ARCHITECTURE-AWARE LDPC CODE DESIGN FOR SOFTWARE DEFINED RADIO

Yuming Zhu and Chaitali Chakrabarti

Department of Electrical Engineering

Arizona State University, Tempe, AZ 85287

Email: {yuming,chaitali}@asu.edu

ABSTRACT

Low-Density Parity-Check (LDPC) codes have been adopted in the physical layer of many communication systems because of their superior performance. The direct implementation of these codes onto an existing software defined radio (SDR) platform is likely to be inefficient. Our approach is to design the LDPC code to match the constraints imposed by the existing architecture, without compromising the communication performance. We present a procedure for architecture-aware code design that involves feature identification, code construction and verification. Details of the procedure for the case when the SDR platform is equipped with a multi-stage interconnection network (MIN) is presented. By analyzing the characteristics of the MIN, simple yet explicit constraints are derived and used in the code construction step. The resulting LDPC code can not only be mapped very efficiently onto the SDR platform but also has very good bit error rate (BER) performance.

1. INTRODUCTION

Low-Density Parity-Check (LDPC) [1] codes are linear block codes with sparse parity check matrices. They have superior asymptotic performance; the performance of these codes can be as close as one tenth dB away from the Shannon limit [2] Another big advantage of LDPC codes is that the decoding algorithm is inherently parallel and so a wide variety of hardware implementations can be derived that exploit this feature [3, 4, 5, 6, 7]. Because of the superior performance, LDPC codes have been adopted in the physical layer of many recent communication standards such as DVB-S2, 10GBase-T, 802.16e (WiMax) and 802.11n.

The algorithms used in the physical layer for different protocols is quite diverse. In order to support multiple protocols and reduce the time to market, software defined radios (SDR) have emerged as a very attractive alternative. Successful SDR platforms are those that have been optimized based on profile studies of the target protocols. These typically result in better performance and lower power. However, when a new protocol is introduced and mapped directly to the existing SDR platform, it results in a possible inefficient implementation. If instead, the protocol could be somehow tailored to match the constraints imposed by the existing architecture, without compromising the communication performance, the mapping of the new protocol to the SDR platform would result in a much more efficient implementation.

In this paper, we present our attempt at designing LDPC codes that exploit the architectural features of an SDR platform. Fig. 1 shows the overall flow. The first step is to obtain the code features based on both system specifications, such as bit error rate (BER) performance, block size, code rate, and architectural features of the target platform, such as number of processing units, bus width and arithmetic complexity. After that, the LDPC code is constructed based on code features, which include number of block columns, row weight, level of parallel processing, etc. The last step is to verify that the LDPC code meets both the system constraints and the architectural constraints. In some cases, several iterations may be necessary to ensure that all the constraints are met.

As a case study, we show the LDPC code design procedure for a SDR platform that consists of multiple processors that communicate with each other via a multi-stage interconnection networks (MIN). The constraints posed by the interconnection network are first identified. Specifically, the combinations of routing paths that cause conflicts in the MIN are identified and mechanisms to avoid them are translated into constraints for the code construction step. "Pair-wise" conflict patterns are studied and explicit code construction rules that result in a sub-optimal solution are provided. The resulting LDPC code can be mapped very efficiently onto the SDR platform. The decoding throughput is very high since routing conflicts through the MIN have been minimized. At the same time the BER performance of these codes is not compromised. In fact, in some cases, they are better than the randomly generated counterparts.

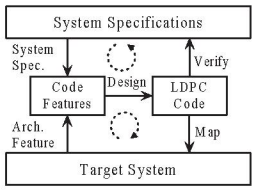

Fig. 1. Architecture-aware LDPC code design flow

The rest of this paper is organized as follows. In Section 2, the relationship between architectural features and LDPC code features is studied. Then in Section 3, the LDPC code design procedure for a SDR platform with existing MIN is presented. The explicit constraints are derived from the analysis of the MIN, and then the LDPC code is constructed under these constraints. The hardware mapping and the performance results are also presented. The paper is concluded in Section 4.

2. ARCHITECTURE-AWARE LDPC CODE DESIGN

2.1. SDR Platform

A typical SDR platform consists of multiple processing units (PU) and multiple global storage units as shown in Fig. 2. The PUs and the global storage units communicate with each other via an

This research was supported by the NSF-ITR 0325761 and NSF CSR-EHS 0615135.

interconnection network. Each PU consists of a local memory, a processing element (PE), which consists of a scalar unit and a single-instruction multiple-data (SIMD) unit, and an application specific element (ASE). The combination of a scalar unit and a SIMD unit enables a large class of algorithms to be mapped very efficiently as shown in [8]. The ASEs are typically included to enhance the performance of some target protocols. The software control unit coordinates all the operations in the system.

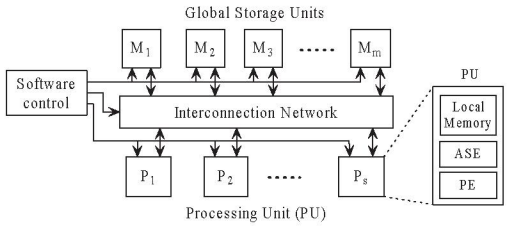

Fig. 2. Software defined radio (SDR) platform

The efficiency of mapping the protocols to the target SDR platform is greatly affected by the features of the architecture. There are two types of architectural features: general features and application specific features. General features are architectural descriptors such as the number of PU, SIMD width of each PU, and the arithmetic operations that can be performed within each PU. Application specific features are the features of ASEs which were added to enhance the performance of some protocol.

2.2. LDPC decoding

The computations in the LDPC decoding are generally simple and can be easily parallelized. The bottleneck, however, lies in the interconnection between the different processing and storage units. The root of this problem can be traced back to the construction of the LDPC codes: the codes rely on scrambling of the bits to achieve good performance.

Because of the interconnection complexity, the fully parallel LDPC decoder is huge for large block sizes [3]. Thus the partial parallel decoder, which makes use of small block matrices with ordered structure, is highly preferred. Several LDPC codes with ordered structures based on geometrical or algebraic constructions have been proposed [9, 5, 10]. These codes make use of geometrical or algebraic properties that achieve good BER performance.

In this paper, we use the algebraic construction method presented in [10]. The block parity check matrix H_b has dimension of $m_b \times n_b$; each element in H_b matrix is a $Z \times Z$ circulant shifted identity matrix or zero matrix, where Z is a prime number. The shift value for the sub-matrix (i, j) is $P_{i,j} = b^{i-1}a^{j-1}$ where a and b have multiplicative order of n_b and m_b respectively, in Galois field GF(Z). Thus the non-zero item in H_b can be represented as $H_b(i, j) = I_{b^{i-1}a^{j-1}}$, where I_x is the circulant identity matrix with shift value of x. For simplicity, we use '1' or '0' to represent the I_x or zero matrix.

The LDPC code features can be divided into two categories: some features come from the code itself, such as the code rate (R), number of block columns (n_b), number of block rows (m_r), submatrix size (Z), average/maximum row weight (d_c), minimum column weight (d^v_{min}), maximum column weight (d^v_{max}), etc.; other features come from the decoding process, such as the decoding algorithm, level of parallelism in parallel decoding, etc.

Table 1. General architectural features and related code features

Architectural Feature	Code Feature
Number of PU (S)	Maximum Row weight in H matrix
SIMD width of each PU (P)	Maximum Parallel level in decoder
Arithmetic operations supported by PU	Selection of decoding algorithm

2.3. LDPC code design from code features

The steps for designing LDPC code based on code features can be summarized as follows:

1. First, obtain the code features based on system requirement and architectural specifications. The code features include: n_b, m_b, Z, d_c, d^v_{min}, d^v_{max}, etc.
   While system requirements determine the code rate R and the range of block size ($n_b \times Z$), architectural features are used to determine the other code features. Some general architectural features and their related LDPC code features are briefly summarized in Table 1. From the SIMD width (P) and number of PU assigned in LDPC decoding (S), the parallel level and row weight (d_c) can be determined. Based on the arithmetic operation supported by PU, the appropriate decoding algorithm can be selected.
2. Compute the degree distribution based on the targeted channel condition, code rate (obtained from m_b and n_b), d_c, d^v_{min} and d^v_{max}. The optimal degree distribution for asymptotic cases (infinite block size, infinite iteration number) can be computed based on density evolution (DE) [11]. An online computing program is available at [12]. To use the optimal degree distribution results of the infinite block size to finite block size, we may need to round off the actual node numbers to integers.
3. Construct the H_b matrix based on the degree distribution result in previous step. The construction procedure can be briefly written as:
   (i) Initialize the H_b of size $m_b \times n_b$ with '0'.
   (ii) Randomly place d_c '1' in each row.
   (iii) Move around the '1' among columns to satisfy the bit node degree distribution.
   (iv) Switch columns to facilitate further modification. For example, some lower degree columns are needed to move to the right side of the H_b matrix before step **(v)** in order to make the code encoder efficient [13].
   (v) Exchange the '1'-pair among different columns and rows to meet more constraints. By 'exchange' we mean the movement which does not affect both check and bit node degree distribution. In Section 3.1, the constraints obtained from the interconnection network are applied in this step.
   (vi) Output the H_b matrix.
   Note that steps **(iv)** and **(v)** may have to be processed multiple times until all the constraints have been met or the maximum iteration number has been reached.
4. Assign shift values for all the non-zero elements in H_b matrix.

We can see that the optimal degree distribution obtained from step 2 is asymptotically optimal, which may not be optimal for finite length codes decoded with finite number of iterations. However, to the best of our knowledge, there does not exist any mathematically rigorous finite-length optimization method for additive white Gaussian noise (AWGN) or more complicated channel (the

most related result is for binary erasure channel (BEC) [14]). Empirically, the borrow-from-asymptotic method provides reasonably good performance and can serve as a temporary solution. Other criteria for optimizing finite-length LDPC codes such as girth (smallest size of cycles in underlying Tanner graph), minimum distance or cycle properties can be applied in step 4. In this paper, we simply apply the construction shown in [10].

3. CASE STUDY: INTERCONNECTION-AWARE LDPC CODE DESIGN

As an example, we will perform the LDPC code design for an SDR system with built-in interconnection networks. We assume that each PU is assigned to process one non-zero block column (variable node) in a block row of H_b matrix. Thus multiple PUs are used in decoding in parallel which lead to very high throughput.

Ideally, connecting the processors with a completely connected network results in minimum processing delay and communication overhead. However, the completely connected network of N nodes requires $O(N^2)$ connections, and is thus formidably complex for large N. The MIN is much less complex: it supports N inputs and N outputs with $\lceil \log_2(N) \rceil$ stages of interconnection and switches. Many of the known MINs, such as butterfly, perfect k-shuffle, can be proved to be topologically and functionally equivalent [15]. In this paper, we only show the example of butterfly and perfect shuffle network. However, the analysis method and results are not limited to a specific type of MIN, and can be generalized to other MINs.

3.1. Feature Identification

To utilize the MIN in LDPC decoding, we need to first characterize the constraints imposed by the MIN and translate the constraints to the corresponding code features.

In LDPC decoding, when the layered belief propagation (BP) algorithm [4] is used, the decoding is scheduled based on row order. There are n_b storage units to store the bit node information, which is the log-likelihood ratio (LLR) of *a posteriori* probability (APP) for each bit node. Each storage unit corresponds to one block column and can perform read/write operations independently. The check-to-bit information is stored in the local memory of each PU. The data from the n_b storage units are routed to/from the $S = d_c$ processing units. The bidirectional MIN that connects the processing units and storage units can be visualized as two unidirectional MIN as in Fig. 3. The bit node information of the bit nodes which are connected to the check node being processed, are read out from SU and routed through MIN I to the PU. In the PU, the bit node information together with the check-to-bit information are used to calculate the new check-to-bit information and new bit node information as shown in [4]. The new bit node information from the PU are routed back to SU through MIN II. In the rest of this paper, we study the operation of the interconnection network MIN II and assume the PUs are connected with input nodes and storage units are connected with output nodes.

Note that the data stream for each processing unit and storage unit can be either single data when only one row is processed at a time or the ensemble of multiple data when parallel processing is employed. If parallel processing is used, the "alignment" and "reverse alignment" functions discussed in [4] should be implemented. It can be done within the PU or also be implemented in the MINs. The details are omitted due to lack of space.

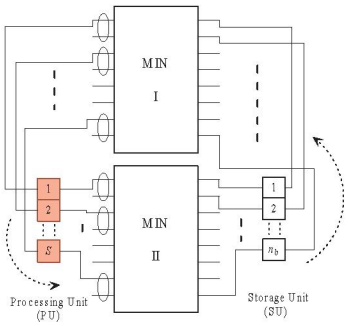

Fig. 3. The connection of the processing and storage units of LDPC decoder through the multi-stage interconnection network.

3.1.1. *Architectural Features*

In the MIN interconnection network, if the data from only one input node is to be routed through the network, there is no conflict. However, if the data from more than one input node is to be routed, there exist combinations of routing paths among input and output nodes that can not be activated simultaneously. The relative position of the input/output nodes plays an important role in determining these combinations. To minimize the delay in routing through the MINs, it is important that these combinations not be used. Since the study of these combinations for large N values is very involved and do not provide guidance to code construction, we study "pair-wise" conflicts instead. These are used to derive explicit constraints that are then used in the code construction.

For any given input node X, the remaining $N-1$ input nodes are divided into different neighborhood groups $G_i(X)$:

$$G_i(X) \triangleq \{\text{all the input nodes that can be routed to the same } 2 \times 2 \text{ switch as X at stage } i\} \quad (1)$$

From this definition, we can see that $Y \in G_i(X) \Leftrightarrow X \in G_i(Y)$.

If the input node X is represented by $X = (x_{n-1}, \cdots, x_1, x_0)$, $x_i \in \{0,1\}$, the necessary and sufficient condition that an input node $Y = (y_{n-1}, \cdots, y_1, y_0)$ belongs to the i-th neighborhood group $G_i(X)$ for butterfly MIN is given in Eqn. (2). The corresponding relationship for the perfect shuffle MIN is given in Eqn. (3).

$$Y \in G_i^{butterfly}(X) \Leftrightarrow \begin{cases} y_j = x_j, & j > i \\ y_j \neq x_j, & j = i \\ \text{don't care}, & \text{otherwise} \end{cases} \quad (2)$$

$$Y \in G_i^{perfect}(X) \Leftrightarrow \begin{cases} y_j = x_j, & j < n-i-1 \\ y_j \neq x_j, & j = n-i-1 \\ \text{don't care}, & \text{otherwise} \end{cases} \quad (3)$$

Fig. 4 shows the neighborhood groups of $(010)_b$ in an 8-point butterfly MIN: $G_0(010) = \{(011)_b\}$, $G_1(010) = \{(000)_b, (001)_b\}$ and $G_2(010) = \{(100)_b, (101)_b, (110)_b, (111)_b\}$. Similarly, Fig. 5 shows the neighborhood groups of $(010)_b$ in an 8-point perfect shuffle MIN respectively. It is clear that although the set $G_i(X)$ for different MIN are different, the norm is the same, i.e. for $\forall i, X$, $|G_i^{butterfly}(X)| = |G_i^{perfect}(X)|$.

If the destination of node X is fixed, any node $Y \in G_i(X)$ can be routed to only $N - 2^{n-i-1} = 2^n[1 - 2^{-(i+1)}]$ possible destination positions. To illustrate this, the pairwise coverage

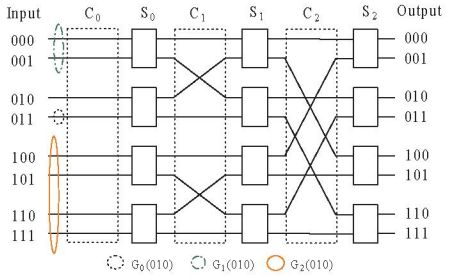

Fig. 4. Butterfly MIN: neighborhood groups of $(010)_b$

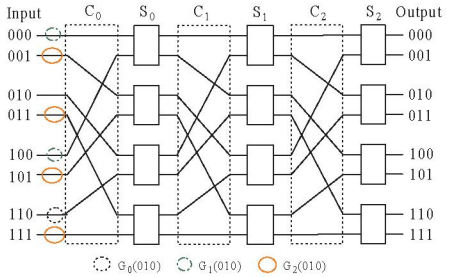

Fig. 5. Perfect 2-shuffle MIN: neighborhood groups of $(010)_b$

$C_i(X,Y)$ is defined as the number of destination nodes (in percent) that can be routed from node $Y \in G_i(X)$ without creating any conflicts with the routing of node X. It is easy to see that $C_i(X,Y) = [1 - 2^{-(i+1)}] \times 100\%$ and $C_i(X,Y)$=$C_i(Y,X)$. The $C_i(X,Y)$ for different i values are depicted in Fig. 6. The $C_i(X,Y)$ monotonously increases with the increase of i, which means that the input node pair (X,Y), $Y \in G_i(X)$ has higher probability of conflict for smaller values of i. It is worth noting that the coverage is only a function of i and is independent of N.

To derive a method to reduce the probability of conflict, we introduce the concept of exclusive node set (ENS). Let $E_i(X)$ be a set of input nodes which includes X and all the elements in $G_j(X)$,$\forall j \leq i$, i.e.:

$$E_i(X) \triangleq \{X\} \cup G_0(X) \cup \cdots \cup G_i(X) \tag{4}$$

It is easy to obtain several properties of $E_i(X)$: (1) $|E_i(X)| = 2^{i+1}$. (2) $\forall X' \in E_i(X)$, $E_i(X') = E_i(X)$. (3) $\forall X' \in \overline{E_i(X)}$ $\Rightarrow \forall X'' \in E_i(X'), E_i(X'') \cap E_i(X) = \phi$.

From the properties above, we can see that the N input nodes can be divided into $\frac{N}{2^{i+1}}$ non-intersecting i-th level ENS (numbered as $E_i^1, E_i^2, \ldots, E_i^{N/2^{i+1}}$). To reduce the possibility of routing conflict, no more than one node from each ENS should be selected as input node. Thus the maximum number of input nodes that can be routed simultaneously is the same as the number of ENS, i.e., $N/2^{i+1}$. This has been illustrated in Fig. 7.

From Fig. 6 and Fig. 7, we see that as i increases, the minimum pair-wise coverage for any two nodes among different i-th level ENS increases with i, while the maximum number of input nodes that can be simultaneously routed decreases exponentially. For example, if 0-th level ENS is applied, there are at most $\frac{N}{2}$ nodes that can be simultaneously routed with minimum pair-wise coverage of 75%. If 1-th level ENS is applied, the minimum pair-wise coverage increases to 87.5% while the number of nodes that can be simultaneously routed decreases to $\frac{N}{4}$. Since the row weight of the parity check matrix is upper bounded by the number of nodes that can be simultaneously routed, it is desirable to have this number as large as possible. This is because for the same block size, the LDPC code with higher row weight can achieve the same BER performance with higher code rate (and thus higher throughput), compared with the code with lower row weight.Therefore, if there are no routing conflicts, lower level of ENS will result in higher throughput. Thus the guideline for selection of ENS is clear: start from lower level of ENS (0-th level); if there are routing conflicts that can not be solved, use higher level of ENS. Fortunately, as shown in the next section, in most cases there are no conflicts even for 0-th level of ENS.

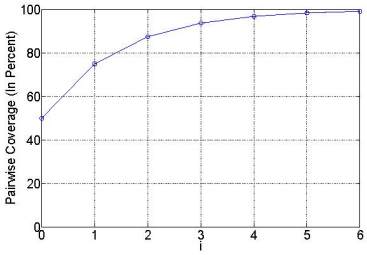

Fig. 6. Pairwise coverage $C_i(X,Y)$ (in percent) for different i, $\forall X, Y \in G_i(X)$.

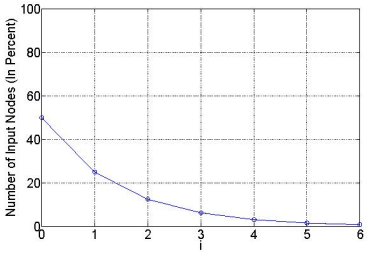

Fig. 7. Number of input nodes (in percent) that can be simultaneously routed when applying i-th level ENS.

3.1.2. Code Features

In the previous section, we derived the simple rule that by avoiding simultaneous routing of nodes from the same ENS, we can map the code more efficiently into the target SDR platform. So for the case of 0-th level of ENS, only $\frac{N}{2}$ input nodes should be routed simultaneously.

An efficient mapping of the LDPC code onto the SDR platform equipped with MINs requires that the number of input nodes equals the maximum row weight (d_c), and the number of block column (n_b) is less than the size of the MIN (N). So the compromise mentioned above can be translated to $n_b \leq N$ and $d_c \leq \frac{N}{2}$. Because of the sparseness of the parity check matrix for LDPC codes, we can easily find such configurations.

The concept of neighborhood groups can also be defined from the output node point of view. Let $P_i(X)$ be the set of output nodes that can be traced back to the same 2×2 switch at stage $n-i-1$. It is desirable to avoid routing X and $Y \in P_0(X)$ in the same block row when the decoding is scheduled based on row number. Because the MIN structures that are used in this paper all have a switch stage before the output, this translates to no consecutive '1's in each row.

The constraints on n_b and d_c can be further refined by considering the encoding complexity. It was shown in [13] that linear

encoding complexity can be achieved by restricting the shape of H_b to be lower triangular. In this case, the conditions become $n_b \leq N$ and $d_c \leq \frac{N-m_b-1}{2} + d^v_{min}$, where d^v_{min} is the lowest variable node degree. Fig. 8 sketches the idea of the derivation. The shaded part of the parity check matrix corresponds to the positions of possible non-zero elements: the first row is the one with the densest '1's in the shaded part and so can be used to determine the d_c value. At the lower part of the H_b matrix, there are rows that have the '1's distributed over all the n_b block columns. The number of such rows for an interconnection-aware LDPC code is $2d^v_{min} - 1$ instead of d^v_{min} (applying the pigeonhole principle to the right most two block columns). If $n_b = N$, the maximum value of $d_c = \frac{N-m_b-1}{2} + d^v_{min}$; if $n_b < N$, the result remains the same since we can amortize the un-used $N - n_b$ output nodes for the left part of H_b.

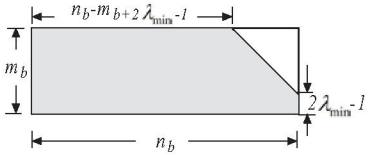

Fig. 8. Parity check matrix for the encoder efficient case.

The rule that only one node should be used from each ENS can be translated into the following property of the H_b matrix: there is no more than one '1' in the elements belonging to the same ENS for any row (since every block column corresponds to one node, the name ENS is used for both nodes and block columns of H_b). Thus it is clear from the pigeonhole principle, that the maximum variable node degree $d^v_{max} \leq m_b - d^v_{min}$.

The constraints are thus summarized as follows:

- $n_b = N, d_c \leq \frac{N-m_b-1}{2} + d^v_{min}$.
- $d^v_{min} \geq 2, d^v_{max} \leq m_b - d^v_{min}$.
- No more than one '1' within each ENS for any row.

By applying the constraints presented above in step **(v)** in Section 2.3, the routing conflicts greatly reduce. Our experimental results show that the algorithm converges very well. Next we will show a code design example following these rules.

3.2. LDPC Code Design Example

In this section, we illustrate our procedure in design of a rate $\frac{3}{4}$ LDPC code which fully exploits the characteristics of an SDR platform equipped with a 32-point butterfly MIN. From the discussion above, we can derive the design parameters as follow:

- n_b = 32, m_b = 8, code rate $r = \frac{3}{4}$
- d^v_{min}=2, $d^v_{max} \leq 6$, check node degree $d_c = 13$

The sub-matrix size Z can be determined by the calculation of multiplicative order. A prime number Z qualifies for a block size if there exist two generator elements a and b in $GF(Z)$ that have multiplicative order of m_b and n_b respectively. In this setting, $Z = 97, 193, 257, 353, 449, ...$ can serve as the size of each sub-matrix. For $Z = 97$, the shift values for (i, j) element in H_b can be generated by $a = 33$ (row) and $b = 19$ (column), i.e. shift value=$a^{i-1}b^{j-1}$.

The optimal degree distribution can be obtained with the online tool [12]. After that we round the bit node number array to an integer version with the total edge number constraint. The result is shown in Table 2.

Table 2. Bit/Check node degree design result

Node Degree	Optimal Distribution	Design Result	
Bit Node	Optimal Bit Dist.	Node #	Actual Dist.
2	0.44277513606938	14	0.43750
3	0.21083725527093	7	0.21875
5	0.34638760865969	11	0.34375
Check Node	Optimal Check Dist.	Node #	Actual Dist.
13	1	8	1

Next, we generate the H_b matrix with the given degree distribution. The constraints introduced by the MIN are included in the generating process. Fig. 9 and 10 shows one of the H_b matrices before and after the interconnection-aware construction. In the H_b matrix before interconnection-aware construction, there exist cases where there are two '1's in the same row of an 0-th level ENS. Note that each 0-th ENS for butterfly network consists of two elements that only differ in the last bit in their binary forms. These have been alternatively highlighted in Fig. 9 and Fig. 10. In the H_b matrix after the interconnection-aware construction, all such cases are eliminated.

After the code is constructed, we map the code to the target architecture. We only present the result of mapping the input and output nodes of MINs. There are $d_c = 13$ input nodes and $n_b = 32$ output nodes. The mapping of input nodes and output nodes is not unique: one possible input mapping is $\{2, 4, 6, \ldots, 26\}_d$ and one possible output mapping is $\{0, 1, 2, \ldots, 31\}_d$. Table 3 shows the settings of all the 2×2 switches in each stage for the different block rows. Here '0' and '1' represents straight and cross connection respectively, '_' means that the switch is not used thus the state can be either '0' or '1', the conflict on the switch state will be represented by an 'X'. In this example there are no conflicts in any switch and thus no conflicts in the routing. Consequently, there is no additional delay in the routing and the implementation of this LDPC code on the MIN based architecture has a very high throughput.

In contrast, if the original H_b matrix is used, there are 4.1 conflicts per block row on average (4.56% of all the switches). When a conflict happens, the corresponding data have to be routed separately, thus slowing the system down. For the original H_b matrix, the 4.56% conflicts will cause the mapped architecture to slow down by 34%. The slowing down factor is calculated by dividing the additional number of cycles, which are introduced by separately routing the data that caused conflicts, by the overall number of cycles.

Finally, we verify the performance of the code before and after interconnection oriented optimization. Fig. 11 shows the BER performance of for AWGN channel when the LDPC encoded data is modulated with binary phase shift keying (BPSK). The interconnection oriented code has less than 0.2 dB degradation in the "waterfall" region and has much lower error floor. Thus the proposed code generation procedure did not introduce any degradation in the BER performance.

4. CONCLUSION

In this paper, we presented a procedure to design LDPC codes that can be mapped efficiently onto an existing SDR system. The general design flow is given, followed by specifics of the case in which

H_b=

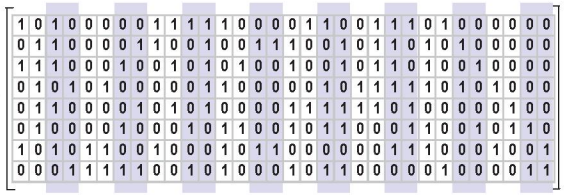

Fig. 9. H_b matrix before interconnection-aware construction.

H_b=

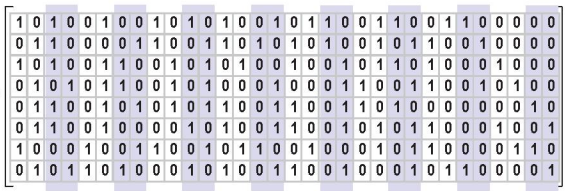

Fig. 10. Interconnection-aware H_b matrix for a rate $\frac{3}{4}$ code.

Table 3. Settings of the Switches in the butterfly MIN for the interconnection-aware LDPC code

Row	Stage 0	Stage 1	Stage 2	Stage 3	Stage 4
1	_0100101010101__	1_000110011001__	011_1100001111__	000_1111111100__	001_0001010011__
2	_0110101010101__	1__00110011001__	01_11100001111__	00_01111111100__	10_10100110100__
3	_0101010010100__	1_00110001100__	011_100000111_1_	0001111_11110_0_	0010111_10011_1_
4	_0101011101010__	1_0011_0001100__	011_10_00001_11_	000111_1_111100_	111011_1_011000_
5	_0101010101001__	1_001100110000__	011_100001111__1	0001111_111_0__0	10111100101_1__1
6	_0101010101001__	1_000011001110__	011_11100001__11	000__111_1111_00	101__00100010_10
7	_0010101001001__	1_1001100_1110__	0_1111000_01__11	0_0011111_111_00	0_0101100_010_01
8	_0101101011011__	1_00_1100111_0__	011__10000_1_1_1	0001_11111_110_0	1100_00110_001_0

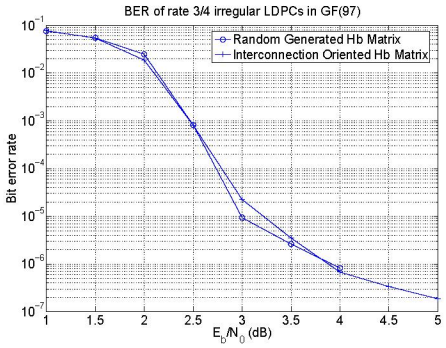

Fig. 11. BER performance of the code under AWGN.

the SDR platform is equipped with MIN interconnection networks such as butterfly interconnection network and perfect shuffle network. The characteristics of the MIN are studied and translated into explicit constraints in the code construction step. The resulting LDPC code for butterfly MIN has superior timing performance since the routing conflicts through the MIN have been minimized. Furthermore, the BER performance is comparable to that of a randomly generated code with the same degree distribution. The procedure and analysis method used in this paper is quite general and can be used for any kind of MIN.

5. REFERENCES

[1] R. G. Gallager, "Low-Density Parity-Check Codes," *IRE Trans. Inform. Theory*, vol. IT-8, no. 1, pp. 21–28, Jan. 1962.

[2] Sae-Young Chung; G.D. Forney; T.J. Richardson; R. Urbanke, "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit," *IEEE Commun. Lett.*, vol. 5, no. 2, pp. 58–60, Feb. 2001.

[3] C. Howland; A. Blanksby, "A 220 mW 1 Gb/s 1024-bit rate-1/2 low density parity check code decoder," in *IEEE Conf. on Custom Integrated Circuits (CICC)*, May 2001, pp. 293–296.

[4] Y. Chen; D. Hocevar, "A FPGA and ASIC implementation of rate 1/2, 8088-b irregular low density parity check decoder," in *Proc. of Global Telecom. Conf. (GlobeCom)*, Dec. 2003, vol. 1, pp. 113–117.

[5] M.M. Mansour; N.R. Shanbhag, "High-throughput LDPC decoders," *IEEE Trans. VLSI*, vol. 11, no. 6, pp. 976–996, Dec. 2003.

[6] H. Zhong; T. Zhang, "Block-LDPC: A Practical LDPC Coding System Design Approach," *IEEE Trans. Circuits and Systems I*, vol. 52, no. 4, pp. 766–775, April 2005.

[7] Y. Zhu; C. Chakrabarti, "Aggregated Circulant Matrix based LDPC codes," in *Proc. of IEEE ICASSP*, May 2006, vol. 3, pp. 916–919.

[8] Y. Lin, H. Lee, M. Woh, Y. Harel, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner, "SODA: a low-power architecture for software radio," in *The 33rd Ann. Int. Symp. on Computer Arch. (ISCA)*, June 2006.

[9] Y. Kou; S. Lin; M.P.C. Fossorier, "Low density parity check codes: construction based on finite geometries," in *IEEE Global Telecomm. Conf. (GLOBECOM)*, Nov. 2000, vol. 2, pp. 825–829.

[10] R.M. Tanner; D. Sridhara; A. Sridharan; T.E. Fuja; D.J Costello, "LDPC Block and Convolutional Codes Based on Circulant Matrices," *IEEE Trans. Inform. Theory*, vol. 50, no. 12, pp. 2966–2984, Dec. 2004.

[11] T.J. Richardson; M.A. Shokrollahi; R.L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. on Inform. Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001.

[12] R. Urbanke, "LDPC optimization program," Available at: http://lthcwww.epfl.ch/research/ldpcopt/.

[13] T.J. Richardson; R.L. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Trans. on Inform. Theory*, vol. 47, no. 2, pp. 638–656, Feb. 2001.

[14] A. Amraoui; A. Montanari; R. Urbanke, "Finite-length scaling of irregular LDPC code ensembles," in *IEEE Information Theory Workshop, 2005*, Aug. 2005, pp. 1–5.

[15] C.L. Wu and T. Y. Feng, "On a class of multistage interconnection networks," *IEEE Trans. on computers*, vol. C-29, pp. 694–702, Aug. 1980.