Multi-Split-Row Threshold Decoding Implementations for LDPC Codes

Tinoosh Mohsenin, Dean Truong and Bevan Baas
ECE Department, University of California, Davis

***Abstract*— The recently introduced Split-Row Threshold algorithm significantly improves the error performance when compared to the non-threshold Split-Row algorithm while requiring a very small increase in hardware complexity. The Multi-Split-Row Threshold decoding algorithm presented in this paper enables further reductions in routing complexity for greater throughput and smaller circuit area implementations. Several Multi-Split-Row Threshold decoder designs have been implemented in 65 nm CMOS and the impact of the different levels of partitioning on error performance, wire interconnect complexity, decoder area, and speed are investigated. The Split-Row-16 Threshold decoder occupies 3.8 mm^2, runs at 100 MHz, delivers a throughput of 13.8 Gbps at 15 iterations and is only 0.28 dB and 0.22 dB away from SPA and MinSum Normalized.**

I. Introduction

Low density parity check codes were first introduced by Gallager [1] in 1962. Due to their excellent error performance, LDPC codes have recently received significant attention and have been adopted by many recent communication standards such as 10 Gigabit Ethernet (10GBASE-T) [2], digital video broadcasting (DVB-S2) [3] and WiMAX(802.16e) [4]. However, due to the high interconnect complexity and high memory bandwidth requirements of existing decoding algorithms, implementing high throughput and energy efficient LDPC decoders remains a challenge.

A (W_c, W_r) (N, K) regular LDPC code is a block code defined by a binary parity check matrix with code length N, information length K, column weight W_c which is the number of ones per column, and row weight W_r which is the number of ones per row.

LDPC codes are commonly decoded by an iterative message passing algorithm consisting of two sequential operations (or two-phases): check node update (row processing) and variable node update (column processing). A simple illustration of this algorithm is shown in Fig. 1 (a). The parity check matrix defines the assignment (i.e. dependency) between which check nodes to which variable nodes, and vice versa. In row processing, check nodes receive messages (β) from their assigned variable nodes, perform parity check operations and send their results (α) back to the variable nodes. For column processing, the variable nodes update their estimates of the decoded bits using the messages (α) from their assigned check nodes and the channel data (λ) and send their results (β) back to the check nodes. This process continues iteratively until all errors are corrected or the number of iterations reaches a user defined maximum.

Sum-Product (SPA) [5], MinSum (MS) [6] and MinSum Normalized [7] are near-optimum decoding algorithms which are widely used in LDPC decoders and are considered as standard decoders. These algorithms propose different check node processing methods but use the same variable node update.

The major drawback of standard decoding algorithms is that they require variable nodes to send their messages to all their assigned check node for a single check node update. This communication leads to greater global interconnect complexity for large row weight codes ($W_r \geq 8$) and for large parity check matrices. Considering the fact that even if a decoding message is represented by a few bits (e.g. $6\ bits$), the interconnect complexity will sharply increase with every increment in row weight resulting in larger and slower circuits [8], [9]. Previous studies for reducing wire interconnect complexity are based on reformulating the message passing algorithm [10], [11], [12].

This paper introduces Multi-Split-Row Threshold decoding which significantly reduces wire interconnect complexity and considerably improves the error performance compared to non-threshold Multi-Split decoding [13]. The paper is organized as follows: Section II reviews the Split-Row and Split-Row Threshold decoding algorithms. Multi-Split-Row Threshold and its error performance results are presented in Section III. In Section IV, the results of several Multi-Split-Row Threshold decoder designs have been implemented highlighting the impact of the different levels of partitioning on wire interconnect complexity, decoder area, and decoder speed.

II. Split-Row and Split-Row Threshold Decoding Algorithms

Recall that a standard message passing two-phase algorithm consists of a check node update followed by a variable node update as shown in Fig. 1 (a). For Split-Row [14], [13] and Split-Row Threshold [15], [16], their decoders partition the check node processing into two or multiple nearly-independent partitions, where each check node processor simultaneously computes a new message α while only using minimal information from its adjacent partitions. Split-Row is illustrated in Fig. 1 (b) showing how the check node processing is partitioned into two blocks. A single bit of information ($Sign$) for each check node processor must be sent between partitions to improve error performance.

The major drawback of Split-Row is that it suffers from a 0.5–0.7 dB error performance loss, when compared to MinSum Normalized and SPA decoders. This degradation in performance is dependent on the number of check node partitions, and is the key obstacle in practical Multi-Split implementations. For MinSum Split-Row each partition has no information of the minimum value of the other partition, and a $Sign$ signal is sent to minimize the error due to incorrect sign information of the true check node output α.

The Split-Row Threshold algorithm mitigates the error caused by incorrect magnitude by providing a signal, $Threshold\_en$, which indicates whether a partition has a minimum less than a given threshold (T). This causes all check nodes to take the min of their own local minimum or T. Thus, any large deviations from the true minimum because of the partitioning is reduced, which then makes Multi-Split implementations feasible.

Figure 1 (c) shows the additional single bit signal ($Threshold\_en$) added to the original Split-Row architecture by the Split-Row Threshold algorithm. This signal allows the Split-Row to remain essentially unchanged while adding some extra logic and minimal wiring to improve error significantly [15].

Therefore, the Split-Row architectures reduce the communication between check node and variable node processors, which is the

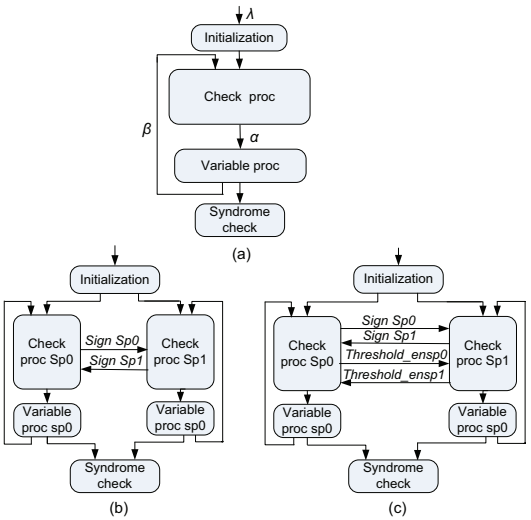

Fig. 1. Block diagram of (a) standard two-phase decoding (b) Split-Row (c) Split-Row Threshold block diagram

major cause of the interconnect complexity found in existing LDPC decoder implementations. In addition, the area of each check node processor, because of the reduction of inputs going into each check node, reduces. This is elaborated further in Section IV. However, note that the variable node operation in the SPA, MinSum, Split-Row and Split-Row Threshold algorithms are all identical. Thus, the logic of the variable node processor is left unchanged.

III. Multi-Split-Row Threshold

Multi-Split-Row Threshold is similar to *Multi-Split-Row* [13] where the parity check matrix is divided into Spn partitions and each one is processed simultaneously. However, unlike Multi-Split-Row, each partition of Multi-Split-Row Threshold sends the status of its local minimum against a threshold T to the next partition with a single wire, called $Threshold\_en$.

The block diagram of Multi-Split-Row Threshold decoding with *Spn* partitions, highlighting the $sign$ and $Threshold\_en$ passing signals, is shown in Fig. 2. These are the only wires passing between the partitions. In each partition, local minimums are generated and compared with a threshold T simultaneously. If the local minimum is smaller than T then the $Threshold\_en$ signal is asserted high. The magnitude of the check node outputs are finally computed using local minimums and the $Threshold\_en$ signal from neighboring partitions. If a local partition's minimums are larger than T, and at least one of the $Threshold\_en$ signals is high, then T is used to update its check node outputs. Otherwise, local minimums are used to update check node outputs.

The check node outputs are normalized with a correction factor which is in the range of 0.4-0.2 (depending on the level of split) before being sent to the variable nodes. The optimal values of correction factor and threshold T are found empirically through simulation [16].

Unlike Multi-Split-Row decoding whose error performance is not acceptable beyond a certain level of splitting, Multi-Split-Row Threshold partitioning can be arbitrary so long as there are two variable nodes per partition and the error performance loss is less than 0.3 dB. For example, Fig. 3 shows the error performance results for a (6,32) (2048,1723) LDPC code for SPA, MinSum Normalized, and MinSum Split-Row Threshold Improved with different levels of splitting and with optimal correction factors. The error performance simulations assume an additive white Gaussian noise channel with

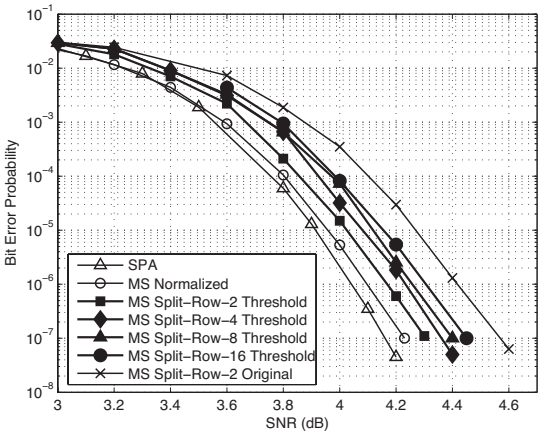

Fig. 3. BER comparison of Multi-Split-Row Threshold Improved with SPA and MinSum Normalized

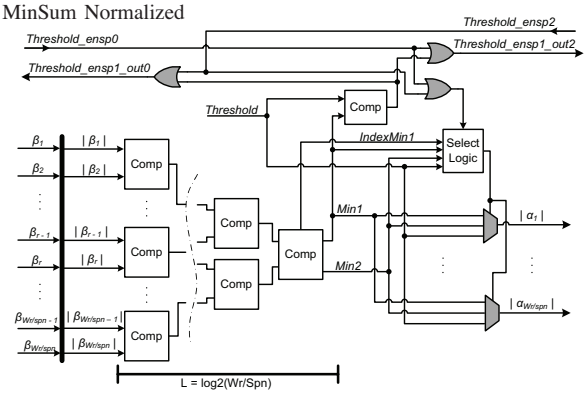

Fig. 4. Check node processor implementation block diagram for partition $Sp1$ using the MinSum Split-Row Threshold Improved method.

BPSK modulation. Simulations were made for 80 error blocks and with either a maximum of 15 decoding iterations or less when the decoder converged early. As the figure shows, MinSum Split-Row-2 Threshold is about 0.13 dB and 0.07 dB away from SPA and MinSum Normalized, respectively. From Split-2 Threshold through Split-4, Split-8 and Split-16 Threshold the error performance loss are less than 0.05 dB and total loss from Split-Row-16 Threshold to Split-Row-2 Threshold is 0.15 dB at $\mathrm{BER} = 10^{-7}$. Also shown in the plot is the Split-Row-2 original algorithm which is still 0.12 dB away from Split-Row-16 Threshold algorithm.

IV. Multi-Split-Row Threshold Hardware Implementation

In theory we can arbitrarily split a standard MinSum decoder into any Spn partitions. However, there are practical limitations to the level of partitioning that can be done in hardware. For a given W_r, the split-number, Spn must satisfy: $W_r/Spn \geq 2$, where W_r/Spn is the number of input β signals and output α signals to and from a check node processor, respectively. Having this, going below two represents splitting below the logic of a comparator.

A. Delay Analysis

Figure 4 shows an example $Spn1$ check node processor's magnitude calculation logic using Multi-Split-Row Threshold MinSum decoding. Given W_r/Spn, the critical path of a check node processor is essentially a function of the depth of comparator stages (L), which lies along a path from β input to α output. Thus, for $L = log_2(W_r/Spn)$, the critical path of a check node processor,

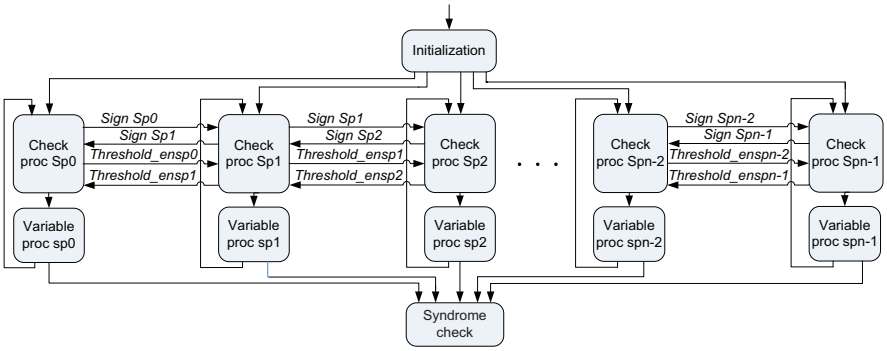

Fig. 2. Block diagram of Multi-Split-Row Threshold decoding with Spn partitions, highlighting sign and threshold passing signals between partitions.

$\Delta Spn_{CheckProc}$, is:

$$\Delta Spn_{CheckProc} = L \cdot \Delta_{Comp} + \Delta Th_{Logic} + \Delta_{Mux} \quad (1)$$

where Δ_{Comp}, ΔTh_{Logic}, and Δ_{Mux} are the worst case delays through a comparator, the threshold *select logic*, and a $Min1/Min2/Threshold$ mux, respectively.

When using original Multi-Split-Row, the critical path was most often the path along a partition's local logic and wire consisting of the longest path through the check node and variable node processors. Thus, the main advantage of the original Multi-Split-Row is the fact that the critical path decreases proportionally with the area of a single split. In this case the critical path for an original Multi-Split-Row, ΔSpn_{orig}, is:

$$\Delta Spn_{orig} = \Delta Spn_{CheckProc} + \Delta_{VarProc} + \Delta Spn_{wire} \quad (2)$$

where $\Delta Spn_{CheckProc}$ is given by Eq. 1, and $\Delta_{VarProc}, \Delta Spn_{wire}$, are the worst case delays through a variable node processor and wire, respectively. Note that the delay of a check node processor and wire delay is dependent on the number of splits (Spn) while the variable node processor's delay is largely unchanged.

When compared with the original Multi-Split-Row, the Multi-Split-Row Threshold's local delay is increased due to the select logic as depicted in Fig. 4. But notice that since the select logic is also dependent on the neighboring $Threshold\_en$ signals from $Sp0$ and $Sp2$, one possible critical path is from a $Threshold\_en$ signal that finally propagates to $Sp1$ and changes the mux select bits influencing check node output α. For large Spn, the Multi-Split-Row method becomes heavily dependent on the worst case propagation of $Threshold\_en$ signals to and from distant processors. Thus, the critical path for the Multi-Split-Row Threshold decoding is calculated as follows:

$$\Delta Spn_{thprop} = (L+1) \cdot \Delta_{Comp} + (Spn-2) \cdot \Delta Th_{prop} + \Delta_{Misc} + \Delta Th_{Logic} + \Delta_{Mux} + \Delta_{VarProc} \quad (3)$$

$$\Delta Spn_{threshold} = max\{\Delta Spn_{thprop}, \Delta Spn_{orig}\} \quad (4)$$

In summary, the final critical path of a Multi-Split-Row Threshold decoder, $\Delta Spn_{threshold}$, is the worst (i.e. largest) of the two possible critical paths: threshold propagate path (ΔSpn_{thprop}), or the original local check and variable processor path (ΔSpn_{orig}). The $(L+1) \cdot \Delta_{Comp}$ term represents the critical path required to generate a $Threshold\_en$ signal that is sent to other partitions; $(Spn-2) \cdot \Delta Th_{prop}$ is the total propagation delay of the $Threshold\_en$ signal across $Spn-2$ intermediate partitions, while Δ_{Misc} is the sum of the miscellaneous wire and gate delays along the path; $\Delta_{VarProc} + \Delta Th_{Logic} + \Delta_{Mux}$ is the total delay from the select logic to final variable node output β.

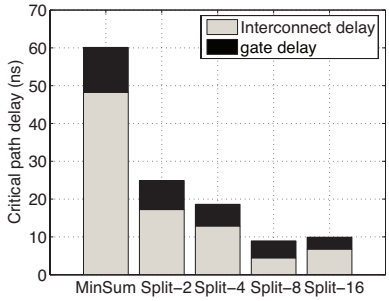

Fig. 6. Post route maximum delay in MinSum Normalized and Multi-Split-Row Threshold decoders, partitioned into interconnect and gate delay.

To illustrate the two possible critical paths, Fig. 5 highlights both worst case paths in bold. The propagation of the $Threshold\_en$ signal passing through $Spn-2$ partitions in one direction is shown next to the dotted lines labeled as Path 1. While Path 2 shows the original delay path through the check and variable processors. For small Spn, Path 2 (see Eq. 2) is dominant but, typically, Path 1 is the dominant critical path for Multi-Split-Row Threshold as quantified by Eq. 3. This path is indicative of one possible worst case propagation path beginning from the leftmost partition's ($Sp0$) check node processor, through $Spn-2$ wire delays, and finally to the variable node processor of the rightmost partition (Sp_{n-1}).

To further investigate the impact on the hardware implementation due to partitioning, several Multi-Split-Row Threshold full-parallel decoders are implemented for the (6,32) (2048,1723) LDPC code in 65 nm CMOS. Figure 6 summarizes the critical path of MinSum Normalized and various Multi-Split-Row Threshold implementations, partitioned into interconnect and gate delay. For $Spn > 4$, the threshold propagate path begins to dominate due to the overwhelming contribution of the $(Spn-2) \cdot \Delta Th_{prop}$ term in Eq. 3. So even though as the level of partitioning increases, where the delay inside a check node processor decreases, the propagation term negates this benefit. The figure shows that for MinSum Normalized and Split-2 Threshold the interconnect delay is largely dominant. This is due to the fact that as the number of check nodes, variable nodes and interconnection increase, the wire interconnect complexity increases nonlinearly which results in larger delays.

B. Area Analysis

Figure 7 shows the decoder area after synthesis and layout. Notice that for the MinSum Normalized decoder the area of the layout deviates significantly from the synthesized area. The reason is because of the inherent interdependence between the many number of check and variable nodes for large row weight LDPC codes, the number of timing critical wires that the automatic place and route tool must constrain becomes an exponentially challenging problem.

	Normalized MinSum	Split-2 Threshold MinSum	Split-4 Threshold MinSum	Split-8 Threshold MinSum	Split-16 Threshold MinSum
Logic Utilization (%)	25	40	85	95	**98**
Area (mm^2)	18.2	8.9	5	4.5	**3.8**
Avg. wire length per sub-block (μm)	142.5	88.1	59.1	27.1	**20.3**
Worst case speed (MHz)	17	40	53	**112**	**100**
Throughput @ 15 iterations (Gbps)	2.3	5.5	7.7	**15.2**	**13.8**

TABLE I

COMPARISON OF FULL-PARALLEL DECODERS IN 65 *nm* CMOS, FOR A (6,32) (2048,1723) CODE IMPLEMENTED USING MINSUM NORMALIZED AND MINSUM SPLIT-ROW THRESHOLD WITH DIFFERENT LEVELS OF SPLITTING.

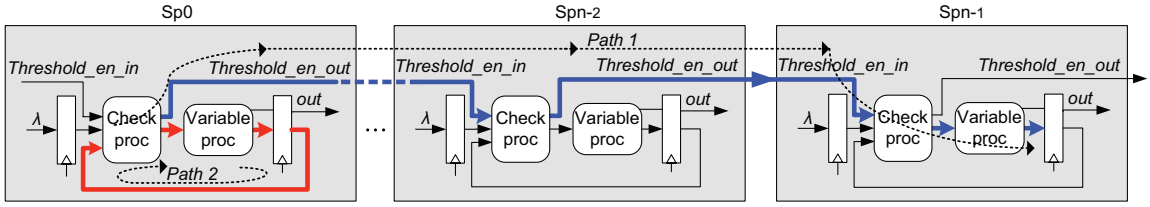

Fig. 5. Critical path and the pipeline diagram for Multi-Split-Row Threshold decoding method

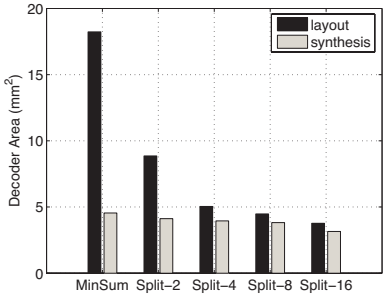

Fig. 7. The core area from synthesis and layout for MinSum Normalized and Multi-Split-Row Threshold decoders. The effect of wire interconnect complexity is shown by the area difference between layout and synthesis results for MinSum Normalized decoder.

Typically, the layout algorithm will try to spread standard cells apart because of the increased metal density between gate, drain and source connections of transistors and the upper metal layers. This results in a lower logic (i.e. silicon/transistor) utilization and a larger overall area.

For instance, wire complexity is exemplified by the very low utilization percentage (25%) and large average wire length (142.5 μm) of MinSum Normalized as shown in Table I. The table also summarizes the chip implementation results of Multi-Split-Row Threshold decoders. For a high Spn such as Split-16 the utilization is 98% which is about 4 times higher, while its average wire length is 7 times shorter than MinSum. It occupies 3.8 mm^2 which is 4.8 times smaller, it runs at 100 MHz and with 15 decoding iterations, it delivers 13.8 Gbps which is 6 times higher than MinSum. Thus, although Split-16 runs slightly slower than Split-8, it is still the smallest decoder; this represents the inflection point along the tradeoff curve between area and speed.

V. CONCLUSION

In this work we have analyzed the benefits and costs of Multi-Split implementations using the recently proposed Split-Row Threshold architecture. This has allowed us to achieve a better error performance for a chip that has a high level of partitioning when compared to the Split-Row original algorithm. Indeed, even Split-16 Threshold outperforms an original Split-2 by as much as 0.12 dB while being only 0.22 dB from MinSum Normalized. Apart from error performance, in Split-16 Threshold, the throughput improves 6 times and 2.5 times, while area improves 4.8 times and 2.4 times when compared to MinSum Normalized and Split-2 Threshold, respectively. The highest throughput is obtained by a Split-8 implementation with a throughput of at least 15.2 Gbps at 15 iterations. This shows that we can meet the demands of high speed applications while obtaining very low areas when compared to standard decoding methods.

REFERENCES

[1] R.G. Gallager, "Low-density parity check codes," *IRE Transaction Info.Theory*, vol. IT-8, pp. 21–28, Jan. 1962.

[2] "IEEE P802.3an, 10GBASE-T task force," `http://www.ieee802.org/3/an`.

[3] "T.T.S.I. digital video broadcasting (DVB) second generation framing structure for broadband satellite applications.," `http://www.dvb.org`.

[4] "IEEE 802.16e. air interface for fixed and mobile broadband wireless access systems. ieee p802.16e/d12 draft, oct 2005.," .

[5] D.J.MacKay, "Good error correcting codes based on very sparse matrices," *IEEE Transaction Info.Theory*, vol. 45, pp. 399–431, Mar. 1999.

[6] M. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *IEEE Transaction Communications*, vol. 47, pp. 673–680, May 1999.

[7] J. Chen and M. Fossorier, "Near optimum universal belief propagation based decoding of low-density parity check codes," *IEEE Transaction Communications*, vol. 50, pp. 406–414, Mar. 2002.

[8] A. Blanksby and C. J. Howland, "A 690-mW 1-Gb/s 1024-b, rate 1/2 low-density parity-check code decoder," *JSSC*, vol. 37, no. 3, pp. 404–412, Mar. 2002.

[9] M. Mansour and N.R. Shanbhag, "A 640-Mb/s 2048-bit programmable LDPC decoder chip," *JSSC*, vol. 41, pp. 684–698, Mar. 2006.

[10] S. Kang and I. Park, "Loosely coupled memorybased decoding architecture for low density parity check codes," *IEEE Transaction Cicuits and Systems I*, vol. 53, pp. 1045–1056, May 2006.

[11] A. Darabiha, A.C. Carusone, and F.R. Kschischang, "Block-interlaced LDPC decoders with reduced interconnect complexity," *IEEE Transaction Cicuits and Systems II*, vol. 55, pp. 74–78, Jan. 2008.

[12] Z.Cui and Z. Wang, "Efficient message passing architecture for high throughput LDPC decoder," in *ISCAS*, 2007, pp. 917–920.

[13] T. Mohsenin and B. Baas, "High-throughput LDPC decoders using a multiple Split-Row method," in *ICASSP*, 2007, vol. 2, pp. 13–16.

[14] T. Mohsenin and B. Baas, "Split-row: A reduced complexity, high throughput LDPC decoder architecture," in *ICCD*, Oct. 2006, pp. 13–16.

[15] T. Mohsenin, P. Urard, and B. Baas, "A thresholding algorithm for improved Split-Row decoding of LDPC codes," in *ACSSC*, Nov. 2008.

[16] T. Mohsenin, D. Truong, and B. Baas, "An improved Split-Row Threshold decoding algorithm for LDPC codes," in *ICC*, June 2009.