A Thresholding Algorithm for Improved Split-Row Decoding of LDPC Codes

Tinoosh Mohsenin[1], Pascal Urard[2] and Bevan Baas[1]

[1] ECE Department, University of California, Davis, [2] STMicroelectronics, Crolles Codex, France

***Abstract*— The recently proposed Split-Row decoding algorithm provides significant improvements in the throughput, hardware efficiency and energy efficiency when compared to existing soft decision decoding algorithms at the cost of some error performance loss. In this paper we propose Split-Row Threshold which outperforms the Split-Row algorithm while maintaining the same level of complexity. Simulation results show that the algorithm provides 0.2 dB coding gain over Split-Row decoding and is within 0.15-0.2 dB of SPA and MinSum normalized.**

I. Introduction

Low density parity check codes first introduced by Gallager [1] have recently received significant attention due to their being near the Shannon limit error correction performance and their inherently parallelizable decoder architectures. Many recent communication standards such as 10 Gigabit Ethernet (10GBASE-T) [2], digital video broadcasting (DVB-S2) [3] and WiMAX (802.16e) [4] have adopted LDPC codes. Implementing high throughput and energy efficient LDPC decoders remains a challenge largely due to the high interconnect complexity and high memory bandwidth requirements of existing decoding algorithms stemming from the irregular and global communication inherent in the codes.

The recently proposed Split-Row decoder [5], [6] partitions the row processing into two or multiple nearly-independent partitions, where each block is simultaneously processed using minimal information from an adjacent partition. The key idea of Split-Row is to reduce communication between row and column processors which has a major role in the interconnect complexity of existing LDPC decoding algorithms such as Sum Product (SPA) [7] and MinSum (MS) [8].

To illustrate further, Fig. 1 (a) shows the block diagram and a parity check matrix example highlighting the first row processing stage using MinSum decoding. The row processor output is shown by α, and the column processor output is shown by β which goes back to the row processor. The check node C_1 corresponding to the first row of the parity check matrix is also shown at the bottom, which connects to four variables nodes. In the Split-Row method which is shown in Fig. 1 (b), row processing is partitioned into two blocks, where each block is simultaneously processed almost independently. With this partitioning, the number of inputs sent to check node C_1 in each partition is reduced to half which results in less communication between row and column processors. In addition, each row processor's area is reduced because it processes only half the number of inputs.

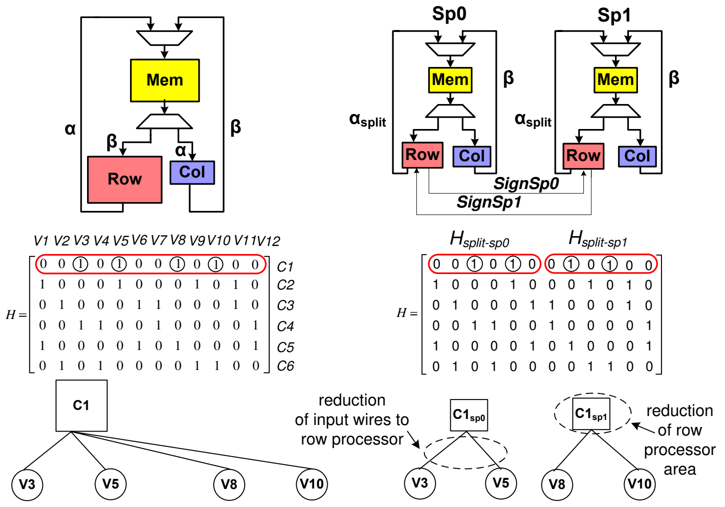
Mem
Row
Col
α
β
Sp0
Sp1
αsplit
SignSp0
SignSp1
V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12
C1 C2 C3 C4 C5 C6
H =
Hsplit-sp0
Hsplit-sp1
C1
C1sp0
C1sp1
V3
V5
V8
V10
reduction of input wires to row processor
reduction of row processor area

Fig. 1. Block diagram, a parity check matrix example highlighting the first row processing and the check node C_1 for (a) MinSum (b) MinSum Split-Row decoding methods.

Eq. 1 and Eq. 2 show the row processing equations of the MinSum normalized [9] and MinSum Split-Row decoding methods. In the following equations, β is the input to row processing and α is the output of row processing. In MinSum Split-Row the sign bit is computed using the sign bit of all messages across the whole row of the parity check matrix (because we pass the sign to the next partition). However, the magnitude of the α message in each partition is computed by finding the minimum among the messages within each partition. S_{MS} and S_{Split} are correction factors which normalize α values in MinSum and MinSum Split-Row to improve the error performance.

$$\alpha_{ij} = S_{MS} \times \prod_{j' \in V(i)\backslash j} sign(\beta_{ij'}) \times \min_{j' \in V(i)\backslash j} (|\beta_{ij'}|) \tag{1}$$

$$\alpha_{ijSplit} = S_{Split} \times \prod_{j' \in V(i)\backslash j} \text{sign}(\beta_{ij'}) \times \min_{j' \in V_{split}(i)\backslash j} (|\beta_{ij'}|) \tag{2}$$

.

The chip implementation results while using the MinSum Split-Row method show significant improvements in area, speed and energy dissipation over the MinSum decoder. The

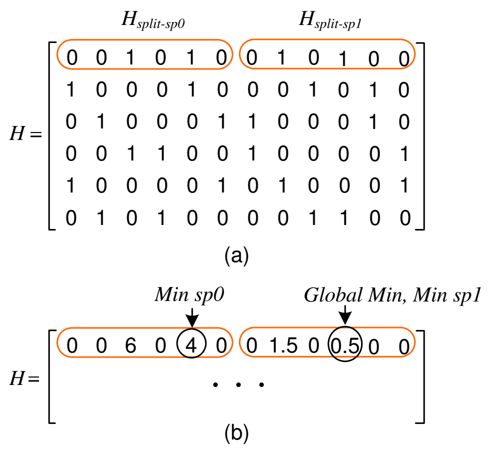

Fig. 2. The parity check matrix of a 12-bit LDPC code highlighting the first row processing using (a) original Split-Row decoding and (b) after being initialized with channel information, where $Min\ Sp0 \gg Min\ Sp1$

(2048,1723) LDPC decoder chip that was previously implemented using MinSum Split-Row with the same technology delivers 6.1 Gbps throughput and dissipates 79 pJ/bit which is 3.6 times faster and 1.8 times more energy efficient compared to a regular MinSum decoder.

The major drawback of Split-Row is that it suffers from a 0.4-0.7 dB error performance loss (depending on the number of row partitions) compared to MinSum and SPA decoders. The main reason for its error performance degradation is that in MinSum Split-Row each partition has no information about the minimum value of the other partition. Therefore, when the minimum value in one partition is much larger than the global minimum, the α values in that partition which are calculated by that minimum are all overestimated when compared to those in the other partition. This leads to a possible incorrect estimation of the bits which reside in that partition. Figure 2 (a) again shows the parity check matrix example and Fig. 2 (b) shows the values in the first row after being initialized with the received channel data. According to the MinSum Split-Row decoding method the local minimum in each partition is calculated independently and is used to update the α values. However, as shown in the figure, the local minimum in the left side, $Min\ sp0$, is eight times larger than the local minimum in the right side, $Min\ sp1$, which is also the global minimum of the entire row. This results to an overestimation of α values for bit 3 (V_3) and bit 5 (V_5) which can possibly cause an incorrect decision for these two bits.

II. MinSum Split-Row Threshold

A threshold decoding method is proposed for Split-Row to compensate for the difference between minimums among the partitions and therefore improve the error performance with negligible additional hardware. The basic idea is that each partition sends a signal to the next partition if its own local minimum is smaller than a threshold (T). Thus, the other partition is notified if there exists a minimum smaller than the threshold. The algorithm is explained below.

Similar to the MinSum decoder, the first and second minimums ($Min1$ and $Min2$) in each partition are computed

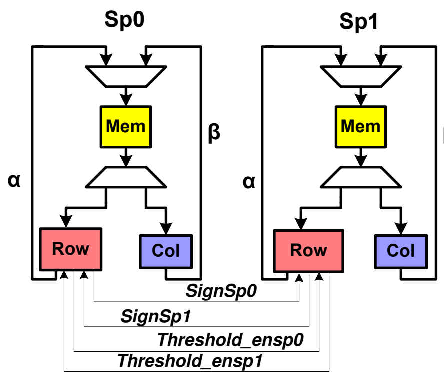

Fig. 3. Split-Row Threshold block diagram

locally. The proposed algorithm checks if $Min1$ is less than threshold T, then both $Min1$ and $Min2$ are used to update α values. Additionally, a threshold signal ($Threshold\_en$) which goes to the next partition is asserted high, indicating that the minimum in this partition is smaller than threshold T.

On the other hand, if the local $Min1$ is larger than the threshold but $Threshold\_en$ from the neighboring partition is asserted, it uses threshold T to update α values in that partition.

The last condition is when the local $Min1$ is larger than threshold T and $Threshold\_en$ is not asserted, which indicates that $Min1$ in the other partition is also larger than the threshold T. In this case the local $Min1$ and $Min2$ are used to calculate α values. Figure 3 shows the block diagram of Split-Row Threshold where $Threshold\_en$ signals are passed between partitions. These two wires alongside the two sign wires are the only wires passed between the partitions.

The MinSum Split-Row Threshold algorithm for the row processing in the $Sp0$ partition is summarized as,

```
If ($Min1 \leq Threshold$)
  $Threshold\_ensp0 = 1$,
  if ($|\beta_j|$ =$Min1$)
      $|\alpha_j|$=$Min2$
  else
      $|\alpha_j|$=$Min1$
else if ($Threshold\_ensp1 = 1$)
  $|\alpha_j|$=$Threshold$
else
  if ($|\beta_j|$ =$Min1$)
      $|\alpha_j|$=$Min2$
  else
      $|\alpha_j|$=$Min1$.
```

Column processing in MinSum, MinSum Split-Row, and MinSum Split-Row Threshold are all identical.

III. The Determination of Threshold and Error Performance Results

The error performance simulations presented here assume an additive white Gaussian noise channel with BPSK modulation. Simulations were made for 80 error blocks and with

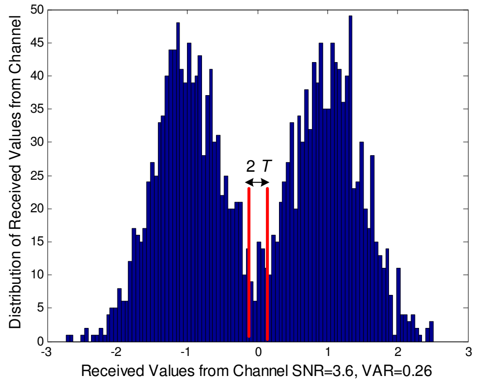

Fig. 4. The distribution of received channel values for a 2048-bit code at $SNR = 3.6\ dB$, and noise variance $VAR = 0.29$. The likely values of threshold T at which the algorithm performs best is with received values of approximately 0.1–0.2 .

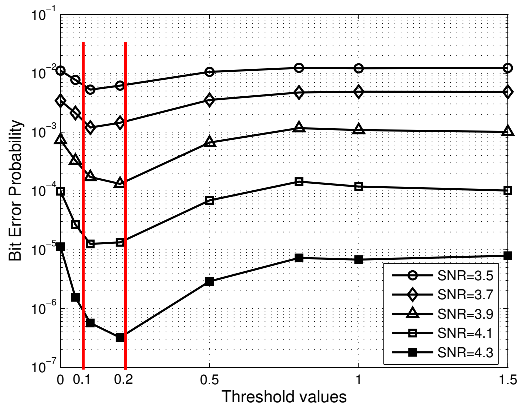

Fig. 5. Determination of Threshold T for a (6,32) (2048,1723) code using MinSum Split-Row Threshold with different SNRs. The red vertical lines signify the optimal threshold T interval.

either a maximum of 15 decoding iterations or earlier when the decoder converged.

Figure 4 shows the distribution of received values from a channel at $SNR = 3.6\ dB$ and noise variance $VAR = 0.29$. The likely location of the threshold T at which the algorithm performs is shown in the figure. To further show the effect of choosing the optimal threshold value, Fig. 5 plots the error performance of a (6,32) (2048,1723) LDPC code versus threshold values for different SNRs. There are two limits for the threshold value. If threshold T is zero, then local minimums are always larger than the threshold, meaning that local minimums are used to update α. Thus, the algorithm converges to the original MinSum Split-Row. On the other hand, if threshold T is very large, local minimums are always smaller than the threshold. This again results in using local minimums to calculate α and therefore, the algorithm converges to the original MinSum Split-Row. As shown in the figure the optimal value of the threshold for this code, within the SNR ranges of 3.5-4.3 dB, is in the interval of 0.1-0.2. Thus, one of the benefits of the MinSum Split-Row Threshold method is that the threshold T is not dependent on the SNR and channel statistics.

Figure 6 shows the error performance results for a (6,32) (2048,1723) LDPC code for SPA, MinSum normalized, MinSum Split-Row original and MinSum Split-Row Threshold with different threshold values (T). The simulation results show that the optimal correction factor S for MinSum is 0.5, for MinSum Split-Row original is 0.30, and for MinSum Split-Row Threshold is 0.35. The threshold is chosen to be fixed over different SNRs and different decoding iterations. As shown in the figure the coding gain of MinSum Split-Row Threshold, with $T = 0.2$, over the original is 0.2 dB, and it is only 0.15 dB away from MinSum normalized.

IV. Row Processor Implementation with Split Threshold Method

The implementation of the row processor in MinSum Split-Row Threshold is similar to that of the MinSum decoder while using half of the total number of inputs (β) and with some small additional hardware. The column processor implementation remains the same as in the MinSum

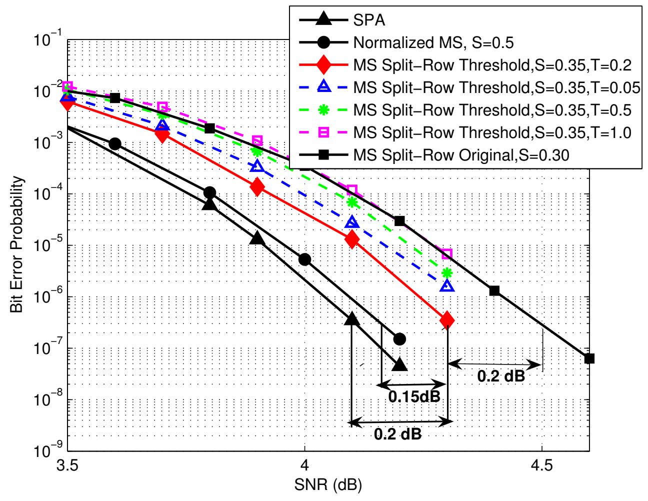

Fig. 6. BER performance of (6,32) (2048,1723) code using MinSum normalized, MinSum Split-Row original and MinSum Split-Row Threshold with different threshold values T.

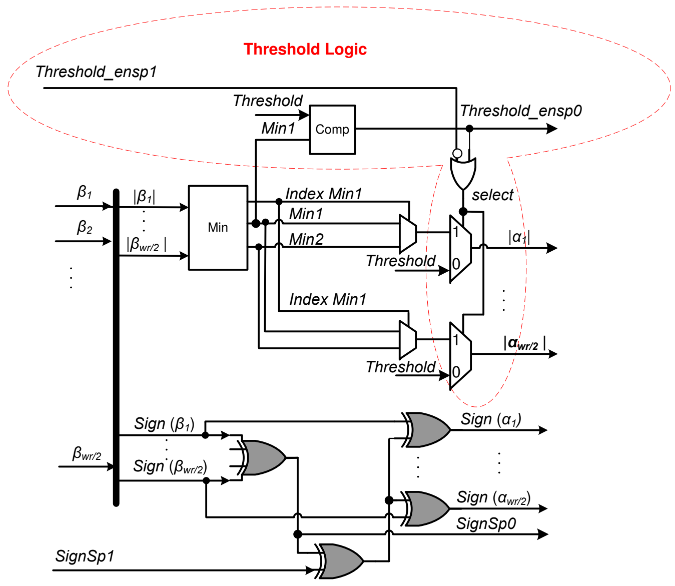

Fig. 7. The row processor implementation block diagram using MinSum Split-Row Threshold method, the threshold logic is shown within the dashed red line.

and MinSum Split-Row. Figure 7 shows the row processor $sp0$ implementation block diagram using the MinSum Split Threshold method. The magnitude update of α is shown on the top and the sign calculation is shown at the bottom. Similar to MinSum Split-Row the sign bit calculated from partition $sp1$ is passed to $sp0$ to correctly calculate the global sign bit according to row processing equation Eq. 2. The threshold logic implementation is shown within the dashed line which consists of one comparator, an OR gate and $W_r/2$ muxes. Assuming the row weight of the parity check matrix is W_r, there are $W_r/2$ inputs (β) to each row processor. Similar to MinSum decoding, the first minimum ($Min1$) and the second minimum ($Min2$) are found alongside the signal ($IndexMin1$), which indicates whether Min1 or Min2 is chosen for this particular index α. Min1 is then compared with the Threshold to generate $Threshold\_ensp0$. This does not add extra delay because Min2 is generated one comparator delay after Min1. $Threshold\_ensp0$ and $Threshold\_ensp1$ are combined together, based on the algorithm, to generate the $Select = \overline{Threshold\_ensp1} + Threshold\_ensp0$ signal. The α values are finally produced using the $Select$ and $IndexMin1$ values.

	MinSum Split-Row Original	MinSum Split-Row Threshold
Row processor area (μm^2)	766	805
Gate count	296	317

TABLE I

THE SYNTHESIS RESULTS FOR ROW PROCESSOR USING SPLIT-ROW ORIGINAL AND THRESHOLD DECODING METHODS IN 65 *nm* CMOS

Table I summarizes the synthesis results for the row processor implemented with the Split-Row original and Threshold decoding methods in 65 nm CMOS technology using Synopsys Design Compiler. As shown in the table the row processor area in MinSum Split-Row Threshold is only 5% larger than the row processor in MinSum Split-Row. The (2048,1723) LDPC decoder chip that was previously implemented using MinSum Split-Row with the same technology delivers 6.1 Gbps throughput and dissipates 79 pJ/bit which is 3.6 times faster and 1.8 times more energy efficient compared to a regular MinSum decoder. Because the MinSum Split-Row Threshold only requires one additional global wire to pass the $Threshold\_en$, very few additional gates to implement the threshold logic, and one additional mux delay into the critical path, the performance, area, and energy of the MinSum Split-Row Threshold will be comparable to that of the MinSum Split-Row original. .

V. CONCLUSION

The threshold decoding method is proposed to facilitate hardware implementations capable of: high-throughput, high hardware efficiency, and high energy efficiency. Simulation results show that the Split-Row Threshold outperforms the Split-Row algorithm for 0.2 dB while maintaining the same level of complexity. Our simulation results show that for a given LDPC code keeping threshold T constant at any SNR does not cause any error performance degradation.

VI. ACKNOWLEDGMENTS

The authors gratefully acknowledge support from STMicroelectronics, Intel, UC Micro, NSF Grant 0430090 and CAREER Award 0546907, SRC GRC Grant 1598 and CSR Grant 1659, Intellasys, S Machines, and a UCD Faculty Research Grant; LDPC codes and assistance from Shu Lin and Lan Lan; and thank Dean Truong, Jean-Pierre Schoellkopf and Patrick Cogez.

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] T. Mohsenin and B. Baas, "Split-row: A reduced complexity, high throughput LDPC decoder architecture," in *ICCD*, Oct. 2006, pp. 13–16.
[6] T. Mohsenin and B. Baas, "High-throughput LDPC decoders using a multiple split-row method," in *ICASSP*, 2007, vol. 2, pp. 13–16.
[7] D.J.MacKay, "Good error correcting codes based on very sparse matrices," *IEEE Transaction Info.Theory*, vol. 45, pp. 399–431, Mar. 1999.
[8] 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.
[9] J. Chen, A. Dholakia, E. Eleftheriou, and M. Fossorier, "Reduced-complexity decoding of LDPC codes," *IEEE Transaction Communications*, vol. 53, pp. 1288–1299, Aug. 2005.