Low-Power Design of a Functional Unit for Arithmetic in Finite Fields GF(p) and GF(2^m)

Johann Großschädl and Guy-Armand Kamendje

Graz University of Technology
Institute for Applied Information Processing and Communications
Inffeldgasse 16a, A–8010 Graz, Austria
{Johann.Groszschaedl,Guy-Armand.Kamendje}@iaik.at

Abstract. Recent multi-application smart cards are equipped with powerful 32-bit RISC cores clocked at 33 MHz or even more. They are able to support a variety of public-key cryptosystems, including elliptic curve systems over prime fields GF(p) and binary fields GF(2^m) of arbitrary order. This flexibility is achieved by implementing the cryptographic primitives in software and taking advantage of dedicated instruction set extensions along with special functional units for low-level arithmetic operations. In this paper, we present the design of a low-power multiply/accumulate (MAC) unit for efficient arithmetic in finite fields. The MAC unit combines integer arithmetic and polynomial arithmetic into a single functional unit which can be configured at run-time to serve both types of fields, GF(p) and GF(2^m). Our experimental results show that a properly designed unified (dual-field) multiplier consumes significantly less power in polynomial mode than in integer mode.

Keywords: Multi-application smart cards, elliptic curve cryptography, finite fields, unified multiplier datapath, instruction set extensions.

1 Introduction

In general, embedded systems incorporate a mixture of general-purpose hardware (e.g. processor core, memory, etc.), application-specific hardware[1] (e.g. special functional unit or co-processor), and software. An example for such embedded systems are smart cards, whereby the application-specific hardware may include a cryptographic co-processor or a random number generator. The majority of the smart cards on the market today are equipped with 8-bit micro-controller cores such as Intel's 8051, Motorola's 6805, or Hitachi's H8. More sophisticated smart cards contain a cryptographic co-processor for enhanced arithmetic computation capabilities [12].

The smart card market is currently shifting from the traditional 8-bit smart cards to 16 and 32-bit cards equipped with powerful RISC processors. "Second generation" operating systems like *MULTOS* or *Windows for Smart Cards* sit

[1] Application-specific hardware shall be understood as hardware tailored to a certain application or application domain like multimedia processing or cryptography.

K. Chae and M. Yung (Eds.): WISA 2003, LNCS 2908, pp. 227–242, 2003.

on top of the card's proprietary system and can host several applications separated by secure firewalls. The RISC cores of these multi-application smart cards support advanced cryptography processing by means of dedicated instruction set extensions, following the idea of multimedia extensions found in today's high-end processors (e.g. Intel's MMX). Well-known microprocessor vendors such as ARM Limited or MIPS Technologies developed optimized 32-bit RISC cores for smart cards with advanced security features and custom instructions for accelerating both symmetric and asymmetric cryptosystems [1,18].

Integrating security technology into the processor core is necessary because cryptographic co-processors are inadequate for next-generation multi-application cards due to limitations in scalability and algorithm agility. For instance, elliptic curve cryptosystems [3] offer a variety of implementations options, which calls for more flexible solutions than fixed-function hardware blocks allow. Augmenting a general-purpose processor with special instructions for performance-critical operations facilitates fast yet flexible implementations of elliptic curve cryptography. Typical candidates for custom instructions are the inner loop operations of low-level arithmetic primitives (e.g. multiplication in finite fields).

The circuitry that actually executes a given kind of instruction is called a *functional unit* (FU). There are two fundamental differences between an FU and a co-processor. Firstly, an FU is tightly coupled to the processor core and directly controlled by the instruction stream, which eliminates the hardware cost of a co-processor interface and dedicated control logic. Secondly, FUs have access to the register file and perform their operations on operands stored in general-purpose registers, whereas a co-processor typically incorporates extra registers for holding the operands involved in the computations.

In this paper, we present a special functional unit (FU) for multiplications and multiply-and-add operations on signed/unsigned integers and binary polynomials. The FU is realized in terms of a multiply/accumulate (MAC) unit and uses the same datapath for both types of operands, i.e. it consists of a so-called unified multiplier datapath. A custom instruction for the word-level multiplication of binary polynomials is very useful for an efficient software implementation of elliptic curve cryptosystems over binary extension fields $\mathrm{GF}(2^m)$ [19]. The basic building block of the MAC unit is a (32×32)-bit array multiplier made up of radix-4 partial product generators and unified (4:2) compressors. Our design is optimized for low power consumption and exploits the fact that a properly designed dual-field multiplier consumes significantly less power in polynomial mode than in integer mode. A general-purpose RISC processor equipped with this MAC unit allows fast yet flexible implementations of elliptic curve cryptography over $\mathrm{GF}(2^m)$, thereby eliminating the need for a co-processor.

1.1 Related Work

The first approach for combining the hardware of a conventional multiplier with a multiplier for finite fields $\mathrm{GF}(2^m)$ was introduced by Drescher *et al.* [6]. They investigated the implementation of arithmetic in $\mathrm{GF}(2^m)$, $m \leq 8$, on a DSP datapath for signal coding, in particular BCH codes. Later, polynomial arithmetic

was also integrated into the datapath of modular multipliers for cryptographic applications [4]. Savaş *et al.* presented a scalable and unified architecture for Montgomery multiplication in finite fields GF(p) and GF(2^m) [26]. Their design is based on the observation that polynomial addition can be realized by using an integer addition circuit and simply disabling the carry path. The basic building block of their processing unit (PU) is a so-called dual-field adder (DFA), which is nothing else than a full adder with a small overhead of logic for setting the carry output to zero.

Goodman and Chandrakasan described the implementation of a domain-specific reconfigurable cryptography processor (DSRCP) capable of performing a number of different public-key cryptosystems [8]. The major component of the DSRCP is a reconfigurable 1024-bit datapath made up of simple computation units (CUs), each consisting of two full adders and two NAND gates. These CUs can be reconfigured on the fly to carry out radix-2 Montgomery multiplication of integers modulo a prime p, MSB-first bit-serial multiplication in GF(2^m), as well as inversion in GF(2^m) according to the binary extended Euclidean algorithm. During multiplication in GF(2^m), the CUs use one full adder to compute a 3-input GF(2) addition, whereas the second full adder is disabled to avoid unnecessary switching activities.

A completely different approach for combining integer and polynomial addition was introduced by Au and Burgess [2]. They implemented the addition of integers by using a *redundant binary representation* based on the digit set $\{0, 1, 2\}$, whereby the digits have the following encoding: $0 \sim (0, 0)$, $1 \sim (0, 1)$, and $2 \sim (1, 0)$. A fourth digit, denoted 1* and encoded as (1,1), represents the 1 in GF(2), which means that the addition of binary polynomials is performed with the digit set $\{0, 1^*\}$. Taking advantage of this special encoding, Au and Burgess proposed a unified (4:2) adder for integers and binary polynomials. The (4:2) adder has a critical path of only three XOR gates and does not need extra control logic to suppress the carries for polynomial addition.

Satoh and Takano presented a scalable elliptic curve cryptographic processor for finite fields GF(p) and GF(2^m), which is able to handle arbitrary field orders without changing the hardware configuration [25]. The core of the processor is a fully parallel $(r \times r)$-bit multiplier based on a Wallace tree architecture, whereby the word-size r is either 8, 16, 32, or 64 bits, depending on the desired trade-off between area and performance. Satoh and Takano's tree network is made up of standard full and half adders, respectively. The adders of the first stage (i.e. the leaf cells) take in three partial products and produce a sum and carry vector. In the subsequent stages, the sum and carry vectors are processed in separate datapaths, i.e. separate sub-trees of the multiplier. That way, the delay for summing up r binary polynomials is only $\lceil \log_3 r \rceil$ full adder delays since any full adder performs a 3-input GF(2) addition. On the other hand, an integer multiplication has a delay of about $\lceil \log_{3/2} r \rceil$ adder stages, coming from the fact that not only sum but also carry vectors need to be processed. Satoh and Takano's design is highly scalable and can be used for a variety of applications, ranging from embedded micro-controllers to high-speed security servers.

1.2 Our Contributions

Arithmetic in GF(2^m) has several advantages over conventional integer arithmetic used for prime fields. For instance, Satoh and Takano demonstrated that polynomial multiplication is much faster (i.e. allows higher clock frequencies) than the multiplication of integers, simply because of the "carry-free" addition of binary polynomials. Integer multipliers typically use a redundant representation (e.g. carry save form) to avoid time-consuming carry propagation, which causes extra delay in a tree multiplier since both sum and carry vectors must be processed.

In the present paper, we show that polynomial arithmetic has another major advantage over integer arithmetic, namely *less power consumption* and therefore better energy efficiency. The proliferation of portable battery-driven devices makes a good case to design low-power functional units, especially multipliers since they can make a notable contribution to the total power consumption of a RISC processor. We demonstrate that a properly designed dual-field multiplier consumes about 30% less power in polynomial mode than in integer mode.

Furthermore, we present a unified radix-4 partial product generator (PPG) for signed and unsigned integer multiplication as well as for multiplication of binary polynomials. The PPG is based on our previous work [9], but extended to support also the processing of signed operands. In integer-mode, the generation of partial products is performed according to the modified Booth recoding technique [15]. In polynomial mode, on the other hand, the PPG works like a digit-serial polynomial multiplier with a digit-size of $d = 2$ (see [16]). Note that all designs mentioned in the previous subsection are based on radix 2.

2 Architecture of the MAC Unit

Various RISC processors tailored to the embedded systems field contain a fast multiply/accumulate (MAC) unit to facilitate common DSP routines like computation of matrix multiplies, vector dot products, or fast Fourier transforms (FFT). For instance, the MIPS32 4Km features a MAC unit consisting of a 32×16 Booth recoded multiplier, result/accumulation registers (referenced by the names `HI` and `LO`), and the necessary control logic [17]. The multiply-and-add (`MADD`) instruction multiplies two 32-bit words and adds the product to the 64-bit concatenated values in the `HI/LO` register pair. Then, the resulting value is written back to the `HI` and `LO` registers. In other words, the `MADD` instruction can carry out computations of the form $A \times B + Z$, whereby the operand Z is a double-precision (i.e. 64-bit) quantity. Various signal processing kernels can make heavy use of that instruction, which optimizing compilers automatically generate when appropriate.

In the following sections, we present the design and implementation of a single-cycle $(32 \times 32 + 64)$-bit MAC unit that can execute multiply (`MULT`) as well as multiply-and-add (`MADD`) instructions on three different kinds of operands: Unsigned integers, signed integers in two's complement representation, and binary polynomials. The core of the proposed MAC unit is a unified (32×32)-bit

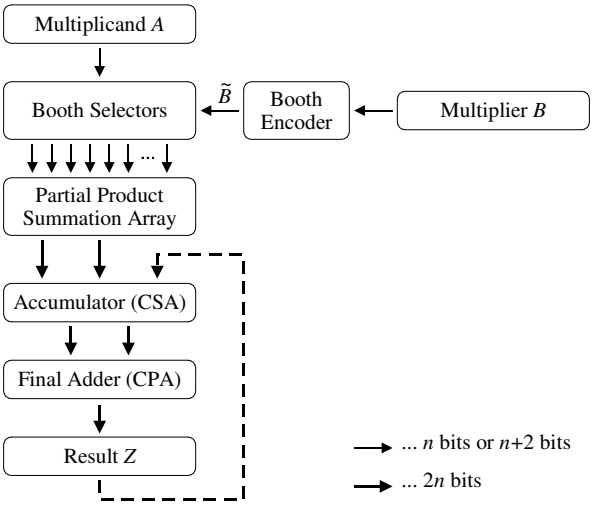

Fig. 1. Main components of the multiply/accumulate unit

multiplier made up of so-called dual-field adders (DFAs). Figure 1 illustrates the architecture of the MAC unit with its main components. Modified Booth recoding [15] is widely employed in parallel multipliers because it reduces the number of partial products by half. We present a unified Booth selector (partial product generator) for signed/unsigned integers and binary polynomials in Section 3. The partial products are summed up by using a redundant representation (e.g. carry-save form) to avoid the time-consuming carry propagation.

A multitude of efficient techniques for partial product summation have been proposed in literature, see e.g. [22] and the references therein. In the simplest case, the addition of partial products can be accomplished with (3:2) counters (carry-save adders) arranged in an array or tree structure. Higher-order counters or compressors generally help to reduce the cycle time. Array multipliers are simple to implement and result in a very regular layout, whereas tree multipliers allow to reach much higher clock frequencies, but at the expense of irregular wiring. We decided to implement the multiplier by means of an array architecture since smart cards, in general, are clocked at moderate frequencies (typically less than 33 MHz).

An Accumulator, which is nothing else than a carry-save adder (CSA), allows the addition of the product to a 64-bit cumulative sum when a `MADD` instruction is executed[2]. The outputs of the Accumulator represent the result in carry-save form. Now we have convert the sum and carry vector to a standard binary number to obtain the final result. This final addition is accomplished with a carry-propagation adder (CPA). The key point is that the carry-propagation delay occurs only once, at the very end, rather than in each addition step.

[2] MIPS32 processors store the 64-bit cumulative sum in the `HI/LO` register pair.

The availability of `MULT` and `MADD` instructions for binary polynomials allows very efficient implementation of arithmetic operations in binary extension fields GF(2^m). Many standard algorithms for multiple-precision arithmetic of integers — such as Montgomery multiplication or Barrett reduction — can be applied to binary polynomials as well [14,5]. We refer to [10] for a detailed description of word-level algorithms for arithmetic in GF(2^m).

3 Radix-4 Partial Product Generation

The multiplication of two numbers A and B is realized by generation and summation of partial products. Consequently, there are two major options to speed up multiplication: Reduce the number of partial products or accelerate their summation. For the former we need a higher-radix multiplication scheme where two or more bits of the multiplier B are examined at a time. Well-known examples include Booth's algorithm and modified Booth's algorithm (also called Booth-MacSorley algorithm [15]). Figure 1 indicates that modified Booth recoding is performed within two steps: Encoding and Selection (partial product generation).

A unified partial-product generator (PPG) that works for both unsigned integers and binary polynomials was presented in [9]. In this section, we extend the unified PPG to support also signed integers. This is necessary because most modern RISC processors, including the MIPS32 4Km, can deal with both signed and unsigned multiplication. We introduce a radix-4 encoding and partial product generation technique for handling three different types of operands: unsigned integers, signed integers in two's complement form, and binary polynomials. In any case, the number of partial products is reduced by one half compared to the conventional radix-2 (binary) multiplication scheme.

Encoding of Unsigned Integers. The Booth Encoder depicted in Figure 1 converts the n-bit multiplier B into an equivalent radix-4 number $\widetilde{B}$ with digits in $\{-2,-1,0,1,2\}$. From a mathematical point of view, the digit set conversion is performed according to the following equation.

$$B = \sum_{i=0}^{n-1} b_i \cdot 2^i = \sum_{k=0}^{\lfloor n/2 \rfloor + 1} \widetilde{b}_k \cdot 4^k \quad \text{with} \quad \widetilde{b}_k = -2 \cdot b_{2k+1} + b_{2k} + b_{2k-1} \qquad (1)$$

We set $b_{n+1} = b_n = 0$ when B is an n-bit unsigned integer. Furthermore, b_{-1} is always 0, and therefore the least significant digit of the encoded multiplier can be computed as $\widetilde{b}_0 = -2 \cdot b_1 + b_0$. Given an n-bit unsigned multiplier B, radix-4 Booth recoding reduces the number of partial products to $\lfloor n/2 \rfloor + 1$. Applying radix-4 Booth recoding to (32×32)-bit multiplication means that we have to generate and sum up 17 partial products $P_k = \widetilde{b}_k \cdot A$ for $0 \le k \le 16$. Note that the partial product P_{16} is either 0 or the multiplicand A since $b_{33} = b_{32} = 0$ and consequently $\widetilde{b}_{16} = b_{31} \in \{0,1\}$. The fourth column of Table 1 shows the value of the partial product P_k depending on the multiplier bits b_{2k+1}, b_{2k}, and b_{2k-1}.

Table 1. Partial product generation for integers and binary polynomials

Multiplier bits			Integer mode ($fsel = 1$)				Polynomial mode ($fsel = 0$)			
b_{2k+1}	b_{2k}	b_{2k-1}	P_k	inv	trp	shl	$P_k(t)$	inv	trp	shl
0	0	0	0	0	0	0	0	0	0	0
0	0	1	$+A$	0	1	0	0	0	0	0
0	1	0	$+A$	0	1	0	$A(t)$	0	1	0
0	1	1	$+2A$	0	0	1	$A(t)$	0	1	0
1	0	0	$-2A$	1	0	1	$t \cdot A(t)$	0	0	1
1	0	1	$-A$	1	1	0	$t \cdot A(t)$	0	0	1
1	1	0	$-A$	1	1	0	$t \cdot A(t) \oplus A(t)$	0	1	1
1	1	1	0	0	0	0	$t \cdot A(t) \oplus A(t)$	0	1	1

Using the digit set $\{-2, -1, 0, 1, 2\}$ for the encoded multiplier $\widetilde{B}$ has the advantage that only the multiples $\pm 2A$, $\pm A$ of the multiplicand A are required when a radix-4 multiplication is performed with $\widetilde{B}$. The generation of partial products is therefore simply a matter of shifting and/or inverting[3] the multiplicand A.

Encoding of Signed Integers. A Booth multiplier requires to handle singed numbers properly since the partial products can have positive or negative values (see Table 1). The two's complement (TC) form is the most widespread method for representing signed numbers because it allows to treat addition and subtraction equally [22]. On the other hand, radix-4 multiplication of TC numbers requires a different Encoding since the MSB of a number in TC form has a negative weight, whereas the weight of an unsigned number's MSB is positive.

$$B = b_{n-1} \cdot (-2^{n-1}) + \sum_{i=0}^{n-2} b_i \cdot 2^i = \sum_{k=0}^{\lceil n/2 \rceil} \widetilde{b}_k \cdot 4^k ; \quad \widetilde{b}_k = -2 \cdot b_{2k+1} + b_{2k} + b_{2k-1} \quad (2)$$

For an n-bit signed multiplier B given in TC form, the encoded version $\widetilde{B}$ can be computed according to Equation (2). However, it is important to consider that modified Booth's algorithm yields the correct result only if the sign bit b_{n-1} is extended properly, i.e. we have to set $b_{n+1} = b_n = b_{n-1}$ when B is represented in TC form (b_{-1} is always 0). A second difference between unsigned and signed (TC) Booth multiplication is the number of digits in the radix-4 representation of the multiplier. Applying Equation (2) to encode an n-bit TC number B leads to a radix-4 representation $\widetilde{B}$ consisting of exactly $\lceil n/2 \rceil$ digits $\widetilde{b}_k$. Given a 32-bit multiplier B, we have to sum up 17 partial products when B is unsigned, but only 16 partial products when B is a signed number in TC form. Therefore, Booth multipliers for both signed and unsigned multiplication are generally designed

[3] Negative multiples (in two's complement form) can be obtained by inverting the corresponding positive multiple (i.e. producing the one's complement) and adding a "1" at the least significant position of the partial product [22]. This addition is typically performed in the partial product summation array or tree.

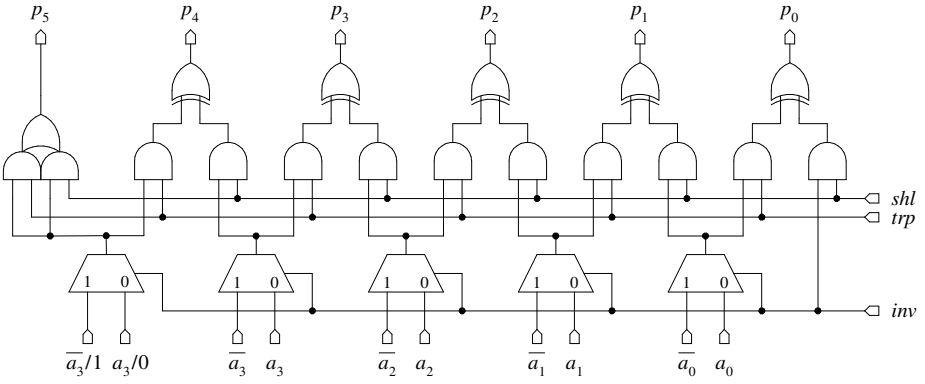

Fig. 2. Unified radix-4 PPG for signed/unsigned integers and binary polynomials

to sum up $\lfloor n/2 \rfloor + 1$ partial products, whereby the last partial product is set to 0 when a signed multiplication has to be performed [7]. This requires no extra control logic since the last partial product is automatically 0 due to the sign extension of B, e.g. for $n = 32$ we get $\widetilde{b}_{16} = -2 \cdot b_{33} + b_{32} + b_{31} = 0$ because $b_{33} = b_{32} = b_{31}$.

Encoding of Binary Polynomials. The multiplication of two binary polynomials $A(t), B(t)$ is performed in a similar way as integer multiplication, namely by generation and addition of partial products. However, signed-digit recoding techniques like Booth recoding do not work for binary polynomials [9]. On the other hand, the generation of partial products for high-radix (digit-serial) multiplication of binary polynomials is simply a matter of AND and XOR operations [16]. Digit-serial polynomial multiplication with a digit-size of $d = 2$ requires a two-coefficient operand scan in which either 0, 1, t, or $t + 1$ times the multiplicand $A(t)$ is added, depending on the corresponding coefficients of $B(t)$.

$$B(t) = \sum_{i=0}^{n-1} b_i \cdot t^i = \sum_{k=0}^{\lceil n/2 \rceil} \widetilde{b}_k(t) \cdot t^{2k} \quad \text{with} \quad \widetilde{b}_k(t) = b_{2k+1} \cdot t + b_{2k} \tag{3}$$

Consequently, any partial product $P_k(t) = \widetilde{b}_k(t) \cdot A(t)$ is a polynomial from the set $\{0,\ A(t),\ t \cdot A(t),\ t \cdot A(t) \oplus A(t)\}$. The multiplication of $A(t)$ by t is nothing else than a 1-bit left shift of the coefficients of $A(t)$.

Design of a Unified PPG. Figure 2 shows a partial product generator for signed/unsigned integers and binary polynomials. The PPG is controlled by an Encoder circuit via the three signals *inv* (invert), *trp* (transport), and *shl* (shift left). These signals depend on the multiplier bits b_{2k+1}, b_{2k}, and b_{2k-1} according to Equations (4)-(6), as can be easily derived from Table 1. An additional control signal, *fsel* (field select), allows to switch between integer-mode (*fsel* = 1) and

polynomial-mode ($fsel = 0$). The circuit shown in Figure 2 needs A (the multiplicand) and $\overline{A}$ as an input, and the multiplexors select between A and $\overline{A}$. The AND/XOR combination allows to perform a 1-bit left-shift operation.

$$inv = fsel \cdot \left(b_{2k+1} \cdot \overline{b}_{2k} \; + \; b_{2k+1} \cdot b_{2k} \cdot \overline{b}_{2k-1} \right) \tag{4}$$

$$trp = fsel \cdot \left(\overline{b}_{2k} \cdot b_{2k-1} \; + \; b_{2k} \cdot \overline{b}_{2k-1} \right) \; + \; \overline{fsel} \cdot b_{2k} \tag{5}$$

$$shl = fsel \cdot \left(\overline{b}_{2k+1} \cdot b_{2k} \cdot b_{2k-1} + b_{2k+1} \cdot \overline{b}_{2k} \cdot \overline{b}_{2k-1} \right) + \overline{fsel} \cdot b_{2k+1} \tag{6}$$

In integer mode ($fsel = 1$), the PPG works as follows: When $inv = 1$, the corresponding partial product is negative, i.e. $P_k = -2A$ or $-A$. Control signal $trp = 1$ means $P_k = \pm A$ (no left-shift). On the other hand, when $shl = 1$, a 1-bit left-shift has to be performed, i.e. $P_k = \pm 2A$. Last but not least, $P_k = 0$ is generated by $trp = shl = 0$. In integer mode, it is possible that inv and trp or shl are equal to 1 at the same time, but $trp = shl = 1$ will never occur. In polynomial mode ($fsel = 0$), the signal inv is always 0 and the PPG is directly controlled by the multiplier bits, i.e. $trp = b_{2k}$ and $shl = b_{2k+1}$. Therefore, the unified PPG works like a digit-serial polynomial multiplier with a digit-size of $d = 2$.

A parallel $(n \times n)$-bit multiplier for signed/unsigned multiplication contains $\lfloor n/2 \rfloor + 1$ PPGs and the same number of Encoder circuits. The partial products are $n + 2$ bits long as they are represented in two's complement form. However, the generation of partial products for signed multiplication differs slightly from unsigned multiplication. To process a signed multiplicand A, the sign bit a_{n-1} must be extended by one position. The leftmost multiplexor of the PPG is either connected to a_{n-1} (signed multiplication), or to 0 (unsigned multiplication).

4 Addition of Partial Products

A standard $(n \times n)$-bit radix-4 array multiplier consists of $n/2$ stages of carry-save adders (CSAs) [22]. Higher-order counters or compressors can be used to reduce the cycle time. Each CSA stage takes three inputs of the same weight and generates two outputs, i.e. the intermediate result is kept in redundant representation (the so-called carry-save form) to avoid carry propagation.

The rules of two's complement arithmetic demand an extension of the sign bit. Therefore, the $(n + 2)$-bit partial products have to be sign extended up to the $2n$-th bit position as illustrated in Figure 3. However, the actual width of the sign bit extension depends on the chosen addition method [24]. In general, sign extension should be minimized since the addition of extended sign bits increases the hardware cost and contributes to power consumption. On the other hand, the sign bits are necessary to process partial products which have a negative value. As mentioned in Section 3, a typical Booth PPG performs merely an inversion when the generation of a negative partial product is intended. This requires that a "1" is added at the least significant position of the partial product in order to get the correct two's complement representation. Most Booth multipliers do this "correction" at the CSA stage where the partial product is added, e.g. by setting the LSB of the output carry vector to 1.

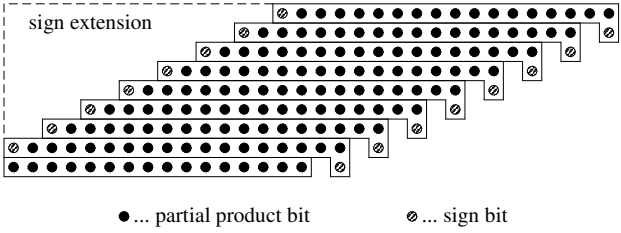

Fig. 3. Radix-4 multiplication of 16-bit unsigned integers

Each partial product, except for the last one, is $n+2$ bits long. The partial products are added according to their "weight" (i.e. in the appropriate relative position) to obtain the correct result. For instance, partial product P_1 has four times the weight of P_0, which means that it is offset by two bit positions. Note that the last partial product (i.e. P_{16} in our case) is either 0 or the multiplicand A, and hence it can never have a negative value[4]. The cumulative sum Z is a 64-bit quantity and added in the Accumulator. In our implementation, we merged the CSA stages for partial product summation with the Accumulator in order to reduce overall hardware cost and delay.

The first three partial products, P_0, P_1, and P_2, can be summed up in one CSA stage [11]. All remaining partial products require an additional CSA stage, which means that we need 16 CSA stages altogether to sum up the 17 partial products and the cumulative sum Z. The datapath of an $(n \times n + 2n)$-bit MAC unit consists of $(\lfloor n/2 \rfloor - 1) \cdot (n+2)$ adder cells to sum up the $\lfloor n/2 \rfloor + 1$ partial products. Furthermore, $2n$ adder cells are required to add the $2n$-bit quantity Z. The overall delay of the array including Accumulator is $\lfloor n/2 \rfloor$ adder cells.

We will not discuss further details in this paper since the implementation of an array multiplier is fairly simple. However, we refer the interested reader to Huang's thesis [13], which serves as an extensive and state-of-the-art reference for low-power multiplier design.

4.1 Counters and Compressors for Dual-Field Arithmetic

Savaş *et al.* define a dual-field adder (DFA) as *a full adder which is capable of performing addition both with carry and without carry* [26]. A DFA has an extra input, called *fsel* (field select), that allows to switch between integer mode and polynomial mode. When $fsel = 1$ (integer mode), the DFA operates like a standard full adder, i.e. it performs a bitwise addition with carry. On the other hand, when $fsel = 0$, the DFA's *carry* output is always 0, regardless of the input values. The *sum* output computes a 3-input XOR operation, which corresponds to the addition of binary polynomials. Therefore, we can use exactly the same hardware for the addition of both integers and binary polynomials.

[4] Recall that a radix-4 Booth multiplier has to sum up $\lfloor n/2 \rfloor + 1$ partial products when the operands are unsigned, but only $\lceil n/2 \rceil$ partial products for signed integers.

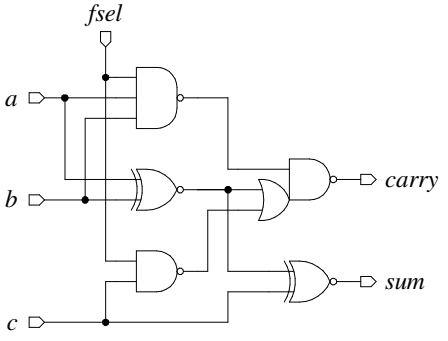

Fig. 4. Dual-field adder presented in [10]

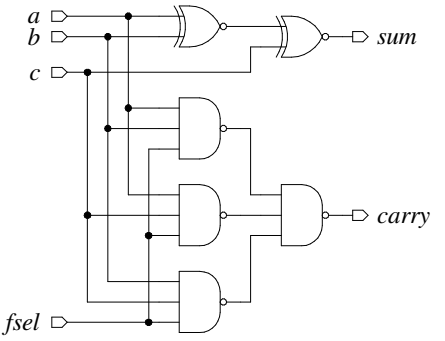

Fig. 5. DFA for (4:2) compressor design

Figure 4 shows a DFA for standard cell implementation, which was first presented in our previous work [10]. An important aspect when designing a DFA is not to increase the critical path of the circuit compared to a conventional full adder. The critical path of the DFA depicted in Figure 4 is determined by the two XNOR gates. Thus, the delay of the DFA is not larger than that of a standard full adder. We point out that in polynomial mode ($fsel = 0$), the outputs of the two NAND gates are forced to 1 and the output *cout* is always 0. Hence, only the two XNOR gates are active and contribute to power consumption. Simulations with random input patterns show that this DFA consumes approximately 42% less power in polynomial mode than in integer mode.

The DFA illustrated in Figure 5 also avoids unnecessary switching activities when polynomial addition is performed. This design has the advantage that it allows very efficient implementation of (4:2) compressors. In the past, (4:2) compressors have been widely used in tree multipliers, whereas there is relatively little previous work on applying them in array multipliers. A (4:2) compressor has the same gate complexity as two carry-save adders, but is typically faster than two cascaded CSAs because an efficient design has only three XOR delays, while each single CSA has a delay of two XORs. Consequently, when (4:2) compressors are used in an array multiplier, the delay for partial product summation is reduced by about 25%. This also reduces the power consumption as less switching activities arise when signals propagate fewer stages [13].

The DFA illustrated in Figure 5 was motivated by the balanced (4:2) compressor presented in [21]. The efficiency of the design is based on the observation that NAND gates are typically much faster than XOR/XNOR gates (in typical standard cell libraries up to twice as fast). Under this assumption, the DFA shown in Figure 5 has "fast" and "slow" inputs and outputs, respectively. For instance, input c is a fast input since the delay from c to either output is only one equivalent XNOR delay. Inputs a and b are slow inputs because their path to the *sum* output contains two XNOR gates. Similarly, the output *carry* is a fast output, whereas *sum* is a slow output. Oklobdžija *et al.* introduced an algorithmic approach for designing high-speed multipliers, taking the different

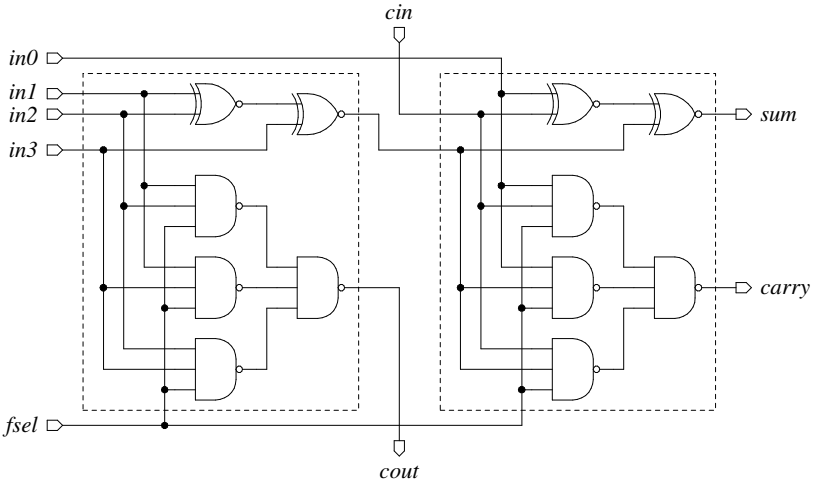

Fig. 6. Balanced (4:2) compressor composed of dual-field adders

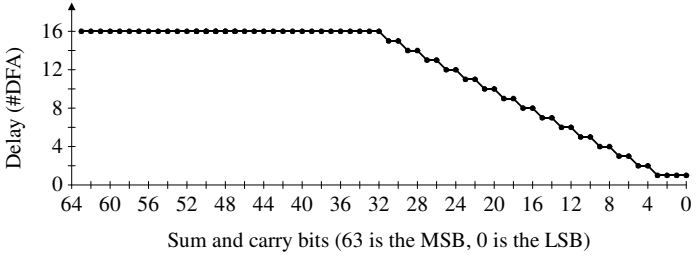

Fig. 7. Signal arrival profile of a $(32 \times 32 + 64)$-bit array multiplier

input-to-output delays of counters or compressors into account [21]. By connecting slow outputs with fast inputs and vice versa, as shown in Figure 6, a fast (4:2) compressor can be built with two DFAs. The delay of this "balanced" (4:2) compressor is only three XNOR gates (provided, of course, that two NANDs do not cause more delay than an XNOR gate).

5 Final Adder

When designing a final adder, it is important to consider the signal arrival profile of the sum and carry vector [20]. Figure 7 shows the signal arrival profile of a $(32 \times 32 + 64)$-bit array multiplier as described in Section 4. The low-order bits arrive sequentially, whilst the upmost 32 bits of the sum and carry vector arrive simultaneously. It is easily observed from Figure 3 that the first four bits have to pass only the DFA stage of the Accumulator, i.e. they arrive after a delay of one DFA cell. The next two bits of the sum and carry vector have an additional

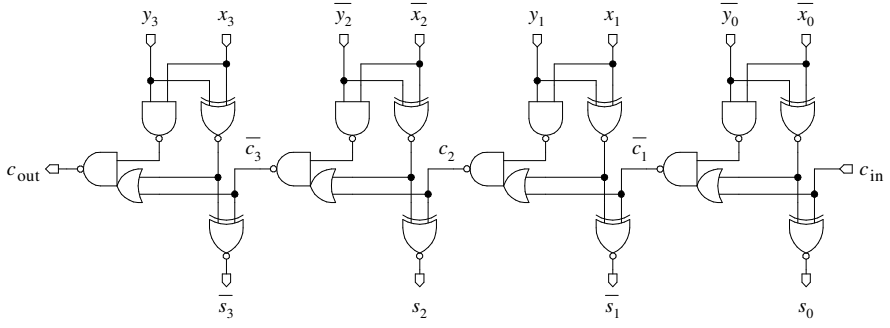

Fig. 8. Example of a 4-bit ripple-carry adder for standard-cell implementation

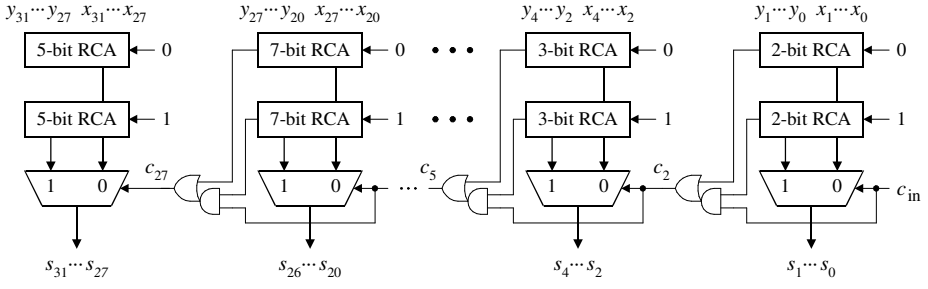

Fig. 9. Typical implementation of a 32-bit carry-select adder (see [23] for details)

delay of one DFA cell, and the following two bits have an overall delay of three DFA cells, etc. Finally, the upmost 32 bits arrive after passing 16 DFA cells.

Our final adder is realized in the same way as described in [11], namely by means of a "hybrid" adder made up of elements of carry-select addition and ripple-carry addition. For the redundant to non-redundant conversion of the topmost 32 bits, we employ a carry-select adder composed of "small" ripple-carry adders (RCAs). A carry-select adder divides the words to be added into blocks (not necessarily of the same size) and forms two sums for each block in parallel, one with a carry-in of "0" and the other with a carry-in of "1". When the actual carry of the previous block arrives, the appropriate result is selected with the help of multiplexors [22]. A major attraction of carry-select adders is the regularity of the layout.

Considering the non-equal arrival times of the low order bits, it becomes evident to use concatenated 2-bit ripple-carry adders (RCAs) for the final addition of these bits. Ripple-carry adders are generally designed so that the carry-in to carry-out delay is minimized. Figure 8 shows a possible implementation of an RCA, whereby the carry-in to carry-out delay of a k-bit RCA is only k OAI gates (excluding inverters). Figure 9 depicts the carry-select adder that we use to convert the 32 MSBs of the redundant result into binary representation.

Table 2. Comparison of (4:2) compressors for unified (i.e. dual-field) arithmetic

Implementation	#Gates (#Tr.)	Critical path	Act. gates GF(2)
Savaş *et al.* [26]	10 (60)	XOR+2AOI+2NOR	6
Au & Burgess [2]	10 (56)	3XOR	10
Prev. work [10]	10 (64)	2XNOR+2OAI	4
This work	12 (80)	3XNOR	4

6 Experimental Results and Discussion

We developed a $(32 \times 32 + 64)$-bit prototype of the MAC unit based on a 0.6 µm CMOS standard cell library. The partial product summation array (including Accumulator) consists of 572 DFA cells and has a critical path of 16 DFA cells for the addition of 17 partial products along with the extra operand Z. Measured in terms of gate delays, the critical path of the partial product summation array is only 24 XNORs since we used the balanced (4:2) compressors described in Subsection 4.1. The overall gate count of the MAC unit is roughly 10k.

Simulations based on a netlist with extracted parasitics show that the delay of our prototype is slightly less than 30 ns, which corresponds to a maximum clock frequency of 33 MHz. This clock speed is sufficient for smart card applications, even though a significant performance gain could be achieved by using a state-of-the-art CMOS technology. We also simulated the power dissipation of the MAC unit for both integer and polynomial mode. Our results indicate that the overall power consumption of the MAC unit is about 30% lower when polynomial multiplications are executed. This is primarily because the dual-field adders were designed to suppress all unnecessary switching activities when polynomial arithmetic is performed (see Figure 5)

Table 2 provides a comparison of our design with previous works that disclosed gate-level schematics of the implemented dual-field adders. We analyzed four different (4:2) compressors based on the DFAs published in [26], [2], and our previous work [10] (i.e. the DFA depicted in Figure 4). The second column indicates the gate and transistor count of the compressors. The critical path shown in the third column was determined by interconnecting two DFAs so that the overall delay is minimized. The fourth column lists the number of active gates in polynomial mode, which is an indicator for the power consumption.

The (4:2) adder introduced by Au and Burgess is small and very fast, but has high power consumption in polynomial mode, which is due to the encoding they used. Savaş *et al's* design has a longer critical path and does also show unnecessary power consumption in polynomial mode. The critical path of our previous design [10] is slightly longer than the minimum of three XOR/XNOR gates. Note that unnecessary switching activities do not take place in our previous design, which is also the case for the design described in this paper. The present design features the minimum delay of three XNOR gates, i.e. it combines a short critical path with low power consumption. However, the gate count of the proposed (4:2) compressor is slightly higher than that of the others.

7 Conclusions

We presented the design of a $(32 \times 32 + 64)$-bit multiply/accumulate (MAC) unit for signed/unsigend integers and binary polynomials. It performs multiply and multiply-and-add operations within one clock cycle and can be easily integrated into general-purpose RISC processors. The datapath of the MAC unit is modular, consists of only very few basic cells, and results in a very regular and compact layout. Our design is based on a unified radix-4 partial-product generator in combination with an optimized (4:2) compressor for signed/unsigned integers and binary polynomials. The extra functionality for polynomial arithmetic is very simple to integrate into a standard multiplier and causes virtually no speed penalty. A slight increase in silicon area is tolerable for most applications, especially when considering that a cryptographic co-processor would require much more silicon area than the modification of a standard multiplier datapath for dual-field arithmetic.

The main result of our work is that a properly designed dual-field multiplier consumes significantly less power in polynomial mode than in integer mode (approximately 30% in our implementation). Instruction set extensions along with low-power functional units open up new options for fast and energy-efficient software implementation of elliptic curve cryptography. For many applications, including smart cards and other kinds of embedded systems, the functional unit can eliminate the need for a cryptographic co-processor.

References

1. ARM Limited. ARM SecurCore Solutions. Product brief, available for download at `http://www.arm.com/aboutarm/4XAFLB/$File/SecurCores.pdf`, 2002.
2. L.-S. Au and N. Burgess. A (4:2) adder for unified GF(p) and GF(2^n) Galois field multipliers. In *Conference Record of the 36th Asilomar Conference on Signals, Systems, and Computers*, vol. 2, pp. 1619–1623. IEEE, 2002.
3. I. F. Blake, G. Seroussi, and N. P. Smart. *Elliptic Curves in Cryptography.* Cambridge University Press, 1999.
4. M. Bucci. Dual mode (integer, polynomial) fast modular multipliers. Presentation at the Rump Session of EUROCRYPT '97, Konstanz, Germany, May 13, 1997.
5. J.-F. Dhem. Efficient modular reduction algorithm in $\mathbb{F}_q[x]$ and its application to "left to right" modular multiplication in $\mathbb{F}_2[x]$. In *Cryptographic Hardware and Embedded Systems — CHES 2003*, LNCS 2779, pp. 203–213. Springer Verlag, 2003.
6. W. Drescher, K. Bachmann, and G. Fettweis. VLSI architecture for datapath integration of arithmetic over GF(2^m) on digital signal processors. In *Proceedings of the 22nd IEEE Int. Conference on Acoustics, Speech, and Signal Processing (ICASSP '97)*, vol. 1, pp. 631–634. IEEE, 1997.
7. A. A. Farooqui and V. G. Oklobdžija. General data-path organization of a MAC unit for VLSI implementation of DSP processors. In *Proceedings of the 31st IEEE Int. Symposium on Circuits and Systems (ISCAS '98)*, vol. 2, pp. 260–263. IEEE, 1998.
8. J. R. Goodman and A. P. Chandrakasan. An energy efficient reconfigurable public-key cryptography processor architecture. In *Cryptographic Hardware and Embedded Systems — CHES 2000*, LNCS 1965, pp. 175–190. Springer Verlag, 2000.

9. J. Großschädl. A unified radix-4 partial product generator for integers and binary polynomials. In *Proceedings of the 35th IEEE Int. Symposium on Circuits and Systems (ISCAS 2002)*, vol. 3, pp. 567–570. IEEE, 2002.
10. J. Großschädl and G.-A. Kamendje. Instruction set extension for fast elliptic curve cryptography over binary finite fields GF(2^m). In *Proceedings of the 14th IEEE Int. Conference on Application-specific Systems, Architectures and Processors (ASAP 2003)*, pp. 455–468. IEEE Computer Society Press, 2003.
11. J. Großschädl and G.-A. Kamendje. A single-cycle $(32 \times 32 + 32 + 64)$-bit multiply/accumulate unit for digital signal processing and public-key cryptography. Accepted for presentation at the 10th IEEE Int. Conference on Electronics, Circuits and Systems (ICECS 2003), scheduled for Dec. 14-17, 2003 in Sharjah, U.A.E.
12. H. Handschuh and P. Paillier. Smart card crypto-coprocessors for public-key cryptography. In *Smart Card Research and Applications — CARDIS '98*, LNCS 1820, pp. 372–379. Springer Verlag, 2000.
13. Z. Huang. *High-Level Optimization Techniques for Low-Power Multiplier Design.* Ph.D. Thesis, University of California, Los Angeles, CA, USA, 2003.
14. Ç. K. Koç and T. Acar. Montgomery multiplication in GF(2^k). *Designs, Codes and Cryptography*, 14(1):57–69, Apr. 1998.
15. O. L. MacSorley. High-speed arithmetic in binary computers. *Proceedings of the IRE*, 49(1):67–91, Jan. 1961.
16. M. C. Mekhallalati, A. S. Ashur, and M. K. Ibrahim. Novel radix finite field multiplier for GF(2^m). *Journal of VLSI Signal Processing*, 15(3):233–245, Mar. 1997.
17. MIPS Technologies, Inc. MIPS32 4Km$^{\text{TM}}$ processor core family data sheet. Available for download at `http://www.mips.com/publications/index.html`, 2001.
18. MIPS Technologies, Inc. SmartMIPS Architecture Smart Card Extensions. Product brief, available for download at `http://www.mips.com/ProductCatalog/P_SmartMIPSASE/SmartMIPS.pdf`, 2001.
19. E. M. Nahum, S. W. O'Malley, H. K. Orman, and R. C. Schroeppel. Towards high performance cryptographic software. In *Proceedings of the 3rd IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems (HPCS '95)*, pp. 69–72. IEEE, 1995.
20. V. G. Oklobdžija. Design and analysis of fast carry-propagate adder under non-equal input signal arrival profile. In *Conference Record of the 28th Asilomar Conference on Signals, Systems, and Computers*, vol. 2, pp. 1398–1401. IEEE, 1994.
21. V. G. Oklobdžija, D. Villeger, and S. S. Liu. A method for speed optimized partial product reduction and generation of fast parallel multipliers using an algorithmic approach. *IEEE Transactions on Computers*, 45(3):294–306, Mar. 1996.
22. B. Parhami. *Computer Arithmetic: Algorithms and Hardware Designs.* Oxford University Press, 2000.
23. J. M. Rabaey. *Digital Integrated Circuits – A Design Perspective.* Prentice Hall, 1996.
24. O. Salomon, J.-M. Green, and H. Klar. General algorithms for a simplified addition of 2's complement numbers. *IEEE Journal of Solid-State Circuits*, 30(7):839–844, July 1995.
25. A. Satoh and K. Takano. A scalable dual-field elliptic curve cryptographic processor. *IEEE Transactions on Computers*, 52(4):449–460, Apr. 2003.
26. E. Savaş, A. F. Tenca, and Ç. K. Koç. A scalable and unified multiplier architecture for finite fields $GF(p)$ and $GF(2^m)$. In *Cryptographic Hardware and Embedded Systems — CHES 2000*, LNCS 1965, pp. 277–292. Springer Verlag, 2000.