HIGH PERFORMANCE ELLIPTIC CURVE CRYPTO-PROCESSOR FOR FPGA PLATFORMS

Debdeep Mukhopadhyay
Dept. of Computer Science and Engg.
IIT Kharagpur

Outline

- Elliptic Curve Cryptography
- Finite Field Arithmetic
- The Elliptic Curve Crypto Processor
- Performance Comparisons
- Conclusion

Two Important Paradigms

- **Faster**
  - Higher Clocks
  - Parallelism
  - Large Bandwidth
- **Smaller**
  - Low Power
  - Smaller Area

Both Applications require security

Security on Miniature Devices

- Current Security Algorithms (RSA)
  - Key size of 2048 bits … too large
  - Too Computationally Intensive … (too much power and too slow)

Solution : Elliptic Curve Cryptography

Why ECC

- Smaller key and yet large security

Key Comparison

ECC Key size	RSA Key size	Key Ratio
163	1024	1:6
256	3072	1:12
384	7680	1:20
512	15360	1:30

- Small size of implementation and less power consumption

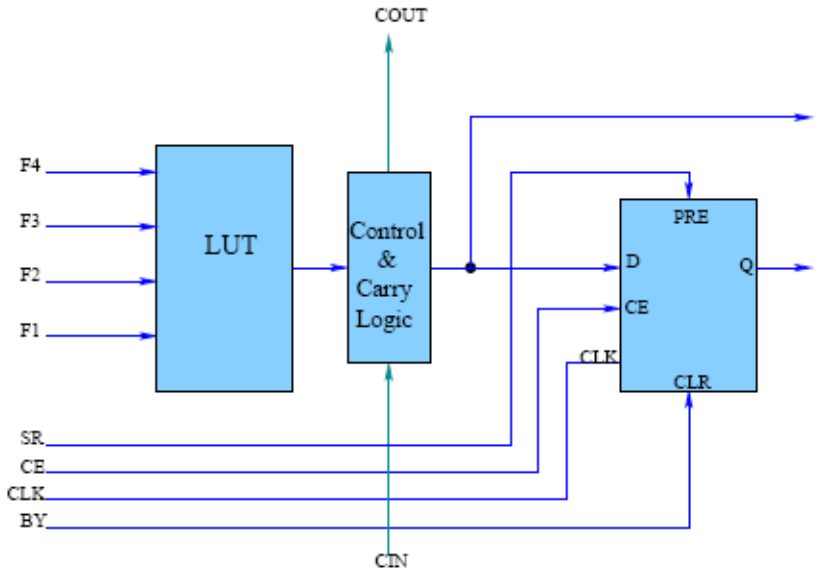
BUT IS IT FAST ENOUGH ??

SOLUTION : HARDWARE ACCELERATORS

Objective of the Work

- To build hardware accelerators for ECC on FPGA platforms for high performance applications.

FPGA Logic Block



- **LUT**
  - Four Input , One Output.
  - Can contain 16x1 SRAM.
  - Can implement any four input truth table.

LUT Utilization

$y_1 = x_1 \oplus x_2 \oplus x_3 \oplus x_4$ requires 1 LUT

$y_2 = x_1 \oplus x_2$ also requires 1 LUT

y_2 Results in an ***under utilized LUT***

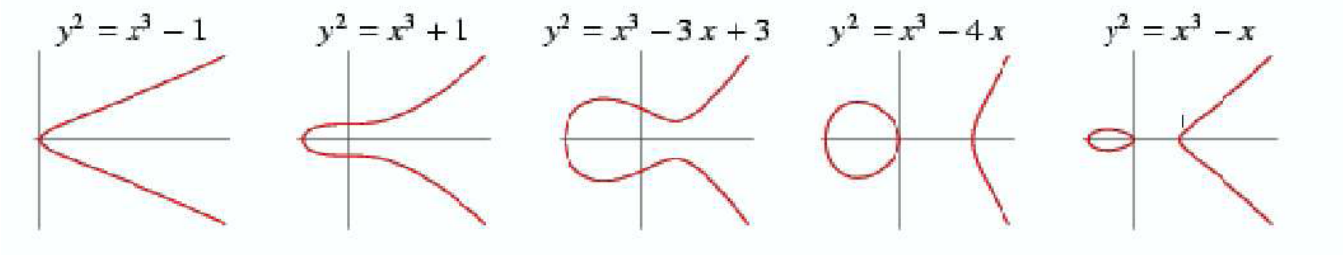
Design goal is to reduce the number of under utilized LUTs

Elliptic Curves

- An Elliptic Curve is the set of points which satisfy the equation

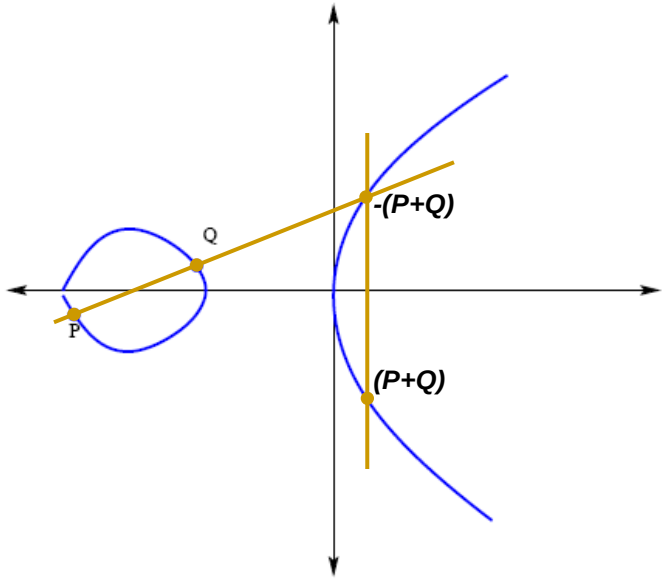
$$y^2 + xy = x^3 + ax^2 + b$$

- Inserting different values of constants would give different elliptic curves.



- Points on the elliptic curve along with a special point called *the point at infinity* form a Group under addition.

Point Addition (P+Q)



Equation for Point Addition

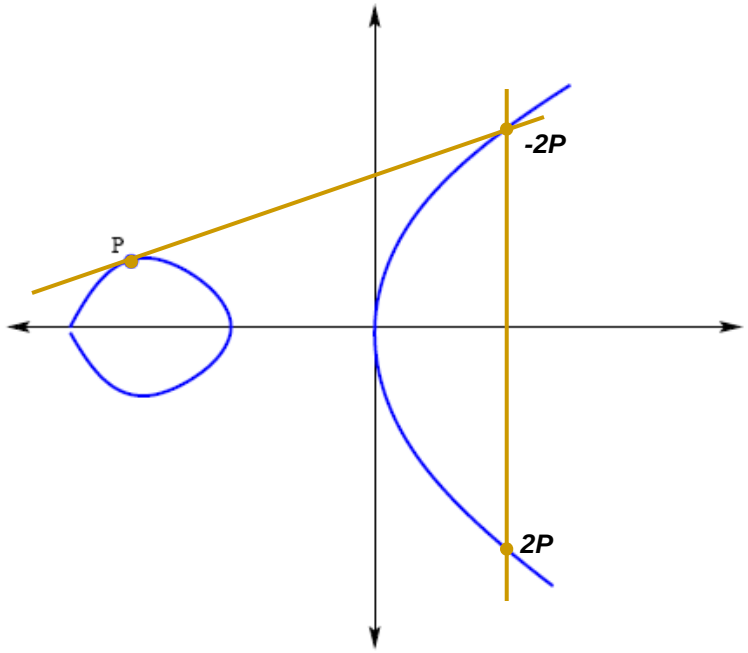
$P = (x_1, y_1)$, $Q = (x_2, y_2)$ and $(P + Q) = (x_3, y_3)$

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$$

$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1$$

where $\lambda = (y_1 + y_2)/(x_1 + x_2)$.

Point Doubling



Equation for Point Doubling, $P = (x_1, y_1)$, $(2P) = (x_4, y_4)$

$$x_3 = \lambda^2 + \lambda + a = {x_1}^2 + \frac{b}{{x_1}^2}$$

$$y_3 = {x_1}^2 + \lambda x_3 + x_3$$

where $\lambda = x_1 + (y_1/x_1)$.

Scalar Multiplication

- Given a point P=(x,y)
  determine kP = P + P + P + …. (k times)
- Double and Add algorithm

$22P$

$22 = (10110)_2$

bit	Double/Add	Value
1	-	P
0	D	2P
1	D&A	$4P + P = 5P$
1	D&A	$10P + P = 11P$
0	D	22P

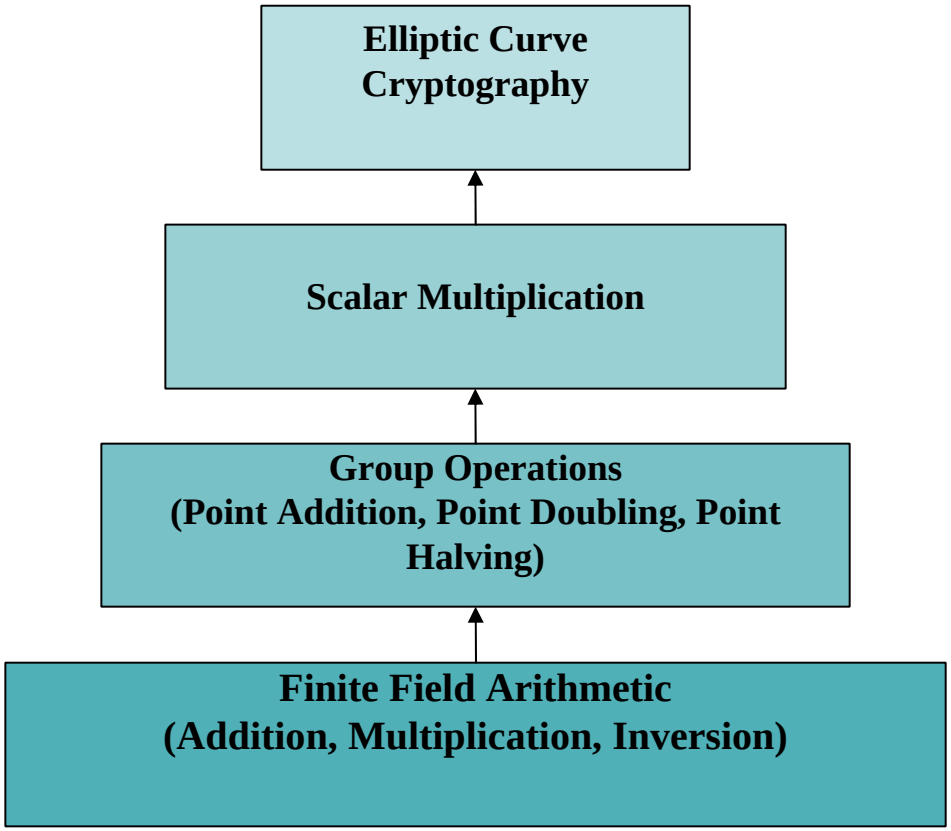
Elliptic Curves in Cryptography

- Given k and P it is easy to find kP
- Given kP and P it is hard to find k.
- Therefore, k is the *Private Key*, and kP is the *Public Key.*

Elliptic Curves in Cryptography

- Each x and y coordinate on the elliptic curve is taken from a finite field.
- Two finite fields are considered
  - Prime Field (GF(p))
  - Binary Finite Field (GF(2^m))

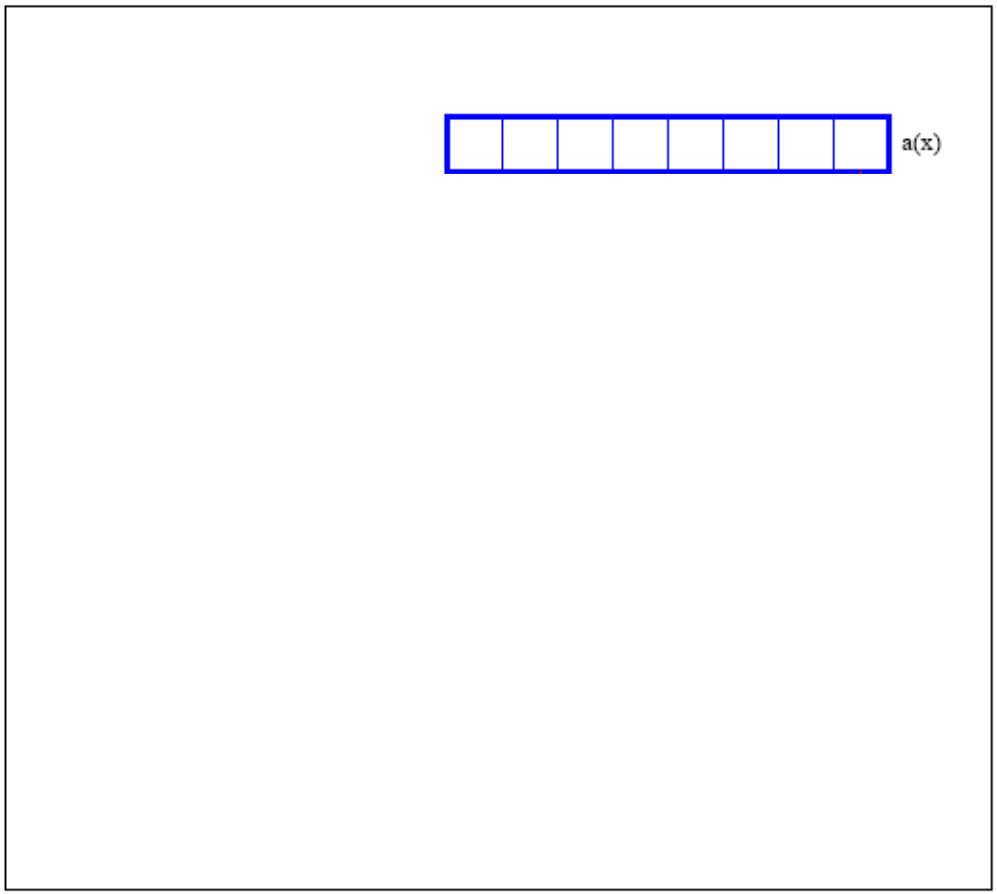
ECC Construction



Binary Finite Fields

- Addition
  - Is done by a simple XOR operation.
- Subtraction
  - Same as addition.
- Multiplication
  - Multiplication is done using polynomial multiplication, followed by a modular operation with the irreducible polynomial.

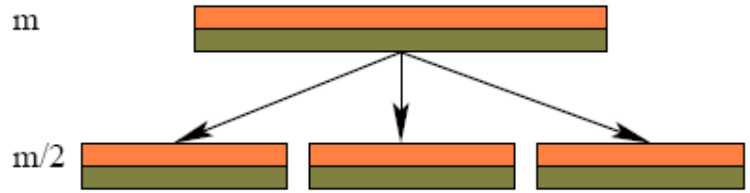
Squaring



Finite Field Multiplication

- We choose the **Karatsuba multiplier** as it is the fastest.
- For Elliptic Curves there are three types of combinational Karatsuba multipliers.
  - Simple Karatusba Multiplier.
  - Binary Karatsuba Multiplier.
  - General Karatsuba Multiplier.

Simple Karatsuba Multiplier



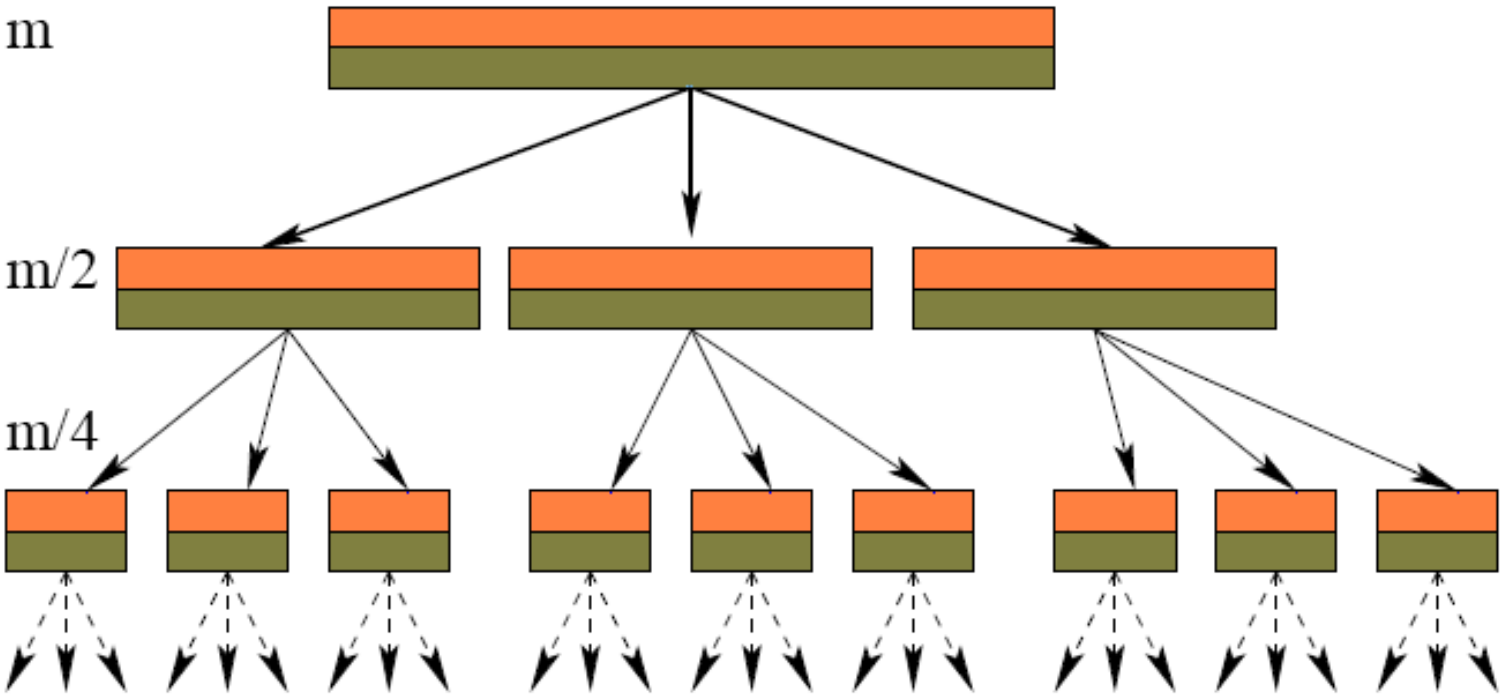
Split multiplicands into two

$$A(x) = A_h x^{m/2} + A_l$$
$$B(x) = B_h x^{m/2} + B_l$$

Use three $m/2$ bit multiplications

$$\begin{aligned} C'(x) &= (A_h x^{m/2} + A_l)(B_h x^{m/2} + B_l) \\ &= A_h B_h x^m + (A_h B_l + A_l B_h) x^{m/2} + A_l B_l \\ &= A_h B_h x^m \\ &\quad + ((A_h + A_l)(B_h + B_l) + A_h B_h + A_l B_l) x^{m/2} \\ &\quad + A_l B_l \end{aligned}$$

Recursive Simple Karatsuba Multiplier



General Karatsuba Multiplier

- Instead of splitting into two, splits into more than two.
  - For example, an m bit multiplier is split into m different multiplications.

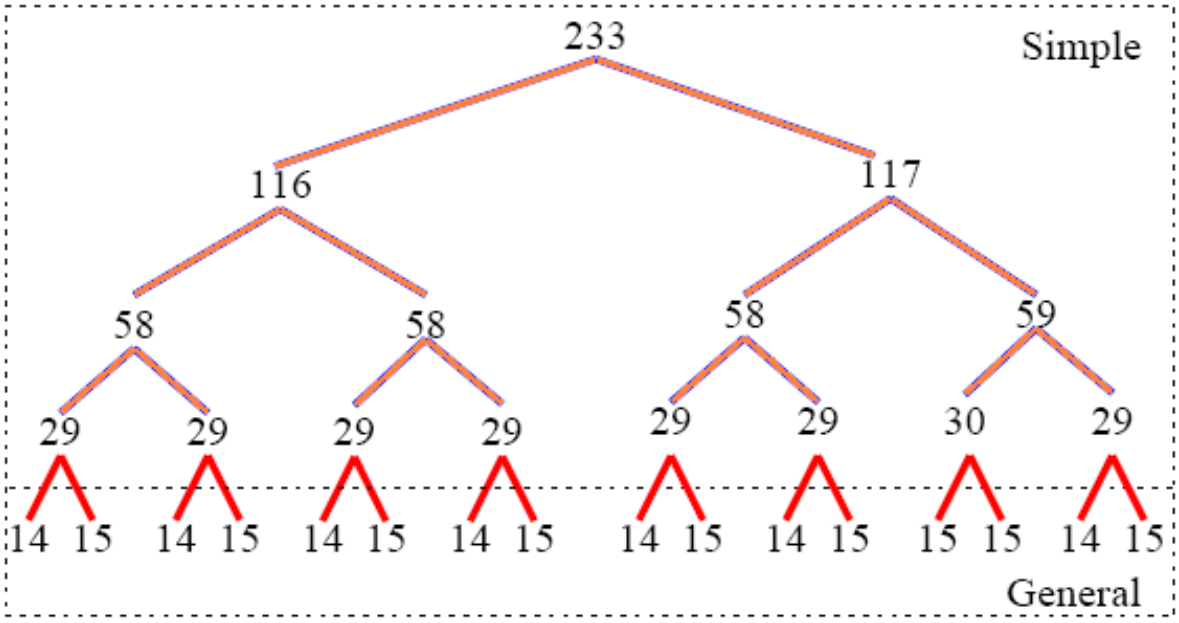
Comparing the General and Simple

m	General			Simple		
	Gates	LUTs	LUTs Under Utilized	Gates	LUTs	LUTs Under Utilized
2	7	3	66.6%	7	3	66.6%
4	37	11	45.5%	33	16	68.7%
8	169	53	20.7%	127	63	66.6%
16	721	188	17.0%	441	220	65.0%
29	2437	670	10.7%	1339	669	65.4%
32	2977	799	11.3%	1447	723	63.9%

- **Hybrid Karatsuba Multiplier**
  - For all recursions less than 29 use the General Karatsuba Multiplier.
  - For all recursions greater than 29 use the Simple Karatsuba multiplier

Example

- 233 bit Hybrid Karatsuba Multiplier



Multiplicative Inverse

- Given an element ‘a’, find a^{-1} in the field such that $a.a^{-1} = 1$.
- Techniques
  - Extended Euclid’s Algorithm
  - Fermat’s Little Theorem
- Fermat’s Little Theorem is more efficient on hardware than the Euclidean technique.
- Fermat’s Little Theorem
  - $a^{-1} = a^{n-2} \bmod n$
  - For example $3*3^{(5-2)} \bmod 5 = 3*2 \bmod 5 = 1$

Itoh-Tsujii method for Binary Finite Fields

- We need to find $a^{-1} = a^{2^m-2}$ for m=233
- We first define an addition chain for m – 1
  - (1,2,3,6,7,14,28,29, 58,116, 232)
- Then, define $\beta_k = a^{2^k-1}$ then $a^{-1} = (a^{2^{232}-1})^2 = (\beta_{232})^2$
- Also define the recursion $\beta_{k+j} = (\beta_k)^{2^j}\beta_j$

Computing the inverse of 'a'

	$\beta_{u_i}(a)$	$\beta_{u_j+u_k}(a)$	Exponentiation
1	$\beta_1(a)$		a
2	$\beta_2(a)$	$\beta_{1+1}(a)$	$(\beta_1)^{2^1}\beta_1 = a^{2^2-1}$
3	$\beta_3(a)$	$\beta_{2+1}(a)$	$(\beta_2)^{2^1}\beta_1 = a^{2^3-1}$
4	$\beta_6(a)$	$\beta_{3+3}(a)$	$(\beta_3)^{2^3}\beta_3 = a^{2^6-1}$
5	$\beta_7(a)$	$\beta_{6+1}(a)$	$(\beta_6)^{2^1}\beta_1 = a^{2^7-1}$
6	$\beta_{14}(a)$	$\beta_{7+7}(a)$	$(\beta_7)^{2^7}\beta_7 = a^{2^{14}-1}$
7	$\beta_{28}(a)$	$\beta_{14+14}(a)$	$(\beta_{14})^{2^{14}}\beta_{14} = a^{2^{28}-1}$
8	$\beta_{29}(a)$	$\beta_{28+1}(a)$	$(\beta_{28})^{2^1}\beta_1 = a^{2^{29}-1}$
9	$\beta_{58}(a)$	$\beta_{29+29}(a)$	$(\beta_{29})^{2^{29}}\beta_{29} = a^{2^{58}-1}$
10	$\beta_{116}(a)$	$\beta_{58+58}(a)$	$(\beta_{58})^{2^{58}}\beta_{58} = a^{2^{116}-1}$
11	$\beta_{232}(a)$	$\beta_{116+116}(a)$	$(\beta_{116})^{2^{116}}\beta_{116} = a^{2^{232}-1}$

- In all we need 232 squarings and 10 multiplications.

Using Quads instead of Squarers

Field	Squarer Circuit		Quad Circuit		Size ratio
	$\#LUT_s$	Delay (ns)	$\#LUT_q$	Delay (ns)	$\frac{\#LUT_q}{2(\#LUT_s)}$
$GF(2^{233})$	153	1.48	230	1.48	0.75

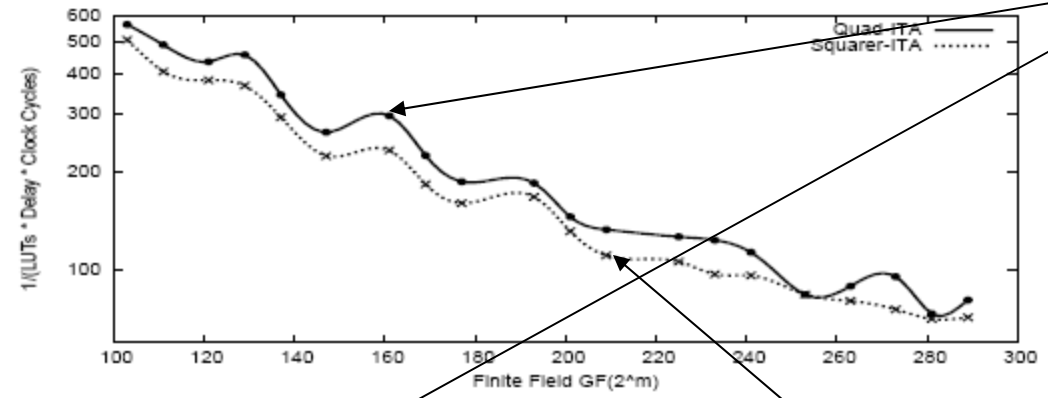
- On an FPGA, Quads have better LUT utilization compared to squarers.
- Delay of a quad and a squarer is the same
- Therefore we propose a **Quad Itoh-Tsujii** algorithm which uses quad circuits instead of squarers..

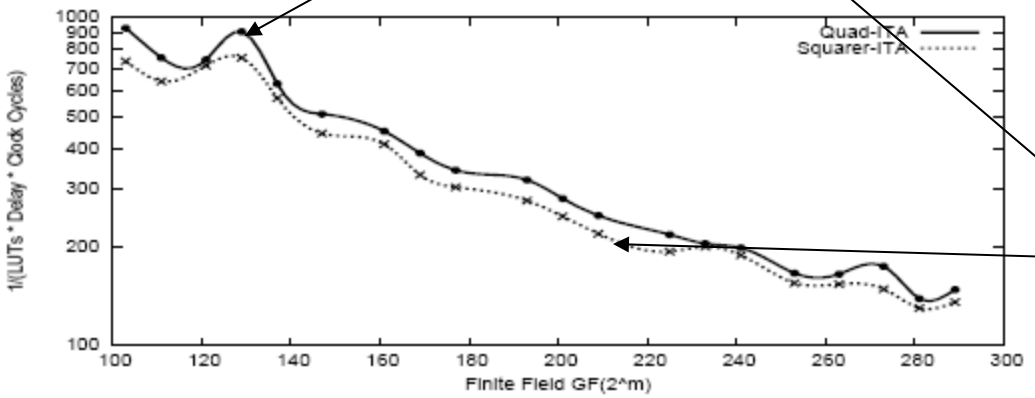
Quad Itoh Tsujii Algorithm

	α_{u_i}	$\alpha_{u_j+u_k}$	Exponentiation
1	α_1		a^3
2	α_2	α_{1+1}	$(\alpha_1)^{4^1}\alpha_1 = a^{4^2-1}$
3	α_3	α_{2+1}	$(\alpha_2)^{4^1}\alpha_1 = a^{4^3-1}$
4	α_6	α_{3+3}	$(\alpha_3)^{4^3}\alpha_3 = a^{4^6-1}$
5	α_7	α_{6+1}	$(\alpha_6)^{4^1}\alpha_1 = a^{4^7-1}$
6	α_{14}	α_{7+7}	$(\alpha_7)^{4^7}\alpha_7 = a^{4^{14}-1}$
7	α_{28}	α_{14+14}	$(\alpha_{14})^{4^{14}}\alpha_{14} = a^{4^{28}-1}$
8	α_{29}	α_{28+1}	$(\alpha_{28})^{4^1}\alpha_1 = a^{4^{29}-1}$
9	α_{58}	α_{29+29}	$(\alpha_{29})^{4^{29}}\alpha_{29} = a^{4^{58}-1}$
10	α_{116}	α_{58+58}	$(\alpha_{58})^{4^{58}}\alpha_{58} = a^{4^{116}-1}$

- We now require 115 quads (instead of 232) and 10 multiplications. We save 7 clock cycles.

Performance of Quad-Itoh Tsujii on Virtex 4 and 5 Platforms





Quad ITA

Virtex 4

Virtex 5

Squarer ITA

Comparisions for NIST Binary Curves having irreducible Trinomials

Field	Algorithm	LUTs	Delay (ns)	Clks	T (ns)	P
Virtex 4						
$GF(2^{233})$	Squarer-ITA	27897	10.2	36	367.2	97.6
$GF(2^{233})$	Quad-ITA	26122	10.3	30	309	123.9
$GF(2^{409})$	Squarer-ITA	64970	12.9	40	516	29.8
$GF(2^{409})$	Quad-ITA	60644	15.8	32	505.6	32.6
Virtex 5						
$GF(2^{233})$	Squarer-ITA	21379	7.09	33	234	199.9
$GF(2^{233})$	Quad-ITA	20950	7.79	30	233.7	204.2
$GF(2^{409})$	Squarer-ITA	47785	8.56	40	342.4	61.1
$GF(2^{409})$	Quad-ITA	44948	9.2	35	322	69.1

Comparision for Inversion in GF(2^{193}) on XCV3200efg1156 Platform

Implementation	Resources Utilized (Slices, Brams)	Freq (MHz) (f)	Clock Cycles (c)	Time μsec (c/f)	P
Sequential [3]	10065, 12	21.2	28	1.32	75.2
Parallel [4]	11081, 12	21.2	20	0.94	95.7
Quad-ITA	10420, 0	35	21	0.6	160

- The Quad-ITA offers the best performance than sequential or parallel ITA.
- We have tested on this on various fields and on modern FPGAs.

Multiplication vs Inversion

- Finite field inversion is several times more expensive than multiplication.
- Therefore we need to reduce the inversions present.
- One solution is to use a 3 coordinate system (**Projective Coordinates**)
- Each 2 coordinate point (x,y) called **Affine Point,** is mapped to a unique point in the projective plane with coordinates (X,Y,Z)

Projective Coordinates

- Point Addition

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

$$D = B^2 \cdot (C + a \cdot Z_1^2) \; ; \; Z_3 = C^2 \; ; \; E = A \cdot C \; ; \; X_3 = A^2 + D + E$$

$$F = X_3 + x_2 \cdot Z_3 \; ; \; G = (x_2 + y_2) \cdot Z_3^2 \; ; \; Y_3 = (E + Z_3) \cdot F + G$$

- Point Doubling

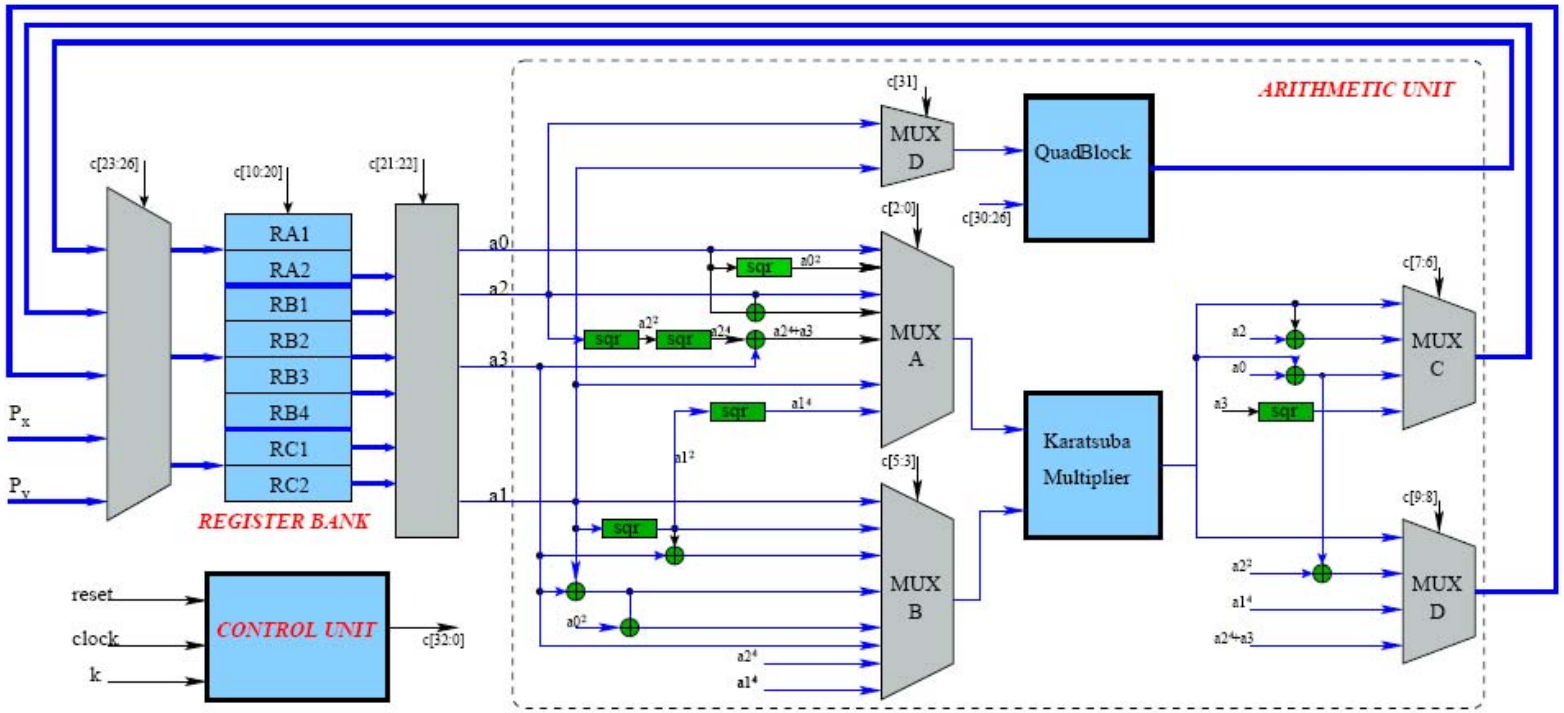
$$Z_4 = X_1^2 \cdot Z_1^2 \; ; X_4 = X_1^4 + b \cdot Z_1^4$$

$$Y_4 = b \cdot Z_1^4 \cdot Z_4 + X_4 \cdot (a \cdot Z_4 + Y_1^2 + b \cdot Z_1^4)$$

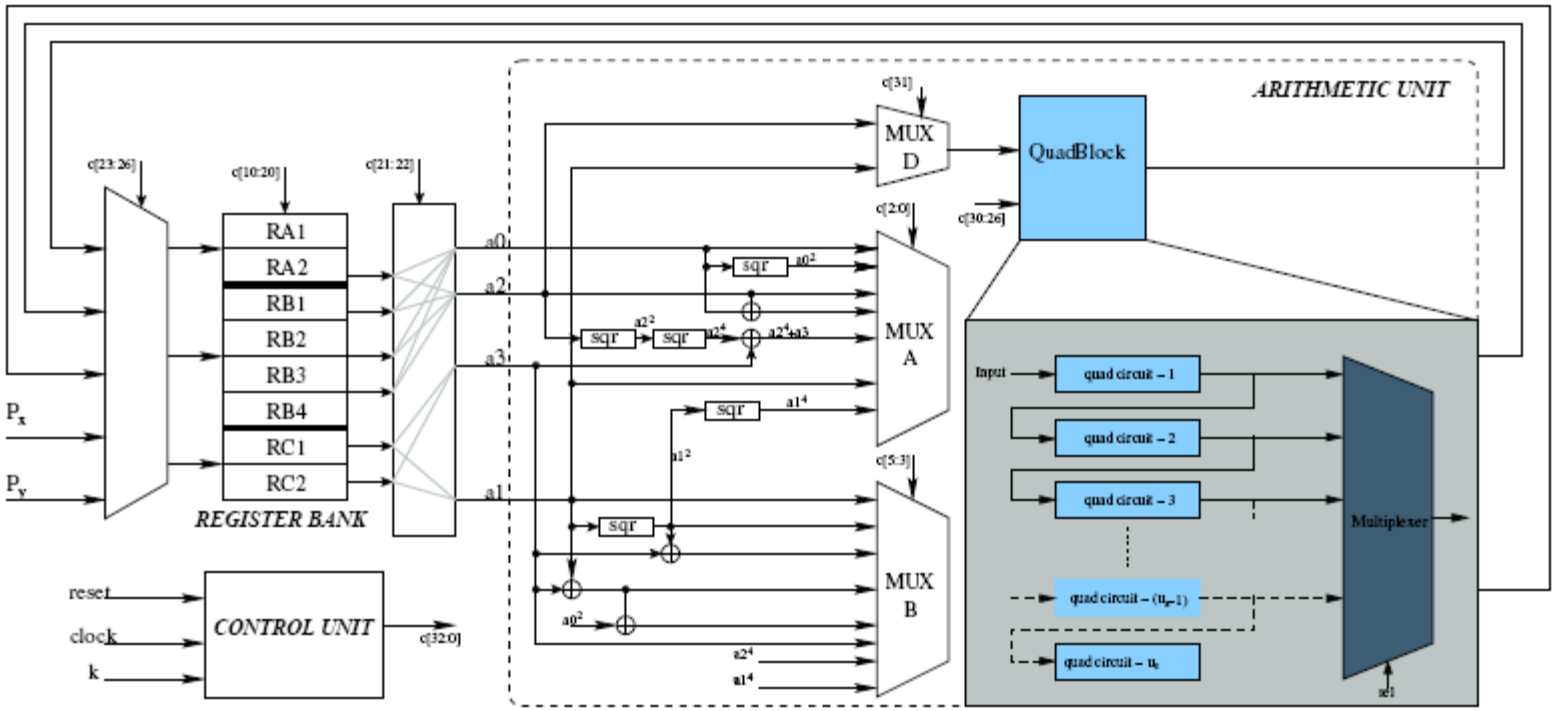
- Conversion between Projective and Affine

$$x = X/Z \qquad y = Y/Z^2$$

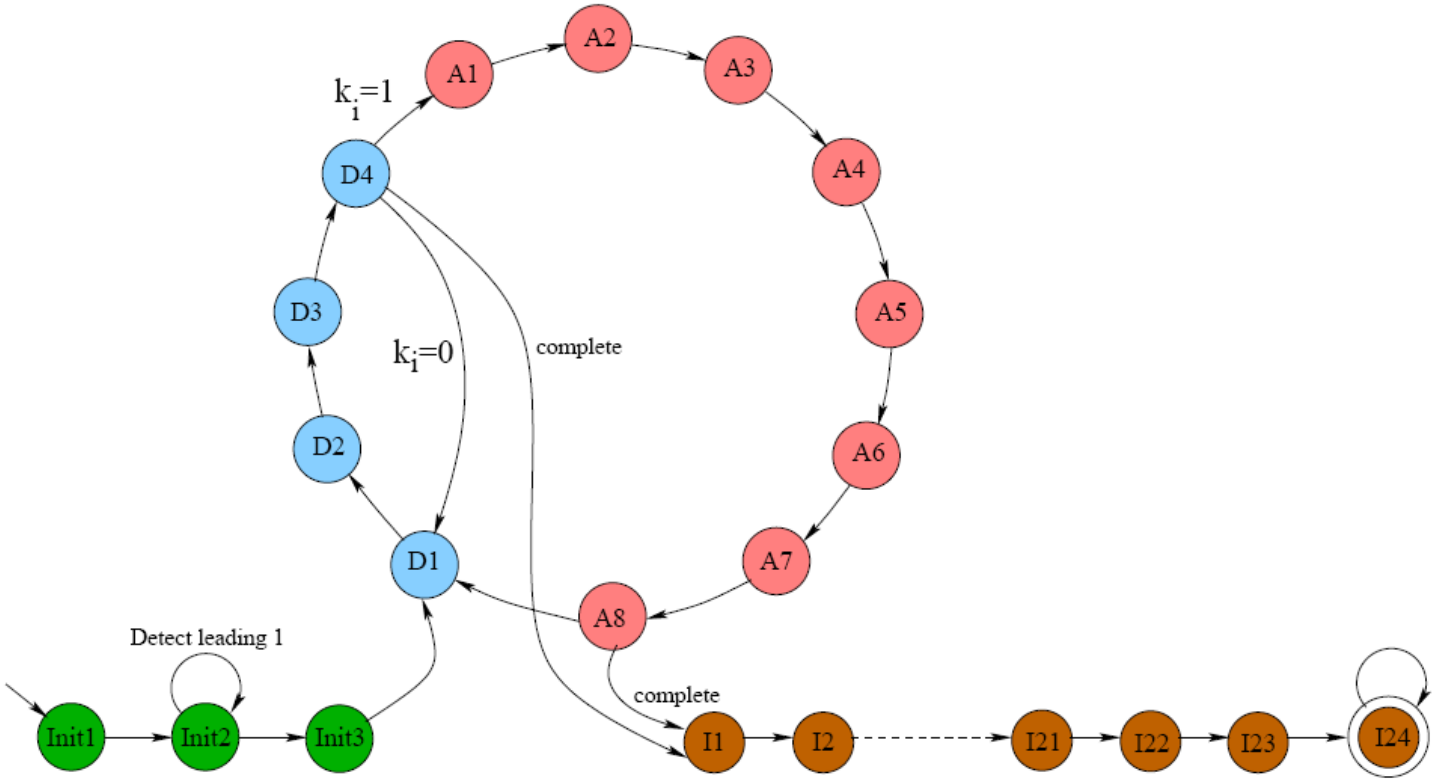
The Elliptic Curve Crypto Processor



QuadBlock


ARITHMETIC UNIT
MUX D
QuadBlock
MUX A
MUX B
RA1
RA2
RB1
RB2
RB3
RB4
RC1
RC2
REGISTER BANK
CONTROL UNIT
reset
clock
k
sqr
Input
quad circuit – 1
quad circuit – 2
quad circuit – 3
Multiplexer

The FSM


A1
A2
A3
A4
A5
A6
A7
A8
D1
D2
D3
D4
k_i=1
k_i=0
complete
complete
Detect leading 1
Init1
Init2
Init3
I1
I2
I21
I22
I23
I24

Parallel Point Arithmetic

Point Addition

Clock	Operation 1 (c_0)	Operation 2(c_1)
1	$RB_1 = RB_2 \cdot RC_1^2 + RB_1$	-
2	$RA_1 = RA_2 \cdot RC_1 + RA_1$	-
3	$RB_3 = RA_1 \cdot RC_1$	-
4	$RA_1 = RA_1^2 \cdot (RC_1^2 + RB_3)$	-
5	$RC_2 = RB_1 \cdot RB_3$	$RA_1 = RB_1 \cdot RB_3 + RA_1 + RB_1^2$
6	$RC_1 = RB_3^2$	$RB_3 = RA_2 \cdot RB_3^2 + RA_1$
7	$RB_1 = (RA_2 + RB_2) \cdot RC_1^2$	-
8	$RB_1 = (RC_2 + RC_1) \cdot RB_3 + RB_1$	-

Point Doubling

Clock	Operation 1 ($C0$)	Operation 2($C1$)
1	$RC_1 = RA_1^2 \cdot RC_1^2$	$RB_3 = RC_1^4$
2	$RB_3 = RB_3 \cdot RB_3$	
3	$RC_2 = (RA_1^4 + RB_3) \cdot (RC_1 + RB_1^2 + RB_3)$	$RA_1 = (RA_1^4 + RB_3)$
4	$RB_1 = RB_3 \cdot RC_1 + RC_2$	

Control Words

Clock Cycle	QuadBlock Control ($c_{31} \cdots c_{27}$)	Register Input ($c_{26} \cdots c_{23}$)	Register Output ($c_{22} c_{21}$)	Register addressing and read and write ($c_{20} \cdots c_{10}$)	Mux D ($c_9 c_8$)	Mux C ($c_7 c_6$)	Mux B ($c_5 c_4 c_3$)	Mux A ($c_2 c_1 c_0$)
D1	0 0 x x x	x 0 1 x	x 0	x 1 0 1 x x 1 0 0 x 0	1 1	0 0	0 0 1	0 0 1
D2	0 0 x x x	x 0 0 1	1 1	1 0 0 1 0 0 1 0 1 0 0	1 0	0 0	1 0 0	1 0 0
D3	0 0 x x x	x 0 0 0	0 1	1 0 0 1 1 0 0 0 0 x x	0 0	1 0	1 0 1	1 0 1
A1	0 0 x x x	x 0 0 x	0 1	x 0 0 1 0 1 0 0 0 x x	x x	0 1	0 0 1	0 0 0
A2	0 0 x x x	x 0 0 x	1 0	x 0 0 0 x x 0 0 1 1 0	x x	1 0	0 0 0	0 1 0
A3	0 0 x x x	x 0 x x	x 0	0 1 1 0 x x x x 0 x 0	x x	0 0	1 0 1	0 0 0
A4	0 0 x x x	x 0 x 1	x 0	1 1 0 0 x x x x 1 x 0	0 0	1 1	0 1 0	0 0 1
A5	0 0 x x x	x 0 x 1	0 0	x 1 1 0 x x 0 0 1 x 0	0 1	0 0	0 0 0	0 1 0
A6	0 0 x x x	x 0 0 x	1 0	x 0 0 1 x x 1 0 0 0 1	x x	0 1	0 0 0	0 0 0
A7	0 0 x x x	x 0 0 x	1 1	x 0 0 1 0 1 0 0 0 1 x	x x	0 0	0 0 1	0 1 1
A8	0 0 x x x	x 0 0 x	0 1	0 0 1 1 1 0 0 0 0 x x	x x	0 1	0 1 1	0 0 0
I1	0 0 x x x	x 0 x x	x x	x 1 0 x x x x x 0 x x	x x	0 0	0 0 1	1 0 1
I2	0 0 x x x	x 0 0 x	0 x	x 0 0 1 x x 1 0 0 x x	x x	0 0	0 0 0	1 1 0
I3	0 0 x x x	x 0 0 x	0 x	x x 0 1 x x 1 0 0 x x	x x	0 0	1 1 0	1 0 1
I4	0 1 0 0 1	1 0 x x	0 x	x 1 1 0 x x 1 0 0 x x	x x	x x	x x x	x x x
I5	0 0 x x x	x 0 0 x	0 x	x 0 1 1 x x 1 0 0 x x	x x	0 0	0 0 0	0 1 0
I6	0 0 x x x	x 0 0 x	0 x	x 0 0 1 x x 1 0 0 x x	x x	0 0	1 1 0	1 0 1
I7	0 1 0 1 1	1 0 x x	0 x	x 1 1 0 x x 1 0 0 x x	x x	x x	x x x	x x x
I8	0 0 x x x	x 0 0 x	x x	x 0 1 1 x x 1 0 0 x x	x x	0 0	0 0 0	0 1 0
I9	0 1 1 1 1	0 0 x x	0 x	x 1 1 0 x x 1 0 0 x x	x x	x x	x x x	x x x
I10	0 0 x x x	x 0 0 x	x x	x 0 1 1 x x 1 0 0 x x	x x	0 0	0 0 0	0 1 0
I11	0 0 x x x	x 0 0 x	0 x	x 0 0 1 x x 1 0 0 x x	x x	0 0	1 1 0	1 0 1
I12	0 1 1 1 1	0 0 x x	0 x	x 1 1 0 x x 1 0 0 x x	x x	x x	x x x	x x x
I13	1 1 1 1 1	0 0 x x	0 x	x 1 1 0 x x x x 0 x x	x x	x x	x x x	x x x
I14	0 0 x x x	x 0 0 x	0 x	x 0 1 1 x x 1 0 0 x x	x x	0 0	1 1 1	0 1 0
I15	0 1 1 1 1	0 0 x x	0 x	x 1 1 0 x x 1 0 0 x x	x x	x x	x x x	x x x
I16	1 1 1 1 1	0 0 x x	0 x	x 1 1 0 x x x x 0 x x	x x	x x	x x x	x x x
I17	1 1 1 1 1	0 0 x x	0 x	x 1 1 0 x x x x 0 x x	x x	x x	x x x	x x x
I18	1 1 1 1 1	0 0 x x	0 x	x 1 1 0 x x x x 0 x x	x x	x x	x x x	x x x
I19	1 1 0 0 1	0 0 x x	0 x	x 1 1 0 x x x x 0 x x	x x	x x	x x x	x x x
I20	0 0 x x x	x 0 0 x	0 x	x 1 1 0 x x 1 0 0 x x	x x	0 0	0 0 0	0 1 0
I21	0 0 x x x	x x x x	x x	1 1 0 0 x x x x 0 x x	x x	1 1	x x x	x x x

Performance Results

	LUTs	Frequency $freq$	Clock Cycles CC	Performance η_1 $(freq/(LUTs * CC))$	Performance η_2 $(freq/LUTs)$
Hybrid Karatsuba Quad-Itoh Tsujii	34394	37.611MHz	33	33.137	1093
Binary Karatsuba [9] Quad-Itoh Tsujii	36970	35.433MHz	33	29.043	958
Hybrid Karatsuba Squarer-Itoh Tsujii[10]	33326	37.853MHz	43	26.414	1135
Binary Karatsuba [9] Squarer-Itoh Tsujii[10]	35805	35.669MHz	43	23.167	996

Comparisons

Work	Platform	Field m	Slices	LUTs	Gate Count	Freq (MHz)	Latency (ms)	Latency /bit (ns)
Orlando 2000	XCV400E	163	-	3002	-	76.7	0.21	1288
Bednara 2002	XCV1000	191	-	48300	-	36	0.27	1413
Kerins 2002	XCV2000	239	-	-	74103	30	12.8	53556
Gura 2003	XCV2000E	163	-	19508	-	66.5	0.14	858
Mentens 2004	XCV800	160	-	-	150678	47	3.810	23812
Lutz 2004	XCV2000E	163	-	10017	-	66	0.075	460
Saqib 2004	XCV3200	191	18314	-	-	10	0.056	**293**
Pu 2006	XC2V1000	193	-	3601	-	115	0.167	865
Ansari 2006	XC2V2000	163	-	8300	-	100	0.042	257
Chelton 2008	XCV2600E	163	15368	26390	238145	91	0.033	**202**
	XC4V200	163	16209	26364	264197	153.9	0.019	**116**
This Work	XCV3200E	233	20325	40686	333063	25.31	0.074	**317**
	XC4V140	233	20917	39303	334709	64.46	0.029	**124**

- **Saqib 2004**, does not do the final conversion from projective to affine coordinates
- **Chelton** 2008, has better latency than our implementation, but we have a better area time product.
- **Area time product** : Chelton is 894 while our area time product is 606.

Conclusion

- The paper presents an implementation of an Elliptic Curve Crypto Processor.
- High speed obtained by implementations of
  - Hybrid Karatsuba multiplier
  - Quad Itoh Tsujii inversion algorithm.
- The Hybrid Karatsuba multiplier can be used to minimize LUT requirements and increase operating frequency.
- The Quad Itoh Tsujii algorithm can be used to reduce computation time.

References

- Chester Rebeiro, Debdeep Mukhopadhyay: High Speed Compact Elliptic Curve Cryptoprocessor for FPGA Platforms. INDOCRYPT 2008: 376-388
- Chester Rebeiro, Sujoy Sinha Roy, Sankara Reddy and Debdeep Mukhopadhyay, "Revisiting the Itoh-Tsujii Inversion Algorithm for FPGA Platforms", To Appear in IEEE Transactions on VLSI Systems.


Thank You