A Parallel Turbo Decoding Message Passing Architecture for Array LDPC Codes

Kiran Gunnam, Pankaj Bhagawat, Weihuang Wang, Gwan Choi, Mark Yeary[*]
Dept. of Electrical Engineering, Texas A&M University, College Station, TX-77840
[*] Dept. of Electrical and Computer Engineering, University of Oklahoma, Norman, OK-73109

Abstract

The VLSI implementation complexity of a low density parity check (LDPC) decoder is largely influenced by interconnect and the storage requirements. Here, the proposed layout-aware layered decoder architecture utilizes the data–reuse properties of min-sum, layered decoding and structured properties of array LDPC codes. This results in a significant reduction of logic and interconnects requirements of the decoder when compared to the state-of-the-art LDPC decoders. The ASIC implementation of the proposed fully parallel architecture achieves throughput of 4.6 Gbps (for a maximum of 15 iterations). The chip size is 2.3 mm x 2.3 mm with a gate count of 787 K in 0.13 micron technology.

1. Introduction

Low-Density Parity Check (LDPC) codes and turbo codes are among the best known codes that operate near Shannon limit [1]. When compared to the decoding of turbo codes, LDPC decoders require simpler computational processing and they are more suitable for parallelization, low implementation complexity, and low latency. While initial LDPC decoder designs suffered from complex interconnect issues, structured LDPC codes simplify the interconnect complexity. LDPC codes are considered for error correction coding in next generation digital video broadcasting (DVB-S2), MIMO-WLAN 802.11n,, 802.12, 802.20, Gigabit Ethernet 802.3, magnetic channels(storage/recording systems),and long-haul optical communication systems .

A binary (N,K) LDPC code is a linear block code of codeword length N and information block length K that can be described by a sparse $M \times N$ parity check matrix where M denotes the number of parity check equations. LDPC codes can be decoded by the Gallager's iterative two-phase message passing algorithm (TPMP) which involves check-node update and variable-node update as two phase schedule. Various algorithms are available for check-node updates and widely used algorithms are sum of products (SP), min-sum (MS) and Jacobian based BCJR (named after its discoverers Bahl, Cocke, Jelinik and Raviv). Mansour and Shanbhag [2] introduced the concept of turbo decoding message passing (TDMP), which is sometimes also called as layered decoding, using BCJR for their architecture-aware LDPC (AA-LDPC) codes. TDMP offers 2x throughput and significant memory advantages when compared to TPMP. This is later studied and applied for different LDPC codes using sum of products algorithm and its variations in [3]. TDMP is able to reduce the number of iterations required by up to 50% without performance degradation when compared to the standard message passing algorithm. A quantitative performance comparison for different check updates was given by Chen and Fossorier et al [4]. Their research showed that the performance loss of offset based min-sum with 5-bit quantization is less than 0.1dB in SNR as compared with that of floating point SP and BCJR. While fully-parallel LDPC decoder designs [5] suffered from complex interconnect issues, various semi-parallel and parallel implementations based on structured LDPC codes [2, 6-9] alleviate the interconnect complexity.

In this paper, we propose a novel parallel micro-architecture structure for check-node message processing unit (CNU) for the min-sum decoding of LDPC codes. This new micro-architecture structure employs the minimum number of comparator units by exploiting the concept of survivors in the search, which results in reduced number of comparisons, addition operations and additional reduction of the memory requirement. TDMP is applied for the offset MS for array LDPC codes. The resulting decoder architecture has significantly lower requirements of logic and interconnects when compared to the published decoder implementations. The rest of the paper is organized as follows. Section 2 introduces the background of array LDPC codes and min-sum the decoding algorithm. Section 3 presents the value-reuse property and proposed micro-architecture structure of CNU. The data flow graph and parallel architecture for TDMP using offset MS is included in Section 4. Section 5 shows the ASIC implementation results and performance comparison with related work and Section 6 concludes the paper.

2. Background

2.1 Array LDPC Codes

A brief review of the array codes is provided to facilitate the TDMP and the decoder architecture. The array LDPC parity-check matrix is specified by three parameters: a prime number p and two integers k (check node degree) and j (variable node degree) such that $j,k < p$. It is given by

$$H_A = \begin{bmatrix} I & I & I & \cdots & I \\ I & \alpha & \alpha^2 & \dots & \alpha^{k-1} \\ I & \alpha^2 & \alpha^4 & \dots & \alpha^{2(k-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ I & \alpha^{j-1} & \alpha^{(j-1)2} & \cdots & \alpha^{(j-1)(k-1)} \end{bmatrix} \quad (1)$$

where I is the $p \times p$ identity matrix , and α is a $p \times p$ permutation matrix representing a single left or right cyclic shift of I . Power of α in H denotes multiple cyclic shifts, with the number of shifts given by the value of the exponent. In the following discussion, we use the α as a $p \times p$ permutation matrix representing a single left cyclic shift of I .

2.2 Min-sum decoding of LDPC

Assume binary phase shift keying (BPSK) modulation (a 1 is mapped to -1 and a 0 is mapped to -1) over an additive white Gaussian noise (AWGN) channel. The received values y_n are Gaussian with mean $x_n = \pm 1$ and variance σ^2 .

The reliability messages used in belief propagation (BP) based min-sum algorithm can be computed in two phases, viz. (1) check node processing and (2) variable node processing. The two operations are repeated iteratively until the decoding criterion is satisfied. This is also referred to as standard message passing or two-phase message passing (TPMP). For the i^{th} iteration, $Q_{nm}^{(i)}$ is the message from variable node n to check node m , $R_{mn}^{(i)}$ is the message from check node m to bit node n , $\mathrm{M}(n)$ is the set of the neighboring check nodes for variable node n , $\mathrm{N}(m)$ is the set of the neighboring variable nodes for check node m . The message passing for TPMP are calculated in the following steps as in [4]:

(1) check-node processing: for each m and $n \in \mathrm{N}(m)$,

$$R_{mn}^{(i)} = \delta_{mn}^{(i)} \kappa_{mn}^{(i)} \quad (2)$$

$$\kappa_{mn}^{(i)} = \left| R_{mn}^{(i)} \right| = \min_{n' \in \mathrm{N}(m) \backslash n} \left| Q_{n'm}^{(i-1)} \right|. \quad (3)$$

The sign of check-node message $R_{mn}^{(i)}$ is defined as

$$\delta_{mn}^{(i)} = \left(\prod_{n' \in \mathrm{N}(m) \backslash n} \operatorname{sgn}\left(Q_{n'm}^{(i-1)} \right) \right) \quad (4)$$

where $\delta_{mn}^{(i)}$ takes value of $+1$ or -1

(2) Variable-node processing: for each n and $m \in \mathrm{N}(n)$,

$$Q_{nm}^{(i)} = L_n^{(0)} + \sum_{m' \in \mathrm{M}(m) \backslash m} R_{m'n}^{(i)} \quad (5)$$

where the log-likelihood ratio of bit n is $L_n^{(0)} = y_n$.

(3) Decision

$$P_n = L_n^{(i)} + \sum_{m \in M(n)} R_{mn}^{(i)} \quad (6)$$

A hard decision is taken where $\hat{x}_n = 0$ if $P_n(x_n) \geq 0$, and $\hat{x}_n = 1$ if $P_n(x_n) < 0$. If $\hat{x}_n H^T = 0$, the decoding process is finished with $\hat{x}_n$ as the decoder output; otherwise, go to step (1). If the decoding process doesn't end within some maximum iteration $it_{\max}$, stop and output an error messenger.

The reliability messages $R_{mn}^{(i)}$ computed in min-sum algorithm are overestimated. The overestimation is corrected in offset min-sum algorithm by subtracting a positive constant β from the magnitude of the $R_{mn}^{(i)}$ in the following way:

$$R_{mn}^{(i)} = \delta_{mn}^{(i)} \max\left(\kappa_{mn}^{(i)} - \beta, 0 \right) \quad (7)$$

For (3, 6) rate 0.5 code, β is computed as 0.15 using the density evolution technique [4].

2.3 Parallel TDMP for array LDPC

In TDMP, the array LDPC with j block rows can be viewed as concatenation of j layers or constituent sub-codes similar to observations made for AA-LDPC codes in [2]. After the check-node processing is finished for one block row, the messages are immediately used to update the variable nodes (2), whose results are then provided for processing the next block row of check nodes(1). This differs from TPMP where all check nodes are processed first and then the variable node messages will be computed. Each decoding iteration in TDMP has j number of sub-iterations. In the beginning of the decoding process, variable messages are initialized as channel values and they are used to process the check nodes of the first block row. After completion of that block row, variable messages are updated with the new check node messages. This concludes the first sub-iteration. In similar fashion, result of check-node processing of the second block row is immediately used in the same iteration to update the variable node messages for third block row. The completion of check-node processing and associated variable-node processing of all block rows constitutes one iteration. The TDMP can be described with equation (8-11):

$$\vec{R}_{l,n}^{(0)} = 0, \vec{P}_n = \vec{L}_n^{(0)} \quad (8)$$

$\forall i = 1, 2, \cdots it_{\max}$, [Iteration loop]

$\forall l = 1, 2, \cdots, j$, [Sub-iteration loop]

$\forall n = 1, 2, ...k$ [Block column loop]

$$\left[\vec{Q}^{(i)}_{l,n'}\right]^{-S(l,n)} = \left[\vec{P}_n\right]^{-S(l,n)} - \vec{R}^{(i-1)}_{l,n} \quad (9)$$

$$\vec{R}^{(i)}_{l,n} = f(\left[\vec{Q}^{(i)}_{l',n'}\right]^{-S(l',n)}) \quad (10)$$

$$\left[\vec{P}_n\right]^{-S(l,n)} = \left[\vec{Q}^{(i)}_{l,n'}\right]^{-S(l,n)} + \vec{R}^{(i)}_{l,n} \quad (11)$$

where the vectors $\vec{R}^{(i)}_{l,n}$ and $\vec{Q}^{(i)}_{l,n}$ represent all the R and Q messages in each block of H matrix, $s(l,n)$ denotes the shift coefficient for the block in l^{th} block row and n^{th} block column of the H matrix $\left[\vec{R}^{i-1}_{l,n}\right]^{S(l,n)}$ denotes that the vector $\vec{R}^{i-1}_{l,n}$ is cyclically shifted up by the amount $s(l,n)$, k is the check-node degree of the block row. A negative sign on $s(l,n)$ indicates that it is cyclic down shift (equivalent cyclic left shift). $f(\cdot)$ denotes the check-node processing, which can be done using BCJR, SP or MS. For the proposed work we use MS as defined in equation (2-7). As soon as the output vector $\vec{R}^{(i)}_{l,n}$ corresponding to each block column n in H matrix for a block row l is available, this could be used to produce updated sum $\left[\vec{P}_n\right]^{-S(l,n)}$ (10). This could be immediately used in (9) to process block row $l+1$ except that the shift $s(l,n)$ imposed on $\vec{P}_n$ has to be undone and a new shift $s(l+1,n)$ has to be imposed. This could be simply done by imposing a shift corresponding to the difference of $s(l+1,n)$ and $s(l,n)$. Due to the structure of array LDPC H matrix, the $\left[\vec{P}_n\right]^{-S(l,n)}$ in each block column n need to go through either a) a cyclic down shift of $n-1$ or b)cyclic up shift of $(j-1)n$ if we do layered decoding. In addition we can do parallel processing of block-column loop using p parallel CNUs. The property a) is possible due to constant difference of shift across block columns in array code's H matrix and layers are processed in the order from 1 to j in each iteration while the property b) is due to the fact that we need to process layer 1 after processing layer j. We utilize this property in the proposed layout aware architecture by implementing the two required cyclic shifts with wiring (and concentric layout) and by processing all block columns in a layer in parallel.

3. Value-reuse Properties of Check Node Processing of Min-Sum Decoding

One of the key contributions of this paper is the observation that offset min-sum decoding algorithm share the value-reuse property, as explained below, hence reduce the logic and message passing requirement of the decoder. For each check node m, $\left|R^{(i)}_{mn}\right|$ $\forall n \in \mathrm{N}(m)$ takes only 2 values and they are the two least minimum of input magnitude values. Since $\forall n \in \mathrm{N}(m)$, $\delta^{(i)}_{mn}$ takes value of either $+1$ or -1 and $\left|R^{(i)}_{mn}\right|$ takes only 2 values. Equation (3) gives rise to only 3 possible values for the whole set $R^{(i)}_{mn}$ $\forall n \in \mathrm{N}(m)$. In a VLSI implementation, this property greatly simplifies the logic and reduces the memory. This result would greatly simplify the number of comparisons required as well as the memory needed to store CNU outputs. This also means reduction in correction computations. Normally CNU (check-node unit) processing is done using the signed magnitude arithmetic for (3) and VNU (variable-node unit processing) equation (4) is done in 2's complement arithmetic. This requires 2's complement to signed conversion at the inputs of CNU and signed to 2's complement at the output of CNU. This would mean that we need to apply 2's complement to only two values instead of k values at the output of CNU. The work in [8] proposed a parallel CNU for MS that requires $k(1+0.5(\log_2 k - 1))$ comparators, $4k$ 5-bit adders-the reason for this complexity is not using the value re-use property of MS. In steps 1 and 2 below, consider the case of rate 0.5 (4,8) code so that $k = 8$ and assume the word length of signed magnitude bit node messages is 5 so that there are 4 bits allocated for magnitude.

Step 1: Locate the two minimum values of the array

Step1.1: Find the first minimum using the binary tree. Figure 1a gives a binary tree of comparators. Each comparator take two 4 bit input words and produces the minimum and a flag which is set to 1 if the upper input is less than the lower input

Step 1.2: Select the survivors by using the comparator output flags as the control inputs to multiplexes. For example in the last stage of the comparator tree the value other than the least minimum is the survivor. No further comparisons are necessary along the tree path to the survivor. We trace back the survivors using the comparator outputs. For a binary tree for input vector of length 8, we need 2 to 1, 4 to 1 and 8 to 1 multiplexers. At any stage of binary tree we have only one survivor. So there would be $\log_2 k$ survivors and $\log_2 k - 1$ comparisons in sequential fashion

Step 1.3: Perform the required comparisons among survivors in sequential fashion.

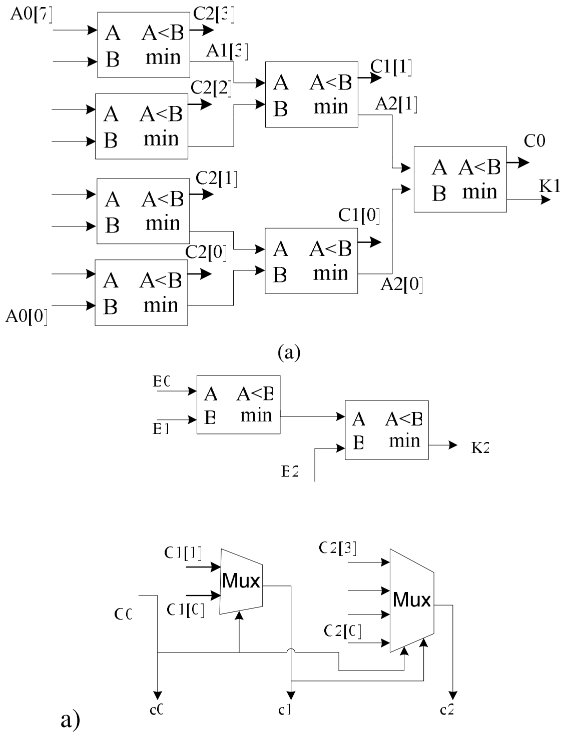

Figure 1: Finder for the two least minimum in CNUBinary tree to find the least minimum. (b) Trace-back multiplexers and comparators on survivors to find the second minimum. Multiplexers for selecting survivors are not shown.

In Figure 1, C0, C1 and C2 are 1 –bit outputs corresponding to A<B condition. '0' in C0 notation is used to denote first level of outputs from right and so on C2[0] does mean the 1 bit comparator output at the first output of comparators at 3rd level outputs from right and so on. A2,A1 and A0 are magnitudes(usually 4 bits wide[2]) of bit node messages. '0' in A0 notation is used to denote first level of inputs and so on. A0[0] does mean the 4bit input word at the first input of first level of inputs and so on. It does not mean 0th bit of A0.

K1 =A0 [C0 C1[C0] C2[C0C1[C0]]] is the least minimum. The 3 bit trace back C0 C1[C0] C2[C0C1[C0]] gives the location of the least minimum. See Figure 2c. The following inputs are obtained from the intermediate nodes of the search tree. We have to use 2-in 1, 4-in 1 and 8-in 1multiplexers respectively to obtain the following survivors.

B2= A2[c0]; ,

B1= A1[!c1 c0]];

B0= A0[!c2 c1 c0]

Step 2: This steps produces the R outputs according to (7) in the following optimized fashion. Apply the offset to K1 and K2 to produce M1 and M2. Now compute –M1 and/or –M2. Now based on computed sign information by XOR logic(4) and index of K1, R is set to one of the 3 possible values M1,-M1, +/- M2. (see Figure 2).

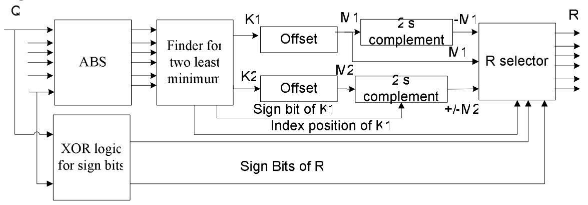

Figure 2: Proposed Parallel CNU based on Value-reuse property of Min-Sum

4. Fully Parallel Architecture Using TDMP and Min-Sum

A new data flow graph , see Figure 3, is designed based on the TDMP and on the value reuse property of min-sum algorithm described above. For ease of discussion, we will illustrate the architecture for a specific structured code: Array code of length 2082 described in section 2, $j=3$, $k=6$ and $p=347$. First, functionality of each block in the architecture is explained. A check node process unit (CNU) is the parallel CNU based on offset min-sum described in previous section. The CNU array is composed of p computation units that compute the R messages for each block row in fully parallel fashion. Since R messages of previous $j-1$ block rows, are needed for TDMP, the *compressed* information of them is stored in FS register banks. There is one register bank of depth $j-1$, which is 2 in this case, connected with each CNU. Each final state contains M1,-M1,+/-M2 and index for M1. The sign bits are stored in sign flip-flops. The total number of sign flip-flops is k and each block row has pk sign flip-flops. We need $j-1$ of such sign flip-flop banks in total. p number of R select units is used for R_{old} . An R select unit, whose functionality and structure is the same as the block denoted as R select in CNU, generates the R messages for $6(=k)$ edges of a check node from 3 values stored in final state register in parallel fashion. This unit can be treated as de-compressor of the check node edge information which is stored in compact form in FS registers.In the beginning of the decoding process (9-11), *P* matrix(of dimensions *p*x *k*) is set to received channel values in the first clock cycle(i.e. the first sub-iteration) while the output matrix of R select unit is set to zero matrix. The multiplexer array at the input of P buffer is used for this initialization. The P matrix is computed by adding the shifted $\vec{Q}_l$ matrix(labeled as Q^{shift} in Figure 3)to the output matrix $\vec{R}_l$ of the CNU array(labeled as R_{new}). The compressed information is stored in the register banks FS and sign so that these can be used to generate R_{old} for the l^{th} sub-iteration in next iteration.

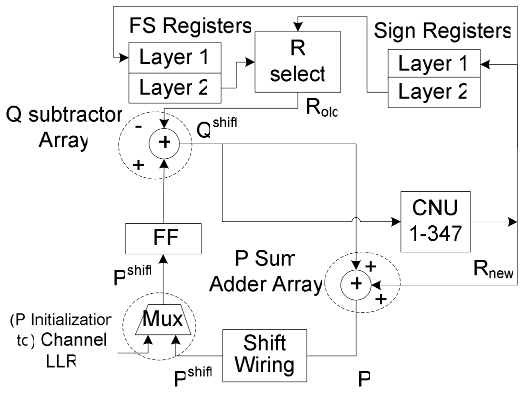

Figure 3: Dataflow graph of the proposed parallel architecture for layered decoder

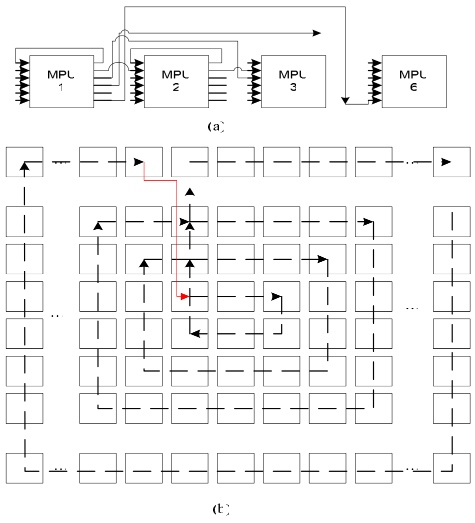

Figure 4: a)Illustration of connections between Message Processing units to achieve cyclic down shift of (n-1) on each block column n b) Concentric layout to accommodate 347 message processing units. Connections for cyclic up shift of 2n(=(j-1).n) are not shown.

Note that the *P* matrix that is generated can be used immediately to generate the Q matrix as the input to the CNU array as CNU array is ready to process the next block row. Now each block column in the *P* message matrix will undergo a cyclic shift by the amount of difference of the shifts of the block row that is processed and the previous block row that was just processed in the previous sub-iteration. Figure 4 gives the layout details for the proposed decoder. The 2*k* adder units, *k* R select unit is termed as the parallel variable-node unit (VNU). A message processing unit (MPU) consists of a parallel CNU, a parallel VNU and associated registers belonging to each row in *H* matrix. MPU *I* communicates with its 5(=*k-1*) adjacent neighbors (MPUs whose numbers are mod(*i+1,p*)...mod*(i+5,p))* to achieve cyclic down shifts of 1,2,..5 respectively for block columns 2, 3 …6 in *H* matrix (1) as shown in Figure 4.

5. ASIC IMPLEMENTATION RESULTS

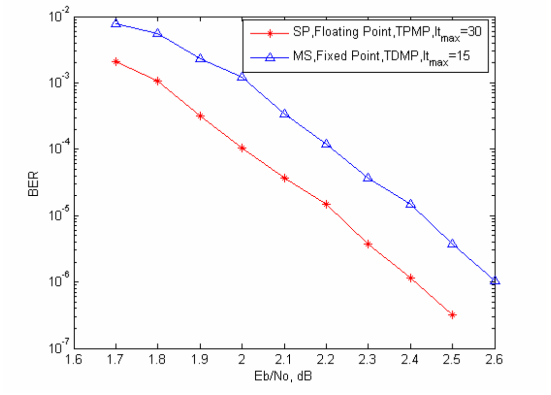

Figure 5: BER Performance of the decoder

To achieve the same BER as that of TPMP schedule on SP, TDMP schedule on offset MS need half the number of iterations. This essentially doubles the throughput for the given amount of parallelization. Parallel decoder for TDMP or layered decoding needs to use the hardware to decode one layer only while the parallel decoder for TPMP will have the hardware to decode all the layers simultaneously. So TDMP decoder takes *j* clock cycles to finish *j* sub-iterations(for *j* layers) to constitute iteration as explained in Section 2. If we use 5-bit precision in R and Q messages and 6-bit precision in P messages to achieve almost the same performance as that of the floating point SP with only 0.2 dB performance degradation. The proposed parallel CNU for offset MS is of very low complexity and requires $k+\log_2 k-2$ 4-bit comparators and $k+4$ 5-bit adders. The work in [8] proposed a parallel CNU for MS that requires $k(1+0.5(\log_2 k-1))$ comparators, 4 k 5-bit adders-the reason for this complexity is not using the value re-use property of MS. It is also observed that instead of storing all the R messages, the compressed R information is stored along with the sign bits of the R messages. This result in a reduction of R memory is around of 20%-72% for 5 bit messages based on the check node degree k of the code. So, in the proposed architecture designed to support the code with $k=6$, the savings of R memory is 20%. In addition there are significant savings due to the layered decoding and parallel architecture. We implemented the proposed fully parallel layered decoder architecture on 0.13 micron technology. We used the standard cells *vsclib013*[10].

TABLE I: COMPARISON WITH THE EXISTING WORK

	[2]	[5]	[9]	Proposed
Decoded Throughput	640 Mbps	1.0 Gbps	1.0 Gbps	4.6 Gbps
Area	14.3 mm^2,	52.5 mm^2	8.74 mm^2	5.29 mm^2
Memory	51,680 bits (excluding flip-flops and FIFO s)	34,816 bits (scattered flip-flops)	34,816 bits (scattered flip-flops)	27046 bits (scattered flip-flops)
Interleaver/Router	3.28 mm^2-Network	26.25 mm^2-Network	NA	0.89 mm^2-Network
Frequency	125 MHz	64 MHz	64 MHz	100 MHz
LDPC Code	AA-LDPC, Structured rate 0.5 and (3,6) code	Random and irregular rate 0.5 code	Modified Array Code rate 0.5 and (3,6) code	Array Code rate 0.5 and (3,6) code
Block Length	2048	1024	1024	2082
Check Node update	BCJR	SP	SP	Offset MS
Decoding Schedule	TDMP	TPMP	TPMP	TDMP
CMOS Technology	0.18 μ	0.16 μ	0.16 μ	0.13 μ

The area of the chip is 2.3 mm x 2.3 mm and the post routing frequency is 100 MHz. The estimated gate count is 787 K 2-input NAND gates in 0.13 micron technology. The ASIC implementation of the proposed parallel architecture achieves a decoded throughput of 4.6 Gbps for 15 iterations and user data throughput of 2.3 Gbps. The area distribution of the CNU, VNU, storage flip-flops to support layered decoding, synchronization flip-flops to support multiple iterations and wiring is respectively 32%, 29%, 12%, 10% and 17%.

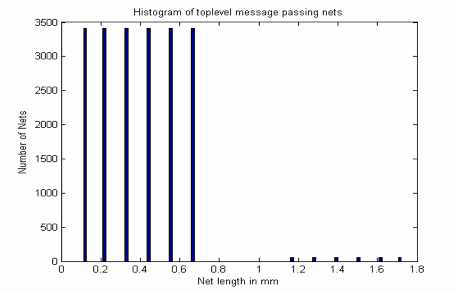

Figure 6: Net length distribution of top level message passing nets for code with k=6 and p=347

The net length distribution of the message passing wires P is given in Figure 6. Most of the wires are short due to concentric layout. The net length distribution of the channel LLR message nets(which need not be shifted as the first block row of array H matrix have all zero shift coefficients) and decoded bit nets has uniform bins at i*0.1222 mm, $\forall i = 1,2,...,10$ assuming that the inputs and outputs are also arranged in ring fashion around the chip. Note that channel LLRs are inputs to decoder message processing units for every new frame-every 45 clock cycles assuming 15 iterations and 45(=15*3) sub-iterations. However the additional IO circuitry (the serial to parallel and parallel to serial conversion around the chip) which is application dependent is not accounted in the chip area and is estimated not to exceed 15% of chip area. Table I gives the performance comparison with the state of the art work. This work offers a synergetic combination of array LDPC codes [6], reduced complexity decoding [4], layered decoding [2] by providing the novel micro-architecture structures and architectures to offer significant advantages.

6. Conclusion

We present layout-aware parallel decoder architecture for turbo decoding message passing of array LDPC codes using the min-sum algorithm for check node update. Our work offers several advantages when compared to the other state of the art LDPC decoders in terms of significant reduction in logic and interconnect.

References

[1] D.J.C. MacKay and R.M. Neal. "Near Shannon Limit Performance of Low Density Parity Check codes" *Electronics Letters*, volume 32, pp. 1645-1646, Aug 1996.

[2] M. Mansour and N. Shanbhag, "A 640-Mb/s 2048-bit programmable LDPC decoder chip," *IEEE Journal of Solid-State Circuits*, vol. 41, no.3, pp. 684- 698, March 2006.

[3] Hocevar, D.E., "A reduced complexity decoder architecture via layered decoding of LDPC codes," IEEE SiPS 2004 pp. 107- 112, 13-15 Oct. 2004

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

[5] Blanksby, A.J.; Howland ,C.J, 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, Mar 2002 Pages:404-412

[6] S. Olcer, "Decoder architecture for array-code-based LDPC codes," *Global Telecommunications Conference*, 2003. GLOBECOM '03. IEEE , vol.4, no.pp. 2046- 2050 vol.4, 1-5 Dec. 2003

[7] E. Kim, G. Choi, "Diagonal low-density parity-check code for simplified routing in decoder," IEEE SiPS 2005, vol., no.pp. 756- 761, 2-4 Nov. 2005

[8] M. Karkooti and J. Cavallaro, "Semi-parallel reconfigurable architectures for real-time LDPC decoding," Proceedings of International Conference on Information Technology, Coding and Computing, vol. 1, pp. 579-585, 2004.

[9] V. Nagarajan, *et al*, "High-throughput VLSI implementations of iterative decoders and related code construction problems,", GLOBECOM '2004. IEEE , vol.1, pp. 361- 365, vol. 1, 29 Nov.-3 Dec. 2004.

[10] Open source standard cell library, http://www.vlsitechnology.org