 25 °C, but the on-die temperature can be higher while operating at higher power levels. The corresponding $V_{\rm dd}^2 f$ trend (normalized to the measured power at 1.2 V) is

TABLE IV
COMPARISON TO PRIOR WORK

	Proposed[1]	E. Yeo[1] *et al.* [9]	S. Lee[1] *et al.* [17]	P. Urard[2] *et al.* [18]	M.Bickerstaff[2] *et al.* [19]	C. Wong[2] *et al.* [20]	B.Bougard[2] *et al.* [21]
Algorithm	FOMAP	SOVA	BCJR	BCJR	BCJR	Max-log-MAP	BCJR
Technology	0.13μm	0.18μm	0.18μm	0.13μm	0.18μm	0.13μm	0.18μm
Application	HDD	HDD	W.Comm	W.Comm	W.Comm	W.Comm	W.Comm
Frequency	750MHz	500MHz	285MHz	350MHz	145MHz	80MHz	160MHz
Throughput: MAP core Turbo Dec. (iteration)	750Mb/s	500Mb/s	285Mb/s	350Mb/s 350Mb/s (5)	290Mb/s 24Mb/s (6)	160Mb/s (8)	75.6Mb/s ($\sim$ 4)
Power	2400mW	400mW	300mW	2464mW	1450mW	275mW	N/A
Energy	6.4† nJ/b/iter	1.6† nJ/b/iter	4.8(est) nJ/b/iter	1.4 nJ/b/iter	10 nJ/b/iter	0.22 nJ/b/iter	2.2 nJ/b/iter
Core Area	7.1mm^2	0.5mm^2	4.3mm^2	10mm^2	14.5mm^2	17.8mm^2	7.1mm^2
No.Tr.(Gates)	2M	170K	150K	N/A	(410K)	(2.67M)	(373K)
No.States	16	8	8	16	8	8	8

Max-log-MAP method is based on BCJR(simplified) algorithm.

[1]: Implement a single MAP component detector or decoder.

[2]: Implement turbo decoder with multiple MAP component decoders and memory banks.

†: estimated energy = 2×Power/(MAP core throughput). The factor 2 is for two passes through the detector -per iteration.

HDD: Hard Disk Drives.

W.Comm: Wireless Communications.

superimposed on the plot. Here, f is the maximum attainable frequency for a given $V_{\rm dd}$. The figure shows that the power requirement increases significantly with frequency and voltage. The measured power curve tracks $V_{\rm dd}^2 f$ closely, but deviates slightly at higher voltages. This deviation can be attributed to higher leakage power, which is not taken into account in the $V_{\rm dd}^2 f$ curve.

While the design initially targeted 1 GHz operation, the maximum achievable frequency was constrained due to several reasons. One reason is the underestimation of wire parasitics in the CAD flow, which directly affects propagation delay and maximum operational frequency. Further, excess parasitic capacitance results in fanout mismatches along the clock distribution network within the detector. Post fabrication simulations show that mismatch in rise and fall times within a buffer, combined with fanout mismatch along the clock buffer chain, results in decreasing the duty cycle of the clock signals in our design. This decrease in duty cycle can severely degrade the maximum achievable clock rate because it reduces the overlap period between the three clock phases, availability of time borrowing, and time for signals to propagate through adjacent clock domains. Due to these reasons, the measured frequencies deviate from the original frequency targets of the detector. The key characteristics of the test-chip prototype are summarized in Table III.

Table IV shows the comparison of this work with prior work in the literature. We limit this comparison to work with chip implementations and measured results. Several other authors such as Raghupathy and Liu, Miyauchi *et al.*, and Bickerstaff *et al.* have done considerable work in this area and we defer the reader to [22]–[24] for comparison of their work. As far as we know, this work and the SOVA implementation described in [9] are the only chips that specifically target magnetic storage devices, which require very high throughputs. Other implementations given in the table target various wireless communication standards where low power, in contrast to throughput, is the main objective.

Given that these devices target different applications and channel models, and are implemented in different technologies, it is not straightforward to make a fair comparison. Hence, we make the comparisons within a given subset. This work and designs described in [17]–[19] and [21] implement the optimal MAP method whereas designs in [9] and [20] are based on sub-optimal methods. Among the optimal MAP implementations we infer that this work achieves the highest throughput. The next recorded highest throughput in this category is 350 Mb/s. Among the two high-throughput detectors targeting magnetic storage devices, our work implements the optimal MAP algorithm. The detector described by [9] implements SOVA which is sub-optimal in terms of BER performance. Further, our implementation targets high density magnetic storage devices where ISI is modeled as a 16-state EEPR4 channel. Other implementations, except [18], target systems with eight states.

VI. CONCLUSION

In pursuit of higher performance for future generation communications applications, iterative detection and decoding methods are being considered where MAP detection and decoding ought to be used given their superior BER. Towards

addressing this, we propose a maximum *a posteriori* probability detector that implements FOMAP, a recently developed algorithm and achieves throughputs greater than 750 Mb/s. The algorithm inherits attractive features from both Viterbi and traditional BCJR algorithms, namely parallel survivor updating and an ability to compute *a posteriori* probabilities. High throughput is achieved by exploiting key aspects of this forward-only MAP algorithm and leveraging high-speed circuit techniques. The detector has been implemented in a 0.13 μm CMOS technology and experimentally verified. This is the first published implementation of the forward-only MAP algorithm and the highest throughput demonstrated for a MAP algorithm in VLSI.

Achieving very high-throughputs, the primary objective of this work, led to a circuit design based on dynamic logic that consumes high power. Power consumption is further exacerbated by the aggressive channel model we have targeted. With technology scaling we may be able to reduce the power requirement. We believe the most feasible method to implement the FOMAP architecture in nanoscale technologies would be to use static logic, which consumes less power. Static logic would also alleviate the complications associated with leakage power. Leakage power has been steadily increasing with technology scaling and especially problematic for dynamic gates. While static logic is generally slower than dynamic logic and requires latches, the inherent speed up offered by scaling process technology ought to compensate for the slower logic family used. Trading dynamic logic for static logic with latches shifts the burden of clocking dynamic gates to clocking latches. Thus, the proposed FOMAP detector architecture implemented in 65 nm or 45 nm CMOS should be able to meet the higher throughput rates (on the order of several Gb/s) demanded by future devices.

In order to achieve high speeds for this FOMAP design we have leveraged algorithm and circuit level tradeoffs. Continual research for such tradeoffs in other algorithms such as BCJR ought to yield similar gains in throughput. We hope that the proposed architecture for the FOMAP algorithm, detector implementation and tradeoff methods described in this paper enables the use of optimal algorithms for future high-speed magnetic storage devices.

APPENDIX
HIGH-THROUGHPUT BCJR METHOD

This Appendix describes the high-throughput BCJR algorithm that was considered in Section III. Fig. 18 shows the forward, backward recursions of this method. There are two forward processors denoted by α_1 and α_2 and two backward processors denoted by β_1 and β_2. The APP computing processors are Λ_1 and Λ_2. We define the training length L as the minimum number of trellis steps required to be processed in order to obtain reliable state metrics after starting from all equally probable states.

Each processor starts and process through the training segment and obtain sufficiently reliable state probabilities. The state metrics computed within the training segment are not used for APP computations. After completing the training segment, each processor continues processing in the same direction for another segment of length $3L$. The state metrics generated within this segment are used for APP computations (Hence the sliding window length is $3L$). Thus, each forward and backward processor pair generates $3L$ APPs for each $4L$ of channel outputs processed. Thus, each pair has APP throughput of 3/4 times the rate of the state updates. Since there are two such pairs operating in parallel, the combined throughput is 3/2 times the rate of state updates.

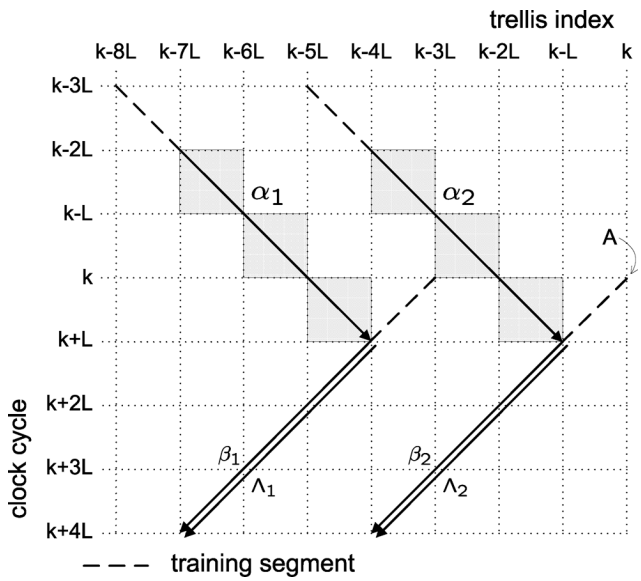


Fig. 18. Forward and backward processing of a high throughput BCJR method. There are two forward (α_1, α_2) and two backward processors (β_1, β_2). Each processor generates APPs for window of length $3L$.

We define latency as the number of clock cycles a channel output required to be stored before the corresponding APP is computed. We assume the channel outputs arrive one per cycle and adequate number of them are stored to be processed. Thus, kth channel output can be processed on or after the kth clock cycle. Assume kth channel output is processed on kth clock cycle by backward processor β_2 as shown by A in Fig. 18. This dictates the starting time of all other processors. Thus, α_1 and α_2 starts at clock cycle $k - 3L$. Therefore, the longest latency occurs for channel output $k - 7L$ since the corresponding APP output is generated at clock cycle $k + 4L$ resulting a latency of $11L$. Hence, the system needs to store at least $11L$ channel outputs or corresponding branch metrics.

ACKNOWLEDGMENT

The authors thank E. F. Haratsch, Z. Keirn, and Agere Systems for their generous support of this work and chip fabrication.

REFERENCES

[1] F. Sun and T. Zhang, "Quasi-reduced-state soft-output Viterbi detector for magnetic recording read channel," *IEEE Trans. Magn.*, vol. 43, no. 10, pp. 3921–3924, Oct. 2007.

[2] B. Zand and D. A. Johns, "High-speed CMOS analog Viterbi detector for 4-PAM partial-response signalling," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Jul. 2002, vol. 37, pp. 895–903.

[3] P. J. Black and T. H. Y. Meng, "A 1-Gb/s, four-state, sliding block Viterbi decoder," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Jun. 1997, vol. 32, pp. 797–805.

[4] I. Lee and J. L. Sonntag, "A new architecture for the fast Viterbi algorithm," in *Proc. IEEE GLOBECOM*, Nov. 2000, pp. 1664–1668.

[5] M. Tuchler, R. Koetter, and A. C. Singer, "Turbo equalization: Principals and new results," *IEEE Trans. Commun.*, vol. 50, no. 5, pp. 754–767, May 2002.
[6] G. Bauch and V. Franz, "A comparison of soft-in/soft-out algorithms for Turbo detection," in *Proc. Int. Conf. Telecomm.*, Jun. 1998, pp. 259–263.
[7] C. Douillard, M. Jezequel, C. Berrou, A. Picart, P. Didier, and A. Glavieux, "Iterative correction of intersymbol interference: Turbo-equalization," *Eur. Trans. Telecomm.*, vol. 6, pp. 507–511, Sep. 1995.
[8] X. Wang and H. V. Poor, "Iterative (Turbo) soft interference cancellation and decoding for coded CDMA," *IEEE Trans. Commun.*, vol. 47, no. 7, pp. 1046–1061, Jul. 1999.
[9] E. Yeo, S. A. Augsburger, W. R. Davis, and B. Nikolic, "A 500-Mb/s soft-output Viterbi decoder," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Jul. 2003, vol. 38, pp. 1234–1241.
[10] J. Hagenauer and P. Hoeher, "A Viterbi algorithm with soft-decision outputs and its applications," in *Proc. IEEE GLOBECOM*, Nov. 1989, vol. 3, pp. 1680–1686.
[11] B. Vucetic and J. Yuan, *Turbo Codes: Principals and Applications*. Boston, MA: Kluwer Academic, 2000.
[12] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimum decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inf. Theory*, vol. 20, no. 2, pp. 284–287, Mar. 1974.
[13] X. Ma and A. Kavčić, "Path partition and forward-only trellis algorithm," *IEEE Trans. Inf. Theory*, vol. 49, no. 1, pp. 38–52, Jan. 2003.
[14] P. J. Black and T. H. Y. Meng, "A 140-Mb/s, 32 state, radix-4 Viterbi decoder," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Dec. 1992, vol. 27, pp. 1877–1885.
[15] D. Harris, *Skew-Tolerant Circuit Design*. San Mateo, CA: Morgan Kaufmann, 2001.
[16] N. H. E. Weste and D. Harris, *CMOS VLSI Design*. Reading, MA: Addison Wesley, 2004.
[17] S. Lee, N. Shanbhag, and A. C. Singer, "A 285 MHz pipelined MAP decoder in 0.18 μm CMOS," *IEEE J. Solid-State Circuits*, vol. 40, no. 8, pp. 1718–1725, Aug. 2005.
[18] P. Urard, L. Paumier, M. Viollet, E. Lantrebecq, H. Michel, S. Muroor, B. Coates, and B. Guptha, "A generic 350 Mb/s Turbo-codec based on a 16-states SISO decoder," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2004, vol. 1, pp. 424–425.
[19] M. Bickerstaff, L. Davis, C. Thomas, D. Garrett, and C. Nicol, "A 24 Mb/s radix-4 logMAP Turbo decoder for 3GPP-HSDPA mobile wireless," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2003, vol. 1, pp. 150–151.
[20] C. Wong, C. Tang, M. Lai, Y. Zheng, C. Lin, H. Chang, C. Lee, and Y. Su, "A 0.22 nJ/b/iter 0.13 μm Turbo decoder chip using inter-block permutation interleaver," in *Proc. IEEE Custom Integrated Circuits Conf. (CICC)*, Sep. 2007, pp. 273–276.
[21] B. Bougard, A. Giulietti, V. Derudder, J. Weijers, S. Dupont, L. Hollevoet, F. Catthoor, L. V. der Perre, H. D. Man, and R. Lauwereins, "A scalable 8.7 nJ/bit 75.6 Mb/s parallel concatenated convolutional (Turbo-) codec," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2003, vol. 1, pp. 152–153.
[22] A. Raghupathy and K. J. R. Liu, "VlSI implementation considerations for Turbo decoding using a low latency log-MAP," in *Proc. Int. Conf. Consumer Electronics (ICCE)*, Jun. 1999, pp. 182–183.
[23] T. Miyauchi, K. Yamamoto, T. Yokokawa, M. Kan, Y. Mizutani, and M. Hattori, "High-performance programmable SISO decoder VLSI implementation for decoding Turbo codes," in *Proc. IEEE GLOBECOM*, Nov. 2001, vol. 1, pp. 305–309.
[24] M. Bickerstaff, D. Garrett, T. Prokop, C. T. B. Widdup, Z. Gongyu, C. Nicol, and Y. Ran-Hong, "A unified Turbo Viterbi channel decoder for 3GPP mobile wireless in 0.18 μm CMOS," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Nov. 2002, vol. 37, pp. 1555–1564.

Ruwan Ratnayake (M'02) received the B.S. degree in electrical engineering from the University of Sydney, Sydney, Australia, in 1998, and the M.S. degree in electrical and computer engineering from the University of California at Santa Barbara in 2002. He received the Ph.D. degree in electrical engineering from Harvard University, Cambridge, MA, in 2008.

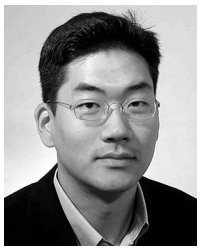
From 1998 to 2001, he was a senior research engineer at the Institute for Infocomm Research in Singapore. His research interests include signal processing and VLSI design at systems architecture level.

Dr. Ratnayake is a member of the IEEE ComSoc Data Storage Technical committee and has one patent in this area.

Aleksandar Kavčić received the Dipl. Ing. degree in electrical engineering from Ruhr-University, Bochum, Germany, in 1993, and the Ph.D. degree in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, in 1998.

Since 2007, he has been with the University of Hawaii, Honolulu, where he is presently an Associate Professor of electrical engineering. Prior to 2007, he was with the Division of Engineering and Applied Sciences at Harvard University, as an Assistant Professor of electrical engineering from 1998 to 2002, and as John L. Loeb Associate Professor of Natural Sciences from 2002 to 2006. While on leave from Harvard University, he served as a Visiting Associate Professor at the City University of Hong Kong in the Fall of 2005 and as a Visiting Scholar at the Chinese University of Hong Kong in the Spring of 2006. He held short-term research positions at Seagate Technology in 1995, Read-Rite Corporation in 1996, and Quantum Corporation from 1997 to 1998. He served as a technical consultant for Quantum Corporation in 1999 and 2000, Link-A-Media in 2004, and Agere Corporation in 2004. He is presently serving on the Advisory Board of Link-A-Media Corporation.

Prof. Kavčić received the IBM Partnership Award in 1999 and the NSF CAREER Award in 2000. He is a co-recipient, with X. Ma and N. Varnica, of the 2005 IEEE Best Paper Award in Signal Processing and Coding for Data Storage. He served on the Editorial Board of the IEEE TRANSACTIONS ON INFORMATION THEORY as Associate Editor for Detection and Estimation from 2001 to 2004. From 2005 until 2007, he was the Chair of the Signal Processing for Storage Technical Committee of the IEEE Communications Society.



Gu-Yeon We received the B.S., M.S., and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 1994, 1997, and 2001, respectively.

He is currently an Associate Professor of electrical engineering in the School of Engineering and Applied Sciences, Harvard University, Cambridge, MA. After a brief stint as a Senior Design Engineer at Accelerant Networks, Inc., Beaverton, OR, he joined the faculty at Harvard as an Assistant Professor in January 2002. His research interests span several areas, including high-speed, low-power link design, mixed-signal circuits for communications, ultra-low-power hardware for wireless sensor networks, and co-design of circuits and computer architecture for high-performance and embedded processors to address PVT variability and power consumption that plague nanoscale CMOS technologies.