Investigation of Error Floors of Structured Low-Density Parity-Check Codes by Hardware Emulation

Zhengya Zhang, Lara Dolecek, Borivoje Nikolić, Venkat Anantharam, and Martin Wainwright

Department of Electrical Engineering and Computer Sciences

University of California, Berkeley

Berkeley, CA 94720, USA

Abstract—**Several high performance LDPC codes have parity-check matrices composed of permutation submatrices. We design a parallel-serial architecture to map the decoder of any structured LDPC code in this large family to a hardware emulation platform. A peak throughput of 240Mb/s is achieved in decoding the (2048,1723) Reed-Solomon based LDPC (RS-LDPC) code. Experiments in the low bit error rate (BER) region provide statistics of the error traces, which are used to investigate the causes of the error floor. In a low precision implementation, the error floors are dominated by the fixed-point decoding effects, whereas in a higher precision implementation the errors are attributed to special configurations within the code, whose effect is exacerbated in a fixed-point decoder. This new characterization leads to an improved decoding strategy and higher performance.**

I. INTRODUCTION

Low-density parity-check (LDPC) codes have been demonstrated to perform very close to the Shannon limit when decoded iteratively [1]. However, they have yet to be widely employed in systems that require very low bit error rate (BER) for two reasons: 1) implementation of the decoder is complex, as it requires either long interconnects or large memory bandwidths, which make it difficult to map the decoder onto a very large scale integration (VLSI) system, and 2) the current lack of adequate theory to analytically predict the performance of the code past some moderate values of BER.

The empirical way to study LDPC codes is via simulations. An optimized decoder implemented in C and executed on a high-end microprocessor provides a peak throughput of the order of hundreds of kb/s. Consequently, months of simulation time are required to obtain a confident estimate of the BER at 10^{-10}, thus rendering the approach infeasible. Using a field-programmable gate array (FPGA) platform, emulation of LDPC codes can be accelerated [2][3].

This paper explores practical LDPC decoder design issues using an emulation-based approach. The main contribution is to shed light on the nature of the error floor, which is caused both by intrinsic properties of the code as well as aspects of the quantization scheme. This insight suggests an improved decoding strategy that leads to higher performance.

The message passing algorithm is reviewed in Section II. In Section III a family of high-performance regular LDPC codes with structured parity check matrices is surveyed, and a flexible, high-throughput decoder architecture for this family of LDPC codes is proposed. The performance results of a (2048,1723) RS-LDPC decoder are presented in Section IV.

Error traces are analyzed against the structure of the code to reveal the nature of error floors. Interesting observations are made: after sufficiently many iterations and when the decoder has not converged to a codeword, in a low numerical precision decoder, the error floor is exhibited as the oscillations of hard decisions in going from one iteration to the next. As a contrast, under the same assumptions, in a higher precision decoder the soft decisions do not change over subsequent iterations, and thus converge to a non-codeword.

II. MESSAGE PASSING DECODING OF LDPC CODES

A low-density parity-check code is defined by a sparse $M \times N$ parity check matrix $\boldsymbol{H}$ where N represents the number of bits in the code block and M represents the number of parity checks. The rate of such code is lower bounded by $r = (N - M)/N$. The $\boldsymbol{H}$ matrix of an LDPC code can be illustrated graphically using a Tanner graph, where each bit is represented by a bit node and each check is represented by a check node. An edge exists between the bit node n and the check node m if $\boldsymbol{H}_{mn} = 1$. The set $\mu(n) = \{i: \boldsymbol{H}_{in} = 1, i = 1, 2, \ldots, M\}$ consists of the check nodes connected to the bit node n. Likewise, the set $v(m) = \{j: \boldsymbol{H}_{mj} = 1, j = 1, 2, \ldots, N\}$ consists of the bit nodes connected to the check node m.

An LDPC code can be efficiently decoded with a message passing algorithm. The algorithm relies on the exchange of soft messages between bit nodes and check nodes to achieve correct bit decisions. In the first step, bit nodes x_i, $i = 1, 2, \ldots, N$, are initialized with the prior log-likelihood ratios defined in (1) using the channel outputs y_i, $i = 1, 2, \ldots, N$,

$$ LLR^{prior}(x_i) = \log \frac{\Pr(x_i = 0 \mid y_i)}{\Pr(x_i = 1 \mid y_i)}. \tag{1} $$

Bit nodes propagate the prior *LLR*s to the check nodes via the edges of the Tanner graph. Check nodes sum up the messages received from the bit nodes adjacent to them and send the extrinsic information back to them. Bit nodes follow up by updating bit decisions based on extrinsic information in the next round of iteration.

Bit-to-check and check-to-bit messages are calculated using equations (2), (3), and (4). The messages Q_{nm} and R_{mn} refer to the bit-to-check and check-to-bit messages that are passed between the bit node n and the check node m,

$$Q_{nm} = \sum_{i \in \mu(n)} R_{in} - R_{mn} + LLR^{prior}(x_n), \qquad (2)$$

$$R_{mn} = \Phi\left[\sum_{j \in \nu(m)} \Phi(|Q_{nj}|) - \Phi(|Q_{nm}|)\right]$$

$$\times \left[\mathrm{sgn}(Q_{nm}) \prod_{j \in \nu(m)} \mathrm{sgn}(Q_{nj})\right], \qquad (3)$$

$$\Phi(x) = -\log\left(\tanh(\tfrac{1}{2}x)\right), \ x \geq 0. \qquad (4)$$

The message passing algorithm is allowed to run for a fixed number of iterations. After this a hard decision is performed based on the posterior *LLR* as in (5) and (6),

$$LLR^{posterior}(x_n) = \sum_{i \in \mu(n)} R_{in} + LLR^{prior}(x_n), \qquad (5)$$

$$\hat{x}_n = \begin{cases} 0, & if \quad LLR^{posterior}(x_n) \geq 0 \\ 1, & if \quad LLR^{posterior}(x_n) < 0. \end{cases} \qquad (6)$$

III. Decoder Architecture of Structured LDPC Codes

A practical high-throughput LDPC decoder can be implemented in a fully parallel manner by directly mapping the Tanner graph onto an array of processing elements interconnected by wires. In this parallel implementation, all messages in one direction are processed concurrently, yielding a complex, interconnect-dominated design. On the other hand, the memory bandwidth limits the throughput of a serial decoder [4]. A balance between throughput and memory bandwidth can be achieved if the underlying parity check matrix is regular and structured. The structure of the **H** matrix enables a parallel-serial architecture and a compact memory design.

A. LDPC Codes with Parity Check Matrices Composed of Permutation Matrices

Several known high performance LDPC code constructions, including the Reed-Solomon based [5], array-based [6], lattice-based [7], as well as the ones proposed by Tanner et. al [8], share the same property that their parity check matrices can be written as a two-dimensional array of component matrices of equal size, each of which is a permutation matrix. The constructions using the ideas of Margulis and Ramanujan [9] have a similar property that the component matrices in the parity check matrix are either permutation or all-zeros matrices. A rendition of two examples of this type of parity check matrices is provided in Fig. 1.

In this family of LDPC codes, the $M \times N$ **H** matrix can be partitioned along the boundaries of $\delta \times \delta$ permutation submatrices. For $N = \delta\rho$ and $M = \delta\gamma$, column partition results in ρ column groups and row partition results in γ row groups. This structure of the parity check matrix proves amenable for efficient decoder architecture.
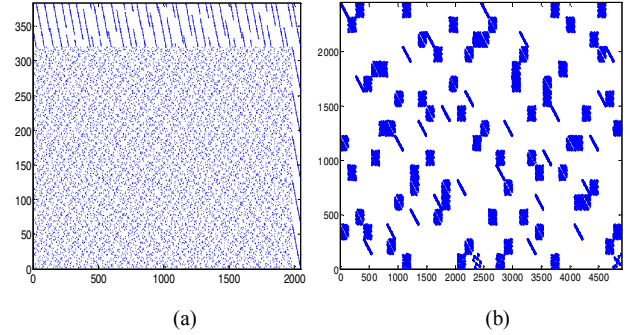

Figure 1. Illustration of parity check matrices of (a) Reed-Solomon based LDPC code, and (b) Ramanujan-Margulis based LDPC code.

B. Parallel-Serial Architecture of LDPC Decoder

The LDPC code under investigation is the (6,32)-regular (2048,1723) Reed-Solomon LDPC code (RS-LDPC). The **H** matrix contains $M = 384$ rows and $N = 2048$ columns. This matrix can be partitioned into $\gamma = 6$ row groups and $\rho = 32$ column groups of $\delta \times \delta = 64 \times 64$ permutation submatrices. Column partition divides the decoder into 32 parallel units, where each unit processes a group of 64 bits.

Fig. 2 illustrates the architecture of the RS-LDPC decoder. Two sets of memories, M0 and M1, are designed to be accessed alternatively. M0 stores bit-to-check messages and M1 stores check-to-bit messages. Each set of memories is divided into 32 banks. Each bank is assigned to a processing unit that can access it independently. During a check-to-bit operation, messages are read from M0 and written to M1. During a bit-to-check operation, messages are read from M1 and written to M0. Messages are stored in the order they are accessed in the bit-to-check operation, so that M1 is accessed sequentially, whereas a lookup table controls the access sequence of M0 in the check-to-bit operation.

The check node accepts 32 bit-to-check messages in every cycle. It takes 384 cycles to complete processing of the entire 384 checks. Each of the 32 bit nodes is assigned to a processing unit. A bit node works inside the processing unit, so it accepts check-to-bit messages serially. It takes 6 cycles to accumulate all the check-to-bit messages for one bit, which leads to 384 cycles to complete processing of all 64 bits in a processing unit. It follows that the 32 bit nodes are able to process the entire block of 2048 bits in 384 cycles.

The message passing algorithm is executed in the following order. M0 is first loaded with the prior *LLR*s. Loading is performed in parallel among the 32 parallel banks. Each bank loads 64 bits serially. A decoding iteration is divided into two steps. In the first step, bit-to-check messages are read from the 32 banks of M0, and then undergo the Φ transformation in parallel before being sent to a check node. A check node calculates the sum, which is then dispatched to each processing unit. The sum is marginalized locally in the processing unit and stored in M1. In the second step, check-to-bit messages are read from the 32 banks of M1 and undergo the Φ^{-1} transformation. The bit node in each processing unit accumulates

the check-to-bit messages and produces a sum every 6 cycles. The sum is marginalized locally and stored in M0. This architecture minimizes the number of global interconnects by performing marginalization within the local processing unit.

The complete decoder is pipelined. A pipeline stall is inserted between check-to-bit and bit-to-check operations to ensure the read-before-write consistency between check-to-bit and bit-to-check operations. The decoder takes 64 cycles to load, 384 cycles to produce check-to-bit messages, and 384 cycles to produce bit-to-check messages. In a high SNR regime, the majority of the received frames can be decoded in one iteration. Thus, the peak throughput of this decoder is 2048 bits/(64+384+384) cycles = 2.46 bits/cycle. The peak throughput is approximately 2.41 bits/cycle after accounting for the pipeline overhead.

The parallel-serial architecture is designed specifically for LDPC code emulation. The architecture allows efficient mapping of a practical decoder onto configurable hardware platforms. For example, an RS-LDPC code of up to 4kb in block length can be easily supported on a Xilinx Virtex-II Pro XC2VP50 FPGA. This area-efficient decoder architecture permits a variety of experiments to be accommodated on the same emulation platform. The main parameters of this architecture are:

Figure 2. Architecture of (2048,1723) RS-LDPC decoder.

1) Memory size: $2w\rho M$ bits,
   Address lookup tables: $\rho M \log_2 M$ bits,
   Function Φ lookup tables: $2w\rho 2^w$ bits, where w denotes the wordlength.
2) Area: proportional to the number of parallel units, ρ, and the wordlength, w.
3) Peak throughput: $N/(\delta + 2M) \approx 0.5/(1 - r)$ bits/cycle.

In practical high-rate codes, $N > M > \delta$, and therefore a high throughput can be achieved with a low number of processing units.

Finally, this architecture is reconfigurable. Any member of the LDPC code family described in Section II.A. can be accommodated. Lookup tables can be reconfigured based on the **H** matrix. Some of the processing units can be enabled or disabled depending on the block length, and the memory size can be increased to allow variable code rates.

C. Decoder Implementation and Emulation Setup

The (2048,1723) RS-LDPC decoder is designed and implemented using Xilinx Virtex-II Pro XC2VP50 FPGA. The decoder is implemented using wordlength $w = 5, 6, 9$ bits, following 3.2 (3 bits for integer and 2 bits for fraction), 3.3, and 4.5 uniform quantization schemes. The Φ lookup table is based on the uniform sampling of the Φ function using the specified wordlength and quantization. The device utilization is listed in Table I. AWGN noise generator is implemented on chip. Final iterations of soft decisions are stored in an onboard static memory module when decoding fails. An on-chip PowerPC microprocessor controls the decoder, noise generator, and the interface with the memory module. The hardware emulation platform is illustrated in Fig. 3. It allows the characterization of the code and evaluation of practical implementation parameters. Error traces enable the exploration of patterns that cause the decoder to fail.
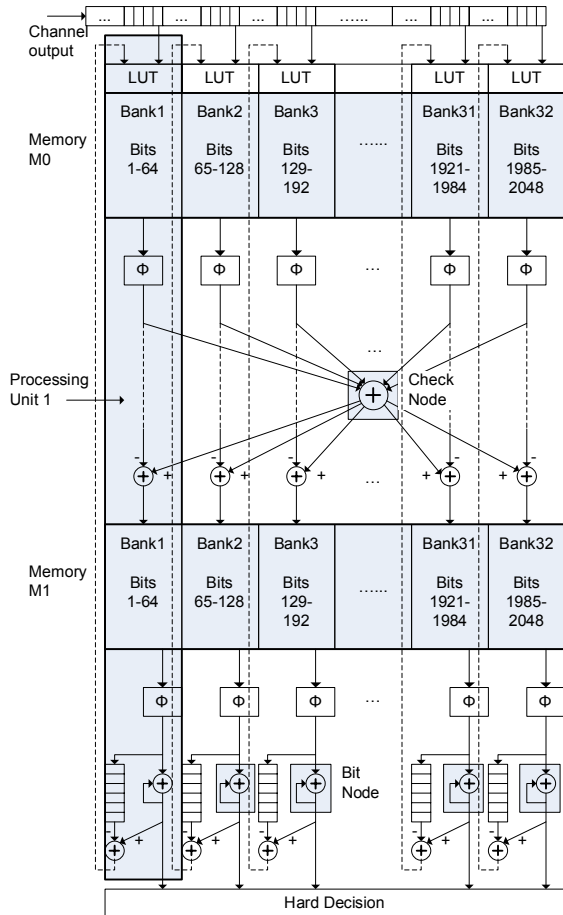
TABLE I
DEVICE UTILIZATION OF (2048,1723) RS-LDPC DECODER IMPLEMENTATIONS

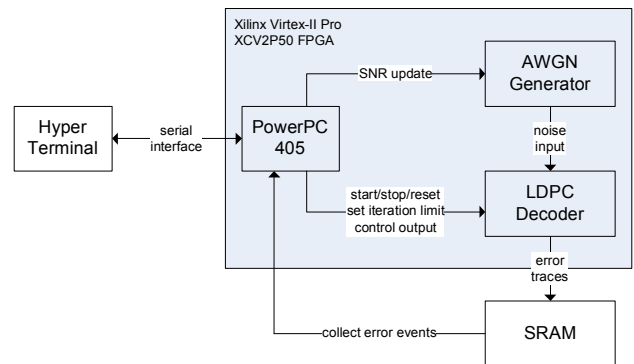|	5-bit decoder	6-bit decoder	9-bit decoder	Noise generator	Peripherals
Slice Flip Flops	11204	13658	21020	6543	1269
4-Input LUTs	2856	3780	7503	5765	1370
Occupied Slices	6763	7320	16247	5404	1281
Block RAMs	129	129	129	24	32

Figure 3. LDPC decoder emulation platform.

The decoder reaches a peak throughput of 240 Mb/s using a 100 MHz clock rate. This throughput can potentially be doubled using a 200 MHz clock rate with buffered global interconnects and control lines. Hardware emulation of this LDPC decoder extends the BER curve beyond 10^{-10} within an hour. For comparison, an optimized implementation of the same decoder in C provides a peak throughput of 260 kb/s (without accounting for noise generation latency) on an Intel Xeon 2.4 GHz microprocessor. The comparison demonstrates the advantage of hardware emulation.

IV. RESULTS

The wordlength and the number of decoding iterations are important design parameters that determine the area, power, and performance of an LDPC decoder. A short wordlength and a small number of iterations are always desirable in practical implementations.
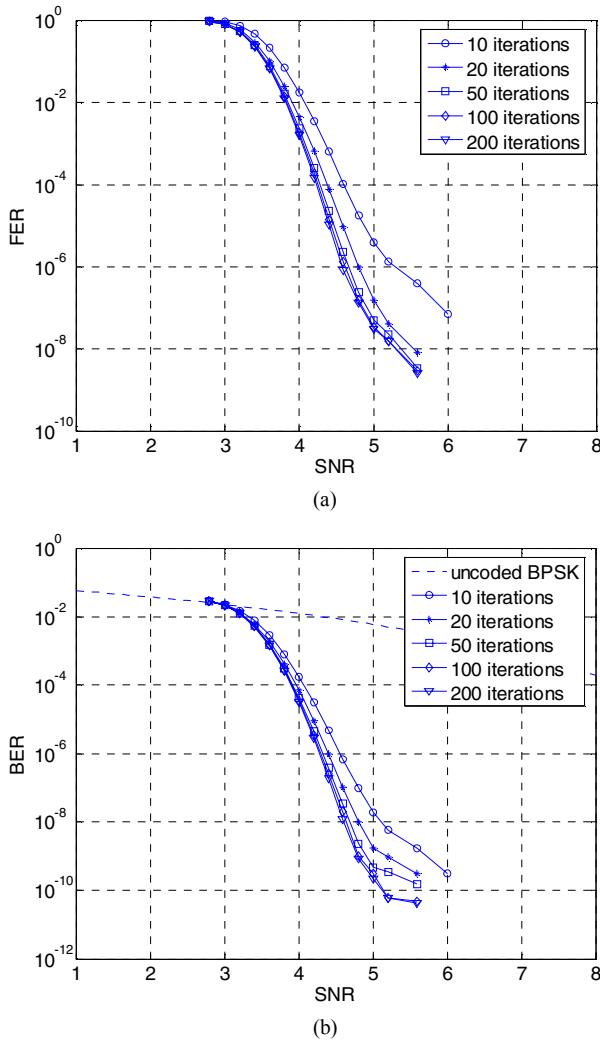

Figure 4. (a) FER and (b) BER performance of a 6-bit fixed-point implementation of the (2048,1723) RS-LDPC code for various number of decoding iterations.

The frame error rate (FER) and the bit error rate versus the signal-to-noise ratio (SNR) are plotted in Fig. 4 showing the effect of iteration number on the performance of a 6-bit fixed-point implementation of the (2048,1723) RS-LDPC decoder. More iterations result in better performance, though the gain becomes marginal after 50 iterations. So as to minimize the effect of iteration number and to isolate the error events caused by fixed-point implementations, we perform up to 200 iterations. The FER and BER versus SNR curves are shown in Fig. 5 for fixed-point decoder implementations using $w = 5$, 6, and 9 bits.

A. Characterization of Error Events

We begin by introducing a notion of an absorbing set which is used to describe an error event that can occur when the message passing decoding fails to converge to a codeword after a large number of iterations.

Definition of an absorbing set: Let T_H be the bipartite Tanner graph corresponding to the parity check matrix $\boldsymbol{H}$ of the given code. We say that the subset of a bit nodes and their neighboring b check nodes in T_H constitute an (a, b) set if in the subgraph induced by these a bit nodes, exactly $b > 0$ check nodes have odd degrees, each of these a bit nodes is connected to more even-degree checks than odd-degree checks, and all remaining check nodes outside of the induced subgraph have even degree with respect to T_H. We say that an (a, b) set is an (a, b) absorbing set if for all a', $a' < a$, it does not contain an (a', b) set as its subgraph. ∎

Related notions have been previously introduced in the literature in the attempt to characterize the behavior of the algorithm when it does not converge to a codeword, such as stopping sets [10], near-codewords [11], and trapping sets [12]. We are specifically motivated by the trapping set definition [12] but choose to instead use the absorbing set as defined above in order to explicitly distinguish the convergence of the decoder to a non-codeword from its oscillatory behavior.

B. Error Analysis

In all experiments, an all-zeros codeword is transmitted. The final 16 iterations are recorded when the decoder fails to converge to a codeword after 200 iterations. If all bit values remain the same for the final 16 iterations, and the decoder has not converged to a codeword, we say that the observed error event is an (a,b)-absorbing-set error if b checks are unsatisfied and same a bits have wrong values in each of the final iterations.

In the 5-bit fixed-point implementation, an error floor appears at the BER of approximately 4.5×10^{-8} as shown in Fig. 5. A total of 74 error frames is collected at an SNR of 5.2 dB, of which 71 give rise to an oscillatory behavior and 3 are (8,8)-absorbing-set errors. Examples of the bit error counts illustrating the oscillatory behavior are given in Table II. Observe that the entries have a periodic structure in which every third entry is relatively small, followed by a medium-sized entry, which is in turn followed by a large entry. This behav-

ior can be attributed to the dynamics of the message exchange in which a small number of bits, in our experiments on the order of 20-30, propagate negative messages through their neighboring checks. These in turn make some of their other neighboring bits admit incorrect values (50-80 is observed), which are propagated further to more bits (120-200 is observed). As the number of incorrect bits increases, so do their neighboring sets, which means that after two steps there is a sufficient number of available checks to enforce the correct values. As a result, the total number of incorrect bits goes down again. Since the number of incorrectly decoded bits varies significantly from one phase of the oscillation to the next (or from one iteration to the next), we perform averaging when reporting the bit error rates.

Since the oscillation errors are a consequence of the limited precision of the 5-bit decoder, all error frames are post-processed with the 6-bit fixed-point decoder. The errors manifested in oscillating sets are corrected within 9 iterations, thereby eliminating the short wordlength-induced error floor. The absorbing set errors are not corrected with the 6-bit post-processing because the post-processing cannot overcome the strong incorrect likelihoods associated with the nodes in the absorbing set due to the special configuration of these nodes. Post-processing with the floating point decoder is also not successful, further suggesting the "absorbing" property of these configurations, on which we elaborate later.

The 6-bit fixed-point implementation demonstrates a better performance. The BER curve still incurs a significant change of slope at 10^{-10}. A total of 31 error frames is captured between 5.2 dB and 5.6 dB of SNR, 27 of which are due to (8,8) absorbing sets and 4 are due to oscillations. All error frames are post-processed with an 8-bit fixed-point decoder. As before, all oscillation errors are corrected within 4 iterations but none of the absorbing set errors are corrected, even when a floating-point decoder is used for post-processing.

The 9-bit implementation shows the best performance in high SNR, although all 6 error frames collected at 5.6 dB SNR exhibit the (8,8) absorbing set structure.

C. Absorbing Set Characterization

As previously discussed, all encountered absorbing-set errors are of (8,8) type. They all share the same structure in which these 8 bit nodes participate in a total of 28 checks. Of these 28 checks, 20 are connected with degree-2 to the 8 bit nodes. Since the girth of the code is at least 6 [5], these bit node pairs are all different. The remaining 8 checks are each connected to a different bit node in the absorbing set. The illustration of such configuration is provided in Fig. 6. For an intuitive explanation of why the failures occur in such set, suppose that, to begin with, all 8 bits in the absorbing set have extremely in-

correct values and all other bits have extremely correct values. Then all but 8 checks are satisfied. These incorrect bits then reinforce each other's incorrect values through the checks they share. In particular, each such bit, along with its extreme incorrect prior, receives 5 such messages. The (correct) message from its remaining neighboring check cannot overcome this joint effect. As a result, the values remain incorrect.

This behavior is also verified experimentally by simulating a floating point decoder for channel realizations with extremely noisy inputs in precisely 8 coordinates that constitute an absorbing set, and observing that even the floating point decoder cannot successfully decode such realizations.
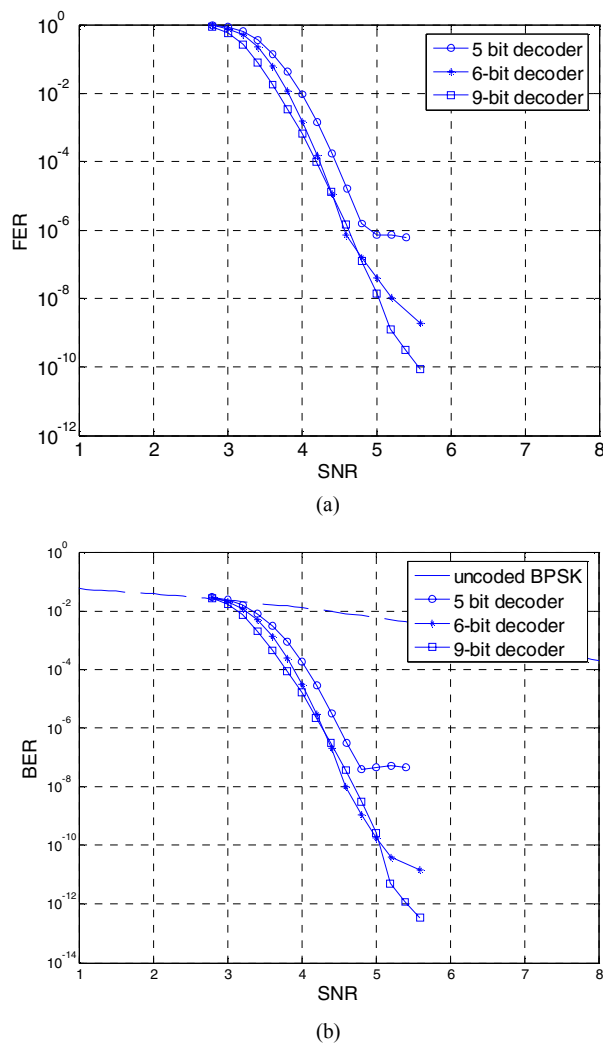

(a)

(b)

Figure 5. (a) FER and (b) BER performance of the (2048,1723) RS-LDPC code with 5-bit, 6-bit, and 9-bit fixed-point implementations.

TABLE II
EXAMPLES OF BIT ERROR COUNTS IN THE FINAL 16 ITERATIONS OF DECODING

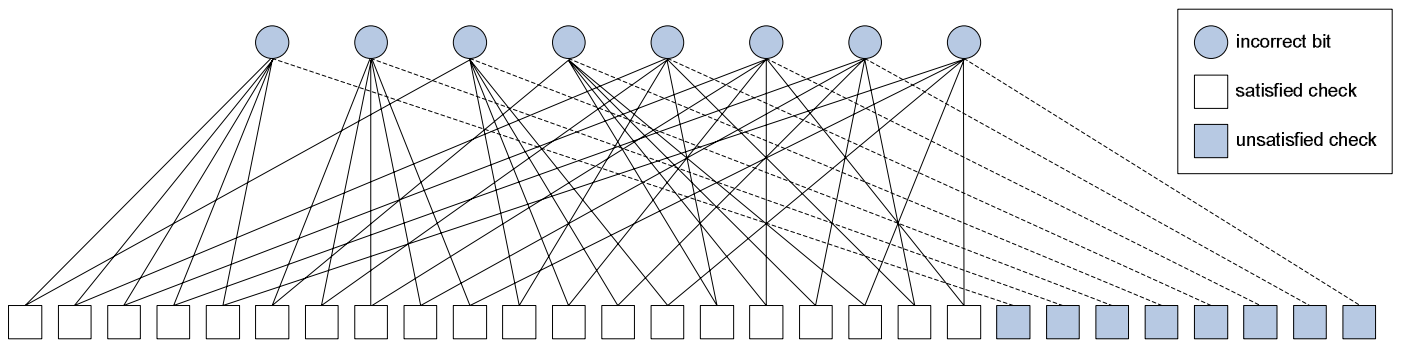| Iteration #	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200
Error 1	143	24	75	147	23	68	137	23	72	143	24	69	136	24	75	121
Error 2	125	28	60	149	28	61	125	28	60	149	28	61	125	28	60	149
Error 3	123	29	53	133	28	52	148	28	49	138	28	49	128	27	51	139

Figure 6. Illustration of the subgraph induced by the incorrect bits in an (8,8) absorbing set.

Even though this special (8,8) configuration is intrinsic to the code, when the dynamic range is finite, the power of the neighboring check of an incorrect bit that is itself not a part of the absorbing set, is limited. As a consequence, the occurrence of absorbing sets is increased in a shorter wordlength implementation. This is demonstrated in the difference between the performance of the 6-bit and 9-bit decoders at the same SNR level, whereby in the former case the number of failures due to absorbing sets is significantly higher.

Using the characterization of an (8,8) absorbing set given earlier one can create a list of all such sets. For each group of 8 checks participating in an absorbing set, one has a bit sublist consisting of all collections of 8 bit nodes with each collection completing an absorbing set with the given group of 8 checks. Upon observing that the decoder remains in an absorbing set with 8 unsatisfied checks, one bit at a time is selected from the bit sublist determined by the group of 8 unsatisfied checks. The decoder is then run for a small number of additional iterations, e.g., 10, but with the selected bit flipped. For all collected errors due to (8,8) absorbing sets, this post-processing results in successful decoding to the transmitted codeword.

V. CONCLUSIONS AND FUTURE WORK

The proposed parallel-serial, flexible, high-throughput architecture allows mapping of a family of high-performance LDPC decoders on an emulation platform. Low BER error traces reaching 10^{-13} are captured using this emulation platform for a (2048,1723) RS-LDPC code. Analysis of the error traces reveals interesting properties of the error floor. In a low-precision implementation, the dominant cause of the error floor is the oscillatory behavior, which can be corrected with a slight increase in the wordlength. Absorbing sets dominate error floors in a higher precision implementation and are due to the code construction, exacerbated by the finite wordlength implementation. Absorbing sets can be enumerated and used to predict the location of error floors. Enumeration of the most frequent absorbing set configuration is used in post processing to improve the overall performance and lower the error floor. How to exploit the knowledge of the absorbing set configuration for a better code design and error floor prediction is the subject of our future work.

REFERENCES

[1] R.G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.
[2] L. Sun, H. Song, Z. Keirn, and B.V.K.V. Kumar, "Field programmable gate array (FPGA) for iterative code evaluation," *IEEE Trans. on Magnetics*, vol. 42. no. 2, pp. 226-231, Feb. 2006.
[3] L. Yang, H. Liu, and R. Shi, "Code construction and FGPA implementation of capacity approaching low error-floor LDPC decoder," to appear in *IEEE Trans. on Circuits and Systems*, 2006.
[4] E. Yeo, B. Nikolic, and V. Anantharam, "Iterative decoder architectures," *IEEE Communications Magazine*, pp.132-140, Aug. 2003.
[5] I. Djurdjevic, J. Xu, K. Abdel-Ghaffar, and S. Lin, "A class of low-density parity-check codes constructed based on Reed-Solomon codes with two information symbols," *IEEE Communications Letters*, vol. 7, no. 7, pp. 317-319, July 2003.
[6] J. Fan, "Array codes as low-density parity check codes," in *Proc. 2nd Int. Symp. Turbo Codes and Related Topics*, Brest, France, pp. 543-546, Sept. 2000.
[7] B. Vasic, K. Pedagani, and M. Ivkovic, "High-rate girth-eight low-density parity-check codes on rectangular integer lattices," *IEEE Trans. on Communications*, vol. 52, no. 8, Aug. 2004.
[8] R. M. Tanner, D. Sridhara, A. Sridharan, T. E. Fuja, and D. J. Costello, "LDPC block and convolutional codes based on circulant matrices," *IEEE Trans. on Information Theory*, vol. 50, no. 12, pp. 2966-2984, Dec. 2004.
[9] J. Rosenthal and P.O. Vontobel, "Constructions of LDPC codes using Ramanujan graphs and ideas from Margulis," in *Proc. of the 38th Annual Allerton Conference on Communications, Control, and Computing*, pp. 248-257, Oct. 2000.
[10] C. Di, D. Proietti, I. E. Telatar, T. J. Richardson, R. L. Urbanke, "Finite-length analysis of low-density parity-check codes on the binary erasure channel," *IEEE Trans. on Information Theory*, vol. 48, no. 6, pp. 1570-1579, Jun 2002.
[11] D. MacKay and M. Postol, "Weaknesses of Margulis and Ramanujan-Margulis low-density parity check codes," *Electronic Notes in Theoretical Computer Science*, vol. 74, 2003.
[12] T. Richardson, "Error floors of LDPC codes," in *Proc. of the 41st Annual Allerton Conference on Communications, Control, and Computing*, Oct. 2003.