VLSI ARCHITECTURE FOR DATAPATH INTEGRATION OF ARITHMETIC OVER $GF(2^m)$ ON DIGITAL SIGNAL PROCESSORS

Wolfram Drescher, Kay Bachmann, and Gerhard Fettweis

Mobile Communications Systems
Dresden University of Technology, D-01062 Dresden
drescher,bachmann,fettweis@ifn.et.tu-dresden.de

ABSTRACT

This paper examines the implementation of Finite Field arithmetic, i.e. multiplication, division, and exponentiation, for any standard basis $GF(2^m)$ with $m \leq 8$ on a DSP datapath. We introduce an opportunity to exploit cells and the interconnection structure of a typical binary multiplier unit for the Finite Field operations by adding just a small overhead of logic. We develop division and exponentiation based on multiplication on the algorithm level and present a simple scheme for implementation of all operations on a processor datapath.

1. INTRODUCTION

Arithmetic over Finite Fields is usually applied in coding theory, cryptography and their applications [1]. Even though there would exist many applications in mobile and satellite communication, codes based on Finite Fields are not commonly used yet. For example, BCH codes have very desirable properties regarding burst error correction, but are currently not used in mobile communication network systems. At present, hardware providers offer a variety of chipsets for specific applications having heavy constraints for arithmetic over Finite Fields and therefore code properties as well. For applications based on flexible field width and variable primitive polynomials, to our knowledge, no hardware solution is available. If programmable logic components would be utilized for Finite Field arithmetic, the programmer could freely choose between different codes for different applications. That means, any theoretical designed code could be easily customized to hardware implementations using a programmable processor. No longer would special and expensive circuits need to be designed for a particular system implementation. The system implementation and testing could be executed straightforwardly in a software environment. Once DSPs support Finite Field arithmetic, we anticipate a breakthrough even for low budget coding applications.

2. DATAPATH DESIGN REQUIREMENTS FOR DOMAIN SPECIFIC DSPs

A processor datapath is where the vital arithmetic manipulations of operands take place. To achieve a higher performance, a DSP datapath is specialized on the types of computations mostly performed in signal processing. Present DSPs support these computations by functional units as multiplier-accumulators, bit manipulation units or barrel shifters. As demand grows for Domain Specific Digital Signal Processors (DSDSP) for special applications, e.g. in mobile communications systems, manufacturers have begun to tailor their processors to the algorithms of the application for which the DSP was requested. For the datapath this means more functionality and parallelism needs to be added [2].
Our intention was to design a datapath for a DSDSP for the domain of error correction coding[1]. From the arithmetical point of view, this requires addition, multiplication, division, and exponentiation over Galois Fields. However, in addition conventional binary fixed point multiply and accumulate operation as a basic feature of a DSP need to be supported as well.

3. ALGORITHMIC APPROACH FOR ARITHMETIC OVER FINITE FIELDS $GF(2^m)$

The first non sequential architecture for Finite Field Multiplication was described in [3]. In [4] an attempt has been made to integrate a Finite Field multiplier in a economical way on a DSDSP datapath. In this paper, we go one step further and present an architecture for integration of all important arithmetical operations over Finite Fields on a DSDSP datapath.
Many papers have focused on performing arithmetic over Finite Fields in a sequential way [5], [6], or by using systolic structures [7], [8]. Mostly, these approaches exploit properties of normal basis and cannot be applied for standard basis. Hence, they cannot be used for programmable architectures. Our proposed architecture allows a general usage for any standard basis Finite Field.
Finite fields, so called Galois fields, consist of p^m elements where p is a prime and $m \in N - \{0\}$. $GF(p^m)$ is an extension field of $GF(p)$. Codes with symbols from the binary field

1. This work was sponsored in part by the Deutsche Forschungsgemeinschaft within the Sonderforschungsbereich SFB 358

GF(2) of 2 elements ($GF(2) = \{0,1\}$) or its extension $GF(2^m)$ are most widely used in digital systems.
α is a root of a primitive irreducible polynomial $f(x) = x^m + f_{m-1}x^{m-1} + \ldots + f_1x + f_0$ in $GF(2^m)$. All nonzero elements of $GF(2^m)$ can be represented as powers of a primitive element α. Since $f(\alpha) = 0$, $\alpha = f_{m-1}\alpha^{m-1} + \ldots + f_1\alpha + f_0$, i.e. an element of $GF(2^m)$ can be expressed as a polynomial of α with degree less than m. The polynomial representation is used to represent the finite field $GF(2^m)$. Since the elements of a Finite Field can be represented by vectors with its coefficients from *GF(2)* {0,1}, the addition in $GF(2^m)$ is equivalent to the subtraction. Hence both operations can be realized by the same hardware. In contrast to addition, the computation of multiplication and division over Finite Fields is more difficult. Following, basic algorithms for multiplication, division, and exponentiation over $GF(2^m)$ will be discussed.

3.1 Multiplication

If $A = a_0 + a_1\alpha + \ldots + a_1\alpha^{m-1}$ and $B = b_0 + b_1\alpha + \ldots + b_{m-1}\alpha^{m-1}$ are two elements from $GF(2^m)$, then

$$A + B = C = c_0 + c_1\alpha \ldots + c_{m-1}\alpha^{m-1}, \qquad (1)$$

with $c_i = a_i + b_i \bmod 2 \qquad (0 \le i \le m-1).$ (2)

Hence, addition over $GF(2^m)$ can be realized easily by m independent XOR gates. Multiplication over $GF(2^m)$ is defined through

$$a(x) \otimes b(x) := a(x) \cdot b(x) \bmod f(x). \qquad (3)$$

It can be calculated by building partial products

$$P_i(x) = a(x) \cdot b_i x^i \qquad (0 \le i < m) \qquad (4)$$

first. Second, all partial products $P_i(x)$ have to be added modulo 2. A partial result $C_p(x)$ can be obtained with

$$C_p(x) = \sum P_i(x) \bmod 2 \qquad (5)$$

$$C_p(x) = c_0 + c_1x + \ldots + c_{2m-2}x^{2m-2} \qquad (6)$$

which is obviously not an element of the field $GF(2^m)$. To reduce the highest order term x^h of $C_p(x)$ the property of the irreducible polynomial

$$x^m = f_0 + f_1x + \ldots + f_{m-1}x^{m-1} \qquad (7)$$

is exploited. The substitution starts with the highest order term $c_{2m-2}\,x^{2m-2}$ from (6). (7) is used to produce a partial result C'_p of one degree less than C_p:

$$C'_p(x) = [c_0 + c_1x + \ldots + c_{2m-2}x^{2m-3}] + [f_0 \cdot x^{m-2} + f_1x \cdot x^{m-2} + \ldots + f_{m-1}x^{m-1} \cdot x^{m-2}]. \qquad (8)$$

This operation is altered until $C'_p(x)$ is of degree m-1, i.e. it becomes an element of $GF(2^m)$. Details are given in [12].

3.2 Division

Division is not directly defined in a straight forward manner in a Finite Field. However, a possibility to perform division over $GF(2^m)$ is by multiplication of the inverse element of a divisor β with the dividend γ:

$$\frac{\beta}{\gamma} = \beta \cdot \gamma^{-1} \qquad (9)$$

It turn out that the real problem of division over $GF(2^m)$ is to find an efficient computation algorithm for the multiplicative inverse element of the divisor. Two algorithms are known:

- Euclid's algorithm [12],
- Continuous square algorithm.

In this paper, the continuous square algorithm is used which is based on a fundamental property of $GF(2^m)$:

$$\gamma^{2^m - 1} = 1 . \qquad (10)$$

We multiply (10) by γ^{-1} and get (11):

$$\gamma^{-1} = \gamma^{2^m - 1} \cdot \gamma^{-1} = \gamma^{2^m - 2} \qquad (11)$$

Since

$$2^m - 2 = \sum_{i=1}^{m-1} 2^i \qquad (12)$$

γ^{-1} can be computed successively by squaring and multiplication. To give an example, we use $GF(2^4)$. Since $14 = 2 + 4 + 8$,

$$\gamma^{14} = \gamma^2 \cdot \gamma^4 \cdot \gamma^8 = \gamma^{-1}. \qquad (13)$$

3.3 Exponentiation

Exponentiation over Finite Fields is useful for decoding Reed-Solomon-Codes, e.g. for Forney's algorithm to compute error values. A simple algorithm for exponentiation over $GF(2^m)$ based on multiplication was proposed in [11]:

$$y = \beta^e \text{ with } \beta \in GF(2^m) \qquad e \in N \qquad (14)$$

Each integer can be presented in its binary representation as a n-bit-vector according to (14).

$$e = e_0 + e_1 2 + e_2 2^2 + \ldots + e_{n-1}2^{n-1} \qquad (15)$$

The substitution of (15) into (14) leads to:

$$y = \beta^{e_0 + e_1 2 + e_2 2^2 + \ldots + e_{n-1}2^{n-1}}$$

$$y = \beta^{e_0} + \beta^{e_1 2} + \beta^{e_2 2^2} + \ldots + \beta^{e_{n-1}2^{n-1}}$$

$$y = \beta^{e_0}(\beta^{e_1}(\beta^{e_2}(\ldots(\beta^{e_{n-1}})^2)^2)^2)^2 \qquad (16)$$

and

$$\beta^{e_i} = \begin{cases} \beta & \text{if } e_i = 1 \\ 1 & \text{if } e_i = 0 \end{cases} \qquad (17)$$

From (15) and (16) an exponentiation algorithm can be derived:

```
y = 1
for i = n - 1 to 0
    { y = y^2
      if e_i = 1 then y = β · y
    }
β^e = y
```

4. IMPLEMENTATIONAL APPROACH

4.1 Multiplication

In [4] opportunities have been investigated to merge a conventional binary multiplier and a multiplier over $GF(2^m)$ at the gate-level. This leads to a new multiplier architecture that performs both types of multiplications employing the same logical cells with a low complexity and a marginal propagation overhead. This architecture will serve as the basis for our error correction coding DSDSP datapath introduced in this paper.

The switching function of a full-adder with 3 inputs X_1, X_2, X_C and two independent outputs Y_S and Y_C is:

$$Y_S = X_1 \otimes X_2 \otimes X_C\,, \qquad (18)$$

$$Y_C = X_1X_2 \vee X_1X_C \vee X_2X_C\,. \qquad (19)$$

Typically Y_S feeds X_1, X_2 and Y_C feeds X_C of the next adder stage. Comparing (18) and (2) we see that both logical functions involve XOR gates to calculate their result. In fact, if X_C in (18) is forced to logical 0 and Y_C of the feeding full-adder stage is left open, we can use the full-adder to perform the XOR operation needed in (2). Beyond the logical function of a basic cell, also the cell interconnection scheme of a binary partial product reduction array can be maintained, because as we can conclude from (2), (5), (18) and (19) it is equal for binary as well as for finite field addition arrays.

For physical implementation a binary partial product reduction (PPR) array is usually divided into subarrays to yield a better regularity [9]. For merging the two different modulo-addition arrays representing (6) and (8) with a binary PPR array, we propose to split the binary PPR array in at least two equally sized parts with an adder column of length m-2. Each part houses logic for GF(2^m) addition ((6), (8)) and binary addition concurrently. Fig. 1 shows a simplified

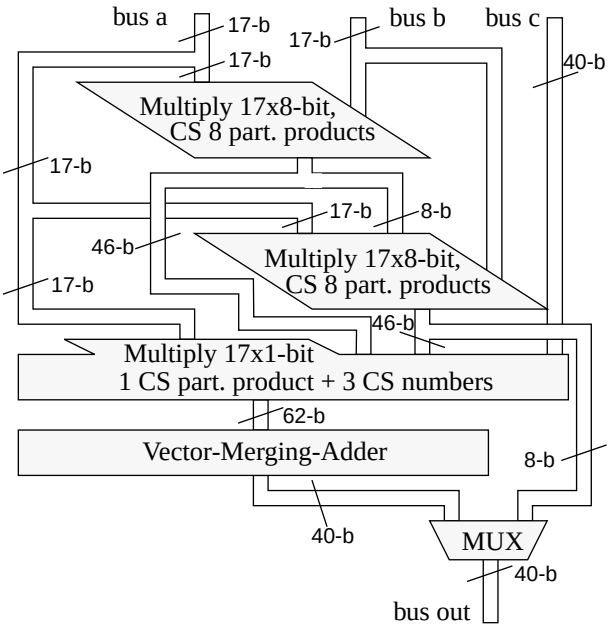

Fig. 2 Block diagram of a 17-b binary / $GF(2^8)$ multiplier

schematic of a signed17x8-bit multiplication and PPR array based on a Wallace-tree [10] for (signed) integers, that shares the cells with an 8-bit Finite Field multiplier as discussed in [4] to perform (8). This array is part of a 17x17-bit combined binary and $GF(2^8)$ multiplier depicted in Fig. 2 and consists of 6 different cells. Fig. 2 shows how the complete multiplier is divided into two equal 17x8-bit PPR arrays in carry-save (CS) format and one CS PPR array which calculates the sum of the highest order partial product and the two 17x8-bit PPR arrays. A multiplexer (MUX) after the final vector merging adder can select between the result of the binary and GF(2^m) multiplication.

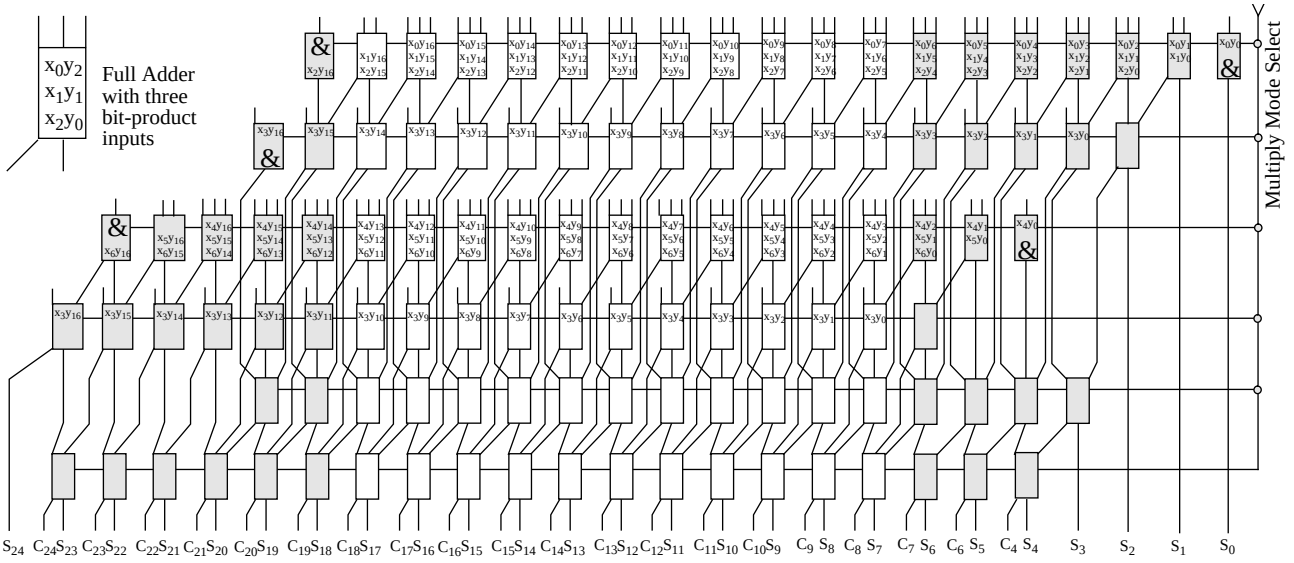

Fig. 1 Design example of a 17x8-bit combined multiplier PPR array

In Fig. 1 Full-adders employed for binary and GF(2^m) multiplication concurrently are depicted darker. To perform (6), the array Fig. 1 can easily be used in the same manner.

4.2 *GF*(2^m) Division and *GF*(2^m) Exponentiation

As we have shown in chapter 3, division and exponentiation over Finite Fields can be calculated by continuous square operations. Our studies on implementing Euclid´s algorithm [7] did not lead to a simple and feasible design result. However, a quotient or an exponentiation cannot be obtained in one clock cycle with the proposed architecture, but it still accelerates calculations by orders of magnitudes. Beyond that, no extra hardware on the datapath needs to be added and programming becomes more transparent.

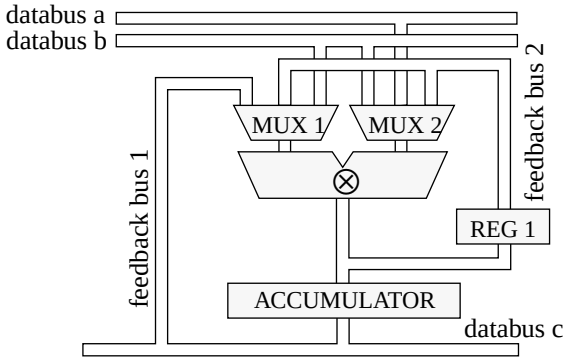

Fig. 3 Block diagram of the GF(2^m) arithmetical unit

Fig. 3 shows a block diagram of a simple architecture for Finite Field datapath. If REG1 is replaced by a second accumulator, this architecture can be partly found in present DSPs. For the computation of multiplications databus a and b feed the two factors. The result is stored in the accumulator. For squaring databus b is used only. Division is more complex and uses the accumulator and REG1. During the first division step the divisor (databus b) is squared and the result is stored in both registers. After that, the content of REG1 is squared again and the multiplication of this square to the content of the accumulator is performed alternately. After the $2m\text{-}1^{th}$ step the inverse of the divisor is located in the accumulator and is multiplied to the dividend (databus a). For exponentiation first REG1 has to be preset to the value of α^0 (01H). Then, squaring of the content of REG1 and the multiplication to β (databus a or b), dependent on the currently active bit of the exponent, is performed alternately. During the last step of this operation the result is stored in the accumulator.

5. BENCHMARKS

Benchmarks for a Reed-Solomon (255,223) decoder implemented on a TMS320C25-50 DSP compared to an implementation of our proposed datapath architecture have shown, that an implementation with our proposed architecture is more than 10 times faster if the Finite Field operations are carried out on the DSP in an algorithmic way. If precalculated lookup tables would be used, our approach would still be about 2 times faster. For longer codes the benchmark results become even better for our datapath.

6. CONCLUSIONS

We have developed a VLSI architecture for all basic arithmetical operations over Finite Fields for implementation on a DSDSP datapath. Multiplication, inversion, exponentiation, and addition over *GF*(2^m) can be computed. We proposed to implement exponentiation and addition based on multiplication. A simple implementation of an arithmetical multiplication unit has been introduced, which can process binary numbers and elements of a Finite Field as well.

7. REFERENCES

[1] S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, Englewood Cliffs, NJ: Prentice-Hall, 1983.

[2] G. Fettweis, „DSPs for Mobile Communications: Where are we going?," Proc. of ICASSP 1997.

[3] B. A. Laws Jr. and C. K. Rushforth, „A Cellular-Array Multiplier For *GF*(2^m)," *IEEE Transactions on Computers*, pp. 1573-1578, Dec. 1971.

[4] W. Drescher and G. Fettweis, „VLSI Architectures for Multiplication in *GF*(2^m) for Application Tailored Digital Signal Processors," Proc. of 1996 IEEE Workshop on VLSI Signal Processing.

[5] E. R. Berlekamp, „Bit-Serial Reed-Solomon Encoders," *IEEE Transactions on Information Theory*, vol. IT-28, no. 6, pp. 869-874, Nov. 1982.

[6] C. C. Wang, T. K. Troung, H. M. Shao, L. J. Deutsch, and I. S. Reed, „VLSI Architectures For Computing Multiplications and Inverses in *GF*(2^m)," *IEEE Transactions on Comp.*, vol. c-34, pp. 709-717, Aug. 1985.

[7] Y.-J. Jeong and W. Burleson, „VLSI Array Synthesis for Polynomial GCD Computation and Application to Finite Field Division," *IEEE Transactions on Circuits and Systems*, pp. 891-897, Dec. 1994.

[8] M.A. Hasan, V.K. Bhargava, „Bit-Serial Systolic Divider and Multiplier for Finite Fields *GF*(2^m)," *IEEE Transactions on Computers*, vol. 41, pp. 972-980, Aug. 1993.

[9] G.J. Hekstra, R. Nouta, „A Fast Parallel Multiplier Architecture," Proc. of 1992 IEEE International Symposium on Circuits and Systems.

[10] C.S. Wallace, „A Suggestion for A Fast Multiplier," *IEEE Transactions on Elec. Computers*, vol. EC-13, pp. 14-17, Jan. 1964.

[11] P.A. Scott, S.J. Simmons, S.E. Tavares, L.E. Peppard, „Architectures for Exponentiation in *GF*(2^m)," *IEEE Journal on Selected Areas in Communications*, pp. 578-586, April . 1988.

[12] A. G. Akritas, *Elements of Computer Algebra*, New York, NJ: Wiley, 1989.