Scalable Unified Dual-Radix Architecture for Montgomery Multiplication in GF(P) and GF($2^n$)

Kazuyuki TANIMURA,
Ryuta NARA, Shunitsu KOHARA, Kazunori SHIMIZU,
Youhua SHI, Nozomu TOGAWA, Masao YANAGISAWA,
Tatsuo OHTSUKI

Department of Computer Science,
Waseda University
Outline

- Introduction
- Montgomery Multiplication
- Proposed Montgomery Multiplier
- Experimental Result
- Conclusion
Introduction

- Elliptic Curve Cryptography (ECC)
  - Public-key Cryptography
  - Security level
    - ECC with 160-bit key = RSA with 1024-bit key
  - Area and operation-time efficient hardware can be implemented

- Montgomery multiplication
  - Most dominant arithmetic operation

<table>
<thead>
<tr>
<th>Layer</th>
<th>Protocols</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layer-4</td>
<td>(EC-DSA)</td>
</tr>
<tr>
<td>Layer-3</td>
<td>Point multiplication</td>
</tr>
<tr>
<td>Layer-2</td>
<td>Elliptic curve addition and doubling</td>
</tr>
<tr>
<td>Layer-1</td>
<td>Finite field arithmetic</td>
</tr>
<tr>
<td></td>
<td>(ADD/SUB/XOR/MAC in GF(2^p)/MAC in GF(p))</td>
</tr>
</tbody>
</table>
Montgomery Multiplication

- **Input:** \( A, B \)
- **Output:** \( C = A \cdot B \cdot 2^{-n} \mod P \)
  - \( P \): parameter of ECC
  - \( n \): bit width of \( P \)

### Algorithm

\[ C := 0 \]

for \( i = 0 \) to \( m - 1 \)

\[ t_i := (c_0 + a_i b_0) \mod 2^k \]

\[ C := (c_i + a_i b + t_i P) / 2^k \]

if \( (C > P) \) then \( C := C - P \)
Scalable Montgomery Multiplication

- Bit width of operands
  - Varies from 160 to 256 bits depending on security levels

- Scalability
  - Divide operands by 64 bits
    - Use 64-bit $\times$ 64-bit multiplier iteratively
  - Radix-$2^{64}$ architecture
High Radix vs. Low Radix

- Radix($=2^k$)
  - Low radix: $2^1 \sim 2^3$ [2,4,6,7,8,9,10]
    - Short delay time
    - More clock cycles is required
  - High radix: $2^{16} \sim 2^{64}$ [3,5]
    - Long delay time
    - Fewer clock cycles is required

- The total operation time of high radix architecture is shorter
  - Decreasing the clock cycles affects the total operation time more than the delay time does
Galois Field

The operands of Montgomery multiplier are in...

- **GF(P): Prime field**
  - P is a 160～256-bit prime number
    - E.g. 010b+011b=101b

- **GF(2^n): Binary extension field**
  - n is a 160～256-bit number
  - Addition is defined as XOR
    - E.g. 010b+011b=001b
Unified Montgomery Multiplier

- Elliptic curve-based signature scheme EC-DSA[1] is standardized in both GF(P) and GF(2^n) fields

- Unified Montgomery Multiplier [2,5,6,7]:
  - Can compute both GF(P) and GF(2^n) numbers

Problem of Unified Architecture

- Delay time difference between $\text{GF}(P)$ and $\text{GF}(2^n)$ circuits
  - Delay of the circuit
    - $\text{GF}(P) > \text{GF}(2^n)$

![Diagram showing the carry propagation produces the delay time difference between $\text{GF}(P)$ and $\text{GF}(2^n)$ circuits.](attachment:diagram.png)
Delay time difference $\text{GF}(2^n)$ and $\text{GF}(P)$

- The larger radix, the longer delay time difference
  - The merit (short delay) of $\text{GF}(2^n)$ is ruined in unified architecture

![Graph showing the comparison between $\text{GF}(P)$ and $\text{GF}(2^n)$ delay times](image)

- Synthesized by DesignCompiler with STARC90nm process library

Delay (ns)

$k$ bits

3.7 times longer
Proposed Montgomery Multiplier

- Scalable
- Unified: GF(P) and GF(2^n)
- Dual-radix
  - GF(P): radix-2^{16} × 4 units
  - GF(2^n): radix-2^{64}
    - Applying lower radix to GF(P) enables to reduce its delay as long as GF(2^n)
    - Reduce the clock cycles in GF(P) by computing in 4 parallel
Delay time difference between $GF(P)$ and $GF(2^n)$

- Delay time of radix-$2^{16}$ multiplier in $GF(P)$ is as long as that of radix-$2^{64}$ multiplier in $GF(2^n)$

\[
\begin{array}{c}
\text{Sythesized by DesignCompiler} \\
\text{with STARC90nm process library}
\end{array}
\]
Implementation

- Described in VHDL
- Synthesized using DesignCompiler
- Library
  - STARC90nm process library

```
• MAC
• ADD
• SUB
• XOR
```

```
• MAC
• ADD
• SUB
• XOR
```
## Comparison in Operation Time

<table>
<thead>
<tr>
<th>Ref.</th>
<th>Tech.</th>
<th>Field</th>
<th>Frequency</th>
<th>Radix</th>
<th>256-bit time</th>
</tr>
</thead>
<tbody>
<tr>
<td>This work</td>
<td>90n m</td>
<td>GF(P)</td>
<td>714 MHz</td>
<td>$2^{16}$</td>
<td>0.23 $\mu$s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GF($2^n$)</td>
<td></td>
<td>$2^{64}$</td>
<td>63 ns</td>
</tr>
<tr>
<td>[5]</td>
<td>0.13 $\mu$m</td>
<td>GF(P)</td>
<td>137.7 MHz</td>
<td>$2^{64}$</td>
<td>0.36 $\mu$s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GF($2^n$)</td>
<td>510.2 MHz</td>
<td>$2^{64}$</td>
<td>88 ns</td>
</tr>
</tbody>
</table>

- Proposed Montgomery multiplier in GF(P) is 11% faster at 510.2MHz
- Dual-radix approach requires more clock cycles, but parallel architecture can cancel them out.
Comparison in Area

- Area of ALU + Controller is 29% of [5], 103% of [7]
Conclusion

- Scalable Unified Dual-Radix Montgomery Multiplier
  - Applying lower radix to GF(P) enables to reduce its delay as long as GF(2^n)
  - Reduce the clock cycles by parallel architecture in GF(P)
  - The area of logic part in proposal is almost same or smaller than other approaches.
  - Can drive both field multipliers at same frequency, which will result in shortening encryption and decryption time
Thank you

Kazuyuki TANIMURA

tanimura@yanagi.comm.waseda.ac.jp
Appendix
Previous Works

- Radix-$2^{64}$ approach in [5] is the fastest Montgomery multiplier

Scalable Algorithm in GF(P)

- **FIOSM[5,11]**
  - Double loop
  - Clock cycles increase proportionally to \( m^2 \), where \( m \) is the number of digits
    - E.g. 160=5\( \cdot \)32
    - \( n=m \cdot k \)
      - \( n \): bit width of operands
      - \( m \): number of digits
      - \( k \): bit width of digits

```plaintext
INPUT: A = (a_{m-1}, \ldots, a_1, a_0)_{2^k}
INPUT: B = (b_{m-1}, \ldots, b_1, b_0)_{2^k}
INPUT: P = (p_{m-1}, \ldots, p_1, p_0)_{2^k} (0 \leq A, B < P)
INPUT: q = -P^{-1} \mod 2^k
OUTPUT: C = AB2^{-n} \mod P

C := 0
for i = 0 to m - 1
  z := 0
  t_i := (c_0 + a.ib_0) q \mod 2^k
  for j = 0 to m - 1
    S := c_j + a_ib_j + t_ip_j + z
    if (j \neq 0) then c_{j-1} := S \mod 2^k
    z := S/2^k
  c_{m-1} := z
  if (C > P) then C := C - P
```
Scalable Algorithm in \( GF(2^n) \)

- FIOSM[5,11]
  - Double loop
  - Clock cycles increase proportionally to \( m^2 \)
  - Same as GF(P) except final subtraction is not required

```latex
\text{INPUT: } A(x) = (a_{m-1}, \cdots, a_1, a_0) \mod x^r
\text{INPUT: } B(x) = (b_{m-1}, \cdots, b_1, b_0) \mod x^r
\text{INPUT: } P(x) = (p_{m-1}, \cdots, p_1, p_0) \mod x^r
\text{INPUT: } q(x) = P(x)^{-1} \mod x^r
\text{OUTPUT: } C(x) = A(x) B(x) x^{-n} \mod P(x)

C(x) := 0
\text{for } i = 0 \text{ to } m - 1
z(x) := 0
t_i(x) := [c_0(x) + a_i(x) b_0(x)] q(x) \mod x^r
\text{for } j = 0 \text{ to } m - 1
S(x) := c_j(x) + a_i(x) b_j(x) + t_i(x) p_j(x) + z(x)
\text{if } (j \neq 0) \text{ then } c_{j-1}(x) := S(x) \mod x^r
z(x) := S(x) / x^r
c_{m-1}(x) := z(x)
```
Previous Approaches for Reducing The Delay in $GF(P)$

- Drive $GF(P)$ circuit slower than $GF(2^n)$\cite{5}
  - Clock dividers are needed to change the field

- Use lower radix for $GF(P)$ and reduce the delay of the $GF(P)$ circuit\cite{7}
  - Clock cycles increase dramatically

\cite{5} A.Sato et al., IEEE Transactions on Computers, 2003
\cite{7} E.Savas et al., Computers and Digital Techniques, 2004
Block Diagram of the Proposal

- 64-bit multiplier in GF($2^n$)
- 4 16-bit multipliers in parallel in GF(P)
Reduce Clock Cycles

- Reduce the clock cycles with 4 parallel architecture in GF(P)

![Graph showing reduction in clock cycles with 4 parallel architecture compared to non-parallel approach.]
Comparison in Clock cycles

<table>
<thead>
<tr>
<th>Reference</th>
<th>Number of Clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GF(P)</td>
</tr>
<tr>
<td>This work</td>
<td>165</td>
</tr>
<tr>
<td>[5]</td>
<td>49</td>
</tr>
</tbody>
</table>

- However, frequency of proposal in GF(P) reaches 3.7 times faster than that of [5], so that proposed Montgomery multiplier is faster.

Future Work

- Incorporate the proposed multiplier into the entire cryptographic system

- Implement the multiplier into cryptographic system LSIs

<table>
<thead>
<tr>
<th>Layer</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layer-4:</td>
<td>Protocols (EC-DSA)</td>
</tr>
<tr>
<td>Layer-3:</td>
<td>Point multiplication</td>
</tr>
<tr>
<td>Layer-2:</td>
<td>Elliptic curve addition and doubling</td>
</tr>
<tr>
<td>Layer-1:</td>
<td>Finite field arithmetic (ADD/SUB/XOR/MAC in GF(q^2)/MAC in GF(P))</td>
</tr>
</tbody>
</table>