Instruction Set Extensions for Reed-Solomon Encoding and Decoding

Suman Mamidi and Michael J. Schulte
Dept. of ECE
University of Wisconsin-Madison
{mamidi, schulte}@cae.wisc.edu
http://mesa.ece.wisc.edu

Daniel Iancu, Andrei Iancu, and John Glossner
Sandbridge Technologies
White Plains, NY
{diancu, aiancu, glossner}@sandbridgetech.com
http://www.sandbridgetech.com

Abstract

Reed-Solomon codes are an important class of error correcting codes used in many applications related to communications and digital storage. The fundamental operations in Reed-Solomon encoding and decoding involve Galois field arithmetic, which is not directly supported in general purpose processors. On the other hand, pure hardware implementations of Reed-Solomon coders are not programmable. In this paper, we present a novel Reed-Solomon encoding algorithm, which avoids polynomial division and is suitable for Single Instruction Multiple Data (SIMD) processors. We also propose four new instructions for Galois field arithmetic. We show that by using the new instructions, we can speedup Reed-Solomon decoding by more than a factor of 12, while still maintaining programmability.

1 Introduction

Reed-Solomon codes are used to perform Forward Error Correction (FEC). FEC in general introduces redundancy in data before it is transmitted. The redundant data, known as check symbols or parity symbols, are transmitted with the original data to the receiver. This type of error correction is widely used in data communication applications, such as Digital Video Broadcast (DVB) and optical carriers like OC-192. The number and type of errors that can be corrected depends on the characteristics of the Reed-Solomon code.

Reed-Solomon codes are referred to as RS(N, K) codes with M-bit symbols. This means the encoder takes K data symbols of M bits each and adds $N - K$ parity symbols to make an N symbol codeword. On the receiving end, the Reed-Solomon decoder can correct up to T symbols that contain errors in the codewords, where

$$T = \frac{(N - K)}{2} \tag{1}$$

Reed-Solomon codes are particularly well suited to correcting burst errors, in which a continuous sequence of bits is received with errors.

The Reed-Solomon coding algorithms described in this paper are tuned to the DVB standard requirements for the broadcast of MPEG2 transport packets [1]. In this standard, the Reed-Solomon error correction coding technique is employed on 188-byte packets $(K = 188, M = 8)$ with the capability of correcting 8 errors per packet $(T = 8)$. This requires 16-check symbols, resulting in a total codeword length of 204 bytes $(N = 204)$.

Reed-Solomon encoding and decoding can be carried out in software or in customized hardware. Software implementations on general purpose processors often do not meet the high-speed requirements of Reed-Solomon encoding and decoding for DVB. General purpose processors do not support Galois field arithmetic, which is fundamental to Reed-Solomon encoding and decoding. There are several approaches to perform Galois field multiplication in software. One approach is to precompute all possible results for two M-bit inputs, store them in a 2-D array, and then perform Galois field multiplication via a lookup with the inputs as indices to this array. This approach requires large amounts of memory, which are not usually available in embedded systems. Another approach involves a test for zero, two log table lookups, modulo addition, and an antilog table lookup. Yet another approach involves computing the Galois field product using a sequential and-xor-shift algorithm.

For high-speed Reed-Solomon decoding, a number of hardware implementations are available [2]. Most common hardware implementations are based on the use of a Linear Feedback Shift Register (LFSR) that provides a convenient way to perform polynomial division [3]. A summary of hardware and software implementations for Reed-Solomon coding is presented in [4]. Hardware schemes, however, have limited flexibility.

The authors of [5] introduce a hardware-software codesign approach to perform Reed-Solomon encoding and decoding. They propose two new instructions for general purpose processors that allow low power Reed-Solomon encoding and decoding. [6] proposes instruction set extensions for the configurable Xtensa processor that facilitate software implementations on a single-issue processor to achieve high-speed Reed-Solomon decoding.

In this paper, we present a novel algorithm to perform Reed-Solomon encoding on Single Instruction Multiple Data (SIMD) processors. This algorithm does not use Galois field polynomial division to calculate the codewords. We also present instruction set extensions for Galois field arithmetic to the Sandblaster Digital Signal Processor (DSP) [7]. These instructions improve the performance of the Reed-Solomon decoder by more than a factor of 12.

This paper is organized as follows: Section 2 discusses the Reed-Solomon encoding and decoding algorithms for SIMD architectures. Section 3 gives a brief overview of the Sandbridge processor and its vector processing unit. Section 4 discusses the proposed instruction set extensions and the experimental results. Section 5 concludes the paper.

2 The Reed-Solomon Encoder and Decoder Algorithms

2.1 Encoder Algorithm

Cyclic codes, such as Reed-Solomon codes, are described in numerous coding theory books [8] [9] [10]. Given a data polynomial $a(x)$ of degree $k < n$, $n = 2^m$, in Galois field $GF(2^m)$ and a code generating polynomial $g(x)$ of degree p, where $p \leq n - k$ and

$$g(x) = \prod_{i=0}^{p-1} (x + \alpha^i), \quad a(x) = \sum_{i=0}^{k} a_i x^i \tag{2}$$

with α^i successive unity roots in $GF(2^m)$ and a_i elements of the same field, the systematic encoding of $a(x)$ is given by

$$C(x) = a(x)x^{n-k} - R(x) \tag{3}$$

where R(x) is the remainder of the division $a(x)x^{n-k}$ by $g(x)$.

Given two polynomials $P(x)$ and $G(x)$ with coefficients in the $GF(2^m)$ field,

$$P(x) = \sum_{i=0}^{M} a_i x^i, G(x) = \prod_{i=0}^{N-1} (x + \alpha^i) \tag{4}$$

the remainder $R(x)$ of $P(x)$ divided by $G(x)$ can be expressed as:

$$R(x) = \sum_{i=0}^{N-1} \beta^{(i)} \prod_{k=i}^{N-1} (x + \alpha^k) \tag{5}$$

where $N = n - k$, α^i are the roots of the polynomial $G(x)$, and $\beta^{(i)}$ are GF polynomials generated by successive Horner reductions of $P(x)$ by α^i.

Successive Horner reductions of the polynomial $P(x)$ by α^i are illustrated in Table 2.1. The entries in the first row are the coefficients of the polynomial to be reduced and the entries in the rightmost column are the unity roots. If the remainder $\beta^{(j)}$ is zero, then α^j is a root of $P(x)$. The entries of the second row are the coefficients of the reduced polynomial with α^0, and so on. Each row represents the coefficients of the polynomial to be reduced with the root of the $g(x)$ polynomial in the corresponding position of the rightmost column. The result of the reduction is the next row.

The encoding algorithm is performed as follows:

- Perform the successive Horner reductions on the polynomial $a(x)x^{n-k}$ to obtain the coefficients $\beta^{(i)}$,
- Calculate $R(x)$ as described in Equation 5,
- Calculate $C(x)$ as described in Equation 3

The steps for Reed-Solomon encoding are shown in Figure 1. The text in parentheses indicates the function name that implements that particular block and the percent of the total encoding time spent by each function in the original software implementation on the Sandblaster DSP.

2.2 Decoder Algorithm

Suppose the received code word is

$$C(x) = \sum_{i=0}^{n-1} c_i x^i \tag{6}$$

Instead of checking the validity of the code word by encoding the data portion of the received code word and then comparing the computed parity bits to the received parity bits, the decoder starts by directly computing the syndromes of the received codeword using Equation 7.

$$S_i = C^T \Sigma_i \tag{7}$$

where C is a vector formed from the coefficients of the data polynomial and Σ_i are the syndrome vectors computed using,

$$\Sigma_i = \sum_{k=0}^{2^n-1} x^k (\alpha^k)^i \tag{8}$$

If all the syndromes are zeroes, there are no errors. If any syndrome differs from zero, the following syndrome matrix equation is solved, where Λ_i are the coefficients of the error locator polynomial.

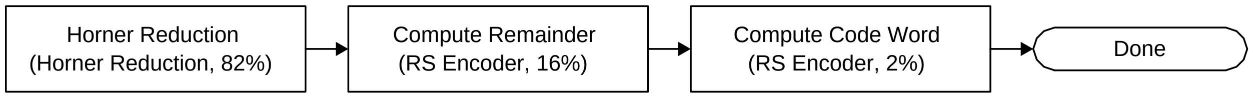

Figure 1. Reed-Solomon Encoder

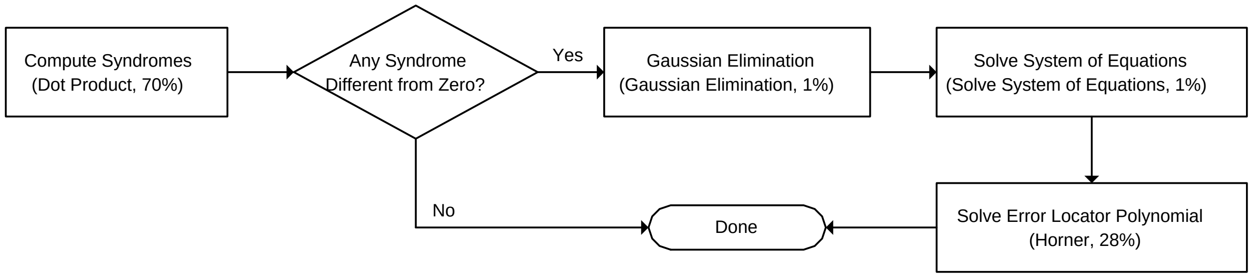
Compute Syndromes
(Dot Product, 70%)
Any Syndrome
Different from Zero?
Yes
No
Gaussian Elimination
(Gaussian Elimination, 1%)
Solve System of Equations
(Solve System of Equations, 1%)
Solve Error Locator Polynomial
(Horner, 28%)
Done

Figure 2. Reed-Solomon Decoder

a_n	a_{n-1}	a_{n-2}	...	a_0	α^0
a_n	$a_n\alpha^0 + a_{n-1}$	$(a_n\alpha^0 + a_{n-1})\alpha^0 + a_{n-2}$	...	β^0	α^1
a_n	$a_n\alpha^1 + a_n\alpha^0 + a_{n-1}$	$(a_n\alpha^1 + a_n\alpha^0 + a_{n-1})\alpha^1 + (a_n\alpha^0 + a_{n-1})\alpha^0 + a_{n-2}$	...		α^2
.	.	.	.	.	.
a_n	$\beta^{(N-1)}$	...	...	...	α^{N-1}

Table 1. Horner Reduction Table

$$\begin{pmatrix} S_0 & S_1 & \cdot & S_{\frac{N}{2}-1} \\ S_1 & S_2 & \cdot & S_{\frac{N}{2}} \\ S_2 & S_3 & \cdot & S_{\frac{N}{2}+1} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ S_{\frac{N}{2}-2} & S_{\frac{N}{2}-1} & \cdot & S_{N-1} \\ S_{\frac{N}{2}-1} & S_{\frac{N}{2}} & \cdot & S_{N-2} \end{pmatrix} \begin{pmatrix} \Lambda_{\frac{N}{2}-1} \\ \Lambda_{\frac{N}{2}-2} \\ \Lambda_{\frac{N}{2}-3} \\ \cdot \\ \cdot \\ \cdot \\ \Lambda_1 \\ \Lambda_0 \end{pmatrix} = \begin{pmatrix} S_{\frac{N}{2}} \\ S_{\frac{N}{2}+1} \\ S_{\frac{N}{2}+2} \\ \cdot \\ \cdot \\ \cdot \\ S_{N-2} \\ S_{N-1} \end{pmatrix} \tag{9}$$

The above system of equations is solved using Gaussian elimination to obtain Equation 10.

$$\begin{pmatrix} 1 & \Xi_1 & \cdot & \Xi_{\frac{N}{2}-1} \\ 0 & 1 & \cdot & \Xi_{\frac{N}{2}} \\ 0 & 0 & \cdot & \Xi_{\frac{N}{2}+1} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \cdot & \Xi_{N-1} \\ 0 & 0 & \cdot & 1 \end{pmatrix} \begin{pmatrix} \Lambda_{\frac{N}{2}-1} \\ \Lambda_{\frac{N}{2}-2} \\ \Lambda_{\frac{N}{2}-3} \\ \cdot \\ \cdot \\ \cdot \\ \Lambda_1 \\ \Lambda_0 \end{pmatrix} = \begin{pmatrix} S_{\frac{N}{2}} \\ S_{\frac{N}{2}+1} \\ S_{\frac{N}{2}+2} \\ \cdot \\ \cdot \\ \cdot \\ S_{N-2} \\ S_{N-1} \end{pmatrix} \tag{10}$$

This system of equations is then solved to produce

$$\Lambda_0 = \Xi_{N-1},\ \Lambda_1 = \Xi_{N-1}\Lambda_0 + \Xi_{N-1},\ ... \tag{11}$$

The error locator polynomial $\Lambda(x)$ is solved using Chien search. The inverse of the solution of the error locator polynomial represents the position of the bit error. The decoding algorithm is performed as shown in Figure 2. The text in the parenthesis in Figure 2 indicates the function name that implements that particular block and the percent of the total decoding time spent by that function in the original software implementation on the Sandblaster DSP. The percentages assume a worst case scenario where at least one error is present in the received codeword, which results in non-zero syndromes. In the more common case, 98% of the execution time is spent on computing the syndromes.

3 Overview of the Sandblaster Processor

Sandbridge Technologies has designed a multithreaded processor capable of executing DSP, embedded control, and Java code in a single compound instruction set optimized for handset radio applications [7]. The Sandbridge Sandblaster design overcomes the deficiencies of previous approaches by providing substantial parallelism and throughput for high-performance DSP applications, while maintaining fast interrupt response, high-level language programmability, and low power dissipation.

Figure 3 shows a block diagram of the Sandblaster microarchitecture. The processor is partitioned into three units; an instruction fetch and branch unit, an integer and load/store unit, and a SIMD vector unit. The design uti-

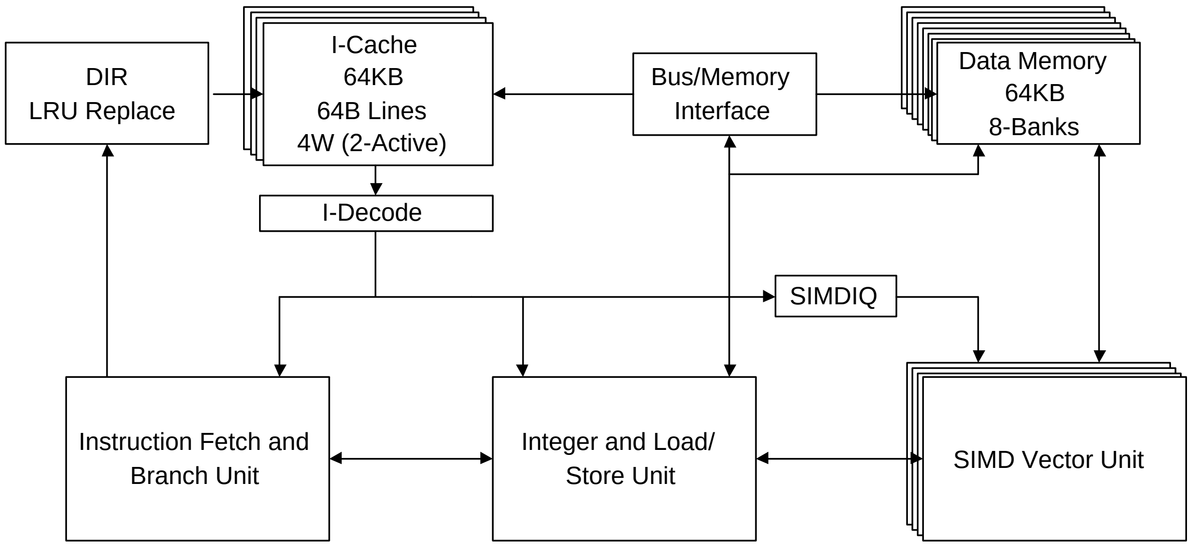

Figure 3. Sandblaster Microarchitecture

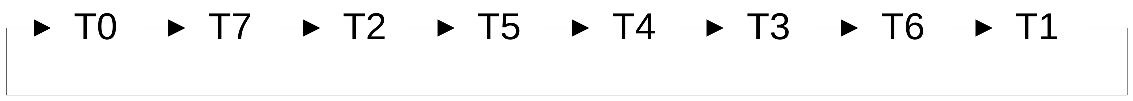

Figure 4. Token Triggered Multithreading

lizes a unique combination of techniques including hardware support for multiple threads, SIMD vector processing, and instruction set support for Java code. Program memory is conserved through the use of powerful compounded instructions that may issue multiple operations per cycle. The resulting combination provides for efficient execution of DSP, control, and Java code. The proposed instructions to speedup Galois field operations are targetted for the Sandbridge SIMD Vector Unit.

The Sandblaster processor uses a form of interleaved multithreading, called Token Triggered Threading (T^3). As shown in Figure 4, with T^3 each thread is allowed to simultaneously execute an instruction, but only one thread may issue an instruction on a cycle boundary. This constraint is also imposed on round robin threading. What distinguishes (T^3) threading is that each clock cycle a token indicates the subsequent thread that is to issue an instruction. Tokens may be sequential (e.g. round robin), even/odd, or based on other communication patterns. Compared to SMT, T^3 has much less hardware complexity and power dissipation, since the method for selecting threads is simplified, only a single compound instruction issues each clock cycle, and dependency checking and bypass hardware are not needed.

Figure 5 shows a high-level block diagram of the SIMD vector processing unit (VPU), which consists of four vector processing elements (VPEs), a shuffle unit, a reduction unit, and a multithreaded 2-bank accumulator register file. The four VPEs perform arithmetic and logic operations in SIMD fashion on 16-bit, 32-bit, and 40-bit fixed-point data types. High-speed 64-bit data busses allow each PE to load or store 16 bits of data each cycle in SIMD fashion. Support for SIMD execution significantly reduces code size, as well as power consumption from fetching and decoding instructions, since multiple sets of data elements are processed with a single instruction.

The proposed Galois field instructions are vector instructions that go through eight pipeline stages. For example, a VGFMAC instruction goes through the following stages: Instruction Decode, VRF Read, EX1, EX2, EX3, EX4, Transfer, and Write Back. The Transfer stage is needed due to the long wiring delay between the bottom of the VPU and the Vector Register File (VRF). Since there are eight cycles between when consecutive instructions issue from the same thread, the result from one instruction in a thread is guaranteed to have written back to the VRF by the time the next instruction in the same thread is ready to read it. Thus, the long pipeline latency of the VPEs is effectively hidden, and no data dependency checking or bypass hardware is needed. Even if there is a data dependency between the two instructions, there is no need to stall the second instruction, since the first instruction has completed the Write Back stage before the second instruction enters the VRF Read stage.

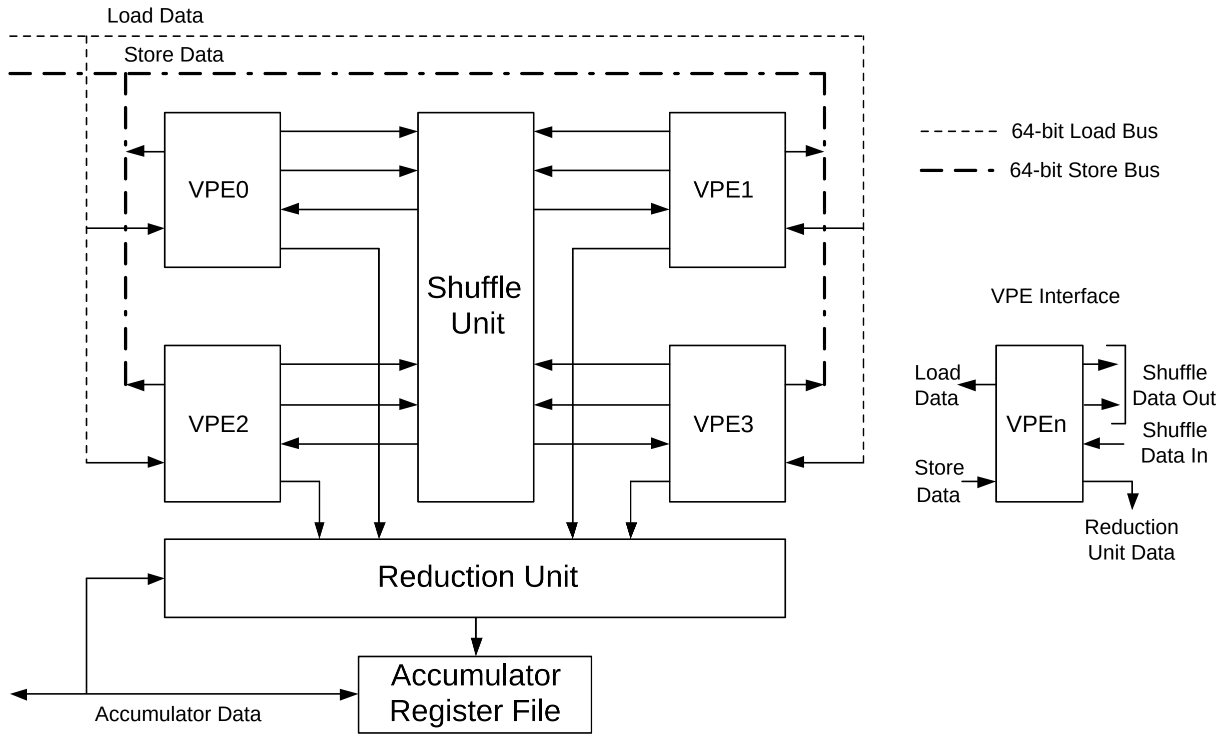

Figure 5. SIMD Vector Processing Unit

Instruction	Parallel Operations	Operations
VGFMUL	4	$v_t = v_a \otimes v_b$
VGFMAC	4	$v_t = (v_a \otimes v_b) \oplus v_c$
VGFMUL2	8	$v_t = v_a \otimes v_b$
VGFMAC2	8	$v_t = (v_a \otimes v_b) \oplus v_c$

Table 2. Proposed Instructions

Instruction	Area	Delay
VGFMUL	$42752\mu^2$	$1.22ns$
VFGFMAC	$49852\mu^2$	$1.32ns$
VGFMUL2	$85504\mu^2$	$1.22ns$
VGFMAC2	$99704\mu^2$	$1.32ns$

Table 3. Area and Delay Estimates

4 Proposed Instructions and Experimental Results

In this section, we investigate the performance benefits of extending the Sandbridge Vector Processing Unit to perform SIMD Galois field operations. The instructions investigated are VGFMUL, VGFMAC, VGFMUL2, and VGFMAC2. Table 2 summarizes the instructions and their functionality. In this table, $x \oplus y$ denotes x XOR y and $x \otimes y$ denotes GF multiplication of x with y to generate a 15-bit intermediate product. The 15-bit intermediate result is then reduced using an irreducible polynomial to generate an 8-bit product.

The instructions are implemented by adding parallel GFMUL or GFMAC units to the SIMD Vector Processing Unit. The VGFMUL/VGFMAC instructions use one VGFMUL/VGFMAC unit per VPE and the VGFMUL2/VGFMAC2 instructions use two VGFMUL/VGFMAC units per VPE. Table 3 shows the area and delay estimates of the hardware that implements the proposed instructions. The estimates are based on synthesis results generated by Synopsys Design Compiler when the hardware was optimized for delay in LSI Logic's Gflxp 0.11 μ standard cell library.

In Table 4, we give the speedup of the Reed-Solomon decoder and its important functions for the proposed Galois field instructions over the original implementation. The original implementation used look up tables to perform Galois field multiplication. In the worst case, the syndromes are different from zero and the Reed-Solomon decoder goes through the path that computes the error locator polynomial shown in Figure 2. VGFMAC2 speeds up Dot Product 27.7 times and Horner 12.4 times. This translates to a 12.7x improvement in the Reed-Solomon decoder, with a total of 6953 cycles/thread for each 204-byte packet. The common case, however, is when all the syndromes are zero

Function	VGFMUL	VGFMAC	VGFMUL2	VGFMAC2
Dot Product	10.3	19.3	18.0	27.5
Horner	8.5	10.4	10.6	12.4
Solve System of Equations	4.1	4.1	4.2	4.2
Gaussian Elimination	5.2	6.4	4.4	4.6
Entire RS Decoder	8.4	12.5	11.5	12.7

Table 4. Speedup Over Baseline Implementation

and the Reed-Solomon decoder is done after checking the syndromes. The speedup of 27.5 times for Dot Product translates to a 20x speedup of the complete Reed-Solomon decoder, with a total of 4171 cycles/thread for each packet. Every thread of the Sandblaster processor runs at a minimum of 75MHz. Since there are 8 independent threads, the Sandblaster processor can perform worst case Reed-Solomon decoding at over 150Mbits/second and common case decoding at over 230Mbits/second.

5 Conclusion

In this paper, we present a novel algorithm to perform Reed-Solomon encoding that does not require Galois Field polynomial division and is suitable for SIMD processors. We also propose four instruction set extensions that speed up Galois field arithmetic. Additionally we analyze the performance benefits of the proposed instruction set extensions on the Reed-Solomon decoder. The instruction set extensions allow high-speed Reed-Solomon decoding, while maintaining programmability.

References

[1] "Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television." European Standard (Telecommunications series), 2001.

[2] T. D. Wolf, "Programmable, Reconfigurable DSP Implementation of a Reed-Solomon encoder/decoder." United States Patent No. 6385751, 2002.

[3] Y. Katayama and S. Morioka, "One-Shot Reed-Solomon Decoder," *Annual Conference on Information Science and Systems*, vol. 33, pp. 700–705, 1999.

[4] V. K. Bhargava, T. Le-Ngoc, D. Gregson, "Software and Hardware R-S Codec for Wireless Communications," *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, vol. 2, pp. 739–740, 1997.

[5] L. Song, K. K. Parhi, I. Kuroda, T. Nishitani, "Hardware/Software Codesign of Finite Field Datapath for Low-energy Reed-Solomon codecs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 2, pp. 160–172, 2000.

[6] H. M. Ji, "An Optimized Processor for Fast Reed-Solomon Encoding and Decoding," *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 3, no. III, pp. 3097–3100, 2002.

[7] M. J. Schulte, J. Glossner, S. Mamidi, M. Moudgill, and S. Vassiliadis, "A Low-Power Multithreaded Processor for Baseband Communication Systems," *Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation, Lecture Notes in Computer Science*, vol. 3133, pp. 393–402, 2004.

[8] E. R. Berlekamp, "Algebraic Coding Theory," *Aegean Park Press*, 1984.

[9] R. E. Blahut, "Algebraic Codes for Data Transmission," *Cambridge University Press*, 2003.

[10] S. B. Wicker, "Error Control Systems for Digital Communication and Storage," *Prentice Hall Inc.*, 1995.