Memory efficient FPGA implementation for flooded LDPC decoder

Alexandru Amaricai, *Member, IEEE*, Oana Boncalo, *Member, IEEE*, and Ioana Mot

***Abstract* — This paper proposes an FPGA based flooded architecture for quasi-cyclic (QC) LDPC decoder. The message computation for both check and variable node update is done using a parallel scheme of a number of processing units equal to the expansion factor of the QC matrix. The proposed architecture performs serial processing of the messages by dedicated check node and variable node processing units. This way, a reduced memory word size is used, which lead to a reduction of the BRAM blocks. Multiple frame decoding is used in order to both increase the throughput and to increase the BRAM usage. Implementation results for the WiMAX (1152, 2304) QC irregular LDPC code indicate that the proposed architecture has up to 4x less slices resource utilization and up to 1 order of magnitude less BRAM blocks with respect to other flooded architectures, while maintaining a throughput of several hundreds of Mbps.**

***Keywords* —Flooded scheduling; FPGA; LDPC decoding**

I. Introduction

LDPC codes are error correction schemes used in modern telecommunication systems, being adopted in a wide range of standards, such as IEEE 802.16 WiMAX or DVB-S2 [1][2]. LDPC decoding can be performed using two different scheduling strategies: flooded scheduling and layered scheduling [1][3][4]. Layered decoders require a lower amount of memory bits - due to a reduced number of stored messages - , and have better convergence – due to the increased number of updates on the a-posteriori log likelihood messages. However, the layered architecture is subject to patents, which restricts is usability [4]. Another advantage of the flooded LDPC decoder is represented by its high reliability to hardware faults which affect its internal circuitry: the error correction capability of the decoder is not affected by a certain amount of hardware faults within the decoder [5]. This makes the flooded strategy the preferred choice for applications with increased fault tolerance requirements.

LDPC decoders have significant memory requirements, of several tens of kilobits [6][7]. Thus, for FPGA architectures, implementing the LDPC memories with BRAM modules would be preferable rather than distributed RAM. However, several limitations in the BRAMs lead to their inefficient use even in LDPC decoders with modest throughputs. These include: limited number of read/write ports (2 read/write ports) and limited memory word size (maximum 72 bits). Therefore, LDPC decoders require significant number of BRAM blocks, but only a few entries of these blocks are used (less than 30 entries out of the 512 which are available in 36 kb sized BRAM).

In this paper, we propose a memory efficient architecture for FPGA implementation of flooded LDPC decoders. In order to obtain reduced cost, while having acceptable throughput (of several hundreds of Mbps), several techniques are employed: *(i)* parallel processing units (variable node units – VNU and check node units – CNU) at B matrix row/column level *(ii)* serial processing of messages corresponding to a VNU/CNU *(iii)* aggressive pipelining *(iv)* multiple codeword processing. By applying these techniques, we use 456 entries out of the 512 corresponding to a BRAM for the extrinsic message memories and 144 entries for the input log likelihood ratios (LLR) memory. The obtained throughput is of almost 300 Mbps for coded bits, for 20 decoding iterations.

This paper is organized as follows: Section II represents a brief introduction into flooded LDPC decoding; related work is presented in Section III; the proposed architecture is detailed in Section IV; implementation results are discussed in Section V; last section is dedicated to the concluding remarks.

II. Flooded Decoding of LDPC Codes

QC LDPC codes are a class of LDPC codes, which present highly structured parity check matrix, defined by blocks of circulant matrices [3]. Regarding the decoding algorithm, Min-Sum (MS) and its variants (such as offset MS or normalized MS) are the most used for hardware implementations. Because it uses only additions and comparisons on a small number of bits (i.e. < 10 bits), it has the lowest hardware complexity.

Flooded decoding requires two type of processing: the CNUs compute the check node messages (denoted as β) based on the messages received from the VNUs (denoted as α); these updated β messages are passed to the VNUs, which will update α messages. This message passing is performed for several iterations. The flooded MS decoding has as inputs the channel LLR messages (denotes as γ), and consists of the following steps:

1. Initialization $\alpha_i = \gamma_i$ (1)

This work has been supported by the European Commission's project "I-Risc - Innovative Reliable Chip Design from Low Power Unreliable Components", grant number 309129.

Alexandru Amaricai, Oana Boncalo and Ioana Mot are with the Computer Engineering Department, University Politehnica Timisoara, Vasile Parvan, Blvd, Nr. 2, Timisoara, Romania (email: alexandru.amaricai@cs.upt.ro)

2. Check node update

$$\beta_{z,i} = \left(\prod_{j \in H(z) \setminus i} \operatorname{sgn}(\alpha_{z,j}) \right) \times \min\left(\alpha_{z,j}\right) \quad (2)$$

3. Variable node update $\alpha_{z,i} = \gamma_i + \sum_{j \in H(z) \setminus i} \beta_j$ (3)

4. A-posteriori update $\tilde{\gamma}_i = \gamma_i + \sum_{j \in H(z)} \beta_j$ (4)

The above four steps are repeated until one codeword is found (if a stopping criteria circuit is implemented) or a maximum number of iterations is reached. A disadvantage of the flooded scheduling is represented by the fact that the check node update has to wait for the variable node update and vice-versa [12]. Thus, during the VNU processing, the CNUs are stalled and vice-versa. In order to overcome this disadvantage, dual frame decoding is used: VNUs perform update on one frame, while CNUs perform the corresponding updates on the other frame

III. Related Work

Fully flooded architectures are the faithful hardware implementation of the Tanner graph. This type of decoders require a number of VNUs equal to the number of columns in the H matrix, and a number of CNUs equal to the number of rows in the H matrix. The main advantage of this architecture is represented by the high throughput. However, it has a major drawback represented mainly by the very high cost of the routing network. Several approaches, such as [8][9][11], tried to reduce the interconnection size, mainly by reducing the message quantization, at the expense of error correction capability. FPGA implementations of such architectures present very large slice count.

One way to reduce the complexity of such decoders is to serialize the variable node and/or check node updates. These approaches require memories in order to partially store messages. The approach in [12] uses a number of VNU equal to the number of columns in the *H* matrix, while the number of CNU is equal to the circulant size. The VNU process β messages in a serial manner: the d_v messages corresponding to a VNU are read in consecutive clock cycles. The CNU process the α messages in a parallel manner: the dc α messages corresponding to a CNU are read in a single clock cycle. Barrel shifters are used for routing messages to the corresponding processing node. The implementation is ASIC based. The memory is implemented using a dedicated non-refresh DRAM; the size of the memory is equal to the number of required bits.

The architectures in [6][7][10] have been implemented in FPGA. The ones in [10] use a number of VNUs proportional to the number of columns in the B matrix, while the number of CNUs is proportional to the number of rows in the B matrix. It does not use any kind of routing network. The number of memory modules required for α and β messages is equal to the number of elements in the B matrix. One message (4-6 bits) is stored in a BRAM word. Thus, the usage of the BRAM for this architecture is rather small. Improvements of this architecture have been proposed in [6]; these rely on packing multiple messages in a single in the same BRAM word (vectorization) or on techniques which allow several circulant matrices share the same BRAM (folding). However, in both cases, the number of BRAM blocks used is high - 80 BRAM for (8176,7156) LDPC code using vectorization technique and 330 BRAM for (3369,3213) LDPC code using folding technique.

IV. Proposed Architecture

A. Overall architecture

The design parameters for the proposed LDPC decoder are: γ message quantization (γ_quant), β message quantization (β_quant), α message quantization (α_quant), the size of the circulant matrix *m*, number of columns in *B* matrix *NrCols*, and number of rows in *B* matrix *NrRows*. The proposed architecture is depicted in Fig. 1 and is composed of the following modules:

1. *Input LLR memory* – a single frame requires a memory module with word size of (*m* x γ_quant) bits and a depth of *NrCols*; the number of BRAM modules for this memory is equal to (*m* x γ_quant)/72; processing a single frame for WiMAX ½ rate code will lead to a BRAM usage of less than 5%; in the last iteration, this memory will be updated with the a-posteriori LLR, which will be used for computing the hard decision.
2. *VNU processing block* – it consists of m VNUs; the m VNUs perform the variable node and a-posteriori updates corresponding to a column in the B matrix; each VNU process d_v β messages in a serial manner; the VNU outputs d_v α messages also in a serial manner (one α message per clock cycle); the VNU processing blocks requires the reading of *m* β messages in one clock cycles and perform *m* α messages write operations in one clock cycle.
3. α *message memory* – the word size for this memory is (*m* x α_quant) bits; for a frame, the depth of this memory is equal to the number of non-negative elements in the B matrix; for the considered WiMAX code, the depth of the α message memory is equal to 76; the number of BRAM blocks required for this unit is equal to (*m* x α_quant)/72.

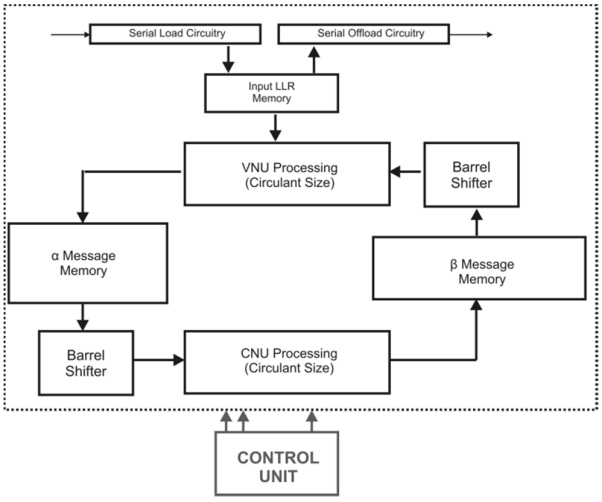

Fig. 1. LDPC Decoder Architecture

4. *α read barrel shifter* – it represents the interconnection network between the outputs of the VNUs to the inputs of the CNUs; for considered WiMAX code, it has 7 level of multiplexers; the number of multiplexers per level is equal to (*m* x *α_quant)*; the shift amounts are provided by the control unit and correspond to the non-negative values in the B matrix.
5. *CNU processing block* - it consists of *m* CNUs; the m CNUs perform the check node updates corresponding to a row in the B matrix; each CNU process d_c *α* messages in a serial manner; the CNU outputs d_c *β* messages also in a serial manner (one *β* message per clock cycle); the CNU processing blocks requires the reading of m *α* messages in one clock cycles and perform m *β* messages write operations in one clock cycle.
6. *β messages memory* - the word size for this memory is (*m* x *β_quant)* bits; for a frame, the depth of this memory is equal the depth of the *α message memory*; in many ASIC implementations, but also in some FPGA, compressed *β* message format is stored in the memory; using the compressed format, the number of memory bits is reduced; for BRAM based implementation of this memory module, using the compressed format will lead to an increase of BRAM blocks used; this is due to the higher memory word sized used for storing compressed *β* messages; using the uncompressed form, the *β* messages, which would have been composing the compressed message, are stored in separate memory words; this way, the size of the memory words is reduced; thus, the number of the BRAM blocks used is decreased with respect to the compressed format;
7. *β read barrel shifter* – it represents the interconnection network between the outputs of the CNUs to the inputs of the VNUs; it has the same number of multiplexer levels as the *α* read barrel shifter; the number of multiplexers per level is equal to (*m* x *β_quant)*; the shift amounts are provided by the control unit and correspond to the non-negative values in the B matrix.
8. *Control unit* – it provides: (i) memory addresses and memory enable signals for all three memories, (ii) the shift amounts for the two barrel shifters, (iii) the control signals for the processing units, (iv) because the used WiMAX LDPC code is irregular, the control unit provides the number of *β* messages which need to be processed by the VNUs for each columns in the B matrix, as well as the number of *α* messages which need to be processed by the CNUs for each row in the *B* matrix; the addresses, shift amounts and the number of processed messages are provided by dedicated ROM memories.

Due to the serial type of processing at both CNU and VNU level, the proposed architecture can be easily adapted for other LDPC codes.

B. Multiple frame processing

Dual frame processing has been employed in flooded architectures, such as [12]. It has the goal to improve hardware resources efficiency, as well as to double the throughput of the LDPC decoder. For the proposed architecture, dual frame processing has no cost increase with respect to the single frame decoding. Therefore, the usage (depth) of the BRAM modules is doubled with respect to the single frame decoding.

In order to further increase the usage of the BRAM blocks, we employ a 6 frame processing. The CNUs perform updates for 3 frames, while the VNUs perform the updates corresponding to the other 3 frames. For the BRAM blocks used to implement the α message memory and β messages memories a number of 456 memory words out of 512 are used. For the input LLR memory, the number of entries used is equal to 144 out of the 512. Regarding the throughput, the total VNU/CNU processing time (the time required to process all the messages corresponding to all 3 frames) is increased by a factor of 3 corresponding to dual frame processing. Therefore, the throughput for 6 frames decoding is equal to the one for dual frame decoding. Increasing the number of decoding frames (such as 8 frames), would lead to the increase in the number of BRAM modules. This is due to the fact that 8 frames would require 608 memory words.

C. Processing Units

The CNU and the VNU are depicted in Fig. 2. The CNU update is performed according to eq. (2). It consists of computing the first two minimums from the absolute values of the input α messages, as well as the index of the first minimum. The CNU performs the following operations: (i) conversion of the input α message from a two's complement format to a sign magnitude format (ii) comparison between the incoming *α* message and the previous two minimums, as well as updating the index of the first minimum (iv) computing the *β* message from the values of the first two minimums and the current index of the *β* message.The VNU performs the update of the α message based on the input LLR message and d_v *β* messages, according to (3). This is performed by computing first the a-posteriori LLR, and then subtracting the corresponding *β* message. For a 4-bit β message quantization, a-posteriori LLR is computed using a 6-bit accumulator. The α message is computed on 6-bits and then is saturated to 4 bits for memory storage.

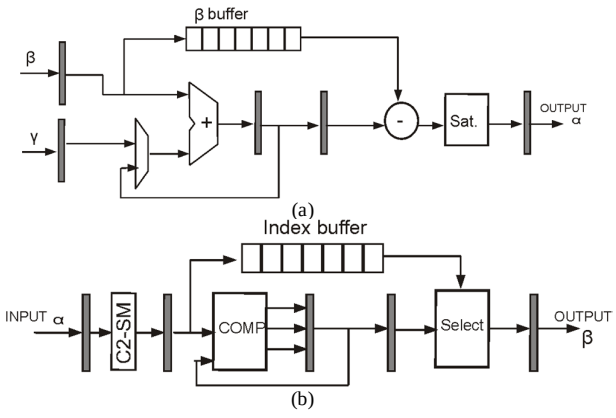

Fig.2. VNU Architecture (a) CNU Architecture (b)

TABLE I. RESULTS COMPARISON FOR DIFFERENT FPGA ARCHITECTURES

	Code	Quantization	Device	Frequency (MHz)	Throughput (Mbps)	Resources	
						Slices	BRAM
Chandrasetty2012 [9]	(576,1152)	4,2	Virtex-5	138	11400	10823	
Balatsoukas-Stimming2012 [8]	(1152,2304)	4,3 3,2	Virtex-5	154 211	8900 12200	21688 11700	
Vector overlapped [6] Folded [6]	(7156,8176) (3213,3969)	6,4 6,4	Virtex-4	212-228 200-226	195-713 101-460	4021-17100 6600-14100	80 330
Torres12 [11]	(1723,2048)	4,2	Virtex-6	30	3050	24860	
Proposed	(1152,2304)	6,4	Virtex-7	260	290	3880	38

Both VNUs and CNUs use dedicated FIFO buffers. For VNUs, these buffers are used to synchronize the current α message index for β message computation; this index is used to select between the first and the second minimum. For CNUs, these FIFO buffer are used to synchronize the β message for the subtraction from the a-posteriori LLR. A fixed size FIFO would have required the introduction of stall cycles for both CNU and VNU processing. This is due to the irregularity of the considered WiMAX code. We tackle this issue by implementing the FIFO buffers using a modified register file. For both the CNU and the VNU, the number of processed messages is used to address the buffer.

V. SYNTHESIS RESULTS

We have implemented the proposed architecture for the WiMAX (1152, 2304), with the circulant size m=96. The decoder has been synthesized using Xilinx ISE and implemented on a Virtex-7 VX485T device with speed grade -2. Implementation results are depicted in Table I. Regarding the comparison with decoders implemented on Virtex-4 devices, these are using 4-input LUT instead of the 6-input LUT present in Virtex-5 or Virtex-7. Results show that the proposed design has the smallest slice count with respect to other flooded architectures. Considering the non-fully parallel flooded architectures, the proposed one has the lowest BRAM count. If we consider the cost (slice count and BRAM count) per processing frame, the proposed requires less than 7 BRAM blocks and less than 700 slices per frame. This is due to the fact that 6 frames processing has a negligible cost increase with respect to dual frame processing. Furthermore, it has a throughput of approximately 300 Mbps for 20 iterations, and 400 Mbps for 15 iteration, which is comparable to the throughput of the decoders in [6], but at a smaller cost both in BRAM and slices. The [8][9] approaches try to obtain high throughput at reasonable cost by using smaller quantization. Thus, these decoders sacrifice error correction capability for better cost-throughput trade-off. Using smaller quantization will also result in significant memory and slice count reduce in the proposed architecture. Regarding the working frequencies, Table I indicates that the proposed decoder has the highest frequency with respect to other flooded architectures.

VI. CONCLUSIONS

This paper presents a memory efficient flooded architecture for QC LDPC decoders for FPGA implementations. The main advantages of the proposed architecture: efficient BRAM utilization, efficient slice based utilization due to the serial nature the processing of at both CNU and VNU level, multiple frame decoding - 6 frames are decoded in the same decoding round, without any hardware overhead; this way, we significantly increase the usage of BRAM blocks, by taking advantage of the generous depth of BRAM modules; efficient processing unit implementation – due to serial processing. This also favors multi-rate and easiness in changing the code.

REFERENCES

[1] T. Richardson and R. Urbanke, "The Renaissance of Gallager's Low-Density Parity-Check Codes", *IEEE Comm. Magazine*, Aug. 2003.

[2] IEEE-802.16e, "Physical and medium access control layers for combined fixed and mobile operation in licensed bands," 2005, amendment to Air Interface for Fixed Broadband Wireless Access Systems

[3] M.P.C. Fossorier, "Quasicyclic Low-Density Parity-Check Codes from Circulant Permutation Matrices," *IEEE Trans. on Information Theory*, vol. 50, no. 8, pp. 1788–1793, 2004.

[4] D. Hocevar "Layered decoding of low density parity check (LDPC) codes" European Patent EP 1622276 A3, 2006

[5] CK Ngassa, V Savin, D Declercq "Min-Sum-based decoders running on noisy hardware" IEEE Global Communications Conference (GLOBECOM), 2013

[6] X. Chen, J. Kang and S. Lin and V. Akella, "Memory System Optimization for FPGA Based Implementation of Quasi-Cyclic LDPC Codes Decoders", *IEEE Trans. on CAS*, 2011

[7] Z. Wang, Z. Cui, "A Memory Efficient Partially Parallel Decoder Architecture for Quasi-Cyclic LDPC Codes", *IEEE Trans. on VLSI Systems*, Vol. 15, No. 4, April 2007

[8] A. B. Stimming and A. Dollas, "FPGA-based design and implementation of a multi - GBPS LDPC decoder", *FPL*, 2012.

[9] V. A. Chandrasetty and S. M. Aziz, "An area efficient LDPC decoder using a reduced complexity min-sum algorithm", *VLSI Journal*, 2012.

[10] Y. Dai, Z. Yan, and N. Chen, "Optimal overlapped message passing decoding of quasi-cyclic LDPC codes," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 16, no. 5, pp. 565–578, May 2008.

[11] V.Torres, A. Perez-Pascual, T. Sansaloni, J. Valls "Fully-parallel LUT-based (2048,1723) LDPC Code Decoder for FPGA" Proc 19th Int. Conf. on Electronic Circuits and Systems (ICECS), 2012

[12] YS Park, D Blaauw, D Sylvester, Z Zhang "Low-Power High-Throughput LDPC Decoder Using Non-Refresh Embedded DRAM" IEEE Journal of Solid State Circuits, Vol. 49, Issue 3, 2014