SPEEDING UP A SCALABLE MODULAR INVERSION HARDWARE ARCHETECTURE

Research Final Report of

Dr. Adnan Abdul-Aziz Gutub

Computer Engineering Department

King Fahd University of Petroleum and Minerals

Email: gutub@kfupm.edu.sa

Submitted to complete
the British Council Research Program of summer 2005

in collaboration with

Dr. Tatiana Kalganova
Bio-Inspired Intelligent System (BIIS) research group
Electrical and Computer Engineering Department
Brunel University
Uxbridge, United Kingdom

September 2005

1. Abstract

The modular inversion is a fundamental process in several cryptographic systems. It can be computed in software or hardware, but hardware computation proven to be faster and more secure. This research focused on improving an old scalable inversion hardware architecture proposed in 2004 for finite field GF(p). The architecture has been made of two parts, a computing unit and a memory unit. The memory unit is to hold all the data bits of computation whereas the computing unit performs all the arithmetic operations in word (digit) by word bases known as scalable method.

The main objective of this project was to investigate the cost and benefit of modifying the memory unit to include parallel shifting, which was one of the tasks of the scalable computing unit. The study included remodeling the entire hardware architecture removing the shifter from the scalable computing part embedding it in the memory unit instead. This modification resulted in a speedup to the complete inversion process with an area increase due to the new memory shifting unit. Quantitative measurements of the speed area trade-off have been investigated. The results showed that the extra hardware to be added for this modification compared to the speedup gained, giving the user the complete picture to choose from depending on the application need.

Table of Contents

2. Introduction

Modular inverse arithmetic is an essential arithmetic operation in public-key cryptography. It is used in the Diffie-Hellman key exchange method [5], and it was also adopted to calculate private decryption key in RSA [4]. Modular inversion is a basic operation in the elliptic curve cryptography (ECC) [1,2,9-12,20-25]. This work is targeted mainly toward the use of ECC because of its promise to replace older public-key cryptographic systems [9-12,20]. ECC arithmetic consists mainly in modular computations of addition, subtraction, multiplication, and inversion.

Inversion is well known to be the lowest computation among all other arithmetic calculations in ECC [1,2,11,16-18]. Many researchers propose minimizing the use of modular inversion by adopting elliptic curves defined for projective coordinates [9-12], which substitutes the inverse by several multiplication operations. Inversion, in the projective coordinate systems, is required only once at the end, to convert the projective coordinate points back to affine coordinates. However, if this single inversion is not fast enough, it will cause the complete ECC system to be slow.

A fast modular inverse calculation is the main reason to do inversion in hardware instead of software [16-18]. If it is possible to compute the inverse in less time than nine multiplication operations, then it is more efficient to use the affine coordinate system instead of going to the projective coordinate systems [2,10]. Even if the speed to compute the inverse is not that good to justify the use of affine coordinates, the computation with hardware is still faster than software [6,16-18,20-25], which will provide better performance for the overall cryptographic system based on projective coordinates.

Another main reason to implement the inverse in hardware is security. For cryptographic applications, it is more secure to have all the computations handled in hardware, inside an IC-chip, instead of mixing some computations performed in software with others processed in hardware. Software-based systems can be interrupted and trespassed by intruders much easier than hardware, which can jeopardize the security of the whole application. Moreover, stealing information from software systems is easier than from hardware.

3. Literature Review

Modular inversion is often performed by algorithms based on the Extended Euclidean algorithm [11]. Several inversion VLSI designs are described in the literature [16-18, 20-25, 29-31]. Most of them [17, 18, 20-25] are for inversion in Galois Fields GF(2^k). Several [17, 18, 21-25] are based upon extensive combinational networks. The inversion in GF(2^k) is fast due to the elimination of the carry propagation in GF(2^k) calculations. However, the area used in these parallel organizations are very large, of order O(n^2). Hasan in [20] proposed to implement the GF(2^k) inversion algorithm in a smaller area but with slower speed. His hardware performs word-by-word computation on the operands instead of computing all the words in parallel. Since we focus on GF(p), the designs proposed for GF(2^k) in [17,18, 20-25] have no direct link to this work.

Takagi in [16], proposed an inverse algorithm for hardware with a redundant binary representation. Each number is represented by a digit in the set {0,1,-1}. Redundant representation is used to avoid the carry propagation delay problem. However, the hardware in [16] requires more area than the design proposed here and also needs data transformations that are usually expensive.

Zhou in [30, 31], designed a VLSI implementation for GF(p) inversion computation using one simple adder. Zhou's hardware suffers from the long propagation carry chain which made the operation clock frequency limited and the design area and complexity not flexible to accommodate the changing demand of the crypto applications.

Several attempts [29, 32, 34, 36] have investigated the GF(p) inversion targeted to field programmable gate array (FPGA) implementations. Fiaz in [36] described an FPGA divider which can be used for inversion by representing the dividend by '1'. Daly [29] and Dormale [34] shortened the critical path and carry-chain addition within the inversion process according to the FPGA column limitations. The designs minimize the extra delay of the top to bottom carry chain mapping between different FPGA columns by specific physical routing. Daly presented an architecture technique "for implementation on any FPL (field programmable logic) device which has dedicated carry logic capability" [29]. Dormale presented an FPGA carry conditional adder implementation that demonstrates improvement especially when the carry-chain exceeds the specific FPGA column height [34].

An ECC arithmetic hardware unit has been proposed by Feldhofer in [35]. It contained asynchronous modules to compute all prime field computations including inversion. The inversion hardware was slow and complex depending on Fermat's Theorem. Feldhofer's idea was not to consider the inversion as a main problem, assuming all computations are performed through projective coordinates, and the inversion was calculated by a number of multiplications. The inversion process needed multiplication operations that can be equal to double the number of bits of the data length used.

McIvor in [32] improved the inversion algorithm presented by Savas in [1] involving modular multiplication. McIvor reduced the number of needed multiplication operations to speedup the complete computation [32]. McIvor benefited from the built-in carry-look-ahead adders on the FPGA to gain in area reduction.

Tawalbeh in [37] presented a unified inversion hardware for both GF(p) and GF(2^k). He replaced all comparisons by the use of counters to keep track of the difference between field elements which are usually expensive and time-consuming. It uses a scheduling method to reduce the number of hardware resources without significantly increasing the total execution time.

4. Problem and Objectives

The standard modular inverse over GF(p) can be defined by the following example. Assume a is an integer in the range [1, p-1]. Integer x is called the modular inverse, or modulo inverse, of integer a if-and-only-if: $ax \equiv 1 \pmod p$; where $x \in [1, p\text{-}1]$. It is normally represented as $x = a^{-1} \bmod p$ [1]. The Montgomery modular inverse algorithm and hardware suitable for this research is presented in [27, 28]. 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 two months project aimed to investigate the possibility of speeding the process by modifying the registers of the non-scalable part to incorporate the shifting operation. This way, the shifting operation will be part of the memory unit instead of the scalable computing unit. This feature is predicted to reduce the shifting operation delay which will

improve the total computation performance. Therefore, the main objective has been to investigate the advantages and disadvantages behind this modification and its practicality to be implemented. Quantitative measurements of the trade-off between speed and area, and the modification criteria will be the subject of investigation.

5. Engineering Approach

The scalable inversion hardware is built of two main parts, a memory unit and a computing unit, as shown in Fig. 1. It is very similar, in principle, to the scalable hardware presented in [27]. The memory unit is not scalable because it has a limited storage defined by the value of n_{max}. The data values of a and p are first loaded in the memory unit. Then, the computing unit read/write (modify) the data using a word size of w bits. The computing unit is completely scalable. It is designed to handle w bits every clock cycle. The computing unit does not know the total number of bits, n_{max}, the memory is holding. It computes until the controller indicates that all operands' words were processed. Note that the actual numbers used may be way smaller than n_{max} bits.

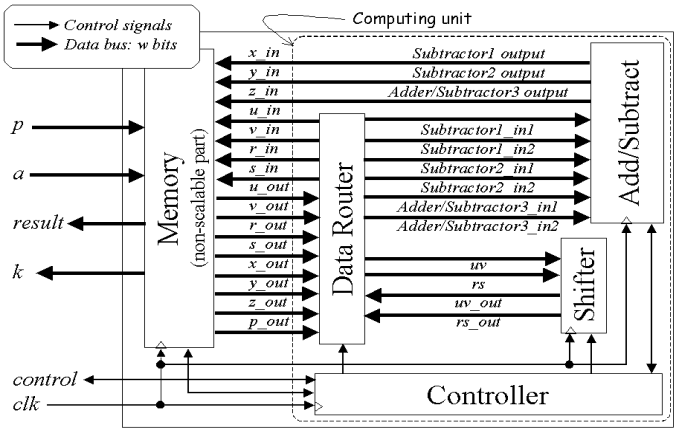

Fig. 1 inversion scalable hardware block diagram

The memory unit contains a counter to compute variable k and eight first-in-first-out (FIFO) registers used to store the inversion algorithm's variables. All registers, u, v, r, s, x, y, z and p, are limited to hold at most n_{max} bits. Each FIFO register has its own reset signal generated by the controller. They have counters to keep track of n (the number of bits actually used by the application).

The computing unit is made of four hardware blocks, the add/subtract, shifter, data router, and controller block. All these blocks functions and hardware design are detailed in [27, 28]. Our focus of this research is about the *shifter*. The original shifter is made of two multiplexers and two registers with special mapping of some data bits, as shown in Fig. 2. The two multiplexers are used to select the correct set to be used in the multi-bit shifter. Depending on the controller signal *Distance*, the shifter acts as a one, two, or three-bit shifter, as clarified in [28]. Two types of shifting are needed in the inversion algorithm, right shifting an operand (*u* or *v*) through the *uv* bus (one, two, or three bits) and left shifting another operand (*r* or *s*) through the *rs* bus (by similar number of bits). Right shifting *u* or *v* is performed through Register1, which is of size *w*-1 bits. For each word, *w-1* bits of *uv* are stored in Register1. The LS bit(s) of each word is (are) read out immediately as the most significant bit(s) of the output bus *uv_out*. Left shifting *r* or *s* is performed via Register2, which is of size *w*+3 bits, in a similar fashion.

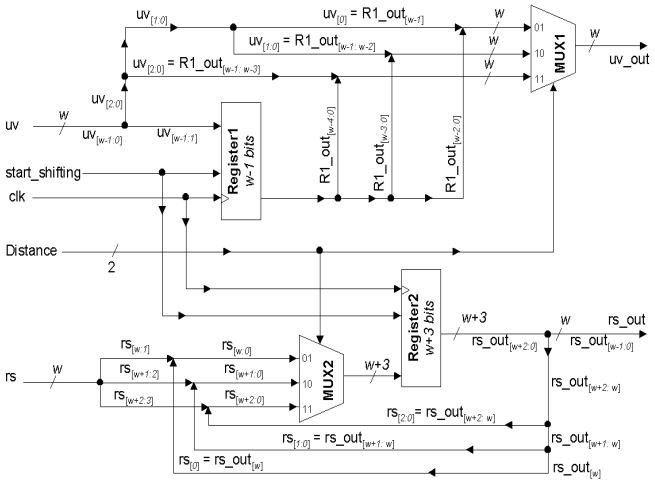

Fig. 2 Multi-bit shifter (max distance = 3)

The study plan is to redesign both the scalable and non-scalable hardware units. The shifter will be removed from the computation unit. It will be embedded into the non-scalable memory unit. The project results are expected to show the speedup gained and the extra hardware area needed. The conclusion will indicate whether the extra hardware is worth the expected speedup with quantitative measurements.

6. Hardware and Algorithms

Several methods considered for hardware computation of the Montgomery inverse are shown in Fig. 3; including the procedures proposed by Savas and Koç in [1] using MonPro (Montgomery Product). Each path in the graph has its own set of routines and its total computation time. Fig. 3 presents the approximate number of iterations for each routine. Note that the number of iterations for multiplication is estimated considering serial-parallel multipliers, because fully parallel multipliers are impractically large [6].

All approaches of Fig. 3 lead to the same final result. However, the number of iterations in each path proves that our two-phase method, the *AlmMonInv* followed by the *correction phase* (the bold path shown in Fig. 3), is the fastest. It requires only *2n* iterations to complete the inversion, the *AlmMonInv* needs *1.5n* iterations, and the correction phase (*CorPh*) needs *0.5n* iterations, assuming an average value of $k=1.5n$, as detailed in [28].

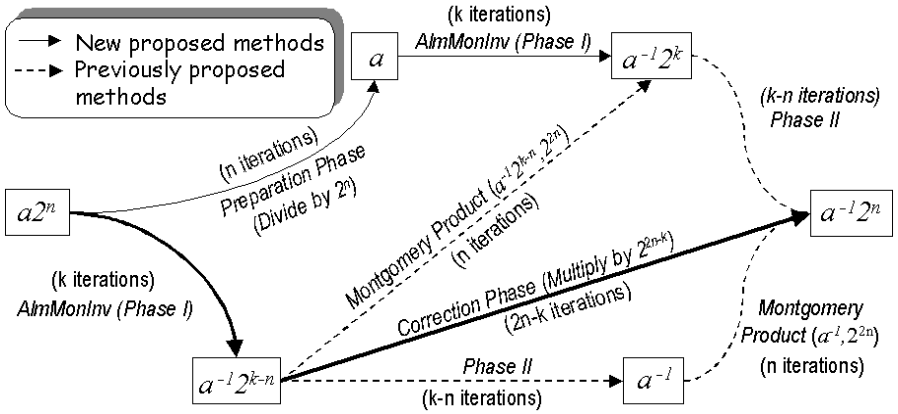

Fig. 3 Different ways to compute the Montgomery inversion

Two hardware algorithms of the AlmMonInv procedure are shown in [28], depending on the number of bits of shifting used. We will start this study by single bit shifting since the shifter will be eliminated from the computing unit. The AlmMonInv algorithm of single-bit shifting is shown below.

AlmMonInv Hardware Algorithm (HW-Alg1)
Registers: u, v, r, s, & p (all five registers hold n bits).
Input: $a \in [1, p-1]$, p = modulus; where $2^{n-1} \le p < 2^n$
Output: result$\in[1, p-1]$ & 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 ≠ 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)* [28] algorithm *(see Fig. 3)* 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 ($r= a^{-1}2^{k-n} mod\ p$)& k from *AlmMonInv*
Output: result; where result = $a^{-1}2^n$ *(mod p)*.
11. $j= 2n-k-1$
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}
16. return result = r

The hardware is modified as shown in Fig. 4. The memory block is improved to perform shifting by adding n_{max} 4x1 multiplexers to each FIFO. The non-scalable block is resized increasing the hardware area by $(8*n_{max}*10)$ *gates.*

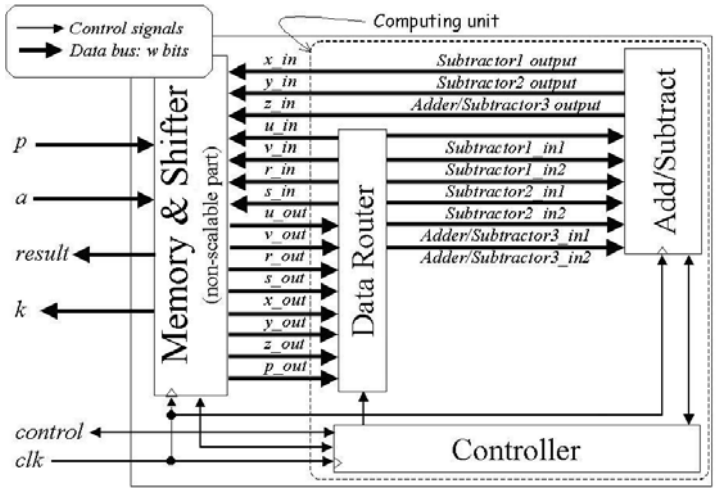

Fig. 4 Improved inversion scalable hardware block diagram

7. Effect on Area and Speed

The study will concentrate on the *AlmMonInv* algorithm first, then the *CorpPh* one, similar to our old scalable design presented in [28]. The new hardware area and speed will be estimated depending on the number of bits to be shifted through the non scalable 'memory & shifter' unit.

7.1 AlmMonInv Single Bit Shifting

The analysis showed that single bit shifting through the memory is performed by simply adding a multiplexer at the input of every memory cell. The multiplexer size used is ten gates making the area increase over the single bit shifting scalable hardware in [28] as follows:

$$Area\ increase = 8*n_{max}*10 = 80\ n_{max}$$

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 [28]. 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 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:

$$Cycles\ for\ steps\ 4,5,6 = 0.5\ k\ (\lceil n/w \rceil + 1)$$

$$Cycles\ for\ step\ 9 = \lceil n/w \rceil$$

$$Cycles\ for\ steps\ 2,3 = 0.5\ k$$

$$Total\ AlmMonInv\ Clock\ Cycles = 0.5\ k\ (\lceil n/w \rceil + 1) + \lceil n/w \rceil + 0.5\ k$$

7.2 AlmMonInv Two Bit Shifting

The new hardware, shown in Figure 4, is improved to have its memory & shift unit to perform two bit shifting in addition to its original ability of single bit shifting. The

shifting can now be performed as one bit shifting right, one bit shifting left, two bits shifting right, and two bits shifting left. This will modify the routing multiplexer inserted between the memory cells increasing the multiplexer size by eight gates more than the single bit shifting presented in 7.1. The new hardware with two bit shifting will increase the area of the original design of [28] by:

$$Area\ increase = 8*n_{max}*(10+4+4) = 144\ n_{max}$$

The AlmMonInv computation time will be similar to the single bit shifting except in steps 2, and 3, which will be reduced by 6% each. The overall time reduction of steps 2, and 3, together is estimated by 12%, as described in detail in [28]. This two bit shifting made the AlmMonInv computation time as follows:

$$Cycles\ for\ steps\ 4,5,6 = 0.5\ k\ (\lceil n/w \rceil + 1)$$

$$Cycles\ for\ step\ 9 = \lceil n/w \rceil$$

$$Cycles\ for\ steps\ 2,3 = 0.38\ k$$

$$Total\ AlmMonInv\ Clock\ Cycles = 0.5\ k\ (\lceil n/w \rceil + 1) + \lceil n/w \rceil + 0.38\ k$$

7.3 AlmMonInv Multi Bit (Three) Shifting

In [28], it was shown that increasing the multi-bit shifting over three-bits is not beneficial. The time reduction probability will be too low compared to three-bit shifting making three bit shifting as the appropriate hardware to build. The hardware area increase of three bit shifting within the memory is estimated as eight gates more within all multiplexers between the memory cells. The overall area modification of this three-bit shifting hardware compared to the original hardware of [28] is:

$$Area\ increase = 8*n_{max}*(10+8+8) = 208\ n_{max}$$

The AlmMonInv computation time is affected similarly to the two-bits shifting (section 7.2) with the difference in time reduction to calculate steps 2, and 3. This three-bit shifting made the AlmMonInv computation time as follows:

$$Cycles\ for\ steps\ 4,5,6 = 0.5\ k\ (\lceil n/w \rceil + 1)$$

$$Cycles\ for\ step\ 9 = \lceil n/w \rceil$$

$$Cycles\ for\ steps\ 2,3 = 0.35\ k$$

$$Total\ AlmMonInv\ Clock\ Cycles = 0.5\ k\ (\lceil n/w \rceil + 1) + \lceil n/w \rceil + 0.35\ k$$

7.4 CorPh Single Bit Shifting

The correction phase algorithm (HW-Alg2) can run on the new hardware with single bit shifting and two bits shifting. It cannot benefit from three bits shifting since it will need an impractical increase in the number of adders of the scalable design as clarified in [28]. The area of the hardware design is not affected when running HW-Alg2 while the computation time is. The computation time of 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:

$$number\ of\ iterations = 2n\text{-}1.5n\text{-}1 \approx 0.5n.$$

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:

$$Cycles\ for\ step\ 13 = 0.5\ n$$

$$Cycles\ for\ step\ 14 = 0.5\ n * \lceil n/w \rceil$$

$$Total\ CorPh\ Clock\ Cycles = 0.5\ n + 0.5\ n * \lceil n/w \rceil$$

7.5 CorPh Multi-Bit Shifting

When two-bits shifting method is involved within HW-Alg2, the average computation time will be hafted. The average number of cycles to compute HW-Alg2 using the new hardware with multi-bit shifting is as follows:

$$\text{Cycles for step 13} = 0.5\ n/2$$

$$\text{Cycles for step 14} = 0.5\ n/2 * \lceil n/w \rceil$$

$$\text{Total } CorPh \text{ Clock Cycles} = (0.5\ n + 0.5\ n * \lceil n/w \rceil)/2$$

The exact computation time is computed by the number of cycles multiplied by the clock cycle period. It was found 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 scalable hardware of [28] as listed in Table 3.

w	**4**	**8**	**16**	**32**	**64**	**128**
Period	12	14	19	28	47	82

Table 1 Clock cycle period for all scalable designs (nsec)

8. Comparisons and Analysis

8.1 Area Comparison

The hardware area of any VLSI architecture depends on the technology and minimum feature size. For technology independence, the number of equivalent gates are used as area measure [14]. Fig. 5, shows the area of the two types of new scalable designs, single and multi-bit shifting, compared to the old scalable designs of [28]. All types of designs are having smooth relation to the maximum number of bits n_{max}. As n_{max} and w increase, all designs areas are getting larger. Observe that as n_{max} is very low, i.e. n_{max} around 128 bits, the multi-bit shifting hardware with small w is smaller than the single bit shifting one with large w. Similarly, for the single bit shifting new hardware compared to the old hardware, as n_{max} is low, the new hardware with small w is smaller than the old hardware with large w.

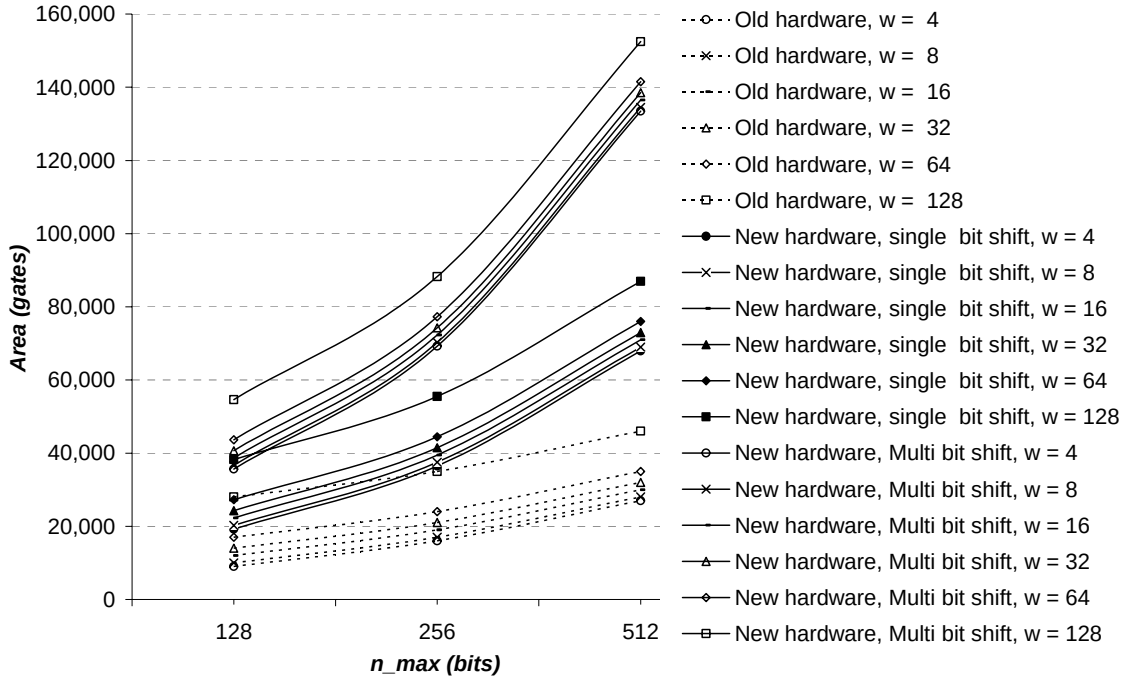

Fig. 5 Area comparison of all scalable designs

The percentage of area increases with relation to w for different scalable designs are shown in Fig. 6. All the percentages shown are for the new hardware designs compared to the old designs of [28]. Observe that the area increase goes low as w gets larger. In fact, the complete option is given to the application and its hardware capability. If area is available, the hardware chosen can be the biggest.

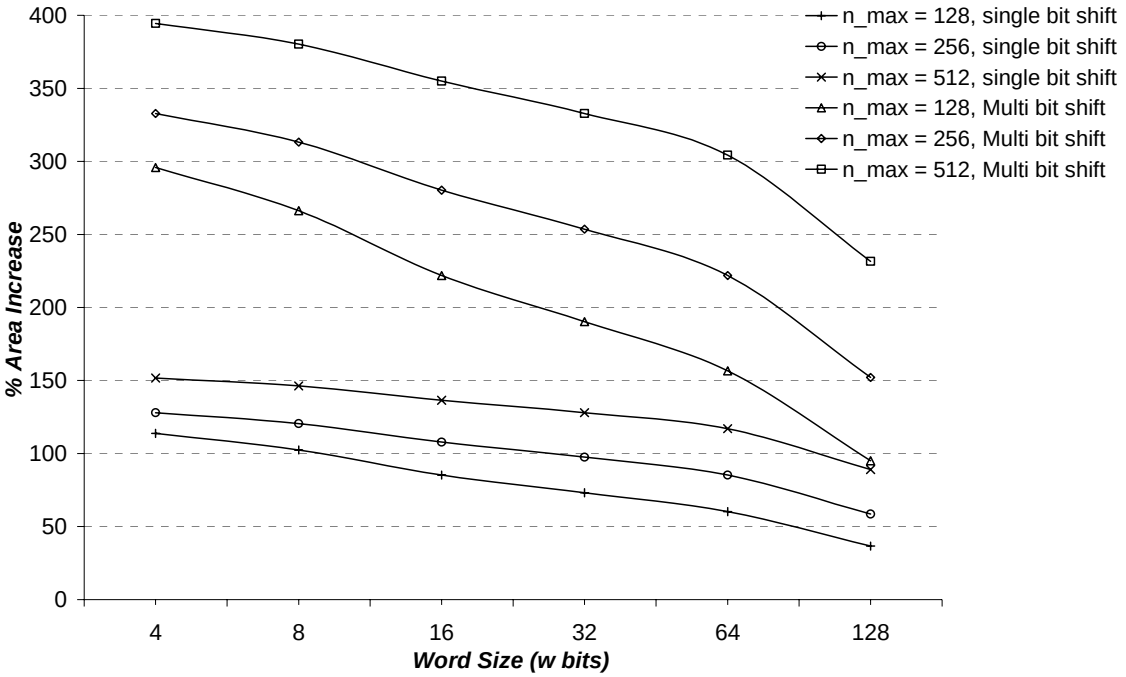

Fig. 6 Percentage of area increase of different scalable designs

8.2 Delay Comparison

Several scalable hardware configurations are designed depending on different n_{max} and w parameters. Each configuration can have different computation time depending on the actual number of bits, n, used. For example, Fig. 7 compares the delay of six scalable hardware designs of all types, the new single bit shifting hardware, the new multi-bit shifting hardware, and the old hardware of [28]. The study assumes all architectures are designed for maximum bits of $n_{max}=512$ bits, which is the practical number for future ECC applications [11]. Note that the difference in the number of bits of the actual data size (n) affects on the number of cycles that changes the speed of the designs. In other words, as n reduces and w is small, the overall computing time of any scalable design reduces. This is a major advantage of the scalable hardware over all other non-scalable

designs where the computation time relate to the actual number of bits and do not depend on the hardware capability number of bits only.

Fig. 7 shows that the computation time of all new designs are less than the old ones in all cases. Similarly, the new hardware with multi-bit shifting is always faster than the single bit shifting hardware. 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 $n \geq w$.

Observe also in Fig. 7, as n increases to the maximum, i.e., $n = n_{max} = 512\text{-}bits$, the fastest hardware is the new multi-bit shift scalable design with $w=128$ bits and $w=64$ bits, which are almost the same speed. This implies that even if you go to a bigger design you are not going to gain in speed anymore. Another interesting observation for the maximum n is that the new multi-bit shift hardware with $w = 16\text{-}bits$ is slower than the single bit shift new hardware with $w \geq 32$ *bits*, which indicates the important of this study and not to go the larger design immediately, assuming the bigger the designs always give higher speed.

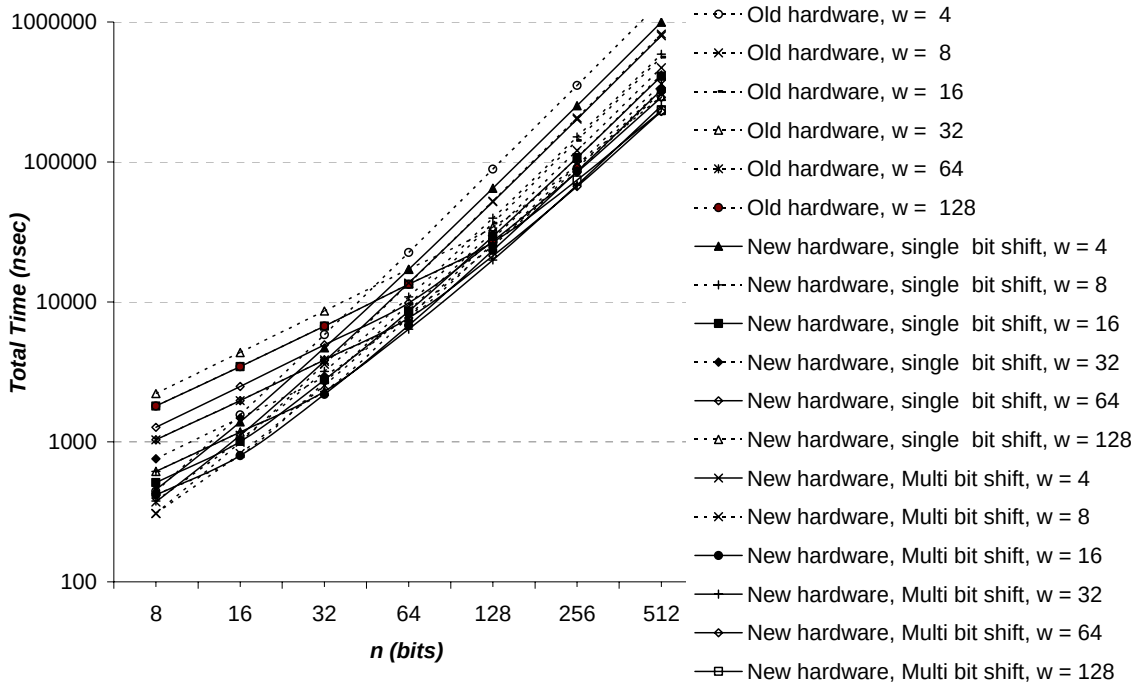

Fig. 7 Total computation time comparison of all scalable designs

The percentage speedup of the new hardware compared to the old one is shown in Fig. 8. The speedup percentage shown is for both types of new hardware designs, i.e. single bit shifting and multi bit shifting architectures. Interestingly, the multi-bit shifting new hardware is having a positive speedup percentage in most of the cases. On the other hand, the new hardware with single bit shifting is having negative speedup when $2n \geq w$. In other words, the single bit shifting new hardware is too slow compared to the old hardware whenever w is larger or near the value of half n.

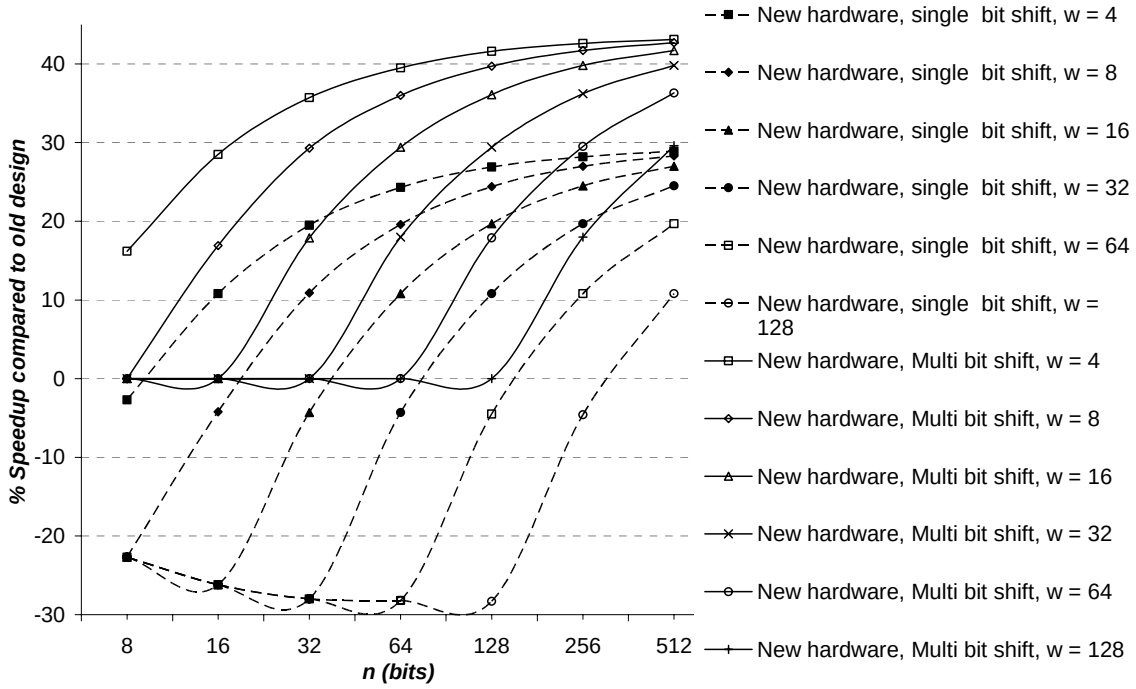

Fig. 8 Percentage of speedup of all scalable designs

The multi bit shifting new hardware is faster than the single bit shifting one. The speedup of these two types of new hardware architectures are shown in Fig. 9. It can be observed that the multi bit shifting design is faster than the single bit shifting one within the range from 18% to 22%. Note that the percentage of speedup depends on the value of n. The n values that give the best speedup percentage for all designs is summarized in Table 2.

w	**4**	**8**	**16**	**32**	**64**	**128**
n	16	32	16-64	32	32-64	32-256

Table 2 New hardware n value for best speedup of multi-bit over single bit shift architectures.

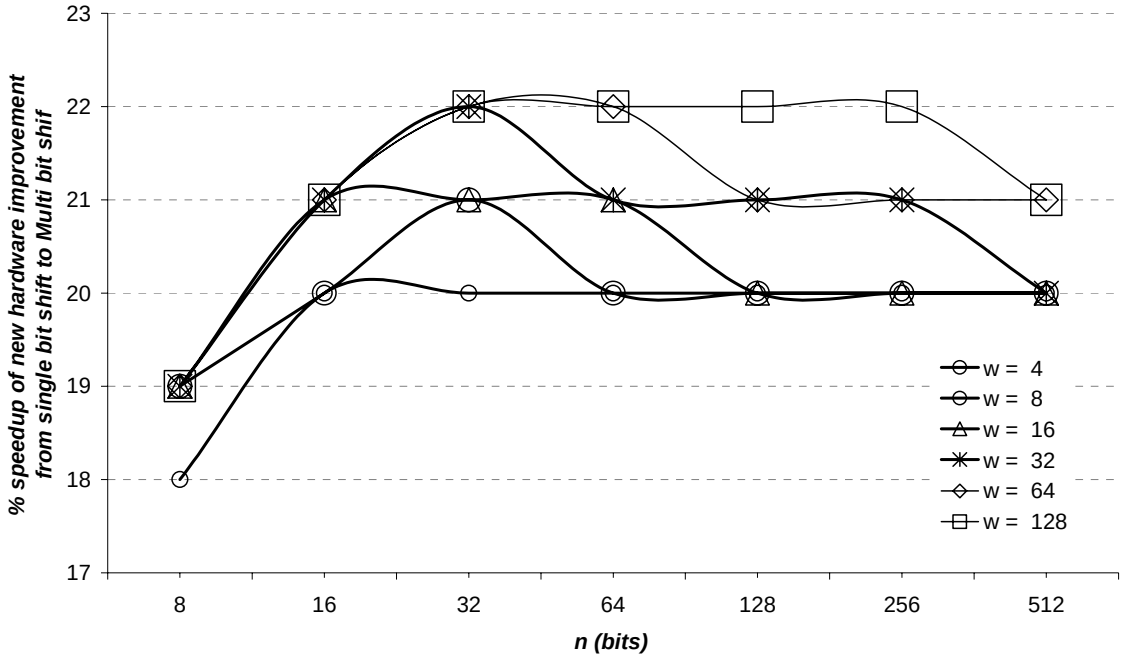

Fig. 9 New hardware speedup improvement from single bit to multi bit shifting.

8.3 Area × Time of the New Hardware

Choosing the appropriate scalable design is depending on the importance of speed and area. In fact, as seen from the area study, Figure 5, and the delay one, Figure 7, 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, two *figure of merit* values are used depending on each factor importance. If area is assumed to have the same importance as time, AT (Area×Time) is used to decide the best design. On the other hand, if the time is the most important factor, AT^2 (Area×Time×Time) is considered. It is assumed that as the figure of merit values reduces as the design is better.

Figures 10 and 11 show the AT results of the scalable designs with respect to the number of bits *n* for single bit shifting and multi bit shifting architectures, respectively.

Both AT figures show that our proposed designs with single bit shifting are giving the best designs at similar *w* values.

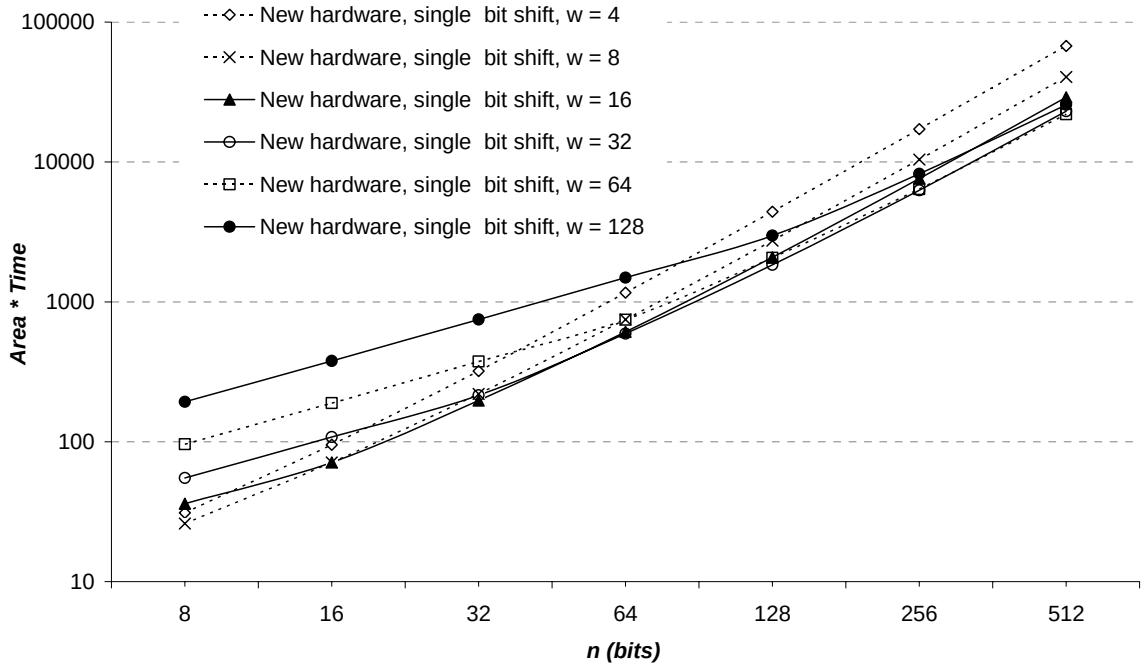

Fig. 10 Area×Time figure of merit of different new hardware single bit shifting architectures

The best AT scalable architectures depends on the application actual number of bits *n*. For example, if the number of bits is impractically low, i.e. *n = 8 bits*, the best design would be with *w = n*. If the actual number of bits: $16 \leq n \leq 64$, the best hardware would be with *w* as the smallest *n* (*w = 16 bits*). The design with *w = 32 bits* is the appropriate for the actual number of bits: $128 \leq n \leq 256$. If *n > 265 bits*, the suitable design would be with *w = 64 bits*. Figures 10 and 11 confirm that there is no need to build scalable designs with $w \geq 128$ *bits*, as long as the hardware time and area are both having the same importance. The AT best architectures *w* values related to *n* are summarized in Table 3.

n	***8***	***16***	***32***	***64***	***128***	***256***	***512***
w	8	16	16	16	32	32	64

Table 3 AT best architectures depending on the actual number of bits (n).

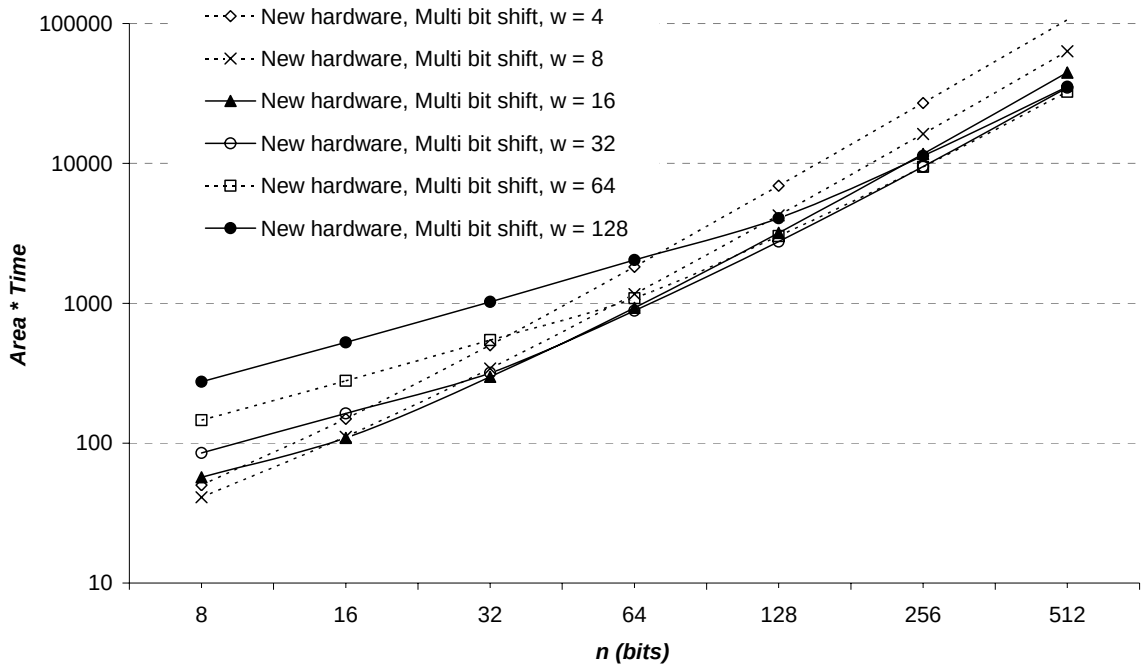

Fig. 11 Area×Time figure of merit of different new hardware multi bit shifting architectures

8.4 Area ×Time^2 of the New Hardware

AT^2 is the appropriate figure of merit to find the right and proper hardware assuming the time is much more important than the area. The best new hardware architecture with single bit shifting can be derived from Figure 12. Depending on the actual number of bits n the appropriate design word size w is chosen.

Recall that all designs are built to handle the maximum number of bits $n_{max} = 512\ bits$. If $n = 8\ bits$, the proper design to be selected is the one with $w = 8\ bits$. When the actual number of bits: $16 \le n \le 32$, the suitable architecture is with $w = 16\ bits$. As the actual number of bits goes practically large, i.e. $n \ge n_{max}/8$ ($n \ge 64\ bits$), the best AT^2 single bit shifting design is always the one with $w = n_{max}/4$ ($w=128\ bits$), as in Table 4.

n	***8***	***16***	***32***	***64***	***128***	***256***	***512***
w	8	16	16	128	128	128	128

Table 4 AT^2 best single bit shifting architectures.

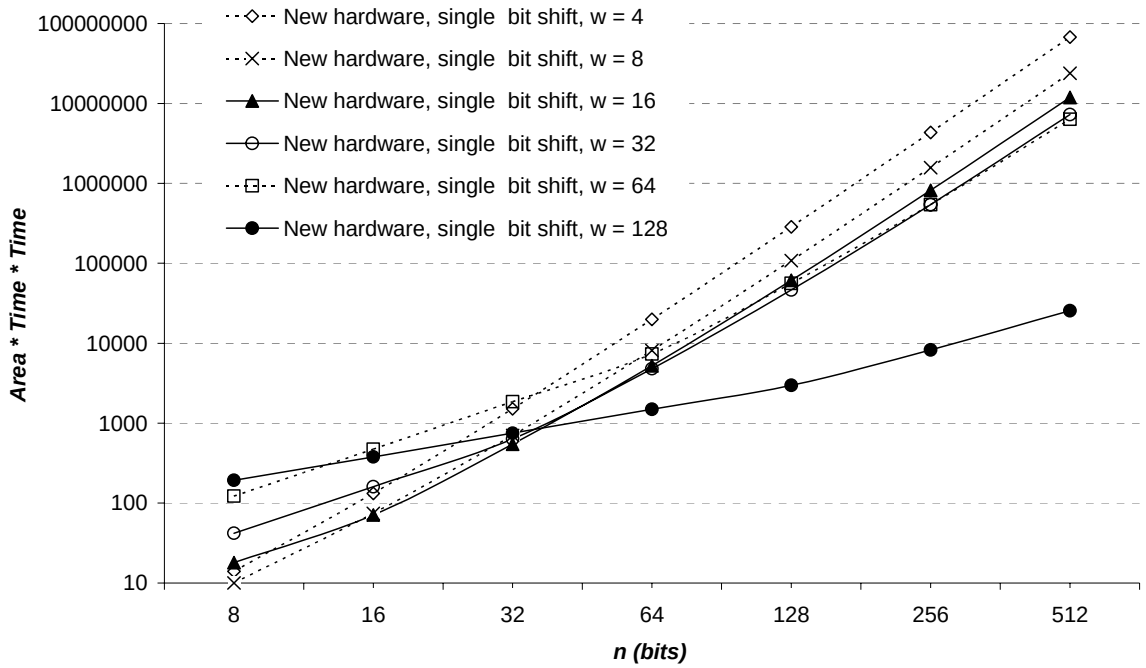

Fig. 12 Area×Time^2 figure of merit of different new hardware single bit shifting architectures

The AT^2 of multi bit shifting architectures are shown in Figures 13. The appropriate multi bit shifting hardware for $n \geq 256$ *bits* is the one with $w = 64$ *bits*. If the actual number of bits: $64 \leq n \leq 128$, the suitable design is with $w = 32$ *bits*. Whenever $16 \leq n \leq 64$, the correct architecture to choose is with $w = 16$ *bits*. For $n = 8$ *bits* the design to be used should be with $w = 8$ *bits*. Note that the biggest hardware to be used is not to exceed $w = 64$ *bits*, according to this AT^2 study. Figure 13 is giving different suitable hardware designs than Figure 12 making a new summary table of best multi bit shifting hardware architectures related to n shown as Table 5.

n	***8***	***16***	***32***	***64***	***128***	***256***	***512***
w	8	16	16	32	32	64	64

Table 5 AT^2 best multi bit shifting architectures.

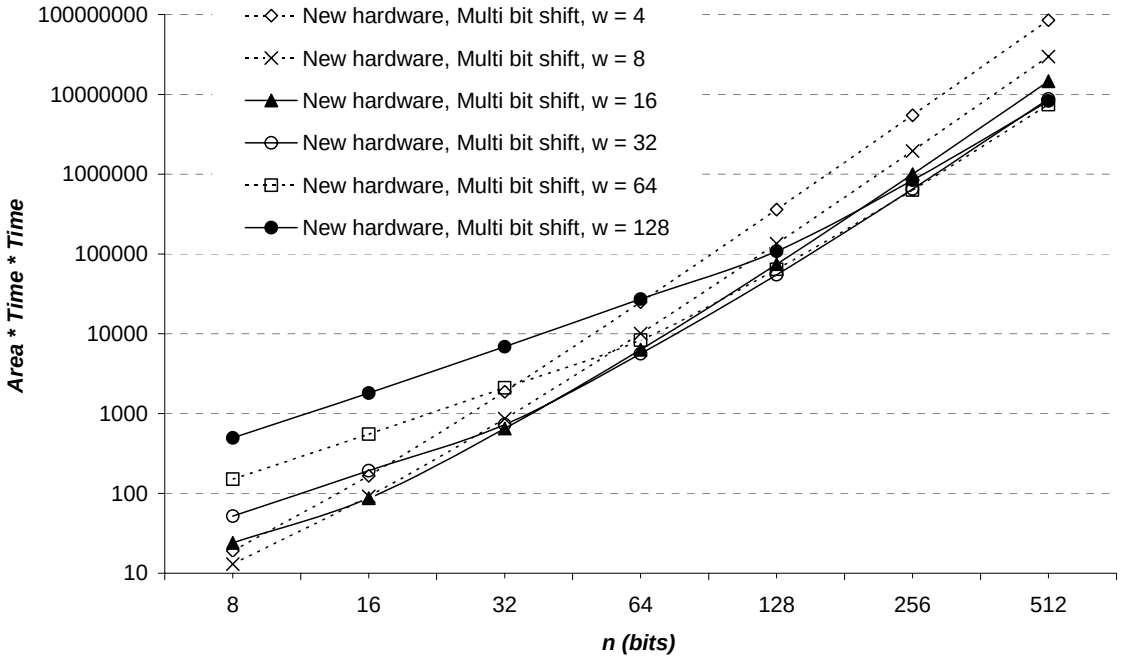

Fig. 13 Area×Time^2 figure of merit of different new hardware multi bit shifting architectures

9. Conclusion

This work modified a scalable VLSI architecture for GF(p) Montgomery modular inverse computation to gain in speed. The architecture is scalable allowing a specific computing module to handle operands of any precision. The word-size that the module operates can be selected depending on the area and performance requirements. The maximum limit (n_{max}) on the operand precision of the entire inverter hardware is limited only by the available memory to store the operands and internal results. If the operand precision exceeds the memory size, the memory unit is the only part that needs to be modified, while the scalable computing unit does not change.

The original old hardware had shifting operation performed within the computing unit. This shifting operation has been moved from the scalable computing unit to the non-scalable memory part. Two shifting strategies have been investigated, single bit shifting and multi bit shifting, which gave different speedup and hardware area results. In general,

the new hardware with single bit shifting was double the area of the original old one gaining the speedup that can reach 28%. The multi bit shifting new hardware increased the original hardware area by a range from two times to four times, depending on the word size of the scalable unit *w*. It gained speedup that can reach 40% depending on the increase of the actual number of bits *n*. On the other hand, as *n* goes low, in both new deigns shifting types, the speedup reduces and may go down to negative values. Negative speedup indicates that the features of the new proposed scalable hardware can be a burden instead of being a benefit, it increased the area and gave lower speed.

Depending on the actual number of bits *n* and the figure of merit AT or AT^2, different designs can be chosen. Table 6 below summarizes all best designs according to the actual number of bits *n* used. All designs are capable to handle up to 512 bits, but the appropriate one is selected depending on the actual number of bits *n* the application is expected to commonly have. The study show that our scalable structure is very attractive for cryptographic 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.

Range of actual number of bits n	***AT Best architecture word w single& multi bit shift***	***AT^2Best architecture word size w***	
		single bit shift	***multi bit shift***
8	8	8	8
16	16	16	16
32	16	16	16
64	16	128	32
128	32	128	32
256	32	128	64
512	64	128	64

Table 6 Best new architectures according to *n*.

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 Dr. Tatiana Kalganova at 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 finish this work.

References

[1] Savas, and Koç, "The Montgomery Modular Inverse – Revisited", *IEEE Trans. on Computers*, 49(7): 763-766, July 2000.

[2] Kobayashi, and Morita, "Fast Modular Inversion Algorithm to Match Any Operation Unit", *IEICE Trans. Fundamentals*, E82-A(5):733-740, May 1999.

[3] Kaliski, "The Montgomery Inverse and its Applications", *IEEE Trans. on Computers*, 44(8):1064-1065, Aug. 1995.

[4] Rivest, Shamir, and Adleman, "A Method for Obtaining Digital Signature and Public-Key Cryptosystems", Comm. ACM, 21(2):120-126, Feb. 1978.

[5] Diffie, and Hellman, "New Directions on Cryptography", *IEEE Trans. on Information Theory*, 22:644-654, Nov. 1976.

[6] Tenca, and Koç, "A Scalable Architecture for Montgomery Multiplication", *In Cryptographic Hardware and Embedded Systems*, no. 1717 in Lecture notes in Computer Science, Springer, Berlin, Germany, 1999.

[7] Savas, Tenca, and Koç, "A Scalable and Unified Multiplier Architecture for Finite Fields GF(p) and GF(2^k)", *In Cryptographic Hardware and Embedded Systems*, Lecture notes in Computer Science. Springer, Berlin, Germany, 2000.

[8] Tenca, Todorov, and Koç, "High-Radix Design of a Scalable Modular Multiplier", *Workshop on Cryptographic Hardware and Embedded Systems, CHES 2001*, Paris, France, May 14-16 2001.

[9] Chung, Sim, and Lee, "Fast Implementation of Elliptic Curve Defined over GF(p^m) on CalmRISC with MAC2424 Coprocessor", *Workshop on Cryptographic Hardware and Embedded Systems, CHES 2000*, Massachusetts, Aug. 2000.

[10] Atsuko Miyaji, "Elliptic Curves over F_P Suitable for Cryptosystems", *Advances in cryptology- AUSCRUPT'92*, Australia, Dec. 1992.

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

[12] Hankerson, Hernandez, and Menezes, "Software Implementation of Elliptic Curve Cryptography Over Binary Fields", *Workshop on Cryptographic Hardware and Embedded Systems, CHES 2000*, Massachusetts, Aug. 2000.

[13] Tocci, R. J. and Widmer, N. S., "*Digital Systems: Principles and Applications*", Eighth Edition, Prentice-Hall Inc., New Jersey, 2001.

[14] Ercegovac, M. D., Lang, T., and Moreno, J. H., *Introduction to Digital System*, John Wiley & Sons, Inc., New York, 1999.
[15] Montgomery, P.L., "Modular Multiplication Without Trail Division", *Mathematics of Computation*, 44(170): 519-521, April 1985.
[16] Naofumi Takagi, "Modular Inversion Hardware with a Redundant Binary Representation", *IEICE Transactions on Information and Systems*, E76-D(8): 863-869, Aug. 1993.
[17] Guo, J.-H., and Wang, C.-L., "Hardware-Efficient Systolic Architecture for Inversion and Division in GF(2^m)", *IEE Proceedings: Computers and Digital Techniques*, 145(4): 272-278, July 1998.
[18] Choudhury, P. Pal., and Barua, R., "Cellular Automata Based VLSI Architecture for Computing Multiplication and Inverses in GF(2^m)", *Proceedings of the 7th IEEE International Conference on VLSI Design*, Calcutta, India, January 5-8 1994.
[19] http://www.mentor.com/partners/hep/AsicDesignKit/dsheet/ami05databook.html, Mentor Graphics Co., *ASIC Design Kit*.
[20] Hasan, M. A., "Efficient Computation of Multiplicative Inverse for Cryptographic Applications", *Proceeding of the 15th IEEE Symposium on Computer Arithmetic*, Vail, Colorado, June 11-13 2001.
[21] Guo, J.-H., and Wang, C.-L., "Systolic Array Implementation of Euclid's Algorithm for Inversion and Division in GF(2^m)", *IEEE Trans. on Computers*, 47(10): 1161-1167, Oct. 1998.
[22] Fenn, S. T. J., Benaissa, M., and Taylor, D., "GF(2^m) Multiplication and Division Over the Dual Basis", *IEEE Trans. on Computers*, 45(3): 319-327, March 1996.
[23] Wang, C. C., Truong, T. K., Shao, H. M., Deutsch, L. J., Omura, J. K., and Reed, I. S., "VLSI Architectures for Computing Multiplications and Inverses in GF(2^m)", *IEEE Trans. on Computers*, C-34(8): 709-717, Aug. 1985.
[24] Feng, G.-L., "A VLSI Architecture for Fast Inversion in GF(2^m)", *IEEE Trans. on Computers*, 38(10):1383-1386, Oct. 1989.
[25] Kovac, M., Ranganathan, N. and Varanasi M., "SIGMA: A VLSI Systolic Array Implementation of Galois Field GF(2^m) Based Multiplication and Division Algorithm", *IEEE Trans. on VLSI*, 1(1):22-30, March 1993.
[26] Charles J. Stone, *A course in probability and statistics*, Duxbury Press, Belmont, 1996.
[27] Adnan Gutub, A. F. Tenca, and C. K. Koç, "Scalable VLSI Architecture for GF(p) Montgomery Modular Inverse Computation", *ISVLSI 2002: IEEE Computer Society Annual Symposium On VLSI*, Pittsburgh, Pennsylvania, April 25-26 2002.
[28] Adnan Gutub and Alexandre F. Tenca, "Efficient Scalable VLSI Architecture for Montgomery Inversion in GF(p)", *Integration, the VLSI Journal*, 37(2):103-120, May 2004.
[29] Alan Daly, William Marnane and Emanuel Popovici, "Fast Modular Inversion in the Montgomery Domain on Reconfigurable Logic", *Irish Signals and Systems Conference (ISSC 2003)*, pages 362-367, Limerick, July 2003.
[30] Tao Zhou, Xingjun Wu, Guoqiang Bai, and Hongyi Chen, "New Algorithm and Fast VLSI Implementation for Modular Inversion in Galois Field GF(p)", *IEEE International Conference on Communications, Circuits and Systems*, 2:1491-1495, 29 June -1 July 2002.

[31] Tao Zhou, Xingjun Wu, Guoqiang Bai, and Hongyi Chen, "Fast GF(p) Modular Inversion Algorithm Suitable for VLSI Implementation", *Electronics Letters*, 38(14):706-707, July 2002.

[32] C. McIvor, M. McLoone and J.V. McCanny, "Improved Montgomery Modular Inverse Algorithm", *Electronics Letters*, 40(18):1110-1111, September 2004.

[33] Adnan Abdul-Aziz Gutub, *New Hardware Algorithms and Designs for Montgomery Modular Inverse Computation in Galois Fields GF(p) and GF(2^n)*. Ph.D. Thesis, Electrical and Computer Engineering Department, Oregon State University, 2002.

[34] Guerric Meurice de Dormale, Philippe 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, December 2004.

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

[36] Fareena Fiaz and Shahid Masud, "Design and Implementation of a Hardware Divider in Finite Field", *National Conference on Emerging Technologies*, 2004.

[37] Lo'ai A. Tawalbeh, Alexandre F. Tenca, Song Park and Cetin K. Koc, "A Dual-field Modular Division Algorithm and Architecture for Application Specific Hardware", Thirty-Eighth Asilomar Conference on Signals, Systems and Computers, Page(s):483 – 487, Vol.1,7-10 Nov 2004.

Researcher Biography

Dr. Adnan Abdul-Aziz Gutub, Visiting researcher, summer 2005

Dr. Adnan Gutub is a Faculty Member in the Computer Engineering Department at King Fahd University of Petroleum and Minerals in Saudi Arabia. He received his Ph.D. degree in June 2002 from the Department of Electrical and Computer Engineering at Oregon State University in Cryptographic hardware design under the supervision of Prof. Alexandre Ferreira Tenca.

Adnan received his BSc degree in Electrical Engineering in 1995 and MSc degree in Computer Engineering in 1998 both from King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia.

Dr. Adnan Gutub's research interests are in modeling, simulating, and synthesizing VLSI hardware for computer arithmetic operations. He worked on designing efficient integrated circuits for the Montgomery inverse computation in different finite fields. He has been awarded the visiting internship for 2 months sponsored by British Council at Brunel University to collaborate with Bio-Inspired Intelligent System (BIIS) research group in a project to speed-up a scalable modular inversion hardware architecture.

Further details can be found on http://faculty.kfupm.edu.sa/coe/gutub/