FPGA Implementation of Point Multiplication on Koblitz Curves Using Kleinian Integers

V.S. Dimitrov[1] **K.U. Järvinen**[2] **M.J. Jacobson, Jr.**[3]
W.F. Chan[3] Z. Huang[1]

[1]Department of Electrical and Computer Engineering, University of Calgary, 2500 University Drive NW, Calgary, Alberta, Canada T2N 1N4, (dimitrov,huangzh)@atips.ca

[2]Signal Processing Laboratory, Helsinki University of Technology, Otakaari 5A, 02150, Espoo, Finland, kimmo.jarvinen@tkk.fi

[3]Department of Computer Science, University of Calgary, 2500 University Drive NW, Calgary, Alberta, Canada T2N 1N4, (chanwf,jacobs)@cpsc.ucalgary.ca

October 13, 2006

Koblitz Curves

Koblitz curves (defined over $\mathbb{F}_2$):

$$E_a : y^2 + xy = x^3 + ax^2 + 1, \quad a \in \{0, 1\}$$

$|E_a(\mathbb{F}_{2^m})|$ easily computed for any integer $m > 0$

Frobenius endomorphism $\tau(x, y) = (x^2, y^2)$ for $(x, y) \in E_a(\mathbb{F}_{2^m})$:

- almost free to compute
- satisfies minimal polynomial $x^2 - \mu x + 2 = 0$ where $\mu = (-1)^{1-a}$
- can view τ as a root, i.e., $\tau = (\mu + \sqrt{-7})/2$
- leads to efficient τ-adic point multiplication algorithms (eg. τNAF)

Double Base Expansions

Dimitrov, Jullien, Miller (1998): compute kP using $k = \sum \pm 2^a 3^b$

- requires only $O(\log k / (\log \log k))$ $(2,3)$-integers
- find closest $\pm 2^a 3^b$ to k, subtract and repeat

Our contribution: efficient point multiplication on Koblitz curves

- first provably sublinear point multiplication algorithm (3 complex bases)
- efficient method using bases τ and $\tau - 1$ (no proof, conjectural sublinearity)
- no precomputations based on k or P
- efficient FPGA implementation

Kleinian Integer Expansions

Kleinian integers: $x + y\tau \in \mathbb{Z}[\tau]$

- $(\tau, \tau - 1)$-Kleinian integers: $\pm\tau^a(\tau - 1)^b$
- $(\tau, \tau - 1, \tau^2 - \tau - 1)$-Kleinian integers: $\pm\tau^a(\tau - 1)^b(\tau^2 - \tau - 1)^c$

Theorem: $k \in \mathbb{Z}[\tau]$ can be represented by a sum of $O(\log N(k)/(\log\log N(k)))$ $(\tau, \tau - 1, \tau^2 - \tau - 1)$-Kleinian integers

Conjecture: same for $(\tau, \tau - 1)$-Kleinian integers

- Proof for bases 2 and 3 doesn't generalize (only for real bases)
- Greedy algorithm doesn't generalize well:
  - hard to find closest $(\tau, \tau - 1)$-Kleinian integer to k

Conversion Algorithm

Compute $k = \sum_{i=1}^{d} \pm\tau^{a_i}(\tau - 1)^{b_i}$ for $k \in \mathbb{Z}[\tau]$

Precomputation: *minimal* representation for every $q = \sum_{i=0}^{w-1} d_i \tau^i, d_i \in \{0, 1\}$

1. Compute unsigned τ-adic expansion of k.
2. Divide τ-adic expansion into blocks of length w.
3. Substitute each block with minimal $(\tau, \tau - 1)$-expansion times appropriate power of τ

Assuming the conjecture, d and $\max(b_i)$ are both sublinear in $\log N(k)$

Example

$k = 6465$, $E_1(\mathbb{F}_{2^{163}})$, $\tau = (1 + \sqrt{-7})/2$

- partial reduction modulo $(\tau^{163} - 1)/(\tau - 1)$: $k \equiv \xi = -104 + 50\tau$

Using block size 7 we have:

$$\xi = \tau^{13} + \tau^{12} + \tau^{11} + \tau^{9} + \tau^{5} + \tau^{2}$$

Example

$k = 6465$, $E_1(\mathbb{F}_{2^{163}})$, $\tau = (1 + \sqrt{-7})/2$

- partial reduction modulo $(\tau^{163} - 1)/(\tau - 1)$: $k \equiv \xi = -104 + 50\tau$

Using block size 7 we have:

$$\begin{aligned} \xi &= \tau^{13} + \tau^{12} + \tau^{11} + \tau^{9} + \tau^{5} + \tau^{2} \\ &= \tau^{7}\left(\tau^{6} + \tau^{5} + \tau^{4} + \tau^{2}\right) + \left(\tau^{5} + \tau^{2}\right) \end{aligned}$$

Example

$k = 6465$, $E_1(\mathbb{F}_{2^{163}})$, $\tau = (1 + \sqrt{-7})/2$

- partial reduction modulo $(\tau^{163} - 1)/(\tau - 1)$: $k \equiv \xi = -104 + 50\tau$

Using block size 7 we have:

$$\begin{aligned} \xi &= \tau^{13} + \tau^{12} + \tau^{11} + \tau^{9} + \tau^{5} + \tau^{2} \\ &= \tau^{7}\left(\tau^{6} + \tau^{5} + \tau^{4} + \tau^{2}\right) + \left(\tau^{5} + \tau^{2}\right) \\ &= \tau^{7}\left(\tau(\tau - 1) + \tau(\tau - 1)^{6}\right) \end{aligned}$$

Example

$k = 6465,\ E_1(\mathbb{F}_{2^{163}}),\ \tau = (1 + \sqrt{-7})/2$

- partial reduction modulo $(\tau^{163} - 1)/(\tau - 1) : k \equiv \xi = -104 + 50\tau$

Using block size 7 we have:

$$
\begin{aligned}
\xi &= \tau^{13} + \tau^{12} + \tau^{11} + \tau^{9} + \tau^{5} + \tau^{2} \\
&= \tau^{7}\left(\tau^{6} + \tau^{5} + \tau^{4} + \tau^{2}\right) + \left(\tau^{5} + \tau^{2}\right) \\
&= \tau^{7}\left(\tau(\tau - 1) + \tau(\tau - 1)^{6}\right) + \left(\tau^{2}(\tau - 1)^{2}\right)
\end{aligned}
$$

Example

$k = 6465,\ E_1(\mathbb{F}_{2^{163}}),\ \tau = (1+\sqrt{-7})/2$

- partial reduction modulo $(\tau^{163}-1)/(\tau-1) : k \equiv \xi = -104 + 50\tau$

Using block size 7 we have:

$$\begin{aligned}
\xi &= \tau^{13} + \tau^{12} + \tau^{11} + \tau^{9} + \tau^{5} + \tau^{2} \\
&= \tau^{7}\left(\tau^{6} + \tau^{5} + \tau^{4} + \tau^{2}\right) + \left(\tau^{5} + \tau^{2}\right) \\
&= \tau^{7}\left(\tau(\tau-1) + \tau(\tau-1)^{6}\right) + \left(\tau^{2}(\tau-1)^{2}\right) \\
&= \tau^{8}(\tau-1) + \tau^{8}(\tau-1)^{6} + \tau^{2}(\tau-1)^{2}
\end{aligned}$$

Point Multiplication Algorithm

Given $k = \sum_{i=1}^{d} s_i \tau^{a_i} (\tau - 1)^{b_i}$ can write

$$k = \sum_{j=0}^{\max(b_i)} (\tau - 1)^j \left(\sum_{i=1}^{\max(a_{i,j})} s_{i,j} \tau^{a_{i,j}} \right)$$

Compute kP using $\max(b_i)$ τ-adic expansions

Cost:

- multiply by $(\tau - 1)$: one τ, one point subtraction
- overall: $\max(b_i) + d - 1$ point adds/subs
- number of point additions required is sublinear in $\log N(k)$

Numerical Evidence

Avg number of point adds to compute kP on $E_a(\mathbb{F}_{2^m})$

m	τNAF	Greedy	Blocking		
			$w = 5$	$w = 10$	$w = 16$
163	54.25	36.37	47.86	40.00	37.22
233	77.59	49.31	66.23	54.96	50.76
283	94.25	58.64	79.37	65.66	60.49
409	137.12	81.84	113.64	93.63	85.68
571	190.25	111.90	154.98	127.21	117.04

Fewer point adds than τNAF in all cases

- $w = 5$ requires < 1 KB ROM (no points need to be stored)

Computation of Algorithms

Specifications

NIST curve K-163

$\mathbb{F}_{2^{163}}$, normal basis

Computation of Algorithms

Specifications

NIST curve K-163

$\mathbb{F}_{2^{163}}$, normal basis

Point multiplication algorithm

Input: k, P
Output: $Q = kP$
  $P_0 \leftarrow P$; $Q \leftarrow \mathcal{O}$
  for $i = 0$ to $\max(b_i)$ **do**
    $S \leftarrow r_i(k)P_i$
    $P_{i+1} \leftarrow \tau P_i - P_i$
    $Q \leftarrow Q + S$
  end for

Computed one row, i.e. $(\sum_j k_{i,j}\tau^j)(\tau - 1)^i P$, at a time

- Each row is computed as a τNAF point multiplication

Computation of Algorithms

Specifications

NIST curve K-163

$\mathbb{F}_{2^{163}}$, normal basis

Point multiplication algorithm

Input: k, P

Output: $Q = kP$

$P_0 \leftarrow P$; $Q \leftarrow \mathcal{O}$

for $i = 0$ to $\max(b_i)$ **do**

$S \leftarrow r_i(k)P_i$

$P_{i+1} \leftarrow \tau P_i - P_i$

$Q \leftarrow Q + S$

end for

Computed one row, i.e. $(\sum_j k_{i,j}\tau^j)(\tau - 1)^i P$, at a time

- Each row is computed as a τNAF point multiplication

Point addition in mixed coordinates ($\mathcal{LD}/\mathcal{A}$) and Frobenius map in $\mathcal{LD}$

- $S \leftarrow S \pm P_i$; $S \leftarrow \tau S$

Computation of Algorithms

Specifications

NIST curve K-163

$\mathbb{F}_{2^{163}}$, normal basis

Point multiplication algorithm

Input: k, P

Output: $Q = kP$

$P_0 \leftarrow P$; $Q \leftarrow \mathcal{O}$

for $i = 0$ to $\max(b_i)$ **do**

$S \leftarrow r_i(k)P_i$

$P_{i+1} \leftarrow \tau P_i - P_i$

$Q \leftarrow Q + S$

end for

Computed one row, i.e. $(\sum_j k_{i,j}\tau^j)(\tau - 1)^i P$, at a time

- Each row is computed as a τNAF point multiplication

Point addition in mixed coordinates ($\mathcal{LD}/\mathcal{A}$) and Frobenius map in $\mathcal{LD}$

- $S \leftarrow S \pm P_i$; $S \leftarrow \tau S$

Frobenius map and point subtraction in $\mathcal{A}$

- $P_{i+1} \leftarrow \tau P_i - P_i$

Computation of Algorithms

Specifications

NIST curve K-163

$\mathbb{F}_{2^{163}}$, normal basis

Point multiplication algorithm

Input: k, P

Output: $Q = kP$

$P_0 \leftarrow P$; $Q \leftarrow \mathcal{O}$

for $i = 0$ to $\max(b_i)$ **do**

$S \leftarrow r_i(k)P_i$

$P_{i+1} \leftarrow \tau P_i - P_i$

$Q \leftarrow Q + S$

end for

Computed one row, i.e. $(\sum_j k_{i,j}\tau^j)(\tau - 1)^i P$, at a time

- Each row is computed as a τNAF point multiplication

Point addition in mixed coordinates $(\mathcal{LD}/\mathcal{A})$ and Frobenius map in $\mathcal{LD}$

- $S \leftarrow S \pm P_i$; $S \leftarrow \tau S$

Frobenius map and point subtraction in $\mathcal{A}$

- $P_{i+1} \leftarrow \tau P_i - P_i$

Point addition in $\mathcal{LD}$

- $Q \leftarrow Q + S$

Computation of Algorithms

Specifications

NIST curve K-163

$\mathbb{F}_{2^{163}}$, normal basis

Point multiplication algorithm

Input: k, P

Output: $Q = kP$

$P_0 \leftarrow P$; $Q \leftarrow \mathcal{O}$

for $i = 0$ to $\max(b_i)$ **do**

$S \leftarrow r_i(k)P_i$

$P_{i+1} \leftarrow \tau P_i - P_i$

$Q \leftarrow Q + S$

end for

Computed one row, i.e. $(\sum_j k_{i,j}\tau^j)(\tau - 1)^i P$, at a time

- Each row is computed as a τNAF point multiplication

Point addition in mixed coordinates $(\mathcal{LD}/\mathcal{A})$ and Frobenius map in $\mathcal{LD}$

- $S \leftarrow S \pm P_i$; $S \leftarrow \tau S$

Frobenius map and point subtraction in $\mathcal{A}$

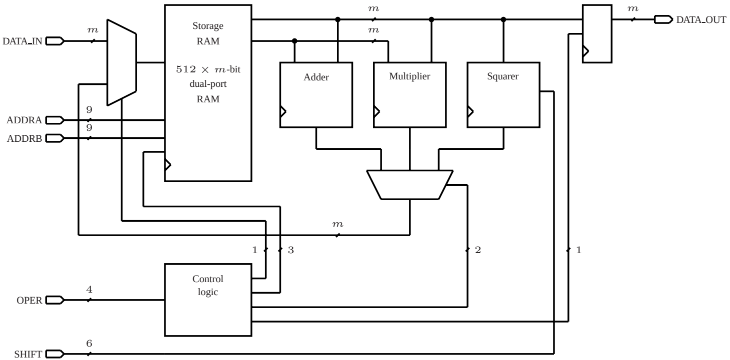
- $P_{i+1} \leftarrow \tau P_i - P_i$

Point addition in $\mathcal{LD}$

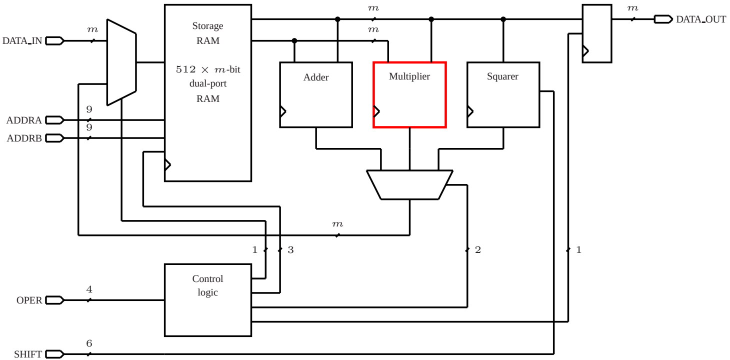
- $Q \leftarrow Q + S$

$\mathcal{LD} \mapsto \mathcal{A}$ mapping

Field arithmetic processor (FAP)



Field arithmetic processor (FAP)

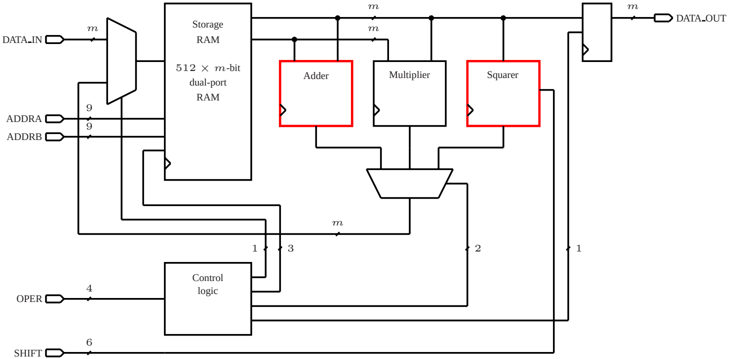


Multiplier

Digit-serial Massey-Omura multiplier

Latency: 9 clock cycles

Field arithmetic processor (FAP)

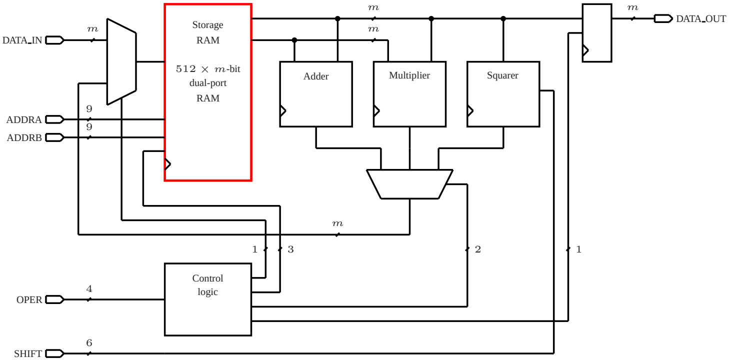


Adder and squarer

Adder: bitwise exclusive-or (xor)

Squarer: shifter (max shift $2^6 - 1$)

Field arithmetic processor (FAP)



Storage RAM

Dual-port RAM implemented in BlockRAMs

5 BlockRAMs needed (One B-RAM: 512 × 36-bits)

System architecture

$\{\tau, \tau - 1\}$-converter

- Converts k into $\{\tau, \tau - 1\}$-expansion
- Partial reduction (Solinas, 2000), computation of τ-adic expansion and blocking algorithm ($w = 10$)

Control logic and FAP

- FAP controlled by hand-optimized control sequences stored in a ROM (BlockRAM)
- k parsed and the ROM controlled by an FSM

Converter and the rest of the design use different clocks

Latency of a point multiplication (excluding conversion):

$$L_{kP} = 104\, d + 243 \max(b_i) + 84$$

Results

Xilinx Virtex-II XC2V2000-6

Maximum clock frequency: 128 MHz

Resource requirements: 6,494 slices and 6 BlockRAMs

Converter: 88 MHz, 2,251 slices, 2 BlockRAMs and 2 multipliers

- One conversion requires 3.81 μs

$\max(b_i)$	d	$\mathcal{LD}/\mathcal{A}$	$\mathcal{A}$	$\mathcal{LD}$	$\mathrm{L_{kP}}$	Time (μs)
0	54.33	53.33	0	0	5735	44.80
2	39.47	36.47	2	2	4675	36.52
3	36.18	32.18	3	3	4576	35.75
4	34.74	29.74	4	4	4669	36.48
5	33.42	27.42	5	5	4775	37.30
6	32.22	25.22	6	6	4893	38.23

Future work

Compare with ASIACRYPT 2006 (Avanzi, Dimitrov, Doche, Sica):

- proof of sublinear density for 2 complex bases
- memory-free conversion algorithm

Window method analogues (fixed base point):

- two-dimensional windows?

Analogue for hyperelliptic curves?

Implementation improvements:

- Computing rows in parallel leads to shorter latency
- Polynomial basis implementation