Hardware Implementation of Matrix Inversion for Raptor Decoder on Embedded System

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

***Abstract*—Raptor codes have been proven very suitable for mobile multimedia content delivery, and yet they have not been investigated in the context of embedded systems where the energy dissipation is as important as the timing performance. At the heart of Raptor codes is the matrix inversion operation. This paper proposes a dedicated hardware block, for two matrix inversion algorithms, as a part of Raptor decoder implemented on a system on a chip (SoC) platform with a soft-core embedded processor. The performance, energy profile and resource implication are analyzed and compared with a pure software implementation.**

I. Introduction

Multimedia on mobile devices requires secure delivery of various sized data with minimum negotiation overhead. Here is where Raptor codes [1], [2] have come quite useful and outperformed the already well known coding schemes. Recently there have been two standards, namely 3GPP MBMS (Multimedia Broadcast/Multicast Services) [3] and DVB-H [4], which have included systematic Raptor codes in their specifications for content delivery.

Although Raptor codes are growing as a preferred mobile multimedia delivery scheme, experimental data relating to their implementations are reported from simulation on a workstation platform. As far as we are aware, their hardware implementations for mobile embedded systems have not yet been investigated.

This paper looks at the implementation of Raptor codes on an embedded system platform, where resources in terms of computational and power dissipation are limited. The most demanding part of the Raptor decoder, profiled to take 92% of the decoding time, is the matrix inversion operation. This motivates us to look for a hardware implementation of the matrix inversion that would reduce both the decoding time and the energy dissipation. We propose such dedicated hardware blocks for the well known Gaussian elimination (GE) algorithm and the efficient matrix inversion algorithm (SA) proposed in [3], [4]. The relative performance of these two algorithms in terms of decoding time, power, energy and area trade offs are demonstrated. We propose and design hardware enhancements for GE and SA based on their algorithmic structures. Finally, based on the profiling data, suitability assessments are made for the implementation of GE and SA on an embedded system platform.

The chosen embedded system platform is a NIOS soft-core processor, running on an *EP1S40F780C5* Altera Stratix FPGA, with $41,250$ logical elements, $3,423,744$ total memory bits ($2,097,152$ bits maximum single memory size), 14 DSP blocks and 129 (9-bit) embedded multipliers. This device is housed on the NIOS Development Board Stratix Professional Edition with 16MB of SDRAM memory. NIOS soft-core processor can be augmented with custom instructions and additional peripheral devices. Fig. 1 depicts the high level block diagram of the embedded SoC platform for the implementation of Raptor codes.

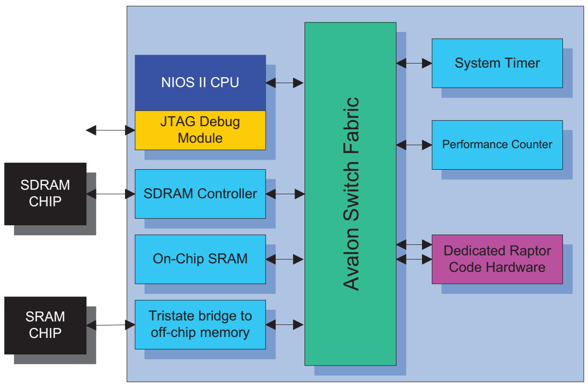

Fig. 1. Raptor codes on hardware/software NIOS embedded system

This paper is organized as follows. In Section II, we briefly explain the operation of a Raptor decoder. Section III describes and presents the details of GE and SA, the two algorithms for the matrix inversion used in this paper. Sections IV presents the hardware implementation performance results in terms of execution time, power and energy, and hardware resources and compares them to the software implementation presented in [5]. Section V concludes the paper.

II. Systematic Raptor Decoding

A Raptor code can be viewed as a regular linear block code, which makes it possible to be represented by a generator matrix. 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. 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, where K is the number of source symbols). 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 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.

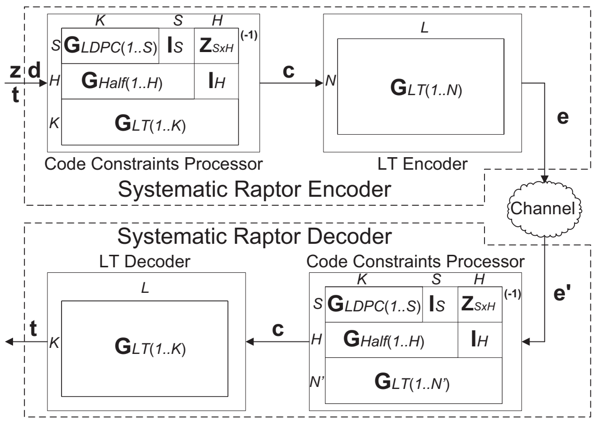

Fig. 2. Block diagram of the systematic Raptor codes

III. Matrix Inversion Algorithms

The most common matrix inversion algorithm is GE [6]. The pseudo code for a GF(2) GE algorithm where elimination and backward substitution are performed together, is shown in Algorithm I .The main operations involved in this algorithm are "row exchange" and "row XOR" (Exclusive-OR).

The specifications in [3] and [4], recommend SA as a more efficient technique for matrix inversion. A version of SA technique is presented in Algorithm II. The operation of SA is as follows.

In *Phase I* matrix $\mathbf{A}$ is reduced to the following form:

$$\mathbf{A}_{Phase_I(M \times L)} = \left[\begin{array}{c|c} \mathbf{I}_i & \\ \hline \mathbf{Z}_{(M-i) \times i} & \mathbf{U}_{M \times u} \end{array} \right] \tag{3}$$

This reduction is performed iteratively, by first relocating the rows containing the minimum number of "1s" to the top, and then moving the first column having "1" in this row to

Algorithm 1 The Gaussian Elimination algorithm (GA) for matrix inversion over GF(2)

```
Require: $A \in \{0,1\}^{n \times m}$
 1: for $i = 0 : n-1$ do
 2:     $j = i$;
 3:     while $a_{ji} == 0$ do
 4:         $j = j + 1$;
 5:     if $i \neq j$ then
 6:         row_exchange($\vec{a}_i$,$\vec{a}_j$);
 7:     for $(k = 0 : n-1) \& (k \neq i)$ do
 8:         if $a_{ki} == 1$ then
 9:             for $l = 0 : m-1$ do
10:                 $a_{kl} = a_{kl} \bigoplus a_{il}$
```

Algorithm 2 The efficient matrix inversion algorithm (SA) over GF(2) proposed in [3], [4]

```
Require: $A \in \{0,1\}^{n \times m}$
Ensure: $i = 0; u = 0$;
 1: Phase I
 2: while $i + u < m$ do
 3:     $r = 1$;
 4:     while $r \leq m$ do
 5:         for $s = i : n-1$ do
 6:             if (count_ones_&_store_pos($s$) == $r$) then
 7:                 break;
 8:         if (row_found) then
 9:             break;
10:         else
11:             $r = r + 1$;
12:     if $(i \neq s)$ then
13:         row_exchange($\vec{a}_i$,$\vec{a}_s$);
14:     if $(a_{ii} == 0)$ then
15:         col_exchange($i$,$ones\_col[0]$);
16:     for $h = 1 : r-1$ do
17:         col_exchange($(m - u - 1)$ , ones_col[$h$]);
18:         $u = u + 1$;
19:     for ($k$ =$(i+1)$: $n-1$) do
20:         if $a_{ki} == 1$ then
21:             for $l = 0 : m-1$ do
22:                 $a_{kl} = a_{kl} \bigoplus a_{il}$
23:     $i = i + 1$;
24: Phase II
25: Gaussian Elimination
    on submatrix $S = [i+1 : n-1][i+1 : m-1]$
26: Phase III
27: for $(k = 0 : i)$ do
28:     for $(s = i+1 : m-1)$ do
29:         if $a_{ks} == 1$ then
30:             $a_{ks} = a_{ks} \bigoplus a_{ss}$
```

the beginning at column location i, and the remaining columns with "1" to the end of the row at column locations $m-u-1$. Note that i and u are initialized to 0. While, in each row, i increments only once per row, u can increment multiple times.

In each iteration of the algorithm one row from the top is excluded from the consideration. Further, the count of "1s" within a row is confined to columns i to $(m-u-1)$. *Phase I* is completed when $(L = i + u)$.

In *Phase II* submatrix $\mathbf{U}$ is partitioned into lower and upper submatrices $\mathbf{U}'_{i\times u}$ and $\mathbf{U}''_{M-i\times u}$, respectively. The lower matrix $\mathbf{U}''_{M-i\times u}$ is transformed into the identity matrix $\mathbf{I}_u$ through the normal Gaussian elimination technique. The $(M-L)$ rows that are left below $\mathbf{I}_u$ are discarded. The form of the matrix produced at the end of *Phase II* is:

$$A_{Phase_{II}} = \begin{bmatrix} \mathbf{I}_i & \mathbf{U}_{i\times u} \\ \mathbf{Z}_{u\times i} & \mathbf{I}_u \end{bmatrix} \tag{4}$$

In *Phase III* the upper matrix U' is zeroed by XOR of its individual rows with the sufficient number of rows from the lower matrix $\mathbf{I}_u$.

$$A_{Phase_{III}} = \begin{bmatrix} \mathbf{I}_i & \mathbf{Z}_{i\times u} \\ \mathbf{Z}_{u\times i} & \mathbf{I}_u \end{bmatrix} \tag{5}$$

It was shown in [5] that for pure software implementation and the "PACKED WORD" memory organization (where 32 matrix elements are packed together into a single 32-bit memory word) the simple GE algorithm outperforms SA by a factor of 20.48.

IV. Hardware Implementation

The inversion of the *Code Constraints Processor* matrix in Fig. 2 has been profiled to be the most time consuming part of the Raptor decoder. To reduce this computational bottleneck, in what follows, a dedicated hardware block for matrix inversion is proposed (*Dedicated Raptor Code Hardware* in Fig. 1), and its performance, power, energy dissipation, and the required hardware resources are presented and analyzed.

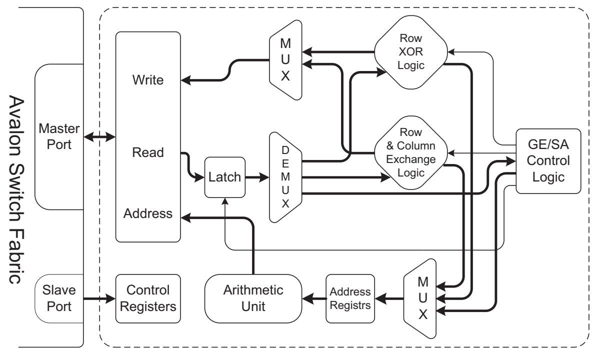

Fig. 3. Hardware accelerator block diagram

Fig 3 shows the block diagram of the hardware accelerator block. The Avalon switch fabric uses a slave port to set the *Control Registers* that initialize the hardware accelerator. During the initialization the size and initial addresses for matrix and vectors are set. *Control Registers* also control operation 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 processor, and send interrupts to NIOS 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. They share a common address generation path through the *Arithmetic Unit* that contains two 32-bit adders and one 16-bit multiplier. The *Latch* circuit only exists in the SA version of the hardware accelerator.

A. Performance

SA algorithm includes a procedure for counting the number of "1s" in the matrix rows (Phase 1, lines 4 to 11) in Algorithm II). After a memory read access the Avalon switch fabric does not retain the data word just read, but goes into the "high impedance" state until the next read cycle. Since we use the "PACKED WORD" memory organization we need a mechanism to retain the read data to avoid multiple memory accesses and bit masking to acquire all the bits in a row.

1) Memory access interface enhancement: The proposed hardware block in Fig. 3 for the SA algorithm employs a special circuit that latches the data word read from the memory into the *Latch* until the next word is needed. The subsequent memory accesses for the matrix elements are made to this register, as long as the required matrix entries are in the current word. Fig. 4 depicts the performance of the SA algorithm with and without the addition of this circuit for several values of K. This simple enhancement improves the performance by a factor of 5.5.

It should be noted that the performance improvement is solely due to the "the count of 1s" operation in Phase I of Algorithm II for SA where memory accesses are sequential. However, there is no performance improvement in Phases II and III of this algorithm, and Algorithm I for GE, where "row exchange", "column exchange" and 'row XOR" operations involve search for the elements of the matrix from non sequential addresses.

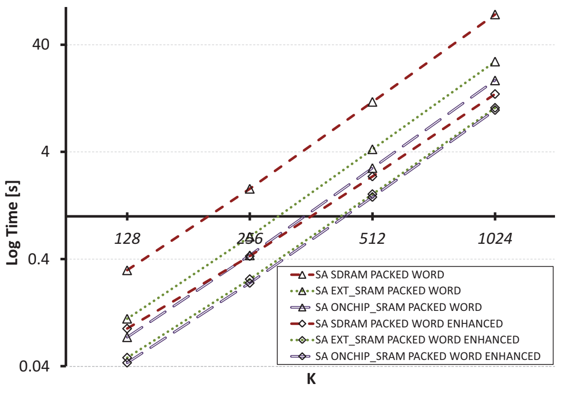

Fig. 4. Performance of the hardware–enhanced Phase I of SA algorithm

At 100 and 35 MHz speeds for the SDRAM, and the logic, the hardware versions of GE and SA perform better than their software versions, as reported in [5], by factors of 1.76 and 20.31, respectively. This is in spite of the fact the the software versions of the algorithms run on a 100 MHz NIOS processor; a clock rate almost 3 times faster.

It should also be noted that hardware version of GE performs better than the hardware version of SA by a factor of 1.48. This ratio is less than the equivalent 20.48 ratio for the software implementations in [5]. The reduction in the performance ratio indicates that the SA algorithm benefits far more that the GE in switching from software to the hardware platform. This is mainly due to the speed up in the "the count of 1s" procedure due to the simple enhancement.

2) Effect of SDRAM and logic clock rates: Fig. 5 shows the effect of the SDRAM speed on GE and SA algorithms for ($K = 1024$). The data is plotted for three different logic clocks - 25, 30 and 35 MHz. While both algorithms benefit from a faster SDRAM, the effect of memory speed is more noticeable on the operation of the GE algorithm. This is specially true for the slower SDRAM region where the plots for SA and GE cross over each other. The point of cross over moves to right for the higher logic speed. This indicates that a faster logic requires a faster SDRAM if the relative superior performance of GE algorithm over SA were to be maintained. This highlights the fact that the SA algorithm benefits more from a faster logic.

From Fig. 5 it can be inferred that the enhancement in the "the count of 1s" procedure has the following effect. The increase in the logic speed from 25 MHz to 35 MHz is slightly more effective in improving the performance of the SA algorithm. This is because the number of slow memory accesses are significantly reduced due to the enhancement which causes the computation to be more dependent on the speed of the logic.

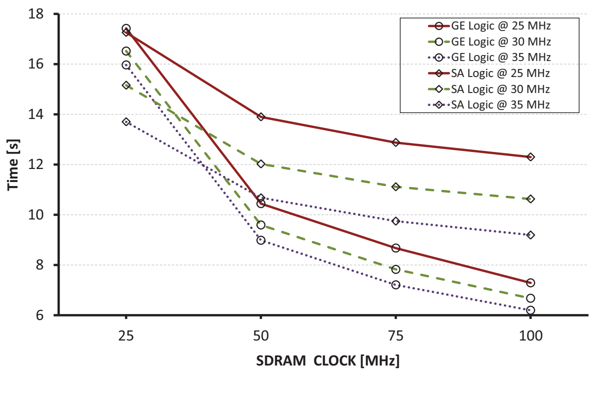

Fig. 5. Effect of SDRAM memory speed on GE and SA algorithms

B. Power and Area

Table I shows the power, energy and hardware resource requirements for the FPGA implementation of the NIOS soft-core processor with the proposed dedicated hardware accelerator block for the matrix inversions, with the SDRAM and logic operating at the speeds of 100 and 35 MHz, respectively. The NIOS processor is placed in idle state when the control is passed to the dedicated hardware accelerator. In spite of the addition of a new block, the power dissipation is less for all cases, when compared to the corresponding software implementations in [5]. From the data in Table I it can be seen that the energy required for the matrix inversion is 1.44 times less for the GE algorithm compared to the SA.

The implementation with the dedicated hardware block for GE needs several times less energy to invert the same matrix compared to entirely software implementation [5]. The saving factor is 3.43. The similar energy saving factor for SA is, impressively, much higher at 22.04.

TABLE I
POWER, ENERGY AND RESOURCES FOR THE HARDWARE IMPLEMENTATION FOR ($K = 128$) WITH 100 MHZ SDRAM AND 35 MHZ LOGIC SPEEDS.

GE	**Power [mW]**	1,564.71
SA	**Power [mW]**	1,564.02
GE	**Energy [mJ]**	53.86
SA	**Energy [mJ]**	77.64
Hardware	**Logic Elements**	8,172
Resources	**Memory Bits**	47,360

V. CONCLUSION

This paper has evaluated the performance of Raptor decoder on embedded system. The performance of two matrix inversion algorithms on dedicated hardware block for embedded system have been presented. It was shown that the hardware implementation achieves better performance with less energy compared to a software implementation. Furthermore, it was found that contrary to the recommendation, based on the profiling on a workstation platform, even under hardware implementation the Gaussian elimination performs significantly better than the alternative reportedly more efficient algorithm, in terms of execution speed and energy saving.

ACKNOWLEDGMENT

This work was, in part, supported by the Center for Distributed Sensor Network at GIST.

REFERENCES

[1] A. Shokrollahi, "Raptor codes," *IEEE Trans. Inf. Theory*, vol. 52, pp. 2551–2567, Jun. 2006.
[2] M. Luby, M. Watson, T. Gasiba, T. Stockhammer, and W. Xu, "Raptor codes for reliable download delivery in wireless broadcast systems," in *Third IEEE Consumer Communications and Networking Conference*, vol. 1, Jan. 2006, pp. 192–197.
[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] T. Mladenov, S. Nooshabadi, A. Dassatti, and K. Kim, "Analysis and implementation of raptor codes on embedded system," in *Global Telecommunications Conference. GLOBECOM '09. IEEE*, 2009.
[6] D. Parkinson and M. Wunderlich, "A compact algorithm for gaussian elimination over GF(2) implemented on highly parallel computers," *Parallel Computing*, vol. 1, no. 1, pp. 65–73, Aug. 1984.