A 54 MBPS $(3, 6)$-REGULAR FPGA LDPC DECODER

Tong Zhang and Keshab K. Parhi

Department of Electrical and Computer Engineering
University of Minnesota, Minneapolis, MN 55455, USA
E-mail:{tzhang, parhi}@ece.umn.edu

ABSTRACT

Applying a joint code and decoder design methodology, we develop a high-speed $(3, k)$-regular LDPC code partly parallel decoder architecture, based on which a 9216-bit, rate-$1/2$ $(3, 6)$-regular LDPC code decoder is implemented on Xilinx FPGA device. When performing maximum 18 iterations for each code block decoding, this partly parallel decoder supports a maximum symbol throughput of 54 Mbps and achieves BER 10^{-6} at 2dB over AWGN channel.

1. INTRODUCTION

Thanks to its excellent performance, Low-Density Parity-Check (LDPC) code [1][2] has been widely considered as a next-generation error-correcting code for telecommunication and magnetic storage. Defined as the null space of a very sparse $M \times N$ parity check matrix $\mathbf{H}$, an LDPC code is typically represented by a bipartite graph, called Tanner graph, in which one set of N *variable* nodes corresponds to the set of codeword, another set of M *check* nodes corresponds to the set of parity check constraints and each edge corresponds to a non-zero entry in the parity check matrix $\mathbf{H}$. An LDPC code is known as (j, k)-regular LDPC code if each column and each row in its parity check matrix have j and k non-zero entries, respectively. The construction of LDPC code is typically random. As illustrated in Fig. 1, LDPC code is decoded by the iterative belief-propagation (BP) algorithm [2] that directly matches its Tanner graph.

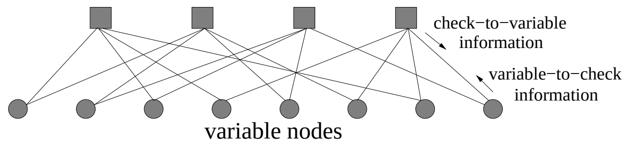


Fig. 1. Tanner graph representation of a LDPC code and the decoding message flow.

This research was supported by the Army Research Office by grant number DA/DAAG19-01-1-0705.

A fully parallel decoder is realized by directly instantiating the BP decoding algorithm to hardware. Such fully parallel decoder could achieve extremely high decoding speed, *e.g.,* a 1024-bit, rate-$1/2$ LDPC code fully parallel decoder [4] with the maximum symbol throughput of 1 Gbit/s has been implemented using ASIC technology. However, the primary disadvantage of fully parallel design is that with the increase of code length the hardware complexity will become more and more prohibitive for many practical purposes, *e.g.,* the ASIC LDPC decoder [4] with only 1K-bit code length consumes 1.7M gates. Moreover, as pointed out in [4], the routing overhead is quite formidable due to the large code length and randomness of the Tanner graph.

A *joint* code and decoder design methodology [5] was recently proposed for $(3, k)$-regular LDPC code and partly parallel decoder design to achieve appropriate trade-offs between hardware complexity and decoding throughput. In this paper, applying the proposed joint design methodology, we develop an elaborate $(3, k)$-regular LDPC code high-speed partly parallel decoder architecture based on which we implement a 9216-bit, rate-$1/2$ $(3, 6)$-regular LDPC code decoder using Xilinx Virtex FPGA device. We significantly modify the original decoder structure [5] to improve the decoding throughput and simplify the control logic design. We propose a novel *concatenated* scheme to realize the random connectivity by using two concatenated routing networks, where the random hardwire routings are localized to significantly reduce the routing overhead. Based on the post-routing static timing analysis, with the maximum 18 decoding iterations, this decoder supports a maximum symbol throughput of 54 Mbps and achieves BER 10^{-6} at 2dB over AWGN channel.

2. JOINT CODE AND DECODER DESIGN

In this section we briefly describe the joint $(3, k)$-regular LDPC code and decoder design methodology according to [5]. The essential objective of this joint design approach is to construct an LDPC code that not only fits to a high-speed partly parallel decoder but also has large average cycle length in its 4-cycle free Tanner graph. This joint design process is outlined as follows and the corresponding schematic flow diagram is shown in Fig. 2.

1. Construct two matrices, $\mathbf{H}_1$ and $\mathbf{H}_2$, in such a way that $[\mathbf{H}_1^T, \mathbf{H}_2^T]^T$ defines a $(2,k)$-regular LDPC code whose Tanner graph has the girth[1] of 12;
2. Develop a partly parallel decoder that is configured by some constrained random parameters and defines a $(3,k)$-regular LDPC code ensemble, in which each code has the parity check matrix $\mathbf{H} = [\mathbf{H}_1^T, \mathbf{H}_2^T, \mathbf{H}_3^T]^T$;
3. Select a good $(3,k)$-regular LDPC code from the code ensemble.

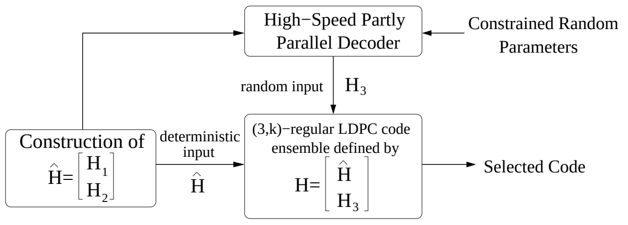

High−Speed Partly Parallel Decoder
Constrained Random Parameters
random input
H3
Construction of
Ĥ= [H1 H2]
deterministic input
Ĥ
(3,k)−regular LDPC code ensemble defined by
H= [Ĥ H3]
Selected Code

Fig. 2. Joint design flow diagram.

Construction of $\mathbf{H}_1$ and $\mathbf{H}_2$: As illustrated in Fig. 3, both $\mathbf{H}_1$ and $\mathbf{H}_2$ are $L \cdot k$ by $L \cdot k^2$ submatrices. Each block matrix $\mathbf{I}_{x,y}$ in $\mathbf{H}_1$ is an $L \times L$ identity matrix and each block matrix $\mathbf{P}_{x,y}$ in $\mathbf{H}_2$ is obtained by a cyclic shift of an $L \times L$ identity matrix. Let T denote the right cyclic shift operator where $T^u(\mathbf{Q})$ represents right cyclic shifting matrix $\mathbf{Q}$ by u columns, then $\mathbf{P}_{x,y} = T^u(\mathbf{I})$ where $u = ((x-1) \cdot y) \bmod L$ and $\mathbf{I}$ represents the $L \times L$ identity matrix. We can prove that $[\mathbf{H}_1^T, \mathbf{H}_2^T]^T$ defines a $(2,k)$-regular LDPC code whose Tanner graph has the girth of 12.

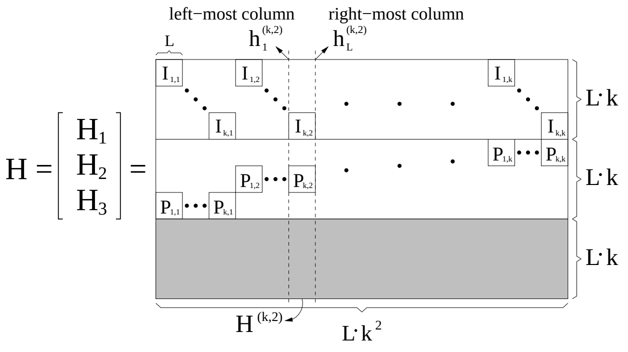


Fig. 3. The parity check matrix.

Partly Parallel Decoder: The authors presented a principal $(3,k)$-regular LDPC code partly parallel decoder structure in [5]. Configured by a set of constrained random parameters, this decoder defines a semi-random $(3,k)$-regular LDPC code ensemble in which each code is 4-cycle free and has the parity check matrix as shown in Fig. 3. For real applications, we can select a good code from this code ensemble based on average cycle length comparison combined with computer simulations. For the detailed description of this joint design methodology and the principal partly parallel decoder structure, readers are referred to [5].

[1]Girth is the length of a shortest cycle in a graph.

3. PARTLY PARALLEL DECODER ARCHITECTURE

Applying the joint design methodology, we develop a high-speed $(3,k)$-regular LDPC code partly parallel decoder architecture and implement a 9216-bit, rate-$1/2$ $(3,6)$-regular LDPC decoder using Xilinx Virtex FPGA device. Compared with the structure presented in [5], this partly parallel decoder architecture has the following distinct properties:

- A *concatenated* configurable random 2-D shuffle network implementation scheme is proposed to realize the random-like connectivity with minimized routing overhead, which is especially desirable for FPGA implementations;
- To improve the decoding throughput, this decoder contains k^2 *Variable Node processor Units* (VNU's) and $3k$ *Check Node processor Units* (CNU's);
- To simplify the control logic design and reduce the memory bandwidth requirement, this decoder completes each iteration in $2L$ clock cycles in which CNU's and VNU's work in the 1^{st} and 2^{nd} L clock cycles, alternatively.

This decoder defines a semi-random $(3,k)$-regular LDPC code ensemble in which each code has the parity check matrix as illustrated in Fig. 3. To facilitate the succeeding description, we introduce following definitions: Denote the submatrix consisting of the L consecutive columns in $\mathbf{H}$ that go through $\mathbf{I}_{x,y}$ as $\mathbf{H}^{(x,y)}$, in which from left to right each column is labeled as $h_i^{(x,y)}$ with i increasing from 1 to L, as shown in Fig. 3. We label the variable node corresponding to column $h_i^{(x,y)}$ as $v_i^{(x,y)}$ and the L variable nodes $v_i^{(x,y)}$ for $i = 1, \cdots, L$ constitute a variable node group VG$_{x,y}$. We arrange the $L \cdot k$ check nodes corresponding to all the $L \cdot k$ rows of submatrix $\mathbf{H}_i$ into check node group CG$_i$.

Fig. 4 shows the partly parallel decoder principal structure. It mainly contains k^2 PE Blocks PE$_{x,y}$ for $1 \le x, y \le k$, 3 bi-directional shuffle networks π_1, π_2 and π_3 and $3 \cdot k$ CNU's. Each PE$_{x,y}$ contains one memory bank RAMs$_{x,y}$ that stores all the decoding information associated with all the L variable nodes in the variable node group VG$_{x,y}$, and contains one VNU to perform the variable node computations for these L variable nodes. Each bi-directional shuffle network π_i realizes the decoding information exchange between all the $L \cdot k^2$ variable nodes and the $L \cdot k$ check nodes in CG$_i$. The k CNU$_{i,j}$'s for $j = 1, \cdots, k$ perform the check node computations for all the $L \cdot k$ check nodes in CG$_i$.

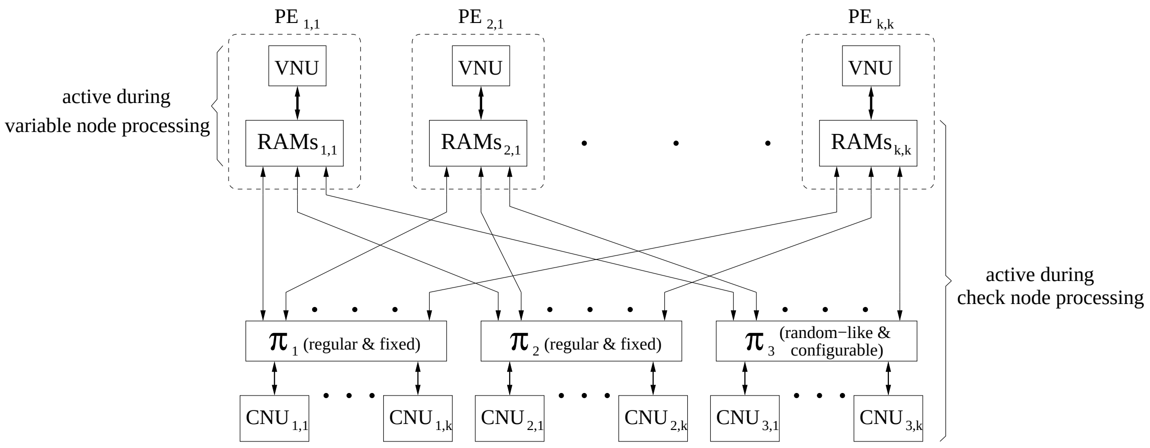


Fig. 4. The principal $(3, k)$-regular LDPC code partly parallel decoder structure.

This decoder completes each decoding iteration in $2L$ clock cycles. During the 1^{st} and 2^{nd} L clock cycles, it works in *check node processing* (CNP) mode and *variable node processing* (VNP) mode, respectively. In the CNP mode, the decoder performs both the computations of all the check nodes and the decoding information exchange between neighboring nodes. In the VNP mode, the decoder only performs the computations of all the variable nodes.

All the intrinsic, check-to-variable and variable-to-check information are quantized to 5 bits. The iterative decoding datapath is illustrated in Fig. 5, where the datapath in CNP mode and VPN mode are represented by solid lines and dash dot lines, respectively. As shown in Fig. 5, each PE Block PE$_{x,y}$ contains five RAM blocks: EXT_RAM_i for $i = 1, 2, 3$, INT_RAM and DEC_RAM. Each EXT_RAM_i has L memory locations and the location with the address $d-1$ $(1 \leq d \leq L)$ contains the decoding information exchanged between the variable node $v_d^{(x,y)}$ in VG$_{x,y}$ and its neighboring check node in CG$_i$. The INT_RAM and DEC_RAM store the intrinsic information and hard decision associated with node $v_d^{(x,y)}$ at the memory location with the address $d-1$ $(1 \leq d \leq L)$. As we will see later, such decoding information storage strategy greatly simplifies the control logic for generating the memory access address.

3.1. Check node processing

In the CNP mode, decoder performs the computations of all the check nodes and decoding information exchange between neighboring nodes. At the beginning, in each PE$_{x,y}$ the memory location with address $d-1$ in EXT_RAM_i contains 6-bit *hybrid* data consisting of 1-bit hard decision and 5-bit variable-to-check information associated with the variable node $v_d^{(x,y)}$. Each clock cycle this decoder performs the *read-shuffle-modify-unshuffle-write* operations to *convert* one variable-to-check information in each EXT_RAM_i to its check-to-variable counterpart. We outline the datapath loop in CNP mode as follows: 1. *Read*: One 6-bit hybrid data $h_{x,y}^{(i)}$ is read from each EXT_RAM_i; 2. *Shuffle*: Each $h_{x,y}^{(i)}$ goes through the shuffle network π_i and arrives CNU$_{i,j}$; 3. *Modify*: Each CNU$_{i,j}$ performs the parity check on the 6 input hard decision bits and generates the 6 output 5-bit check-to-variable information $\alpha_{x,y}^{(i)}$; 4. *Unshuffle*: Send each $\alpha_{x,y}^{(i)}$ back to the PE Block via the same path as its variable-to-check counterpart; 5. *Write*: Write check-to-variable information $\alpha_{x,y}^{(i)}$ to the same memory location in EXT_RAM_i as its variable-to-check counterpart.

We implement each bi-directional I/O connection in the 3 shuffle networks by two distinct sets of wires with opposite directions so that the hybrid data from PE Blocks to CNU's and the check-to-variable information from CNU's to PE Blocks are carried on distinct set of wires. Compared with sharing one set of wires in time-multiplexed fashion, this approach has higher wire routing overhead but eliminates the logic gate overhead due to the realization of time-multiplex and, more importantly, make it feasible to directly pipeline the datapath loop for higher decoding throughput.

Each EXT_RAM_i associates with one address generator AG$_{x,y}^{(i)}$ that provides the read address in each clock cycle. The write address for writting the check-to-variable information is obtained via delaying the read address by the pipelining stages of the datapath loop. The connectivity among all the variable nodes and check nodes realized by this decoder is jointly specified by all the address generators and the 3 shuffle networks. Moreover, for $i = 1, 2, 3$, submatrix $\mathbf{H}_i$ or the connectivity among all the variable nodes and the check nodes in CG$_i$ is completely determined by all AG$_{x,y}^{(i)}$'s and π_i.

Implementations of AG$_{x,y}^{(i)}$ and π_i for $i = 1, 2$: Recall that node $v_d^{(x,y)}$ corresponds to the column $h_i^{(x,y)}$ as illustrated in Fig. 3 and the decoding information associated with node

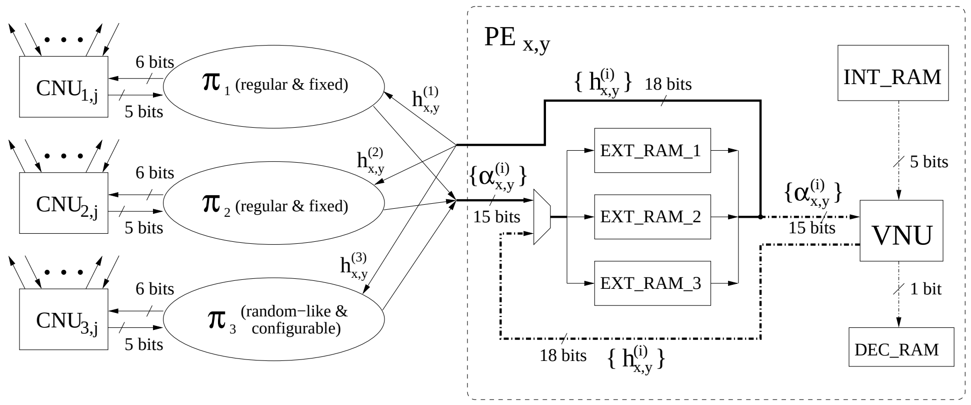


Fig. 5. Iterative decoding datapath.

$v_d^{(x,y)}$ is always stored at address $d-1$. Exploiting the explicit structure of $\mathbf{H}_1$ and $\mathbf{H}_2$, we have

- Each $\mathrm{AG}_{x,y}^{(1)}$ is realized as a $\lceil \log_2 L \rceil$-bit binary counter that is cleared to zero at the beginning of CNP mode;
- The shuffle network π_1 connects the k $\mathrm{PE}_{x,y}$'s with the same x-index to the same CNU;
- $\mathrm{AG}_{x,y}^{(2)}$ as a $\lceil \log_2 L \rceil$-bit binary counter that only counts up to the value $L-1$ and is loaded with the value of $((x-1)\cdot y) \bmod L$ at the beginning of CNP mode;
- The shuffle network π_2 connects the k $\mathrm{PE}_{x,y}$'s with the same y-index to the same CNU.

Notice that the counter load value for each $\mathrm{AG}_{x,y}^{(2)}$ comes from the construction of each block matrix $\mathbf{P}_{x,y}$ in $\mathbf{H}_2$.

Implementations of $\mathbf{AG}_{x,y}^{(3)}$ and π_3: The bi-directional shuffle network π_3 and $\mathrm{AG}_{x,y}^{(3)}$'s jointly define the connectivity between all the variable ndoes and check nodes in CG_3, which is represented by $\mathbf{H}_3$. The design of each $\mathrm{AG}_{x,y}^{(3)}$ and π_3 are not trivial because of the following requirements:

- The parity check matrix $\mathbf{H} = [\mathbf{H}_1^T, \mathbf{H}_2^T, \mathbf{H}_3^T]^T$ should correspond to a 4-cycle freeTanner graph;
- To make $\mathbf{H}$ be random to some extent, $\mathbf{H}_3$ should be random-like.

To simplify the design process, we *separately* conceive $\mathrm{AG}_{x,y}^{(3)}$'s and π_3 so that the design of $\mathrm{AG}_{x,y}^{(3)}$'s and π_3 accomplish the above 1^{st} and 2^{nd} requirement, respectively.

Implementations of $\mathbf{AG}_{x,y}^{(3)}$: We implement each $\mathrm{AG}_{x,y}^{(3)}$ as a $\lceil \log_2 L \rceil$-bit binary counter that counts up to the value $L-1$ and loads a constant value $t_{x,y}$ at the beginning of CNP mode. Each $t_{x,y}$ is generated in random under the following two constraints:

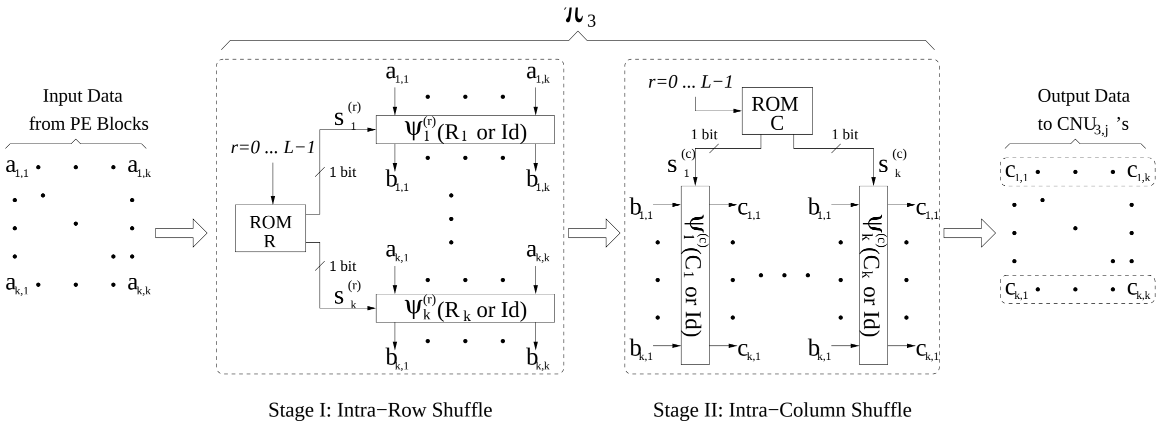
1. Given x, we have $t_{x,y_1} \neq t_{x,y_2}, \forall y_1, y_2 \in \{1, \cdots, k\}$;
2. Given y, we have $t_{x_1,y} - t_{x_2,y} \not\equiv ((x_1 - x_2)\cdot y) \bmod L$, $\forall x_1, x_2 \in \{1, \cdots, k\}$.

We can prove that the above constrains are sufficient to make $\mathbf{H}$ always correspond to a 4-cycle free Tanner graph no matter how we implement π_3.

Implementation of π_3: We develop a novel concatenated configurable random shuffle network implementation scheme for π_3 as described in the following.

Fig. 6 shows the forward path (from $\mathrm{PE}_{x,y}$ to $\mathrm{CNU}_{3,j}$) of the bi-directional shuffle network π_3. In each clock cycle, it realizes the data shuffle from $a_{x,y}$ to $c_{x,y}$ by two concatenated stages: *intra-row* shuffle and *intra-column* shuffle. First, the $a_{x,y}$ data block, where each $a_{x,y}$ comes from $\mathrm{PE}_{x,y}$, passes an *intra-row* shuffle network array in which each shuffle network $\Psi_x^{(r)}$ shuffles the k input data $a_{x,y}$ to $b_{x,y}$ for $1 \le y \le k$. Each $\Psi_x^{(r)}$ is configured by 1-bit control signal $s_x^{(r)}$ leading to the fixed random permutation R_x if $s_x^{(r)} = 1$, or to the identity permutation (Id) otherwise. The k-bit configuration word $\mathrm{s}^{(r)}$ changes every clock cycle and all the L k-bit control words are stored in ROM R. Next, the $b_{x,y}$ data block goes through an *intra-column* shuffle network array in which each $\Psi_y^{(c)}$ is configured by 1-bit control signal $s_y^{(c)}$ and shuffles the k data $b_{x,y}$ to $c_{x,y}$ for $1 \le x \le k$. The k-bit configuration word $s_y^{(c)}$ changes every clock cycle and all the L k-bit control words are stored in ROM C. As the output of forward path, the k $c_{x,y}$'s with the same x-index are delivered to the same $\mathrm{CNU}_{3,j}$. To realize the bi-directional shuffle, we only need to implement each configurable shuffle network $\Psi_x^{(r)}$ and $\Psi_y^{(c)}$ as bi-directional so that π_3 can *unshuffle* the k^2 data backward from $\mathrm{CNU}_{3,j}$ to $\mathrm{PE}_{x,y}$ along the same route as the forward path on distinct sets of wire.

To make the connectivity realized by π_3 be random-like and change each clock cycle, we randomly generate the control word $s_x^{(r)}$ and $s_y^{(c)}$ for each clock cycle and each R_x and C_y. Since most modern FPGA devices have multiple metal layers, the implementations of the two shuffle arrays can be overlapped from the bird's-eye view. Therefore, such concatenated implementation scheme confines all the routing wires to small area (in one row or one column), which will significantly reduce the routing overhead.



π_3
Input Data from PE Blocks
Output Data to CNU$_{3,j}$'s
r=0 ... L-1
ROM R
ROM C
1 bit
Stage I: Intra−Row Shuffle
Stage II: Intra−Column Shuffle

Fig. 6. Forward path of π_3.

3.2. Variable node processing

The operations performed in the variable node processing is quite simple since the decoder only performs all the variable node computations. At the beginning of variable node processing, the 3 5-bit check-to-variable information associated with each variable node $v_d^{(x,y)}$ are stored at the address $d-1$ of the 3 EXT_RAM_i's in PE$_{x,y}$. The 5-bit intrinsic information associated with variable node $v_d^{(x,y)}$ is also stored at the address $d-1$ of INT_RAM in PE$_{x,y}$. As illustrated in Fig. 5, in each clock cycle, this decoder performs the *read-modify-write* operations to *convert* the 3 check-to-variable information associated with the same variable node to 3 hybrid data consisting of variable-to-check information and hard decision.

3.3. Data Input/Output

This decoder works simultaneously on 3 consecutive code frames in two-stage pipelining mode: while one frame is being iteratively decoded, the next frame is loaded into the decoder and the hard decisions of the previous frame are read out from the decoder. Thus each INT_RAM contains two RAM blocks to store the intrinsic information of both current and next frames. Similarly, each DEC_RAM contains two RAM blocks to store the hard decisions of both current and previous frames.

The intrinsic information input and hard decision output schemes are heavily dependent on the floor planning of the k^2 PE Blocks. To minimize the routing overhead, we develop a square-shaped floor planning as illustrated in Fig. 7 and the data I/O scheme is described in the following:

Intrinsic Data Input: The intrinsic information of next frame is loaded 1 symbol per clock cycle. As shown in Fig. 7, the memory location of each input intrinsic data is determined by the input load address that has the width of $(\lceil\log_2 L\rceil + \lceil\log_2 k^2\rceil)$ bits in which $\lceil\log_2 k^2\rceil$ bits specify

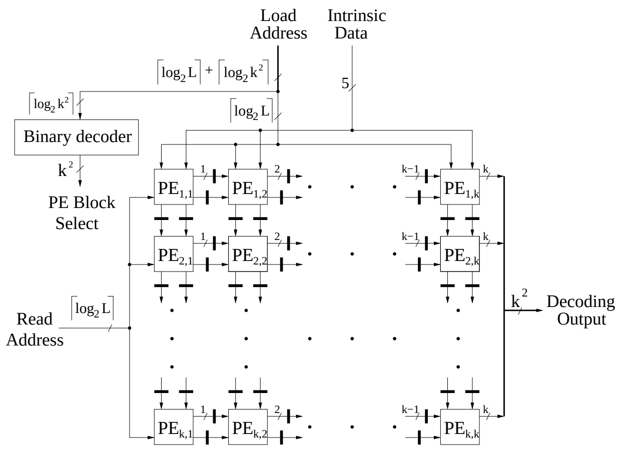


Fig. 7. Data Input/Output structure.

which PE Block is being accessed and the other $\lceil\log_2 L\rceil$ bits represent the memory location in the INT_RAM. The primary intrinsic data and load address input directly connect to the k PE Blocks PE$_{1,y}$ for $1 \le y \le k$, and from each PE$_{x,y}$ the intrinsic data and load address are delivered to the adjacent PE Block PE$_{x+1,y}$ in pipelined fashion.

Decoded Data Output: As shown in Fig. 7, the primary $\lceil\log_2 L\rceil$-bit read address input directly connects to the k PE Blocks PE$_{x,1}$ for $1 \le x \le k$, and from each PE$_{x,y}$ the read address are delivered to the adjacent block PE$_{x,y+1}$ in pipelined fashion. Each PE Block outputs 1-bit hard decision per clock cycle. Therefore, as illustrated in Fig. 7, the width of pipelined decoded data bus increases by 1 after going through one PE Block, and at the rightmost side, we obtain k k-bit decoded output that are combined together as the k^2-bit primary data output.

4. FPGA IMPLEMENTATION

Based on the above architecture, we implemented a $(3, 6)$-regular LDPC code partly parallel decoder for $L = 256$ using Xilinx Virtex-E XCV2600E device. The LDPC code length is $N = L \cdot k^2 = 256 \cdot 6^2 = 9216$ and code rate is $1/2$. The target XCV2600E FPGA device contains 184 on-chip block RAMs, each one is a dual-port 4K-bit RAM. We configure each dual-port 4K-bit RAM as two independent single-port 256×8-bit RAM blocks so that each EXT_RAM_i can be realized by one single-port 256×8-bit RAM block. Since each INT_RAM contains two RAM blocks for storing the intrinsic information of both current and next code frame, we use two single-port 256×8-bit RAM blocks to implement one INT_RAM. The DEC_RAM is realized by distributed RAM that provides shallow RAM structures implemented in CLBs. Because all the RAM blocks have fixed locations, the placement of the decoder is primarily carried out based on the RAM block locations and we manually configured the placement of each PE Block according to the floor planning scheme as shown in Fig. 7. Notice that such placement scheme exactly matches the structure of the configurable shuffle network π_3.

Table 1. FPGA resources utilization statistics.

Resource	Number	Utilization rate
Slices	11,792	46%
Slices Reg.	10,105	19%
4 input LUTs	15,933	31%
IOBs	68	8%
Block RAMs	90	48%

This decoder is described in VHDL and SYNOPSYS FPGA Express was used to synthesize the VHDL implementation. The Xilinx Development System tool suite was used to place and route the synthesized implementation for the target XCV2600E device with the speed option -7. The resource utilization statistics are listed in Table 1. Notice that 74% of the total utilized slices, or 8691 slices, are used for implementing all the CNU's and VNU's.

The post-routing static timing analysis results suggest that the maximum decoder clock frequency can be 56 MHz. If this decoder performs s decoding iterations for each code frame, the total clock cycles for decoding one frame will be $2s \cdot L + L$ where the extra L clock cycles is due to the initialization process, and the maximum symbol decoding throughput will be $56 \cdot k^2 \cdot L/(2s \cdot L + L) = 56 \cdot 36/(2s + 1)$ Mbps. If we set $s = 18$, the maximum symbol decoding throughput is 54 Mbps. Fig. 8 shows the corresponding performance over AWGN channel with $s = 18$, including the BER (Bit Error Rate), FER (Frame Error Rate) and the average iteration numbers.

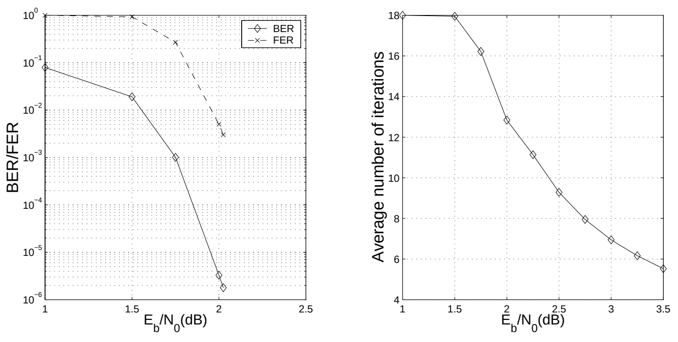


Fig. 8. Simulation results on BER (Bit Error Rate), FER (Frame Error Rate) and the average iteration numbers.

5. CONCLUSION

Following a joint design methodology, we develop a $(3, k)$-regular LDPC code high-speed partly parallel decoder architecture and implement a 9216-bit, rate-$1/2$ $(3, 6)$-regular LDPC decoder on the Xilinx XCV2600E FPGA device. The detailed decoder architecture has been presented. With the maximum 18 decoding iterations, this decoder can achieve upto 54Mbps symbol decoding throughput and the BER 10^{-6} at 2dB over AWGN channel.

6. REFERENCES

[1] R. G. Gallager, *Low-Density Parity-Check Codes*, M.I.T Press, 1963. available at http://justice.mit.edu/people/gallager.html.

[2] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Transactions on Information Theory*, vol. 45, pp. 399–431, Mar. 1999.

[3] T. Zhang, Z. Wang, and K. K. Parhi, "On finite precision implementation of low-density parity-check codes decoder," in *Proc. of 2001 IEEE Int. Symp. on Circuits and Systems*, Sydney, May 2001. available at http://www.ece.umn.edu/groups/ddp/turbo/.

[4] A. J. Blanksby and C. J. Howland, "A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity-check code decoder," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 3, pp. 404–412, March 2002.

[5] T. Zhang and K. K. Parhi, "VLSI implementation-oriented $(3, k)$-regular low-density parity-check codes," IEEE Workshop on Signal Processing Systems (SiPS), Sept. 2001. available at http://www.ece.umn.edu/groups/ddp/turbo/.

[6] T. Zhang and K. K. Parhi, "Joint code and decoder design for implementation-oriented $(3, k)$-regular ldpc codes," in *Proc. of IEEE Asilomar Conference*, Nov. 2001, pp. 1232–1236.