Code Construction and FPGA Implementation of a Low-Error-Floor Multi-Rate Low-Density Parity-Check Code Decoder

Lei Yang, *Student Member, IEEE*, Hui Liu, *Senior Member, IEEE*, and C.-J. Richard Shi, *Fellow, IEEE*

***Abstract*—With the superior error correction capability, low-density parity-check (LDPC) codes have initiated wide scale interests in satellite communication, wireless communication, and storage fields. In the past, various structures of single code-rate LDPC decoders have been reported. However, to cover a wide range of service requirements and diverse interference conditions in wireless applications, LDPC decoders that can operate at both high and low code rates are desirable. In this paper, a 9-k code length multi-rate LDPC decoder architecture is presented and implemented on a Xilinx field-programmable gate array device. Using pin selection, three operating modes, namely, the irregular 1/2 code mode, the regular 5/8 code mode, and the regular 7/8 code mode, are supported. Furthermore, to suppress the error floor level, a characterization on the conditions for short cycles in a LDPC code matrix expanded from a small base matrix is presented, and a cycle elimination algorithm is developed to detect and break such short cycles. The effectiveness of the cycle elimination algorithm has been verified by both simulation and hardware measurements, which show that the error floor is suppressed to a much lower level without incurring any performance penalty. The implemented decoder is tested in an experimental LDPC orthogonal frequency division multiplexing system and achieves the superior measured performance of block error rate below 10^{-7} at signal-to-noise ratio of 1.8 dB.**

***Index Terms*—Block-error rate, channel encoding, cycle elimination, forward error correction (FEC), field-programmable gate array (FPGA), low-density parity-check (LDPC) codes, multi-rate, orthogonal frequency division multiplexing (OFDM), signal-to-noise ratio (SNR), VLSI.**

I. Introduction

RECENTLY, low-density parity-check (LDPC) codes have attracted an ever increasing amount of attention due to their superior error correction capacity. It has been shown that with the block length 10^7, it is possible to achieve 0.04 dB from the Shannon limit at a bit-error rate (BER) of 10^{-6} [1]; this yields significant advantages over other forward error correction codes including turbo codes. Furthermore, the LDPC decoding algorithm is inherently parallel and is easy to be implemented.

Manuscript received January 26, 2005; revised July 9, 2005. This work was supported in part by Defense Advance Research Projects Agencies under Grant 66001-05-1-8918 and in part by the National Scienc Foundation ITR Program under Grant 0086032. This paper was recommended by Associate Editor Z. Wang.

The authors are with the Department of Electrical Engineering, University of Washington, Seattle, WA 98195 USA (e-mail: yanglei@ee.washington.edu; hliu@ee.washington.edu; shi@ee.washington.edu).

Digital Object Identifier 10.1109/TCSI.2005.862074

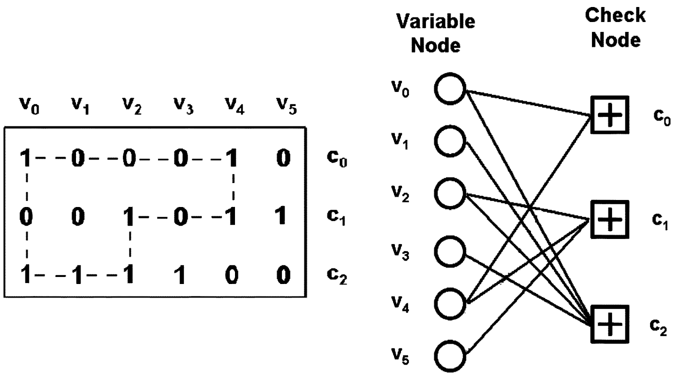

Fig. 1. Irregular H-matrix and its corresponding Tanner graph.

Therefore, it can be applied in optical networking, magnetic recoding, digital video broadcast satellite (DVB-S) communications, and other fields [2]–[4].

As illustrated in Fig. 1, an LDPC code is a linear block code described by a binary sparse $M \times N$ parity-check matrix H. Each row of matrix H corresponds to a parity check and each column represents a demodulated symbol. The number of demodulated symbols N is the LDPC *code length*. The number of nonzero elements in a row (column) is defined as the row (column) weight d_C (d_V). If all rows and all columns are of uniform weight, the LDPC code is called a *regular* code, otherwise an *irregular* code. The notion of Tanner graph has been introduced to represent LDPC codes. As illustrated in Fig. 1, a Tanner graph is a bipartite graph with *variable* nodes on one side and *check* nodes on the other side. Each variable node corresponds to a received symbol, each check node corresponds to a particular set of parity-check constraints, and each edge corresponds to a nonzero entry in the parity-check matrix. For example, an irregular LDPC H-matrix and its corresponding Tanner graph are shown in Fig. 1. A *cycle* of *length* $2g$ is defined as a closed path that traversals a set of g variable nodes and g check nodes through edges but without going through the same edge twice. For example, Fig. 1 contains a length-6 cycle $(v_0, c_0, v_4, c_1, v_2, c_2, v_0)$.

Typically, iterative belief propagation (*BP*) algorithms are exploited to decode the LDPC codes. In the decoding process, messages are exchanged along the graph edges, and computed at the variable/check nodes. If an LDPC code is well designed, the messages will converge exponentially to the correct bits after a finite number of iterations. However, proper operations of the iterative BP algorithms hinges on the assumption that neighbors of a node in the Tanner graph are *conditionally independent*. If there are too many short cycles in the graph, the above assumption is violated, and the BP algorithms will converge slowly and even result in wrong decoded results. Hence, how to increase the Tanner graph *girth*, which is defined as the length of the shortest cycles, is critically important in the construction of LDPC codes. Several algorithms, such as bit-filling [5] and progressive edge-growth (PEG) [6], have been proposed to improve the girth of a Tanner graph. But those algorithms are developed with no consideration of the decoder VLSI implementation. Consequently, the resulting LDPC matrix with randomly located nonzeros elements is hard to be mapped to a hardware structure.

To design LDPC codes that are amenable to VLSI implementation, several methods [7]–[10] have been developed to jointly consider the code design and the hardware implementation. The basic idea of all these methods is that before constructing an $M \times N$ parity-check matrix, a small $M_b \times N_b$ *base matrix* is built, in which each nonzero element at the position (i, j) is to be expanded to a square permutation $L \times L$ matrix ($M = M_b \cdot L$, $N = N_b \cdot L$). The square permutation matrix is an identity matrix whose rows have been *cyclically* shifted by a set of amount $P_{i,j}$, where $P_{i,j}$ is a function of i and j. The resulting $M \times N$ matrix is referred to as the *expanded matrix*. This LDPC matrix structure can be conveniently converted to a partially parallel decoder architecture consisting of a structured array of memory and computation units [7].

The above VLSI-oriented LDPC codes can achieve the gain performance as good as the random codes. As for the girth performance, [7]–[9] employ different functions to compute the cyclic shifted value $P_{i,j}$. Those functions can ensure that the girth of the expanded matrix is no more than that of the base matrix [7], but cannot guarantee the girth of the expanded LDPC matrix to be improved to a user specified value. In our research, we observed that short cycles in the base matrix can result in a collection of the same length cycles in the expanded matrix. Furthermore, we characterize the necessary and sufficient conditions for such scenarios to occur. With this characterization, we develop a *cycle elimination (CE)* algorithm. In this algorithm, short cycles existing in the base matrix are checked first. Then through setting the cyclically shifted values $P_{i,j}$, those short cycles will not incur the same length cycles in the expanded matrix, thus yielding high girth performance. The cycle elimination process carries on until the maximum cyclically shifted $P_{i,j}$ value exceeds its upper limit $L-1$ (L is the size of a square permutation matrix). With this, even if a base matrix contains potentially many short cycles, the girth of the expanded matrix can be improved to be cycle-6 free, cycle-8 free, or even cycle-10 free. This is very helpful to suppress the error floor to a much lower level.

Even if hardware-oriented LDPC codes with high girth performance have been achieved, efficiently mapping them to a VLSI implementation remains to be challenging. On one hand, the architecture of LDPC codes is characterized by many parallel memories and a lot of long interconnect wires; On the other hand, various LDPC applications require diverse dominant performances, such as high throughputs, low area, and low power. Several approaches implement the BP algorithm to hardware in a way that each variable node is mapped to a variable-node processor, each check node is mapped to a check-node processor, and all the processors are connected through a Tanner graph interconnection network. This architecture gains high parallelism and high throughputs [11], but is not scalable to LDPC codes with large code length due to the heavy burden of interconnect wires. The excessive amount of interconnection could lead to routing congestion, which might exceed the available chip routing area. To elude this problem, partially parallel architectures compromising between the operating speed and interconnection complexity are introduced, in which a certain number of variable nodes and check nodes are mapped to one hardware unit in the time-division multiplexing mode [12]. This approach eases the routing difficulty but at the expense of decreasing parallelism. Furthermore, all the reported implementations up to now are limited to single-rate LDPC code designs. In practice, especially for wireless applications, it is highly desirable that a design scheme could adapt to different coding rates to meet various service requirements and interference conditions.

In this paper, a multi-rate LDPC architecture is presented. This architecture is not only suitable for different regular LDPC coding rates, but also can support irregular LDPC codes. The proposed architecture is demonstrated by implementing a 9 k bit, multi-rate LDPC decoder in a Xilinx FPGA device. By configuring two pins of the FPGA device at “00,” “01,” or “10,” the decoder works on three coding rate modes: 1/2 as irregular code, 5/8 and 7/8 as regular codes constructed by employing the CE algorithm to suppress the error floor. The designed decoder is tested in an LDPC plus orthogonal-frequency-division-multiplexing (OFDM) system [13] and achieves the superior measured performance of block error rate below 10^{-7} at a signal-to-noise ratio (SNR) of 1.8 dB without the presence of error floor.

Some primary results of this work have appeared in [14]–[16]. The rest of this paper is organized as follows. An overview of the LDPC decoding algorithm is given in Section II. The proposed cycle elimination algorithm is presented in Section III. The three rate codes design process and simulation results are described in Section IV. The architecture of the multi-rate decoder is presented in Section V. Measurement results of the implemented FPGA device are shown in Section VI. Concluding remarks are made in Section VII.

II. LDPC Decoding Algorithm

For hardware implementation, the BP algorithms are often re-formulated as the Log-BP algorithms [17], [18], in which multiplication operations are converted to addition operations to decrease the computational complexity. In the following, μ represents the log-likelihood ratio (LLR) messages exchanged between variable nodes and check nodes, and γ_i stands for intrinsic probability for every bit from a demodulator. The LDPC decoding algorithm can be summarized in the following four major steps.

1) **Initialization**

All variables nodes and their outgoing variable messages are initialized to the values of the intrinsic messages. The intrinsic message is defined as

$$\gamma_i = \log \left[\frac{P(x_i = 0|y_i)}{P(x_i = 1|y_i)} \right]. \tag{1}$$

Here, y_i is the received symbol and x_i is the transmitted symbol.

2) **Check-Node Computation**

After the incoming messages are gathered in each check node from its connected variable nodes in the Tanner graph, the following check-node computation is performed:

$$\mu_{CV} = \operatorname{sgn}\left(\prod_{j=1}^{d_C-1} \mu_{V_jC}\right) \Psi^{-1}\left(\sum_{j=1}^{d_C-1} \Psi\left(\mu_{V_jC}\right)\right). \quad (2)$$

Here, d_C is the degree of the check node C, μ_{V_jC} represents the incoming message from neighbor variable node $V_j \neq V$ to check node C, and μ_{CV} is the outgoing message from check node C. Function Ψ is equal to Ψ^{-1}, and is expressed in the following equation:

$$\Psi(x) = \Psi^{-1}(x) = \log\left(\frac{1+\exp(-|x|)}{1-\exp(-|x|)}\right). \quad (3)$$

After the check-node computation, the outgoing messages are passed to variable nodes along the edges.

3) **Variable-Node Computation**

The variable-node computation is expressed as follows:

$$\mu_{VC} = \sum_{i=0}^{d_V-1} \mu_{C_iV} \quad (4)$$

where μ_{C_iV} is the incoming message from the neighbor check node $C_i \neq C$ to variable node V and d_V is the number of check nodes connected to V, and μ_{VC} is the outgoing message from variable node V.

4) **Check Stop Criterion**

When the variable-node computation is finished, the LLR of every symbol i is updated as

$$\lambda_i = \gamma_i + \sum_{i=0}^{d_V-1} \mu_{C_iV}. \quad (5)$$

From the updated LLR vector $\lambda = \{\lambda_1, \lambda_2, \ldots \lambda_i, \ldots \lambda_N\}$, a hard decision result $X = \{x_1, x_2, \ldots x_i, \ldots x_N\}$ is calculated as

$$x_i = \begin{cases} 1, & if\ \lambda_i \le 0 \\ 0, & if\ \lambda_i > 0. \end{cases} \quad (6)$$

The calculated hard decision vector X is then checked against the parity check matrix H. A case of $H \cdot X = 0$ means the iterative process has converged to a codeword and decoding stops. Otherwise, steps 2) and 3) have to be repeated until $H \cdot X = 0$ or until a fixed number of iterations is reached.

III. Cycle Elimination Algorithm

A. Idea

Suppose that we already constructed a base matrix shown in Fig. 2, which is a 6 × 12 irregular matrix with variable-node degree {2,2,2,2,2,2,3,3,3,3,4,4} and check-node degree {5,3,6,6,6,6}. If we employ the base matrix expansion method in [7] to construct a large LDPC H-matrix, every 1 at position (i,j) is expanded to a square permutation $L \times L$ (here $L = 453$) matrix, each of which is an identity matrix with rows cyclically shifted by a set of amount $P_{i,j}$. The size of the generated LDPC H-matrix is 2718 × 5436. It is obvious that the above matrix expansion is beneficial in terms of error correction performance because the girth of the expanded matrix is always larger than or equal to the girth of the base matrix [7]. However, in order to further improve the girth of the expanded matrix, we need to study the relationship between the cycles in the expanded matrix and the cycles in the base matrix. Two issues arise here.

1) Will the short cycles in a base matrix result in a collection of same length cycles in the expanded H-matrix?
2) If the answer is yes, then how to eliminate those short cycles?

The above two questions are answered in the following, which form the basic idea behind the proposed CE algorithm.

In fact, if there exists a length-$2g$ cycle (it has $2g$ 1's in the cycle path) in the base matrix, this cycle can result in L length-$2g$ cycles in the expanded LDPC matrix. For example, in Fig. 3, the four 5 × 5 square matrices represent the square permutation matrices expanded from a length-4 cycle in the base matrix with cyclically shifted values $P_{1,1} = 1$, $P_{1,2} = 0$, $P_{2,1} = 4$, $P_{2,2} = 3$. Clearly, there are five length-4 cycles after the expansion.

Consider two adjacent 1's in the same row of a base matrix located respectively at (i,j) and (i,m). Let their corresponding permutation matrices to be $H_{i,j}$ and $H_{i,m}$, and their respective shifted values to be $P_{i,j}$ and $P_{i,m}$. Then let $\Delta P_{i,j->i,m} = P_{i,j} - P_{i,m}$ *to be the (cyclically) shifted-value drop* from the 1 at location (i,j) to the 1 at location (i,m). As illustrated in Fig. 4, the shifted-value drop from the 1 at position (1,1) to its neighbor at (1,2) is $\Delta P_{1,1->1,2} = P_{1,1} - P_{1,2} = 1 - 0 = 1$.

Furthermore, for two adjacent 1's in the same row (column), we refer to its shifted-value drop as a *horizontal (vertical)*

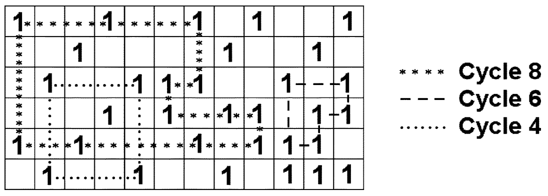

Fig. 2. Base matrix with cycles.

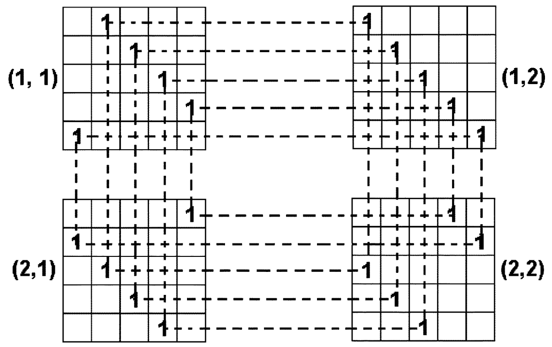

Fig. 3. Illustration of the number of cycles increase after expansion.

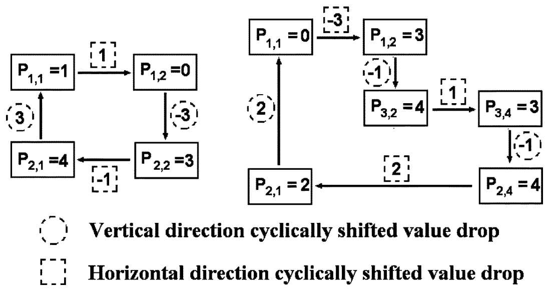
P1,1 =1
1
P1,2 =0
3
-3
P2,1 =4
-1
P2,2 =3
P1,1 =0
-3
P1,2 =3
-1
2
P3,2 =4
1
P3,4 =3
-1
2
P2,1 = 2
P2,4 =4
Vertical direction cyclically shifted value drop
Horizontal direction cyclically shifted value drop

Fig. 4. Another illustration of the cycle number increase after expansion.

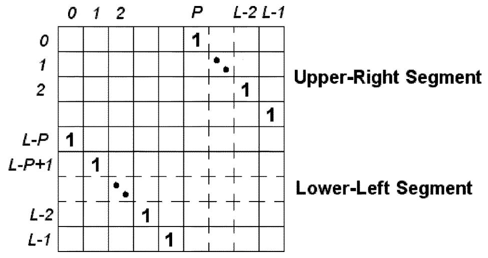

Fig. 5. Square permutation $L \times L$ matrix with row cyclically shifted P.

shifted-value drop. For example, in Fig. 4, along each edge, the number inside a dotted cycle is a vertical shifted-value drop, and the number inside a dotted box is a horizontal shifted-value drop.

Consider a cycle in a base matrix. Let $\Sigma_V(\Delta P)$ to be the sum of all the vertical shifted-value drops along the cycle. Similarly, let $\Sigma_H(\Delta P)$ to be the sum of all the horizontal shifted-value drops along the cycle. For example, in Fig. 4, the left picture and right picture in represent a length-4 cycle and a length-6 cycle in the base matrix. The sum of all the vertical shifted-value drops for the cycle-4 can be computed as: $\Sigma_V(\Delta P) = (-3) + 3 = 0$. For the cycle 6, we have $\Sigma_V(\Delta P) = (-1) + (-1) + 2 = 0$. Obviously in this case, if each 1 in the cycle-4/cycle-6 is expanded to a $L \times L$ square permutation matrix, L length-4/length-6 cycles will be generated in the expanded matrix.

In general, we have the following result.

Lemma: Any cycle of length $2g$ in a base matrix will lead to L cycles of length $2g$ in the expanded matrix, if and only the sum of all the vertical (horizontal) shifted-value drops along the cycle is equal to kL, where $k = 0, \pm 1, \pm 2 \ldots \pm g$.

Proof: We only need to prove the lemma for the vertical shifted-value drops case, since the horizontal case is symmetric to the vertical case.

Sufficient Condition (if part): We need to show that if a length-$2g$ cycle exists in the $2g$ permutation matrices expanded from a length-$2g$ cycle in the base matrix, the sum of all vertical shifted-value drops along the cycle $\Sigma_V(\Delta P) = k \cdot L$ (here $k = 0, \pm 1, \pm 2 \ldots g$).

First, consider a square permutation $L \times L$ matrix whose row cyclically shifted value is P, as illustrated in Fig. 5. The 1's in the matrix are divided into two parts—the upper-right segment and lower-left segment. The coordinates of the 1's are expressed symbolically as (7), shown at the bottom of the page. It can be re-written in the following *compact* form:

$$(i, i + P - 0//L). \tag{8}$$

Here, i represents the row index, and the notation $//$ represents "or."

Now consider the expanded matrix consisting of $2g$ such permutation $L \times L$ matrices $H_{1,1}, H_{1,2}, H_{2,1}, H_{2,2} \ldots H_{g,1}$ and $H_{g,2}$ expanded from a length-$2g$ cycle in the base matrix, as shown in Fig. 6. Let their cyclically shifted values to be $P_{1,1}, P_{1,2}, P_{2,1}, P_{2,2} \ldots P_{g,1}$ and $P_{g,2}$. To form a length-$2g$ cycle, the $2g$ 1's must come from the $2g$ different permutation matrices and the following two conditions must be satisfied.

Condition 1: The $2g$ 1's are located in g different rows, and each row contains two 1's.

In Fig. 6, since the $2g$ permutation matrices origin from a length-$2g$ cycle in the base matrix, the $2g$ permutation matrices can be divided into g levels, and every level contains two permutation matrices.

To satisfy condition 1, the two 1's from the i^{th} level must be located on the same row s_i. According to (8), the location of the two 1's can be expressed as

$$(s_i, s_i + P_{i,1} - 0//L) \tag{9}$$
$$(s_i, s_i + P_{i,2} - 0//L). \tag{10}$$

Condition 2: The $2g$ 1's are located at g different columns, and each column contains two 1's.

In Fig. 6, permutation matrices $H_{1,2}$ and $H_{2,1}$ are expanded from two 1's, and the two 1's are on the same column in the base matrix. Similarly $H_{2,2}$ and $H_{3,1}$, $H_{i,2}$ and $H_{i+1,1}$, $H_{g-1,2}$ and $H_{g,1}$, $H_{g,2}$ and $H_{1,1}$, are on the same column too.

If two 1's are randomly selected from $H_{1,2}$ and $H_{2,1}$, the following (11) can guarantee the two 1's are on the same column in the expanded matrix:

$$s_1 + P_{1,2} - (0//L) = s_2 + P_{2,1} - (0//L). \tag{11}$$

Similarly, the following equations are derived to meet condition 2:

$$s_2 + P_{2,2} - (0//L) = s_3 + P_{3,1} - (0//L) \tag{12}$$
$$s_3 + P_{3,2} - (0//L) = s_4 + P_{4,1} - (0//L) \tag{13}$$
$$\cdots$$
$$s_i + P_{i,2} - (0//L) = s_{i+1} + P_{i+1,1} - (0//L) \tag{14}$$

$$\text{1's coordinate} = \begin{cases} (i, i+P), & \text{when } 0 \le i < L-P \quad \text{(Upper-right segment)} \\ (i, i+P-L), & \text{when } L-P \le i \le L-1 \quad \text{(Right-left segment)} \end{cases} \tag{7}$$

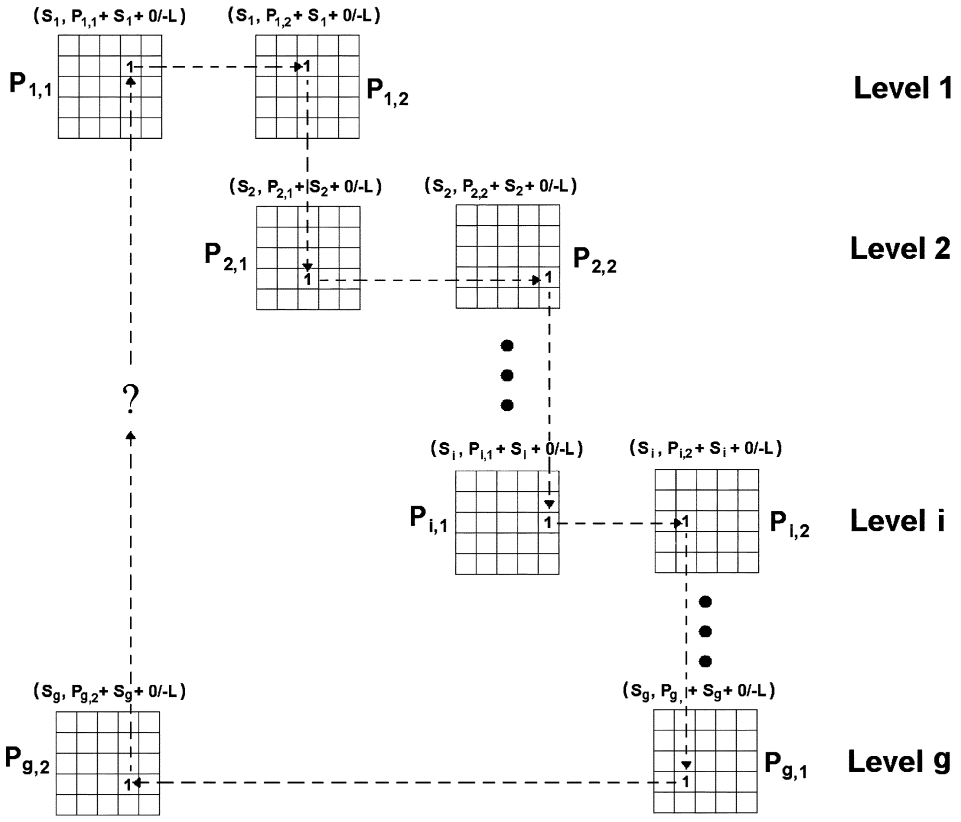

Fig. 6. 1's from $2g$ permutation matrices constitute a length-$2g$ cycle.

$$\cdots$$

$$s_{g-1} + P_{g-1,2} - (0//L) = s_g + P_{g,1} - (0//L) \quad (15)$$

$$s_g + P_{g,2} - (0//L) = s_1 + P_{1,1} - (0//L). \quad (16)$$

Summarizing all the equations from (11)to (16), we have

$$(P_{1,2} + P_{2,2} + \cdots + P_{i,2} + \cdots + P_{g,2}) + \sum_{i=1}^{g} s_i + x \cdot (-L)$$
$$=(P_{2,1}+P_{3,1}+\cdots+P_{i+1,1}+\cdots+P_{1,1})+\sum_{i=1}^{g} s_i+y\cdot(-L). \quad (17)$$

Here, x and y stands for an integer number between 0 and g.

Equation (18) can be derived from (17) as

$$(P_{1,2} - P_{2,1}) + (P_{2,2} - P_{3,1}) + \cdots (P_{i,2} - P_{i+1,1}) + \cdots + (P_{g-1,2} - P_{g,1}) + (P_{g,2} - P_{1,1}) = (y - x) \cdot (-L). \quad (18)$$

That is

$$(P_{1,2} - P_{2,1}) + (P_{2,2} - P_{3,1}) + \cdots (P_{i,2} - P_{i+1,1}) + \cdots + (P_{g-1,2} - P_{g,1}) + (P_{g,2} - P_{1,1}) = k \cdot (-L). \quad (19)$$

Here, k can be any integer number between $-g$ to $+g$.

Equation (19) can be re-written as

$$\sum_{V}(\Delta P) = k \cdot (-L). \quad (20)$$

This completes the proof of the "if" part of the lemma.

Necessary Condition (only-if part): We need to prove that if there are no length-$2g$ cycles existing in the $2g$ permutation matrices, the sum of all vertical shifted value drops along the cycle $\Sigma_V(\Delta P) \neq k \cdot L$ (here $k = 0, \pm 1, \pm 2 \ldots \pm g$).

Suppose that we have the freedom to pick a 1 freely from each permutation matrix in Fig. 6, in the following steps, we will try to connect them to construct a length-$2g$ cycle and study in which condition that the $2g$ 1's cannot form a length-$2g$ cycle.

1) The cycle construction process starts from the leftmost upper permutation matrix $H_{1,1}$ in Fig. 6, and the picked $1'$ location is $(s_1, s_1 + P_{1,1} - 0//L)$, here s_1 is the row index.
2) Next, we need to pick a 1 from $H_{1,2}$, which will be on the same row as the 1 in $H_{1,1}$. Its location is decided as $(s_1, s_1 + P_{1,2} - 0//L)$.
3) Start from the 1 in $H_{1,2}$, we pick the 1 in $H_{2,1}$, and the following can ensure that the two 1's are on the same column:

$$s_1 + P_{1,2} - (0//L) = s_2 + P_{2,1} - (0//L). \quad (21)$$

4) Similarly, from the 1 in $H_{2,1}$, we can find the 1 in $H_{2,2}$ at $(s_2, s_2 + P_{2,2} - 0//L)$ on the same row.
5) From the 1 in $H_{2,2}$, the 1 in $H_{3,1}$ are found to be on the same column, so the following (22) is obtained:

$$s_2 + P_{2,2} - (0//L) = s_3 + P_{3,1} - (0//L). \quad (22)$$

6) Repeating the above the process, if we try to find the next 1 in the cycle loop, this newly found 1 must be on the same

row or column as the previous one. Therefore, in the ith step, (23) can ensure two 1's to be on the same column

$$s_i + P_{i,2} - (0//L) = s_{i+1} + P_{i+1,1} - (0//L). \qquad (23)$$

7) From the 1 in $H_{g-1,2}$, we can find the 1 in $H_{g,1}$ on the same column, and derive

$$s_{g-1} + P_{g-1,2} - (0//L) = s_g + P_{g,1} - (0//L). \qquad (24)$$

9) Finally, from $H_{g,1}$, we reach the 1 in $H_{g,2}$ on the same row at location $(s_g, s_g + P_{g,2} - 0//L)$.

8) At this stage we obtain all the $2g$ 1's as well as their locations, as seen in Fig. 6, if and only if the last found 1 in $H_{g,2}$ and the first found 1 in $H_{1,1}$ are on the same column, the $2g$ 1's can form a length-$2g$ cycle. Therefore, to have the $2g$ 1's not forming a cycle, the following inequality must hold:

$$s_g + P_{g,2} - (0//L) \neq s_1 + P_{1,1} - (0//L). \qquad (25)$$

Adding the equations from (21)to (25) together, we have the following result:

$$\sum_V (\Delta P) \neq k \cdot (-L). \qquad (26)$$

This completes the proof of the "only-if" part of the lemma.

B. Cycle Elimination Algorithm

Based on the Lemma, a constraint $\sum_V (\Delta P) \neq k \cdot (-L)$ is generated for any cycle detected in the base matrix. Since each 1 in the base matrix may be contained in many cycles, a constraint list is generated from all the cycles containing this 1. The purpose of the CE program is to find suitable $P_{i,j}$ values for each 1 in the base matrix that could satisfy all the constraint-lists.

The pseudo-code of the proposed CE algorithm is depicted in Listing 1.

LISTING 1. The Cycle Elimination Algorithm

```
(1) All the $P_{i,j}$ of the 1's in the base matrix are initialized to 0
(2) Set up a empty constraint-list for every 1 in the base matrix
(3) Choose column j to be the current column (from column 1 to column $N_b$)
(4)    clear all the 1's generation marking information in the base matrix.
(5)    //Starting from the 1's in the current column, find and mark their first, second, third, fourth generations.
(6)    for each nonzero $H(i,j)$ in the current column ($0 < i < d_v$)
(7)        Find $H(i,j)$'s second offspring set $\{H(i,m)\}$, here $m > j$, and each $H(i,m)$ is marked as the second generation.
(8)        for each $H(i,m)$ in the second generation set $\{H(i,m)\}$
(9)            Find $H(i,m)$'s offspring set $\{H(p,m)\}$, and every $H(p,m)$ is marked as the third generation.
(10)           for each $H(p,m)$ in the third generation set $\{H(p,m)\}$
(11)               Find $H(p,m)$'s offspring set $\{H(p,q)\}$, and every $H(p,q)$ is marked as the fourth generation
(12)           end
(13)       end
(14)   end
(15)   // Find the cycles
(15)   Check every row and column in the base matrix
(16)   if two second generation $H(i_1,m)$ and $H(i_2,m)$ are on the same column, backtrack to their ancestors to form a cycle-4.
(17)   if two third generation $H(p,m_1)$ and $H(p,m_2)$ are on the same row, backtrack to their ancestors to form a cycle-6.
(18)   if two fourth generation $H(p_1,q)$ and $H(p_2,q)$ are on the same column, backtrack to their ancestors to form a cycle-8.
(19)   each found cycle is transformed to a constraint, and this constraint is added to the constrain-list of the 1's in this cycle.
(19)   // cycle elimination function ()
(20)   for each 1 at position $(i,j)$ in the found cycle
(21)       record its original $P_{i,j}$ value
(22)       while all the constraints in its constraint-list is not satisfied
(23)           $P_{i,j} = P_{i,j} + 1$
(24)       end while
(25)       Check all the current $P_{i,j}$ of all the 1's along the cycles, choose the one with the smallest $P_{i,j}$ and keep this changed value, others 1's' P are restored to their original values.
(26)   end
(27) end
```

The above pseudo-code is explained in the following.

1) First, the base matrix is initialized. All $P_{i,j}$ values are set to zero and an empty constraint-list is set up for every 1 in the matrix. Since each 1 may be contained in several cycles, and every cycle will be translated into a cycle constraint and stored in the constraint-list.
2) In each iteration of the algorithm, a column is picked from the base matrix as a current column (from the order of left to right). This current column divides the base matrix to the left and right parts. All the current-column-related cycles in the right part of the base matrix will then be searched and eliminated through setting the cyclically shifted value P in step 3) and step 4).
3) The cycle searching process is illustrated using the cycle-6 in Fig. 4. Starting from $H(1,1)$ at the current column, an offspring $H(1,2)$ is obtained, the offspring here is defined as the 1's on the same row or the same column with the ancestor 1. From $H(1,2)$, its offspring $H(3,2)$ is found, and $H(3,2)$ is called the third generation of $H(1,1)$. Exploiting the same idea to $H(2,1)$, which is also located at the current column, a second generation offspring $H(2,4)$ and a third generation offspring $H(3,4)$ are found. Then,

the marked generation values are checked. If two third generations $H(3,2)$ and $H(3,4)$ are on the same row, a cycle-6 is detected and the program backtracks to their ancestors until all 1's in the cycle path are found. Because cycle searching proceeds only in the right part of the base matrix, all the found cycles are right-side-related with the current column. In Fig. 4, cycle-4 and cycle-6 are right-side-related with the current column, which contains 1's at $H(1,1)$ and $H(2,1)$.

4) All the cycles right-side-related to the current column are eliminated by setting the cyclically shifted values of the 1's in the cycle path. The number of cycles in the base matrix increases exponentially with the cycle length. The cyclically shifted value P is limited within the ranges $[0, L-1]$. Furthermore, if a cycle is found, each 1 in the cycle is checked to see whether its cycle constraints have been satisfied. If not, its P is incremented until all the constraints are satisfied. After comparing all the P values in the cycle, the 1 with the minimum P value is chosen and its current P value is accepted, while others P values are restored to their original values.

5) The above shifted value setting algorithm in step 4) is very effective in achieving a slow increase of the $P_{i,j}$ value. However, it can still increase rather quickly (not exponentially) and may exceed its upper limit $L-1$ if the target girth is too large. For example, in our experiment of a 18 $\times$ 36 base matrix, the average number of cycles for each element 1 is 1174 if the maximum cycle length is 8. Therefore, in the CE program, the maximum girth optimization target supported is 10. The program will automatically detect the cycle-4, cycle-6 and cycle-8 in the base matrix and eliminate them. If the set $P_{i,j}$ value exceeds the upper limit, the program produces a warning message and the user may lower the girth target.

IV. LDPC Codes Design

Since the fully parallel LDPC decoding structure has high hardware complexity [11], while the fully serial decoding structure can only achieve low throughput, partially parallel decoder architecture [7] is adopted in our design to achieve a balance between the speed and silicon complexity. Moreover, the partially parallel code is VLSI-oriented, and it can be nicely mapped to the partially parallel decoder structure. The partially parallel decoder contains N_b variable-node computation units (VNU) and M_b check-node computation unit (CNU), which corresponds to the size of the base matrix $M_b \times N_b$. Those VNUs and CNUs work in parallel, and every VNU and CNU contain time-multiplexed $L(L = N/N_b)$ variable nodes and $L(L = M/M_b)$ check nodes, respectively.

Among the three rate codes implemented: irregular rate 1/2, regular rate 5/8 and 7/8, the rate 1/2 code is most critical dealing with the worst transmission situation. Based on previous observations [19], [20], irregular codes can outperform regular codes in term of coding gain. Hence, the irregular code is implemented for rate 1/2. Rates 5/8 and 7/8 are designed as regular structures to simplify the hardware complexity.

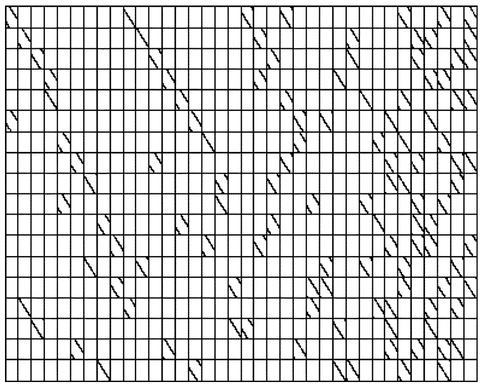
Fig. 7. H-matrix of irregular 1/2 LDPC code.

A. LDPC Code Base Matrix Design

1) Regular Rate 5/8 Code: A $(3,k)$ regular LDPC structure is adopted to construct the rate 5/8 code because this structure can provide a good BER performance for moderate code lengths. The base matrix size of this $(3,k)$ structure is $3k \times k^2$. If every 1 in the base matrix is expanded to an $L \times L$ square permutation matrix, the H-matrix size is $3 \cdot k \cdot L \times L \cdot k^2$ [12]. The rate of the regular$(3,k)$ code is calculated as

$$\text{Rate} = \frac{k-3}{k}. \tag{27}$$

Here, if $\text{rate} = 5/8$, k should be equal to 8.

The structure of the rate 5/8 H-matrix is expanded from a 24 $\times$ 64 ($3k = 24$, and $k^2 = 64$) base matrix. Every square permutation matrix is 149 $\times$ 149, so the size of the H-matrix is 4768 $\times$ 9536. This code structure can be mapped to hardware with 64 VNUs and 24 CNUs, each VNU/CNU contains time-multiplexed L variable nodes and L check nodes, respectively. However, the implementation of 64 VNUs and 24 CNUs would require a heavy hardware cost. So in our architecture, every two VNUs/CNUs are combined and there are a total of 32 VNUs and 12 CNUs in the design. Overall, $2 * L = 298$ clock cycles are needed to complete one VNU computation process and one CNU computation process.

2) Regular Rate 7/8 Code: $(3,k)$ regular code structure is used to design the regular rate 7/8 code too. According to (27), $k = 24$ is for $\text{rate} = 7/8$ case, thus its base matrix size is 72 $\times$ 576 ($3k = 72$, and $k^2 = 576$). Every 1 is expanded to an $L \times L$ ($L = 17$) square permutation matrix, therefore, the size of the H-matrix is 1224 $\times$ 9792. Similarly, every 24 $L \times L$ matrices are combined to a VNU and CNU, so the hardware contains 24 VNUs and 3 CNUs. In total, $24 * L = 408$ clock cycles are needed to finish one VNU computation process and one CNU computation process.

3) Irregular Rate 1/2 Code: Irregular LDPC codes do not outperform regular ones unless their degree distribution is carefully designed. The PEG program [6] is employed to determine the node degree distribution of the base matrix. The designed variable-node degree distribution is (2, 2, . . ., 2, 3,. . .3, 7, . . .7).

The H-matrix of the irregular 1/2 code is shown in Fig. 7. It contains an 18 $\times$ 36 base matrix. The square permutation matrix size is 251 $\times$ 251, and the expanded H-matrix size is 4518 $\times$

TABLE I
CODE DESIGN PARAMETERS

Rate	1/2	5/8	7/8
Variable node degree d_v	2,3 or 7	3	3
Check node degree d_c	5 or 6	8	24
Code length	9036	9536	9792
Base matrix size	18×36	24×64	72×576
Expanded matrix size L×L	251	149	17
Average girth (ZP-constructed codes)	8.0556	8.25	5.9132
Average girth (CE-constructed codes)	10	10	8
CE program run CPU time (minutes)	22.5	18.7	1.13

9036. With each of the square permutation matrix mapped to one VNU/CNU, the hardware contains 36 VNUs and 18 CNUs.

B. Square Permutation Matrix Shifted Value Design

With the above base matrix designed for the three rates, two different methods are employed to determine the cyclically shifted value of every square permutation matrix. One is the CE program described in Section III, the other one is the technique given by Zhang and Parhi (ZP) [12], in which cyclically shifted value P for the 1 at position (i, j) is determined from formula $P_{x,y} = (i-1) \cdot j$. The detailed parameters of the codes are listed in Table I.

Seen from Table I, the codes constructed using the ZP method (ZP-constructed codes) have the average girth lengths 8.0556, 8.25, and 5.9132 for rates 1/2, 5/8, and 7/8, respectively. In contrast, the average girths of the codes constructed using the CE method (CE-constructed codes) can be improved to 10, 10, and 8, respectively. With a SUN Fire V480 server with 4 900 MHz UltraSparc-III processors, the CE program uses 22.5, 18.7, and 1.13 min to generate the three codes, respectively. It should be noted that for the regular 7/8 case, due to the large base matrix size and the small L value, the constructed code girth is 8.

C. Simulation Results

According to the Shannon theory, all "random" codes are "good codes." Unfortunately, random codes are not easy to implement. Thanks to the work of [7]–[10], we can construct VLSI implemented LDPC codes as good as the random codes. Similar to the ZP-constructed codes in [7], the CE-constructed codes are also pseudo-random codes. Fig. 8 shows the simulation comparison results between the performances of the CE and ZP-constructed codes. It is seen that the gain difference between the two codes is within 0.1 dB.

Note that the error floor problem is not seen in the figure. This is due to the fact that the error floor normally occurs at a very low SNR, and a great amount of LDPC blocks and much more simulation time are needed to reveal the error floor curve. For the software simulation, it may take several days to obtain a simulation point with BER below 10^{-7}. So we will employ hardware implementation measurements to show that the CE-constructed

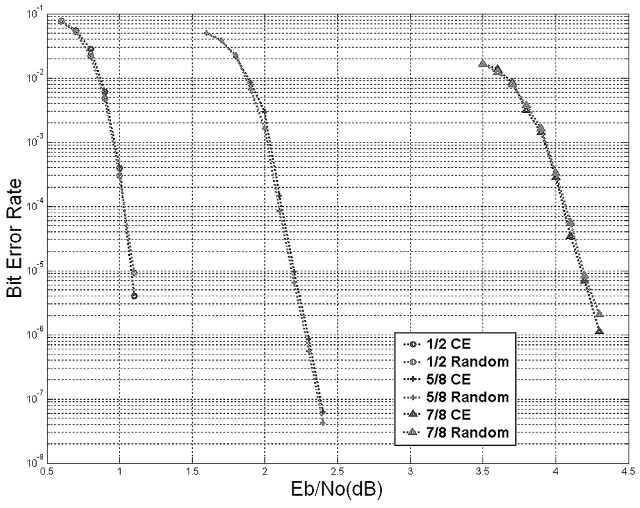

Fig. 8. Simulation results of the ZP and CE-constructed codes.

codes exhibit better error floor performance than the ZP-constructed codes, which will be shown in Section VI.

V. DECODER ARCHITECTURE

The developed configurable partial parallel decoder architecture is shown in Fig. 9. This architecture is not only suitable for different rates, but also can be used for both regular and irregular LDPC codes. At the top level, the architecture follows the general idea of [12], but has the following key distinct characteristics in comparison to the previous architectures described in [7]–[10], [12].

- It exploits configurable structures for the VNU block, the CNU block and the memory banks so different rate LDPC regular/irregular codes can be fit.
- The min-sum with correction (MSC) algorithm [21] is adopted to replace the normally applied table-lookup quantization method to reduce the large performance loss for irregular codes and high-rate regular codes.

A. Finite-Precision Implementation

A finite word length can impact both the decoding performance and the hardware complexity. Let $(q : f)$ to represent the quantization scheme, where q means that totally q bits are utilized, in which f bits are used for the fractional part of the value. If q is kept constant, the precision will be proportional to f, while the dynamic range will be inversely proportional to f.

First of all, an efficient way to emulate function $\Psi = \Psi^{-1}$ in equation (3) needs to be determined since this function is performed frequently for every check-node computation. Up to now, most of the publications on LDPC VLSI implementation employ look-up tables (LUTs) to quantize Ψ. However, the LUT quantization method suffers from the tradeoff between the dynamic range and the precision. For high rate designs and deep fading channels, the maximum magnitude of input channel messages may exceed 100. This requires decreasing f to achieve a

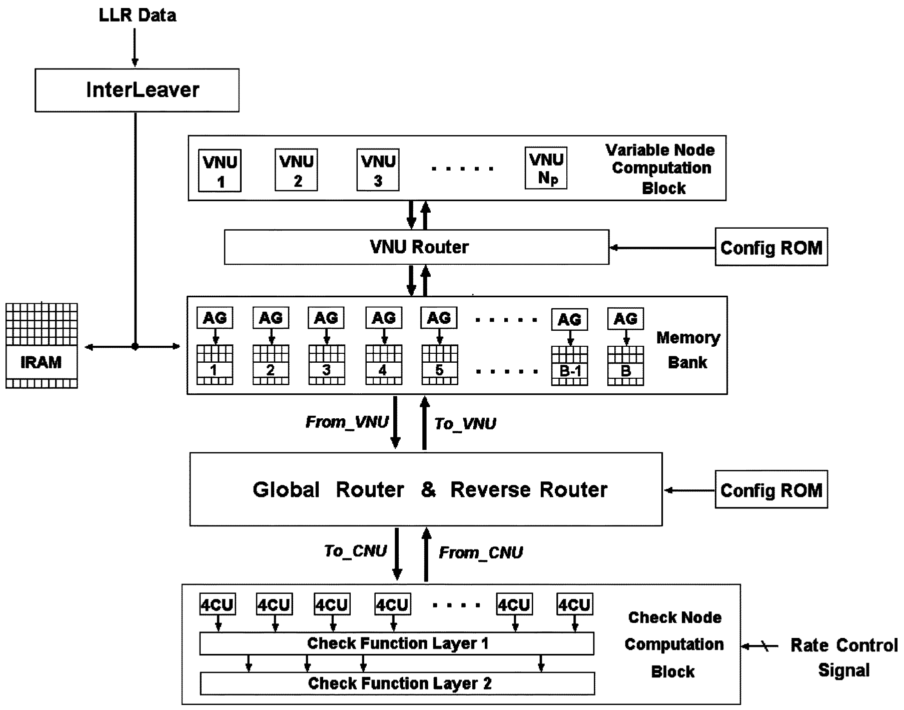

Fig. 9. Multirate decoding architecture.

high dynamic range, which unfortunately contaminates the precision and results in the decoding failure. To solve this problem, [21] and [22] presented the MSC algorithm to emulate $\Psi(x)$

$$L(x\oplus y) = \operatorname{sgn}(xy)\min\{|x|,|y|\}+\ln\left(\frac{1+e^{-|x+y|}}{1+e^{-|x-y|}}\right). \quad (28)$$

The correction term in (28) can be further simplified as

$$\ln\left(\frac{1+e^{-|x+y|}}{1+e^{-|x-y|}}\right) = \begin{cases} 0.5, & |x+y|\le 1, |x-y|>1 \\ -0.5, & |x+y|>1, |x-y|\le 1 \\ 0, & \text{else.}\end{cases} \quad (29)$$

The two equations above describe the MSC algorithm and elude the problem of computing nonlinear function $\Psi(x)$. There is no requirement on the precision and thus there exists the freedom to increase the dynamic range. In our design, f is set to the minimum value 1 to achieve the maximum dynamic range.

To show the MSC performance, both the LUT and MSC methods are used to simulate the three codes in our design. Quantization schemes (6:3) and (6:1) are used for LUT and MSC, respectively. The simulation results are shown in Fig. 10. As seen from the curves, the LUT method achieves good results for low-rate regular LDPC codes (for example, regular 5/8 code). But for high-rate regular codes (for example, regular 7/8), there is more than 0.5 dB gain lost. For irregular 1/2 codes, the LUT method is unable to converge, while other quantization schemes show even worse results than that of the scheme (6:3). Fortunately, the MSC algorithm achieves good results for all the three cases.

Based on the above observation, the MSC method is adopted to emulate $\Psi(x)$ in our design. Through fix point simulations, quantization scheme (6:1) of the input LLR data is chosen because it only has 0.1 dB loss compared to the floating point simulation result.

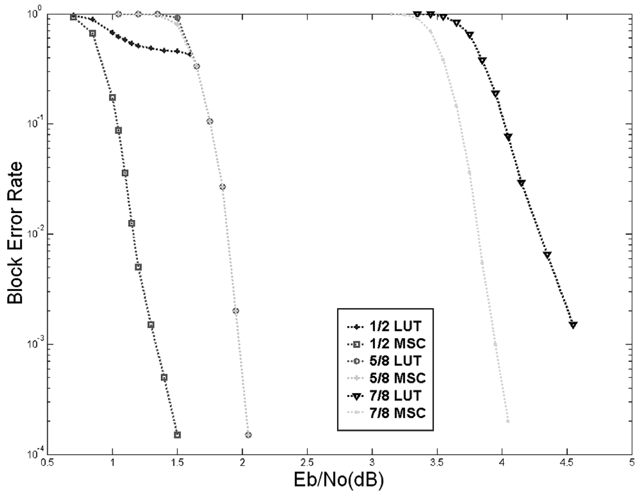

Fig. 10. Comparison of LUT and MSC simulation results.

B. Check-Node Computation Block

As stated earlier, the MSC algorithm is adopted to improve the decoder performance. However, since every pair of messages needs to be performed by the check function, which is used to realize the functions of (28) and (29), the operation of MSC would be time consuming. Below we describe a careful implementation to improve the speed.

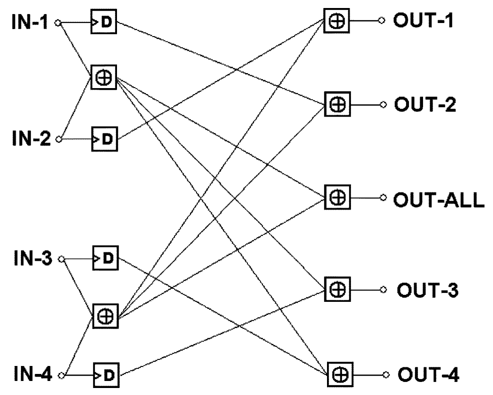

Fig. 11. Architecture of a 4-CNU (4CU).

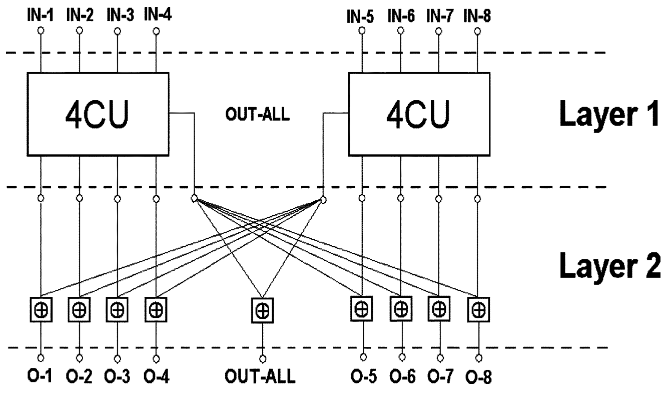

Fig. 12. Architecture of an 8-CNU (8CU) made of two 4CUs.

Considering a check node with degree k, it has the following input-output relationship:

$$\mathrm{OUT}_n = \sum_{i=1->k}^{i\neq n} \oplus \mathrm{IN}_i. \tag{30}$$

Every output OUT_n is equal to the checking result of all the other $k-1$ input messages in the check node. When implementing the check node in hardware, one natural way to calculate OUT_n is checking the other $k-1$ inputs one by one. But this serial operation is time consuming, especially when k is a large number. In this work, a multilayer tree structure is proposed for parallel check-node computation, as shown in Fig. 9.

The first layer of the network contains a set of 4-CNUs (4CUs). Each 4CU has four input messages and five computed output messages, as shown in Fig. 11. One of the outputs is "OUT-ALL," which is the checking result of all the four input messages and will be used in the next layer. The 4CU structure in Fig. 11 contains seven check function operators and four set of D-flip-flops to delay the input messages.

In the second layer, 4CU sets are connected to a set of 8CUs or 12CUs. Fig. 12 shows an example of an 8CU made of two 4CU's. Every 8CU can be easily modified to a 7CU or 6CU by deleting one or two check function operators. The 8CUs and 12CUs can be further connected to form 24CUs or 36CUs in the third layer.

The check-node degrees of the irregular 1/2 code are 6 and 7. Each check node is either a 6CU or 7CU, which can be constructed using two layers of 4CUs. The regular 5/8 (7/8) has check nodes of degrees 8 (24) and requires two (three) layers of 4CUs to construct a 8CU (24CU).

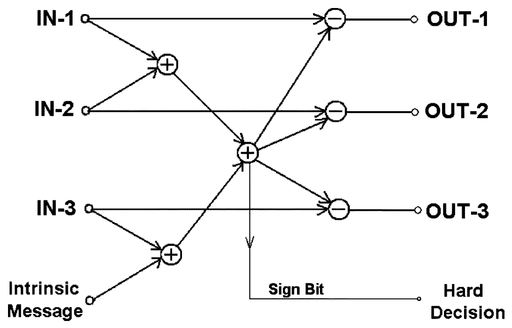

Fig. 13. Architecture of $j=3$ VNU without LUTs.

C. Variable-Node Computation Block

In our architecture, variable-node computation becomes simpler than that in [12] due to the use of the MSC checking function operators in CNUs. Therefore, there is no need for lookup tables and fix-point format conversion (between the sign-magnitude and two's complement). Fig. 13 shows the VNU architecture with node degree 3. The input of this VNU is one intrinsic message and three check-to-variable messages. The output is three computed variable-to-check messages and 1-bit hard decision result.

D. Configurable Routers

- *VNU Router*

  In rate 5/8 and 7/8 regular codes, variable-node degree is uniformly distributed $(j=3)$ and each VNU is connected to 3 RAMs. However, in irregular codes, every VNU is associated with different numbers of RAM's due to the nonuniform distribution of the variable-node degrees. To have different rate code designs co-exist in one implementation, a VNU router is inserted between the memory bank and the variable-node computation block to adapt to different code rates, as shown in Fig. 9. This VNU router is configured by a ROM. The ROM contains three arrays corresponding to the three code rates, where each array describes the node degree distribution of the current rate design. The values of the arrays are shown as

Rate $5/8$: $\{3,3,3,3,\ \ldots\ldots.3,3\}$ Array Length $= N_P = 32$
Rate $7/8$: $\{3,3,3,3,\ \ldots\ldots.3,3\}$ Array Length $= N_P = 24$
Rate $1/2$: $\{2,2,\ \ldots\ 3,3,\ldots\ 7,7\}$ Array Length $= N_P = 36$

- *Global Router and Reverse Router*

  In Fig. 9, the global router and reverse router connect VNUs and CNUs together. There are two message flowing directions going through the global router and reverse router: one is from VNUs to CNUs, and another one is the reverse direction from CNUs to VNUs. Seen from the figure, four buses are involved in this message exchange: "*from-VNU*," "*to-CNU*," "*from-CNU*" and "*to-VNU*." For each rate mode, two arrays are used to configure how "*from-VNU*" is permutated and connected to "*to-CNU*," and how "*from-CNU*" is permutated and connected to "*to-VNU*." A total of six arrays that is stored in a configuration ROM are needed to configure the global router and reverse router to work at three different rate modes.

E. Memory Bank

The RAMs in the memory bank are used to store exchanged messages between variable nodes and check nodes. Each RAM is associated with an address generator (AG) to provide reading and writing addresses. Since each RAM contains one or more $L \times L$ circularly shifted permutation matrices, an address generator consists of only simple counters starting from an offset address during the check-node processing period or starting from address zero during the variable-node processing period.

VI. FPGA Implementation and Measurement Results

Employing the (6:1) quantization scheme and the architecture described in Section IV, a multi-rate LDPC decoder is implemented on a Xilinx Virtex-II XC2V8000 FPGA device. The design is described in VHDL, synthesized by Synplicity, placed and routed using Xilinx development tool ISE6.0. It works at the 100 MHz clock frequency, and has codeword length about 9 kbits. By configuring two pins of the FPGA device at "00," "10" or "11," the decoder can operate at three different code modes: irregular 1/2 mode, regular 5/8 mode and regular 7/8 mode.

The XC2V8000 FPGA belongs to the Xilinx Virtex-II family, and is developed for data communication and DSP applications. The device contains 168 18-kbit dual-port SelectRAM blocks and 46 592 slices, and possesses the capacity to handle 8-million-gate design. One of the most challenging problems in the LDPC decoder design is the memory usage since the LDPC code structure requires a large number of parallel working memories to store exchanged messages. Therefore, a proper partition of the memory blocks to fully utilize the FPGA SelectRAM resources is needed. In our architecture, memory bank and IRAM (used to store intrinsic messages) are the main sources of memory usage. The memory bank contains 117 512$*$7 independent RAM blocks, and every small RAM needs one independent port for reading/writing, so two of them are combined to one dual-port RAM's and occupy a SelectRAM unit resource in the FPGA. IRAM is a large 4.5K$*$32 memory, and takes eight SelectRAM units. Adding the memory resource used in the data loading, unloading and the interleaver blocks, a total of 102 SelectRAM units are used. The resource utilization statistics is shown in Table II.

Another challenge of designing an LDPC decoder is the routing congestion caused by the complex top-level connections. In our implementation, this problem is alleviated by carefully pipelining the data paths between VNU blocks, routers, and CNU blocks. In addition, the critical path delays are minimized, and the decoder is able to operate at the 100 MHz clock frequency. At this frequency, the decoder achieves a maximum throughput of 40 Mbps for regular 5/8 and 7/8 codes when performing maximum 24 decoding iterations. For the irregular 1/2 LDPC code, more iterations are required for the codes to converge, so maximum decoding iteration number is set to 60, and the maximum 15 Mbps throughput has been achieved.

TABLE II
FPGA Resource Usage Statistics

Resource	Number	Usage Rate
Slices	34,127	73%
Slice Flip Flops	24,570	26%
4 Input LUTs	53,327	57%
Block RAMs	102	60%
Bonded IOBs	75	9%
DCMs	1	8%

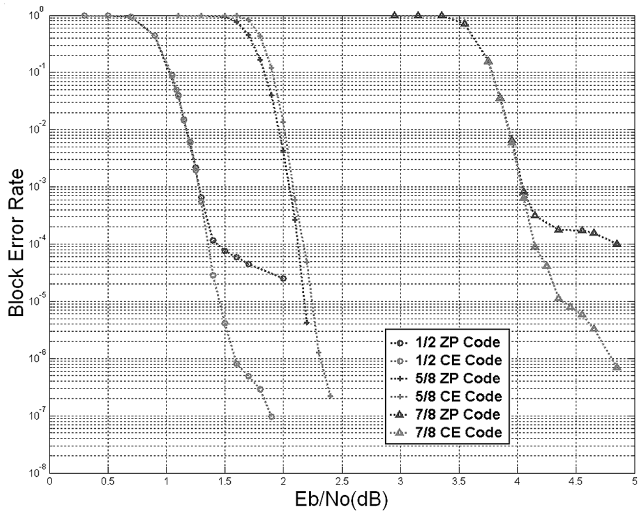

Fig. 14. Measurement result comparison between CE and ZP-constructed codes.

The implemented LDPC decoder has been used together with an OFDM chip to form an OFDM-LDPC system. The measurement results of the ZP-constructed codes and CE-constructed codes are compared in Fig. 14. In the figure, the block-error rate verse SNR curves are plotted. Note that the y axis is the block error rate (block length >9 k), and it is approximately two orders of magnitude higher than the BER. As seen from Fig. 14, the ZP-constructed regular 5/8 code achieves the good block error rate performance and does not have the error floor problem. However, the ZP-constructed irregular 1/2 and regular 7/8 codes experience the serious error floor problem at the block error rate of about 10^{-4}. The above phenomena can be explained using the girth histogram shown in Fig. 15. In Fig. 15, the variable nodes inside one square permutation matrix have the same girth value. For the ZP-constructed regular 5/8 code, because it is cycle-4 and cycle-6 free, the error floor can be efficiently removed. As to the ZP-constructed irregular 1/2 code, although it is cycle-4 free and has the average girth of 8.0556, the error floor problem still occurs at a much high level because irregular codes normally have the worse error floor problem than regular codes [23]. Moreover, there exists a tradeoff between the threshold SNR and the error floor BER [24]. In our design, the performance of the irregular 1/2 code is near the Shannon limit, making the error floor to appear at a higher level. For the ZP-constructed regular 7/8 code, its base matrix size is 72 $\times$

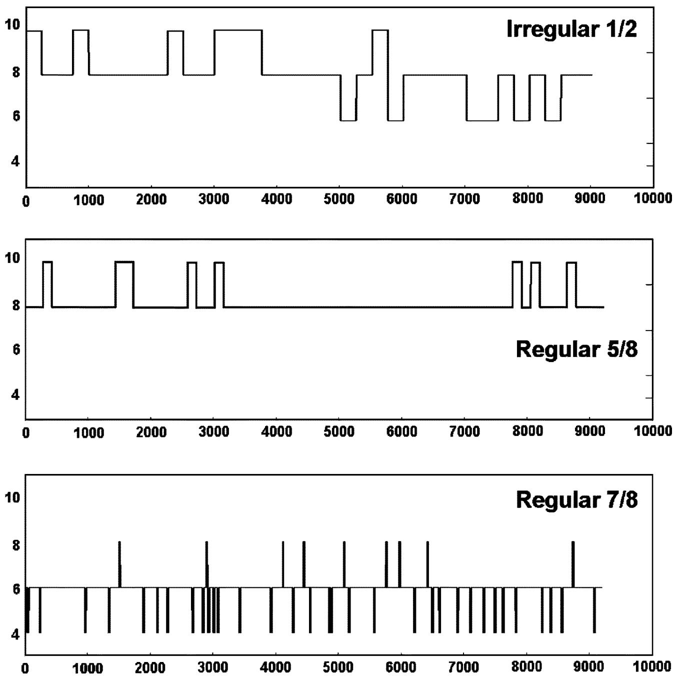

Fig. 15. Girth histogram of the ZP-constructed codes.

576. It contains a lot of length-4 cycles, and those short cycles are difficult to be eliminated with a very small L value 17.

To suppress the error floor in irregular 1/2 and regular 7/8 codes, cycle elimination is used to generate the LDPC H-matrix, and the average girth values of the three rate codes are improved to 10, 10, and 8, respectively. This can suppress the error floor to a much lower level. Furthermore, the decrease of the error floor does not bring any penalty to the coding gain, and the ZP-constructed codes and the CE-constructed codes have the gain difference within 0.1 dB.

It should be noted that after cycle elimination, the irregular 1/2 code outperforms the regular 1/2 code by approximate 0.5 dB, achieving a block error rate lower than 10^{-7} at SNR 1.8 dB. With this superior error correcting ability, the irregular 1/2 mode decoder in our design can be utilized to handle deep fading channels and the most severe interference conditions. The regular 5/8 and 7/8 codes enjoy the higher throughput capability, and can operate at better transmission conditions.

VII. Conclusion

This paper presented the construction, architecture and VLSI implementation of a 9-kbit multi-rate LDPC code decoder. The implementation used a Xilinx XC2V8000 FPGA device. By configuring two pins of the FPGA device at "00," "01" or "10," the decoder can work at three different rates: irregular 1/2, regular 5/8 and regular 7/8. The finite precision effect has been carefully analyzed, and the best quantization scheme that fit for three rates has been developed to improve the decoder performance.

To suppress the error floor for the irregular 1/2 code and regular 7/8 code, a cycle elimination (CE) algorithm has been developed. By carefully setting the cyclically shifted value for all the square permutation matrices, the CE algorithm attempts to eliminate the short cycles in an LDPC matrix. Both the simulation and measurement results showed that the error floor of the LDPC codes constructed with cycle elimination has been significantly lowered without incurring the SNR threshold penalty.

The principle of the CE algorithm is to find and eliminate the short cycles existing in the LDPC base matrix. However, it has been pointed out that short cycles are not the main cause of the error floor problem [20], [25]. Therefore, the future work include the generalization of the CE algorithm to find and eliminate the so-called stopping sets, instead of short cycles.

Acknowledgment

The authors thank Dr. Z. Wang, Oregon State University, Corvallis, for several helpful discussions, and the anonymous reviewers for their constructive comments.

References

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correction coding and decoding," in *Proc. Int. Conf. Commun.*, Geneva, Switzerland, May 1993, pp. 1064–1070.
[2] T. Richardson and R. Urbanke, "The renaissance of Gallager's low-density parity-check codes," *IEEE Commun. Mag.*, pp. 126–131, Aug. 2003.
[3] E. Yeo, B. Nikolic, and V. Anantharam, "Iterative decoder architectures," *IEEE Commun. Mag.*, pp. 132–140, Aug. 2003.
[4] P. Urard, E. Yeo, and B. Gupta *et al.*, "A 135 Mb/s DVB-S2 compliant CODEC based on 64 800 b LDPC and BCH codes," in *Proc. IEEE Int. Solid-State Circuits Conf.*, Jan. 2005, pp. 547–548.
[5] J. Campello, D. S. Modha, and S. Rajagopalan, "Designing LDPC codes using bit-filling," in *Proc. Int. Conf. Commun.*, Helsinki, Finland, 2001, pp. 55–59.
[6] X. Y. Hu, E. Eleftheriou, and D. M. Arnold, "Progressive edge-growth tanner graphs," in *Proc. IEEE Global Telecomm. Conf.*, vol. 2, Nov. 2001, pp. 995–1001.
[7] H. Zhang and T. Zhang, "Design of VLSI implementation-oriented LDPC codes," in *Proc. IEEE Veh. Technol. Conf.*, vol. 1, 2003, pp. 670–673.
[8] D. E. Hocevar, "LDPC code construction with flexible hardware implementation," in *Proc. Int. Conf. Commun.*, 2003, pp. 2708–2712.
[9] A. Delvarathinam, E. Kim, and G. Choi, "Low-density parity-check decoder architecture for high throughput optical fiber channels," in *Proc. Int. Conf. Comp. Design*, 2003, pp. 520–525.
[10] E. Boutillon, J. Castura, and F. R. Kschischang, "Decoder-first code design," in *Proc, Int. Symp. Turbo Codes Related Topics*, Brest, France, Sep. 2001, pp. 459–462.
[11] 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 J. Solid-State Circuits*, vol. 37, no. 3, pp. 404–412, Mar. 2002.
[12] T. Zhang and K. K. Parhi, "VLSI implementation-oriented (3,k)-regular low-density parity-check codes," in *Proc. IEEE Worksh. Signal Process. Syst.*, Sep. 2001, pp. 25–36.
[13] G. Xing, M. Shen, and H. Liu *et al.*, "An LDPC-based Terrestrial Multimedia Broadcasting (TMB) system: design, implementation and experimental results," *Acoust., Speech, Signal Process.*, vol. 4, pp. 17–21, May 2004.
[14] L. Yang, M. Shen, H. Liu, and C.-J. R. Shi, "An FPGA implementation of low-density parity-check code decoder with multi-rate capability," in *Proc. Asia South Pacific Design Automation Conf.*, Shanghai, China, Jan. 2005, pp. 760–763.
[15] L. Yang, H. Liu, and C.-J. R. Shi, "Cycle elimination method to construct VLSI oriented LDPC codes," in *Proc. IEEE Veh. Technol. Conf.*, Dallas, TX, Sep. 2005, pp. 522–526.
[16] L. Yang, H. Liu, and C.-J. R. Shi, "VLSI implementation of low-error-floor and capacity-approaching performance low-density parity-check codes with multi-rate capacity," in *Proc. IEEE Global Telecomm. Conf.*, St. Louis, MO, Nov. 2005, pp. 1261–1266.
[17] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.
[18] M. Chiani, A. Conti, and A. Ventura, "Evaluation of low-density parity-check codes over block fading channels," in *Proc. Int. Conf. Commun.*, 2000, pp. 1183–1187.

[19] D. J. C. Mackay, S. T. Wilson, and M. C. Davey, "Comparison of constructions of irregular Gallager codes," *IEEE Trans. Comm.*, vol. 47, pp. 1449–1454, Oct. 1999.

[20] T. Tian, C. Jones, J. D. Villasenor, and R. D. Wesel, "Construction of irregular LDPC codes with low error floors," in *Proc. Int. Conf. Commun.*, vol. 5, May 2003, pp. 3125–3129.

[21] A. Anastasopoulos, "A comparison between the sum-product and the min-sum iterative detection algorithms based on density evolution," in *Proc. IEEE Global Telecomm. Conf.*, vol. 2, Nov. 2001, pp. 25–29.

[22] X. Y. Hu, E. Eleftheriou, and D. M. Arnold, "Efficient implementation of the sum-product algorithm for decoding LDPC codes," in *Proc. IEEE Global Telecomm. Conf.*, vol. 2, Nov. 2001, pp. 1036–1036.

[23] T. Tian, C. Jones, J. D. Villasenor, and R. D. Wesel, "Construction of irregular LDPC codes with low error floors," in *Proc. Int. Conf. Commun.*, vol. 5, May 2003, pp. 3125–3129.

[24] T. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 47, pp. 619–637, Feb. 2001.

[25] T. Richardson, "Error floors of LDPC codes," in *Proc. 41st Allerton Conf. Commun. Contr. Comput.*, Oct. 2003, pp. 1426–1435.

Lei Yang received the B.S. degree from HuaZhong University of Science and Technology, Wuhan, China, in 1998, and the M.S. degree from TsingHua University, Beijing, China, in 2001. He is currently working toward the Ph.D. degree in the Mixed-Signal CAD Lab, Electrical Engineering Department, University of Washington, Seattle.

During his M.S. study, he did a lot of work on RF filter design, field–programmable gate arrays, and digital application-specific integrated circuit (ASIC) designs. Currently, his research interests are in the field of VLSI implementation and communication systems such as low-density-parity-check and orthogonal frequency division multiplexing design and chip realization, and mixed-signal VLSI and analog IC design automation.

Hui Liu (S'92–M'92–S'93–M'95–SM'04) received the B.S. degree from Fudan University, Shanghai, China, in 1988, the M.S. degree from Portland State University, Portland, OR, in 1992, and the Ph.D. degree from the University of Texas at Austin, in 1995, all in electrical engineering.

He held the position of Assistant Professor in the Department of Electrical Engineering at University of Virginia from September 1995 to July 1998. He was the Chief Scientist at Cwill Telecommunications, Inc., and was one of the principal designers of the UMTS TD-SCDMA 3G standard. In 2000, he founded Broadstorm Inc. and pioneered the development of the world first OFDMA-based mobile broad-band network. He is currently an Associate Professor in the Department of Electrical Engineering, University of Washington, Seattle. His research interests include broad-band wireless networks, array signal processing, digital signal processing and VLSI applications, and multimedia signal processing. He has published more than 40 journal articles and has twelve awarded or pending patents. He is the author of two textbooks: *OFDM-Based Broad-Band Wireless Networks: Design and Optimization* (Wiley, 2005), and *Signal Processing Applications in CDMA Communications* (Artech House, 2000).

Dr. Liu's activities for the IEEE Communications Society include membership on several technical committees and serving as an Editor for the IEEE TRANSACTIONS IN COMMUNICATIONS. He is the General Chairman for the 2005 Asilomar conference on Signals, Systems, and Computers. He is a recipient of 1997 National Science Foundation (NSF) CAREER Award, The Best Patent Award in China, and 2000 Office of Naval Research (ONR) Young Investigator Award.

C.-J. Richard Shi (M'91-SM'99–F'06) is currently a Professor in Electrical Engineering at the University of Washington. His research interests include computer-aided design and test of integrated circuits and systems, as well as VLSI implementation of communication systems. He is a key contributor to IEEE std 1076.1-1999 (VHDL-AMS) language standard for the description and simulation of mixed-signal circuits and systems. He founded IEEE International Workshop on Behavioral Modeling and Simulation (BMAS) in 1997, and has served on the technical program committees of several international conferences. He has authored or coauthored over 100 papers published in international journals and conferences, and has served as the principal investigator of over 10 research projects supported by DARPA, SRC and NSF.

Dr. Shi received the Best Paper Award from the IEEE/ACM Design Automation Conference, a Best Paper Award from the IEEE VLSI Test Symposium, a National Science Foundation CAREER Award, and a Doctoral Prize from the Natural Science and Engineering Research Council of Canada. He has been an Associate Editor, as well as a Guest Editor, of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: ANALOG AND DIGITAL SIGNAL PROCESSING. He is currently an Associate Editor of IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATE CIRCUITS AND SYSTEMS and IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: EXPRESS BRIEFS.