Fast 160-Bits GF (P) Elliptic Curve Crypto Hardware of High-Radix Scalable Multipliers

Adnan Abdul-Aziz Gutub
Computer Engineering Department, King Fahd University of Petroleum & Minerals, SA

Abstract: *In this paper, a fast hardware architecture for elliptic curve cryptography computation in Galois Field, GF (p), is proposed. The architecture is implemented for 160-bits, as its data size to handle. The design adopts projective coordinates to eliminate most of the required GF (p) inversion calculations replacing them with several multiplication operations. The hardware is intended to be scalable, which allows the hardware to compute long precision numbers in a repetitive way. The design involves four parallel scalable multipliers to gain the best speed. This scalable design was implemented in different versions depending on the area and speed. All scalable implementations were compared with an available FPGA design. The proposed scalable hardware showed interesting results in both area and speed. It also showed some area-time flexibility to accommodate the variation needed by different crypto applications.*

Keywords: *Modulo multipliers, elliptic curve cryptography, scalable hardware designs.*

Received September 24, 2005; accepted August 1, 2006

1. Introduction

Public Key Cryptography (PKC) is becoming very important for today's computer applications security. Most of the systems that use PKC for data encryption and digital signature involve RSA [19]. By time, the number of bits (key size) used in RSA is increasing, making the computation process very lengthy and unpractical which motivated for the use of Elliptic Curve Cryptography (ECC) as a promising substitute [20].

ECC has been proposed independently by Koblitz [7] and Miller [10]. ECC is based on the discrete logarithm problem providing equal security to RSA for a far shorter key size. "A typical example of the size in bits of the keys used in different public key systems, with a comparable level of security (against known attacks), is that a 160-bit ECC key is equivalent to RSA with a modulus of 1024-bits" [15]. This advantage of ECC is being recognized in many standards [18]. The Elliptic Curve Digital Signature algorithm is now included in the ISO/IEC 15946 draft standards. Other standards that include elliptic curves as part of their specifications are the IEEE P1363 (http://grouper.ieee.org/groups/1363), the ATM Forum (http://www.atmforum.com/meetings/rich_bios.html), and the internet engineering task force (http://www.ietf.cnri.reston.va.us).

ECC systems can be implemented in software as well as hardware. Hardware is preferred due to its better speed and security [14, 15, 16, 17]. Software on the other hand, provides flexibility in the choice of the key size [6], which will be gained by our hardware using special multipliers named scalable multipliers thatwill be clarified la ter. Hardware processes provide more security. For crypto applications, the security improves when the computations are handled in hardware instead of software. Software based systems can be terminated and/or trespassed by intruders easier than hardware, which risk the entire security of the application [9].

ECC computations' complexity depends on the efficiency, speed of elliptic curve scalar multiplications, and finite field that is defined over. ECC is normally defined in one of Galois Fields GF (p) or GF (2^m) [2]. The focus in this paper is on GF (p) since it is more complex and lengthy than GF (2^m) due to its carry propagation problem [3].

It is well-known that GF (p) ECC involve point adding operations over an elliptic curve which require a division (or inversion) operation. This inversion operation is the most expensive and complex calculation over GF (p) [3]. We avoid most of the inversion computations by a substitution of several multiplications, replacing the elliptic curve points as projective coordinate points similar to the research work presented in [2, 11, 14, 15]. There are several projective coordinate systems candidates. The choice thus far has been based on selecting the system that has the least number of parallel multiplication steps, since multiplication over GF (p) is a common operation and the next most time consuming process - after inversion - in ECC. We choose the projective coordinates system depending on its inherent parallelism to four parallel multipliers as proven in [5].

In this paper, we use Tenca's high-radix scalable GF (p) multiplier proposed in [22]. Scalable multipliers benefit the trade-offs between area and time, compared

to conventional GF (p) multipliers, giving the hardware designer the priority option between area and time as required by the crypto-application. The scalable multiplier calculation is based on Montgomery modular multiplication method [12]. Normal GF (p) multiplication involves division by the modulus. Division, however, is a very expensive operation (more complex than inversion) [4]. Montgomery in [12] proposed an algorithm to perform modular multiplication that replaces the usual complex division with division by two, which is easily performed in the binary representation of numbers. The cost behind using Montgomery's method is paid in some extra computations to represent the numbers into Montgomery domain and vice-versa. Once the numbers are transformed into Montgomery domain, all operations (addition, subtraction, multiplication, and inversion) are performed in this domain. The result is then converted back to the original integer values.

The scalable ECC design, in principal, can be generalized to compute any number of key size bits. However, it is modeled in this work for 160-bits. This number of bits is specified to make the design comparable to another similar hardware implemented on FPGA by Ors [15], with the assumption that 160-bits ECC gives equivalent security to 1024-bits RSA.

The paper is organized as follows. In section 2, some elliptic curve background is presented followed by a simple crypto demonstration of encryption and decryption. Section 2 also outlines the elliptic curve scalar multiplications algorithm giving some details on the elliptic curve operations using projective coordinates. Section 3 provides a description of the proposed ECC hardware architecture with elaboration on the scalable multiplier used. The section derives the formulae to estimate the area and computation time of the ECC architecture. Section 4 briefly introduces an FPGA implementation as another available hardware to compare with in terms of area and computation time (speed). Finally, the conclusion of the paper is presented as Section 5.

2. Elliptic Curves Over GF(P)

2.1. Elliptic Curve Theoretical Background

It will be assumed that the reader is familiar with the arithmetic over elliptic curves. The reader is directed to reference [2] for more details. In brief, the GF (p) elliptic curve arithmetic is the usual mod *p* arithmetic. The elliptic curve equation over GF (p) is:

$$y^2 = x^3 + ax + b$$

where $p > 3$, $4a^3 + 27b^2 \neq 0$, *and* $x, y, a, b \in GF(p)$.

There is also a single element named the *point at infinity* or the *zero point* denoted 'φ'. The point at infinity is computed as the sum of any three points on an elliptic curve that lie on a straight line. If a point on the elliptic curve is to be added to another point on the curve or to itself, some special elliptic curve addition rules are applied as shown below:

$$(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$$
$$x_3 = \lambda^2 - x_1 - x_2$$
$$y_3 = \lambda (x_1 - x_3) - y_1$$

Where λ is calculated as:

$$\lambda = (y_2 - y_1) / (x_2 - x_1); \text{ if } x_1 \neq x_2$$
or
$$\lambda = (3 (x_1)^2 + a) / (2y_1); \text{ if } x_1 = x_2 \text{ and } x_1 \neq 0$$

Notes that if $x_1 = x_2$ then $y_1 = y_2$ and the elliptic curve addition operation is known as point doubling [2].

Considering the squaring of a number as multiplication, to add two different elliptic points in GF (p) the required operations are: Six additions, one inversion, and three multiplication computations. To double a point the needed operations are: Four additions, one inversion, and four multiplication computations. Because the inversion operation is too lengthy, as introduced earlier, the normal (*x, y*) affine coordinate is converted to projection coordinates (X, Y, Z) as will be discussed later in section 2.4.

2.2. Encryption and Decryption

Several ways can use elliptic curves for encryption and decryption [2] where one method is given here as an example. Users randomly chose a *base point* $G = (x, y)$, lying on the elliptic curve *E*. The plain text (the original message to be encrypted) is coded into an elliptic curve point $P_m = (x_m, y_m)$. Each user selects a secret key '*s*' and generates his public key $P = sG$. For example, user *A*'s private key is s_A and his public key is $P_A = s_A G$.

For any one to encrypt and send the message point P_m to user *A*, the sender chooses a random integer *r* and generate the ciphertext:

$$C_m = \{rG, P_m + kP_A\}$$

The ciphertext pair of points uses *A*'s public key, where only user *A* can decrypt the plaintext using his private key.

To decrypt the ciphertext C_m, the first point in the pair of C_m, rG, is multiplied by *A*'s private key to get the point $s_A(rG)$. Then this point is subtracted from the second point of C_m, the result will be the plaintext point P_m. The complete decryption operations are:

$$(P_m + rP_A) - s_A (rG) = P_m + r (s_A G) - s_A (rG) = P_m$$

The most time consuming operation in the ECC encryption and decryption procedure is finding the multiples of the base point, *G* (the elliptic curve scalar multiplications). The algorithm used to implement this is discussed in the next subsection.

2.3. Scalar Multiplication Algorithm

The ECC scalar multiplication algorithm used for calculating the multiples of an elliptic point, can be expressed by finding *nP* from *P*. This operation is based on a binary scalar multiplication method, known to be efficient and practical to implement in hardware [2, 3, 6, 20, 21]. This binary method algorithm is shown below:

Binary Algorithm

Define k: Number of bits in n; and n_i : i^{th} bit of n
Input: P (a point on the elliptic curve)
Output: Q = nP (another point on the elliptic curve)

1. *If n_{k-1} = 1, then Q := P else Q := 0;*
2. *For i = k - 2 down to 0;*
3. *Q := Q + Q;*
4. *If n_i = 1 then Q := Q +P ;*
5. *End for*
6. *Return Q;*

Basically, the binary algorithm scans the bits of *n* and doubles the point *Q* *k*-times. Whenever, a particular bit of *n* is found to be one, an extra operation of point addition (*Q* + *P*) is needed. Every point addition or point doubling requires the three modulo operations of multiplication, inversion, and addition/subtraction as clarified earlier in Section 2.1.

2.4. Projective Coordinates

The projective coordinates are used to eliminate the need for performing the lengthy inversion similar to the crypto processor idea presented in [2, 11, 14, 15]. For elliptic curve defined over *GF (p)*, two different forms of formulas are found [2, 11] for point addition and doubling. One form projects *(x, y) = (X / Z^2, Y / Z^3)* [2], while the second projects *(x, y) = (X / Z, Y / Z)* [11]. Both projection methods were visualized and studied in [5]. The dependency within all formulae showed that both projective coordinate forms can be parallelized to the maximum possibility when using four multipliers, but with different critical path stages (different number of multiplication cycles steps). The results in [5] showed that projective coordinate *(x, y) = (X / Z, Y / Z)* is faster than *(x, y) = (X / Z^2, Y /Z^3)* with the same hardware. The parallel data flow graph of the projection *(x, y) = (X / Z, Y / Z)* that is suitable for our design is shown in Figures 1 and 2, for elliptic curve point addition and doubling, respectively.

The number of computations of point additions and point doubling depend on the binary value of *n* and its number of bits *k* (see the binary algorithm in section 2.3). In fact, the number of point doubling is always equal to *k*, while the point additions depend on the number of bits that are one. Using the average assumption that half the bits of *n* are ones, the number of point additions is half the number of bits, *k/2*. Projection *(x, y) = (X / Z^2, Y / Z^3)* has on the average *6.5k* multiplication cycles, whereas the *(x, y) = (X / Z, Y / Z)* has on the average *5k* multiplications [5].

Obviously, projection *(x, y) = (X / Z, Y / Z)* would be the projection of choice for our implementation. Remark a further benefit to implement the projective coordinate *(x, y) = (X / Z,Y / Z)* is the 100% utilization of the four multipliers in all multiplication cycles, as seen in Figures 1 and 2, which is not the case of the projection *(x, y) = (X / Z^2,Y / Z^3)* [5].

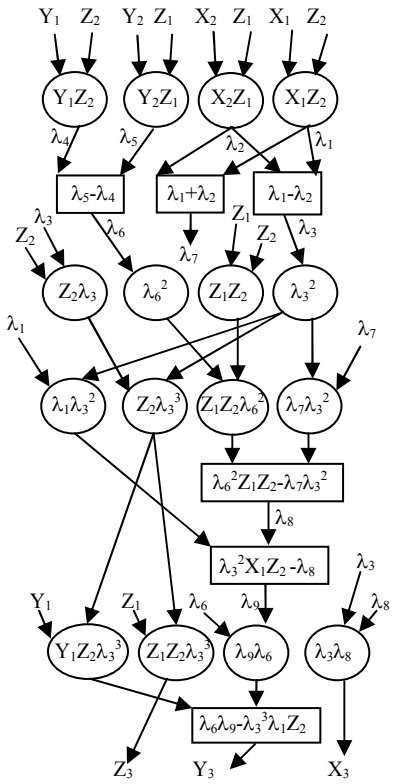

Figure 1. Projecting *(x, y)* to *(X / Z, Y / Z)* adding two points dataflow.

3. Modelling the Proposed Architecture

Many interesting crypto architectures have been proposed in the literature, such as [14, 15]. The usual method in these designs is to adopt serial computations at both the algorithmic level by using a single multiplier, as well as at the arithmetic level by using a serial multiplier. The reason behind serial multiplier and sequential operation is the thought that they provide the lowest area for large word lengths as needed for secure cryptography (i. e., 160 bits [2]). This classical approach shows the way to the reduction of area but with very slow speed that is moreover fixed. The new architecture proposed in this paper has four parallel multipliers, an adder/subtractor, registers and a controller, as shown in Figure 3. The design is straight implemented as the dependency graphs shown in Figures 1 and 2. Its controller is constructed of a state machine to direct the flow of data to conduct the

required projective point operation depending on the binary algorithm (described previously in section 2.3).

The improvement in our crypto-architecture, other than the multipliers architectural parallelism (seen in Figures 1 and 2), is in the basic GF (p) multiplier. The designs proposed in [14, 16] use multiplier hardware that is limited by the number of bits they are meant to be for, if the number of bits are needed to be increased for any application reason, the complete hardware is to be replaced. Furthermore, if the number of bits is much less than the intention of the VLSI design, the unneeded bits will be considered as zeros and they will be included in the computation casing the same delay exactly as if all bits are essential. These weaknesses made-up our choice of adopting special *scalable multipliers* instead of conventional ones.

3.1. Scalable Multipliers

An arithmetic unit is called scalable if it can be reused or replicated in order to generate long precision results independently of the data path precision for which the unit was originally designed. To speed up the multiplication operation, various dedicated multiplier modules were developed. These designs operate over fixed finite fields. For example, the multiplier designed for 155-bits [1] cannot be used for any other field of higher degree. When a need for multiplication of larger precision appears, a new multiplier must be designed.

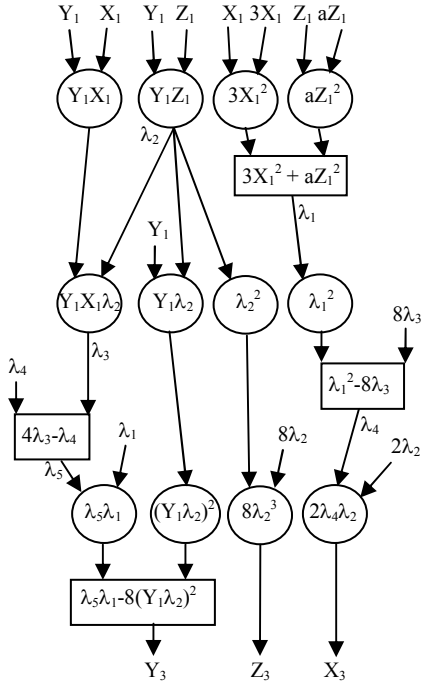

Figure 2. Projecting *(x, y)* to *(X / Z, Y / Z)* doubling a point dataflow.

Another way to avoid redesigning the module is to use software implementations and fixed precision multipliers. However, software implementations (other than their security problem) are inefficient in utilizing inherent concurrency of the multiplication because of the inconvenient pipeline structure of the microprocessors being used. Furthermore, software implementations on fixed digit multipliers are more complex and require excessive effort in coding.

Therefore, a scalable hardware module specifically tailored to take advantage of the concurrency of the multiplication algorithm becomes extremely attractive [22]. Also computation of elliptic point doubling, addition and the algorithm of computing multiples of the base point is such that the multiplication of one stage must be completed before starting the multiplication of the subsequent stage. Therefore, pipelining the digits to further stages is not applicable, even if fast digit serial multipliers are used, the throughput of such multipliers can not be exploited since the next multiplication operation can not begin until the multiplication operations in the previous stage has fully completed.

Another benefit of this scalable multiplier is the flexibility in its hardware modeling stage. It provides an area range that provides the capability to fit in very limited hardware areas such as smart cards [13], of course, with the price compensated from the number of clock cycles to complete the ECC computation. This trade-off between area and speed can be achieved by reducing the *bits per word* size and/or the *number of stages*that makes the scalable multipliers. The reader can go through paper [22] for more information on the scalable multiplier used and its details.

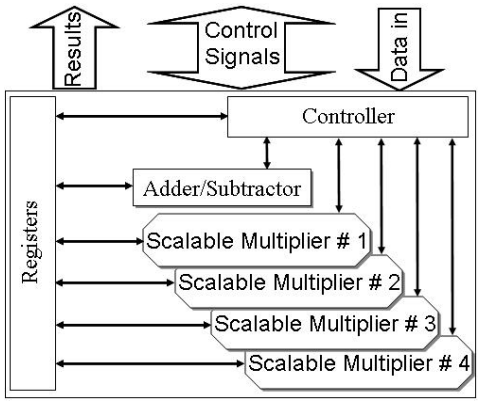

Figure 3. Proposed 160-bits scalable architecture.

3.2. Proposed Hardware Area

The exact area of any design depends on the technology and minimum feature size. For technology independence, we use the number of gates as an equivalent area measure [4].

A modeling package from Mentor Graphic, Leonardo, takes the VHDL design code to generate the hardware gate count (area) and longest path delay (clock period) [22]. The target technology was set to AMI0.5 slow (*0.5μm* CMOS) provided in the ASIC Design Kit (ADK) from the same company [8]. Note that the ADK is generated for academic reasons and cannot be thoroughly compared to technologies developed for commercial ASICs. However, it

provides contrast method to study the different hardware designs.

The scalable multiplier is the unit to make the difference between our proposed design and any other. Thus the scalable multiplier is considered the main factor in calculating the speed and area. The high-radix scalable multiplier area depends on the *number of stages* (*NS*) and the *bits per word* size (*BPW*). Varying *NS* and *BPW* provide different scalable designs with different areas [22], the area of any scalable multiplier can be approximated as:

$$Area_{Scalable\text{-}Multiplier} \approx 92 * BPW * NS + 269 * NS - 9.42 * BPW - 35.5$$

This area of scalable multiplier is multiplied by four and summed to the areas of the adder/subtractor, the controller and the registers, to compute the total hardware area as shown below:

$$Hardware\ Area = 4 * Area_{Scalable\text{-}Multiplier} + Area_{Adder} + Area_{controller} + Area_{Registers}$$

The areas of the adder / subtractor and registers depend on the design maximum number of bits used, $n_{max} = 160 - bits$.

3.3. Proposed Hardware Computation Time

The total computation time is the product of three terms: The average number of multiplication steps, the number of clock cycles each multiplication takes, and the clock period of the VLSI hardware. The number of clock cycles each multiplication takes depends on the relation between two factors, the *number of words* (*NW*) and the number of stages (*NS)*. The high-radix scalable multiplier cycles is estimated in [22] as:

$$Number\text{-}of\text{-}Clock\text{-}Cycles\ per\ Multiplication\ (NCC) = \lceil n_{max} / (3 * NS) \rceil * (2 * NS + 1) + NW + 1,\ If\ NW \leq 2 * NS\ or\ \lceil n_{max} / (3 * NS) \rceil * (NW + 1) + 2 * NS,\ If\ NW > 2 * NS$$

The clock period (t_p) generated by the CAD tool (Leonardo) in [22] changes according to *NS* and *BPW* as listed in Table 1. Table 1 does not show t_p for *NS* between 9 and 15 since they show insignificant difference unimportant to report.

The average number of *multiplication steps* (*MS*) in our proposed design is estimated as:

$$MS = 5k = 5 * n_{maz}$$

As clarified earlier in section 2.4. This makes the total hardware computation time formulated as:

$$Total\ Time = NCC * t_p * MS$$

3.4. Scalable Hardware Area-Time Tradeoff

Depending on the importance of speed and area, the design considered necessary is chosen. In fact, as we pay in terms of area, we generally gain in speed. But is the speed gained worth the area paid. To estimate an evaluation standard that relates between area and time two *figure of merit* estimations are developed, namely, AT (Area × Time), and AT^2 (Area × Time^2). The aim is to have the design with low AT or AT^2. The value of AT assumes area and speed have the same priority weight, while AT^2 gives speed the most priority. Figure 4, shows the AT values of several scalable hardware designs for 160-Bits data size. It is clear that the cost increase as the design *NS* increase, which indicates that choosing the smallest hardware with one stage (*NS = 1*) is the best for all sizes of words. However, surprisingly observed, that the best design (cost-point-of-view) is with the word size of sixteen (*BPW = 16*) and not smaller.

Table 1. High-radix scalable multiplier clock cycle periods (nanoseconds).

NS	Bit Per Word (BPW)				
	8	**16**	**32**	**64**	**128**
1	10.7	10.3	13.1	18.9	20.2
2	10.8	12.1	14.4	20.5	30.4
3	10.9	12.5	15.7	23	
4	11	12.9	17	25.4	
5	11.1	12.7	17.6		
6	11.1	13.5	18.2		
8	11.2	14.9	19.2		
9	11.2	15.1			
15	11.3	15.5			
20	11.4				
26	13				

Figure 5 shows the AT^2 values with respect to the number of bits per word (*BPW*) for all the designs built for $n_{max}=160$-bits. The best AT^2 values changes extraordinarily with both *BPW* and *NS* parameters, i.e. not following a normal logical rule. The best AT^2 scalable designs according to *BPW* are listed in Table 2. If the designer had the scalable multiplier stage designed for *BPW* of 8 bits, the best number of stages is 9, assuming area is not important compared to speed as AT^2. If the design is intended to handle 16-bits at a time (*BPW=16*), the number of stages should be 20.

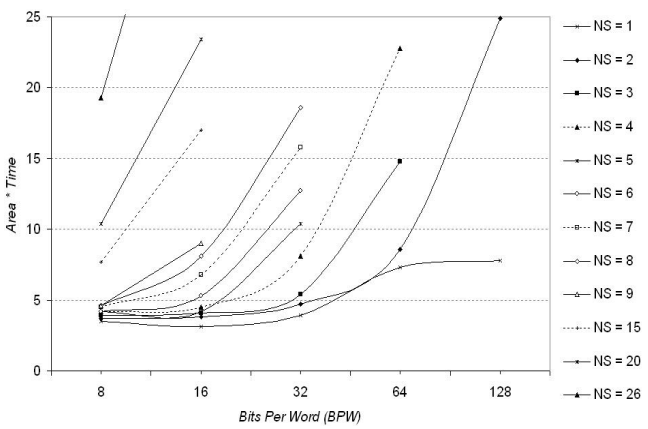

Figure 4. AT of several versions of 160-bits scalable designs.

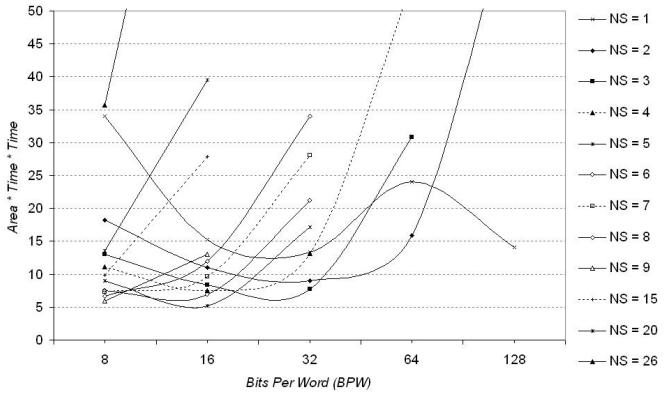

Figure 5. AT^2 of versions of the 160-bits scalable designs.

Table 2. Best AT^2 scalable designs according to BPW.

BPW	**8**	**16**	**32**	**64**	**128**
NS	9	20	3	3	1

4. Comparison With an Available 160-Bits FPGA Implementation

The proposed scalable hardware is compared with another published hardware implementation presented by Ors in 2003 [15]. Ors designed an FPGA hardware to perform ECC in GF (p) for data size of 160-bits. The FPGA hardware consists of special operational blocks for memory, Montgomery multiplication, modular addition/subtraction, and a controller of finite state machines to organize the ECC flow operations. The FPGA design uses projective coordinate arithmetic, similar to our scalable hardware, to avoid inversion complexity. The critical path of the FPGA design is determined by the multiplier, which is based on a systolic array structure built specifically for crypto applications. It has been reported that the FPGA implementation have been built in an area equivalent to 115,520 gates with longest path delay of 10.952 nanoseconds. Its average computation time has been estimated to 14.414 milliseconds using our same average computation time method (number of point additions half the number of doubling) [15].

4.1. Area Comparison

The areas of different scalable designs and the FPGA one are compared in Figure 6. The area of the FPGA hardware does not relate to the *BPW* value, however, it is shown as a constant line in Figure 6 to complete the area comparison study. Observe that most scalable hardware designs are having area smaller than the FPGA one. The cases of the FPGA implementation to have better area are when the number of stages is large, i. e., *NS > 20*, and the *BPW > 8*, which is unrealistic to implement.

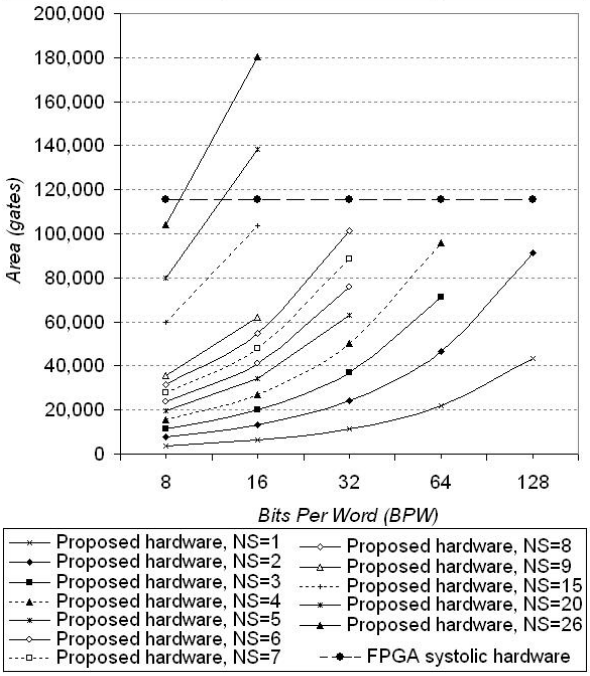

Figure 6. Area comparison of different *160-bits* designs.

4.2. Speed Comparison

The computation time of several different scalable designs is shown in Figure 7 compared to the ECC FPGA design. All the designs are assumed to operate for n_{max} *= 160 - bits*. The FPGA single multiplier design is very slow compared to all parallel scalable ones. For example, the FPGA hardware needs almost double the time of the parallel designs with *NS = 1*. It needs more than triple the time of all other scalable designs. In fact, the FPGA hardware needs about ten times the time of the scalable hardware with the large number of stages, such as when *NS = 20*. However, this large *NS* may be impractical to implement because of its large area, as mentioned earlier.

Note the affect of increasing the *BPW* number for each *NS* scalable design. As *BPW* goes high in most of the designs, the total computation time start increasing. This indicates that it is not necessarily by increasing the *BPW*, time reduces, in fact, the time may increase. This extra hardware will cause extra computation delay.

5. Conclusion

This paper presents scalable hardware models of a procedure used in the computation of 160-bits elliptic curve cryptography. The models act as if the inverse operation is converted into consecutive multiplication steps using a method known as projective coordinates, projecting *(x, y)* to *(X / Z,Y / Z)*. The proposed hardware architectures implement the ECC procedures into four parallel multipliers that enjoy 100% utilization.

An important comment about the implementation of our proposed architecture is that we propose to use

scalable multipliers which depend on digit serial multiplications. Digit serial computation is more suitable for the elliptic curve crypto algorithm discussed above since the computation of elliptic point doubling, addition and the algorithm of computing multiples of the base point is such that the multiplication of one stage must be completed before starting the multiplication of the subsequent stage. Therefore, even if a pipelined bit-parallel multipliers is used, the throughput of such a multiplier can not be exploited since the next multiplication operation can not commence until the multiplication operations in the previous stage has completed. The scalable multiplier used is flexible to give different hardware versions of the same basic multiplier depending on the number of stages (*NS*) and the number of bits per word (*BPW*) each *stage* is handling.

The proposed design is compared to an available FPGA implementation showing interesting area and speed results.

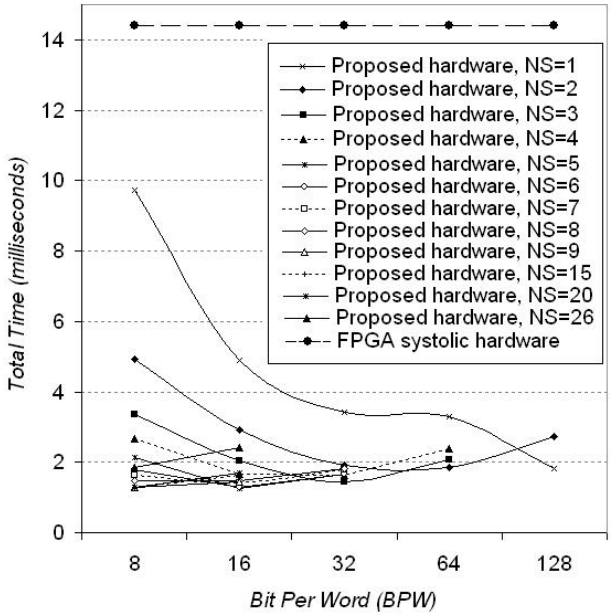

Figure 7. Total time comparison of different *160-bits* designs.

Acknowledgments

Thanks are due to Professor Ibrahim M. K. for his valuable suggestions. Thanks to King Fahd University of Petroleum and Minerals, KFUPM-Dhahran, for its support.

References

[1] Agnew G. B., Mullin R. C., and Vanstone S. A., "An Implementation of Elliptic Curve Cryptosystems Over F_2155," *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 5, pp. 804-813, June 1993.

[2] Blake I., Seroussi G., and Smart N., *Elliptic Curves in Cryptography*, Cambridge University Press, New York, 1999.

[3] Crutchley D. A., "Cryptography and Elliptic Curves," *Master Thesis*, Faculty of Mathematics, University of Southampton, England, May 1999.

[4] Ercegovac M. D., Lang T., and Moreno J. H., *Introduction to Digital Systems*, John Wiley & Sons, New York, 1999.

[5] Gutub A., and Ibrahim M. K., "High Radix Parallel Architecture for GF (P) Elliptic Curve Processor," *in Proceedings of IEEE Conference on Acoustics, Speech, and Signal Processing (ICASSP'2003)*, Hong Kong, pp. 625-628, April 2003.

[6] Hankerson D., Hernandez J., and Menezes A., "Software Implementation of Elliptic Curve Cryptography Over Binary Fields," *in Proceedings of Workshop on Cryptographic Hardware and Embedded Systems (CHES 2000*), Massachusetts, August 2000.

[7] Koblitz N., "Elliptic Curve Cryptosystems," *Mathematics Computing*, vol. 48, pp. 203-209, 1987.

[8] Mentor Graphics Co., ASIC Design Kit, available at: http://www.mentor.com/partners/hep /AsicDesignKit/dsheet/ami05databook.html.

[9] Michener J. R. and Mohan S. D., "Internet Watch: Clothing the e-Emperor," *Computer Innovative Technology for Computer Professionals*, IEEE Computer Society, vol. 34, no. 9, pp. 116-118, September 2001.

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

[11] Miyaji A., "Elliptic Curves Over FP Suitable for Cryptosystems," *in Proceedings of Advances in Cryptology (AUSCRUPT'92)*, Australia, December 1992.

[12] Montgomery P. L., "Modular Multiplication Without Trail Division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519-521, April 1985.

[13] Naccache D. and M'Raihi D., "Cryptographic Smart Cards," *IEEE Micro*, vol. 16, no. 3, pp. 14-24, June 1996.

[14] Orlando G. and Paar C., "A Scalable GF (P) Elliptic Curve Processor Architecture for Programmable Hardware," *Cryptographic Hardware and Embedded Systems (CHES 2001)*, Paris, France, May 2001.

[15] Ors S. B., Batina L., Preneel B., and Vandewalle, J., "Hardware Implementation of an Elliptic Curve Processor Over GF (P)," *in Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP'03)*, pp. 433-443, June 2003.

[16] Orton G. A., Roy M. P., Scott P. A., Peppard L. E., and Tavares S. E., "VLSI Implementation of Public-Key Encryption Algorithms," *in*

Proceedings of Advances in Cryptology (CRYPTO'86), pp. 277-301, August 1986.

[17] Paar C., Fleischmann P., and Soria-Rodriguez P., "Fast Arithmetic for Public-Key Algorithms in Galois Fields with Composite Exponents," *IEEE Transactions on Computers*, vol. 48, no. 10, October 1999.

[18] Raju G. V. S., and Akbani R., "Elliptic Curve Cryptosystem and its Applications," *in Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, vol. 2, pp. 1540 - 1543, October 2003.

[19] Rivest R. L., Shamir A., and Adleman L., "A Method for Obtaining Digital Signatures and Public-key Cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120-126, 1978.

[20] Stallings W., *Cryptography and Network Security: Principles and Practice*, Prentice Hall, NJ, 1999.

[21] Stinson D. R., *Cryptography: Theory and Practice*, CRC Press, Boca Raton, Florida, 1995.

[22] Tenca A. F., Todorov G., and Koc C. K., "High-Radix Design of a Scalable Modular Multiplier," *Cryptographic Hardware and Embedded Systems (CHES'2001)*, Paris, France, pp. 185-201, May 2001.

Adnan Abdul-Aziz Gutub is a faculty member in the Computer Engineering Department at King Fahd University of Petroleum and Minerals (KFUPM), Saudi Arabia. He received his BSc degree in electrical engineering in 1995, his MSc degree in computer engineering in 1998 both from King Fahd University of Petroleum and Minerals, and his PhD degree in 2002 from the Department of Electrical and Computer Engineering at Oregon State University in cryptographic hardware design. His research interests include 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 of summer 2005 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.