High-Speed Elliptic Curve Cryptography Accelerator for Koblitz Curves

Kimmo Järvinen Jorma Skyttä

Helsinki University of Technology
Department of Signal Processing and Acoustics
Otakaari 5A, FIN-02150, Finland
{Kimmo.Jarvinen,Jorma.Skytta}@tkk.fi

April 14, 2008

Outline

1. Preliminaries
   - Elliptic Curve Cryptography
   - Koblitz Curves
   - Window Method and Multiple Point Multiplication
2. FPGA Implementation
   - Design Specifications
   - Architecture of the Implementation
3. Results, Comparisons and Conclusions
   - Results
   - Comparisons
   - Conclusions and Future Work

Introduction to Elliptic Curve Cryptography

- Public-key cryptography method which uses a group of points on an elliptic curve, E, defined over a finite field, $\mathbb{F}_q$
- Faster and shorter keys than, e.g., RSA

Introduction to Elliptic Curve Cryptography

- Public-key cryptography method which uses a group of points on an elliptic curve, E, defined over a finite field, $\mathbb{F}_q$
- Faster and shorter keys than, e.g., RSA

Elliptic Curve Point Multiplication

$$Q = kP$$

where k is a positive integer and $P = (x, y)$ is a point on E

- Computed with point additions, $P_1 + P_2$, and point doublings, $2P_1$

Point Multiplication on Koblitz Curves

Koblitz curves

Frobenius maps, $\phi(P_1)$, instead of point doublings
$\Rightarrow$ faster computation

- k must be converter to τ-adic representation

Point Multiplication on Koblitz Curves

Koblitz curves

Frobenius maps, $\phi(P_1)$, instead of point doublings
$\Rightarrow$ faster computation

- k must be converter to τ-adic representation

Point multiplication

- Frobenius map for all bits of k
- Point addition if the bit is 1

Point Multiplication on Koblitz Curves

Koblitz curves

Frobenius maps, $\phi(P_1)$, instead of point doublings
$\Rightarrow$ faster computation

- k must be converter to τ-adic representation

Point multiplication

- Frobenius map for all bits of k
- Point addition if the bit is 1

Example

```
10011100010011110001
A  AAA   A  AAAA  A   10
```

Point Multiplication on Koblitz Curves

Koblitz curves

Frobenius maps, $\phi(P_1)$, instead of point doublings
$\Rightarrow$ faster computation

- k must be converter to τ-adic representation

Point multiplication

- Frobenius map for all bits of k
- Point addition if the bit is `1`, point subtraction if $\bar{1}$

Example

```
1001110001001111001
A  AAA   A  AAAA  A   10
```

```
10100̄1000101000̄1001
A A  S   A A   S  A   7
```

Window Method

Windowing further reduces the number of point additions

Window Method

Windowing further reduces the number of point additions

Idea of windowing

Instead of computing `AAA` several times:

- Precompute `AAA`
- Use the precomputed value every time for the string `111`!

We precompute values for the strings $\texttt{10}\bar{\texttt{1}}$, `101`, and `1001`

Window Method

Windowing further **reduces the number of point additions**

Idea of windowing

Instead of computing `AAA` several times:

- Precompute `AAA`
- Use the precomputed value every time for the string `111`!

We precompute values for the strings $10\bar{1}$, 101, and 1001

Example

τNAF

$\underline{10\bar{1}}0\underline{1}000\underline{1001}00\underline{101}00\underline{\bar{1}0\bar{1}}$

`A S A   S  A  A A  S S` 9

Width-4 τNAF

$3010000000\bar{7}0000500005\bar{5}$

`A A     S   A   S` 5

Precomputations: 3

Multiple Point Multiplication

Sum of n point multiplications

$$Q = k^{(1)}P^{(1)} + k^{(2)}P^{(2)} + \ldots + k^{(n)}P^{(n)}$$

Multiple Point Multiplication

Sum of n point multiplications

$$Q = k^{(1)}P^{(1)} + k^{(2)}P^{(2)} + \ldots + k^{(n)}P^{(n)}$$

Efficient computation with Shamir's trick

- Precompute all combinations of $P^{(1)} \ldots P^{(n)}$, e.g. $P^{(1)} + P^{(2)}$ and $P^{(1)} - P^{(2)}$
- Interpret $k^{(1)} \ldots k^{(n)}$ as n-row table, e.g. $\begin{smallmatrix} 100100\bar{1}01001010 \\ 10\bar{1}0010010100\bar{1}0 \end{smallmatrix}$
- Frobenius map for all columns
- Point addition with precomputed point if column is nonzero

Multiple Point Multiplication

Sum of n point multiplications

$$Q = k^{(1)}P^{(1)} + k^{(2)}P^{(2)} + \ldots + k^{(n)}P^{(n)}$$

Efficient computation with Shamir's trick

- Precompute all combinations of $P^{(1)} \ldots P^{(n)}$, e.g. $P^{(1)} + P^{(2)}$ and $P^{(1)} - P^{(2)}$
- Interpret $k^{(1)} \ldots k^{(n)}$ as n-row table, e.g. $\begin{matrix} 100100\bar{1}01001010 \\ 10\bar{1}00100101001\bar{1}0 \end{matrix}$
- Frobenius map for all columns
- Point addition with precomputed point if column is nonzero

τ-adic joint sparse form (τJSF)

τJSF maximizes the number of zero columns in the table

Algorithmic Comparison

Window method

Input: Integer k, point P
Output: Result point $Q = kP$

$\langle k_{\ell-1} \ldots k_0 \rangle \leftarrow w\text{-}\tau\mathrm{NAF}(k)$
$P_1, P_3, \ldots, P_{2^{w-1}-1} \leftarrow \mathrm{PreC}(P)$
$Q \leftarrow \mathcal{O}$
for $i = \ell - 1$ **down to** 0 **do**
  $Q \leftarrow \phi(Q)$
  if $k_i \neq 0$ **then**
    $Q \leftarrow Q + \mathrm{sign}(k_i) P_{|k_i|}$
  end if
end for
$Q \leftarrow \mathrm{xy}(Q)$

Multiple point multiplication

Input: n integers $k^{(i)}$, n points $P^{(i)}$
Output: Result point $Q = \sum_{i=1}^{n} k^{(i)} P^{(i)}$

$\langle k_{\ell-1} \ldots k_0 \rangle \leftarrow \tau\mathrm{JSF}(k^{(1)}, \ldots, k^{(n)})$
$P_1, P_2, \ldots, P_{(3^n-1)/2} \leftarrow \mathrm{PreC}(P^{(1)}, \ldots, P^{(n)})$
$Q \leftarrow \mathcal{O}$
for $i = \ell - 1$ **down to** 0 **do**
  $Q \leftarrow \phi(Q)$
  if $k_i \neq 0$ **then**
    $Q \leftarrow Q + \mathrm{sign}(k_i) P_{|k_i|}$
  end if
end for
$Q \leftarrow \mathrm{xy}(Q)$

Algorithmic Comparison

Window method

Input: Integer k, point P
Output: Result point $Q = kP$

$\langle k_{\ell-1} \ldots k_0 \rangle \leftarrow w\text{-}\tau\text{NAF}(k)$
$P_1, P_3, \ldots, P_{2^{w-1}-1} \leftarrow \text{PreC}(P)$
$Q \leftarrow \mathcal{O}$
for $i = \ell - 1$ **down to** 0 **do**
  $Q \leftarrow \phi(Q)$
  if $k_i \neq 0$ **then**
    $Q \leftarrow Q + \text{sign}(k_i) P_{|k_i|}$
  end if
end for
$Q \leftarrow \text{xy}(Q)$

Multiple point multiplication

Input: n integers $k^{(i)}$, n points $P^{(i)}$
Output: Result point $Q = \sum_{i=1}^{n} k^{(i)} P^{(i)}$

$\langle k_{\ell-1} \ldots k_0 \rangle \leftarrow \tau\text{JSF}(k^{(1)}, \ldots, k^{(n)})$
$P_1, P_2, \ldots, P_{(3^n-1)/2} \leftarrow \text{PreC}(P^{(1)}, \ldots, P^{(n)})$
$Q \leftarrow \mathcal{O}$
for $i = \ell - 1$ **down to** 0 **do**
  $Q \leftarrow \phi(Q)$
  if $k_i \neq 0$ **then**
    $Q \leftarrow Q + \text{sign}(k_i) P_{|k_i|}$
  end if
end for
$Q \leftarrow \text{xy}(Q)$

Algorithmic Comparison

Window method

Input: Integer k, point P
Output: Result point $Q = kP$

$\langle k_{\ell-1} \ldots k_0 \rangle \leftarrow w\text{-}\tau\text{NAF}(k)$
$P_1, P_3, \ldots, P_{2^{w-1}-1} \leftarrow \text{PreC}(P)$
$Q \leftarrow \mathcal{O}$
for $i = \ell - 1$ **down to** 0 **do**
  $Q \leftarrow \phi(Q)$
  if $k_i \neq 0$ **then**
    $Q \leftarrow Q + \text{sign}(k_i) P_{|k_i|}$
  end if
end for
$Q \leftarrow \text{xy}(Q)$

Multiple point multiplication

Input: n integers $k^{(i)}$, n points $P^{(i)}$
Output: Result point $Q = \sum_{i=1}^{n} k^{(i)} P^{(i)}$

$\langle k_{\ell-1} \ldots k_0 \rangle \leftarrow \tau\text{JSF}(k^{(1)}, \ldots, k^{(n)})$
$P_1, P_2, \ldots, P_{(3^n-1)/2} \leftarrow \text{PreC}(P^{(1)}, \ldots, P^{(n)})$
$Q \leftarrow \mathcal{O}$
for $i = \ell - 1$ **down to** 0 **do**
  $Q \leftarrow \phi(Q)$
  if $k_i \neq 0$ **then**
    $Q \leftarrow Q + \text{sign}(k_i) P_{|k_i|}$
  end if
end for
$Q \leftarrow \text{xy}(Q)$

Algorithmic Comparison

Window method

Input: Integer k, point P
Output: Result point $Q = kP$

$\langle k_{\ell-1} \dots k_0 \rangle \leftarrow w\text{-}\tau\text{NAF}(k)$
$P_1, P_3, \dots, P_{2^{w-1}-1} \leftarrow \text{PreC}(P)$
$Q \leftarrow \mathcal{O}$
for $i = \ell - 1$ **down to** 0 **do**
  $Q \leftarrow \phi(Q)$
  if $k_i \neq 0$ **then**
    $Q \leftarrow Q + \text{sign}(k_i) P_{|k_i|}$
  end if
end for
$Q \leftarrow \text{xy}(Q)$

Multiple point multiplication

Input: n integers $k^{(i)}$, n points $P^{(i)}$
Output: Result point $Q = \sum_{i=1}^{n} k^{(i)} P^{(i)}$

$\langle k_{\ell-1} \dots k_0 \rangle \leftarrow \tau\text{JSF}(k^{(1)}, \dots, k^{(n)})$
$P_1, P_2, \dots, P_{(3^n-1)/2} \leftarrow \text{PreC}(P^{(1)}, \dots, P^{(n)})$
$Q \leftarrow \mathcal{O}$
for $i = \ell - 1$ **down to** 0 **do**
  $Q \leftarrow \phi(Q)$
  if $k_i \neq 0$ **then**
    $Q \leftarrow Q + \text{sign}(k_i) P_{|k_i|}$
  end if
end for
$Q \leftarrow \text{xy}(Q)$

Algorithmic Comparison

Window method

Input: Integer k, point P
Output: Result point $Q = kP$

$\langle k_{\ell-1} \ldots k_0 \rangle \leftarrow w\text{-}\tau\mathrm{NAF}(k)$
$P_1, P_3, \ldots, P_{2^{w-1}-1} \leftarrow \mathrm{PreC}(P)$
$Q \leftarrow \mathcal{O}$
for $i = \ell - 1$ **down to** 0 **do**
  $Q \leftarrow \phi(Q)$
  if $k_i \neq 0$ **then**
    $Q \leftarrow Q + \mathrm{sign}(k_i) P_{|k_i|}$
  end if
end for
$Q \leftarrow \mathrm{xy}(Q)$

Multiple point multiplication

Input: n integers $k^{(i)}$, n points $P^{(i)}$
Output: Result point $Q = \sum_{i=1}^{n} k^{(i)} P^{(i)}$

$\langle k_{\ell-1} \ldots k_0 \rangle \leftarrow \tau\mathrm{JSF}(k^{(1)}, \ldots, k^{(n)})$
$P_1, P_2, \ldots, P_{(3^n-1)/2} \leftarrow \mathrm{PreC}(P^{(1)}, \ldots, P^{(n)})$
$Q \leftarrow \mathcal{O}$
for $i = \ell - 1$ **down to** 0 **do**
  $Q \leftarrow \phi(Q)$
  if $k_i \neq 0$ **then**
    $Q \leftarrow Q + \mathrm{sign}(k_i) P_{|k_i|}$
  end if
end for
$Q \leftarrow \mathrm{xy}(Q)$

Objectives of the Implementation

Specifications

- NIST K-163, Koblitz curve
- Finite field $\mathbb{F}_{2^{163}}$ with polynomial basis
- (Multiple) point multiplications with $n = 1$, $n = 2$, and $n = 3$
- FPGAs offer combination of high-speed and flexibility
- Primary application: Proof-of-concept implementation for Packet-Level Authentication (PLA) communication scheme

Objectives of the Implementation

Specifications

- NIST K-163, Koblitz curve
- Finite field $\mathbb{F}_{2^{163}}$ with polynomial basis
- (Multiple) point multiplications with $n = 1$, $n = 2$, and $n = 3$
- FPGAs offer combination of high-speed and flexibility
- Primary application: Proof-of-concept implementation for Packet-Level Authentication (PLA) communication scheme

Design Principles

Maximize throughput and maintain low computation time

Objectives of the Implementation

Specifications

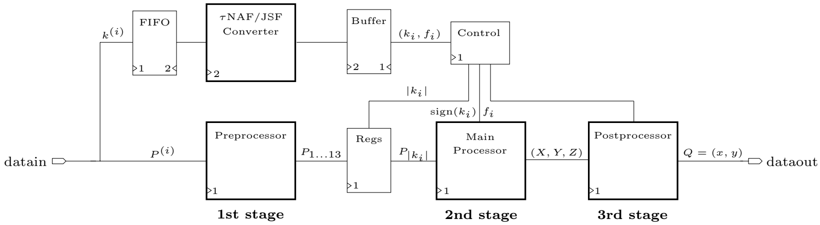
- NIST K-163, Koblitz curve
- Finite field $\mathbb{F}_{2^{163}}$ with polynomial basis
- (Multiple) point multiplications with $n = 1$, $n = 2$, and $n = 3$
- FPGAs offer combination of high-speed and flexibility
- Primary application: Proof-of-concept implementation for Packet-Level Authentication (PLA) communication scheme

Design Principles

Maximize throughput and maintain low computation time by...

1. Utilizing the common structure of the algorithms
2. Using specific processing units from our previous works

Top Level Architecture



Specialized processing units

- τNAF/JSF converter ⇒ Width-4 τNAF or 2/3-term τJSF
- Preprocessor ⇒ Precomputations
- Main processor ⇒ For loop
- Postprocessor ⇒ Coordinate conversion, xy(Q)

Main Processor: Idea

Background

- Point additions computed sequentially
- Data dependencies prevent efficient parallelization in point additions

$(X_3, Y_3, Z_3) = (X_1, Y_1, Z_1) + (x_2, y_2):$

$$A = Y_1 + y_2 Z_1^2; \quad B = X_1 + x_2 Z_1$$

$$C = BZ_1; \quad Z_3 = C^2; \quad D = x_2 Z_3$$

$$X_3 = A^2 + C(A + B^2 + aC)$$

$$Y_3 = (D + X_3)(AC + Z_3) + (y_2 + x_2)Z_3^2$$

Main Processor: Idea

Background

- Point additions computed sequentially
- Data dependencies prevent efficient parallelization in point additions

Idea

- Most operations do not need Y_1

$$(X_3, Y_3, Z_3) = (X_1, Y_1, Z_1) + (x_2, y_2):$$

$$A = Y_1 + y_2 Z_1^2; \quad B = X_1 + x_2 Z_1$$

$$C = BZ_1; \quad Z_3 = C^2; \quad D = x_2 Z_3$$

$$X_3 = A^2 + C(A + B^2 + aC)$$

$$Y_3 = (D + X_3)(AC + Z_3) + (y_2 + x_2)Z_3^2$$

Main Processor: Idea

Background

- Point additions computed sequentially
- Data dependencies prevent efficient parallelization in point additions

Idea

- Most operations do not need Y_1
- Point additions (and Frobenius maps) can be interleaved

$$(X_3, Y_3, Z_3) = (X_1, Y_1, Z_1) + (x_2, y_2):$$

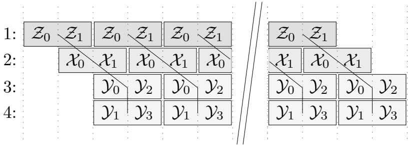
$$A = Y_1 + y_2 Z_1^2; \quad B = X_1 + x_2 Z_1$$

$$C = BZ_1; \quad Z_3 = C^2; \quad D = x_2 Z_3$$

$$X_3 = A^2 + C(A + B^2 + aC)$$

$$Y_3 = (D + X_3)(AC + Z_3) + (y_2 + x_2)Z_3^2$$

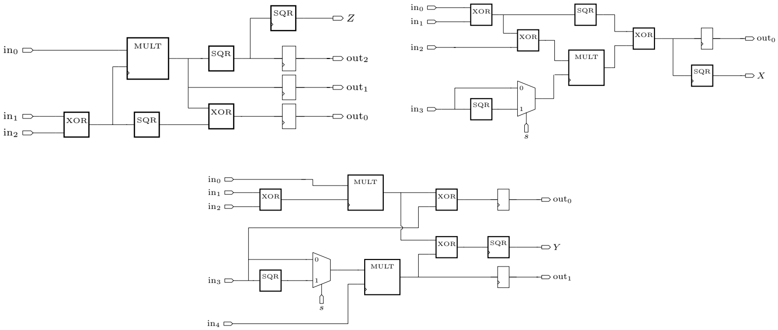
$\mathcal{Z}_{0/1}$: Computation of Z_3
$\mathcal{X}_{0/1}$: Computation of X_3
$\mathcal{Y}_{0-3}$: Computation of Y_3



Main Processor: Implementation

Implementation strategy

Design coordinate-specific processing units build around field multipliers with latencies: multiplication + 1



Up-left: *Z* unit, **Up-right:** *X* unit, **Down:** *Y* unit

Optimizations

- The design includes 6 finite field multipliers:

Preprocessor	1
Main processor	4
Postprocessor	1

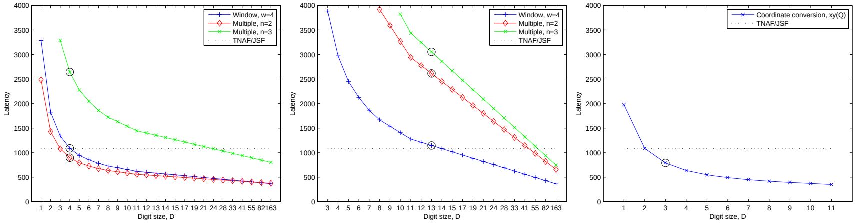
- Multiplier digit size, D, defines both latency and area

Optimizations

- The design includes 6 finite field multipliers:

Preprocessor	1
Main processor	4
Postprocessor	1

- Multiplier digit size, *D*, defines both latency and area



Left: preprocessor, **Middle:** main processor, **Right:** postprocessor

Results from Quartus II 6.0 SP1

Area consumption in Stratix II S180C3

Component	ALUTs	Regs.	ALMs	M4Ks
Converter	4,906	2,862	2,862	7
Preprocessor	2,037	1,546	1,332	14
Main processor	16,642	10,045	10,930	0
Postprocessor	2,874	2,336	1,953	0
Total	26,616	16,966	16,930	21

Results from Quartus II 6.0 SP1

Area consumption in Stratix II S180C3

Component	ALUTs	Regs.	ALMs	M4Ks
Converter	4,906	2,862	2,862	7
Preprocessor	2,037	1,546	1,332	14
Main processor	16,642	10,045	10,930	0
Postprocessor	2,874	2,336	1,953	0
Total	26,616	16,966	16,930	21

Computation time and throughput

Operation	Time (μs)	Throughput (ops)
Window, $w = 4$	16.36	161,290
Multiple, $n = 2$	24.28	70,773
Multiple, $n = 3$	35.06	60,603

Comparisons

FPGA-based implementations using NIST K-163

Ref.	n	Device	Area	μs	ops
Dimitrov	1	Vir.-II	6,494 slices + memory	35.75	27,972
Järvinen[1]	3	Str. II	67,467 ALMs + memory	114.2	166,000
Järvinen[2]	1	Str. II	13,472 ALMs + memory	25.81	49,318
Lutz	1	Vir.-E	10,017 LUTs, 1,930 FFs	75	13,333
Okada	1	F. 10K	——	45600	22
Ours	1	Str. II	16,930 ALMs, 21 M4Ks	16.36	161,290
Ours	3	Str. II	16,930 ALMs, 21 M4Ks	35.06	60,603

- Faster than other published implementations
- Only 1/4 of area compared to Järvinen[1] $\Rightarrow$ $4 \times 60,603 \approx 242,000 \Rightarrow$ Speedup 46 %

Comparisons

FPGA-based implementations using NIST K-163

Ref.	n	Device	Area	μs	ops
Dimitrov	1	Vir.-II	6,494 slices + memory	35.75	27,972
Järvinen[1]	3	Str. II	67,467 ALMs + memory	114.2	166,000
Järvinen[2]	1	Str. II	13,472 ALMs + memory	25.81	49,318
Lutz	1	Vir.-E	10,017 LUTs, 1,930 FFs	75	13,333
Okada	1	F. 10K	——	45600	22
Ours	1	Str. II	16,930 ALMs, 21 M4Ks	16.36	161,290
Ours	3	Str. II	16,930 ALMs, 21 M4Ks	35.06	60,603

- Faster than other published implementations
- Only 1/4 of area compared to Järvinen[1] $\Rightarrow$ $4 \times 60,603 \approx 242,000 \Rightarrow$ Speedup 46 %

Conclusions and Future Work

Conclusions

We showed that very high throughput and low computation time are achievable with reasonable cost in modern FPGAs by...

- Selecting the most efficient algorithms (Koblitz curves, window methods, multiple point multiplications)
- Utilizing the common structure of the algorithms
- Pipelining carefully optimized dedicated processing units

Conclusions and Future Work

Conclusions

We showed that very high throughput and low computation time are achievable with reasonable cost in modern FPGAs by...

- Selecting the most efficient algorithms (Koblitz curves, window methods, multiple point multiplications)
- Utilizing the common structure of the algorithms
- Pipelining carefully optimized dedicated processing units

Future work

We will study at least the following aspects...

- Other field sizes, faster τNAF/JSF converter, latency-area product optimizations, side-channel resistivity, etc.

Thank you.
Questions?