FPGA Implementation of Rate-Compatible QC-LDPC Code Decoder

Anton Blad, Oscar Gustafsson
Electronics Systems, Linköping University
SE-581 83 Linköping, Sweden
Email: antonb@isy.liu.se, oscarg@isy.liu.se

Abstract—The use of rate-compatible error correcting codes offers several advantages as compared to the use of fixed-rate codes: a smooth adaptation to the channel conditions, the possibility of incremental Hybrid ARQ schemes, as well as simplified code representations in the encoder and decoder. In this paper, the implementation of a decoder for rate-compatible quasi-cyclic LDPC codes is considered. The decoder uses check node merging to increase the convergence speed of the algorithm. Check node merging allows the decoder to achieve the same performance with a significantly lower number of iterations, thereby increasing the throughput.

The feasibility of a check node merging decoder is investigated for codes from IEEE 802.16e and IEEE 802.11n. The faster convergence rate of the check node merging algorithm allows the decoder to be implemented using lower parallelization factors, thereby reducing the logic complexity. The designs have been synthesized to an Altera Cyclone II FPGA, and results show significant increases in throughput at high SNR.

I. INTRODUCTION

Low-density parity-check (LDPC) codes [1], [2] have emerged as an error-correcting scheme for many wired and wireless technologies. Communication standards that have adopted LDPC codes as either mandatory or optional channel coding schemes include DVB-x2, IEEE 802.11n, IEEE 802.16e, and IEEE 802.3an. Typically, individual codes with different rates are employed for different channel conditions. However, an alternative is the use rate-compatible codes [3], where related codes with a wide range of rates are used. Commonly, these are obtained from a mother code through puncturing or extension. Rate-compatibility offers several benefits:

• Better adaptation to channel conditions is allowed by the large number of available rates.
• Incremental redundancy Hybrid ARQ significantly reduces the amount of data that has to be retransmitted in the case of communication errors.
• The similarity of codes of different rates may allow simpler configuration of the encoder and decoder.

Punctured LDPC codes can be decoded by a fixed-rate belief propagation decoder by initializing the LLR values of the punctured nodes to zero. However, it was shown in [4] that a significant reduction of the convergence time of the algorithm is achievable by merging the check nodes involving the punctured nodes. Later, an architecture for the check merging decoding algorithm was outlined in [5].

Fig. 1. Example of check-merging for a degree-2 punctured variable node.

In this paper, the suitability of check merging for QC-LDPC codes from the 802.16e and 802.11n standards are evaluated. The original architecture from [6] and the proposed one from [5] have been synthesized to an Altera Cyclone II FPGA in order to determine the overhead required by enabling the check merging. In addition to efficiently handling rate-compatible LDPC codes, the proposed architecture also gives a small throughput increase by eliminating the wait states present in the original architecture.

The remainder of the paper is organized as follows. In Sec. II, rate-compatible LDPC codes and check node merging are reviewed. Section III describes the proposed architecture for the check node merging algorithm. In Sec. IV, parameters and performance of the proposed architecture for codes from the 802.16e and 802.11n standards are presented, and synthesis results of are included. Finally, conclusions are in Sec. V.

II. RATE-COMPATIBLE LDPC CODES

A class of rate-compatible codes is defined as a class of codes where the codewords in the higher-rate codes can be obtained by removing bits from the codewords of lower-rate codes [3]. Thus the information content is the same in the different codes, but the numbers of parity bits differ. To simplify the presentation, rate-compatible LDPC codes obtained through puncturing are considered, although the results are equally valid with codes obtained through other means.

Check node merging is shown conceptually in the three steps in Fig. 1, where a degree-2 variable node is punctured and its check node neighbors are merged:

1) Define punctured variable nodes and associated purged check nodes.
2) Merge the edges of the purged check nodes into the other neighbors of its punctured variable node.
3) Remove the punctured variable nodes and the purged check nodes.

•
Check node merging reduces the overall complexity of the code, and increases the propagation speed of the messages during the iterative decoding, thereby reducing the convergence time of the algorithm. Check node merging can also be applied to variable nodes with higher degrees, although this impacts the performance of the code and increases the complexity of the decoding algorithm. However, many LDPC codes used in practice (e.g., those used by 802.16e and 802.11n) have a dual-diagonal structure for the parity bits and are thus suited well for check node merging. An example of the performance of check node merging applied to the 802.16e rate-1/2 code is shown in Fig. 2, where it is shown that the block error rate and the average number of iterations is slightly decreased while the average number of iterations is significantly reduced in the higher SNR region.

### III. FPGA IMPLEMENTATION

#### A. Decoder schedule

The schedule of the decoder and relevant definitions (similar to [6]) are shown in Fig. 3:

- A **layer** is a group of check nodes where all the involved variable nodes are independent. It may be bigger than the expansion factor of the base parity-check matrix.

#### B. Architecture overview

An overview of the check node merging decoder architecture is shown in Fig. 5. At the start of each block, the contents of the bit-sum memories are initialized with the log-likelihood ratios received from the channel. Then the contents of the bit-sum memories are updated during a number of iterations. In each iteration, the hard-decision values are also written to the decision memory.

![Architecture of decoder utilizing check merging.](image-url)
decision memory, from where they can be read when decoding of the block is finished.

In each iteration, bit-sums are read from the bit-sum memories, routed by a cyclic shifter to the correct check function unit (CFU), and then routed back. The output of the CFUs are bit-sum differences which are accumulated for each read bit-sum value in the bit-sum update block. The updated bit-sums are then rewritten to the bit-sum memories. The two counters count the current input and output indices of the bit-sum blocks of the currently processed check node group.

The following parameters are defined for the architecture:

- \( Z \) is the maximum expansion factor
- \( Q \) is the parallelization factor
- \( C \) is the maximum check node degree
- \( w_q \) is the bit-sum wordlength
- \( w_r \) is the wordlength of the min-sum magnitudes
- \( w_e \) is the index wordlength

C. Cyclic shifters

As in the original architecture [6], the shifters are implemented as barrel shifters with programmable wrap-around. However, unlike the original architecture the return path needs a cyclic shift as a bit-sum may need updates produced in its input. A programmable offset value is used to correct for the overly optimistic estimates of the min-sum algorithm. Also, the parity of the check node is computed.

D. Check function unit

In the CFU, shown in Fig. 6, the old check-to-variable messages are stored for each of the check nodes’ neighbors. As the min-sum algorithm is used, only two values are needed. These are stored as \( w_r \)-bit magnitudes along with a \( w_e \)-bit index of the node with the minimum message and one bit containing the check node parity. These bits are grouped together and stored in the min-sum memory. Along with the signs of each check-to-variable message stored in the S memory, the check message generator generates the old check-to-variable message in two’s complement \( w_r + 1 \)-bit representation for each variable node neighbor. The old check-to-variable messages are subtracted from the bit-sum input to generate the variable-to-check messages in the bit message generator. These are used by the min-sum update unit to compute the new min-sum values, which are subsequently stored in the min-sum memory. They are also sent to the second check message generator, which generates the new check-to-variable messages. The difference between the old and new check-to-variable messages is computed, and forms the output of the CFU.

The check message generator consists of a mux to choose the correct check-to-variable message and a converter from signed magnitude to two’s complement. The bit message generator computes the variable-to-check message by adding the inverted old check-to-variable message to the bit-sum and then converting the result back to signed magnitude representation. The min-sum update unit is unchanged from the original in [6], and performs a basic search for the two minimum magnitudes in its input. A programmable offset value is used to correct for the overly optimistic estimates of the min-sum algorithm. Also, the parity of the check node is computed.

E. Bit-sum update unit

In Fig. 7, the bit-sum update unit is shown. When processing of a check node group starts, the bit-sums are loaded in the shift registers with a delay equal to the current node degree, and thus equal to the delay of the CFUs. Then, when the bit-sum differences from the CFUs start to arrive, they are added to the old bit-sums. The sum is truncated and sent to the output to be written back to the bit-sum memories. When several CFUs have updates to the same bit-sum, the updated bit-sum is reloaded in the shift register to replace the obsolete value.
the check-merging algorithm converges in fewer iterations than the standard sum-product algorithm. In Fig. 9, for each SNR a punctured rate has been chosen that results in a target BLER of $10^{-3}$. Then, the throughputs of the decoders have been determined considering the average number of iterations required at that SNR and rate. For higher SNRs, higher rates can be used, and the benefit of the check-merging decoder is seen as a significant throughput increase.

C. Synthesis results

The reference decoder [6] and the proposed check merging decoder have been synthesized for an Altera Cyclone II EP2C70 FPGA for a clock frequency of 100 MHz. Mentor Precision was used for the synthesis, and the results are shown in Table I for 802.16e and 802.11n. The $C$ values for the reference decoders were set to the maximum check node degrees of the rate-1/2 codes in the respective standards. Due to the smaller parallelization factors of the check-merging decoders, the logic utilization is reduced. However, increased check node degrees require longer FIFOs in the bit-sum update unit, increasing the register complexity.

V. Conclusions

In this paper, an architecture for decoding of rate-compatible QC-LDPC codes using the min-sum algorithm was presented. The decoder uses check node merging to reduce the convergence time of the algorithm. The check-merging algorithm was evaluated for QC-LDPC codes from the IEEE 802.16e and 802.11n standards, and the architecture was implemented in an Altera Cyclone II EP2C70 FPGA. Reduced convergence time allows the check-merging algorithm to be implemented with a lower parallelization factor, thereby reducing the implementation complexity.

REFERENCES


<table>
<thead>
<tr>
<th>$Q$</th>
<th>$C$</th>
<th>LUTs</th>
<th>Regs</th>
<th>Mem. bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>802.16e, ref. [6]</td>
<td>24</td>
<td>7</td>
<td>4730</td>
<td>5231</td>
</tr>
<tr>
<td>802.16e, prop.</td>
<td>12</td>
<td>6</td>
<td>2997</td>
<td>4649</td>
</tr>
<tr>
<td>802.11n, ref. [6]</td>
<td>7</td>
<td>6</td>
<td>16706</td>
<td>10849</td>
</tr>
<tr>
<td>802.11n, prop.</td>
<td>7</td>
<td>4</td>
<td>7727</td>
<td>13529</td>
</tr>
</tbody>
</table>

Table I: Synthesis results of reference and check merging decoders.

Fig. 8. Maximum check node degrees of check node merged matrices for 802.16e and 802.11n rate-1/2 codes.

Fig. 9. Information bit throughput of fixed decoder [6] and check-merging decoder as function of SNR with puncturing for target BLER=$10^{-3}$. 

IV. Results

The reference architecture and the proposed architecture have been synthesized for the rate-1/2 QC-LDPC codes in 802.16e and 802.11n. For the proposed architecture, a lower parallelization factor has been used due to the faster convergence. For all implementations, $w_q = 8$ and $w_r = 5$.

A. Maximum check node degrees

When check node merging is used, the check node degrees increase with increased punctured rate, and the maximum check node degree supported by the decoder affects the length of the register chain in the bit-sum update unit. Therefore a limit must be set based on the degree distributions of the check node merged matrices. Figure 8 shows the maximum check node degrees as a function of the punctured rate for the rate-1/2 codes of the 802.16e and 802.11n standards. $C = 32$ and $C = 44$ have been chosen for 802.16e and 802.11n, respectively, in order to handle rates between 0.5 and 0.85.

B. Decoding throughput

As the complexity of the code is reduced when checks are merged, the number of clock cycles per iteration is reduced as the punctured rate increases. Further, as seen in Fig. 2(b), the check-merging algorithm converges in fewer iterations than the standard sum-product algorithm. As the complexity of the code is reduced when checks are merged, the number of clock cycles per iteration is reduced.

F. Memories

The bit-sums are organized into $Q$ memory banks with data wordlengths of $w_q$ bits. Each memory $l$ contains the bit-sums of the variable nodes $v_k$ where $k = l \mod Q$. The min-sums are organized into $Q$ memory banks with data wordlengths of $w_q + 1 + 2w_r$ bits, which holds a $w_q$-bit edge index, 1-bit parity, and two $w_r$-bit magnitudes. Each memory $l$ contains the min-sums of the check nodes $c_k$ where $k = l \mod Q$.  

1For standard codes and parallelization degrees, these memory arrangements may result in sparingly used memories. To circumvent this, the same techniques as in [6] may be used.