A Fast Parallel Reed-Solomon Decoder On a Reconfigurable Architecture

Arezou Koohi,
EECS Department
University of California Irvine
949-8242481
akoohi@ece.uci.edu

Nader Bagherzadeh,
EECS Department
University of California, Irvine
949-8242481
Nader@ece.uci.edu

Chengzi Pan
EECS Department
University of California, Irvine
949-8242481
panc@uci.edu

ABSTRACT

This paper presents a software implementation of a very fast parallel Reed-Solomon decoder on the second generation of MorphoSys reconfigurable computation platform, which is targeting on streamed applications such as multimedia and DSP. Numerous modifications of the first-generation of the architecture have made a scalable computation and communication intensive architecture capable of extracting parallelisms of fine grain in instruction level. Many algorithms and the whole Digital Video Broadcasting base-band receiver as well, have been mapped onto the second architecture with impressing performance. The mapping of a Reed-Solomon decoder proposed in this paper highly parallelizes all of its sub-algorithms, including Syndrome Computation, Berlekamp Algorithm, Chein Search, and Error Value Computation, in a SIMD fashion. The mapping is tested on a cycle-accurate simulator, "Mulate", and the performance is encouragingly better than other architectures. The decoding speed of the RS (255,239,16) decoder using two different methods of GF multiplication can be 1.319Gbps and 2.534Gbps, respectively. Furthermore, since there is no functionality specifically tailored to Reed-Solomon decoder, the result has demonstrated the capability of MorphoSys architecture to extracting Instruction Level Parallelism from streamed applications.

Categories and Subject Descriptors

C.1.2 [**Computer System Organization**]: Processor Architectures, Multiple Data Stream Architectures (Multiprocessors), Single-instruction-stream, Multiple-Data - Stream Processors

General Terms

Algorithms, Performance, Design, Experimentation

Keywords: Reconfigurable Architecture, SIMD Processor, Reed_Solomon codes, Berlekamp Algorithm, Chein Search

1. INTRODUCTION

Toward a coming billion-transistor era, today's computation platform design has already foreseen the end of the road for conventional micro-architectures [4], and numerous new approaches have arisen above the horizon, such as EPIC (Itanium 2) [5], RAW [6], Imagine [7], and VIRAM [8], etc. Many of them target on stream applications, which have already been consuming more than 90% of total computing cycles nowadays [7]. The biggest challenge of architecture design is the scalability, only with which can one follow up the step of Moore's Law. The difficulty of scalability is imposed by slower decrease of wire transmission delay than that of transistor switching delay. This discrepancy requires a new philosophy on design of scalar operand network [9] and memory hierarchy. In this paper, we will introduce the 2nd-generation MorphoSys reconfigurable architecture called M2, a computation and communication intensive platform capable of extracting fine grain parallelisms at the instruction level. Many algorithms and the whole Digital Video Broadcasting base-band receiver as well, have been mapped onto M2 with impressing performance. The mapping of a Reed-Solomon decoder is proposed in this paper. RS codes are powerful block codes widely used as an error correction method in the areas such as digital communication, digital disc error correction, etc. Recently, concatenated codes made up of convolutional codes followed by RS codes have been proved as an efficient way of error correction in wireless data communication systems. We present M2 Architecture at the first section of the paper, including the architecture different parts, M2 implementation and its programming model. Section 3 introduces the RS decoding Algorithms and the mapping procedure on the M2. The section also includes the M2 architecture feasibility for different parts of the Algorithms. In section 4 we compare the result of the RS (255,239,16) decoder with the existing benchmarks including the TI64x and different Asics. The decoding speed of the RS (255,239,16) decoder using two different methods of GF multiplication can be 1.319Gbps and 2.534Gbps, respectively

2. MORPHOSYS RECONFIGURABLE ARCHITECTURE

2.1. Introduction of MorphoSys

MorphoSys is a reconfigurable computation platform targeting computation intensive data parallel applications, including streamed applications. M1 [1-2], the first prototype of MorphoSys, has been used as a platform for many applications such as multimedia, wireless communication, signal processing, and computer graphics. M2, the 2nd generation of MorphoSys, follows the basic concepts of MorphoSys; however, it is redesigned in both scalar operand network and memory hierarchy, thus greatly enhanced in performance. Feedbacks from numerous

kernel and system mappings pointed out M1's bottlenecks, which have been revisited in M2.
M2 architecture consists of three main subsystems: a core processor called TinyRISC, an array of 64 Reconfigurable Cells (RCs) organized in SIMD fashion, and a special data movement unit called Frame Buffer (FB). The programming model is simple. TinyRISC takes charge of the whole Data Control Flow (DCF); RC array and Frame Buffer are only triggered by TinyRISC and executing on their own configurations (called context) continuously for a given number of cycles specified by TinyRISC. The main difference between M1 and M2 is described in [3]. Figure 1 shows the MorphoSys diagram. Main Memory can be either on-chip or off-chip without significant difference of the connection interface, as long as Main Memory is also composed of 64 banks.

2.2. M2 implementation

The following table-1 gives out the characteristics of M2 and compares it with the other processors. Results of first order simulation using Synopsis tools show that the critical-path delay is about 1.8~2.3ns. Hence, 450 MHz is used in our simulation. Other data are either based on M1 implementation and M2 post-synthesis (current status), or projected as our design aim, which are chosen no more aggressive than commercial processors.

Though independently designed, M2 combines the structural advantages of three important architecture parameters: the overall structure of host-processor/slave-computation-fabric, the controlled size of distributed memory within each AlU clusters and the powerful sequential code, which is hard to be parallelized. Second, the modest size of distributed memory enables more AlUs to be integrated in the array and, more importantly, reduces the latency of scalar operand network, which is essential to extract to adopt wide range of optimization techniques, both coarse-grain and fine-grain, and avoids usage of hardware-specific languages as StreamC.

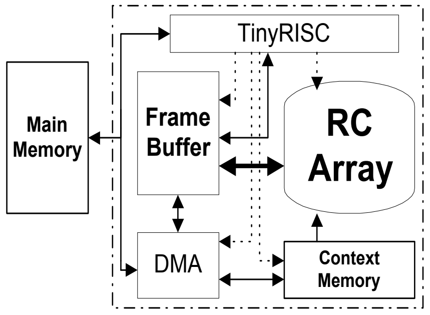


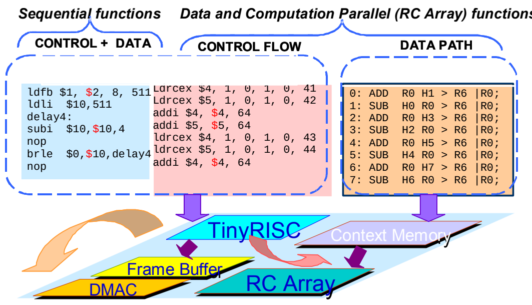
Figure 1. MorphoSys Diagram

Table 1. Comparison of M2 and other architectures* the latency is analyzed in 5-tuple <send occupancy, send latency, network hop latency, receive latency, receive occupancy> [9].

		VIRAM	Imagine	RAW	M2 (with/without on-chip memory)
Parallelism	Parallelism model	Vector	SIMD	MIMD	SIMD
Capability	Peak 16-bit OPS	6.4G	23.7G	3.6G (32-bit)	28.8G
	Clock Speed	200MHz	296MHz	225MHz	450MHz
	Chip area	$15*18mm^2$	$12*12mm^2$	$18*18mm^2$	$8*8mm^2/16*16mm^2$
	# Transistors	120M	21M	122M	20M/120M
	Power	2W(average)	4W	25W	4W(peak MAC)
	Technology	0.18μm	0.15μm	0.15μm	0.13μm
Scalar Operand Network	# Network nodes	8(Banks)	8	16	64
	Total bandwidth	51.2Gbps	75.8Gbps	922Gbps	922Gbps
	Latency*	<0,1,1,1,0>	<0,1,1,1,0>	<0,1,1~6,1,0>	<0,0,1~2,0,0>
	Full permutation	No	Yes	Yes	Yes
Memory Hierarchy	1st level size	64Kb(VRF)	96Kb(LRF)	16.4Kb(RF)	16.4Kb(RF)
	2nd level size	104Mb(DRAM)	1Mb(SRF)	16Mb(Local Mem.)	2Mb(Local Mem.)
	3rd level size	Off-chip	Off-chip	Off-chip	16Mb/off-chip
	1st level bandwidth	205Gbps	758Gbps	230Gbps	1382Gbps
	2nd level bandwidth	51.2Gbps	829Gbps**	334Gbps	461Gbps
	3rd level bandwidth	N/A	12.4Gbps	200Gbps	115Gbps/56.7Gbps

2.3. Programming Model

One potential drawback of M2 is the SIMD model of RC array. However, mapping experience on MorhpoSys, VIRAM, Imagine, and other SIMD extension of general-purpose processor like Pentium MMX, all show that SIMD is sufficient for streamed multimedia applications, not mentioning the much smaller code size than that of MIMD model. For example, mapping of DVB-T system on M2 does not need MIMD generality. Although SPMD feature might make mapping of MPEG-2 decoder easier, other subsystems are applying mostly affine array access and small portion of non-affine but static access involving random communication, which can be done efficiently by the scalar operand network. Since SIMD programming has long been a matured field in general, it is reasonable to suppose that an efficient C compiler for streamed applications is highly possible, with some previous experience already [10-11].Moreover, parallel programming in M2 can adopt domain decomposition or function decomposition, the former more straightforward, while Imagine has to stick with cumbersome function decomposition often experiencing load-unbalanced problem. The following figure illustrates a real code segment for The Berelekamp algorithm operation which also shows programming model of MorphoSys.



```
ldfb $1, $2, 8, 511
ldli  $10,511
delay4:
subi  $10,$10,4
nop
brle  $0,$10,delay4
nop
```

```
Ldrcex $4, 1, 0, 1, 0, 41
Ldrcex $5, 1, 0, 1, 0, 42
addi $4, $4, 64
addi $5, $5, 64
ldrcex $4, 1, 0, 1, 0, 43
ldrcex $5, 1, 0, 1, 0, 44
addi $4, $4, 64
```

```
0: ADD  R0 H1 > R6 |R0;
1: SUB  H0 R0 > R6 |R0;
2: ADD  R0 H3 > R6 |R0;
3: SUB  H2 R0 > R6 |R0;
4: ADD  R0 H5 > R6 |R0;
5: SUB  H4 R0 > R6 |R0;
6: ADD  R0 H7 > R6 |R0;
7: SUB  H6 R0 > R6 |R0;
```

Figure 2- Programming Model in MorphoSys

2.4. Scalability

There is a misunderstanding in many places, that scalability is equal to even distribution of resources. Actually, this is not true in large scale. For example, as the tile number of RAW processor grows, the scalar operand network latency increases, and more severely, traffic load per tile increases, which will finally destroy the scalability. In order to avoid this load increase, one would expect to keep the atomic size of 8X8 basic array and instead use an incremental level of inter-array communication network accommodating this incremental load. This leads to a hierarchical scaling behavior, in each level of which there are both distributed and centralized recourses. The real billion-transistor architecture coming around year 2007 can be made up of 8 M2 together with the counterpart of TinyRISC and operand network at a high level.3 Mapping of Reed-Solomon Decoding Algorithm

3. MAPPING OF REED-SOLOMON DECODING ALGORITHM

3.1 Reed-Solomon Decoding Algorithm [14]

In the RS (n,k) code, n is the block length, k is the transmitted code word length and n-k is the minimum distance over GF (2^n). Decoder is capable of correcting $t = \frac{(n-k)}{2}$ error in the received code word. RS decoder introduced in this paper is developed for the SIMD reconfigurable processor. It is necessary to find a RS decoding algorithm, which makes the parallel processing possible. Let's consider $v(x)$ is the transmitted code word and $r(x)$ is the received code word. Then the error added by channel is

$$e(x) = r(x) - v(x) \qquad (1)$$

Considering υ errors are occurred in the transmitted code word the syndromes are computed as follows:

$$S_j = v(\alpha^j) = r(\alpha^j) + e(\alpha^j) = \sum_{k=0}^{n-1} e_k (\alpha^j)^k = \sum_{l=1}^{\upsilon} e_{i_l} X_l^j \quad (2)$$

Eq (3) shows the key equation to find the error locations and error magnitudes.

$$\Lambda(x) S(x) = \Gamma(x) \bmod x^{2t} \qquad (3)$$

$\Lambda(x)$ is the error location polynomial and its roots are the inverses of the error locations. It can be presented as follows:

$$\Lambda(x) = 1 + \sigma_1 x + \ldots + \sigma_\nu x^\upsilon = \prod_{l=1}^{\nu} (1 - x\alpha^{i_l}) \qquad (4)$$

After calculating the error location polynomial roots the error value corresponding to each error location can be easily calculated using the Eq (5)

$$e_l = -(\alpha^{i_l})^{-1} \frac{\Gamma((\alpha^{i_l})^{-1})}{\Lambda((\alpha^{i_l})^{-1})} \qquad (5)$$

The Berlekamp algorithm [14] is used to find the error location polynomial for the SIMD implementation of this decoder.

3.2. GF Multiplication and Addition Inside Each RC

GF multiplication is the basic operation should be considered in the implementation of RS decoder. There are two ways of implementing GF multiplication inside each RC. The first method is using look up tables for converting the vector representation to the power representation. Being able to change the power representation to its vector, we can consider GF elements in the whole decoding procedure as their vector representations. The GF multiplication will be simply the addition of the two powers. After addition, Mod 255 of the result should be computed. We avoid this time consuming computation by storing two look up tables continuously instead of one for converting the power to vector. the second look up table will be the vector corresponding for the power of 255 to 512. This method works if we just have the multiplication of two GF elements and will convert the result to its vector presentation soon after that. This is the case in implementing the GF Multiplication. Total application is mainly consists of GF polynomial computation which is the GF additions after each GF multiplication. In order to make each RC the computational cell for the GF addition and multiplication, we are taking advantage of the local memory inside each RC. These memories are specially implemented for storing look up tables, which are extremely, used in the data streamed applications. The second method comes from the feasibility of implementing the GF multiplier hardware inside each RC using the fine grain block. Using this way there is no need to know about the power presentation of GF elements and all the computation will go on using the vector presentation. Using the GF multiplier instead of the look up tables will speed up total application, as now we are able to multiply two GF elements and get the result in one clock cycle.

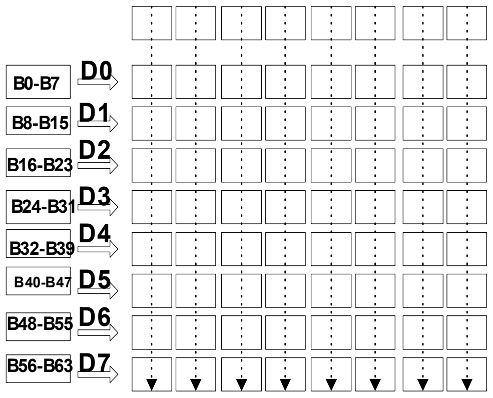


Figure 3. context broadcast to RC columns from the context memories of each column

3.3. Developing the Parallel Algorithm of the RS Decoder

In order to achieve the highest performance of the decoder implemented on the Morphosys, We should develop the introduced decoding algorithm for the SIMD processor. Fig 3 shows the parallelism exploited at the data level for the RS (255, 239,16). Each row of RC has been reconfigured for decoding one block of the data. 8 blocks of data will be decoded in parallel with each other. This will decrease the throughput to 1/8 of the total time needed for decoding 8 blocks in parallel. Fig 4 shows the task-flow diagram of the decoder. Each RC holds one coefficient of the error location polynomial, error magnitude polynomial and error correction polynomial. Each RC is the basic operational block for GF addition and multiplication. In the figure S_i Stands for the different syndromes computed during the syndrome computation. σ_i and T_i's are representing the error location and error correction polynomials coefficients respectively which are computed during the Berelekamp algorithm. In the Chein search part β_i stands for the GF (255) element, which should be substituted into the error location polynomial, to find the root of the error location polynomial. δ_i presents a coefficient of the error magnitude polynomial which is computed during the Forney's method stage and will be used to calculate the error values.

3.3.1. Syndrome Computation

The first step is the Syndrome calculation out of the received code word. Received code word is saved in the frame buffer and will be transmitted to each RC .Each RC computes two Syndromes in parallel with the other RCs of the row. This scheme is generalized depending on the number of Syndromes should be computed. This has been made easy since there is one bank of the Frame buffer corresponding to each RC. So we can easily scale the Syndrome numbers transmitting the data from frame buffer's bank to each RC.

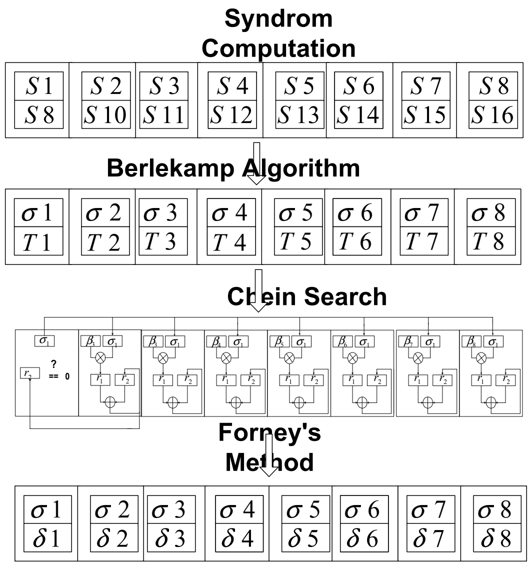


Figure 4. Task diagram of the decoding procedure in the time domain

3.3.2. Berlekamp Algorithm

Berlekamp Algorithm is consisting of three main parts. Discrepancy computation, calculation of the new error location polynomial and error correction polynomial. These calculations are basically made of addition of two polynomials together or multiplying one of them by the GF element. Addition has been made possible by storing the coefficients of the same order in the same RC. Addition of the polynomials will be adding of these two coefficients together in parallel in all RCs. In order to compute the discrepancy we need to take advantage of the data movement, between RCs. This data movement is possible in the same clock cycle as the multiplication happens using that element from the other RC. This is depicted in Fig 5.Taking advantage of the data movement between RCs is like increasing the register file of each RC to the ones from the RCs of the same row and same column. This is the strong tool to enhance the level of parallelism of the algorithm. Each RC can use the results of the other computation, as it has been stored in its register file.

3.3.3. Chein Search

Chein search can be done in parallel in the 7 RCs of the row. The first RC will check the result for the possible error detections. Each RC is responsible for trying 33 $GF(2^8)$ elements. The whole search has been done in parallel in the 7 RCs. FB will be used as a secondary memory here to store the elements of the $GF(2^8)$. Each bank will keep the necessary elements and their power, since we need up to 8 power of each element to reduce the computation time. In fact memory hierarchy is one of the important feature of the Morphosys. We can take advantage of the different parameter storage in the different levels of the memory.

3.3.4. Error Value computation

Error value calculation will be performed sequentially for the different error locations. This way we take advantage of parallel GF addition and multiplication in different RCs. Fig 6 shows different terms calculation of the error values inside each RC. Δ_k^{-1} is the discrepancy calculated during the Berlekamp algorithm part and σ_i s are the error location coefficients.

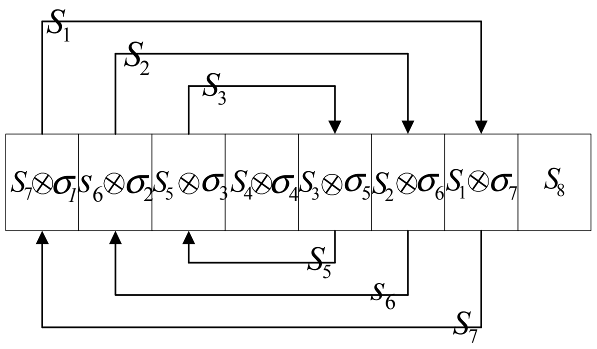


Figure 5. Data movement between RCs in order to calculate different terms of the Discrepancy in the different RC

4 CONCLUSION AND COMPARISON

MorphoSys as a reconfigurable architecture provides a very flexible platform for implementing different DSP applications. This is perfectly demonstrated through the RS decoder implementation with different error correcting length. Being able to reconfigure each RC for the different parts of the wireless communication receiver will provide the best functionality of each RC for the whole application. TI DSPs have the same approach to perform GF multiplication. In comparison Morphosys takes advantage of the higher data parallelism obtained through the SIMD characteristics of it. DSP C6400 provides the hardware support for performing the Galois Field multiplies. In the absence of hardware to effectively perform Galois field math, previous DSP implementations made use of logarithms to perform multiplication. This has decreased performance to ¼ of the present one with hardware GF multiplication. Many different Reed-Solomon ASIC Architectures are proposed in the literature [12-13]. These blocks are mostly specified and optimized for the RS decoder. They can be considered as a single chip RS decoder. In comparison Morphosys can be viewed as the programmable architecture optimized for the whole receiver Morphosys outperforms Asics, which have the smaller size. We have simulated the result on the Morphosys hardware simulator called Mulate. The result is given for the RS decoder of (255,239,16) using two different methods of GF multiplication. The decoding speed is 1.319Gbps and 2.534Gbps, respectively. Fig 7 shows the bit rate of the decoder and its comparison with the different Asics and Also TI64x

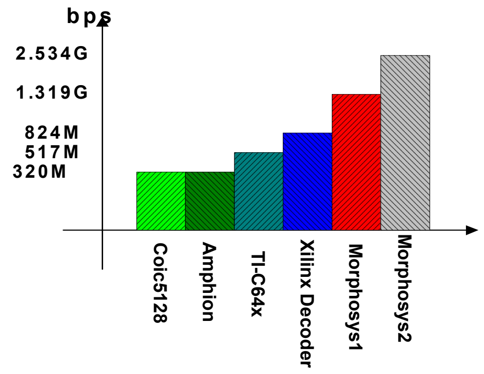


Figure7. Bit rate comparison of two decoders implemented on Morphosys with the other Asics and TIC64x

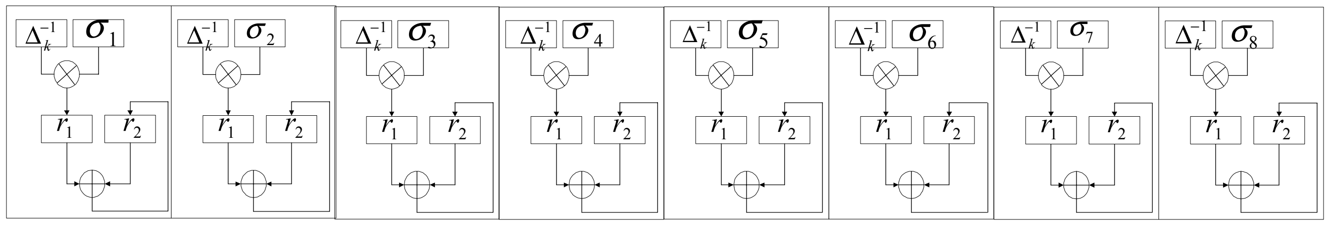


Figure 6. Different terms calculation of the error values

References:

[1] H. Singh, Lee, Lu, Bagherzadeh, Kurdahi, "MorphoSys: An Integrated Reconfigurable System for Data-Parallel and Computation –Intensive Applications," IEEE Transactions on Computers, vol. 49, No. 5, pp. 465-481, May 2000.

[2] Lee, Singh, Lu, Bagherzadeh, Kurdahi, "Design and Implementation of the MorphoSys Reconfigurable Computing Processor," Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology, vol. 24, No. 2-3, Kluwer Academic Publishers, pp. 147-164, Mar 2000.

[3] Pan, Kamalizad, Koohi, Bagherzadeh, "Design and Analysis of a Programmable Single-Chip Architecture for DVB-T Base-Band Receiver," To Appear in DATE 2003.

[4] Agarwal, Hrishikesh, Keckler, Burger, **"**Clock rate versus IPC: the end of the road for conventional microarchitectures," Computer Architecture, 2000. Proceedings of the 27th International Symposium on, 2000 Page(s): 248 -259

[5] Schlansker, Rau, "EPIC: Explicitly Parallel Instruction Computing," Computer, Volume: 33 Issue: 2, Feb 2000 Page(s): 37 -45

[6] Taylor, et.al. **"**The Raw microprocessor: a computational fabric for software circuits and general-purpose programs," Micro, IEEE , Volume: 22 Issue: 2 , Mar/Apr 2002, Page(s): 25 -35

[7] Rixner, et.al. **"**A bandwidth-efficient architecture for media processing," Microarchitecture, 1998. MICRO-31. Proceedings. 31st Annual ACM/IEEE International Symposium on , 30 Nov-2 Dec 1998

[8] Kozyrakis, Patterson, "Vector vs. superscalar and VLIW architectures for embedded multimedia benchmarks," Microarchitecture, 2002. (MICRO-35). Proceedings. 35th Annual IEEE/ACM International Symposium on, 2002 Page(s): 283 -293

[9] Taylor, Lee, Amarasinghe, Agarwal, "Scalar Operand Networks: On-chip Interconnect for ILP in Partitioned Architectures," HPCA 2003, February 2003.

[10] Venkataramani, Najjar, Kurdahi, Bagherzadeh, Bohm, "A Compiler Framework for Mapping Applications to a Coarse-grained Reconfigurable Computer Architecture," CASES 2001, Atlanta, GA, November 2001.

[11] Maestre, Kurdahi, Bagherzadeh, Singh, Hermida, Fernandez, "Kernel scheduling in reconfigurable computing," DATE 1999.

[12] Implementation of high speed Reed-Solomon decoder. [Conference Paper] 42nd Midwest Symposium on Circuits and Systems (Cat. No.99CH36356). IEEE. Part vol. 2, 2000, pp.808-12 vol. 2. Piscataway, NJ, USA.

[13] Martina M, Masera G, Piccinini G, Vacca F, Zamboni M. VLSI Reed Solomon decoder architecture for networked multimedia applications. [Conference Paper] *Proceedings 14th Annual IEEE International ASIC/SOC Conference (IEEE Cat. No.01TH8558). IEEE. 2001, pp.347-51. Piscataway, NJ,USASystems II-Analog & Digital Signal Processing, vol.47, no.11, Nov. 2000, pp.1254-70. Publisher: IEEE, USA.*

[14] Shu LIN/ Daniel J Costello Error Control Coding Fundamentals and applications

[15] www.amphion.com

[16] www.ti.com

[17] www.xilinx.com