Scalable Instruction Set Extension for Dual-field Public-key Cryptosystem

Wang Liao, Meilin Wan, Kui Dai and Xuecheng Zou

Abstract—**As the higher security per bit compared with traditional symmetric-key cryptography, public-key cryptography has always been attractive in security system. But usually the cost is high and efficiency is low because of complex algorithm. Unlike common hardware solution based on FPGA or system on chip (SOC), an instruction set architecture (ISA) extension of embedded processor has been proposed. Firstly the extended function unit is introduced, and the structure is scalable according to different applications. Then the extended instruction set is proposed under a new architecture, to overcome the weakness of traditional ISA extension, such as the flexibility for multiple extended functions and the difficulty of implementation. Opposite to original ISA, detail operation of extended instruction has been treated as side effects of data transfer, to keep the architecture of embedded processor and compilation tools basically unchanged. Test results show that point multiplication on GF(2^{160}) can be done in 181 us, by the cost of 124k gates.**

Index Terms—**dual-field, flexibility, ISA extension, public-key, scalability**

I. INTRODUCTION

Public-key cryptography such as Rivest-Shamir-Adleman (RSA) and Elliptic Curve Cryptography (ECC) has been used in many areas like securing e-mail, wireless communication and electronic commerce since the introduction by Difffie and Hellman [1]. Lots of previous work to implement RSA and ECC can be divided into software solution running on general embedded processor and hardware solution based on co-processors or FPGA. They represent the extreme pursuit to cost or performance respectively. But in practice the balance between performance and cost, along with the flexibility to satisfy various applications are mostly important. We have proposed a scalable instruction set architecture (ISA) extension aimed at public-key cryptography based on embedded processor. Requirements of different applications focused either on performance or cost can be met by reconfigurable basic arithmetic units. Moreover, our work does not impact the original architecture and compilation tools, to provide a universal architecture of instruction extension for various functions.

The paper is organized as follows. In Section II background knowledge and some algorithm is discussed. Section III proposed the scalable function units that can meet different requirement. Detail architecture of instruction extension is shown in Section IV. Evaluation of performance and cost of some typical circumstance is reported in Section V. Finally, concluding remarks are presented in Section VI.

Wang Liao, Meilin Wan, Kui Dai and Xuecheng Zou are with the School of Optical and Electronic Information, Huazhong University of Science and Technology, Wuhan, CO 430074 P.R.China
Wang Liao (e-mail: liaowangww@163.com).
Meilin Wan: (email: D201277512@hust.edu.cn)
Kui Dai (email: josh.maxview@gmail.com)
Xuecheng Zou (email: estxczou@gmail.com)

II. PRELIMINARIES

As we know, modular arithmetic which consists of modular multiplication, addition/subtraction, power and inverse is the basic operation of most public-key algorithm. Power and inverse can be transformed to modular multiplication by certain algorithm. To avoid division which is expensive and inefficient in modular multiplication, Montgomery algorithm based on words is introduced and widely accepted [2], which is shown in Algorithm 1, *m* is the digit of prime number *p* over finite field or the degree of irreducible polynomial over binary field, *r* is the width of arithmetic units, thus *w* = *m/r* is words.

Algorithm 1: *Montgomery modular multiplication*

```
Input: $a,b,p,q,2^m \times 2^{-m} - p \times q = 1$
Output: $c = a \times b \times 2^{-m} \pmod p$
1. $c = 0$
2. for $i = 0$ to $w-1$ by $+1$ do
3.   $z = 0$
4.   $\{z,c_0\} = c_0 + a_i \times b_0$;
5.   $t = c_0 \times q \pmod{2^r}$
6.   $\{z,c_0\} = \{z,c_0\} + t \times p_0$
7.   for $j = 1$ to $w-1$ by $+1$ do
8.     $\{z,c_j\} = c_j + a_i \times b_j + z$
9.     $\{z,c_{j-1}\} = \{z,c_j\} + t \times p_j$
10.  endfor
11.  $c_{w-1} = z$
12. endfor
```

Modular addition/subtraction over finite field can be replaced by normal addition/subtraction, as the modulo can be handled by following modular multiplication. While for binary field it simply equals XOR as there is no carry bit. Therefore a specific instruction support modular multiplication over dual-filed will be the main purpose of our ISA extension.

III. SCALABLE MODULAR MULTIPLIER

It can be concluded from Algorithm 1 that Montgomery modular multiplication mainly consists of successive multiplication and addition with long digits, which can be divided into internal and external loops. A basic architecture is proposed in Fig. 1, including memory block, operand fetch, operation core (OC), temporary queue and writing back control. Due to the massive data amount, an independent memory block is needed, and usually SRAM is adopted because of the compact capacity. The operand fetch is divided into internal loops (B_j and P_j) and external loops (A_i). Arithmetic units including multiplier and adder are in operation core for most computing task. Temporary queue is designed to store the intermediate results next loop needed (Z_i). There is also writing back path to control the final results back to the specific address in memory block.

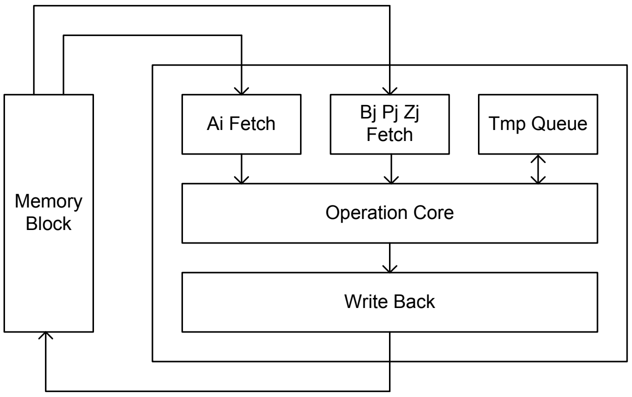


Fig 1. Basic architecture of Montgomery multiplier

A. Optimization within internal loops

The possibility of parallelism among both internal loops and external loops is investigated to optimize the performance at most. Finite field is taken as example. The maximum parallelism among internal loops is easy to achieve. Firstly the operation of internal loops can be classified into 6 types which are listed below.

$$A: c_i = Z_{00} + a_i \times b_0$$

$$B: t_i = c_i \times q$$

$$C: t_i \times p_0$$

$$D_1: a_i \times b_{j+1}$$

$$D_2: t_i \times p_{j+1}$$

$$D_3: Z_{ij} = Z_{i-1\,j} + a_i \times b_j + t_i \times p_j$$

The data dependence either among the first three operation or between the first three and the last three is obvious and inevitable, while the last three can be executed in parallel provided there is sufficient computing resource, which means two multipliers and one adder is needed in the operation core.

B. Optimization within internal loops

Then for the parallelism among external loops, firstly the data structure of Algorithm 1 is expressed in Fig. 2. Having sufficient arithmetic units, $D_1D_2D_3$ have been merged into D. From left to right the consecutive D make up internal loops, while from top to bottom each row stands for one of the external loops. The arrows between each row represent the data dependence among external loops. Despite the data dependence, partial overlap can be achieved given there are multiple operation cores.

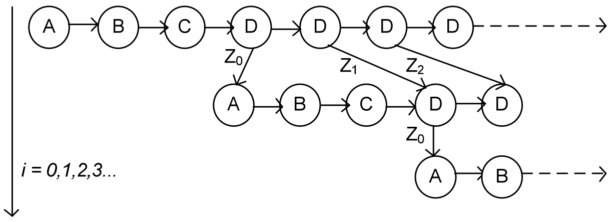


Fig 2. Data structure of Algorithm 1 with multiple OCs

It can be figured out that the number of rows needed is decided by the length of each row, which means to achieve maximum parallelism among external loops, the width of arithmetic units (AU) and the parameter of specific application need to be considered. If the length of operands is n bits, and the width of operation core is w, it means the words $r=n/w$. The length of each row can be represented as $l=3+n/w+1$, as there is an additional external loop for carry bit on binary field. So when the first row ends, which means the first operation core can handle next external loop, the total number of operation cores can be represented as $c=l/4=(4+r)/4=1+n/4w$.

We have chosen $w=8$ and $w=32$ as two most typical circumstance, and the relationship between operand length and operation cores needed is shown in Fig.3. Considering the length of critical path and exponential growth of cost, the width of arithmetic units should better not exceed 32 bits.

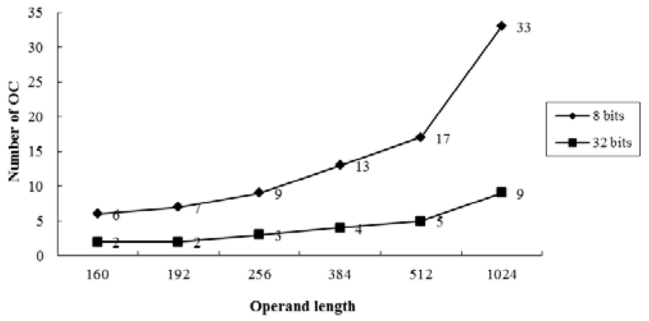


Fig 3. Relationship between operand length and OC needed with different width of arithmetic units

It should be reminded that the operands' width and the corresponding architecture means the maximum parallelism will be achieved. Larger operands are also supported, although the potential of parallelism will not be released to most. For example, if for some application, the performance over GF(256) is most important, 8 bits width with 9 OCs or 32 bits width with 3 OCs should be considered, further decision can be made according to the detail requirements of performance or cost.

C. Support for dual-field

To support the operation over binary field at the same time, another two multipliers and one adder aimed for binary field in each OC are needed. Since the algorithm of both field are basically identical, the arithmetic units for different fields can be combined under the same data path. Moreover, as there is no carry bit over binary field, the adder can be simply implemented by XORs, while the 4-2 compressors in typical multiplier can be replaced by XORs with 4 inputs. Also there will be one less external loop and the performance of operand with same length will be better. Finally, architecture of the multiplier with multiple OCs is shown in Fig. 4, where 3 OCs are taken as example.

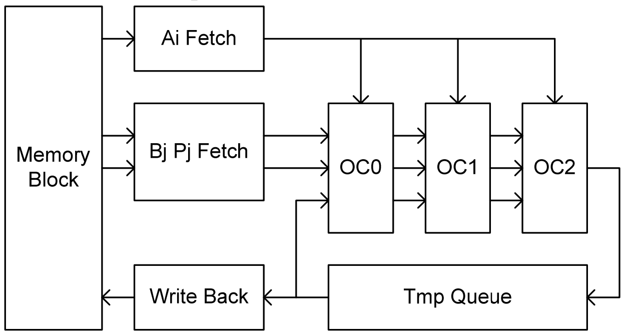
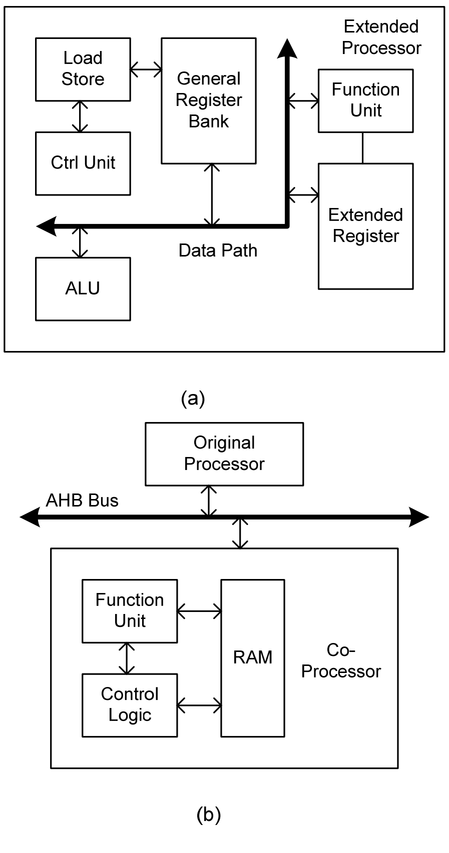


Fig 4. Architecture of Montgomery multiplier with 3 OCs

IV. ISA EXTENSION ARCHITECTURE

Solutions to implement public-key cryptography can be classified according to the relationship between function unit and the embedded processor. Fig. 5a represents traditional ISA extension whose function unit has a direct connection with the microprocessor core and is embedded into the pipeline [3]. The typical hardware solution is shown in Fig. 5b, whose function unit together with other control logic is treated as co-processor, and the connection to microprocessor may be general IO or some general bus like AHB [4][5][6]. Fig. 5c can be regarded as a compromise of the previous two, the function unit is implemented by some programmable logic arrays like FPGA and the connection is similar to the co-processor solution [7]. Besides, there is also the basic software solution based on original ISA, which is a special case as there is no extended function unit.



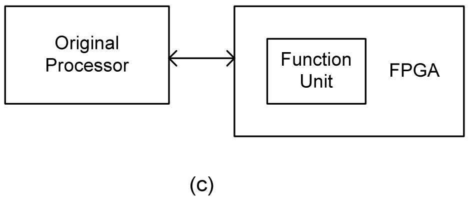


Fig 5. Typical solution for public-key cryptography

The comparison result is shown in Table I. Three aspects including performance, flexibility and difficulty of implementation are rated from A to C. Except basic software solution, the premise is the computing resource of function unit is identical, so the performance here mainly reflect the efficiency of interface between microprocessor and function unit.

TABLE I
COMPARISON BETWEEN SOLUTIONS

	Performance	Flexibility	Implementation
ISA extension	A	C	C
Co-processor	C	B	B
Programmable logic	B	A	A
Basic software	C	A	A

It can be figured that each solution has their advantages and weakness. We have proposed a new ISA extension architecture which can be regarded as an application of Transport Triggered Architecture (TTA) [8], to improve the flexibility and lower the difficulty of implementation while keep the merit in performance at the same time.

A. Overall architecture

All the disadvantages of traditional ISA extension can be concluded to the modification of original architecture and compilation tools when new instruction is added. We have noticed that all extended function can be treated as dataflow among different module. If all data transfer is programmed explicitly and all function units are addressed under a unified space, only instructions for data access is needed no matter what function to be extended.

The architecture of embedded processor extended for public-key cryptography is shown in Fig. 6, where the original architecture of ARM is taken as an example.

We can figure out that as an extended function unit, the Montgomery multiplier along with its local memory is connected to B bus and can be simply treated as the extension of general register bank under a unified address space. The address space Montgomery multiplier occupied can be divided into three types, which consist of operand register, trigger register and result register. The meaning of operand and result register is obvious, for trigger register, writing into it triggers the corresponding function unit.

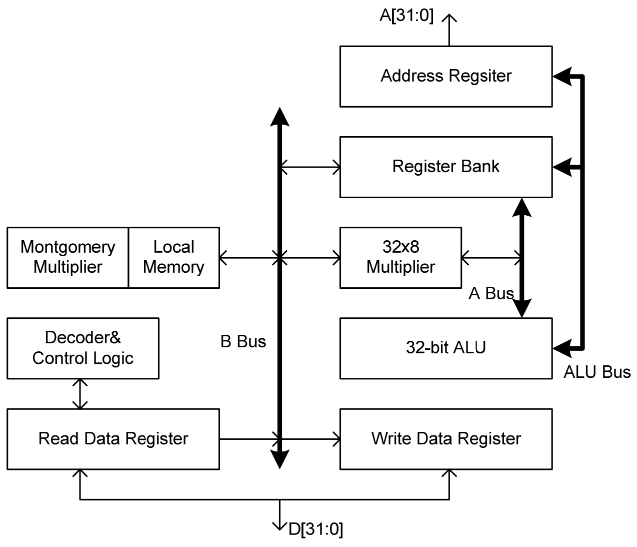
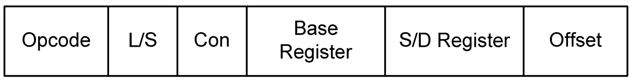


Fig 6. Architecture of extended embedded processor based on ARM

B. Instruction definition

Still ARM is taken as example. As address space in original LOAD/STOR only support 16 general registers, new data access instruction namely LOADF/STORF need to be defined for extra address space. The format and bits allocation is shown in Fig. 7.



31~28: operation code
27: 0 for store, 1 for load
26: condition guard
25~21: register stored base address
20~16: source or destination register
15~0: offset address to be added on base address

Fig 7. New data access instruction definition

LOADF and STORF share one operation code, and they are distinguished by the L/S bit. Condition guard is used to control conditional execution. The bits allocation of addressing related content can be customized according to the number of function units extended and the size of their local memory, while in Fig.6 the address bits have been extended to 5, so that 32 general registers are supported. The value of offset can be either immediate data or another register ID.

The pipeline diagram for LOADF/STORF is shown in Fig. 8, which is basically identical with original ARM pipeline despite that Execute (EX) is replaced by Transport (TR). PO means Possible Operation, which is treated as side effect of TR and varies according to the specific function unit. As all data access is explicit, programmers can arrange their program according to the PO of specific applications to maximize the potential of pipeline.

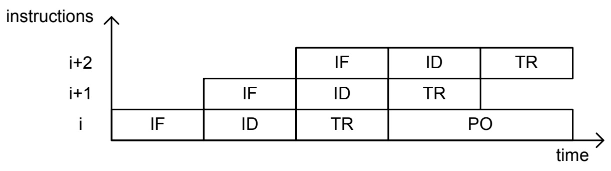


Fig 8. Pipeline Diagram for LOADF/STORF

V. EVALUATION AND COMPARISON

Performance and cost of related works are listed in Table II, and point multiplication on GF (2^{160}) is taken as example. 8 bits with 6 OCs and 32 bits with 2 OCs of our work are evaluated to meet different requirements focused on cost or performance respectively. Firstly compared to traditional ISA extension in [3], with little additional computing resource, the promotion in performance is great. The frequency is not mentioned and we assume it is the same as ours since the processor is also ARM. Then co-processors implemented by ASIC or FPGA is listed. The cost of [4][5][7] is a little higher than the 32 bits version of our work, but our performance is much better, and [7] only support the binary field. The performance of [6] is slightly better than our work but the cost of resource is ten times more. Moreover, co-processors still need an embedded general processor to form the entire system on chip (SOC), which has already been included in our works. For the 8 bits version of our work, although the performance is not very outstanding, the cost is quite low. It is suitable for some application need extremely low cost with an acceptable performance.

Table II
EVALUATION AND COMPARISON

Design	Tech	Area	Logic gates	Field	Frequency	Time
This work (8 bits)	0.13 um CMOS	0.47 mm^2	59k gates	GF (2^{160})	233MHz	746 us
This work (32 bits)	0.13 um CMOS	0.98 mm^2	124k gates	GF (2^{160})	233MHz	181 us
[3]	ARM	\	\	GF (2^{160})	233MHz	3519 us
[4]	0.13 um CMOS	1.35 mm^2	179k gates	GF (2^{160})	158MHz	272 us
[5]	0.18 um CMOS	1.64 mm^2 *	175k gates	GF (2^{160})	249MHz	220 us
[6]	0.18 um CMOS	18.6 mm^2	1984k gates *	GF (2^{160})	250MHz	169 us
[7]	EP3SL340H1152C3	\	97899 LUTs	GF (2^{160})	143MHz	355 us

*: estimated

VI. Conclusion

A scalable ISA extension of embedded processor for public-key cryptography is proposed, while the scalability can be reflected in two aspects. Firstly, the scalable Montgomery multiplier is adopted to achieve balance between performance and cost. Different requirements of various applications focused either on performance or cost can be met by adjusting the parameter of basic arithmetic units. Then the scalability at system level is achieved by a new architecture for ISA extension. Opposite to traditional ISA, detail operation is treated as the side effect of data transfer, so the original architecture and compilation tool need not to be modified when new instruction is added. Furthermore, not only the public-key related instruction discussed in this paper, but any new instruction can be extended under this architecture simply after a definition in address space. At last, performance and cost are compared with previous works, where 8 bits with 5 OCs and 32 bits with 2 OCs are taken as the two most representative versions.

References

[1] W. Diffie, M.E. Hellman, "New Directions in Cryptography," *IEEE Trans. Information Theory*, vol. 22, pp. 644-654, 1976.

[2] A. F. Tenca and C¸ etin Kaya Koc¸ "A scalable architecture for modular multiplication based on montgomery's algorithm" ， *IEEE Trans. Computers*, 52(9):1215–1221, 2003.

[3] Bartolini, S. ; Dipt. di Eng. dell"Inf., Univ. di Siena, Siena, Italy ; Castagnini, G. ; Martinelli, E., " Inclusion of a Montgomery Multiplier Unit into an Embedded Processor's Datapath to Speed-up Elliptic Curve Cryptography", Information Assurance and Security, 2007. IAS 2007. Third International Symposium on

[4] J.-Y. Lai and C.-T. Huang, "Energy-Adaptive Dual-Field Processor for High-Performance Elliptic Curve Cryptographic Applications" IEEE Trans. on very large scale intergration (VLSI) systems.vol 56, no. 4,pp.356-360,March 2010.

[5] ZHONG Xian-hai, XU Jin-fu, YAN Ying-jian, "Parallel and Reconfigurable ECC Application Specific Instruction-set Coprocessor" *Computer Engineering*, vol 5, pp 153-155, March 2009.

[6] YANG Xiao-hui, DAI Zi-bin, LI Miao, ZHANG Yong-fu "Research and design of parallel architecture processor for elliptic curve cryptography," *Journal on Communications*, vol. 5, pp 70-76, 2011

[7] ZHANG Jun, YANG Xiao-hui, ZHAO Qian-jin, YANG Tong-jie, DAI Zi-bin, "Elliptic curve cryptography coprocessor based on specific instruction set," *Computer Engineering*, vol. 3, pp 111-113, 2011.

[8] H.Corporaal, "Microprocessor Architectures: From VLIW to TTA", Chichester,UK:John Wiley&Sons,1997