ARCHITECTURES FOR DECODING OF STRUCTURED LDPC CODES USING THE ON-THE –FLY COMPUTATION PARADIGM

Kiran Gunnam, Gwan Choi, Mark Yeary[*]

Dept. of Electrical Engineering, Texas A&M University, College Station, TX-77840
[*] Dept. of Electrical and Computer Engineering, University of Oklahoma, Norman, OK-73109

ABSTRACT

Recent research efforts based on joint code-decoder design methodology have shown that it is possible to construct structured LDPC (Low Density Parity Check) codes without any performance degradation. An interesting new data independence property between the two classes of messages viz. check to bit and bit to check involved in decoding, is observed. This property is a result of the specific structuring of parity check matrix. By exploiting this property, we propose an architecture in which the computation of messages is synchronized such that each class of messages is consumed immediately by the computational unit for another class of messages. The internal memory of the check to bit units is increased in tune with the storage requirement of the check to bit messages. The separate memories for check to bit and bit to check messages are eliminated. This approach has memory savings of 75% and reduces the overall memory accesses by 66% when compared to some of existing work. When compared to recent implementations the savings are mainly in reduction of memory accesses and the speed of the proposed data path. The architecture renders itself to efficient implementation of various LDPC decoding algorithms such as Min-Sum and layered decoding.

1. INTRODUCTION

Low-Density Parity Check (LDPC) codes and Turbo codes are among the best known near Shannon limit codes [1]. LDPC decoding algorithm has more parallelization when compared to the decoding algorithm of Turbo codes. The major issues surrounding the VLSI implementation of LDPC decoders are the complex interconnects and large memory requirements due to the sparse nature of the parity generator matrix [2-3]. This paper proposes low complexity architecture with reduced memory requirements for LDPC decoding based on the recent work on structured LDPC codes [4-7].

LDPC codes can be described by an $m \times n$ parity check matrix H in which the average number of non-zero elements (i.e. one in GF2) in each row is a constant. In a (r, c) regular code, each of the n bit nodes $(b_1, b_2, ..., b_n)$ has connections to r check nodes and each of the m check nodes $(c_1, c_2, ..., c_m)$ has connections to c bit nodes. LDPC codes can be decoded by the Gallager's iterative belief-propagation (BP) algorithm [1].

The rest of the paper is organized as follows. Section 2 explains the code construction used and the relevant decoding schedule property which is exploited in the new architecture. Section 3 presents architecture to eliminate the message storage and Section 4 presents comparison with the existing work.

2. DECODING SCHEDULE

2.1. Code Design

We focus on the construction which structures the parity check matrix H into blocks of $p \times p$ matrices such that: 1. a bit in a block participates in only one check equation in the block and 2. each check equation in the block involves only one bit from the block. We show that this specific construction of LDPC codes has some interesting properties which are not exploited in the previous art.

One method to perform this construction is through cyclotomic cosets [7]. Another method is to achieve this property by employing random bit filling algorithm (for low rate codes such as rate ½ codes) and geometric constructions (for high rate codes such as rate 8/9 codes) [5, 6]. The work [7] reports no performance degradation for a (3, 5) - LDPC code of length 1055, rate 0.4; constructed from cyclotomic cossets. The work [5] reports a minimal performance degradation up to 0.2 dB for a (3, 6) - LDPC code of length 2040, rate 0.5; constructed with random bit filling algorithm and when the "parallelization factor" [5] is set to p=2040/6=340 (the case of our interest). The work [6] reports a minimal performance degradation up to 0.2 dB for a (3, 30) - LDPC code of length 2070, rate 0.9; constructed with random bit filling algorithm and when "parallelization factor" [5] is set to p=2070/30=69 (the case of our interest).

The H matrix can be constructed with filling with matrices obtained by permuting identity matrix by the

appropriate shift coefficients [7]. Say $B_{j,k}\ \forall j = 1,2..r; k = 1,2,..c$ is a $p \times p$ matrix, located at the j^{th} block row and k^{th} block column of H matrix. The scalar value $s(j,k)$ denotes the shift applied to $I_{p\times p}$ identity matrix to obtain the $(j,k)^{th}$ block, $B_{j,k}$, and the rows in the $I_{p\times p}$ identity matrix are cyclically shifted to the right $s(j,k)$ positions for $s(j,k) \in \{0,1,2,..., p-1\}$.

Let us define S as a $c \times r$ shift coefficient matrix in which $S_{j,k} = s(j,k)\ \forall j = 1,2..r; k = 1,2,..c$. (1)

So an H matrix, in this construction, can be completely characterized by these two simple matrices viz. $I_{p\times p}$ and $S_{c\times r}$.To define H matrix, we start with fixing c, r and finding an appropriate p and shift coefficient matrix S such that the BER performance is maintained when compared to a random construction.

For example if $c = 5, r = 3$ and $p = 211$ the use of cyclotomic cosets [7] results in the following shift coefficient matrix for the code of length $1055 (n = cp)$.

$$S_{3\times 5} = \begin{bmatrix} 2 & 3 & 110 & 142 & 165 \\ 5 & 64 & 96 & 113 & 144 \\ 7 & 50 & 75 & 116 & 174 \end{bmatrix} \quad (2)$$

The resulting H matrix has same BER performance when compared to a similar code with random construction [7].

2.2. Block Message Independence Property

The reliability messages used in Gallager's Belief Propagation algorithm can be computed in two phases viz. check node processing (3) and bit node processing (4) and this is repeated iteratively till the decoding criterion is satisfied [1,3]. The message passing equations are given by

$$R_{cj,bi} = \psi^{-1}\left[\left(\sum_{i'=Row[cj][1]}^{Row[cj][c]} \psi\left(Q_{i',cj}\right)\right) - \psi\left(Q_{bi,cj}\right)\right].\delta(cj,bi) \quad (3)$$

$$Q_{bi,cj} = \left(\sum_{j'=Col[bi][1]}^{Col[bi][r]} R_{j',bi}\right) - R_{cj,bi} + \wedge(bi) \quad (4)$$

where

$R_{cj,bi}$ is the message from check c_j to bit b_i , $Q_{bi,cj}$ is the message from bit b_i to check c_j , $\psi(x) = -\log(|\tanh(x/2)|$ is the Gallager's function which is invariant under its inverse, $\delta(cj,bi)$ is ± 1 and is given by

$$\delta(cj,bi) = \left(\operatorname{sgn}\left(Q_{bi,cj}\right).\prod_{i'\in Row[cj]} \operatorname{sgn}\left(Q_{i',cj}\right)\right).(-1)^{|Row[cj]|} \quad (5)$$

$(-1)^{|Row[cj]|} = 1$ for codes constructed with even parity. $\wedge(bi)$ is the intrinsic reliability metric of bit i. $Row[c_j][1...c]$ ($Col[b_i][1...r]$) gives the locations of bits (checks) connected to the check node c_j (bit node b_i).

We can represent R and Q messages by the following matrices for deriving the new data independence property. This arrangement is similar to physical message storage employed in [3] except that these matrices are not really stored in the proposed architecture.

$$Rm = \begin{bmatrix} R_{1,Row[1][1]} & R_{1,Row[1][2]} & \cdots & R_{1,Row[1][c]} \\ R_{2,Row[2][1]} & R_{2,Row[2][2]} & \cdots & R_{2,Row[2][c]} \\ \vdots & \vdots & \vdots & \vdots \\ R_{p\bullet r,Row[p\bullet r][1]} & R_{p\bullet r,Row[p\bullet r][1]} & \cdots & R_{p\bullet r,Row[p\bullet r][c]} \end{bmatrix}$$

$$Qm = \begin{bmatrix} Q_{1,Col[1][1]} & Q_{1,Col[1][2]} & \cdots & Q_{1,Col[1][r]} \\ Q_{2,Col[2][1]} & Q_{2,Col[2][2]} & \cdots & Q_{2,Col[2][r]} \\ \vdots & \vdots & \vdots & \vdots \\ Q_{p\bullet c,Col[p\bullet c][1]} & Q_{p\bullet c,Col[p\bullet c][2]} & \cdots & Q_{p\bullet c,Col[p\bullet c][r]} \end{bmatrix} \quad (6)$$

If we employ the partitioning of H matrix into r rows and c columns of p x p matrices, the R and Q messages in a p x p block can be processed simultaneously. The recent architectures [4,5,6,7] exploit this property to store messages in the memory partitioned into p independent memory banks and employ p copies of message computation units.

We now represent the R and Q messages in a p x p block as p x 1 vectors

$$\vec{R}_{j,k} = \left[Rm_{1+(j-1)p,k},..., Rm_{l+(j-1)p,k},..., Rm_{p+(j-1)p,k}\right]^T$$

$$\vec{Q}_{k,j} = \left[Qm_{1+(k-1)p,j},..., Qm_{l+(k-1)p,j},..., Qm_{p+(k-1)p,j}\right]^T$$

$$l = 1,2,..., p\ \forall j = 1,2,..., r, k = 1,2,..., c \quad (7)$$

Then R and Q messages in block matrix format are:

$$\vec{R} = \begin{bmatrix} \vec{R}_{1,1} & \vec{R}_{1,2} & ... & \vec{R}_{1,c} \\ \vec{R}_{2,1} & \vec{R}_{2,1} & ... & \vec{R}_{2,c} \\ : & : & : & : \\ \vec{R}_{r,1} & \vec{R}_{r,1} & ... & \vec{R}_{r,c} \end{bmatrix} \vec{Q} = \begin{bmatrix} \vec{Q}_{1,1} & \vec{Q}_{1,2} & ... & \vec{Q}_{1,r} \\ \vec{Q}_{2,1} & \vec{Q}_{2,1} & ... & \vec{Q}_{2,r} \\ : & : & : & : \\ \vec{Q}_{c,1} & \vec{Q}_{c,1} & ... & \vec{Q}_{c,r} \end{bmatrix} \quad (8)$$

Now the Gallager's equations can be written as

$$\vec{R}_{j,k} = \psi\left[\left(\sum_{k=1}^{c} \psi\left(\vec{Q}_{k,j}^{s(j,k)}\right)\right) - \psi\left(\vec{Q}_{k,j}^{s(j,k)}\right)\right].\vec{\delta}_{k,j} \quad (9)$$

$$\vec{Q}_{k,j} = \left(\sum_{j=1}^{r} \vec{R}_{j,k}^{p-s(j,k)}\right) - \vec{R}_{j,k}^{p-s(j,k)} + \vec{\wedge}_k \quad (10)$$

$$\vec{\delta}_{k,j} = \left(\operatorname{sgn}\left(\vec{Q}_{k,j}\right).\prod_{k=1}^{r} \operatorname{sgn}\left(\vec{Q}^{s(k,j)}{}_{k,j}\right)\right) \quad (11)$$

$$\vec{\wedge}_k = \left[\wedge\left(1+(k-1)p\right),...,\wedge\left(p+(k-1)p\right)\right] \quad (12)$$

where $\vec{Q}_{k,j}^{s(j,k)}$ ($\vec{R}_{j,k}^{p-s(j,k)}$) is the modified p x1 vector $\vec{Q}_{k,j}$ ($\vec{R}_{j,k}$), whose elements are circularly shifted in location by the amount $s(j,k)$ ($p - s(j,k)$).

Say $\vec{A}_j = \sum_{k=1}^{c} \psi\left(\vec{Q}_{k,j}^{s(j,k)}\right), \vec{B}_{k,j} = \psi\left(\vec{Q}_{k,j}^{s(j,k)}\right)$ (13)

$$\vec{C}_k = \sum_{j=1}^{r} \vec{R}_{j,k}^{\,p-s(j,k)} \,,\; \vec{D}_{j,k} = \vec{R}_{j,k}^{\,p-s(j,k)} \qquad (14)$$

Now

$$\vec{R}_{j,k} = \psi\left[\vec{A}_j - \vec{B}_{k,j}\right]\vec{\delta}_{k,j} \qquad (15)$$

$$\vec{Q}_{k,j} = \vec{C}_k - \vec{D}_{j,k} + \vec{\Lambda}_k \qquad (16)$$

Observation

We can observe that the j^{th} block row of R messages is only dependent on the j^{th} block column of Q messages and similarly the k^{th} block row of Q messages is only dependent on the k^{th} block column of R messages. Only one class of messages has to be stored if we schedule the pipeline of the R and Q message computation unit such that the either one of R and Q message units output the block row at once and multiplexing the other units schedule such that it is able to produce the output in block column fashion.

If p Check to Bit serial message computation units, which have internal FIFOs of size $(c\times(r-1)+1) \approx c.r$ are employed, this is approximately equivalent to storage requirement of one class of messages $(p.c.r)$. We do not need any additional memory for storing R and Q messages. By scheduling we can efficiently use the internal memory of the computational units.

3. ARCHITECTURE

For the example (3, 5) - LDPC code of length 1055 described in section 2, $r = 3, c = 5$ and $p = 211$. We can generalize the following discussion to any LDPC code with similar structure.

According to the observation made in Section 2, the pipeline is designed such that Q messages are produced block row wise and R messages are produced in block column fashion (Fig.1). Initially the Q messages are available in row wise as they are set to soft log likelihood information of the bits coming in chunks of p (10). The Q Initializer (Q Init) is an SRAM of size $n+p$ and holds the Λ values of two different frames. It can supply p intrinsic values to the BCUs each clock cycle and also can simultaneously read p intrinsic values from the channel at the start of iterations of the next frame. The data path of the design is set to 5 bits. ψ and ψ^{-1} are implemented with identical SRAM lookup tables The maximum number of iterations is set to 20 and the iterations will stop when the decoded vector d (using Majority function of Bit to check messages)satisfies the relation $dH^T = 0$.

The p by p interleaver is constructed with two input - two output switches and $\log 2(p)$ stages of $p/2$ switches are used. The Switching Sequence (SS) unit supplies the binary sequences to toggle switches in order to produce the shifts in the matrix $S_{3\times5}$ (2). The interleavers of R and Q messages will receive sequences column wise corresponding to the shifts (2, 5, 7, 3… 174) for cyclic shift up and down respectively (refer to eq.9 and 10).

The Check to Bit processing unit is composed of p serial computation units which computes the partial sum for each block row in a multiplexed fashion to produce the R messages in block column fashion. The registers ps1,ps2 and ps3 correspond to the partial sum for block row 1,2 and 3 respectively.

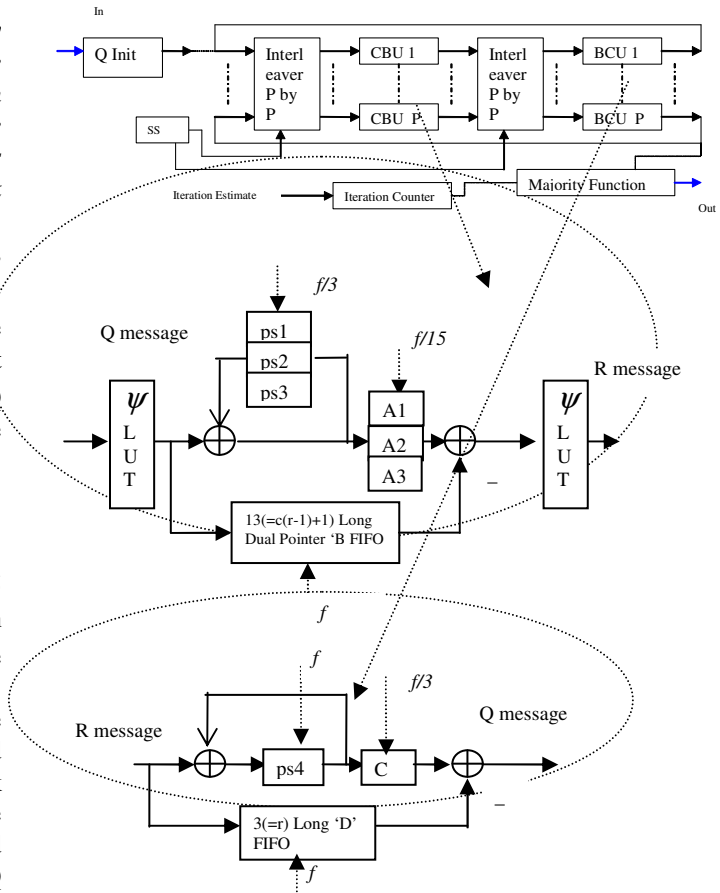

Fig. 1. Block Diagram of the Decoder Architecture

I	CBU Adders	CBU Sub tractors	BCU Adders	BCU Sub tractors
1	1-15	14-28	17-31	20-34
2	22-36	35-49	38-52	41-55

I=Iteration Number.

Table 1. Occupation of Resources for a decoding iteration in terms of clock cycles. (Shown for two iterations.)

I	CBU Adders	CBU Sub tractors	BCU Adders
1	1-15	14-28	17-31
2	19-33	32-46	35-49

Table 2. Occupation of Resources for a decoding iteration in terms of clock cycles; with the equivalent implementation of BCU where only required terms are added to reduces idle cycles. This BCU has two adders, three latches, two 2-input mux and one 3-input mux. The explanation of the architecture is still based on Fig1 for the simplicity.

Clock,I	13,1	15,1	22,1
$ps\vec{1}$	$\sum_{k=1}^{5}\psi\left(\vec{Q}_{k,1}^{s(1,k)}\right)$	$\sum_{k=1}^{5}\psi\left(\vec{Q}_{k,1}^{s(1,k)}\right)$	$\sum_{k=1}^{1}\psi\left(\vec{Q}_{k,1}^{s(1,k)}\right)$
$ps\vec{2}$	$\sum_{k=1}^{4}\psi\left(\vec{Q}_{k,2}^{s(2,k)}\right)$	$\sum_{k=1}^{5}\psi\left(\vec{Q}_{k,2}^{s(2,k)}\right)$	0
$ps\vec{3}$	$\sum_{k=1}^{3}\psi\left(\vec{Q}_{k,3}^{s(3,k)}\right)$	$\sum_{k=1}^{5}\psi\left(\vec{Q}_{k,3}^{s(3,k)}\right)$	0

Table 3. Snapshot of partial sum registers in p CBU s operating in parallel to compute p R messages.

The CBU B FIFO corresponds to (13) stores the intermediate computations. Its snapshot at 15th clock cycle is $\left[\vec{B}_{5,3,},\vec{B}_{5,2,}\vec{B}_{5,1,},...,\vec{B}_{1,1,}\right]$.The registers A1, A2 and A3 (which correspond to (13)) latch the ps1, ps2 and ps3 (Table 3) in 14,15 and 16 clock cycles respectively and one of these values (from 14- 28th clock cycle for 1st iteration) will be selected sequentially as one of the inputs to the subtractor and each subtraction operation during this period produces R messages in block column fashion.

The Bit to Check processing unit is composed of p serial computation units which compute the partial sum ps4 for each block row in a sequential fashion to produce the Q messages in block row fashion.

Clock,I	17,1	19,1	31,1
$ps\vec{4}$	$\sum_{j=1}^{1}\vec{R}_{j,1}^{p-s(j,1)}$	$\sum_{j=1}^{3}\vec{R}_{j,1}^{p-s(j,1)}$	$\sum_{j=1}^{3}\vec{R}_{j,5}^{p-s(j,5)}$

Table 4. Snapshot of partial sum registers in p BCU s operating in parallel to compute p Q messages

The BCU D FIFO corresponds to (14). Its snapshot at 19th clock cycle is $\left[\vec{R}_{1,1},\vec{R}_{2,1},\vec{R}_{3,1}\right]$ and at 31st clock cycle is $\left[\vec{R}_{1,5},\vec{R}_{2,5},\vec{R}_{3,5}\right]$.The register C (which correspond to (14)) latch the ps4 (Table 4), every three clock cycles and is one of the inputs to the subtractor and each subtraction operation during this period produces Q messages in block row fashion.

4. PERFOMANCE COMPARISON

Table5. shows the comparison with the related work. The memory savings are 75% and savings in memory accesses are 66% when compared to [3,4] . When compared to [5,8] the memory accesses are 50% less while the memory requirement is almost the same and this results in better low power characteristic for the proposed architecture. For example [8] reported that the NA-Mm accounts for 50% of their decoder power.

	Yeo [3]	Zhang [4]	Anand [5]	Mansour [8]	Proposed
Mm	$4p.c.r$	$2p.c.r$	$p.c.r$	$p.c.r$	0
Mc	$p.c$	$p.c$	$p.r$	$p.c$	$p.c.r$
NA_Mm	$4p.c.r$	$4p.c.r$	$2p.c.r$	$2p.c.r$	0
NA_Mc	$2p.c.r$	$2p.c.r$	$2p.c.r$	$2p.c.r$	$2p.c.r$

Table5. Memory requirement comparison

Mm: Memory for message storage Mc: Internal Memory in Check to Bit Serial Computational Units NA_Mm: No. of R/W accesses from Mm for a decoding iteration NA_Mc: No. of R/W accesses from Mc for a decoding iteration

5. CONCLUSION

A new decoding schedule for LDPC is presented based on a property of structured LDPC codes. A new architecture based on this decoding schedule is described. The design has reduced memory and correspondingly low power requirements when compared to the existing work. This proposed configuration can be thought of a parallel architecture scaled down by a factor c x r and still having the simple interconnection structure associated with a serial architecture.

EXTENSIONS

Advantages

Though the architecture presented here is for fixed rate decoding, it can serve as scalable architecture or to serve a decoder for different LDPC codes.

The proposed architecture data path (Cyclic Shifter->CBU with Internal State Memory and Final State Memory->Cyclic Shifter ->VBU) has several advantages.

It has uniform timing of each pipeline stage. For instance in [5], the partial sums will go through a router and reverse router and the final sum will have to go through another reverse router. This would affect the timing as well as increase the complexity of the design. In addition, with the code construction used in [5], the shits needed are not necessarily cyclic- this may result in costly implementation of switching network instead of simple multi-stage cyclic shifters. In addition, as pointed in previous section, the

memory requirements of the proposed architecture for the various levels of scaling are almost the same when compared to [5], but the number of memory accesses are 50% less for the specific configuration with number of units equal to p.

Multi-Rate Decoding

By introducing a memory equivalent to one block column of R messages and another block column of Q messages in Fig 1, this architecture can support decoding of any code length with a limited number of units M(1<M<p). However, now the internal state of CBU can not be minimized- this implies that partial sums will have to be stored in SRAM while the final sums will have to be stored in another SRAM. The switching circuit will be M x M cyclic shift network. The minimal memory that gets introduced has to be read in a specific order. Table 5 gives the possible architectures based on block independence property. In this case, the number of memory accesses are similar to the other architectures [3,4,5,8].

Min Sum algorithm and its variants

This architecture can support MinSum algorithm and its variations if CBU is modified. In this case the 'B' FIFO are of the width 1 bit and the depth remains the same- as now only sign bits need to be stored. The internal state of CBU will be Minimum1, Minimum2 and index of Minimum1 position instead of partial sum [9, 10, 11]. The final state of CBU will be Minimum1with correction, -(Minimum1 with correction), +/-(Minimum2 with correction), index of Minimum1 instead of final sum [9,10,11].

Multi-Rate Decoding for MinSum and its variants

By introducing a memory equivalent to one block column of Q messages, this architecture can support decoding of any code length with a limited number of units M(1<M<p). To support multi-rate decoding or to serve as scalable architecture, all the internal state and final state need to be stored in SRAM. If the SRAM for internal state and SRAM for final state are configured in a specific order. Cyclic shifts of P x P are possible on the final state by still using a switching circuit that can do M x M cyclic shift network.

Layered Decoding

This architecture can support layered decoding of Row Message Passing [12] with few minor changes. In this case further reduction of internal state is possible. Now instead of doing check node processing of block rows of R messages in time division fashion, one block row of R is done at a time and this will be sent to update the Q messages.

Simplification of Routers

Dynamic state assignment is used for a different class of architecture in [13] to eliminate the routers by using simple shift circuits..

Array codes are defined [14] as Here r is the row weight and c is column weight.

$$H = \begin{bmatrix} I & I & I & \dots & I \\ I & \sigma & \sigma^2 & \dots & \sigma^{r-1} \\ I & \sigma^2 & \sigma^4 & \dots & \sigma^{2(r-1)} \\ \vdots & & & & \\ I & \sigma^{c-1} & \sigma^{(c-1)2} & \dots & \sigma^{(c-1)(r-1)} \end{bmatrix}$$

$$\sigma = \begin{bmatrix} 0 & 0 & \dots & 0 & 1 \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & .0 & 0 \\ \vdots & & & & \\ 0 & 0 & \dots & .1 & 0 \end{bmatrix}$$

It is possible to eliminate both the routers if proposed On the Fly Type 3 is used with dynamic state assignment. In this case both partial state and final state will be wired for incremental shifts. The Cyclic shifter between Q and R messages can be realized by simple wiring of partial state in Check to Bit unit and no multiplexers are necessary. The Cyclic shifter between R and Q messages can be realized by wiring of final state in Check to Bit unit to permit cyclic shift and having 2-to 1 muxes to have the transfer from partial state to final state[See the Appendix for details]. The same concept can be extended for Type 1 and Type 3 architectures also with minor modifications.

REFERENCES

[1] R. G. Gallager, Low-Density Parity-Check Codes, M.I.T Press, 1963. Available at http://justice.mit.edu/people/gallager.html

[2] Blanksby, A.J.; Howland ,C.J, A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity-check code decoder, Solid-State Circuits, IEEE Journal of, Vol.37, Iss.3, Mar 2002 Pages:404-412

[3] Yeo, E.; Pakzad, P.; Nikolic, B.; Anantharam, V., "High throughput low-density parity-check decoder architectures" in proceedings of Global Telecommunications Conference, 2001.Volume: 5, Page(s): 3019 –3024

[4] T. Zhang and K. K. Parhi, "Joint (3, k)-Regular LDPC Code and Decoder/Encoder Design", to appear in IEEE Transactions on Signal Processing. Available at http://www.ecse.rpi.edu/homepages/tzhang/

[5] A. Selvarathinam, G.Choi, K. Narayanan, A.Prabhakar, E. Kim, "A Massively Scalable Decoder Architecture for Low-Density Parity-Check Codes", in proceedings of ISCAS'2003, Bangkok, Thailand.

[6] A. Selvarathinam, G.Choi, K. Narayanan, A.Prabhakar, E. Kim, "A Massively Scalable Decoder Architecture for Low-Density Parity-Check Codes". Journal version in submission.

[7] M. M. Mansour and N. R. Shanbhag, "Low Power VLSI Decoder Architectures for LDPC codes," in proceedings of International Symposium on Low Power Electronics and Design (ISLPED), Monterey, CA, Aug. 2002, pp. 284-289.

[8] M. M. Mansour, M. M. Mansour, and N. R. Shanbhag, "A Novel Design Methodology for High-Performance Programmable Decoder Cores for AA-LDPC Codes," in proceedings of IEEE Workshop on Signal Processing Systems (SiPS), Seoul, Korea, August 2003

[9] J. Chen and M. Fossorier, ``Near Optimum Universal Belief Propagation Based Decoding of Low-Density Parity Check Codes," IEEE Transactions on Communications, vol. COM-50, pp. 406-414, March 2002

[10] Cocco, M.; Dielissen, J.; Heijligers, M.; Hekstra, A.; Huisken, J.; "A scalable architecture for LDPC decoding" Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings Volume 3, 16-20 Feb. 2004 Page(s):88 - 93 Vol.3
[11] K.Gunnam and, G.Choi,"A Low Power Architecture for Min-Sum Decoding of LDPC codes"TAMU-ECE Technical Report. Avaialble at http://dropzone.tamu.edu
[12] Radosavljevic, P.; de Baynast, A.; Cavallaro, J.R.; "Optimized Message Passing Schedules for LDPC Decoding" Signals, Systems and Computers, 2005. Conference Record of the Thirty-Ninth Asilomar Conference on October 28 - November 1, 2005 Page(s):591 – 595
[13] Bhagawat, P.; Uppal, M.; Choi, G "FPGA based implementation of decoder for array low-density parity-check codes".; Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP '05). IEEE International Conference on Volume 5, 18-23 March 2005 Page(s):v/29 - v/32 Vol. 5
[14] E. Eleftheriou and S. Olcer, "Low density parity-check codes for digital subscriber lines", Proc. ICC'2002, New York, pp.1752-1757(2002).

Possible Architectures For Structured LDPC Codes[Array Codes, Cyclotomic Cosets]	*VNU*	*CNU*	*Switching N/W*
Basic Serial	*1Serial*	*1*	*0*
Programmable [SP,MinSum]	*1<K<P serial/Parallel*	*1<K<P serial/Parallel*	*K x K*
On the Fly-Type1[Proposed Decribed Here]	*P Serial*	*P Hybrid Serial*	*P x P*
On the Fly-Type2[proposed]	*P Parallel*	*dc*P Serial*	*dc P xP Cyclic Shifter. For array codes no shifter are necessary*
On the Fly-Type3[Proposed]	*dr*P Serial*	*P Parallel*	*dr P x P Cylic Shifter. For array codes no shifter are necessary*
On the Fly-Type4[Fully Parallel]	*dr * P Parallel*	*dc * P Parallel*	*dc*dr *P wires*
Proposed Programmable [SP,MinSum] Memory requirement would differ from the other programmable architectures. In addition the architecture has well balanced data flow and has good pipelining	*1<K<P Parallel for Standard Message Passing(SMP) 1<K<P Serial For Row Message Passing*	*dc *(1<K<P) Serial for high rate codes Parallel for low rate codes Tradeoff will be based on latency and throughput requirements*	*(dr-1)K x K For SMP K x K for Row Message Passing*
Proposed Programmable [SP,MinSum] Memory requirement would differ from the other programmable architectures. In addition the architecture has well balanced data flow and has good pipelining	*1<K<P Parallel For Column Message Passing*	*(1<K<P) Serial for high rate codes Parallel for low rate codes*	*K x K For Column Message Passing*

Table 5: Possible architectures for structured LDPC codes.
For array codes, the first block row and column does not need cyclic shift. This can be used to advantage in implementation.

Appendix:

Minimal Shifting based on constant incremental shifts in Array LDPC:

Say

$$\vec{A}_j = \sum_{k=1}^{c} \psi\left(\vec{Q}_{k,j}^{s(j,k)}\right) \quad (13)$$

$$\vec{B}_{k,j} = \psi\left(\vec{Q}_{k,j}\right) \quad (14)$$

$$\vec{S}_k = \vec{A}_j^{s(j,k)} \quad (15)$$

$$\vec{\delta}_j = \left(\prod_{k=1}^{r} \mathrm{sgn}\left(\vec{Q}_{k,j}^{s(j,k)}\right)\right) \quad (16)$$

$$\vec{\delta}_{k,j} = \left(\mathrm{sgn}\left(\vec{Q}_{k,j}\right)\vec{\delta}^{s(j,k)}{}_j\right) \quad (17)$$

$$\vec{R}_{j,k}^{s(j,k)} \equiv \vec{R}1_{j,k} = \vec{\delta}_{k,j}\psi\left(\vec{S}_k - \vec{B}_{k,j}\right) \quad (18)$$

$$\vec{C}_k = \sum_{j=1}^{r} \vec{R}1_{j,k} \quad (19)$$

$$\vec{D}_{j,k} = \vec{R}1_{j,k} \quad (20)$$

Now

$$\vec{R}_{j,k}^{s(j,k)} \equiv \vec{R}1_{j,k} = \vec{\delta}_{k,j}\psi\left(\vec{S}_k - \vec{B}_{k,j}\right) \quad (21)$$

$$\vec{Q}_{k,j} = \vec{C}_k - \vec{D}_{j,k} + \vec{\wedge}_k \quad (22)$$

Shift on values $\vec{S}_k$ in (15) and $\vec{\delta}^{s(j,k)}$ in (17) can be implemented as cyclic shift down register which would obviate the need for Omega n/w for R messages.

For example for block row 2 in array codes, output sum register in CBU unit i is connected to both accumulator as well as the output sum register of CBU unit i-1. For example for block row 3 in array codes, output sum register in CBU unit i is connected to both accumulator as well as the output sum register of CBU unit i-3.Assuming that all the block rows are done on separate units, first block row wont need any switching circuits either in CBU or in VBU. For the second and third rows, this would lead to reduction of muxes from log2(p)*p*5 to p*(6+1) . 5 represents the word length of R message- since now we are routing sum, we need 6 bits. To accommodate routing for sign product, we need another bit. We finally note that this property is already used in [4], however here all the block rows are time multiplexed – this may lead to routing congestion. We note that our approach will be very advantageous as all the block rows are done in parallel and has no routing congestion.

Elimination of Router between VNU and CNU:

$$\vec{A}_j = \sum_{k=1}^{c} \psi\left(\vec{Q}_{k,j}^{s(j,k)}\right)$$

If $s(j,k) - s(j,k-1) = w(j) \ \forall 1 < k \le c$.

Here w is a constant. For array codes, for the first block row w =0

for the second block row w =1

for the second block row w =2

$$\vec{A}_{j,1} = \vec{0}$$

$$\vec{A}_{j,k} = \vec{A}_{j,k-1}^{w(j)} + \psi\left(\vec{Q}_{k,j}\right) \quad \forall 1 < k \le c$$

Then partial sum for the first block row can be written as

$$\vec{A}_{1,1} = \vec{0}$$

$$\vec{A}_{1,k} = \vec{A}_{1,k-1} + \psi\left(\vec{Q}_{k,2}\right) \quad \forall 1 < k \le c$$

Then partial sum for the second block row can be written as

$$\vec{A}_{2,1} = \vec{0}$$
$$\vec{A}_{2,k} = \vec{A}^{1}_{2,k-1} + \psi\left(\vec{Q}_{k,2}\right) \quad \forall 1 < k \leq c$$

Then partial sum for the second block row can be written as

$$\vec{A}_{3,1} = \vec{0}$$
$$\vec{A}_{3,k} = \vec{A}^{2}_{3,k-1} + \psi\left(\vec{Q}_{k,2}\right) \quad \forall 1 < k \leq c$$

As we can see from above equations, we do not need shifters or any muxes. We just need constant wiring on the partial state register such that shift by appropriate constant $w(j)$ is provided for each block row.

Extension to Min Sum

The same logic can be applied for Min Sum as well. For Min-Sum, (19) would have a state space representing Minimum 1, Minimum 2 and locator for Minimum 1. So now a cyclic shift down register which can shift a total of 16 bits is needed. So now we need p*(16) muxes as opposed to the original requirement of log2(p)*p*5 . If

This appendix presents an efficient way to implement switching network for array codes. Savings would depend on paralleization p. If p is more than 8 then only we can get savings. Since p can be chosen as a large number such as 64, this method would give a savings of around 33% in Forward Shuffling network. If p =211, then the savings would be 60%.

Savings in reverse router can also be realized if shifting on partial state of CBU is done.