HIGH THROUGHPUT LOW POWER DECODER ARCHITECTURES FOR LOW DENSITY PARITY CHECK CODES

A Dissertation

by

ANAND MANIVANNAN SELVARATHINAM

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 2005

Major Subject: Computer Engineering

HIGH THROUGHPUT LOW POWER DECODER

ARCHITECTURES FOR LOW DENSITY PARITY CHECK CODES

A Dissertation

by

ANAND MANIVANNAN SELVARATHINAM

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	Gwan Choi
Committee Members,	Krishna Narayanan
	Xi Zhang
	Rabi Mahapatra
Department Head,	Chanan Singh

August 2005

Major Subject: Computer Engineering

ABSTRACT

High Throughput Low Power Decoder Architectures for Low Density Parity Check Codes. (August 2005)

Anand Manivannan Selvarathinam, B.E., Anna University;

M.S., Texas A&M University

Chair of Advisory Committee: Dr. Gwan Choi

A high throughput scalable decoder architecture, a tiling approach to reduce the complexity of the scalable architecture, and two low power decoding schemes have been proposed in this research. The proposed scalable design is generated from a serial architecture by scaling the combinational logic; memory partitioning and constructing a novel H matrix to make parallelization possible. The scalable architecture achieves a high throughput for higher values of the parallelization factor M. The switch logic used to route the bit nodes to the appropriate checks is an important constituent of the scalable architecture and its complexity is high with higher M. The proposed tiling approach is applied to the scalable architecture to simplify the switch logic and reduce gate complexity.

The tiling approach generates patterns that are used to construct the H matrix by repeating a fixed number of those generated patterns. The advantages of the proposed approach are two-fold. First, the information stored about the H matrix is reduced by one-third. Second, the switch logic of the scalable architecture is simplified. The H matrix

information is also embedded in the switch and no external memory is needed to store the H matrix.

Scalable architecture and tiling approach are proposed at the architectural level of the LDPC decoder. We propose two low power decoding schemes that take advantage of the distribution of errors in the received packets. Both schemes use a hard iteration after a fixed number of soft iterations. The dynamic scheme performs X soft iterations, then a parity checker cH^T that computes the number of parity checks in error. Based on cH^T value, the decoder decides on performing either soft iterations or a hard iteration. The advantage of the hard iteration is so significant that the second low power scheme performs a fixed number of iterations followed by a hard iteration. To compensate the bit error rate performance, the number of soft iterations in this case is higher than that of those performed before cH^T in the first scheme.

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr Choi for his help and guidance in making this dissertation possible. He has encouraged me throughout this dissertation. I also thank him for teaching me VLSI concepts. He has taught me how to conduct research and other qualities such as problem exploration and problem solving. He helped instill confidence in me about doing research. I am thankful to him for creating a good learning environment and for his patience in my times of crisis.

I would like to thank Dr Narayanan for teaching channel coding, advanced channel coding and information theory and had it not been for him, I would not have worked on this dissertation topic. I also thank him for his ideas and suggestions in the course of this research.

I would like to thank Dr Reddy for providing valuable feedback to my dissertation as a former committee member and the work on hard iterations in the later half of this research would not have been possible but for his instigation. I also like to thank Dr Zhang for sharing his knowledge on tornado codes and for providing valuable feedback for my dissertation. I would also like to thank Dr Miller for his valuable suggestions towards this research. I also thank Dr Mahapatra for providing suggestions and feedback in the course of this dissertation. I also thank Dr Khatri for providing valuable suggestions and insight during our discussions.

I thank Suresh Sivakumar for teaching me the concepts of LDPC codes and sharing his knowledge on decoding algorithms and decoder architectures. He has provided a lot of help in the initial stages of this dissertation. I thank Abhiram Prabhakar for teaching me techniques to generate H matrices for LDPC codes and also for providing suggestions throughout this period of study. I thank Euncheol Kim for teaching me VLSI concepts and helping me to understand low level circuit simulations. I also thank Nikhil Jayakumar for his help in running SPICE simulations for 90nm technology. I also thank Sanghoan Chang, Kiran Gunnam, Rohit Singhal, Pankaj Bhagavat and Rajeshwary Tayade for their valuable suggestions and feedback.

I also thank my family for standing by me in times of crisis and for motivating me to complete this dissertation. Above all, I am thankful to this PhD experience for making me learn a lot of valuable lessons in life.

TABLE OF CONTENTS

CHAPTER Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

CHAPTER I

INTRODUCTION

Linear Block Codes

Block codes have a set of message and parity bits. Message and parity bits together constitute a code word. Linear block codes are a special class of block codes where each bit can be expressed as the linear combination of other bits in the code word. Parity bits are included for error detection and correction. Parity information is redundant information that is used to detect and correct the errors caused in the message bits due to the noise in the communication channel. Parity bits are governed by parity check equations. Each equation is an exclusive-or of bits evaluating to 0. The parity is even. N is the number of coded bits. K represents the message bits. Therefore, N-K represents the number of parity bits in the codeword. The code can also be referred as (N, K) code. There a set of N-K parity equations corresponding to N-K parity bits. A unique set of N-K equations form a codeword. A different set of equations form a different code. For a given value of N, there are N-2 possible values of K ranging from 1 to N-1. For each value of N and K, there are different codes possible based on the parity check equation set. Each such code has 2^K code words, each codeword corresponding to a unique message vector of length K.

This dissertation follows the style of *IEEE Transactions on Computers*.

For example, consider a (7, 4) code. X1, X2, X3, X4 are the message bits and X5, X6, X7 are the parity bits. The parity equations are given by,

$X1 + X4 + X7 = 0$

$X2 + X3 + X6 = 0$

$X1 + X2 + X5 = 0$

The addition involved is modulo-2 addition, also expressed as two variable exclusive-or. If X1 and X4 are 0 and X7 is 1, the first parity check equation will be violated. The set of parity check equations can also be expressed in a two dimensional matrix format. Each row represents an equation. For N-K parity bits, N-K parity check equations are required. Hence there are N-K rows in the matrix. Each column represents a bit of code word. There are N bits that include K message bits and N-K parity check bits and hence N columns. Thus, the matrix is of dimension N-K by N. In each row of the matrix or the corresponding parity check equation, the bits that are involved in the parity check or the corresponding columns have a 1 on that row. The other entries are marked 0s. The matrix constructed in this fashion is called a parity check matrix, also referred to as an H matrix.

A valid codeword must satisfy $cH^T = 0$. The equation is equivalent to the system of parity check equations being satisfied. The parity check equations represented above correspond to the following H matrix.

$$H = \begin{matrix} 1001001 \\ 0110010 \\ 1100100 \end{matrix}$$

The error correcting capability can also be related to the properties of the parity check matrix. Random block codes are hard to decode. One of the earliest decoding algorithms was to compute error syndrome S.

The received code word is r, c is the transmitted codeword and e is the error code word. The e vector has 1s in positions where the code bits are in error.

$S = r H^T = (c + e) H^T = c H^T + e H^T = e H^T$, since $c H^T = 0$

Each syndrome can be associated with different error patterns. The error pattern that has the least number of errors is more likely to occur than the one with most errors. Hence, the one with smaller errors is associated with syndrome.

A table is generated that contains the different syndromes and the error patterns. This table is significantly large for a code length of a few hundreds. A large amount of memory is required to store this table which is prohibitive. This led to developing block codes with specific properties that has simple decoding algorithms.

One such class of linear block codes is Low density parity check codes (LDPC). LDPC codes have a sparse parity check matrix H. There are very few 1s in the matrix. Sparseness allows for very good error correcting capabilities. LDPC codes were first proposed by Gallager in 1962. He proposed regular LDPC codes where the H matrix contains j number of 1s in each column and k number of 1s in each row. The code is also termed a (j, k) regular LDPC code. The memory requirements for implementing LDPC

decoders are huge at that time. LDPC codes were completely forgotten until the 90s. Turbo codes [1], an extension of convolutional codes, were proposed in ’93 that has good error correcting properties. LDPC codes were then rediscovered in 1998 as the implementation of the decoders for these codes is feasible by that time. Irregular LDPC codes at high code lengths have better error correction performance than turbo codes [2]. Irregular LDPC codes do not have the same number of 1s in all rows/columns.

Here is an example of an H matrix for a regular linear block code with 3 1s in each column and 6 1s in each row.

$$H = \begin{matrix} 100011000100000101 \\ 010000101001001010 \\ 001100010010110000 \\ 110000110011000000 \\ 001100001100001100 \\ 000011000000110011 \\ 101000101000101000 \\ 000101000101000101 \\ 010010010010010010 \end{matrix}$$

There are uniform number of 1s in each row and column. The above code has code length N = 18 and K = 9. The relation connecting j, k, N and K can be obtained as follows.

Number of 1s in H matrix = N * number of 1s in a column

Number of 1s in H matrix = (N-K) * number of 1s in a row

$N * j = (N-K) * k$

$(N-K)/N = 1 – K/N = j/k$

$K/N = 1 – j/k$

Rate of a code R = fraction of message bits in a codeword = $K/N = 1 – j/k$

In a typical LDPC code, j = 3. Gallager [3] observed for regular LDPC codes that when j is equal to 3, the minimum distance of LDPC codes increases linearly with block length. The parameter k depends on the rate of the code for a given j. We consider rate ½ codes in this research as it is used in a wide variety of low power communication and storage applications. For a rate ½ code, R = ½ = 1- j/k, j/k = ½ and hence k = 6.

A typical short LDPC code has code length in the order of 1000s while a large LDPC code has N in the order of 10,000s.

Number of 1s in the H matrix = N * j

Number of entries in the H matrix = N * (N-K)

Fraction of 1s in the H matrix = j / (N-K)

For j = 3 and N-K = 1000, fraction of 1s in the H matrix is a low 0.003. The number of 1s is small compared to the number of 0s and hence the parity check matrix has sparse 1s.

Encoding of LDPC Codes

A modified parity check matrix is used for encoding LDPC codes. A systematic parity check matrix is derived from the original H matrix. The systematic parity check matrix has an identity matrix of dimension N-K by N-K along with a sub matrix of dimension N-K by K. This is obtained by guassian elimination which involves row and column transformations.

The systematic parity-check matrix H can be written as, $\mathbf{H} = \mathbf{[I \mid P\ ]}$

where **P** is the N-K x K parity check part and **I** is (*N*-*K*) x (*N*-*K*) identity matrix.

The generator matrix can be obtained as, $\mathbf{G} = \mathbf{[P^T \mid I]}$

where $\mathbf{P^T}$ represents *K* x (*N*-*K*) transposed parity check part and **I** represents *K* x *K* identity matrix.

$\mathbf{c} = \mathbf{mG}$, where c is the final codeword, m is the message vector of length K.

Encoding of binary LDPC code thus involves (*N*-*K*)x*K* binary multiplication and (*N*-*K*)x(*K*-1) binary addition to generate (*N*-*K*) parity check digits. Hence, LDPC encoding has complexity of the order of N^2. Several low complexity encoders have been presented in [4], [5], [6], [7].

Decoding of LDPC Codes

Decoding of LDPC codes is based on iterative message passing between the bit nodes and check nodes. LDPC codes are also represented by a bipartite graph. The information in the H matrix is contained in the bipartite graph. The bipartite graph contains

information about which coded bits (also referred as variable nodes or bit nodes) are involved in which parity checks (check nodes).

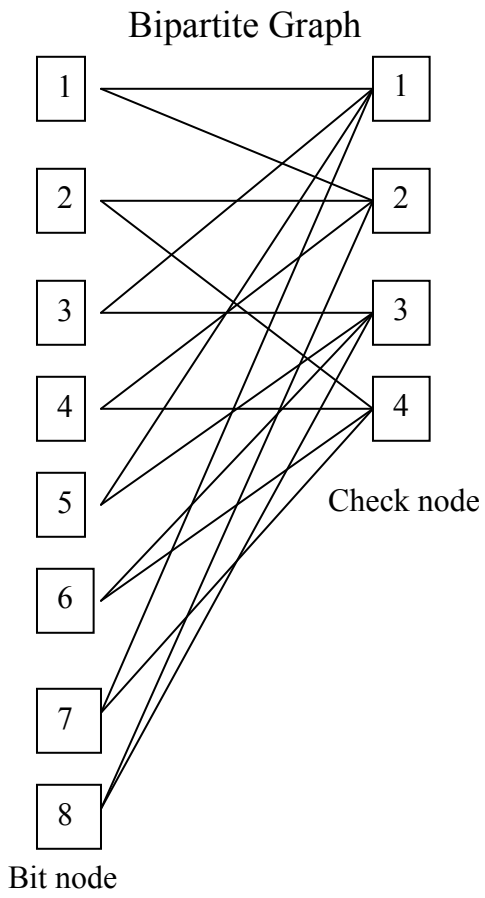

Figure 1. Message passing between the bit nodes and the check nodes

Each edge between the bit node and a check node in the bipartite graph indicates that the bit node is involved in that parity check equation (check node). Bit node 1 is connected to check nodes 1 and 2 as shown in figure 1. Bit node 1 is involved in two parity checks. Check node 1 is connected to bit nodes 1, 3, 5 and 7.

The message passing algorithm has two components – bit to check message and check to bit message. The bit nodes first pass messages to check nodes along the edges through which they are connected. The check nodes receive the messages and compute the parity check constraint. The parity check constraint is the modul0-2 addition of the received bit information. If the parity check is satisfied, i.e. if the number of 1s is even, the check node sends the same bit information back to the bit node. This is to inform the bit node that the bit node message is correct as it satisfies the parity check constraint. If the parity check is violated, the check node passes the compliment of the bit node message back to that bit node. It sends back 0 if the original bit node message was 1 and 1 if the original bit node message was 0 along the same edge through which the bit node sent a message to it. Each bit node receives messages from several check nodes and computes its message by performing majority voting on the received check node messages. For maximum efficiency, the check node message is excluded during the computation of an update for the same check node. This is to avoid circulating the same message back and forth between bit and check nodes. Each time, the bit node sends a new message received from other check nodes and hence convergence to the correct bit information is made possible. The computation at the bit node involves majority voting on the messages received from other check nodes and the original value received on the communication channel.

The above two operations, bit to check messages and check to bit messages constitute one iteration of message passing. The decoder hence involves several iterations until all

the parity check equations are satisfied or until sufficient number of iterations is carried out that constitute the 90 percent confidence interval.

The hard decoder discussed above has low error correction performance. To improve performance, more precision is included in the algorithm. Instead of one bit precision as in the hard decoder case, the message passed between bit nodes and check nodes are several bits of precision. 0 and 1 can be replaced with 16 different values between 0 and 1. The original information received from the channel is now sampled to soft information and it is converted to a log likelihood ratio (LLR), i.e. the natural logarithm of the ratio of the probability that the bit is a 1 to the probability that the bit is a 0. Probability ratios are used as messages in the decoding algorithm and messages converge either to 0 or a large value. In other words, the LLR value is expected to reach a large value. A large value of the LLR implies there is complete confidence in deciding the bit information and the sign bit (positive or negative large number) indicates whether the bit is a 0 or a 1.

In the soft value message passing algorithm, also referred to as the sum product algorithm, bit nodes pass LLR values to the check node. The parity check equation is computed at the check node as the product of hyperbolic tangent values of the LLRs. Then the hyperbolic tangent of the bit node LLR is removed from the product and after taking inverse hyperbolic tangent is passed to the bit node. The bit node implements the majority logic function in the hard decoder as the sum of LLR values. The description of the sum product algorithm is presented below.

We assume a BPSK modulation scheme where 1 is mapped to +1 and 0 is mapped to –1 and an Additive White Gaussian Noise (AWGN) Channel.

Let the transmitted codeword be 'c'= $c_0, c_1,\ldots,c_{N-1}$.

Let the noise added in the channel be 'n'= $n_0, n_1,\ldots,n_{N-1}$ where each n_i is an independent Gaussian random variable with mean 0 and variance σ^2.

The received channel values are given by $r_i = c_i + n_i$ for i=0 to N-1.

The received channel values for each bit are first converted to log-likelihood ratios (LLR). The channel LLR value for the i^{th} bit is given by,

$$Lch\langle c_i \,|\, r_i \rangle = \log\left(\frac{\Pr\langle c_i = 1 | r_i \rangle}{\Pr\langle c_i = 0 | r_i \rangle}\right) \qquad (1)$$

$$= \log\left(\frac{\Pr\langle r_i \,|\, c_i = 1\rangle}{\Pr\langle r_i \,|\, c_i = 0\rangle}\right) \; if \; \Pr(c_i = 1) = \Pr(c_i = 0)$$

$$\therefore Lch\langle c_i \,|\, r_i \rangle = \log\left(\frac{e^{\frac{(r_i - 1)^2}{-2\sigma^2}}}{e^{\frac{(r_i + 1)^2}{-2\sigma^2}}}\right) = \frac{2}{\sigma^2} r_i \qquad (2)$$

Let us define $\Psi(x) = \log(\tanh(|\frac{x}{2}|)$ (3)

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. Two different computations have to be performed during a decoding iteration, namely the bit node update and the check node update.

Consider the check node update at the nth check node. Let $L_c^q(i)$ represent the check to bit value along the i^{th} edge connected to the n^{th} check node during the q^{th} iteration. Enforcing the parity check constraint on the incoming bit to check values, $L_c^q(i)$ is given by,

$$L_c^q(i) = \Psi^{-1}\left(\sum_{k=1,k\neq i}^{k=t} \Psi(L_b^{q-1}(k)\Pi(-1)^k)\right), \forall i \qquad (4)$$

where t is the number of edges connected to the check node and index k refers to the k^{th} edge connected to the check node. This operation can be split into two steps to minimize the hardware required. First, a set of values M1, S1 are calculated.

$$\mathrm{M1} = \sum_k \Psi(L_b^{q-1}(k)) \qquad (5)$$

$$S1 = (-1)^{t-1} \prod_k sign(L_b^{q-1}(k)) \qquad (6)$$

M2 and S2 for the i^{th} edge are given by,

$$M2 = M1 - \Psi(L_b^{q-1}(i)) \qquad (7)$$

$$S2 = -S1 \times sign(L_b^{q-1}(i)) \qquad (8)$$

Now $L_c^q(i)$ is given by,

$$L_c^q(i) = S2 \times (\Psi^{-1}(M2)) \qquad (9)$$

Consider the bit node update. Let $L_b^{q+1}(i)$ represent the bit to check value along the i^{th} edge connected to the n^{th} bit node. $L_b^{q+1}(i)$ is given by

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

where index k refers to the edges connected to the n^{th} bit node.

After several iterations, the hard decision is performed on the coded bits. The hard decision for the nth bit is given by,

$$sign\,(L_{ch}(r_n) + \sum_k L_b(k)) \qquad (11)$$

There is another algorithm called min-sum algorithm, in which we just use min operations and sum operation. During the computation of the parity check constraint, the bit node with the least LLR value dominates the parity check. In other words, the result is as confident as the lowest LLR value involved in the parity check. Therefore, the minimum LLR among all the LLRs in a given parity check will be used as the check to bit update for all the bits in that check except the bit node having that minimum value. For the bit node with minimum LLR, the minimum value among the remaining bit nodes is used as the update. The bit to check update is same as that of the sum product algorithm and involves sum of the LLRs. This algorithm does not have a significant advantage over the sum product algorithm in terms of hardware implementation. Therefore, the sum product algorithm is used for hardware implementation of the decoder.

CHAPTER II

PREVIOUS DECODER ARCHITECTURES

Serial Architectures

LDPC decoder architectures are referred to as serial or parallel depending on how the bit node and check node updates are computed. The serial architecture proposed in [4] performs parity check computation in N clock cycles. There are three edges for each bit node. If the three edges were to be processed serially, the latency for one iteration stage will be 3*N clock cycles. In each clock cycle, the LLR values along the three edges of a bit node update the corresponding partial sums of the parity check. Therefore N clock cycles are needed to complete the N-K parity checks. Another N clock cycles are required to read parity check sums from memory and compute the check to bit node update and followed by the bit node update to be sent to the check node in the next iteration. In the second N cycles, the check to bit values are computed by subtracting the appropriate check sum with the bit to check value, bit node updates are completed and passed as inputs for the next stage. The author in [4] processes the parity check computation for the second frame in the second N clock cycles and therefore the net latency for one iteration stage per frame is N clock cycles. With 20 iterations, the total decoder latency is 20 * N clock cycles. A detailed description of the serial architecture [4] is presented below.

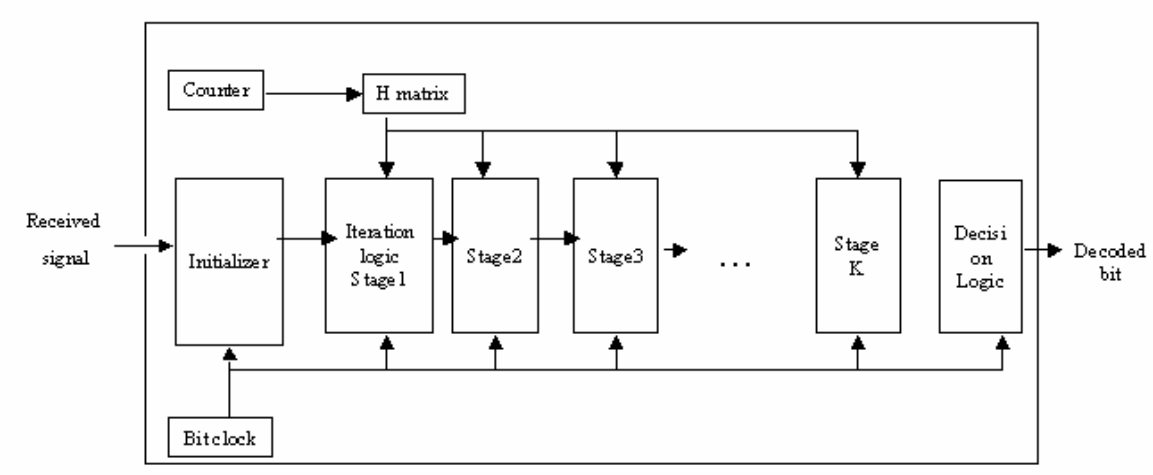

Figure 2. Overall decoder structure

The LDPC decoder structure of the serial architecture [4] is shown in Figure 2. The decoder has the following blocks, namely, Bit Clock, Counter, Initializer, H Matrix ROM, Iteration Stage and Decision Logic. The bit clock is the rate at which the all the components of the decoder are clocked. A counter is used to synchronize the decoding of the bits. The counter is 11 bit counter. The counter output is in the range 0...2039 and provides the address for the H matrix ROM. The H matrix ROM stores the positions of 1's in the parity check matrix. The H matrix provides the address of three memory locations (three parity checks) that correspond to the incoming bit node. The Initializer finds the LLR values for the received signal through a simple combinational circuit that implements equation 2. Each iteration stage takes bit to check LLR values from previous stage and generates bit to check LLR values for the next iteration stage. The decision logic computes the hard decision for each bit node and involves a few adders. The output of the decision logic is the decoded bit.

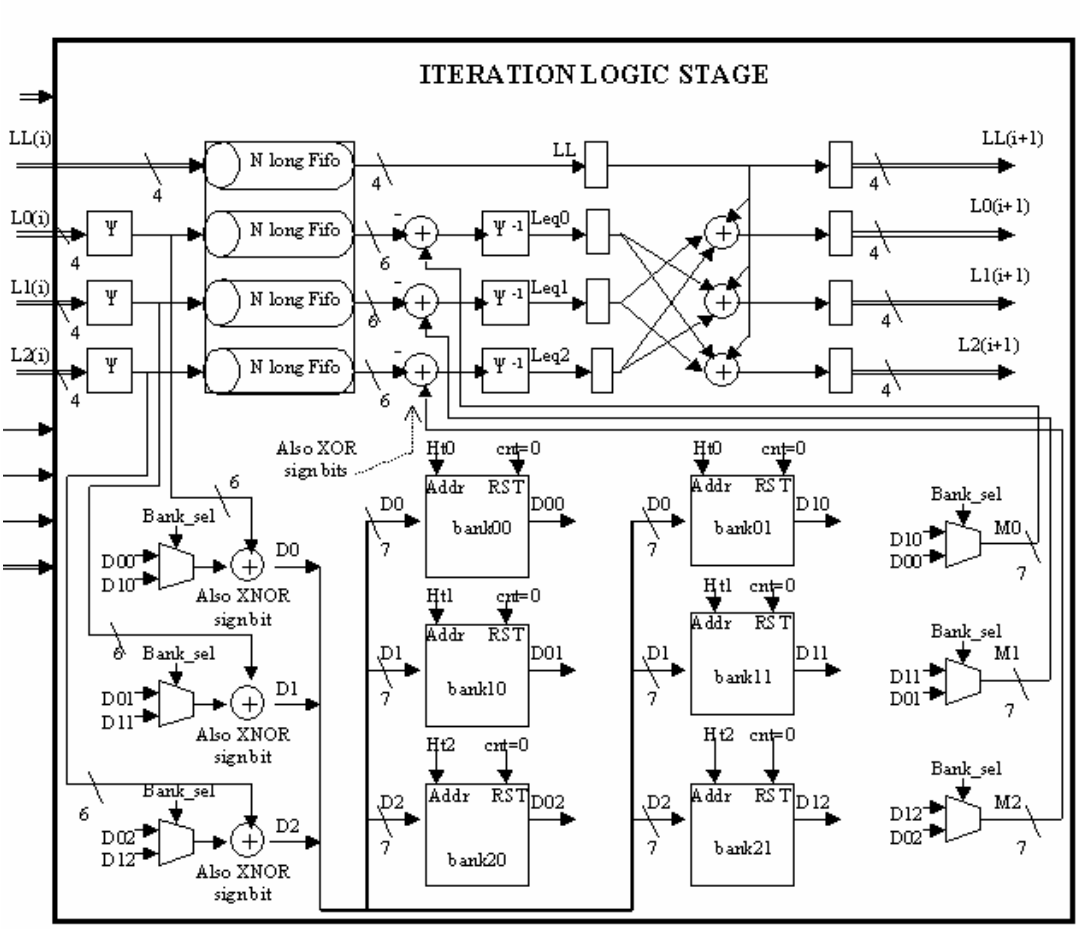

Figure 3. Iteration logic stage

The iteration logic stage is shown in Figure 3. The inputs to the iteration logic stage are the LLRs of the incoming bits. L0, L1 and L2 represent the LLRs along the three edges connected to a bit node. LL represents the output of the initializer and is the estimate from the channel. Ψ circuit (equation 3) operates on the incoming LLR values of each bit.

The check node update is performed by reading the previous sum from memory D_{00} and adding to the incoming bit LLR and stored back in the same memory location D_{00} (check node). Signal "addr" generated by the H matrix provides the exact location of the parity

checks within the memory block (BANK RAMs) that the edges connected to incoming bit node participate in. The “bank_sel” signal is 0 for even frames and 1 for odd frames so that even frames use the first set of banks and odd frames use the second set. The Ψ(LLR) values are stored in a FIFO while the check sums are computed, as the values are required for the subtraction operation. Subtraction (equations 7and 8) is performed to obtain the check to bit values. The bit node update is performed and the three bit to check values along with the Channel LLR values are passed to the next stage. Decision logic stage uses the sum of the bit to check values (L0, L1, L2 in Figure. 2) and LL to make a hard decision on the bit. The architecture uses 4 bits to represent the LLR values, 6 bits to represent the Ψ (LLR) values and 20 iterations of the decoding algorithm. There are two sets of bank RAMs to process two frames at same time. While bit node update is computed for the first frame by reading from the first set of bank RAMs, the partial check sum is computed for the second frame using the second set of bank RAMs. This is made possible by multiplexers that switch between D_{00} and D_{10}, D_{01} and D_{11}, D_{02} and D_{12}. Serial architecture, in general, has a low hardware complexity, but has a huge latency and hence a low throughput.

A staggered decoding schedule has been presented in [8] to reduce memory access and reduce gate complexity of the serial architecture. Approximations on the decoding algorithm that do not degrade the code performance on magnetic channels have been proposed in [9], [10]. Trade offs between serial and parallel architecture and their implementations on different platforms such as ASICs and FPGAs has been explored in [11], [12]. Hierarchical parity check matrix constructions that also make use of

ramanujam graphs to enable partial check node processing as a trade off between serial and parallel architecture is proposed in [13].

Fully Parallel Architectures

LDPC decoder can also be implemented as a fully parallel decoder. Parallel architecture processes all the bit nodes simultaneously as shown in figure 4.

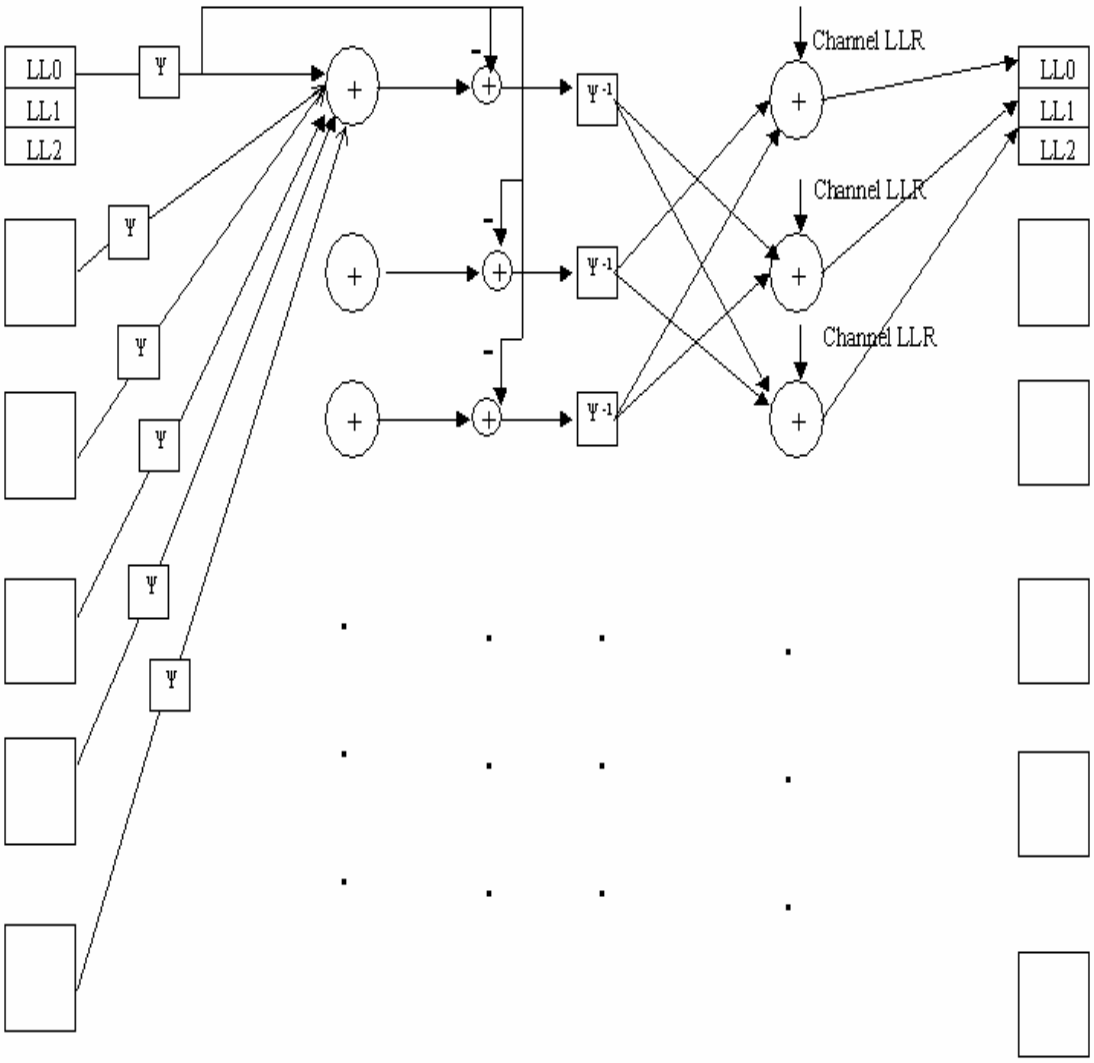

Figure 4. Fully parallel architecture

The bit node and check node updates are performed in one clock cycle. All the bit nodes and their LLRs are stored in registers. In figure 4, three registers are used to store the three different updates for each bit, namely LL0, LL1 and LL2. Thus the number of registers are 3*N, where N is the block length. In iteration, the LLR values in the registers are transformed in the ψ domain. ψ circuits are needed for all the three lines of each bit node and hence 3*N circuits required. At the check node, ψ^{-}values are added for all bit nodes. Hence, subtraction is required to obtain the update for an individual bit node. After subtraction, ψ-1 circuit is required to transform back to the LLR domain and again 3*N circuits are required. Then the final bit node update for each edge is obtained by a three input adder as shown in figure 3. ψ^{--1} computation, along with the processing of Lextrinsic values, is thus performed in the same clock cycle for all bit nodes. Hence, the iteration stage processes all the bit nodes in one clock cycle. This restricts the clock speed of the design, but the enhancement due to parallel processing is significant over the clock speed reduction resulting in a very high throughput. The high throughput is achieved at the expense of high hardware resource requirements. This is the general architecture for a fully parallel decoder.

Lee presented an approach that reduces decoder complexity, latency and intermediate storage in [14]. Lee also provided an in place algorithm and table look up for real-time cellular personal communication in his work. Another key issue is numerical precision during quantization and there is a trade off between precision and hardware resource requirement. The effect of precision on error performance is studied in [15], [16] and [17].

Hardware implementation issues were considered in great detail by the authors in [18], [19] with emphasis on reconfigurable hardware. An architecture that achieves low power by inducing structural regularity into the decoder is presented in [20]. An alternative method of implementing the check node function is presented in [21] by using a ROM based LUT in place of a linear piece wise approximation for check node computation.

High-throughput memory-efficient parallel architecture is presented in [22]. Three different optimizations have been performed to achieve a high throughput of over 1Gbps. First, the interconnect complexity is reduced by designing architecture-aware LDPC codes that have shifts of several parity check sub matrices used to construct the H matrix [23], [24]. This structured LDPC code helps localizing message passing between bit nodes and check nodes. Second, the memory overhead [25] is reduced by three-fourths using a turbo decoding based algorithm. Proper scheduling [26] further eliminates overhead due to storing all parity checks. Thus the code optimization, algorithm enhancement and a modified scheduling achieves a high throughput while maintaining low power. This together with programmable architecture having distributed memory is used to save power while achieving a high throughput of 1.6Gbps in [27]. AA LDPC code design is also extended to repeat accumulate codes [28], [29] and an unified decoder architecture is presented in [30].

Parallel decoders have been implemented in [31], [32], [33], [34]. The decoder in [34] achieves a throughput of 1Gbps and a low power requirement of 690mW. The low power

is attributed to structured code constructions based on quasi cyclic codes (cyclic shifts of identity matrix as sub matrices) that simplify interconnection network between the bit nodes and check nodes. Further optimizations resulted in a low power decoder with 220-mW power consumption and are presented in [35].

A memory based decoding architecture to achieve high throughput is presented in [36]. Registers are replaced by segmented memory and scheduling is performed that allows for less memory area in [37]. The architecture is also scalable due to the above reason.

LDPC decoder architecture is mapped onto parallel machines in [38], [39]. The mapping involves several steps and the search space is huge. To simplify the search space, clustering and cluster allocation is employed. Both these techniques are based on a modified min-cut algorithm for smaller codes and hence for smaller interconnection networks. For large codes, a genetic algorithm is best suited for clustering allocation. The architecture presented in [40] is extended to provide a domain programmable architecture that works for a variety of codes in [41].

Parallel architectures are proposed in [42], [43] that utilize parallely concatenated code structures that provide for a sparse generator matrix and hence less complex encoder and decoder structures.

Partly Parallel Architectures

A FPGA and ASIC implementation of a LDPC decoder for a partly parallel architecture is proposed in [44], [45] for irregular LDPC codes. A partly parallel decoder is presented in [46], [47], [48], [49] that constructs H matrix using several sub matrices that are shifts of the identity matrix and results in simplified decoder and encoder architectures. The implementation of the partly parallel decoder on a FPGA is shown in [44] and has a throughput of 54 Mbps. Issues of finite precision associated with LDPC decoder has been completely analyzed in [50] to arrive at the optimum bits to represent each message in the belief propagation algorithm. An overlapped message passing decoder using quasi cyclic LDPC codes constructed form shifts of the identity matrix is proposed in [51]. This approach reduces latency by overlapping bit to check and check to bit computations. A scalable decoder architecture for a class of structured LDPC codes derived from proto graphs is presented in [52]. A scalable check node centric architecture that achieves partial parallelization in processing the check nodes is presented in [53].

Special Architectures

LDPC decoder architecture has also been implemented on network-on-chip to achieve high throughput in [54]. The author also proposed novel power aware optimization techniques to save power. Several works [55], [56], [57] has been performed to implement LDPC encoder and decoder on a single chip. A vector signal processor that has independent computational units and an interleaver that routes bit-to-check and check-to-bit messages is presented in [58]. A LDPC IP core that satisfies DVB-S2 LDPC has been proposed in [59] that perform on a high code length of 65,536 bits.

A power efficient LDPC decoder architecture is presented in [60] that reduce power consumption due to memory access by providing a modified decoding schedule. Semi-parallel decoder architecture is presented in [61] that uses min-sum algorithm and structured LDPC codes to achieve low gate complexity. Partial data independence between the bit-to-check and check-to-bit messages has been used to provide a new decoder schedule [62] that reduces memory storage by 75% and memory access by 66% compared to previous architectures. Trellis decoding schedule for LDPC codes based on reliability metrics of BCJR algorithm results in faster convergence and low decoder latency for the decoder architecture is presented in [63]. The decoder also achieves low interconnect complexity, improved coding gain and lower memory access in addition to the above merits.

Structured code constructions that permit layered decoding are presented in [64] that achieve improved code performance with low gate complexity. A network of

programmable logic array based decoder is presented in [65]. The proposed decoder has low interconnect complexity by reducing routing congestion and the underlying code is constructed based on array codes. A decoder using structured LDPC codes that allows for full parameterization and a reconfigurable FPGA core that caters to a broad class of structured LDPC codes is presented in [66].

The architecture presented in this paper achieves scalability by extending the serial architecture [4]. We partition the partial check sum memory and scale the combinational logic to process multiple incoming bit values simultaneously. To reduce latency, our approach instantiates iteration stages multiple times (to pipeline the iteration stages for the code-word blocks). Since our implementation is less complex than that of the fully parallel architecture, the implementation is more practical. Our proposed architecture is a scalable design with a parallelization factor M.

CHAPTER III

PROPOSED SCALABLE ARCHITECTURE

Motivation

The purpose of this research is to develop a novel LDPC decoder architecture that achieves high throughput than the serial architecture. The serial architecture [4] takes N clock cycles to process an iteration of message passing for a frame. This limits the throughput of the decoder to a maximum of 50 Mbps. This severely restricts application of LDPC codes to low throughput applications and cannot be used in applications that require Gbps. Serial architecture cannot be used for optical communications.

A Parallel architecture was then proposed [23] that required prohibitively large hardware resources. We tried to explore an intermediate configuration that does not require large hardware resources but has an increased throughput. The serial architecture can process two bits if the conflict associated with memory access of two bit nodes can be removed after scaling the combinational logic that processes a bit node. The key idea is to construct a multi ported memory structure, which is switch logic along with a memory module. To make sure that two bit nodes access different memory blocks, the LDPC code has to be constructed so that the bit nodes fall into two groups of check nodes. Structuring or constraining H matrix usually results in degradation of code performance. It is interesting to note that the H matrix constructed in the above fashion resulted in no performance degradation.

If two bits can be processed in parallel, we started to look at 4 or 5 bit parallel processing. We were surprised to note that there was no degradation in code performance. Therefore, we looked at higher values of the parallelization factor, which is the number of bit nodes processed in parallel. We performed simulation of LDPC code with constraints for processing 20, 40, 80, 170 bits in parallel. The code structure allows us to have only parallelization factors that can divide N/6. This value N/6 is the number of parity checks (N/2 for rate ½ codes) divided by 3. We divide it by 3 because each bit node has three edges or three 1s in a column.

Finally, we observed a limit on the number of bit nodes that can be processed in parallel. This maximum value is N/6. When trying to process bit nodes more than N/6, we experience overlap in the placement of 1s in the rows of the H matrix. Therefore, we achieved a scalable architecture that can process integral divisors of N/6 ranging from 2 to N/6.

Parity Check Matrix Constructions

For a regular LDPC code, Gallager's constraints [3] for a regular LDPC code are given by, a) Each column has j (>=3) 1's.

b) Each row has k (> j) 1's.

For j =3, the minimum distance of these codes increases linearly with code length. Hence, we use j = 3 in our work. We also work on rate ½ codes and hence k = 6. Therefore, we employ a (3, 6) code.

For hardware implementation, the author in [4] has used an additional constraint given by, c) First, second and third 1's in a column are in three different row groups. The rows are initially divided into three groups. The H matrix is then constructed such that each of the three 1s in a column (three edges of the bit nodes) is in different row groups. A unique memory block in the serial architecture processes each row group. Therefore, the three edges of a bit node can be processed simultaneously by three different memory blocks.

The proposed scalable architecture imposes the following additional constraint on the H matrix given by, d) The row groups in iii) are further divided into M sub-groups. Each of the M subgroups is a block of memory that can read and write in parallel with the remaining M-1 sub-groups, making M bit parallel processing possible. The columns are grouped into blocks of M columns, where the ith block contains columns (i-1)*M+1 to

i*M. In each block, the first 1s in M consecutive columns are in distinct row sub-groups. Similarly, the second and third 1s along a column satisfy this parallelization constraint.

$$H = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

We explain the above constraint with an example. The H matrix that satisfies the above constraints for M=2 is shown above. Since M=2, two columns belong to each block. Block 1 contains columns 1 and 2. Block 2 contains columns 3 and 4. In this example, 6 checks are first divided into 3 groups of 2 checks to satisfy constraint iii). Each group is further divided into 2 subgroups of one check each. Overlapping of 1s within a block are avoided i.e. the 1s in consecutive columns within a group belong to check nodes of different subgroups and hence can be accessed simultaneously. Thus parallel processing of 2 bits is possible. In this approach, random permutations of the identity matrix are used and it provides more H matrix configurations than the structured H matrix with cyclic shifts of identity matrix as blocks as proposed in [17]. Moreover, the block size can be varied and specified as M. The variability in M gives rise to the proposed scalable architecture.

The maximum theoretical possible value of M depends on the value of N-K and column weight j: M=(N-K)/j. For the maximum value of M, there are k = j*N/(N-K) blocks to be processed serially. For a ½ rate code, the value of N/(N-K) is 2. The column weight (j) is 3 in our implementation. Therefore, 6 blocks need to be processed with each block being processed in one clock cycle. For a rate ½ code, the latency of the maximum parallelization case is always 6 clock cycles for the iteration logic stage. Therefore, the throughput of the decoder increases linearly with M.

LDPC Code Performance

Figure 5 shows the performance plot for a rate 1/2 code of length 2040 for various parallelization values between 20 and 340. The Signal to noise ratio (Energy per bit to noise ratio) is plotted on the X-axis and the bit/frame error rate (BER/FER) is plotted on the Y-axis. These are software simulation results with double precision accuracy for bit to check and check to bit messages. The message-passing algorithm was implemented in log domain and the extrinsic information (likelihood ratios) was clipped to 10 in magnitude. That is probabilities are between 10^{-10} and $1-10^{-10}$. An all zero data sequence and an Additive White Gaussian Channel (AWGN) are assumed in these simulations. Fifty block errors (frame or codeword) were observed and 30 iterations were used for decoding. The pseudo-random codes with parallelization constraints are all within 0.1db (for both BER and FER) of a pseudo-random code without any constraints. Hence there is no appreciable deterioration in the performance of the code due to the added parallelization constraints. Similar results were obtained with fixed metrics.

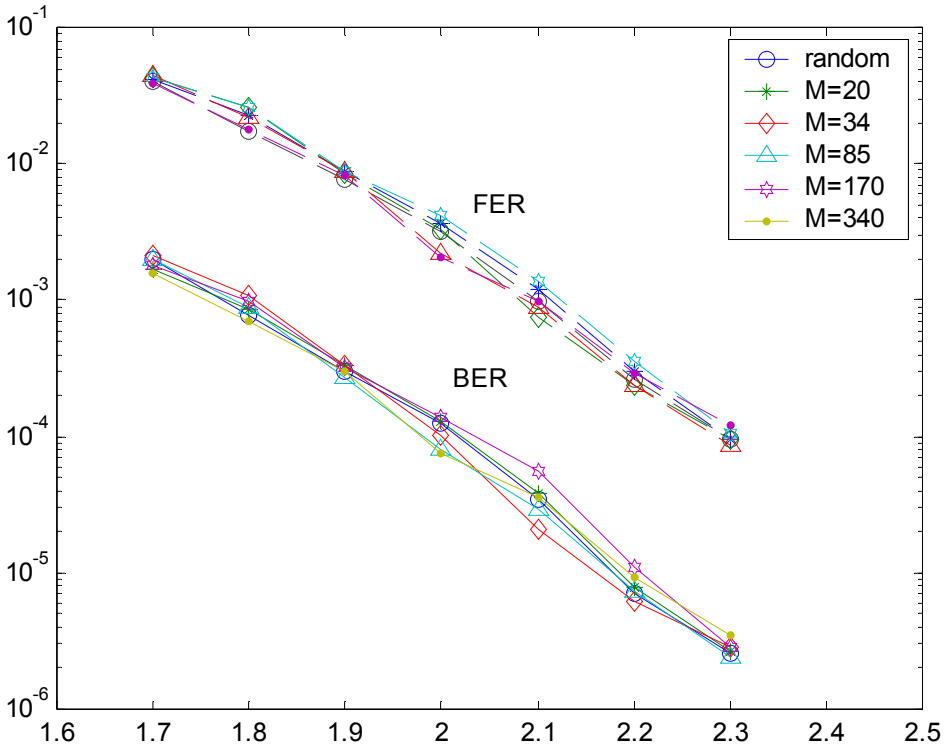

Figure 5. BER/FER simulation result for different values of M

Our results (Figure 5) indicate that there is no significant performance degradation even at a high degree of parallelization, N=2040, j=3, M=340.

Architecture Description

The serial architecture in [7] is extended for M bit parallel processing. The serial architecture takes N clock cycles for one stage of iterative decoding. The components of the serial architecture include an initializer, an iteration logic stage and a decision stage. There are 20 iteration stages. Since most of the decoder processing delay is due to the long chain of iteration stages, the initializer and decision stage overheads are ignored. Hence, decoder latency reduction is possible by minimizing the processing delay of the iteration logic stage. Instead of serially processing one bit every clock cycle, a group of M bits can be processed in each clock cycle. This implies that 3*M memory blocks (Serial architecture has 3 memory blocks to process three edges of a bit node) are required to process M bit nodes (3*M edges) in parallel. Also, the combinational logic part of the iteration stage is required for each edge of every bit node that is being processed in one clock cycle. All circuits that operate on LLRs of the bit nodes (the upper half of the iteration logic stage) have to be increased by M as shown in figure 6. This includes adder, subtractor, ψ circuit and FIFOs. The parallelization can be achieved because there is no explicit computation dependency between any two-bit nodes among the block of M consecutive bit nodes.

The partitioning of 3 memory blocks into 3*M memory blocks is shown in figure 7 for M=2. The multiplexers and adders required to compute the partial sum are also increased in number. The partial sum updated by the incoming LLR has to be routed to the appropriate memory block (parity check). This requires an additional switch-logic that takes M LLR values as inputs and routes them to M different memory blocks. A second switch logic is required at the output of the memory block to re-order the data that confers with the order of occurrence of the incoming bit node LLRs. The routing complexity of the switch logic is significant. In general, M*M lines have to be placed on the chip for routing M inputs to M different outputs. Layer assignment becomes difficult for the routing-lines. A NAND network can be used to implement the switch. Alternatively, an 8-to-1 MUX for each memory block can be used for switch. Each MUX needs different select lines; appropriate code design enables sharing select lines minimizing complexity.

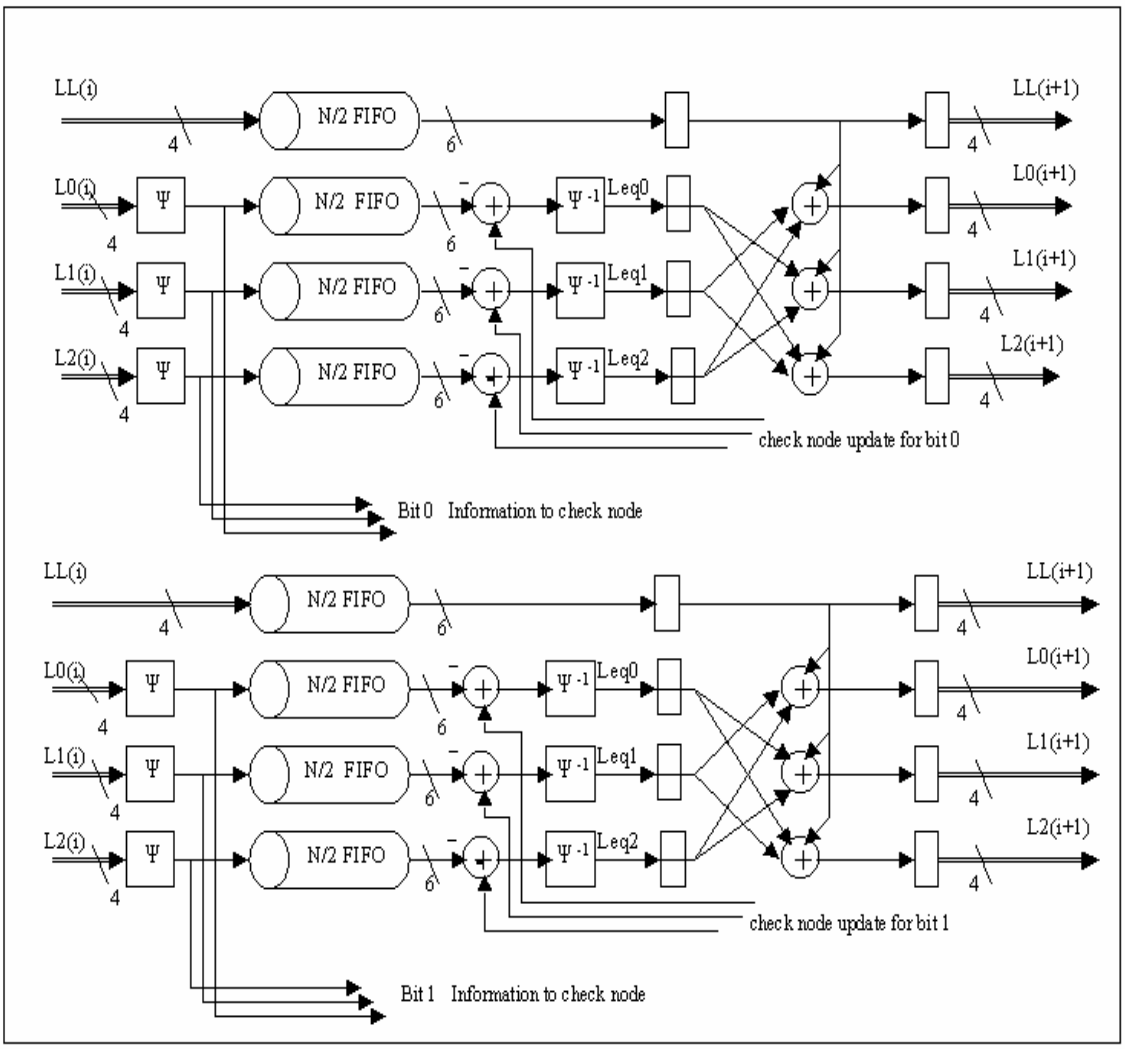

Figure.6 Iteration stage – combinational logic scaling

The individual block of memory is constructed using a register stack for an ASIC implementation. The code construction and the presence of 3*M memory blocks ensures that all the edges of M consecutive bit nodes participate in different parity check subgroups and all these 3*M parity checks are processed in parallel (figure 7). The maximum parallelization is when the number of memory blocks (3*M) is equal to the

total number of check nodes (N-K parity bits) and hence each block of memory is a single register. For a (3, 6) code and maximum possible parallelization the iteration logic stage takes six clock cycles.

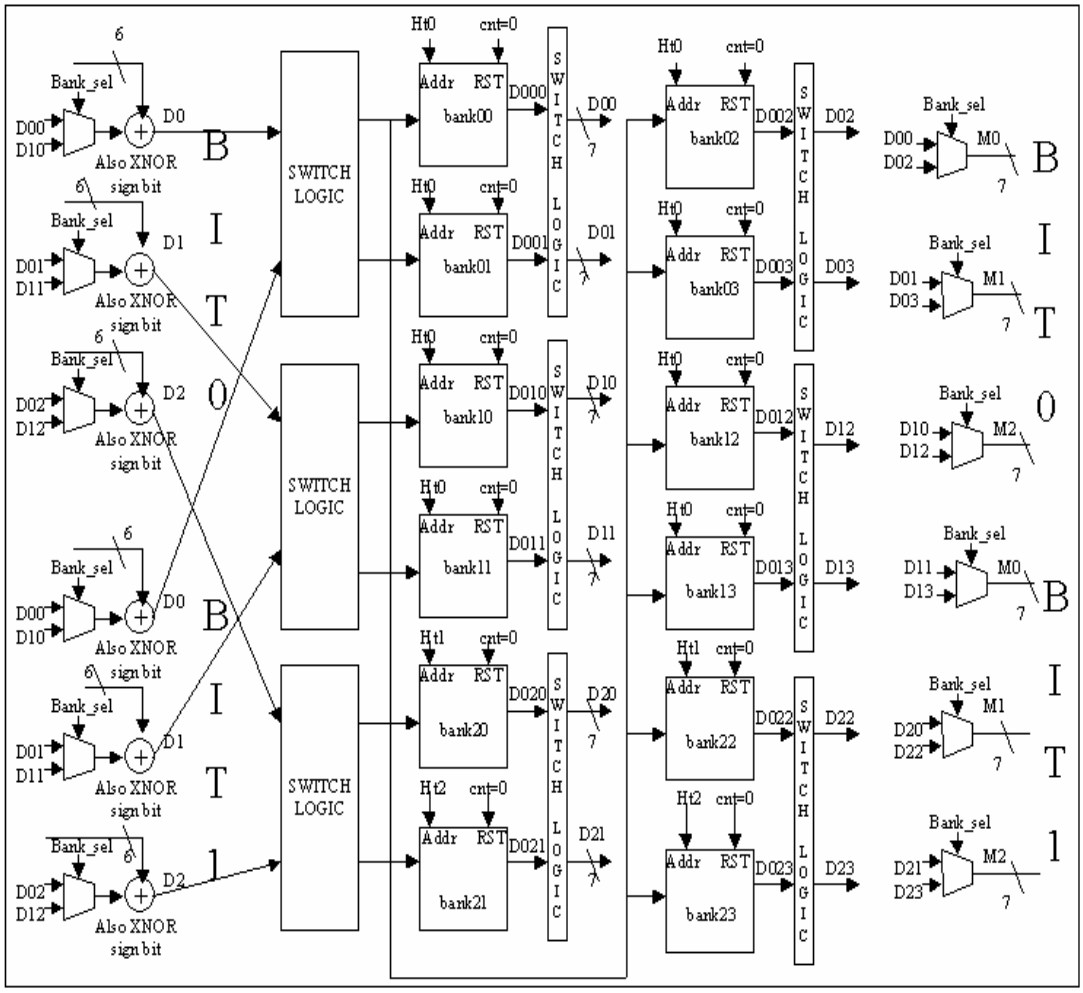

Figure.7 Iteration stage - memory partitioning

Since the information for all the M bits are available in parallel, the combinational logic part, namely, adder, subtractor and Ψ circuits required for bit node processing are scaled by a factor of M as shown in figure 6. The FIFO width is increased by a factor of M to store all the log-likelihood information for the M parallel processed bit nodes at the same instant. The length of the FIFO indicates the number of clock cycles required to complete parity check computation. FIFO length is hence reduced by a factor of M. Hence, the hardware complexity of FIFO is fixed. 3*M Ψ conversion circuits are required to process each of the 3 edges of the M parallel processed bit nodes. To process M bits simultaneously, subtractors and adders required for computing check to bit values and bit node updates are scaled by a factor of M. The log-likelihood ratios of all the M bit nodes are fed to the next iteration stage in parallel.

Decoder Throughput and Hardware Complexity

Parallelization is thus achieved by increasing the bus width, scaling the combinational logic and partitioning the memory. The other components of the decoder, initializer and decision logic are not affected by parallelization. Table I shows the hardware requirements of different components of the decoder for a serial architecture. The initializer and decision logic requirements are insignificant. Memory part of the decoder is the significant part of the hardware. But, the memory requirement is constant and does not change with M.

Table I. Hardware complexity of different decoder components

Decoder Components	Complexity (# of gates)
Initializer	100
H matrix ROM	367200
Iteration logic stage (1 stage)	
Combinational logic	2700
FIFO	326400
BANK RAM	142800
Decision logic	550

The FIFO and BANK RAM sizes are fixed. Hence the complexity of the logic (not affected by parallelization) can be obtained as

Constant = 100+367200+326400+142800+550 = 837,050 gates

The hardware complexity can be written in terms of M as

# of gates = 837,050 + M*2700 + 6*M*M

Table II. Key features of different architectures

Architecture	Memory (in bits)	Switch Logic
Serial	1.23828M	None
M = 20	1.23828M	6* (20 X 20)
M = 40	1.23828M	6* (40 X 40)
M = 170	1.23828M	6* (170 X 170)
M = 340	1.23828M	6* (340 X 340)

Table II lists the memory requirements and switch complexity of the different architectures. Memory requirement does not increase for the scalable architecture compared to the serial architecture, while switch complexity increases with M. The hardware complexity for different values of M, from the above equation, is presented in table III.

For lower values of M, the complexity of memory logic dominates the combinational logic complexity. For M = 170, the hardware complexity due to combinational logic scaling is lower than that of memory complexity. At M = 340, the hardware complexity of combinational logic part including switch logic is twice that of the memory part. Hence, high throughput can be obtained with hardware complexity comparable to that of the serial architecture with lower values of M.

Table III. Implementation complexity for different M values

Parallelization factor M	Implementation Complexity (# of gates)
1	839,750
20	893,450
34	935,786
85	1,109,900
170	1,469,450
340	2,448,650

Parallelization also increases delay overhead that limits the clock speed. Routing the LLR values dynamically to the memory blocks require switching logic. The switch and memory constitute the critical path of the logic stage and hence determine the clock speed for higher values of M. For value of M below 34, the combinational circuits that include the adder, ψ and ψ^{-1} circuits form the critical path and have a delay of 1.6ns. Timing analysis of the scalable architecture has been performed for various values of M for an ASIC implementation using TSMC 0.09μ technology. The speed estimation is based on worst-case speed model. Critical path is analyzed by hand estimation. The parasitic resistances and capacitances have been estimated for the 0.09μ technology model. The schematics are drawn in cadence and the netlist obtained from SPECTRE has been imported to SPICE simulation environment developed by Berkeley for 90nm technology. The path delay of the critical path is then determined through simulations and the clock speed is determined from this path delay.

The throughput of the decoder is calculated below:

# of bits processed per clock cycle = M

Maximum Throughput = M * Maximum Clock Rate/20

Table IV. Throughput of different architectural configurations

M	Maximum Clock Rate (Mhz)	Throughput (Gbps)
1	625	0.0313
20	625	0.625
34	500	0.85
85	386.1	1.641
170	297.6	2.53
340	129	2.193

The throughputs for different values of M are tabulated in table IV. Since the clock speed does not decrease significantly with different values of M (except M = 340), the

throughput is scaled by a factor of M. At M = 170, a very high throughput of 1.23Gbps is achievable through the scalable architecture.

The trade off between throughput and Implementation complexity (# of gates) is explored. The cost curve to determine the optimal value of M is shown in figure 8. The cost is computed as COST = Gates (Millions) + 1/(10*Throughput (Gbps))

The weights for gate count and throughput are such that low gate count is emphasized. Figure 8 shows the cost curve with cost along the Y-axis and parallelization factor M along the X-axis. The cost curve shown in the figure reaches a minimum at M = 40. Therefore, the optimal M value is 40. For M = 170, the design can be clocked at 297.6 Mhz and therefore can achieve a maximum throughput of (170/20) * 297.6 Mhz = 2.53 Gbps.

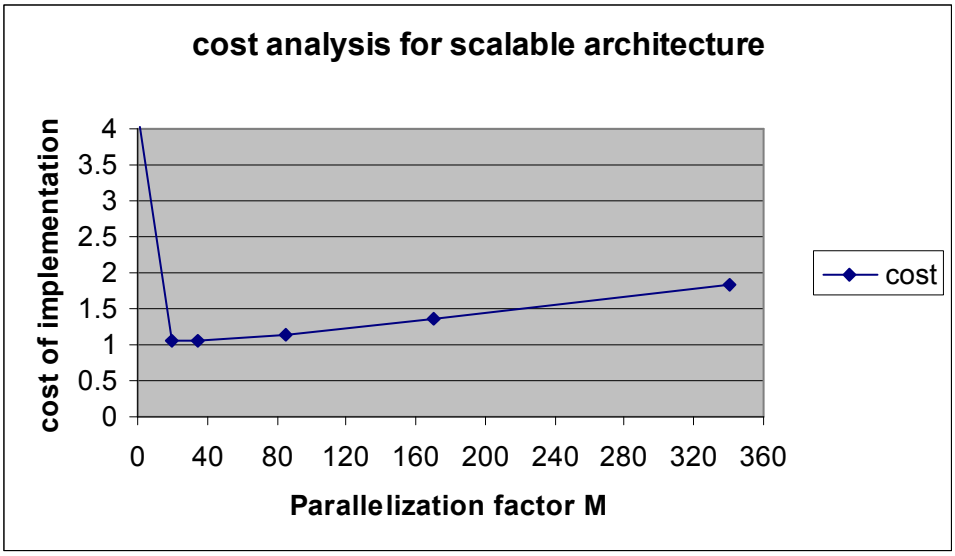

Figure.8 Throughput – gate complexity cost curve

CHAPTER IV

TILING APPROACH

Motivation

The scalable architecture has nice features such as reconfigurability and fine tuning ability. One of the important blocks in the iteration stage is the switch logic whose logic complexity and delay are significant. We studied several approaches to simplify switch logic. One of the approaches includes localizing routing among the bits and checks proposed in [67], [68].

In addition to switch logic complexity, H matrix ROM also requires hundreds of thousands of gates. H matrix ROM complexity can be reduced only if fewer entries are stored in ROM. We tried to propose an approach where lesser number of entries generates the complete H matrix. Hence, we propose constructing a limited number of patterns/blocks and repeat them to generate the H matrix. The patterns are repeated according to a set of rules that are discussed in detail in the following sections.

The tiling approach proposed in this research reduces storage of H matrix by $1/3^{rd}$. We observed that the switch logic can also be simplified by tiling H matrix. Each pattern or tile is a small switch that has the information about which bit node is routed to which check node. By embedding the patterns into the switches, switch logic is greatly simplified and there is no need to store the H matrix in a separate ROM.

Introduction

The performance of LDPC codes depends on the construction of the H matrix. Random constructions and irregular LDPC codes have been found to have better code performance. However, random constructions and irregular codes are difficult to implement in hardware. Hence constraints are imposed on the H matrix to permit reasonable hardware implementation cost and higher throughput. In [4], the rows of H matrix are divided into three groups so that all three edges of a column are into different row groups. This facilitates parallel processing of the three edges of each bit node in one clock cycle in the serial architecture.

In our scalable architecture, the constraint that successive columns do not overlap in row position is used to process M bits simultaneously (throughput enhanced by a factor of M compared to serial architecture). In [69], several identity matrix patterns are used to construct the H matrix for interconnect optimization. These constraints do not significantly degrade the performance of the LDPC code. We present additional constraints that permit tiling on H matrix for our high throughput scalable architecture to reduce the hardware cost.

Besides error correction, Tiling approach addresses key issues, namely latency and hardware implementation costs in LDPC decoder implementation. The tiling approach for the H matrix has the following advantages given by, a) Low hardware cost – The tiling approach reduces the hardware complexity of the decoder.

b) Low memory requirements – This approach reduces information stored in the design about the H matrix by one-third. Further, the H matrix information can be in built into the hardware components and hence the H matrix need not be stored in hardware as ROM. This leads to reduction of memory requirements of the decoder.

In general, there are several novel methods proposed to eliminate storage of H matrix or reduce design complexity by creating regularity in H matrix. Cyclic shifts of the identity matrix are used to construct H matrix [70] [71]. At high rates, anti pasch affine geometries [72] are used to construct H matrices. In [70], the author considered a pattern in the H matrix that has a single one in each row and each column. Cyclic shifts of an Identity matrix generate several such patterns. The patterns are used in [71] to construct the H matrix for interconnect optimization. In our approach, patterns that satisfy the parallelization constraint of the scalable architecture are generated. A small number of patterns are used repeatedly to construct the complete parity check matrix. The repetition of patterns facilitates reduction of hardware complexity and delay for the decoding of LDPC codes. The number of patterns that can be generated in this approach are higher than the patterns generated in [70] because [70] uses only cyclic shifts of the identity matrix. In this approach, any random permutation of 1s in the identity matrix is considered. In contrast to [70], patterns are also repeated so that the decoder stores fewer patterns or sub-matrices.

We can present the tiling approach on anyone of the above platforms that partition and arrange codes for reduced switching. In this paper, tiling is exemplified using random H matrix. The H matrix is constructed using the tiling approach by repeating random patterns for a code length N = 2040 and simulated with the software implementation of the decoder to determine the code performance. Simulation results indicate that the BER and FER performance are not compromised due to H matrix tiling.

The scaleable architecture hence imposes the following additional constraint on the H matrix for parallelizing bit node processing by a factor of M (Parallelization factor). The columns are grouped in to blocks of M columns, where the ith block contains columns (i-1)M+1 to iM. In each block, the first 1s in M consecutive columns do not overlap in row position. Similarly, the second and third 1s do not overlap in row position. Tiling approach is built on top of our parallel (and scaleable) architecture constraint to reduce hardware complexity.

Tiling Approach

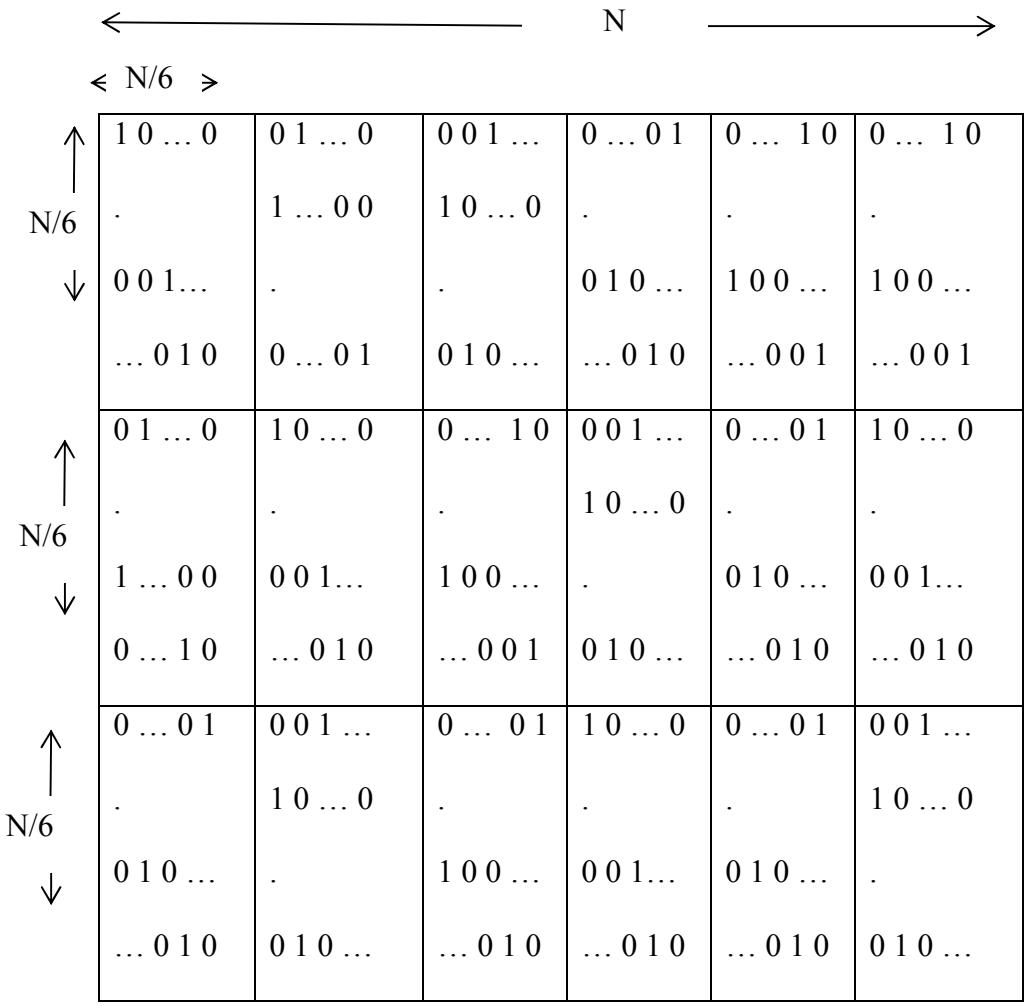

Figure 9. Parity check matrix for a parallelization factor M = N/6

The example H matrix shown in figure 9 is a 1020-by-2040 matrix. The block length N = 2040 for this example and is a rate ½ code. The maximum parallelization possible for a (3,6) code with three 1s along a column is given by parallelization factor M = N/6. There are N/2 rows in the H matrix and three 1s in each column that prompts partitioning the rows into three groups. Each row group has N/6 rows. In this work, N = 2040 and hence

the maximum parallelization is 340. With 6 1s in each row, the block free of 1s overlap in row position (for any two columns) is N/6 columns. For a rate ½ code, the maximum parallelization possible is a square matrix block of N/6 by N/6. Tiling applied on parallelization implies each N/6 by N/6 block treated as a pattern. The pattern (tile) is hence a square matrix of dimension 340-by-340. There are 18 tiles in the H matrix. Only five different patterns are used in the above H matrix. The 18 tiles are constructed using only five different patterns. These five patterns can be stored in memory without having to store the entire H matrix causing a significant reduction in memory requirements. The placement of the five patterns satisfies the following constraints for high BER performance.

If two patterns are placed next to each other along a column block, the two patterns are not placed in another column block in the same manner. Pattern 2 is placed next to pattern 1 in the first column block. If pattern 1 is placed along the same row block in another column block, pattern 2 is not placed close to it. This constraint eliminates cycles of length 4 among the patterns themselves.

If there are l patterns and m tile positions, each pattern is repeated a minimum of [m/l] for fairness. The patterns are placed as randomly as possible to have high BER performance.

P1	P4	P2	P3	P5	P1
P2	P5	P1	P4	P3	P4
P3	P1	P4	P2	P2	P5

P1	P2	P3	P4	P5	P6
P2	P4	P1	P6	P3	P5
P3	P6	P5	P2	P1	P4

Figure10. H matrix for parallelization factor of N/6 and 5, 6 patterns

P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P1	P2
P2	P4	P1	P10	P6	P4	P9	P5	P7	P3	P8	P3
P3	P7	P10	P8	P4	P9	P1	P7	P2	P5	P6	P8

P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12
P2	P4	P7	P3	P6	P8	P1	P12	P11	P5	P9	P10
P11	P10	P9	P1	P12	P2	P5	P7	P6	P8	P3	P4

Figure 11. H matrix for M = N/12 and 10, 12 patterns

In figure 10, 5 patterns have been used. All the patterns are repeated at least once in each row, and with an extra slot, one of the patterns is repeated in each row. Patterns 1, 4 and 2

are repeated in rows 1, 2 and 3 respectively. The Distribution is fair for almost all pattern numbers. The pattern numbers by themselves make no significant difference. Pattern 1 and 2 can be interchanged and the error correction performance remains the same. In figure 2, there are six patterns and hence each pattern takes one slot in each row.

The situation is slightly different in figure 11. The parallelization factor is M = N/12 which is exactly one half of the maximum parallelization, 170 in this case. Hence, the tile size is 340 by 170 and that gives place to 36 pattern slots to fill with patterns. The H matrix is filled with 10 and 12 patterns in figure 3. The placement of patterns in the H matrix has been manual, while each pattern is generated randomly satisfying the parallelization constraint. A reduction of $1/3^{rd}$ is achieved in H matrix memory requirements.

Also, the patterns can be embedded into the decoder logic without the need for external memory. Patterns can be used to design static switches that route bit nodes to check nodes and vice versa. By repeating patterns, the number of static switches required decrease. The switch design can be implemented using multiplexers and with appropriate connection of multiplexer inputs to the bit nodes that have to be routed. This eliminates the need for external memory to store H matrix and hence any memory access is not required. This improves the timing of the switch logic and hence the decoder. The decoder can be clocked at a higher rate that gives rise to higher throughput.

Array codes also use cyclic shifts of the identity matrix as patterns to fill the H matrix. Cyclic shifts of identity matrix are also used as patterns in [49], [50], [51]. In [52], Storage of multiple check-to-bit messages is avoided and interconnection network simplified by adopting identity matrix and its cyclic shifts in constructing H matrix. Other than randomly generated patterns, tiling approach can also use cyclic shifts of identity matrix as patterns as used by other approaches. In [53], high rate LDPC codes have been constructed using anti-pasch affine geometries. Tiling can be applied to high rate codes and in that case, these geometry codes can be used to construct patterns. The difference between tiling approach and other approaches proposed in the past is that patterns or tiles are repeated in the H matrix to simplify decoder architecture.

Simulation Results

H matrix is constructed using the tiling approach for a code length N = 2040 and simulated with the decoder implemented in software to determine the bit error rate (BER) and frame error rate (FER) performance. For a given value of M, the number of slots in the H matrix is determined.

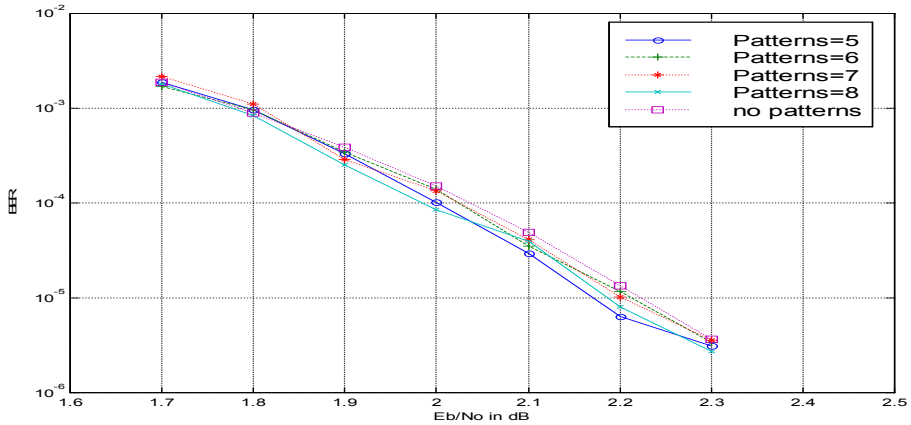

Figure 12. BER performance for M = N/6 and different patterns

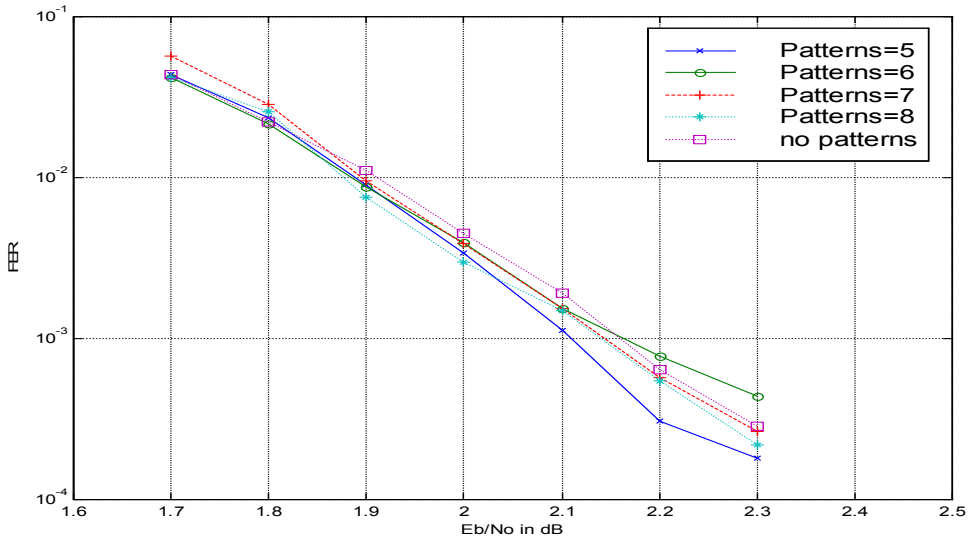

Figure 13. FER performance for M=N/6 and different patterns

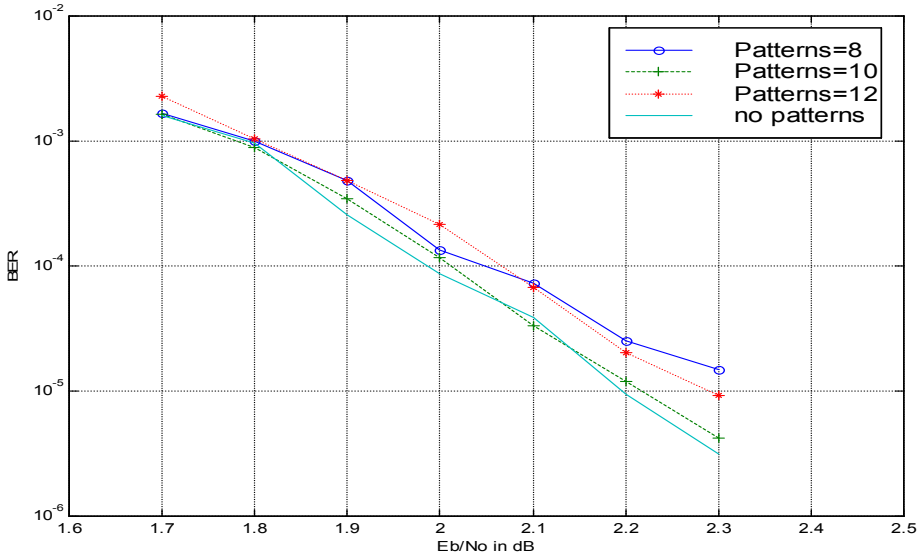

Figure 14. BER performance for M = N/12 and different patterns

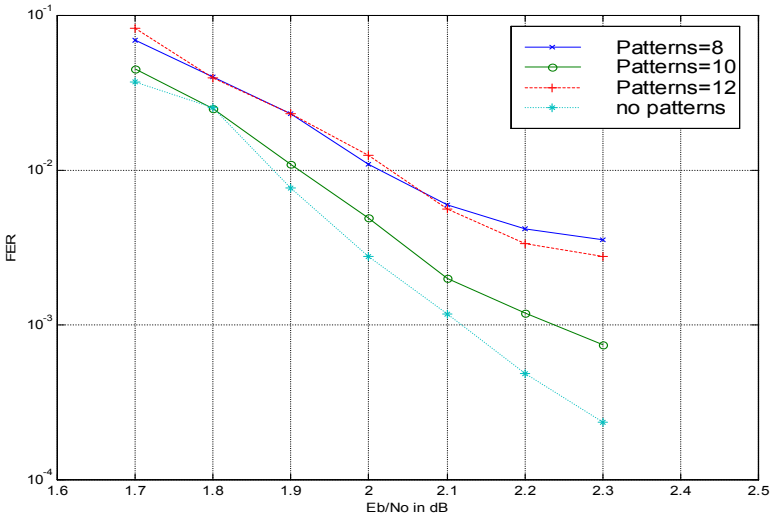

Figure 15. FER performance for M=N/12 and different patterns

For M=340, the H matrix is constructed with 5, 6, 7 and 8 patterns. The BER curve in figure 12 shows that there is no significant degradation in BER performance (less than 0.2 dB) with limited number of patterns to fill the H matrix. The FER curve (figure 13)

also shows very good performance of the tiling approach with less than 0.2 db of difference in Eb/No at various bit error rates.

Simulations are also performed for a different parallelization factor of 170 (M = N/12). The number of H matrix slots is 36 and has been filled with 8, 10 and 12 patterns

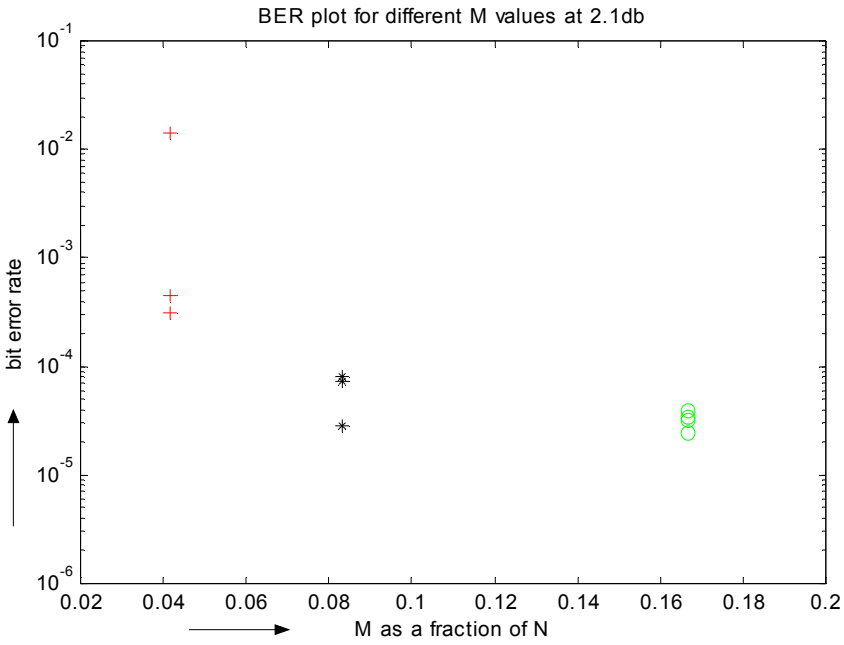

Figure16. BER performance for different M values at 2.1db

respectively. The BER and FER results (figures 14, 15) for 10 patterns are close to that of the random construction (no tiling). Hence, this illustrates that tiling is effective for H matrix constructions and preserves the error correction capability of the code.

Figures 16 and 17 show the bit error rate plot for three different values of M. Figure 16 shows the BER plot at Eb/No = 2.1 db and figure 17 shows the plot for Eb/No = 2.3 db. Both the figures plot M = N/6, N/12 and N/24. The bit error rate performance degrades as M decreases. The BER plot for M = N/24 has significant performance degradation for all the different patterns used, namely 8, 12 and 16. As the number of patterns increases with decrease in the value of M, the patterns generated have more cycles among them and hence degrade the BER performance. The FER performance also degrades for small values of M with the tiling approach. The hardware complexity and throughput increases as the number of patterns increase for lower values of M. Hence tiling approach reduces the hardware complexity and has high error correction capability for higher values of M (N/6, N/12) for a typical code length N = 2040.

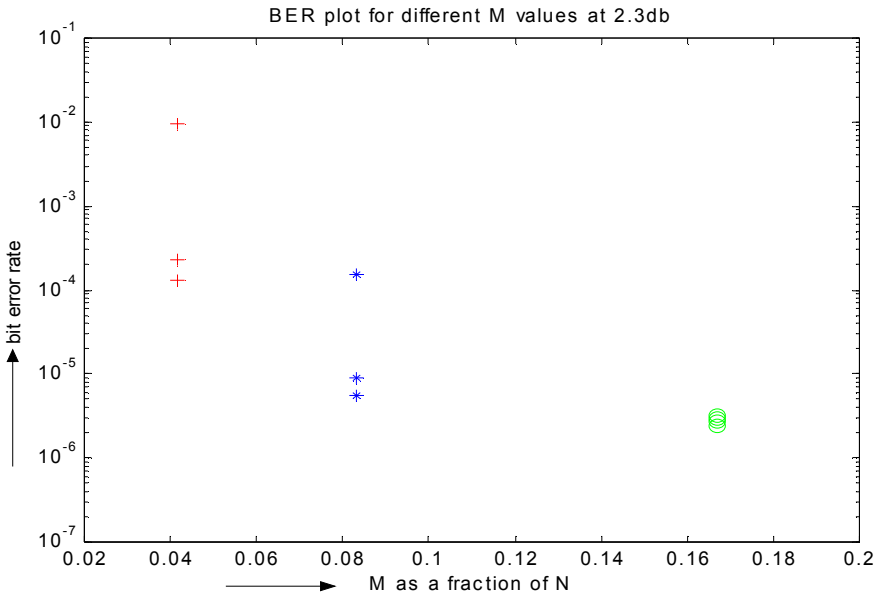

Figure 17. BER plot for different M values at Eb/No= 2.3db

Tiling Approach: Architecture Enhancements

There are several benefits due to H matrix tiling on the decoder architecture. The significant merit of tiling is that the H matrix need not be stored in memory. It can be directly embedded into the switch logic. The switch logic routes the bit nodes to the appropriate memory block. For a given parallelization factor M, there are M memory blocks. Each Memory block contains N/6M entries. In a given clock cycle, M bit nodes are to be routed at a time to M different memory blocks.

In H matrix, M bit nodes connected to M checks form a pattern. Each bit node has three edges that are placed in different check groups. There are three patterns for each group of M bit nodes. There are N bit nodes and hence 3*N/M patterns in the H matrix. For N= 2040 and M = 340, there are a total of 18 patterns. For the switch logic, 3 patterns are used per clock cycle. Without Tiling, 18 patterns have to be stored. By repeating patterns regularly, 5 or 6 patterns are sufficient. In each clock cycle, three of the six patterns are used for three different components of switch logic.

Each pattern can be implemented using M 8-to-1 MUX's, one for each check node. The 6 inputs for each MUX are the six bit nodes that are connected to check node. The select lines can be shared between all the check nodes that belong to the same pattern. This sharing of select lines between the different multiplexers simplifies the switch logic hardware. It results in less gate delay and less gate complexity.

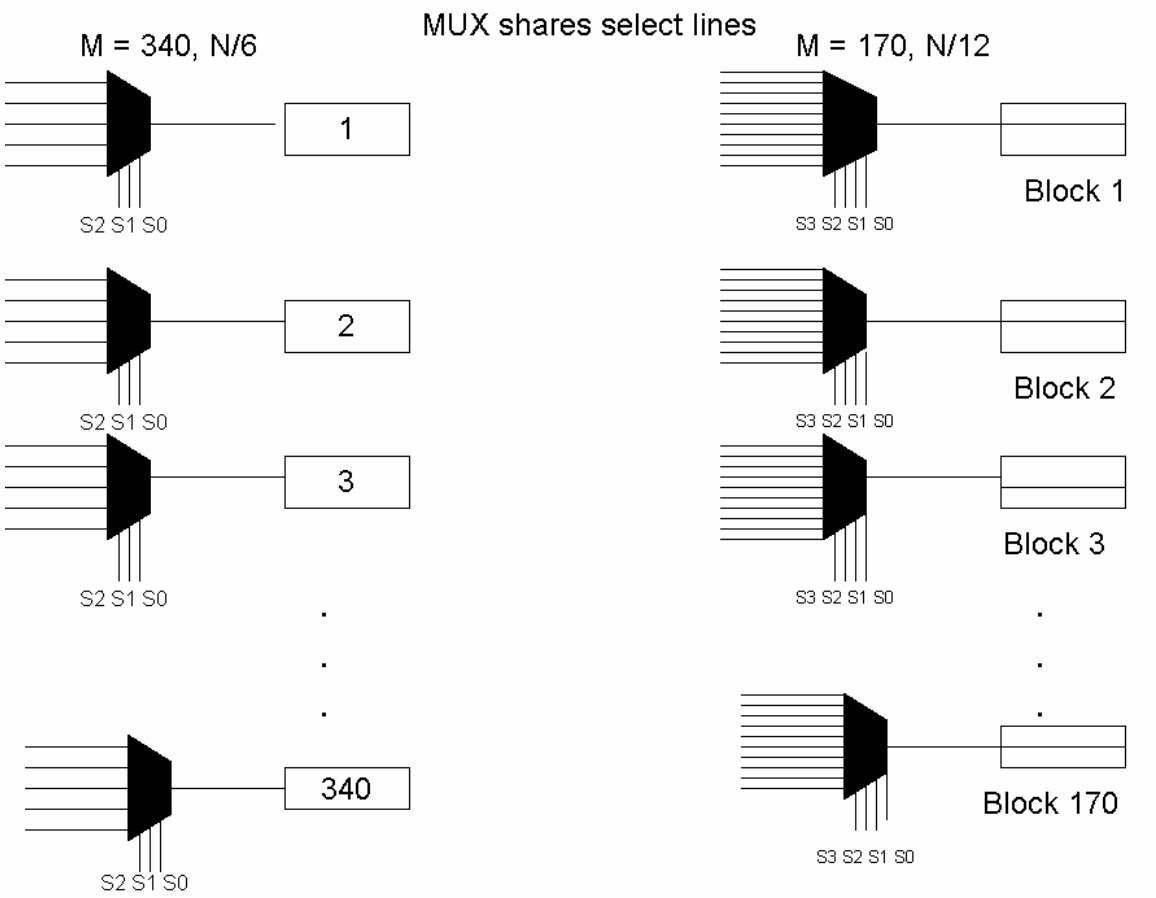

Figure18. Tiling: Switch logic

There are several benefits due to H matrix tiling on the decoder architecture. There is significant reduction in hardware complexity as H matrix is no longer stored in ROM. The patterns that generate H matrix are implemented using 8-to-1 MUX's, one for each check node. There a total of 340 8-to-1 MUXs for M = 340 and 170 16-to-1 MUXs shown in figure 18. The number of inputs to the MUX is equal to the number of patterns. Therefore 8-to-1 MUX and 16-to-1 MUX are custom designed as 6-to-1 MUX and 10-to-1 MUX respectively. Further, the select lines are shared between all the M MUXs. This sharing of select lines between the different multiplexers simplifies the switch logic

hardware. It results in less gate complexity compared to the scalable architecture. The architecture schematics for M = 170 and M = 340 are imported to SPICE and SPICE simulations have been performed using 90nm technology and the clock speed of the decoder for the critical path involving the switch logic and memory is computed and shown in table V. The corresponding gate count for M = 170 and M = 340 are also shown in table V. The clock speeds are identical for M = 340 but there is a small decrement in clock speed for M = 170. There is a significant reduction in the gate count due to tiling. For both values of M, the hardware complexity is almost halved. We observe a significant reduction in gate complexity due to the proposed tiling approach for higher values of M (N/6, N/12).

Table V. Tiling approach – gate count and clock speed

	Clock Speed (Mhz)		Gate Count (Millions)	
Architecture	M=170	M=340	M=170	M=340
Scalable architecture	297.6	129	1.469	2.448
Tiling on scalable architecture	150.2	147	0.947	1.424

CHAPTER V

POWER SAVING SCHEME

Motivation

We proposed a scalable architecture and then a tiling approach that enhances the scalable architecture. Architectural improvements are coming to a phase when there is no room for further modification. Current research has had a look at all major and minor architecture modifications arising out of the sum product algorithm. As we near saturation at the architectural level, it is a good venture to explore algorithmic optimizations, or develop a different code [54], [72], [73] on the lines of LDPC that can have a simple decoding algorithm. The approximations of sum product algorithm and the min sum algorithm have already been explored.

Another feature of the LDPC decoder to be explored is the iterative nature of the decoder. Significant effort is underway to unroll the iterations and construct a one-shot computation to generate the result. Analog circuits are one such method of implementing iterations by letting the capacitance or device float to achieve convergence. The level of accuracy achieved from these computations and whether the error rate performance is acceptable is still a subject of concern.

Controlled iterations [57], [58] is a better alternative and is a trade-off between fixed iterations and one shot computation. Packet profile is studied to arrive at a dynamic iteration controller. In this approach, different packets go through different number of

iterations. This is a traditional approach already known to researchers. After each soft decision iteration, a parity checker can be instantiated to check if all the parity checks are satisfied. This increases the latency of the decoder and therefore, researchers have settled for a fixed number of iterations.

After observing the packet profile, we recognized the placement of a single parity checker that can achieve a better decoder throughput than that of fixed iterations (which wastes lot of iterations) and the "parity checker after every iteration" scheme that spends lot of clock cycles on parity check evaluation. The optimum placement of the parity checker has to be determined.

We determined the range of values for the placement of parity checker by observing the packet profile. The exact value is determined after simulating the decoder with different placements of the parity checker.

In a totally new development while exploring the application of a 1 bit soft decision stage (hard decision iteration stage), we observed that the hard iteration stage is able to correct all errors if the number of parity checks in error (also a measure of the number of errors in the packet) in the packet is below a threshold value irrespective of the signal to noise ratio of the packet. It is to be noted that at the receiver, we have access to only the number of checks in error unless a fixed code word is used which is rarely the case in practical applications.

Low thresholds of parity checks in error can only be achieved at high signal to noise ratios where the number of errors in a packet is at a minimum. But such high signal to noise ratios is rarely used in practical applications. So, we tried to explore scenarios where this high signal to noise ratio occurs. We found that after a fixed number of soft decision iterations, the number of parity checks in error is reduced to a low value which is within the threshold. The placement of a parity checker after a fixed number of soft iterations makes sure that the threshold is achieved for most packets that they can be corrected through hard iteration stages. A few packets still go through soft decisions after parity checker. We call them critical packets and that cannot be avoided. Merging the dynamic scheme with the hard iteration stage has resulted in a remarkable improvement in decoder performance. Reduced gate count and high throughput are achieved through the above configuration. LDPC bit error rate performance does not suffer in this new configuration and is on par with the other decoders.

The advantages of the proposed dynamic scheme are so significant that we also propose a fixed scheme that works with hard iterations after a fixed number of soft decisions as an alternative to conventional decoders that work with fixed number of soft iterations. The increase in hardware resources for hard iteration decoder is minimum as the hard iteration stage has a negligible gate count compared to the soft decision iteration stage. We observe a reduction in the number of soft iterations as compared to conventional decoders and that result in significant hardware savings and improved throughput.

Introduction

The LDPC decoder is typically set to run for fixed number of iterations, say 20. In practice this number depends on the code rate. The proposed low-power approaches explore the fact that not all packets have the same severity of errors. Some packets need less number of iterations to yield error-free state. If all packets are run with the decoder with fixed 20 iterations, there is considerable waste of resource and energy.

Our design is based on a fully-parallel decoder that has 20-iteration instantiations [49]. The packets are thus decoded in the 20-stage pipeline processor. We studied a variety of architectural configurations dynamically reduce power. However, our initial simulation results pointed to the fact that, vast majority of all packets require initial specific number (say X) of soft iterations in order for the code word to eventually converge. Apparently, dynamically managing the first X iterations turned out to be less beneficial.

The low-power decoder scheme presented in this paper initially processes packets through an X number of soft iterations. Then the partially decoded packets are subjected to a hard decision, a parity checker that computes cH^T. If the result of cH^T evaluation is a zero, then a valid code word is reached and therefore no additional iteration is necessary. If any of the checks in error, i.e., the number of non-zero entries in cH^T, then the packet is passed to the second stage of soft decision decoder unit for the remaining 20-X iterations. For high signal-to-noise-ratio (SNR) ranges, the second stage rarely becomes active (**STEP**s up) and operates at a higher-**GAIN** mode. We refer to this process as **GAIN-STEP** for low-power.

For applications where a very high SNR is expected, we further reduce the circuit complexity by completely eliminating the dynamically managed two step process. Instead, we install a hard decision stage at the end of a fixed number of soft-iterations. Number of soft-iteration stages is empirically optimized in this case, and the trade off is a slight degradation in the code performance. The number of soft iterations, in this static design, is determined as 12 for the rate ½ code. Thus the power consumption for the static design is slightly increased for iteration units compared to that of dynamic design. However, overall energy use has decreased significantly since the absence of control and packet buffering of the dynamic scheme. The BER/FER simulation results show that the two proposed schemes negligibly compromise the coding gain. The soft decision iteration stage has already been discussed. We discuss the hard iteration stage below.

Hard Iteration

Check Node Update

$$Check(i) = \sum_{k=1}^{k=t} Bit(k), \forall i$$

The summation is modulo-2 addition (XOR)

If (check(i) = 0) , each bit node receives the same binary value (message) that it passed to the check node.

If (check(i) = 1), each bit node connected to the check node receives the compliment of the binary value that it passed to the check node in the current iteration. Each bit node

receives three updates from three different checks (Bit-1_q(i), Bit-2_q(i) and Bit-3_q(i)) along with the channel value Bit-ch_q(i).

Bit Node Update

The bit node update for the next iteration q+1 is computed as follows,

Bit-1_{q+1}(i) = Bit-2_q(i) * Bit-3_q(i) + Bit-2_q(i) * Bit-ch_q(i) + Bit-3_q(i) * Bit-ch_q(i)

Bit-2_{q+1}(i) = Bit-1_q(i) * Bit-3_q(i) + Bit-1_q(i) * Bit-ch_q(i) + Bit-3_q(i) * Bit-ch_q(i)

Bit-3_{q+1}(i) = Bit-1_q(i) * Bit-2_q(i) + Bit-1_q(i) * Bit-ch_q(i) + Bit-2_q(i) * Bit-ch_q(i)

Thus, the message passed from a given check node is excluded when messages are computed to be passed to that particular check node for the next iteration. Hard decision is finally performed by doing majority voting on the three bit node updates Bit-1_q(i), Bit-2_q(i) and Bit-3_q(i).

Proposed Low-Power Scheme

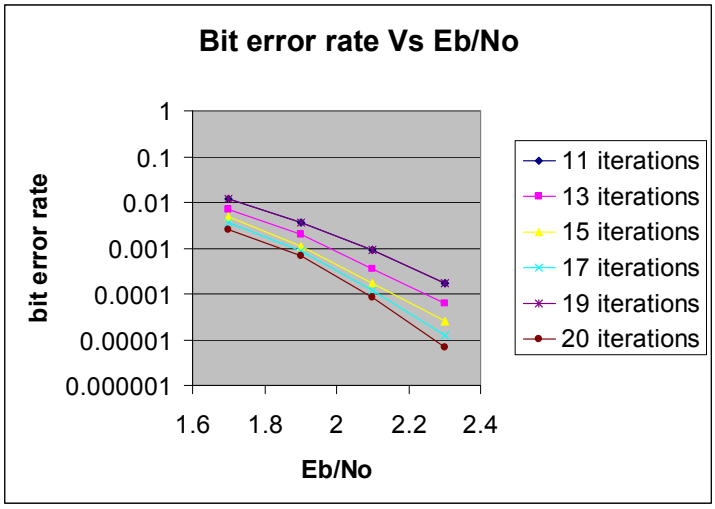

Figure19. BER plot for different decoding iterations

It is generally accepted that almost all packets become error free after LDPC decoder carries out about 20 soft iterations. Most packets require less number of soft iterations to achieve convergence as observed through simulations. But there are a few critical packets that require 20 iterations. If low number of iterations is used on these packets, the decoded packets contain more errors and the BER/FER performance will degrade significantly. Hence an adaptive decoding scheme will significantly reduce power consumption.

Figure 19 shows the packet profile. The bit error rate performance for different number of decoding iterations as a function of Eb/No is shown in figure 19. The bit error rate is plotted on the Y axis and the Eb/No value is plotted on the X axis. Different curves are shown corresponding to the different iterations used for decoding. There is a large

difference in coding gain between 11 and 13 soft iterations. The coding gain decreases between 13 and 15 iterations in terms of gain per unit increase in soft iteration. Beyond 19, there is no improvement and iterations saturate at 20. For low signal to noise ratios, the difference between different iteration numbers is small, but in high Eb/No values, the difference is higher than that of the low Eb/No range. This also makes it clear that high Eb/No values make a significant difference in decoding iteration stages.

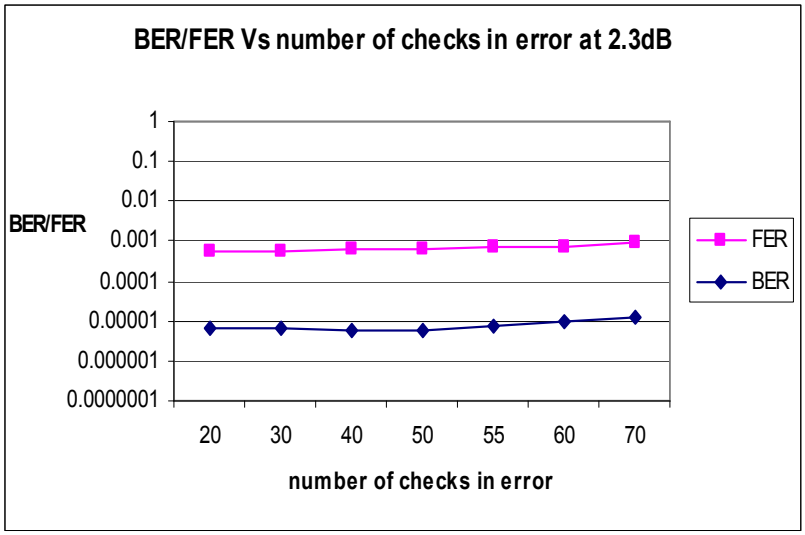

Figure 20. Determining threshold value for cH^T

It has been observed at high signal to noise ratios that a hard iteration (an iteration based on just 0s and 1s for the bit nodes and check nodes) corrects all errors if the total number of parity checks is below a threshold value (e.g. 50). At low SNRs, if the packets are passed through sufficient number of soft iterations, it is equivalent to the high SNR situation.

Figure.20 shows the BER/FER simulation results for a N = 2040, rate ½ code at a Eb/No of 2.3dB. After 13 soft iterations, hard iterations are performed based on syndrome check cH^T. The value of cH^T is plotted on the X axis which is the number of parity checks in error based on which a decision to perform hard iteration is taken. If cH^T value is above the X axis parameter value, then soft iterations are performed. If cH^T value is below the X axis parameter value, a hard iteration is performed. We make an interesting observation from the figure. From Figure.20, it can be observed that the BER/FER performance improves with number of parity checks in error until 50 and then degrades significantly above 50. The fact that the bit error rate performance improves until 50 parity checks in error are being used as the threshold implies that hard iteration works better than soft iterations when the number of checks in error is low. This could be attributed to the fact that hard iterations can correct all errors in a packet compared to soft iterations at these low error rates. Therefore, we conclude that hard iteration achieved high BER performance if the number of parity checks in error is less than 50. The number of hard iterations is again determined through simulations.

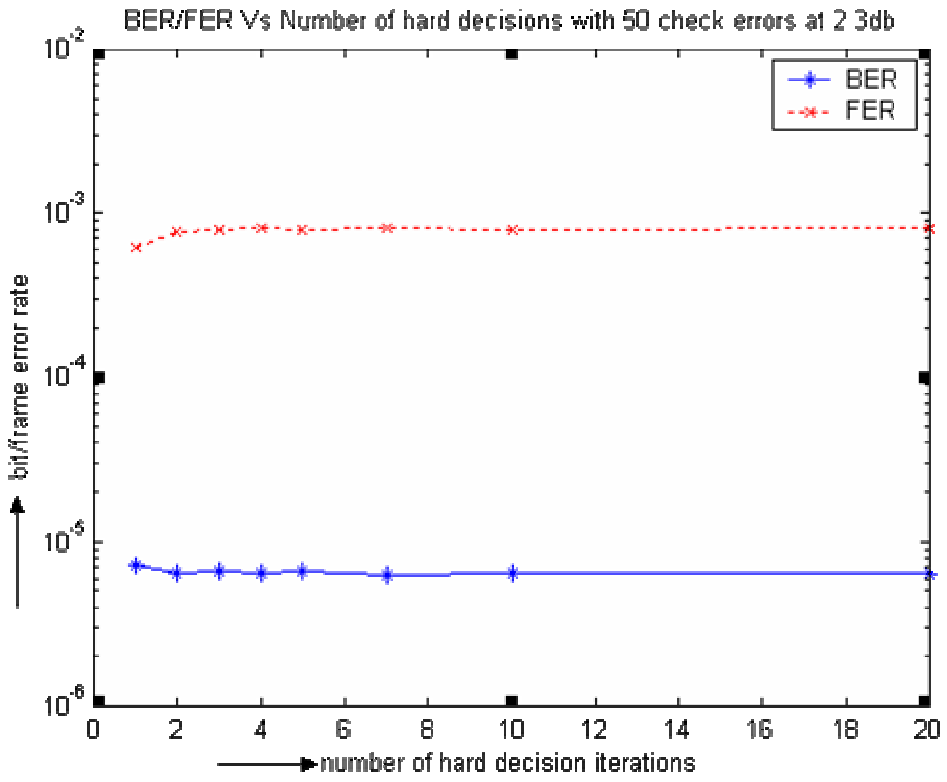

Figure 21. Determining number of hard iterations

Figure.21 shows the BER performance with different number of iterations for the same LDPC code as above at 2.3db and with a threshold value of 50 parity checks in error. The bit error rate performance is plotted along the Y axis and the number of hard iterations is plotted along the X axis. All other parameters of the simulation are fixed like the number of soft iterations before cH^T and the threshold value of cH^T that has already been determined. Zero hard iteration has not been plotted as that implies an average of 40 checks in error for each packet. The bit error rate is a very large value and beyond the scale shown. It can be seen that a single hard iteration achieves a good BER performance and saturates after that. One hard decision achieves the same BER performance as that of 10 hard iterations. Hard iterations saturate beyond a single iteration.

Therefore, a hard iteration stage after a fixed number of soft iterations achieves the same BER performance as that of 20 soft iterations for most packets that have threshold less than 50. It also results in significant power savings. Henceforth, we propose two low power schemes that are based on dynamic/static soft iterations and a hard iteration.

The two proposed schemes are:

A. The dynamic decoding scheme takes as input a data frame and 1) performs a fixed number of iterations, 2) determines if further processing is necessary, and finally 3) an adaptive decoding is carried out for remaining iterations.

B. Static scheme aims to simplify hardware complexity in aforementioned dynamic decoding by carrying out fixed number of soft iterations and hard-decision without dynamically configuring the decoding mode. This is because we observed experimentally that the BER/FER performance does not degrade if the number of soft iterations is higher than that of dynamic scheme (but significantly less than 20).

Dynamic Scheme

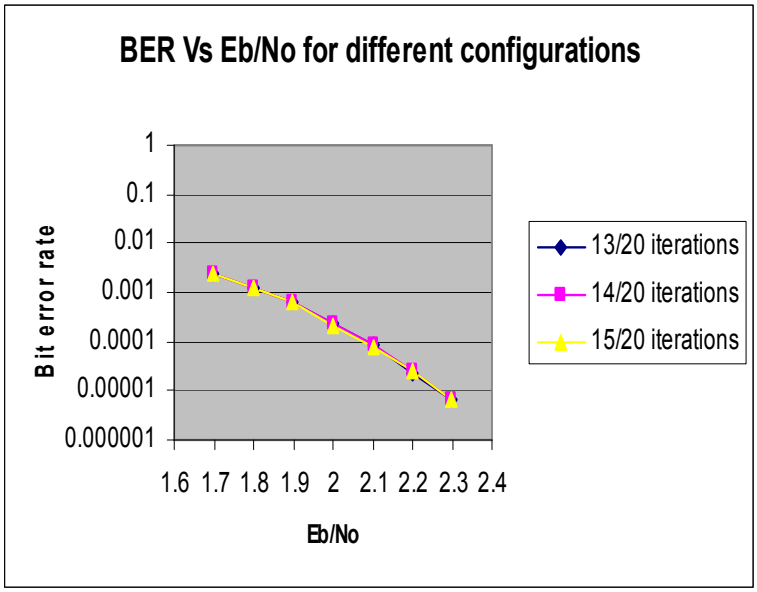

Figure 22. BER performance for different X

We conducted a set of extensive experiments and have observed that most packets require a fixed number of iterations, but very few packets require the complete 20 iterations. If the decoder configuration is dynamically changed for the decoded packets, a significant amount of power can be saved. LDPC code performance can also be unaltered. The power savings result from selectively shutting down soft iteration stages when not used, with unaltered BER performance as shown in figure 22.

Based on empirical observations, we propose a scheme wherein a fixed number of soft iterations X are performed on the packet and then the syndrome check criterion is used to determine further decoding. If syndrome-check $cH^T < 50$, a hard iteration is employed and the packet decoding is complete. If the number of parity check errors determined by the above criterion is above 50, 20-X number of soft iterations is used. Figure.23 shows

the block diagram of the dynamic scheme. The first few blocks represent the different soft iterations. X number of soft iterations are performed. The parameter X will be systematically determined through simulations. The cH^T block represents the parity checker circuit that takes as input the 2040 code bits hard values and checks if parity check constraints are satisfied. It provides 1020 outputs which correspond to the 1020 parity check constraints. If a parity check is satisfied, the output is 0, otherwise the output is 1. The outputs are summated to get the cumulative number of parity checks in error. There is a branch at this point. If the number of checks in error is greater than 50, a separate circuit receives the code word soft outputs and iterates starting from stage X+1 to 20 soft decision iterations and then decision logic takes hard decision on each decoded bit. If the number of parity checks in error is less than or equal to 50, a hard iteration is performed on the hard decision bit values and then passed to the decision logic.

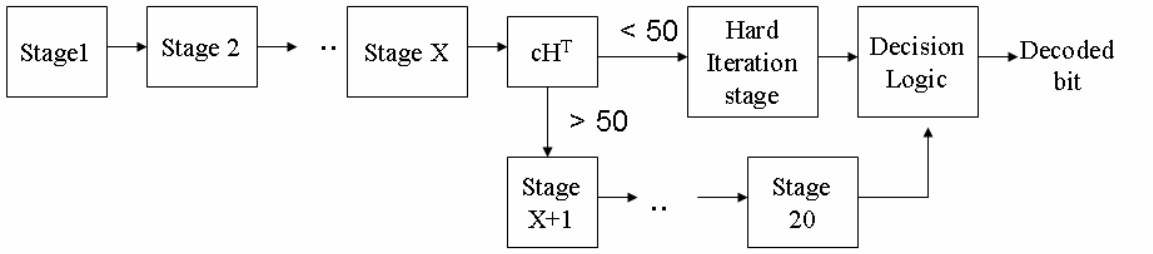

Figure 23. Dynamic decoding scheme

Figure.22 shows that independent of X, the BER performance does not degrade and is intuitive. The bit error rate performance is plotted on the Y axis and the Eb/No value is plotted along the X axis. The curves are plotted for the dynamic scheme for three different values of X. Irrespective of X, any packet that has threshold greater than 50

checks in error will always go through 20 iterations. But, decoder power is dependent on X.

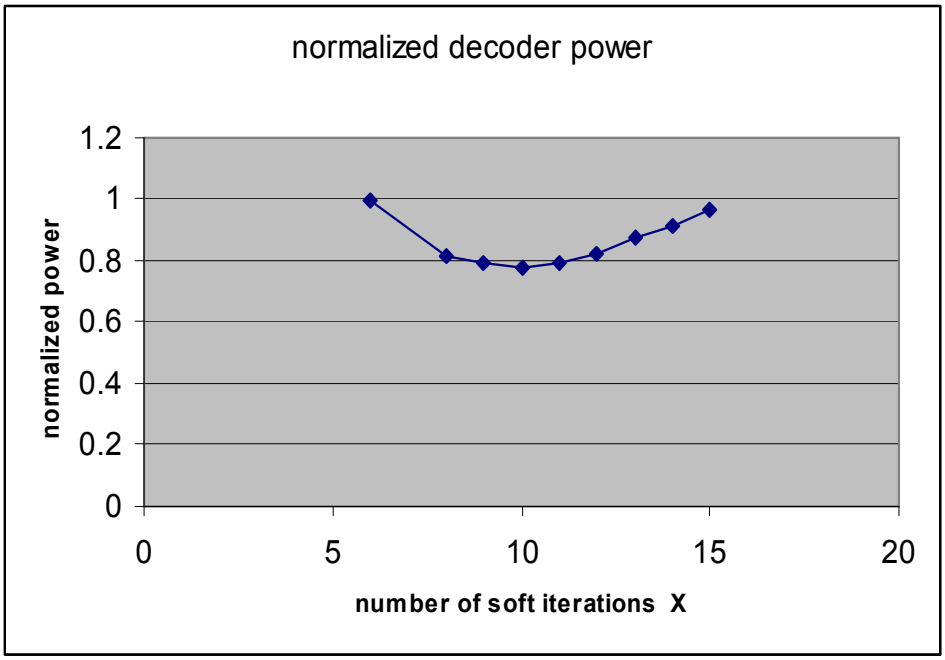

Figure 24. Decoder power for different soft iterations X

A high value of X implies all packets go through more iterations leading to wastage of power and a low value of X also imply that most packets go through parity checker and the remaining 20-X iterations. The optimal trade off point has to be determined for X.

The software version of the decoder is simulated for N=2040, rate ½ code for different Eb/No values and the graph is shown in figure 24 for the case of Eb/No = 1.7dB. Each value of X is associated with a different decoder power. We assume all iteration stages consume the same amount of power. Through simulations, we observed the number of

packets that go through the hard iteration stage and the packets that complete 20 iterations. The decoder power is composed of a fixed component and a variable component. The decoder power due to X soft iterations is always fixed. The decoder power due to a hard iteration is negligible. The decoder power due to 20-X iterations is variable as only a fraction of the total packets complete 20 soft iterations. For lower values of X, this fraction becomes huge and hence higher power. For higher values of X, this fraction is small but X is already large and the decoder power is large. The optimal decoder power and optimum X is determined through the graph. Power is normalized in the figure. The normalized decoder power is plotted along the Y axis and the value X (number of soft iterations before syndrome check) is plotted along the X axis. From figure 24, decoder power decreases with X as X increases from 5 to 10 and reaches the lowest value at X = 10 and then increases for higher values of X. Therefore, the optimum value of X for the dynamic scheme is 10 soft iterations.

Static Scheme

The Static scheme simplifies the decoder of the dynamic scheme by fixing the number of soft iterations before hard iteration. There is no additional soft iteration or syndrome check in this scheme. This is still a low power scheme benefiting from the hard iteration after Y soft iterations as shown in figure 25.

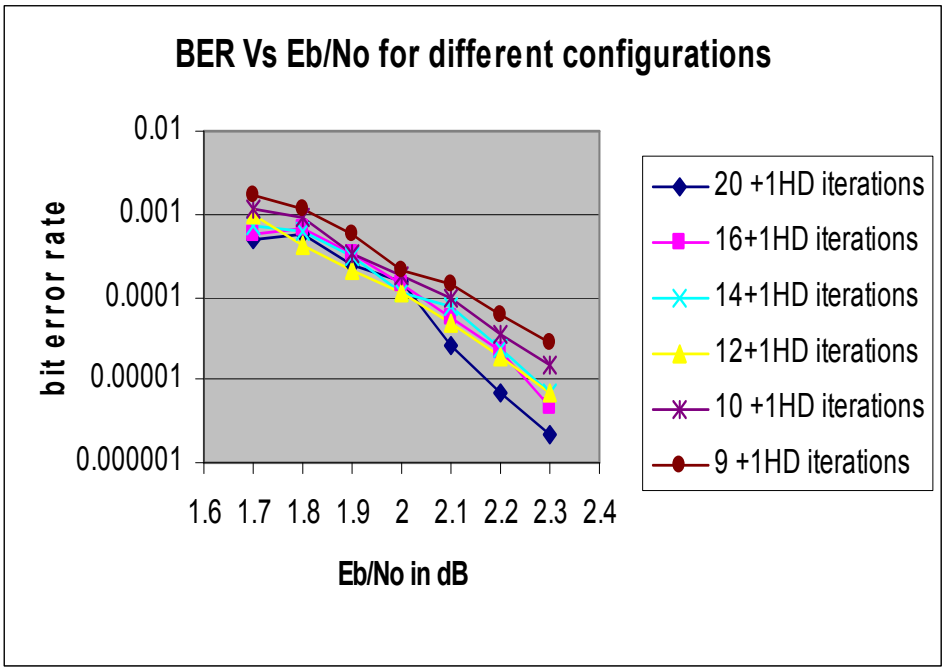

Figure 25. BER performance for different soft iterations Y

The block diagram of the static scheme is shown in figure 26. There are Y soft iteration stages. After completing Y soft iterations, packets pass through a hard iteration and followed by decision logic to determine the decoded bit. Thus in the Static scheme, packets pass through a fixed number Y of soft decision stages followed by a hard iteration stage. The parameter Y is higher than X to achieve the same BER performance.

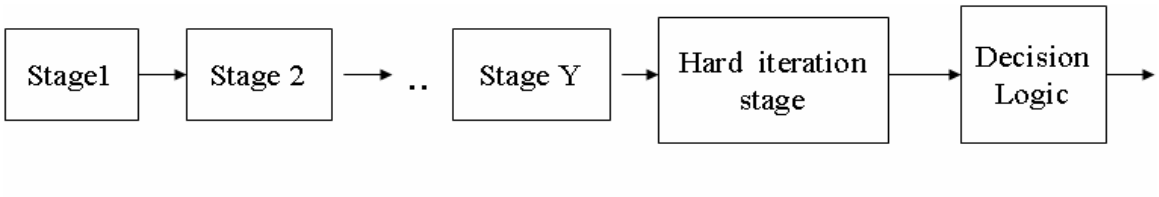

Figure 26. Fixed decoding scheme

Figure.25 shows the simulation results of the decoder at different Eb/No values for different values of Y. The X axis is Eb/No in dB. The Y axis is the bit error rate of the decoder for different values of Y. The graph is plotted for different values of Y ranging from 9 to 16 and compared with the original decoder with 20 soft iterations and a hard iteration. From Figure.25, 9 soft iterations with a hard iteration have a very poor performance. The coding gain at a high bit error rate of 10^{-4} is more than 0.2dB. The decoder with 10 soft iterations and a hard iteration stage is slightly better than 11 soft iterations. With 12 soft iterations, the bit error rate is close to that of the other decoders that have up to 16 soft iterations followed by a hard iteration. Therefore, the optimal value of Y is 12. Hence, the static scheme has 12 soft iterations and a hard iteration. This scheme has less gate complexity than the first scheme, but Y has to be larger than X to achieve the same BER/FER performance.

Dynamic and Static Scheme: Architecture

Once the decoding scheme is determined, there are a variety of architectures for an iteration of soft decision decoding. Serial, partly parallel and fully parallel architectures have been proposed in the past. In this paper, fully parallel architecture is used for soft decision decoding iteration stage. The fully parallel architecture has very high throughput and high hardware complexity. The fully parallel architecture is presented here for both schemes and is widely used [10], [24] to achieve high throughput.

The hard iteration first computes the parity checks that involve XOR gates. Then a bit node update is computed based on majority voting logic that uses a few "AND" and "OR" gates. The hard decision iterations involve 'XOR' and 'AND' gates and their number is proportional to the code length. The gate complexity is only a few thousand gates and is negligible compared to the complexity of soft decision iteration stage. The gate delay is small and hence a stage of hard decision is completed in one clock cycle with a fully parallel hard decision iteration stage.

Comparison of Dynamic and Static Scheme

In the dynamic scheme, a few packets pass through the complete 20 iterations and most of the packets require less than 11 iterations. The packets that go through fewer iteration stages will be decoded earlier than the earlier packets that go through all 20 iteration stages. The following table lists the scheduling for two consecutive packets that require 20 iterations.

Table VI. Scheduling packets in dynamic scheme

Clock cycle	Packet 1 processing	Packet 2 processing
1	1^{st} iteration	
2	2^{nd} iteration	
3	3^{rd} iteration	
4	4^{th} iteration	
5	5^{th} iteration	
6	6^{th} iteration	
7	7^{th} iteration	
8	8^{th} iteration	
9	9^{th} iteration	
10	10^{th} iteration	
11	cH^{T}	1^{st} iteration
12	11^{th} iteration	2^{nd} iteration
13	12^{th} iteration	3^{rd} iteration
14	13^{th} iteration	4^{th} iteration
15	14^{th} iteration	5^{th} iteration
16	15^{th} iteration	6^{th} iteration
17	16^{th} iteration	7^{th} iteration
18	17^{th} iteration	8^{th} iteration
19	18^{th} iteration	9^{th} iteration
20	19^{th} iteration	10^{th} iteration
21	20^{th} iteration	cH^{T}
22	Output	11^{th} soft iteration or Hard iteration

Only two instantiations of the parallel architecture iteration stage are used in the dynamic scheme. One instantiation is used before cH^{T} is computed and the other is used for those

packets that need 10 soft iterations after cH^T is computed. This configuration achieves higher throughput for the same power consumed.

Table VI clearly shows that there is no conflict in scheduling between any two packets irrespective of whether the packet is operated on by soft iterations or a hard iteration after cH^T. In the case of static scheme, one instantiation of the soft iteration stage is used. Both schemes use only one instantiation of the hard iteration whose gate complexity is approximately 6000 gates.

The gate count, decoder power per packet and throughput for the two low power schemes on the fully parallel decoder architecture is shown in the table below. A fully parallel decoder has a clock speed of 100 Mhz. The hardware complexity of one iteration stage is 2,693,450 gates and the iteration stage power consumption is 600mW. The power consumed by the hard iteration stage is negligible. It can be seen from table VII that the static scheme has a higher throughput and lower complexity than the dynamic scheme.

Table VII. Dynamic and static schemes optimized for throughput

Different schemes	# of gates	Decoder Power	Throughput
Static Scheme	2,699,450	600mW	15.38 Gbps
Dynamic Scheme	5,393,900	744mW	14.9 Gbps
Regular decoders	2,693,450	600mW	10 Gbps

The two schemes can be optimized to save power. The dynamic scheme uses only one instantiation of the soft iteration stage. The static scheme achieves the same throughput of regular decoders, but shuts down the iteration stage for 8 of the 20 clock cycles used for decoding.

Table VIII Dynamic and static schemes optimized for power

Different schemes	# of gates	Decoder Power	Throughput
Static Scheme	2,699,450	360mW	10 Gbps
Dynamic Scheme	2,700,450	372mW	9.52 Gbps
Regular decoders	2,693,450	600mW	10 Gbps

Table VIII shows the complexity, power and throughput of the two proposed schemes and compares it with regular decoders. The static scheme has the lowest decoder power closely followed by the dynamic scheme. In Dynamic scheme, 24 percent of the packets complete 20 soft iterations. This implies that the average number of soft iterations is 12.4 in the dynamic scheme which is slightly higher than 12 iterations used in the static scheme. Therefore, the static scheme has an edge over the dynamic scheme in throughput, gate complexity and decoder power.

CHAPTER VI

APPLICATIONS AND COMPETING CODES

Bit Error Rates

Each application has different requirements. The noise levels in deep space communication are completely different from mobile wireless communication or storage networks. Also, the acceptable levels of error in the transmitted content also vary with different applications. Figure 27 shows the BER requirements of a wide range of applications.

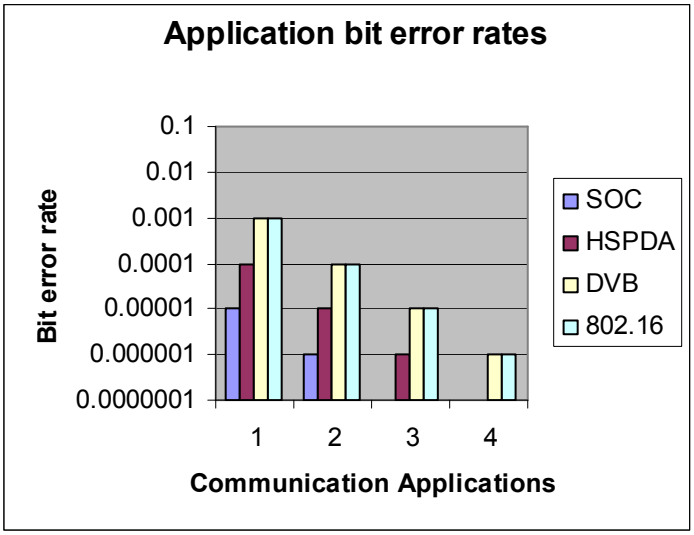

Figure 27. Bit error rates of different applications

System on Chip (SOC) has very low levels of error. Other applications account for a wide range of errors. The same trend is observed in frame error rates.

The main competitor to LDPC codes is Turbo codes with similar bit error and frame error rates. Figure 28 shows the error rate levels under which codes operate for similar block sizes for different error correcting and detecting codes. Cyclic Redundancy Check (CRC) codes operate only on low bit error rates, but in conjunction with LDPC or turbo codes as in Hybrid ARQ (HARQ) scheme can operate on higher error levels.

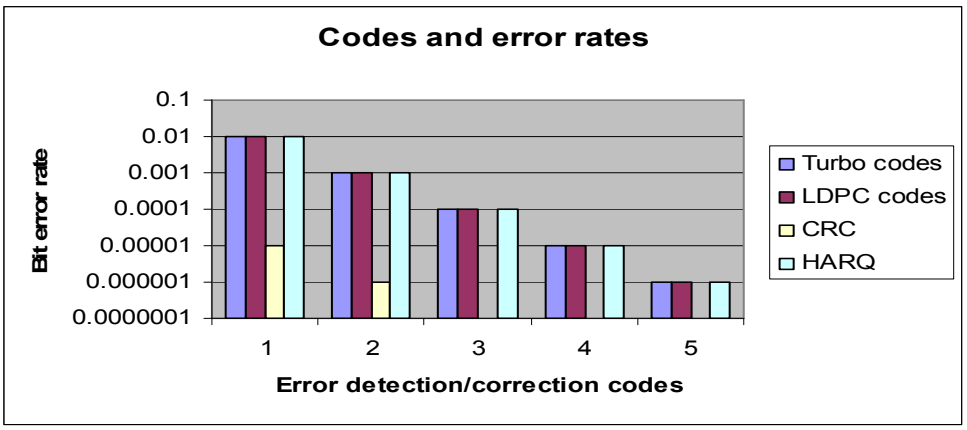

Figure 28. Bit error rates of different codes

Throughput Requirements

The Throughput requirements of different applications are also different. While Optical Communication applications operate in the range of Gbps, other wireless and satellite communication operate at a maximum of a few hundred Mbps. Figure 29 shows the data rates of different applications.

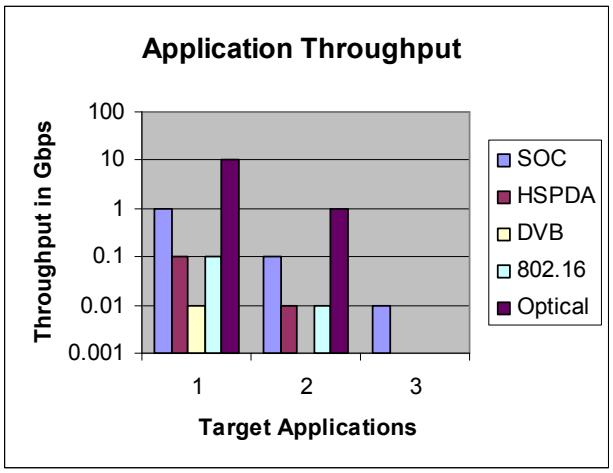

Figure 29. Throughput requirements of different applications

With varying degree of flexibility in implementing LDPC codes, LDPC codes support a wider throughput range than turbo codes as shown in figure 30.

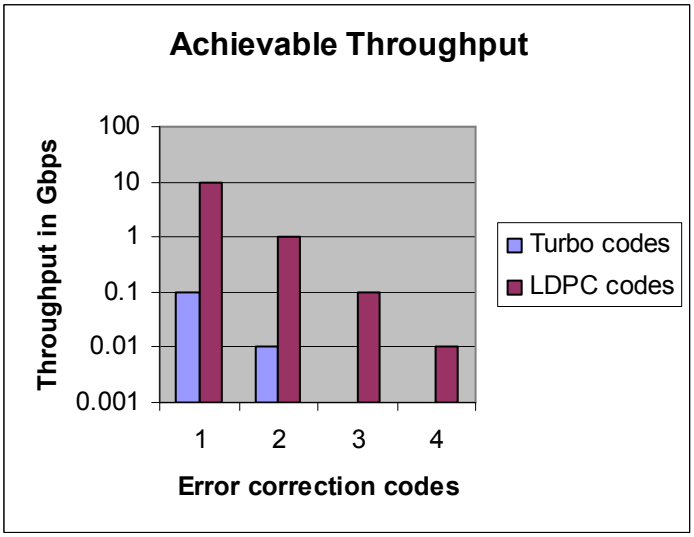

Figure 30. Data rates supported by LDPC and turbo decoders

CHAPTER VI

CONCLUSION

Summary

A high throughput scalable architecture, a tiling approach for an improved scalable design and two low power decoding schemes have been proposed in this research. The existing serial architecture has been scaled to form a scalable architecture by scaling the combinational logic, partitioning the memory used to store the parity checks and constructing a novel H matrix that ensures that the parallely processed bit nodes access different memory partitions. The scalable architecture achieves a high throughput for higher values of the parallelization factor M and the value of M can be carefully chosen to suit available hardware resources. The switch logic is an important constituent of the scalable architecture and whose complexity becomes significant for higher values of M. The tiling approach proposed on top of the scalable architecture simplifies the switch logic.

The tiling approach generates tiles or patterns of size M by M and constructs the H matrix by repeating a fixed number of patterns. For example, for a parallelization factor of N/6, 5 patterns of size N/6 by N/6 are repeated to fill the 18 blocks in the H matrix. The advantages of the proposed approach are two-fold. First, the information stored about the H matrix is reduced by $1/3^{rd}$. Second, the switch logic of the scalable architecture is simplified as each pattern is a switch. There are fewer switches to control in this approach. The routing is greatly simplified. The H matrix information is also embedded in the switch and no external memory is required to store the H matrix. The tiling

approach is a very general approach and can also be applied to other architectures to achieve high throughput through simplified routing.

Scalable architecture and tiling approach are proposed at the architectural level of the LDPC decoder. We propose two low power decoding schemes that take advantage of the Gaussian and random distribution of errors in the received packets. Both schemes explore the use of a hard iteration after a fixed number of soft iterations. The dynamic scheme performs X soft iterations, followed by a parity checker cH^T that computes the number of parity checks in error. If the number of checks in error is below a threshold value, a hard iteration is performed and is able to correct all the errors. If the number of checks in error is above the threshold value, 20-X soft iterations are performed. This is because it has been shown that 20 iterations achieve a low bit error rate for any packet. The advantage of the hard iteration is so significant that the second scheme performs a fixed number of iterations Y followed by a hard iteration. To compensate the bit error rate performance, the parameter Y is slightly higher than X. These two low power schemes match the bit error rate performance of the best known decoders at the given code length and rate.

Future Work

Applications of scalable architecture will be the focus of future research. Partial parallel processing of bits can be applied to irregular code processing using simple decoders. We will also explore application of scalable architecture to large LDPC codes and other novel codes such as repeat accumulate codes.

Tiling approach can also be applied to different architectures. The advantages gained by applying tiling to other LDPC architectures will be explored. In a more general sense, tiling implies a block code constructed from several smaller block codes of uniform sizes. This principle can be used to decode large length codes. We will also explore tiling approach with irregular block sizes.

The hard iteration stage has a significant contribution in low power decoders. The hard iteration stage has low gate count and lower gate delay. We have presented hard iteration stages as a substitute for soft iteration stages after the packet has already been processed by several soft iterations. The presence of a threshold below which hard iterations correct all errors is a significant finding and can be used to develop a novel code that can work with soft iterations. It also implies hard iterations can be used for applications where it is proven that the channel bit error is below the threshold.

REFERENCES

1. C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon Limit Error Correcting Coding and Decoding: Turbo-Codes", *Proc. International Communications Conference*, pp. 1064-1070, 1993.
2. D.J.C. Mackay and R.M. Neal, "Near Shannon Limit Performance of Low Density Parity Check Codes", *Electron. Letters*, vol.32, no. 18, pp. 1645-1646, 1996.
3. R.G. Gallager, "Low-Density Parity-Check Codes", *IRE Transactions on Information Theory*, vol. IT-8, pp.21-28, 1962.
4. S. Sivakumar, "VLSI Implementation of Encoder and Decoder for Low-Density Parity-check codes", M.S. Thesis, Texas A & M University, 2001.
5. S-C. Chae, and Y-O. Park, "Low Complexity Encoding of Regular Low Density Parity Check Codes", *Proc. Vehicular Technology Conference*, vol. 3, pp.1822 – 1826, 2003.
6. D.-U. Lee, W. Luk, C. Wang, and C. Jones, "A Flexible Hardware Encoder for Low-Density Parity-Check Codes", *IEEE Symp. on Field-Programmable Custom Computing Machines*, pp.101 – 111, 2004.
7. D. Haley, A. Grant, and J. Buetefeur, "Iterative Encoding of Low-Density Parity-Check Codes", *Proc. Global Telecommunications Conference*, vol. 2, pp. 1289 – 1293, 2002.
8. E. Yeo, P. Pakzad, B. Nikolic, and V. Anantharam, "High Throughput Low-Density Parity-Check Decoder Architectures", *Proc. IEEE Global Telecommunications Conference*, vol. 5, pp. 3019-3024, 2001.

9. E. Yeo, P. Pakzad, B. Nikolic, and V. Anantharam, "VLSI Architectures for Iterative Decoders in Magnetic Recording Channels", *IEEE Transactions on Magnetics*, vol. 37, no. 2, pp. 748 – 755, 2001.

10. E. Yeo, B. Nikolic, and V. Anantharam, "Architectures and Implementations of Low-Density Parity Check Decoding Algorithms", *Proc. 45th Midwest Symposium on Circuits and Systems*, vol. 3, pp: III-437 - III-440, 2002.

11. Y. Liao, E. Yeo, and B. Nikolic, "Low-Density Parity-Check Code Constructions for Hardware Implementation", *Proc. IEEE International Conference on Communications*, vol. 5, pp. 2573 -2577, 2004.

12. Y. Pei, L. Yin, and J. Lu, "Design of Irregular LDPC Codec on a Single Chip FPGA", *Proc. Circuits and Systems Symposium on Emerging Technologies: Frontiers of Mobile and wireless communication*, vol. 1, pp. 221-224, 2004.

13. M. Mohiyuddin, A. Prakash, A. Aziz, and W. Wolf, "Synthesizing Interconnect Efficient Low Density Parity Check Codes", *Proc. Design Automation Conference*, pp. 488-491, 2004.

14. W.L.Lee, and A. Wu, "VLSI Implementation for Low Density Parity Check Decoder", *Proc. Conference on Electronics, Circuits and Systems*, vol.3, pp: 1223 –1226, 2001.

15. Y. C. He, S. H. Sun, and X. M. Wang, "Fast Decoding of LDPC Codes Using Quantization", *Electronics Letters*, vol.38, no. 4, pp. 189 – 190, 2002.

16. T. Zhang et al, "On Finite Precision Implementation of Low Density Parity Check Codes Decoder", *Proc. IEEE International Symposium on Circuits and Systems*, vol. 4, pp. 202 -205, 2001.

17. D. Changyan, D. Proietti, I.E. Telatar, T.J. Richardson, and R.L. Urbanke, "Finite-Length Analysis of Low-Density Parity-Check Codes on the Binary Erasure Channel", *IEEE Transactions on Information Theory*, vol. 48, no. 6 , pp. 1570 – 1579, 2002.

18. B. Levine et al, "Implementation of Near Shannon Limit Error-Correcting Codes Using Reconfigurable Hardware", *IEEE Symp. on Field-Programmable Custom Computing Machines*, pp. 217-226, 2000.

19. M. Karkooti and J. Cavallaro, "Semi-parallel Reconfigurable Architectures for Real-time LDPC Decoding", *Proc. IEEE International Conference on Information Technology (ITCC)*, vol. 1, pp. 579-585, 2004.

20. J.K.-S. Lee, B. Lee, J. Thorpe, K. Andrews, S. Dolinar, and J. Hamkins, "A Scalable Architecture of a Structured LDPC Decoder", *Proc. International Symp. on Information Theory*, pp. 292, 2004.

21. T. Theocharides, G. Link, E. Swankoski, N. Vijaykrishnan, M.J. Irwin, and H. Schmit, "Evaluating Alternative Implementations for LDPC Decoder Check Node Function", *Proc. IEEE Computer society Annual Symp. on VLSI*, pp. 77 – 82, 2004.

22. C. Howland, and A. Blanksby, "Parallel Decoding Architectures for Low Density Parity Check Codes", *IEEE International Symp. on Circuits and Systems*, vol. 4, pp. 742 -745, 2001.

23. M. M. Mansour, and N. R. Shanbag, "Memory-Efficient Turbo Decoder Architectures for LDPC Codes", *Proc. IEEE Workshop on Signal Processing Systems*, pp. 159 - 164, 2002.

24. M. M. Mansour, and N.R. Shanbag, "Turbo Decoder Architectures for Low-Density Parity-Check Codes", *Proc. Global Telecommunications Conference*, vol. 2, pp. 1383 – 1388, 2002.

25. M. M. Mansour, and N.R. Shanbag, "Design Methodology for High-Speed Iterative Decoder Architectures", *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 3, pp. III-3085 - III-3088, 2002.

26. M. M. Mansour, and N. R. Shanbag, "Low-Power VLSI Decoder Architectures for LDPC Codes", *Proc. International Symp. on Low Power Electronics and Design*, pp. 284-289, 2002.

27. M. M. Mansour, and N. R. Shanbag, "High Throughput LDPC Decoders", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 6, pp. 976 – 996, 2003.

28. M. M. Mansour, "High Performance Decoders for Regular and Irregular Repeat-Accumulate Codes", *Proc. Global Telecommunications Conference*, vol. 4, pp. 2583 – 2588, 2004.

29. M. M. Mansour, and N. R. Shanbag, "A Novel Design Methodology for High-Performance Programmable Decoder Cores for AA-LDPC Codes", *Proc. IEEE Workshop on Signal Processing Systems*, pp. 29 - 34, 2003.

30. M. M. Mansour, "Unified Decoder Architectures for Repeat-Accumulate and LDPC Codes", *Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers*, vol. 1, pp. 527 – 531, 2004.

31. S. Kim, G. E. Sobelman, and J. Moon, "Parallel VLSI Architectures for a Class of LDPC Codes", *Proc. IEEE International Symp. on Circuits and Systems*, vol. 2, pp. 93 –96, 2002.
32. G. Al-Rawi, J. Cioffi, and M. Horowitz, "Optimizing the Mapping of Low-Density Parity Check Codes on Parallel Decoding Architectures", *Proc. International Conference on Information Technology, Coding and Computing*, pp. 578 –586, 2001.
33. G. Al-Rawi, J. Cioffi, R. Motwani, and M. Horowitz, "Optimizing the Mapping of Low-Density Parity Check Codes on Parallel Decoding Architectures", *Proc. IEEE Global Telecommunications Conference*, vol.5, pp. 3012- 3018, 2001.
34. A.J. Blanksby, and C.J. Howland, "A 690-mW 1-Gb/s 1024-b, rate-1/2 Low-Density Parity-Check Code Decoder", *IEEE Journal of Solid State Circuits*, vol. 37, no. 3, pp. 404 –412, 2002.
35. C. Howland, and A. Blanksby, "A 220-mW 1-Gb/s 1024-b, Rate-1/2 Low-Density Parity-Check Code Decoder", *IEEE Conference on Custom Integrated Circuits*, pp. 293 – 296, 2001.
36. S. Kang, and I. Park, "Memory-Based Low Density Parity Check Code Decoder Architecture Using Loosely Coupled Two Data-Flows", *Proc. International Symp. on Circuits and Systems*, vol. 2, Pages: II - 397-400, 2004.
37. D.E. Hocevar, "LDPC Code Construction with Flexible Hardware Implementation", *Proc. IEEE International Conference on Communications*, vol. 4, pp. 2708 -2713, 2003.

38. G. Al-Rawi, and J. Cioffi, "A Highly Efficient Domain-Programmable Parallel Architecture for Iterative LDPC Decoding**",** *Proc. International Conference on Information Technology: Coding and Computing*, pp. 569 –577, 2001.

39. G. Murphy, E.M. Popovici, R. Bresnan, W.P. Marnane, and P. Fitzpatrick, "Design and Implementation of a Parameterizable LDPC Decoder IP Core", *Proc. of 24th International Conference on Microelectronics*, vol. 2 , pp. 747 – 750, 2004.

40. Y. Chen, and D.E. Hocevar, "A FPGA and ASIC Implementation of Rate ½ 8088 b Irregular Low-Density Parity-Check Decoder", *Proc. Global Communications Conference*, vol. 1, pp. 113-117, 2003.

41. D.E. Hocevar, "A Reduced Complexity Decoder Architecture Via Layered Decoding of LDPC Codes", *IEEE Workshop on Signal Processing Systems*, pp. 107 - 112, 2004.

42. H. Zhang, and T. Zhang, "Joint Code-Encoder-Decoder Design for LDPC Coding System VLSI Implementation", *Proc. International Symp. on Circuits and Systems*, vol. 2, pp: II - 389-92, 2004.

43. H. Zhang, and T. Zhang, "Design of VLSI Implementation-Oriented LDPC Codes", *Proc. IEEE 58th Vehicular Technology Conference*, vol. 1, pp. 670 – 673, 2003.

44. L. Sun, and B.V.K. Vijaya Kumar, "Field Programmable Gate Array Implementation of a Generalized Decoder for Structured Low-Density Parity Check Codes", *Proc. IEEE International Conference on Field-Programmable Technology*, pp. 17 – 24, 2004.

45. V. Nagarajan, N. Jayakumar, S. Khatri, and G. Milenkovic, "High-Throughput VLSI Implementations of Iterative Decoders and Related Code Construction Problems", *Proc. IEEE Global Telecommunications Conference*, vol. 1, pp: 361 – 365, 2004.

46. H. Zhang, and T. Zhang, "Block-LDPC: A Practical LDPC Coding System Design Approach", *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 52, no. 4, pp. 766 – 775, 2005.

47. Y. Mao, and A. H. Banihashemi, "A New Schedule for Decoding Low-Density Parity-Check Codes", *Proc. Global Telecommunications Conference*, vol. 2, pp. 1007 – 1010, 2001.

48. A. Selvarathinam, G. Choi, K. Narayanan, A. Prabhakar, and E. Kim, "A Massively Scaleable Decoder Architecture for Decoding Low-Density Parity-Check Codes", *Proc. IEEE International Symp. on Circuits and Systems*, vol. 2, pp. II-61 – II-64, 2003.

49. A. Selvarathinam, E. Kim, and G. Choi, "Low-Density Parity-Check Decoder Architecture for High Throughput Optical Fiber Channels", *Proc. 21st International Conference on Computer Design*, pp. 520 – 525, 2003.

50. J. Chen and M. P. C. Fossorier, "Decoding Low-Density Parity Check Codes with Normalized APP-based Algorithm", *Proc. IEEE Global telecommunications Conference*, vol.2, pp.1026 – 1030, 2001.

51. Y. Chen, and K.K. Parhi, "Overlapped Message Passing for Quasi-Cyclic Low-Density Parity Check Codes", *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 51, no. 6, pp. 1106 – 1113, 2004.

52. M. Fossorier, M. Mihaljevic, and H. Imai, “Reduced Complexity Iterative Decoding of Low Density Parity Check Codes Based on Belief Propagation”, *IEEE transactions on Communications*, vol.47, pp. 673-680, 1999.

53. R. Singhal, G. Choi, P. Koteeswaran, and N. Mickler, “Scalable Check Node Centric Architecture for LDPC Decoder”, *Proc. IEEE International Symp. on Circuits and Systems*, vol.4, pp. IV-189- IV-192, 2004.

54. T. Theocharides, G. Link, N. Vijaykrishnan, and M.J. Irwin, “Implementing LDPC Decoding on Network-on-Chip”, *Proc. 18th International Conference on VLSI Design*, pp. 134 – 137, 2005

55. G. Lechner, J. Sayir, and M. Rupp, “Efficient DSP Implementation of an LDPC Decoder”, *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 4, pp. iv-665 - iv-668, 2004.

56. F. Kienle, and N. Wehn, ”Joint Graph-Decoder Design of IRA Codes on Scalable Architectures [LDPC codes]” *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 4, pp. iv-673 - iv-676, 2004.

57. T. Zhang, and K.K. Parhi, “A 54 Mbps (3,6)-Regular FPGA LDPC Decoder”, *IEEE Workshop on Signal Processing Systems*, pp. 127- 132, 2002.

58. L. Yanping, M.H. Lee, G.Y. Hwang, and J.Y. Park, “New Implementation for the Scalable LDPC-Decoders”, *Proc. Vehicular Technology Conference*, vol. 1, pp. 343 – 346, 2004.

59. F. Kienle, T. Brack, and N. Wehn, “A Synthesizable IP Core for DVB-S2 LDPC Code Decoding”, *Proc. of Design, Automation and Test in Europe*, pp. 100-105, 2005.

60. Y. Li, M. Elassal, and M. Bayoumi, "Power Efficient Architecture for (3,6)-Regular Low-Density Parity-Check Code Decoder", *Proc. International Symp. on Circuits and Systems*, vol. 4, pp: IV - 81-4, 2004.

61. K. Gunnam, G. Choi, and M. Yeary, "An LDPC Decoding Schedule for Memory Access Reduction", *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 5, pp: V - 173-6, 2004.

62. Flarion Technologies, Inc, "Vector-Low Density Parity-Check Coding Solution Data Sheet", www.flarion.com/products/overviews/Vector-LDPC_Product_ Overview.pdf, 2001.

63. Y. Zhang, Z. Wang, and K.K. Parhi, "Efficient High-Speed Quasi-Cyclic LDPC Decoder Architecture", *Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers*, vol. 1, pp. 540 – 544, 2004.

64. T. Zhang, and K.K. Parhi, "Joint Code and Decoder Design for Implementation-Oriented (3,k)-Regular LDPC Codes", *Proc. Thirty-Fifth Asilomar Conference on Signals, Systems and Computers*, vol. 2, pp. 1232 –1236, 2001.

65. X. Y. Hu, E. Eleftheriou, D. M. Arnold, and A. Dholakia, "Efficient Implementations of the Sum-Product Algorithm for Decoding LDPC Codes", *Proc. Global Telecommunications Conference*, vol. 2, pp. 1036 -1036E, 2001.

66. H. Sankar, and K.R. Narayanan, "Memory-Efficient Sum-Product Decoding of LDPC Codes", *IEEE Transactions on Communications*, vol. 52, no. 8, pp. 1225 – 1230, 2004.

67. S.Y. Chung, T.J. Richardson, and R.L. Urbanke, "Analysis of Sum-Product Decoding of Low-Density Parity-Check Codes Using a Gaussian

Approximation", *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 657 – 670, 2001.

68. T. Zhang, and K.K. Parhi, "VLSI Implementation-Oriented (3, k)-Regular Low-Density Parity-Check Codes", *Proc. IEEE Workshop on Signal Processing Systems*, pp. 25-36, 2001.
69. T. J. Richardson, and R. L. Urbanke, "The Capacity of Low-density Parity-Check Codes Under Message- Passing Decoding", *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 599 – 618, 2001.
70. M. Yang, William E. Ryan, and Y. Li, "Design of Efficiently-Encodable Moderate-Length High-Rate Irregular LDPC Codes", *IEEE Transactions on Communications*, vol. 52, pp. 564-571, 2004.
71. J.L. Fan, "Array Codes as Low-Density Parity-Check Codes", *Proc. 2nd International Symp. on Turbo Codes and Related Topics*, pp. 543-546, 2000.
72. S. Olcer, "Decoder Architecture for Array-Code-Based LDPC Codes", *Proc. IEEE Global Telecommunications Conference*, vol.4, pp. 2046 – 2050, 2003.
73. B. Vasic, "High-Rate Low-Density Parity Check Codes Based on Anti-Pasch Affine Geometries", *Proc. IEEE International Conference on Communications*, vol. 3, pp. 1332 -1336, 2002.

VITA

Anand Manivannan Selvarathinam received his B.E. degree in electronics and communication engineering from Anna University, Chennai, India in 2000. He received his M.S. degree in computer engineering from Texas A&M University in 2001. He received his doctoral degree in August 2005 from Texas A&M University.

He did an internship at Texas Instruments, Houston from September to December 2001. His research interests are in the field of VLSI design, verification, testing and VLSI architectures for LDPC decoders. He can be reached at B-4/198, BHEL Township, Tiruchirappalli 620014, Tamil Nadu, India. His email address is anandmanivannan@yahoo.com