VLSI Implementation of a Soft Bit-Flipping Decoder for PG-LDPC Codes

Junho Cho, Jonghong Kim, Hyunwoo Ji, and Wonyong Sung
School of Electrical Engineering and Computer Science
Seoul National University
Gwanak-gu, Seoul, 151-744 Korea
E-mail: {jhkim, juno, hwji}@dsp.snu.ac.kr; wysung@snu.ac.kr

Abstract—Implementation of high throughput VLSI chips for low-density parity-check codes has been considered very difficult especially when the row or column weight of the code is high. In this paper, a projective-geometry (PG) LDPC code is implemented in VLSI employing the proposed soft bit flipping (SBF) algorithm. The SBF algorithm requires only simple interconnections, but its error correcting performance is close to the sum-product algorithm (SPA). Parallel processing architecture is employed for increasing the throughput. With the (1057, 813) PG-LDPC code, the implemented 4-bit SBF decoder consumes only a small area of 2.5mm² while providing 6.5Gbps and good performance close to the floating-point SPA by 0.6dB at the frame error rate of 10⁻⁴.

I. INTRODUCTION

Low-density parity-check (LDPC) codes are known to show near Shannon-limit performance when decoded with a belief-propagation based iterative algorithm [1-3]. There have been intensive works to achieve both good error performance and low complexity based on LDPC codes. Among various decoding algorithms, the soft-decision based sum-product algorithm (SPA) [3-4] and the hard-decision based bit-flipping (BF) algorithm [1] are the extreme counterparts, which have lead many variants to overcome their weaknesses. The SPA shows the best error performance but is not easily implemented in hardware due to high computational complexity. On the contrary, the BF algorithm requires only small amount of simple computations while the error performance is far behind the SPA.

The soft bit-flipping (SBF) algorithm [5] is one of the variants which mediate between SPA and BF algorithm. It has been developed in consideration of the low-cost implementation. The SBF algorithm follows the basic structure of BF algorithm to reduce the interconnection complexity and the amount of computations, while it applies reliability estimation which imitates the SPA to bridge the error performance gap. Since the messages transferred from a variable/check node to all the connected check/variable nodes is singular, the computational complexity of the SBF algorithm grows much more slowly than the SPA when the column or row weight of the parity-check matrix increases.

Therefore, the LDPC codes which show good error performance but have a high column or row weight, such as projective geometry (PG)-LDPC codes [6], can be practically implemented using the SBF algorithm.

In this paper, the VLSI circuits of the serial and parallel SBF decoders are implemented using a cyclic PG-LDPC code. In order to reduce the area redundancy appeared in the parallel SBF decoder architecture, a method to use a common processing unit is proposed instead of using two different node processing units. Since the cyclic SBF decoder requires very little area overhead for parallelization, a parallel decoder with a large parallel factor can be easily implemented for high throughput. Pipelining scheme is also employed to further increase the decoding throughput. With the (1057, 813) PG-LDPC code, the developed SBF decoder achieves a throughput of 6.5Gbps with only 2.5mm² area when implemented in a 0.18um process, while providing 0.6 dB close performance to the floating-point SPA at the frame error rate of 10⁻⁴. Note that the (1057, 813) PG-LDPC code can hardly be implemented in SPA due to the large column and row weights of 33.

This paper is organized as follows. In Section II, the SBF algorithm is introduced. Architectures for SBF decoding and the optimization methods are described in Section III. VLSI implementation results are presented in Section IV, and the concluding remarks are made in Section V.

II. SOFT BIT-FLIPPING ALGORITHM

Assume we have a regular \((N, K)\) LDPC codes defined by \(M \times N\) parity-check matrix \(H = [h_{m,n}]\). Let \(B(m) = \{n| h_{m,n} = 1\}\) be the set of variable nodes that participate in the \(m\)-th check node. The information is transmitted over the AWGN channel with BPSK modulation. Let us denote the received word and the hard-decision vector of it as \(y = (y_1, y_2, ..., y_N)\) and \(z = (z_1, z_2, ..., z_N)\), respectively. Then the weight assigned to the \(m\)-th check node is computed by

\[
w_m = \sum_{i \in B(m)} |y_i|, \quad m \in [1, M].
\]
If the variable and check nodes are quantized into q bits, there are \(2^q - 1\) quantization thresholds for each node; i.e., \(\delta_1^i < \delta_2^i < \cdots < \delta_{2^q-1}^i\) for the variable node, and \(\delta_1^c < \delta_2^c < \cdots < \delta_{2^q-1}^c\) for the check node. The quantized values are represented by the sign-magnitude method except \(\pm 0\). The flipping strength is decided by the three thresholds of \(\delta_1^f < \delta_2^f < \delta_3^f\). Then the soft bit-flipping (SBF) algorithm performs decoding simply by the following procedure:

**Initial:** Set the iteration number to 0.

**Step 1:** Calculate the syndrome vector \(s^k = z^k H^T\). If \(s^k\) is a zero vector, output \(z^k\) as the decoded codeword and finish the decoding.

**Step 2:** Quantize the variable node \(y_n^k\) \((n \in [1, N])\) into q bits according to the quantization level \(\delta_i^v\) \((i \in [1, 2^q - 1])\). Compute the weight of the \(m\)-th check node \(w_{mn}^k\) \((m \in [1, M])\), and then quantize it using \(\delta_i^c\) \((i \in [1, 2^q - 1])\).

**Step 3:** For every variable node, compute the flipping function defined by

\[
e_n^k = \sum_{m \in A(n)} (2s_{mn}^k - 1)w_{mn}^k - \alpha|y_n^k|.
\]

**Step 4:** Update each variable node with the soft bit-flipping as follows:

i) (Strong flip) When \(e_n^k \in (\delta_3^f, \infty)\),

\[
y_n^{k+1} = \text{sgn}(y_n^k - 2.5) \cdot \max(1, |y_n^k| - 2),
\]

ii) (Weak flip) When \(e_n^k \in (\delta_2^f, \delta_3^f)\),

\[
y_n^{k+1} = \text{sgn}(y_n^k - 1.5) \cdot \max(1, |y_n^k| - 1),
\]

iii) (Maintaining) When \(e_n^k \in (\delta_1^f, \delta_2^f]\),

\[
y_n^{k+1} = \text{sgn}(y_n^k) \cdot |y_n^k|,
\]

iv) (Strengthening) When \(e_n^k \in (-\infty, \delta_2^f]\),

\[
y_n^{k+1} = \text{sgn}(y_n^k) \cdot \min(2^q, |y_n^k| + 1).
\]

If there occurs no bit-flipping in any variable node, adjust the flipping thresholds as \(\delta_2^f = \max_n e_n^k\) and \(\delta_3^f = \delta_2^f - \beta\).

**Step 5:** Increase the iteration number \(k = k + 1\). If \(k\) becomes larger than the predefined iteration limit \(K_{MAX}\), decoding is finished with the decoding failure message, otherwise go to Step 1.

As stated in the above procedure, the SBF algorithm basically operates in a similar structure of the conventional bit-flipping algorithm devised by Gallager in early 1960’s. However the SBF-based decoding employs not only the hard-decision bits of the symbols and parity-checks but also the reliability of them using a few bits of precision. The SBF algorithm efficiently narrows the performance gap to the ideal SPA decoding at the expense of little computational overhead. The error rate performance curves of various decoding algorithms for the \((1057, 813)\) PG-LDPC code are depicted in Fig. 1, where the performance of SPA, weighted bit-flipping (WBF) algorithm [6] and improved modified WBF (IMWBF) algorithm [7] are also plotted for comparison. Note that only the performance of SBF algorithm is the result of fixed-point computations, whereas the other curves are that of floating-point computation.

III. ARCHITECTURES FOR SOFT BIT-FLIPPING DECODER

This section describes architectural alternatives of the 4-bit soft bit-flipping decoder. For the baseline parallel architecture, some methods to reduce the hardware area and to increase the throughput are presented in detail.

A. Serial Architecture

A straightforward implementation of the SBF decoder consists of two \(n\)-bit shift registers for storing the variable and check nodes, one variable node processing unit (VPU) and one check node processing unit (CPU). Fig. 2 shows the overall architecture. Once the decoding starts, all the received signals are stored in the variable nodes. Then the VPU computes a parity-check from the connected 33 variable nodes, and store it to the corresponding check node. This VPU operation is continued until all the check nodes are updated, while all the registers are shifted by one stage every cycle. After finishing the check node update, the CPU adds up the connected 33 check nodes and compares the result with the flipping thresholds, thereby generating a flipping strength for the output variable node. Each variable node is updated by adding the flipping strength if the current variable node is negative, or by subtracting it otherwise. The overflow caused by the flipping is saturated at the last stage.

Detailed structures of the processing units are illustrated in Fig. 3. Let us denote the \(n\)-th 4-bit variable node connected to the VPU as \([s_n; m_n]\), where \(s_n\) is the inverted 1-bit hard-decision value, and \(m_n\) is the 3-bit precision reliability of this hard-decision value. Then the VPU generates a 4-bit parity-check with the sign-magnitude representation; the most significant 1 bit for the inverted hard-decision parity-check is obtained by the modulo-2 sum \(\sum_{n \in [0, 32]} s_n\), and the remaining 3 bits are calculated by the ordinary sum \(\sum_{n \in [0, 32]} m_n\) followed by the quantization with threshold \(\delta_i^q\)’s, as shown in Fig. 3(a). The result is then formatted to the 4-bit 2’s complement representation before being stored in the check node. Using the 2’s complement format is efficient.
for the check nodes, whereas the sign-magnitude format is advantageous for the variable nodes. It is because the CPU requires the signed summation of check nodes, while the VPU sums up the absolute value of variable nodes. The format conversion unit is attached at the last stage of the VPU rather than the first stage of CPU, since the critical path is on the CPU. As a result of VPU operations, the check nodes rather than the first stage of CPU, since the critical path is on the VPU, as well as adding the check nodes as they are, as illustrated in Fig. 3(b). In this way, the complexity of conditional additions can be quite alleviated. The lower part of Fig. 3(b) shows the hardware for threshold adaptation. If no flipping operations occur in a variable node update phase, even though there is at least one unsatisfied parity-check, $\delta_2^f$ is modified to the maximum reliability value which has been detected in the last variable node update phase, and also $\delta_2^f$ is modified as $\delta_2^f = \delta_3^f - \beta$. Both in the VPU and CPU, the Wallace tree and a carry look-ahead (CLA) adder are employed to minimize the critical path delay caused by the large-input addition.

**B. Parallel Architecture**

Denoting the parallel factor by $p$, a parallel SBF decoder uses $p$ times many VPUs and CPUs, while using the same number of node registers, threshold adaptation modules, and interconnection networks as a serial decoder. The number of shift registers storing the variable and check nodes is not changed no matter how much $p$ increases. Therefore, although the shift registers occupy the dominant portion of total cell area in a fully serial architecture ($p = 1$), the portion of VPUs and CPUs becomes larger as $p$ increases. Also, there is only one threshold adaptation module which operates by the DETECT_MAX and DETECT_FLIP signals delivered from the $p$ CPUs.

1) **Shared Node Processing Unit (SNPU) Architecture:** A way of sharing two node processing units is studied to reduce the cell area of the parallel SBF decoder. In the SBF decoding procedure, VPU and CPU operate one after another. Furthermore, because both the modulo-2 summation required for computing the inverted hard-decision parity-checks in VPU and the ordinary summation required for compensating the range mismatch in CPU can be implemented by 1-bit XOR gates, it is possible to share these hardware units. Moreover, the 33-input 3-bit or 4-bit ordinary adders of VPU and CPU can be shared also. The SNPU architecture is shown in Fig. 4, which reduces the cell area of the parallel decoder especially when the parallel factor is large.

2) **Pipelining:** On the SNPU architecture, pipelining technique is applied to increase clock frequency. The determined number of pipeline stages is two because the critical path delay (7.97ns) of the whole decoder circuit is approximately two times larger than the path delay (3.12ns) of the finite state control logic. The pipeline registers are then placed between the Wallace tree and the CLA in the SNPU, which is almost the middle point of the critical path. The path delay of the front path is 4.10ns, and that of the remaining path is 3.87ns. Pipelining is required also in the threshold adaptation module, for which the path delay increases at the rate of $O(\log p)$. This is because the threshold adaptation module searches for a maximum value among the $p$ inputs delivered from $p$ SNPUs. In this case, as many pipeline
stages can be used as required, since the threshold adaptation will be used after \(N/p\) clock cycles, which is consumed for the variable node update. Fig. 5 depicts the case when \(p = 32\) and there are 3 pipeline stages.

### Table I. Implementation Results

<table>
<thead>
<tr>
<th>Area ((\mu m^2))</th>
<th>Serial (p = 16)</th>
<th>(p = 32)</th>
<th>(p = 64)</th>
<th>Serial (p = 16)</th>
<th>(p = 32)</th>
<th>(p = 64)</th>
<th>Serial (p = 16)</th>
<th>(p = 32)</th>
<th>(p = 64)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPU</td>
<td>27,775</td>
<td>1,778,901</td>
<td>1,886,415</td>
<td>499,637</td>
<td>981,691</td>
<td>1,499,648</td>
<td>2,509,915</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU</td>
<td>891,153</td>
<td>1,507,467</td>
<td>2,487,762</td>
<td>450,474</td>
<td>460,054</td>
<td>470,273</td>
<td>470,273</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subtotal</td>
<td>8,337,090</td>
<td>3,289,368</td>
<td>5,004,177</td>
<td>550,115</td>
<td>1,037,445</td>
<td>1,579,921</td>
<td>1,979,186</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>8,414,827</td>
<td>3,397,256</td>
<td>5,112,349</td>
<td>550,115</td>
<td>1,037,445</td>
<td>1,579,921</td>
<td>1,979,186</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

V. CONCLUDING REMARKS

VLSI circuits for SBF decoding are synthesized for the (1057, 813) PG-LDPC code. The proposed SNUA architecture combined with the pipelining technique achieves a throughput of 6.5Gbps with only 2.5mm² cell area, while showing 0.6dB close performance to the SPA. This is the first implementation result of a high weight PG-LDPC code that shows such close performance to SPA, since the other decoding algorithms are almost infeasible to implement due to the complexity problem.

### REFERENCES