A Novel Architecture for Scalable, High throughput, Multi-standard LDPC Decoder

***Abstract*—This paper presents a a novel bottom up parallel approach for implementing high throughput , scalable multi-standard, layered LDPC decoding architecture. Proposed solution includes three elements of novelty i.e. 1) A parallel Min Sum check node based on ”Tree way” approach 2) Channel Memory organization scheme at block level to support parallel access for proposed check node 3) Flexibility in terms of supporting multiple codes, block lengths, code rates and parallelisms to realize fully scalable LDPC decoding architecture. The proposed decoder IP supports different LDPC codes adopted in WiMAX (802.16e) and WiFi (802.11n). Synthesis results are based on $130-nm$ Standard Cell ASIC technology.**

***Index Terms*—Low Density Parity Check codes, Min Sum, Layered Decoding, Tree way approach, Flexible architectures**

I. Introduction

Present research in the field of Forward Error Correction (FEC) is aimed at finding the best possible error correcting codes which could allow high throughput decoding with efficient VLSI implementation meeting area, power and speed constraints. Low Density Parity Check (LDPC) Codes [1], a class of linear block codes have gained huge attention in wireless communication domain. Near Shannon limit error correcting capabilities, low error floor, affordable complexity and intrinsic parallelism make LDPC codes an eligible candidate for a number of wireless standards [2]. LDPC codes have been proposed as an optional channel coding scheme for upcoming WiMax 802.16e [3] and WiFi 802.11n [4] standards. However, VLSI implementation of LDPC decoders is a challenging task. Initial designs [5] reported severe layout congestion problems due to pseudo-random message exchange between the computing kernels. But structured LDPC codes greatly simplified the interconnect complexity [6]. Code-decoder co-design is the most suitable approach to ease the implementation. This leads to adoption of architecture-aware (AA) LDPC codes by almost all upcoming standards.

Traditional parallel architectures provide maximum throughput but result in huge complexity [5], partially (block-level) parallel have sufficient throughput and affordable complexity but limited to specific structured codes only [7]–[10], while fully serial architectures have minimum complexity and throughput. State of the art mainly relies on multiprocessor implementations to achieve flexible, high throughput iterative decoding. In [9], a highly flexible LDPC decoder able to process all specified codes has been proposed, but it consisted of serial check node and interconnection network, which limits the achievable throughput to large extent.

Realizing high throughput decoders e.g. for wireless backbone networks (supporting data rates up to few hundred Mbps) either asks for massive parallelism or increasing the clock frequency which results in significant area and power overhead. However, incorporating parallelism at check node level is still not fully explored and can bring significant increase in throughput with affordable complexity. In this regard, the key contributions to this work i.e. a novel ”Tree-way” approach for parallel Min Sum check node is presented in its generic form supporting check node degrees up to $d_c = 32$, which is sufficient to cover a large number of LDPC codes. Starting from ”Tree way” parallel check node, a bottom up approach for complete decoder implementation is proposed. A modular channel memory design technique at block and sub-block level is presented which supports parallel access for proposed ”Tree-way” check node and allows flexibility in terms of codes, rates, block lengths and parallelisms. The proposed architecture is exploited to design a fully scalable LDPC decoder for WiMax and WiFi standards.

II. Basics of LDPC Decoding

A binary LDPC code is represented by a sparse parity check matrix *H* with dimensions $M \times N$ such that each element $h_{mn} \, \epsilon \, \{0,1\}$. *N* is the total length of code word and *M* is the number of parity bits. Each row $H_i (1 \leq i \leq M)$ introduces one parity check constraint on input data vector $x = \{x_1, x_2, \cdots, x_n\}$ i.e.

$$H_i.x^T \; = \; 0 \, mod \, 2$$

The complete *H* matrix can best be described by a Tanner graph [11], a graphical representation of the associations between code bits and parity checks. Each row of this *H* matrix corresponds to a Check Node (CN) while each column corresponds to a Variable Node (VN) in Tanner graph. An *(N, K)* LDPC code is mapped on Tanner Graph such that there are *N* Variable (or bit) Nodes (VNs) and $M = N - K$ Check Nodes (CNs). An edge e_{ji} on the Tanner Graph connects a VN_j with CN_i only if the corresponding element h_{ij} is a '1' in *H*. An LDPC code is called regular if the node degree (i.e. the number of incoming edges) is constant for all nodes, otherwise irregular. Irregular LDPC codes have better communication performance than regular LDPC codes.

LDPC decoding is done using well known Belief Propagation (BP), Sum Product (SP) or Message Passing (MP) algorithms [12], in an iterative way which involves bilateral exchange of Log-Likelihood Ratio (LLR) messages between VNs and CNs. The aim of BP algorithm is to compute the a-posteriori probability (APP) that a given bit in the transmitted codeword $c = [c_0, c_1, \cdots, c_{N-1}]$ equals 1, given the received word $y = [y_0, y_1, \cdots, y_{N-1}]$.

The Min Sum (MS) Algorithm is an area efficient, sub optimal approximation to the SPA. It is also an iterative, soft decoding algorithm. Let α_{ij}^{n} represents the message sent from variable node VN_j to check node CN_i in n^{th} iteration, β_{ij}^{n} represents the message sent from check node CN_i to variable node VN_j in n^{th} iteration, $\mathcal{M}(j) = \{i : H_{ij} = 1\}$ i.e. set of parity checks in which variable node VN_j participates, $\mathcal{N}(i) = \{j : H_{ij} = 1\}$ i.e. set of variable nodes that participate in parity check i, $\mathcal{M}(j)\backslash i$: the set $\mathcal{M}(j)$ with check i excluded, $\mathcal{N}(i)\backslash j$: the set $\mathcal{N}(i)$ with variable j excluded. The conventional Min-Sum algorithm formulation is described as follows

1) Initialization
For $j \,\epsilon\, \{1, \cdots, N\}$

$$\alpha_{i,j}^{0} = ln\frac{P(VN_j = 0|y_j)}{P(VN_j = 1|y_j)} = \frac{2y_j}{\sigma^2} \quad (1)$$

where y_j is received data bit at position j and σ^2 is the variance of channel noise. Practically, $\alpha_{i,j}^{0} = y_j$

2) Horizontal Scan (Check Node Update Rule)
For all check nodes CN_i , $i \,\epsilon\, \{1, \cdots, M\}$ do

$$\beta_{i,j}^{n} = \prod_{j' \epsilon \mathcal{N}(i)\backslash j} sign(\alpha_{i,j'}^{n-1}) \min_{j' \,\epsilon\, \mathcal{N}(i)\backslash j} \{|\alpha_{i,j'}^{n-1}|\} \quad (2)$$

3) Vertical Scan (Variable Node Update Rule)
For all Variable nodes VN_j , $j \,\epsilon\, \{1, \cdots, N\}$ do

$$\alpha_{i,j}^{n} = \alpha_{i,j}^{0} + \sum_{i' \epsilon \mathcal{M}(j)\backslash i} \beta_{i',j}^{n} \quad (3)$$

4) Decoding
For each bit, compute its posteriori LLR

$$\alpha_{j}^{n} = \alpha_{j}^{0} + \sum_{i' \epsilon \mathcal{M}(j)} \beta_{i',j}^{n} \quad (4)$$

Estimated codeword is $\hat{C} = (\hat{c}_1, \hat{c}_2, \cdots, \hat{c}_N)$ where element $\hat{c}_j$ is calculated as

$$\hat{c}_j = \begin{cases} 0 & \text{if } \alpha_j^n > 0 \\ 1 & \text{elsewhere} \end{cases} \quad (5)$$

If $H(\hat{C})^T = 0$ then stop and output $\hat{C}$ as decoded codeword. Modifying the Variable node update rule (3) as

$$\alpha_{i,j}^{n} = \alpha_{j}^{n} - \beta_{i,j}^{n} \quad (6)$$

We can merge horizontal and vertical scans in to a single horizontal scan where the check node messages $\beta_{i,j}^{n}$ are computed from $\alpha_{j}^{(n-1)}$ and $\beta_{i,j}^{(n-1)}$. This technique is called "Layered Decoding" [13], or "Turbo Decoding Message Passing (TDMP)" [14]. Formally, the algorithm for layered decoding Min Sum can be described as

1) Initialization : $\beta_{i,j}^{0} = 0$

2) Horizontal Scan (Check Node Update Rule) :

$$\alpha_{j}^{n} = \alpha_{j}^{0}$$

$$\beta_{i,j}^{n} = sgn\{\beta_{i,j}^{n}\} \times |\beta_{i,j}^{n}| \quad (7)$$

$$sgn\{\beta_{i,j}^{n}\} = \prod_{j' \epsilon \mathcal{N}(i)\backslash j} sgn\{\alpha_{j}^{(n-1)} - \beta_{i'j}^{(n-1)}\} \quad (8)$$

$$|\beta_{i,j}^{n}| = \min_{j' \epsilon \mathcal{N}(i)\backslash j} |\alpha_{j}^{(n-1)} - \beta_{i,j'}^{(n-1)}| \quad (9)$$

$$\alpha_{j}^{n} = \alpha_{j}^{n} + \beta_{i,j}^{n} \quad (10)$$

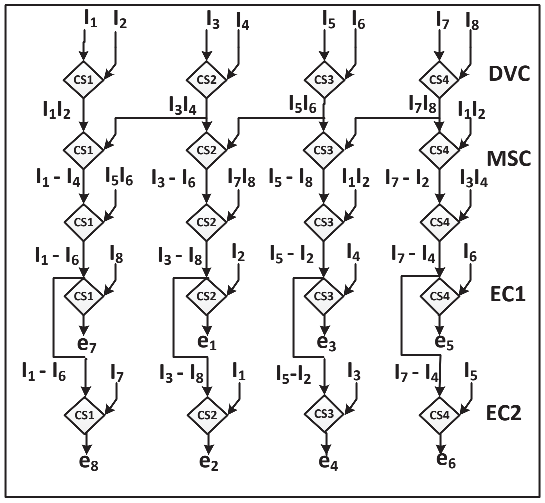
3) Decoding : $\hat{c}_j^n = 0$, if $\alpha_j^n > 0$, $\hat{c}_j^n = 1$ otherwise

Layered decoding is based on the principle of using intermediate results directly in the next sub-iteration so that updated information is available to check node. This results in 50% decrease in number of iterations to meet a certain BER (equivalent to 2x increase in throughput) and significant memory savings as compared to standard two phase message passing.

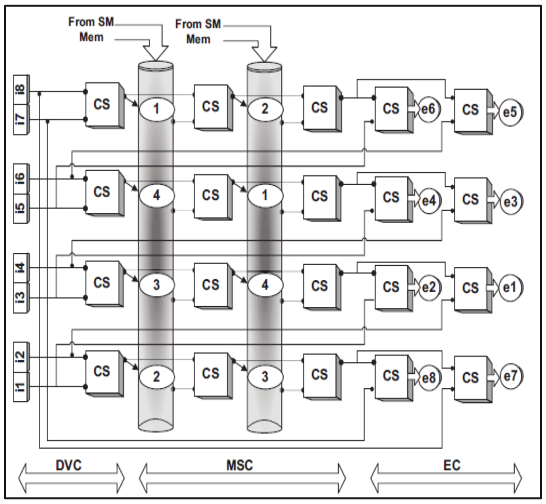
III. Check node parallelism : A Tree-Way Approach

Close examination of check node operation described in previous section reveals a fact that in Min Sum decoding, out of all LLRs of a CN only two magnitudes are of interest i.e. minimum and the second minimum. The state of the art for Min Sum decoding of LDPC for many applications consists of serial check nodes incorporating on the fly calculation of running minimum, second minimum and sign product [15]. The interconnection network is also serial providing one input to all check nodes in each clock cycle. However, decreasing the check node and interconnection network latencies is mandatory to realize high throughput LDPC decoders.

Incorporation of parallel processing inside the check node is a key contribution of this work whereby a novel "Tree-way" check node receives all variable to check node messages in parallel and writes back the updated extrinsic information simultaneously to all connected variable nodes. This significantly decreases the number of clock cycles required for check node update and helps in achieving high throughput with smaller parallelism. Figure 1a is a particular example of this scheme for $d_c = 8$. Each VN i.e. $(I_1, I_2, \cdots, I_8)$ is represented as a leaf node and tree is traversed performing min calculation at branch nodes until all VN extrinsic values are derived at the root nodes $(e_1, e_2, \cdots, e_8)$. One of the key contribution of this paper is that for a parallel check node architecture we generalize the tree network connectivity and the data flow for any value of d_c up to 32 and present a fairly simple control mechanism for it. For the sake of architecture uniformity odd d_c values are considered as their even counterpart with extra VN intrinsic value initialized at $+\infty$ (i.e $d_c' = d_c$ if d_c is even; else $d_c' = d_c + 1$). Figure 1b shows the different stages of VN extrinsic calculation for proposed "Tree-Way" scheme (magnitude only).



(a) Tree for calculation of VN extrinsics



(b) Generalized data flow

Fig. 1. Proposed Tree way scheme for Parallel Check Node

Direct VN Comparison (DVC) stage : As seen in Fig. 1b, for $d_c = 8$, the intrinsic values $(I_1, I_2, \cdots, I_8)$ are fed parallel to 4 CS (compare select) units. For all values of d_c there is only one direct comparison stage. The outputs of DVC and each subsequent stage are passed on to next stage through a local shuffling network known as ACS network as well as are stored in switch matrix (SM) memory for use in later stages.

Multiple Shuffled Comparison (MSC) Stage : The shuffle network implements a circular shifting permutation. The rotational shift depends on d_c and the sub stage of the shuffled comparison stage. There are multiple shuffled stages depending on d_c and equal to N_{cc}, where N_{cc} is the required number of clock cycles for check node update. For different values of dc Table I provides the information on N_{cc} values and shift associated with each shuffled stage.

TABLE I
CLOCK CYCLE REQUIREMENT AND SHIFT PERMUTATION

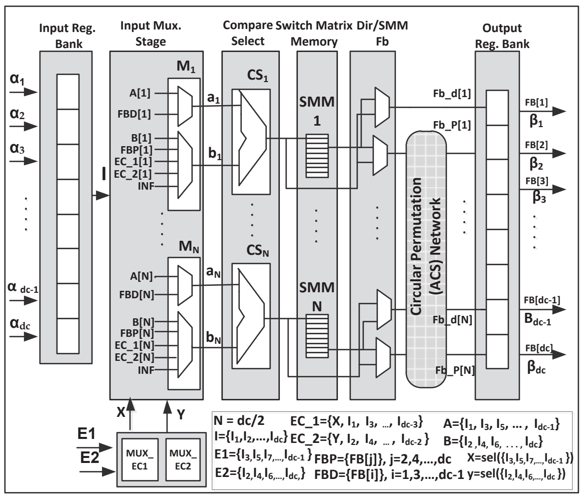
d_c	N_{cc}	Permutation	d_c	N_{cc}	Permutation
5,6	4	1	19,20	7	1,2,4,8
7,8	5	1,2	21,22	7	1,2,4,8
9,10	5	1,2	23,24	8	1,2,4,8,10
11,12	6	1,2,4	25,26	7	1,2,4,8
13,14	6	1,2,4	27,28	8	1,2,4,8,12
15,16	7	1,2,4,6	29,30	8	1,2,4,8,12
17,18	6	1,2,4	31,32	9	1,2,4,8,12,14

As can be seen from Fig. 1b, the input to the shuffle network is either the output from the immediate previous stage or output of a much earlier stage stored in the SM memories. Irrespective of the input source of the shuffle network, the shift associated with a stage is fixed.

Extrinsic Calculation (EC) Stage : The last two stages for any value of d_c are extrinsic calculation stage. As seen in figure 1b input to these stages is the output from the last MSC stage and the shifted VN intrinsic values. This shift is circular over d_c and equal to 1 and 2 for EC stage 1 and 2 respectively.

IV. TREE WAY PROCESSING ELEMENT ARCHITECTURE

Figure 2a shows the generic data path architecture for "Tree-way" processing element (magnitude). The α_n are the incoming LLR values, while β_n are the updated outgoing LLRs values where $(n = 1, 2, \cdots, d_c)$ and d_c is the check node degree. The first block i.e. Input Reg Bank consists of d_c parallel registers to store α_n values which are input to the DVC and EC(1,2) stages. The next stage is the Input Mux Stage which consists of $N = dc/2$ multiplexer units. Each unit M_i consists of a mux2 and mux5. For compare select unit $CS_i \;:\; i = 1, 2, \cdots, N$, mux2 selects the first input (among α_n $(n \;:\; n + 1 \bmod 2 \;=\; 0)$ and FBD[i]) while mux5 selects the second input (among α_n $(n : n \bmod 2 \;=\; 0)$, FBP[i], intrinsic information contained in vectors EC_[j], EC_2[j] and infinity). The output of each compare select unit is stored in its corresponding SM memory. There are $dc/2$ memories and size of each memory is equal to number of stages (or Ncc) which depends on dc. For retrieval of outputs stored in SM memory from previous stages, a simple mechanism is formulated. Output values at each stage are assigned a binary label e.g. label 10 represents DVC stage output, while labels 100 and 1000 represent first two MSC stage outputs and so on. For a given stage, the address to be accessed in SM memory depends on d_c value and can be derived using a relatively simple control based on binary representation of the value $d_c - 2$. For example for $d_c = 16$, corresponding binary representation of $d_c - 2 = 1110$ i.e. $1000 + 100 + 10$, thus address for memory access during the shuffle stage corresponds to values at labels of 100 and 10. The inputs to ACS network come either directly from CS unit, or from SM memory i.e the output of much earlier stage. ACS network is implemented to allow for all possible permutations for a particular degree as given in Table I. A $d_c/2 \times d_c/2$ Barrel shifter supports the implementation of all possible permutations


(a) Magnitude Processing

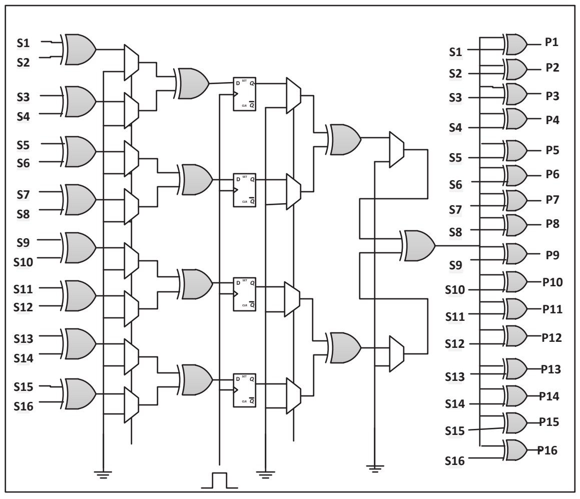

(b) Sign product

Fig. 2. Tree Way Check Node PE : Generalized Data Path

for a degree d_c. To process an Irregular LDPC code with multiple check node degrees, the check node is synthesized for maximum d_c supporting all CN degrees for that code less than and equal to maximum. For example, WiMax $1/2$ rate LDPC has minimum and maximum d_c of 5 and 7 respectively. So the proposed ”Tree-way” check node is synthesized for $d_c = 8$, supporting multiple trees for $d_c = 5$ and $d_c = 7$. During EC_1 and EC_2 the inputs to CS_1 change with degree. Thus the blocks Mux_EC1 and Mux_EC2 consist of pipelined multiplexers to route proper inputs to CS_1. A, B, EC_1, EC_2, FBD, FBP, E1, E2, X and Y are intermediate signal vectors that show connectivity between different stages of data path. Data path reuse is a distinguishing feature of proposed ”Tree-way” processing element, which is not only capable of achieving high clock frequency but also results in considerably small area on chip. The sign product tree is based on the property that product of all sign bits except the sign bit s_i is the exclusive or of all sign bits(including the sign bit s_i) with the sign bit s_i. Figure 2b shows the tree way implementation of flexible sign product block. The intermediate multiplexers are used to select proper inputs to support variable d_c as in case of irregular LDPC codes.

V. Proposed Layered Decoding ”Tree-Way” Check Node Architecute

Layered decoding [16] is compatible with the parallelization of architecture with a limitation that in order to achieve collision free decoding, the parity check matrix is divided in groups such that in each group the maximum column weight must be equal to one [6]. Key idea behind layered decoding is that there is no processing inside variable node. It just acts as a Channel Memory unit to store the soft output estimate of previous sub-iteration. A layered data path architecture has been reported in [10] . The check node is serial with latency equal to d_c and consists of a FIFO, message ram and Check node Functional Unit (CFU). To ensure correct information input to CFU, the corresponding edge messages are immediately subtracted from the message RAM and result is passed to CFU. After processing, the same locations in the message RAMs are updated by newly calculated extrinsic messages. The output of CFU is added to corresponding input passed by a FIFO. Thus the correct a-posteriori information is passed back to channel memory which always holds the updated VN information. This serial Check Node architecture is de-coupled such that the CFU is replaced by our proposed parallel ”Tree-way” processing element. As shown in Fig. 3, single message RAM of $k \times d_c$ words used in serial architecture has been replaced by d_c number of message RAMs each consisting of k words, where k is the number of sub iterations in which a variable node is accessed. The architecture shown is generalized and can be implemented for any value of d_c.

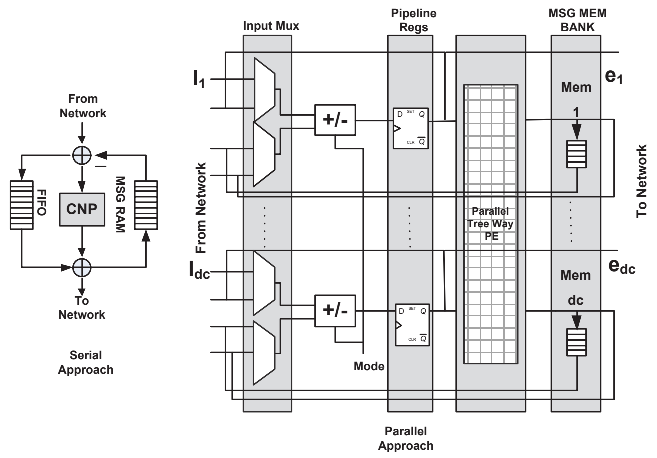
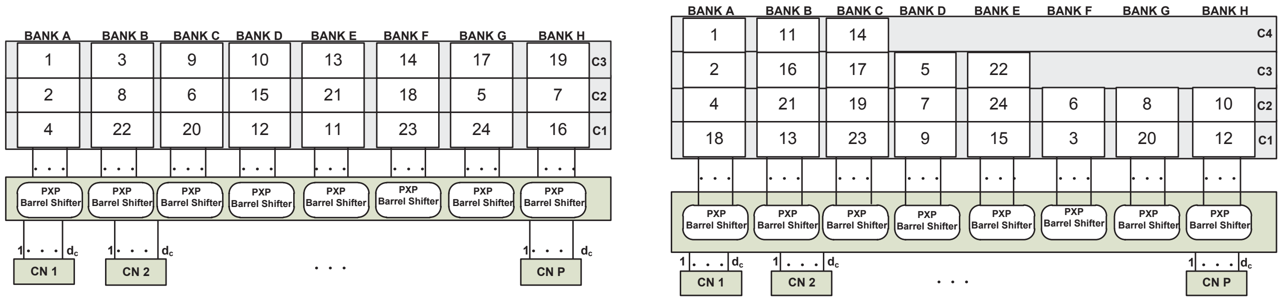


Fig. 3. Serial v.s. Proposed Layered Decoding Check Node



(a) Regular (b) Irregular

Fig. 4. WiMax 1/2 Rate LDPC : Channel Memory Organization

VI. Block level memory organization

As discussed earlier, the state of the art for min sum decoding of LDPC consists of serial check node. The interconnection network is also serial providing one out of dc inputs to all P check nodes per clock cycle, where P is called parallelism. Therefore, d_c clock cycles are required before all incoming messages are received by P check nodes in a sub-iteration. To fully exploit the inherent advantage of proposed check node, a fully parallel network is desired which could move d_c messages to all P check nodes with minimum number of clock cycles.

Up coming wireless standards adopt structured LDPC codes which solve the interconnect, memory overhead and scalability problems associated with LDPC decoders. The H_{BASE} matrix defined as [14] :

$$H_{BASE} = \begin{bmatrix} \Pi_{0,0} & \Pi_{0,1} & \dots & \Pi_{0,N} \\ \Pi_{1,0} & \Pi_{1,1} & \dots & \Pi_{1,N} \\ \vdots & \vdots & \vdots & \vdots \\ \Pi_{M,0} & \Pi_{M,1} & \dots & \Pi_{M,N} \end{bmatrix}$$

is associated to a parity check matrix *H*. It has M_b block rows and N_b block columns. The H_{BASE} is expanded, in order to generate *H* matrix, by replacing each of its entries $\Pi_{i,j}$ with a *Z*-by-*Z* permutation matrix known as circulant, where *Z* is the expansion factor. The circulant can be formed by cyclically shifting right the *Z*-by-*Z* identity matrix. The amount of shift is equal to the value of $\Pi_{i,j}$. The code word length is equal to $N_b \times Z$ and code rate is $(N_b - M_b)/N_b$. In this work a technique is also presented to design memory organization supporting parallel access for proposed "Tree-way" check node. The objective is to break a single memory bank in to a number of parallel memory banks and assign circulants to them such that data dependencies between circulants accessed in the same sub-iteration are minimum. This technique can be described as follows.

To provide parallel access up to arbitrary d_c, the channel memory consists of of d_c memory banks with each bank containing N_b/d_c circulants. Proceed as follows

1) For $i = 1$ to N_b repeat
2) Select a column C_i and highlight all rows corresponding to non negative entries of column C_i
3) Check all other columns in highlighted rows. Columns which have negative entries in all highlighted rows of C_i are data independent from C_i and are not accessed in the same sub-iteration when C_i is accessed and can be placed in same memory bank with C_i.

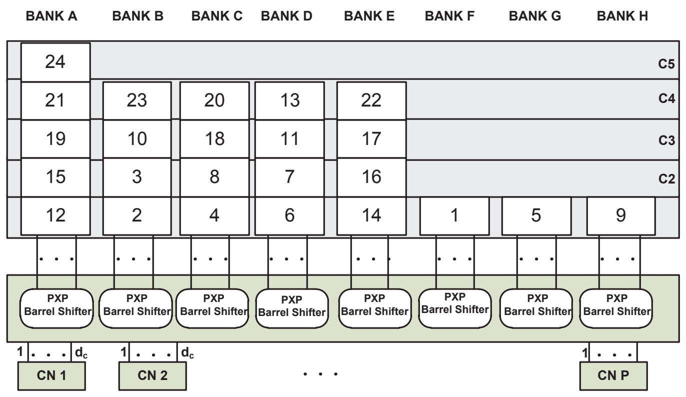
In this way data dependencies between all circulants are noted down in an iterative manner. The N_b circulants are stored in d_c memory banks. Each bank contains P single port memories with $(Z/P) \times (N_b/d_c)$ words each. For verification, this technique is applied on $1/2$ rate H_{BASE} matrices of 802.16e WiMax and 802.11n WiFi. Table II shows specifications for WiMax and WiFi LDPC codes where W_r and W_c the maximum row and column weight of H_{BASE} for a particular code rate.

TABLE II
H_{BASE} PARAMETERS FOR WIMAX AND WIFI LDPC CODES.

Code Rate		**1/2**	**2/3**	**3/4**	**5/6**
H_{BASE} **matrix (WiMax / WiFi)**		12×24	8×24	6×24	4×24
M_b **(WiMax / WiFi)**		12	8	6	4
W_r - W_c	WiMax	$7 - 6$	$11 - 6$	$15 - 6$	$20 - 4$
	WiFi	$8 - 12$	$11 - 8$	$15 - 6$	$22 - 4$

Test Case 1 : 802.16n WiMax Rate $\frac{1}{2}$ ***Code****:* For $1/2$ rate WiMax LDPC case, $W_r = 7$ therefore, the proposed "Tree-way" check node must be synthesized for $d_c = 8$. According to proposed technique, one of the possible memory organizations for $1/2$ rate WiMax LDPC is shown in Fig. 4a where the total LLR memory is partitioned into 8 parallel banks namely A,B,C,D,E,F,G and H. Each bank is further divided into $24/8 = 3$ sub banks, to store three circulants C1,C2 and C3. Each bank consists of *P* single port memories where *P* is parallelism. Size of each single port memory is $(Z/P) \times 3$ words.

Depending upon the structure of H_b matrix, it is not guaranteed that all such data conflicts would be removed. Circulants C1 and C2 in Bank D have data conflicts in sub-iterations 2 and 3, also the circulants C1 and C2 have data conflicts in sub-iterations 9 and 10. So in total, there is penalty of 4



(a) Irregular

Sub iter. #	Circulants Accessed								Clock Cycles	Memory Conflict Penalty
	Memory Banks									
	A	B	C	D	E	F	G	H		
1	C1			C1 / C4	C1	C1	C1	C1	2	1
2	C2	C1	C2	C2	C1	C1	C1	C1	1	0
3	C2	C2		C3	C2	C1	C1	C1	1	0
4		C3	C1		C2 / C3	C1	C1	C1	2	1
5	C2		C3	C3	C3	C1	C1	C1	1	0
6	C3	C2	C1 / C3	C2		C1	C1	C1	2	1
7	C3	C3	C4	C4		C1	C1	C1	1	0
8	C4	C1	C4	C2		C1	C1	C1	1	0
9	C4	C1		C3	C4	C1	C1	C1	1	0
10	C1	C4		C3	C4	C1	C1	C1	1	0
11	C5		C2						1	0
12	C5	C4 / C2	C2	C1		C1	C1	C1	2	1

(b) Access Scheme

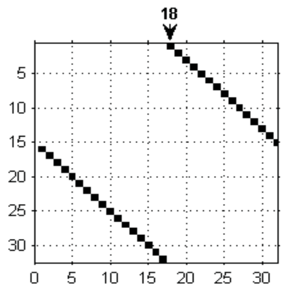
Fig. 5. WiFi 1/2 Rate LDPC : Channel Memory Organization

clock cycles per iteration which is quite tolerable. This scheme can be further improved by a so called irregular memory scheme in which banks A,B,...,H are different in length and each bank stores maximum independent circulants. The over all LLR memory size (in bytes) remains the same, only length of memory bank that changes. The advantage of such a scheme is that all data dependencies are removed thus increasing the throughput. Irregular memory organization for WiMax rate 1/2 is shown in Fig. 4b. The irregular memory access scheme in Fig. 6 shows all data conflicts between the circulants within same memory bank have been removed, which significantly increases the throughput.

Sub iter. #	Circulants Accessed								Clock Cycles	Memory Conflict Penalty
	Memory Banks									
	A	B	C	D	E	F	G	H		
1	C3	C1	C4	C1		C1		C2	1	0
2	C3		C4	C2	C1	C2	C2	C1	1	0
3	C2	C3		C3	C1	C2	C2	C1	1	0
4	C4	C3	C3	C1		C1		C2	1	0
5	C1	C4	C3	C2		C1		C2	1	0
6	C1	C1	C2	C3		C2	C2	C1	1	0
7	C2	C4	C2			C1	C1	C2	1	0
8	C3	C2		C2		C1	C1	C2	1	0
9	C4	C2		C3	C3	C2	C2	C1	1	0
10		C4	C1		C3	C2	C2	C1	1	0
11	C2		C1	C1	C2	C1		C2	1	0
12	C4	C1			C2	C2	C2	C1	1	0

Fig. 6. WiMax Rate 1/2, Access Scheme for Irregular Memory Organization

***Test Case 2 : 802.11n WiFi Rate 1/2 Code**:* As shown in Table II, for $1/2$ rate WiFi LDPC code $W_r = 8$ so the same approach can be applied. The check node is synthesized for $d_c = 8$, the channel memory is divided in to 8 banks. However, the structure of H_{BASE} results in larger number of data conflicts between the circulants therefore, the irregular memory organization results in fewer access penalties. Applying the proposed technique on this H_{BASE}, one possible memory organization with access scheme is shown in Fig. 5. As observed, four clock cycles are wasted per iteration for this case, which is quite affordable as compared to serial approach.



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

Fig. 7. $Z = 32$ circulation angle 18, proposed memory

VII. Sub block level memory organization to achieve scalability

A full mode architecture for WiMax and WiFi applications must support

- 6 code rates for WiMax and 4 code rates for WiFi.
- 19 expansion factors (24-96) for WiMax and 3 expansion factors (27-81) for WiFi.
- 19 codeword sizes (576-2304) for WiMax and 3 codeword sizes (648-1944) for WiFi.
- Arbitrary values of parallelism (P) to achieve variable throughput.

Rows Processed				VNs Required				Memory Address	Right Rotation
1	9	17	25	18	26	2	10	2	2
2	10	18	26	19	27	3	11	3	2
3	11	19	27	20	28	4	12	4	2
4	12	20	28	21	29	5	13	5	2
5	13	21	29	22	30	6	14	6	2
6	14	22	30	23	31	7	15	7	2
7	15	23	31	24	32	8	16	8	2
8	16	24	32	17	25	1	9	1	2

Fig. 8. Verification : Sub Block Memory Organization

To support all expansion factors and parallelisms, enormous flexibility is required for interconnection network. One solution

TABLE III
THROUGHPUT AND AREA COMPARISON OF PROPOSED AND PUBLISHED LDPC DECODER IMPLEMENTATIONS

Implementation	[9]	[17]	[18]		[10]		[19]	This Work	
Technology (nm)	130	130	65	65	65	65	90	130	130
Code	WiMax	WiMax	WiMax	WiFi	WiMax	WiFi	WiMax	WiMax	WiFi
Dec. Modes	114	19	114	12	114	12	114	114	12
Frequency(MHz)	333	83.3	400	400	400	400	400	300	300
Iterations	10-15	8	20	20	25	25	8-12	20	20
Quantization (bits)	6	8	7	7	6	6	7	7	7
Parallelism	24-96	4	27	27	24-96	27-81	2-4	13-24	27
Area $[mm^2]$	3.834	8.29	0.5	0.5	1.337	1.023	0.679	2.764	2.744
Scaled Area $[mm^2]$	3.834	8.29	2	2	5.345	4.092	1.30	2.764	2.744
T.P (Mbps)	83-610	60-222	27.7-237.8	34.5-257	96-399	108-337	66.67-200	56-167	116-187
TAR (Mb/s/mm^2)	21.6-159	7.2-26.8	13.8-119	17-128.5	17.9-74.6	26.4-82.3	51.2-153.8	20.2-60.4	42.2-68.14

could be to make the width of each memory bank equal to maximum expansion factor (Z_{max}) for a particular code and use a fully connected $(Z_{max} \times Z_{max})$ network able to support all expansion factors and parallelisms less than Z_{max}. But this is expensive from implementation point of view, since smaller parallelisms yield hardware resources idle most of the time.

Scalable sub block parallel architecture requires assignment of variable nodes in memory such that given any value of *Z* and *P*, all *P* variable nodes are stored in the same memory word irrespective of the circulant shift. This is done by scheduling of parity equations within a sub-matrix such that given an expansion factor *Z* and parallelism *P* be an integer factor of *Z* i.e. Z mod $P = 0$, two consecutive variable nodes within a same memory word are at a distance of Z/P. This is explained by an example, suppose $Z = 32$, then all values of *P* satisfying Z mod $P = 0$ are 2, 4, 8, 16 and 32. Considering $P = 4$, the 32 VNs are stored in to $Z/P = 8$ memory locations with $P = 4$ VNs in each location. Figure 7 shows a 32×32 sub matrix with circulation angle 18 and proposed sub block level memory partitioning with $P = 4$. This sub matrix is processed in 8 sets of 4 parity check equations. Rows 1, 9, 17 and 25 are processed in 1st set and require VNs 18, 26, 2 and 10 which are stored at memory location 2 and obtained after rotating it left by 3. For the next set of 4 parity check equations VNs 19, 27, 3 and 11 are required which are stored at memory word 3 and accessed in the same way. Verification of this scheme is shown in Fig. 8. This scheme guarantees all *P* variable nodes accessed in single clock cycle no matter where the diagonal starts in a sub-matrix. Supporting *P* values that does not satisfy $Z\%P = 0$ constraint, requires additional words in memory occupying redundant variable nodes resulting in some penalty in terms of extra clock cycles.

VIII. IMPLEMENTATION RESULTS

This section deals with ASIC implementation of proposed Layered LDPC decoder which features parallel "Tree-way" check node implementation. The VHDL IP core for proposed decoder has been synthesized on $130-nm$ Standard Cell ASIC technology using Synopsys Design View tool. The operating frequency has been set to 300 MHz. The synthesized decoder is a full-mode architecture, able to decode all 114 codes for WiMax and 12 codes for WiFi. Table IV shows the throughput results for proposed decoder. The maximum number of 20 iterations has been selected to meet a satisfactory error performance for each code rate. The table displays the maximum achievable throughput for both cases, starting from the smallest code size with lowest code rate up to largest high-rate code.

TABLE IV
THROUGHPUT RESULTS OF PROPOSED DECODER IP

Throughput (TP) 300MHz, 20 Iterations			
	802.16e WiMax		
Code Rate	Code Size	Parallelism	TP (Mbps)
1/2	546-2304	13-24	56 - 103
2/3	546-2304	13-24	59 - 107
3/4	546-2304	13-24	71 - 131
5/6	546-2304	13-24	91 - 167
	802.11n WiFi		
1/2	648-1944	27	116
2/3	648-1944	27	122
3/4	648-1944	27	135
5/6	648-1944	27	187

TABLE V
SYNTHESIS RESULTS FOR PROPOSED DECODER IP

Area @ 300MHz, 7-bit Quantization		
Module	**Area WiMax $[mm^2]$**	**Area WiFi $[mm^2]$**
CNP	1.79	1.94
Memory	0.94	0.75
Network	0.03	0.05
Controller	0.004	0.004
Total	2.764	2.744

Table V shows the synthesis area results for proposed decoder. Parallelisms of 13-24 and 27 have been selected for WiMax and WiFi respectively to realize a full mode architecture to support low to moderate throughput applications for both standards. Both implementations show almost 50% of total area occupied by check node logic which is reflected by parallelism at check node level and huge flexibility to support variable check node degrees. The proposed decoder is able to achieve throughput well above 70 Mbps as specified by WiMax standard [3]. However, the proposed architecture is fully compliant to support higher parallelisms for applications requiring throughputs of the order of Gbps. For example parallelism of 96 results in a throughput of 1.7 Gbps for WiMax at 15 iterations.

IX. Comparison With Other Related Works

Table III compares the proposed work with some already published decoders. The Table shows for each decoder the main characteristics: implementation technology, supported codes and decoding modes (combination of codes length and code rate), achieved throughput, quantization, internal parallelism and occupied area. A parameter called throughput to area ratio (TAR) [20] defined as TAR=Throughput/Area has also been included in the Table to evaluate the efficiency of the designed decoder. However comparison to other similar implementations of LDPC code decoders is difficult because of the differences in terms of design choices, implementation technology, finite precision arithmetic. To simplify the comparison in Table III, area of each decoder has been scaled up to 130 nm process with a scaling factor of 2 and 4 respectively for 90 nm and 65 nm processes. On the contrary clock frequency has not been scaled as a universal conversion method between different technologies is not available. The proposed decoder architecture achieves sufficient throughput for both WiMAX and WiFi standards at the cost of an occupied area which is smaller than required for most of alternative solutions. In particular, implementations in [9], [17] and [10] use more area than the proposed decoder; comparable area is required for the recent, 65 nm architecture in [18]. In [19] a significantly smaller decoder is presented, but it only supports WiMAX codes.

X. Conclusion

LDPC codes have been adopted in a number of next-generation wireless standards for forward error correction. Implementing a fully flexible architecture while satisfying area, speed and power metrics is still a challenging task. In this paper we presented to best of our knowledge first implementation of LDPC decoder based on bottom up parallel approach. Fully scalable architecture for LDPC decoding based on proposed Parallel ”Tree-Way” Check Node realization has been presented supporting different codes of WiMAX and WiFi standards. In addition, a comparison between the proposed and state of art implementations is also presented. Even if state of the art low complexity techniques have been applied however area figures are still high when scaled to 130nm technology. The overall cost of proposed decoder is mainly determined by cost of huge flexibility and can be improved to a greater extent by supporting only a limited number of codes. The proposed decoder architecture is fully compliant to support large parallelisms for high throughput applications.

References

[1] R. G. Gallager, “**Low Density Parity Check Codes**,” *IEEE Trans. Inform. Theory*, vol. IT 8, no. 1, pp. 21–28, 1962.

[2] D. J. C. MacKay and R. M. Neil, “**Near Shannon Limit Performance of Low Density Parity Check codes**,” *Electronics Letters.*, vol. 32, pp. 1645–1646, Aug. 1996.

[3] “**IEEE Standard for Local and Metropolitan Area Networks-Part 16, Air Interface for Fixed broadband wireless access system,IEEE Std 802.16**,” 2004.

[4] “**IEEE 802.11n Wireless LAN Medium Access Control and Physical Layer specifications: Enhancements for Higher Throughput. IEEE P802.16n/D.1.0**,” Mar 2006.

[5] A. Blanksby and C. Howland, “**A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity-check code decode**,” *In IEEE Journal of Solid-State Circuits*, vol. 37, pp. 404–412, Mar. 2002.

[6] D. Hocevar, “**A Reduced Complexity Decoder Architecture via Layered Decoding of LDPC Codes**,” *In IEEE Workshop on Signal Processing Systems, SISP 2004*, pp. pp: 107–112, 2004.

[7] a. P. Urard et, “**A 135Mb/s DVB-S2 compliant codec based on 64800b LDPC and BCH codes**,” *In IEEE Solid-state Circuits Conference (ISSCC)*, 2005.

[8] T. B. F. Kienle and N. Wehn, “**A synthesizable IP core for DVB-S2 LDPC code decoding**,” *In IEEE Conference on Design Automation and Test in Europe (DATE)*, 2005.

[9] F. K. N. W. Torben Brack, Matthias Alles, “**A Synthesizable IP Core for WiMax 802.16E LDPC Code Decoding**,” *17th Annual IEEE Intl. Symposium on Personal, Indoor and Mobile Radio Commnications (PIMRC06), Helsinki, Finland*, pp. 1–5, September 2006.

[10] T. L.-E. F. K. N. W. L. I. F. R. M. R. L. F. T. Brack, M. Alles, “**Low Complexity LDPC Code decoders for Next Generation Standards.**” *Proceedings of the conference on Design, automation and test in Europe*, 2007.

[11] R. M. Tanner, “**A recursive approach to low complexity codes**,” *IEEE Trans. Inform. Theory*, vol. IT-27, p. 533 547, Nov. 1981.

[12] R. G. Gallager, “**Low Density Parity Check Codes**,” *Cambridge, MA : MIT Press*, 1963.

[13] B. N. E. Yeo, P. Pakzad and V. Anantharam, “**High throughput low-density parity-check decoder architectures**,” *In IEEE proceedings of GLOBECOM*, vol. 5, pp. 3019–3024, 2001.

[14] M. Mansour and N. Shanbhag, “**A 640-Mb/s 2048-bit programmable LDPC decoder chip**,” *IEEE J. of Solid-State Circuits*, vol. 41, no. 3, pp. 684– 698, Mar 2006.

[15] V. S. T. Bhatt, V. Sundaramurthy and D. McCain, “**Pipelined block serial decoder architecture for structured LDPC codes**,” *in Acoustics, Speech and Signal Processing, 2006*, 2004.

[16] V. B. John Dielissen, Andries Hekstra, “**Low cost LDPC decoder for DVB-S2**,” *Philips Research, High Tech Campus 5*.

[17] C.-H. L. X.-Y. Shih, C.-Z. Zhan and W. A.-Y, “**An 8.29 $mm2$ 640-Mb/s 2048-bit programmable LDPC decoder design for Mobile WiMax system in 0.13 μm CMOS process**,” *IEEE J. of Solid-State Circuits*, vol. 33, no. 3, pp. 672– 683, Mar 2008.

[18] T. V. M. Alles and N. Wehn, “**FlexiChap: A reconfigurable ASIP for convolutional, turbo and LDPC code decoding**,” *In Proc. Turbo codes related topics*, pp. 84– 89, Sep 2008.

[19] C.-L. C.-J. Y. Yu-Luen Wang, Y-L Ueng, “**Processing Task Arrangement for a Low -Complexity Full-Mode WiMax LDPC Codec**,” *IEEE Transactions on Circuits and Systems-1*, vol. 58, no. 2, pp. 415– 428, Feb 2011.

[20] F. Q. G. Masera and F. Vacca, “**Implementation of a flexible LDPC decoder**,” *IEEE transactions on circuit and systems II, Express Briefs*, vol. 24, no. 6, pp. 542–546, 2007.