A Decoder Architecture for High-Speed Free-Space Laser Communications

Michael Cheng, Michael Nakashima, Jon Hamkins, Bruce Moision, and Maged Barsoum*
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109-8099

ABSTRACT

We present a decoding architecture for high-speed free-space laser communications. This system will be used by NASA's Mars Laser Communication Demonstration (MLCD) project, the first use of high-speed laser communication from deep space. The Error Correction Code (ECC) and modulation techniques for this project have been motivated by an analysis of capacity, and existing designs have been shown to operate within 0.9 dB of the Shannon limit on the nominal operating point. In this paper, we give the algorithmic description and FPGA implementation details that led to the development of a 50 Mbps hardware decoder.

Keywords: Laser Communication, Error Correction Code, Turbo Codes, MAP decoding, FPGA implementation.

1. INTRODUCTION

The Mars Laser Communications Demonstration (MLCD) is planned to be the first demonstration of optical communications from a satellite in Deep Space to a Ground Receive Terminal on Earth. Nominal downlink (from spacecraft to Earth) data rates of over 1 Mbps are desired. However, with certain operating conditions, communication at 50 Mbps can be demonstrated.

NASA's legacy ECC design for RF communication is the concatenation of an inner convolutional code and an outer Reed-Solomon (RS) code [1]. Decoding is performed in one pass and hard bit-decisions are made. The decoding cost is high. The discovery of Turbo codes [2] and their suboptimal but effective low-complexity iterative decoding approach has generated much excitement in the coding community. The MLCD ECC design is the serial concatenation of an inner modulation code and an outer convolutional code. Modulation is a mapping of bits to symbols transmitted on the channel. This mapping may be considered a code and demodulation as decoding of the code. Conventionally, the modulation and ECC are decoded independently, with the demodulator sending it's results to the ECC decoder. However, we may consider the combination of the modulation and the ECC as a single large code, which maps user information bits directly to the symbols transmitted on the channel. We could gain several dB in performance by decoding the ECC and modulation jointly as a single code relative to decoding them independently. An exact ML decoding of the joint modulation–ECC code would, in most cases of practical interest, be prohibitively complex.

However, we may approximate true ML decoding while limiting the decoder complexity by iteratively decoding the modulation and the ECC. This is in fact the "Turbo" principle and more details can be found in [3].

2. CHANNEL DESCRIPTION

This work models the optical communication systems as seen in Figure 1. The information bits $\mathbf{U} = (U_1, U_2, \cdots, U_k)$ are independent identically distributed (i.i.d.) binary random variables assumed to take on the values 0 and 1 with equal probability. The vector $\mathbf{U}$ is encoded to $\mathbf{C} = (C_1, C_2, \cdots, C_n)$, a vector of n M-ary Pulse Position Modulation (PPM) symbols. Each M-PPM symbol is a number in $\{0, 1 \cdots, M-1\}$ and represents a block of $\log_2 M$ bits. There is one signal slot and $M-1$ nonsignal slots for each M-PPM symbol. On the Poisson channel, a nonsignal slot has average photon count n_b and a signal slot has average count $n_s + n_b$ so that the likelihood ratio (LR) of slot i is calculated by

$$\begin{aligned} LR(k_i) &= \frac{e^{-(n_s+n_b)}\left(n_s+n_b\right)^{k_i}/k_i!}{e^{-n_b} n_b^{k_i}/k_i!} \\ &= e^{-n_s}\left(1+\frac{n_s}{n_b}\right)^{k_i}. \end{aligned} \tag{1}$$

*Portion of this paper is the subject of a patent application NTR 4123 by the California Institute of Technology.

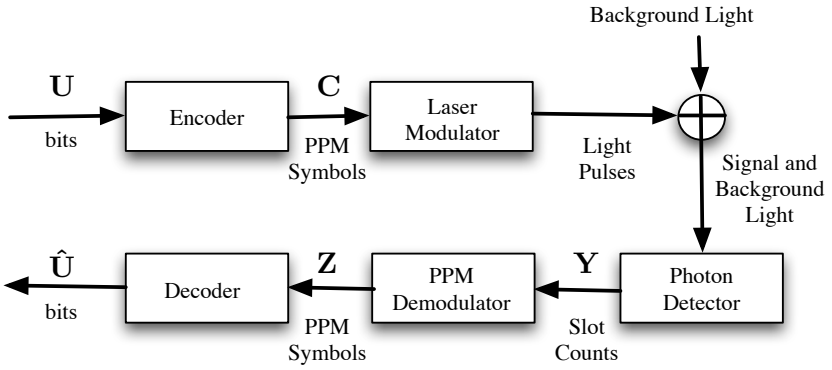

Figure 1. An optical communication system.

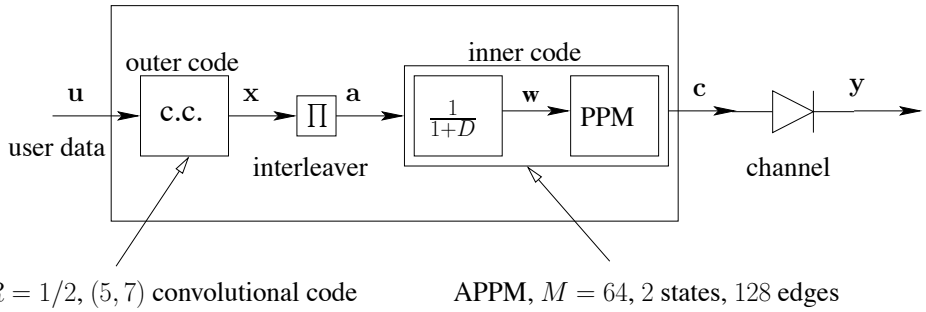

Figure 2. The SCPPM encoder. In our system, we adopt the $(5,7)$ convolutional code as the outer code. The PPM order M can be of 32, 64, or 128.

From the slot likelihood ratios, we can calculate the probability that symbol i was transmitted as $LR(k_i) / \sum_{j=1}^{M} LR(k_j)$ and the probability that the i^{th} bit is a one as $\sum_{l:\text{bit } i \text{ is } 1} LR(k_l) / \sum_{j=1}^{M} LR(k_j)$. The log likelihood ratio (LLR) of bit i is then $\log\left(\sum_{l:\text{bit } i \text{ is } 1} LR(k_i) / \sum_{l:\text{bit } i \text{ is } 0} LR(k_i)\right)$. The capacity of an optical channel employing PPM and APD is presented in both [4] and [5].

3. ENCODER ARCHITECTURE

Sections 3 through 5 are materials taken out of [6], we include the contents here for completeness.

The Serial Concatenated Pulse Position Modulation (SCPPM) encoder is shown in Figure 2. A block of information symbols $\mathbf{u} = (\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_K)$ is encoded by an outer convolutional code to yield a coded sequence $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_K)$. Each is $\mathbf{u}_k$ a binary vector of length p_o and each $\mathbf{x}_k$ a binary vector of length q_o. The code rate is $\frac{p_o}{q_o}$. The i-th components of these vectors are denoted $u_{k,i}, x_{k,i}$.

The sequence $\mathbf{x}$ is permuted, bit-wise, to produce the sequence $\mathbf{a}$. Both $\mathbf{x}$ and $\mathbf{a}$ have length $q_o K$. This permutation is commonly referred to as an interleaving of the sequence. The sequence $\mathbf{a}$ is encoded by an inner code consisting of an accumulator and PPM mapping to produce the coded sequence $\mathbf{c}$. The trellis that describes the inner code consists of 2 states and $M/2$ parallel branches between connecting states.

Moision and Hamkins [6] compared the SCPPM design to a coded PPM scheme that uses Reed-Solomon (RS) code as the outer code. Their results showed that SCPPM with iterative decoding performs better. The increase in the decoding complexity of SCPPM is also manageable. In fact, for the nominal operating point, SCPPM has a 3 dB signal energy gain over RS-PPM of the same code rate. They further demonstrated that the $(5,7)$ rate $1/2$ convolutional code is an effective outer code selection that led to a performance that is only 0.9 dB away from the Shannon capacity at a Bit Error Rate (BER) of 10^{-6}.

The interleaver used is characterized by a 2nd order polynomial $f(x) = ax + bx^2$. The bit in position x is mapped via the interleaver to position $f(x) \mod N$. Here it is required that b is divisible by the prime factors

(a)

Anti-Gray construction	Corresponding Label
000	0
111	1
100	2
011	3
110	4
001	5
010	6
101	7

(b)

a	w	Natural	Gray	Anti-Gray
000	000	0	0	0
001	111	7	7	5
010	110	6	3	6
011	001	1	4	3
100	100	4	1	2
101	011	3	6	7
110	010	2	2	4
111	101	5	5	1

Figure 3. (a) The M-ary anti-Gray mapping is constructed by taking the first $M/2$ entries of the M-ary Gray mapping (marked by normal font) and inserting the inverse (marked by bold) pattern in-between each entries. (b) Bit-to-Symbol APPM Mapping (from a to Anti-Gray). Initial accumulator state is 0.

of N and that a is not [7]. For MLCD we have $N = 15120 = 2^4 \cdot 3^3 \cdot 5 \cdot 7$. Candidate interleavers are of the form $f(x) = ax + 210mx^2$, where m is a positive integer and a does not have $2, 3, 5$ or 7 as a factor. Among this class we have observed good performance with the polynomial $f(x) = 11x + 210x^2$.

Barron and Robinson [8] have derived a recursive implementation of a permutation polynomial interleaver mapping that requires only additions. Let $[\cdot]$ be the mod N operator. Expand $f(x+1)$ as

$$\begin{aligned} f(x+1) &= \left[a(x+1) + b(x+1)^2\right] \\ &= \left[f(x) + g(x)\right], \end{aligned} \tag{2}$$

where $g(x) = [a + b + 2bx]$. Expanding $g(x)$ similarly yields $g(x+1) = [g(x) + 2b]$.

4. BITS TO PPM SYMBOL MAPPING AND PARTIAL STATISTICS

Due to inter-slot-interference (ISI) and timing jitter, a fraction of each pulse energy may appear in adjacent slots. PPM symbols with pulses in adjacent slots will be more likely to be confused with one another in ISI. To mitigate the effects of ISI, the input bit streams $\mathbf{a}$ mapped by adjacent slots should have a large Hamming distance. To accomplish this, the anti-Gray mapping is used, where bit streams represented by pulses in adjacent slots are Hamming distance $\log_2 M$ or $\log_2 M - 1$ apart. The anti-Gray mapping can be constructed from the Gray mapping by, for example taking the Gray mapping entries that has a prefix 0 and inserting the inverse pattern between each entry. The anti-Gray construction and bit mapping for $M = 8$ are shown in Figure 3. A gain of 0.3 dB in signal energy has been observed of the anti-Gray mapping relative to the Gray or natural mapping [9].

To realize the gains of iterative decoding algorithms would nominally require a likelihood be computed and stored for every slot of each PPM symbol in a codeword. However, high data rates, large values of M, and large interleavers can make likelihood storage and processing prohibitively expensive.

To reduce the complexity of iterative decoding, we may discard the majority of the channel likelihoods [6], operating the decoder using only the remainder. This may be accomplished by transmitting only a subset consisting of the largest likelihoods during each symbol duration–the likelihoods corresponding to the slots with the largest number of observed symbols. The observation of the remaining slots is set to the mean of a noise slot. In low background noise, a small subset may be chosen with negligible loss.

5. DECODER ARCHITECTURE

5.1. Overview of Architecture

The SCPPM decoder diagram is given in Figure 4. The decoder operates iteratively to refine the overall decision through repeated decoding trials. The symbol I indicates input to the constituent decoders and O indicates output. The LLR's passed between the soft-in-soft-out (SISO) decoders are extrinsic information obtained by

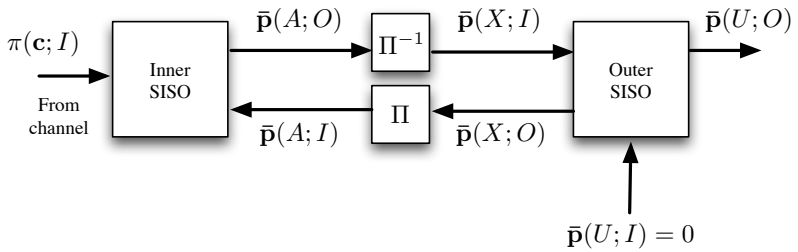

Figure 4. SCPPM decoder

subtracting the LLR's provided to their respective inputs. Iterating on only the extrinsic information reduces undesirable feedback that could bias the decisions. The Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm [10] is used to update the likelihoods. The procedure involves traversing the trellis that represents each of the constituent codes in a forward and backward fashion and therefore, calculating the state and transition metrics that would lead to maximum *a-posteriori* (MAP) bit decisions.

5.2. The Log-Domain SISO Decoding

Let $\mathcal{V}$ be the set of states and $\mathcal{E}$ be the set of directed labeled edges in a trellis. Each edge $e \in \mathcal{E}$ has an initial state $i(e)$, a terminal state $t(e)$, an input label $a(e)$ and an output label $c(e)$. The details of the SISO algorithm, which uses the BCJR approach as its underlying principle, are presented in [6], [10], and [11]. To facilitate hardware realization, the SISO algorithm, is implemented in the log-domain, which translates multiplications into additions, and may be less sensitive to round-off errors in fixed-point arithmetic. The algorithm may be converted in a straight-forward manner by defining [6], [11]

$$\begin{aligned}
\bar{\alpha}_k(s) &= \log\alpha_k(s) \\
\bar{\beta}_k(s) &= \log\beta_k(s) \\
\bar{\gamma}_k(s) &= \log\gamma_k(s) \\
\bar{\lambda}_k(e) &= \log\lambda_k(e)
\end{aligned}$$

$$\begin{aligned}
\bar{p}_{k,i}(A;I) &= \log\frac{p_{k,i}(a=0;I)}{p_{k,i}(a=1;I)} \\
\bar{p}_{k,i}(A;O) &= \log\frac{p_{k,i}(a=0;O)}{p_{k,i}(a=1;O)} \\
\pi_k(\mathbf{a};I) &= \log p_k(\mathbf{a};I) + \text{ constant} \\
\pi_k(\mathbf{c};I) &= \log p_k(\mathbf{c};I) + \text{ constant}
\end{aligned}$$

As input to the SISO algorithm, we receive the bit LLRs $\bar{p}_{k,i}(A;I)$ from the outer SISO decoder. We desire to compute output bit LLRs $\bar{p}_{k,i}(A;O)$. Note that for $a \in \{0,1\}$

$$\log p_{k,i}(a;I) = \frac{1}{2}(-1)^a \bar{p}_{k,i}(A;I) + \frac{1}{2}\log(p_{k,i}(0;I)p_{k,i}(1;I)). \tag{3}$$

The second term is a constant relative to a and can be factored out. Hence, we may compute edge input symbol LRs as

$$\pi_k(\mathbf{a};I) = \sum_{i=1}^{\log_2 M} \frac{1}{2}(-1)^{a_i}\bar{p}_{k,i}(A;I) + \text{constant} \tag{4}$$

Transforming the α and β iterations yields

$$\bar{\gamma}_k(e) = \pi_k(\mathbf{a};I) + \pi_k(\mathbf{c};I) \tag{5}$$

$$\begin{aligned}
\bar{\alpha}_k(s) &= \log \sum_{e:t(e)=s\in\mathcal{V}} \alpha_{k-1}(i(e))\gamma_k(e) \\
&= \log \sum_{e:t(e)=s\in\mathcal{V}} \exp\left(\bar{\alpha}_{k-1}(i(e)) + \bar{\gamma}_k(e)\right) \qquad (6)
\end{aligned}$$

$$\bar{\beta}_k(s) = \log \sum_{e:i(e)=s\in\mathcal{V}} \exp\left(\bar{\beta}_{k+1}(t(e)) + \bar{\gamma}_{k+1}(e)\right) \tag{7}$$

5.3. The $\overset{*}{\max}$ Operation

Implementation of the BCJR algorithm in the log domain requires taking the log of sums of exponentials. This function is defined as the $\overset{*}{\max}$ operation [12]:

$$\begin{aligned} \overset{*}{\max}(x,y) &\triangleq \log\left(e^x + e^y\right) \\ &= \max(x,y) + \log\left(1 + e^{-|x-y|}\right). \end{aligned} \tag{8}$$

It is also noted that

$$\begin{aligned} \overset{*}{\max}(x,y,z) &= \log\left(e^x + e^y + e^z\right) \\ &= \overset{*}{\max}\left(\overset{*}{\max}(x,y), y\right). \end{aligned} \tag{9}$$

By pre-computing $\log(1 + e^{-|x-y|})$ and storing the results in a table, we have a low complexity implementation of $\overset{*}{\max}$ in hardware. More details are presented in Section 6.2.

5.4. Simplified Computation with Parallel Edges

The APPM trellis has 2-states and $2M$ edges per stage as seen in Figure 5. The forward and backward recursions on this trellis require taking the $\overset{*}{\max}$ of $\frac{M}{2}$ edges. Suppose each 2-input $\overset{*}{\max}$ operation incurs a delay of one clock cycle. A direct implementation of the forward-backward algorithm would require a delay of $\log_2(M/2)$ cycles just for the $\overset{*}{\max}$'s. We show here how the computation may be pipelined, reducing the $M/2$-input $\overset{*}{\max}$ operation to a 2-input $\overset{*}{\max}$ operation that may be computed in one clock cycle. We consider the $\bar{\beta}$ recursion, all others follow in the same manner [13].

In the product domain, it is straightforward to see an application of the distributive law (multiplication distribution over addition) saves computations on a trellis with parallel edges:

$$\begin{aligned} \beta_k(s) &= \sum_{e:i(e)=s} \beta_{k+1}(t(e))\gamma_{k+1}(e) \\ &= \sum_{e:i(e)=s,t(e)=s} \beta_{k+1}(s)\gamma_{k+1}(e) + \sum_{e:i(e)=s,t(e)=s'\neq s} \beta_{k+1}(s')\gamma_{k+1}(e) \\ &= \left(\beta_{k+1}(s) \sum_{e:i(e)=s,t(e)=s} \gamma_{k+1}(e)\right) + \left(\beta_{k+1}(s') \sum_{e:i(e)=s,t(e)=s'} \gamma_{k+1}(e)\right) \\ &= \beta_{k+1}(s)\gamma'_{k+1}(s,s) + \beta_{k+1}(s')\gamma'_{k+1}(s,s') \end{aligned} \tag{10}$$

where $\gamma'_{k+1}(s,s)$, a sum over parallel edges, is referred to as the *Super Gamma* for the state pair (s,s) at stage $k+1$, the same calculation can be made for the state pair (s,s').

We have an analogous simplification in the log domain via the distributive law (addition distributive over $\overset{*}{\max}$) which can be seen by taking logarithms of both sides of (10)

$$\begin{aligned} \bar{\beta}_k(s) &= \log\left(\exp\left(\bar{\beta}_{k+1}(s) + \bar{\gamma}'_{k+1}(s,s)\right) + \exp\left(\bar{\beta}_{k+1}(s') + \bar{\gamma}'_{k+1}(s,s')\right)\right) \\ &= \overset{*}{\max}_{\tilde{s}\in\{s,s'\}}\{\bar{\beta}_{k+1}(\tilde{s}) + \bar{\gamma}'_{k+1}(s,\tilde{s})\} \end{aligned} \tag{11}$$

where

$$\begin{aligned} \bar{\gamma}'_k(s,s') &= \log\left(\sum_{e:i(e)=s,t(e)=s'} e^{\bar{\gamma}_{k+1}(e)}\right) \\ &= \overset{*}{\max}_{e:i(e)=s,t(e)=s'}\{\bar{\gamma}_{k+1}(e)\} \end{aligned} \tag{12}$$

Since the $\bar{\gamma}'_k$s are not a function of a recursively computed quantity, they may be pre-computed via a pipeline as illustrated in Figure 6. Mapping $\bar{\lambda}$s to bit probabilities require taking the $\overset{*}{\max}$ of M inputs and this latency can be reduced similarly using the pipeline implementation.

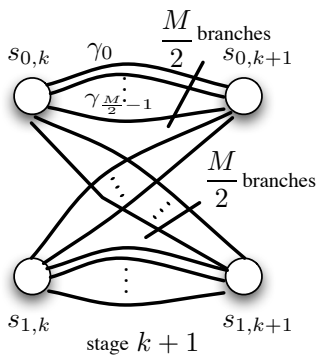

Figure 5. A single trellis stage with $\frac{M}{2}$ parallel edges between connecting states.

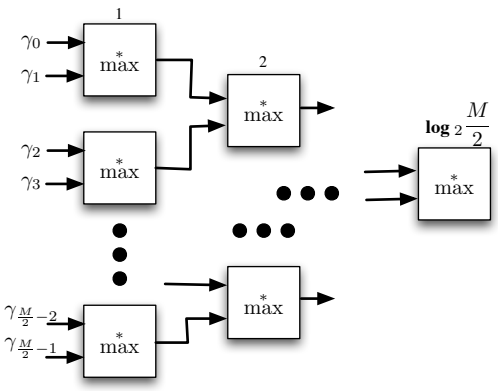

Figure 6. Pipelined $\max^*$ to compute the Super γ's.

5.5. SCPPM Decoding Algorithm Flow

We describe the flow of our log-domain decoding algorithm. The inner code is an accumulate-PPM with $M = 64$ shown in Figure 7. The outer code is the rate 1/2, 4-state $(5, 7)$ convolutional code shown in Figure 8. The interleaver has length 15120-bits.

1. (INNER SISO) The channel symbol log-likelihoods $\pi_k(\mathbf{c}; I)$ are delivered from the receiver to the decoder via the Receiver Interface Block. In our current design, only the top 8 symbol statistics are used. This reduces the amount of data transfer required from the receiver and the block memory (BRAM) needed for storage. When the γ calculation is ready, the de-multiplexer (de-mux) in Figure 7 directs the partial symbols statics from the Channel Memory Block and sets the remaining symbol statics to the mean of the noise slot (indicated by filler data). On initialization, the bit LLRs $\bar{p}_{k,i}(A; I)$, indicated by $P_k(A)$ in Figure 7, are set to zero. On successive iterations, bit LLRs $\bar{p}_{k,i}(A; I)$ are obtained from the outer code via the interleaver. The Vector Sums Block computes edge input symbol LLRs by

$$\pi_k(\mathbf{a}; I) = \sum_{i=1}^{\log_2 M} \frac{1}{2}(-1)^{\mathbf{a}_i} \bar{p}_{k,i}(A; I)$$

   for $\mathbf{a} \in \{0, 1, \ldots, 63\}$ (meaning the corresponding 6-bit vectors $\mathbf{a}$) and $k = 1, \ldots, 2520$.

2. The γ-block takes as input the Vector Sums and LLRs from the de-multiplexer (de-mux) to compute

$$\bar{\gamma}_k(e) = \pi_k(a(e); I) + \pi_k(c(e); I)$$

   for $e \in \mathcal{E}$ and $k = 1, \ldots, 2520$. There are 128 edges in $\mathcal{E}$.

3. The β-block uses the γ's for each edge to compute

$$\bar{\gamma}'_k(s, s') = \max^*_{e:i(e)=s, t(e)=s'} \{\bar{\gamma}_k(e)\}$$

   for each pair of initial and terminal states $(s, s') \in \{(s_{0,k-1}, s_{0,k}), (s_{0,k-1}, s_{1,k}), (s_{1,k-1}, s_{0,k}), (s_{1,k-1}, s_{1,k})\}$ and each $k = 1, \ldots, 2520$. These are 32-input $\max^*$ operations, and may be computed using the Super-$\bar{\gamma}$ pipelined algorithm. The β-block then initializes the log β's to $\bar{\beta}_{2520}(s) = 0$ for all s, and recursively computes (and stores in BRAM)

$$\bar{\beta}_k(s) = \max^* \left(\bar{\beta}_{k+1}(s_{0,k+1}) + \bar{\gamma}'_{k+1}(s, s_{0,k+1}), \bar{\beta}_{k+1}(s_{1,k+1}) + \bar{\gamma}'_{k+1}(s, s_{1,k+1})\right)$$

   for $s \in \{s_{0,k}, s_{1,k}\}$ and $k = 1, \ldots, 2520$. Note, these are 2-input $\max^*$ operations.

4. The $\alpha\backslash\lambda$-block initializes
$$\bar{\alpha}_0(s) = \begin{cases} 0, & s = s_{0,0} \\ -\infty, & s = s_{1,0} \end{cases},$$
follows the same procedure as the β-block to calculate the Super $\bar{\gamma}$'s, and recursively computes
$$\bar{\alpha}_k(s) = \overset{*}{\max}\,(\bar{\alpha}_{k-1}(s_{0,k-1}) + \bar{\gamma}'_k(s_{0,k-1}, s), \bar{\alpha}_{k-1}(s_{1,k-1}) + \bar{\gamma}'_k(s_{1,k-1}, s))$$
for $s \in \{s_{0,k}, s_{1,k}\}$ and $k = 1, \ldots, 2520$. The $\bar{\alpha}$'s are used immediately to form the current $\bar{\lambda}$'s and the next $\bar{\alpha}$'s; therefore, only $\bar{\alpha}$'s of the current stage are stored. The $\bar{\lambda}$'s are calculated as
$$\bar{\lambda}_k(e) = \bar{\alpha}_{k-1}(i(e)) + \bar{\gamma}_k(e) + \bar{\beta}_k(t(e))$$
for $e \in \mathcal{E}$ and $k = 1, \ldots, 2520$. In hardware, $-\infty$ is stored as the most negative fix-point value, see (14).

5. The $p_k\,(A)$-block takes the $\bar{\lambda}$'s and computes
$$\bar{p}_{k,i}(A; O) = \overset{*}{\max}_{e \in \mathcal{E}^A_{0,i}}\{\bar{\lambda}_k(e)\} - \overset{*}{\max}_{e \in \mathcal{E}^A_{1,i}}\{\bar{\lambda}_k(e)\} - \bar{p}_{k,i}(A; I)$$
for $k = 1, \ldots, 2520$, $i = 0, \ldots, 5$. These are 64-input $\overset{*}{\max}$ operations and are calculated using the pipelined algorithm.

6. The $\bar{p}_{k,i}(a; O)$'s are stored in the de-interleaver in sequential order and read out in (de)permuted order.

7. (OUTER SISO) The bit LLRs $\bar{p}_{k,i}(X; I)$ from the inner code are read via the de-interleaver. The Vector Sum Block in Figure 8 then computes edge output symbol log-likelihoods (LRs) by
$$\pi_k(\mathbf{x}; I) = \sum_{i=1}^{2} \frac{1}{2}(-1)^{\mathbf{x}_i}\bar{p}_{k,i}(X; I)$$
for $\mathbf{x} = 0, 1, 2, 3$ and $k = 1, \ldots, 7560$. $\mathbf{x}$ lists the 4 outputs possible each stage and k is the stage counter.

8. The LRs are passed to the γ-block
$$\bar{\gamma}_k(e) = \pi_k(x(e); I)$$
for $e \in \mathcal{E}$ and $k = 1, \ldots, 7560$. There are 8 edges in $\mathcal{E}$. Note: $\pi_k(u(e); I) = 0$ for all iterations, hence does not appear in the addition.

9. The β-block initializes
$$\bar{\beta}_{7560}(s) = \begin{cases} 0, & s = s_{0,7560} \\ -\infty, & s = s_{j,7560}, j = 1, 2, 3 \end{cases}$$
and recursively computes
$$\bar{\beta}_k(s) = \overset{*}{\max}_{e:i(e)=s}\{\bar{\beta}_{k+1}(t(e)) + \bar{\gamma}_{k+1}(e)\}$$
for $s \in \{s_{0,k}, s_{1,k}, s_{2,k}, s_{3,k}\}$ and $k = 1, \ldots, 7560$. These are 2-input $\overset{*}{\max}$ operations.

10. The $\alpha\backslash\lambda$ block initializes
$$\bar{\alpha}_0(s) = \begin{cases} 0, & s = s_{0,0} \\ -\infty, & s = s_{j,0}, j = 1, 2, 3 \end{cases}$$
and recursively computes
$$\bar{\alpha}_k(s) = \overset{*}{\max}_{e:t(e)=s}\{\bar{\alpha}_{k-1}(i(e)) + \bar{\gamma}_k(e)\}$$
for $s \in \{s_{0,k}, s_{1,k}, s_{2,k}, s_{3,k}\}$ and $k = 1, \ldots, 7560$. These are 2-input $\overset{*}{\max}$ operations. These $\bar{\alpha}$'s are then used to form
$$\bar{\lambda}_k(e) = \bar{\alpha}_{k-1}(i(e)) + \bar{\gamma}_k(e) + \bar{\beta}_k(t(e))$$
for $e \in \mathcal{E}$ and $k = 1, \ldots, 7560$. Again $-\infty$ is stored in hardware as the most negative fix-point value, see (14)

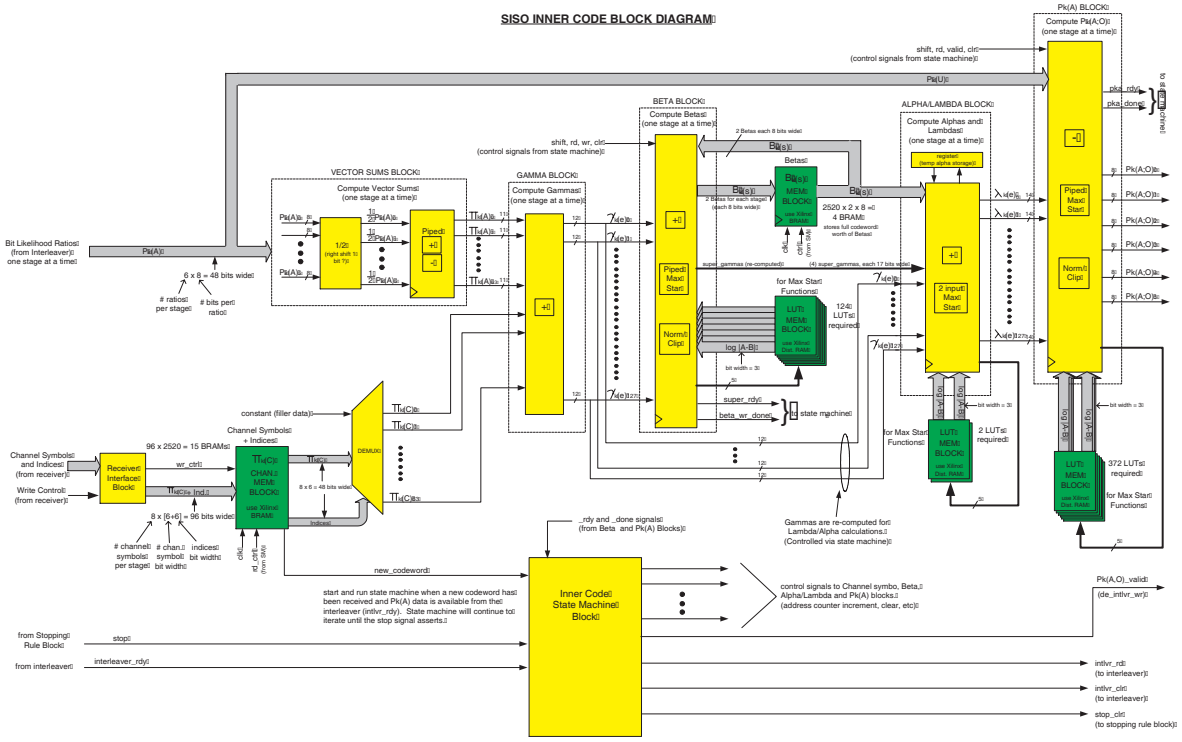

Figure 7. Block diagram of inner SCPPM decoder.

11. The $\bar{\lambda}$'s are input to the $P_k\left(U;O\right) \setminus P_k\left(X;O\right)$ block to calculate output LLRs

$$\bar{p}_{k,i}(U;O) = \max^{*}_{e \in \mathcal{E}^{U}_{0,i}} \{\bar{\lambda}_k(e)\} - \max^{*}_{e \in \mathcal{E}^{U}_{1,i}} \{\bar{\lambda}_k(e)\}$$

$$\bar{p}_{k,j}(X;O) = \max^{*}_{e \in \mathcal{E}^{X}_{0,j}} \{\bar{\lambda}_k(e)\} - \max^{*}_{e \in \mathcal{E}^{X}_{1,j}} \{\bar{\lambda}_k(e)\} - \bar{p}_{k,j}(X;I)$$

for $k = 1, \ldots, 7560$, $i = 0$, and $j = 0, 1$.

12. The $\bar{p}_{k,j}(X;O)$'s are then written to the interleaver in a permuted order, the $\bar{p}_{k,i}(U;O)$'s are sliced to form the bit-decisions. The bits are checked using the stopping rule. If satisfied, the decoding algorithm terminates; otherwise, the procedure starts again from Step 1.

The stopping rule [6] consists of a check on whether the output bits form a valid convolutional codeword and that the bits pass a 16-bit Cyclic-Redundancy-Check (CRC). If both of these conditions are met, the decoding iteration is stopped and the codeword is output as the decision.

6. HIGHLIGHTS OF DECODER IMPLEMENTATION

6.1. Metric Quantization

In software, the decoder metrics such as the states $\bar{\alpha}$'s, $\bar{\beta}$'s, the edges $\bar{\gamma}$'s, $\bar{\lambda}$'s, and the likelihoods are of floating point values. However, in hardware, these metrics are represented by fix-point integers in two's complement form.

The input quantization parameters are available bit width w and decimal precision p. The decoder variables (or metrics) are represented as integers, that is,

$$quantized = (int)\,\mathrm{round}\left(float \cdot base^{p}\right), \tag{13}$$

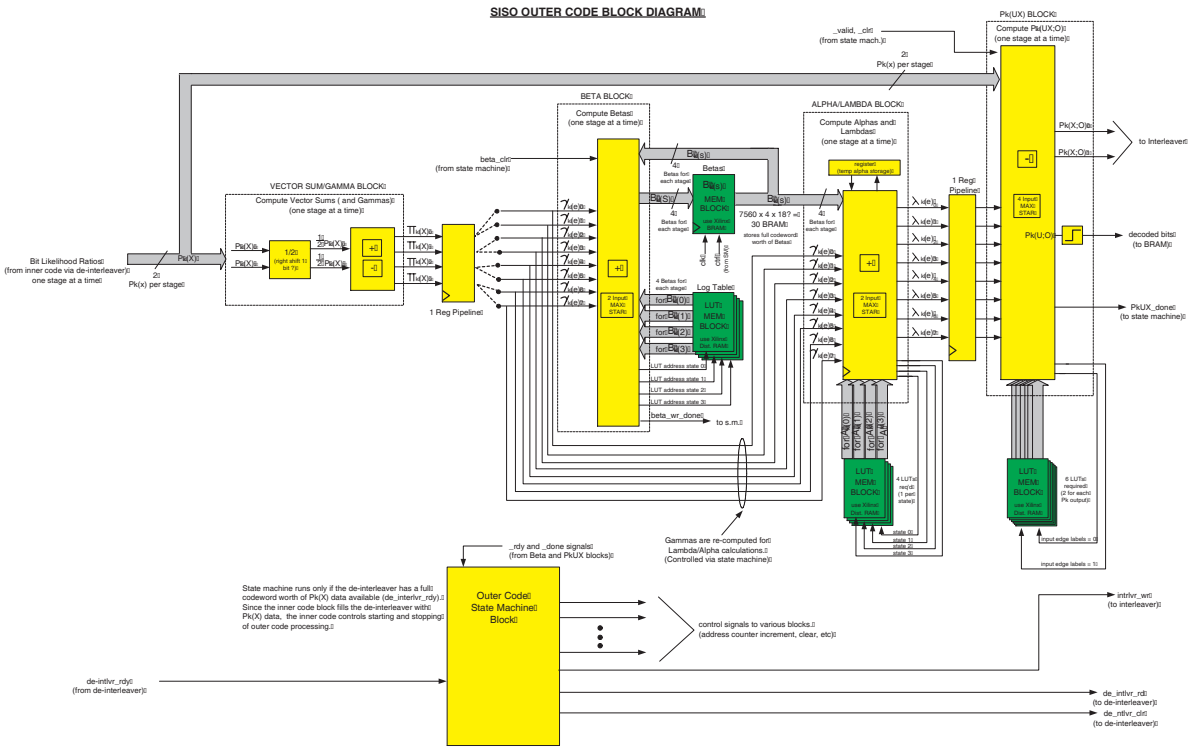

Figure 8. Block diagram of outer SCPPM decoder.

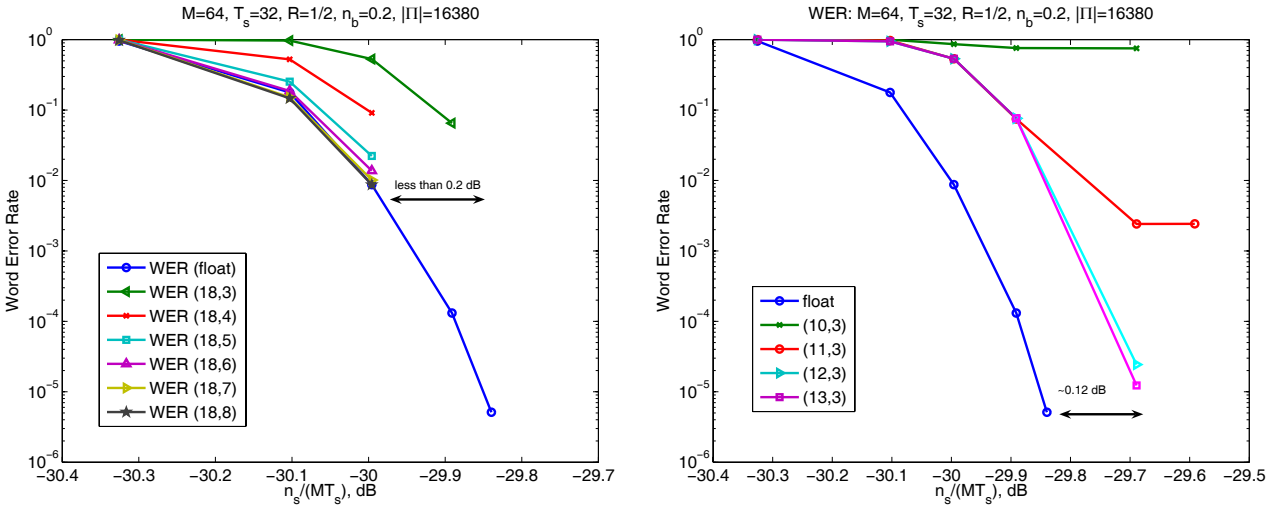

Figure 9. (a) Decimal precision and (b) Dynamic range Word Error Rate (WER) for a typical operating point.

where $base = 2$ but could be any number. The quantized variable are also clipped at maximum and minimum allowable integer values, that is,

$$quantized \in \left[-2^{w-p-1}+1, 2^{w-p-1}-1\right]. \tag{14}$$

Uniform quantization and clipping is applied in the same way to all decoder variables. This includes the Log-Likelihood Ratios (LLR)'s provided by the channel.

The quantization results are plotted versus WER in Figure 9. The parameters are: M the PPM order, T_s the slot width in nanoseconds, n_b the background noise in photons per slot, and n_s the signal in photons per

(A)

δ	0	1	2	$\cdots$	$m-1$
$v(\delta)$	$v(0)$	$v(1)$	$v(2)$	$\cdots$	$v(m-1)$

(B)

δ	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
$v(\delta)$	6	5	5	4	4	3	3	3	3	2	2	2	2	1	1	1	1	1	1	1	1	1

Table 1. (A) The max star Look-Up-Table (LUT) and (B) the LUT with $p = 3$.

pulse. The number pair in the legend indicates the total number of bits used and the number of bits used for decimal precision. For example, $(18, 3)$ means 18 total bits used, 15-bit for dynamic range and 3-bit for decimal precision. These curves are generated with an S-random interleaver of length 16380. The effects of quantization on decoding performance using the length 15120 polynomial interleaver will be the same. Note the occurrence of an error floor when there is an insufficient number of bits used to represent the dynamic range.

The signal energy gap can be reduced by clipping (or saturating) only the $\bar{\alpha}$'s, $\bar{\beta}$'s, and channel LLRs. The other decoder variables are allowed to grow throughout the data path. Another benefit of limited clipping is the ease of debugging. Every clipping point adds a source of potential discrepancy between software and hardware values (if a mistake exists in hardware implementation). Therefore, limiting the clipping points also facilitate hardware debugging because possible locations of mismatches are reduced. The cost is in the increased bus width (and logic) required to represent all other variables in the hardware.

Performance results for an 8-bit quantization, 3 bits for decimal precision, and 5 bits for dynamic range, are plotted in Figure 10.

6.2. The $\overset{*}{\max}$ Look-Up-Table

The log function is costly to realize in hardware. The $\overset{*}{\max}$ operation discussed in Section 5.3 is therefore, implemented as a Look-Up-Table (LUT). Since all variables are to be represented by fixed-point integers, for any real number x, we assign its quantized value to be

$$x_q = \min\left(\max\left(\text{round}\left(x \cdot 2^p\right), -2^{w-p-1}+1\right), 2^{w-p-1}-1\right) \tag{15}$$

Let the adjustment term in (8) be defined as

$$\Delta \triangleq \ln\left(1+e^{-|x-y|}\right) \approx \ln\left(1+e^{-\frac{|x_q-y_q|}{2^p}}\right). \tag{16}$$

Montorsi and Benedetto [14] suggested a way of generating the fixed-point $\overset{*}{\max}$ LUT with m entries, where m is the smallest positive integer that satisfies

$$\ln\left(1+e^{-m/2^p}\right) \leq 2^{-(p+1)} \tag{17}$$

rearranging the terms we have

$$m = \left\lceil -2^p \cdot \ln\left(e^{2^{-(p+1)}}-1\right)\right\rceil. \tag{18}$$

Each entry in the fixed-point $\overset{*}{\max}$ LUT is indexed by the difference in the two fix point arguments $\delta = |x_q - y_q|$ and has a value calculated as

$$v(\delta) = \text{round}\left(\ln\left(1+e^{-\delta/2^p}\right)\cdot 2^p\right). \tag{19}$$

The fixed-point $\overset{*}{\max}$ LUT would look like Table 1(A). Calculating $\max^*(x, y)$ in fixed-point representation is therefore done by

$$\overset{*}{\max}(x_q, y_q) = \max(x_q, y_q) + v(|x_q - y_q|). \tag{20}$$

Using the Example in [14], let $p = 3$, from (18) m is computed to be 22. The max star LUT is then populated as seen in Table 1(B). If $\delta = |x_q - y_q| = 7$, then the $\overset{*}{\max}$ operation would return $\max(x_q, y_q) + 3$. If $\delta = 0$, then $\overset{*}{\max}$ would return $\max(x_q, y_q) + 6$. If $\delta = 21$, then $\overset{*}{\max}$ would return $\max(x_q, y_q) + 1$. Any fixed-point number x_q can be converted back to floating point value by $x = \frac{x_q}{2^p}$. Another fast $\overset{*}{\max}$ hardware implementation can be found in [15]. The effects of using $\overset{*}{\max}$ LUTs on performance is also included in the $(8, 3)$ quantization curve of Figure 10. The partial-statistics result is also presented in the Figure.

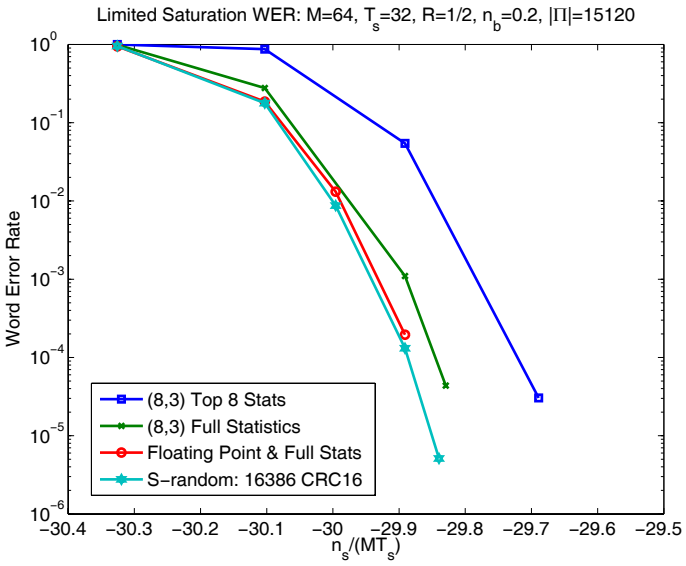

Figure 10. Performance of partial statistics and quantization of a typical operating point.

Full Decoder	used/total	utilization	Inner Decoder	Outer Decoder	Other Blocks
BRAM	101/168	60 %	19 % of total resource	9 % of total	32 % of total
Flip Flops	17311/93184	18 %	16 % of total resource	1 % of total	1 % of total
Slices	30174/46592	64 %	52 % of total resource	6 % of total	6 % of total

Table 2. SCPPM decoder on the Xilinx Virtex II-8000 FPGA.

6.3. Other Optimizations

Other optimizations of the SCPPM hardware implementation includes a fast clipping circuit and efficient interleaver and de-interleaver design. The fast clipping circuit requires no comparators and this reduces propagation delays. The interleaver and de-interleaver are partitioned into submodules that would allow parallel access and this enables reads and writes of multiple LLRs in one clock cycle.

7. DECODER PERFORMANCE

The SCPPM decoder for $M = 64$ is currently implemented on the Xilinx Virtex II-8000 FPGA, speed grade 4 (XC2V8000-4) which sits on a commercial off the shelf (COTS) Nallatech BenDATA WS board. The resource utilization is given in Table 2. Other hardware overheads are incurred by the interleavers, interface circuitry, and interface memory. The $\overset{*}{\max}$ Look-Up Tables (LUTs) are implemented as ROMs using Xilinx internal distributed RAM. This was made possible by the small number of entries in the LUT. The channel symbol memory, β-storage memory, and interleaver LUT's are all implemented using Xilinx internal, dual-ported BRAM. The equivalent gate count for our design as reported by the Xilinx place and route tool is 6.5 million. On the grade 4 part, a maximum clock speed of 23 MHz can be obtained which translates into a throughput of 1.23 Mbps based on an average of 7 decoding iterations. We can increase the data rate by using more advanced parts. On a grade 5, a clock speed of 26 MHz and throughput of 1.39 Mbps can be met and on a Virtex II-Pro FPGA, a clock speed of 28 MHz and throughput of 1.5 Mbps can be achieved.

Potential improvements exist in using a window-based BCJR, efficient scheduling algorithms, and next generation Virtex IV FPGAs. Calculations show that the enhanced design can in principle run at 24 Mbps per slice and this will allow an aggregate 50 Mbps SCPPM decoder implementation that requires only 3 FPGAs.

8. SUMMARY

We demonstrated an FPGA implementation of a Serial Concatenated Pulse Position Modulation (SCPPM) decoder for Deep Space Laser Communication. With optimizations, our decoder can achieve a throughput of 50 Mbps using 3 FPGAs and perform within 0.9 dB from the Shannon capacity at a WER of 10^{-4}.

REFERENCES

1. CCSDS, "Telemetry channel coding." Consultative Committee for Space Data Systems Standard 101.0-B-6, Oct. 2002.
2. C. Berrou and A. Glavieux, "Near optimum error-correcting coding and decoding: Turbo Codes," *IEEE Trans. Comm.,* vol. 44, pp. 1261-1271, Oct. 1996.
3. S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "A soft-input soft-output maximum a posteriori (MAP) module to decode parallel and serial concatenated codes," *The Telecom. and Data Acquisition Progr. Rep.,* vol. 42, pp. 1-20, Nov. 1996, JPL.
4. J. Hamkins, "The Capacity of APD-detected PPM," *JPL Telecommunications and Mission Operations Progress Report,* vol. 42-138, Aug. 1999.
5. J. Hamkins and J. Ceniceros, "The Capacity of Avalanche Photodiode-Detected Pulse Position Modulation," in *Proceedings of SPIE,* vol. 3932, (San Jose, CA), pp. 90-101, 2000.
6. B. Moision and J. Hamkins, "Low complexity serially concatenated coding for the deep space optical channel," to appear in *JPL Interplanetary Network Progress Report,* Feb. 2005.
7. J. Sun and O. Y. Takeshita, "Interleavers for Turbo Codes Using Permutation Polynomials Over Integer Rings," *IEEE Trans. Info. Theory.* vol. 51, pp. 101-119, Jan 2005.
8. R. J. Barron and B. Robinson, "Recursive polynomial interleaver algorithm." MLCD project Memo, Sept. 2004.
9. B. Moision, M. Srinivasan, and C. Lee, "Sequence detection for the optical channel in the presence of ISI." JPL Inter-Office Memo, June 2004.
10. 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. 20, pp. 284-287, March 1974.
11. W. E. Ryan, "A turbo code tutorial." Available on-line at http://www.ece.arizona.edu/ryan/turbo2c.ps.gz, 1997.
12. A. J. Viterbi, "An intuitive justification and a simplified implementation of the MAP decoder for convolutional codes," *IEEE J. Select. Areas Comm.*, vol. 16, pp. 260-264, Feb. 1998.
13. M. Barsoum and B. Moision, "Method and apparatus for fast digital turbo decoding for trellises with parallel edges." JPL Tech. Rep., July 2004.
14. G. Montorsi and S. Benedetto, "Design of fixed point iterative decoders for concatenated codes with interleavers," *IEEE J. Select. Areas Comm.,* vol. 19, pp. 871-882, May 2001.
15. T. Miyauchi, *et al.* "High-performance programmable SISO decoder VLSI implementation for decoding turbo codes," *IEEE Global Telecomm. conf.*, vol. 1, 25-29 Nov. 2001.