Memory Efficient Scalable Decoder Architectures for Low Density Parity Check Codes

A. Prabhakar and K.R. Narayanan, Dept. of ECE, TAMU,
College Station, TX 77843

WCL Technical Report
WCL-TR-06-104

05/12/2006

Wireless Communications Laboratory
Department of Electrical and Computer Engineering
Texas A&M University, College Station

Memory efficient scalable decoder architectures for Low Density Parity Check codes

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

May 12, 2006

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 exploits the properties of a check constraint to reduce the storage of extrinsic messages which forms the bulk of the hardware. Extrinsic message 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.

1 Introduction

LDPC codes discovered by Gallager [1] have been the focus of intense research in recent years. It has been showed that LDPC codes can be designed to achieve communication at rates close to Shannon capacity limit [2]. Better coding gains and low complexity decoding 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. 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 and limits its implementation to codes of about thousand bits long. 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. The various architectures implement some of the most popular decoding algorithms such as Sum Product Algorithm(SPA), Min-Sum algorithm and their modifications to meet coding, throughput, latency, decoder area and power requirements.

A serial decoder architecture that uses regular SPA decoding algorithm and decodes one bit at a time was presented in [8]. A staggered decoding schedule which uses an approximation of belief propagation and requires memory dependant on the number of bits in the codeword was given by Yeo *et al* [17]. The staggered schedule suffers a coding loss compared to SPA and is suitable only for less than five decoding iterations. An hierarchial formulation of LDPC code that allows a structured realization of the decoder was presented in [16]. A joint LDPC code-encoder-decoder design approach called Block-LDPC taking into account hardware-oriented constraints was given by Zang *et al* [11]. A partly parallel implementation with a joint code-decoder design approach with 56 Mbps throughput was presented in [13]. A parallel decoder architecture with a decoding throughput of 1 Gbps was give by Blanksby and Howland [18]. A partly parallel decoder architecture for irregular LDPC codes was given by Chen and Hocevar [22]. Code design and parallelization concepts were explained in the paper. An LDPC decoding schedule for memory access reduction was presented in [21].

While most of the above references implement the SPA decoding algorithm using the log-tanh function, reduced complexity algorithms for LDPC decoding have been widely used. Issues related to implementation of Min-Sum algorithm and its modifications were explored by Zhao *et al* [24], Chen*et al* [5]. Recently Chen *et al* [6] have presented the various log-likelihood-ratio-based belief-propagation decoding algorithms and their reduced complexity derivatives for LDPC codes. The performance and complexity of various reduced complexity algorithms were compared in the paper.

In spite of the huge coding gain offered by LDPC codes compared to the presently used Forward Error Correction(FEC) systems based on Reed Solomn(RS) codes and BCH codes their implementation for wide range of applications have been limited due to the large hardware requirements of the LDPC decoder. We observe that the bulk of the hardware required for the serial and partly

parallel LDPC decoders lie in the memory used for extrinsic messages(bit to check or check to bit) and memory for storing partial update values and hence attempt to reduce it. The proposed reduced memory implementation based on the modified SPA algorithm allows intrinsic feedback of soft values at the check update unit. This proposed modification is similar to the Approximate-Min Constraint independently proposed by Jones *et al* [3] but we implement the approximation in log-tanh domain rather than implementing directly on the LLR values based on the Jacobian logarithmic identity. Also no specific decoder architecture has been proposed in [3]. The proposed decoder stores only few bit to check messages or check to bit messages which is very efficient when compared to architectures proposed in [13], [17] which store all the bit to check messages and check to bit messages and architectures proposed in [22], [21] which store either check to bit or bit to check messages. Memory efficient turbo decoding algorithms were proposed by Mansour *et al* [9] and Shankar *et al* [10]. Such a decoding architecture uses layered decoding approach to save memory and the approach adopted in [9] requires lesser number of iterations as the decoder converges faster compared to regular 2 way SPA scheduling. Layered decoding offers memory reduction only when a large portion of the graph is decoded in parallel. For serial/partly parallel architectures where a small portion of the graph is decoded there would be hardly any memory reduction. Hocevar [23] has also presented a reduced complexity decoder using layered decoding. Our proposed decoder with serial scheduling has faster convergence as presented in [9] and requires lesser memory. Guilloud et al [4] presented an offset min-sum decoding algorithm offering a tradeoff between performance and complexity to save extrinsic message memory. Our proposed architecture scheduling requires lesser memory compared to [4].

The next section introduces LDPC codes and SPA, Min-Sum decoding algorithms. The proposed finite precision scheme to implement ψ function in the SPA decoding is discussed in section 3. The proposed modified SPA algorithm is discussed in section 4. The concept of scalability and requirements for parallelization, memory mapping are discussed in section 5. The proposed architectures with parallel and serial scheduling are discussed in section 6. Section 7 summarizes and concludes the paper.

2 Introduction to LDPC codes

Let $a = a_0, a_1, a_2, ..., a_{K-1}$ be a K bit information word. Let the K bit information be mapped to a N bit codeword $c = c_0, c_1, c_2, ..., c_{N-1}$. Such a mapping for a linear block code is defined by a parity check check matrix having N columns and $N - K$ rows. LDPC codes are linear block codes with sparse parity check matrices(H matrix). Each column has λ number of 1's and each row has $\rho > \lambda$ number of 1's. If each row and column has same number of 1's it is referred to as a regular LDPC code. If the number of 1's differ between columns and rows its 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 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.

2.1 SPA decoding of LDPC codes

Although the decoding algorithm is well known, we explain this in detail since the proposed modification is based on this algorithm. Let us assume a BPSK modulation scheme where a 1 is mapped to $+1$ and 0 is mapped to -1 and an Additive White Gaussian Noise(AWGN) channel with mean '0' and variance σ^2. The received channel values are given by,

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

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

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

Let us define the function,

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

It can be shown that the functions $\psi(x)$ and $\psi^{-1}(x)$ are identical.

The decoding of LDPC codes is based on passing messages between bit nodes and check nodes along the edges through which they are connected in an iterative manner. The messages represent estimates of the coded bits (in LLR form) based on the received signals and from the parity check constraints. 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 n^{th} check node during the q^{th} iteration(we will not explicitly use the index n to denote quantities associated with the n^{th} node as the operations are identical at all nodes). Similarly, let $L_b^q(i)$ represent the bit to check message along the i^{th} edge connected to the n^{th} bit node during the q^{th} iteration.

Enforcing the parity check constraint on the incoming bit to check values, the check node update

is given by,

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

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

Where t is the degree of the check node and index k refers to the k^{th} edge connected to the check node. For serial implementation in hardware, Eq. 5 is divided into two steps. First, we find two quantities $M1$ and $S1$.

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

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

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

Now,

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

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

$$L_{soft}^n = L_{ch}(c_n) + \sum_k L_c^q(k) \quad (9)$$

The bit node update is given by,

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

Alternatively, the bit node update can be written as,

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

The n^{th} bit is decoded as 1 if $L_{soft}^n < 0$, else as 0.

2.2 Min-Sum Decoding

Min-Sum(MS) 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 Min-Sum algorithm. In Min-Sum 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 is given by,

$$|L_c^q(i)| = \min_{k=1,k\neq i}^{k=\rho} |L_b^{q-1}(k)| \tag{12}$$

To save the number of operations this can be written as,

$$\begin{aligned} |L_c^q(i)| &= L_b^{q-1}(min1) \quad \text{if i} \neq \text{min1 edge} \\ |L_c^q(i)| &= L_b^{q-1}(min2) \quad \text{if i} = \text{min1 edge} \end{aligned} \tag{13}$$

Where, min1 and min2 are the minimum and second least minimum. Clearly we can see that there are only two different magnitudes for the check to bit messages. 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 $\beta(|L_c|/\beta)$ or by adding a fixed correction factor $\alpha(|L_c| - \alpha)$ [5]. The correction factor can also be added on a conditional basis [24].

3 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 and Leung [15]. A parity likelihood function was proposed to improve performance. The effects of finite word length on log-BP based SPA decoding was previously studied by Zhang *et al* [12]. Non-uniform quantization was proposed for extrinsic values with both ψ and ψ^{-1} functions implemented using the same look up tables. The authors limit the range of LLR values between 0 and 4 and consider only rate 1/2 codes for their analysis. We observe that this limited dynamic range is suitable only for low rate codes and leads to error floors in high rate codes. We present schemes that are suitable for decoding a wide range of code rates.

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

We observe from simulations that for good performance of the decoder without error floors, (a) The range of L_c messages(output of ψ^{-1}) should be large. (b) The transformation $x = \psi^{-1}(\psi(x))$ should be preserved more accurately for larger values of x than smaller values of x. For (a) we need $\psi(x)$ to have very small quantization steps close to zero. Eg: For $\psi^{-1}(x) = 8$, we need $x = 6.4 \times 10^{-4}$. 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. The smaller the range of $\psi(x)$, higher will be the lower

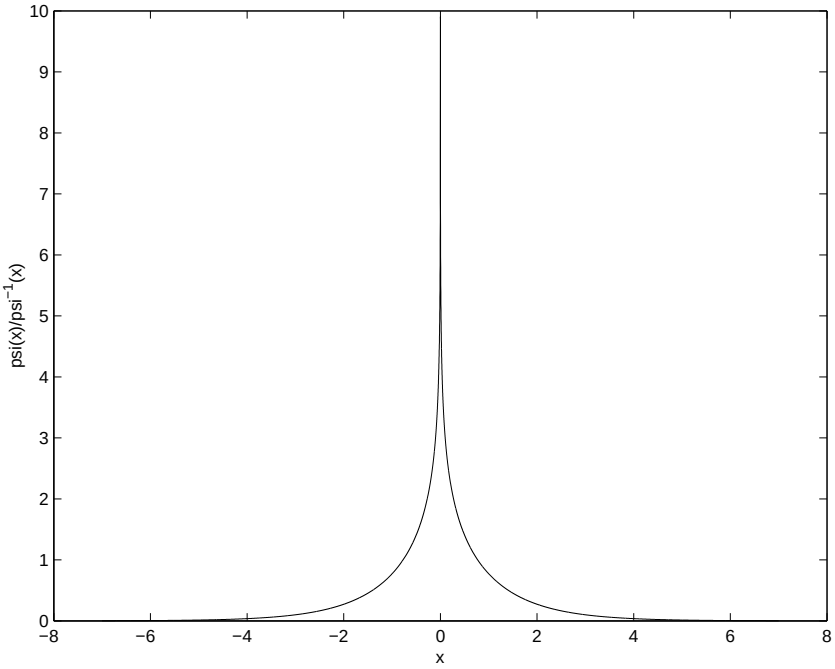

Figure 1: $\psi(x)/\psi^{-1}(x)$ function

end clipping for the L_c messages and this would lead to overestimation of lesser reliable(smaller) L_c messages and hence loss in coding performance. Eg: $\psi_{max}(x) = 4.0$ gives $L_c(min) = 0.037$, $\psi_{max}(x) = 1.0$ gives $L_c(min) = 0.78$ $\psi_{max}(x) = 0.5$ gives $L_c(min) = 1.4$. Hence the range of $\psi(x)$ should be carefully selected. 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 size of ψ LUT is linearly proportional to q whereas the size of $\psi^{-1}(x)$ is proportional to 2^q. The check update adder works only on uniform representation of ψ values and hence we need simple conversion circuits to covert from non-uniform representation to uniform representation and vice versa. We observe that there is very little performance gain for the decoder in using non-uniform quantization for L_c, L_b and L_{ch} messages and hence use only uniform quantization to represent these values and avoid any uniform/non-uniform conversion logic. First we propose uniform quantization schemes for ψ values. We limit the range of $\psi(x)$ to minimize adder size and to decrease quantization step size there by increasing the dynamic range of L_c messages for a given number of bits. Next we propose non-uniform quantization schemes for ψ values that have simple non-uniform/uniform conversion circuits. Even though mathematically $\psi(x)$ and $\psi^{-1}(x)$ transformations are identical we use different LUTs as the input/output range over which they operate is different.

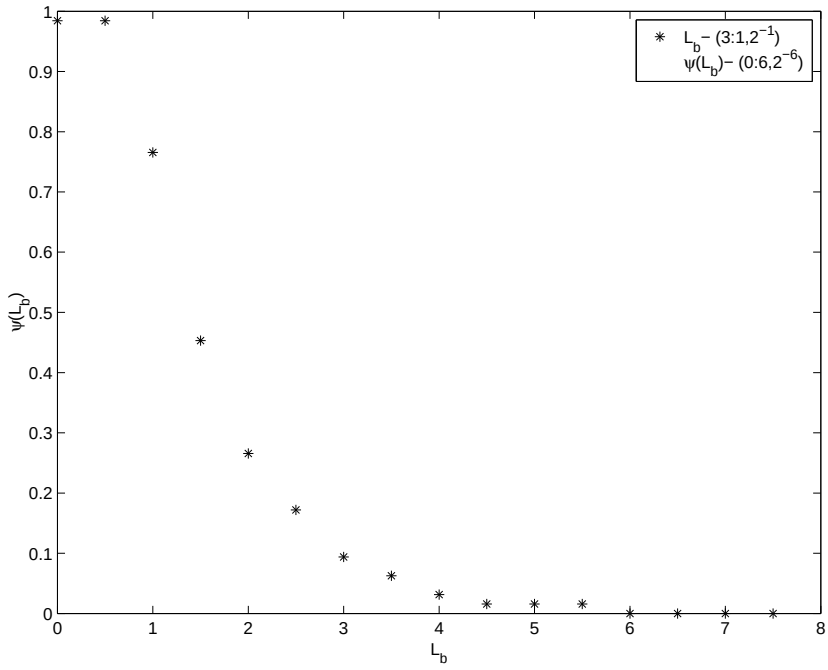

Figure 2: Finite precision ψ function

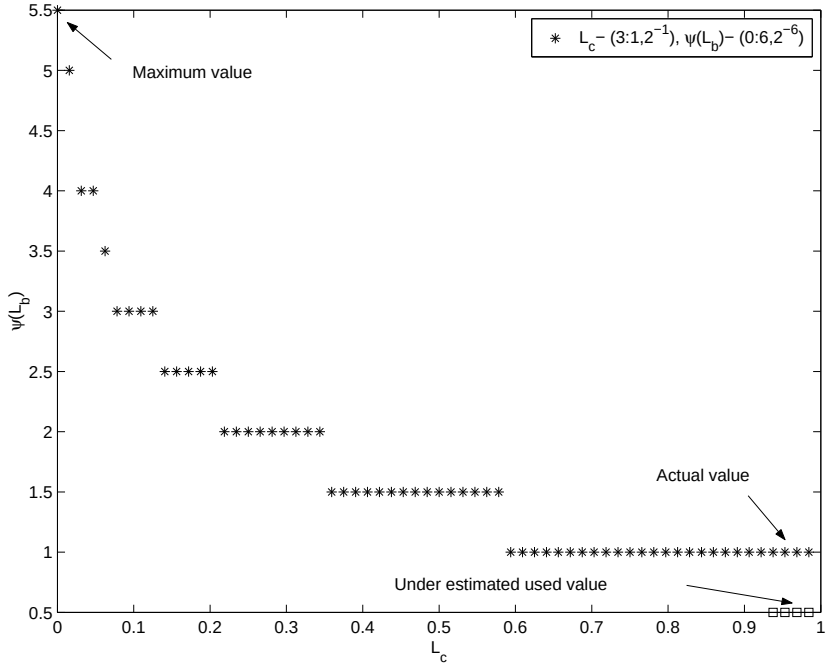

Figure 3: Finite precision ψ^{-1} function

3.1 Uniform Quantization

We use the following convention to represent finite precision in our paper. The number of bits used to represent a quantity does not include its sign bit. The word length for the quantity includes only the representation of magnitude. If the quantity has a sign then the sign bit is extra. For $\psi(L_b)$ there is no sign involved since all the quantities are negative and hence no sign bit. L_{ch}, L_b and L_c have sign and magnitude and hence one extra bit will be used for sign bit. A general uniform quantization will be defined by the number of bits N and range (L_{lower}, L_{upper}). The step size can be derived as $(L_{upper} - L_{lower})/N$. A special case of uniform quantization is where the step size is a power of two. Such a representation is defined by the pair $(N_i : N_f, 2^{-s})$, where N_i is the number of bits to represent the integer part and N_f is the number of bits to represent the fraction part and 2^{-s} is the step size. Fig. 2 shows the finite precision points for the $\psi(L_b)$ function. It can be seen that $\psi(0)$ and $\psi(0.5)$ have undergone clipping to the maximum value of $(1 - 2^{-6})$. Fig. 3 shows the corresponding ψ^{-1} function. The input range for this function is the range of $\psi(L_b)$ function. The output of this function never reaches zero and gets clipped to 1.0 due to the limited range of ψ values. This would lead to loss in performance and we solve this problem by underestimating the last 4 values to 0.5 rather than 1.0. Another thing to be noted is about $\psi^{-1}(0)$. This value can be at most fixed to $L_{c(max)} = 7.5$. To prevent any loss of performance due to large overestimation we fix this value close to $\psi^{-1}(0.5)$ and hence choose 5.5. It should be noted that for a check node with degree 30 the corresponding L_c value when all the L_b values are around 6 would be only 2.5 but the finite precision implementation would still over estimate it to 5.5.

Fig. 4 and Fig. 5 show the BER and Frame error rate(FER) performance of a ($\lambda = 3$,$\rho = 30$), rate 0.9, regular LDPC code of length 7680. It can be seen that limiting the LLR values to 3.75 $(LLR - (2 : 2, 2^{-2}), \psi - (2 : 4, 2^{-4}))$ leads to a large performance loss. Increasing the range of LLR values while allowing the ψ function to have a large range $(LLR - (3 : 1, 2^{-1}), \psi - (3 : 3, 2^{-3}))$ leads to an error floor since the step size increases and the range of L_c messages decrease. The error floor is lowered significantly by allowing a smaller range for ψ $(LLR - (3 : 1, 2^{-1}), \psi - (0 : 6, 2^{-6}))$. 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 $0.2dB$ loss compared to floating point implementation due to small word length and large underestimation of smaller L_c values but the underestimation has little effect at lower BER and the gap closes. Using an integer range with 4 bits for ψ would have resulted in a high error floor.

Fig. 6 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 6 bits representation for ψ with its range limited to $(1 - 2^{-6})$. When only 4 bits are used for representing ψ, a 0.1 dB gain is achieved by decreasing the range of ψ from 3.75

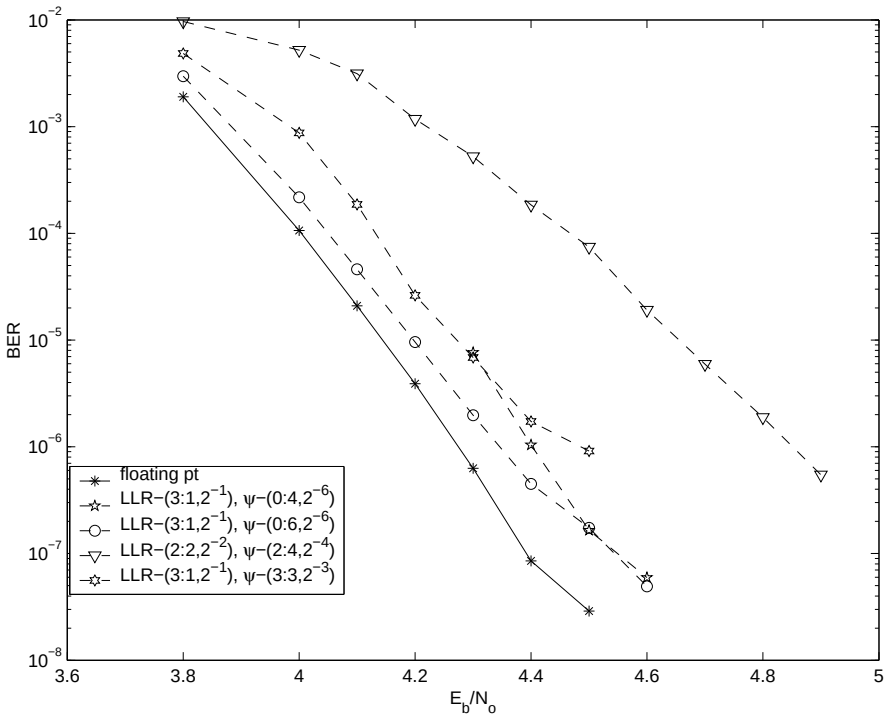

Figure 4: Bit error plot of a $(3, 30)$ rate 0.9 code of length 7680

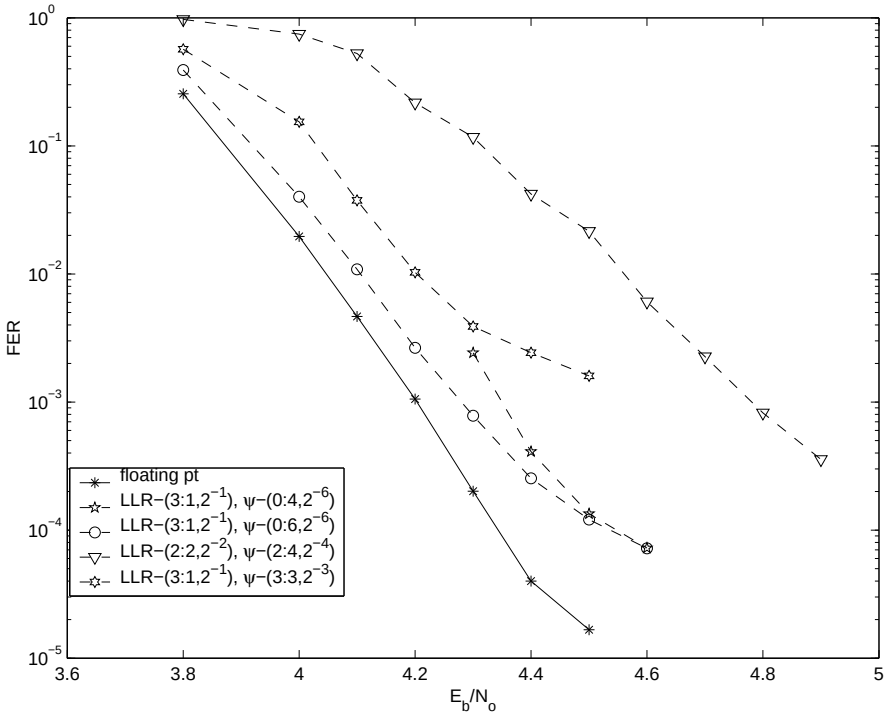

Figure 5: Frame error plot of a $(3, 30)$ rate 0.9 code of length 7680

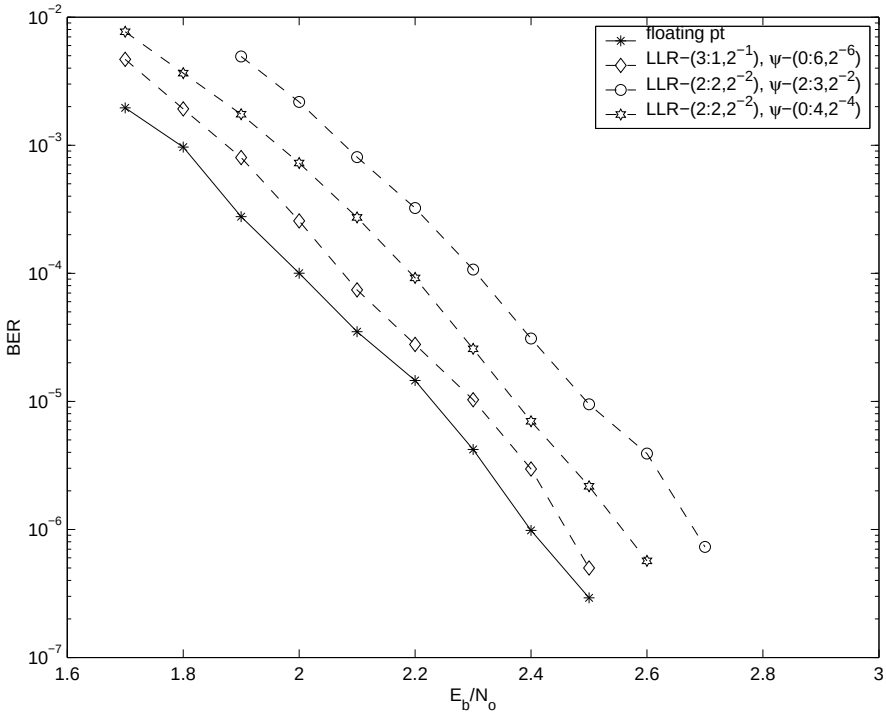

Figure 6: Bit error plot of a $(3, 6)$ rate 0.5 code of length 2040

(which represents using the same look up table for ψ and ψ^{-1}) to $(1 - 2^{-4})$. In general the range for ψ would depend upon the rate of the code, word length used and the SNR range over which the decoder operates.

3.2 Non-Uniform Quantization

A Large range for L_c messages would require extremely small quantization steps close to zero for ψ values and a uniform quantization would require large LUTs. From Fig. 1 we can clearly see that 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 save on the size of the LUTs 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.

3.2.1 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

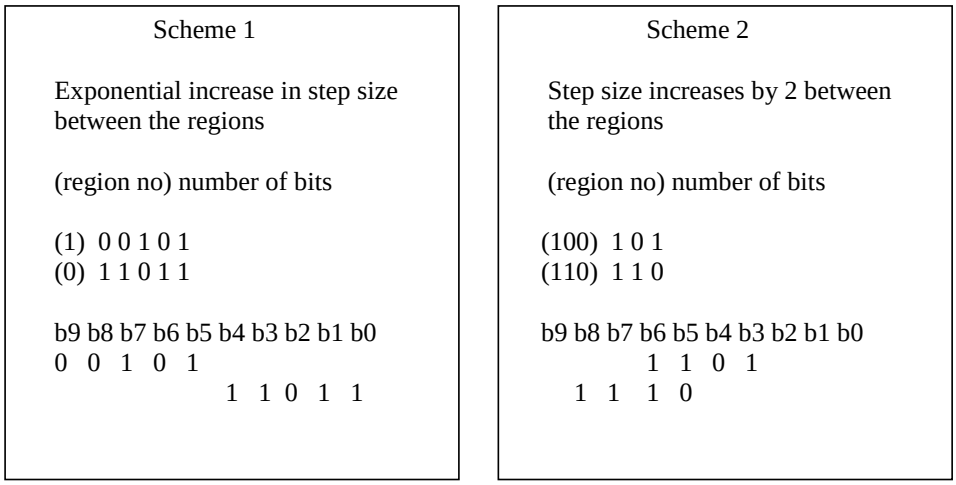

Figure 7: Example:proposed non-uniform schemes

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. 7. 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. Windowing is just a multiplexing between the 2 regions. 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. 8 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. Fig. 9 shows the BER for the proposed scheme. It can be seen that the error floor is lower compared to the uniform quantization scheme.

3.2.2 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

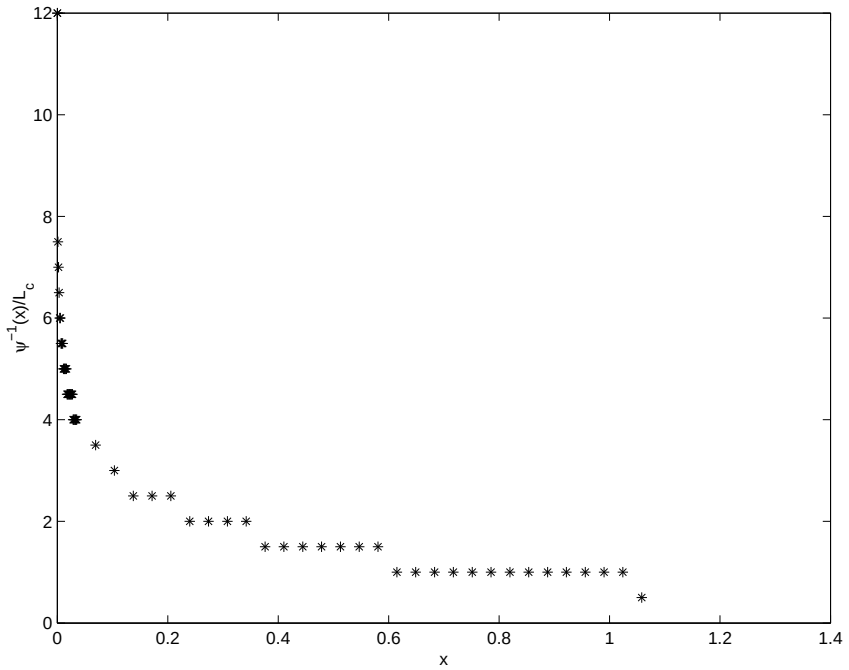

Figure 8: Finite precision ψ^{-1} function with non-uniform quantization

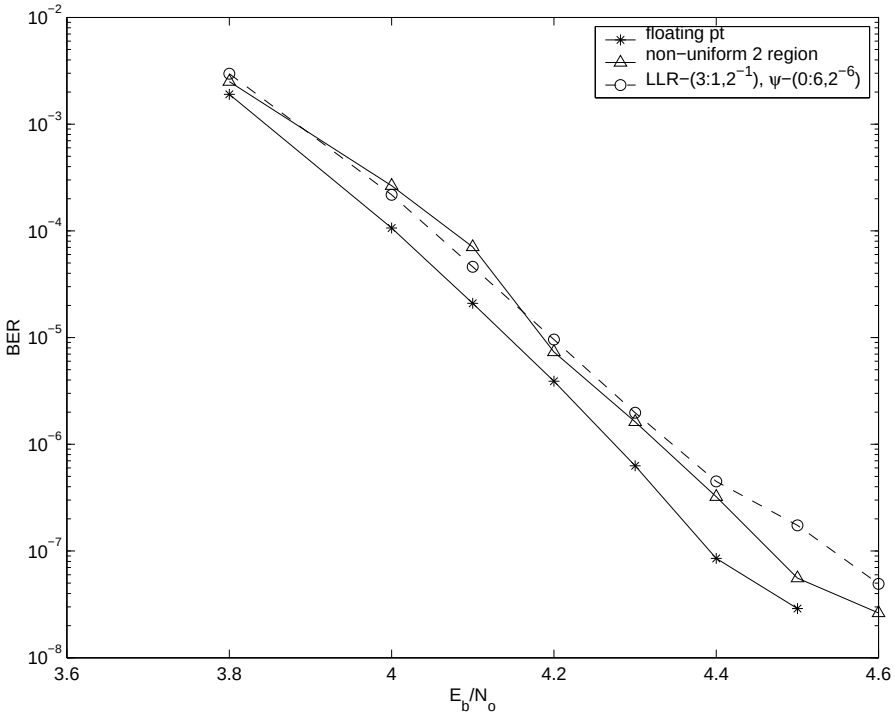

Figure 9: Bit error plot of a $(3, 30)$ rate 0.9 code of length 7680

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. 7. 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.

A generalized non-uniform quantization would combine the ψ and adder circuits(subtractor and ψ^{-1}) together into a single combinational logic block. The error correction performance depends on the implementation of the function whose design is an optimization problem.

4 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. The sign of a message depends on the data bit and always needs to be stored at the decoder. In method 1 we store only two different quantities for a check node and in method 2 the approximation reduces the number of bits needed to be stored for a message and we combine this with method 1 to store only few bits for the quantity stored in method 1. 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 should cause only simple modifications in the architecture used for a regular SPA decoder thereby not affecting the overall design and requirements.

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

$$\begin{aligned} M2(j) &= M1 - \psi(L_b^{q-1}(j)) \\ M2(i) &= M1 \quad \text{for all other edges}(i \neq j) \end{aligned} \tag{14}$$

This intrinsic feedback for the magnitude of L_c messages was previously presented by us in [34] and similar to the Approximate-Min check update [3] 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 6.

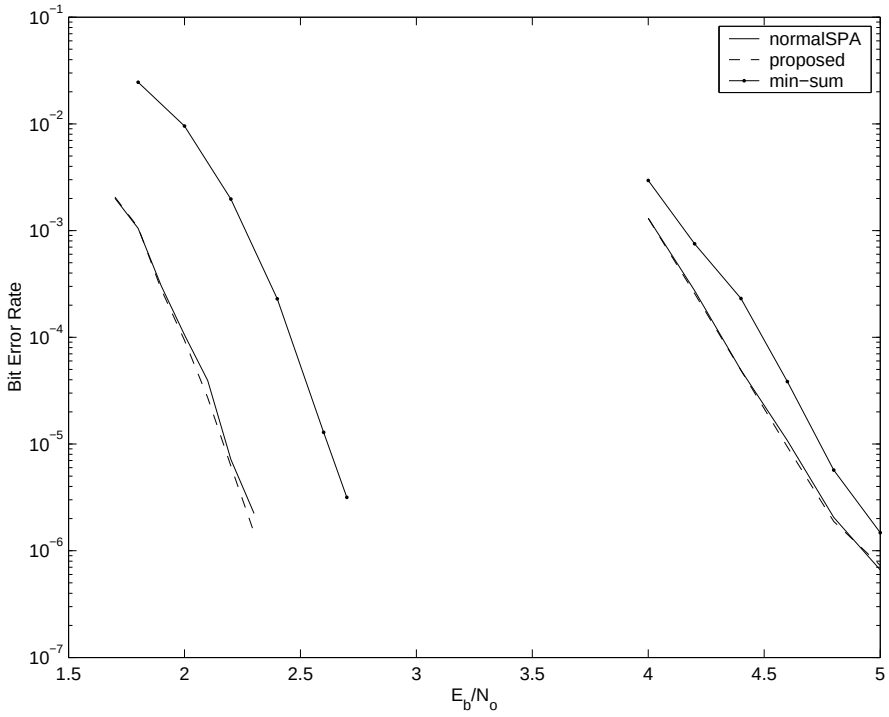

Figure 10: performance comparison plot

Fig. 10 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 the under estimation of check to bit messages in the proposed scheme compared to the overestimation of check to bit messages in the Min-Sum decoding.

From the properties of $\psi(x)/\psi^{-1}(x)$ function we know that it is a decreasing function. Also the function exhibits sharp slope for $0 < |x| < 1$ and close to zero for $3 < |x| < \inf$. 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 wont 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 [5], [6], [24]. 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 [7]. Hence when the same hardware is used for decoding codes of various rates and different degree profiles the finite precision requirements for the various correction factors is not well understood.

4.2 Method 2

In this method, instead of a self-tuning approach we follow a fixed-tuning approach where a reliability threshold is selected. Feed-back is allowed on those edges with reliability above a threshold(Th) and for the other edges regular SPA check update is performed.

$$\begin{aligned} M2(j) &= M1 - \psi(L_b^{q-1}(j)) \quad \text{if } |L_b^{q-1}(j)| < Th \\ M2(i) &= M1 \quad \text{otherwise} \end{aligned} \tag{15}$$

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(eq. 7) to remove the the intrinsic feedback. Fig. 11 illustrates this procedure.

Here $Th = 3.5$ and we do not subtract the intrinsic magnitude from the ψ sum when $L_b \geq 3.5$. 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(eq. 7) will have only 3 entries. Intuitively lets understand why this procedure should work. L_b messages greater than Th are more reliable and hence contribute very little to the ψ

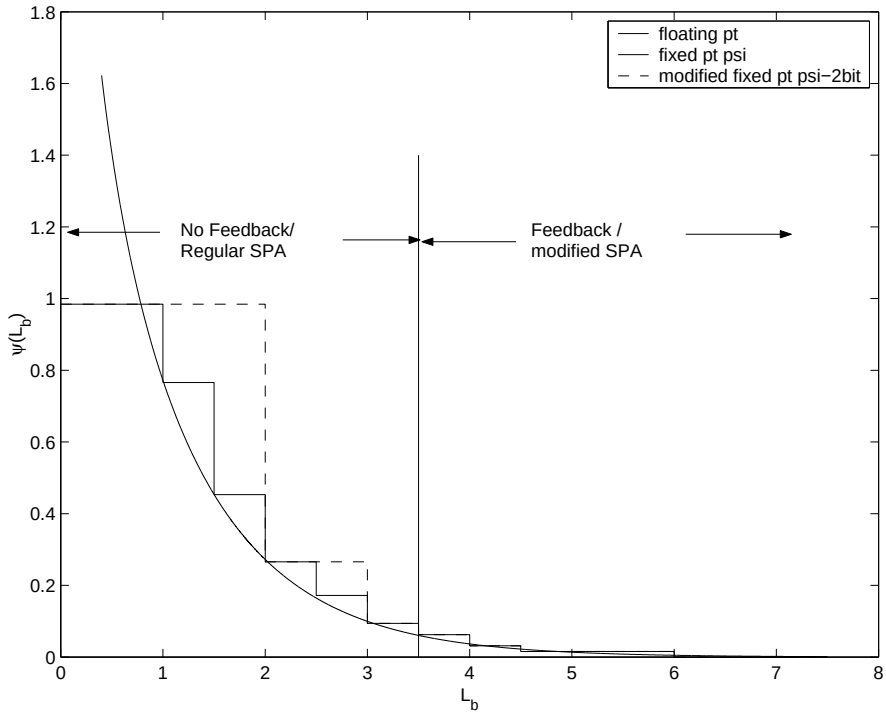

Figure 11: Modified PSI LUT-Method 2

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 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. The change in step size would affect the performance when most of the L_b messages have a good reliability but only a few of them are below Th. In this case the ψ sum would lie in a region where variations to the sum would cause noticeable variations in L_c message magnitudes. 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,

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

Fig. 12 and Fig. 13 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

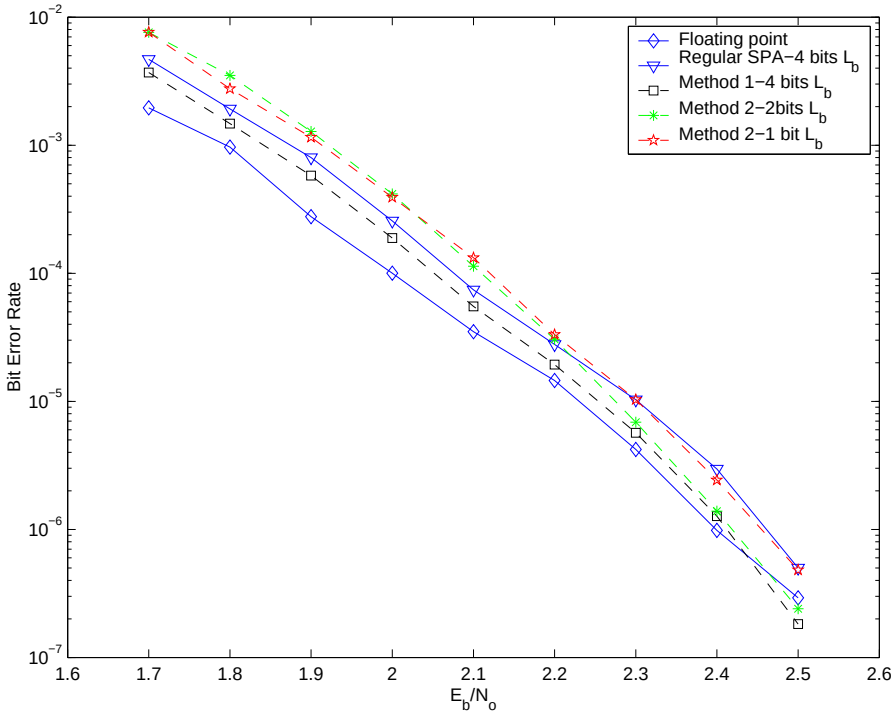

Figure 12: Rate 0.5 (3,6) LDPC code of length 2040

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 Signal to Noise ratios(SNR) and almost gives same performance at higher SNR. Fig. 14 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 its slightly slower.

5 Parallelization and Scalability

The parity check matrix of an LDPC code is defined 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 made at random with constraints to avoid cycles of short length [32]. Random construction is not suitable for hardware implementation due to the following reasons. (i)The bit and check node connection for each edge has to be stored at the decoder. (ii) Random connections lead to switching

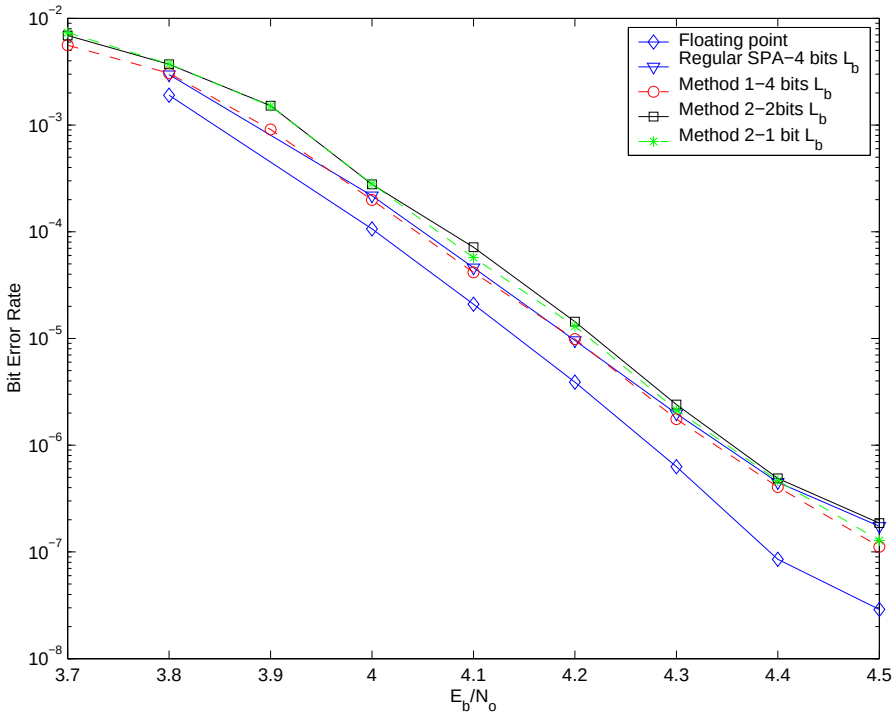

Figure 13: Rate 0.9 (3,30) LDPC code of length 7680

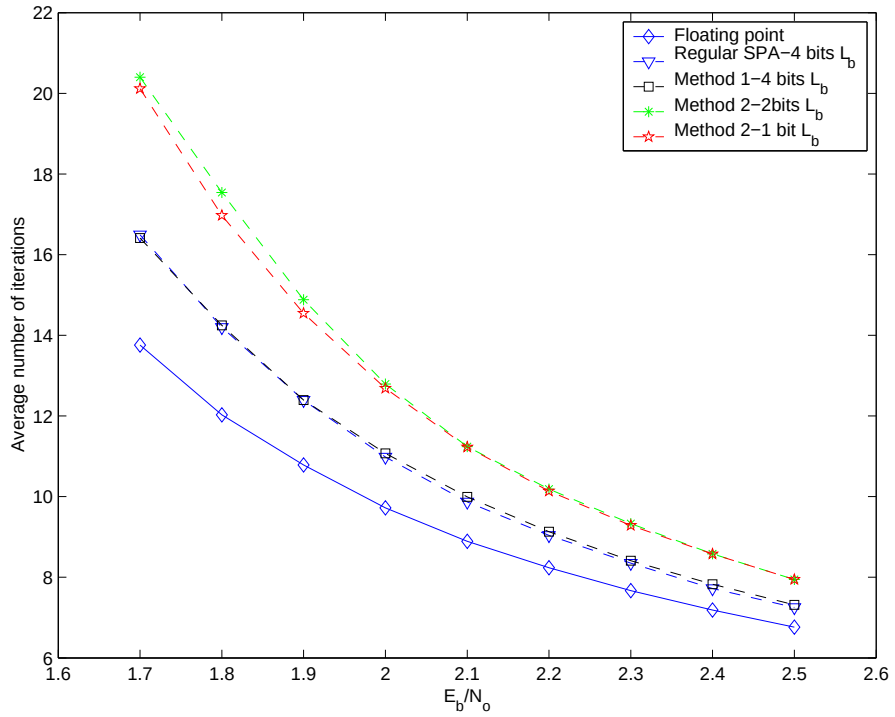

Figure 14: Comparison of average number of iterations for various methods:max 30 itr

complexity for routing messages between bit and check nodes. (iii) Random connections will lead to memory access conflicts for partly parallel architectures and shall be explained in more detail.

Consider M bit node updates taking place in parallel. Each bit node update unit reads one L_c message from the memory during a clock cycle and therefore M L_c messages have to be read simultaneously and hence need to be stored in different memory blocks to avoid access conflict. On completion of the bit updates the new M L_b messages are written back simultaneously into the same locations. For check node update the constraint would be that M L_b messages have to be read simultaneously and the updated M L_c messages have to be written back simultaneously into the same locations. Hence the same memory should allow simultaneous M way read/write for both bit and check nodes and a random construction will not satisfy this constraint and would lead to memory access conflicts. Therefore partly parallel decoding of LDPC codes require special structural constraints on the parity check matrix. The exact constraints depend on the implementation of the decoder and the scheduling used. A random bit filling procedure for constructing the parity check matrix taking into consideration the parallelization constraints was proposed in our previous work [20]. This procedure has implementation difficulties due to reasons (i) and (ii). Recently, LDPC codes constructed from Circular Permutation Matrices(CPM) have been proposed [25], [27], [14]. Our architecture uses these family of codes and our heuristic construction procedure works 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 profile is chosen such that the $\lambda_i's$ are of the form $\frac{x}{n}$ for some x and similarly $\rho_i's$ of the form $\frac{y}{k}$ for some y). 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 each 1 is chosen one at a time and should minimize cycles of short length(4,6,8,10) for the part of the matrix already expanded from the base matrix. The necessary and sufficient conditions given in [25]. Fig. 15 shows the procedure for illustration purpose with smaller dimensions.

Fig. 15 shows a circularly shifted identity matrix of size $P = 12$. 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 M bit node updates and M check node updates 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. It can be clearly seen that M divides P and let $\frac{P}{M} = D$. Rearrange the columns of the square matrix such that first we have all the columns which have modulus D as 0. 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. We observe two levels of circular shifting. Using the base matrix representation for this block we find that the smaller non zero block matrices form a circularly shifted identity matrix in the base matrix. Let the shift on the larger block be S. The circular shift for the base matrix is given by $A = S \bmod D$. The circular shift for the smaller non-zero blocks take two values, $s = \lfloor \frac{S}{D} \rfloor = \lfloor \frac{S \times M}{P} \rfloor$ for the blocks before wrap around and $s + 1$ for the blocks after wrap around. Hence for each larger block we need to store only two values namely A and s to operate the decoder at a lower parallelization level. Fig. 15 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 decoding performance for various parallelization factors remain the same since there is only a reassignment of columns and rows between them and the flexibility of choosing various parallelization levels offer scalability to the architecture. The above procedure eliminates the need for an alignment and reverse alignment unit as used in [22] which would require additional storage of size $M \times \lambda_{max} \times check - state - size$ or $M \times \rho_{max} \times bit - state - size$ or twice the amount of read and write operations with extra multiplexing when used in our architectures.

The cyclic shift between the bit nodes and check nodes can be achieved with a Logarithmic shifter or with an 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. A 4×4 Logarithmic shifter is shown in Fig. 16. It has two stages of multiplexers. The first stage can shift values by $2^0 = 1$ and the second stage by $2^1 = 2$. The select signal for multiplexers in each column is defined by the binary representation of the shift needed. 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 $(\lfloor \frac{S \times M}{P \times t} \rfloor + 1)$. Array codes [14] are highly structured and the shift between adjacent blocks in a row or column is a constant. This property of constant shift difference has been used in [29] to eliminate the need for cyclic shifters. Similar structured parity check matrices can be formed in our above mentioned procedure if we can choose the shift on adjacent permutation blocks in a row or column to differ by a minimum of D and a maximum

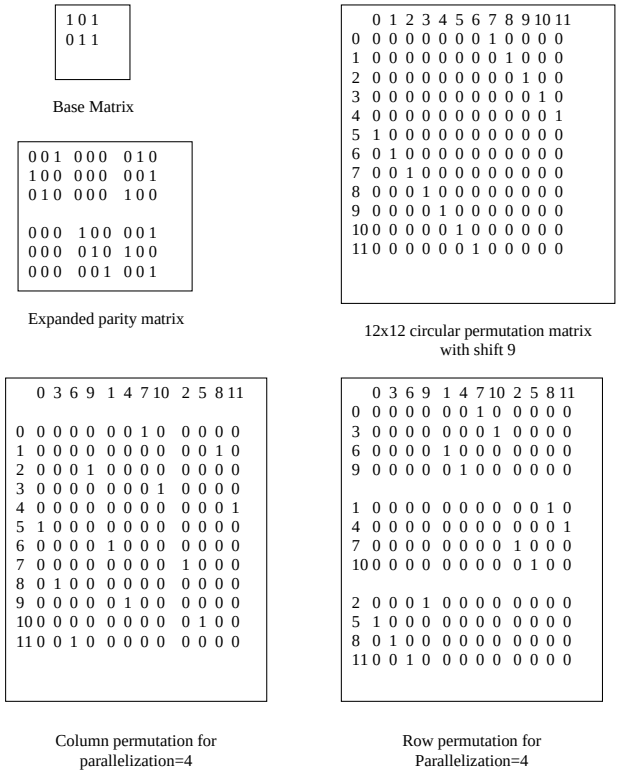

Figure 15: Base Matrix Expansion and parallelization

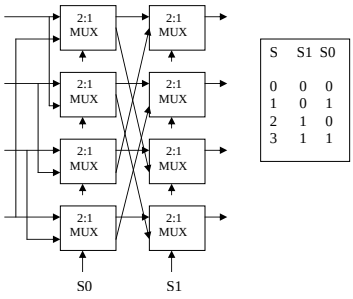

Figure 16: 4×4 Cyclic Shifter

of $2D - 1$.

6 Decoder Architectures

Based on the direction of scheduling used, we broadly classify the partly parallel decoder architectures into two types namely column schedule and row schedule architectures. In column schedule architectures the 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 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 based on the scheduling used for update procedure. In normal parallel/flooding scheduling all the bit-node updates are completed before making any check-node updates and vice versa. In layered/turbo scheduling [30], [23], [26], [9] only a set of bit-node updates are done and the newly formed L_b messages are used for a set of check-node updates. Now the newly formed L_c messages are used in another set of bit-node updates and this procedure continues. It has been shown that the layered architecture improves the convergence speed of the decoder and hence offers lesser latency. The power consumption also decreases since the average number of iterations for a packet come down.

A Communication standard might use LDPC codes of various rates and lengths depending upon the condition of the channel. The LDPC decoder at the receiver must then handle codes of various rates and lengths. Hence, we implement both bit and check update units as serial-type units to make them independent of λ and ρ. In our architectures we use both column and row scheduling that employ flooding or turbo scheduling and analyze the memory requirements.

Fig. 17 shows the top level block diagram for the proposed partly parallel decoder architecture with parallelization M. It consists of M check-node update(CNU) units and M bit-node update(BNU) units with cyclic shifters sitting between them. The cyclic shift to be used is as described in the previous section. The same set of units are used for all decoding iterations and hence we have an iteration multiplexer(MUX) that feeds in Lch 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.

6.1 Column Schedule Architectures

6.1.1 Regular SPA-Flooding Schedule

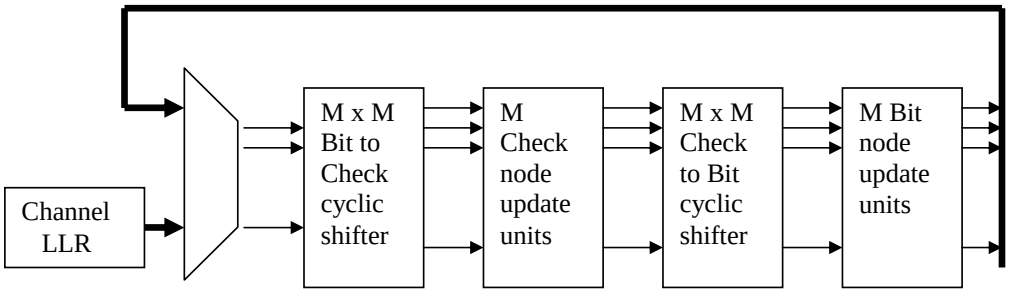

Figure 17: Top level block diagram

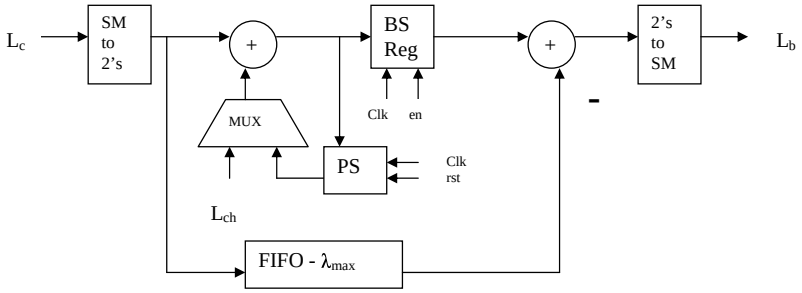

Figure 18: Implementation of Bit Node Update unit for column schedule architecture

A. Bit-Node Update Unit: Fig. 18 shows the the BNU unit for a column schedule architecture. The L_c message from the CNU unit is converted from Sign-Magnitude(SM) representation to 2's complement representation and added together with L_{ch} to form L_{soft} as given by equation 9. Due to column scheduling the λ successive L_c inputs would belong to the same bit-node. The partial sum is stored in PS register and once the the λ values are added the L_{soft} is transferred to BS register. The L_c values that were stored in a First-In-First-Out(FIFO) stack when the sum was formed are used for subtraction to form L_b messages as given by equation 11. The use of PS and BS registers help compute equations 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.

B. Check-Node Update Unit: Fig. 18 shows the the CNU unit for a column schedule architecture. Since the successive L_b inputs belong to different check nodes, quantities $M1$ and $S1$ as defined by equation 6 for each check node have to be stored in a memory. We use the concept of mirror memory to pipeline the check update operations of the present iteration with the check update operations of the previous iteration through the bit update units. While the mirror memory bank stores $(M1, S1)$ values of previous iteration the other memory bank stores the partially formed $(M1, S1)$ values for the present iteration. The size of these memories is proportional to the number of parity checks and precision of ψ values. The iteration process takes place as following.

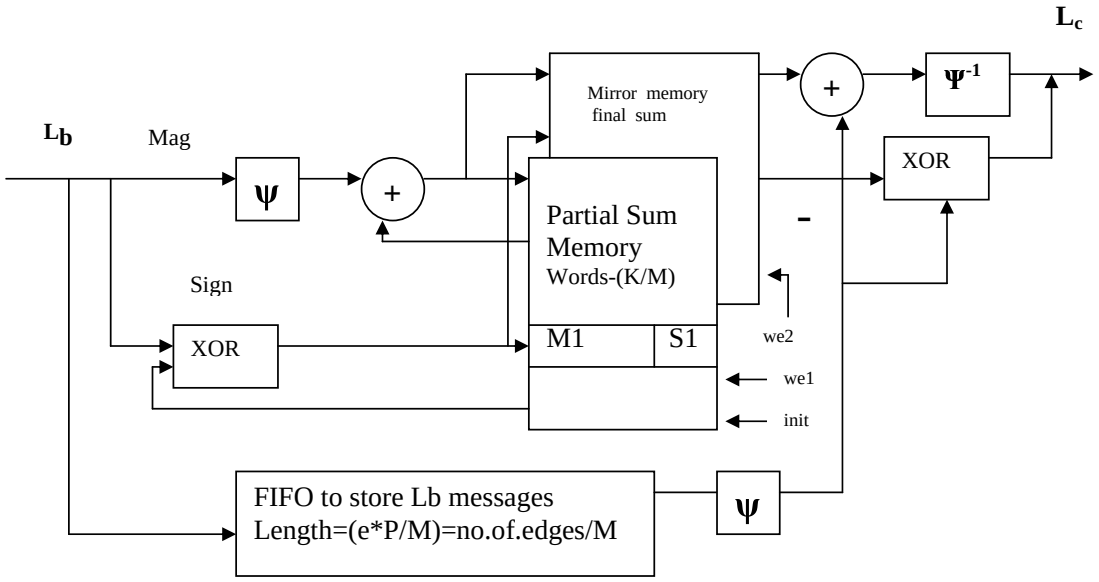

Figure 19: Implementation of Check Node Update unit for column schedule architecture using flooding update and regular SPA

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 equations 7 and 8. These messages are passed to the bit-node update units and the updated L_b messages are stored in the FIFO and also 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 number of bits used for representing $M1$ is one greater than the number of bits used to represent ψ because only one subtraction is needed to get $M2$. Saturation adders and subtractors are used for this purpose. We store only L_b messages in this architecture and the size of the FIFO is proportional to the number of 1's in the parity check matrix and precision of L_b messages. This forms the bulk of the memory in the decoder hardware.

6.1.2 Modified SPA-Flooding Schedule

Since there is no change in the bit node update process the structure of BNU unit remains the same as described for regular SPA.

Check-Node Update Unit: Fig. 18 shows the 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 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 location. The number of bits required for storing the present minimum location is given by $\lfloor \log_2(\rho) \rfloor$ bits. We transfer

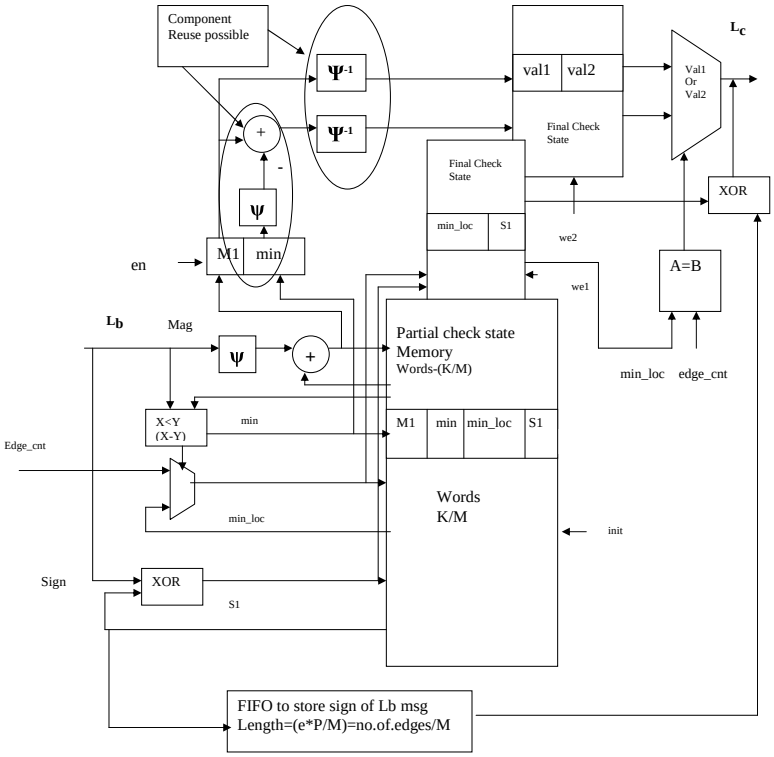

Figure 20: Implementation of modified SPA Check Node Update unit for column schedule architecture using flooding update

the final $(M1, S1)$ values to a register before feeding them to the circuit required to implement equation 16. Using this register we avoid unnecessary switching of $M2$ circuit on partial check values. This adds one clock cycle latency per iteration. 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} circuits 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 $(1/\rho)\%$ and therefore the loss in throughput is insignificant only for higher rate codes when ρ is large. It should be noted that due to TDM there is additional switching of signals every clock cycle in the multiplexers used with $M1$ circuit. The check node unit can be built with only one ψ^{-1} circuit and without the registers [34] at the cost of additional switching and memory. 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 FIFO widths have become one bit and store only the sign of the L_b messages and hence the memory reduction. The memory reduction was possible since we store only one type of messages and used value reuse for L_c messages. The memory reduction increases with the rate of the code since we have lesser number of check nodes.

6.1.3 Min-Sum decoding with offset correction-Flooding Schedule

For MS decoding we again have only two different values for L_c message magnitudes as defined by equation 13. Hence we can use the same techniques used in the implementation of modified SPA. The BNU unit remains the same as in regular SPA except for the addition of an offset correction unit similar to the one shown in Fig. 21. When the column degrees are regular this correction unit can be eliminated by lumping together with the correction term used in check update unit.

Check-Node Update Unit: The concepts and working of this unit shown in Fig. 21 is similar to that used in modified SPA but the values stored for each check and the updating circuits differ. For every check we store min1, min2, location of min1 and the XOR of sign bits S1. The min1, min2 values are updated using 2 comparators and 3 2:1 MUXs as shown in Fig. 22. The update can be achieved in two different ways. In parallel min update the incoming L_b is compared with both $min1$ and $min2$ simultaneously. In serial min update first we compare L_b with $min2$ and then we compare the result with $min1$. When the successive L_b messages to a check-node update unit do not belong to the same check-node as in the case of column scheduling both the methods result in 2ρ comparisons and hence we choose parallel update method which is faster. In the case of row scheduling the successive L_b messages belong to the same check-node update unit and we would choose serial method since the inputs to the second comparator do not change unless $min2$ changes. Hence we have between $\rho + 1$ to 2ρ comparisons which is lesser than the parallel method. Once the final min1,min2 have been found they are transferred to a register. Offset correction is made for these two values and the greater of zero or offset corrected value is written into the mirror memory. For regular LDPC codes this is the only offset correction unit present and 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. TDM can be used for these offset correction unit in the check-node update unit. When implementing on Xilinx series of FPGA's a multiplexer and subtractor require same number of look-up-tables(LUTs) and hence not having TDM would be better to avoid unnecessary switching of signals in the MUXs.

6.1.4 Regular SPA-Layered scheduling

For layered scheduling at the check node only $M1$ and $S1$ change from flooding scheduling(equation 6). At any time instant the $M1$ and $S1$ values for a check node depend on L_b messages from the present

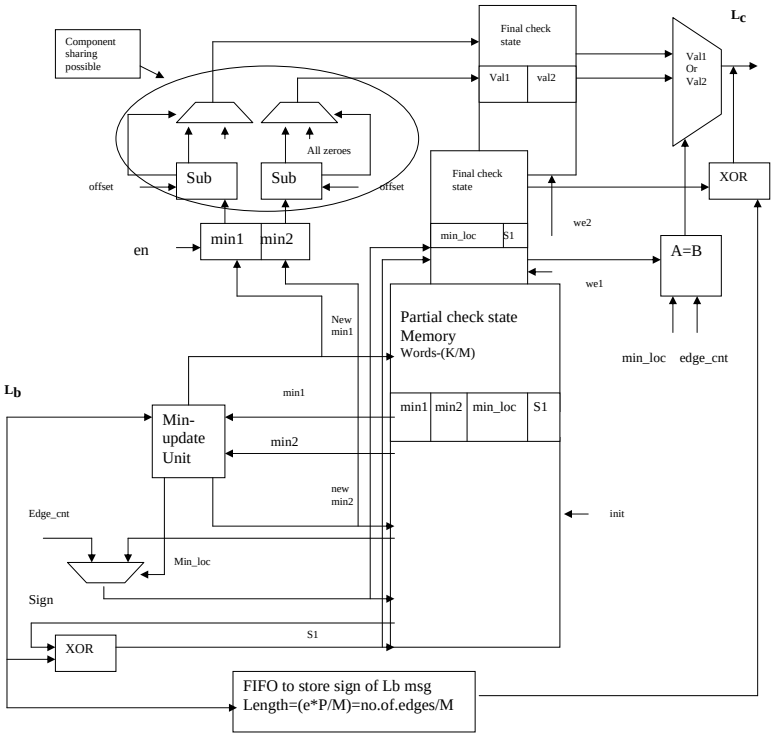

Figure 21: Implementation of min-sum algorithm with offset correction for column schedule architecture using flooding update

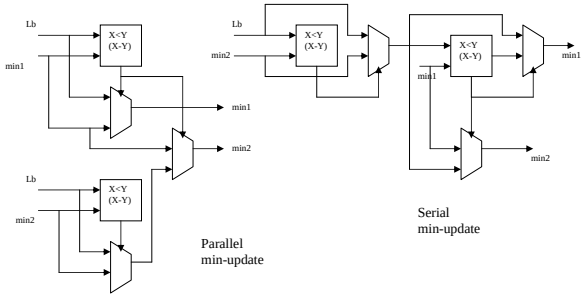

Figure 22: Min1,Min2 Update Unit

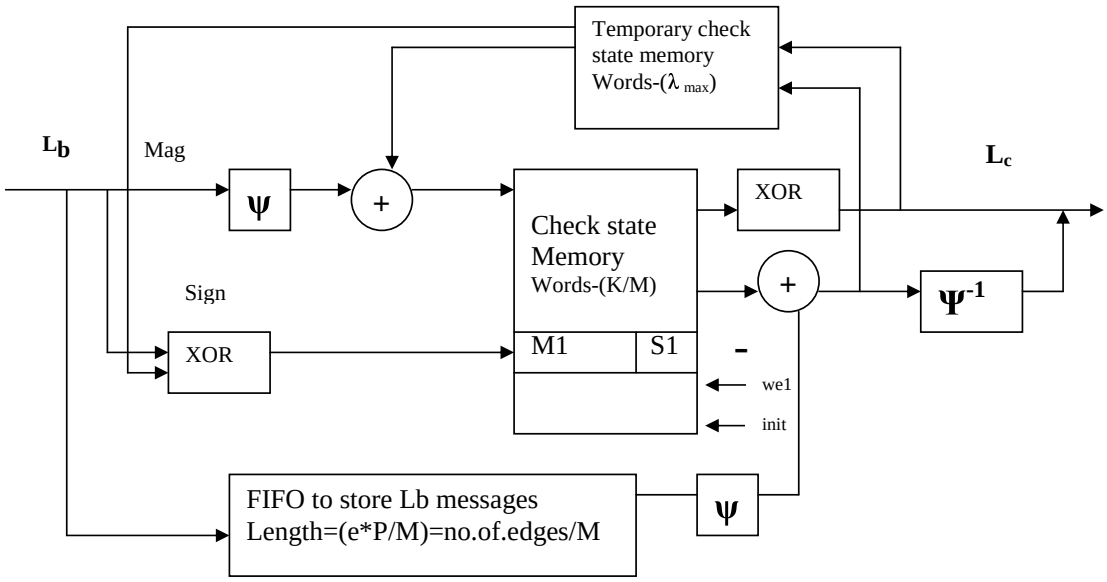

Figure 23: Implementation of regular SPA Check Node Update unit with Column scheduling and layered updating procedure

iteration as well as the previous iteration. This is given by,

$$M1_{lay} = \sum_{k=1}^{k=\rho-t} \psi(L_b^q(k)) + \sum_{k=t}^{k=\rho} \psi(L_b^{q-1}(k)) \qquad S1_{lay} = \prod_{k=1}^{k=\rho-t} sign(L_b^q(k)) \prod_{k=t}^{k=\rho} sign(L_b^{q-1}(k)) \quad (17)$$

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

$$M1_{lay} = \sum_{k=1}^{k=\rho-t+1} \psi(L_b^q(k)) + \sum_{k=t+1}^{k=\rho} \psi(L_b^{q-1}(k)) \qquad S1_{lay} = \prod_{k=1}^{k=\rho-t+1} sign(L_b^q(k)) \prod_{k=t+1}^{k=\rho} sign(L_b^{q-1}(k)) \quad (18)$$

Fig. 23 shows the implementation of a CNU unit. Since there is no concept of partial and final $(M1, S1)$, we have only one memory to store the present state of $(M1, S1)$. Instead of the mirror memory we have a small memory of length λ to store $(M2, S2)$ values due to the latency of the bit node update unit. Since we add and subtract values from $M1$ no saturation of values is possible. Hence the number of bits to represent $M1$ would be $log_2(\rho) + \psi$ 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 and the gain is faster convergence of the decoder leading to lesser latency and average number of iterations.

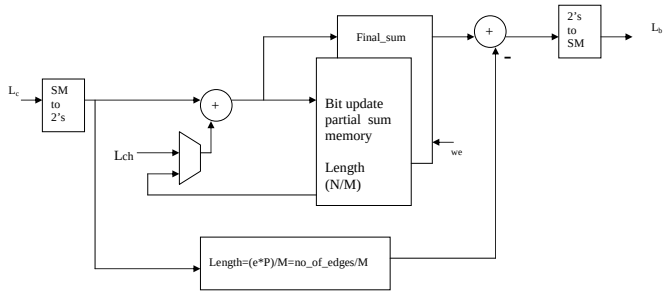

Figure 24: Implementation of Bit Node Update unit with row scheduling and flooding update

6.1.5 Modified SPA and Min-Sum decoding-Layered scheduling

Since we store only the minimum 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.

6.2 Row Schedule Architectures

6.2.1 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 sum(eq. 9). Fig. 24 shows the implementation of the BNU unit. There are two memories storing the bit node update sums. 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 equation 9 is formed they are transferred to the mirror memory. The size of these memories is proportional to the length of the code 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 $log_2(\lambda) + L_c$ bits. For $M1$ we used the concept of saturation and hence the size of $M1$ was independent of the check node degree. 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. Also, the size of the sign magnitude adder and subtractor is determined by the columns with high degree. The FIFO containing L_c messages is now located in the bit node update unit rather than check node update unit for column scheduling.

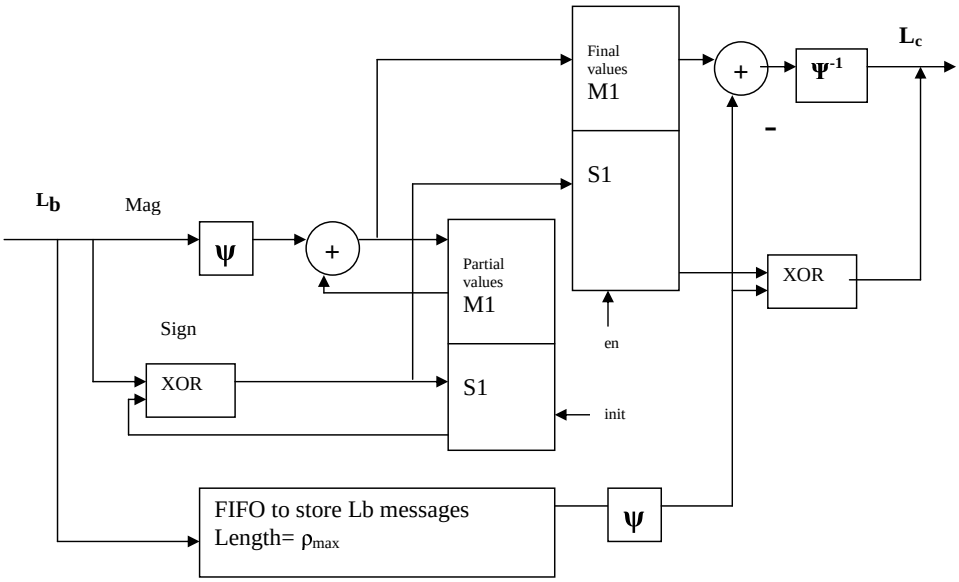

Figure 25: Implementation of Check update unit with row scheduling

A. 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 and hence there is no need for memory to store $(M1, S1)$. Fig. 25 shows the implementation for a CNU unit. The structure is same as the CNU unit used for column scheduling(Fig. 19) 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 since 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 structure for the check-node update unit with row scheduling for modified-SPA and MS decoding will resemble their column scheduling counterparts with the changes as mentioned for regular SPA unit. For these two cases the FIFO in the BNU unit will be moved to the CNU unit. The FIFOs will store only the signs of L_c/L_b messages and not the magnitude. We will be using the value reuse property of L_c messages twice and also 2's complement is done only on two different L_c messages. A more detailed explanation is given next with a layered decoding model which differs from flooding model only in the number of bit-node sum memories used.

6.2.2 modified SPA-Layered Scheduling

The layered decoding model at the bit node can be implemented for regular SPA, modified SPA and MS decoding. We look at the modified SPA implementation in detail and the structure for

MS decoding remains similar except for a different check update procedure. Fig. 26 shows such an implementation. We have a FIFO that stores the check state values of the previous iteration namely $value1 = \psi^{-1}(M1)$, 2's complement or SM representation of $value2 = \psi^{-1}(M1 - \psi(L_b^{q-1}))$ depending on the sign of $L_b^{q-1}(min)$ message, min location and the XOR of signs $S1$. Another FIFO stores the sign of L_c^{q-1} messages. A bit-node sum memory stores the L_{soft} values. Using the value reuse property and the sign FIFO we can generate the L_c^{q-1} messages for a check-node. These messages pass through a check-node to bit-node cyclic shifter and using equation 11 we can find L_b^q messages. These messages pass through a bit-node to check-node cyclic shifter and arrive at the check-node-update unit. These messages are also stored in a temporary memory equal to the size of check-node unit latency. Since we have row scheduling the successive L_b^q messages belong to the same check node and hence we have only registers to store the temporary check state values. Compared to the flooding row schedule the only difference in this unit is that we also store the sign of $L_b^q(min)$ message. Once the final check state values are transferred to the mirror memory we find the two different $M2$ values. With the sign of $L_b^q(min)$ and $S1$, either SM or 2's complement representation for $value2$ is selected. The updated check FIFO states are written back to it. We again use the value reuse property to generate the L_c^q messages for the present iteration. The sign of these messages is written into the sign FIFO. The L_c^q messages pass through a check-node to bit-node router and gets added with the the output of the temporary memory to form the L_{soft} values and are written back to the bit-node sum memory. During the last write to each bit-node in the final iteration we store the L_{ch} message for the new packet.

Fig. 27 shows the performance of the modified SPA-layered decoding with that of regular SPA decoding with flooding update as a function of iterations for a regular LDPC code of rate 0.5 and length 2040. It can be seen that the BER performance for regular SPA decoding at 30 iterations is comparable to the BER performance for modified SPA-layered decoding. Hence we have a 2x reduction in latency. Fig. 28 shows the average number of iterations required for both the methods. The regular SPA with flooding update requires approximately 1.8x the number of iterations for modified SPA with layered decoding and hence we can shut-off the decoder in fewer number of iterations leading to power savings. Also we need only one bit-node sum memory compared to two blocks for flooding update. Hence we have memory savings too.

We summarize the memory requirements and check node operations for our various implementations and decoding algorithms in tables 1 and 2. We present results for regular LDPC codes for easier comparison and understanding. The requirements for irregular LDPC codes are straightforward extensions.

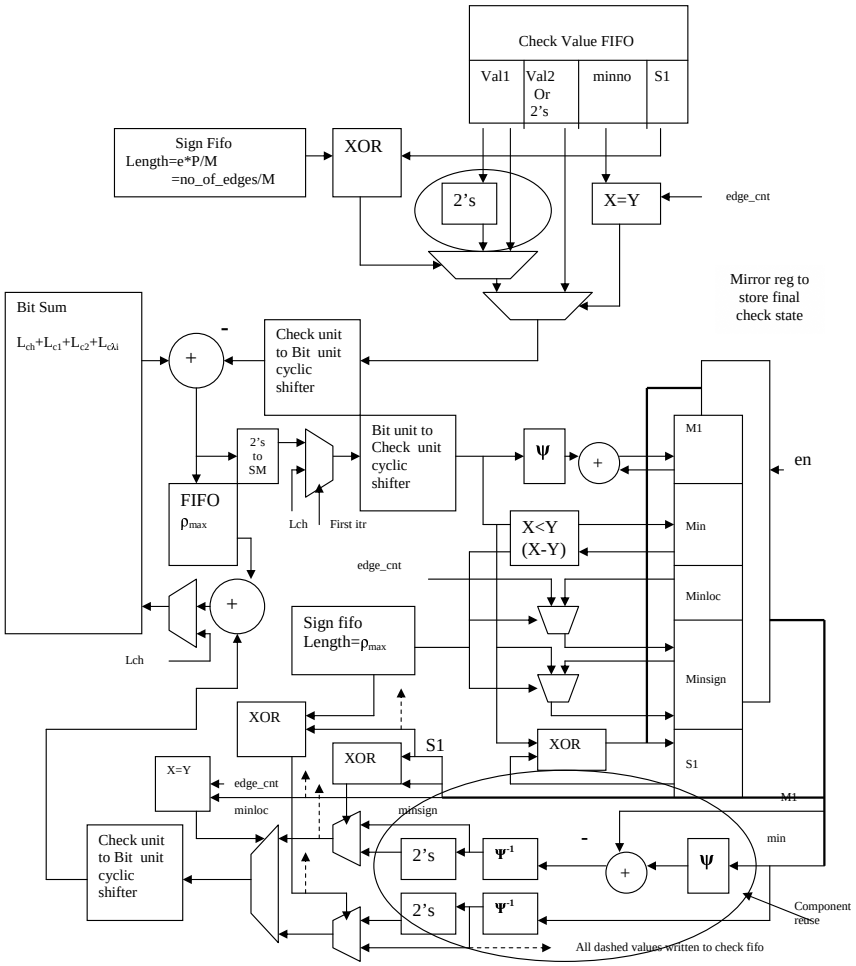

Figure 26: Implementation of Modified SPA with row scheduling and layered update procedure

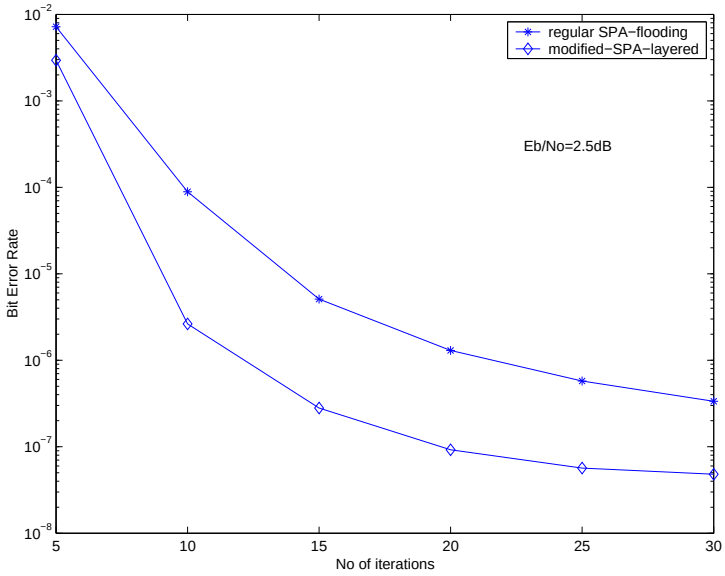

Figure 27: Bit error plot as a function of iterations for a(2040,1020) regular LDPC code constructed from cyclic permutation matrix

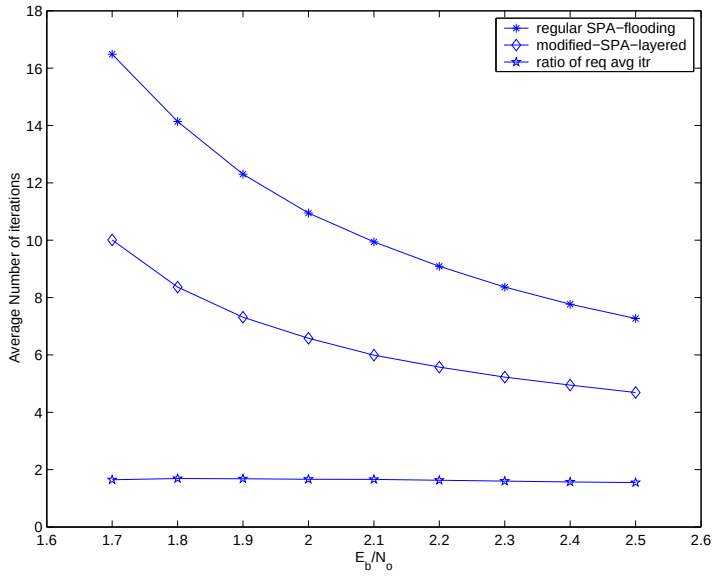

Figure 28: Average number of iterations as a function of E_b/N_o for a(2040,1020) regular LDPC code constructed from cyclic permutation matrix

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

Table 1: Memory requirement for various algorithms and schedules

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
SPA	$\psi = \rho + 1$ $\psi^{-1} = 2$ $ADD = \rho$ $SUB = 1$ $lessthan - comp = \rho$ $equal - comp = \rho$ SM to 2's= ρ 2's to SM= ρ	$\psi = \rho + 1$ $\psi^{-1} = 2$ $ADD = \rho$ $SUB = 1$ $lessthan - comp = \rho$ $equal - comp = 2 \times \rho$ SM to 2's= 3 2's to SM= ρ
Min-Sum-offset	$SUB = 2$ $lessthan - comp = 2 \times \rho$ $equal - comp = \rho$ SM to 2's= ρ 2's to SM= ρ	$SUB = 2$ $lessthan - comp = \rho + 1$ to 2ρ $equal - comp = 2 \times \rho$ SM to 2's= 3 2's to SM= ρ

Table 2: Number of operations for each check update

6.3 Parallel Update units

We have used serial bit-node update and check-node update units to provide flexibility to the architecture so that the same 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 update units. A column schedule architecture in such case will have a parallel bit-node update unit and serial check-node update unit. A parallel bit-node update takes all λ L_c messages simultaneously and finds λ L_b messages. In such a case we can have λ memory blocks instead of one as in the case of serial update units. This provides an inherent parallelization of λ. To achieve a parallelization of M we now need a parallelization of M/λ in each block and hence the number of stages in the cyclic shifter reduces by $\lfloor \log 2(\lambda) \rfloor$. A row schedule architecture will have a parallel check-node update unit(binary tree) and a serial bit-node update unit. We can divide the memory into ρ blocks and hence this reduces the number of stages in the cyclic shifter by $\lfloor \log 2(\rho) \rfloor$. In the parallelization procedure described for serial units, M different update units operate on M different nodes. Hence even though we have reduced the number of ψ, ψ^{-1}, subtraction, SM to 2's operations we still have no reduction in the number of units at the decoder unless TDM is done. For a parallel check-node update unit the reduction in the number of operations directly leads to reduction in the number of hardware instantiations required. Also the equal to comparisons can be replaced by a single $\log_2(\rho) : \rho$ encoder. For a MS decoder with parallel binary tree CNU it has been shown that the number of comparisons can be reduced to as low as $\rho + \lfloor \log_2(\rho) \rfloor - 2$. The cost of row scheduling is the large size for bit-node sum memories compared to column scheduling.

7 Conclusion

In this paper we have presented a modified SPA that has lower complexity and results in only two different L_c magnitudes and therefore can be implemented with a lesser complexity CNU unit and memory requirements. The clipping and precision issues related to the implementation of SPA decoding algorithm were studied. Parity check matrix construction procedures for various decoder parallelization levels were studied. Architectures for regular SPA, modified SPA and Min-Sum decoding with various scheduling schemes were presented. Their complexity, memory requirements, BER performance were discussed.

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] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

[10] H. Sankar, K. R. Narayanan. Memory-Efficient Sum-Product Decoding of LDPC Codes. *IEEE Tansactions on Communications*, vol 52, Issue 8, Aug 2004, pages: 1225- 1230.

[11] 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.

[12] 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.

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

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[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. *IEEE Global Telecommunciations Conference, 2003. GLOBECOM'03*, vol:4, 1-5 Dec.2003, 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] T. Okumora. Designing LDPC codes using cyclic shifts. *IEEE international symposium on information theory, ISIT 2003*, page: 151.

[32] 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.

[33] Y. Kou, S. Lin, M.P.C. Fossorier. Low-density parity-check codes based on finite geometries: a rediscovery and new results. *IEEE Trans. on Information theory*, vol:47, Issue 7, Nov 2001, pages: 2711- 2736.

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