A Reduced Routing Network Architecture for Partial Parallel LDPC decoders

By

HOUSHMAND SHIRANI MEHR
B.S. (Sharif University of Technology) July, 2009

THESIS

Submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

in

Electrical and Computer Engineering

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Bevan M. Baas, Chair

Venkatesh Akella

Soheil Ghiasi

Committee in charge
2012

Abstract

A novel partial parallel decoding scheme based on the matrix structure of LDPC codes proposed in IEEE 802.15.3c and IEEE 802.11ad standards is presented that significantly simplifies the routing network of the decoder, and the class of parity-check matrices for which the method can be used is defined. The proposed method results in an almost complete elimination of logic gates on the routing network, which yields improvements in area, speed and power, with an identical error correction performance to conventional partial-parallel decoders. A decoder for the (672,588) LDPC code adopted in the IEEE 802.11ad standard is implemented in a 65 nm CMOS technology including place & route with both proposed permutational decoder, and conventional partial-parallel architecture. The proposed permutational LDPC decoder operates at 235 MHz and delivers a throughput of 7.9 Gbps with 5 decoding iterations per block. Compared to a conventional partial-parallel decoder, the proposed decoder achieves a throughput 30% higher and at the same time requires a chip area approximately 24% smaller.

Acknowledgments

First, I would like to sincerely thank my advisor Professor Bevan Baas for his support and guidance throughout my years at UC Davis. I was extremely lucky to have the opportunity of working in VLSI Computation Lab under his supervision, and it was thoroughly an enriching and delightful experience. I am grateful to Professor Shu Lin for introducing me to concepts of Error Correction Coding. Without his through instructions, I would not be able to complete this work. I would also like to thank Professor Soheil Ghiasi for his constant support and valuable advice through my graduate study. Many thanks to Professor Ghiasi and Professor Venkatesh Akella for their time and consideration in reviewing my thesis.

There were many people without their help this work could not be accomplished. I would like to express my appreciation to Tinoosh for introducing me to LDPC decoders, helping me with learning the tools, and her endless support and valuable advice through the projects we worked together. I would also like to thank all the people at VCL lab for providing me with a friendly and exciting environment, inspiring me to keep following my research aspirations. Many thanks to all my friends at UC Davis for giving me some of my most enjoyable memories, specially Saeed, Mahnoosh, Mohammad, Mina, Marjan, Ladan, Jon, Luis, and Matin.

I am grateful for the support from our sponsors, ST Microelectronics, UC Micro, NSF Grant 0430090, CAREER Award 0546907, NSF Grant 0903549 and 1018972, SRC GRC Grants 1598 and 1971 and CSR Grant 1659, and Intellasys. I am also thankful for the support of Cadence and Synopsys for the tools provided, and C2S2 Focus Center, one of six research centers funded under the Focus Center Research Program (FCRP), a Semiconductor Research Corporation entity.

Finally, my special thanks goes to my beloved family for all their support and patience throughout these years. I am grateful to my brothers Hooman and Houtan for always being an example in my career, and I am mostly thankful to my parents for giving me the confidence to follow my passion, and for their endless patience and love.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Low density parity check (LDPC) codes were first introduced by Galallager in 1962 [1], and after being rediscovered in 1996 [2], these error correction codes have been constantly in the center of attention for researchers. Their superior correction performance and highly parallelizable decoding algorithms have resulted in adoption in many recent communication standards. In recent years, digital video broadcasting via satellite (DVB-S2) [3], WiMax standard for microwave communications (802.16e) [4], 10GBASE-T standard for 10 Gigabit Ethernet (802.3an) [5], Wireless Personal Area Network (802.15.3c) [6], and new WiMax standards (802.11ad) [7] have utilized LDPC codes as their error correction method. However, there are certain challenges in efficient hardware implementation of codecs for LDPC codes. One of the major issues is that their decoder circuit exhibits high interconnection complexity. As a result, the inherit parallelism in the decoding algorithm of LDPC codes cannot be efficiently exploited. On the other hand, throughput and power requirements of emerging communication systems have been constantly growing. For instance, recent wireless communication standards have increased their throughput to beyond 1 Gbps [8,9,10]. These factors have resulted in high demand for innovative and adequate methods to tackle design challenges for LDPC codes.

Depending on the approach to exploit parallelism in the decoding algorithm of the code, LDPC decoders are divided into two main categories: full-parallel and partial-parallel decoders. In full-parallel decoders, every processing node is implemented in hardware, and these nodes are connected through global wires based on the parity check matrix [11].

In partial-parallel architectures, a subset of check nodes and variable nodes of the parity check matrix is implemented in hardware. By employing memory units and changing routing network

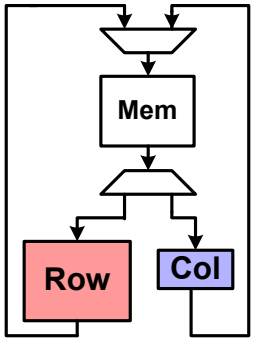

Figure 1.1: The schematic for a general partial-parallel decoder

between nodes, the update stage for different partitions of the matrix are processed. Figure 1.1 shows a general partial-parallel decoder architecture. Adjusting the interconnection network to different partitions is achievable by utilizing permuter networks [12], or in general a network of muxes [13]. The objective of these networks is to change the path of every bit of messages transmitted between check nodes and variable nodes in different cycles. Therefore, a significant number of muxes is required, which results in a substantial hardware overhead, and a considerable power dissipation due to constant toggling over cycles. Additionally, since these muxes are in the critical path of the signals passing through decoder, a decline in the throughput is observed.

In this work, the parity check matrix structure of architecture-aware LDPC codes proposed for two recent IEEE standards is investigated, and based on its characteristics, a new decoding technique is proposed. The objective of this new method is to minimize the gate count on the routing network of the decoder, and subsequently decrease the routing network complexity. As the result, improvements in area, throughput and power are observed. This new method can be generalized for any LDPC code with the same characteristics (defined here as *permutational*). The new scheme is based on the layered belief propagation (LBP) algorithm [14], which improves the speed of convergence by a factor of two compared to regular belief propagation decoding [15].

1.1 Previous Work

Alleviating the interconnection complexity by improving implementation techniques based on properties of structured LDPC codes [16], or improving the switching networks in general [17], has been a constant field of research in recent years. However, still a considerable number of logic

gates are used to change the configuration of routing network between cycles.

Two major publications have discussed LDPC decoders for codes similar to the ones discussed in this work. In first one, a decoder supporting four LDPC codes in IEEE 802.15.3c is presented with the idea of simplifying the routing network by utilizing same decoding behavior between different layers and rates [18]. The proposed decoder still requires 63,168 multiplexer inputs on the routing network. In the second work, a flexible fully pipelined architecture is implemented for 802.11ad, supporting power and throughput requirements of the standard [19]. The advantage of the proposed decoder is the flexibility to support different codes. However, it is achieved by applying shifter networks which contain 14 levels of 672 2:1 multiplexers.

Although the architecture presented in this work is capable of supporting all codes in IEEE 802.15.3c and 802.11ad, it is not flexible enough to support any LDPC code. However, for designs specific to these two standards or codes designed with same structure, this architecture can achieve throughput requirements with far improved area and power characteristics compared to previous work.

1.2 Project Contributions

The matrix structure of LDPC codes proposed for IEEE 802.15.3c and IEEE 802.11ad standards is investigated. By defining *valid mapping* and *permutational parity check matrix*, the class of matrices for which the new decoding scheme can be utilized is fully described. This class of matrices (called *permutational*) contain all LDPC codes in 802.15.3c and 802.11ad. The general architecture of the new decoding method is proposed for codes in this class with details of check node and variable node processes. The dynamics of the *permutational* LDPC decoding is fully explained on a sample matrix. It is shown how the constant shifting of values in the datapath of the decoder over multiple cycles results in decoding process corresponding to different layers of parity check matrix. the proposed scheme is implemented in 65 nm CMOS technology including place & route for the (672,588) LDPC code adopted in IEEE 802.11ad standard. The results are compared with a conventional partial-parallel decoder implemented with the same characteristics, and other similar LDPC decoders including the ones presented in previous work section.

Implementation results for the proposed decoder in 65 nm technology for LDPC code in 802.15.3c shows over 96% reduction in number of gates on the routing network of the decoder. This reduction in complexity of the routing network yields to 1.26× improvement in area and 1.3× improvement in throughput of the decoder.

1.3 Organization

This thesis is organized as follows: Chapter 2 introduces LDPC codes and decoding techniques in details. Sum-Product, Normalized Min-Sum, and Layered Normalized Min-Sum algorithms are explained, and differences between decoding architectures are described. Chapter 3 introduces the generalized set of parity check matrices for which the method can be applied. It also presents a general architecture for these codes, and detailed description of decoding scheme and its properties. The hardware implementation of check nodes, variable nodes, shift network and routing network are described as well. Chapter 4 shows how the architecture can be applied to LDPC codes in 802.11ad and 802.15.3c standards. The decoder is implemented in 65 nm CMOS technology including place & route for (672,588) code in 802.11ad, and the results are compared to similar decoders. Finally, Chapter 5 concludes the work and states future work directions.

Chapter 2

LDPC Decoding

There are certain challenges in efficient hardware implementation for LDPC codes. In order to meet the expectations of communication standards, LDPC codec designs should have high throughput, low area, and low power. High performance decoders need many processing nodes and large interconnection circuitry [20]. This interconnection complexity is a major draw-back in efficient VLSI designs for LDPC codes. LDPC decoders are divided based on how they face this challenge into two categories: Full-parallel decoders, and partial-parallel decoders.

In full-parallel decoders, every computational unit in decoding algorithm is realized in hardware, and global wires are used for connecting the units [11]. Full-parallel decoders have higher throughput [21], since they can process one iteration of decoding in just one cycle. Another advantage of these decoders is that their energy efficiency is theoretically the best [22]. However, the interconnection complexity and irregularities in connection matrix cause inefficiencies in their hardware implementation [20]. Their hardware utilization is low, and large amount of power is dissipated in their long global wires. Finally, the clock rate is decreased because of long critical path in the decoder.

Partial-parallel decoders realize a part of parity-check matrix in hardware instead, and other parts are processed by configuring the realized hardware [23]. These decoders usually use pipelining, large memory resources, and shared computational blocks [20]. Throughput is decreased due to several cycles needed to process one iteration. Memory resources also cause decline in throughput, power, and area. Changing the interconnection in every cycle is another factor that adds to the power dissipation.

LDPC decoding is mainly done by iterative message passing algorithm [24, 25]. In this

algorithm, information is received from channel, and then by passing it between check nodes and variable nodes for a couple of iterations, the received data is decoded. The iterative process of LDPC decoding consists of two stages: 1) check node processing; 2) variable node processing. The equations for check node processing and variable node processing are published in many papers. There are two major variants of LDPC decoding: Sum-Product(SP) [26] which uses an *arctg* function and is more accurate; and an approximate to Sum-Product algorithm called Min-Sum (MS) [27], which is used widely in LDPC decoders due to its smaller hardware and low BER degradation. Layered Normalized Min-Sum decoding [14] is another variant which is utilized in this thesis as the decoding algorithm.

In this chapter, iterative message passing algorithm for decoding LDPC codes is explained in details and its variants including layered decoding are described. Finally, the LDPC decoding architectures are mentioned.

2.1 LDPC codes

LDPC codes are among linear error correcting codes, which are used to transmit data through noisy channels. In general, error correction is a method of obtaining reliable transmission over unreliable channels by sending more bits than necessary to represent data. The added redundancy to the transmitted information increases the minimum distance between two distinct codewords, so if the noise in the channel changes a limited number of bits, correct information can still be retrieved. An encoder implemented in the transmitter circuit maps the information to codewords, and the decoder at the receiver end is responsible to retrieve the transmitted data. There are many error correcting codes available today and LDPC codes are one of the highly efficient codes adopted in different communication standards.

Definition *H parity check matrix*: The binary $M \times N$ matrix that uniquely defines the LDPC code. The number of rows in the parity check matrix, M, is equal to the number of check nodes in the decoder and the number of parity check equations for the code. The number of columns, N, is the number of variable nodes and the number of bits in each codeword. The 1's in the parity check matrix determine how check nodes and variable nodes are connected in the decoder. Furthermore, $K = N - rank$ specifies the information length, and the rate of the code, K/N, shows how much information is transmitted by each bit.

$$H = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

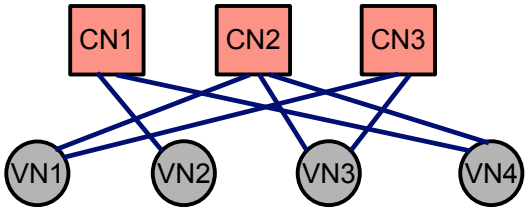

Figure 2.1: A parity check matrix and its corresponding tanner graph

Definition *Row weight*: The number of 1's in a row of a parity check matrix is specified by the row weight, W_r.

Definition *Column weight*: The number of 1's in a column of a parity check matrix is specified by its column weight, W_c.

Definition *Irregular LDPC codes*: The LDPC code is irregular if its rows or columns have different weights. Consequently, in regular LDPC codes, row weights are the same for all rows and column weights are the same for all columns.

Definition *Tanner graph:* For every column and row in the parity check matrix, a variable node or check node is defined respectively. Then two nodes are connected if their corresponding element in the matrix is 1. The result is called a Tanner graph [28].

Figure 2.1 shows a very simple parity check matrix and its corresponding tanner graph. The matrix has three check nodes and four variable nodes ($M = 3$, $N = 4$). The connections between variable nodes and check nodes are based on the 1's in the matrix. The row weights are not the same for all rows, therefore the corresponding code is irregular. First row in the matrix implies that there is a parity check equation between second and fourth bits in its codewords.

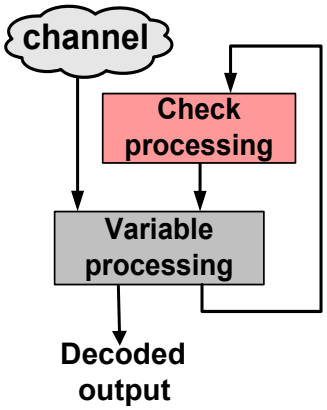

Figure 2.2: The iterative decoding process for LDPC codes

2.2 LDPC Decoding Algorithms

As mentioned earlier, LDPC decoding is usually done through an iterative message passing algorithm. The simplified flow for the process is shown in Figure 2.2. The information from channel is initialized in variable node messages, and check node messages are initialized to zero. Then updated messages from check nodes are transmitted to variable nodes based on connections in Tanner graph. After variable node update stage is done, they pass their messages to check nodes again, and generate output bits. The output bits go through a syndrome check stage to check if they pass parity check equations and match any of the codewords. Iterative message passing between check nodes and variable node continue until the input is decoded and output bits match one of the codewords, or the number of iterations pass a certain threshold.

In order to explain the decoding algorithms and its variations in detail, the following definitions are used throughout this work:

λ_j Log-likelihood ratio of channel information (*a priori value*) for j-th variable node.

R_{ij} Message from check node i to variable node j.

Q_{ij} Message from variable node j to check node i.

Q_j Sum of check node messages and channel information in variable node j (*a posteriori probability ratio*).

The set of variable nodes connected to check node i is denoted by $V(i)$, and this set excluding variable node j is shown by $V(i)\backslash j$.

2.2.1 Sum-Product Algorithm

Sum-Product algorithm [26] is the accurate version of LDPC decoding based on iterative message passing algorithm. Assume that a binary codeword $(x_1, x_2, ..., x_N)$ is transmitted over an additive white Gaussian noise (AWGN) channel using binary phase-shift keying (BPSK) modulation, and $(y_1, y_2, ..., y_N)$ is received instead. Log-likelihood ratio of channel information (λ_j) is calculated from:

$$\lambda_j = \log_e(\frac{P(x_j = 0|y_j)}{P(x_j = 1|y_j)}) \tag{2.1}$$

Sum-Product algorithm is explained in these steps:

1. *Initialization:* Q_j values are initialized by the log-likelihood ratio of channel information (λ_j), and all the messages between check nodes and variable nodes are set to zero. In other words, variable node outputs (Q_{ij}) are initialized by the log-likelihood ratio of channel information in the corresponding bit for that variable node (λ_j).

2. *Update stage - check node update:* For every check node i, the following equation is calculated for every variable node j connected to that check node $(j \in V(i))$:

$$R_{ij} = \underbrace{\prod_{j' \in V(i) \setminus j} \text{sign}(Q_{ij'})}_{\text{Sign Calculation}} \times \underbrace{\phi^{-1}(\sum_{j' \in V(i) \setminus j} \phi(|Q_{ij'}|))}_{\text{Magnitude Calculation}} \tag{2.2}$$

$$\phi(x) = -\log(\tanh\frac{|x|}{2}) \tag{2.3}$$

   The $\phi(x)$ function is implemented in hardware by using a look-up table.

3. *Update stage - variable node update:* After check node to variable node messages are calculated, for each variable node, j, the message going to check node i is calculated as follows:

$$Q_{ij} = \lambda_j + \sum_{i' \in C(j) \setminus i} R_{i'j} \tag{2.4}$$

   Same way as in the check node update stage, information from all connected check nodes are used in the calculation of Q_{ij} except message from check node i. The variable node and check node update stages are processed iteratively.

4. *Syndrome check and early termination:* In variable node update stage, in addition to messages

to check nodes, a posteriori probability ratio is calculated for each variable node:

$$Q_j = \lambda_j + \sum_{i \in C(j)} R_{ij} \tag{2.5}$$

The sign of Q_j is used as an estimation for the output bit corresponding to variable node j. In other words, assuming the output code vector to be $\hat{X} = (\hat{x}_1, \hat{x}_2, ..., \hat{x}_N)$, the estimated code vector is calculated from:

$$\hat{x}_j = \begin{cases} 1, & \text{if } Q_{ij} \leq 0 \\ 0, & \text{if } Q_{ij} > 0 \end{cases} \tag{2.6}$$

Then the syndrome check is done by observing if $H \cdot \hat{X}^T = 0$. Without early termination, the decoding process is finished after a pre-defined number of iterations, I_{max}, has passed. However, if the syndrome check was satisfied, the output code vector is a valid codeword for the parity check matrix, and the decoding process can be terminated earlier than I_{max}.

2.2.2 Normalized Min-Sum Algorithm

As mentioned before, the check node update stage in Sum-Product algorithm is implemented in hardware by utilizing a look-up table for the $\phi(.)$ function, which results in hardware overhead. By accepting a negligible degradation in bit-error rate performance, Min-Sum algorithm [27] simplifies the check node processing for efficient hardware implementation. In this algorithm, instead of computing a non-linear function, $min(.)$ is calculated over check node inputs. In order to compensate for BER degradation, a normalizing factor ($Sfactor_{MS}$) is utilized as well. The check node update equation for Min-Sum algorithm is then:

$$R_{ij} = Sfactor_{MS} \times \underbrace{\prod_{j' \in V(i) \setminus j} \text{sign}(Q_{ij'})}_{\text{Sign Calculation}} \times \underbrace{\min_{j' \in V(i) \setminus j} (|Q_{ij'}|)}_{\text{Magnitude Calculation}} \tag{2.7}$$

Other steps in the decoding process such as initialization, variable node update stage, and syndrome check are exactly the same as Sum-Product algorithm.

In Min-Sum algorithm, in order to find the check node output R_{ij}, $min(.)$ over all variable node outputs connected to check node i excluding variable node j should be calculated. This process is simplified by finding first and second minimum (min_1, min_2) over magnitude of variable node outputs connected to check node i. Assuming min_1 comes from variable node j_{min1} and min_2

from variable node j_{min2}:

$$R_{ij} = \begin{cases} \prod_{j' \in V(i) \setminus j} \text{sign}(Q_{ij'}) \times min_1, & \text{for } j \neq j_{min1} \\ \prod_{j' \in V(i) \setminus j} \text{sign}(Q_{ij'}) \times min_2, & \text{for } j = j_{min1} \end{cases} \tag{2.8}$$

Moreover, sign calculation can efficiently be implemented in hardware by XOR gates.

2.2.3 Layered Normalized Min-Sum Algorithm

In conventional belief propagation algorithm for LDPC decoding, the processing stage for check nodes and variable nodes are done separately in each iteration. In other words, processing variable nodes in one iteration is done after processing of all the check nodes is completed for that iteration. However, in layered decoding the variable or check nodes can update their outputs after partial processing of the other set of nodes. Layered LDPC decoding [14] can increase the speed of convergence for code blocks by two times [15].

In horizontal version of layered LDPC decoding used here, the check nodes are divided into a number of layers, Y. In each iteration, after the processing of check nodes in one layer is finished, their messages to variable nodes get updated. In belief propagation algorithm, variable nodes update their outputs after processing of all check nodes in layers are complete. However in layered decoding, variable nodes use these inputs to update their messages to check nodes in layers which are not processed yet in current iteration. In this work, layered scheduling with normalized Min-Sum [29] as the update procedure in the check nodes is utilized.

The layered decoding algorithm using normalized Min-Sum in check nodes is summarized in the following steps:

1. *Initialization:* Q_j values are initialized by the log-likelihood ratio of channel information (λ_j), and all the messages between check nodes and variable nodes are set to zero.

2. *Processing of layers:* Assume $L_0, L_1, ..., L_{Y-1}$ to be the layers of the matrix, then for every decoding iteration:

   for $k = 0 : (Y - 1)$ **do**

   for $i \in$ check nodes of L_k **do**

$$Q_{ij} = Q_j - R_{ij(old)} \tag{2.9}$$

$$R_{ij} = Sfactor_{MS} \times \underbrace{\prod_{j' \in V(i) \setminus j} \text{sign}(Q_{ij'})}_{\text{Sign Calculation}} \times \underbrace{\min_{j' \in V(i) \setminus j} (|Q_{ij'}|)}_{\text{Magnitude Calculation}} \tag{2.10}$$

$$Q_j = Q_{ij} + R_{ij} \tag{2.11}$$

end for

end for

Where $R_{ij(old)}$ represents the stored value of R_{ij} from previous iteration and $Sfactor_{MS}$ is the correction factor for the normalized Min-Sum. By storing $R_{ij(old)}$ values in check nodes and passing Q_j values through them in each cycle, all the processing for this step can be done in check nodes.

3. *Syndrome check and Termination of Decoding:* Based on Q_j values in every step, the estimated bits for the output are generated by variable nodes based on the following:

$$\hat{x_i} = \begin{cases} 1, & \text{if } Q_i \leq 0 \\ 0, & \text{if } Q_i > 0 \end{cases} \tag{2.12}$$

If the estimated bits satisfy all the parity check equations, or the number of iterations exceeds a predefined maximum value (I_{max}), then the decoding is terminated.

2.3 LDPC Decoding Architectures

The inherent parallelism in LDPC decoding algorithms on one hand, and the large interconnection complexity caused by implementing all parallel nodes in the hardware on the other hand, has resulted in different approaches towards decoding architectures. Based on to what degree the parallelism is exploited, the decoding architectures are divided into two main categories: full-parallel and partial-parallel.

2.3.1 Full-parallel Decoders

In full-parallel architectures, every check node and variable node in the parity check matrix is implemented in hardware, along with all the connections required for passing messages between them. The update stage for check nodes is done in parallel since all the check nodes are implemented in hardware, then the messages are transmitted to the variable nodes, and later variable nodes are processed in parallel as well. Consequently, one iteration of decoding can be done in just one clock cycle. This means that this architecture has the best throughput, and needs no memory to store intermediate messages. Full-parallel implementations have also the best power efficiency among LDPC decoders [22].

However, full-parallel architectures are not widely used unless very high throughputs are required. The main reason is the high interconnection complexity caused by large number of long global wires between check nodes and variable nodes [cite trends]. This problem grows dramatically when the code length increases. The high interconnection complexity results in lower than expected improvements in throughput compared to partial-parallel decoders. Additionally, since the interconnection network is implemented fully in wires, there is no efficient way to reconfigure the architecture to support more than one LDPC code. As the result, this architecture cannot be used as the decoding hardware in standards with multiple code rates. Large circuit area is also another drawback for full-parallel architectures.

2.3.2 Partial-parallel Decoders

The main idea behind partial-parallel implementations is to build a balance between parallelism in decoding and the interconnection wiring complexity. In this architecture, a subset of variable nodes and check nodes are implemented in hardware, and by changing the routing network between implemented nodes, different partitions of parity check matrix are processed. Since intermediate messages need to be stored, memory resources are essential for this architecture. One iteration of decoding takes multiple cycles, which means the throughput is lower compared to full-parallel decoders. However, the decoding circuit is much smaller. Figure 1.1 on page 1 shows the general partial-parallel architecture.

Most of LDPC codes adopted in recent standards belong to Quasi-cyclic class of codes [24, 30]. The parity check matrices of these codes are block structured, which makes them well suited for partial-parallel implementations. The parity check matrix for this class of codes is constructed by smaller submatrices, each is either an all-zero submatrix or a permutation of an identity matrix.

For example,

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

is a quasi-cyclic parity check matrix constructed from 3×3 identity matrix and its permutations:

$$p^0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \; p^1 = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \; p^2 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

The H matrix can also be represented in a shorter format by the permutation index of its submatrices as following:

$$H = \begin{bmatrix} 0 & 1 & - \\ 2 & 0 & 2 \end{bmatrix}$$

where $-$ shows an all-zero submatrix. Although the routing network is much simpler in partial-parallel decoders comparing to full parallel ones, it is still a challenge for efficient hardware implementation, specially when the high throughput requirements force large number of implemented processing nodes in hardware. Adjusting the interconnection network to different partitions is achievable by utilizing Permuter networks [12], or in general a network of muxes [13]. Since the routing network becomes adjustable, reconfigurability can be entered into the design, therefore this architecture is ideal for standards supporting multiple code rates.

The objective of muxes on the routing network of partial-parallel decoders is to change the path of every bit of messages transmitted between check nodes and variable nodes in different cycles. Therefore, a significant number of muxes is required, which results in a substantial hardware overhead, and a considerable power dissipation due to constant toggling over cycles. Additionally, since these muxes are in the critical path of the signals passing through decoder, a decline in the

throughput is observed.

In following chapters, a new partial-parallel architecture based on the characteristics of parity check matrix of LDPC codes adopted in some of recent IEEE standards is presented. This new architecture significantly simplifies the routing network of the decoder, which results in improvements in area, power and throughput.

Chapter 3

Proposed Architecture

In this chapter, our new LDPC decoding architecture is presented. First the class of matrices for which this decoding architecture can be implemented for is introduced. then the decoding method is explained in details on a sample matrix. The class of matrices the method works for is named *permutational* and the method is called *permutational LDPC decoding* here. The decoding scheme uses layered normalized min-sum as the update procedure for check nodes.

3.1 Basic Definitions

3.1.1 Valid Mapping

As mentioned earlier, in layered scheduling the parity check matrix (H) is divided into a number of layers, Y. Each of these layers contains a certain number of check nodes, M_l (i.e. $M_l = M/Y$). Furthermore, assume any horizontal partitioning of the matrix, dividing it into U groups of N_c columns (i.e. $N_c = N/U$). Each of these column partitions are called a *column group* here. Additionally, assume $Submatrix(l, c)$ to be the submatrix in layer l and column group c. Figure 3.1 shows a simple matrix represented by its submatrices. As shown in the figure, the matrix has four layers and four *column groups*, and $Submatrix(1, 3) = C$.

A mapping from the column groups of layer L_1 to column groups of layer L_2 is called *valid* if it has the following two properties:

1. It is one-to-one, meaning each column group in layer L_1 is mapped to one and only one column group in layer L_2,

2. Every non-zero submatrix in layer L_1 is mapped to an equal or an all-zero submatrix in layer

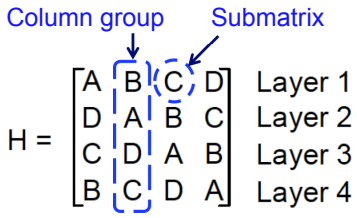

Figure 3.1: Layers, a *column group*, and $Submatrix(1, 3)$ are shown for the parity check matrix H

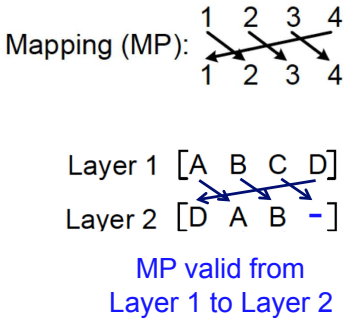

Figure 3.2: The mapping MP is *valid* based on the definition from *Layer* 1 to *Layer* 2

L_2,

Figure 3.2 shows a mapping (MP) that is *valid* from *Layer* 1 to *Layer* 2.

The ability to find a *valid* mapping between two layers implies that the basic elements building these two (submatrices) are the same. This definition is valuable since by finding an efficient way to implement this mapping in the hardware, the number of different connection submatrices required to support both layers cuts in half.

3.1.2 Permutational Matrix

For a parity check matrix, if a permutation of layers shown by $P = (L_0, L_1, ..., L_{Y-1})$ and a mapping MP exists such that:

1. $\forall k \in \{0, ..., Y-2\} : MP$ is *valid* from layer L_k to L_{k+1},
2. MP is *valid* from layer L_{Y-1} to layer L_0.

then the matrix is called *Permutational* with valid mapping MP. Figure 3.3 shows that the matrix in Figure 3.1 is *permutational*.

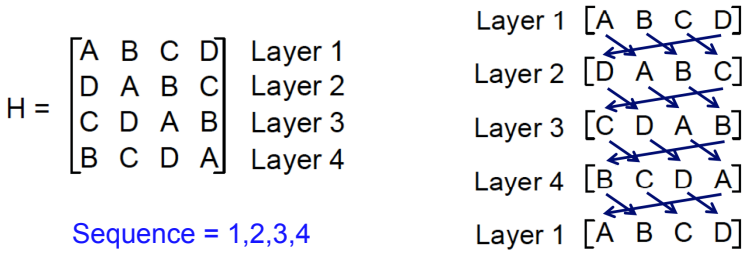

Figure 3.3: A permutation of layers mapped by MP from Figure 3.2 implying matrix H is permutational

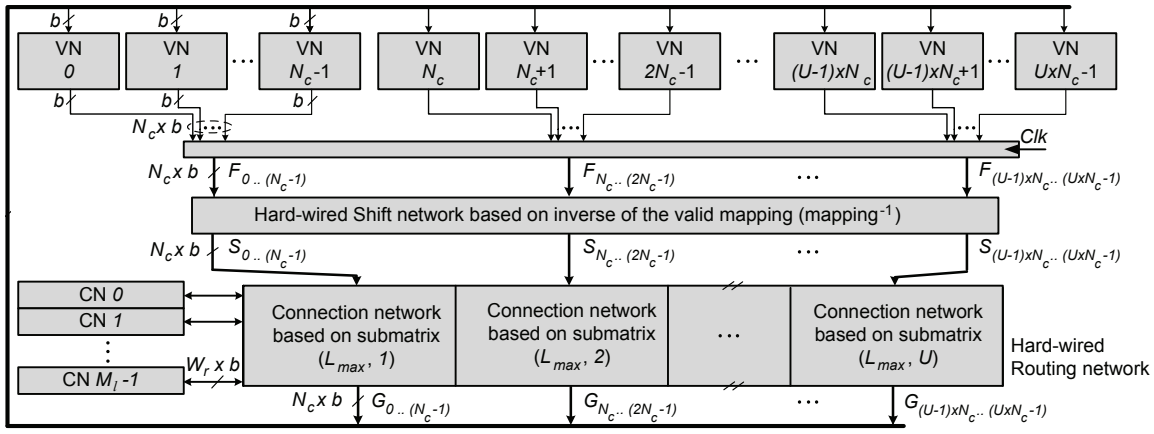

Figure 3.4: The schematics for general permutational LDPC Decoder, the routing network is a constant wiring network implemented based on inverse of *valid* mapping defined on parity check matrix and the layer with maximum row weight, L_{max}

In a permutational parity check matrix, the set of submatrices building layers are the same. Additionally, a constant mapping over one layer can rearrange these submatrices to generate the configuration of all layers.

The definition of permutational parity check matrices covers a wide range of codes, including (672,336), (672,504) and (672,588) LDPC codes adopted in IEEE 802.15.3c (WPAN) standard and (672,336), (672,504) and (672,588) codes proposed in IEEE 802.11ad (WiFi). In the next section, it is shown that a new partial-parallel decoding scheme (called *permutational LDPC decoding* here) can be developed for these matrices that significantly simplifies the routing network of the decoder, by eliminating almost all the gates on the network.

3.2 Permutational LDPC Decoding

In the method proposed in this work, the V-to-C and C-to-V routing networks are hard-wired based on one of the layers in the parity check matrix. These two networks need no gates to adjust decoding process to different layers, except a minimal number of gates for cases where the code is irregular. This exception is explained later in details. In this architecture, an additional fixed wiring network is coupled with V-to-C routing. The responsibility of this network is to perform a constant shift over messages from variable nodes. Although the physical routing network has no change over different sub-iterations, the shift in the variable node messages produces different decoding process, matching to different layers. The method is described in this section.

For a permutational parity check matrix with $Y \times M_l$ rows and $U \times N_c$ columns, the architecture of the general permutational LDPC decoder is shown in Figure 3.4. The number of implemented check nodes are the number of rows in a layer (i.e., M_l), and the number of implemented variable nodes are the overall number of columns in the matrix (i.e., $U \times N_c$). The original routing network between check nodes and variable nodes is hard-wired based on the layer with highest row degree, L_{max}, and the coupled shifting network is implemented based on the inverse of valid mapping defined on matrix ($Mapping^{-1}$). In the first iteration, the variable node output signals are initialized by the log-likelihood ratio of channel information.

In order to explain how the decoding process works, the decoding scheme is implemented for the parity check matrix in Figure 3.1. For this simple matrix, the architecture for the permutational decoder is shown in Figure 3.5. Since the row weights are the same for all layers in the matrix, there is no preferences in choosing the implemented layer in hardware, and layer 4 is chosen arbitrarily. The shifting network is based on the inverse of mapping in Figure 3.2. The naming for the signals in Figure 3.5 is as following:

F_j : Registered output from variable node j,

$F_{0..(N_c-1)}$: The vector of F_j values from first N_c variable nodes,

S_j : Signals after the shifting network,

G_j : Contains check node results after their process is finished (Q_j in equation 2.11)

Table 3.1 shows how the Q_j values corresponding to different column groups in the matrix in Figure 3.1 change over time and in location. The table follows these values in four sub-iterations of the first decoding iteration, and shows how they are updated by check nodes. In this architecture,

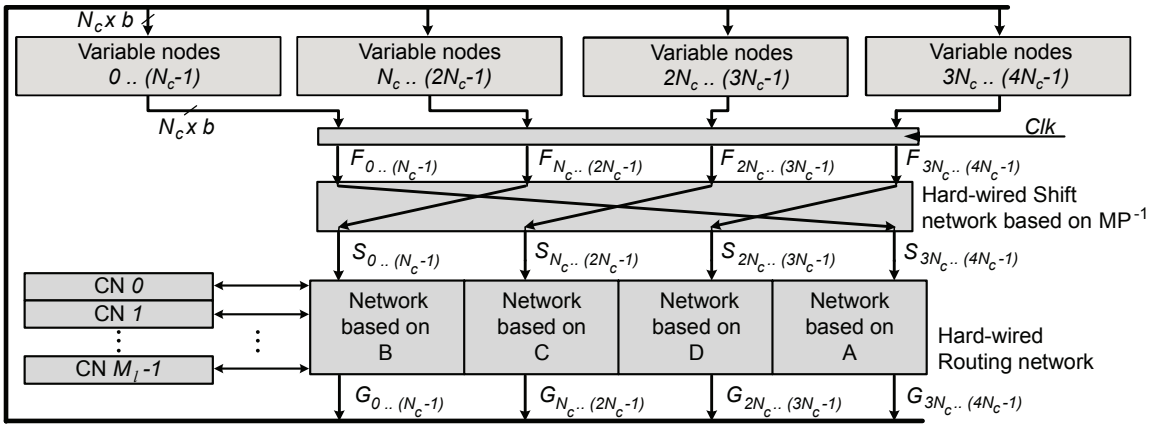

Figure 3.5: The schematics of a permutational LDPC Decoder for the matrix defined in Figure 3.1

Clk	Phase of Process	Q_j of first N_c columns	Q_j of second N_c columns	Q_j of third N_c columns	Q_j of fourth N_c columns	Cycle Result
0	Initial	$\lambda_{0..(N_c-1)}$	$\lambda_{N_c..(2N_c-1)}$	$\lambda_{2N_c..(3N_c-1)}$	$\lambda_{3N_c..(4N_c-1)}$	-
1	Loc. in F	$0..(N_c-1)$	$N_c..(2N_c-1)$	$2N_c..(3N_c-1)$	$3N_c..(4N_c-1)$	
	Loc. in S	$3N_c..(4N_c-1)$	$0..(N_c-1)$	$N_c..(2N_c-1)$	$2N_c..(3N_c-1)$	Layer
	Eff. Conn.	A	B	C	D	1
	Loc. in G	$3N_c..(4N_c-1)$	$0..(N_c-1)$	$N_c..(2N_c-1)$	$2N_c..(3N_c-1)$	proc.
2	Loc. in F	$3N_c..(4N_c-1)$	$0..(N_c-1)$	$N_c..(2N_c-1)$	$2N_c..(3N_c-1)$	
	Loc. in S	$2N_c..(3N_c-1)$	$3N_c..(4N_c-1)$	$0..(N_c-1)$	$N_c..(2N_c-1)$	Layer
	Eff. Conn.	D	A	B	C	2
	Loc. in G	$2N_c..(3N_c-1)$	$3N_c..(4N_c-1)$	$0..(N_c-1)$	$N_c..(2N_c-1)$	proc.
3	Loc. in F	$2N_c..(3N_c-1)$	$3N_c..(4N_c-1)$	$0..(N_c-1)$	$N_c..(2N_c-1)$	
	Loc. in S	$N_c..(2N_c-1)$	$2N_c..(3N_c-1)$	$3N_c..(4N_c-1)$	$0..(N_c-1)$	Layer
	Eff. Conn.	C	D	A	B	3
	Loc. in G	$N_c..(2N_c-1)$	$2N_c..(3N_c-1)$	$3N_c..(4N_c-1)$	$0..(N_c-1)$	proc.
4	Loc. in F	$N_c..(2N_c-1)$	$2N_c..(3N_c-1)$	$3N_c..(4N_c-1)$	$0..(N_c-1)$	
	Loc. in S	$0..(N_c-1)$	$N_c..(2N_c-1)$	$2N_c..(3N_c-1)$	$3N_c..(4N_c-1)$	Layer
	Eff. Conn.	B	C	D	A	4
	Loc. in G	$0..(N_c-1)$	$N_c..(2N_c-1)$	$2N_c..(3N_c-1)$	$3N_c..(4N_c-1)$	proc.

Table 3.1: The table follows Q_j values corresponding to columns of the matrix in Figure 3.1 as they change location in decoder of Figure 3.5 in different sub-iterations, so that in every sub-iteration these values are fed to check nodes through proper connection matrix, and all four layers of the matrix are processed in one decoding iteration

each sub-iteration is processed in one clock cycle. For the purpose of clarification, following Q_j values for first N_c columns (first column group) is summarized in these steps:

1. *Sub-iteration 0:*

   Q_j values should be initialized by the log-likelihood ratio of channel information (λ_j). The Q_j

values are the outputs of the variable nodes in layered decoding algorithm. These outputs are registered to generate F_j signals. Therefore, when the first clock cycle begins, all signals at the output side of main registers in datapath are registered as their corresponding log-likelihood ratio from channel. This means $F_{0..(N_c-1)} = \lambda_{0..(N_c-1)}$,

2. *Sub-iteration 1:*

   (a) The shifting network results in movement of Q_j values corresponding to column groups. This movement is based on the inverse of valid mapping defined on the parity check matrix. Consequently, $S_{3N_c..(4N_c-1)} = F_{0..(N_c-1)} = \lambda_{0..(N_c-1)}$,

   (b) The S_j signals are transmitted to check nodes through V-to-C connection network. This network is implemented based on layer 4 in parity check matrix. As the result, $S_{3N_c..(4N_c-1)}$ $(= \lambda_{0..(N_c-1)})$ are fed to check nodes through submatrix A. This submatrix is the proper connection matrix for $\lambda_{0..(N_c-1)}$ in processing of layer 1. Other λ_j values are connected to check nodes based on layer 1 as well. Therefore, layer 1 is processed properly in first sub-iteration. The updated Q_j's for first N_c columns are produced at $G_{3N_c..(4N_c-1)}$ as the output from check nodes. These signals are then transmitted to last N_c variable node units in the architecture. If the decoded bits from variable nodes are registered at the end of this cycle, the order of output bits is corrupted. For instance, first N_c bits of the codeword are generated at last N_c variable nodes. Let ${Q_j}^{(1)}$ represent Q_j after layer 1 is processed,

3. *Sub-iteration 2:*

   (a) After direct registration of variable node outputs, ${Q_j}^{(1)}$ values corresponding to first N_c columns are at the last N_c register outputs, $F_{3N_c..(4N_c-1)}$. The fixed shift based on inverse mapping is done again, so $S_{2N_c..(3N_c-1)} = F_{3N_c..(4N_c-1)} = {Q_{0..(N_c-1)}}^{(1)}$.

   (b) $S_{2N_c..(3N_c-1)}$ are transmitted to check nodes based on submatrix B. However, in this sub-iteration these signals contain values corresponding to first column group. Passing ${Q_{0..(N_c-1)}}^{(1)}$ to check nodes through B is the proper configuration in layer 2. Examining other column groups shows that layer 2 is processed properly in current sub-iteration. ${Q_{0..(N_c-1)}}^{(2)}$ are generated in $G_{2N_c..(3N_c-1)}$, and are sent to variable nodes $2N_c$ to $3N_c-1$,

4. *Sub-iterations 3:*

   In third cycle of the decoding process, ${Q_{0..(N_c-1)}}^{(2)}$ values are transmitted to $S_{N_c..(2N_c-1)}$ $= F_{2N_c..(3N_c-1)}$ after passing the shift network. These signals contain posteriori probability

ratios for first N_c columns updated after first two layers are processed. They are then connected to check nodes in the architecture based on connection matrix C. The way Q_j values corresponding to other columns of the matrix match the connection matrices on the routing network shows that layer 3 is processed in this sub-iteration. Before the current cycle ends, ${Q_{0..(N_c-1)}}^{(3)}$ are ready to get registered at the output side of second N_c variable nodes.

5. *Sub-iterations 4:*

   Layer 4 is processed properly in this cycle. ${Q_{0..(N_c-1)}}^{(3)}$ values are shifted, and they are passed to check nodes through a connection matrix based on D. At the end of fourth sub-iteration, all four layers are processed and one decoding iteration is complete. Additionally, at the output side of check nodes, ${Q_{0..(N_c-1)}}^{(4)}$ go through first N_c variable nodes in the architecture. Therefore, these variable nodes represent the first N_c columns of the matrix. The generated bits from all the variable nodes in the architecture are in the correct order in current cycle. By registering the outputs when the cycle ends, no shifting is needed in the output side.

In general, the decoding process in the proposed scheme starts for each iteration with layer L_1, for which M is valid from L_{max} to L_1. Afterward, layer L_2 gets processed for which M is valid from L_1 to L_2. The last layer that gets processed in a decoding iteration is L_{max}, for which the output bits are registered. The only disadvantage of the proposed architecture comparing to the original layered-decoding partial-parallel architecture is that the syndrome check can not be done after processing each layer. Without adding extra logic to shift back the output bits to their proper place, syndrome check can only be done at the end of one complete iteration.

Although the shifting of Q_j values is done in hardware, for R_ij values shifting is not necessary, since registering is done inside every check node separately. Overall, by using a constant routing network, the movement between layers is managed without using any gates. The only gates that should be used in the routing network are for the column groups which have irregularity in the matrix structure. Also It should be mentioned that the complexity of the routing network is not changed dramatically. The shifting network and the connection network based on matrix are in series, so they can be assumed as one overall shifting network, comparable to any other V-to-C routing network in conventional partial-parallel decoders, but with no gates.

Although the physical routing network has no change over different sub-iterations, the shift in the variable node outputs produces different decoding process, matching to different layers.

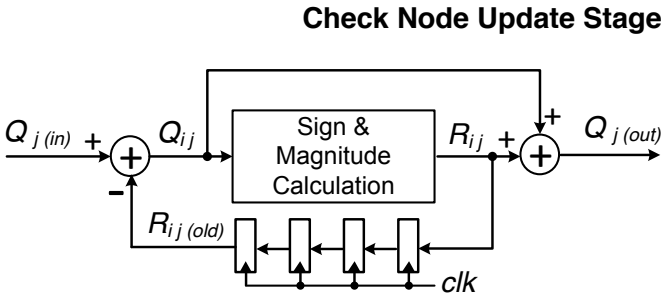

Figure 3.6: The update stages inside check nodes: Q_j values coming from variable nodes in the architecture get updated in the check nodes.

3.3 Architecture Overview

3.3.1 Check Nodes

As mentioned before, the number of implemented check nodes in the proposed architecture is the number of rows in one layer, M_l. The check nodes in the architecture are responsible for processing update equations for Normalized Min-Sum algorithm. Check node update in Normalized Min-Sum algorithm consists of three stages which are shown in Figure 3.6.

In first update stage, the check node to variable node messages from last iteration are subtracted from *a posteriori probability ratio* for current iteration (Eqn. 2.9) to produce variable node to check node messages. Therefore, the check node messages from last iteration should be stored. Additionally, Check nodes in the proposed architecture are utilized to process update stages for all layers in the parity check matrix. For instance, CN 0 processes check node update stage for first row in each layer of parity check matrix. In this architecture, a queue of Y flops is used to store check node messages from last iterations. The check node to variable node messages are registered in first flop of this queue when process of current layer is finished. After Y clocks, when the process of one iteration is complete and check nodes are processing same layer in parity check matrix again, these messages are at the output of the last flop in the queue and ready to get subtracted from variable node outputs.

Second update stage in check node for Layered Normalized Min-Sum is the main check node update stage in Min-Sum algorithm (Eqn. 2.11). Using variable node to check node messages produced in the first stage, check node to variable node messages are generated by sign and magnitude calculation. The sign calculation is done by multiplying signs of variable node messages except the one the check node message is calculated for. In this architecture, sign calculation is implemented

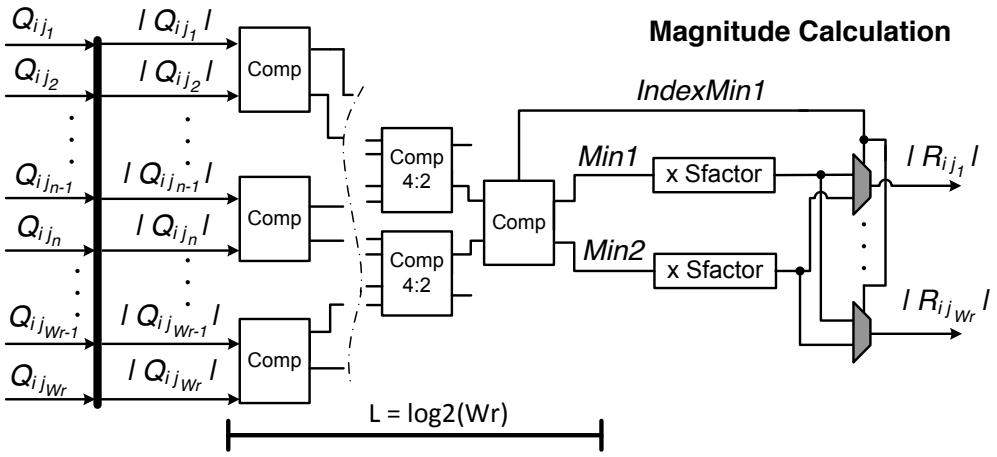

Figure 3.7: Magnitude calculation in check nodes

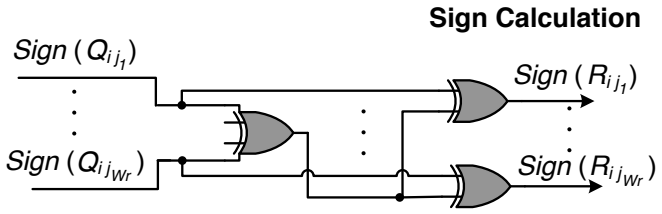

Figure 3.8: Sign calculation in check nodes

by doing XOR over all sign bits of variable node messages, as shown in Figure 3.7. Eventually, for each check node message, the sign of corresponding variable node message is XORed with the overall sign.

The magnitude calculation is processed by finding the first and second minimum over the magnitude of all variable node messages connected to the check node. The first minimum is used for the magnitude part of all check node messages except the one corresponding to the variable node with that minimum as its message. For that variable node, the second minimum is used. Finding the first and second minimum of magnitude of all variable node messages is done by first changing their format to sign magnitude, and then comparing them two by two in a number of levels. The magnitude calculation is shown in Figure 3.8.

In last update stage, the produced check node messages are substituted with the messages from last iteration in the *a posteriori probability ratio* for variable nodes. The produced values are

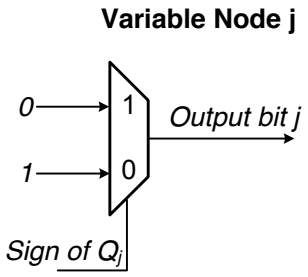

Figure 3.9: The simple logic in each variable node to generate decoded bits

the outputs to variable nodes in the architecture. They are registered and given to variable nodes in the next cycle. The overall check node update process is shown in Figure 3.6.

3.3.2 Variable Nodes

The number of implemented variable nodes in permutational decoder architecture is equal to the number of columns in the parity check matrix, N. In conventional Min-Sum LDPC decoding, variable nodes are responsible for accumulating incoming check node messages, generating decoded bit corresponding to the variable node based on the sign of the result, and subtracting each check node message from the result to produce variable node messages. However, in the architecture in this work, accumulating check node messages and subtracting each of them later from the result are summarized in check node update process. The input to the variable nodes are Q_j values instead of Q_{ij}. Therefore the variable nodes here are only responsible to generate output bits. Figure 3.9 shows the simple logic in each variable node.

3.3.3 Shift Network

The shift network in the architecture is a constant wiring network based on the inverse of valid mapping for the parity check matrix. Since this shifting network is in series with the hard-wired variable node to check node routing, these two can be considered as one overall routing network comparable to any conventional network between variable nodes and check nodes, but with almost no gates. Additionally, the shift network is only implemented in the datapath of variable node to check node messages. The check node to variable node messages only go through the routing network based on one layer in parity check matrix.

Usually in communication standards, multiple LDPC codes with different code rates and

parity check matrices are adopted to support different bit-error rate ranges. For instance, 802.11ad draft has mentioned three LDPC codes with code rates 1/2, 3/4 and 7/8. Interestingly, all of these three LDPC codes have permutational structure, and their parity check matrices have the same valid mapping. This gives the permutational LDPC decoding architecture the potential to effectively support multiple code rates.

3.3.4 Hard-wired Routing Network

The hard-wired routing network includes a wiring network passing shifted variable node outputs to check nodes, and another wiring network passing check node outputs to variable nodes. These two wiring networks are implemented based on the layer with highest row degree in parity check matrix. They are equivalent to the routing network of a full-parallel LDPC decoder based on a matrix consisting of only the layer with the highest row degree in original parity check matrix. The big advantage in this architecture comes from the fact that these two routing networks need almost no gate to support decoding process for other layers. The only gates needed in the routing network of this decoder is to support irregularities in the parity check matrix. It was mentioned in the definition of valid mapping that 1) there is no constraint on the submatrix it maps to an all-zero submatrix, and 2) if it maps a non-zero submatrix to an all-zero one, it is still valid. These two conditions are essential in including irregular LDPC codes adopted in 802.11ad and 802.15.3c in permutational LDPC codes. The architecture is implemented based on the layer with the highest row degree. To process layers with an all-zero submatrix, the Q_j values corresponding to columns of that submatrix should neither affect the process of check nodes nor get updated based on check node outputs. In other words, those Q_j values should be disconnected from check nodes in that sub-iteration. This is done by using a multiplexer between Q_j and its positive maximum possible at the input side of check nodes, and another multiplexer between Q_j from variable node and updated Q_j from check node at the output side. These multiplexers are only used for columns in submatrices that get all-zero once in layers of the matrix.

3.3.5 Output Bit Registers

Due to circulation of Q_j values in the architecture during different sub-iterations, unless a shifting mechanism is implemented for variable nodes, output bits are not in correct order in all sub-iterations except the last one. As the result, registering of variable node outputs is done only at the last sub-iteration (the last cycle of one iteration). The only degradation in the performance

for this architecture compared to conventional layered LDPC decoders is that syndrome check can be done only after one iteration is complete.

Chapter 4

Hardware Implementation for IEEE 802.11ad

In this chapter, the permutational decoder for the (672,588) LDPC code proposed in IEEE 802.11ad standard is implemented in 65 nm CMOS technology, and the results are compared to a conventional partial-parallel decoder for the same code and other similar partial-parallel decoders. The gate count for logic on the routing network of the proposed decoder shows over 96% reduction, which results in improvements in area, power, and throughput comparing to similar decoders. The same architecture can be implemented for (672,504) and (672,336) LDPC codes proposed for IEEE 802.11ad or LDPC codes adopted in IEEE 802.15.3c standard.

4.1 LDPC Codes in IEEE 802.11ad

The parity check matrices of (672,588), (672,504) and (672,336) LDPC codes proposed for 802.11ad are shown in Figure 4.1. Each numerical value in the matrices represent a permutation of 21×21 identity matrix, and submatrices with no value are all-zero ones. A mapping is valid between two submatrices if it maps submatrices with same numerical value, or maps from or to an all-zero submatrix. Since the row degrees and column degrees are different between different rows and columns of each parity check matrix, all three codes are irregular. Table 4.1 shows the distribution of row degrees between layers of three codes. Row degrees of rows inside one layer of these parity check matrices are the same. However, different layers have different row degrees, and their values are different between three codes. On the other hand, column degrees have the same

(672,336), Code rate: 1/2

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	-	-	-	5	-	18	-	-	-	-	3	-	10	-	-	-	-	-	-	5	-	-	-	-	-	-	-	-	-	-	-
2	0	-	-	-	-	-	16	-	-	-	-	6	-	-	-	0	-	7	-	-	-	-	-	-	-	-	-	-	-	-	-
3	-	-	6	-	7	-	-	-	-	2	-	-	-	-	9	-	20	-	-	-	-	-	-	-	-	-	-	-	-	-	-
4	-	18	-	-	-	-	-	0	10	-	-	-	-	16	-	-	-	-	9	-	-	-	-	-	-	-	-	-	-	-	-
5	5	-	-	-	-	-	18	-	-	-	-	3	-	10	-	-	5	-	-	-	-	-	-	4	5	-	-	-	-	-	-
6	-	0	-	-	-	-	-	16	6	-	-	-	0	-	-	-	-	-	7	-	4	-	-	-	-	10	-	-	-	-	-
7	-	-	-	6	-	7	-	-	-	-	2	-	-	-	-	9	-	20	-	-	-	4	-	-	-	-	19	-	-	-	-
8	-	-	18	-	0	-	-	-	-	10	-	-	-	-	16	-	-	-	-	9	-	-	12	-	-	4	-	-	-	-	-
9	-	5	-	-	-	-	-	18	3	-	-	-	-	-	10	-	-	5	-	-	4	-	-	-	-	-	-	-	-	-	-
10	-	-	0	-	16	-	-	-	-	6	-	-	-	0	-	-	-	-	-	7	-	4	-	-	-	-	-	-	-	-	-
11	6	-	-	-	-	-	7	-	-	-	-	2	9	-	-	-	-	-	20	-	-	-	4	-	-	-	-	-	-	-	-
12	-	-	-	18	-	0	-	-	-	-	10	-	-	-	-	16	9	-	-	-	-	-	-	12	-	-	-	-	-	-	-
13	-	-	5	-	18	-	-	-	-	3	-	-	-	-	-	10	-	-	5	-	-	4	-	-	-	-	5	-	7	-	-
14	-	-	-	0	-	16	-	-	-	-	6	-	-	-	0	-	7	-	-	-	-	-	4	-	10	-	-	-	-	-	19	-
15	-	6	-	-	-	-	-	7	2	-	-	-	-	9	-	-	-	-	-	20	-	-	-	4	-	19	-	-	-	-	-	10
16	18	-	-	-	-	-	0	-	-	-	-	10	16	-	-	-	-	9	-	-	12	-	-	-	-	-	-	4	-	17	-	-

(672,504), Code rate: 3/4

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	0	-	-	5	-	18	16	-	-	-	3	6	10	-	-	0	-	7	-	5	-	-	4	4	-	10	-	5	-	-	-	-
2	-	18	6	-	7	-	-	0	10	2	-	-	-	16	9	-	20	-	9	-	4	12	-	-	4	-	19	-	-	-	-	-
3	5	0	-	-	-	-	18	16	6	-	-	3	0	10	-	-	5	-	7	-	4	-	-	4	5	-	10	-	19	-	-	-
4	-	-	18	6	0	7	-	-	-	10	2	-	-	-	16	9	-	20	-	9	-	4	12	-	-	4	-	19	-	10	-	-
5	-	5	0	-	16	-	-	18	3	6	-	-	-	0	10	-	-	5	-	7	4	4	-	-	-	5	-	-	-	-	-	-
6	6	-	-	18	-	0	7	-	-	-	10	2	9	-	-	16	9	-	20	-	-	-	4	12	19	-	-	-	-	-	-	-
7	-	-	5	0	18	16	-	-	-	3	6	-	-	-	0	10	7	-	5	-	-	4	4	-	10	-	5	-	7	-	19	-
8	18	6	-	-	-	-	0	7	2	-	-	10	16	9	-	-	-	9	-	20	12	-	-	4	-	19	-	4	-	17	-	10

(672,588), Code rate: 7/8

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	0	18	6	5	7	18	16	0	10	2	3	6	10	16	9	0	20	7	9	5	4	12	4	4	4	10	19	5	10	-	-	-
2	5	0	18	6	0	7	18	16	6	10	2	3	0	10	16	9	5	20	7	9	4	4	12	4	5	4	10	19	19	10	-	-
3	6	5	0	18	16	0	7	18	3	6	10	2	9	0	10	16	9	5	20	7	4	4	4	12	19	5	4	10	17	19	10	-
4	18	6	5	0	18	16	0	7	2	3	6	10	16	9	0	10	7	9	5	20	12	4	4	4	10	19	5	4	7	17	19	10

Figure 4.1: The parity check matrices of LDPC codes proposed in IEEE 802.11ad

range across different code rates. The highest row degree (W_r) for all three matrices is 32, and the highest column degree (W_c) is 4.

Figures 4.2, 4.3 and 4.4 show how a valid mapping exits over the layers of parity check matrices for codes proposed in 802.11ad. This valid mapping and the incremental permutation of layers show that all these three matrices are *permutational*. Therefore, based on the valid mapping and the layer with highest row degree, a permutational LDPC decoding architecture can be implemented for all three code rates. Figures 4.2, 4.3 and 4.4 also show that the valid mapping for all three codes in 802.11ad are the same. Additionally, the layer with highest row degree in all three codes has the same location in parity check matrices, and the connection matrix based on this layer

	(672,588)	(672,504)	(672,336)
Layer 1	29	14	5
Layer 2	30	15	7
Layer 3	31	13	6
Layer 4	32	16	8

Table 4.1: Row degrees (W_r) for different layers of 802.11ad LDPC codes

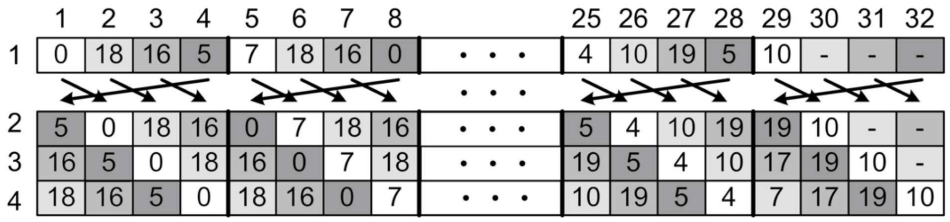

Figure 4.2: The valid mapping shown and the permutation (1,2,3,4) demonstrate that the parity check matrix of IEEE 802.11ad (672,588) LDPC code is *permutational*

can be easily reconfigured to support all three codes by separating a check node in higher rate into multiple check nodes in lower rates. This implies that the architecture in this work has the potential to support multiple code rates and reconfigurability in the hardware efficiently.

In this work,the permutational LDPC decoder is implemented for (672,588) code, which has the highest code rate among three codes, 7/8. The parity check matrix of the (672,588) code has M= 84 rows, N= 672 columns. These rows and columns can be divided into Y= 4 layers and U= 32 column groups. As the result, every submatrix has 21×21 dimension. The mapping shown in Figure 4.2 with the sequence of layers 1,2,3,4 implies that the parity check matrix is permutational.

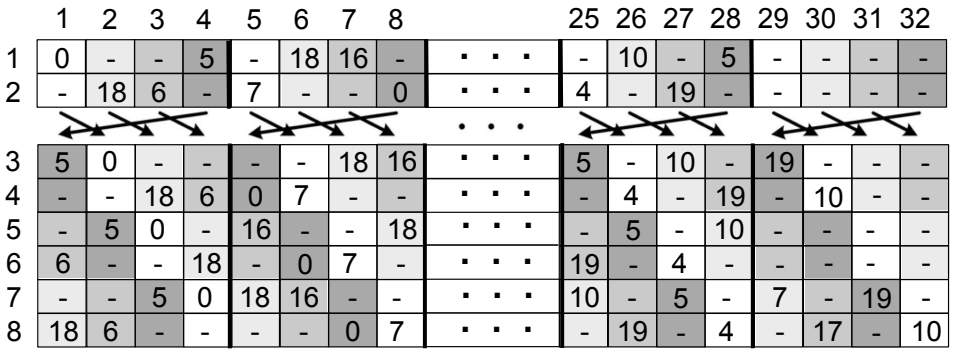

Figure 4.3: The parity check matrix of (672,504) LDPC code in IEEE 802.11ad is *permutational*.

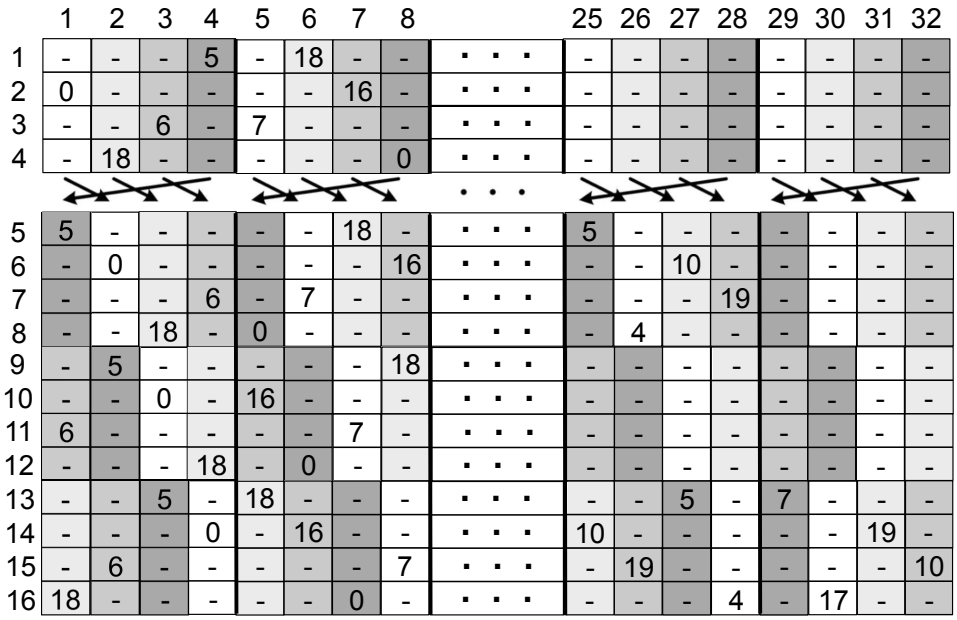

Figure 4.4: The parity check matrix of (672,336) LDPC code in IEEE 802.11ad is *permutational*.

4.2 Design Parameters

As mentioned earlier, the number of implemented check nodes in permutational decoder is equal to the number of check nodes in one layer, and all of the variable nodes are implemented in hardware. Therefore, the architecture for (672,588) code in 802.11ad has 21 check nodes and 672 variable nodes in hardware. The number of implemented variable nodes stay the same for all the rates in 802.11ad, however the number of check nodes get 2× for (672,504) and 4× for (672,336) codes.

The V-to-C and C-to-V routing networks in the architecture are based on the connection matrix of layer with highest row degree, which is layer 4 for all three code rates. Consequently, each check node is connected to 32 variable nodes. The shifting network coupled with V-to-C routing is implemented based on the inverse of valid mapping on the parity check matrix. The valid mapping is shown in Figure 4.2. The inverse of this mapping is implemented by fixed wiring in the hardware, and is responsible for shifting data in multiple cycles of one iteration to generate decoding process corresponding to different layers. There are no other gates required to adjust the decoding process to different layers in regular codes. However, since the parity check matrix of the code is irregular, a couple of muxes are required to bypass all-zero submatrices in processing of layers with less row weights. These muxes are only applied to output and input ports of variable nodes corresponding to columns of all-zero submatrices in matrix (column groups 29, 30, 31 and 32 in (672,588) code).

At the output side of these variable nodes, the mux chooses between the real output of variable node and maximum positive value possible for that signal. At the input side, another mux chooses between the output from check nodes and the output of variable node in current cycle. These are the only gates required on the routing network of decoder.

The variable node messages in decoder are initialized by channel information. After processing of check nodes and variable nodes corresponding to four layers of parity check matrix is finished in four cycles, one iteration of decoding is complete. The decoded bits coming from variable nodes are in correct order after 4 cycles, and they are registered at the end of one iteration. Early termination by syndrome check can be applied after each iteration.

The remaining design parameters to be discussed are correction factor in Min-Sum algorithm ($Sfactor_{MS}$), maximum number of iterations for decoding (I_{max}), and decoding datapath quantization. These parameters are determined based on MATLAB simulation of the decoding algorithm with different combinations to get to an optimum trade-off between bit-error rate performance and architecture requirements. For (672,588) LDPC code proposed in IEEE 802.11ad, $Sfactor_{MS} = 0.75$, $I_{max} = 5$, and data-path quantization of 6 bits (4 bits before and 2 bits after the radix point) yield to minimum degradation compared to floating point algorithm.

Figure 4.5 compares different data-path quantizations for proposed architecture assuming $Sfactor_{MS} = 0.75$ and $I_{max} = 5$. As shown in the figure, 6-bit quantization of messages results in less than 0.1 dB degradation comparing to floating point algorithm until 10^{-5} bit error probability. On the other hand, 5-bit quantization shows significant degradation in BER performance, to the extent that it is comparable to no decoding at all.

The BER performance diagrams for comparing different $Sfactor_{MS}$ values is shown in Figure 4.6. Like previous simulations for data-path quantization, other parameters such as quantization and I_{max} are fixed at their optimum values, and $Sfactor_{MS}$ is changed to find the value yielding to best BER performance. The BER diagram shows that $Sfactor_{MS} = 0.75$ gives over 0.25 dB improvement in bit error probability compared to other values such as 0.5 or 1. In check node update equations, $Sfactor_{MS}$ multiplication is applied to every check node to variable node message. However, in hardware implementation in check nodes, $Sfactor_{MS} = 0.75$ is only applied to first and second minimum over absolute values of variable to check node messages. Therefore the number of multiplications is decreased from row degree to only two for each check node. $0.75\times$ multiplication is implemented in hardware as $0.5 \times +0.25\times =<< 1+ << 2$.

Finally, Figure 4.7 compares different I_{max} values for the decoder. I_{max} is the threshold

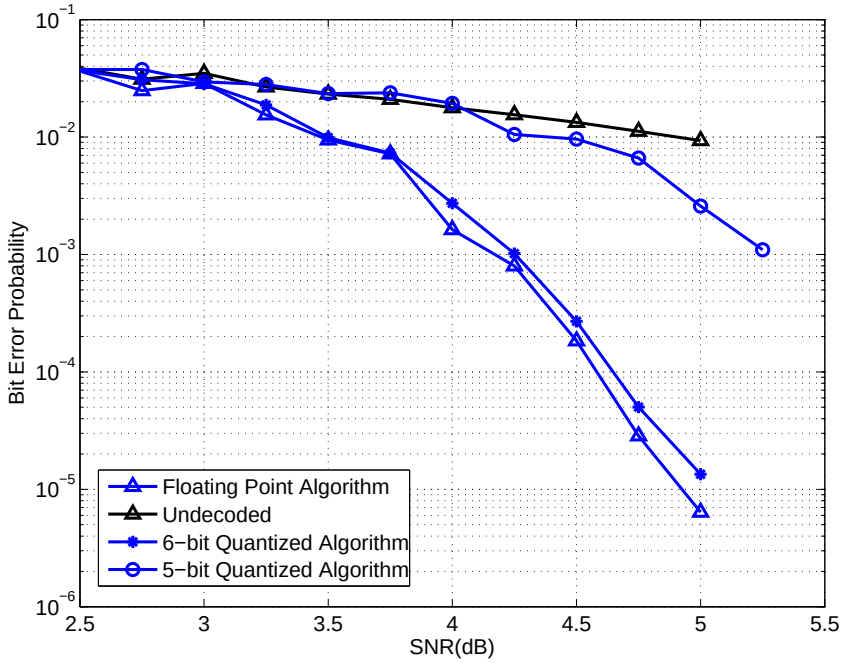

Figure 4.5: BER performance of decoder with different quantization's when $Sfactor_{MS} = 0.75$ and $I_{max} = 5$

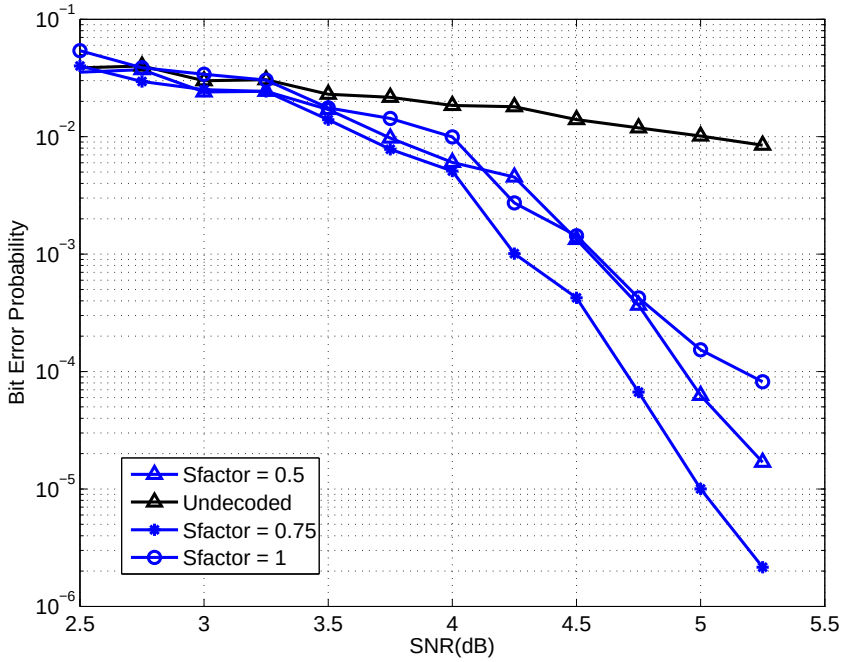

Figure 4.6: BER performance of decoder with different $Sfactor_{MS}$ values when $I_{max} = 5$ and 6-bit quantization

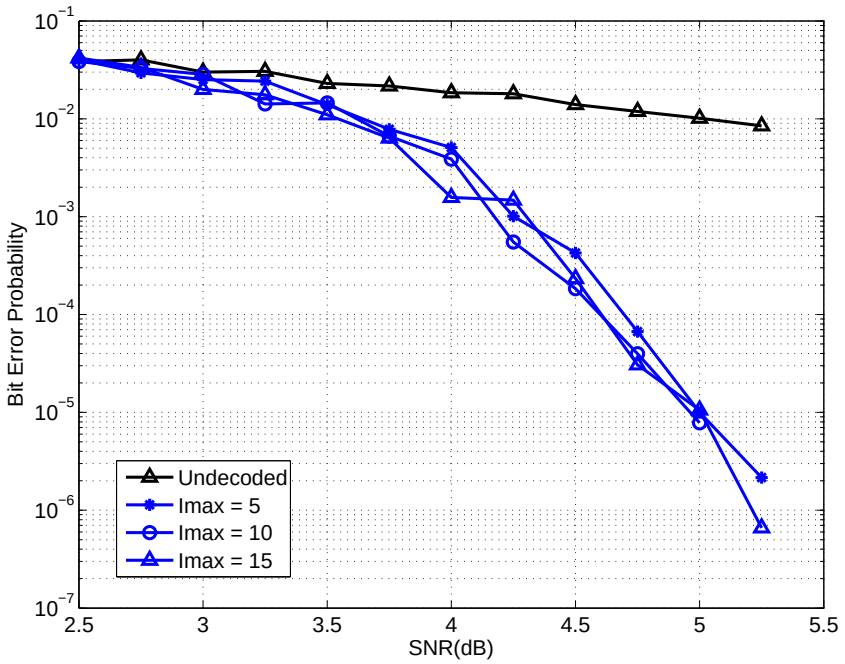

Figure 4.7: BER performance of decoder with different I_{max} values when $Sfactor_{MS} = 0.75$ and 6-bit quantization

for the maximum number of iterations used for decoding input message. If the number of iterations required for decoding a message from channel gets more than this threshold, the decoding process is finished without resulting in a codeword. In this scenario, usually another transmission of the message is requested by the receiver. Increasing this threshold always improves BER performance. Yet, Figure 4.7 shows that there is no significant improvement from $I_{max} = 5$ to $I_{max} = 15$ in performance, although $I_{max} = 15$ has three time worse throughput and power. $I_{max} = 5$ is chosen as the maximum number of iterations allowed for decoding input messages.

4.3 Results

For hardware implementation of the decoder, NC Verilog by Cadence is used for hardware simulations and verification of functional behavior. Synthesis is done by Synopsys Design Compiler XG-T. Finally layout is simulated in 65 nm CMOS technology using Cadence Encounter 09.12-s159_1.

Figure 4.8 shows the layout of implemented decoder. Table 4.2 shows the performance characteristics of the decoder, and compares them with a conventional partial-parallel decoder with the same number of check nodes and variable nodes. It should be noted that the only difference

between the proposed decoder and conventional partial-parallel decoders is in the routing network. Therefore, assuming same design parameters such as correction factor, quantization, and maximum number of permitted iterations, there is no degradation in the BER performance of the decoder.

The code length of the decoders represented in Table 4.2 are in the same range. Consequently, the performance characteristics of these designs can effectively get compared. As the input quantization of a decoder decreases, its complexity and area decrease with same respect, and its BER performance shows degradation. The input quantization of the proposed decoder is the highest among decoders in the table, meaning the improvements in area and throughput are not due to compensating the BER performance for hardware efficiency. However, the gate count and core area in the permutational decoder shows substantial improvement (over 5.1×) comparing to decoders from literature represented in the table. The proposed decoder shows around 10% decrease in gate count comparing to a conventional partial-parallel decoder with same characteristics. This improvement in number of gates is entirely due to the gate count reduction on the routing network of the decoder. The simplification of the routing network containing global connection wires between check nodes and variable nodes improves the hardware utilization of the decoder, which results in 1.26× reduction in core area.

The critical path of signals passing through the decoder is almost the same in both proposed decoder and conventional partial-parallel decoder. This path starts with the signal corresponding to sub-iteration count in the architecture. In permutational decoder, this signal is entered as the select to few 2-to-1 multiplexers on the V-to-C routing network. These multiplexers are responsible for cutting check nodes from certain variable nodes in the sub-iterations that those variable nodes correspond to an all-zero submatrix. In conventional decoder this critical path starts with the same signal acting as the select to 4-to-1 multiplexers on the V-to-C routing network. The critical path for both decoders then passes through the check nodes. After check nodes, the path passes through 2-to-1 multiplexers in proposed decoder and 4-to-1 multiplexers in conventional one in C-to-V routing. Finally the critical path ends in the main flops of the datapath before the variable nodes. The reduction in core area, complexity of the routing network and global wiring, and logic in critical path of signals passing through decoder results in 1.3× increase in the throughput of the permutational decoder. The throughput numbers at Table 4.2 are calculated at maximum number of allowed iterations for the decoder, assuming that syndrome check and early termination are not applied. Applying early termination by syndrome check results in further improvements in throughput.

	ASSCC'10 [18]	ISCAS'11 [19]	CICC'07 [21]	Conventional Partial-Parallel Arch.	Proposed Arch. [31]
CMOS fabrication process	65 nm	65 nm	0.13 μm	65 nm	65 nm
Code Length	672	672	660	672	672
Supported Code rates	1/2, 5/8 3/4, 7/8	1/2, 5/8 3/4, 13/16	0.73	7/8	7/8
Input Quantization (bits)	6	5	4	6	6
Gate count(k)	647	-	690	138	125
Core area (mm^2)	1.562	1.3	7.3	0.891	**0.718**
Maximum clock frequency (MHz)	197	150	300	180.2	**235**
Maximum Iteration Count (I_{max})	5	15	15	5	5
Throughput @ I_{max} (Gbps)	5.79	3.08	2.44	6.05	**7.9**
Throughput/area (Gbps/mm^2)	3.71	2.37	0.33	6.79	**11**
Power @ I_{max} (mW)	361	84	1383	75.24	**83.84**

Table 4.2: Comparison of results of Permutational LDPC Decoder with similar decoders

Functional verification for the proposed decoder hardware is done by simulations in MATLAB environment. A model for the decoder is implemented in MATLAB code as close as possible to the architecture. Then one codeword for the (672,588) LDPC code is generated based on the parity check matrix. This codeword is passed through an additive white Gaussian noise (AWGN) channel using binary phase-shift keying (BPSK) modulation. The result is the message from channel at the receiver end, and the log-likelihood ratio of channel information (λ_j) is calculated. λ_j values are dumped inside a text file, and are applied to the hardware module by verilog testbench. Then the testbench prints decoded bits at the end of each iteration in an output file. This log file is compared bit by bit with the outputs of MATLAB simulation to verify MATLAB model is simulating the hardware behavior. Finally, the BER performance of MATLAB model is examined and compared with decoders in literature by generating various inputs in different SNR ranges.

As mentioned before, the savings in the proposed method are the result of reducing the number of gates used for adjusting interconnection network to different layers of parity check matrix. In the conventional partial-parallel architecture, for each input pin of every check node, a 4-to-1 mux is used to adjust the routing network to four layers of the matrix in different cycles. Therefore, the overall number of gates used in the V-to-C routing network is *Number of check nodes* $\times$ *Row weight* 4-to-1 muxes. On the input side of variable nodes (C-to-V routing network), the same number of muxes are required to change the path of check node messages in four cycles of an iteration. This results in overall $2 \times 21 \times 32 (= 1344)$ 4:1 muxes on the routing network of a conventional decoder

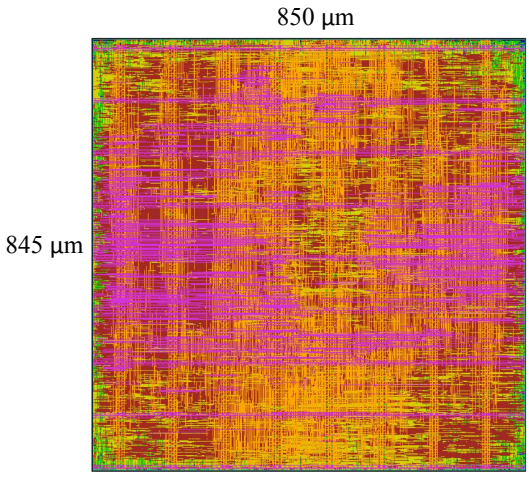

Figure 4.8: Layout of proposed decoder for (672,588) LDPC code

for (672,588) LDPC code discussed here.

On the other hand, in the proposed permutational architecture, the only gates required on the routing network of the decoder are for the columns corresponding to irregularities in the matrix structure (last 3 column groups in this code). In other words, the number of gates on the V-to-C or C-to-V routing networks is *Number of columns once in an all − zero submatrix* 2-to-1 muxes. This means the proposed decoder has only $2 \times 3 \times 21 (= 126)$ 2-to-1 muxes on its interconnection network.

Over 96% reduction in gate count on the routing network in permutational decoder for (672,588) code results in 1.26 times improvement in hardware area. Additionally, since the 4-to-1 muxes in partial-parallel architecture are on the critical path of signals passing through check nodes, reducing them to at most 2-to-1 muxes yields further improvement in throughput. The proposed decoder is 1.3 times faster.

Chapter 5

Conclusion

5.1 Advances

A new routing network architecture based on characteristics of parity check matrix of LDPC codes proposed in the IEEE 802.15.3c and 802.11ad is presented which results in almost complete elimination of gates on the routing network of the decoder, and provides improvements in area, throughput and power, with no decline in BER performance. The class of matrices for which the architecture could be used is defined, and the general architecture is presented. Implementing the decoder in 65 nm CMOS technology including place & route for (672,588) LDPC code in the IEEE 802.11ad shows over 96% reduction in number of gates on the routing network, which results in 1.26× improvement in area and 1.3× improvement in throughput. The definition of permutational parity check matrices in this work can also be used as a guideline in generating LDPC codes for emerging standards resulting in more efficient hardware implementations.

5.2 Future Work

As mentioned earlier, communication standards usually adopt multiple codes with different code rates to support various SNR ranges and channel conditions. For instance, 802.15.3c has adopted LDPC codes with rates 1/2, 3/4, and 7/8. A big advantage for a decoder architecture of a communication standard is the ability to effectively support these multiple code rates. As shown in previous chapters, the architecture proposed in this work is implemented based on the inverse of the valid mapping on layers of the parity check matrix. However, for LDPC codes proposed in 802.11ad and 802.15.3c, the valid mapping is the same. As the result, the architecture in this work

has the potential to support all codes in 802.11ad and 802.15.3c with minimal changes. One future work direction is to implement a reconfigurable decoder for all codes in one standard based on this architecture. By applying a number of gates between check nodes to merge or split them in different rates, the changes between routing networks of codes in one of discussed standards can be simulated in the hardware efficiently.

Another interesting future work direction is to fully define LDPC codes with characteristics discussed in this work, and analyze mathematical aspects of their performance. As architecture-aware LDPC codes were able to define a class of codes with efficient hardware implementations, the permutational LDPC codes can define another class with highly simplified routing networks and same BER performance as other types. Therefore, by following certain guidelines, LDPC codes proposed for future communication standards can have higher efficiency in their hardware implementations.

Bibliography

[1] R. G. Gallager. Low-density parity check codes. *IRE Transaction Info. Theory*, IT-8:21–28, Jan. 1962.

[2] D.J.C. MacKay and R.M. Neal. Near shannon limit performance of low density parity check codes. *Electronics Letters*, 32(18), Aug. 1996.

[3] T.T.S.I. digital video broadcasting (DVB) second generation framing structure for broadband satellite applications. http://www.dvb.org.

[4] IEEE 802.16e. air interface for fixed and mobile broadband wireless access systems. ieee p802.16e/d12 draft, Oct. 2005.

[5] IEEE P802.3an, 10GBASE-T task force. http://www.ieee802.org/3/an.

[6] IEEE standard for information technology - specific requirements. part 15.3. *IEEE Std 802.15.3c-2009 (Amendment to IEEE Std 802.15.3-2003)*, pages c1 –187, 12 2009.

[7] 802.11ad full proposal submission. http://mentor.ieee.org/802.11/dcn.

[8] F. Clermidy, C. Bernard, R. Lemaire, J. Martin, I. Miro-Panades, Y. Thonnart, P. Vivet, and N. Wehn. A 477 mw NoC-based digital baseband for MIMO 4G SDR. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, pages 278–279, Feb. 2010.

[9] T. Limberg, M. Winter, M. Bimberg, R. Klemm, E. Matus, M.B.S. Tavares, G. Fettweis, H. Ahlendorf, and P. Robelly. A fully programmable 40 GOPS SDR single chip baseband for LTE/WiMAX terminals. In *Solid-State Circuits Conference, 2008. ESSCIRC 2008. 34th European*, pages 466–469, Sep. 2008.

[10] T. Mohsenin and B. Baas. Trends and challenges in LDPC hardware decoders. In *Signals, Systems and Computers, 2009 Conference Record of the Forty-Third Asilomar Conference on*, pages 1273–1277, Nov. 2009.

[11] A. Blanksby and C. J. Howland. A 690-mW 1-Gb/s 1024-b, rate 1/2 low-density parity-check code decoder. *IEEE Journal of Solid-State Circuits*, 37(3):404–412, Mar. 2002.

[12] M. Karkooti, P. Radosavljevic, and J.R. Cavallaro. Configurable, high throughput, irregular LDPC decoder architecture: Tradeoff analysis and implementation. In *ASAP*, pages 360 –367, Sep. 2006.

[13] Zhengya Zhang, V. Anantharam, M.J. Wainwright, and B. Nikolic. An efficient 10GBASE-T ethernet LDPC decoder design with low error floors. *Solid-State Circuits, IEEE Journal of*, 45(4):843–855, Apr. 2010.

[14] M. Mansour and N.R. Shanbhag. High-throughput LDPC decoders. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 11(6):976–996, Dec. 2003.

[15] P. Radosavljevic, A. de Baynast, and J.R. Cavallaro. Optimized message passing schedules for LDPC decoding. In *ACSSC*, pages 591 –595, Nov. 2005.

[16] T. Brack, F. Kienle, and N. Wehn. Disclosing the LDPC code decoder design space. In *DATE*, volume 1, page 6 pp., Mar. 2006.

[17] G. Malema and M. Liebelt. Interconnection network for structured low-density parity-check decoders. In *APCC*, pages 537 –540, Oct. 2005.

[18] Shiang-Yu Hung et al. A 5.7gbps row-based layered scheduling LDPC decoder for IEEE 802.15.3c applications. In *A-SSCC*, pages 1 –4, Nov. 2010.

[19] M. Weiner, B. Nikolic, and Z. Zhang. LDPC decoder architecture for high-data rate personal-area networks. In *ISCAS*, pages 1784 –1787, May. 2011.

[20] T. Mohsenin, D.N. Truong, and B.M. Baas. A low-complexity message-passing algorithm for reduced routing congestion in LDPC decoders. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 57(5):1048–1061, May. 2010.

[21] A. Darabiha, A.C. Carusone, and F.R. Kschischang. A 3.3-Gbps bit-serial block-interlaced Min-Sum LDPC decoder in 0.13-um CMOS. In *IEEE Custom Integrated Circuits Conference*, pages 459–462, Sep. 2007.

[22] A. Darabiha, A.C. Carusone, and F.R. Kschischang. Power reduction techniques for LDPC decoders. *IEEE Journal of Solid-State Circuits*, 43(8):1835–1845, Aug. 2008.

[23] M. Mansour and N.R. Shanbhag. A 640-Mb/s 2048-bit programmable LDPC decoder chip. *IEEE Journal of Solid-State Circuits*, 41(3):684–698, Mar. 2006.

[24] S. Lin and D. J. Castello Jr. *Error Control Coding*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2004.

[25] D.J.C. MacKay. *Information Theory Inference and Learning Algorithms*. Cambridge University Press, Cambridge, UK, 3rd edition, 2003.

[26] D. J. MacKay. Good error correcting codes based on very sparse matrices. *IEEE Transactions on Information Theory*, 45(2):399–431, Mar. 1999.

[27] M. Fossorier et al. Reduced complexity iterative decoding of low-density parity check codes based on belief propagation. *IEEE Transactions on Communications*, 47(5):673–680, May. 1999.

[28] R. M. Tanner. A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, 27(5):533–547, Sep. 1981.

[29] J. Chen, A. Dholakia, E. Eleftheriou, and M. Fossorier. Reduced-complexity decoding of LDPC codes. *IEEE Transactions on Communications*, 53(7):1288–1299, Aug. 2005.

[30] M. Fossorier. Quasi-cyclic low-density parity-check codes from circulant permutation matrices. *IEEE Transaction Information Theory*, 50(8):1788–1793, Aug. 2004.

[31] Houshmand Shirani-Mehr, Tinoosh Mohsenin, and Bevan Baas. A reduced routing network architecture for partial parallel LDPC decoders. In *IEEE Asilomar Conference on Signals, Systems and Computers (ACSSC)*, Nov. 2011.

Index