Memory efficient scalable decoder architectures for Low Density Parity Check codes

Abhiram Prabhakar and Krishna Narayanan
Department of Electrical & Computer Engineering,
Texas A&M University,
College Station, TX-77843
Email: {pabhiram,krn}@ece.tamu.edu

***Abstract*— We present a memory efficient low density parity check (LDPC) decoder that implements a modified Sum Product Algorithm (SPA). A memory efficient implementation for the Min-Sum algorithm is also presented. Serial and scalable partly parallel architectures with both parallel flooding schedule and serial message passing schedule are presented for the regular SPA, modified SPA and Min-Sum algorithms. Compared to a regular SPA decoder the proposed modified SPA architecture reduces the number of extrinsic messages stored at the decoder that forms the bulk of the hardware. Memory reduction increases with the rate of the code and up to 75% savings is achieved for a rate 9/10 code. The architecture for serial scheduling has a faster convergence over parallel scheduling and can save up to 40% decoding time for 20 decoding iterations. Simulation results show that the proposed changes to the SPA do not degrade the bit error rate (BER) performance and convergence of the decoder. The proposed architecture is scalable in logic for achieving various throughput and latency requirements by scaling the computational units of a serial architecture and partitioning the memory for parallel read and write access.**

I. Introduction

Low density parity check (LDPC) codes, discovered by Gallager [1] have been the focus of intense research in recent years. It has been shown that LDPC codes can be designed to achieve communication at rates close to the capacity limit [2]. Better coding gains and lower complexity compared to turbo codes have made them a strong candidate for application in upcoming communication standards.

Recently, a number of LDPC decoder architectures have been proposed. These architectures can be broadly classified into three types - namely serial, parallel and partly parallel architectures.

A serial architecture performs one bit node update, one check node update or processes a message between a check/bit node at a time. These architectures require very little logic and the area of the decoder is dominated by the memory blocks. Such a serial decoder architecture that decodes one bit at a time was presented in [3].

A parallel architecture instantiates logic components for decoding the whole LDPC graph simultaneously. Although no memory is required to store intermediate values, routing between bit and check update units become highly impractical for long codes. The routing complexity limits fully parallel implementation to codes of about thousand bits long such as what has been presented in [4].

A partly parallel architecture processes several edges, bit node updates or check node updates simultaneously. For these decoders, the parity check matrix of the LDPC code must have special structural constraints to read/write several messages from/to the memory simultaneously. Partly parallel decoders have been presented by Yeo *et al.* [5] [6], Zang *et al.* [7] [8], Hocevar *et al.* [9], Gunnam *et al.* [10] [11], Karkooti *et al.* [12] [13], Brack *et al.* [14].

One of the main limitations in partly parallel architectures is the memory used for storing the extrinsic messages (bit to check or check to bit) and memory for storing partial update values. Our main contribution in this paper is to propose a partly parallel scalable architecture that results in reduced memory requirements. This is accomplished by modifying the sum product algorithm (SPA) to allow intrinsic feedback of soft values at the check update. As a result of this modification, only a few bit to check messages or check to bit messages need to be stored, thereby significantly reducing the memory requirement. Our proposed modification (first published in a conference version in [15]) is similar to the approximate-min constraint independently proposed by Jones *et al.* [16].

The proposed architecture requires lesser memory compared to architectures proposed in [8], [5] which store all the bit to check messages and check to bit messages, and architectures proposed in [9], [10] which store either check to bit or bit to check messages. Our proposed architecture can also be used with a modified min-sum algorithm instead of the modified sum product algorithm. When used with the modified min-sum algorithm, our proposed architecture requires lesser memory compared to those in [17] and [12].

The next section introduces LDPC codes, SPA and Min-Sum decoding algorithms. Finite precision implementation of SPA decoding is discussed in section III. The proposed modified SPA algorithm is discussed in section IV. The scalability of the architecture and requirements for parallelization, memory mapping are discussed in section V. The proposed architectures with parallel and serial scheduling are discussed in section VI. Section VII summarizes and concludes the paper.

II. Introduction to LDPC codes

Let $\mathbf{a} = a_0.....a_{K-1}$ be a K bit information word which is mapped onto an N bit codeword $\mathbf{c} = c_0....c_{N-1}$ of an LDPC

code. An Low density parity check (LDPC) code is a code which has a sparse parity check matrix. Let the i^{th} column of the parity check matrix have λ_i 1's and let the j^{th} row have ρ_j 1's. If λ_i's are same for all columns and ρ_j's same for all rows, then it is referred to as a regular LDPC code. When they are different it it is referred to as an irregular LDPC code.

LDPC codes are well represented by bipartite graphs. One set of nodes, the variable or bit nodes correspond to elements of the code word and the other set of nodes, viz. check nodes, correspond to the set of parity check constraints satisfied by the codewords. Typically, the edge connections are chosen at random. The error correction capability of the LDPC code is improved if cycles of short length are avoided in the graph.

A. SPA decoding of LDPC codes

Although decoding of LDPC codes using the SPA is well known, we explain this in a bit of detail since the proposed modification is based on this algorithm. For the ease of exposition, we will restrict our attention to regular LDPC codes. The extension to irregular codes is straight forward. Further, we will assume binary phase shift keying (BPSK) as the modulation scheme where a 1 is mapped to $+1$ and 0 is mapped to -1 and additive white Gaussian noise (AWGN) with mean 0 and variance σ^2. The received channel values are given by,

$$r_i = c_i + n_i \qquad \text{for } i = 0 \text{ to } N-1 \tag{1}$$

The received channel values are converted to log-likelihood ratios (LLR) given by,

$$L_{ch}(c_i) = \log\Big(\frac{Pr(c_i=0|r_i)}{Pr(c_i=1|r_i)}\Big) = \frac{-2}{\sigma^2} r_i \tag{2}$$

Let us define the function,

$$\psi(x) = \log\left(\tanh\left(\frac{|x|}{2}\right)\right) \tag{3}$$

Decoding of LDPC codes is based on iteratively passing messages between bit nodes and check nodes along the edges through which they are connected. Two different computations have to be performed during a decoding iteration, namely the bit node update and the check node update. Let $L_c^q(i)$ represent the check to bit message along the i^{th} edge connected to the l^{th} check node during the q^{th} iteration (we will not explicitly use the index l to denote quantities associated with the n^{th} node as the operations are identical at all nodes). Similarly, we represent bit to check messages by $L_b^q(i)$.

Enforcing the parity check constraint on the incoming bit to check values, the outgoing message at a check node

$$|L_c^q(i)| = \psi^{-1}\left(\sum_{k=1,k\neq i}^{k=\rho} \psi(L_b^{q-1}(k))\right), \forall i \tag{4}$$

$$sign(L_c^q(i)) = \prod_{k=1,k\neq i}^{k=\rho} sign(L_b^{q-1}(k)) \tag{5}$$

Where t is the degree of the check node and index k refers to the k^{th} edge connected to the check node. It can be shown that the functions $\psi(x)$ and $\psi^{-1}(x)$ are identical.

For serial implementation in hardware, Eq. 5 is divided into two steps. First, we find two quantities $M1$ and $S1$ given by

$$M1 = \sum_{k=1}^{k=\rho} \psi(L_b^{q-1}(k)) \qquad S1 = \prod_{k=1}^{k=\rho} sign(L_b^{q-1}(k)). \tag{6}$$

Note that by definition of ψ, $M1$'s are always negative. Next, we find $M2(i)$ and $S2(i)$ for all the edges $i = 1$ to ρ according to

$$M2(i) = M1 - \psi(L_b^{q-1}(i)) \qquad S2(i) = S1 \times sign(L_b^{q-1}(i)) \tag{7}$$

Now,

$$L_c^q(i) = S2(i) \times \psi^{-1}(M2(i)) \tag{8}$$

The soft output for the l^{th} bit is given by,

$$L_{soft}(c_l) = L_{ch}(c_l) + \sum_k L_c^q(k) \tag{9}$$

The outgoing message from a bit node is given by,

$$L_b^{q+1}(i) = L_{ch}(c_l) + \sum_{k\neq i} L_c^q(k) \tag{10}$$

Alternatively, the bit node update can be written as,

$$L_b^{q+1}(i) = L_{soft}(c_l) - L_c^q(i) \tag{11}$$

The l^{th} bit is decoded as 1 if $L_{soft}(c_l) < 0$, else as 0.

B. Min-Sum Decoding

The Min-Sum (MS) algorithm is one of the reduced complexity decoding algorithms and we discuss it in brief as we compare the performance of the modified SPA algorithm to it. Also, our proposed architectures implement variations of the MS algorithm. In MS decoding, only the check update differs from the regular SPA algorithm. While the sign of L_c^q messages remain the same as in (7), the magnitude can take only two different values.

Let $j1$ and $j2$ be the two edges connected to a check node that correspond to the two least values of $|L_b^{q-1}(k)|$, i.e.,

$$j1 = \arg\min_{k=1}^{k=\rho} |L_b^{q-1}(k)| \tag{12}$$

$$j2 = \arg\min_{k=1,k\neq j1}^{k=\rho} |L_b^{q-1}(k)| \tag{13}$$

Let $min1$ and $min2$ be the two least values of the magnitude of the LLR's, i.e., $min1 = L_b^{q-1}(j1)$ and $min2 = L_b^{q-1}(j2)$. The magnitude of the outgoing messages at a check node is given by,

$$\begin{aligned} |L_c^q(i)| &= min1 \quad \text{if } i \neq j1 \\ |L_c^q(i)| &= min2 \quad \text{if } i = j1 \end{aligned} \tag{14}$$

There is a performance loss for the min-sum decoder compared to the regular SPA decoder. The performance can be improved by scaling the L_c messages by a fixed factor $\frac{|L_c|}{\beta}$ or by adding a fixed correction factor $(|L_c| - \alpha)$ [18]. Issues related to implementation of MS algorithm and its modifications were explored by Zhao *et al.* [19], Chen *et al.* [20].

III. Finite Precision implementation

Choosing the word length for representing L_{ch}, L_b, L_c and ψ values offers a trade-off between performance and hardware requirements of the decoder. Finite precision performance of SPA in its original form was studied by Ping *et al.* [21] and later by Zhang *et al.* [22]. The authors in [22] limit their study to rate 1/2 codes and propose a non-uniform quantization scheme. We study the effect of finite precision in more detail for various rate codes.

The following can be noted about the $\psi(x)$ function - (i) $\psi(x)$ is always negative and, hence, there is no need to consider the sign of the output, (ii) $\psi(x)$ and $\psi^{-1}(x)$ are same functions, (iii) the function $\psi(x)$ has a large slope for $0 < |x| < 1$ and hence, x should be very small for $\psi(x)$ and $\psi^{-1}(x)$ to be large.

We observe from simulations that the range of L_c messages (output of ψ^{-1}) should be large for good performance of the decoder in the error floor region. For this, we need $\psi(x)$ to have very small quantization steps close to zero. The word length of the check update adder (ψ adder for $M1$) will be determined by the smallest quantization step and the range of $\psi(x)$. To minimize the adder size, the range of $\psi(x)$ should be limited to a small value. Since $\psi(x)$ is a non-linear function, choosing a non-uniform quantization for its finite precision representation would save hardware size in the look up tables(LUT) used for implementing the $\psi(x)$ and $\psi^{-1}(x)$ transformations. Let q be the number of bits used for representing $\psi(x)$. The word length of the $\psi(x)$ LUT is linearly proportional to q, whereas the length of the $\psi^{-1}(x)$ LUT is proportional to 2^q. The check update adder works only on uniformly quantized ψ values and, hence, we need simple conversion circuits to convert from non-uniform quantization to uniform quantization and vice versa.

A. Clipping and Uniform quantization

We use the following convention to represent quantities with finite precision. Uniform quantization of any quantity Z is denoted as Z-$(N_i : N_f, step)$, where N_i is the number of bits used to represent the integer part and N_f is the number of bits used to represent the fraction part and $step$ is the step size. Signed quantities have an extra bit. The range used for $\psi(x)$ determines the range for L_c messages($|L_{c_{min}}|$,$|L_{c_{max}}|$). A lower maximum value for $\psi(x)$ will result in higher $L_{c_{min}}$ and this would lead to overestimation of lesser reliable L_c (smaller L_c) messages and, hence, a loss in error performance. For example, $\psi_{max}(x) = 4.0$ gives $L_{c_{min}} = 0.037$, $\psi_{max}(x) = 1.0$ gives $L_{c_{min}} = 0.78$ $\psi_{max}(x) = 0.5$ gives $L_{c_{min}} = 1.4$. We limit ψ_{max} to save hardware and minimize loss in performance by underestimating (reducing) the $\psi^{-1}(x)$ for values close to $L_{c_{min}}$. Another thing to be noted is $\psi^{-1}(0)$ can be at most set to $L_c(max)$. To prevent any loss of performance due to large overestimation we set this not far higher from $\psi^{-1}(0.5)$.

Fig. 1 show the bit error rate(BER) performance of a ($\lambda = 3$,$\rho = 30$), rate 0.9, regular LDPC code of length 7680. We simulate enough frames to observe at least 50 frame errors for all our plots for statistical accuracy. For all our plots we do 30 iterations at the decoder. It can be seen that limiting the

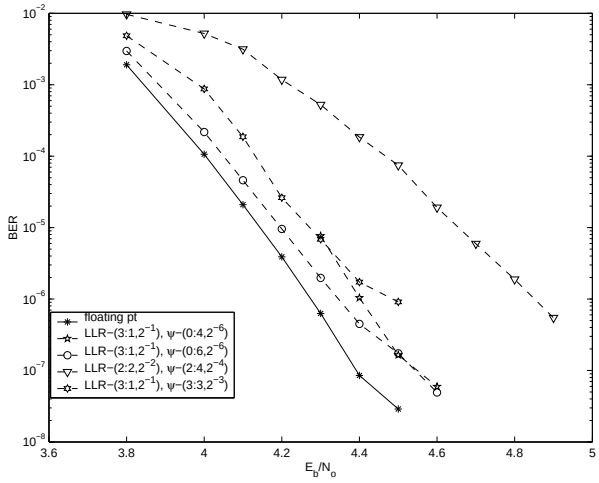

Fig. 1. Bit error plot of a $(3, 30)$ rate 0.9 code of length 7680

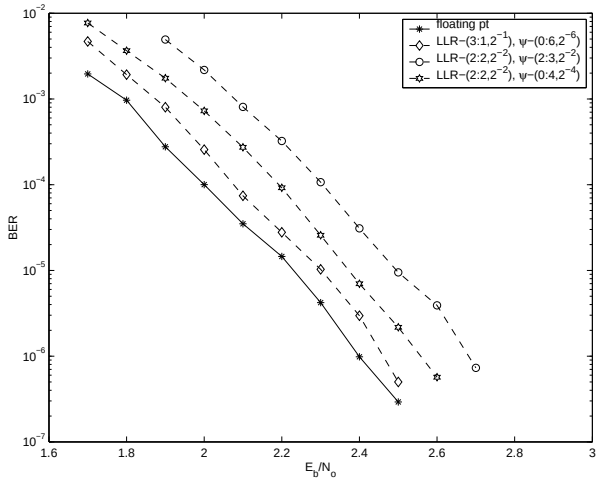

Fig. 2. Bit error plot of a $(3, 6)$ rate 0.5 code of length 2040

LLR values to 3.75 with $(LLR-(2:2,2^{-2}), \psi-(2:4,2^{-4}))$ leads to a large performance loss. Increasing the range of LLR values while allowing the ψ function to have a large range, for example, $(LLR-(3:1,2^{-1}), \psi-(3:3,2^{-3}))$ leads to an error floor since the step size is not small enough for L_c messages to take large values. The error floor is lowered significantly by allowing a smaller range for ψ, for example, $(LLR-(3:1,2^{-1}), \psi-(0:6,2^{-6}))$. Finally we use a 4 bit ψ function by limiting the range to 0.25 $(LLR-(3:1,2^{-1}), \psi-(0:4,2^{-6}))$. This scheme has a 0.2 dB loss compared to floating point implementation. The loss would be much higher without the underestimation of L_c messages close to $L_{c_{min}}$.

Fig. 2 shows the BER performance of a ($\lambda = 3$,$\rho = 6$), rate 0.5, regular LDPC code of length 2040. A performance loss of less than 0.05 dB is attained by having a $(3:1,2^{-1})$ representation for LLR values and $(0:6,2^{-6})$ for ψ values. When only 4 bits are used for representing ψ, a 0.1 dB gain is achieved by decreasing the clipping range for ψ and, thereby, decreasing the step size. The quantization step size, maximum and minimum clipping levels for ψ, ψ^{-1}, L_b and

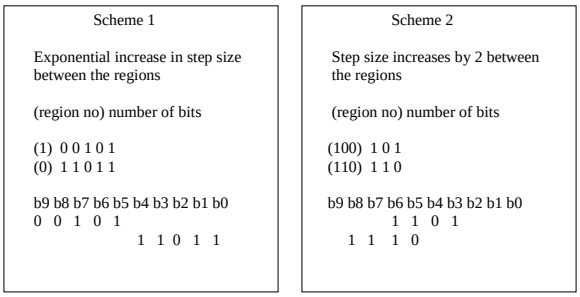

Fig. 3. Example:proposed non-uniform schemes

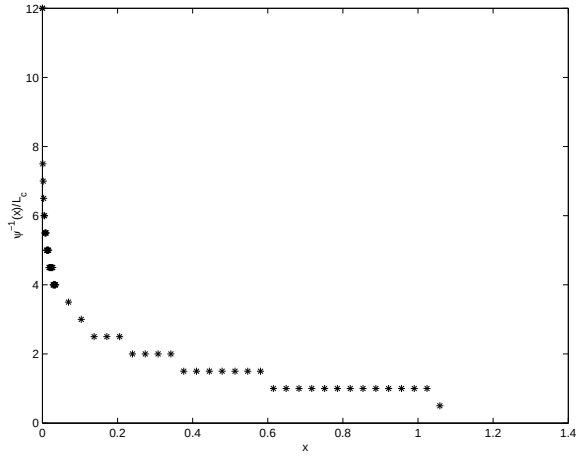

Fig. 4. Finite precision ψ^{-1} function with non-uniform quantization

L_c were picked from empirical observations with the objective to minimize the number of bits for representation without a significance loss in coding gain.

B. Non-Uniform Quantization

A large range for the L_c messages would require extremely small quantization steps close to zero for the ψ values and a uniform quantization would require large look up tables (LUT's). The ψ^{-1} output is less sensitive for larger values of ψ and, hence, a larger step size would be sufficient. Clearly, a non-uniform quantization would result in savings in the size of the LUT's as explained before. We propose two simple non-uniform quantization schemes for ψ values that offer easy conversion to uniform quantization. The range for ψ is limited as discussed before.

1) Scheme 1: The range of ψ is divided into x non overlapping regions numbered $i = 0$ to $i = (x-1)$(defined by $\log_2(x)$ bits) each having 2^{y_i} points. The step size for the i^{th} region is given by the total range of 0 to $i-1$ regions. Conversion to uniform quantization for a point in the i^{th} region($i \neq 0$) is achieved by a left shift of $\sum_{t=0}^{t=i-1} y_t$ bits. The uniform representation and hence the ψ adder will have $\sum_{t=0}^{t=i-1} y_t$ bits. The representation is illustrated with an example in Fig. 3. For region 1 the bits are shifted to the left by 5 bits. To take ψ^{-1} we need to convert the ψ sum that has a uniform representation back to non-uniform representation. Let the the ψ^{-1} look up table have v bits. First we look at a window of left most v bits. If this is non zero we ignore all the other bits to the right. Else, we move the window and look at the next v bits and the process is continued till we find the left most v non zero bits or we reach the last v bits. Typically 2 regions with equal number of points and window size of one region is used. Let us consider the simplest case when there are only 2 regions, with each region having 2^5 points. The step size for the 0^{th} region is chosen as 0.0011. The step size for region 1 will be $.0011 \times 31 = 0.0341$. The total range for ψ values will be 0 to 1.1253. Fig. 4 shows the ψ^{-1} output for the above scheme. It can be seen that the range of L_c values extend to 7.5. The ψ values have large number of points close to zero and hence the ψ^{-1} output has large number of points for higher magnitude.

2) Scheme 2: In the previous scheme the step size was increased rapidly for each region. Alternatively more regions can be used with the step size doubling from the previous region. This leads to gradual increase of step size and would be more desirable for implementation over codes of various rates. Let there be x regions numbered $i = 0$ to $i = (x-1)$ defined by $\log_2(x)$ bits each having 2^y points. To convert from non-uniform quantization to uniform quantization we first look at the region number. The uniform representation is given by a 1 in the $i+y$ th bit followed by the y bits. The range of ψ values represented by this scheme is from 0 to $\sum_{t=0}^{t=y} 2^{x+t-1} \times \delta$, where δ is the smallest steps size. For $i = 0$ it will be just the y bits since the 1 will overlap with the y bits. An example is shown in Fig. 3. Here each region has 3 bits and there are a total of 8 regions. To take ψ^{-1} we need to convert the uniform representation back to non-uniform quantization. We can use an inverse procedure and the ψ^{-1} would require an LUT of size $\log_2(x)+y$ bits. Alternatively we can use the windowing method of v bits described in Scheme 1.

The motivation for non-uniform quantization was to have more quantization levels for ψ values close to zero than for large values. The idea behind the two proposed schemes is to grow the step size rapidly for larger values. The exact step sizes are chosen empirically.

IV. Proposed modification to SPA

The motivation behind the modifications to the SPA is to reduce the memory requirement for storing the L_b and L_c messages without undergoing any significant loss in the coding gain. We propose two methods that achieve this. The sign of a message depends on the data bit and always needs to be stored at the decoder. The proposed approximation to SPA decoding results in reducing the number of unique magnitudes for L_c messages from ρ to only two different values and, hence, reduces the number of operations in finding them. Also, the approximations cause only simple modifications in the architecture used for a regular SPA decoder and, therefore, do not affect the overall design and requirements.

A. Method 1

We propose to modify $M2(i)$ given by Eq. 7. Among the edges $1, 2, ...\rho$ connected to a check node, let the j^{th} edge

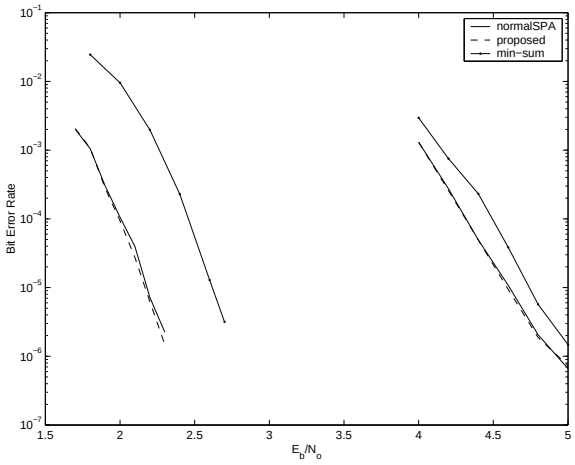

Fig. 5. performance of different decoding algorithms over an AWGN channel for code rates 0.5 and 0.9 and length 2040

have the lowest magnitude of L_b. That is $j = \arg\min_i |L_b(i)|$. Then $M2(j)$ is given by,

$$\begin{aligned} M2(j) &= M1 - \psi(L_b^{q-1}(j)) \\ M2(i) &= M1 \quad \text{for all other edges}(i \neq j) \end{aligned} \tag{15}$$

This intrinsic feedback for the magnitude of L_c messages was previously presented by us in [15] and similar to the Approximate-Min check update [16] but differs in implementation to suit our check update procedure which is done in parallel with finding the least reliable edge as explained in section VI.

Fig. 5 shows the BER performance for the proposed scheme for a rate 0.5 and rate 0.9 $(3, k)$ regular code of length 2040. It can be seen that there is no noticeable loss in performance compared to the regular SPA decoding. The performance is better than Min-Sum decoding which is another possible approximation to reduce memory requirement. The reason for this is discussed below.

From the properties of $\psi(x)$ function, we know that $\psi(x)$ and $\psi^{-1}(x)$ are decreasing functions. Also, the function exhibits a sharp slope for $0 < |x| < 1$ and is close to zero for $3 < |x| < \infty$. Hence, large values of $|L_b|$ contribute very little to $M1$ and less reliable L_b values contribute larger values to $M1$. It should be noted that for a normal SPA check update, the L_c message along an edge depend on the L_b messages along all other edges and is dominated by the the least reliable edge among them. When one or more edges are unreliable, $M1$ is large and is dominated by the unreliable edges. When $M1$ is large, $M2$ is large and, hence, L_c will be small and will be insensitive to small variations in $M2$. Since the more reliable edges contribute very little to $M1$, not subtracting their intrinsic input to get $M2$ will cause very little underestimation in their L_c values. The maximum underestimation will be for the edge that has a reliability same as that of the most unreliable edge but this will not matter since the check will be very weak when two edges are unreliable. When all the edges are reliable then $M1$ will be very small and hence L_c values will be very large. Even though ψ^{-1} function is highly sensitive in this region, the underestimated L_c values are large enough and do not cause any major performance loss. Also, for finite precision implementations the range of the function will be limited and hence the underestimation will have no effect. The update is self-tuning in the sense that the feed-back is applied only on non-min edges. For the least reliable edge we do not do any underestimation and hence $L_{c(prop)} = L_{c(norm)}$. The result is interesting since not subtracting the *a priori* information in the LLR domain(for example, at bit node update) can result in more optimistic extrinsic values, which cause error propagation. For a Min-Sum decoder $M2_{Min-Sum} > M2_{SPA}$ and hence the L_c values are more optimistic. This positive feedback results in error propagation and hence results in the range of 0.2 dB to 0.8 dB depending on the rate of the code used. The performance loss can be reduced to $0.1 - 0.2$ dB by applying correction terms to decrease this overestimation [18], [20], [19]. The correction factor depends on the rate of the code and also an irregular code would need different correction factors for bit and check nodes of different degrees [23]. Hence when the same hardware is used for decoding codes of various rates and different degree profiles, finite precision requirements for the various correction factors is not well understood.

B. Method 2

In this method, instead of a self-tuning approach we follow a fixed-tuning approach where a reliability threshold is selected. The magnitude of M2 is then given by,

$$\begin{aligned} M2(j) &= M1 - \psi(L_b^{q-1}(j)) \quad \text{if } |L_b^{q-1}(j)| < T \\ M2(i) &= M1 \quad \text{otherwise} \end{aligned} \tag{16}$$

We exclude the intrinsic input $(\psi(L_b^{q-1}(j)))$ from $M1$ if the input is larger than T otherwise we allow intrinsic feedback. This feedback would not result in a major loss in performance only when the L_b input is reliable. Hence, very small values for T would lead to a major loss in performance. We simulate the coding gain by varying T at higher values and also varying the quantization levels to choose the best among them.

The number of finite levels below the threshold for the L_b messages is adjusted to have a representation with 2 or 1 bits. Only these bits need to be stored in the decoder for each message along the graph which will be used in the subtraction (7) to remove the intrinsic feedback. Fig. 6 illustrates this procedure.

From empirical observations, we picked T to be 3.5.We have only 4 levels to represent L_b when it is stored in the memory. They correspond to values 0,2,3,3.5 coded as '0','1','2','3' from the actual levels '0'-'7'. The ψ table used for the subtraction (7) will have only 3 entries. Intuitively, let us understand why this procedure should work. L_b messages greater than T are more reliable and, hence, contribute very little to the ψ sum. Hence, with a similar reasoning as in Method 1 we can say that feedback in magnitude for these edges would affect the L_c magnitude very little. When several of the edges are unreliable, the ψ sum would saturate or be very high such that variations to the sum would hardly cause

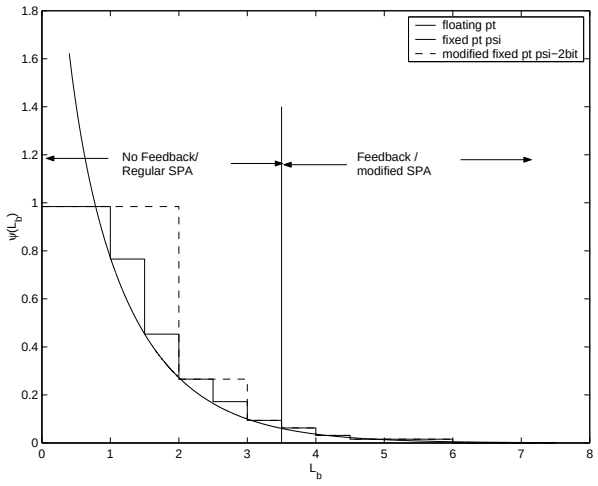

Fig. 6. Modified PSI Look Up Table used for Method 2

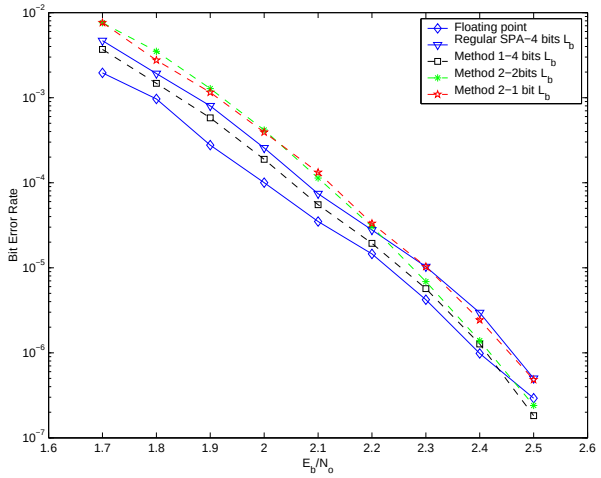

Fig. 7. Bit error plot of a (3,6) rate 0.5 LDPC code of length 2040

a large difference in L_c message magnitude. Hence a larger step size for less reliable L_b messages i.e. grouping of two or more points in the uniform quantization representation in this region would be acceptable. When this scheme is combined with Method 1, we need to store only 1 or 2 bits to represent L_b magnitude per check node. This is given by,

$$\begin{aligned} M2(j) &= M1 - \psi(L_b^{q-1}(j)) \quad \text{if } |L_b^{q-1}(j)| \text{ is min and } < T \\ M2(i) &= M1 \quad \text{otherwise} \end{aligned} \tag{17}$$

Fig. 7 and Fig. 8 show the performance of the proposed methods for a rate 0.5 (3,6) LDPC code of length 2040 and rate 0.9 (3,30) LDPC code of length 7680. It can be seen that using method 1 with 4 bits of precision the BER performance is even slightly better compared to that of regular finite precision SPA. Method 2 with 2 bits and 1 bit gives a small loss of 0.05 db compared to method 1 and is slightly worse than regular SPA at lower SNR and almost gives same performance at higher SNR. Fig. 9 shows the average number of decoding iterations for various methods with 30 iterations as the maximum limit. It can be seen that the decoder convergence for method 1 is similar to that of a finite precision regular SPA implementation and for method2 it's slightly slower.

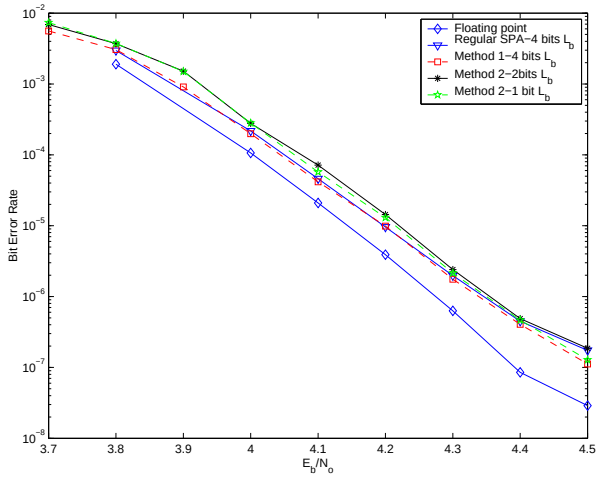

Fig. 8. Bit error plot of a (3,30) rate 0.9 LDPC code of length 7680

V. Parallelization and Scalability

An ensemble of LDPC codes is specified by its degree profile polynomials $\lambda(x) = \sum_{i=2}^{d_b} \lambda_i x^{i-1}$ and $\rho(x) = \sum_{j=2}^{d_c} \rho_i x^{j-1}$ where λ_i is the fraction of bit nodes of degree i and ρ_j is the fraction of check nodes of degree j. A regular LDPC code is a special case where all the bit and check nodes have same degree and hence the polynomials have a single term. Given the rate of the code, the degree distributions can be optimized to achieve near Shannon capacity for an AWGN channel [2]. Once the profile polynomials are known the connection between a bit and check node is typically made at random with constraints to avoid cycles of short length [24]. Random construction is not suitable for hardware implementation due to the following reasons. (i)The bit and

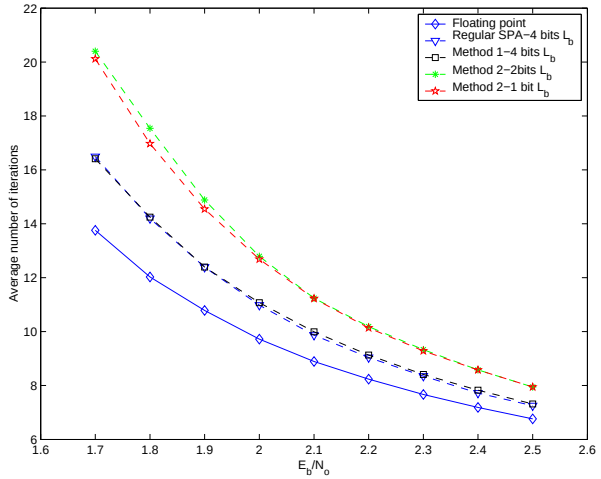

Fig. 9. Comparison of average number of iterations for various methods:max 30 itr, Rate 0.5 (3,6) LDPC code of length 2040

check node connection for each edge has to be stored at the decoder. (ii) Random connections lead to switching complexity for routing messages between bit and check nodes. (iii) Random connections will lead to memory access conflicts for partly parallel architectures since, for a partly parallel architecture with M check node update(CNU) units and M bit node update(BNU) units, M L_b/L_c messages will have to be read/written into the memory simultaneously.

A bit filling procedure for constructing the parity check matrix taking into consideration the parallelization constraints was proposed in our previous work [25]. This procedure has implementation difficulties due to reasons (i) and (ii). Recently, LDPC codes constructed from Circular Permutation Matrices(CPM) have been proposed [26], [27], [28]. Our architecture uses this family of codes and the heuristic construction procedure is as follows. (i) First construct a base matrix of size n columns and k rows such that the number of 1's in each column and number of 1's in each row is as given by the degree profile. The value of (n, k) are chosen to be as small as possible. Some heuristic such as bit filling can be applied to construct the base matrix. Let e be the total number of 1's in this base matrix. (ii) Expand the 1's in the base matrix with a cyclicly shifted identity matrix of size P and the 0's in the base matrix with an all zero square matrix of size P to get an LDPC parity check matrix of dimension $N = n \times P, K = k \times P$. (iii) The cyclic shift for the identity matrices are chosen to avoid cycles of short length(4,6,8,10) using the procedure given in [26].

Fig. 10 shows the procedure for illustration purpose with smaller dimensions. A circularly shifted identity matrix of size $P = 12$ is shown. If these P extrinsic messages(L_c or L_b) are stored together, a partly parallel architecture with parallelization $M = P$ can be realized without memory conflicts. The order in which these messages have to be delivered for the M BNU and the M CNU units differ only by a circular shift. A fixed M does not provide any scalability to the architecture as only a parallelization of P is achievable. To change the parallelization factor to a smaller number one has to increase the size of the base matrix and correspondingly decrease P. This would lead to an entirely new parity check matrix whose structure would be different for each parallelization factor and also the performance for each parallelization factor might be different. It is observed that parallelization factors smaller than P can be achieved for the same parity check matrix using column and row reassignment. Any parallelization factor that is a prime factor or product of prime factors of P is achievable. Rearrange the columns of the square matrix such that first we have all the columns i, such that $i \mod D = 0$, where $D = \frac{P}{M}$. Next we have columns which have modulus D as 1 and so on. Now perform a row rearrangement following a similar procedure. The circularly shifted identity matrix breaks up into smaller circularly shifted identity matrices. Let the shift for the larger block be S. In the equivalent base matrix representation each 1 will be replaced by an identity matrix of size D with a shift $A = S \mod D$. The circular shift for each of the $1's$ in this base matrix is given by $s = \lfloor \frac{S}{D} \rfloor = \lfloor \frac{S \times M}{P} \rfloor$ for the blocks before wrap around and $s+1$ for the blocks after wrap around. Hence we need to store only two values, namely

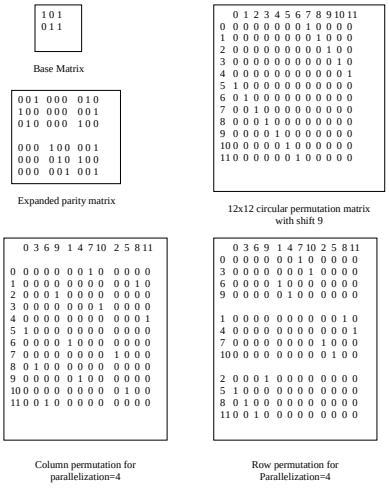

Fig. 10. Base Matrix Expansion and parallelization

A and s to operate the decoder at a lower parallelization level with the same decoding performance since the parity checks have not changed. Fig. 10 illustrates the procedure for $P = 12$, $M = 4$, $S = 7$. Here $D = 3$ and therefore $A = 1$, $s = 2$. M can take values 2,3,4,6,12.

The cyclic shift between the bit nodes and check nodes can be achieved with a logarithmic shifter/Omega network. These networks achieve a cyclic shift on N values using $\log_2(N)$ stages of shifting with each stage employing N $2:1$ multiplexers. In our code construction procedure we can add constraints for selecting S such that the resulting codes requires lesser routing complexity between the check and bit nodes or vice versa. When P is large and ρ_{max} is not large it might be possible to restrict the values of S to less than $\frac{P}{t}$ for $t \geq 1$ instead of choosing from the whole range 0 to $P-1$ without any significant loss is performance. The maximum cyclic shift in such cases required for the partly parallel decoder is given by $(\lfloor \frac{S \times M}{P \times t} \rfloor + 1)$. A decoder implementation where the difference in shift between two adjacent permutation blocks is used instead of the absolute shift on a single block was presented in [29]. A logarithmic shifter with only two stages can produce shifts from 0 to 3. One can use such a router with only two stages by choosing the shift on the two blocks to differ by less than $3D$. Array codes [28] are highly structured class of codes for which the shift between adjacent blocks in a row is a constant and given by the row number. A two stage shifter is enough as long as the number of rows i.e. the column weight of the code is less than $3D$. The above procedure eliminates the need for an alignment and reverse alignment unit as used in [9] which would require additional muxing and storage.

VI. Decoder Architectures

In this section, we compare partially parallel architectures based on the scheduling used as they differ in memory requirements and implementation. Based on the direction of scheduling used, we broadly classify partly parallel decoder architectures into two types namely, column schedule and row schedule architectures. In column schedule architectures, the

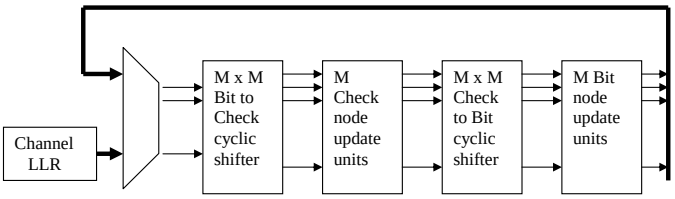

Fig. 11. Top level block diagram of the partly parallel decoder architecture

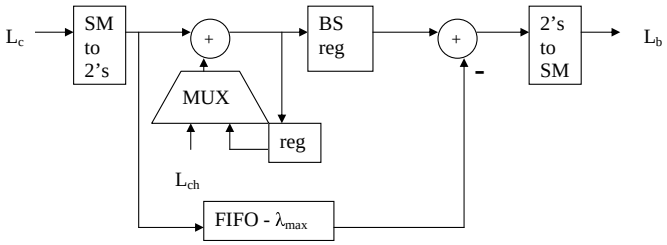

Fig. 12. Implementation of Bit Node Update unit for column schedule architecture

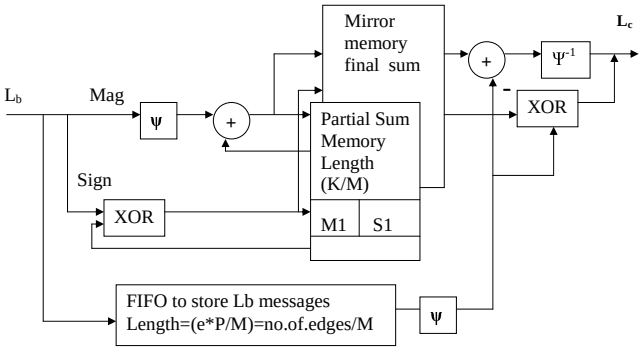

Fig. 13. Implementation of Check Node Update unit for column schedule architecture using flooding update and regular SPA

processing takes place column wise in the parity check matrix or in other words, the bit-node update unit receives the λ L_c messages in parallel (parallel-type) or consecutively one by one in succession (serial-type). In row schedule architectures, the processing takes place row wise in the parity check matrix or in other words, the check-node update unit receives the λ L_b messages in parallel or one by one in succession.

The row or column schedule architectures can again be sub-divided into two types namely, flooding or layered schedule architectures. In flooding scheduling, all the BNUs are completed before making any CNUs and vice versa. In layered/turbo scheduling [30], [31], [32], [33], only a set of BNUs are performed and the newly formed L_b messages are used in the CNUs in which they participate. Now, the newly formed L_c messages are used in the BNUs in which they participate and this procedure continues. It has been shown that the layered architecture improves the convergence speed of the decoder and, hence, offers lesser latency [33], [30]. The decoder power consumption also decreases since the average number of iterations for a packet decreases.

A communication standard might use LDPC codes of various rates and lengths depending upon the estimate of the channel condition. The LDPC decoder at the receiver must then handle codes of various rates and lengths. Hence, our implementation of both BNU and CNU units are independent of λ and ρ. Fig. 11 shows the top level block diagram for the partly parallel decoder architecture with parallelization M. It consists of M serial-type CNU units and M serial-type BNU units. Such a structure was also used in [10] for decoding regular LDPC codes. The L_c and L_b messages are routed to the appropriate update units with cyclic shifters. The same set of units are used on a time division basis for performing BNU and CNU for all bit-nodes and check-nodes and also for all the decoding iterations. We have an iteration multiplexer (MUX) that feeds in L_{ch} messages as L_b messages during first iteration for the CNU units. We now describe the CNU and BNU units in detail for various scheduling schemes and algorithms.

A. Column Schedule Architectures

1) Regular SPA-Flooding Schedule:

a) Bit-Node Update Unit: Fig. 12 shows the BNU unit for a column schedule architecture. The L_c message from the CNU unit is added together with L_{ch} to form L_{soft} as given by (9). Due to column scheduling, the λ successive L_c inputs would belong to the same bit-node. Once the L_{soft} is obtained, it is transferred to a bit sum(BS) register which helps compute (9) and (11) of two different bit node updates in parallel. The latency for the BNU unit is λ cycles and we have M such units in the architecture. The memory requirements for this unit is equal to the latency of the unit required to store the Lc messages which is quite small.

b) Check-Node Update Unit: Fig. 13 shows the CNU unit for a column schedule architecture. Since the successive L_b inputs belong to different check nodes, the L_c values are produced only after accumulating the L_b values over an iteration and, hence, the latency of this unit is equal to one iteration. Each decoding iteration requires storing L_c and L_b messages which are equal to the number of 1's in the parity check matrix. In our column schedule architecture, we derive L_c messages from $(M1, S1)$ and L_b values and hence do not store them. The number of $(M2, S2)$ values required is equal to the number of parity checks, which is smaller when compared to the number of 1's in the parity check matrix and decrease with the rate of the LDPC code. A pipelined architecture for the iteration takes place as following. Using the $(M1, S1)$ values of previous iteration and the L_b messages stored in the FIFO we find $(M2, S2)$ and hence L_c according to(7) and (8). These messages are passed to the bit-node update units and the updated L_b messages are used for forming the partial $(M1, S1)$ values for the present iteration. The fully formed $(M1, S1)$ values are transferred to the mirror memory. The storage requirements include storing the partial and final $(M1, S1)$ values for each parity check and L_b message for each 1 in the parity check matrix. This forms the bulk of the memory in the decoder hardware.

2) Modified SPA-Flooding Schedule: Since there is no change in the bit node update process, the structure of BNU

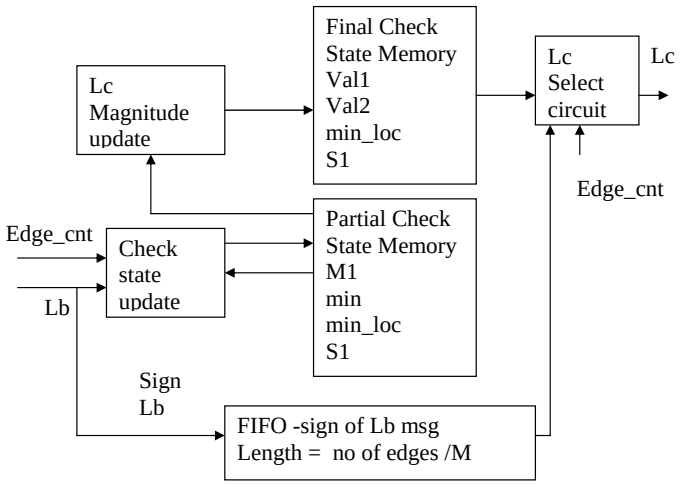

Fig. 14. Implementation of modified SPA Check Node Update unit for column schedule architecture using flooding update

unit remains the same as described for regular SPA.

a) Check-Node Update Unit: Fig. 14 shows the CNU unit for the proposed modified SPA decoder employing Method 1. The concept of mirror memory remains the same as in regular SPA but the values stored are different. For the present iteration we store partial values for $(M1, S1)$ and in addition we store the magnitude of the present minimum L_b message and its index. The number of bits required for storing the index is given by $\lfloor \log_2(\rho) \rfloor$ bits. Due to the proposed method, L_c, which is computed from $M2$ can take only two different values for the magnitude according to(17) and needs only the magnitude of the lowest L_b value. The L_c magnitude update block computes these two different values and stores them for each parity check in the 'final check state' memory. While there is a small increase in the quantities stored in the partial and final check state memory compared to regular SPA, the FIFO now stores only the sign of L_b values and this forms the bulk of the memory reduction. Since only two different L_c values are to be found we can use the ψ and adder circuits in $M1$ computation unit for $M2$ computation unit on a Time Division Multiplexing basis(TDM). Also the two ψ^{-1} operations can be shared on TDM basis. The cost of TDM in terms of throughput is an additional clock cycle for every ρ clock cycles. Hence, we have a throughput reduction of $\frac{1}{\rho}$ and therefore the loss in throughput is insignificant for higher rate codes when ρ is large. The edge_cnt value used to identify the min location can be generated with a clock for regular LDPC codes. For irregular LDPC codes 'e' edge_cnt values have to be stored at the decoder. The savings in memory compared to regular SPA is given by,

$$S_{MSPA} = \left(1 - \frac{3(L_b - 1) + 2\lceil \log_2 \rho \rceil + \rho - (\psi + 1)}{\rho \times L_b}\right) \times 100\% \tag{18}$$

3) Min-Sum decoding with offset correction-Flooding Schedule: MS decoding is another reduced complexity decoding and has the property of having only two different magnitudes for L_c messages as defined by (14). Hence, we can use the same techniques used in the implementation of modified SPA to achieve reduced memory implementation. The BNU unit remains the same as in regular SPA except for the addition of an offset correction unit. When the column degrees are regular this correction unit can be eliminated by lumping together with the correction term used in check update unit.

a) Check-Node Update Unit: The concepts are similar to the ones used in modified SPA but the values stored for each check and the updating circuits differ. In the partial check state memory we store min1, min2, location of min1 and the XOR of sign bits S1. The offset correction is performed on min1 and min2 values and stored in the final state memory. For regular LDPC codes the bit node update unit offset term is lumped together with the check update unit offset term. For irregular LDPC codes, with constant check node degree but with different bit node degrees the check offset correction unit can be completely removed and the correction term lumped with bit node update offset unit. The proposed architecture for MS decoding requires much lesser memory compared to the implementation for MS decoding by Karkooti *et al.* [12] and Guilloud *et al.* [17] where all the messages are stored at the decoder instead of only two different messages per check node. The savings in L_b memory compared to SPA is giving by,

$$S_{MS} = \left(1 - \frac{4(L_b - 1) + 2\lceil \log_2 \rho \rceil + \rho - 2(\psi + 1)}{\rho \times L_b}\right) \times 100\% \tag{19}$$

4) Regular SPA-Layered scheduling: For layered scheduling at the check node only computation and storage of $M1$ and $S1$ change from flooding scheduling (6). At any time instant, the $M1$ and $S1$ values for a check node depend on L_b messages from the present iteration as well as the previous iteration. This is given by,

$$M1_{lay} = \sum_{k=1}^{k=\rho-t} \psi(L_b^q(k)) + \sum_{k=t}^{k=\rho} \psi(L_b^{q-1}(k)) \tag{20}$$

A similar equation applies for $S1$.

From $(M1, S1)$ we find $(M2, S2)$, L_c^q and perform the bit node update to find L_b^q using(7)-(11). With the new L_b^q we update $(M1, S1)$ which is given by,

$$M1_{lay} = \sum_{k=1}^{k=\rho-t+1} \psi(L_b^q(k)) + \sum_{k=t+1}^{k=\rho} \psi(L_b^{q-1}(k)) \tag{21}$$

Since there is no concept of partial and final $(M1, S1)$, we have only one memory to store the present state of $(M1, S1)$. Since we add and subtract values from $M1$ no saturation of values is possible. Hence, the number of bits to represent $M1$ would be $\lceil \log_2(\rho) + \psi \rceil$ bits compared to $\psi + 1$ bits used in the parallel flooding update. Hence the width of the ψ adders and subtractors are larger. This is the cost for layered decoding at the check-node and the gain is faster convergence of the decoder leading to lesser latency and average number of iterations.

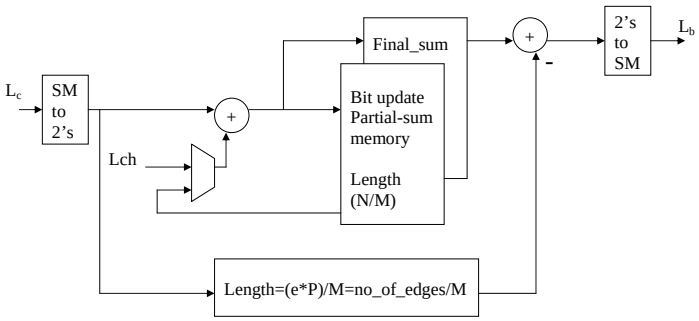

Fig. 15. Implementation of Bit Node Update unit with row scheduling and flooding update

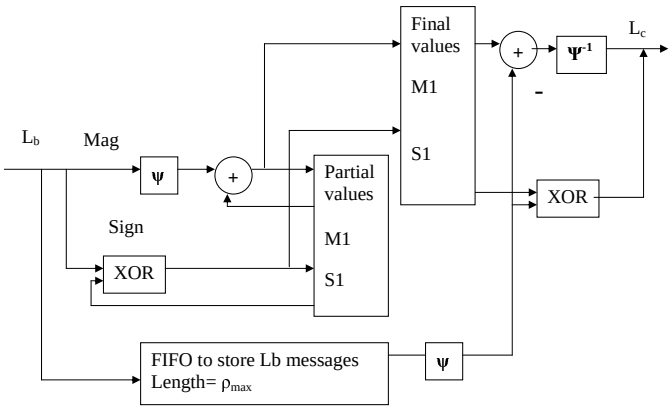

Fig. 16. Implementation of Check update unit with row scheduling

5) Modified SPA and Min-Sum decoding-Layered scheduling: Since we store only the min(L_b) message for each check which is different for each iteration a minimum for L_b messages from two different iterations is not possible. Hence, modified SPA and MS decoding with layered updating procedure at the check node is not possible and layered updating procedure at bit node is usually preferred.

B. Row Schedule Architectures

1) Regular SPA-Flooding Schedule:

a) Bit-Node Update Unit: In row scheduling, the consecutive L_c inputs to the bit node update unit do not belong to the same bit node and, hence, we need bit node memory to store the intermediate partial sum as in (9). Fig. 15 shows the implementation of the BNU unit. As the L_c messages enter the bit node update unit they are added to the corresponding partial sums to form the L_{soft} output. Once the soft output given by (9) is formed they are transferred to the final sum memory. The L_b values are formed according to (11). The BNU partial sum and final sum memories are equal to the length of the codeword rather than the number of checks in column scheduling. Hence, the memory requirements for row scheduling is much higher than that of column scheduling. The number of bits required to represent the L_{soft} message is given by $\lceil \log_2((\lambda+1)/2) + L_c \rceil$ bits. This is the maximum requirement for the case when half the L_c messages are of one sign, maximum magnitude and get accumulated successively and the rest of the messages are of the opposite sign and maximum magnitude. This is a very rare case and typically $L_c + (1 \text{ to } 2)$ bits is good enough for the decoder to perform without any noticeable loss in BER. For a regular LDPC code we can implement the bit node memory as a single block with same width since all the bit nodes have same degree and, hence, require same number of bits for L_{soft}. Irregular codes typically have few columns with very high degree. Hence, the memory corresponding to this column should be implemented separately and multiplexed with the block with lower bit degree. The size of the FIFO containing L_c messages is equal to the number of 1's in the parity check matrix. This forms the bulk of the hardware memory and hence we attempt to reduce it.

b) Check-Node Update Unit: In row scheduling the successive L_b inputs to the check-node update unit would belong to the same check-node. Fig. 16 shows the implementation for a CNU The structure is same as the CNU unit used for column scheduling(Fig. 13) except that there is no memory but just registers for partial and final $(M1, S1)$ values. The size of the FIFO is ρ_{max} which is the maximum latency for the unit. It should be noted that none of the combinational circuits in this unit can be clocked for pipelining. It will add to the latency for every check update. In the case of column scheduling the successive updates were on different check-nodes and hence clocking of combinational logic would add latency once for the whole iteration and not for each check-node update.

The Modified SPA and MS algorithms can be implemented for flooding schedule to decrease the size of Lc FIFO in the BNU unit by storing only the signs. The layered decoding model at the bit node can be implemented for regular SPA decoding to reduce memory requirements by combining together the partial sum, final sum and Lch memory. Further memory reduction can be achieved for layered decoding at bit node by implementing the modified SPA or MS algorithms that will reduce the L_c FIFO size by storing only the signs. We explain in detail the working of the modified SPA decoder with layered scheduling at the bit-node and the implementation for MS decoding is quite similar.

2) Modified SPA-Layered Scheduling: Here, a set of CNUs are made (current layer) and the L_c values are used in the BNUs they participate in. Hence the L_{soft} values depend on the new L_c messages from the present layer as well as the old L_c messages from previous layers that participate in the BNU.

Fig. 17 shows such an implementation. The L_b inputs to a CNU unit are derived according to (11). The L_{soft} value is read from the bit-sum memory and L_c message is derived from the contents of check value memory. Since L_c can have only two different magnitudes we need to store only $value1 = \psi^{-1}(M1)$, 2's complement or SM representation of $value2 = \psi^{-1}(M1 - \psi(L_b^{q-1}(min)))$ and derive $|L_c|$. The index of $L_b^{q-1}(min)$ is also stored to select between value1 and value2. The signs of L_b messages are stored separately and their XOR with $S1$ is used to derive the sign of L_c messages. Hence, we have achieved memory reduction by not storing all the L_c messages which is proportional to the number of

1's in the parity check matrix but only storing few quantities for each check node and signs of all the L_c messages. The M L_b messages undergo a cyclic shift to route them to the correct CNU unit and also stored back in the bit-sum memory. During the first iteration L_{ch} values are used instead of the derived L_b values. Since we have row scheduling, the successive L_b^q messages belong to the same check node and hence we need only registers to store the temporary check state values. Compared to the flooding column schedule the only difference in this unit is that we also store the sign of $L_b^q(min)$ message in the register. Once the final check state values are transferred to the mirror register, we find value1 and value2, which are used to find the two new L_c message magnitudes. The L_c messages are added with the L_b messages read from the bit-sum memory to obtain new L_{soft} messages that are stored back into the memory. The process is repeated for CNU's in all the layers. We also have memory reduction in the bit-sum memory since two copies are not needed as required in flooding scheduling and also the L_{ch} values are combined into them. It should be noted that the bit-sum memory should support two read operations and two write operations in parallel for all the different codes decoded with this architecture. Hence, the single block of bit-sum memory of length $\frac{N}{M}$ and width M, must be split into 4 or more smaller blocks for single port implementation and 2 or more smaller blocks for dual port implementation and each of the write and read operations should access a different block of memory. The ordering of check node processing must be chosen to minimize memory access conflicts which would help reduce the number of partitions required for the single block. When such a splitting of the memory is not desired, one can have a separate memory block for L_{soft} messages and another FIFO for L_b messages. Such an implementation would require an additional memory block of length ρ to store the L_b messages, which is equal to the latency of CNU unit.

For regular flooding schedule, during the first iteration, all L_c messages are zero and hence $L_b = L_{ch}$. In turbo scheduling, during the first iteration, $L_b = L_{ch}$ only for the first edge of a bit-node. For the other edges of a bit-node, L_b depends on both L_{ch} and L_c messages available from the layers already processed. An optional cyclic shifter to shift L_{ch} and a MUX is needed as shown in Fig. 17, when the first iteration has regular flooding schedule and the rest of the iterations follow turbo scheduling. For codes with high column degree this gives a savings of 15% or more on the average number of iterations required. If turbo scheduling is used for all the iterations the optional MUX and shifter are not required and, hence, only one cyclic shifter is required as compared to flooding schedule which requires two cyclic shifters.

The savings in L_c memory over regular SPA layered decoding for both modified SPA and MS decoding is given by,

$$S_{lay} = \left(1 - \frac{2(L_b - 1) + 2\lceil \log_2 \rho \rceil + \rho}{\rho \times L_b}\right) \times 100\% \quad (22)$$

We have used serial bit-node update and check-node update units to provide flexibility to the architecture so that the same

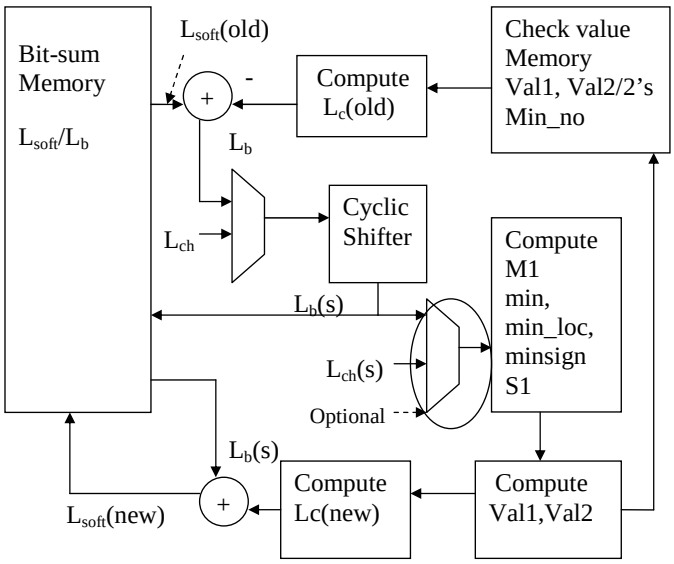

Fig. 17. Implementation of Modified SPA with row scheduling and layered update procedure

decoder can be used for regular codes, irregular codes, codes of different rates and lengths. If flexibility is not a major issue we can use parallel check-update units. To acheive a parallelization of M we need $\frac{M}{\rho}$ parallel-type CNU units.

We summarize the check node operations and memory requirements for our various implementations and decoding algorithms in tables I and II. We present results for regular LDPC codes for easier comparison and understanding. The requirements for irregular LDPC codes are straightforward extensions.

The number of operations for check node update is the least for MS decoding algorithm. The number of LUT operations for ψ^{-1} and SM to 2's complement conversion are lesser for modified SPA algorithm compared to regular SPA algorithm. This is due to the fact that we have only two different L_c message magnitudes for the modified SPA algorithm. The extra comparison operations for the modified SPA algorithm take place in parallel with the addition/subtraction operation and, hence, would not affect the critical path for timing requirements.

We find that, for flooding scheduling the memory requirements for the row update architecture is more than that of the column update architecture. This is due to the fact that the memory required for storing partial update values for the row update architecture is equal to the length of the code whereas, for the column update architecture, it is equal to the number of parity checks which is smaller and decreases with the rate of the code. For layered scheduling architecture it would be better to use row update architecture since the partial update memory can be combined with the L_ch memory. Moreover, memory efficient modified SPA and MS decoding are not available for column update layered scheduling architectures.

The combinational logic used in the check update unit can easily support clock speeds of greater than 100 MHz. The bit-update and check-update units are made of simple combinational logic blocks such as adders, subtractors, comparators

and look up tables. Parallelization of update units in the order of few tenths to a hundred can easily support a decoding throughput of few hundred Mbits for LDPC codes of various redundancy rates with 10-20 decoding iterations. The number of parallel computation units does not depend on the length of the LDPC code whereas, the memory requirements depend of the length of the LDPC code used. For achieving good coding gain, long LDPC codes should be used. The hardware decoder implementation for these systems would be dominated by the memory rather than the logic used. Even for systems using medium length LDPC codes (2k-5k length) and requiring decoding throughput of few hundred Mbits, the savings in memory area is significant for the overall decoder. Hence, we have focussed mainly on reducing the memory in the decoder for the partially parallel architecture.

A CNU unit that exploits the properties of MS algorithm for memory savings in layered decoding was recently proposed by Bhatt *et al.* [34] and Gunnam *et al.* [11]. Our work on memory efficient architectures was initially proposed in [15], [35] which precedes the publication of [34], [11].

VII. Conclusion

In this paper we have presented a modified SPA that has lower complexity and results in only two different L_c message magnitudes. The modification results in no noticeable loss compared to the performance of regular SPA. The clipping and precision issues related to the implementation of SPA decoding algorithm was studied. Scalability and parallelization was investigated for cyclic permutation block based LDPC codes. Architectures for regular SPA, modified SPA and Min-Sum decoding with various scheduling schemes were presented. For modified SPA and MS decoding only two different message magnitudes per check node were stored at the decoder and this was achieved by storing only one type of message, either L_b or L_c message at the decoder and the other type of message was stored in the form of check state or bit sum. Memory reduction increases with the rate of the code as we have fewer checks for a given frame length. The memory requirements for layerd scheduling is lower than that of the regular flooding schedule as there is no need for the mirror memory. To conclude, we have provided architectures that require lesser memory than previously published results.

References

[1] R.G. Gallager. Low-density parity check codes. *IRE Trans. Inform. Theory*, vol IT-8, pp 21-28, Jan 1962.

[2] S.Y. Chung, T.J. Richardson, R.L. Urbanke. Analysis of Sum-Product Decoding of Low-Density Parity-Check Codes Using a Gaussian Approximation. *IEEE Trans. Inform. Theory*, vol 47, no. 2, Feb 2001.

[3] S. Sivakumar. VLSI Implementation of Encoder and Decoder for Low Density parity check codes. *Masters Thesis,Texas A&M University*,December 2001.

[4] A.J. Blanksby, C.J. Howland. A 609-mW 1-Gb/s 1024-b, rate 1/2 low density parity-check code decoder. *IEEE journal on solid state circuits*,March 2002, Pages 402-412, vol:37, Issue 3.

[5] E. Yeo, P. Pakzad, B. Nikolic, V. Anantharaman. High throughput Low-Density Parity-Check decoder architectures. *IEEE Global Telecommunication Conference,2001. GLOBECOM'01*, vol:5, pages:3019-3024.

[6] E. Liao, E. Yeo, B. Nikolic. Low-density parity-check code constructions for hardware implementations *IEEE Intl Conf on Communications, ICC 2004*, vol 5, 20-24 June 2004, pages: 2573-2577.

[7] H. Zhong, T. Zhang. Block-LDPC: A Practical LDPC Coding System Design Approach. *IEEE Trans. on Circuits and Systems-I*, Vol 52, No 4, April 2005, pages:766-775.

[8] T. Zhang, K.K. Parhi. A 54 MBPS (3,6)-regular FPGA LDPC decoder. *IEEE Proc. of SIPS*, pp.127-132, 2002.

[9] Y. Chen, D.E. Hocevar. A FPGA and ASIC implementation of rate 1/2, 8088-b irregular low density parity check decoder. *IEEE Global Telecommunciations Conference, 2003. GLOBECOM'03*, vol:1, 1-5 Dec.2003, pages:113-117.

[10] K. Gunnam, G. Choi, M. Yeary. An LDPC decoding shedule for memory access reduction. *Acoustics, Speech, and Signal Processing, 2004. ICASSP 04*, vol:5, 17-21 May 2004, pages:173-176.

[11] K. Gunnam, G. Choi, W. Wang, E. Kim, M.B. Yeary. Decoding of Quasi-cyclic LDPC Codes Using an On-the-Fly Computation. *Asilomar Conference on Signals, Systems and Computers,2006. ACSSC 2006*, Oct-Nov 2006, pages: 1192-1199.

[12] M. Karkooti, J.R. Cavallaro. Semi-parallel reconfigurable architectures for real-time LDPC decoding. *International Conference on Information Technology: Coding and Computing 2004, ITCC 2004*, vol:1, pages: 579-585.

[13] P. Radosavljevic, A. de Baynast, M. Karkooti, J.R. Cavallaro. Multi-Rate High-Throughput LDPC Decoder: Tradeoff Analysis Between Decoding Throughput and Area *IEEE international Symposium on Personal, Indoor and Mobile Radio Communications,2006.* Sept. 2006, pages: 1-5

[14] T. Brack, F. Kienle, N. Wehn. Disclosing the LDPC code decoder design space. *Design, Automation and Test in Europe, 2006. DATE 06*, 6-10 March 2006, vol1:1, pages: 1-6.

[15] A. Prabhakar, K. Narayanan. A Memory Efficient Serial LDPC Decoder Architecture. *IEEE International Conference on Acoustics, Speech and Signal Processsing. ICASSP 2005*, vol: 5, March 18-23, pages:41-44.

[16] C. Jones, E. Valles, M. Smith, J. Villasenor. Approximate-Min Constraint Node Updating for LDPC code design. *IEEE conference on Military Communications, 2003. MILCOM 2003*, 13-16 Oct 2003, pages:57-162.

[17] F. Guilloud, E. Boutillon, J.L. Danger. λ-Min Decoding Algorithm of Regular and Irregular Codes. *Proceedings of the 3nd International Symposium on Turbo Codes & Related Topics*,Brest,France, Sept 2003.

[18] J. Chen, M.P.C. Fossorier. Near optimum universal beleif propagation based decoding of low-density parity check codes. *IEEE Trans. on Communications*, vol 50, Issue 3, March 2002, pages: 406-414.

[19] J. Zhao, F. Zarkeshvari, A.H. Banihashemi. On the implementation of min-sum algorithm and its modifications for decoding low-density Parity-check codes. *IEEE Trans. on Communications*, vol 53, Issue 4, April 2005, pages: 549-554.

[20] J. Chen, A. Dholakia, E. Eleftheriou, M.P.C. Fossorier, X.Y. Hu. Reduced-Complexity Decoding of LDPC Codes. *IEEE Trans. on Communications*, vol 53, Issue 8, Aug 2005, pages: 1288-1299.

[21] L. Ping, W.K. Leung. Decoding low density parity check codes with finite quantization bits *IEEE Comm. Letters*, Vol 4, Issue 2, Feb 2000. Pages: 62-64.

Decoding Algorithm	Column scheduling	Row scheduling
SPA	$\psi = 2\rho$ $\psi^{-1} = \rho$ $ADD/SUB = 2 \times \rho$ SM to 2's= ρ 2's to SM= ρ	$2\psi = \rho$ $\psi^{-1} = \rho$ $ADD/SUB = 2 \times \rho$ SM to 2's= ρ 2's to SM= ρ
Mod-SPA	$\psi = \rho + 1$ $\psi^{-1} = 2$ $ADD = \rho$ $SUB = 1$ $\leq -comp = \rho$ $eq - comp = \rho$ SM to 2's= ρ 2's to SM= ρ	$\psi = \rho + 1$ $\psi^{-1} = 2$ $ADD = \rho$ $SUB = 1$ $\leq -comp = \rho$ $eq - comp = 2 \times \rho$ SM to 2's= 3 2's to SM= ρ
MS-offset	$SUB = 2$ $\leq -comp = 2 \times \rho$ $eq - comp = \rho$ SM to 2's= ρ 2's to SM= ρ	$SUB = 2$ $\leq -comp = \rho + 1$ to 2ρ $eq - comp = 2 \times \rho$ SM to 2's= 3 2's to SM= ρ

TABLE I

Number of operations for each check update

Decoding Algorithm	Flooding scheduling	Layered scheduling
SPA-Column	$L_{ch} = n \times L_{ch}$ check=$2K((\psi+1)+1)$ check-fifo=$e \times P \times L_b$ bit-fifo=$M \times \lambda \times L_c$	$L_{ch} = n \times L_{ch}$ check=$(K + M\lambda)(\psi + \log_2 \rho + 1)$ check-fifo=$e \times P \times L_b$ bit-fifo=$M \times \lambda \times L_c$
Mod-SPA Column	$L_{ch} = n \times L_{ch}$ check=$K(3 \times (L_b - 1) + 2 + (\psi + 1) + 2 \times \log_2(\rho))$ check-fifo=$e \times P$ bit-fifo=$M \times \lambda \times L_c$	NA NA NA NA
Mod-Sum-Corr Column	$L_{ch} = n \times L_{ch}$ check=$K(4 \times (L_b - 1) + 2 + 2 \times \log_2(\rho))$ check-fifo=$e \times P$ bit-fifo=$M \times \lambda \times L_c$	NA NA NA NA
SPA-Row	$L_{ch} = n \times L_{ch}$ bit-sum=$2N(\log_2((\lambda+1)/2) + L_c)$ bit-fifo=$e \times P \times L_c$ check-latency-fifo=$M \times \rho \times L_b$	$L_{ch} = n \times L_{ch}$ bit-sum=$(N)(\log_2((\lambda+1)/2) + L_c)$ bit-fifo=$e \times P \times L_c$ check-latency-fifo=$M \times \rho$
Mod-SPA Row and Min-Sum-Corr Row	$L_{ch} = n \times L_{ch}$ bit-sum=$2N(\log_2((\lambda+1)/2 + L_c)$ checks-sign-fifo=$e \times P$ check-latency-fifo=$M \times \rho$ check-state-mem=$K(2 \times (L_b - 1) + 1 + \log_2(\rho))$	bit-sum=$(N)(\log_2((\lambda+1)/2 + L_c)$ check-sign-fifo=$e \times P$ check-latency-fifo=$M \times \rho$ check-state-mem=$K(2 \times (L_b - 1) + 1 + \log_2(\rho))$

TABLE II

MEMORY REQUIREMENT FOR VARIOUS ALGORITHMS AND SCHEDULES

[22] T. Zhang, Z. Wang, K.K. Parhi. On finite precision implementation of low density parity check codes decoder. *Circuits and Systems 2001, ISCAS' 01*, Volume 4, 6-9 May 2001. Pages: 202-205 vol.4.

[23] J. Zhang, M. Fossorier, D. Gu, J. Zhang. Two-dimensional correction for min-sum decoding of irregular codes. *IEEE Communication letters*, vol 10, Issue 3, March 2006, pages: 180-182.

[24] J. Campello, D.S. Modha, S. Rajagopalan. Designing LDPC codes with bitfilling. *IEEE International Conference on Communications,2001. ICC 2001*, vol:1, 11-14 Jun 2001, pages: 55-59.

[25] A. Selvarathinam, G. Choi, K. Narayanan, A. Prabhakar, E. Kim. A massively scalable architecture for low-density parity-check codes. *Circuits and Systems 2003, ISCAS' 03*, Volume 2, 25-28 May 2003. Pages:II-61-II64 vol.2.

[26] M.P.C. Fossorier. Quasicyclic low-density parity-check codes from circulant permutation matrices *IEEE Trans. on Information Theory*, vol: 50, Issue 8, Aug 2004, pages: 1788- 1793.

[27] B. Vasic. High-rate low-density parity check codes based on anti-Pasch affine geometries. *IEEE international conference on communications,2002. ICC 2002*, vol: 3, 28 Apr- 2 May 2002. Pages: 1332-1336.

[28] S. Olcer. Decoder architecture for array-code-based LDPC codes. *Global Telecommunication Conference,2003. GLOBECOM'03*, vol:4, pages:2046-2050.

[29] P.Bhagawat, M.Uppal, G. Choi. FPGA based implementation of decoder for array low-density parity-check Codes. *IEEE international conference on Acoustics, Speech and Signal processing,2005. ICASSP 2005*, vol:5, 18-23 Mar 2005, pages: 29-32.

[30] P. Radosavljevic, A. de Baynast, J.R. Cavallaro. Optimized Message Passing Schedules for LDPC Decoding. *Conference record of the Thirty-Ninth Asilomar Conference on Signals, Systems and Computers, 2005*, 28 Oct- 1 Nov 2005. Pages: 591-595.

[31] D.E. Hocevar. A reduced complexity decoder architecture via layerd decoding of LDPC codes. *IEEE worksop on signal processing systems, 2004. SIPS 04*, pages 107-112.

[32] E. Sharon, S. Litsyn, J. Goldberger. An efficient message passing shedule for LDPC decoding. *IEEE convention of electrical and electronics engineers in israel,2004*, 6-7 Sept 2004, pages: 223-226.

[33] M.M. Mansour, N.R. Shanbhag. Memory efficient turbo decoder architecture for LDPC codes. *IEEE workshop on Signal Processing Systems, 2002(SIPS'02)*, 16-18 Oct 02, pages:159-164.

[34] T.Bhatt, V. Sundaramurthy, V. Stolpman, D. McCain. Pipelined Block-Serial Decoder Arthitecture for Structured LDPC codes. *IEEE International Conference on Acoustics, Speech and Signal Processsing,2006*, vol: 4, May 14-29, pages:IV 225- IV 228.

[35] A. Prabhakar, K. Narayanan. Memory Efficient Scalable Decoder Architectures for Low Density Parity Check Codes. *Wireless Communication Laboratory Tech Report,2006*, ID: WCL-TR-06-104, May 12 2006, http://wclb.tamu.edu/research.html