Parallel Scalable Hardware Architecture for Hard Raptor Decoder

T. Mladenov and S. Nooshabadi and K. Kim
Dept. Information & Communications,
Gwangju Institute of Science and Technology,
Gwangju 500-712, South Korea,
Email: {todor,saeid,kskim}@gist.ac.kr,

A. Dassatti
VLSI Lab, Electronic Department,
Politecnico di Torino,
10130 Torino, Italy,
Email: alberto.dassatti@gmail.com

***Abstract*—In this paper we propose a novel parallel hardware architecture for two binary matrix inversion and vector decoding algorithms, for hard Raptor decoder. We compare the achieved performance to a software based implementation in an embedded processor. We demonstrate the superiority of our proposed architecture in terms of performance (by a factor 12), power and energy dissipation (by a factor of 15). We also include the hardware resource requirements in the comparison. Furthermore, the proposed hardware architecture is parameterized and easily scalable. The data processing word size has been successfully extended up to 1024 bits and fitted within the chosen FPGA hardware platform.**

I. Introduction

Raptor Codes [1] are designed to perform as close as possible to the Shannon's channel limit. They are very suitable for communication scenarios where the back channel is not available (satellite communications) or the back channel is unwanted (multimedia broadcasting). They are rateless, thus able to generate as many parity symbols as needed on-the-fly. Raptor Codes come as an improvement to LT-Codes [2], providing linear encoding and decoding time.

The hard Raptor Codes deal with erasure channels, where symbols are either received correctly or dropped in case they have error(s). Recently Raptor Codes have gone mobile, being chosen for the forward error correction (FEC) scheme in 3GPP [3] and DVB-H [4] standards. The documents for these standards present a fully specified version of a systematic Raptor Code. They also propose an efficient binary matrix inversion algorithm for the most time consuming Raptor decoding operation - the *constraints processor's matrix inversion*. It accounts for up to 92% of the decoding time [8]. An improved version of this algorithm is presented in [5]. The idea of incremental decoding has been presented in matrix form in [6]. A quicker re-start algorithm is proposed in [7]. The implementation in hardware of Raptor Codes have been investigated in [8].

In this paper we propose a novel parallel scalable hardware architecture, for the code constraints processor matrix inversion and the intermediate symbol vector decoding. We show a comparison between two matrix inversion algorithms (Gaussian elimination (GE) and the 3GPP/DMB-H standard proposed algorithm (SA)), implemented in software and in hardware, for a range of symbol sizes T, and block sizes K. We further consider two different memory mapping schemes: "PACKED WORD" and "Sparse" matrix representation. For each configuration we have analyzed the power and energy requirements, and outlined the trade offs together with the necessary hardware resources.

The representative embedded system platform, chosen for this work, is a 100 MHz NIOS II soft-core processor, with customizable instruction set, running on an *EP3SL150F1152* Altera Stratix III FPGA. Fig. 1 depicts the high level block diagram of the embedded system on a chip (SoC) platform, used for the implementation of Raptor Codes. The hardware implementation of the proposed architecture is placed in the "Dedicated Hardware" block.

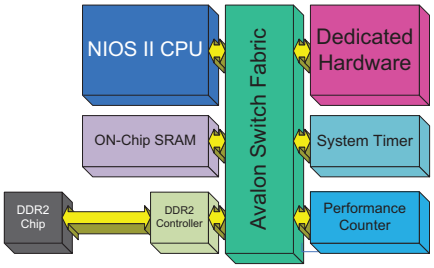

Fig. 1. Raptor Codes on hardware/software NIOS embedded system

This paper is organized as follows. In Section II, we briefly explain the operation of a systematic Raptor decoder. Section III describes and presents the details of GE and SA, the two algorithms for the matrix inversion used in this paper. Section IV presents the software and hardware implementation performance results in terms of execution time, power and energy, and hardware resources. Section V concludes the paper.

II. Systematic Raptor Decoder

A Hard Raptor Code can be viewed as a regular linear block code, and can be represented by a generator matrix. For practical applications the rate of the code is either fix or a feedback channel is provided. A block diagram of systematic Raptor encoder and decoder is shown in Fig. 2. The decoding process of Raptor Codes exchanges the positions of the *Code Constraints Processor* and the *LT Encoder*(*Decoder)* with the proper dimensions for the $\mathbf{G}_{LT}$ LT generator matrices. This

978-1-4244-5309-2/10/$26.00 ©2010 IEEE

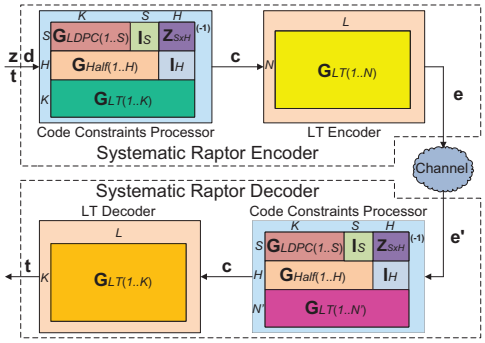

Fig. 2. Block diagram of the systematic Hard Raptor Codes

allows for successful decoding when only the first K encoded symbols (K is the number of source symbols in a block) have been received and no errors are detected in the channel.

The output vector $\mathbf{e}$, containing N symbols, generated by the encoder is received by the decoder across the channel as input vector $\mathbf{e}'$, containing N' $(K \leq N' \leq N)$ encoded symbols (which may be nonconsecutive). Vector $\mathbf{e}'$ is padded with $S+H$ zeroes to dimension it to $(M = N' + S + H)$. Starting with $(N' - K)$, the value of N' is iteratively incremented to make the code constraints preprocessor matrix $\mathbf{A}$ invertible. The difference $(N' - K)$ is equal to or greater than the number of received encoded symbols lost in the channel. The decoding is performed according to (1) and (2), where $\mathbf{G}_{LT}$ is a LT generator matrix with the dimension of $K \times L$. All operations are performed in Galois Field GF(2).

$$\mathbf{c}_{[0:L-1]} = [\mathbf{z}^T \quad \mathbf{e'}^T] \cdot \mathbf{A}^{-1}_{M \times L} \tag{1}$$

$$\mathbf{t}_{[0:K-1]} = \mathbf{G}_{LT} \cdot \mathbf{c}_{[0:L-1]} \tag{2}$$

At the decoder side, the submatrix $\mathbf{G}_{LT}(1..N')$ is first built from the input data. The sequence number of the n^{th} received encoded symbol is used to generate the n^{th} row of the submatrix $\mathbf{G}_{LT}(1..N')$ through the LT encoding process. Further details can be found in [9], [3], [4].

III. Matrix Inversion Algorithms

The most common matrix inversion algorithm is Gaussian Elimination (GE). It involves row exchange and row XOR operations. An example of GE for Galois Field GF(2) is shown on Fig. 3. The structure and simplicity of the algorithm allow for greater design freedom and optimization. That is valid even to a greater extend when it comes to its hardware implementation. As we will show, GE benefits greatly from the proposed parallel scalable architecture.

The SA technique, proposed in [3], [4], on the other hand, aims at reducing the row XOR operations by introducing some additional column exchange operations. It complicates the algorithm, but reduces the amount of memory accesses. An example of the SA algorithm is shown in Fig. 4.

SA algorithm has three phases. In Phase one, it tries to convert as big portion of the matrix as possible, to an identity

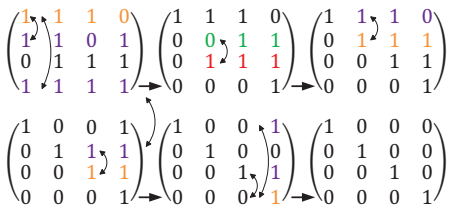

Fig. 3. Example of GE algorithm

matrix, through a minimum number of row XOR operations. In Phase two, whatever portion that is left after this phase on the right side of the original matrix, is split in two, and the lower part of it is converted into identity matrix, through the Gaussian elimination. Phase three completes the inversion by reducing the matrix to the form of an identity matrix. The first line of Fig. 4 shows the form of the matrix after each of the three phases.

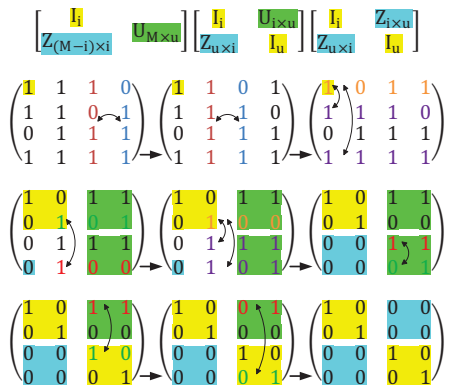

Fig. 4. Example of SA algorithm

IV. Results and Discussion

This section compares and analyzes the performance, power and energy dissipation of various implementations of systematic Raptor decoder. Two separate implementations, for each of SA and GE matrix inversion and vector decoding algorithms, are under consideration: a pure software - on the soft core NIOS II processor platform, and a pure hardware - on the parallel scalable hardware architecture proposed in this paper, as show in Fig. 5. The encoded symbol vector is processed in parallel with the matrix to reduce the required memory footprint and to avoid the need for post matrix multiplications.

In Fig. 5 the Avalon switch fabric uses a slave port to allow the NIOS II processor to configure the *Control Registers*, that initialize the hardware accelerator. During the initialization, the size and initial addresses for the matrix and the vectors are set. The *Control Registers* also control the operations of the hardware, like initiating *START* and *STOP* commands. The hardware accelerator block uses an Avalon master port to access the whole memory mapped address space of the NIOS II processor, send interrupts to NIOS II and receive interrupts from other peripheral devices. For a more flexible design *Row & Column Exchange Logic*, and *Row XOR Logic*

units are designed to be self contained units that interface with the *GE/SA Control Logic* finite state machine. The control logic for the two algorithms and the row and column operation blocks share a common address generation path, through the *Arithmetic Unit*. It contains two 32 bit adders and one 16 bit multiplier. The *Latch* circuit only exists in the SA version of the hardware accelerator.

This architectural approach allows for the processing word size to be easily scalable, through the means of parameters only. Furthermore, adding additional algorithms reduces to only implementing the specific control logic. The design has successfully fitted in the selected FPGA device with word size from 32 bits up to 1024 bits for both SA and GE algorithms. The Avalon switch fabric allows for easy formation of wide word bus, feeding the engine of the proposed hardware accelerator, from various sized memory words. The matrix and vectors are stored in the external DDR2 SDRAM memory chip, running at 400 MHz.

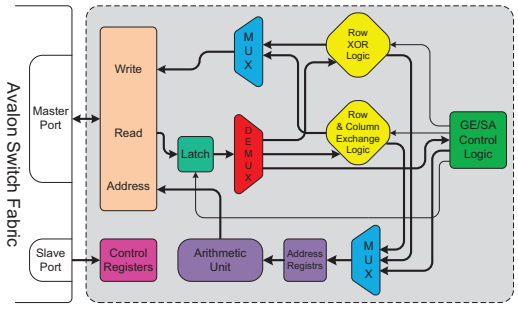

Fig. 5. Block diagram of the parallel scalable hardware architecture

The software version of the SA algorithm is implemented with "Sparse" memory mapping, i.e. only the non zero elements are kept in memory. As it is shown in [8], GE is not performing well under the "Sparse" memory mapping and thus it is implemented with "PACKED WORD" memory mapping scheme. With this representation, thirty two matrix elements are stored in one 32 bit memory word. The memory saving is at best 32 times. Further details about the memory mapping schemes, for the code constraints matrix, can be found in [8].

The proposed hardware architecture uses only "PACKED WORD" for both SA and GE algorithms. Implementing the "Sparse" memory mapping in hardware requires a large logic overload and will negate the benefits of the hardware accelerator. Up to now, we have not come across such an implementation in hardware. Nevertheless, the "PACKED WORD" implementation of SA in hardware has been thoroughly optimized to the extent that it outperforms the corresponding "Sparse" implementation in software.

A. Performance

1) Software Implementation: Fig. 6 shows the performance results of SA and GE algorithms in software. The performance time is plotted against the symbol size T, ranging from 4 to 128 bytes, for several values of the block size K (from 128 to 1024 symbols). It can be seen that for all shown values of T,

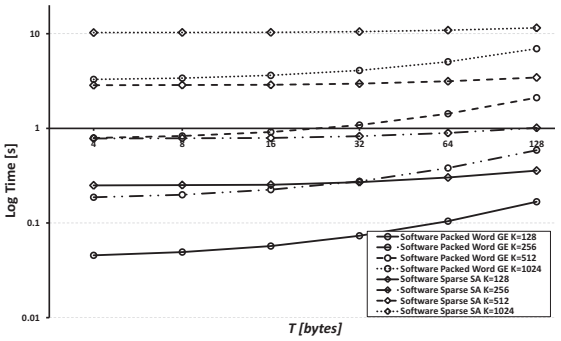

Fig. 6. Software implementation of SA and GE for various T and K

GE outperforms SA for all values of K. That is so because for small values of T the dominant factor in the performance is the matrix inversion and not the vector decoding. As the size of T grows, the decoding time grows much faster for GE compared to SA. This results in SA outperforming GE for values of T larger than 128 bytes in the pair of plots for each value of K. This is due to the reduction in the number of row XOR operations in its favor, and the growing dominance of the vector decoding.

Further, Fig. 6 reveals that the relative performance of GE over SA reduces to the increase in the value of K. It is shown in [8] that the sparsity of the code constraints matrix grows with K. Therefore, the "Sparse" SA implementation, while benefiting from the increased sparsity, is less affected by the larger vector size T, due to its reduced row XOR operations required for the vector decoding.

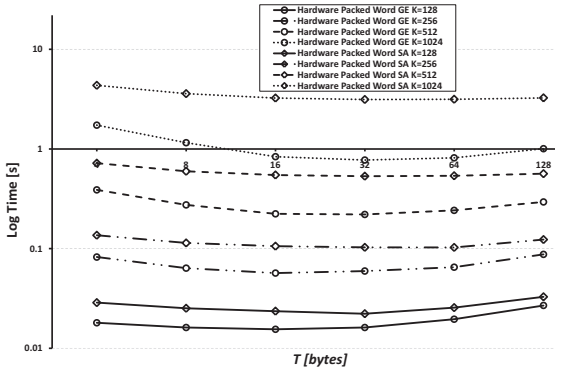

Fig. 7. Hardware implementation of SA and GE for various T and K

2) Hardware Implementation: Fig. 7 shows the performance of the code constrains matrix inversion and vector decoding, implemented with the proposed parallel scalable architecture in hardware. The symbol sizes T and the symbol block sizes K are the same as those used in Fig. 6, for the software implementation.

The main performance difference, compared to the software implementation, is that GE performs better than SA for larger T and all K values. For example, for $T = 32$ and $K = 1024$ GE outperforms SA by a factor of 4. This is due to the structure of GE, as already mentioned, and its better suitability

TABLE I
POWER, ENERGY AND HARDWARE RESOURCES FOR THE HARDWARE AND SOFTWARE IMPLEMENTATION FOR 100 MHZ LOGIC SPEED.

	Hardware Packed Word GE			**Hardware Packed Word SA**			**Software Packed Word GE**			**Software Sparse SA**		
T (bytes)	**4**	**16**	**128**	**4**	**16**	**128**	**4**	**16**	**128**	**4**	**16**	**128**
Power [mW]	1253	1308	1693	1242	1255	1597		1234			1221	
Energy [mJ] K=128	22.6166	20.3002	45.5417	35.6951	29.6306	52.4774	56.2581	70.511	206.991	304.847	309.536	436.141
Energy [mJ] K=256	103.172	74.4121	148.222	169.359	132.842	196.974	230.659	277.699	725.419	954.224	965.542	1,235.79
Energy [mJ] K=512	487.542	292.273	497.877	894.103	687.878	901.922	977.612	1,130.52	2,603.29	3,484.99	3,518.12	4,207.58
Energy [mJ] K=1024	2,173.96	1,096.94	1,707.92	5,407.06	4,077.11	5,206.25	4,058.31	4,474.67	8,575.79	12,540.8	12,611.33	14,072.53
Logic Elements	5,819	6,629	16,006	6,850	8,521	27,888			4,767			
Memory Bits	59,296	87,264	348,800	59,296	87,264	348,800			59,296			

for the parallel processing. Irrespective of its size, once inside the hardware accelerator, all the bits of the data symbol T are processed in parallel. This shifts the dominant performance factor back to the matrix inversion operation and diminishes the vector decoding advantage of the SA algorithm. Going to larger K values further increases the performance gap between GE and SA; a behavior contrary to the software implementation results.

The performance lines for GE and SA have different shapes. The SA lines stay relatively more straight than the GE lines for all T values, especially for large K values. This is due to the limitation in accessing the memory. Although the processing inside the hardware is performed in parallel, even for the large word size symbols, accessing these words from memory is still sequential, and done in small chunks and grouped before processing. The burst nature of the memory allows for improvement in the GE performance, for T values up to 32 bytes. After that value, the sequential memory accesses start to drag down the performance of the GE algorithm. On the other hand, the SA algorithm, with the reduced number of row XOR operations, requires less memory accesses and maintains a relatively constant performance at larger symbol sizes T, for fixed values of K.

The SA implementation with the "PACKED WORD" memory mapping has been optimized, through the *Latch* circuit in Fig. 5, and it delivers better performance compared to the "Sparse" implementation in software. For example, for $T = 32$ and $K = 128$ the proposed hardware implementation of SA outperforms the software version of the "Sparse" SA by a factor of 12.12. The *Latch* circuit operates as a fast application specific cache memory.

B. Power and Area

Table I shows the power and energy dissipation, together with the hardware resources for the two discussed sets of implementations. The power measurements are done very accurately through a dedicated and precise circuitry on the simulation platform.

The hardware implementation of SA takes up to 1.74 times more logic than the hardware implementation of GE, at $T = 128$ bytes. Nevertheless, the hardware implementation of SA dissipates less power for all T values, compared to the GE algorithm (due to the reduced memory accesses), but needs more energy to invert the same code constraints matrix and decode the vector. For example, for $T = 32$ and $K = 1024$ SA requires 3.94 times more energy compared with GE.

On the other hand, the pure software implementations need less logic and dissipate less power in comparison to the hardware implementations, but they require more energy. For example, the "Sparse" SA in software requires 15 times more energy than the GE "PACKED WORD" in hardware, for $T = 32$ and $K = 128$.

V. CONCLUSION

We proposed a new scalable parallel architecture for designing a hardware accelerator, for systematic Raptor decoder. Comparisons between GE and SA matrix inversion algorithms, implemented in hardware and software, were presented. Further, a detailed implementation resources data was presented. The proposed hardware solutions achieved significant performance gains (up to 12 times) and energy savings (up to 15 times).

ACKNOWLEDGMENT

This work was partially supported by the Center for Distributed Sensor Network at GIST, and the World Class University (WCU) program at GIST through a grant provided by the Ministry of Education, Science and Technology (MEST) of Korea (Project No. R31-2008-000-10026-0)

REFERENCES

[1] A. Shokrollahi, "Raptor codes," *IEEE Trans. Inf. Theory*, vol. 52, pp. 2551–2567, Jun. 2006.

[2] M. Luby, "LT codes," in *Proceedings, The 43rd Annual IEEE Symposium on Foundations of Computer Science*, 2002, pp. 271–280.

[3] *3GPP TS 26.346, Technical Specification Group Services and System Aspects; Multimedia Broadcast/Multicast Service (MBMS); Protocols and codecs*, 3GPP Technical Specification, Rev. V7.4.1, Jun. 2007.

[4] *Digital Video Broadcasting (DVB); IP Datacast over DVB-H: Content Delivery Protocols*, ETSI Technical Specification, Rev. V1.2.1, 2006.

[5] S. Kim, S. Lee, and S.-Y. Chung, "An efficient algorithm for ml decoding of raptor codes over the binary erasure channel," *IEEE Commun. Lett.*, vol. 12, pp. 578–580, Aug. 2008.

[6] J. Heo, S. Kim, J. Kim, and J. Kim, "Low complexity decoding for raptor codes for hybrid-arq systems," *IEEE Trans. Consum. Electron.*, vol. 54, no. 2, pp. 390–395, May 2008.

[7] A. AbdulHussein, A. Oka, and L. Lampe, "Decoding with early termination for raptor codes," *IEEE Commun. Lett.*, vol. 12, no. 6, pp. 444–446, Jun. 2008.

[8] T. Mladenov, S. Nooshabadi, and K. Kim, "Hardware implementation of matrix inversion for raptor decoder on embedded system," *Circuits and Systems, Midwest Symposium on*, vol. 0, pp. 687–690, 2009.

[9] M. Luby, T. Gasiba, T. Stockhammer, and M. Watson, "Reliable multimedia download delivery in cellular broadcast networks," *IEEE Trans. Broadcast.*, vol. 53, no. 1, pp. 235–246, March 2007.