Implementation of a Coded Modulation for Deep Space Optical Communications

Michael K. Cheng, Bruce E. Moision, Jon Hamkins, and Michael A. Nakashima
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, 91109-8099
Email: {mkcheng, bmoision, jhamkins, mikenaka}@jpl.nasa.gov

***Abstract*— We present a field programmable gate array (FPGA) implementation of a turbo-like decoder for a serially concatenated pulse-position modulation (SCPPM) code. NASA developed this coded modulation scheme for deep space communications from Mars. Under a nominal mission condition, the SCPPM coded system can operate within a one dB signal energy gap from capacity.**

The structure of SCPPM makes direct application of the conventional turbo decoding algorithm very inefficient. Here, we describe techniques to increase the throughput and performance of a hardware SCPPM decoder. Using our optimizations, we demonstrate a 6 mega-bits per second (Mbps) decoder realization on a single FPGA. Extension to a higher data rate decoder using multiple FPGAs is readily achievable. Similar codes designed for the optical channel can benefit from our optimization techniques.

I. Introduction

Communication over deep-space is difficult. Communications beams spread as the square of the distance between the transmitter and the receiver. For example, geosynchronous Earth orbit (GEO) satellites are about 40,000 kilometers (km) in altitude and the average Mars-Earth distance is 80 million km. Therefore, the extra distance that a communication beam would have to travel from Mars to Earth would make data transfer 4 million times more difficult than from a GEO satellite to Earth. The signal power required to meet this extra effort and to cover this distance squared loss is greater than 66 dBs!

One way to increase the transmission rate from deep-space is through the use of more powerful transmit and receive antennas. However, this comes at a cost in increased antenna sizes which makes realization impractical. Another way is to communicate using frequencies much higher than radio frequency (RF) such as that of optical signals. Beams at higher frequency are more directionally concentrated and this allows a more efficient reception of the transmit energy [1, Ch. 1].

NASA's legacy error-correcting code (ECC) design for RF communication is the concatenation of an inner convolutional code and an outer Reed-Solomon (RS) code [2]. Decoding is performed in one pass utilizing hard bit-decisions. The discovery of turbo codes [3] and their suboptimal but effective low-complexity iterative decoding provided NASA a new code family with improved coding gains. NASA's first use of turbo codes is on the Messenger spacecraft launched in August of 2004.

An efficient ECC design for the deep space optical channel is the serial concatenation of an inner high-order modulation code and an outer convolutional code, namely serially concatenated pulse-position modulation or SCPPM. [4]. We may approximate true ML decoding while limiting the SCPPM decoder complexity by iteratively decoding the modulation and the ECC. This is in fact the "turbo" principle and more details can be found in [5].

This article is a companion to [6] and focuses on design issues critical to hardware implementation that are often not addressed in a high-level design. We also present novel techniques that optimize the SCPPM hardware decoder. The organization is as follows: in Section II we provide a model of the optical communications channel. In Section III, we give an overview of the SCPPM code and its decoding algorithm. In Section IV, we discuss some of the challenges associated with hardware implementation of the SCPPM decoder and describe our efficient approaches in detail. In Section V, we present a fast prototype decoder along with its hardware resource usage and error rate performance.

II. System Description

We consider an optical communications system that uses direct photon detection with a high-order pulse-position modulation (PPM) [1, Ch. 1.2]. An M-order PPM modulation uses a time interval that is divided into M possible pulse locations, but only a single pulse is placed into one of the possible positions. The position of the pulse is determined by the information to be transmitted. A diagram of the optical communications system in discussion is shown in Fig. 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 PPM symbols. At the receiver, light is focused on a detector that responds to individual photons as illustrated in Fig. 2. For each photon sensed, the detector produces a band-limited waveform for input to the demodulator. This waveform is used to estimate the photon count, k_i, within each slot i. On the Poisson channel, a nonsignaling slot has average photon count n_b and a signaling slot has average count $n_s + n_b$ so that the likelihood ratio of slot i is given by

$$LR(k_i) = e^{-n_s}\left(1 + \frac{n_s}{n_b}\right)^{k_i}. \quad (1)$$

More on the receiver design can be found in [7].

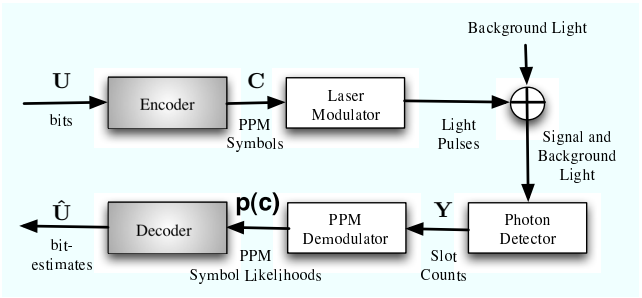

Fig. 1. An optical communication system.

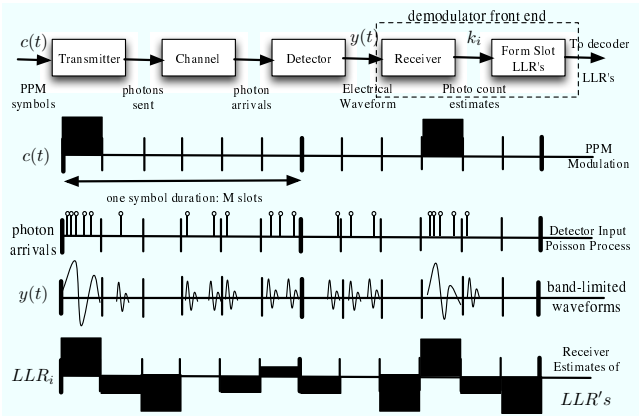

Fig. 2. From PPM symbols to decoder inputs.

III. The Serially Concatenated Pulse-Position Modulation (SCPPM) code

The SCPPM encoder, shown in Fig. 3, consists of an outer $\left(3, \frac{1}{2}\right)$ convolutional code, a polynomial interleaver, and an inner accumulate PPM (APPM) code. The trellis that describes the inner code consists of 2 states and $M/2$ parallel branches between connecting states.

A high level block diagram of the SCPPM decoder is illustrated in Fig. 4. The symbol I indicates input to the constituent decoders and O indicates output. The inner decoder operates on the APPM code and the outer decoder operates on the convolutional code. For each code trellis, the Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm [8] is used to compute the a-posteriori log-likelihood ratios (LLRs) from a-priori LLRs by traversing the trellis in forward and backward directions. Extrinsic information (the difference between the a-posteriori and a-priori LLRs) is exchanged in iteration rather than the

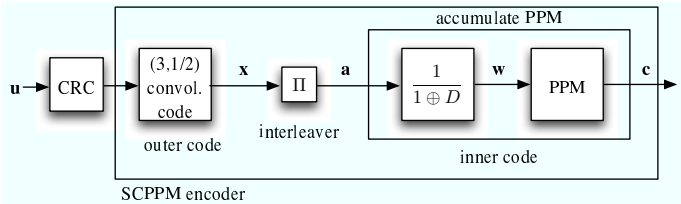

Fig. 3. The SCPPM encoder.

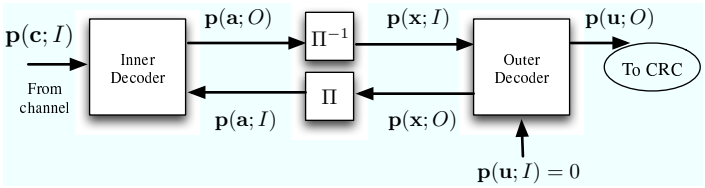

Fig. 4. The SCPPM decoder.

a-posteriori LLRs to reduce undesired feedback. More on the SCPPM code and its decoding algorithm can be found in [4].

IV. Hardware Implementation

We now discuss design considerations that arise in implementing the SCPPM decoder on a field programmable gate array (FPGA). These hardware level details, often not addressed in a top-level design, have an impact on decoder error and throughput performance.

A. Metric Quantization

In a fixed-point implementation, numbers are represented as integers in two's complement form. The quantization parameters are bit width w, base b, and precision p. We set $b = 2$. The integer representation of a floating-point number f is given by

$$f_q = \min\left(\max\left(\text{round}\left(f \cdot 2^p\right), -2^{w-p-1}+1\right), 2^{w-p-1}-1\right). \tag{2}$$

To minimize hardware cost, we select the best pair (p, w) that maintains an acceptable loss between simulated floating-point and fixed-point decoder performance. In this selection procedure, we quantize the channel LLRs and all decoder metrics using the same w and p.

1) Binary Precision: To determine p, we set w sufficiently large as render effects of dynamic range negligible. With our channel model and $p = 2$, the loss in signal energy is less than 0.2 dB and with $p = 3$ the loss is less than 0.1 dB [9]. We use three bits to represent binary precision.

2) Dynamic Range: To determine w, we set $p = 3$ and reduce w until the performance loss becomes unacceptable. An error floor occurs if w is too small. For our setup, using 5 bits for w satisfies our error floor requirement [9]. We use a total of eight bits: five-bit dynamic range and three-bit binary precision for quantization.

B. Log-Domain Decoding

Each constituent decoder applies the BCJR algorithm to the trellis that describes the corresponding code. Operations are performed in the log-domain to avoid multiplications which are costly to implement in hardware. This approach is known as log maximum a-posteriori (log-MAP) decoding [10]. Each log sum of exponentials can be expressed as the max of the exponents plus an adjustment term. This operation is known as the maxstar function:

$$\overset{*}{\max}\,(x, y) \triangleq \ln\left(e^x + e^y\right) = \max(x, y) + \ln\left(1 + e^{-|x-y|}\right). \tag{3}$$

The adjustment term can be precomputed and stored in a lookup table to reduce complexity at an increase in memory usage. We can also ignore the adjustment term entirely to save on memory – this approach is known as max log-MAP decoding. Some of the loss incurred from this approximation can be recovered by scaling the extrinsic information that is passed between the inner and outer decoder [11].

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

To reduce complexity, we implement the $\overset{*}{\max}$ operation as a Look-Up-Table (LUT). Montorsi and Benedetto [12] suggested a way of generating the fixed-point $\overset{*}{\max}$ LUT with m entries, where

$$m = \left\lceil -2^p \cdot \ln\left(e^{2^{-(p+1)}} - 1\right)\right\rceil. \tag{4}$$

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

$$v(\delta) = \text{round}\left(\ln\left(1 + e^{-\delta/2^p}\right) \cdot 2^p\right). \tag{5}$$

Calculating $\overset{*}{\max}\ (x, y)$ in fixed-point representation is therefore done by

$$\overset{*}{\max}\ (x_q, y_q) = \max(x_q, y_q) + v(|x_q - y_q|). \tag{6}$$

D. Fast Modulo Normalization

The BCJR algorithm consists of traversing the code trellis and updating a set of state and branch metrics. Since these metrics are represented by finite bit width variables in two's complement form, we would need to normalize and clip the results of these update operations so that the values do not overflow. However, the clipping operation requires a comparison with the maximum allowed value and this increases the path delay in a circuit. It turns out that we can avoid the need to normalize and clip as long as the quantization bit width is sufficient to account for the maximum differences between any two possible metric values [12], [13].

In modular arithmetic, a metric m_j is mapped into its modulo metric

$$\bar{m}_j \equiv \left(\left(m_j + \frac{C}{2}\right) \bmod C\right) - \frac{C}{2} \tag{7}$$

so that $-\frac{C}{2} \le \bar{m}_j < \frac{C}{2}$. This can be visualized by wrapping the real number line around a circle with circumference C. Going around the circle in a counter clockwise direction traverses a path in increasing magnitude and going around the circle in a clockwise direction traverses a path in decreasing magnitude.

For any two real numbers m_i, m_j such that their absolute difference is bounded by some finite value, that is $\Delta = |m_i - m_j| < \frac{C}{2}$, their modular difference $|\overline{\bar{m}_i - \bar{m}_j}|$ equals their actual difference $|m_i - m_j|$. Proofs are given in [13]. A general description of modulo metric normalization is given in Fig. 5. The angle α is the result of two's complement subtraction of $\bar{m}_2$ from $\bar{m}_1$. We know that $m_1 < m_2$ because

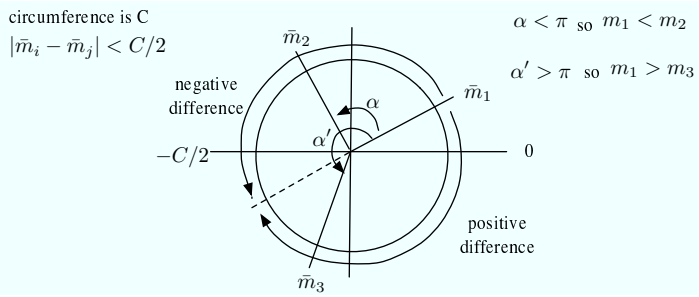

Fig. 5. The idea of modulo metric normalization. The reference metric is $\bar{m}_1$.

$\alpha < \pi$ and its sign bit is 1. The angle α' is the result of two's complement subtraction of $\bar{m}_1$ and $\bar{m}_3$. We know that $m_1 > m_3$ because $\alpha' \ge \pi$ and its sign bit is 0.

E. Parallel Trellis Edges and Partial Statistics

The trellis that describes the inner accumulate-PPM (APPM) code contains many parallel edges. To efficiently handle this large number of parallel edges, Barsoum and Moision [4] developed a method of grouping the many trellis edge calculations per stage into one, and this combined value can be computed in a pipeline. We use notations that are standard in description of the BCJR algorithm. An edge e connects an initial state $i(e)$ with a terminal state $t(e)$. The backward recursion log-domain state metric β for state s and stage k is computed as [14]:

$$\beta_k(s) = \overset{*}{\max}_{\tilde{s}\in\{s,\bar{s}\}}\{\beta_{k+1}(\tilde{s}) + \gamma'_{k+1}(s,\tilde{s})\}. \tag{8}$$

The log-domain edge metrics are calculated as

$$\gamma'_k(s,\tilde{s}) = \overset{*}{\max}_{e:i(e)=s,t(e)=\tilde{s}}\{\gamma_k(e)\}. \tag{9}$$

Since the γ'_ks, or we refer to as "Super Gammas", are not a function of a recursively computed quantity, they may be precomputed via a pipeline and this reduces the edge computation time per trellis stage to one clock cycle. The α's are formed similarly.

To reduce the channel likelihood storage requirements, we may discard the majority of the channel likelihoods and use partial statistics [15]. This may be accomplished by processing 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.

F. Interleaver Design

The interleaver used is characterized by a second order polynomial $f(j) = aj + bj^2$. The bit position j is mapped to the position $[f(j)]_N$ where $[\cdot]_N$ is the mod N operation. Let us factor the codeword length N as products of primes, that is, $N = p_1^{j_1} p_2^{j_2} \cdots p_\ell^{j_\ell}$. Any polynomial [16] with $b = p_1 p_2 \cdots p_\ell$ and a set to a number that does not have p_1,p_2,$\cdots$, or p_ℓ

as a factor is a candidate interleaver. The mapping for the $(j+i)$th interleaver position can be expressed as a function of the current interleaver position j:

$$[f(j+i)]_N = [f(j) + g(i,j)]_N \tag{10}$$

where $g(i,j) = 2ijb + i(a+bi)$. This property enables an algorithmic implementation that does not require the mapping to be precomputed and stored [14]. For SCPPM parameters, we found the polynomial $f(j) = 11j + 210j^2$ to have good performance.

G. Decoder Windowing

The inner trellis consists of $N/\log_2 M$ symbols or segments. The outer code trellis is a rate 1/2 code and has $N/2$ segments. For PPM orders M greater than 4, the outer code trellis will contain more segments than the inner code trellis. If a straightforward scheduling is used to traverse the two trellises, the inner decoder will have to wait longer for the outer decoder to complete a trellis pass. It is advantageous to have both decoders complete an iteration in the same amount of time. The latency in this case is reduced because the wait time of the inner decoder is reduced. To do so, we partition the outer code trellis into distinct windows and apply a window-based BCJR algorithm to decode the windows in parallel.

For SCPPM parameters and $M = 64$, the outer decoder is windowed by three. In this scenario, we observed through simulations that no warmup windows are required to obtain a performance close to that of the non-windowed decoder.

H. Cyclic Redundancy Check

A CRC can be used together with iterative turbo decoding to flag codeword errors or to stop decoding iterations. In windowed-based turbo decoding, the bits to be input to the CRC are generated in parallel more than one at a time. Therefore, the conventional serial input linear feedback shift register (LFSR) circuit that implements a CRC needs to be modified to handle this parallelism.

Let us write a length k binary message block $\mathbf{m} = (m_{k-1}, m_{k-2}, \cdots, m_0)$, that is to be protected by a CRC, in polynomial form:

$$m(x) = m_{k-1}x^{k-1} + m_{k-2}x^{k-2} + \cdots + m_0 \tag{11}$$

Let the length n CRC protected codeword be $\mathbf{c} = (c_{n-1}, c_{n-2}, \cdots, c_0)$ or

$$c(x) = c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \cdots + c_0 \tag{12}$$

and the CRC generator be

$$g(x) = g_{n-k}x^{n-k} + \cdots + g_0. \tag{13}$$

The CRC polynomial $r(x)$ is calculated by first shifting the message polynomial left by $n-k$ positions and then by taking the modulo $g(x)$ operation

$$r(x) = R_{g(x)}\left[m(x) \cdot x^{n-k}\right], \tag{14}$$

where $\deg[r(x)] < n-k$. The codeword polynomial is expressed as $c(x) = m(x) \cdot x^{n-k} + r(x)$.

To verify the CRC of a codeword block $\hat{c}(x) = c(x) + e(x)$ that might be corrupted by an error polynomial $e(x)$, we compute $R_{g(x)}[\hat{c}(x)] = R_{g(x)}[e(x)]$. Therefore, if the remainder is zero, the CRC passes and the error polynomial is zero. If the remainder is nonzero, then the codeword is corrupted. Note that we won't be able to construct the error polynomial $e(x)$ from the CRC remainder $R_{g(x)}[e(x)]$.

In windowed-based turbo decoding, the output bit streams to be fed into the CRC are generated in parallel. We describe how a CRC circuit can be modified to handle this parallelism. Let the code trellis be partitioned into j distinct windows. The codeword polynomial can be written as

$$c(x) = c_1(x)x^{s_1} + c_2(x)x^{s_2} + \cdots + c_j(x). \tag{15}$$

We can then write the check polynomial as

$$\begin{aligned} R_{g(x)}[c(x)] &= R_{g(x)}[c_1(x)x^{s_1} + c_2(x)x^{s_2} + \cdots + c_j(x)] \\ &= R_{g(x)}\left[R_{g(x)}[c_1(x)x^{s_1}] + R_{g(x)}[c_2(x)x^{s_2}]\right. \\ &\quad \left. + \cdots + R_{g(x)}[c_j(x)]\right] \\ &= R_{g(x)}\left[R_{g(x)}[c_1(x)\kappa_1(x)]\right. \\ &\quad \left. + R_{g(x)}[c_2(x)\kappa_2(x)] + \cdots + R_{g(x)}[c_j(x)]\right], \end{aligned} \tag{16}$$

where $\kappa_i = R_{g(x)}[x^{s_i}]$, $i = 1, 2, \cdots, j-1$, and each $\kappa_i(x)$ can be pre-calculated. The CRC LFSR circuit for the window-based decoder will consist of both feed-forward and feedback tap connections. The feed-forward taps are given by the XOR of $\kappa_i(x)$'s and the feedback taps are given by the generator $g(x)$.

V. Decoder Specifications and Performance

The SCPPM decoder for $M = 64$ and $N = 15120$ is currently implemented on a Xilinx Virtex II-8000 FPGA part, speed grade 4 (XC2V8000-4), which sits on a Nallatech BenDATA-WS board. The memory requirement is reduced by taking only the top 8 channel LLRs as decoder input. We have implemented two versions of the decoder: The first is the log-MAP decoder with clipping and normalization circuits. The second is the max log-MAP decoder with fast modulo normalization and windowing. The outer code trellis is windowed by three. The total FPGA resource utilization as well as a breakdown by modules for the two decoders is given in Table I. The miscellaneous blocks that consume resources are the circuitries and memories instantiated for the interleaver, deinterleaver, and FPGA interface. The $\overset{*}{\max}$ lookup tables (LUTs) for the log-MAP decoder are realized as read-only memories (ROMs) using Xilinx internal distributed random access memory (RAM). The channel symbol memory, state metric storage memory, and interleaver LUTs are all implemented using Xilinx internal, dual-ported block RAMs (BRAMs).

The max log-MAP decoder, with all of the proposed throughput optimizations, supports a maximum clock rate of 60 MHz and a data rate of 6.4 Mbps based on 7 average iterations .

The decoder performance is shown in Fig. 6. The word error rate (WER) is plotted versus n_s, the average number of

log-MAP	Used/Total	Utilization	Inner	Outer	Misc.
BRAM	101/168	60%	19%	9%	32%
Slices	30174/46592	64%	52%	6%	6%

max log-MAP	Used/Total	Utilization	Inner	Outer	Misc.
BRAM	158/168	89%	19%	30%	40%
Slices	24587/46592	53%	32%	15%	6%

TABLE I

SCPPM DECODERS ON THE VIRTEX-II 8000 FPGA.

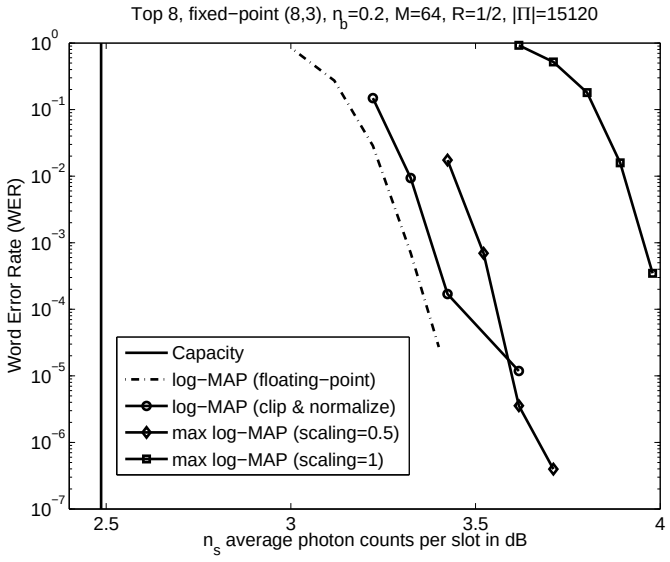

Fig. 6. SCPPM decoder performance on the Poisson channel.

signal photos per PPM signal slot in dB. The average noise photons per slot is $n_b = 0.2$. Each codeword consists of 7560 information bits. A word error is declared when the decoder decision could not converge to the correct codeword in the maximum number of allowed iterations which is set at 32. Out of the 7560 bits, 2 bits are used to terminate the trellis and 22 bits are used for CRC. The CRC polynomial is $x^{22} + x^5 + x^4 + x^3 + 1$ and has an undetected word error probability of approximately $7 \cdot 2^{-22} = 1.67 \times 10^{-6}$ assuming 7 average iterations. To reduce the undetected rate, the decoder runs a minimum number of iterations first before validating the CRC. In doing so, the undetected probability is lowered to roughly the product of the frame loss rate and 1.67×10^{-6}, a very small value.

We make the following observations in the performance plot. Fixed-point implementation (circle-line) has a 0.1 dB loss compared to the floating-point decoder (dashed-line). Clipping and and normalization of the state metrics led to a floor at 10^{-5}. Max log-MAP decoder with fast modulo normalization (square-line) has a 0.6 dB loss compared to log-MAP decoding (circle-line). Max log-MAP decoder with a scaling of the extrinsic information by 0.5 (diamond-line) recovers 0.4 dB out of the 0.6 dB lost.

VI. SUMMARY

NASA developed a serially concatenated pulse position modulation (SCPPM) coding scheme for deep space optical communications. The structure of SCPPM makes direct application of conventional turbo decoding inefficient. In this work, we discussed a set of criteria important in hardware implementation of the SCPPM decoder. These criteria, usually not addressed in a top-level design, have an impact on the decoder cost and performance. To meet the challenges set by these requirements, we developed novel optimization techniques for hardware realization. By applying these techniques, we demonstrated a 6 Mbps SCPPM decoder on an FPGA that performs with in 1.2 dB of the Shannon capacity.

REFERENCES

[1] H. Hemmati, *Deep Space Optical Communications*. John Wiley & Sons Inc., 2006. ISBN 0-470-04002-5.

[2] CCSDS, "Telemetry channel coding." Consultative Committee for Space Data Systems Standard 101.0-B-6, Oct. 2002.

[3] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," in *Proc. IEEE Int. Conf. Commun.*, vol. 2, (Geneva, Switzerland), pp. 1064–1070, IEEE, May 1993.

[4] B. Moision and J. Hamkins, "Coded modulation for the deep space optical channel: serially concatenated PPM," *JPL Interplanetary Network Progress Report*, vol. 42-161, May 2005.

[5] 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.

[6] B. E. Moision, J. Hamkins, and M. K. Cheng, "Design of a coded modulation for deep space optical communications," in *UCSD first workshop on Information Theory and its applications*, (San Diego, CA), Univ. Calif. San Diego, Feb. 2006.

[7] K. J. Quirk and L. B. Milstein, "Optical PPM with sample decision photon counting," in *Proc. IEEE Global Telecom. Conf.*, (St Louis, MO, USA), IEEE, 2005.

[8] 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.

[9] M. Cheng, M. Nakashima, J. Hamkins, B. Moision, and M. Barsoum, "A field-programmable gate array implementation of the serially concatenated pulse-position modulation decoder," *JPL Interplanetary Network Progress Report*, vol. 42-161, May 2005.

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

[11] P. H. Wu and S. M. Pisuk, "Implementation of a low complexity, low power, integer-based turbo decoder," in *Proc. IEEE Global Telecom. Conf.*, vol. 2, (San Antonio, Texas), pp. 946–951, IEEE, 2001.

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

[13] A. P. Hekstra, "An alternative to metric rescaling in Viterbi-decoders," *IEEE Trans. Commun.*, vol. 37, pp. 1220–1222, Nov. 1989.

[14] M. K. Cheng, B. E. Moision, J. Hamkins, and M. A. Nakashima, "Optimizations of a turbo-like decoder for deep space optical communications," in *JPL Interplanetary Network Progress Report*, 2006. Submitted for review.

[15] B. Moision and J. Hamkins, "Reduced complexity decoding of coded-pulse modulation using partial statistics," *JPL Interplanetary Network Progress Report*, vol. 42-161, May 2005.

[16] J. Sun and O. Y. Takeshita, "Interleavers for turbo codes using permutation polynomials over integer rings," *IEEE Trans. Inform. Theory*, vol. 51, pp. 101–119, Jan. 2005.