AMERICAN UNIVERSITY OF BEIRUT

Optimized ASIC Accelerator Chips for LDPC Decoders by Joint Algorithm, Architecture, and Circuit Co-Design

by
Saleh Usman

A dissertation
submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
to the Department of Electrical and Computer Engineering
of the Maroun Semaan Faculty of Engineering and Architecture
at the American University of Beirut

Beirut, Lebanon
April 2020

AMERICAN UNIVERSITY OF BEIRUT

Optimized ASIC Accelerator Chips for LDPC Decoders by Joint Algorithm, Architecture, and Circuit Co-Design

by

Saleh Usman

Approved by:

Dr. Ali Chehab, Professor Electrical and Computer Engineering	Committee Chairperson
Dr. Mohammad M. Mansour , Professor Electrical and Computer Engineering	Advisor
Dr. Louay Bazzi , Professor Electrical and Computer Engineering	Member of Committee
Dr. Emmanuel Boutillon, Professor Universite de Bretagne Sud, France	Member of Committee
Dr. Warren J. Gross, Professor McGill University, Canada	Member of Committee

Date of dissertation defense: April 22, 2020

AMERICAN UNIVERSITY OF BEIRUT

THESIS, DISSERTATION, PROJECT RELEASE FORM

Student Name: USMAN (Last) SALEH (First) ABDULQUDDOOS (Middle)

○ Master's Thesis ○ Master's Project ☑ Doctoral Dissertation

☑ I authorize the American University of Beirut to: (a) reproduce hard or electronic copies of my thesis, dissertation, or project; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes.

☐ I authorize the American University of Beirut, to: (a) reproduce hard or electronic copies of it; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes after: **One ___ year from the date of submission of my thesis, dissertation or project.**
Two ___ years from the date of submission of my thesis , dissertation or project.
Three ___ years from the date of submission of my thesis , dissertation or project.

Saleh Usman. May 14, 2020.

Signature Date

An Abstract of the Dissertation of

Saleh Usman for Doctor of Philosophy
Major: Electrical and Computer Engineering

Title: Optimized ASIC Accelerator Chips for LDPC Decoders by Joint Algorithm, Architecture, and Circuit Co-Design

Multi-Gb/s low-density-parity-check (LDPC) decoding is required for contemporary digital communication standards, like the 5G New Radio, IEEE 802.11 n/ac/ax (WiFi), and IEEE 802.11ad (WiGig), among others. Efficient application-specific integrated circuit (ASIC) implementation of the multi-Gb/s LDPC decoders requires fast-converging algorithms, high-throughput architectures, and optimized circuit choices. This work proposes and implements the algorithm, architecture, and circuit co-design approach for the development of optimized ASIC accelerator chips for LDPC decoders. Algorithmically, fast-converging schedules for decoding of LDPC codes, viz. interlaced column-row message-passing (ICRMP), and fast column-message passing (FCMP) schedules are proposed and investigated in this work. The proposed schedules converge double the speed of already existing fast-converging schedules, which are serial ones and converge two times the speed of initially proposed flooding schedule for LDPC decoding; the proposed schedules converge four times the speed of the flooding schedule with moderate increase in complexity, compared to the serial decoding schedules.

At the architectural level, multi-Gb/s architectures are proposed for the row-message-passing (RMP) version of existing serial decoding schedules by targeting IEEE 802.11n/ac/ax (WiFi) LDPC decoder, and for the proposed FCMP schedule by targeting IEEE 802.11ad (WiGig) LDPC decoder. Circuit-level techniques are then proposed, and an optimized VLSI design of the targeted IEEE 802.11n/ac/ax LDPC decoder is synthesized using the TSMC 40nm CMOS process. The synthesis results of the implementation outperform state-of-the-art in terms of throughput/area. The proposed FCMP-based decoder architecture, for IEEE 802.11ad LDPC codes, is also synthesized using the same technology

node, and the design achieves a throughput of 8.4 Gb/s while operating at a clock frequency of only 200 MHz, which enables the decoder to achieve the best-reported energy-efficiency of 8.6 pJ/bit, for IEEE 802.11ad LDPC decoders. Finally, an energy-efficient and high-throughput multi-core hardware architecture is presented and physically implemented as an ASIC chip. The implemented ASIC chip achieves a peak throughput of 15 Gb/s while operating at a clock frequency of 250MHz. The achieved throughput and the corresponding energy-efficiency are the best reported in the literature for an IEEE 802.11n/ac/ax LDPC decoder.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

LDPC codes have become the coding scheme of choice for high-performance error-correcting codes in many applications owing to their near-optimal performance [1]. Compared to turbo codes [2], LDPC codes provide an attractive additional feature of the inherent parallel structure at the decoder and therefore are better suited for present and future high throughput applications. Efficient application-specific integrated circuit (ASIC) implementation of multi-Gbps LDPC decoders is required, because LDPC codes are opted for contemporary digital communication standards, like the 5G New Radio [3, 4], IEEE 802.11n/ac/ax (WiFi) [5], and IEEE 802.11ad (WiGig) [6], among others.

For ASIC implementation, trivial methods to increase the throughput of LDPC decoders include explicit replication of LDPC decoder instances or by speeding up the clock speed of the LDPC decoder. Targeting these trivial approaches, however, have several drawbacks like, the first approach has a significant area penalty and requires high memory bandwidth, resulting in excessive power consumption, while the second approach has a significant power consumption penalty if timing closure at the desired clock speed is achievable. High-throughput and energy-efficient VLSI implementation of LDPC decoder while maintaining a small decoder footprint, therefore, requires the exploration of non-trivial approaches.

Algorithmically, fast-converging LDPC decoding schedules enable low-power and high throughput VLSI implementation by reducing the number of decoding iterations required for decoding convergence. The reduction in the number of iterations increases decoder throughput by decreasing the processing latency and minimizes power consumption by performing a lower number of decoding computations. The originally proposed LDPC decoding schedule was a flooding schedule, also known as standard message-passing (SMP) schedule, and consists of two distinct phases. In one phase, all *variable* nodes are updated, and then in the second, all the *check* nodes are updated. This technique was originally proposed by Gallager [7] and later used by Mackay [8] in his rediscovery of LDPC codes. Serial decoding schedules later emerged in the literature [9, 10], which converge

in half the number of decoding iterations compared with the flooding schedule. Serial decoding is further divided into two types, known as row message-passing (RMP) [9] and column message-passing (CMP) [10], based on whether the row or column order of the parity-check matrix is followed.

In the RMP schedule, decoding starts from the first check node, and computed messages are immediately passed back to connected variable nodes. This process is repeated one by one for all the check nodes. Immediate propagation of messages computed at check nodes back to variable nodes enables future check nodes to have access to the most updated message nodes, thus accelerating the convergence rate. A turbo-decoding schedule for LDPC decoding, based on RMP, was first proposed in [11, 9], and further analyzed in [12]. In CMP, decoding starts in reverse order compared to RMP, following the variable-nodes order, and messages at check nodes that are connected to a particular variable node are processed and immediately exchanged with that variable node [10, 13]. Serial decoders, in general, provide faster convergence without incurring any significant complexity overhead, i.e., the number of computations stays almost the same in one iteration of RMP and CMP as in SMP. Memory efficiency is an additional advantage of serial decoders; they require significantly lower memory than SMP [11, 9].

To further accelerate the convergence rate of serial decoding schedules, several works have emerged in the literature [14, 15, 16], which determine the best processing order of variable or check nodes for serial decoders. A maximum mutual-information-increase based schedule is presented in [14]. The schedule predicts the next message to be updated based on the mutual-information increase and is used as a metric to improve convergence speed. Similarly, in [15], a column-weight based fixed scheduling that processes variable nodes by following the high-to-low column-weight order for irregular LDPC codes is proposed. Informed fixed scheduling (IFS) is introduced in [16], which finds the appropriate order to ensure that the maximum number of updated messages is utilized within a single iteration. These schedules, however, attain marginal improvement in convergence speed over the original serial decoding schedules.

Informed dynamic scheduling (IDS) strategy is introduced in [17]. The scheme employs residual belief propagation (RBP) and updates only those nodes that correspond to the maximum residuals of the check-nodes messages. Although, RBP achieves faster convergence than existing decoding schedules, however, its decoding performance, after convergence, is even worse than the flooding or serial schedules [17]. Node-wise RBP (NWRBP) is also introduced in [17] to address the "greediness" of the RBP. Motivated by the IDS, several techniques with different message selecting and updating methods have been presented in the literature, such as the silent-variable-node-free (SVNF) scheduling [18], the dynamic-silent-variable-node-free (D-SVNF) [19], the tabu search-based dynamic scheduling (TSDS) [20], and the two-reliability-metrics based RBP (TRM-TVRBP) scheduling [21]. These fast-decoding schedules accelerate the convergence rate of LDPC decoding, but the added cost of computing the residuals and the required search

operations is far from trivial in VLSI implementations. This high computational cost explains the reason why no VLSI implementations of the RBP-based schedules, to the best of authors' knowledge, have been reported in the literature.

At the architectural level, pipelined decoders are used to support the throughput demands of multi-Gbps LDPC decoding. The pipelining, however, slows down the convergence speed of the single-frames based decoders and, consequently, the throughput is affected. Similarly, the number of quantization bits of the processing and memory elements of the decoder is directly related to the area-footprint, while inappropriate selection of the quantization bits affects the decoding performance of the decoder. Proper selection of the bits also allows the replacement of some of the saturated subtractors of the decoder, with unsaturated ones, and thus avoids corresponding costly operations of comparison and clipping involved in the saturation. Also, the selection of register-blocks versus memory-blocks to save variable and check-node messages affects the performance of the decoder, and the proper selection of these blocks is required to get the optimal performance of the decoder.

Furthermore, LDPC codes featured in IEEE standards are Quasi-Cyclic LDPC (QC-LDPC) codes [22]. VLSI implementation of the QC-LDPC decoders involves implementing barrel shifter networks, adders/subtractors, and check-node units (CNUs) along with the memory blocks. A low-power multi-Gbps QC-LDPC decoder implementation with a small footprint requires careful optimizations of all these modules at the circuit level as well. For example, three frame lengths and corresponding sub-block sizes z are defined for IEEE 802.11 ac/ax LDPC codes. The sub-block sizes of long frames are integer multiples of the smaller ones.

For high-throughput decoding, it is desirable to process one sub-block of size Z in a single clock cycle, which requires processing Z number of messages in one cycle, and the size of barrel shifters scales accordingly. For longer frame lengths, however, this increases the decoder area, and the design becomes under-utilized for smaller frame lengths. Processing of one layer per cycle is also used in multi-Gbps LDPC decoders, which implies that a CNU is able to process the layer that has the maximum node degree, in a single clock cycle. The maximum degree of the layer is usually close, but not exactly equal, to the number of columns of the Sparse-Parity-Check-Matrix (SPCM), and therefore, the CNUs and their connected circuitry is designed as per the number of columns of the SPCM to read and write data from memory locations directly. In [23], for example, CNUs with a number of inputs equal to 24 are implemented for IEEE 802.11n/ac LDPC codes, but the maximum degree of any layer is no more than 22 .

Based on the above discussion, it is evident that the optimum performance of the final VLSI implementation requires efficient designing in all the aforementioned domains, and a design-choice in one domain affects the overall performance of the final implementation. An optimized ASIC implementation of multi-Gbps LDPC decoders could be achieved by following a co-design approach that targets the algorithm, architecture, and circuits of the decoder in a joint way. This

work, therefore, proposes and implements the algorithm, architecture, and circuit co-design approach for the development of optimized ASIC accelerator chips for LDPC decoders.

1.1 Contributions

The purpose of this thesis is to develop optimized ASIC accelerator chips for LDPC decoders by following a joint algorithm, architecture, and circuit co-design approach. To this end, the contributions of this work are the following:

- Fast-converging LDPC decoding schedules, viz. interlaced column-row message passing (ICRMP), and fast column message passing (FCMP) schedules are proposed and investigated. The proposed schedules convergence in half the number of iterations than the serial decoding schedules, i.e., RMP and CMP, while incurring significantly lower computational complexity than the RBP-based fast-decoding schedules.

- Complexity and convergence analyses of the proposed schedules are carried out and compared with the existing LDPC decoding schedules. The convergence rate of the proposed schedules is verified using two analytical methods; EXIT (Extrinsic Information Transfer) charts and density-evolution by Gaussian approximation.

- An LDPC decoder architecture which implements the proposed FCMP schedule by targeting IEEE 802.11ad (WiGig) LDPC codes is proposed. The proposed decoder architecture achieves a decoding performance, comparable to state-of-the-art architectures, using a maximum number of iterations equal to only 2 , which allows the architecture to maintain a multi-Gbps throughput with a very high-energy efficiency.

- The proposed FCMP based decoder architecture is synthesized using the TSMC 40 nm standard CMOS technology. The synthesized architecture achieves a throughput of 8.4 Gbps while operating at a clock frequency of 200 MHz. The synthesized architecture achieves an energy-efficiency of 8.6 pJ/bit, which is the highest energy-efficiency of an IEEE 802.11ad LDPC decoder reported in the literature.

- Efficient memory-organization and barrel-shifters design for rotating the longer frames of QC-LDPC codes, in multiple cycles, by using the barrel shifter of the smallest frame is proposed. Multiplexers are then designed to forward a number of inputs equal to the maximum row-degree, instead of the number of columns of the SPCM.

- A direct bit-wise comparator circuit is proposed, and its boolean expressions are derived, which eliminated the need of multiplexers based complex comparators and resulted in an efficient design.

- A simple modification in the RMP decoding schedule is proposed to reduce the dynamic range of the LLR values, which allowed a reduction in bit-width of several modules of the decoder along with replacement of 22 saturated subtractors, involved in one stage of the decoder, with non-saturated ones.

- The proposed optimizations are then employed to implement a fully pipelined IEEE 802.11n/ac/ax LDPC decoder, and the decoder is synthesized in a 40 nm standard CMOS process. The synthesized decoder attains a throughput of **3.8**-**11.4** Gbps, occupies an area of **0.71** mm^2 while achieving **12.5** pJ/bit of energy-efficiency.

- Finally, an energy-efficient and high-throughput multi-core hardware architecture has been presented and physically implemented as an ASIC chip. The implemented ASIC chip achieves a peak throughput of 15 Gb/s while operating at a clock frequency of 250 MHz. The achieved throughput and the corresponding energy-efficiency are the best reported in the literature for an IEEE 802.11n/ac/ax LDPC decoder.

1.2 Outlines

The purpose of this thesis is to develop optimized ASIC chips as accelerator for LDPC decoders by following a joint approach for the algorithm, architecture, and circuits of an LDPC decoder. The structure of this thesis is as follows:

Chapter 2 introduces the background material related to the exiting LDPC decoding schedules,density evolution by Gaussian approximation method, and finally hardware implementation of LDPC decoders.

Chapter 3 discusses the proposed schedules along with their performance. Complexity analysis of the proposed schedules is also carried out in this chapter. Convergence analysis is then presented using extrinsic information transfer (EXIT) charts, and the mean evolution of Gaussian-approximated message-densities of the nodes. Finally the simulations results of the proposed schedules are presented in this chapter.

Chapter 4 presents the proposed architecture for the proposed FCMP schedule along with its decoding performance, and comparison with state-of-the art.

Chapter 5 presents the proposed VLSI optimizations of this work. decoding performance and VLSI synthesis results by applying the proposed optimizations to an IEEE 802.11n/ax/ax LDPC decoder are presented in this chapter. Finally, a multi-core hardware architecture, and physical implementation as an ASIC chip are elaborated in this.

Chapter 6 finally concludes the presented work.

1.3 Publications

At the time of writing, the following publications have been resulted out of this thesis work:

- **Saleh Usman** and M. M. Mansour, "Fast-Converging and Low-Power LDPC Decoding: Algorithm, Architecture, and VLSI Implementation", *submitted to IEEE Transactions on Circuits and Systems I.*

- **Saleh Usman** and M. M. Mansour, "An Optimized VLSI Implementation of an IEEE 802.11n/ac/ax LDPC Decoder", to appear in *IEEE International Symposium on Circuits and Systems (ISCAS)*, Oct. 2020, Seville, Spain.

- **Saleh Usman**, M. M. Mansour, and A. Chehab, " A Multi-Gbps Fully Pipelined Layered Decoder for IEEE 802.11n/ac/ax LDPC Codes," in *Proc. IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 194-199, Jul. 2017, Bochum, Germany.

- **Saleh Usman**, M. M. Mansour, and A. Chehab, "Interlaced column-row message-passing schedule for decoding LDPC codes," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, pp. 1-6, Dec. 2016, Washington, USA.

Chapter 2

Literature Review

LDPC codes are either defined by a sparse parity-check matrix (SPCM), or represented by a sparse bipartite Tanner graph consisting of variable nodes and check nodes. Variable nodes in the Tanner graph represent columns of the SPCM, and check nodes are associated with rows of the SPCM. Consider an LDPC code and let $\mathsf{G}(\mathsf{V} \cup \mathsf{C}, \mathsf{E})$ denote its associated bipartite graph G with vertex set V, as variable nodes, vertex set C, as check nodes, and an edge set E, connecting the vertices. Let P_v is the log-likelihood ratio (LLR) at the variable-node v, L_{cv} represent the message sent from check-node c to variable-node v, and Q_{vc} refers to the message sent from variable-node v to check-node c.

2.1 Standard Message-Passing (SMP)

The SMP scheduling, also called the flooding schedule, updates the variable nodes and check nodes in two phases as follows (e.g., see [9]). Two types of messages will be updated and exchanged: L_{cv} refers to the message sent from check node c to variable node v, and Q_{vc} refers to the message sent from variable node v to check node c:

- **Initialization**: Initialize all variable node messages to the received channel LLRs

$$P_{vc}^{(0)} = \mathsf{LLR}[v] \triangleq \frac{2r_v}{\sigma^2}, \quad \forall v \in \mathsf{G}, \tag{2.1}$$

  where r_v is vth received soft symbol and σ is the estimated standard deviation of channel noise.

- **Phase 1**: Update all check node messages at iteration $k \geq 1$ according to

$$L_{cv}^{(k)} = \Psi^{-1}\Big(\sum_{v' \in \mathcal{R}[c] \setminus v} \Psi \left(\left| P_{v'c}^{(k-1)} \right| \right) \Big), \quad \forall c \in \mathsf{G}, \tag{2.2}$$

where $\Psi(x) \triangleq 0.5\log(\tanh(x/2)) = \Psi^{-1}(x)$. The set $\mathcal{R}[c]$ denotes the set of $d_c = |\mathcal{R}[c]|$ variable nodes connected to check node c. Note that the notation $\mathcal{R}[c]\backslash v$ indicates that the summation does not include the previous message from variable node v that is set to receive the new updated message L_{cv} from check node c.

- **Phase 2**: Update all variable nodes at iteration $k \geq 1$ using the equation

$$P_{vc}^{(k)} = \mathsf{LLR}[v] + \Big(\sum_{c' \in \mathcal{C}[v]\backslash c} L_{c'v}^{(k)} \Big), \quad \forall v \in \mathsf{G}, \tag{2.3}$$

  where $\mathcal{C}[v]$ is the set of $d_v = |\mathcal{C}[v]|$ check nodes connected to the variable node v, and $\mathsf{LLR}[v]$ is the log-likelihood ratio (LLR) of the received channel value of variable node v. As in **Phase 1**, the summation does not include the previous message from neighboring check node c which the variable node is connected to.

For decision making in the final iteration, all neighboring check node messages are included in the summation when evaluating (2.3), i.e., none of the check nodes are excluded:

$$\Lambda[v] \triangleq \mathsf{LLR}[v] + \Big(\sum_{c' \in \mathcal{C}[v]} L_{c'v}^{(k)} \Big), \quad \forall v \in \mathsf{G}. \tag{2.4}$$

Here, the Λ's denote updated channel LLR values at the end of the final decoding iteration. Hard decisions on the bits are then made based on the sign of the Λ LLR values.

2.2 Column Message-Passing (CMP) Schedule

CMP starts from a variable node, and check nodes in the neighboring-set of the variable node send their updated messages to the variable node. The neighboring set of the check nodes update their messages using (2.2). This process of evaluating each variable node by collecting updated messages from its neighboring-set of the check nodes is repeated for all variable nodes of the bipartite graph, in one iteration. Decision making in the final iteration is performed by evaluating (2.4), as in SMP. CMP is also named as *shuffled iterative decoding* in the literature [10]. Fig. 2.1 illustrates the processing of this schedule for the first variable node. The dotted and solid lines in Fig. 2.1 represent the connections among the nodes, while the solid lines are the ones taking part in the update process.

2.3 Row Message-Passing (RMP) Schedule

In RMP, decoding proceeds row-wise instead of column-wise, i.e., check-node messages are computed using (2.2), and then all of its connected variable nodes

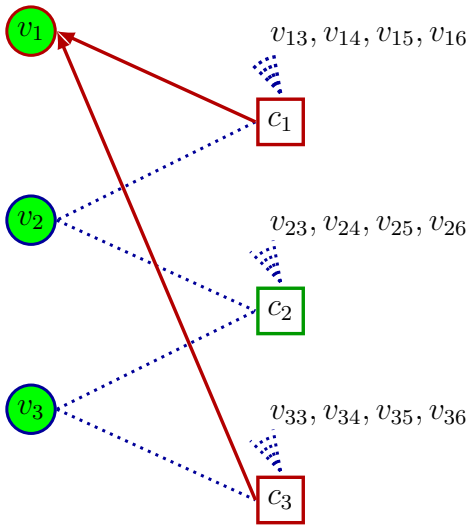

Figure 2.1: CMP processing schedule for variable-node v_1. Solid lines are the ones taking part in the update process.

are updated immediately using equation (2.3). This process is repeated for all check nodes of the bipartite graph, in one iteration. Memory-efficient implementation of the RMP is introduced in [11]. The processing of this schedule for the first check node is illustrated in Fig. 2.2.

2.4 Check-Node Computations

Several low-complexity variants of (2.2) have emerged in the literature [24] because the hardware implementation of (2.2) is not straightforward. The most widely used is the offset min-sum approximation, which is given as [25, 26]:

$$L_{cv}^{(k)} = \max\left\{\left(\min_{v'\in\mathcal{R}[c]\backslash v}|P_{v'c}^{(k-1)}|\right)-\beta, 0\right\}\prod_{v'\in\mathcal{R}[c]\backslash v}\operatorname{sign}(P_{v'c}^{(k-1)}) \tag{2.5}$$

where β is a correction factor determined empirically, and $\mathcal{R}[c]$ is the set of d_c variable nodes connected to check-node c. The notation $\mathcal{R}[c]\backslash v$ indicates that the message from variable-node v, set to receive the updated message L_{cv} from check-node c, is not included in the calculation. If $\min_1$, $\min_2$ are the magnitudes of the first and second minima among the inputs of the check node, and if v_{m1} is the index of the first minimum magnitude, then (2.5) can equivalently be written as:

$$L_{cv}^{(k)} = \begin{cases} \max(\min_2 - \beta, 0)\prod_{v'\in\mathcal{R}[c]\backslash v}\operatorname{sign}(P_{v'c}^{(k-1)}) & v = v_{m1}, \\ \max(\min_1 - \beta, 0)\prod_{v'\in\mathcal{R}[c]\backslash v}\operatorname{sign}(P_{v'c}^{(k-1)}) & \text{otherwise.} \end{cases} \tag{2.6}$$

Here the sign of each input, $\operatorname{sign}(P_{v'c})$, is a single bit, so their product is accomplished by just taking the Exclusive-OR (**XOR**) of all the bits.

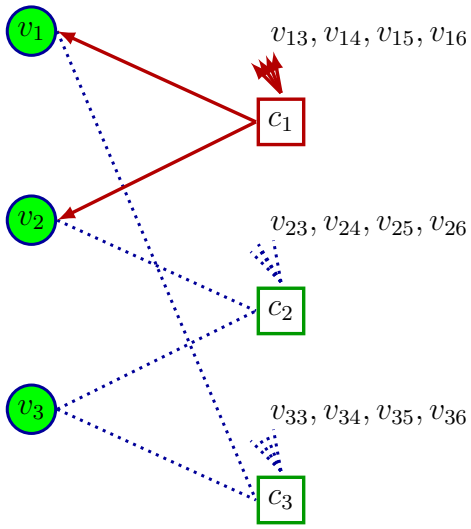

Figure 2.2: RMP processing schedule for first check node.

17	13	8	21	9	3	18	12	10	0	4	15	19	2	5	10	26	19	13	13	1	0	-	-
3	12	11	14	11	25	5	18	0	9	2	26	26	10	24	7	14	20	4	2	-	0	0	-
22	16	4	3	10	21	12	5	21	14	19	5	-	8	5	18	11	5	5	15	0	-	0	0
7	7	14	14	4	16	16	24	24	10	1	7	15	6	10	26	8	18	21	14	1	-	-	0

Figure 2.3: Base-matrix defined for IEEE 802.11n/ac: $F_L = 648$, $R = 5/6$, and $z = 27$.

2.5 QC-LDPC Codes

QC-LDPC codes have been adopted in recent communications standards like 5G New-Radio [3, 4], IEEE 802.11n/ac/ax (WiFi) [5], and IEEE 802.11ad (WiGig) [6], among others. The SPCM of a QC-LDPC code is represented by a base-matrix and a sub-block size parameter z. For example, for IEEE 802.11n/ax/ax LDPC codes, 12 base-matrices are defined; corresponding to 3 frame-lengths, 648, 1296, 1944, and 4 code-rates, $1/2, 2/3, 3/4, 5/6$. Sub-block sizes of $27, 54$, and 84 are defined corresponding to frame-lengths of $648, 1296$, and 1944, respectively. The base-matrix for rate $R = 5/6$ and frame length, $F_L = 648$ is shown in Fig. 2.3. The integers in the base-matrix represent the circular shift of the $z \times z$ identity matrix, while dashes (-) correspond to the $z \times z$ all-zeros matrices. The actual SPCM of the LDPC code is formed by replacing each entry in the base-matrix with a shifted $z \times z$ identity matrix or an all-zeros matrix.

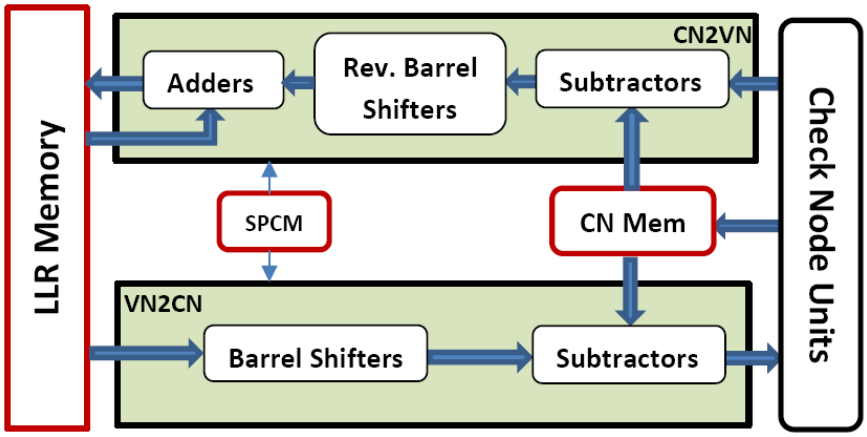

Figure 2.4: Block diagram of a memory-efficient non-pipelined RMP-based LDPC decoder architecture.

2.6 Hardware Implementation of RMP based LDPC Decoders

Several RMP-based QC-LDPC decoder architectures have emerged in the literature [9, 25, 23, 27], and a block diagram of a single-frame based QC-LDPC decoder architecture is shown in Fig. 2.4. The LLRMemory stores LLR values of (2.3) while the CNMem stores L_{cv} messages of (2.2), in Fig. 2.4. The SPCM stores the base-SPCM while the CheckNodeUnits implement (2.5). BarrelShifters and Rev.BarrelShifters in CN2VN and VN2CN blocks implement rotations and their reverse operations, required for QC-LDPC codes. The Adders, Subtractos, and CheckNodeUnits, are instantiated equal to the expansion factor of a QC-LDPC code, to exploit the structure of QC-LPDC codes in supporting the throughput demands of multi-Gbps LDPC decoding.

2.7 Density Evolution by Gaussian Approximation

Density evolution is a mathematical tool used to investigate the behavior of iterative message-passing decoding algorithms. The Gaussian approximation allows to carry out the density evolution analysis by approximating the probability densities of exchanged messages as Gaussian. Under density evolution, the symmetry condition $f(x) = f(-x)e^x$ is preserved for all messages, where $f(x)$ is the density of a message [28]. For a Gaussian density with mean m and variance σ^2, the symmetry condition reduces to $\sigma^2 = 2m$, which means that only the mean needs to be considered [29]. Density evolution with Gaussian approximation can be explained as follows. Assume that variable-node messages and check-node messages

are Gaussian random variables, with means denoted by m_v and m_c, respectively. By considering the flooding schedule for regular LDPC codes, these means get updated at $k \geq 1$ iteration according to the following rules [29]:

$$m_v^{(k)} = m_{co} + (d_v - 1)m_c^{(k-1)} \tag{2.7}$$

$$m_c^{(k)} = \phi^{-1}\left(1 - \left[1 - \phi\left(m_{co} + (d_v - 1)m_c^{(k-1)}\right)\right]^{d_c - 1}\right) \tag{2.8}$$

where m_{co} is the initial mean, $m_c^{(0)} = 0$, and

$$\phi(x) = \begin{cases} 1 - \frac{1}{\sqrt{4\pi x}} \int_{\mathbb{R}} \tanh\left(\frac{u}{2}\right) e^{-\frac{(u-x)^2}{4x}} du & \text{if } x > 0, \\ 1 & \text{if } x = 0. \end{cases}$$

For small x, say $x < 10$, the following approximation of $\phi(x)$ is typically used [29]:

$$\phi(x) \approx e^{-0.4527x^{0.86} + 0.0218}, \ 0 < x < 10. \tag{2.9}$$

Chapter 3

Proposed Decoding Schedules

For the purpose of hardware implementation of LDPC decoders, check-node processing is significantly simplified using the "min-sum" approximation [24]. In the min-sum approximation, each check node identifies the first and second minimum magnitudes among its inputs, and the check-node outputs are then generated by combining these minima with their corresponding signs, determined separately. The min-sum approximation simplifies the check-node processing for RMP, where each row of the SPCM directly corresponds to a check node. For CMP however, complexity reduction is not significant, and several techniques have emerged in the literature to leverage the benefits of the min-sum based check-node processing for the CMP, but the benefits come at the expense of potential degradation in decoding performance [30, 31, 32, 33].

In the CMP schedule, each check node in the neighboring set of a given variable node determines two minima, and the message propagated by each check node to that variable node is determined using these minima. The same minima, however, can be used to determine the messages for other variable nodes, which are in the neighboring sets of the check nodes being processed. The proposed schedules rely on the messages which could be generated for other variable nodes using the already determined minima, instead of applying existing complexity reduction techniques [31, 32] in the CMP schedule. This potential utilization of the check-node outputs creates an advantage in terms of convergence speed, in contrast to the loss in decoding performance incurred by applying the complexity reduction techniques.

Motivated by the above observation, interlaced column-row message-passing (ICRMP) and fast column message-passing (FCMP) schedules are proposed in this work. The proposed schedules propagate messages to all neighboring variable nodes of a check node, instead of just to the variable node, for which the check node is processed, as is the case in CMP. The difference between the ICRMP and FCMP schedules is of serial versus parallel processing of the check nodes, connected to a particular variable node. Referring to Fig. 3.1, the ICRMP schedule starts by considering the first variable-node v_1 and updating it using the messages

from check-nodes c_1 and c_3; processing up to this point is similar to CMP. Next comes the part which makes the ICRMP different than the CMP, and essentially amplifies its convergence speed. Check node c_1 propagates the messages to its neighboring variable nodes, other than v_1, and these variable nodes also get updated. After that, the check node c_3 propagates its messages to its neighboring variable nodes. This additional processing has minimal impact on the overall complexity of the CMP schedule, as shown by the complexity analysis in Section 3.1. The processing order of the proposed ICRMP schedule is depicted in Fig. 3.1, for the first variable node. In Fig. 3.1, the solid lines are the ones taking part in the update process, while v_{ij} represents a connection of ith check node to its jth neighboring variable node, like v_{13} is the connection of first check node to its third neighboring variable node.

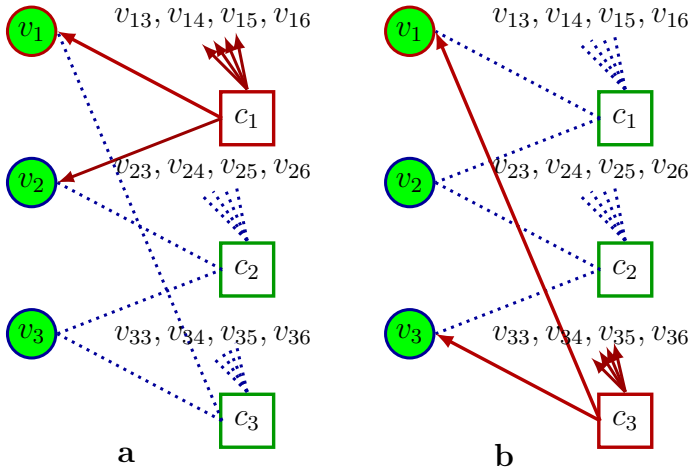

Figure 3.1: Proposed ICRMP schedule for 1st variable node. The check-nodes c_1 and c_3, connected to v_1, not only update v_1 but also their complete neighboring sets of variable nodes, in serial fashion.

In the FCMP schedule, on the other hand, the check-nodes connected to a particular variable node are processed in a parallel fashion, in contrast to the serial processing of the ICRMP. Referring to Fig. 3.2, FCMP schedule starts by considering the first variable-node v_1 and updating it using the messages from check-nodes c_1 and c_3, as is the case for ICRMP. Check nodes c_1 and c_3, then propagate their messages to their neighboring variable nodes, in parallel fashion, and these variable nodes also get updated.

For both the schedules, the process of propagating newly generated messages of the check nodes to all the connected variable nodes, completes a sub-cycle of information exchange that started from the first variable node and concluded at some set of variable nodes, depending on the structure of the SPCM. Sub-cycles equal to the number of variable nodes of the associated SPCM are repeated, one for each variable node, for one iteration.

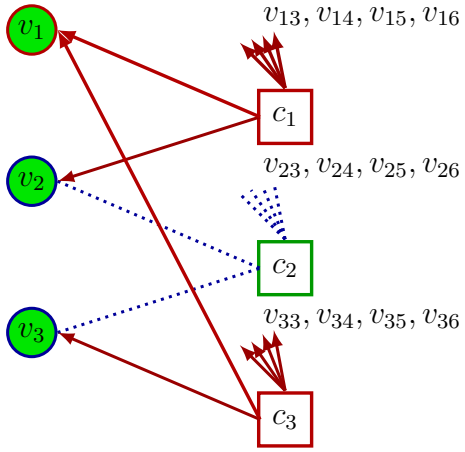

Figure 3.2: Proposed FCMP schedule for variable-node v_1. The check-nodes c_1 and c_3 update their complete neighboring sets of variable nodes, in parallel fashion.

In summary, one sub-iteration of the proposed schedules, for a particular variable node, processes all check nodes connected to that variable node and propagates the check-node messages to their neighboring-sets of the variable nodes. The sub-iteration is completed in parallel fashion for the FCMP, and in serial fashion for the ICRMP. The number of sub-iterations equals the number of variable nodes constitutes one iteration of the proposed schedules. One iteration of the proposed schedules adds $d_c - 1$ times **XOR** operations and an equal number of additions, compared to the CMP, with d_c being the check-node degree.

The proposed schedules, thus, present an optimization of the CMP schedule that improves the decoding throughput of the CMP by accelerating its convergence speed, instead of applying existing complexity reduction techniques which simplify the CMP hardware implementation [31, 32]. The schedules implement the check-node processing as standard min-sum approximation [34], to avoid any loss in decoding performance incurred by existing complexity reduction techniques, and instead modify the CMP schedule to accelerate the convergence speed.

The pseudo-code of the proposed ICRMP schedule is summarized in **Algorithm 1**, while **Algorithm 2** summarizes the pseudo-code of the FCMP. In the pseudo-code, d_{v_j} represent the degree of jth variable node, and c_i is the ith neighboring check node to the variable-node v_j. The parallel loop of the FCMP, demarcated by a bounding box in the pseudo-code, enables the reduction of the propagation delay, by parallel processing of the check nodes connected to a particular variable node.

Algorithm 1 Proposed ICRMP Schedule

for all $v, c \in \mathsf{G}$ **do** ▷ Initialization loop
  $P_v^{(0)} \leftarrow LLR[v]$
  $L_{cv}^{(0)} \leftarrow 0$
end for
for $k = 1$ to K **do** ▷ Iteration loop
  for $j = 1$ to N_v **do** ▷ Variable-node (VN) loop

Serial loop
    for $i = 1$ to d_{v_j} **do**
      $L_{c_i v'}^{(k)} \leftarrow$ Eq. (2.2) for all $v' \in \mathcal{R}[c_i]$
      for all $v' \in \mathcal{R}[c_i]$ **do** ▷ VN updates
        $P_{v'}^{(k)} \leftarrow$ Eq. (2.3)
      end for
    end for

  end for
end for
for all $v \in \mathsf{G}$ **do** ▷ Decision making
  $\Lambda[v] \leftarrow P_v \geq 0$
end for

3.1 Complexity Analysis

In this section, the computational complexities of the CMP and the proposed schedules are derived, and compared with other LDPC decoding schedules. Let N_v and M_c be the total number of variable nodes and check nodes, respectively, then the number of edges is, $E = N_v \times d_v = M_c \times d_c$, where d_v and d_c represent the average-degree of variable and check nodes, respectively.

3.1.1 Computational Complexity of CMP

In CMP, decoding starts from a variable node, and each of its connected check nodes computes its message to send to the variable node. For message computation, each check node determines two minima among the inputs, applies offset correction, and determines the sign of each output by **XOR**-ing the sign bits of the input messages. Each of the connected check nodes then sends its message to the variable node, and these messages are accumulated and get added to the channel LLR value; so the processing of one variable node involves d_v in number **XOR** operations and an equal number of additions at the variable node, along with the minima determination and offset correction operations. The process is repeated for all the N_v variable nodes. If V_c and C_c are the number of V2C

Algorithm 2 Proposed FCMP Schedule

for all $v, c \in \mathsf{G}$ **do** ▷ Initialization loop
  $P_v^{(0)} \leftarrow LLR[v]$
  $L_{cv}^{(0)} \leftarrow 0$
end for
for $k = 1$ to K **do** ▷ Iteration loop
  for $j = 1$ to N_v **do** ▷ Variable-node (VN) loop

Parallel loop
  parfor $i = 1$ to d_{v_j} **do**
    $L_{c_i v'}^{(k)} \leftarrow$ Eq. (2.2) for all $v' \in \mathcal{R}[c_i]$
    for all $v' \in \mathcal{R}[c_i]$ **do** ▷ VN updates
      $P_{v'}^{(k)} \leftarrow$ Eq. (2.3)
    end for
  end parfor

  end for
end for
for all $v \in \mathsf{G}$ **do** ▷ Decision making
  $\Lambda[v] \leftarrow P_v \geq 0$
end for

(variable-node to check-node) and C2V (check-node to variable-node) computations, respectively, then for a single iteration, V_c and C_c are given by the following equations:

$$V_\mathrm{c} = N_v \times d_v = E, \tag{3.1}$$

$$C_\mathrm{c} = N_v \times d_v \times (X + 2 \times M) = E, \tag{3.2}$$

where X represents an **XOR** operation count and M corresponds to the operation count of computing a single minimum with the offset correction. Since $X + 2 \times M$ is considered a single C2V computation, therefore, $C_\mathrm{c} = E$.

3.1.2 Computational Complexity of Proposed Schedules

For the proposed schedules, messages computed at check nodes, connected to a particular variable node, are further propagated to their neighboring variable nodes, as elaborated in Fig. 3.1 and Fig. 3.2. Compared to the CMP, this incurs additional computational cost at the check nodes and the variable nodes. For each check node connected to the variable node, being processed, the operation of determining the two minima stays the same, but each check node has to perform now d_c **XOR** operations to determine the signs of their outputs. Since the number of the connected check nodes to each variable node is d_v, then $d_v \times d_c$ **XOR**

operations are required to complete the processing of one variable node. Similarly, at the variable-nodes side, $d_v \times d_c$ variable nodes in total are updated using the addition operations. This process is repeated for all N_v variable nodes. If V_p and C_f are the number of V2C and C2V computations of the proposed schedules, respectively, then for a single iteration, V_p and C_p are given by the following equations:

$$V_\mathrm{p} = N_v \times d_v \times d_c = d_c \times E, \tag{3.3}$$

$$C_\mathrm{p} = N_v \times d_v \times (d_c \times X + 2 \times M). \tag{3.4}$$

By comparing (3.1) with (3.3), and (3.2) with (3.4), it is evident that the ICRMP and FCMP require d_c times additions at the variable-nodes, and the number of **XOR** operations is multiplied with the same factor at the check-nodes side, compared to the CMP schedule. Addition and **XOR** are simple operations, compared to the minima determination, which is the same for the CMP and the proposed schedules. The additional computational cost is thus moderate, compared to costly search and residual-computations operations, introduced by RBP-based fast-converging LDPC decoding schedules.

The power consumption of a decoder is directly related to the number of computations involved during the processing, along with other factors like switching activity, etc. For the proposed schedules, an increase in power consumption, compared to the CMP, would be moderate even if the required number of iterations would be the same. The proposed schedules, however, converge in almost half the number of iterations compared to the CMP or the RMP, so the power consumption drops in proportionate with the reduced number of iterations required for the convergence of the proposed schedules.

3.1.3 Complexity Comparison

The computational complexities of various LDPC decoding schedules, along with the proposed schedules, are summarized in Table 3.1. In the last column of Table 3.1, $r(v)$ denotes the operations required to calculate the variable-node residuals. The comparison of the complexities reveals that the proposed FCMP and ICRMP schedules, along with the CMP and RMP, do not introduce any costly search operations, or $r(v)$ operations in some cases. In contrast, both the FCMP and ICRMP schedules introduce a moderate increase in complexity, over RMP and CMP schedules. The rest of the fast-converging schedules, NWRBP, SVNF, TSDS, TRM-TVRBP, and RM-RBP, introduce costly search and $r(v)$ operations. These costly operations can be a fundamental reason that no VLSI implementation, to the best of authors' knowledge, has appeared in the literature for comparison with the existing VLSI implementations.

Table 3.1: Computational Complexities of LDPC Decoding Schedules

Schedules	V2C Computations	C2V Computations	Search Operations	$r(v)$
RMP	E	E	0	0
CMP	E	E	0	0
NWRBP*	$E(d_v-1)$	$E[(d_v-1)(d_c-1)+1]$	$M_c(E-1)$	0
SVNF*	$E(d_v-1)$	$E[(d_v-1)(d_c-1)+1]$	$E(d_c-1)d_v$	0
TSDS*	$E(d_v-1)$	$E(d_c-1)$	$E(N_v-T_L-d_v-1)*$	0
TRM-TVRBP**	$E/2$	$5(E/4)(d_c-1)$	$N_v(N_v-1)+E(d_c-1)$	$5(E/4)(d_c-1)$
RM-RBP**	E	$E.d_c$	$E(d_c-1)$	$E.d_c$
ICRMP	$E \cdot d_c$	E + *XOR OPs*†	0	0
Prop. **FCMP**	$\mathbf{E \cdot d_c}$	**E** + XOR OPs†	**0**	**0**

* The complexity expressions have been taken from Table I of [20].

** The complexity expressions have been taken from Table 2 of [21].

* T_L is the length of the tabu list of the TSDS schedule.

† Bitwise XOR are simple operations, compared to other tabulated operations.

3.2 Throughput and Area Tradeoffs

Computational complexity of ICRMP and FCMP schedules is the same, but the fundamental difference between the two is that the ICRMP requires serial processing of the check nodes connected to a variable node, while FCMP does this in parallel fashion. In hardware implementation, the number of clock cycles required to complete one iteration of the ICRMP is $d_v \times C$, while only C clock cycles are required to complete a single iteration of the FCMP schedule, where d_v and C are the average-variable-node degree and number of columns of the SPCM, respectively. So a higher throughput could be achieved with the proposed FCMP schedule in comparison with the ICRMP. As an example, for IEEE 802.11ad, rate-1/2 LDPC codes, the number of clock cycles required to complete one iteration of the FCMP schedule is 16 , equal to the number of columns of IEEE 802.11ad base matrix, while 52 cycles are required to complete one iteration in ICRMP. Thus FCMP attains a speed-up factor proportional to the average variable-node degree of the SPCM compared to ICRMP, and consequently, the overall throughput of the ICRMP is lower than that of FCMP.

On the other hand, d_v check-node units operate in parallel in the FCMP decoder, which increases the area-footprint proportionally. The ICRMP schedule, however, being serial in nature, performs additional computations by a single check-node unit, in d_v clock cycles. The requirement of additional clock cycles de-

creases the throughput but also requires less area for the decoder implementation. Thus the FCMP and ICRMP schedules provide a tradeoff between throughput and area-footprint. Power consumption of both the schedules, however, stays roughly the same since the computational complexity is the same for both cases.

3.3 Convergence Analysis

Convergence analysis of the proposed schedules are carried out using two methods: 1) Extrinsic information transfer (EXIT) charts, and 2) the mean evolution of Gaussian-approximated message-densities of the nodes.

3.3.1 Convergence Analysis using EXIT Charts

EXIT charts are used to analyze the convergence performance of iterative decoders [35], like LDPC decoders. For EXIT chart analysis, this thesis uses the notation of [36], which defines I_A and I_E as the average mutual information between the transmitted bits and the a priori LLRs and extrinsic LLRs, respectively. An approximation of the mutual information between the transmitted bits x_n and the LLRs $L(x_n)$, using the time average, is given as [37]:

$$I(x_n) \approx 1 - \frac{1}{N}\sum_{n=1}^{N} \log_2\left(1 + e^{-x_n . L(x_n)}\right) \tag{3.5}$$

To plot EXIT charts, the approximated mutual information is determined after every iteration, for a variable-node decoder (VND) and a check-node decoder (CND). The corresponding EXIT charts of CMP and the proposed FCMP based decoding are shown in Fig. 3.3 and Fig. 3.4, respectively. The EXIT charts reveal that the transfer of extrinsic information for the proposed FCMP schedule is much faster than CMP based decoding and, consequently, the FCMP converges in half the number of iterations than the CMP based decoding; the FCMP requires 7 iterations to converge, while CMP requires 14 . This is evident from the simulation plots that the convergence speed of the ICRMP schedule is a bit faster than the FCMP schedule, so both the schedules converge faster than the original CMP schedule.

3.3.2 Convergence Analysis by Gaussian Approximation

Convergence analysis of message-passing schedules can also be performed by tracking the mean-evolution of Gaussian-approximated message densities of variable and check nodes. The means of the node-messages of a fast converging schedule evolve rapidly in comparison with slower ones. The convergence speed of a serial LDPC decoding schedule by partitioning the check nodes is analyzed in [38] using this method. To verify the convergence behavior of the proposed

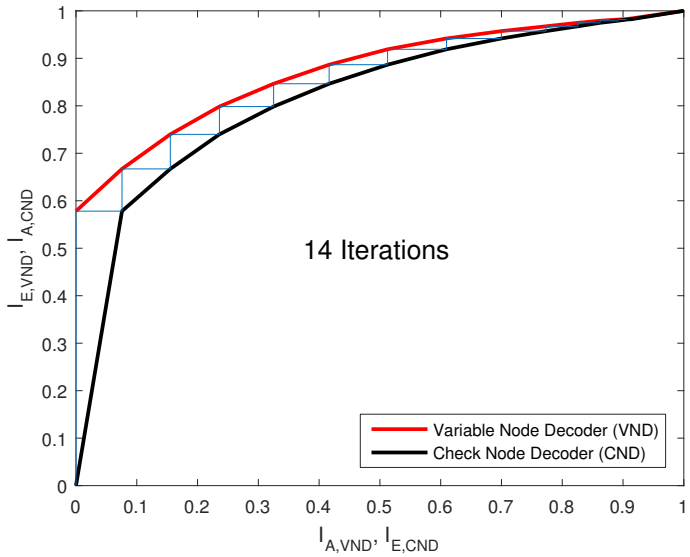

Figure 3.3: EXIT chart of CMP decoding for WiFi LDPC codes, rate 1/2, block-length 1944, at $E_{\mathrm{b}}/N_0 = 1.2$ dB.

schedules and to compare it with the CMP, the mean update equations of the CMP and the ICRMP and FCMP are derived in this section by modifying (2.7) and (2.8).

In the CMP schedule, every update of the variable node receives messages from the check nodes; a fraction of these messages is updated in the current iteration, and the rest are from the previous iteration. Let p_1 and p_2 be the probabilities of the number of check-node messages updated in the previous and the current iterations, respectively, and $p_1 + p_2 = 1$. The total number of the check-node messages for a particular variable node is d_v. Therefore, $p_1 \times d_v$ is the expected number of neighboring check-node messages to a variable node, which depends on the previous iteration, and $p_2 \times d_v$ is the expected number of neighboring check-node messages to a variable node depending on the current iteration. The higher the number of messages that depend on the current iteration (i.e., higher p_2), the faster the means of messages evolve. The means of the variable-node messages $\overline{m}_v^{(k)}$ and the check-node messages $\overline{m}_c^{(k)}$ for the CMP at the kth iteration are given by:

$$\overline{m}_v^{(k)} = m_{co} + p_1 \times (d_v - 1) \times \overline{m}_c^{(k-1)} + p_2 \times (d_v - 1) \times \overline{m}_c^{(k)} \tag{3.6}$$

$$\overline{m}_c^{(k)} = \phi^{-1}\Big(1 - \big[1 - \phi(m_{co} + p_1 \times (d_v - 1) \times \overline{m}_c^{(k-1)} + p_2 \times (d_v - 1) \times \overline{m}_c^{(k)})\big]^{d_c - 1}\Big) \tag{3.7}$$

Other quantities and initial conditions are similar to (2.7)-(2.8). Here p_1 and p_2 depend on the processing order of variable nodes. For instance, for the first

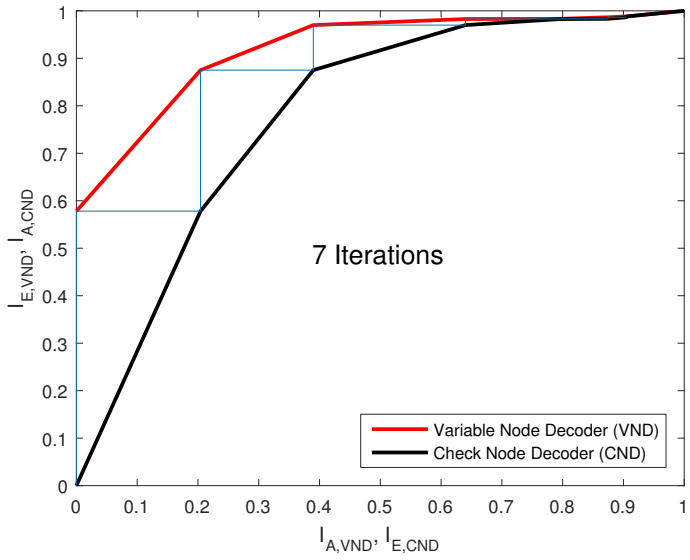

Figure 3.4: EXIT chart of proposed FCMP decoding, for the same LDPC codes and E_b/N_0 as in Fig. 3.3.

variable node in the processing schedule, all check nodes connected to the variable node perform computations that depend on messages from the previous iteration; since no variable node is updated yet so $p_1 = 1$ and $p_2 = 0$. As processing proceeds, p_1 continues to decrease, and p_2 continues to increase because the check nodes connected to the given variable node perform computations that depend more on the input messages of the current iteration rather than the previous iteration. For the last variable node in the processing schedule, $p_1 = 0$ and $p_2 = 1$ since all the nodes have now been updated.

For the proposed schedules, each check node connected to a given variable node, not only updates that variable node but also the remaining $d_c - 1$ variable nodes. Therefore, for a variable node, the probability of getting a check-node message whose computations are based on the current iteration is multiplied by $d_c - 1$. Hence, by using the same p_1 and p_2 as defined above for the CMP, the means of the variable-node messages $\overline{\overline{m}}_v^{(k)}$ and the check-node messages $\overline{\overline{m}}_c^{(k)}$ for kth iteration of the FCMP and ICRMP are updated as:

$$\overline{\overline{m}}_v^{(k)} = m_{co} + \frac{p_1 \times (d_v - 1) \times \overline{\overline{m}}_c^{(k-1)}}{p_1 + p_2 \times d_c} + \frac{p_2 \times d_c \times (d_v - 1) \times \overline{\overline{m}}_c^{(k)}}{p_1 + p_2 \times d_c} \tag{3.8}$$

$$\overline{\overline{m}}_c^{(k)} = \phi^{-1}\Big(1 - \big[1 - \phi(m_{co} + \frac{p_1 \times (d_v - 1) \times \overline{\overline{m}}_c^{(k-1)}}{p_1 + p_2 \times d_c} + \frac{p_2 \times d_c \times (d_v - 1) \times \overline{\overline{m}}_c^{(k)}}{p_1 + p_2 \times d_c -})\big]^{d_c - 1}\Big) \tag{3.9}$$

By considering a check node with different values of p_2 (and therefore different p_1), the mean evolution of the node-message versus the number of iterations is plotted in Fig. 3.5. The figure shows that the evolution of the means of the proposed schedules (plotted in red) is faster than the CMP. In general, different values of p_1 and p_2 for each node result in corresponding mean evolution curves. The overall convergence speed of the schedule depends on the overall effect of all the mean evolution curves.

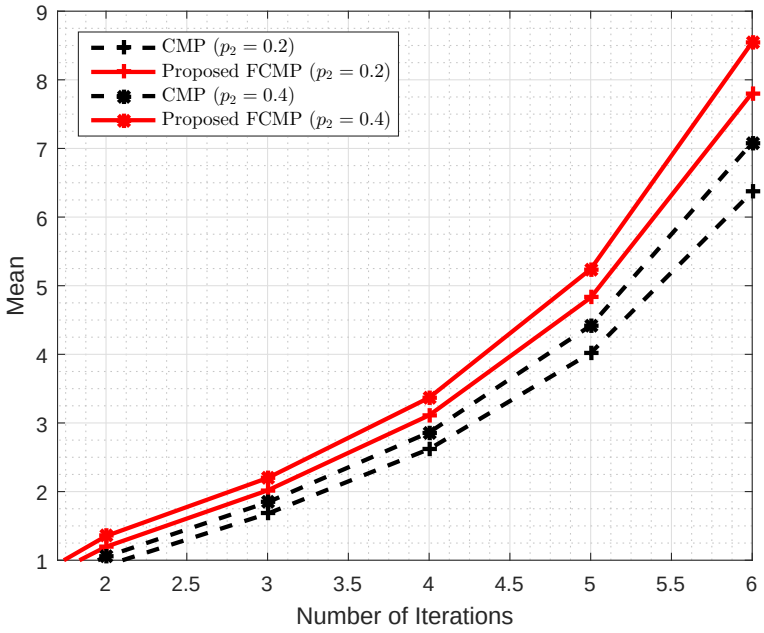

Figure 3.5: Comparison of mean evolution of check-node messages of a (3,6) regular LDPC code using the CMP and the proposed FCMP schedule.

Here p_1 and p_2, for a given variable node, are determined by considering the LDPC graph and figuring out the fractions of check-node messages connected to the variable node that depends on the previous and the current iteration, respectively.

3.4 Simulation Results

The potential advantages of the proposed schedule in terms of convergence speed, BER performance, and FER performance are assessed by simulating the RMP, CMP, RBP-based, and the proposed FCMP and ICRMP decoding schedules. BER and FER curves and the corresponding convergence speed, in terms of the number of iterations taken by the schedules are plotted. Randomly generated, IEEE 802.11ad (WiGig), IEEE 802.11n/ac (WiFi), and IEEE 802.16e (WiMAX)

LDPC codes, are considered while fixing the maximum number of iterations. The comparison of the proposed FCMP and ICRMP schedule with the RMP and CMP is shown in Fig. 3.6 - Fig. 3.9. As is evident from the plots, the fast convergence rate of the FCMP, along with ICRMP, in comparison with RMP and CMP, is verified consistently by these simulations. The simulation plots reveal that the FCMP schedule requires a slightly higher number of iterations than the ICRMP. The higher convergence speed of the ICRMP is because of the serial processing of the neighboring check nodes of a particular variable node, however, the throughput of the FCMP, is higher than the ICRMP, as explained in Section 3.1. The comparison of the proposed schedules with RBP-based fast-decoding schedules, using BER and FER plots against the maximum number of iterations, is shown in Fig. 3.10 - Fig. 3.12. The comparison reveals that the proposed schedules achieve the same decoding and convergence performance as the RBP-based schedule, although the computational complexity of the proposed schedules is significantly lower than the RBP-based schedules, which enables a multi-Gbps low-power VLSI implementation of the proposed schedules.

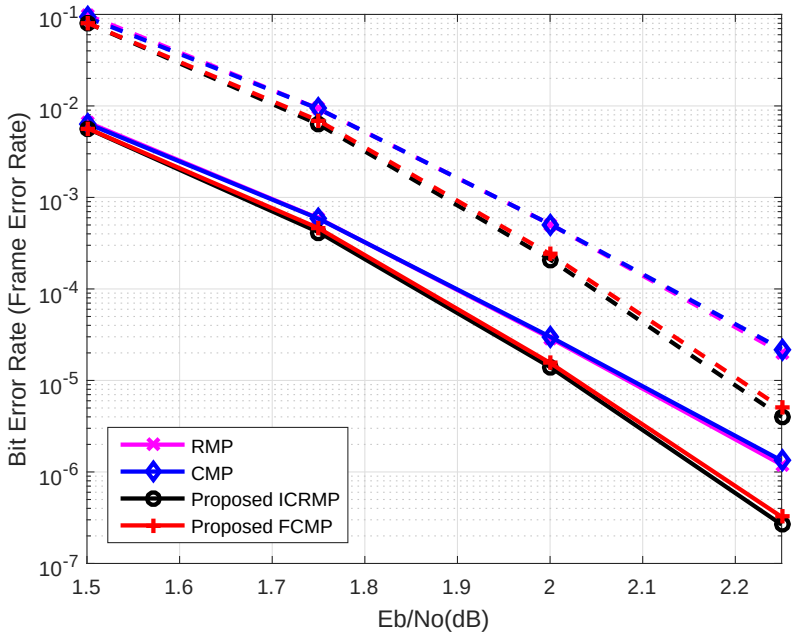

Figure 3.6: BER comparison of the proposed FCMP schedule with the ICRMP, CMP and RMP, for randomly generated LDPC codes with rate $= 1/2, d_v = 3, d_c = 6, N_v = 2000$, and $I_{\max} = 100$.

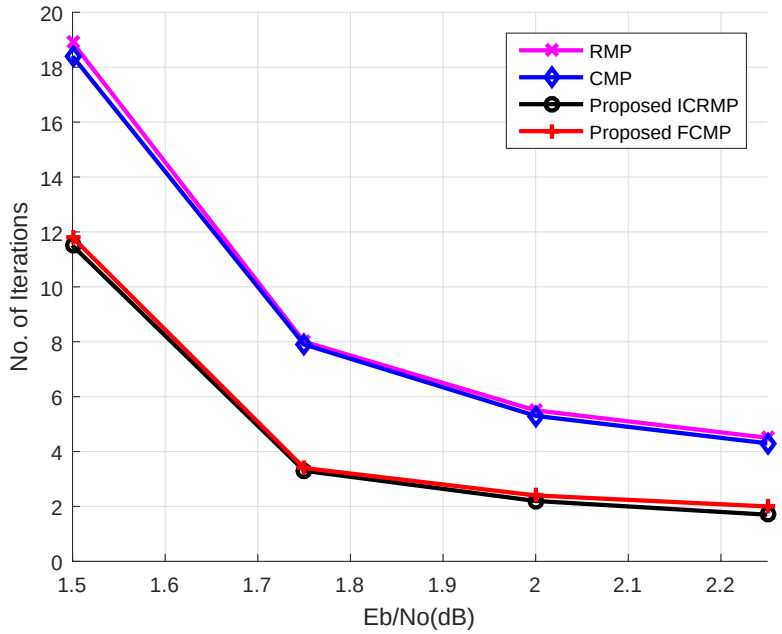

Figure 3.7: Comparison of iterations corresponding to Fig. 3.6.

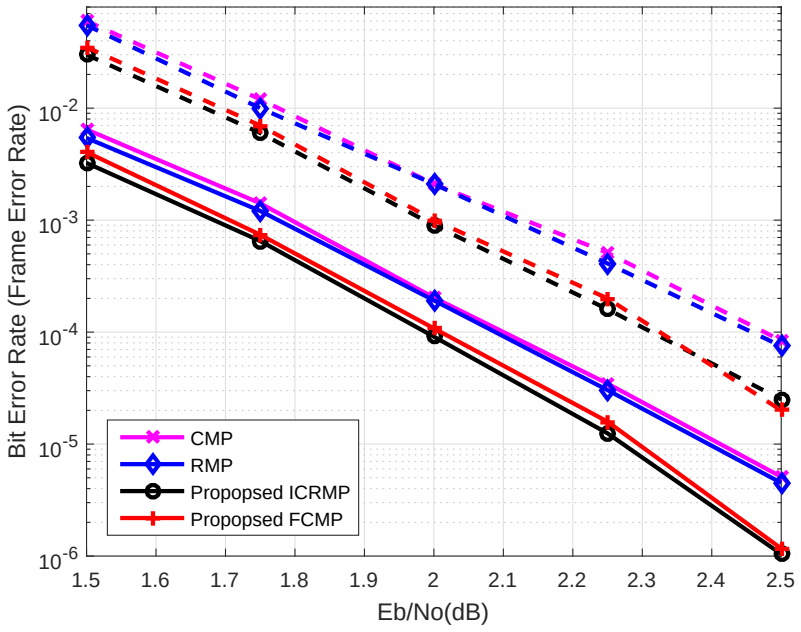

Figure 3.8: BER and FER comparison of the proposed schedules with the CMP and RMP, for IEEE 802.11n/ac (WiFi) LDPC codes, rate 1/2, block-length 648, and $I_{max} = 100$.

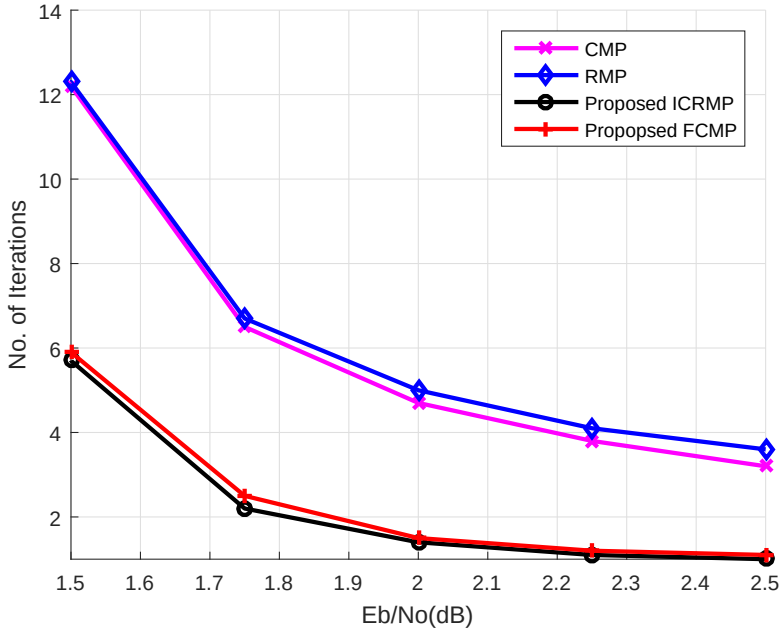

Figure 3.9: Iterations count corresponding to Fig 3.8.

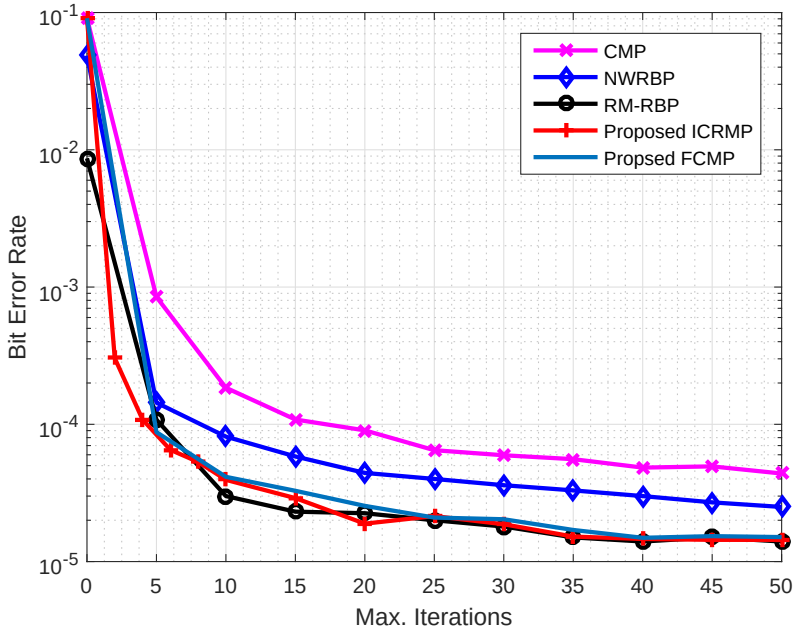

Figure 3.10: BER vs. number of iterations for WiMAX LDPC codes, rate 1/2, block-length 576, at Eb/No=2.5 dB.

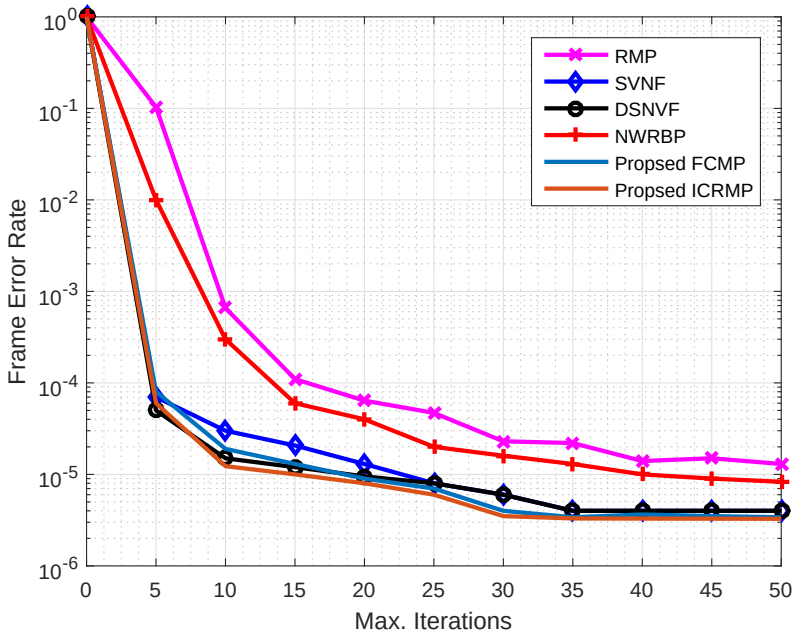

Figure 3.11: FER vs. number of iterations for WiFi LDPC codes, rate 1/2, block-length 1296, at Eb/No=2.25 dB.

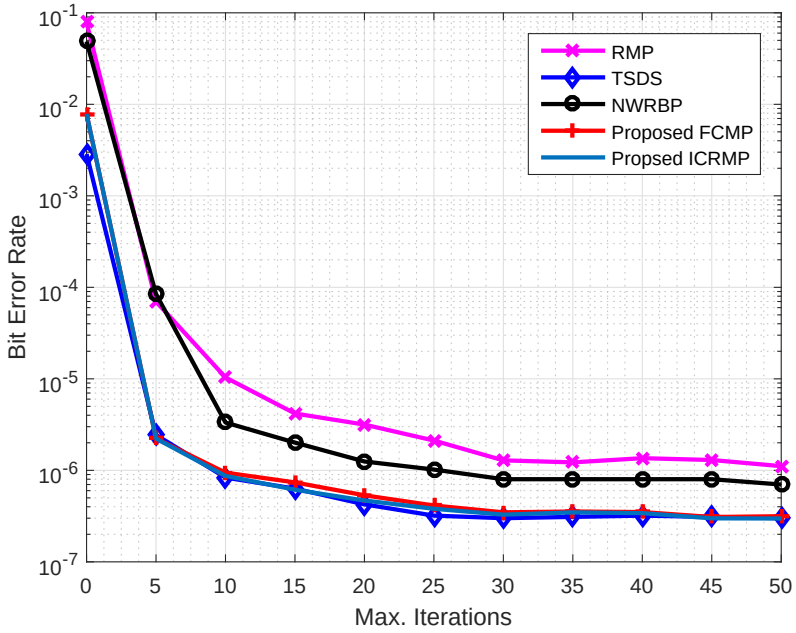

Figure 3.12: BER vs. number of iterations for WiMAX LDPC codes, rate 1/2, block-length 576, at Eb/No=3.0 dB.

Chapter 4

Proposed FCMP Decoder Architecture

An LDPC decoder architecture based on the proposed FCMP schedule is presented in this chapter, by modifying the legacy LDPC decoder architectures. The FCMP decoder is targeted for hardware implementation because of its capability of achieving a higher throughput than the proposed ICRMP schedule, as discussed in Section 3.1. The proposed hardware architecture for the FCMP decoding schedule, targeted for IEEE 802.11ad LDPC codes, is shown in Fig. 4.5. The proposed architecture replicates the processing units, in comparison with Fig. 2.4, by a factor of 4 , equal to the maximum column-degree defined for the IEEE 802.11ad base matrix. The processing units are replicated by 4 in order to process the neighboring set of check-nodes of a particular variable-node in a single clock cycle, as required in FCMP. The proposed architecture is pipelined with a pipeline depth of 2 and processes two frames simultaneously in the pipeline stages to avoid any idle-cycle. The number of memory modules is doubled to store the messages of two frames, instead of a single frame in Fig. 2.4. The memory modules are modified for reading and writing of the messages of multiple blocks of the processing units instead of a single block. Design space exploration, significant processing blocks, and decoding performance of the proposed architecture are elaborated below.

4.1 Design Space Exploration

The design space of the proposed architecture is explored for appropriate selection of correction-factor β, used in (2.6), quantization bits for LLRMem (Q_L), quantization bits for CheckNodeUnits (Q_C), and the maximum number of iterations $I_{\max}$. The simulation corresponding to different values of Q_L, Q_C, and $I_{\max}$ are shown in Fig. 4.1-Fig. 4.4. The simulation plot with $Q_L = 6$ and $Q_C = 5$ is shown in Fig. 4.1, which shows a saturation of decoding curves at high SNR

values, and therefore $Q_L = 7$ is selected for the proposed architecture. By comparing Fig. 4.3 and Fig. 4.4, it is evident that the loss in decoding performance is not significant in changing $I_{\max} = 5$ to $I_{\max} = 2$, compared to a throughput increase which could be achieved using $I_{\max} = 2$, and therefore $I_{\max} = 2$ is chosen for the proposed architecture. $Q_L = 7$, $Q_C = 5$, $\beta = 1$, and $I_{\max} = 2$, are finalized for the proposed architecture.

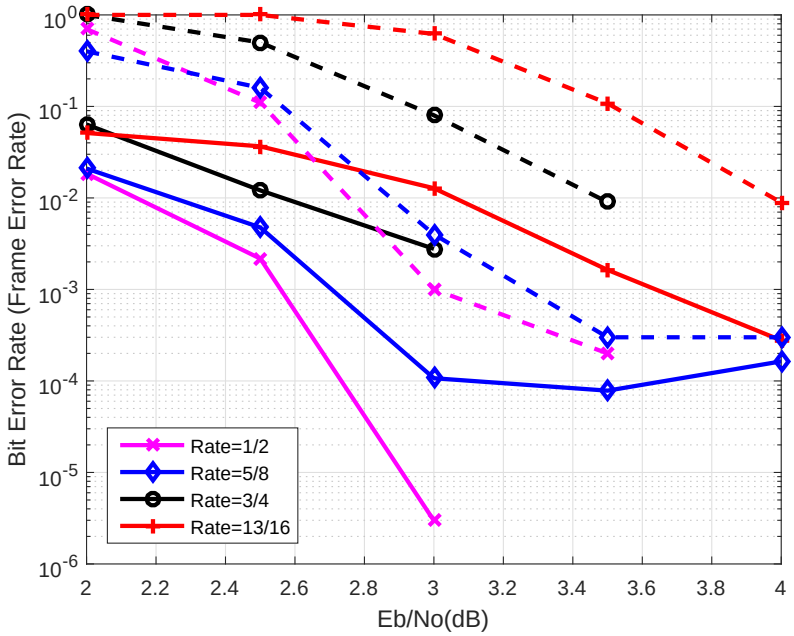

Figure 4.1: Fixed-point decoding performance of proposed FCMP architecture for IEEE 802.11ad (WiGig), with $Q_L = 6$, $Q_C = 5$, and $I_{\max} = 2$. Solid and dotted lines represent BERs and FERs, respectively.

4.2 Pipelining

Pipelining the architecture of an LDPC decoder significantly affects the convergence speed, area, and throughput of the decoder. A single-frame based pipelining alters the decoding schedule by delaying the messages updates, and consequently, slows down the decoding convergence. Therefore, a double-frame based pipelined architecture, with a pipeline depth of 2, is proposed and implemented in this paper. To properly balance the two pipeline stages of the data-path, the pipeline registers are introduced deep in the CNUs, shown in Fig. 4.6

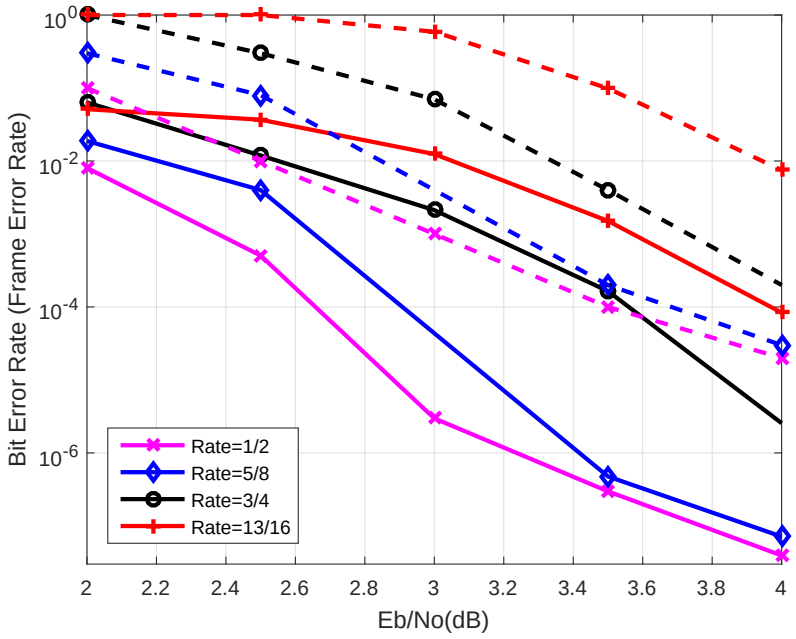

Figure 4.2: Fixed-point decoding performance of proposed FCMP architecture for IEEE 802.11ad (WiGig), with $Q_L = 8$, $Q_C = 5$, and $I_{\max} = 2$. Solid and dotted lines represent BERs and FERs, respectively.

4.3 Check Node Unit (CNU)

The CNU of the proposed architecture is pipelined to balance the two pipeline stages of the proposed decoder architecture, and its block diagram is shown in Fig. 4.6. The pipelining enables the idle-cycle-free processing of two-frames simultaneously. For IEEE 802.11 ad (WiGig), the maximum-degree of a check node is 16, so the number of inputs of each CNU is also 16. Each block of the CheckNodeUnits uses 42 such CNUs while 4 such blocks are used in the proposed architecture, as is evident from Fig. 4.5.

4.4 Check-Node Memory

The Check-Node (CN) Memory in the proposed architecture is a multi-port memory, and its block diagram is shown in Fig. 4.7. The memory saves the magnitudes of first and second minima, as well as the signs and indices of the check-node messages required to regenerate the check-node outputs using (2.6). WritingCircuitry in Fig. 4.7 enables the memory locations for writing while ReadingCircuitry is a network of multiplexers which selects and generates the required check-node outputs. The number of the input and the output lines of the memory are shown as 4 to emphasize the fact that the memory saves and generates messages for

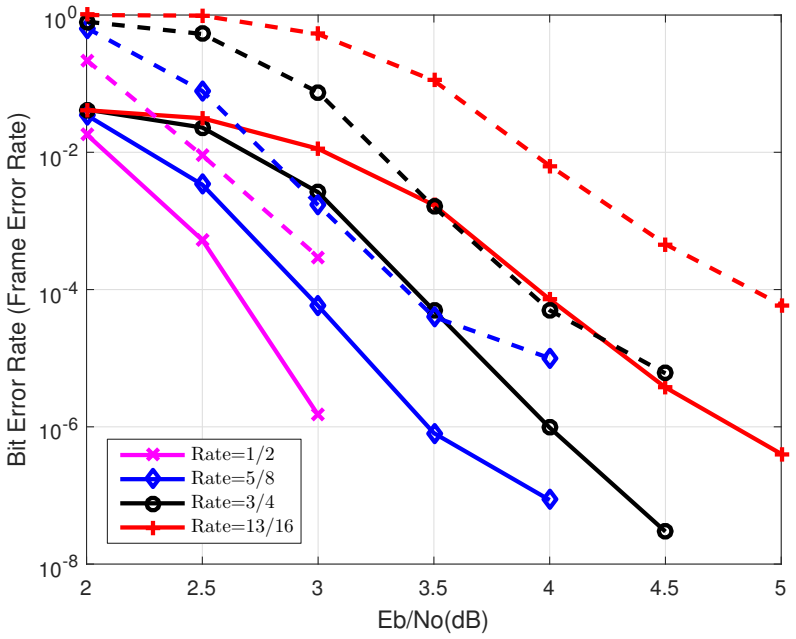

Figure 4.3: Fixed-point decoding performance of proposed FCMP architecture for IEEE 802.11ad (WiGig), with $Q_L = 7$, $Q_C = 5$, and $I_{\max} = 5$. Solid and dotted lines represent BERs and FERs, respectively.

4 blocks of CNUs, in a single clock cycle. Each of the input lines further consists of wires for minima, signs, indices, and addresses of the check-node messages.

4.5 Decoding Performance

The fixed-point decoding performance of the proposed FCMP-architecture, targeted for IEEE 802.11ad (WiGig), is plotted in Fig. 4.4. Design parameters, $Q_L = 7$, $Q_C = 5$, $\beta = 1$, and $I_{\max} = 2$ are determined during the design exploration phase. By considering Fig. 4.4, it is evident that the proposed architecture converges to a BER value of 10^{-6} at $E_b/N_0 = 3.3$, 3.6, 4.0, and 4.8, for $rate = 1/2$, 5/8, 3/4, and 13/16, respectively. The architecture presented in [39], for example, achieves a BER value of 10^{-6} at $E_b/N_0 = 4.6$, 4.6, 5.0, and 5.4, for $rate = 1/2$, 5/8, 3/4, and 13/16, respectively, while operating at 202 MHz and using $I_{\max} = 10$. The proposed architecture thus achieves a better decoding performance and higher energy efficiency than [39] by implementing the proposed FCMP schedule.

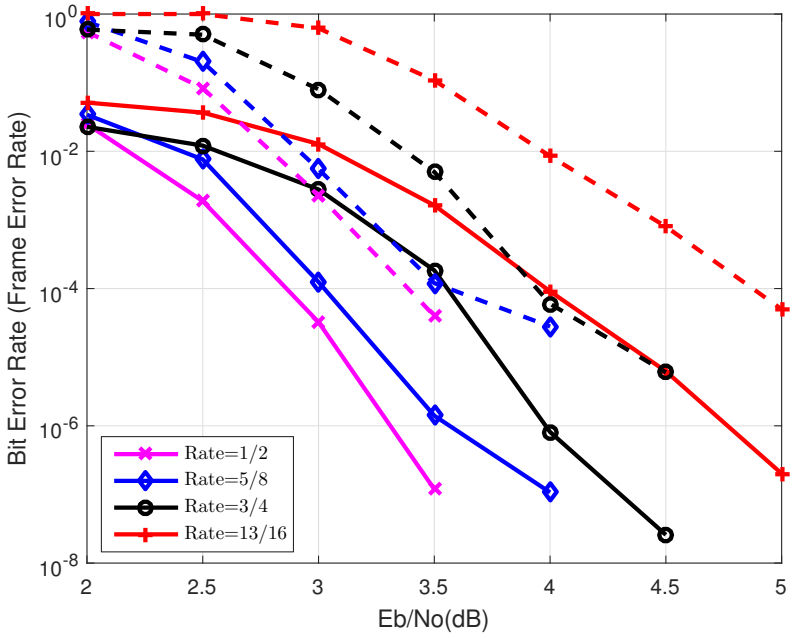

Figure 4.4: Fixed-point decoding performance of the synthesized FCMP architecture for IEEE 802.11ad (WiGig), with $Q_L = 7$, $Q_C = 5$, $I_{\max} = 2$. Solid and dotted lines represent BERs and FERs, respectively.

4.6 VLSI Synthesis Results

The proposed hardware architecture of the FCMP schedule for IEEE 802.11 ad LDPC codes, shown in Fig. 4.5, is synthesized, using the TSMC 40 nm CMOS process, for a clock speed of 200 MHz. The achieved throughput θ is calculated as:

$$\theta = \frac{F_L \times N_F}{I_{\max} \times C} \times f_{\text{clk}}. \tag{4.1}$$

Here, F_L is the frame-length, N_F is the number of frames, $I_{\max}$ is the maximum number of iterations, C is the number of cycles to complete one decoding iteration, and f_{clk} is the clock frequency. For the proposed architecture, $F_L = 672, N_F = 2, I_{\max} = 2, C = 16$, and $f_{\text{clk}} = 200$ MHz which gives us a throughput of 8.4 Gpbs. The latency η of the proposed architecture is calculated as:

$$\eta = \frac{I_{\max} \times C}{f_{\text{clk}}}. \tag{4.2}$$

By using the same values as in (4.1), the latency η of the decoder turns out to be 0.16 μs. For the proposed architecture, it is possible to shorten the critical path even further, and thus a higher throughput can be achieved by increasing the clock speed. However, most of the industry-grade communication/networking

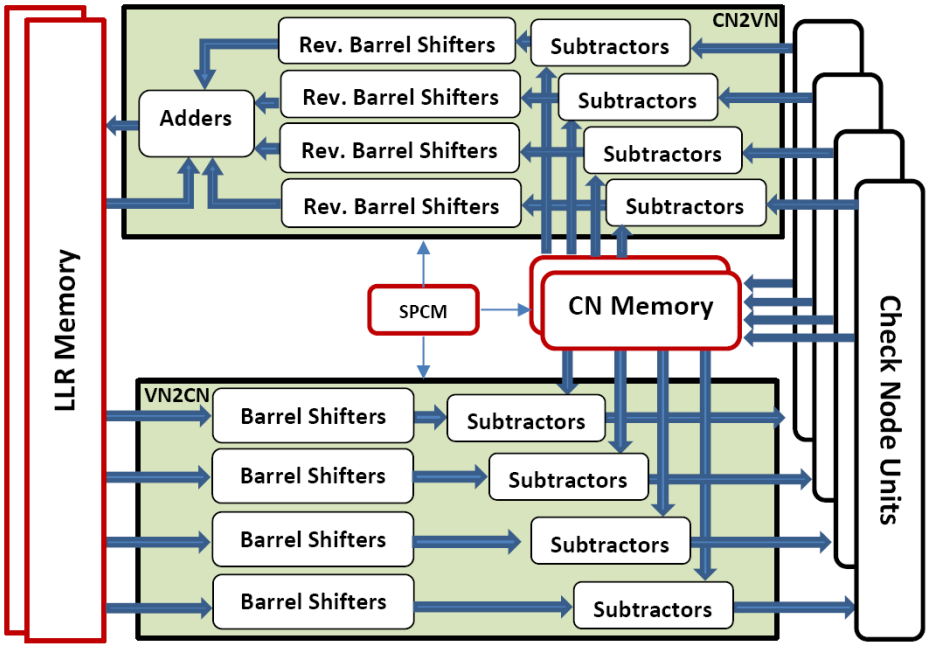

Figure 4.5: A two-frames based block diagram of an FCMP decoder architecture for IEEE 802.11ad LDPC codes.

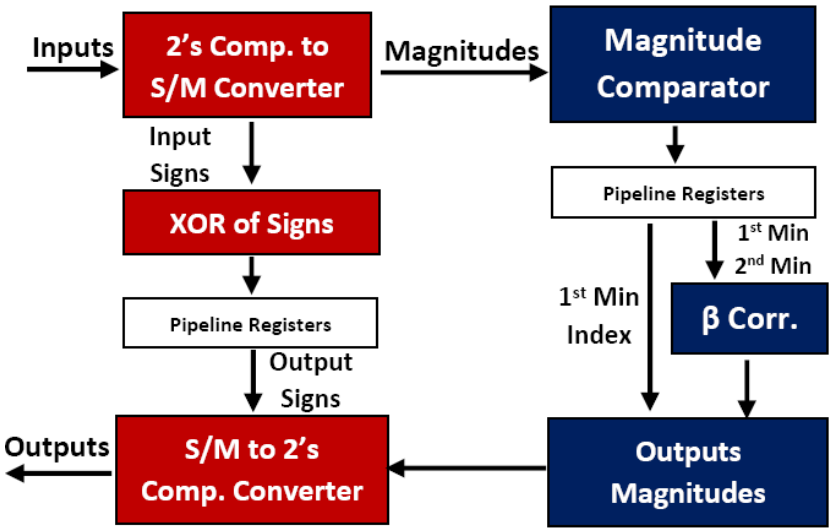

Figure 4.6: Block diagram of a single pipelined Check Node Unit

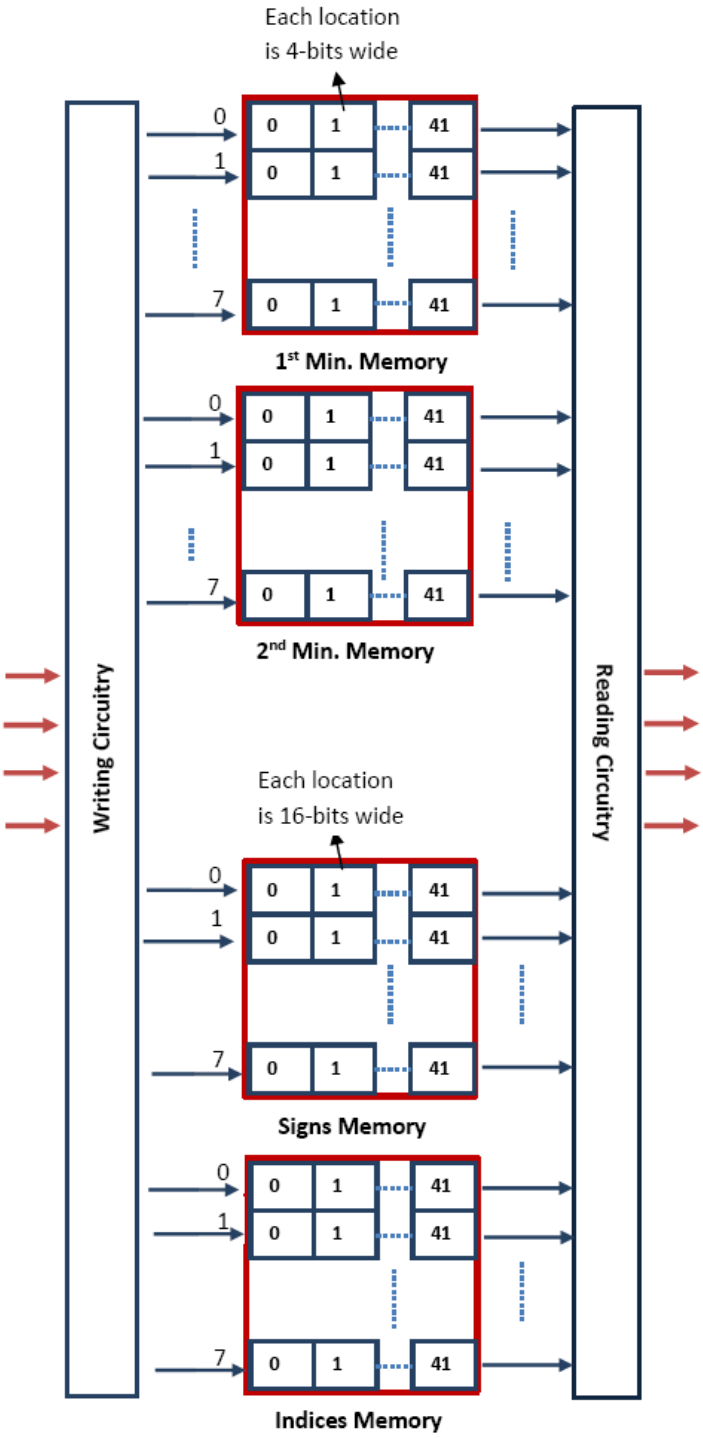

Figure 4.7: Block diagram of Check-Node (CN) Memory

system-on-chips (SoC) operate in a clock speed range of $200-300$ MHz, to be able to close the timing across all the process, voltage, and temperature (PVT) corners of the modern CMOS processes [40]. Therefore, a realistic clock speed of an LDPC decoder should be constrained around 200 MHz, to support the SoC integration capability of the intellectual property (IP) core with other signal processing blocks of the digital baseband [39].

Table 4.1: Synthesis Results of FCMP Decoder

Clock Freq. (MHz)	200
Supply Voltage (V)	1.1
Throughput (Gbps)	8.4
Design Area (mm^2)	1.44
Latency (μs)	0.16
Gate Count (K)	2880
Power (mW)*	60

* Power is estimated at BER = 10^{-6}, which corresponds to single iteration for all code rates. The switching activity for the power estimation is generated by applying test-vectors of each code rate to the synthesized netlist.

4.7 Comparison with State-of-the-Art

A comparison of VLSI synthesis results of the proposed architecture with state-of-the-art is summarized in Table 4.2. The comparison reveals that the proposed architecture attains minimal power consumption while achieving a throughput that is comparable with other architectures. This minimal power consumption, while maintaining a comparable throughput, proves that the proposed FCMP schedule and its architecture are very attractive from an energy-efficiency perspective (the proposed architecture attains an efficiency of 8.6 pJ/bit, which is the best reported in the literature). For the proposed architecture, the clock-speed of the decoder could further be reduced to save additional power, as the achieved throughput of the decoder, 8.4 Gpbs, is still higher than the throughput requirement of IEEE 802.11ad standard, which is up to 7.0 Gpbs. The area of the proposed architecture is larger than some of the other designs, which is a direct consequence of replicating the processing units and decoding the two frames simultaneously; however, the additional area does not result in an increased power consumption. In fact, it indirectly causes a reduction in the overall power consumption by enabling the implementation of the FCMP schedule, which cuts the number of the iterations required for the convergence.

Table 4.2: Comparison of Proposed Decoder with State-of-the-Art

	This work	[26]	[41]	[42]	[39]	[43]	[44]
Technology (nm)	**40**	65	65	40	28	28 (FD-SOI)	40
Quant. Bits	**7/5**	5	5	5	5	5	5
Clock Freq. (MHz)	**200**	150	360	220	202	260	720
Thr. (Gpbs)	**8.4**	3.08	6	6.2	6.78	12	11.8
Latency (μs)	**0.16**	-	0.224	0.109	0.79	0.112	-
Area ($\mathbf{mm^2}$)	**1.44**	1.6*	1.6	0.8	1.99	0.63	0.2
S. Thr. (Gpbs)	**8.4**	4.9 [a]	9.6 [a]	6.2	4.75 [b]	**	11.8
S. Area ($\mathbf{mm^2}$)	**1.7** *	0.6 [c]	0.6 [c]	0.8	3.98 [d]	**	0.2
Power (mW)	**72** *	100*	373.6	203	279	180	182
A. Eff. ($\mathbf{Gbps/mm^2}$)	**4.9**	8.1	16	7.75	1.2	-	60
E. Eff. (**pJ/bit**)	**8.6**	32.47	62.27	32.95	41.15	15	15.42

[a] Throughput is scaled by 1.6 to account for the 65nm technology.
[b] Throughput is scaled by 0.7 to account for the 28nm technology.
[c] Area is scaled by 0.38 to account for the 65nm technology.
[d] Area is scaled by 2.0 to account for the 28nm technology.
* Area and power are scaled by 1.2 for the synthesis results.
** Area and throughput are not scaled because of the used FD-SOI technology.

Chapter 5

Proposed VLSI Optimizations

This chapter presents circuit level optimization techniques for high-throughput and energy-efficient VLSI implementation of LDPC decoders by targeting an IEEE 802.11n/ac/ax decoder. These techniques are then employed to design two variants of a fully-pipelined IEEE 802.11n/ac/ax standard-compliant LDPC decoder. The optimization techniques include efficient barrel shifter and memory organization, usage of multiplexers at I/Os of LLR memory, a bitwise comparator design, and selection of proper quantization bits. The first two techniques could be applied to QC-LDPC decoders as per the structure of their base matrix, while the rest are applicable for the general class of LDPC decoders.

5.1 Barrel Shifters and Memory Organization

Message rotation is required in decoding QC-LDPC codes to align variable-node messages that are passed to check-nodes, and vice versa, and is carried out by BarrelShifters and Rev.BarrelShifters, shown in Fig. 2.4. Designing BarrelShifters and Rev.BarrelShifters, for the largest Z, defined for a particular QC-LDPC code increases the area significantly, since this unnecessarily inflates all BarrelShifters and Rev.BarrelShifters, which are 48 for IEEE 802.11 n/ac LDPC codes, and reduce to 44 , after using the multiplexers, proposed in Section 5.2. To avoid this simplistic approach, processing the longer frames in multiple cycles is an adequate compromise as it does not affect the performance of the decoder other than restricting its throughput, for longer frames, equal to the throughput of the smallest frame.

To this end, proper organization of messages in LLRMemory and its connections with the barrel shifter are necessary (see Fig. 5.1). In Fig. 5.1(A), Z_L, Z_M, and Z_S are the sizes of the largest, medium, and smallest frames, where $Z_M = 2 \times Z_S$ and $Z_L = 3 \times Z_S$, which is the case for IEEE 802.11 n/ac LDPC codes. In Fig. 5.1(B), m_c and m_s are the modified cycles and modified shift values, respectively; they are generated by using the actual cycle number and actual

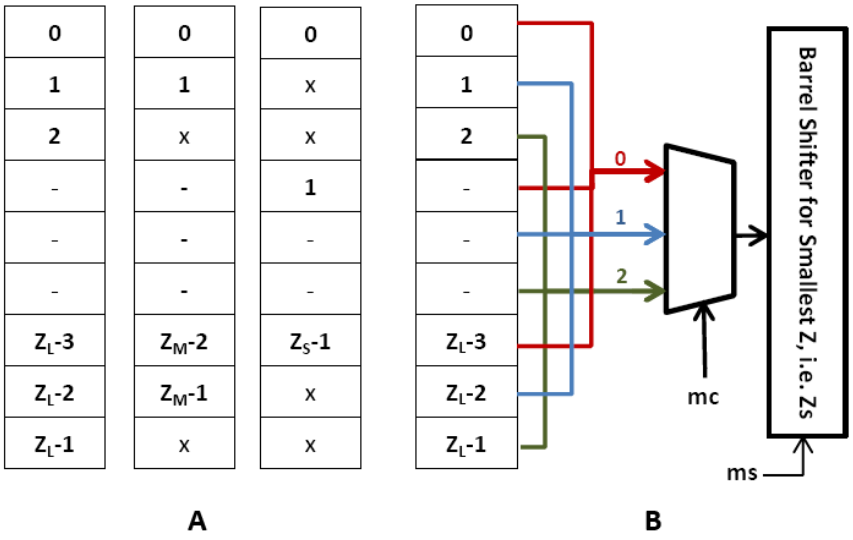

Figure 5.1: (A) Messages organization in LLRMemory. (B) Rotating a frame of length Z_L with a barrel shifter, capable of rotating only frame of length Z_S.

shift values, respectively. The generation of mc and ms is explained as follows.

Let c_t be the total number of cycles required to process a frame, which is 3 for Z_L, 2 for Z_M, and 1 for Z_S, for the case of IEEE 802.11 n/ac LDPC codes. Suppose that the actual required shift value is shamt, which is divided by c_t and the resultant quotient and remainder be q and r, i.e., $q = \mathsf{shamt}/c_t$ and $r = \mathsf{shamt} \bmod c_t$. If c_n is the natural order of cycles, then m_c and m_s are generated by the following rules:

$$m_c = (c_n + r) \bmod c_t, \quad c_n = 0, 1, \cdots, c_t - 1, \tag{5.1}$$

$$m_s = \begin{cases} q+1 & \text{if } c_t = 3; r = 2; m_c = 0 \text{ or } 1 \\ q+1 & \text{if } c_t = 2; r = 1; m_c = 0 \\ q & \text{otherwise.} \end{cases} \tag{5.2}$$

For example, to rotate a frame of length $Z_L = 9$ by 5 in $c_t = 3$ cycles with a barrel shifter of size $Z_S = 3$, we have $q = 1$ and $r = 2$. Using (5.1) and (5.2), the m_c and m_s values are determined to be $2, 0, 1$ and $2, 2, 1$, respectively, by using $c_n = 0, 1, 2$ as the natural order of cycles. The inputs of the barrel shifter, shown in Fig. 5.1, and the corresponding outputs, along with the determined values of m_c and m_s are shown in Fig. 5.2. Inputs to the barrel shifter are read by following the cycle order of m_c, while the outputs are written in the natural order, as shown by the rows of the matrix of Fig. 5.2. In Fig. 5.2, the messages shown in memory are for illustration purposes only, since an immediate write operation is not required in the decoder because the barrel shifter outputs are directly forwarded for further processing.

mc	Barrel Shifter Inputs			ms	Barrel Shifter Outputs		
2	2	5	8	1	5	8	2
0	0	3	6	2	6	0	3
1	1	4	7	2	7	1	4

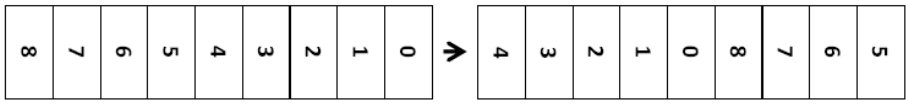

Figure 5.2: Rotating a frame of length $Z_L = 9$ by 5, using a barrel shifter of size $Z_S = 3$, in three cycles.

5.2 Multiplexers for Reduced Number of Inputs

The memory locations of LLRMemory of the decoder correspond to the columns of the base-matrix of QC-LDPC code. To read messages from LLRMemory usage of small multiplexers is proposed to reduce the number of output lines to the rest of the modules to "maximum-row-degree", instead of the number of columns of the base-matrix. For the IEEE 802.11n/ac/ax case, the maximum row-degree is 22 while the number of columns is 24 , so the multiplexers are used to select the 22 messages from 24 memory locations. From Fig. 2.3, it could be observed that the messages corresponding to the first 12 columns can be directly passed from memory, while the messages corresponding to the rest of the 12 columns be passed via multiplexers on 10 output lines, as shown in Fig. 5.3. This reduces the required number of adders, subtractors, and BarrelShifters/Rev.BarrelShifters of Fig. 2.4, from 24 to 22 , as well as simplifies the CNUs to process 22 inputs rather than 24 . Select-lines of the multiplexers are given different names, m_0,m_1,m_2,m_3, and are asserted/de-asserted, based on the connections which are derived from the base-matrix.

5.3 Proposed Comparator Design

A novel comparator is proposed to implement the comparison operation, required for the minima determination, in an efficient way to avoid the multiplexers chain on the critical path. The proposed approach is based on determining the first minima by a bitwise comparison of the inputs. The second minimum is then determined by re-utilizing the bitwise comparator circuit again. For this, the first minimum is excluded from the inputs, and then the rest of the of the inputs are applied to the comparator circuit again. The exclusion operation is carried out by x-nor operations of the first minimum with all of the inputs, which replaces the first minimum with all ones, the maximum value. The comparator circuit determines the second minimum among the applied inputs, after the exclusion of the first minimum. Follows are the boolean expressions and circuit details to

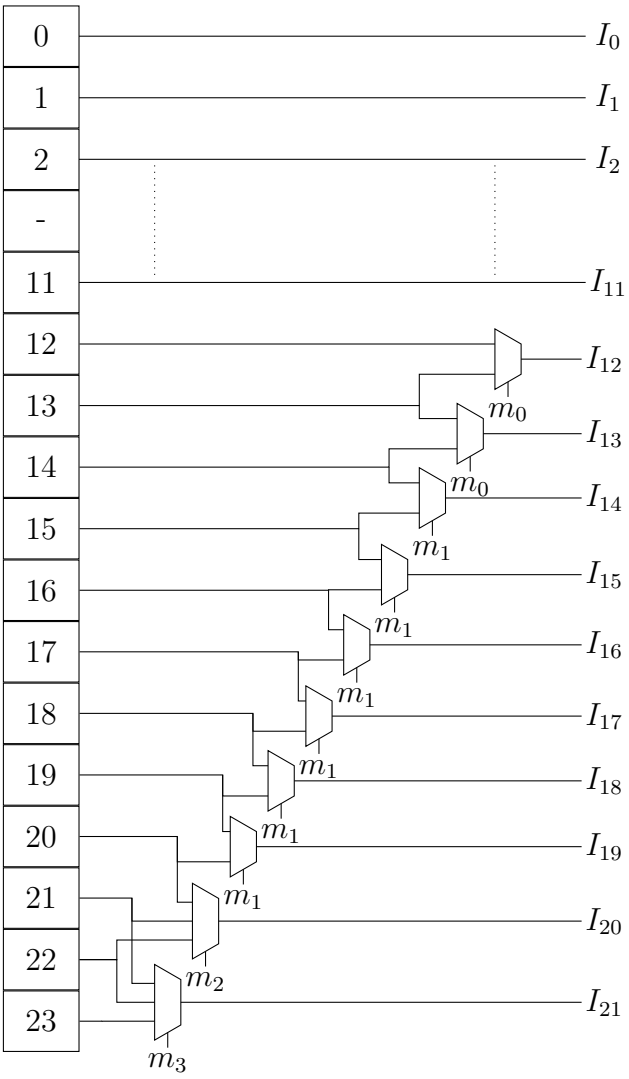

Figure 5.3: Proposed multiplexers organization with LLRMemory

determine the minimum.

The proposed comparator is based on the individual bit-comparison of the inputs starting from MSB and going down to the LSB. Let A, B, C, and D be the numbers, to be compared for determination of the minimum and $a_n a_{n-1}....a_1 a_0$, $b_n b_{n-1}....b_1 b_0$, $c_n c_{n-1}....c_1 c_0$ and, $d_n d_{n-1}....d_1 d_0$ be their corresponding bits. The minimum M_1, with the corresponding bits $m_n m_{n-1}....m_1 m_0$, is determined by the following set of boolean expressions:

$$\begin{aligned}
m_n &= a_n b_n c_n d_n \\
m_{n-1} &= (\bar{m_n} a_n + a_{n-1})(\bar{m_n} b_n + b_{n-1}) \\
&\quad (\bar{m_n} c_n + c_{n-1})(\bar{m_n} d_n + d_{n-1}) \\
&\quad . \\
&\quad . \\
&\quad . \\
m_1 &= (\bar{m_n} a_n + \bar{m_{n-1}} a_{n-1} + .. + \bar{m_2} a_2 + a_1) \\
&\quad (\bar{m_n} b_n + \bar{m_{n-1}} b_{n-1} + .. + \bar{m_2} b_2 + b_1) \\
&\quad (\bar{m_n} c_n + \bar{m_{n-1}} c_{n-1} + .. + \bar{m_2} c_2 + c_1) \\
&\quad (\bar{m_n} d_n + \bar{m_{n-1}} d_{n-1} + .. + \bar{m_2} d_2 + d_1) \\
\\
m_0 &= (\bar{m_n} a_n + \bar{m_{n-1}} a_{n-1} + .. + \bar{m_1} a_1 + a_0) \\
&\quad (\bar{m_n} b_n + \bar{m_{n-1}} b_{n-1} + .. + \bar{m_1} b_1 + b_0) \\
&\quad (\bar{m_n} c_n + \bar{m_{n-1}} c_{n-1} + .. + \bar{m_1} c_1 + c_0) \\
&\quad (\bar{m_n} d_n + \bar{m_{n-1}} d_{n-1} + .. + \bar{m_1} d_1 + d_0)
\end{aligned} \tag{5.3}$$

The circuit diagram corresponding to (5.3) is shown in Fig. 5.4.

5.4 Quantization and Unsaturated Subtractors

By considering (2.1), it is evident that the multiplication of received soft symbol values, r_v by 2, doubles the dynamic range of input LLR values and therefore following modified initialization is proposed to be used instead of (2.1):

$$P_{vc}^{(0)} = \frac{r_v}{\sigma^2}, \quad \forall v \in \mathsf{G}. \tag{5.4}$$

The modification allows us to reduce the bit-width for CheckNodeUnits, CNMem, and the associated circuitry to 4 instead of 5, as is normally used in existing architectures. Quantization-bits for LLRMem, $Q_L = 7$, are chosen to allow the ample growth of the LLR values, at high SNR. Unsaturated subtractors, equal to the maximum row-degree of 22, are used, after the CNUs, to avoid the costly operations of comparison and clipping.

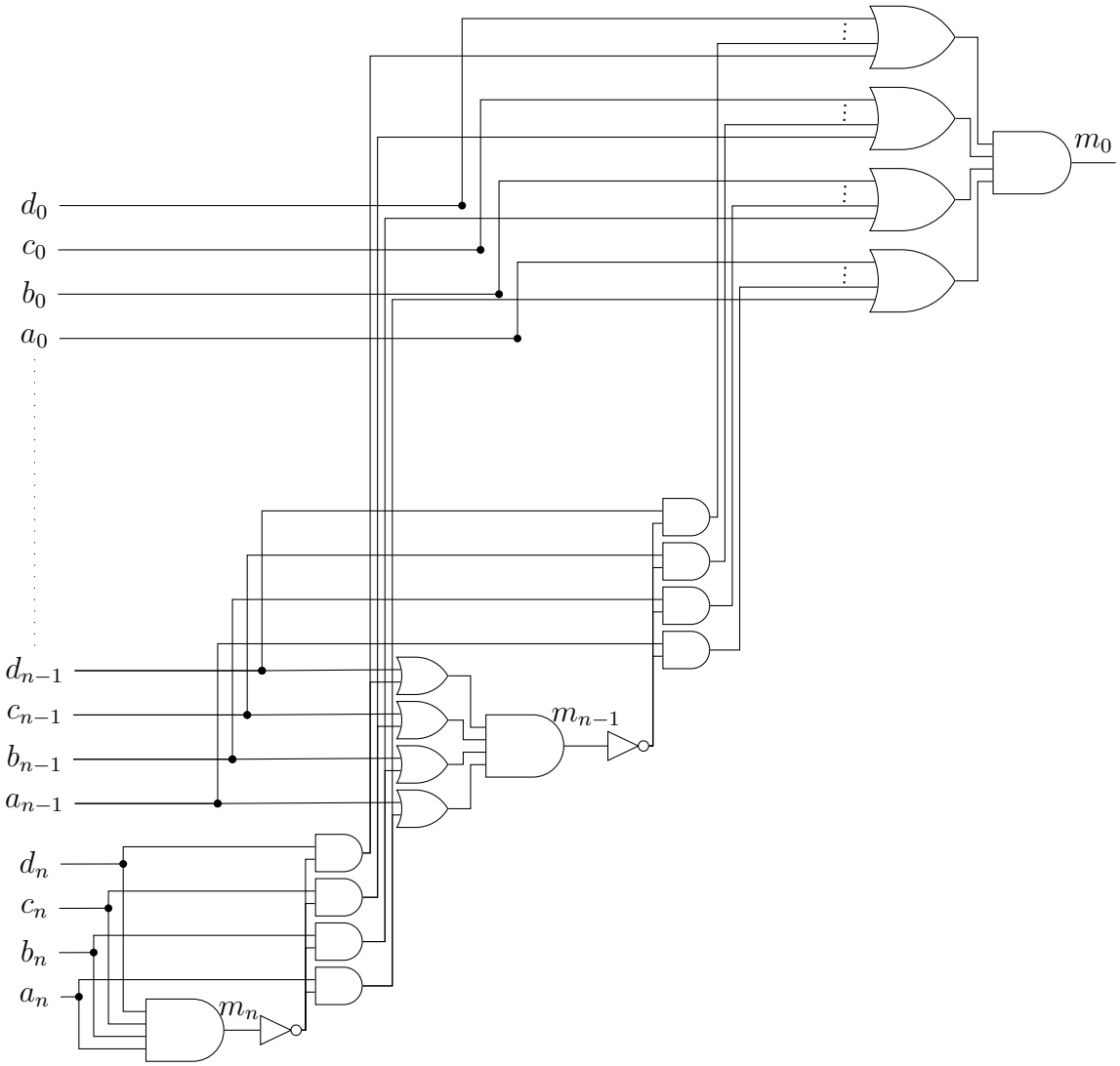

Figure 5.4: Proposed circuit for the minimum determination among 4 inputs while each input comprises of n bits.

5.5 Design Space Exploration

The design space of an RMP based architecture is explored for appropriate selection of correction-factor β, used in (2.6), quantization bits for LLRMem (Q_L), and quantization bits for CheckNodeUnits (Q_C). The simulations corresponding to different values of Q_L, Q_C, and β are shown in Fig. 5.6-Fig. 5.7. Bit Error Rate (BER) curves and corresponding iterations count of the implemented QC-LDPC decoder for IEEE 802.11n/ac/ax are shown in Fig. 5.7 and Fig. 5.8. Max. number of iterations, $I_{\max} = 8$ are used, and $Q_L = 7$, $Q_C = 4$, and $\beta = 1$ are finalized for the implemented design.

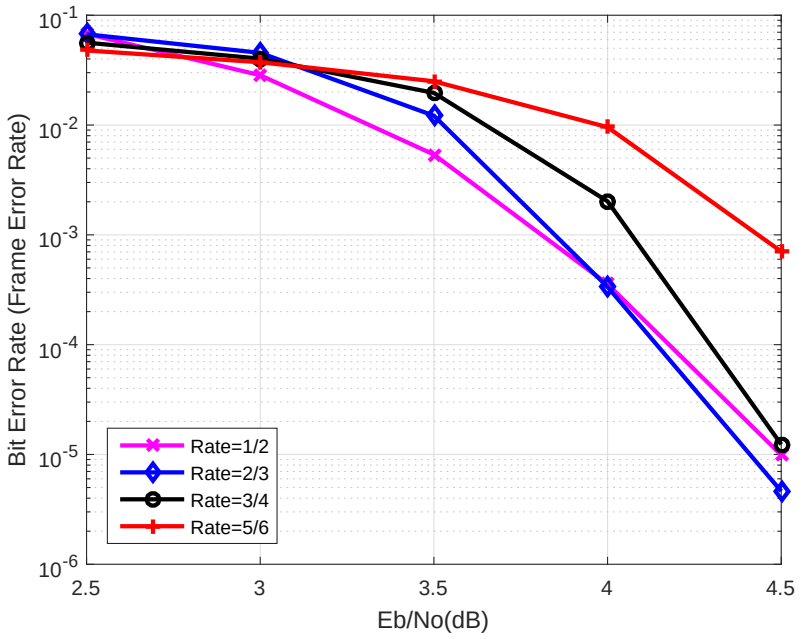

Figure 5.5: Fixed-point BER performance of the proposed decoder for $F_L = 648$, with $Q_L = 6$, $Q_C = 5$, and $\beta = 1$.

5.6 Pipelining

The convergence speed of single-frame based pipelined decoder becomes slower because the new check node messages get delayed at variable nodes. This delay is directly related to the pipeline depth of the decoder. To evaluate the loss in convergence speed as a function of the pipeline depth floating-point pipelined implementations of IEEE 802.11n/ac/ax LDPC codes, block length 648, rate 1/2, are simulated and compared against a non-pipelined (standard RMP) version. As shown in Figs. 5.9 and 5.10, the design is simulated for up to three pipeline

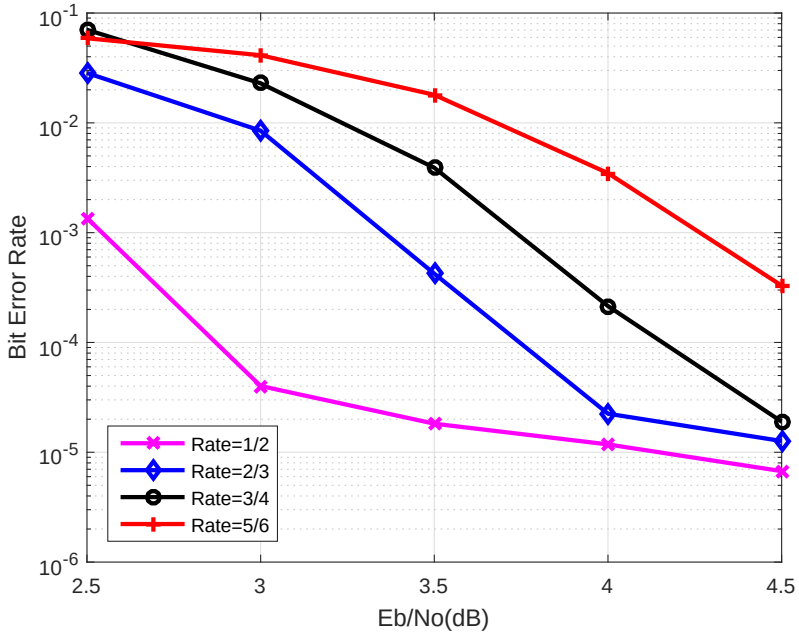

Figure 5.6: Fixed-point BER performance of the proposed decoder for $F_L = 648$, with $Q_L = 7$, $Q_C = 4$, and $\beta = 0$.

stages. The figures show that a pipeline depth of three is an adequate compromise between the convergence speed and pipeline depth of the proposed architecture.

5.7 VLSI Synthesis

The decoder for IEEE 802.11na/ac/ax QC-LDPC codes, implementing the proposed optimizations, is synthesized, using the TSMC 40 nm CMOS process. The critical path delay of the decoder is 1.78 ns. This enables us to operate the decoder at a clock frequency, $f_{\text{clk}} = 562$ MHz. The peak throughput θ is calculated as:

$$\theta = \frac{F_L \times N_F}{I_{\max} \times R} \times f_{\text{clk}}, \tag{5.5}$$

Where F_L is frame length, N_F is the number of frames, and R is the number of rows of the base-matrix of QC-LDPC code. For the proposed decoder, $F_L = 648, N_F = 1, I_{\max} = 8, f_{\text{clk}} = 562$ MHz. The number of rows of the base-matrix are $12, 8, 6, 4$ corresponding to code rates of $1/2, 2/3, 3/4, 5/6$. This gives the throughput of $3.8, 5.7, 7.6, 11.4$ Gpbs corresponding to code rates of $1/2, 2/3, 3/4, 5/6$, respectively. The longer frames, $F_L = 1296$ and $F_L = 1944$, have the same throughput since they are processed in two and three cycles, respectively. The estimated power-consumption of the synthesized decoder is

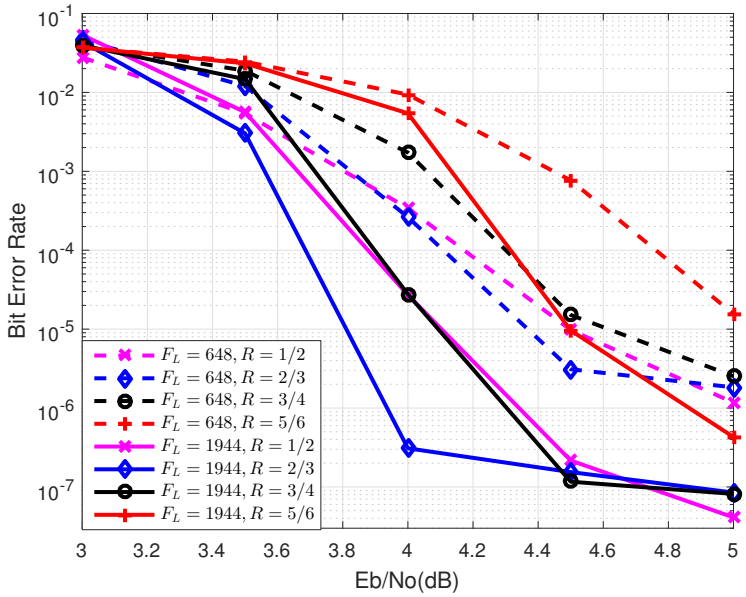

Figure 5.7: BER performance of the synthesized decoder for $F_L = 648$ and $F_L = 1944$, with corresponding code rates.

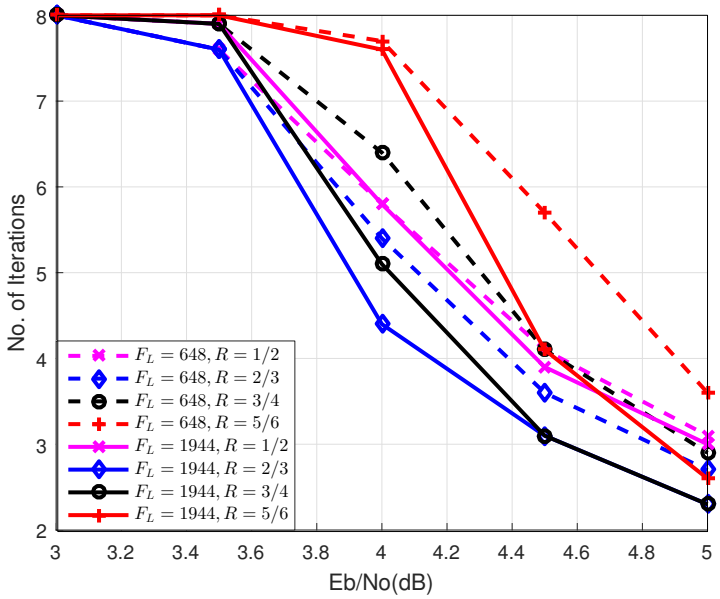

Figure 5.8: Iterations count corresponding to Fig. 5.7.

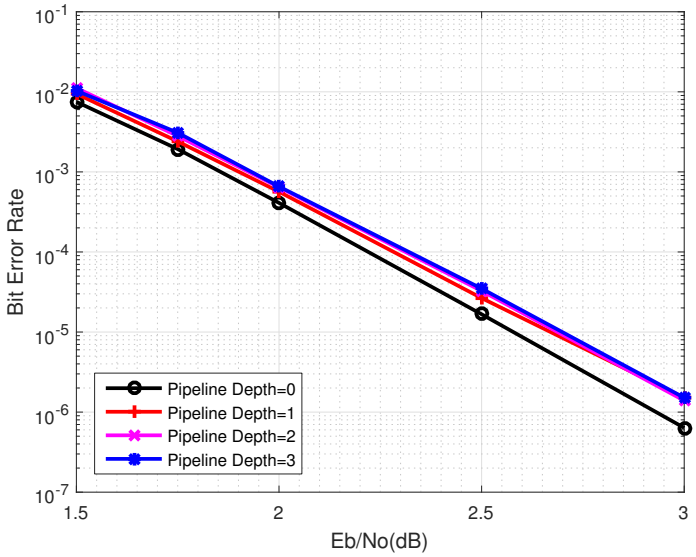

Figure 5.9: BER comparison of pipeline depths for floating-point implementation of rate 1/2, IEEE 802.11n/ac LDPC codes.

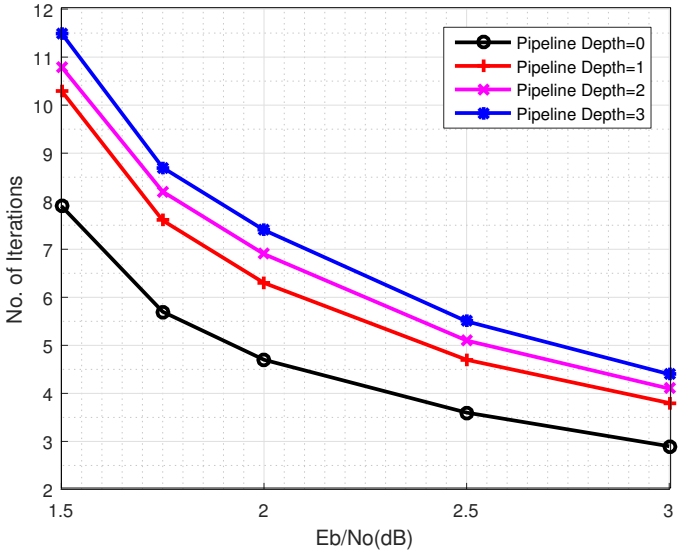

Figure 5.10: Comparison of iteration count corresponding to Fig. 5.9.

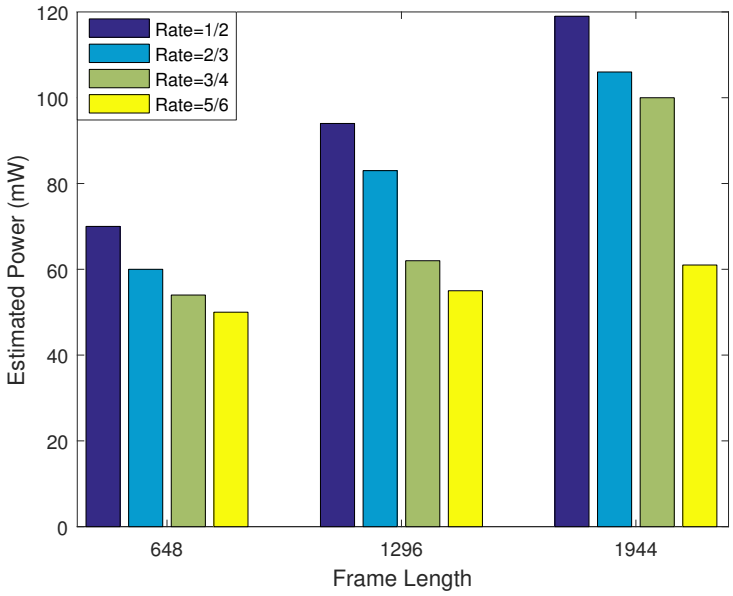

Figure 5.11: Estimated power-consumption of the synthesized decoder for code rates and frame lengths of WiFi LDPC codes*.

* The power is estimated at $BER = 10^{-5}$, corresponding to iterations count values shown in Fig. 5.8. The switching activity for the power estimation is generated by applying test-vectors of each code-rate to the synthesized netlist.

shown in Fig. 5.11. The higher code-rates require a lower number of cycles to complete a decoding iteration, and consequently, their power consumption is reduced. Table 5.7 compares the key parameters of the proposed decoder with the state-of-the-art decoders. It is evident from the comparison that the implemented decoder achieves the highest throughput/area efficiency while consuming the least energy/bit, compared with the state-of-the-art decoders.

5.8 Multi-Core Extension

By considering the above synthesis results of the WiFi decoder, it is evident that more than fifty percent area of the decoder is taken by the memory modules of the decoder, as shown in Table 5.2. The memory modules, however, are designed to store the messages as per the largest frame length of a QC-LDPC code. This points toward an opportunity that the processing blocks of the decoder could be increased without a linear increase of the area-footprint of the overall decoder. The throughput of the decoder, therefore, could significantly be increased by replicating the processing elements of the decoder to process the largest frame of an IEEE 802.11n/ac/ax LDPC codes, which is 1944 , in contrast to the single-core (SC) design, which process the smallest frame of the IEEE 802.11n/ac/ax LDPC

Table 5.1: Comparison of Proposed Decoder with State-of-the-Art

	This work	[45]	[23]	[46]	[47]
Technology	**40 nm**	90 nm	90 nm	90 nm	45 nm
Quant. Bits	**4,7**	4	5,6	8	6
Clock Freq. (MHz)	**562**	555	336	250	815
Throughput (Gpbs)	$\mathbf{3.8-11.4}$	4.5	1.71-5.1	0.67	0.38-3.0
Area ($\mathbf{mm^2}$)	**0.71**	4.9	5.2	3.67	0.81
Power (mW)	$\mathbf{50-119}$	523	450	171	-
S. Through. (Gpbs)	$\mathbf{3.8-11.4}$	10.1 [a]	3.8-11.4 [a]	1.5 [a]	0.43-3.36 [b]
S. Area ($\mathbf{mm^2}$)	**0.85** *	0.98 [c]	1.0 [c]	0.72 [c]	0.64 [d]
S. Power (mW)	$\mathbf{60-142}$ *	209 [e]	180 [e]	68 [e]	-
$\frac{\mathbf{S.Through.}}{\mathbf{S.Area}}(\frac{\mathbf{Gbps}}{\mathbf{mm^2}})$	$\mathbf{4.5-13.4}$	10.3	3.8-11.4	2.1	0.67-5.25
$\mathbf{En.Eff.} = \frac{\mathbf{S.Power}}{\mathbf{S.Through.}}(\frac{\mathbf{pJ}}{\mathbf{bit}})$	**12.5**	20.7	15.8	45	-

[a] Throughput is scaled by 2.25 to account for the 90nm technology.
[b] Throughput is scaled by 1.12 to account for the 45nm technology.
[c] Area is scaled by 0.2 to account for the 90nm technology.
[d] Area is scaled by 0.8 to account for the 45nm technology.
[e] Power is scaled by 0.4 to account for the 90nm technology.
* Area and power are scaled by 1.2 to account for the synthesis results.

Table 5.2: Area comparison of modules of WiFi decoder

	Absolute Area	Percentage Area
LLR Mem	$0.14\,mm^2$	20
CN Mem	$0.27\,mm^2$	37
Process. Elements	$0.3\,mm^2$	43
Total	$0.71\,mm^2$	100

codes. The replication does not cause a proportional increase in the decoder area. This, however, raises an issue of under-utilization of the processing elements for smaller frame lengths of the LDPC codes.

Based on the above observation, this work proposes the replication of the processing elements of the decoder to process the largest frame length of the LDPC codes in a single cycle, and introduces frame-reconfigurability option for the processing elements, so that multiple smaller frames could be processed, in parallel, to address the issue of under utilization. This work, therefore, implements a multi-core (MC) decoder architecture which replicates the processing elements of the decoder by 3 , and makes them reconfigurable, so that multiple smaller frames could be processed, in parallel, when the largest frame is not processed. Since the architecture is design for the frame length of 1944 , so after adding the reconfigurability option, three smallest frames of length 648 , or a single frame of length 1296 along with a frame of length 648 , could be processed in parallel.

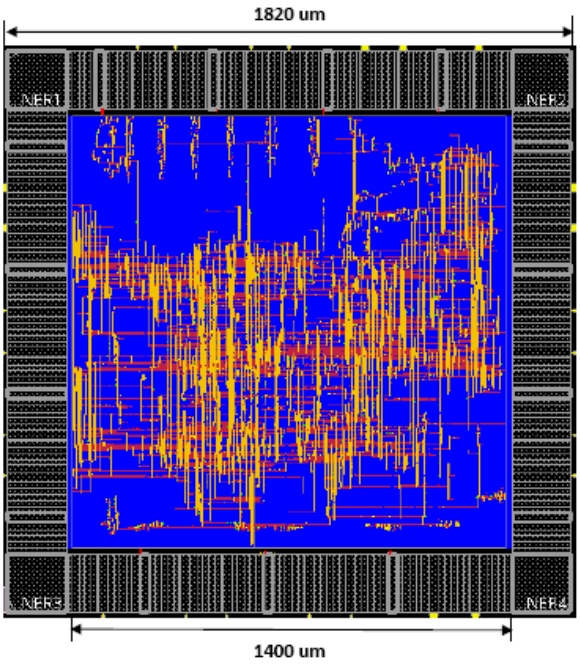

Figure 5.12: The decoder core occupies an area of $1.4 \times 1.4 = 1.96mm^2$. The total die size is $1.82 \times 1.82 = 3.3mm^2$.

Table 5.3: Comparison of Proposed Decoder with State-of-the-Art

	MC	**SC**	[45]	[23]	[46]	[47]
Technology	**40 nm**	**40 nm**	90 nm	90 nm	90 nm	45 nm
Clk Freq. (MHz)	**250**	**562**	555	336	250	815
Through. (Gpbs)	**5.1 – 15.2**	**3.8 – 11.4**	4.5	1.71-5.1	0.67	3.0
Area (mm^2)	**1.96**	**0.71**	4.9	5.2	3.67	0.81
Power (mW)	**75**	**119**	523	450	171	-
S. Thr. (Gpbs)	**5.1 – 15.2**	**3.8 – 11.4**	10.1 [a]	3.8-11.4 [a]	1.5 [a]	3.36 [b]
S. Area (mm^2)	**1.96**	**0.85** *	0.98 [c]	1.0 [c]	0.72 [c]	0.64 [d]
S. Power (mW)	**75**	**142** *	209 [e]	180 [e]	68 [e]	-
En.Eff.(pJ/bit)	**5**	**12.5**	20.7	15.8	45	-

[a] Throughput is scaled by 2.25 to account for the 90nm technology.
[b] Throughput is scaled by 1.12 to account for the 45nm technology.
[c] Area is scaled by 0.2 to account for the 90nm technology.
[d] Area is scaled by 0.8 to account for the 45nm technology.
[e] Power is scaled by 0.4 to account for the 90nm technology.
* Area and power are scaled by 1.2 to account for the synthesis results.

5.9 VLSI Implementation Results

The MC decoder for IEEE 802.11na/ac/ax QC-LDPC codes, implementing the proposed replication of processing elements, is synthesized, and placed and routed, using the TSMC 40 nm CMOS process. The chip-layout of the implemented decoder is shown in Fig. 5.12. To further reduce the power consumption than the SC design, the clock frequency of this design is lowered down to $f_{\text{clk}} = 250\,\text{MHz}$, compared to $f_{\text{clk}} = 562\,\text{MHz}$ of the SC decoder. The peak throughput θ is calculated, using (5.5), as:

$$\theta = \frac{1944}{8 \times R} \times 250, \tag{5.6}$$

This gives the throughput of $5.1, 7.6, 10.1, 15.2\,\text{Gpbs}$ corresponding to code rates of $1/2, 2/3, 3/4, 5/6$, respectively. Table 5.9 compares the key parameters of the proposed architecture with the state-of-the-art decoders. It is evident from the comparison that the implemented decoder achieves the highest reported throughput while consuming the least energy/bit, compared with the state-of-the-art IEEE 802.11na/ac/ax LDPC decoders.

Chapter 6

Conclusions

This work proposed an implemented the algorithm, architecture, and circuit co-design approach for the development of optimized ASIC accelerator chips for LDPC decoders. Algorithmically, fast-converging LDPC decoding schedule that converges in a reduced number of iterations, compared to existing serial decoding schedules, have been presented and investigated. It has been verified that the proposed schedules achieves fast-decoding performance with reduced computational complexity or consumes lower power to achieve a specific throughput, compared with other schedules in the literature.

The advantages of the proposed schedules have been demonstrated through convergence analysis, throughput analysis, complexity analysis, and BER and FER comparisons using empirical simulations of random, IEEE 802.11ad (WiGig), IEEE 802.11n/ac (WiFi), and IEEE 802.16e (WiMAX) LDPC codes. It has also been shown through complexity analysis that the added cost of the proposed schedules is more addition operations at variable nodes and XOR operations at check nodes, compared to existing serial decoding schedules. An FCMP-based decoder architecture targeting IEEE 802.11ad (WiGig) LDPC codes has also been proposed. The architecture is pipelined and optimized for the processing of two frames simultaneously. VLSI synthesis results using the TSMC 40 nm CMOS have revealed that the synthesized decoder achieves an energy efficiency of 8.6 pJ/bit while operating at 8.4 Gbps, which is the best-reported energy-efficiency of a WiGig LDPC decoder.

Optimization techniques are then presented for efficient VLSI implementation of high-throughput IEEE 802.11n/ac/ax LDPC. These optimizations target decoder area and power reduction while still maintaining Gbps processing throughput. The proposed optimizations are applied to design two variants of a fully-pipelined IEEE 802.11n/ac/ax LDPC decoder. The two variants are; single-core design and multi-core design. Both the designs are synthesized using **40 nm** CMOS technology. The single-core decoder attains a peak throughput of **11.4 Gbps** while occupying an area of **0.71 mm^2**. This decoder achieves a troughput/area efficiency of **13.4 Gbps/mm^2** while consuming

12.5 pJ/bit of energy, and thus surpasses the best-reported corresponding efficiencies of 11.4 Gbps/mm^2 and 15.8 pJ/bit, respectively, for IEEE 802.11n/ac/ax LDPC decoders. The multi-core design replicates the processing elements in comparison with the single-core decoder, and achieves a peak throughput of **15.2 Gbps** while operating at clock-frequency of 250 MHz. The achieved throughput and the corresponding energy-efficiency of **5 pJ/bit** are the best reported in the literature for an IEEE 802.11n/ac/ax LDPC decoder.

Appendix A

Abbreviations

ASIC	Application Specific Integrated Circuit
CMOS	Column Message Passing
CMP	Column Message Passing
CNU	Check Node Unit
D-SVNF	Dynamic Silent Variable Node Free
EXIT	Extrinsic Information Transfer
FCMP	Fast Column Message Passing
ICRMP	Interlaced Column Row Message Passing
IFS	Informed Fixed Scheduling
IDS	Informed Dynamic Scheduling
LDPC	Low Density Parity Check
LLR	Log Likelihood Ratio
MC	Multi Core
NWRBP	Node-wise RBP
QC-LDPC	Quasi Cyclic LDPC
RBP	Residual Belief Propagation
RMP	Row Message Passing
SC	Single Core
SVNF	Silent Variable Node Free
SMP	Standard Message Passing
SPCM	Sparse Parity Check Matrix
TSDS	Tabu Search Based Dynamic Scheduling
TSMC	Taiwan Semiconductor Manufacturing Company
VLSI	Very Large Scale Integeration

Bibliography

[1] D. MacKay and R. Neal, "Near shannon limit performance of low-density parity-check codes," *Electron. Lett.*, vol. 33, pp. 457–458, Mar. 1997.

[2] C. Berrou *et al.*, "Near Shannon limit error-correcting coding and decoding: Turbo codes," in *IEEE Workshop on Signal Processing Systems (SiPS 02)*, pp. 1064–1070, Oct. 2002.

[3] T. Richardson and S. Kudekar, "Design of low-density parity check codes for 5G new radio," *IEEE Commun. Mag.*, vol. 56, pp. 28–34, Mar. 2018.

[4] "Multiplexing and channel coding," Technical Specification: 3GPP TS 38.212 (v 15.3.0) Release 15, Oct. 2018.

[5] "IEEE standard for local and metropolitan area networks – part 11: Wireless LAN (MAC) and physical layer (PHY)," no. 802.11, 2012.

[6] "IEEE draft standard for information technology-wireless LANs – part 21: mmWave PHY specification," no. 802.11ad, May 2010.

[7] R. G. Gallager, *Low-Density Parity-Check Codes.* Cambridge, MA: MIT Press, 1963.

[8] D. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inf. Theory*, vol. 45, pp. 399–431, Mar. 1999.

[9] M. M. Mansour and N. R. Shanbhag, "High-throughput LDPC decoders," *IEEE Trans. VLSI Syst.*, vol. 11, pp. 976–996, Dec. 2003.

[10] J. Zhang and M. Fossorier, "Shuffled belief propagation decoding," in *Proc. Asilomar Conf. Signals, Syst., Comput. (Asilomar)*, vol. 1, pp. 8–15, Nov. 2002.

[11] M. M. Mansour and N. R. Shanbhag, "Memory-efficient turbo decoder architectures for LDPC coding," in *Proc. IEEE Int. Workshop on Signal Process. Systems (SiPS)*, pp. 159–164, 2002.

[12] M. M. Mansour, "A turbo-decoding message-passing algorithm for sparse parity-check matrix codes," *IEEE Trans. Signal Process.*, vol. 54, pp. 4376–4392, Nov. 2006.

[13] H. Kfir and I. Kanter., "Parallel versus sequential updating for belief propagation decoding," *Physica A: Statistical Mechanics and its Applications*, vol. 330, pp. 259–270, Dec. 2003.

[14] H.-C. Lee and Y.-L. Ueng, "LDPC decoding scheduling for faster convergence and lower error floor," *IEEE Trans. Commun.*, vol. 62, pp. 3104–3113, Sept. 2014.

[15] C. A. Aslam *et al.*, "Improving the belief-propagation convergence of irregular LDPC codes using column-weight based scheduling," *IEEE Commun. Lett.*, vol. 19, pp. 1283–1286, Aug. 2015.

[16] C. A. Aslam *et al.*, "Informed fixed scheduling for faster convergence of shuffled belief-propagation decoding," *IEEE Commun. Lett.*, vol. 21, pp. 32–35, Jan. 2017.

[17] A. I. V. Casado, M. Griot, and R. D.Wesel, "Informed dynamic scheduling for belief-propagation decoding of LDPC codes," in *Proc. IEEE Int. Conf. Commun. (ICC)*, pp. 932–937, June 2007.

[18] H. C. Lee *et al.*, "Two informed dynamic scheduling strategies for iterative LDPC decoders," *IEEE Trans. Commun.*, vol. 61, pp. 886–896, Mar. 2013.

[19] C. A. Aslam *et al.*, "Low-complexity belief-propagation decoding via dynamic silent-variable-node-free scheduling," *IEEE Commun. Lett.*, vol. 21, pp. 28–31, Jan. 2017.

[20] X. Liu *et al.*, "Dynamic scheduling decoding of LDPC codes based on tabu search," *IEEE Trans. Commun.*, vol. 65, pp. 4612–46211, Nov. 2017.

[21] X. Liu *et al.*, "Improved decoding algorithms of LDPC codes based on reliability metrics of variable nodes," *IEEE Access*, vol. 7, pp. 35769–35778, Mar. 2019.

[22] M. Fossorier, "Quasi-cyclic low-density parity-check codes from circulant permutation matrices," *IEEE Trans. Inf. Theory*, vol. 50, pp. 1788–1793, Aug. 2004.

[23] S. Kumawat *et al.*, "High-throughput LDPC-decoder architecture using efficient comparison techniques and dynamic multi-frame processing schedule," *IEEE Trans. Circuits Syst. I*, vol. 62, pp. 1421–1430, May 2015.

[24] J. Chen *et al.*, "Reduced-complexity decoding of LDPC codes," *IEEE Trans. Commun.*, vol. 53, pp. 1232–1232, July 2005.

[25] S. Usman, M. M. Mansour, and A. Chehab, "A multi-Gbps fully pipelined layered decoder for IEEE 802.11n/ac/ax LDPC codes," in *Proc. IEEE Symposium on VLSI (ISVLSI)*, pp. 194–199, July 2017.

[26] M. Weiner and et al., "LDPC decoder architecture for high-data rate personal-area networks," in *Proc. IEEE Int. Sympos. on Circuits and Systems (ISCAS)*, pp. 1784–1787, July 2011.

[27] S. Usman and M. M. Mansour, "An optimized VLSI implementation of an IEEE 802.11n/ac/ax LDPC decoder," in *Proc. IEEE Int. Sympos. on Circuits and Systems (ISCAS)*, May 2020.

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

[29] S. Chung, T. Richardson, and R. Urbanke, "Analysis of sum-product decoding of low-density parity-check codes using a Gaussian approximation," *IEEE Trans. Inf. Theory*, vol. 47, pp. 657–670, Feb. 2001.

[30] Z. Cui and et al., "Efficient decoder design for high-throughput LDPC decoding," in *Proc. IEEE Asia Pacific Conf. on Circuits and Syst.*, pp. 1640–1643, Nov. 2008.

[31] J. Lin and et al., "An improved min-sum based column-layered decoding algorithm for LDPC codes," in *Proc. IEEE Int. Workshop on Signal Process. Systems (SiPS)*, pp. 238–242, 2009.

[32] Z. Cui, Z. Wang, and X. Zhang, "Reduced-complexity column-layered decoding and implementation for LDPC codes," *IEEE Trans. Commun.*, vol. 5, pp. 2177–2186, Oct. 2011.

[33] X. Zhang *et al.*, "Column-layered message-passing LDPC decoder," *U.S. Patent US20180062666A1*, Mar. 2018.

[34] M. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *IEEE Trans. Commun.*, vol. 47, pp. 673–680, May 1999.

[35] S. t. Brink, "Convergence behavior of iteratively decoded parallel concatenated codes," *IEEE Trans. Commun.*, vol. 49, pp. 1727–1737, Oct. 2001.

[36] S. t. Brink *et al.*, "Design of low-density parity-check codes for modulation and detection," *IEEE Trans. Commun.*, vol. 52, pp. 670–678, Apr. 2004.

[37] M. Tchler and J. Hagenauer, "EXIT charts of irregular codes," in *Proc. 2002 Conf. Information Sciences and Systems*, pp. 748–753, Mar. 2002.

[38] S. Kim and et al., "Analysis of complexity and convergence speed of sequential schedules for decoding LDPC codes," in *Proc. Int. Sympos. on Information Theory and its Applications (ISITA)*, pp. 629–634, Nov. 2006.

[39] M. Milicevic and P. G. Gulak, "A multi-Gb/s frame-interleaved LDPC decoder with path-unrolled message passing in 28-nm CMOS," *IEEE Trans. VLSI Syst.*, vol. 26, pp. 1908–1921, Oct. 2018.

[40] K. Okada *et al.*, "Full four-channel 6.3-Gb/s 60-GHz CMOS transceiver with low-power analog and digital baseband circuitry," *IEEE J. Solid-State Circuits*, vol. 48, pp. 46–65, Jan. 2013.

[41] Y. S. Park *et al.*, "Low-power high-throughput LDPC decoder using non-refresh embedded DRAM," *IEEE J. Solid-State Circuits*, vol. 49, pp. 783–794, Mar. 2014.

[42] H. Motozuka and et al., "A 6.16 Gb/s 4.7 pJ/bit/iteration LDPC decoder for IEEE 802.11ad standard in 40 nm LP-CMOS," in *Proc. IEEE Global Conf. Signal and Inf. Process. (GlobalSIP)*, pp. 1289–1292, Dec. 2015.

[43] M. Weiner and et al., "A scalable 1.5-to-6Gb/s 6.2-to-38.1mw LDPC decoder for 60GHz wireless networks in 28nm UTBB FDSOI," in *ISSCC Dig. Tech. Papers*, pp. 464–465, Feb. 2014.

[44] T. T. B. Nguyen and H. Lee, "Low-complexity multi-mode multi-way split-row layered LDPC decoder for gigabit wireless communications," *Integration*, vol. 65, pp. 189–200, 2019.

[45] I. Tsatsaragkos and V. Paliouras, "A reconfigurable LDPC decoder optimized for 802.11n/ac applications," *IEEE Trans. VLSI Syst.*, vol. 26, pp. 182–195, Jan. 2018.

[46] C. Yu and et al., "Improvement on a block-serial fully-overlapped QC-LDPC decoder for IEEE 802.11n," in *Proc. IEEE Int. Conf. Consum. Electron. (ICCE)*, pp. 446–447, Jan. 2014.

[47] Y. Sun and et al., "Multi-layer parallel decoding algorithm and VLSI architecture for quasi-cyclic LDPC codes," in *Proc. IEEE Int. Sympos. on Circuits and Systems (ISCAS)*, pp. 1776–1779, May 2011.