A Field-Programmable Gate Array Implementation of the Serially Concatenated Pulse-Position Modulation Decoder

M. K. Cheng,[1] M. A. Nakashima,[2] J. Hamkins,[1] B. E. Moision,[1] and M. Barsoum[1]

We describe the development of a hardware turbo decoder on a field-programmable gate array for high-speed free-space laser communications. This system will be used by NASA's Mars Laser Communications Demonstration project on the Mars Telecommunication Orbiter, the first use of high-speed laser communication from deep space. The error-correction code design has been shown to perform within 0.9 dB of the Shannon capacity on a nominal mission operating condition. The implemented decoder achieves a throughput of 1.23 mega-bits per second (Mbps). We outline potential improvements on the current design that can lead to a 50-Mbps decoder.

I. Introduction

The Mars Laser Communication Demonstration (MLCD) is planned to be the first demonstration of optical communications from a satellite in deep space to a ground receiver terminal on Earth. Nominal downlink (from spacecraft to Earth) data rates of over 1 Mbps (mega-bits per second) are desired. However, with certain operating conditions, communication at 50 Mbps can be demonstrated.

NASA's legacy error-correction code (ECC) design for radio frequency (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 due to the large constraint length of the inner code. The discovery of turbo codes [2,3] 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 its 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 decibels in performance by decoding the ECC and modulation jointly as a single code relative to decoding them

[1] Communications Architectures and Research Section.

[2] Flight Systems Electronics Section.

The research described in this publication was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

independently. An exact maximum-likelihood (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 [4].

Moision and Hamkins described in [5] their approach in designing the serially concatenated pulse-position modulation (SCPPM) code for MLCD. They also outlined the soft-in, soft-out (SISO) decoding algorithm used by the decoder. This article is a companion to [5]. We provide the implementation details that led to the hardware development of the SCPPM decoder on a field-programmable gate array (FPGA).

II. Decoder Architecture

A high-level block diagram of the SCPPM decoder is illustrated in Fig. 1. The symbol I indicates input to the constituent decoders, and O indicates output. The inner decoder operates on the modulation code, and the outer decoder operates on the convolutional code. Each code is described by a trellis. For each trellis, the Bahl–Cocke–Jelinek–Raviv (BCJR) algorithm [6] 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 a posteriori LLRs to reduce undesired feedback.

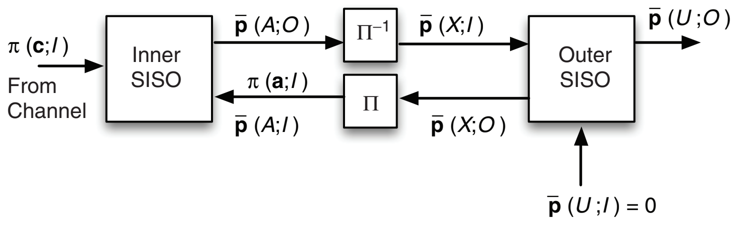
π (**c**;I)
From Channel
Inner SISO
$\bar{\mathbf{p}}$ (A;O)
Π^{-1}
$\bar{\mathbf{p}}$ (X;I)
Outer SISO
$\bar{\mathbf{p}}$ (U;O)
π (**a**;I)
$\bar{\mathbf{p}}$ (A;I)
Π
$\bar{\mathbf{p}}$ (X;O)
$\bar{\mathbf{p}}$ (U;I) = 0

Fig. 1. The SCPPM decoder.

A. The Log-Domain SISO Decoding

Each SISO module in the SCPPM decoder applies the BCJR algorithm to the trellis that describes the corresponding code. We use the notation from [5] and simply restate the calculation of the branch and state metrics inside the inner SISO module. To facilitate hardware realization, the metric computations are done in the log domain, which translates multiplications into additions and is less sensitive to round-off errors in fixed-point arithmetic.

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 $i\,(e)$ and a terminal state $t\,(e)$. For each edge e and stage k of the inner code trellis, the BCJR algorithm traverses the trellis in the forward direction to calculate the branch metric as

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

The term $\pi_k(\mathbf{c}; I)$ is the PPM symbol LLR provided by the channel, and the term $\pi_k(\mathbf{a}; I)$ is the a priori symbol LLR provided by the outer decoder. In the same forward trellis pass, the BCJR algorithm calculates a forward state metric for each state s and stage k as

$$\bar{\alpha}_k(s) = \ln \sum_{e:t(e)=s\in\mathcal{V}} \exp\left(\bar{\alpha}_{k-1}\big(i(e)\big) + \bar{\gamma}_k(e)\right) \tag{1}$$

The algorithm then traverses the trellis in the backward direction to calculate a backward state metric as

$$\bar{\beta}_k(s) = \ln \sum_{e:i(e)=s\in\mathcal{V}} \exp\left(\bar{\beta}_{k+1}\big(t(e)\big) + \bar{\gamma}_{k+1}(e)\right) \tag{2}$$

The output LLRs are a function of $\bar{\alpha}$'s, $\bar{\beta}$'s, and $\bar{\gamma}$'s. The outer SISO operates on the trellis that describes the outer code, using the same principle.

B. 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 [7]:

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

$$= \max\,(x,y) + \ln\left(1 + e^{-|x-y|}\right)$$

It is also noted that

$$\overset{*}{\max}\,(x,y,z) = \ln\left(e^x + e^y + e^z\right) \tag{4}$$

$$= \overset{*}{\max}\left(\overset{*}{\max}\,(x,y)\,,y\right)$$

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 III.C.

C. Simplified Computation of the Inner Modulation Code Trellis

The inner modulation code consists of an accumulator followed by a PPM mapper. The trellis that describes this accumulate-PPM (APPM) code contains many parallel edges, and this causes a bottleneck in the computation of the $\bar{\alpha}$'s and $\bar{\beta}$'s. To reduce this latency, Barsoum and Moision [8,9] developed a method of grouping the individual trellis edge calculations per stage into one, and this combined value can be computed in a pipeline.

III. Hardware Implementation

We now discuss some of the design decisions we made in implementing the SCPPM decoder on an FPGA.

A. Metric Quantization

In software, the decoder metrics $\bar{\alpha}$'s, $\bar{\beta}$'s, $\bar{\gamma}$'s, $\bar{\lambda}$'s, and channel likelihoods $\pi\left(\mathbf{c}, I\right)$'s are floating-point values. However, in hardware, these metrics are represented by fixed-point integers in two's complement form. We discuss the procedure used to determine the quantization bit width.

The input quantization parameters are available bit width w, base b, and precision p. Let f denote a floating-point number and q denote an integer. We represent all decoder variables (metrics) as quantized integers, that is,

$$q = \text{round}\left(f \cdot b^p\right) \tag{5}$$

where $b = 2$ but could be any number. We also clip the quantized variable at maximum and minimum allowable integer values, that is,

$$q \in \left[-2^{w-p-1} + 1, 2^{w-p-1} - 1\right] \tag{6}$$

The $\bar{\beta}$'s are stored in the FPGA block random-access memories (BRAMs) while all other variables are computed as needed. For a fixed performance loss, clipping of all decoder metrics will require an overall larger number of quantization bits than clipping only the $\bar{\alpha}$'s, $\bar{\beta}$'s, and channel LLRs. This larger quantization requirement will translate into a higher BRAM consumption in storing the $\bar{\beta}$'s and therefore can limit the number of decoders that are able to fit onto a single FPGA. Hence, we clip only the $\bar{\alpha}$'s, $\bar{\beta}$'s, and channel LLRs and allow the other metrics in the data path to grow without clipping. The $\bar{\alpha}$'s and $\bar{\beta}$'s are also normalized at every stage by subtracting the largest $\bar{\alpha}$ and $\bar{\beta}$ of that stage. Since the $\bar{\alpha}$'s and $\bar{\beta}$'s are both normalized and clipped, the remaining decoder metrics will not grow without bound. Moreover, limiting the number of clipping points can facilitate debugging because every clipping point adds a potential source of error. In the debugging process, each clipping point will have to be examined during traceback, whereas if most of the metrics are allowed to grow in the data path without being passed through a clipping circuit, the number of potential problem spots can be reduced significantly.

1. Binary Precision. To determine the number of bits required to represent binary precision, we compare the simulated decoder performance using varying p while choosing w large. The results are plotted versus word-error rate (WER) in Fig. 2(a). The parameters are the PPM order, M; the slot width in nanoseconds, T_s; the background noise in photons per slot, n_b; and the signal in photons per pulse, n_s. The number pair (w, p) in the legend indicates the total number of bits used and the number of bits used for binary precision. For example, $(18, 3)$ means 18 total bits used, 15 bits for dynamic range and 3 bits for binary precision. As expected, the decoding performance approaches that of floating point as p is increased. To minimize hardware cost, we select the least number of bits required to maintain an acceptable loss in fixed-point performance. With $p = 2$, the loss in signal energy is less than 0.2 dB, and with $p = 3$, the loss is less than 0.1 dB. We choose to use 3 bits to represent binary precision.

2. Dynamic Range. To determine the number of bits required to represent dynamic range, we fix the binary precision p to 3 and reduce the total number of bits used until the performance loss becomes unacceptable. The results are plotted versus WER in Fig. 2(b). Note the occurrence of an error floor when there is an insufficient number of bits used to represent the dynamic range. To meet a requirement of a WER floor below 10^{-4}, we select 5 bits to use for dynamic range. Our FPGA quantization scheme, therefore, uses a total of 8 bits: 5 bits for dynamic range and 3 bits for binary precision.

B. Partial Statistics

To reduce the channel-likelihood storage requirements of iterative decoding and the amount of data transfer between the decoder and receiver, we may discard the majority of the channel likelihoods [10], 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 photons. 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.

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

The natural log function is costly to realize in hardware. The $\overset{*}{\max}$ operation, therefore, is 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

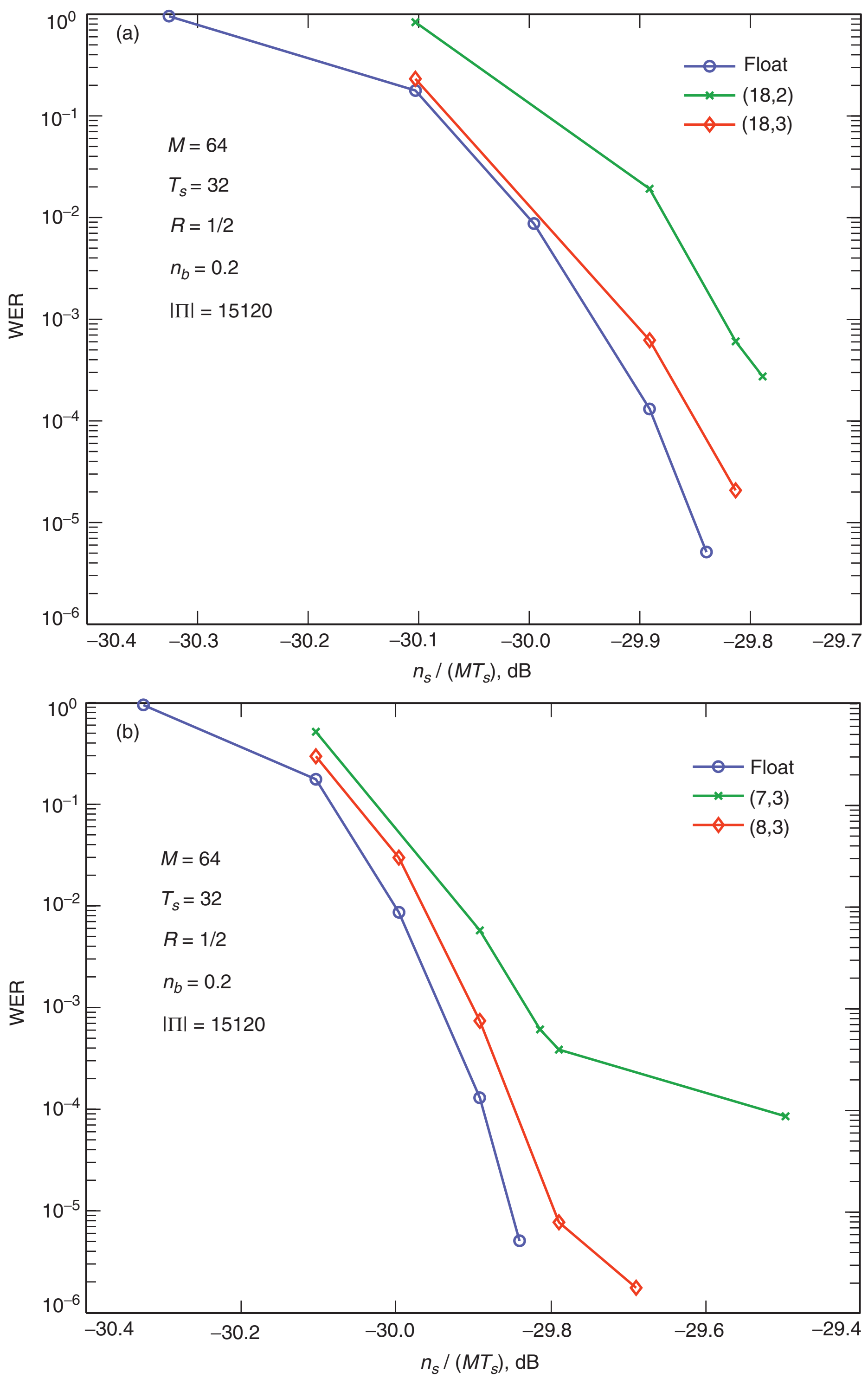

Fig. 2. To determine the quantization parameters, we simulate the decoder WER performance with varying (a) binary precision and (b) dynamic range for a typical operating point.

$$x_q = \begin{cases} -2^{w-p-1}+1 & \text{round}\left(x \cdot 2^p\right) \le -2^{w-p-1}+1 \\ \text{round}\left(x \cdot 2^p\right) & -2^{w-p-1}+1 < \text{round}\left(x \cdot 2^p\right) < 2^{w-p-1}-1 \\ 2^{w-p-1}-1 & 2^{w-p-1}-1 \le \text{round}\left(x \cdot 2^p\right) \end{cases} \tag{7}$$

Let the adjustment term in Eq. (3) be defined as

$$\Delta \triangleq \ln\left(1+e^{-|x-y|}\right) \approx \ln\left(1+e^{-\frac{|x_q-y_q|}{2^p}}\right) \tag{8}$$

Montorsi and Benedetto [11] suggested a way of generating the fixed-point $\overset{*}{\max}$ LUT with m entries, where m is the smallest positive integer that satisfies

$$\ln\left(1+e^{-m/2^p}\right) \le 2^{-(p+1)} \tag{9}$$

Solving for m, we have

$$m = \left\lceil -2^p \cdot \ln\left(e^{2^{-(p+1)}}-1\right)\right\rceil \tag{10}$$

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 calculated as

$$v\left(\delta\right) = \text{round}\left(\ln\left(1+e^{-\delta/2^p}\right) \cdot 2^p\right) \tag{11}$$

Calculating $\max^*\left(x,y\right)$ in fixed-point representation therefore is done by

$$\overset{*}{\max}\left(x_q,y_q\right) = \max\left(x_q,y_q\right) + v\left(|x_q-y_q|\right) \tag{12}$$

Using the example in [11], let $p = 3$; from Eq. (10), m is computed to be 22. The $\overset{*}{\max}$ LUT is then populated as seen in Table 1. If $\delta = |x_q - y_q| = 7$, then the $\overset{*}{\max}$ operation would return $\max\left(x_q,y_q\right)+3$. Any fixed-point number x_q can be converted back to a floating-point value by $x = x_q/2^p$.

Table 1. The $\overset{*}{\max}$ look-up table with $p = 3$.

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

D. A Fast Clipping Circuit

We use a fast clipping circuit to realize two's complement addition. If the result of the addition is larger (or smaller) than what the assigned bit width can represent, that number is clipped to the maximum (or minimum) possible quantized value given in Eq. (6). Conventional clipping circuits consist of comparators that use either an XOR tree or subtractions to determine overflow. Since some of the trellis metrics in our design are allowed to grow to a large width, propagation time through the carry out circuitry or levels of XOR gates can be significant. Our circuit requires no comparators, and this reduces propagation delays. A sample circuit for clipping the sum of two 3-bit numbers is illustrated in Fig. 3. Simple extensions can be made for wider inputs. For positive sums, the first AND gate is used and the other two ignored. The sum is clipped if the overflow bit is set. For negative sums, an inverted overflow bit indicates an overflow; this check is implemented in the second AND gate. However, we also need to check the condition of a negative eight; this occurs when both the sign bit and the overflow bit are set and the remaining bits are all zeros (indicated by the third AND gate.) To clip a sum of more than two terms, allow another overflow bit for each term and OR these extra overflow bits together (NAND for negative numbers) before feeding the result into the AND gates along with the original overflow bit. The inputs to the second AND gate are inverted.

E. Multi-Module Interleaver Design

The interleaver labeled as Π in Fig. 1 maps a bit in position x into a position $f(x)$. The SCPPM decoder uses a second-degree permutation polynomial of the form $f(x) = (ax + bx^2) \bmod N$, where N is the number of bits in a codeword block. In MLCD, N is selected to be 15120 to allow flexibility in the choice of inner and outer code rates. Using the ideas found in [12], Moision and Hamkins [5] set the

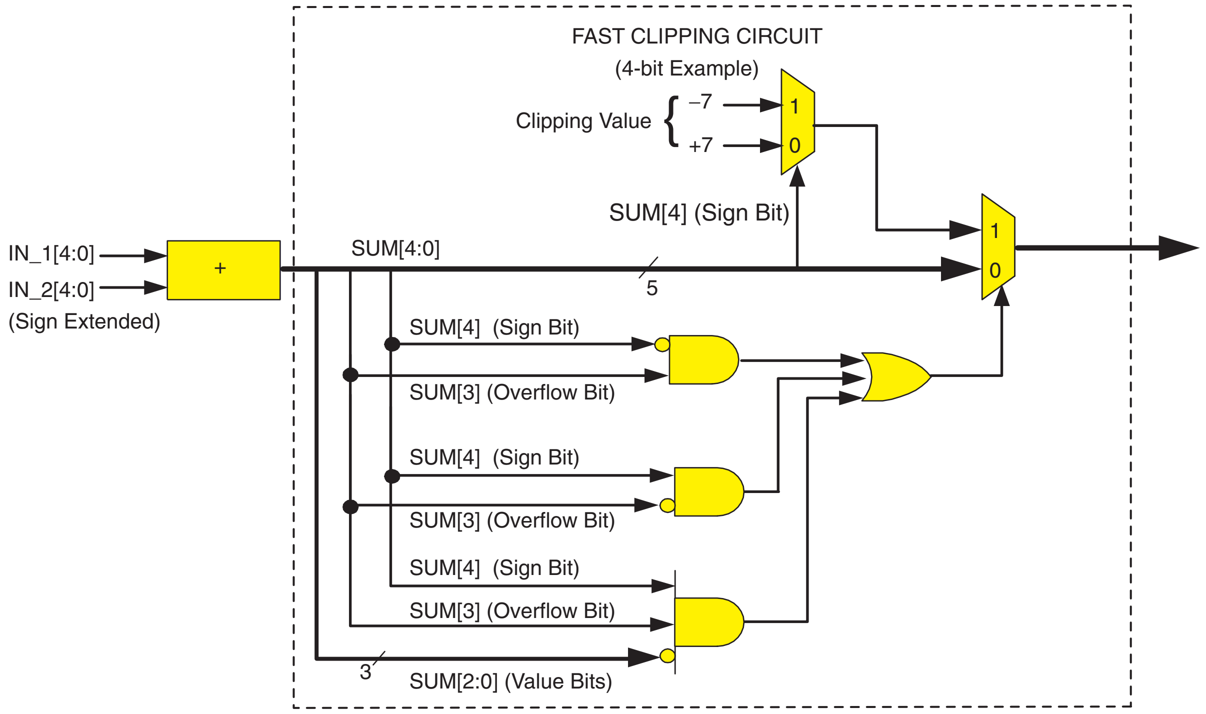

Fig. 3. A fast 3-bit clipping circuit.

coefficients of $f(x)$ to be $a = 11$ and $b = 210$. Moreover, Barron and Robinson[3] have demonstrated a recursive implementation of the second-degree permutation polynomial that requires only additions to facilitate hardware implementation.

For a PPM order of $M = 64$ or six bits per symbol, the inner decoder generates six LLRs each clock to be written into the de-interleaver memory. If a serial interface is adopted for the de-interleaver, it would take six clocks before a stage of LLR is stored. To reduce this latency from six clocks to one clock, we use a scheme in which a set of six LLRs is written into the de-interleaver sequentially in one row all at once, as illustrated in Fig. 4. The de-interleaver mapping is stored in the permuted address table. During outer decoding, the LLRs are read using the address table in permuted order two at a time. This is done by first picking the two (dual-ported BRAM) de-interleaver rows that contain the desired LLRs (in groups of six) and then selecting the correct entry out of the row read. It can be shown that $f(x) = 11x + 210x^2$ maps to positions that are at least six apart, so no two reads will come from the same row. Even if there is a read conflict, the dual-ported BRAM is designed to allow access to the same row.

The interleaver is implemented in the reverse manner as the interleaver and is illustrated in Fig. 5. That is, a stage of two LLRs is written with permuted address into the interleaver. The interleaver is divided into six dual-ported BRAMs, and the two LLRs are stored in one clock. If there is a write collision of the two LLRs to the same BRAM, the dual-port memory realization still allows the writes to complete in the same clock. We use a mapping to store the interleaved entries so that during reads the inner code simply indexes into the same row at each of the six BRAMs and reads six LLR entries simultaneously from the six distinct memory blocks.

The multi-module interleaver design reduces interleaver and de-interleaver access to one clock cycle per trellis stage.

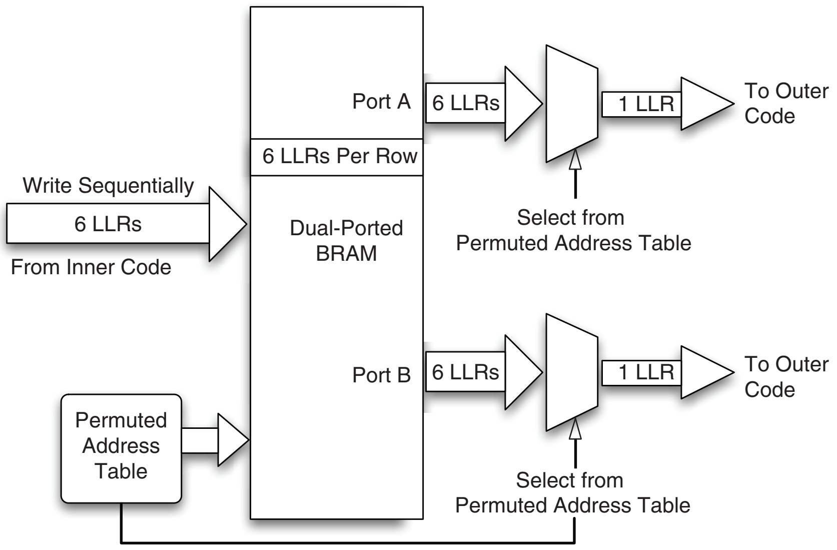

Fig. 4. Block diagram of the de-interleaver.

[3] R. J. Barron and B. Robinson, "Recursive Polynomial Interleaver Algorithm," MLCD Project Memorandum (internal document), Lincoln Laboratory, Massachusetts Institute of Technology, Lexington, Massachusetts, September 2004.

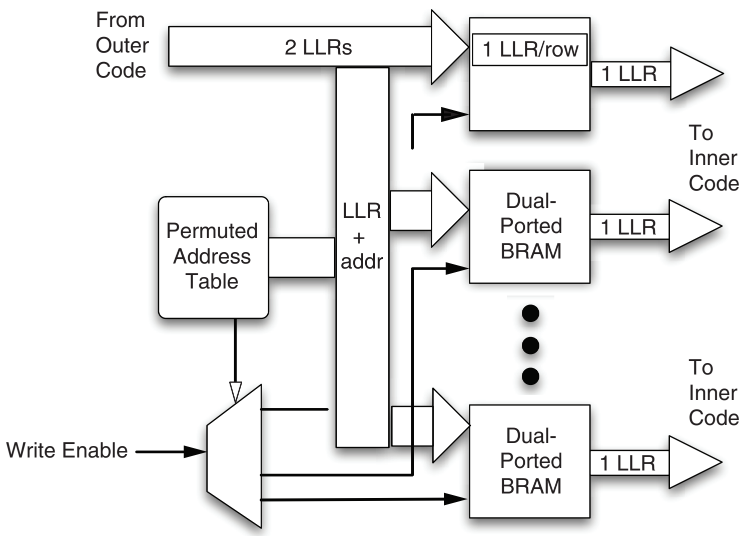

Fig. 5. Block diagram of the interleaver.

IV. Decoder Performance

The SCPPM decoder for $M = 64$ is currently implemented on a Xilinx Virtex II-8000 FPGA, speed grade 4 (XC2V8000-4), which sits on a Nallatech BenDATA-WS board. The resource utilization is given in Table 2. Blocks other than the inner and outer SISO modules that consume resources are the interleavers, interface circuitry, and interface memory. The $\overset{*}{\max}$ look-up tables are implemented as read-only memories (ROMs) using Xilinx internal distributed random access memory (RAM). This was made possible by the small number of entries in the LUT. The channel symbol memory, $\bar{\beta}$-storage memory, and interleaver LUTs 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 achieved, and on a Virtex II-Pro FPGA, a clock speed of 28 MHz and throughput of 1.5 Mbps can be achieved.

The FPGA decoder error performance for three MLCD operating points, nominal, best, and worst, are given in Figs. 6 through 8. To save simulation time, we stopped after approximately 10 word errors were obtained at high signal-to-noise ratios (SNRs). The slight increases in WER and BER at high SNRs for the nominal and worst cases are due to this small number of word errors. These error floors can be smoothed with longer simulation runs. The current decoder takes as input only the top eight slot statistics, and the remaining slots are set to the mean of a noise slot [10]. Extension to a full-statistics

Table 2. SCPPM decoder on the Xilinx Virtex II-800 FPGA.

Full decoder	Used/total	Utilization	Inner decoder	Outer decoder	Other blocks
BRAM	101/168	60%	19% of total resource	9% of total resource	32% of total resource
Flip flops	17311/93184	18%	16%	1%	1%
Slices	30174/46592	64%	52%	6%	6%

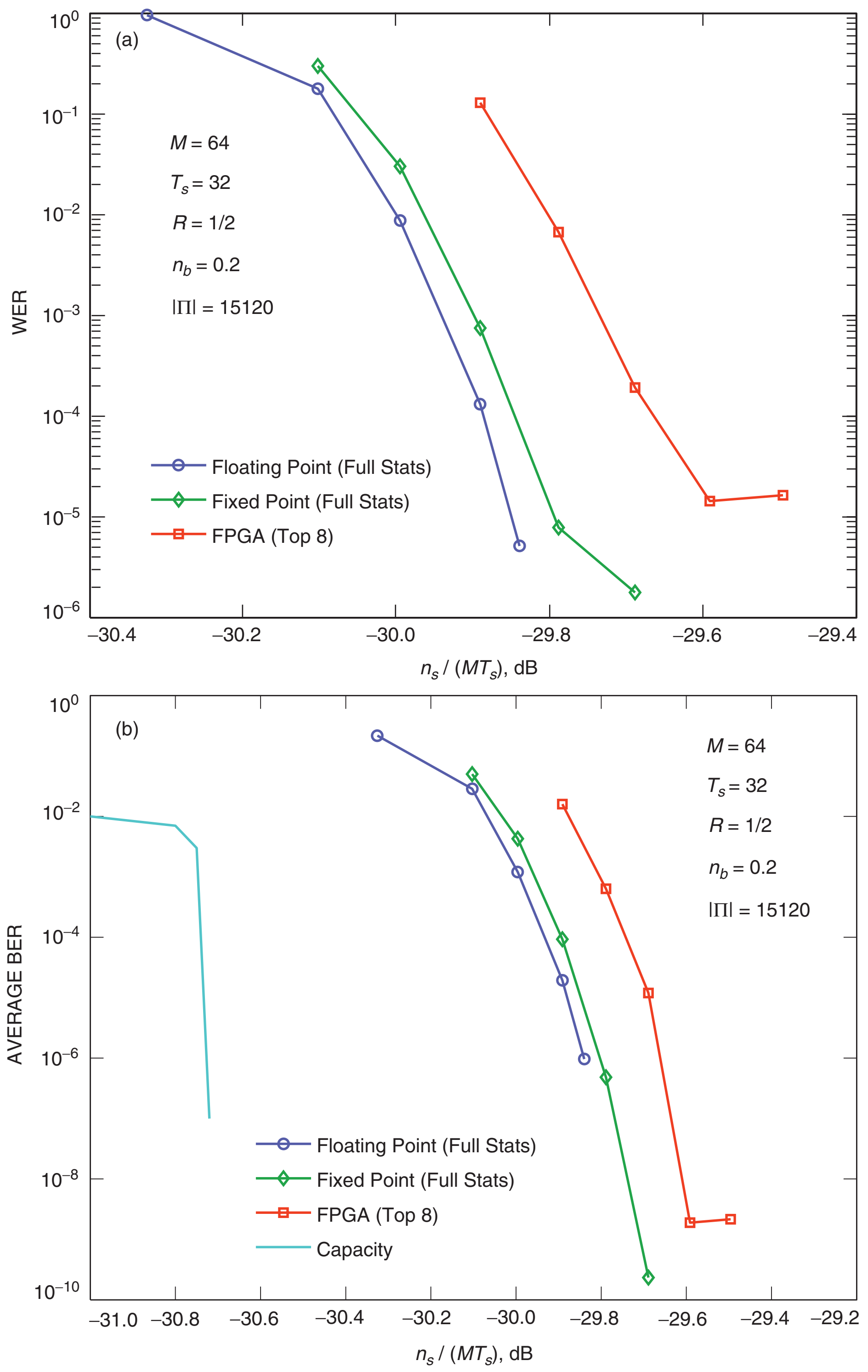

Fig. 6. Performance of the SCPPM FPGA decoder for the nominal operating point: (a) WER and (b) BER. An 8-bit quantization is used, of which 5 bits are for dynamic range and 3 bits are for fractional precision. Only the top 8 slot statistics are input into the decoder, and the remaining statistics are set to the mean of a noise slot. The floating-point performance uses all 64 channel statistics.

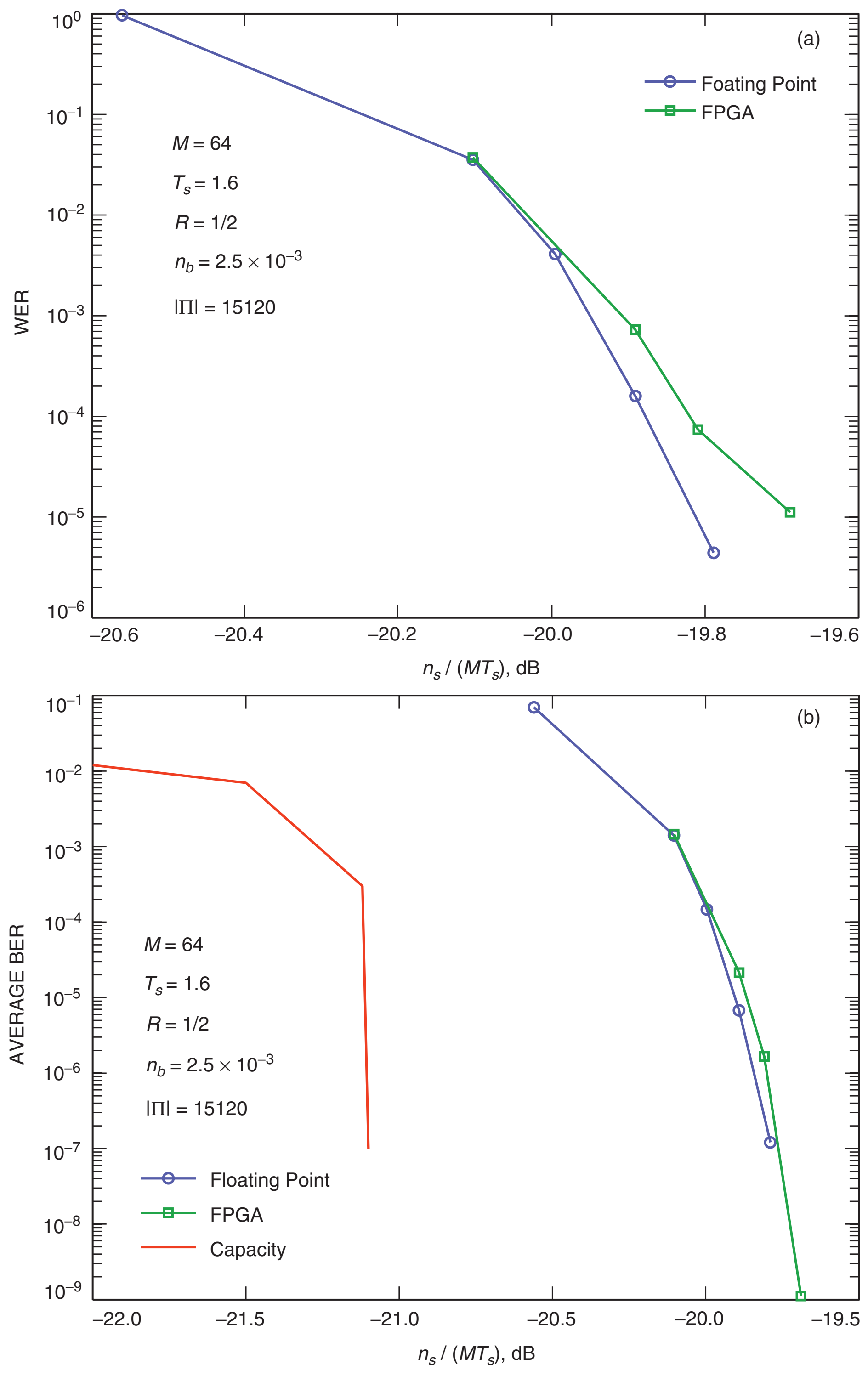

Fig. 7. Performance of the SCPPM FPGA decoder for the best-case operating point: (a) WER and (b) BER.

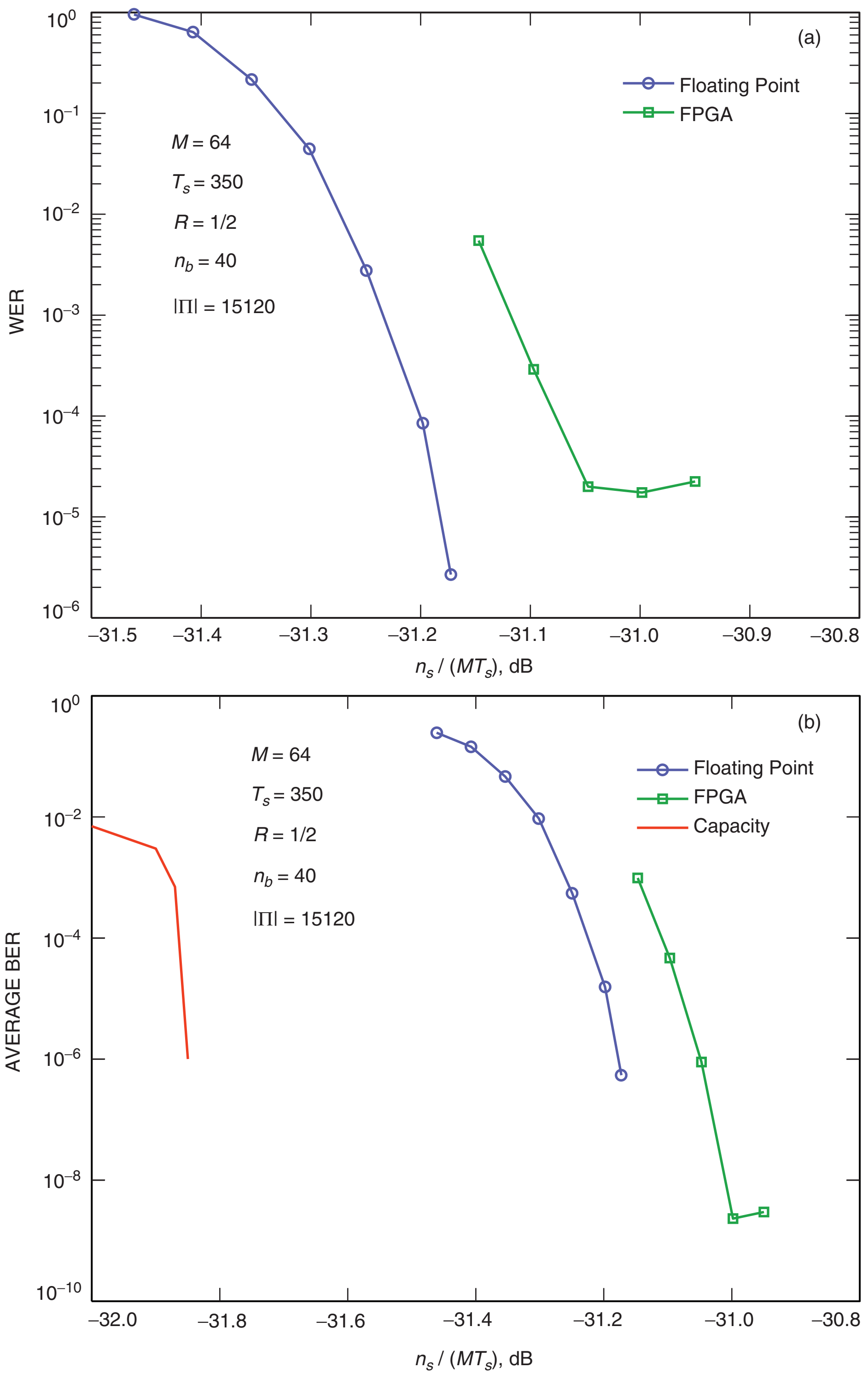

Fig. 8. Performance of the SCPPM FPGA decoder for the worst-case operating point: (a) WER and (b) BER.

decoder is straightforward and involves only the removal of a front-end de-multiplexer. This reduction in hardware can decrease the total FPGA area usage and potentially increase the maximum achievable clock speed.

An 8-bit quantization is used, of which 5 bits are for dynamic range and 3 bits are for binary precision. In all three operating points, the partial-statistics, fixed-point FPGA decoder performs within a 0.2-dB signal energy margin from the full-statistics floating-point decoder. The error floors observed in all cases are caused by the small number of bits used to represent dynamic range. The error floor can be reduced by increasing the dynamic range quantization to 6 bits or more.

There are potential design improvements that can increase the speed and throughput of the SCPPM decoder. Because the outer code has three times the number of trellis stages as the inner code, we can partition the outer code trellis into three segments and apply a window-based BCJR algorithm to all segments simultaneously to reduce the decoding latency. The extra hardware incurred by this approach is manageable since the outer decoder has a small resource utilization. We can also apply a window-based BCJR to the inner code trellis and thus pipeline the $\bar{\alpha}$ and $\bar{\beta}$ computations so that the two circuits are not idle at any time. Moreover, we can build the decoders using next-generation (Virtex IV) FPGAs that are already available on the market. The first two enhancements could lead to 4 and 2 times the speed-up, respectively, for a total increase by a factor of 8, and the third would more than double the speed. Considering the baseline of 1.5 Mbps per decoder slice, the improved design in principle could run at 24 Mbps per slice, and this would allow an aggregate 50 Mbps SCPPM decoder implementation that requires only three FPGAs.

V. Summary

We demonstrated an FPGA implementation of a serially concatenated pulse-position modulation (SCPPM) decoder for deep-space laser communication. With optimizations, our decoder can achieve a throughput of 50 Mbps using three FPGAs and perform within 0.9 dB from the Shannon capacity in floating-point and 1.2 dB in hardware at a BER of 10^{-6}.

References

[1] CCSDS, "Telemetry Channel Coding," Consultative Committee for Space Data Systems Standard 101.0-B-6, October 2002.

[2] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes," *Proc. IEEE Int. Conf. Commun.*, IEEE, vol. 2, Geneva, Switzerland, pp. 1064–1070, May 1993.

[3] C. Berrou and A. Glavieux, "Near Optimum Error-Correcting Coding and Decoding: Turbo Codes," *IEEE Trans. Commun.*, vol. 44, pp. 1261–1271, October 1996.

[4] 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 Telecommunications and Data Acquisition Progress Report, 42-127, July–September 1996*, Jet Propulsion Laboratory, Pasadena, California, pp. 1–20, November 15, 1996.
http://tmo.jpl.nasa.gov/tmo/progress_report/42-127/127H.pdf

[5] B. Moision and J. Hamkins, “Coded Modulation for the Deep-Space Optical Channel: Serially Concatenated Pulse-Position Modulation,” *The Interplanetary Network Progress Report*, vol. 42-161, Jet Propulsion Laboratory, Pasadena, California, pp. 1–25, May 15, 2005.
http://ipnpr.jpl.nasa.gov/progress_report/42-161/161T.pdf

[6] 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.

[7] A. J. Viterbi, “An Intuitive Justification and a Simplified Implementation of the MAP Decoder for Convolutional Codes,” *IEEE J. Select. Areas Commun.*, vol. 16, pp. 260–264, February 1998.

[8] M. Barsoum and B. Moision, *Method and Apparatus for Fast Digital Turbo Decoding for Trellises with Parallel Edges*, JPL Novel Technical Report no. 4123, Jet Propulsion Laboratory, Pasadena, California, July 2004.

[9] M. Cheng, M. Nakashima, J. Hamkins, B. Moision, and M. Barsoum, “A Decoder Architecture for High-Speed Free-Space Laser Communications,” *Proc. of the SPIE*, SPIE, vol. 5712, Bellingham, Washington, pp. 174–185, May 2005.

[10] B. Moision and J. Hamkins, “Reduced Complexity Decoding of Coded Pulse-Position Modulation Using Partial Statistics,” *The Interplanetary Network Progress Report*, vol. 42-161, Jet Propulsion Laboratory, Pasadena, California, pp. 1–20, May 15, 2005.
http://ipnpr.jpl.nasa.gov/progress_report/42-161/161O.pdf

[11] 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.

[12] 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, January 2005.