A.Durga Bhavani, P.Soundarya Mala / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 2, Issue 5, September- October 2012, pp.412-419

Optimized Elliptic Curve Cryptography

A.Durga Bhavani*, P.Soundarya Mala**
Dept of Electronics and Communication Engg,GIET College,Rajahmundry.

Abstract—

This paper details the design of a new high-speed pipelined application-specific instruction set processor (ASIP) for elliptic curve cryptography (ECC) technology. Different levels of pipelining were applied to the data path to explore the resulting performances and find an optimal pipeline depth. Three complex instructions were used to reduce the latency by reducing the overall number of instructions, and a new combined algorithm was developed to perform point doubling and point addition using the application specific instructions In the present work, by using pipeline techniques are optimized the point multiplication speed by implementing on modern Xilinx Virtex-4 and the same to be compare with the Xilinx Virtex-E.

Key Terms*—*Complex instruction set, efficient hardware implementation, elliptic curve cryptography (ECC), pipelining.

I. INTRODUCTION

FIRST introduced in the 1980s, elliptic curve cryptography (ECC) has become popular due to its superior strength- per-bit compared to existing public key algorithms. This superiority translates to equivalent security levels with smaller keys, bandwidth savings, and faster implementations, making ECC very appealing. The IEEE proposed standard P1363-2000 recognizes ECC-based key agreement and digital signature algorithms and a list of secure curves is given by the U.S. Government National Institute of Standards and Technology (NIST).

Intuitively, there are numerous advantages of using field- programmable gate-array (FPGA) technology to implement in hardware the computationally intensive operations needed for ECC. Indeed these advantages are comprehensively studied and listed by Wollinger, *et al.* In particular, performance, cost efficiency, and the ability to easily update the cryptographic algorithm in fielded devices are very attractive for hardware implementations for ECC.

Numerous ECC hardware accelerators and cryptographic processors have been presented in the literature. More recently, these have included a number of FPGA architectures, which present acceleration techniques to improve the performance of the ECC operations.

The optimization goal is usually to reduce the latency of a point multiplication in terms of the number of required cycles. In particular, the works which have duplicate arithmetic blocks to exploit the parallelism in the underlying operations. Yet for most of these implementations, efforts are concentrated on algorithm optimization or improved arithmetic architectures and rarely on a processor architecture particularly suited for ECC point multiplication. Some of the design techniques used in modern high performance processors were incorporated into the design of an application-specific instruction set processor (ASIP) for ECC. Pipelining was applied to the design, giving improved clock frequencies. Data forwarding and instruction reordering were incorporated to exploit the inherent parallelism of the Lopez and Dahab point multiplication algorithm, reducing pipeline stalls.

In this paper, thorough treatment is given to the design of an ASIP for ECC, yielding a new combined algorithm to perform point doubling and point addition based on the instruction set developed, further reducing the required number of instructions per iteration. The same data path is used, but the performance is explored for different levels of pipelining, and a superior choice of pipeline depth is found. The resulting processor has high clock frequencies and low latency, and it has only a single instance of each of the arithmetic units. An FPGA implementation over GF $2^{163}$is presented, which is by far the fastest implementation reported in the literature to date.

In Section II, a background is given in terms of elliptic curve operations, Galois fields arithmetic and the state-of-the-art hardware implementations. Section III develops the ASIP architecture, beginning with the application-specific instructions and a new combined algorithm to perform point doubling and point addition (crux ECC operations) based on the new algorithms. In Section IV, the FPGA implementation of the processor is described, and the performance is analyzed and compared to the state-of-the-art in Section V. This paper is concluded in Section VI.

II. BACKGROUND

A. ECC

ECC is performed over one of two underlying Galois fields: prime order fields or characteristic two fields. Both fields are considered to provide the same level of security, but arithmetic in GF(2^m) will be the focus of this paper because it can be implemented in hardware more efficiently using modulo-2 arithmetic. An elliptic curve over the field is the set of solutions to the equation

$$y^2+xy = x^3+ax^2+b \qquad (1)$$

where a,b € GF(2^m),b≠0

Let $P_1 = (x_1, y_1) \in E$, and $P_2 = (x_2, y_2) \in E$, then summing the two points is $P_1 + P_2 = P_3 = (x_3, y_3) \in E$, where

$$x_3 = \begin{cases} \left(\frac{y_1+y_2}{x_1+x_2}\right)^2 + \frac{y_1+y_2}{x_1+x_2} + x_1 + x_2 + a, & P_1 \neq P_2 \\ x_1^2 + \frac{b}{x_1^2}, & P_1 = P_2 \end{cases} \qquad (2)$$

$$y_3 = \begin{cases} \left(\frac{y_1+y_2}{x_1+x_2}\right)(x_1+x_3) + x_3 + y_1, & P_1 \neq P_2 \\ x_1^2 + \left(x_1 + \frac{y_1}{x_1}\right)(x_3) + x_3, & P_1 = P_2. \end{cases} \qquad (3)$$

Hence, when $P_1 = P_2$ we have the point-doubling operation (DBL), and when $P_1 \neq P_2$ we have the point-adding operation (ADD). These operations in turn constitute the crux of any ECC-based algorithm, known as point multiplication or scalar multiplication. Due to the computational expense of inversion compared to multiplication, several projective coordinate methods have been proposed, which use fractional field arithmetic to defer the in- version operation until the end of the point multiplication; In this paper, the high-performance, generic projective-coordinate algorithm proposed by Lopez and Dahab is used, which is an efficient implementation of Montgomery's method for computing kP. No precomputations or special field/curve properties are required.

In this, procedures to perform DBL and ADD are derived from efficient formulas which use only the coordinate of the points. In the projective coordinate version of the formulas, the x-coordinate of P_i is represented by X_i/Z_i, for i € {1,2,3};

The corresponding DBL and ADD computations are shown respectively, and are used in the projective coordinate point multiplication algorithm shown in Algorithm 1

$$x(2P_i) = X_i^4 + b.Z_i^4$$
$$z(2P_i) = Z_i^2.X_i^2 \qquad (4)$$

$$Z_3 = (X_1.Z_2 + X_2.Z_1)^2$$
$$X_3 = x.Z_3 + (X_1.Z_2)\cdot(X_2.Z_1). \qquad (5)$$

B. *Arithmetic Over* GF(2^{163}) *for ECC*

Addition and subtraction in the Galois field GF(2^m) are equivalent, performed by modulo-2 addition, i.e., a bit-XOR operation. As a result, arithmetic in the field is implemented more efficiently because it is carry free.

Inversion is the most computationally expensive operation in the field, based either on the extended Euclidean algorithm (EEA) or Fermat's little theorem. Many efficient EEA-based inversion and division architectures exist in the literature, but they are usually expensive in terms of area. In- version using Fermat's little theorem, as in the Itoh–Tsujii algorithm, consists of multiplication and squaring only, so can often be implemented without additional hardware resource. It has been shown that similar performance to an EEA-based inverter can be achieved using the Itoh–Tsujii algortihm if the multiplier latency is sufficiently low and repeated squaring can be performed efficiently. Hence, multiplication is considered to be the most resource-sensitive operation in the field.

Algorithm 1: Lopez-Dahab Point Multiplication

```
Input: An integer $k \geq 0$ and a base point $P = (x, y) \in E$.
Output: $Q = kP$.

If $k = 0$ or $x = 0$ then output $(0, 0)$ and stop.
Set $k \leftarrow (k_{l-1}k_{l-2} \ldots k_0)$
Set $X_1 \leftarrow x, Z_1 \leftarrow 1, X_2 \leftarrow x^4 + b, Z_2 \leftarrow x^2$.
For $j$ from $l-2$ downto 0 do
    If $k_j = 1$ then
        ADD$(X_1, Z_1, X_2, Z_2)$, DBL$(X_2, Z_2)$
    Else
        ADD$(X_2, Z_2, X_1, Z_1)$, DBL$(X_1, Z_1)$.
Return$(Q = \mathrm{Proj2}Aff(X_1, Z_1, X_2, Z_2))$.
```

Multipliers are usually implemented using one of three approaches: bit-serial, bit-parallel, or digit-serial. Bit-serial multi- pliers are small, but they require m steps to perform a multiplication. Bit-parallel multipliers are large but perform a multiplication in a single step. Digit-serial multipliers are most commonly used in cryptographic hard- ware, as performance can be traded against area. Recently, in a drive for increased speed, some methods have been proposed, and implementations presented , for ECC hardware based on bit-parallel multipliers.

Squaring is a special case of multiplication, and it is only a little more complex than reduction modulo the irreducible polynomial. Refer to finite field multiplier for efficient architectures to perform polynomial basis squaring, which only require combinational logic.

C. *Previous Work*

Several recent FPGA-based hardware implementations of ECC have achieved high-performance throughput. Various acceleration techniques have been used, usually based on parallelism or pre computation.

The work introduced by Orlando and Paar[2] is based on the Montgomery method for computing *kp* developed by Lopez and Dahab and operates over a single field. A point multiplication over GF(2^{167}) is performed in 210 micro seconds, using a Galois field multiplier with an eleven-cycle latency. This provides an excellent benchmark for all high-performance architectures.

Acceleration techniques based on precomputation can be particularly effective for a class of curves known as anomalous binary curves (ABCs) or Koblitz curves. The implementation from Lutz and Hasan implements some of these techniques achieving a point-multiplication time of 75 µs over, using a Galois field multiplier with a four-cycle latency. How- ever, efforts here will be concentrated on high-performance architectures for generic curves, where such acceleration techniques cannot be used. An important result of , relevant regardless of the point-multiplication algorithm used, is that efficiently performing repeated squaring operations greatly reduces the cost of multiplicative inversion, which is required at the end of the point multiplication.

An ECC processor capable of operating over multiple Galois fields was presented by Gura, *et al.* , which performs a point multiplication over GF(2^{163})in 143 µs, using a Galois field multiplier with a three-cycle latency. Gura, *et al.* stated two important conclusions: the efficient implementation of inversion has a significant impact on overall performance and, as the latency of multiplication is improved, system tasks such as reading and writing have a significant impact on performance. The work introduced by Jarvinen, *et al.* uses two bit-parallel multipliers to perform multiplications concurrently. The multipliers have several registers in the critical path in order to operate at high clock frequencies, but the operations are not pipelined, resulting in long latencies (between 8 and 20 cycles). Hence, while this implementation offers high performance, it is not the fastest reported in the literature but it is one of the largest. Rodriguez, *et al.* introduced an FPGA implementation that performs DBL and ADD in parallel, containing multiple in- stances of circuits to perform the arithmetic functions. It would appear that the inversion required for coordinate conversion at the end of the point multiplication is not performed, and that the quoted point multiplication time does not include the co- ordinate conversion, but the stated point multiplication time is one of the fastest in the literature. However, the complex structure of the multiplier has a long critical path, and as a result the overall performance is let down by quite a low clock frequency (46.5 MHz).

More recently, Cheung, *et al.* presented a hardware design that uses a normal basis representation. The customizable hardware offers a trade between cost and performance by varying the level of parallelism through the number of multipliers and level of pipelining. To the authors' knowledge, this is the fastest implementation in the literature performing a point multiplication in approximately 55 µs, although once again the low clock frequency (43 MHz) limits the potential performance.

III. PIPELINED ASIP FOR ECC

A.Application-Specific Instruction Set and Algorithms

Complex instruction set computers (CISCs) reduce the number of instructions, and consequently the overall latency, by performing multiple tasks in a single instruction. It was shown that complex instructions could be used to reduce latency in the design of an ASIP for ECC. Three new instructions were introduced, which will now be described and their use justified.

Considering the projective-coordinate formula for point addition, an obvious instruction to combine operations is a multiply and accumulate instruction (MULAD), which will save two instruction executions. Also, rewriting the projective-coordinate formula for point doubling , we have

$$x(2P_i) = (X_i.X_i)^2 + b.(Z_i.Z_i)^2$$
$$z(2P_i) = (Z_i.X_i)^2 \qquad (6)$$

so a multiply- and- square operation (MULSQ) would be beneficial, saving three instruction executions.

These two extra instructions reduce the number of instructions executions required to perform the point-adding and point- doubling algorithms from 14 to 9—a 35.7% reduction.

The Itoh–Tsujii inversion algorithm is based on Fermat's little theorem and is used to compute the inversion at the end of the point multiplication. The algorithm uses addition chains to reduce the required number of multiplications, but it contains exponentiations that must be performed through repeated squaring operations—as many as 64 repeated squaring operations for the field GF(2^{163}). It was shown, that relatively low latencies can be achieved using this algorithm if repeated squaring can be performed efficiently. Hence, a third application-specific instruction will be used to perform repeated squaring (RESQR) in order to accelerate the Itoh–Tsujii inversion algorithm.

Using these instructions, a combined algorithm to perform point doubling and point addition can be developed, which has only nine arithmetic instructions, shown in Algorithm 2.

B. Pipelined Data Path

A data path capable of performing the new application specific instructions is required.

Any of the approaches to implementing multiplication mentioned in the background section could be used, but because this work develops high-throughput hardware a bit-parallel multiplier is used. The data path must be constructed such that the result of a multiplication can be squared or added to the previous result. Furthermore, feedback is required so that the result can be repeatedly squared. This functionality is implemented using the SQR/ADD block, described in Section III-D.

To maintain high-throughput performance, high clock frequencies are required and one instruction/cycle or more is desirable. Therefore, the data path must be pipelined. The pipelined data path is shown in Fig. 1. The performance-critical component is the multiplier, so to achieve high clock frequencies, we must sub pipeline the multiplier.

Algorithm 2. New Combined Algorithm to Perform DBL and ADD

Input: x-coordinates X_1/Z_1 for P_1 and X_2/Z_2 for P_2; the curve parameter b;
x, the x-coordinate of the base point P.
Output: $x-x$oordinates X_1/Z_1 for P_1+P_2 and X_2/Z_2 for $2P_2$.

```
$T_1 \leftarrow \mathrm{MUL}(X_2, Z_1)$
$T(2) \leftarrow \mathrm{MULAD}(X_1, Z_2)$
$Z_1 \leftarrow \mathrm{RESQ}(T_2)$
$T_3 \leftarrow \mathrm{MULSQ}(Z_2, Z_2)$
$Z(2) \leftarrow \mathrm{MULSQ}(X_2, Z_2)$
$T_1 \leftarrow \mathrm{MUL}(T_1, T_2)$
$X_1 \leftarrow \mathrm{MULAD}(Z_3, x)$
$X_2 \leftarrow \mathrm{MULSQ}(X_2, X_2)$
$X_2 \leftarrow \mathrm{MULAD}(T_3, b)$
```

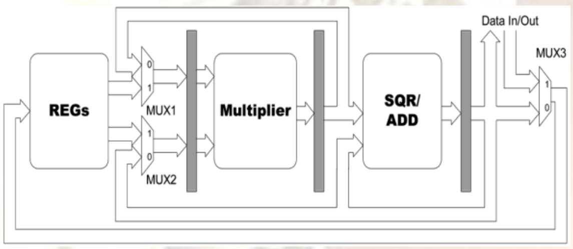


Fig. 1. Pipelined ASIP data path.

C. *Subpipelined Bit-Parallel Multiplier*

Some recent hardware implementations of elliptic-curve accelerators, in addition to some proposed architectures, have used bit-parallel multipliers. Numerous architectures for bit-parallel multiplication exist in the literature. However, very few actual implementations of bitParallel multiplication have been presented for the large fields used in public-key cryptography. Where they have been implemented, poor performance has often been reported either in terms of clock frequency or latency. This is not surprising given the deep sub micrometer fabrics used in modern FPGA devices. The major component of delay is due to routing, and the number of interconnections in a bit-parallel multiplier implemented for ECC will be in the region of many tens of thousands or more making routing very complex. Therefore, to improve routing complexity and to reduce the amount of logic in the critical path, a bit-parallel multiplier that is amenable to pipelining is desirable.

The well-known Mastrovito multiplier is such an architecture; the reader is referred to the original work for full details. The multiplication C(x)=A(x)B(x) mod G(x) is expressed in matrix notation as

$$C = \begin{pmatrix} f_{0,0} & \cdots & f_{0,m-1} \\ \vdots & \ddots & \vdots \\ f_{m-1,0} & \cdots & f_{m-1,m-1} \end{pmatrix} \begin{pmatrix} b_0 \\ \vdots \\ b_{m-1} \end{pmatrix} = ZB.$$

The Z matrix is referred to as the product matrix and the functions f i,j are linear functions of the components of A(x).The columns of represent the consecutive states of the Galois type LFSR with the initial state given by the multiplicand A(x), i.e., the j^{th} column is given by x^j A(x) mod G(x). Therefore, the product matrix can be realized by cascading instances of the logic to perform a left shift modulo G(x). The circuit to perform this modulo shift is referred to as an alpha cell. The same notation will be used here.

The product matrix can be divided into a number of smaller submatrices that calculate the partial products, which give the final product when summed together.

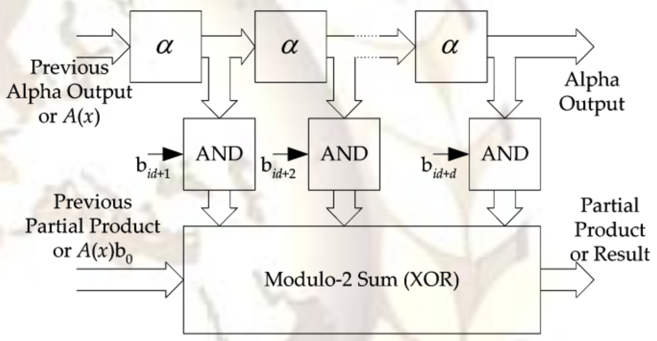


Fig. 2. Pipeline stage of subpipelined bit-parallel multiplier.

$$C = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{m-1} \end{pmatrix} b_0 + \begin{pmatrix} f_{0,1} & \cdots & f_{0,d} \\ f_{1,1} & \cdots & f_{1,1} \\ \vdots & \ddots & \vdots \\ f_{m-1,1} & \cdots & f_{m-1,d} \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_d \end{pmatrix} + \cdots$$
$$+ \begin{pmatrix} f_{0,(D-1)d+1} & \cdots & f_{0,m-1} \\ f_{1,(D-1)d+1} & \cdots & f_{1,m-1} \\ \vdots & \ddots & \vdots \\ f_{m-1,(D-1)d+1} & \cdots & f_{m-1,m-1} \end{pmatrix}$$
$$\times \begin{pmatrix} b_{m-d} \\ b_{m-d+1} \\ \vdots \\ b_{m-1} \end{pmatrix}. \qquad (7)$$

More formally, the product matrix can be divided into D submatrices, which will contain at most d =[m-1/D] columns,i.e,

Hence, the multiplier can be easily pipelined into D stages, each computing the sum of at most d+1 terms, which are the outputs of d alpha cells and the input from the previous multiplication pipeline stage. As a result the logic in the critical path is reduced and the routing is simplified. Note that the number of logic gates in the critical path of each stage will be similar to that of a digit serial multiplier with equivalent d, but the routing is simpler as no feedback is required. The resulting architecture of the i th pipeline stage 0≤i≤D-1, is shown in Fig. 2;

When the alpha input is A(x) and the sum input is A(x)b_0, when i = D-1 the sum output is the multiplication result and the alpha output is unconnected.

D. SQR/ADD Block

The extra functionality required for the new instructions MULAD, MULSQ, and RESQ is provided by the SQR/ADD block, which is appended to the multiplier output as shown in Fig. 1. A block diagram showing the functionality of the SQR/ADD block is shown in Fig. 3; note that the flip-flops shown in Fig. 3 are the pipeline flip-flops placed after the SQR/ADD block in the data path diagram shown in Fig. 1.

Selection 0 on the MUX shown in Fig.3 sets the data path to perform standard multiplication over GF(2^m). Selection 1 sets the data path to perform MULSQ. Selection 2 sets the data path to perform MULAD. Selection 3 sets the data path to perform RESQ; note that RESQ can be performed after any other operation, e.g., MULAD-RESQ is a valid instruction sequence.

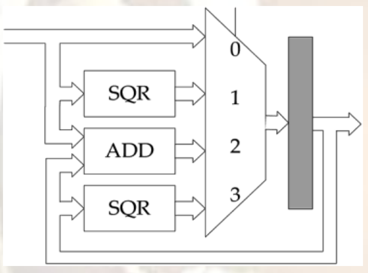


Fig.3. SQR/ADD block.

E. Data Forwarding

The optimal depth of a pipeline is a trade between clock frequency and instruction throughput. Ideally, one instruction per cycle (or more) will be performed, while sufficient pipelining to achieve the desired clock frequency is implemented. However, as the depth of the pipeline is increased, data dependencies can lead to pipeline stalls and even pipeline flushes, resulting in less than the ideal one instruction/cycle being performed.

Data forwarding is a common technique in processor design, used to avoid pipeline stalls that occur due to data dependencies. Typically, data is forwarded to either input of the arithmetic logic unit (ALU) from all subsequent pipeline stages. The scheme used here differs slightly because such generality is not necessary: the order of the instructions and the instances when forwarding is required are known and do not change. Therefore, the control signals may be explicitly defined for each instruction, and, as can be seen in Fig. 3, the data path can be simplified because data forwarding to both ALU inputs from all sub- sequent stages is not required.

IV. FPGA IMPLEMENTATION

The processor was implemented over the smallest field recommended by NIST [8] GF(2163) using the irreducible polynomial given, G(x)=x163+x7+x6+x3+1.Two FPGA devices were used for implementation: the older Virtex-E de- vice (XCV2600E-FG1156-8),for fair comparison with the other architectures presented in the literature, and the more modern Virtex-4 (XC4VLX200-FF1513-11) to demonstrate the performance of the proposed ASIP with modern technology. Each component was implemented and optimized individually, particularly the multiplier, to examine the optimal pipeline depth before the complete processor was implemented. Aggressive optimization for speed was performed, including timing-driven packing and placement. No detailed floor planning was performed, only a few simple constraints were applied.

A. Register File

Xilinx FPGA devices support three main types of storage element: flip-flops, block RAM and distributed RAM. Block RAM is the dedicated memory resource of FPGA devices, which has different sizes and locations depending on the device being used.

TABLE I
DECOMPOSITION OF POINT MULTIPLICATION TIME IN CYCLES

Pipeline Stages	(DBL and ADD)	Itoh-Tsujii Inversion	Coordinate Conversion	Point Multi-plication
5	13	201	226	2332
6	15	211	241	2671
7	17	221	256	3010
8	19	231	274	3352

Distributed RAM uses the basic logic elements of the FPGA device, lookup tables (LUTs), to form RAMs of the desired size, function and location, making it far more flexible. On Virtex-E devices, Block RAMs are 4096 bits each, which increase to 18 k bits if the Virtex-4 is used. So the use of block RAM to store relatively small amounts of data is inefficient. The processor presented in this paper requires only 13 storage locations including all temporary storage, so distributed RAM was used. On Xilinx FPGA devices, 16-bits of storage can be gained from a single LUT. However, as reading from two ports and writing to a third port is required independently and concurrently, a single dual-port distributed RAM is not sufficient. Therefore, the register file utilized in this design is formed by cross-connecting two dual-port distributed RAMs. This approach is still far more efficient than using block RAMs or flip-flops.

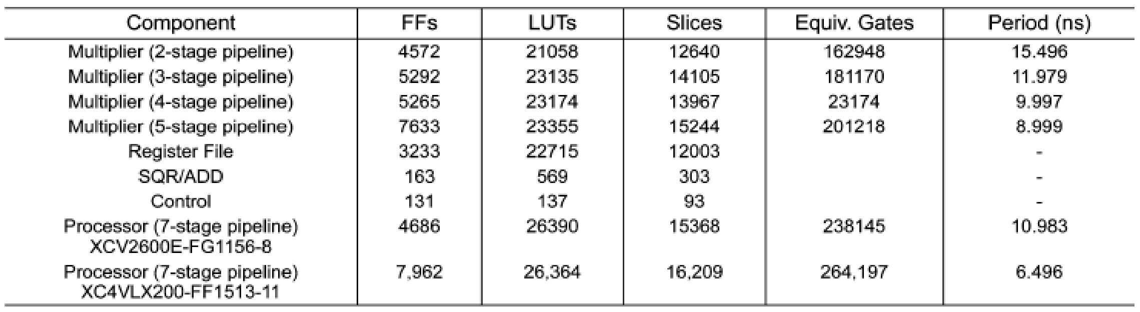
B. Subpipelined Multiplier

The multiplier dominates the processor, both in terms of area (accounting for over 95% of the total) and clock frequency, as the critical path of the other components is considerably shorter than that of the multiplier. The area of a bit parallel multiplier is O(m^2) and the critical path is O(log(m)). However, when implemented on FPGA, these measures have less significance. To implement a bit-parallel multiplier without a pipelining scheme will lead to low clock frequencies, due to the amount of logic in the critical path and the complex routing.

The Mastrovito multiplier architecture was chosen not only because it was suitable for pipelining, but also because its regularity simplifies placement and, consequently, routing. Given the deep sub micrometer fabrics used by FPGAs, the routing is the largest component of delay, and, as shown in Section III-C, the Mastrovito multiplier can be divided into very regular blocks to improve routing. When pipelined, the implemented architecture has a critical path similar to a digit-serial multiplier, but the routing is simplified because no feedback is required.

The multiplier was implemented with several different depths of pipeline; the results are shown in Table II. When timing- driven FPGA placement is performed, the placer may be forced to heavily replicate certain areas of the circuit to meet timing goals, particularly if certain nets have high fan-outs. This can be seen in the implementation results of the multiplier, where the multiplier with three pipeline stages requires more resources than its four-stage equivalent because heavy replication was performed to meet timing goals. In terms of area-time performance, it is clear that the four-stage multiplier offers superior performance, which is an important result when the overall performance of the processor is considered.

TABLE II
FPGA IMPLEMENTATION RESULTS (XILINX XCV2600E-FG1156-8 UNLESS STATED OTHERWISE)



Component	FFs	LUTs	Slices	Equiv. Gates	Period (ns)
Multiplier (2-stage pipeline)	4572	21058	12640	162948	15.496
Multiplier (3-stage pipeline)	5292	23135	14105	181170	11.979
Multiplier (4-stage pipeline)	5265	23174	13967	23174	9.997
Multiplier (5-stage pipeline)	7633	23355	15244	201218	8.999
Register File	3233	22715	12003		-
SQR/ADD	163	569	303		-
Control	131	137	93		-
Processor (7-stage pipeline) XCV2600E-FG1156-8	4686	26390	15368	238145	10.983
Processor (7-stage pipeline) XC4VLX200-FF1513-11	7,962	26,364	16,209	264,197	6.496

C. SQR/ADD Block

By implementing the functionality as a single Boolean function, improved logic optimization can be achieved because the hierarchy will be flattened leading to resource sharing and improved logic minimization.

L

D. Control

Typically, a general-purpose processor with a pipelined data path would use an instruction ROM for control. However, for applications such as this one where the mode of operation is fixed and the control is relatively simple, a state machine is far more efficient, particularly in terms of area, and thus is the choice for this design.

A Moore state machine is most suitable for the proposed architecture, as any extra area required is negligible compared to the overall area of the processor, and the improved clock frequency compared to a Mealy state machine is desirable. The key is stored in a special-purpose register, independent of the data path, and loaded through a separate channel. Thus, the state machine has no dependency on the data path and the control signals can be registered to achieve the desired delay performance. The total resource usage for control is only 93 slices. However, if extra flexibility is required, an instruction ROM can be used with no performance penalty in terms of speed, though more area resources would be required.

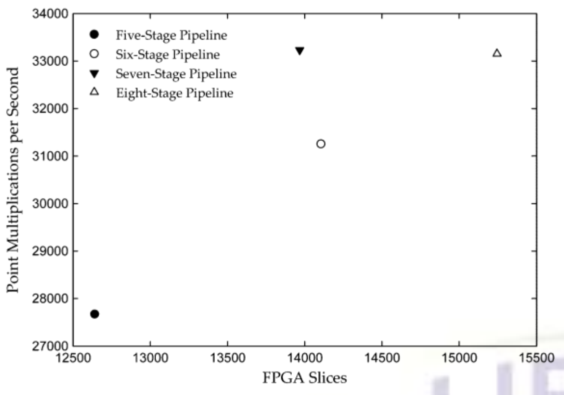
V. PERFORMANCE ANALYSIS

Let be the depth of the pipeline, then the number of cycles required to perform a point multiplication (excluding data I/O and coordinate conversion) is given by

$$\text{Latency(cycles)} = 2L+3$$

The latency in cycles of the point multiplication is decomposed in Table I. To ascertain the optimal pipeline depth for the processor, the bit-parallel multiplier was implemented with different pipeline depths. The results of these implementations (see Table II) and the corresponding number of cycles to perform a point multiplication (see Table I) can be used to estimate the point multiplication times for each pipeline depth, as the critical path of the processor is in the multiplier.

Fig. 4 plots throughput (point multiplications per second) against area (FPGA slices) for each of the multiplier variations implemented. Note that the area figure shown is the area of the multiplier not that of the complete processor, but the multiplier accounts for over 95% of the total area.



It is clear that the relationship between pipeline depth and area throughput performance is approximately linear (as we would expect from the previous equation), but the seven-stage pipeline offers superior performance. Therefore, this was the architecture implemented on both the Virtex-E and Virtex-4 devices, the results of which are detailed in Table II and summarized in Table III for comparison with the state of the art. It appears that none of the architectures in the comparison were floor planned, so for a fairer comparison the implementation of the proposed architecture was not floor planned either. It is important to note for FPGA implementations, being target specific, a detailed comparison of resource usage is not always straightforward, in particular when it is not clear whether the quoted figures are actual post place and route implementation results as in our case or merely synthesis estimates.

TABLE III
COMPARISON OF PROPOSED ASIP WITH ALTERNATIVE ARCHITECTURES IN THE LITERATURE

Architecture	m	FPGA Device	Area (slices)	Freq (MHz)	Time (cycles)	Time (μs)
Lutz [1] (Random Curves)	163	XCV2000E	-	66	15378	233
Lutz [1] (Koblitz Curves)	163	XCV2000E	-	66	4950	75
Orlando [2]	167	XCV400E-BG432-8	-	76.7	16107	210
Gura [4]	163	XCV2000E-FG680-7	-	66.4	9495	143
Jarvninen [5]	163	XC2V8000-5	18 079	90.2	9561	106
Rodriguez [7]	191	XCV2600E	17 630	46.5	2930	63
Cheung [10]	162	XC2V600-4	-	54	3240	60
Cheung [10]	163	XCV2000E-8	-	~43	-	55 (estimate)
Chelton [13]	163	XCV2600E-FG1156-8	15 020	77.01	2831	36.77
Proposed (6-stage)	163	XCV2600E-FG1156-8	15 280	76.85	2671	34.76
Proposed (7-stage)	163	XCV2600E-FG1156-8	15 368	91.05	3010	33.05
Proposed (7-stage)	163	XC4VLX200	16 209	153.9	3010	19.55

The proposed architecture compares very favorably with the state of the art, being smaller and considerably faster than similar works. The implementation of the proposed architecture is several times faster than the first three, lower-resource table entries, which are based on digit-serial multiplication. It is interesting to note that the proposed architecture achieved a higher clock frequency than all three, demonstrating that the simplified routing resulting from the pipelined architecture does indeed improve the critical path.

Comparing against alternative high-speed architectures, the implementation of the proposed architecture performs a point multiplication in only 60.01% of the estimated performance time of the fastest alternative. The area resource of is not stated for the field, only the performance time, but the resource usage (slices) of the proposed architecture is significantly less than the other alternative high-speed architectures. The improvements over the implementation from Jarvinen, *et al.* are due once more to the reduced area requirements but also due to the reduced latency resulting from pipelining and the complex instructions. The further seventh-stage pipelining has increased the clock frequency of the architecture without prohibitively increasing the latency, which leads to the fastest point multiplication time reported in the literature. Hence, the use of an application specific instruction set in conjunction with pipelining has better exploited operation parallelism compared with duplicating arithmetic circuits as proposed.

Comparing against our recently reported pipelined ASIP architecture, a 10.12% improvement in point multiplication time was attained, but the resource usage was only increased by around 2%. The improvement comes as a result of the new combined algorithm to perform DBL and ADD fewer instructions are required, which reduces the latency and the increased pipeline depth, which increases the clock frequency.

VI. CONCLUSION

A high-performance ECC has been implemented using FPGA technology. A combined algorithm to perform DBL and ADD was developed based on the Lopez Dahab Point multiplication algorithm .The data path was pipelined, allowing operation parallelism to be perform fastly and taking less time. Consequently, an implementation with a four-stage pipeline achieved a point multiplication on Xilinx Virtex-4 device, making it the fastest implementation. This work has confirmed the suitability of a pipelined data path and an efficient Galois field multiplier (2^163) is developed and implementations of ECC over GF(2^163) performs the better security with less key size .

REFERENCES

[1] J. Lutz and A. Hasan, "High performance FPGA based elliptic curve cryptographic co-processor," in *Proc. Int. Conf. Inf. Technol.: Coding Comput. (ITCC)*, 2004, p. 486.

[2] F. Rodriguez-Henriquez, N. A. Saqib, and A. Diaz-Perez, "A fast parallel implementation of elliptic curve point multiplication over GF(2m)," *Microprocessors Microsyst.*, vol. 28, pp. 329–339, 2004.

[3] N. Mentens, S. B. Ors, and B. Preneel, "An FPGA implementation of an elliptic curve processor GF(2/sup m/)," presented at the 14th ACM Great Lakes Symp. VLSI, Boston, MA, 2004.

[4] J. Lopez and R. Dahab, "Fast multiplication on elliptic curves over GF(2/sup m/) without precomputation," presented at the Workshop on Cryptographic Hardware Embedded Syst. (CHES), Worcester, MA,1999.

[5] G. Seroussi and N. P. Smart, *Elliptic Curves in Cryptography*. Cam- bridge, U.K.: Cambridge Univ. Press, 1999.

[6] C. H. Kim and C. P. Hong, "High-speed division architecture for GF(2m)," *Electron. Lett.*, vol. 38, pp. 835–836, 2002.

[7] W. Chelton and M. Benaissa, "High-speed pipelined ECC processor over GF(2m)," presented at the IEEE Workshop Signal Process. Syst.(SiPS), Banff, Canada, 2006.

[8] G. Seroussi and N. P. Smart, *Elliptic Curves in Cryptography*. Cam- bridge, U.K.: Cambridge Univ. Press, 1999.

[9] P. L. Montgomery, "Speeding the Pollard and elliptic curve methods of factorization," 1987.

[10] Z. Yan and D. V. Sarwate, "New systolic architectures for inversion and division in GF(2m)," *IEEE Trans. Comput.*, vol. 52, no. 11, pp.1514–1519, Nov. 2003.

[11] C.H.Kim and C. P. Hong, "High-speed division architecture for GF(2m)," *Electron. Lett.*, vol. 38, pp. 835–836, 2002.

[12] T. Itoh and S. Tsujii, "A fast algorithm for computing multiplicative inverses in GF(2/sup m/) using normal bases," *Inf. Computation*, vol.78, pp. 171–177, 1988.

[13]P.A.Scott,S.E. Tavares, and L. E. Peppard, "A fast VLSI multi- plier for GF(2/sup m/)," *IEEE J. Sel. Areas Commun.*, vol. 4, no. 1.

[14] H. Wu, "Bit-parallel finite field multiplier and squarer using polyno- mial basis," *IEEE Trans. Comput.*, vol. 51, no. 7, pp. 750–758, Jul.2002.

[15] A. Reyhani-Masoleh and M. A. Hasan, "Low complexity bit parallel architectures for polynomial basis multiplication over GF(2/sup m/),"*IEEE Trans. Comput.*, vol. 53, no. 8, pp. 945–959, Aug. 2004.

[16] L. Song and K. K. Parhi, "Low-energy digit-serial/parallel finite field multipliers," *J. VLSI Signal Process. Syst.*, vol. 19, pp. 149–166, 1998.

[17] M. C. Mekhallalati, A. S. Ashur, and M. K. Ibrahim, "Novel radix finitefield multiplier for GF(2/sup m/)," *J. VLSI Signal Process.*, vol. 15,pp.233–245, 1997.

[18] P. K. Mishra, "Pipelined computation of scalar multiplication in elliptic curve cryptosystems," presented at the CHES, Cambridge, MA, 2004.