Reconfigurable Computing

Elliptic Curve Processor (from K.H. Leung’s MPhil thesis)

Certainly the fact that the NSA is pushing elliptic-curve cryptography is some indication that it can break them more easily.

- Bruce Schneier
Overview

› Motivation
› Introduction to cryptography
› Mathematical Background
› Introduction to elliptic curves
› Previous Works
› Elliptic curve cryptographic processor (ECP)
› Result
› Conclusion
Motivation

- Cryptography is important to the Internet and E-commerce systems
- **ECC** is better than other public key cryptosystems
  - Lower memory and computational requirements
  - Higher security
- Reconfigurable hardware provides high flexibility, low cost, short turnaround time and possibility of upgrading in the future
Cryptography

- Keeps a message secret during transmission through untrusted and insecure channel

Diagram:

- Alice
  - Plaintext
  - Encryption $E$
  - Key Source
  - $M$
  - $K_e$

- Adversary
  - Insecure channel
  - Ciphertext $C$

- Bob
  - Original Plaintext
  - Decryption $D$
  - Key Source
  - $M$
  - $K_d$
Secret key cryptography
- The encryption key and decryption key are the same
- Example:
  - Data encryption standard (DES) has been widely used for over 20 years
Secret key cryptography
- The receiver’s keys must be distributed in secret ⇒ *key distribution problem*
Public key cryptography
- Different key for encryption and decryption
  - Public-key and private-key
- Key distribution problem is solved
- Mainly used for key exchange & digital signatures
Public key cryptography
One-way function

- Basis of public key cryptosystems
- Easier to compute a function in one direction than in the reverse direction
- “Trap door” can make the inverse computation easy ⇒ private key

Example:
- Factorization of a integer $n$ which is the product of two large prime numbers
- If one of the prime is known, the factorization is easy

(One-way function of RSA)
Discrete Logarithm Problem (DLP)

- One-way function of difficult to find a logarithm in a group

- Definition:
  - For \((g, y \in G)\), easy to calculate \(g^x = y\) (any finite cyclic group \(G\) can be used)
  - DLP: compute \(x\) given \(g\) and \(y\) (difficult)
RSA

- Most commonly used public key cryptosystem
- Based on the problem of factorization $n$ which is the product of two large primes
Advantages of ECC over RSA

- **Smaller key size** for equivalent security
  - Higher security per bit
  - Leads to faster implementations i.e. higher security for the same amount of computation
- Thought to be **more secure** (as of Jan 2008, largest ECC and RSA systems broken to date are 109b & 663b respectively)
Cryptography

› RSA vs. ECC

COMPARISON OF SECURITY LEVELS
ECC and RSA

Time to Break Key (MIPS Years)

Key Size (Bits)

ECC

RSA & DSA

From Certicom Corp.
Key exchange protocol

- Public key cryptosystem solves the key distribution problem
- Transmits a common key securely
Diffie-Hellman key exchange protocol

- Suppose Alice and Bob want to agree on a common key $k$

Public parameters: $p$ and $g$

Alice generates a secret random integer $a$ and sends the point $(g^a \mod p)$ to Bob

Bob generates a secret random integer $b$ and sends the point $(g^b \mod p)$ to Alice

Alice and Bob can both compute the key, $k$

$$k = (g^a)^b = (g^b)^a = g^{ab}$$

- An adversary only knows $p$, $g$, $g^a$, $g^b$
  - To determine $k$ she would have to solve the discrete logarithm problem
Digital signature
- Digital form of traditional signature
- Definition:
  - Signer encrypts the document with his private key
  - Decrypts the signed document with signer’s public key to verify the signature
- Main functions:
  - Signer cannot deny the signature
  - The signature is not reusable
Secret key vs. Public key cryptography

<table>
<thead>
<tr>
<th></th>
<th>Secret key</th>
<th>Public key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>Fast</td>
<td>Slow</td>
</tr>
<tr>
<td>Key distribution problem</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

- In practice, a hybrid method is used
  - Combined of secret and public key cryptosystems
  - Used public key cryptosystem to perform key exchange to make a common key which acts as the key of secret key cryptosystem afterwards
Hybrid cryptosystem

Alice

Random generate a key

Encryption $E_p$

Encryption $E_s$

Secret Plaintext

Insecure channel

Bob

Key generated by Alice

Decryption $D_p$

Key pair $(K_{b\text{ pri}}, K_{b\text{ pub}})$

Decryption $D_s$

Original Plaintext

Public key cryptosystem

Secret key cryptosystem
Overview

› Motivation
› Introduction to cryptography
› Mathematical Background
› Introduction to elliptic curves
› Previous Works
› Elliptic curve cryptographic processor (ECP)
› Result
› Conclusion
Operations of ECC are based on finite fields

Field
- Consists of a set of elements $F$ together with two binary operations
  - $+$ (addition), and
  - $\times$ (multiplication)
- Finite number of elements ($\#F$) in $F$
  $\Rightarrow$ finite field
Finite field $F_{2^n}$

- $\#F = 2^n$
- Elements can be represented as $n$-bit binary number and are in term of a basis
- Two kinds of bases
  - Polynomial basis
  - Normal basis
- The elliptic curve processor is in **optimal normal basis**
  - Normal basis is more suitable for hardware implementation
  - Optimal normal basis gives a minimum complexity
Normal basis

- Any element, $E$ in the $F_{2^n}$ can be written in terms of a normal basis

where $a_i \in F_2, \beta \in F_{2^n}$

- It is a $n$-bit binary number

- Efficient for hardware implementation

$$A = \sum_{i=0}^{n-1} a_i \beta^{2^i}$$
Normal Basis Operations

› Addition

\[ A + B = \left( \sum_{i=0}^{n-1} a_i \beta^{2^i} \right) + \left( \sum_{j=0}^{n-1} b_j \beta^{2^j} \right) = \sum_{i=0}^{n-1} (a_i + b_i) \beta^{2^i} \]

- Bit-wise exclusive-OR (XOR)

› Squaring

\[ A^2 = \left( \sum_{i=0}^{n-1} a_i \beta^{2^i} \right)^2 = \sum_{i=0}^{n-1} a_i \left( \beta^{2^i} \right)^2 = \sum_{i=0}^{n-1} a_i \beta^{2^{i+1}} = \sum_{i=0}^{n-1} a_{i-1} \beta^{2^i} \]

\[ \left( \beta^{2^{n-1}} \right)^2 = \beta^{2^n} = \beta \]

- Rotate left (ROTL) among the coefficients \( a_i \) of \( \beta \)
\( C = A \times B = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j \beta^{2i} \beta^{2j} = \sum_{i=0}^{n-1} c_i \beta^{2i} \)

- Refer to thesis for the derivation of \( \lambda_{ijk} \).
- The multiplication is defined as a cyclic shift to \( \lambda_{ij0} \), since

\[
c_k = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \lambda_{i-k,j-k,0} a_i b_j = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \lambda_{ij0} a_{i+k} b_{j+k}
\]
Normal Basis Operations

› Multiplication

- Minimum number of non-zero terms ($2n-1$) in $\lambda$ matrix

  $\Rightarrow$ Optimal Normal Basis (ONB)

- For $n=5$,

- Since it is ONB, there are 9 non-zero terms in $\lambda$ matrix, thus 9 product terms

\[
c_k = a_k b_{1+k} + a_{1+k} b_k + a_{1+k} b_{3+k} + a_{2+k} b_{3+k} + a_{2+k} b_{4+k} + a_{3+k} b_{1+k} + a_{3+k} b_{2+k} + a_{4+k} b_{2+k} + a_{4+k} b_{4+k}
\]
Normal Basis Operations

Inversion

- Definition of inversion of $a$:

$$aa^{-1} \equiv 1 \mod n$$

- From Fermat’s Little Theorem

$$a^{2^n - 1} = 1 \implies a^{-1} = a^{2^n - 2} = \left(a^{2^{n-1} - 1}\right)^2$$

$$a^{2^{n-1} - 1} = \begin{cases} a^{\left(2^{(n-1)/2} - 1\right)\left(2^{(n-1)/2} + 1\right)} & n \text{ is odd} \\ a^{\left(2^{(n-2)/2} - 1\right)\left(2^{(n-2)/2} + 1\right)+1} & n \text{ is even} \end{cases}$$

- By decompose the power of $a$ continuously, inversion becomes a series of squarings and multiplications

$$\left\lceil \log_2(n - 1) \right\rceil + \left(\# \text{ of bits set in } n + 1 \right) - 1$$
Overview

› Motivation
› Introduction to cryptography
› Mathematical Background
› Introduction to elliptic curves
› Previous Works
› Elliptic curve cryptographic processor (ECP)
› Result
› Conclusion
Elliptic Curves

Elliptic Curve over real number (ℜ)

\[ y^2 = x^3 + a_4x + a_6 \]

with \( x, y, a_4, a_6 \in \mathbb{R} \)

Point addition and doubling on elliptic curve in affine coordinates are defined geometrically
(Also known as curve addition and curve doubling in the thesis)
Elliptic Curves over $\mathbb{R}$

- Curve addition (ESUM)

- $R = P + Q$, $P \neq Q$
- Line connecting $P$ and $Q$ intersects the curve at exactly 1 point $-R$
Elliptic Curves over $\mathbb{R}$

- Curve doubling (EDBL)

- $R = P + Q = 2P$, $\therefore P = Q$
- The tangent to the curve at $P$ is used
Curve addition and doubling in affine coordinates

\[
x_3 = \begin{cases} 
\left( \frac{y_2 - y_1}{x_2 - x_1} \right)^2 - x_1 - x_2 & P \neq Q \\
\left( \frac{3x_1^2 + a_4}{2y_1} \right)^2 - x_1 - x_2 & P = Q 
\end{cases}
\]

\[
y_3 = \begin{cases} 
(x_1 + x_3) \left( \frac{y_2 - y_1}{x_2 - x_1} \right) - y_1 & P \neq Q \\
(x_1 + x_3) \left( \frac{3x_1^2 + a_4}{2y_1} \right) - y_1 & P = Q 
\end{cases}
\]
A non-supersingular elliptic curves over finite fields $F_{2^n}$

\[ y^2 + xy = x^3 + a_2 x^2 + a_6 \]

with $a_6 \neq 0$ and $x, y, a_2, a_6 \in F_{2^n}$

- ECs over $F_{2^n}$ is practical
  - Binary natural of $F_{2^n}$
  - Less hardware
    - $F_{2^{155}}$ ECC processor 11,000 gates
    - 512b RSA 50,000 gates
      (these processors have roughly the same level of security)

Operations of ECs over $F_{2^n}$ are defined as same as ECs over $\mathbb{R}$
Elliptic Curves over $F_{2^n}$

› In affine coordinates

\[
\begin{align*}
    x_3 &= \left\{\frac{y_1 + y_2}{x_1 + x_2}\right\}^2 + \left(\frac{y_1 + y_2}{x_1 + x_2}\right) + x_1 + x_2 + a_2 \quad \text{ESUM} \\
    y_3 &= \left(\frac{x_1 + x_3}{x_1 + x_2}\right) \left(\frac{y_1 + y_2}{x_1 + x_2}\right) + x_3 + y_1 \quad \text{ESUM} \\
    y_3 &= \left(\frac{x_1 + x_3}{x_1 + x_2}\right) \left(x_1 + \frac{y_1}{x_1}\right) + x_3 + y_1 \quad \text{EDBL}
\end{align*}
\]

There are one inversion and two multiplications in both ESUM and EDBL
In projective coordinates

- A non-supersingular curve E can be equivalently viewed as the set of all points in the projective plane which satisfy

\[ y^2 z + xyz = x^3 + a_2 x^2 z^2 + a_6 z^3 \]

- By using projective coordinates, the inversion operations can be eliminated

- Any point \((a, b)\) on ECs over \(F_{2^n}\) in affine can be viewed as a 3-tuple \((a, b, 1)\) on ECs in projective

- In projective coordinates, \((tx, ty, tz) \equiv (x, y, z)\), with \(t \neq 0\)
Conversion between affine and projective

- From affine to projective

\[ A(x, y) = A'(x, y, 1) \]

- From projective to affine

\[ P'(a, b, c) = P'\left(\frac{a}{c}, \frac{b}{c}, 1\right) = P\left(\frac{a}{c}, \frac{b}{c}\right) \quad \text{for} \ c \neq 0 \]
Operations in projective coordinates

- **ESUM**

\[
x_3 = AD \\
y_3 = CD + A^2(Bx_1 + Ay_1) \\
z_3 = A^3z_1 \\
A = (x_2z_1 + x_1), B = (y_2z_1 + y_1) \\
C = A + B, D = A^2(A + a_2z_1) + z_1BC
\]

13 multiplications but no inversion

- **EDBL**

\[
x_3 = AB \\
y_3 = x_1^4A + B(x_1^2 + y_1z_1 + A) \\
z_3 = A^3 \\
A = x_1z_1, B = a_6z_1^4 + x_1^4
\]

7 multiplications but no inversion
### Affine vs. Projective

<table>
<thead>
<tr>
<th>Operation</th>
<th>Affine</th>
<th>Projective</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field Multiplication</td>
<td>2</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>Field Inversion</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Elliptic Curves over $F_{2^n}$

Curve multiplication (EMUL)
- Definition:

$$Q = cP = P + P + \cdots + P$$ \text{ where } P, Q \text{ are on ECs}

- The “doubling and add” algorithm
- Affine coordinates

INPUT: $P$ on EC and $c \in F_{2^n}$
OUTPUT: $Q = cP$

1. $c = \sum_{i=0}^{n-1} b_i 2^i, b_i \in \{0,1\}$ and $b_k$ is the MSB set($k < n$)
2. Set $Q \leftarrow P$
3. For $i$ from $k - 1$ downto 0
   3.1 Set $Q \leftarrow Q + Q$ (Affine EDBL)
   3.2 If $b_i = 1$ then
      3.2.1 Set $Q \leftarrow Q + P$ (Affine ESUM)
4. Return $Q$

Inversion operations occur
Elliptic Curves over $F_{2^n}$

Curve multiplication (EMUL)
- The “doubling and add” algorithm
- Projective coordinates

INPUT: $P$ on EC and $c \in F_{2^n}$
OUTPUT: $Q = cP$

1. $c = \sum_{i=0}^{n-1} b_i 2^i$, $b_i \in \{0,1\}$ and $b_{k-1}$ is the MSB set ($k \leq n$)
2. Convert $P$ to projective $P'$
3. Set $Q' \leftarrow P'$
4. For $i$ from $k-2$ downto 0
   3.1 Set $Q' \leftarrow Q' + Q'$ (Projective EDBL)
   3.2 If $b_i = 1$ then
       3.2.1 Set $Q' \leftarrow Q' + P'$ (Projective ESUM)
5. Convert $Q'$ to affine $Q$
6. Return $Q$

- Number of point additions/doublings required is $k + \left(\# \text{ of bits set in } c\right) - 2$
Elliptic Curve Operations

Field Operations
- Addition
- Squaring
- Multiplication
- Inversion

Curve Operations
- Curve addition (ESUM)
- Curve doubling (EDBL)
- Curve multiplication (EMUL)

Basic operation of an EC processor is EMUL
Elliptic curve discrete logarithm Problem (ECDLP)

- Definition:
  - For \((Y \in G)\), easy to calculate \(Q = xY\) (\(G\) is elliptic curve group)
  - ECDLP: compute \(x\) given \(Q\) and \(Y\) (difficult)

- ECDLP is particularly difficult if the underlying group is based on points on an elliptic curve
  - No sub-exponential time algorithm known which can solve the ECDLP
  - Index calculus methods offer sub-exponential time algorithms for some other groups but not elliptic curve group
Elliptic Curve Cryptography

- e.g. Alice wants to send a message $M$ to Bob secretly

Public parameters: elliptic curve $E$, a point $Q$ on $E$
Public-key of Bob: $(P,Q)$ where $P=xQ$, $P,Q$ on $E$
Private-key of Bob: $x$

Alice: generates random integer $z$

$A(x_a,y_a) = zQ$ and $T(x_t,y_t) = zP$
$B(x_b,y_b) = (x_t,x_m,y_t,y_m)$ where $M$ is embedded to $E$ giving $(x_m,y_m)$
ciphertext $(A,B)$ is sent to Bob

Bob: receives $(U,V)$, $U=(x_u,y_u)$ and $V=(x_v,y_v)$

$xU=xzQ=zP=T(x_t,y_t)$

$\Rightarrow x_m = \frac{x_v}{x_t}, y_m = \frac{y_v}{y_t}$
Elliptic curve Diffie-Hellman key exchange (ECDH)

- Suppose Alice and Bob wish to agree on a common key $k$

Public parameters: elliptic curve $E$ and point $P$ on $E$

Alice generates a secret random integer $c_A$ and sends the point $(c_A P)$ to Bob

Bob generates a secret random integer $c_B$ and sends the point $(c_B P)$ to Alice

Alice and Bob can both compute the key, $k$

$$k = c_A (c_B P) = c_B (c_A P)$$
Overview

› Motivation
› Introduction to cryptography
› Mathematical Background
› Introduction to elliptic curves
› Previous Work
› Elliptic curve cryptographic processor (ECP)
› Result
› Conclusion
Previous Designs

› Hardware implementation

- Field processor
  - A field processor over $F_{2^{155}}$ [AMV93]
    - 11,000 gates, 40 MHz
  - Improved multipliers are reported in [OP99,Gro01]
- Curve processor
  - A FPGA-based EC processor over $F_{(2^n)^m}$ was developed [Ros98b]
  - A scalable, FPGA-based EC processor over $F_p$ was reported in [OP01]

› Software implementation

- Full ECC package with many algorithmic optimizations [Ros98a]
  - Used to evaluate the performance and correctness of the design

High bandwidth interface is required to supply the processor with its data and the field operations are restricted to field of $\text{fix } n$
Overview

› Motivation
› Introduction to cryptography
› Mathematical Background
› Introduction to elliptic curves
› Previous Works
› Elliptic curve cryptographic processor (ECP)
› Result
› Conclusion
Design tested on Annapolis Micro Systems Wildstar Board

- PCI interface

- Consists of
  - 3 Processing elements, PEs
    (Xilinx Virtex FPGA XCV1000-6)
  - Each PE has 128-kbits of BlockRAM and 6144 CLBs (12288 slices) which is equivalent to more than one million system gates
Elliptic Curve Processor

Architecture

- ALU
- Register file
- Control
- Microcode register
Datapath

- Microcode Sequencer
  - Fetch instruction from microcode register
  - Microcode controls datapath (register file, ALU etc)
  - Update PC

- Register file
  - 16 × $n$-bit dual ported registers implemented using Xilinx Virtex distributed memory

- ALU
  - Field addition (XOR)
  - Field squaring (rotate left)
  - Field multiplication
Multiplication implementation

\[ c_k = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \lambda_{ij}^0 a_{i+k} b_{j+k} \]

\[ \Rightarrow c_k = \sum_{j=0}^{n-1} b_{j+k} \sum_{i=0}^{n-1} \lambda_{ij}^0 a_{i+k} \]

\[ \Rightarrow F(k) = b_{2k} \sum_{i=0}^{n-1} \lambda_{ik}^0 a_{i+k} \]

- It defines the connection between register A,B,C
- Property of an ONB:
  - \( a_i \) has a max. fanout of 4
  - High speed
> Multiplication Implementation

\[ c_k = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \lambda_{ij0} a_{i+k} b_{j+k} \]

\[ \Rightarrow c_k = \sum_{j=0}^{n-1} b_{j+k} \sum_{i=0}^{n-1} \lambda_{ij0} a_{i+k} \]

\[ \Rightarrow F(k) = b_{2k} \sum_{i=0}^{n-1} \lambda_{ik0} a_{i+k} \]

- Cyclic shift on A,B,C in each cycle
› Multiplication Implementation
  
  - Structure of $b_k$

- D-type flip flop
- Cyclic shift on register A and B
Multiplication Implementation

- Structure of $c_k$

Each cell calculate:

\[ b_{j+k} \sum_{i=0}^{n-1} \lambda_{ij} a_{i+k} \]

After $n$ cycles:

\[ c_k = \sum_{j=0}^{n-1} b_{j+k} \sum_{i=0}^{n-1} \lambda_{ij} a_{i+k} \]
› Multiplication implementation

- E.g $n=5$

- From multiplication table

$$F(k) = b_{2k} \sum_{i=0}^{\lambda_{ijk}} a_{i+k}$$

and

$$F(0) = b_0(a_1),$$
$$F(1) = b_2(a_1 + a_4),$$
$$F(2) = b_4(a_0 + a_1),$$
$$F(3) = b_1(a_4 + a_0),$$
$$F(4) = b_3(a_1 + a_3)$$

$$c_k = b_k (a_{k+1}) + b_{k+1} (a_k + a_{k+3}) + b_{k+2} (a_{k+3} + a_{k+4})$$
$$+ b_{k+3} (a_{k+1} + a_{k+2}) + b_{k+4} (a_{k+2} + a_{k+4})$$

which is the same as the equation shown in previous slide
Normal Basis Operations

› Multiplication

- Minimum number of non-zero terms \((2n-1)\) in \(\lambda\) matrix
  \[ \Rightarrow \text{Optimal Normal Basis (ONB)} \]
- For \(n=5\),

\[
\begin{align*}
    c_k &= a_k b_{1+k} + a_{1+k} b_k + a_{1+k} b_{3+k} + a_{2+k} b_{3+k} + a_{2+k} b_{4+k} + \\
        &a_{3+k} b_{1+k} + a_{3+k} b_{2+k} + a_{4+k} b_{2+k} + a_{4+k} b_{4+k} \\
\end{align*}
\]
Parallel multiplier

- Field multiplication process can be parallelized easily
- Instead of calculating $c_k$ by one set of inputs from register A and B, we determine $c_k$ by $p$ sets of inputs

- Increase the parallelism by a factor $p$, the multiplier logic can be duplicated $p$ times and reduce the number of cycle to $\left\lfloor \frac{n}{p} \right\rfloor + 2$
› Stores the parameters used in ECC

- A $16 \times n$-bit dual ported registers implemented using Xilinx Virtex distributed RAM
- Updates at the same time
- Outputs two $n$-bit operands simultaneously
For performing high level elliptic curve multiplication
- Stored in Virtex BlockRAMs so control logic does not use FPGA logic resources
- Can be changed without re-compilation of the processor
- Algorithmic optimizations can be performed entirely in microcode
- 16-bit wide microcode

<table>
<thead>
<tr>
<th>Operation</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP</td>
<td>No operation</td>
</tr>
<tr>
<td>XOR r1,r2</td>
<td>r0 ← r1 ⊕ r2</td>
</tr>
<tr>
<td>ROTL r1,r2</td>
<td>r2 ← r1 rotate left</td>
</tr>
<tr>
<td>SHFR r1,r2</td>
<td>r2 ← r1 right shift</td>
</tr>
<tr>
<td>MUL r1,r2</td>
<td>r0 ← r1 × r2</td>
</tr>
<tr>
<td>TFR r1,r2</td>
<td>r2 ← r1</td>
</tr>
<tr>
<td>JKZ,JCZ,JMP</td>
<td>PC ← new address</td>
</tr>
</tbody>
</table>
Parameterized Module Generator

- ECP with different size
  - Generates the codes of elliptic curve processors for any $n$ (key size) with ONB
  - Processors with different strength of security can be generated to fit different requirements

- Parallelized multiplier
  - Generates processor with different level of parallelism of multiplier
  - High resource utilization
  - Different speed/area tradeoffs can be made
Microcode Toolkit
- Facilitates the microcode development
- Simulator and debugger
  - Supports all microcode instructions
  - Supports setting breakpoints
  - Supports single stepping
- Assembler
  - Two pass symbolic assembler
  - Converts symbolic input to binary microcode
› Bitstream reconfiguration
- Modifies the bitstream to change the contents of the register file
- Initializes the register file
- Removes the circuitry to download the parameters from host PC to FPGA
Results

- ECP was generated using parameterized module generator
- Synthesize and implemented by Synopsys FPGA Express 3.4 and Xilinx Foundation 3.2i respectively
- ECP with serial multiplier (XCV1000 has 12288 slices)

<table>
<thead>
<tr>
<th>$n$</th>
<th># of slices</th>
<th>Reported Max. frequency (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>113</td>
<td>1410</td>
<td>31</td>
</tr>
<tr>
<td>155</td>
<td>1868</td>
<td>30</td>
</tr>
<tr>
<td>173</td>
<td>2148</td>
<td>28</td>
</tr>
<tr>
<td>281</td>
<td>3315</td>
<td>26</td>
</tr>
<tr>
<td>371</td>
<td>4247</td>
<td>22</td>
</tr>
<tr>
<td>473</td>
<td>5264</td>
<td>18</td>
</tr>
</tbody>
</table>
Resource requirements are linear with $n$
Results

Compared with software
(SUN Enterprise E4500 with UltraSPARC-II 400 MHz with 8 GBytes of RAM)

<table>
<thead>
<tr>
<th>$n$</th>
<th>SW time ($ms$)</th>
<th>HW time ($ms$)</th>
<th>Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>113</td>
<td>27.6</td>
<td>4.3</td>
<td>6</td>
</tr>
<tr>
<td>155</td>
<td>63.2</td>
<td>8.3</td>
<td>8</td>
</tr>
<tr>
<td>173</td>
<td>86.6</td>
<td>11.1</td>
<td>8</td>
</tr>
</tbody>
</table>

Projective vs. Affine coordinates

<table>
<thead>
<tr>
<th>$n$</th>
<th># of cycles (affine)</th>
<th># of cycles (projective)</th>
<th>HW time affine ($ms$)</th>
<th>HW time projective ($ms$)</th>
<th>Projective: Affine</th>
</tr>
</thead>
<tbody>
<tr>
<td>113</td>
<td>148581</td>
<td>134484</td>
<td>4.8</td>
<td>4.3</td>
<td>0.9</td>
</tr>
<tr>
<td>155</td>
<td>324717</td>
<td>249879</td>
<td>10.8</td>
<td>8.3</td>
<td>0.77</td>
</tr>
<tr>
<td>173</td>
<td>402926</td>
<td>310043</td>
<td>14.4</td>
<td>11.1</td>
<td>0.77</td>
</tr>
</tbody>
</table>
### Dynamic instruction counts and frequencies

<table>
<thead>
<tr>
<th></th>
<th>Projective</th>
<th>Affine</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>113</td>
<td>113</td>
</tr>
<tr>
<td>NOP</td>
<td>291</td>
<td>291</td>
</tr>
<tr>
<td>XOR</td>
<td>616</td>
<td>784</td>
</tr>
<tr>
<td>MUL</td>
<td>128820</td>
<td>127680</td>
</tr>
<tr>
<td></td>
<td>(95.79%)</td>
<td>(85.93%)</td>
</tr>
<tr>
<td>ROTL</td>
<td>673</td>
<td>12825</td>
</tr>
<tr>
<td>TFR</td>
<td>3625</td>
<td>5432</td>
</tr>
<tr>
<td>JKZ</td>
<td>113</td>
<td>113</td>
</tr>
<tr>
<td>JIZ</td>
<td>111</td>
<td>111</td>
</tr>
<tr>
<td>JMP</td>
<td>111</td>
<td>111</td>
</tr>
<tr>
<td>SHFR</td>
<td>123</td>
<td>1233</td>
</tr>
<tr>
<td>Total</td>
<td>134484</td>
<td>148581</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>249879</th>
<th>324717</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>310043</td>
<td>402926</td>
</tr>
</tbody>
</table>
ECP with parallel multiplier

- \( n = 113 \)

<table>
<thead>
<tr>
<th>( p )-way paralleled</th>
<th>slices</th>
<th>cycles</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1410</td>
<td>134484</td>
<td>4.3</td>
</tr>
<tr>
<td>2</td>
<td>1860</td>
<td>71204</td>
<td>2.6</td>
</tr>
<tr>
<td>4</td>
<td>1970</td>
<td>39564</td>
<td>1.7</td>
</tr>
<tr>
<td>6</td>
<td>2076</td>
<td>28264</td>
<td>1.2</td>
</tr>
<tr>
<td>8</td>
<td>2182</td>
<td>23744</td>
<td>1.06</td>
</tr>
<tr>
<td>10</td>
<td>2300</td>
<td>20354</td>
<td>0.93</td>
</tr>
<tr>
<td>12</td>
<td>2434</td>
<td>18094</td>
<td>0.89</td>
</tr>
<tr>
<td>14</td>
<td>2515</td>
<td>16964</td>
<td>0.84</td>
</tr>
<tr>
<td>16</td>
<td>2614</td>
<td>15833</td>
<td>0.81</td>
</tr>
</tbody>
</table>
ECP with parallel multiplier

- $n = 113$

Normalized execution time for one EMUL using $p$-way parallel multiplier

$$\text{Speedup} = \frac{\text{Time}(p = 1)}{\text{Time}(p > 1)}$$
An elliptic curve processor over $F_{2^n}$ using an optimal normal basis was developed.

- To perform a scalar multiplication on elliptic curve which is the central operation of ECC
- Curve multiplication over $F_{2^n}$ with $n=113, 155$ and $173$ was successfully tested on the hardware with reported frequencies $31, 30$ and $28$ MHz respectively
- Curve multiplication in projective coordinates has $10\%$ improvement over affine ($n=113$) and $23\%$ for ($n=155, 173$)
- Projective implementation with serial multiplier gave 6-8 times improvement over optimized software implementation
- 16-way parallel multiplier ($n=113$) gave 5 times improvement over serial multiplier and linear improvement when degree of parallelism $\leq 6$
- More flexible since the design used reconfigurable nature of FPGA
- A parameterized module generator can generate field processors using an ONB for arbitrary $n$ using multiplier with different speed/area tradeoffs
- Microcode approach can implement curve operation on top of field operations
- High I/O requirement interface is not needed
- Algorithmic optimizations can be done in microcode
References

Paper


Thesis

› Make sure you understand how to split a design into datapath + control

› Control can be implemented in different ways including hardwired and microcoded, what are their respective advantages?

› What does a module generator do? How was it used advantageously in the ecc processor?