Scalable VLSI Design for Fast GF(p) Montgomery Inverse Computation

Adnan Abdul-Aziz Gutub[1], Erkay Savas[2], and Tatiana Kalganova[3]

[1] *Department of Computer Engineering,*
King Fahd University of Petroleum & Minerals, Dhahran 31261, Saudi Arabia

[2] *Faculty of Engineering & Natural Sciences,*
Sabanci University, Istanbul, Turkey TR-34956

[3] *Department of Electronic and Computer Engineering*
Brunel University, Uxbridge, Middlesex, UK, UB8 3PH

Email: gutub@kfupm.edu.sa

Abstract

This paper accelerates a scalable GF(p) Montgomery inversion hardware. The hardware is made of two parts a memory and a computing unit. We modified the original memory unit to include parallel shifting of all bits which was a task handled by the computing unit. The new hardware modeling, simulating, and synthesizing is performed through VHDL for several 160-bits designs showing interesting speedup to the inverse computation.

Keywords: Montgomery inverse, Elliptic curve cryptography, Scalable hardware design

1. Introduction

The addition, multiplication and inversion, arithmetic computations in GF(p) have several applications in cryptography, such as RSA algorithm [1], Diffie-Hellman key exchange algorithm [2], the US federal Digital Signature Standard [3] and also elliptic curve cryptography (ECC) [4,5]. ECC is the main focus of this work since its promise to replace older public-key crypto systems [6]. Cryptography with key size of 160-bits in ECC is equivalent in security to 1024 bits RSA [7] which made our design choice of 160-bits. Recently, speeding up inversion operations in GF(p) have been gaining attention since inversion is the most time consuming operation in elliptic curve cryptographic algorithms when affine co-ordinates are selected [6]. Although GF(p) inversion can be performed in software or in hardware, hardware is preferred to gain the best in speed and security [8].

Modular inversion in hardware is often performed by algorithms based on the Extended Euclidean algorithm [6]. Several attempts [9-11] have investigated the GF(p) inversion targeted to field programmable gate array (FPGA) implementations. The FPGA models face extra delay to propagate the carry from top to bottom between different FPGA columns.

Different inversion hardware were proposed by Feldhofer [12] and Zhou [13]. Feldhofer hardware performs inversion but slow and complex due to the usage of Fermat's Theorem [12]. Zhou in [13], designed a VLSI implementation for GF(p) inversion computation using one simple adder that suffered from the long propagation carry chain making the operation clock frequency limited and the design area and complexity not flexible to accommodate the changing demand of the crypto applications.

The Montgomery modular inverse algorithm and hardware suitable for this research is presented in [8]. The algorithm is implemented in hardware using scalability features, which allows the use of a fixed-area scalable circuit to perform inversion of unlimited precision operands. The hardware divides the long-precision numbers in words and each word is processed in a clock cycle. This research proposes speeding up the process by making the shifting operation with the memory unit instead of the scalable computing unit. The results will show the speedup gained and the extra hardware area needed. The conclusion proofs that the extra hardware added is worth the expected speedup with the VHDL measurements.

2. Modified Hardware

The original scalable inversion hardware is built of two main parts, a memory unit and a computing unit. The reader is referred to [8] for detailed information on the original hardware and how it is built. The inversion hardware is to run two hardware algorithms in series to compute the Montgomery inverse needed. These algorithms are known as the almost Montgomery inverse (AlmMonInv) and the correction phase (CorPh) procedures which are represented for hardware as HW_Alg1 and HW_Alg2, respectively, as shown below.

AlmMonInv Hardware Algorithm (HW-Alg1)

Registers: u, v, r, s, & p (all five registers hold n bits).
Input: $a \in [1, p\text{-}1]$, p = modulus; where $2^{n-1} \le p < 2^n$
Output: result$\in[1, p\text{-}1]$ & k;
where result=$a^{-1}2^k \bmod p$ & $n \le k \le 2n$

1. u = p; v = a; r = 0; s = 1; k = 0
2. if (u_0 = 0) then { u = ShiftR(u,1) ;
   s = ShiftL(s,1)}; goto 7
3. if (v_0 = 0) then { v = ShiftR(v,1) ;
   r = ShiftL(r,1)}; goto 7
4. S1 = Subtract (u, v); S2 = Subtract (v, u);
   A1 = Add (r, s)
5. if($S1_{borrow}$=0)then{u=ShiftR(S1,1));r=A1;
   s=ShiftL(s,1)};goto 7
6. s = A1; v = ShiftR(S2,1); r = ShiftL(r,1)
7. $k = k + 1$
8. if ($v \neq 0$) go to step 2
9. S1 = Subtract (p, r); S2 = Subtract ($2p$, r)
10. if($S1_{borrow}$=0)then{return result=S1};
    else {return result=S2}

The *correction phase(CorPh)* [8] algorithm is shown as HW-Alg2 below:

CorPh Hardware Algorithm (HW-Alg2)

Registers: r & p (two registers to hold n bits).
Input: r,p,n,k; where r ($r = a^{-1}2^{k-n} \bmod p$)& k from *AlmMonInv*
Output: result; where result = $a^{-1}2^n \pmod p$.

11. $j = 2n\text{-}k\text{-}1$
12. While $j>0$
13. r = ShiftL(r,1); $j = j$-1
14. S1 = Subtract(r, p)
15. if ($S1_{borrow}$ = 0) then {r = S1}

The new 160-bits modified hardware remodeled both the computing and memory unit, as shown in Fig. 1. The shifting operation is removed from the computing unit. Instead, the memory is changed to a bidirectional single bit shifting register. Each FIFO within the memory block is improved to perform the shifting by adding n_{max} multiplexers. The multiplexers will reroute and organize passing the data to it self or to the next cell for shifting as in the original architecture, or they (the multiplexers) direct the data to be shifted right or left. The memory & shifter unit will follow the operations as controlled by the HW-Alg1 and HW-ALg2.

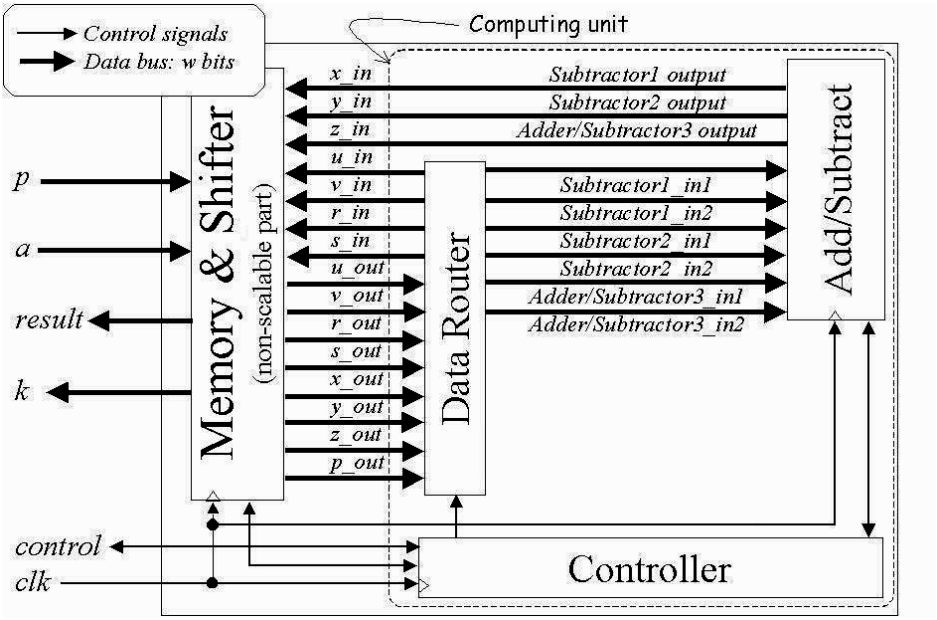

Figure 1: Improved 160-bits inversion hardware block diagram

3. Area Comparison

The area of any VLSI hardware depends on the technology and minimum feature size. For technology independence, the number of equivalent gates are used as area measure. A CAD tool from Mentor Graphics (Leonardo) was used. Leonardo takes the VHDL design code and provides a synthesized design with its area and longest path delay. The target technology is a *0.5µm* CMOS defined by the 'AMI0.5 fast' library provided in the ASIC Design Kit (ADK) from the same Mentor Graphics Company [14].

The areas of scalable hardware depends on the maximum number of bits it can handle (n_{max}) and the scalable word size (w). All designs are built for $n_{max} = 160$-bits. Changing w give different scalable designs areas as shown in Fig. 2; this compares between all new scalable designs and the old scalable ones of [8]. It is clear for all designs that as w increases the area is getting larger.

4. Delay Comparison

The exact computation time is estimated by the number of cycles multiplied by the clock cycle period. It was found from the VHDL synthesis that the new hardware clock period is not affected by the shifting modification of this work, which made the clock period of the new hardware depend on the value of w, exactly as the clock period of the original old hardware of [8].

The number of clock cycles for all designs depends completely on the data and its computation. The computation time of the new hardware to run the AlmMonInv algorithm is estimated by probability study as in [8]. See the AlmMonInv algorithm (HW-Alg1) represented earlier. Simulating this algorithm proofed that almost 25% of the k cycles is consumed by step 2 and 25% is for step 3. Steps 4, 5, and 6 are a sequence that runs consuming 50% of the k iteration. After the k iterations, step 9 is performed once which needs to considered in the time estimation too. Note

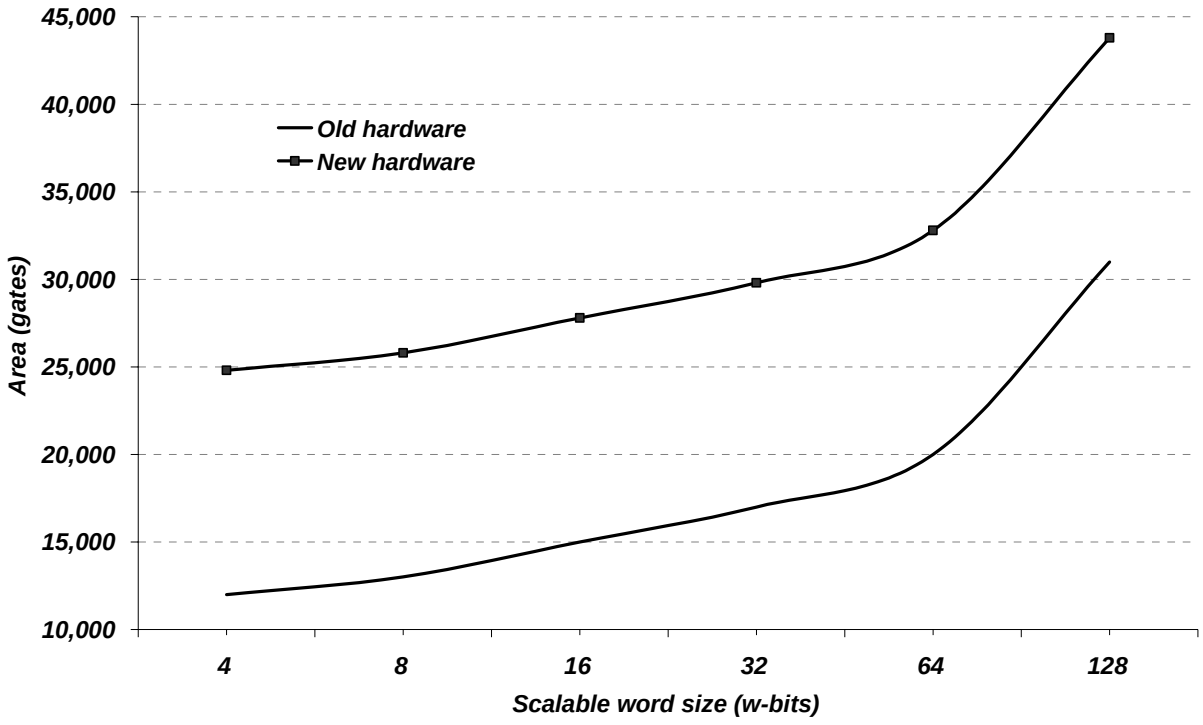

Figure 2: Area comparison of all scalable designs for 160-bits

that each shifting operation is performed in one cycle independent to the number of words the hardware is having, while the addition and subtraction needs to be performed within $\lceil n/w \rceil$ cycles. These points made the AlmMonInv Computation Time as follows:

Cycles for steps 4,5,6 = 0.5 k ($\lceil n/w \rceil$+1)
Cycles for step 9 = $\lceil n/w \rceil$
Cycles for steps 2,3 = 0.5 k
Total AlmMonInv Clock Cycles =
0.5 k ($\lceil n/w \rceil$+1)+ $\lceil n/w \rceil$+ 0.5 k

The computation time of correction phase algorithm (HW-Alg2) depend on the total number of iterations and some extra cycles within the iterations due to scalability. The single bit shifting number of iterations is *2n-k-1*, assuming on average *k=1.5n*, will result: *2n-1.5n-1≈ 0.5n* [8].

HW-Alg2 will need this number of iterations to process step 13 followed by step 14. Step 14 needs the extra scalability cycles of $\lceil n/w \rceil$as detailed below:

Cycles for step 13 = 0.5 n
*Cycles for step 14 = 0.5 n * $\lceil n/w \rceil$*
*Total CorPh Clock Cycles = 0.5 n + 0.5 n * $\lceil n/w \rceil$*

Several scalable hardware configurations are designed depending on different *w* parameters. Each configuration can have different computation time depending on the actual number of bits, *n*, used. For example, Fig. 3 compares the delay of six scalable hardware designs of both types, the new hardware and the old ones of [8]. All architectures are designed for maximum bits of $n_{max}=160$ bits, however, in reality the ECC actual number of bits (*n*) can be less. Note that the difference in *n* affects on the speed of the designs. i.e., as *n* reduces, the overall computing time of any scalable design reduces. This is a major advantage of the scalable hardware over all other non-scalable designs. In this scalable hardware, the computation time is relate to the actual number of bits and do not depend on the hardware capability of $n_{max}=160$ bits.

Fig. 3 shows that the computation time of all new designs are less than the old ones in all cases. However, as the value *w* goes large compared to the actual number of bits *n*, the computation time increase fast, which is a situation that loses the speed benefit of scalability. In other words, as *w* gets bigger the total time decreases fast, which is true in all different scalable designs as long as $n \geq w$.

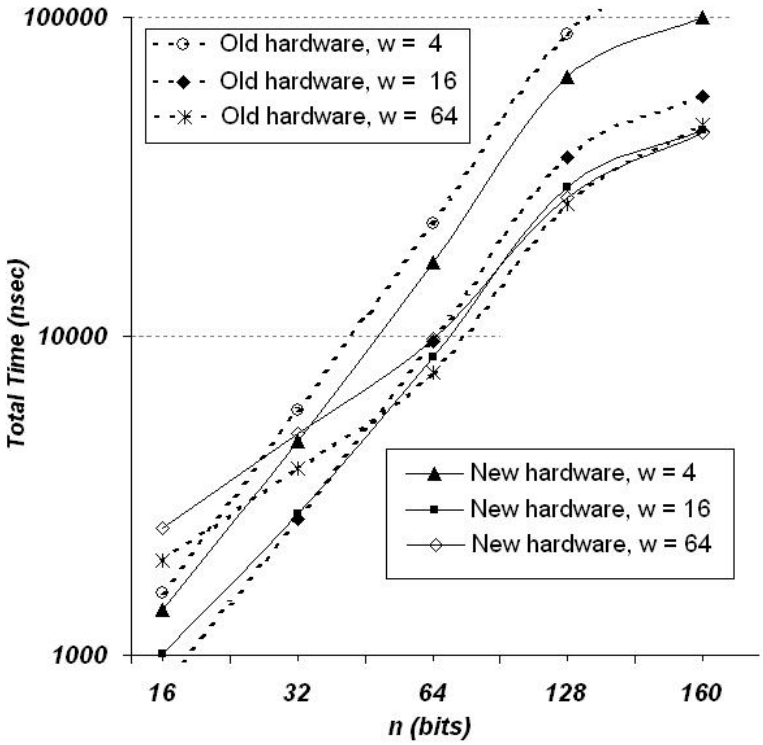

Figure 3: Delay comparison of all scalable designs for 160-bits

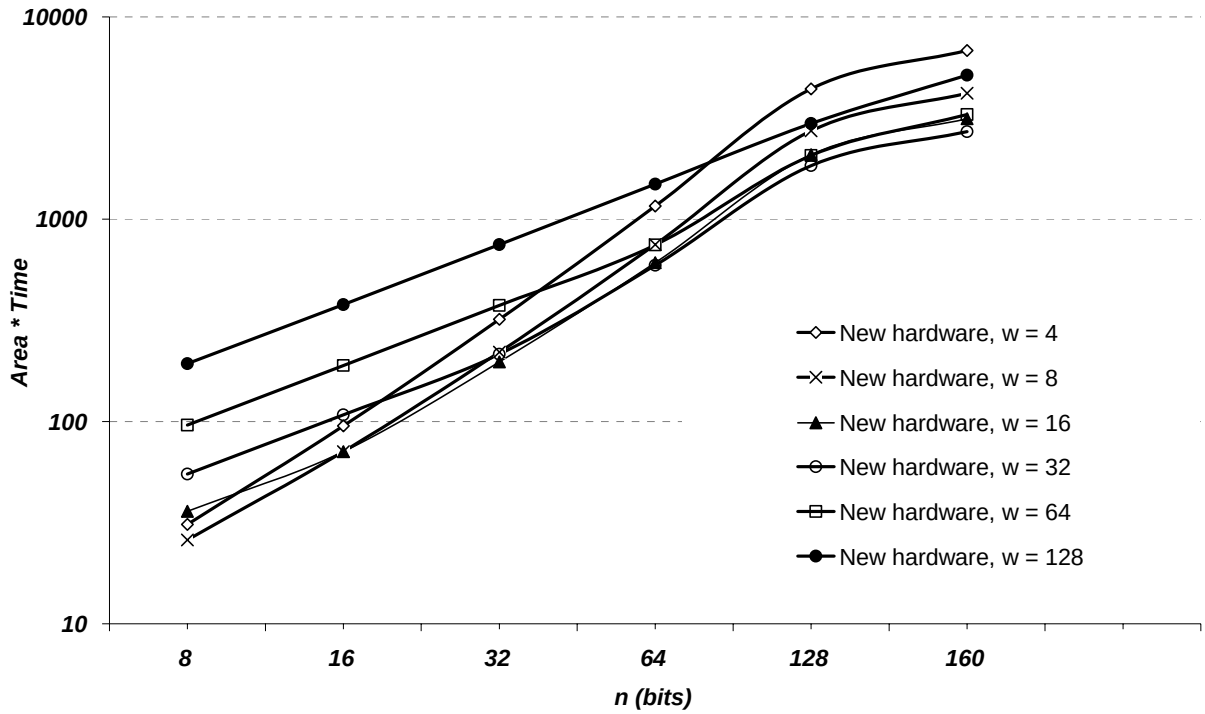

Figure 4: AT figure of merit of different new hardware designs

5. Best Scalable Hardware

Choosing the appropriate scalable design is depending on the importance of speed and area. Consider the area study, Fig. 2, and the delay one, Fig. 3, as we increase in terms of area we gain in most of the cases in speed. However, is the speed gained worth the area paid?

To estimate an evaluation standard that relates between area and time, a *figure of merit:* AT (Area×Time) is used to decide the best design. It is assumed that as AT value reduces as the design is better.

Fig. 4 shows the AT results of the scalable designs with respect to the actual number of bits *n* for the new scalable 160-bits architectures. The AT values show that depending on the actual number of bits *n* expected to be used, the best hardware is chosen. For example, If *n* is too small, less than 16 bits (which is unpractical), the best design would be with $w = 8$ bits. When n is within the range from 16 bits to 64 bits, the appropriate design is the one with $w = 16$ bits. If the value of *n* is grater than 64 bits, the suitable hardware would be with $w = 32$ bits, which is the practical hardware assumption to choose. Note that the bigger designs, i.e. with $w > 32$ bits are found inappropriate according to AT estimate.

6. Conclusion

This paper presents a modified scalable hardware for GF(p) Montgomery modular inverse computation to gain speed. The design is featuring scalability allowing a specific computing module to handle operands of any precision, where its delay depends on the actual data used and not on the hardware capability. The word-size that the scalable module operates can be selected depending on the area and speed requirements. This study found that the best 160-bits hardware to choose is having the scalable word size of 32 bits which has the area of around 30 k-gates.

The proposed new hardware is a modification to an original old hardware that performs shifting operations within the computing unit. This shifting is moved from the scalable computing unit to the non-scalable memory part. The new hardware increased the area to double the area of the original old one but gaining interesting speedup that can reach 28%. The

results show that our scalable structure is very attractive for crypto systems, particularly for ECC where there is a clear need for modular inversion of large numbers, which may differ in size depending on security requirements imposed by applications.

Acknowledgments

I would like to thank the British council in Saudi Arabia, for supporting this research through their postdoctoral program. Continuous support from KFUPM is also appreciated. I would like also to thank the Electrical & Computer Engineering Department of Brunel University in Uxbridge, for hosting me during my visit to the UK and for all fruitful discussions and providing the facilities needed to accomplish this work.

References

[1] Rivest, Shamir, and Adleman, "A Method for Obtaining Digital Signature and Public-Key Cryptosystems", *Comm. ACM,* February 1978, Vol. 21, No. 2, pp. 120-126.

[2] Diffie, and Hellman "New Directions on Cryptography", *IEEE Transactions on Information Theory*, November 1976, Vol. 22, pp. 644-654.

[3] National Institute for Standards and Technology, "Digital signature standard (DSS)", *Federal Register*, August 1991, Vol. 56, pp. 169.

[4] Koblitz, N., "Elliptic Curve Cryptosystems", *Math. Computing*, 1987, Vol. 48, pp. 203–209.

[5] Miller, V., "Use of Elliptic Curves in Cryptography", *Proceedings of Advances in Cryptology (Crypto)*, 1986, pp. 417-426.

[6] Blake, Seroussi, and Smart, "Elliptic Curves in Cryptography", *Cambridge University Press*: New York, 1999.

[7] Ors, Batina, Preneel, and Vandewalle, "Hardware Implementation of an Elliptic Curve Processor over GF(p)", *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP'03)*, June 2003, pp. 433 – 443.

[8] Gutub, Adnan, and Tenca, "Efficient Scalable VLSI Architecture for Montgomery Inversion in GF(p)", *Integration, the VLSI Journal*, May 2004, Vol. 37, No. 2, pp. 103-120.

[9] Daly, Marnane and Popovici, "Fast Modular Inversion in the Montgomery Domain on Reconfigurable Logic", *Irish Signals and Systems Conference (ISSC 2003)*, July 2003, pp. 362-367.

[10] McIvor, McLoone and McCanny, "Improved Montgomery Modular Inverse Algorithm", *Electronics Letters*, September 2004, Vol. 40, No. 18, pp. 1110-1111.

[11] Guerric de Dormale, Bulens and Jean-Jacques Quisquater, "An Improved Montgomery Modular Inversion Targeted for Efficient Implementation on FPGA", *International Conference on Field-Programmable Technology - FPT 2004*, pp. 441-444.

[12] Feldhofer, Trathnigg and Schnitzer, "A Self-Timed Arithmetic Unit for Elliptic Curve Cryptography", *Proceedings of the Euromicro Symposium on Digital System Design (DSD'02)*, 2002.

[13] Zhou, Wu, Bai and Chen, "Fast GF(p) Modular Inversion Algorithm Suitable for VLSI Implementation", *Electronics Letters*, July 2002, Vol. 38, No. 14, pp.706-707.

[14] Mentor Graphics Co. ASIC Design Kit. http://www.mentor.com/partners/hep/AsicDesign Kit/dsheet/ami05databook.html