A reconfigurable FPGA implementation of an LDPC decoder for unstructured codes

Hosseini, S. M. Ehsan; Chan, Kheong Sann; Goh, Wang Ling

2008

Hosseini, S. M. E., Chan, K. S., & Goh, W. L. (2008). A reconfigurable FPGA implementation of an LDPC decoder for unstructured codes. 2nd International Conference on Signals, Circuits and Systems: Hammamet, Tunisia, (pp.1-6).

<https://hdl.handle.net/10356/99867>

<https://doi.org/10.1109/ICSCS.2008.4746952>

© 2008 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE. This material is presented to ensure timely dissemination of scholarly and technical work. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author's copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder. <http://www.ieee.org/portal/site> This material is presented to ensure timely dissemination of scholarly and technical work. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author's copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder.

A reconfigurable FPGA implementation of an LDPC decoder for unstructured codes

S. M. Ehsan Hosseini
School of Electrical and
Electronic Engineering

Nanyang Technological University
50 Nanyang Ave, Singapore
Email: E070013@ntu.edu.sg

Kheong Sann Chan
Data Storage Institute
A*STAR(Agency for Science,
Technology and Research), Singapore
Email: chan_kheong_sann@dsi.a-star.edu.sg

Wang Ling Goh
School of Electrical and
Electronic Engineering
Nanyang Technological University
50 Nanyang Ave, Singapore
Email: ewlgoh@ntu.edu.sg

Abstract—This paper describes the implementation of a general and embedded decoder for the evaluation of unstructured low-density parity-check (LDPC) codes over additive-white Gaussian noise (AWGN) channels. The decoder, which has a serial architecture and moderate throughput, is a peripheral connected to the embedded PowerPC processor of a Xilinx Virtex-II Pro FPGA and is managed by the processor. This method of Hardware/Software implementation provides the maximum flexibility for the development and rapid prototyping of the hardware-based simulator system. The decoding algorithm proposed in this paper belongs to the class of min-sum with correction factor in which the correction factor updates with the log-likelihood ratio (LLR) values.

I. INTRODUCTION

The low-density parity-check (LDPC) codes [1] are a class of codes specified by a parity check matrix which is sparse. The total number of ones in each row and column is small in comparison to the total number of rows and columns. Their superior performance was reported by Mackay and Neal [2] in mid 1990s. The drawback of LDPC codes is their high decoding complexity which is a consequence of the lack of a regular structure in the message communication mechanism [3]. This will lead to complex interconnections and large amount of memory in the decoder implementation, which will then limit the applicable parallelism of the decoding algorithm.

To overcome the decoding complexity, research has been done either on the reduced-complexity decoding algorithms and modifications of the algorithm [4-7] or the decoder architecture [8-11]. These architectures take advantage of inherent parallelism in the decoding algorithm and can achieve high throughputs. However, the structure of implemented decoders is mostly code dependent and can support a class of LDPC codes such as quasi-cyclic LDPC codes. More flexible decoders were also been introduced in recent years. In [12], a serial decoder, that has moderate throughput as compared to the parallel ones, has been reported. A flexible decoder using interconnection networks and implemented on ASIC is presented in [8]. The architecture enables decoding of arbitrarily designed LDPC codes and is area efficient. However, the throughput is less than architectures optimized for specific codes.

For the purpose of code performance evaluation, the key factor of the decoder is flexibility and not being code dependent. The ease of reconfiguration for any given code is also desirable. Since the real-time functionality is not a concern in simulations, the reduction of throughput, which is a consequence of having greater flexibility, is acceptable. In [13] a field-programmable gate array (FPGA) based simulator is presented. This architecture is capable of investigating the bit error rate (BER) performance of LDPC codes at 10^{-10} . The decoder in the system is partially parallel and can support only structured LDPC codes. Another hardware-based simulator is presented in [14] where the decoder is designed for high rate quasi-cyclic codes.

In our design, we aim to build an FPGA-based LDPC decoder that can support high rate unstructured LDPC codes for the purpose of BER performance evaluation. For the evaluation of LDPC codes, it is not appropriate to design a different decoder for different codes. Hence, a flexible decoder that can be readily reconfigured and programmed is considered. In order to support unstructured parity check matrices, we opted for a serial architecture with an aim to achieve maximum throughput. The decoder is proposed as a peripheral that is instantiated in an embedded processor system. The decoder is connected to the local bus of the embedded processor of the FPGA which is the PowerPC of the Virtex-II Pro FPGAs. The decoder is configured and controlled by software. The advantage of making the design peripheral-based is that it can be reused in a system where other peripherals are connected to the processor bus by means of available EDA tools. Such a Hardware/Software system also allows us to simply add other functionalities that are not time-critical by writing C code for the embedded processor. Other than the Hardware/Software implementation, we also propose some simplifications to the sum-product decoding algorithm [15] to make it more appropriate for the hardware implementation.

The remainder of this paper is organized as follows. Section 2 describes the decoding algorithm and the simplifications that have been made. Section 3 discusses the hardware architecture for the decoder. Section 4 presents the BER performance of some LDPC codes evaluated by the system, and section 5 draws the conclusions.

II. DECODING LDPC CODES

The sum-product algorithm is an efficient decoding algorithm of LDPC codes. The parity check matrix can be viewed as a bipartite graph having two kinds of nodes: check-nodes which correspond to the rows and variable-nodes corresponding to the columns of the parity check matrix. Each iteration, the check-nodes receive messages from variable-nodes and then pass updated values back to the variable-nodes. Decoding will stop if all the parity check equations are satisfied or a number of iterations has been reached. We denote λ_i as the log-likelihood ratio (LLR) of variable node i , $u_{m,n}$ as the message passed from check-node m to variable node n and r_n the receiver's observation. The check-node update can be written as:

$$u_{m,n}^{(l)} = \prod_{i \in N(m) \setminus n} \text{sgn}(\lambda_i^{(l-1)} - u_{m,i}^{(l-1)}) \times \varphi \left(\sum_{j \in N(m) \setminus n} \varphi(|\lambda_j^{(l-1)} - u_{m,j}^{(l-1)}|) \right) \quad (1)$$

where:

$$\varphi(x) = \log \left(\frac{e^x + 1}{e^x - 1} \right) \quad (2)$$

And the variable-node update is:

$$\lambda_n^{(l)} = \frac{2}{\sigma^2} r_n + \sum_{m \in M(n)} u_{m,n}^{(l)} \quad (3)$$

From the above equations, it is obvious that the check-node updates are more computationally complex when compared to the variable-node updates. This complexity arises from two factors: first, the computation of $\varphi(x)$ function and second, the exclusion of the LLR value of each variable-node in the computation of messages passed to it from the connected check-nodes. In other words, the check-node should pass different values to each of the connected variable-nodes.

In order to reduce the computation of the check-node processing, different simplifications have been made to it [4]. In this article we propose a new simplification that reduces the complexity of the sum-product algorithm without any significant degradation of the performance. Let $z_{m,i}$ denotes the absolute value passed from the variable-node i to check-node m :

$$z_{m,i} = |\lambda_i - u_{m,i}| \quad (4)$$

Thus (1) can be written as:

$$u_{m,n}^{(l)} = \text{sgn}(u_{m,n}^{(l-1)}) \varphi \left(\sum_{i \in N(m) \setminus n} \varphi(z_{m,i}^{(l-1)}) \right) \quad (5)$$

Because of the behavior of $\varphi(x)$, the minimum value of $z_{m,i}$ has more significant effect in the summation in (5) and leads to an approximation known as the min-sum. However, a better approximation can be made. Let us denote $z_{m,min}$ and $z_{m,min2}$ as the minimum and the next minimum values in $\{z_{m,i} | i \in N(m)\}$ respectively. The value of $z_{m,min}$ exists in the computation of all the check-node values except for

$u_{m,min}$. It has the most contribution in the summation step of (5), for the computation of check-node value. Hence we can make the following approximation:

$$|u_{m,others}| \approx |u_{m,min2}| \quad (6)$$

In which $u_{m,others}$ corresponds to $\{u_{m,i} | i \neq min, min2\}$. (5) then can be simplified to the followings:

$$u_{m,n}^{(l)} = \begin{cases} \text{sgn}(u_{m,n}^{(l-1)}) \varphi(\sum_{i \in N(m) \setminus min} \varphi(z_{m,i}^{(l-1)})) & n = min \\ \text{sgn}(u_{m,n}^{(l-1)}) \varphi(\sum_{i \in N(m) \setminus min2} \varphi(z_{m,i}^{(l-1)})) & n \neq min \end{cases} \quad (7)$$

In (7), the exclusion is performed only twice, and this helps to reduce the number of computations, especially for check-nodes with higher degrees. In both cases of (7), the summation only differs in one term and the sum of the other terms appears in both. Define Z_m as:

$$Z_m = \varphi \left(\sum_{i \in N(m) \setminus min, min2} \varphi(z_{m,i}) \right) \quad (8)$$

Then we can write:

$$|u_{m,n}^{(l)}| = \begin{cases} \varphi(\varphi(z_{m,min2}^{(l-1)}) + \varphi(Z_m^{(l-1)})) & n = min \\ \varphi(\varphi(z_{m,min}^{(l-1)}) + \varphi(Z_m^{(l-1)})) & n \neq min \end{cases} \quad (9)$$

Note that we have used the symmetry property of $\varphi(x)$. In (9), the check-node computation is reduced to one of degree-three and Z_m represents the effect of other variable-nodes connected to check-node m than minimum and next minimum variable-nodes in computation of the magnitude. Using the Jacobian logarithm approach [15], (9) can be expressed as:

$$|u_{m,n}^{(l)}| = \begin{cases} z_{m,min2}^{(l-1)} + \log \left(\frac{1+e^{-|z_{m,min2}^{(l-1)} + Z_m^{(l-1)}|}}{1+e^{-|z_{m,min2}^{(l-1)} - Z_m^{(l-1)}|}} \right) & n = min \\ z_{m,min}^{(l-1)} + \log \left(\frac{1+e^{-|z_{m,min}^{(l-1)} + Z_m^{(l-1)}|}}{1+e^{-|z_{m,min}^{(l-1)} - Z_m^{(l-1)}|}} \right) & n \neq min \end{cases} \quad (10)$$

In here, we had assumed that $z_{m,min} < z_{m,min2} < Z_m$. The above equation is similar to the min-sum approximation with a correction factor. Unlike other methods such as the Offset-minsum [4], the correction factor used in our proposed algorithm is not a constant but depends on the value of the variable-nodes connected to the check-node and is updated dynamically. The computation of the correction factors can be done using the look-up tables. The computation of (9) using (10) is less subjected to the quantization effects. This is because the quantization error only appears in the correction factor.

III. SYSTEM ARCHITECTURE

Based on the decoding algorithm of the previous section, the implementation of the decoder is proposed. The design is a Hardware/Software design that takes advantage of the PowerPC processors embedded in the Virtex-II Pro FPGAs. This methodology provides us more flexibility since the additional functionality can be added to the system simply by writing C code for the PowerPC. The overall system architecture is shown in figure 1.

Fig. 1. The hardware-based simulator architecture.

The PowerPC is connected to the Processor Local Bus (PLB) with three other major components: the DDR external memory, the decoder and the additive white Gaussian noise (AWGN) generator. The processor's executable program as well as the parity check matrix structure is located on the DDR memory since the parity check matrix structure contains a large amount of data for long and unstructured LDPC codes.

The parity check matrix data is segmented equally (i.e. the data of a known number of rows) and each segment is read from DDR memory and stored in the FPGA's Block-RAMs (BRAMs). The communication between the FPGA and the host PC is done through a UART interface. The control commands and the parity check matrix data are sent to the system via UART, which also monitors status of the simulator such as number of error blocks detected and the number of error bits in the blocks. The UART interface that is controlled by the processor enables us to reconfigure the system without having to download the bitstream on the FPGA.

A. The Serial Decoder Architecture

The decoder architecture illustrated in Figure 2 is serial, in which the check-node and variable-node operations are done sequentially. The throughput of the parallel decoders is significantly higher than serial ones since they use the inherent parallelism of the sum-product decoding algorithm. However, in the parallel decoders the complexity increases with the code length as well as the check-node and variable-node degrees, which lead to vast amount of wire interconnections. Consequently, the parallel architectures are generally used for structured codes such as quasi-cyclic ones. Since the decoder is intended for the simulation of any kind of LDPC codes as well as unstructured ones the serial architecture is chosen. The serial architecture also consumes fewer resources which enables us to implement other peripherals in the embedded processor system for further functionality.

The decoder consists of two memory spaces for storing the parity check matrix. One contains the location of ones in each row and the other the location of ones in each column named as *Row-to-Col map* memory and *Col-to-Row map* memory respectively. At the start of each iteration, the indices of columns in a row are read and the corresponding

Fig. 2. Block diagram of the embedded LDPC decoder and its bus interface to the PLB.

LLR values are read from the *LLR memory*. These values are sent to the *Check-node processor (CNP)* for the check-node updating. As will be shown later, the check-node updating is not done immediately after reading the LLR values and requires the same processing time. In order to save on the memory consumed by the Row-to-Col map on the FPGA, and to take advantage of a free time slot, the Row-to-Col memory consists of two identical banks. While one bank is being used for mapping data, the other is filled with the data read from the external DDR memory. The maximum number of ones in each row of the parity matrix is limited to be 64 and each memory bank stores the information of 16 rows, given that each index is represented by 16 bits, each bank is of size of $64 \times 16 = 1024$ 16-bit words. This space can be realized using a single BRAM. Consequently, the total space consumed for Row-to-Col memory is two BRAMs. Unlike the check-node update, the variable-node update can be done instantaneously. If the Col-to-Row memory data is written in a segmented fashion, it will slow down the variable-node process. Based on this, the Col-to-Row is only updated once before the system starts the decoding. Given that in our example, the maximum variable-degree is 4 and the maximum code length is 4608, the memory size needed is therefore $4 \times 4608 = 18432$ 16-bit words. This will require 18 BRAMs.

The messages in all stages are represented in (9:5) signed format. The data read from channel (AWGN) are stored in *Rx Data* memory and LLR memory as the initial LLR values. The data from the LLR memory is sent to the CNP with the architecture provided in figure 3. The CNP handles the check-update for a single row. There is a buffer at the input of the CNP which is filled with the LLR values. These values are then read from the buffer and converted to sign and magnitude and processed individually.

In the CNP, two mathematical functions are evaluated, $\varphi(x)$ and $\psi(x) = \log(1 + e^{-|x|})$. These functions can be realized using the look-up tables (LUTs). However, since $d\varphi(x)/dx$ is

Fig. 3. Block diagram of the check-node processor.

large when x is small, if a uniform quantization is used, it results in a large LUT. In order to reduce the size of the LUT, a non-uniform quantization is preferred. The input to these functions, which is 8-bit length, is passed to the leading zero detector (LZD) module. (see [16] for a discussion on LZD). The output of the LZD is used as the address to the LUTs. Using this approach reduces the resources needed in the non-linear function approximation used in the computation of the correction factor.

According to (8), the computation of $\varphi(x)$ is done on all of the LLR values except for the minimum and the next minimum. Thus the computation cannot be done before these values are known. To overcome this delay, all the LLR values (including min and next min) are fed to the LZD and LUT of $\varphi(x)$ and added together, while the *min finder* unit simultaneously looks for the min and the next min. After the values are added together, the min and the next min are fed again to the LZD and are subtracted from the previously latched value. Consequently, the summation is available just a few clock cycles after the min value is found. This saves time especially for check-nodes with larger degrees. As per equation (8), the evaluation of $\varphi(x)$ is performed once more on the accumulated value after it has been truncated back to 8-bits. The adjusted value is fed back for final computation and the result is passed on to the *correction factor (CF)* unit together with the min and next min values. In the CF unit, the $\psi(x)$ is evaluated using the same method discussed above, to approximate the second term in (10). Each of these terms needs to compute $\psi(x)$ twice (the results are subtracted.). Two separate LZDs and LUTs are used for each of the correction factors in order to reduce the latency in the CF unit.

Using the min-sum decoding algorithm with correction factor reduces the computational load of the check-node update process and also significantly reduces the memory require-

ment. According to (10), the magnitudes of the messages passed to the variable-nodes have one of two values. This allows us to store compressed check-node values. In this method the updated value can be represented using a single slightly longer word, which consists of four parts and is depicted in figure 3. The first part is called the *sign pattern* and carries the sign of the messages passed to the connected variable-nodes. It is 64 bits, for the maximum number of sixty four ones in a row. The second part contains position (index) of the minimum value of the received variable-nodes messages. The next two parts are the messages (magnitude) sent to the min variable-node and the other variable-nodes and are each represented by eight bits, giving a total of $64 + 6 + 8 + 8 = 86$ bits for the register. Assuming the maximum number of rows is 1024, the total memory size required for check-nodes is 86Kb which is implemented on FPGA using 6 BRAMs. This is significantly lower than $64 \times 9 \times 1024 = 576$ Kb, if the updated values were not compressed.

The next step after the check-node updates is updating of the variable-nodes or LLR values. The addresses of the check-nodes connected to a variable-node are read from the Col-to-Row memory and the corresponding values are retrieved from the compressed format stored in check-node memory. These addresses are also sent to the *syndrome check* module as they indicate the parity check equations in which the processed variable-node takes part. The hard decision is also done at this stage. Once all the LLR values are updated the done signal is sent to the PowerPC and the status registers are updated. These registers indicate whether the parity check equations have been satisfied and the number of error bits in the block after making a hard decision. If the parity check is satisfied or the maximum number of iterations is reached, the processing for the current block finishes.

Fig. 4. BER and FER performance of some unstructured LDPC codes.

B. The AWGN Channel

In order to test the performance of the decoder and the LDPC codes, an AWGN channel with the all zero codeword is transmitted. Since the evaluation of codes for higher signal-to-noise ratio (SNR) values requires large numbers of samples, the Gaussian random number generator needs to have very good statistical characteristics. Among the reported designs the one in [17], which uses the Box-Muller method, is chosen and implemented which has a very high throughput (200 million samples running at 100MHz) and is able to produce samples of up to 8.2σ from the mean. This is critical, especially for higher SNR's, since these extreme values are the ones causing the errors at low BER's.

IV. IMPLEMENTATION AND RESULTS

The described architecture for the embedded serial LDPC decoder is implemented on a Virtex-II Pro (XC2VP30) FPGA. The decoder occupies 729 slices which is less than 6% of available slices on the FPGA. This low area implementation is a result of the serial architecture. By comparison, the AWGN generator is implemented using 1568 slices. The decoder and the AWGN generator are controlled by the PowerPC processor, programmed in C. The maximum throughput of our simulator is about 2Mbits/sec. Although, it is less than the throughput of parallel decoders presented in the literature it has distinct characteristics, including low area, ability of decoding arbitrary LDPC codes and Hardware/Software co-design, and that makes it as an affordable and convenient choice for LDPC decoding simulations.

Figure 4. shows the BER and frame error rate (FER) performances of some unstructured LDPC codes available

Fig. 5. Performance comparison of different decoding algorithms.

from Mackay's website [18]. The codes chosen are of high rate, with block lengths of around 4000, and check-degrees ranging from 17 to 63, demonstrating the flexibility of the simulator. For our proposed design we can achieve BER as low as 10^{-9} in a few hours for random LDPC codes.

In order to compare the performance of the proposed simplified algorithm with other Min-Sum family algorithms, the decoder can be readily modified to support Min-Sum and Offset-minsum algorithms. The (4095,3358) LDPC code, which shows error floor, is simulated and performance curves are plotted in figure 5 for different algorithms. The offset value of the Offset-minsum algorithm is chosen to be 0.5 according to software based simulations. The curves reveal that the proposed algorithm performs better in the *waterfall* region. However, all the algorithms, including Min-Sum, performs quite close to each other in the error floor region. The proposed algorithm is also advantageous to the Offset-minsum since it does not require off-line computations such as density evaluation.

Table 1. shows the throughput of the various codes mentioned earlier for a maximum number of 10 iterations. Since

TABLE I
SYSTEM PERFORMANCE FOR DIFFERENT HIGH-RATE CODES.

Code	Max Check-degree	Max Variable-degree	Max Throughput (Mbits/sec)	Simulation Time (hrs)
(4095,3358)	17	3	2	5.5
(4376,4094)	63	4	2.43	3
(4161,3430)	23	4	1.45	14

the decoder checks the syndrome after each iteration, the decoding is done after a couple of iterations for higher SNR regions and leads to higher throughput. It should be noted that the throughput of the simulator is less than the parallel architectures presented in the literature. However, the proposed simulator is able to evaluate any kind of LDPC code regardless of its structure.

V. CONCLUSION

In this paper, we have described the architecture and implementation of an embedded LDPC decoder for the simulation of unstructured LDPC codes. The decoder has a serial architecture and is working along with the embedded processor of the Virtex-II Pro FPGA. The processor manages the simulation by providing handshaking signals and memory data transfer as well as the communication with the PC while the peripherals implemented on the FPGA slices provide fast computations on the flow of data. Such a Hardware/Software co-design provides the flexibility of software and offers high throughput of the hardware.

The decoding algorithm used is a simplified version of the sum-product algorithm which belongs to the class of min-sum algorithms with correction factor. However, in the proposed algorithm, the correction factor is computed based on the LLR values and updates dynamically.

REFERENCES

- [1] R. G. Gallager, "Low-density parity-check codes," *Institute of Radio Engineers Transactions on Information Theory*, vol. IT-8, no. 1, pp. 21–28, 1962.
- [2] D. J. C. MacKay and R. M. Neal, "Near shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 33, no. 6, pp. 457–8, 1997.
- [3] M. M. Mansour and N. R. Shanbhag, "High-throughput ldpc decoders," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 11, no. 6, pp. 976–996, 2003.
- [4] J. Chen, A. Dholakia, E. Eleftheriou, M. P. C. A. F. M. P. C. Fossorier, and X. Y. A. H. X. Y. Hu, "Reduced-complexity decoding of ldpc codes," *Communications, IEEE Transactions on*, vol. 53, no. 8, pp. 1288–1299, 2005.
- [5] S. L. Howard, C. Schlegel, and V. C. Gaudet, "A degree-matched check node approximation for ldpc decoding," in *Information Theory, 2005. ISIT 2005. Proceedings. International Symposium on* (C. Schlegel, ed.), pp. 1131–1135, 2005.
- [6] C. Jinghu, R. M. Tanner, C. Jones, and A. Y. L. Yan Li, "Improved min-sum decoding algorithms for irregular ldpc codes," in *Information Theory, 2005. ISIT 2005. Proceedings. International Symposium on* (R. M. Tanner, ed.), pp. 449–453, 2005.
- [7] S. Papaharalabos, P. Sweeney, B. G. Evans, P. T. A. M. P. T. Mathiopoulos, G. A. A. G. Albertazzi, A. A. V.-C. A. Vanelli-Coralli, and G. E. A. C. G. E. Corazza, "Modified sum-product algorithms for decoding low-density parity-check codes," *Communications, IET*, vol. 1, no. 3, pp. 294–300, 2007.
- [8] G. Masera, F. Quaglio, and F. Vacca, "Implementation of a flexible ldpc decoder," *Circuits and Systems II: Express Briefs, IEEE Transactions on* [see also *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*], vol. 54, no. 6, pp. 542–546, 2007.
- [9] T. Mohsenin and B. M. Baas, "High-throughput ldpc decoders using a multiple split-row method," in *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on* (B. M. Baas, ed.), vol. 2, pp. II–13–II–16, 2007.
- [10] Z. Wang and Z. Cui, "Low-complexity high-speed decoder design for quasi-cyclic ldpc codes," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 15, no. 1, pp. 104–114, 2007.
- [11] Z. Wang and Z. Cui, "A memory efficient partially parallel decoder architecture for quasi-cyclic ldpc codes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 4, pp. 483–488, 2007.
- [12] L. Xiong, Z. Tan, and D. Yao, "The moderate-throughput and memory-efficient ldpc decoder," in *Signal Processing, The 8th International Conference on, 2006* (Z. Tan, ed.), vol. 3, 2006.
- [13] S. Lingyan, S. Hongwei, Z. Keirn, and B. V. K. V. A. K. B. V. K. V. Kumar, "Field programmable gate array (fpga) for iterative code evaluation," *Magnetics, IEEE Transactions on*, vol. 42, no. 2, pp. 226–231, 2006.
- [14] Z. Hao, Z. Tong, and F. H. Erich, "Quasi-cyclic ldpc codes for the magnetic recording channel: Code design and vlsi implementation," *Magnetics, IEEE Transactions on*, vol. 43, no. 3, pp. 1118–1123, 2007.
- [15] H. Xiao-Yu, E. Eleftheriou, D. M. Arnold, and A. A. D. A. Dholakia, "Efficient implementations of the sum-product algorithm for decoding ldpc codes," in *Global Telecommunications Conference, 2001. GLOBECOM '01. IEEE* (E. Eleftheriou, ed.), vol. 2, pp. 1036–1036E, 2001.
- [16] V. G. Oklobdzija, "An algorithmic and novel design of a leading zero detector circuit: comparison with logic synthesis," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 2, no. 1, pp. 124–128, 1994.
- [17] D. U. Lee, J. D. Villasenor, W. Luk, and P. H. W. A. L. P. H. W. Leong, "A hardware gaussian noise generator using the box-muller method and its error analysis," *Computers, IEEE Transactions on*, vol. 55, no. 6, pp. 659–671, 2006.
- [18] D. J. C. MacKay, "Encyclopedia of sparse graph codes." Online. Available: <http://www.inference.phy.cam.ac.uk/mackay/codes/data.html>.