A LOW POWER ARCHITECTURE FOR MIN-SUM DECODING OF LDPC CODES

Kiran Gunnam, Gwan Choi
Dept. of Electrical and Computer Engineering, Texas A&M University, College Station, TX-77840

ABSTRACT

We propose novel micro-architecture structures for check node message processing unit (CNU) for the min-sum decoding of Low-Density Parity-Check codes (LDPC). The construction of these CNU structures is based on a less known property of the min-sum processing step that it produces only two different output magnitude values irrespective of the number of incoming bit-to check messages. These new micro-architecture structures would employ the minimum number of comparators by exploiting the concept of survivors in the search. These would result in reduced number of comparisons and consequently reduced energy use.

1. INTRODUCTION

Low density parity check (LDPC) codes, first invented in are in 1962 [1], are Shannon-limit approaching codes. However these are overlooked for years, mainly due to lack of implementation technology. Recently these are re-discovered for efficient VLSI implementations. LDPC codes are now potentially superior alternatives to widely-accepted and used Turbo codes which also rely on message passing. In terms of coding gain, it has been shown that LDPC codes are asymptotically superior to turbo codes in achieving the Shannon capacity. The operations involved in the LDPC decoding are less compute intensive and have increased parallelism when compared to the Turbo decoder even with the increased number of iterations for LDPC. One of the major obstacle in VLSI implementation of LDPC decoders is the complex and random interconnects and large memory requirements due to storage of messages. Structured LDPC codes addresses the problem of complex interconnects and facilitates the practical implementation of decoding possible.

LDPC codes can be described by a $m \times n$ parity check matrix H in which the average number of non-zero elements (i.e. one in GF2) in each row is a constant. In a (d_v, d_c) regular code, each of the n bit/variable nodes $\left(b_1, b_{2,}, ..., b_n\right)$ has connections to d_v check nodes and each of the m check nodes $\left(c_1, c_{2,}, ..., c_m\right)$ has connections to d_c bit nodes.

LDPC codes can be decoded by the Gallager's iterative belief-propagation (BP) or sum of products algorithm (SP) [1]. The BP algorithm can also be simplified to the BP-based algorithm(also called Min-Sum) [2], which greatly reduces the decoding complexity in implementation, but degrades the decoding performance by up to 0.5 dB for rate 0.5 codes and 0.2 dB for rate 0.9 codes. Normalized BP-based algorithm (Normalized-Min-Sum) [2] and another improved BP-based algorithm, named offset BP-based algorithm (Offset-Min-Sum) [2] eliminates this performance degradation. Both correction algorithms uses an approximation in check to bit message processing like the Min-sum algorithm and either multiplies the check node messages with a constant scaling factor of less than 1(Normalized-Min-Sum) or subtracts the check node messages with a constant offset parameter to reduce the overestimation(Offset-Min-Sum). Both methods have the same complexity and would make the implementation of LDPC decoding an attractive and low power when compared to Turbo decoders and SP decoder of LDPC.

The organization of the paper is as follows. Section 2 presents the decoding based on Min sum algorithm and modified versions of it. Section 3 presents the value-re use property of the search algorithms involved in the Min-Sum algorithms. Section 4 presents new micro-architecture structures for CNU based on the observations made in Section 3. Finally some conclusions are drawn in Section 5 and 6.

2. MIN-SUM DECODING OF LDPC

In this paper, we follow the notation as in [1, 2] and expand equations in alternate forms as necessary for illustrating various properties of the Min Sum decoind scheme. Assume BPSK (Binary Phase Shift Keying) modulation (a 1 is mapped to -1 and a 0 is mapped to -1) over an AWGN (Additive White Gaussian channel). The received values y_n are Gaussian with mean $x_n = \pm 1$ and variance $\sigma^2 = N_0 / 2$.

A. Min Sum Algorithm

The reliability messages used in BP based Min Sum algorithm can be computed in two phases viz. check node processing (1) and bit node processing (2) and this is repeated iteratively till the decoding criterion is satisfied For iteration i, $Z_{mn}^{(i)}$ is the message from bit node n to check node m, $L_{mn}^{(i)}$ is the message from check node m to bit node n, $\mathrm{M}(n)$ is the set of the neighboring check nodes for bit node n, $\mathrm{N}(m)$ is the set of the neighboring bit nodes for check node m.

The message passing equations for each iteration i are given by

1) Check Node Processing: for each m and $n \in \mathrm{N}(m)$, compute

$$L_{mn}^{(i)} = \delta_{mn}^{(i)} \kappa_{mn}^{(i)} \quad (1)$$

$$\kappa_{mn}^{(i)} = \min_{n' \in \mathrm{N}(m) \setminus n} \left| Z_{mn'}^{(i-1)} \right|. \quad (2)$$

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

$$\delta_{mn}^{(i)} = \left(\prod_{n' \in \mathrm{N}(m) \setminus n} \mathrm{sgn}\left(Z_{mn'}^{(i-1)} \right) \right) \quad (3)$$

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

2) Bit Node Processing: for each n and $m \in \mathrm{N}(n)$, compute

$$L_{mn}^{(i)} = L_n^{(0)} + \sum_{m' \in \mathrm{M}(m) \setminus m} L_{m'n}^{(i)} \quad (4)$$

Where

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

B. Normalized Min-Sum:

Check node processing in (2) would overestimate the reliability metric L_{mn}^{i} and can be corrected by a scaling factor λ which depends on the code parameters and the iteration number [2].

$$L_{mn}^{(i)} = \delta_{mn}^{(i)} \lambda \kappa_{mn}^{(i)} \quad (5a)$$

C.Offset Min-Sum:

Overestimation in $L_{mn}^{(i)}$ and can also be corrected by subtracting a positive constant β from the magnitude of the $L_{mn}^{(i)}$ in the following way:

$$L_{mn}^{(i)} = \delta_{mn}^{(i)} \max\left(\kappa_{mn}^{(i)} - \beta, 0 \right) \quad (5b)$$

For (3,6) rate 0.5 code, λ and β is computed as 0.8 and 0.15 using the density evolution technique and can be used as constant for all iterations[2]. We used the procedure in [2] to compute λ and β as 0.8 and 0.52 for (3,32) rate 0.9 code.

3. VALUE-REUSE PROPERTIES OF CHECK NODE PROCESSING OF MIN-SUM DECODING

We note that the following fact without proof is stated in [2]. (2) can be determined by identifying the two minimum values corresponding to this check sum. Check node processing can be achieved with at most $2d_c$ comparisons in serial search, or with at most $d_c + \log 2d_c - 2$ comparisons with the help of a binary tree. No further details on how to perform the parallel search is not given. Perhaps this might be the reason that most of the published results on hardware implementation of parallel Min-Sum unit did not use this result. For example, [3,4] resorts to the use of 2* d_c comparators and additional processing such as offset correction and 2's complement for all d_c messages. However one can easily note that Serial search is straight forward using the partial bubble sort. [5] used some of these properties efficiently for serial unit, however it unnecessarily uses a data width of more than the desired 5 bits[2]. In addition [5] does not use the fact that two's complements are only needed once or twice and resorts to use 2's complement for all d_c messages. [5] stored the two least minimum of incoming messages at check node and index for the least minimum. [6] adopted the method presented in

[5] to have the similar savings in message storage- except for that the underlying architecture is a serial one instead of a scalable architecture In addition, the algorithm proposed in [6] is variation of sum of products algorithm –this is much complex than the Offset Min-Sum for the same performance and same savings in message storage. The contribution of this paper will be formal frame work for value re-use properties of Min-Sum and its variants and improved serial and parallel implementation of check to node units.

We will derive a proof to highlight the result presented in [2] and present ways to achieve the optimal number of additions/comparisons (and comparators) for both serial and binary tree methods for MinSum and its variants.

Min Sum decoding,. Normalized Min-Sum and Offset Minsum have the following property.

Lemma 1:

For Each check node m ,

$\left|L_{mn}^{(i)}\right| \ \forall n \in \mathrm{N}(m)$ takes only 2 values

Proof: In order to prove this, it is sufficient to prove that

$\kappa_{mn}^{(i)} \ \forall n \in \mathrm{N}(m)$ takes only 2 values out of d_c possible values. Either normalization or offset methods operate using constants for all the check node messages. If there is conditional correction , then this result will not apply- however the Bit error rate (BER) performance difference between unconditional correction and conditional correction is negligible. So we restrict ourselves to the correction methods in [2].

Let us define the least minimum of the entire set of the messages from various bit nodes to the check node m as

$$\kappa 1_m^{(i)} = \min_{n' \in \mathrm{N}(m)} \left|Z_{mn'}^{(i-1)}\right|. \qquad (6)$$

Assume that the least minimum $\kappa 1_m^{(i)}$ for check node m is equal to the message from bit node k , i.e.

$$\kappa 1_m^{(i)} = \left|Z_{mk}^{(i-1)}\right|. \qquad (7)$$

Now (2) becomes

$$\kappa_{mn}^{(i)} = \kappa 1_m^{(i)}, \ \forall n \in \mathrm{N}(m) \setminus k$$

$$= \min_{n' \in \mathrm{N}(m) \setminus n} \left|Z_{mn'}^{(i-1)}\right|, \text{ if } n = k \qquad (8)$$

If we define the second least minimum of the entire set of the messages from various bit nodes to the check node m as

$$\kappa 2_m^{(i)} = 2nd \min_{n' \in \mathrm{N}(m)} \left|Z_{mn'}^{(i-1)}\right|. \qquad (9)$$

Now (8) becomes

$$\kappa_{mn}^{(i)} = \kappa 1_m^{(i)}, \ \forall n \in \mathrm{N}(m) \setminus k$$

$$= \kappa 2_m^{(i)}, n = k \qquad (10)$$

It is clear that the least minimum value and second least minimum values of the d_c bit node messages are the result of check node magnitude processing in (2).

Lemma 2

$L_{mn}^{(i)} \ \forall n \in \mathrm{N}(m)$ takes only 3 values

Proof

Since $\forall n \in \mathrm{N}(m)$

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

and $\left|L_{mn}^{(i)}\right|$ takes only 2 values

Equation (1) gives rise to only 3 possible values for the set

$L_{mn}^{(i)} \ \forall n \in \mathrm{N}(m)$

E. Application of above properties

- This result would greatly simplify the number of comparisons required as well as the memory needed to store CNU outputs.
- This would also mean that the correction has to be applied to only two values instead of d_c distinct values.
- Normally CNU (check node unit) processing equations (1-3) are done using the signed magnitude arithmetic for (1-3) 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 2 values instead of d_c values at the output of CNU.

4. NEW MICROARCHITECTURES FOR CHECK NODE PROCESSING OF MIN-SUM DECODING

A. Problem statement

Based on the above observations in Section 3, we propose several new efficient micro-architectures for CNU. The problem in (2) can be done through (6),(9) and (10). This can be re-stated as follows:

1. Define an array Z_m of size d_c and the elements of the array are the magnitudes of the message from d_c bit nodes connected to check node. i.e $Z_m = [Z_{mn'}^{(i)}] \forall n \in \mathrm{N}(m)$. Similarly define the outputs of check node magnitude processing as $\kappa_m = [\kappa_{mn}^{(i)}] \ \forall n \in \mathrm{N}(m)$
2. Find the minimum two numbers $\kappa 1_m^{(i)}$, $\kappa 2_m^{(i)}$ (Equations (6) and (9))
3. Define a register $Flag\_Min$ of size d_c. Set l^{th} bit of $Flag\_Min$ to 1 of the l^{th} element in the array Z_m is the least minimum number $\kappa 1_m^{(i)}$
4. Set the output of the Min-Sum magnitude processing module to $\kappa 1_m^{(i)}$ for the input $Z_m^{(i)}[l]$ if $Flag\_Min[l] = 0$. (so magnitude of d_c -1 messages are the same and equal to $\kappa 1_m^{(i)}$). Set the output of the Min-Sum magnitude processing module to $\kappa 2_m^{(i)}$ for the check node m to bit node n if $Flag\_Min[l] = 1$. [Equation (10)]

Steps 3 and 4 can also be done by storing the position of $\kappa 1_m^{(i)}$ and comparing that to a counter which runs from 0 to d_c. This would be useful when d_c is more than 6 and also when state of one CNU has to be shifted to another CNU as part of cyclic shift.

B. Serial Search CNU

Assume we are doing the search for one of the check nodes of a (3,6) code. $d_c = 6$. This discussion is applicable for any rate code with the change in the size of the one hot counter and $Flag\_Min$ registers.

Step 1: Locate the two least minimum values of the array Z_m

Initialization:

$K1 = Z_m[0]$

$K2 = INFINITY$ (i.e. is set to all ones)

Set two 6 bit registers

$One\_Hot\_Counter\_6bit = 000001$

$Flag\_Min = 000001$

For $l = 1 : d_c - 1$

$One\_Hot\_Counter\_6bit << 1;$

(Rotate the register left by one bit to indicate the position of serial search)

If ($Z_m[l] < K2$) then

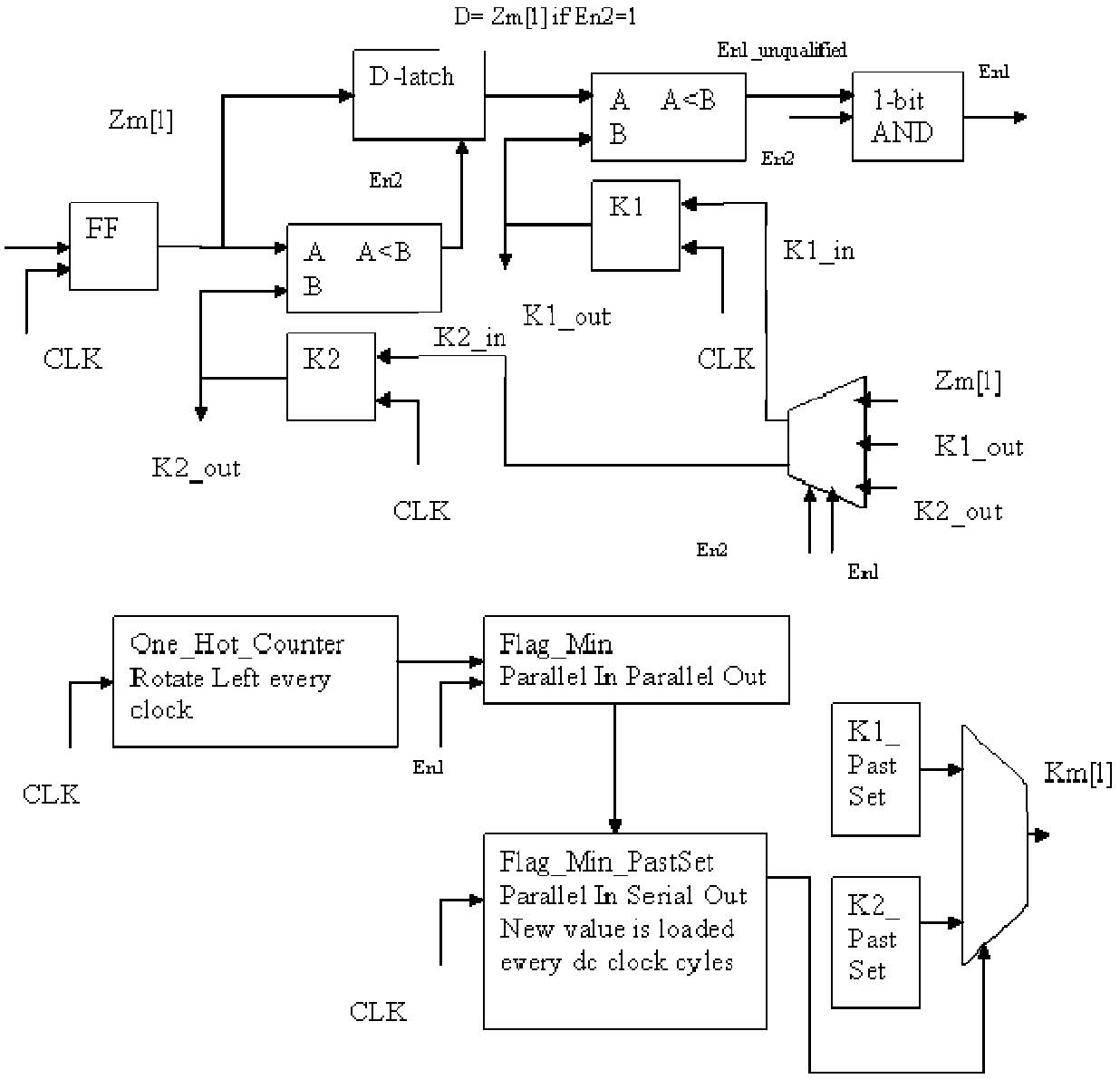


Figure 1: Micro Architecture of the Low Power Serial Check node Processing Unit (CNU)

begin
If ($Z_m[l] < K1$)
begin
$K2 = K1$;
$K1 = Z_m[l]$;
$Flag\_Min = One\_Hot\_Counter\_6bit$;
End
else
begin

$K2 = Z_m[l];$
End
end

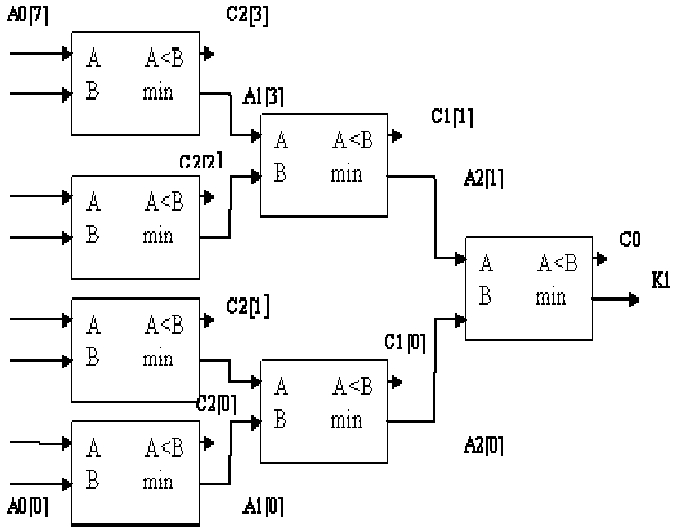
$Flag\_Min\_PastSet = Flag\_Min$
$K2\_PastSet = K2;$
$K1\_PastSet = K1;$
(This $Flag\_Min\_PastSet$ register is a parallel In serial Out register. This register's will be used to select the outputs)

Step 2: Produce the outputs for the previous set. This can be done in parallel with Step 1.

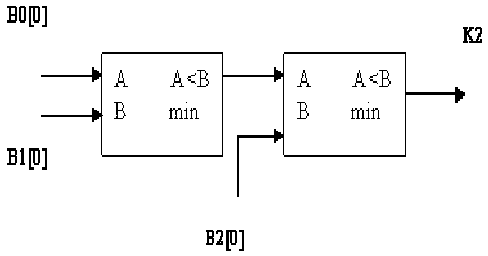
For $l = 0 : d_c - 1$
If($Flag\_Min\_PastSet$ registers serial out)
$\kappa_m(l) = K2\_PastSet$
Else
$\kappa_m(l) = K1\_PastSet$
Detailed Micro architecture block diagram is given in Appendix A.

C. Binary Tree Search Parallel CNU

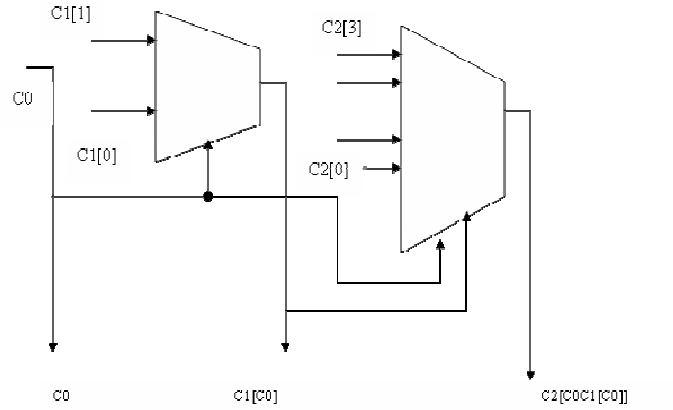
Consider the case of rate 0.5 (4,8) code so that $d_c = 8$ and assume the word length of signed magnitude bit node messages is 5 so that there are 4 bits allocated for magnitude.
Step 1: Locate the two least minimum values of the array
Step1.1: Find the least minimum using the binary tree. Figure 2a 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 in 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 brack 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 muxes. At any stage of binary tree we have only one survivor. So there would be log2(d_c) survivors and log2(d_c) -1 comparisons in sequential fashion*
Step 1.3: Perform the required comparisons among survivors in sequential fashion.
Step 2: Produce the outputs for the previous set. This can be done in parallel with Step 1. This is same as step 2 for serial CNU.
In Figure 2, 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 frist input of first level of inputs and so on. It does not mean 0th bit of A0
K1 =A0[C2[C0C1[C0]] C1[C0] C0] is the least minimum. The 3 bit trace back C2[C0C1[C0]] C1[C0] 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 1 MUXes respectively to obtain the following survivors



a) Binary Tree to find the least minimum



b) Comparators on survivors to find the second least minimum K2



c) Trace back MUX for the position of the least minimum which is equal to C2[C0C1[C0]] C1[C0] C0. A register called Flag_Min similar to the one employed in serial CNU is also used here to do the step 4 of the one bitSimilar trace back MUXes are used for obtaining survivors.

Figure 2: Micro Architecture of the Low Power Parallel Check node Processing Unit (CNU)

B2[0]= A2[[!C0]]; ,

B1[0]= A1[!C1[C0]] B0[0]= A0[!C2[C1[C0]C0] C1[C0]]

Key assumptions are circuit in Figure 2a receives the inputs in registers and they are valid at least until the survivors are produced. So pipeline latches can be inserted between circuit a (including MUXes) and circuit b to speed up the computation.

D.Offset correction or scaling correction

This has to be invoked only on two values – so these computations can be time folded on to the comparison units in CNU if.they are made as 2's complement sub tractors to perform both unsigned comparisons as well as subtraction/addition. It can be noted that scaling can be achieved by shift and additions.

E.Two's complement to signed

Signed to two's complement have to be invoked only on two values – so these computations can be time folded on to the comparison units in CNU if. they are made as 2's complement sub tractors.

5. PERFORMANCE COMPARISION

For each check node	Comparators/	Comparisions(N_s)	Throughput Number of clock cycles for a new set of results to be available	Latency	Additional Hardware for Min Sum Correction a nd 2's complement	Additional Processing operations for Correction and 2's complement
Proposed Serial	2	$d_c - 1 < N_s < 2d_c - 2$	d_c	d_c	0.Time folded Memory storage:	3-4
Serial[5]	2	$d_c - 1 < N_s < 2d_c - 2$	d_c	d_c	0. Time folded for correction. d_c. For 2's complement	2
Proposed Parallel	$d_c + \log 2d_c - 2$	$d_c + \log 2d_c - 2$	1	1	0.Time folded	3-4
Parallel [3,4]	$2d_c$	$2d_c$	1	1	$2d_c$ adders	$2d_c$

LDPC code has N-K check nodes.if N is code length and K is the message length.. d_c is usually from 6 for rate 0.5 codes and around 36 for rate 0.9 codes.

State for proposed architectures are:

Internal State:

Minimum1, Minimum 2, Index of Minimum 1

Final State:

Minimum1 with correction, -Minimum1 with correction, +/- Minimum2 with correction, Index of Minimum 1

(4 locations, 3 bit signed magnitude messages. Index of Minium1 has log(d_c) bits): this is around 18 bits if d_c is 32

Index approach is used for $d_c > 6$,. We need additional d_c equal to comparators.

If d_c is 6, Flag_Min register instead of Index of Minium1. No additional comparisons are necessary.

State for existing serial architectures are:

Internal
Minimum1, Minimum 2, Index of Minimum 1
Final State
Minimum1 with correction, Minimum2 with correction, Index of Minimum 1
(3 locations, 4 bit magnitude messages. Index of Minium1 has log(d_c) bits): this is around 13 bits if d_c is 32

Index approach is used and need additional d_c equal to comparators.

When compared to existing serial micro-architecture for CNU, the advantage lies in storing the 5 extra bits in each CNU by pre computing the two complement by time folding on to one of subtractors. In this case only 1 to 2 two's complement operations are needed. Existing approaches have the two complement unit and its operation for possibly every clock cycle for each CNU.

6. CONCLUSION

A key observation in quantifying the number of distinct outputs of min sum unit and comparisons needed for obtaining them in decoding of LDPC is presented Micro architecture structures which achieve the optimal number of comparisons and hardware circuitry is presented The design has correspondingly low power requirements when compared to the existing work. Several variations of CNU unit design is presented that can be used based on the overall architecture of decoder.

.REFERENCES

[1] R. G. Gallager, Low-Density Parity-Check Codes, M.I.T Press, 1963. Available at http://justice.mit.edu/people/gallager.html
[2] 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.
[3] Karkooti, M.; Cavallaro, J.R.Semi-parallel reconfigurable architectures for real-time LDPC decoding ; Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. International Conference on Volume 1, 2004 Page(s):579 - 585 Vol.1 [
[4] . Hemati, A. H. Banihashemi, and C. Plett, "An 80-Mb/s 0.18-\mum CMOS Analog Min-Sum Iterative Decoder for a (32,8,10) LDPC Code," in Proceedings of the IEEE Custom Integrated Circuits Conference, CICC 2005, San Jose, California, pp. 240-243, Sept. 18 -21, 2005.
[5] Cocco, M.; Dielissen, J.; Heijligers, M.; Hekstra, A.; Huisken, J.;
"A scalable architecture for LDPC decoding" Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings Volume 3, 16-20 Feb. 2004 Page(s):88 - 93 Vol.3
[6] Abhiram Prabhakar and Krishna Narayanan, "A memory efficient Serial LDPC decoder architecture", ICASSP 2005, Philadelphia, Pages V-41 – V-44.