DECODING OF ARRAY LDPC CODES USING ON-THE –FLY COMPUTATION

Kiran Gunnam, Weihuang Wang, Euncheol Kim, 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

Message passing memory takes around 30% of chip area and consumes from 50%-90% power of the typical semi-parallel decoders for the Low Density Parity Check Codes (LDPC). We propose a new LDPC Decoder architecture based on the Min Sum algorithm that reduces the need of message passing memory by 80% and the routing requirements by more than 50%. This novel architecture is based on scheduling of computation that results in "on the fly computation" of variable node and check node reliability messages. The results are memory-efficient and router-less implementations of (3,30) code of length 1830 and (3,6) code of length 1226; each on a Xilinx Virtex 2V8000 FPGA device achieved 1.27 Gbps and 585 Mbps respectively.

EXTENDEDED ABSTRACT

Low Density Parity Check Codes (LDPC) codes which are among the Shannon limit codes have been given intensive attention in recent few years due to their merits in implementing a high throughput, low latency decoder. A suboptimal decoding, Sum of Product (SP), algorithm has been proposed for near Shannon limit performance and its approximate version, Offset Min-Sum (MS), algorithm has also been proposed [1]. Offset MS reduces the complexity of the decoding by removing the non-linear operations needed in SP.

We present a new architecture that exploits the various properties of structured Array LDPC codes [2] and the value–reuse properties of Min-Sum algorithm to reduce the memory, routing and computational requirements. The key features of this architecture are:

1. 80% savings in message passing memory requirements when compared to other semi-parallel architectures based on MS and its variants[10-11]
2. Scalable for any code length due to the concentric and regular layout unlike the fully parallel architecture [3]
3. Reduction of router muxes anywhere from 50% and beyond based on dynamic state concept.

REDUCED MESSAGE PASSING MEMORY AND ROUTER SIMPLIFICATION

Array codes are defined in [2] and have three parameters (d_v, d_c) and length N. Here d_v is the variable node degree and d_c is the check node degree. The size of circulant matrix block in array code is a prime number and is given by $p = N / d_c$. H matrix is given for different codes in Figure 1. Most of the previous work is in the area of semi-parallel implementation of structured LDPC codes, however most of them are based on SP, for instance[5-9]. [10-11] proposed architectures based on MS and its variants. In the architecture of [8], the check node messages in the H matrix are produced block column wise so that all the variable messages in each block column can be produced on the fly. Again these variable-node messages can be immediately consumed by the partial state computation sub-units in Check Node Units. This scheduling results in savings in message passing memory that is needed to store intermediate messages. This work extends above concepts used for SP to the Offset MS.

Cyclic shifters take around 10%-20% of chip area based on the decoder's parallelization and constitute the critical path of the decoder. We make an observation that if all the block rows are assigned to different computational unit arrays of Check Node Unit(CNU) and serial CNU processing across block row is employed, then we need to have a constant wiring to achieve any cyclic shift as each subsequent shift can be realized using the feedback of previous shifted value. This leads to the elimination of forward router between CNU and Variable Node unit (VNU) as well as the reverse router between VNU and CNU. This is possible due to the fact that block-serial processing is employed and Array codes have a constant incremental shift in each block row. For the first block row, the shift and incremental shift is 0. For the second block row, the shifts are [0,1,2,…, $d_c - 1$] and the incremental shift is 1. For the third block row, the shifts are [0, 2 ,…, $2 * (d_c - 1)$] and the incremental shift is 2.

NEW CHECK NODE UNIT MICRO ARCHITECTURE

The proposed serial and parallel Check node unit design[13] utilizes a less known property of the

min-sum algorithm that the check node processing produces only two different output magnitude values irrespective of the number of incoming variable-node messages. [10] 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 and does not utilize this property. This property would greatly simplify the number of comparisons required as well as the memory needed to store CNU outputs. Figure 2 shows the serial CNU architecture for (3, 30) code. In the first 30 clock cycles of the check node processing, incoming variable messages are compared with the two up-to-date least minimum numbers (partial state, PS) to generate the new partial state, which include the least minimum value, M1, the second minimum value M2 and index of M1. Final state (FS) is then computed by offsetting the partial state. It should be noted that the final state includes only three signed numbers, i.e. M1, -M1, +/-M2 with offset correction, and index of M1. VNU micro-architecture is implemented as a parallel unit as the number of inputs is small. It takes 3 Check node messages and one channel value. It is a binary tree adder followed by subtractors and 2's complement to signed magnitude conversion to generate Variable node messages [11].

ARCHITECTURE

Figures 3 and 4 present the proposed architecture and pipeline scheduling for the implementation of (3, 30) – Array LDPC code of length 1830 with the circulant matrix size of 61. The Check Node processing unit array is composed of 3 sub-arrays. Each sub-array contains 61 serial CNUs which compute the partial state for each block row to produce the check-node messages for each block column of H. Block row 1 is array of 61 simple CNUs. CNU array block row 2 and 3 are composed of dynamic CNUs(Fig 2b). The Variable node processing array is composed of 61 parallel VNU units which can process 3*61 messages at each clock cycle. The sign bits will be stored in a FIFO (implemented as RAM), however, there is no need to subject these values to shifts as these values are not modified in check node processing partial state processing.

In the array of simple serial CNU that is designed to do check node processing for first block row in H matrix, the check node processing for each row in H matrix is done such that all the comparisons are performed locally with in one CNU to update the partial state each clock cycle and transfer the partial state to final state d_c once every cycle. In the array of dynamic CNU designed for second block row in H matrix, CNU 122 gets its partial state from CNU 121, CNU 121 gets its partial state from CNU 120 and so on. Array of dynamic CNU designed for third block row in H matrix such that the connection between partial state registers among various units achieve cyclic shifts of [0,2,..,58]. Similar principle is used when making connections for the final state in the CNU array to achieve reverse routing.

As shown in Figure 4, initially the variable messages are available in row wise as they are set to soft log likelihood information (LLR) of the bits coming from the channel. Q Init is an SRAM of size $2^* N$ and holds the channel LLR values of two different frames. It can supply p intrinsic values to the VNUs each clock cycle. The data path of the design is set to 5 bits to provide the same BER performance as that of the floating point Sum of products algorithm with 0.1-0.2 dB SNR loss [1]. Each iteration takes $d_c + 3$ clock cycles. For (3, 30) code this results in 6*33 clock cycles to process each frame when a maximum number of iterations set to 6. For (3,6) code this results in 20*9 clock cycles to process each frame when number of iterations is set to 20.

RESULTS AND PERFOMANCE COMPARISON

The savings in message passing memory due to scheduling are 80% as we need to store only the sign bits of variable node messages. Forward router and reverse routers are eliminated. This results in reduction of number of multiplexers from $2*(d_c - 1)*\log 2(p)*p*wl$ (as routers are eliminated) to $(d_c - 1)*p*(3*wl + \lceil \log 2(d_c) \rceil + 1)$ (to support transfer of partial state to final state in the array of dynamic CNU). Here $wl = 5$ and is the word length of the data path.

Table 1 shows resource consumption of different components used in the design for (3, 30) code of length 1830. Implementations for (3, 30) codes of lengths 1830 and (3,6) code of length 1226 on a Xilinx Virtex 2V3000 device achieved 1.2 Gbps(system frequency 153 MHz) and 340 Mbps(system frequency 140 MHz) respectively. Up to our best knowledge our LDPC implementations achieves the highest throughput per given FPGA resources. Figure 5 gives comparison of design metrics for our designs with the other designs [10, 12] based on similar code parameters and Min Sum implementation.

REFERENCES

[1] 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.

[2] E. Eleftheriou and S. Olcer, “Low density parity-check codes for digital subscriber lines”, Proc. ICC’2002, New York, pp.1752-1757(2002).
[3] Blanksby, A.J.; Howland ,C.J, A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity-check code decoder, Solid-State Circuits, IEEE Journal of, Vol.37, Iss.3, Mar 2002 Pages:404-412
[4] T. Zhang and Parhi, “A 54 Mbps (3, 6)-regular FPGA LDPC decoder”, IEEE Workshop on Signal Proc. Systems, 2002. (SIPS '02), Oct. 2002, pp. 127 -132.
[5] Yijun Li; Elassal, M.; Bayoumi, M., "Power efficient architecture for (3,6)-regular low-density parity-check code decoder," Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium on , vol.4, no.pp. IV- 81-4 Vol.4, 23-26 May 2004
[6] A. Selvarathinam, G.Choi, K. Narayanan, A.Prabhakar, E. Kim, “A Massively Scalable Decoder Architecture for Low-Density Parity-Check Codes”, in proceedings of ISCAS’2003, Bangkok, Thailand.
[7] Mansour, M.M.; Shanbhag, N.R., "A 640-Mb/s 2048-bit programmable LDPC decoder chip," Solid-State Circuits, IEEE Journal of , vol.41, no.3pp. 684- 698, March 2006.
[8] K. Gunnam, G. Choi and M. B. Yeary, “An LDPC Decoding Schedule for Memory Access Reduction”, IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2004)
[9] Bhagawat, P.; Uppal, M.; Choi, G “FPGA based implementation of decoder for array low-density parity-check codes”.; Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP '05). IEEE International Conference on Volume 5, 18-23 March 2005 Page(s):v/29 - v/32 Vol. 5
[10] 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
[11]Yeo, E.; Pakzad, P.; Nikolic, B.; Anantharam, V., “High throughput low-density parity-check decoder architectures” in proceedings of Global Telecommunications Conference, 2001.Volume: 5, Page(s): 3019 –3024
[12] T. Brack, F. Kienle and N. Wehn. “Disclosing the LDPC Code Decoder Design Space” Design, Automation and Test in Europe (DATE) Conference 2006, pages 200-205, March 2006, Munich, Germany
[13] Kiran Gunnam, Gwan Choi, “A Low Power Architecture for Min-Sum Decoding of LDPC Codes”, Available online at http://dropezone.tamu.edu/techpubs. TAMU, ECE Technical Report, May, 2006, Publication No. TAMU-ECE-2006-02.

$$H = \begin{bmatrix} I & I & I & \dots & I \\ I & \alpha & \alpha^2 & \dots & \alpha^5 \\ I & \alpha^2 & \alpha^4 & \dots & \alpha^{10} \end{bmatrix}$$

Check Nodes: C1, C2, C3

Variable Nodes: V1 V2 V3 ... V6

$$\alpha = \begin{bmatrix} 0 & 0 & \dots & 0 & 1 \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & 0 \end{bmatrix}$$

(a)

$$H = \begin{bmatrix} I & I & I & \dots & I \\ I & \alpha & \alpha^2 & \dots & \alpha^{29} \\ I & \alpha^2 & \alpha^4 & \dots & \alpha^{58} \end{bmatrix}$$

(b)

Figure 1: H matrix of LDPC codes: (a) (3, 6) code, rate 0.5, p=211, length 1266; (b) (3, 30) code, rate 0.9, p=61, length 1830

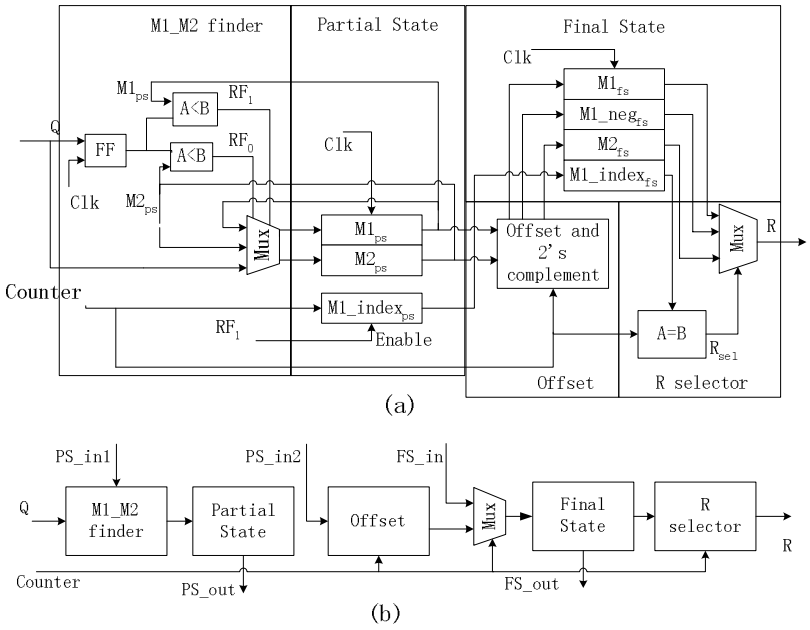

Figure 2: Check node processing unit, Q: Variable node message, R: Check node message. (a) simple scheme; (b) dynamic scheme.

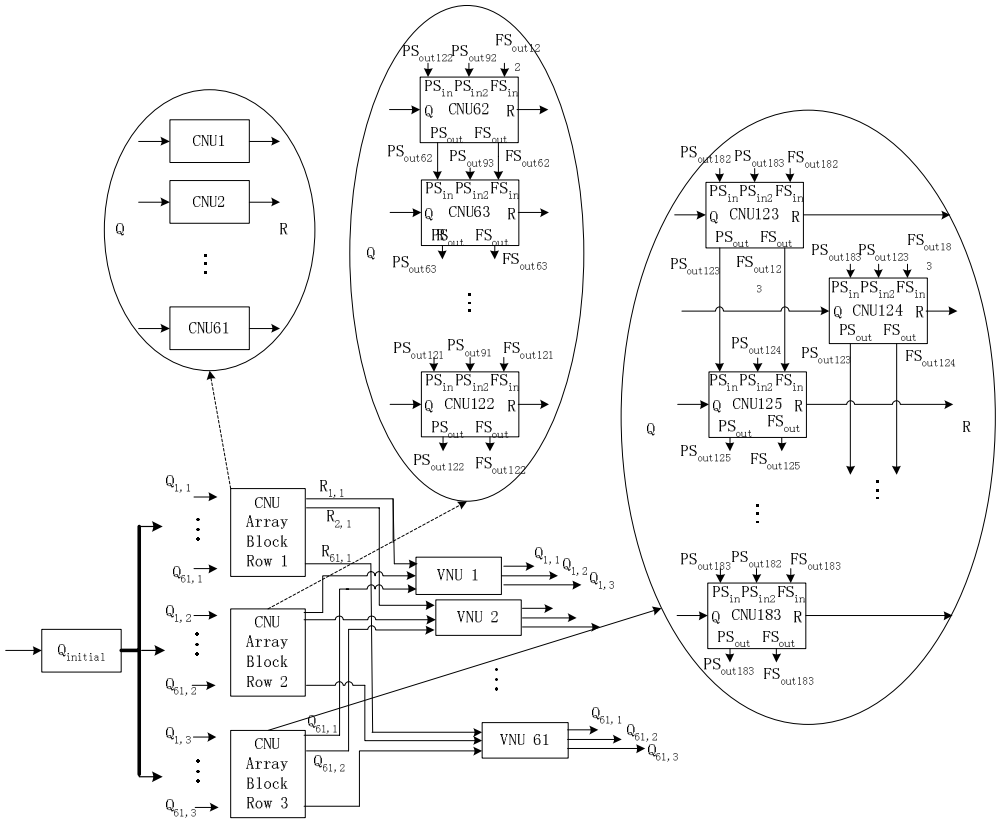

Figure 3: Architecture

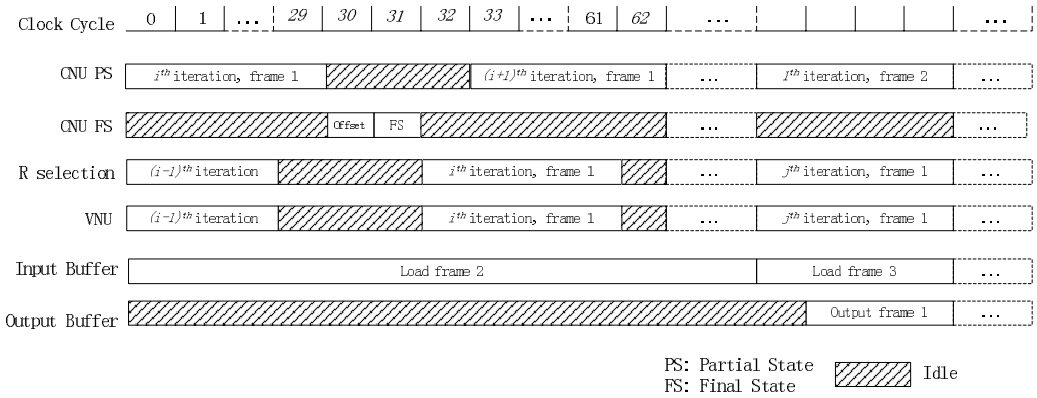

Figure 4: Pipeline

Table 1: FPGA results (Device: Xilinx 2v8000ff1152-5)

	No. Slices	No. 4-input LUT	No. Slice Flip-flops	Operating frequency(MHz)
CNU simple	45	70	53	236
CNU dynamic	55	102	55	193
CNU array block row 1	2599	4285	2980	196
CNU array block row 2,3	3464	6438	2967	
CNU array	9230	17143	8875	
VNU	27	42	25	169
VNU array	1623	2562	1525	169
Top	11695	19732	10733	153
Total number available	46592	93184	93184	

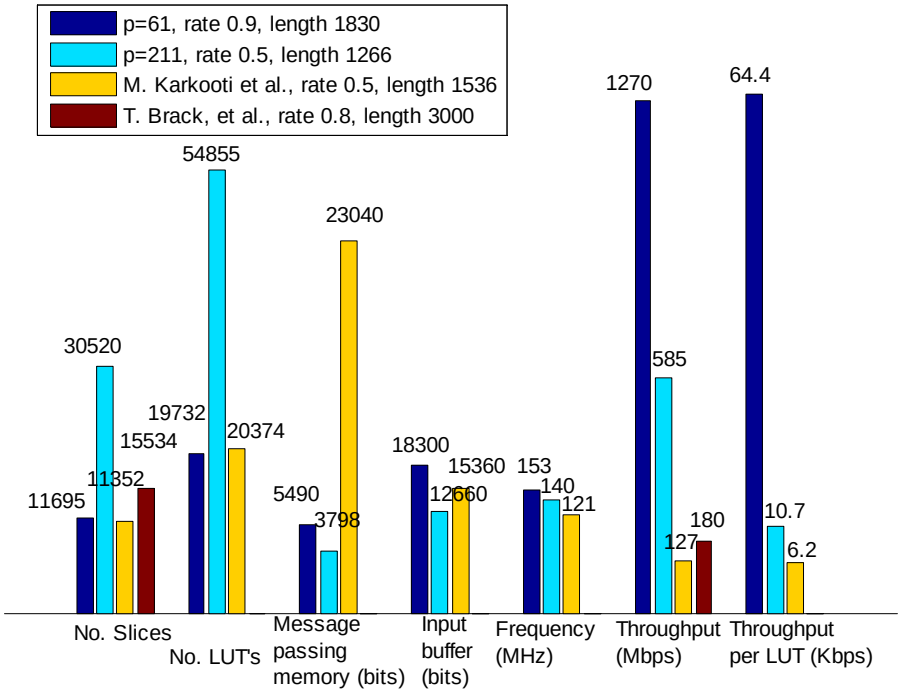

Figure 5: Results comparison