Secured Embedded Architecture Laboratory (SEAL)

Pushing the Limits of High-Speed $GF(2^m)$ Elliptic Curve Scalar Multiplier on FPGAs

Chester Rebeiro, Sujoy Sinha Roy, and Debdeep Mukhopadhyay

Secured Embedded Architecture Lab
Indian Institute of Technology Kharagpur
India

9/12/2012

CHES 2012, Leuven Belgium

Elliptic Curve Scalar Multiplication

- An elliptic curve over $GF(2^m)$ is a set of points which satisfies the equation
$$y^2 + xy = x^3 + ax^2 + b \ ,$$
where $a, b \in GF(2^m)$ and $b \neq 0$. The points on the elliptic curve form an additive group.
- The projective coordinate representation of the curve is
$$Y^2 + XYZ = X^3 + aX^2Z^2 + bZ^4$$
- Scalar Multiplication : Given a base point $P = (X_P, Y_P, Z_P)$ on the elliptic curve and a scalar s compute $Q = sP$ $(i.e.\ Q = P + P + P + \cdots (s times))$

Montgomery Ladder for Scalar Multiplication

Inputs : scalar $s = (s_{t-1}s_{t-2}\cdots s_1 s_0)_2$
basepoint P

Output : Scalar Product $Q = sP$

```
❶ $P_1 = (X_1, Y_1, Z_1) \leftarrow P$ and $P_2 = (X_2, Y_2, Z_2) \leftarrow 2 \cdot P$
❷ For each bit $s_k$ (for $k = t-2, t-3, \cdots, 0$)
    • if $s_k = 1$ then $P_1 \leftarrow P_1 + P_2$ ; $P_2 = 2 \cdot P_2$
    • if $s_k = 0$ then $P_2 \leftarrow P_1 + P_2$ ; $P_1 = 2 \cdot P_1$
❸ $Q = Projective2Affine(P_1)$
```

Montgomery Ladder for Scalar Multiplication

Inputs : scalar $s = (s_{t-1}s_{t-2}\cdots s_1 s_0)_2$
basepoint P

Output : Scalar Product $Q = sP$

```
❶ $P_1 = (X_1, Y_1, Z_1) \leftarrow P$ and $P_2 = (X_2, Y_2, Z_2) \leftarrow 2 \cdot P$
❷ For each bit $s_k$ (for $k = t-2, t-3, \cdots, 0$)
    • if $s_k = 1$ then $P_1 \leftarrow P_1 + P_2$ ; $P_2 = 2 \cdot P_2$
    • if $s_k = 0$ then $P_2 \leftarrow P_1 + P_2$ ; $P_1 = 2 \cdot P_1$
❸ $Q = Projective2Affine(P_1)$
```

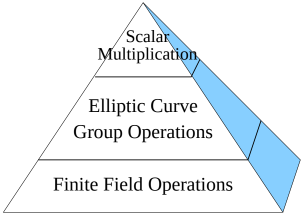
Performing $P_i \leftarrow P_i + P_j$ and $P_j \leftarrow 2 \cdot P_i$

$$X_i \leftarrow X_i \cdot Z_j \ ; \ Z_i \leftarrow X_j \cdot Z_i \ ; \ T \leftarrow X_j \ ; \ X_j \leftarrow X_j^4 + b \cdot Z_j^4$$

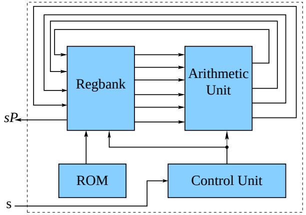
$$Z_j \leftarrow (T \cdot Z_j)^2 \ ; \ T \leftarrow X_i \cdot Z_i \ ; Z_i \leftarrow (X_i + Z_i)^2 \ ; \ X_i \leftarrow x \cdot Z_i + T$$

. . . all operations are in $GF(2^m)$

Engineering the Montgomery Ladder for Scalar Multiplication

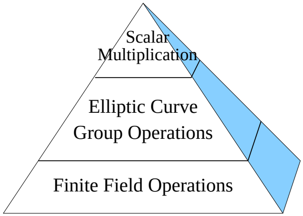


(a) The ECC Pyramid

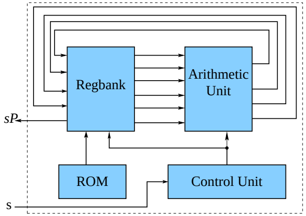


(b) Block Diagram

Engineering the Montgomery Ladder for Scalar Multiplication



(a) The ECC Pyramid



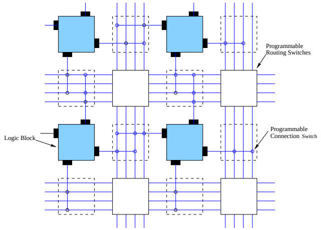
(b) Block Diagram

High-speed scalar multiplication on FPGAs

- Minimize area by maximizing utilization of available resources
- Optimal Pipelining
- Efficient Scheduling of Operations

Field Programmable Gate Arrays

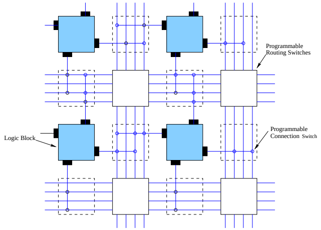
- Provides the speed of hardware and the reconfigurablitity of software
- FPGA Architecture



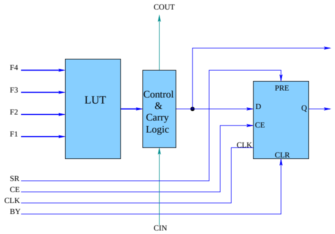
(a) FPGA Island

Field Programmable Gate Arrays

- Provides the speed of hardware and the reconfigurablitity of software
- FPGA Architecture



(a) FPGA Island



(b) Lookup Table

LUT Utilization

LUT

- Four (or six) input $\rightarrow$ one output
- Can implement any four (or six) input truth table

- $y_1 = x_1 \oplus x_2 \oplus x_3 \oplus x_4$ Requires one LUT.
- $y_2 = x_1 \oplus x_2$ Still requires one LUT.

LUT Utilization

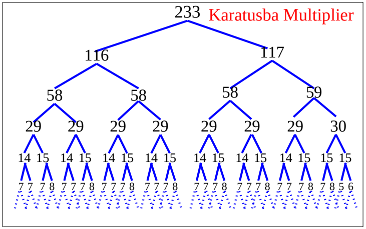
LUT

- Four (or six) input $\rightarrow$ one output
- Can implement any four (or six) input truth table

- $y_1 = x_1 \oplus x_2 \oplus x_3 \oplus x_4$ Requires one LUT.
- $y_2 = x_1 \oplus x_2$ Still requires one LUT.
- y_2 results in an under utilized LUT.

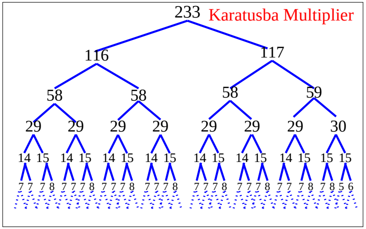
. . . need to maximize LUT utilization to minimize area.

Finite field Multiplier for Best LUT utilization

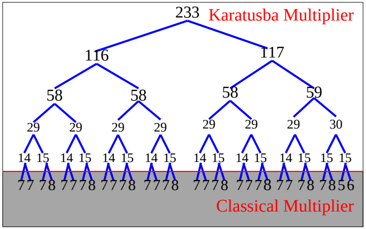


(a) Karatsuba-Ofman Multiplication

Finite field Multiplier for Best LUT utilization

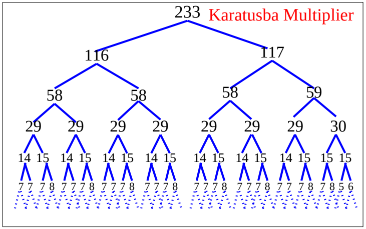


(a) Karatsuba-Ofman Multiplication

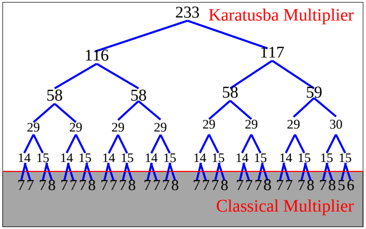


(b) Hybrid Karatsuba Multiplication

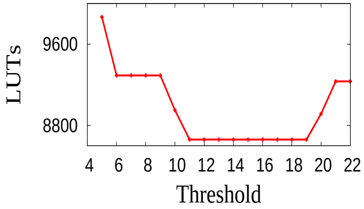
Finite field Multiplier for Best LUT utilization



(a) Karatsuba-Ofman Multiplication

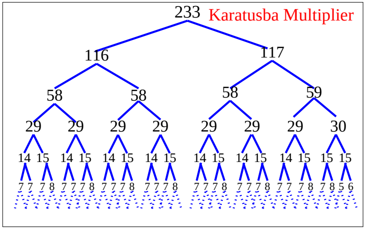


(b) Hybrid Karatsuba Multiplication

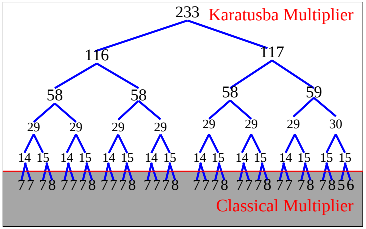


(c) Finding the Right Threshold

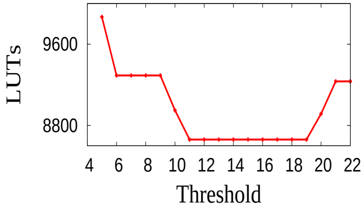
Finite field Multiplier for Best LUT utilization



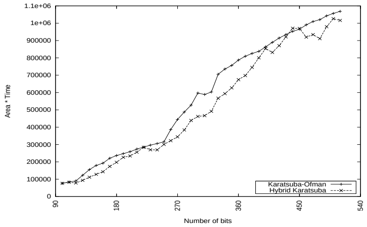
(a) Karatsuba-Ofman Multiplication



(b) Hybrid Karatsuba Multiplication



(c) Finding the Right Threshold



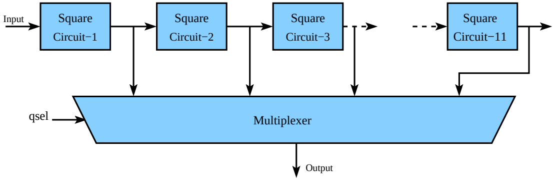
(d) Comparing Multipliers

Finite Field Inversion Using Itoh-Tsujii Algorithm

- Given $a \in GF(2^m)$, find $a^{-1} \in GF(2^m)$ such that $a \cdot a^{-1} = 1$
- Fermat's Little Theorem : $a^{-1} = a^{2^m-2}$
- Itoh-Tsujii Algorithm
  1. Define the addition chain for $m-1$
     (for example $m = 233 :\ (1, 2, 3, 6, 7, 14, 28, 58, 116, 232)$)
  2. Compute
     $a \rightarrow a^{2^2-1} \rightarrow a^{2^3-1} \rightarrow a^{2^6-1} \rightarrow a^{2^7-1} \rightarrow a^{2^{14}-1} \cdots \rightarrow a^{2^{232}-1}$
  3. Square to get $a^{2^{233}-2}$

Finite Field Inversion Using Itoh-Tsujii Algorithm

- Given $a \in GF(2^m)$, find $a^{-1} \in GF(2^m)$ such that $a \cdot a^{-1} = 1$
- Fermat's Little Theorem : $a^{-1} = a^{2^m-2}$
- Itoh-Tsujii Algorithm
  1. Define the addition chain for $m-1$
     (for example $m = 233$: $(1, 2, 3, 6, 7, 14, 28, 58, 116, 232)$)
  2. Compute
     $a \to a^{2^2-1} \to a^{2^3-1} \to a^{2^6-1} \to a^{2^7-1} \to a^{2^{14}-1} \cdots \to a^{2^{232}-1}$
  3. Square to get $a^{2^{233}-2}$
- Exponentiation requires a series of cascaded squarers called powerblock along with a finite field multiplier



Using Higher Exponents in the Itoh-Tsujii Algorithm

Consider using a quad circuit instead of a square.

- This requires an addition chain to $\frac{m-1}{2}$ instead of $m-1$ thus finishes faster.

[IEEE TVLSI 2011, DATE 2011]

Using Higher Exponents in the Itoh-Tsujii Algorithm

Consider using a quad circuit instead of a square.

- This requires an addition chain to $\frac{m-1}{2}$ instead of $m-1$ thus finishes faster.
- The frequency of operation is not affected and area used is less due to better LUT utilization.

Table: Comparison of Squarer and Quad Circuits on Xilinx Virtex 4 FPGA

Field	Squarer Circuit		Quad Circuit		Size ratio
	$\#LUT_s$	Delay (ns)	$\#LUT_q$	Delay (ns)	$\frac{\#LUT_q}{2(\#LUT_s)}$
$GF(2^{193})$	96	1.48	145	1.48	0.75
$GF(2^{233})$	153	1.48	230	1.48	0.75

[IEEE TVLSI 2011, DATE 2011]

Using Higher Exponents in the Itoh-Tsujii Algorithm

Consider using a quad circuit instead of a square.

- This requires an addition chain to $\frac{m-1}{2}$ instead of $m-1$ thus finishes faster.
- The frequency of operation is not affected and area used is less due to better LUT utilization.

Table: Comparison of Squarer and Quad Circuits on Xilinx Virtex 4 FPGA

Field	Squarer Circuit		Quad Circuit		Size ratio
	$\#LUT_s$	Delay (ns)	$\#LUT_q$	Delay (ns)	$\frac{\#LUT_q}{2(\#LUT_s)}$
$GF(2^{193})$	96	1.48	145	1.48	0.75
$GF(2^{233})$	153	1.48	230	1.48	0.75

- Larger exponent circuits can similarly be used to obtain faster results.

[IEEE TVLSI 2011, DATE 2011]

Using Higher Exponents in the Itoh-Tsujii Algorithm

Consider using a quad circuit instead of a square.

- This requires an addition chain to $\frac{m-1}{2}$ instead of $m-1$ thus finishes faster.
- The frequency of operation is not affected and area used is less due to better LUT utilization.

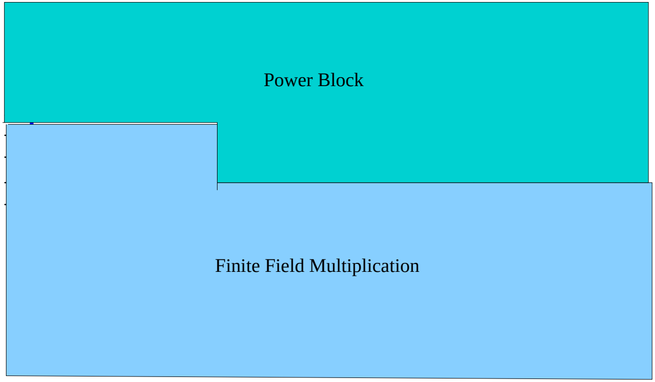
Table: Comparison of Squarer and Quad Circuits on Xilinx Virtex 4 FPGA

Field	Squarer Circuit		Quad Circuit		Size ratio
	$\#LUT_s$	Delay (ns)	$\#LUT_q$	Delay (ns)	$\frac{\#LUT_q}{2(\#LUT_s)}$
$GF(2^{193})$	96	1.48	145	1.48	0.75
$GF(2^{233})$	153	1.48	230	1.48	0.75

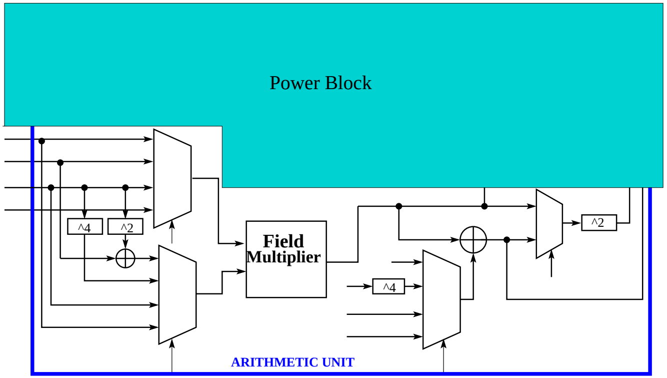
- Larger exponent circuits can similarly be used to obtain faster results.
- However there is an initial overhead of computing a^{2^q-1}, which increases as the exponent circuit increases.

[IEEE TVLSI 2011, DATE 2011]

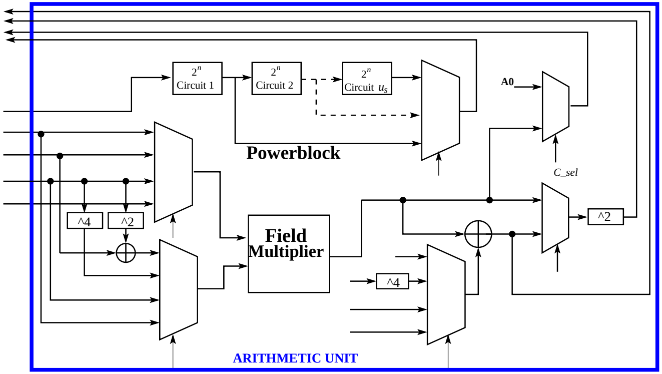
The Arithmetic Unit



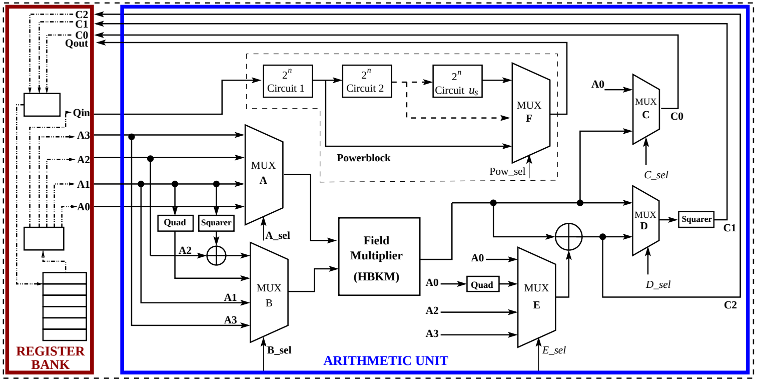
The Arithmetic Unit



The Arithmetic Unit



The Register Bank



Six registers implemented as flip-flops to maximize CLB utilization

Pipelining the Processor

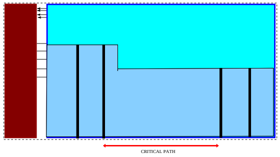
- How many stages of pipelining ?
  - To many would increase the frequency of operation but makes it difficult to schedule instructions without bubbles
  - To few would result in a low frequency of operation

Pipelining the Processor

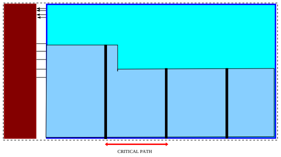
- How many stages of pipelining ?
    - To many would increase the frequency of operation but makes it difficult to schedule instructions without bubbles
    - To few would result in a low frequency of operation
- Where to place the pipeline stages?

Pipelining the Processor

- How many stages of pipelining ?
    - To many would increase the frequency of operation but makes it difficult to schedule instructions without bubbles
    - To few would result in a low frequency of operation
- Where to place the pipeline stages?



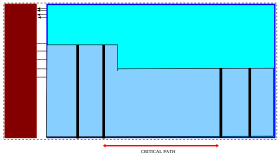
(a) A Bad Pipline Stage Placement



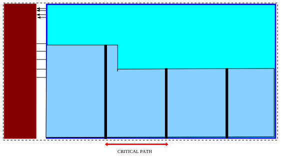
(b) A Good Pipeline Stage Placement

Pipelining the Processor

- How many stages of pipelining ?
    - To many would increase the frequency of operation but makes it difficult to schedule instructions without bubbles
    - To few would result in a low frequency of operation
- Where to place the pipeline stages?



(a) A Bad Pipline Stage Placement



(b) A Good Pipeline Stage Placement

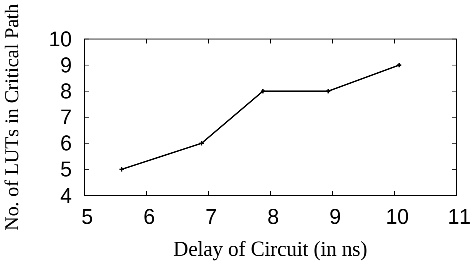
Can we determine the best pipeline strategy at the design exploration phase?

Optimally Pipeline into L Stages Apriori

1. Model the delay of the design
2. Use the model to identify all critical paths (with delay t)
3. Place pipeline registers to ensure that no path has delay greater than $\frac{t}{L}$

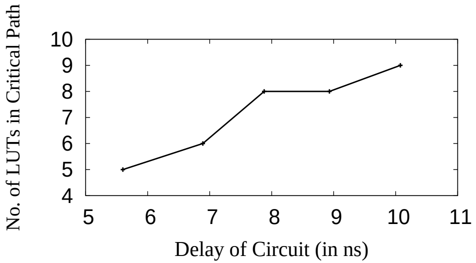
Modeling the Delay

- The delay of a circuit is proportional to the number of LUTs in the critical path



Modeling the Delay

- The delay of a circuit is proportional to the number of LUTs in the critical path

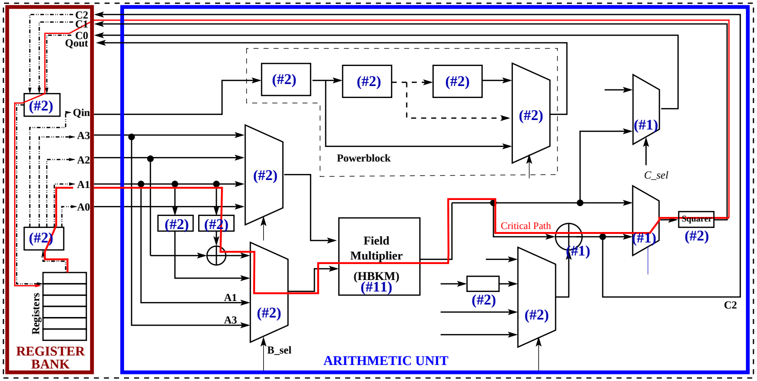


- The number of LUTs in the critical path of an n input Boolean function $f(x_1, x_2, x_3, \cdots x_n)$ is $\lceil \mathbf{log_k(n)} \rceil$, where k is the number of inputs to the LUT.

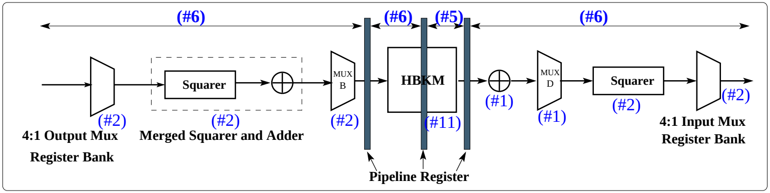
Modeling LUTdelay of all Elements in the Processor

Component	$\mathrm{k} - LUT$ **Delay for** $\mathrm{k} \geq 4$	$m = 163, \mathrm{k} = 4$
m bit field adder	1	1
m bit $n:1$ Mux ($D_{n:1}(m)$)	$\lceil log_{\mathrm{k}}(n + log_2 n) \rceil$	2 (for $n = 4$) 1 (for $n = 2$)
Exponentiation Circuit ($D_{2^n}(m)$)	$max(LUTDelay(d_i))$, where d_i is the i^{th} output bit of the exponentiation circuit	2 (for $n = 1$) 2 (for $n = 2$)
Powerblock ($D_{powerblk}(m)$)	$u_s \times D_{2^n}(m) + D_{u_s:1}(m)$	4 (for $u_s = 2$)
Modular Reduction (D_{mod})	1 for irreducible trinomials 2 for pentanomials	2 (for pentanomials)
HBKM ($D_{\mathrm{HBKM}}(m)$)	$D_{split} + D_{threshold} + D_{combine} + D_{mod}$ $= \lceil log_k(\frac{m}{\tau}) \rceil + \lceil log_k(2\tau) \rceil$ $+ \lceil log_2(\frac{m}{\tau}) \rceil + D_{mod}$	11 (for $\tau = 11$)

Critical Path : Placing Pipeline Registers Optimally



Critical Path : Placing Pipeline Resgisters Optimally



4 stage pipeline

- Critical path length : 23 LUTs
- Time Period : $\lceil \frac{23}{4} \rceil = 6$ LUTs

Scheduling of Operations (for 1 bit)

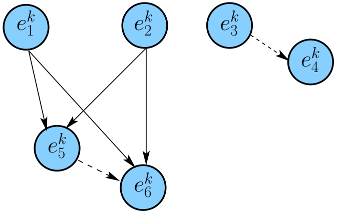
Table: Scheduling Instructions

$e_1^k : X_i \leftarrow X_i \cdot Z_j$	$e_4^k : Z_j \leftarrow (T \cdot Z_j)^2$
$e_2^k : Z_i \leftarrow X_j \cdot Z_i$	$e_5^k : T \leftarrow X_i \cdot Z_i;\ Z_i \leftarrow (X_i + Z_i)^2$
$e_3^k : T \leftarrow X_j;\ X_j \leftarrow X_j^4 + b \cdot Z_j^4$	$e_6^k : X_i \leftarrow x \cdot Z_i + T$

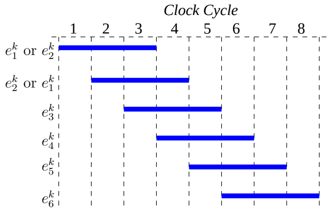
Scheduling of Operations (for 1 bit)

Table: Scheduling Instructions

e_1^k : $X_i \leftarrow X_i \cdot Z_j$	e_4^k : $Z_j \leftarrow (T \cdot Z_j)^2$
e_2^k : $Z_i \leftarrow X_j \cdot Z_i$	e_5^k : $T \leftarrow X_i \cdot Z_i;\ Z_i \leftarrow (X_i + Z_i)^2$
e_3^k : $T \leftarrow X_j;\ X_j \leftarrow X_j^4 + b \cdot Z_j^4$	e_6^k : $X_i \leftarrow x \cdot Z_i + T$



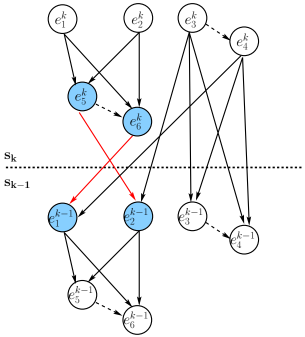
(a) Data Dependencies Between Instructions

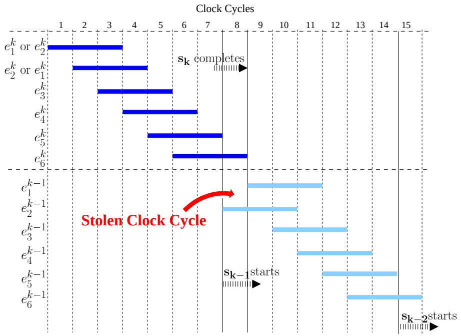


(b) Timing Diagram for 3 Stage Pipeline

Scheduling of 2 Consequtive bits : Overlapping two bits

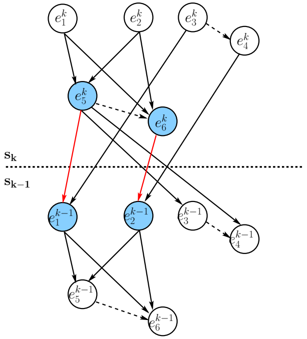
When two consequtive bits are equal

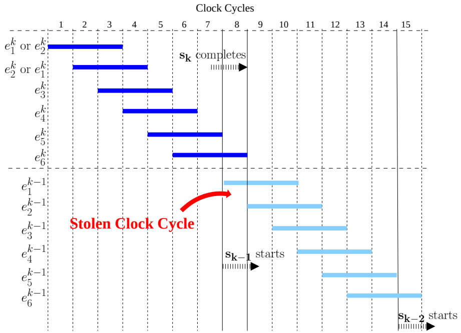




Scheduling of 2 Consequtive bits : Overlapping two bits

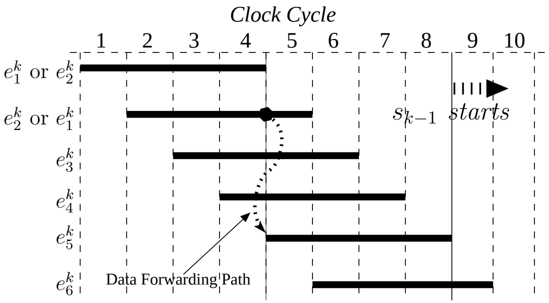
When two consequtive bits are NOT equal





Clock Cycles for L Stage Pipeline

- Without overlapping $2L + 2$ clock cycles are required per bit
- With overlapping, we save a clock cycle, so $2L + 1$
- Additionally,
  - If there is a pipline stage soon after the multiplier, then data forwarding is feasible
  - Clock cycles per bit is $2L$



Clock cycles saved are m or $2m$

Modeling Computation Time to find the Right Pipeline

So, we now know

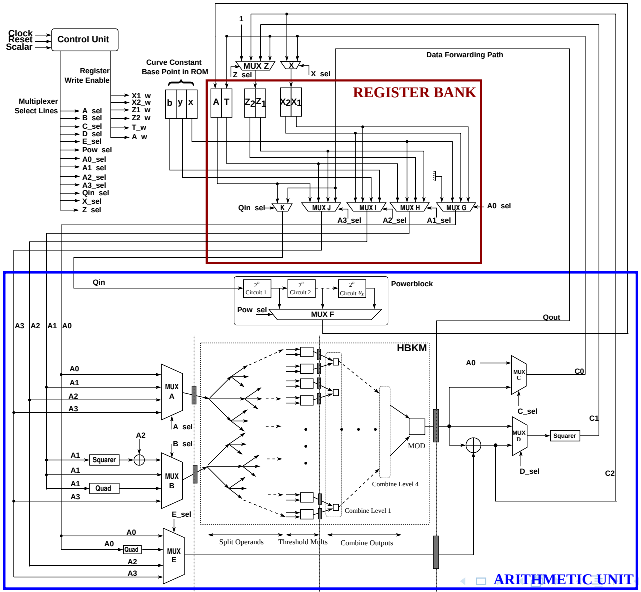
- How to place pipeline registers
- and estimate the clock cycles required

Putting it together, we can estimate how long it takes to perform a scalar multiplication

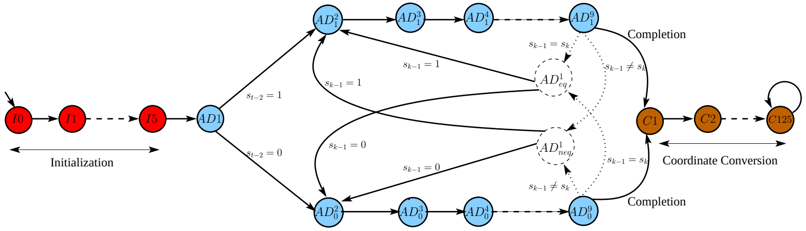
Table: Computation Time Estimates for Various Values of L for an ECM over $GF(2^{163})$ and FPGA with 4 input LUTs

L	u_s	DataForwarding Feasible	Computation Time
1	9	No	1019
2	4	No	524
3	3	No	412
4	2	Yes	357
5	1	No	395
6	1	Yes	360
7	1	Yes	358

Architecture Details



Finite State Machine



Comparisons

Work	Platform	Field (m)	Slices	LUTs	Freq (MHz)	Comp. Time (μs)
Orlando	XCV400E	163	-	3002	76.7	210
Bednara	XCV1000	191	-	48300	36	270
Gura	XCV2000E	163	-	19508	66.5	140
Lutz	XCV2000E	163	-	10017	66	233
Saqib	XCV3200	191	18314	-	10	56
Pu	XC2V1000	193	-	3601	115	167
Ansari	XC2V2000	163	-	8300	100	42
Rebeiro	XC4V140	233	19674	37073	60	31
Järvinen[1]	Stratix II	163	(11800ALMs)	-	-	48.9
Kim [2]	XC4VLX80	163	24363	-	143	10.1
Chelton	XCV2600E	163	15368	26390	91	33
	XC4V200	163	16209	26364	153.9	19.5
Azarderakhsh	XC4CLX100	163	12834	22815	196	17.2
	XC5VLX110	163	6536	17305	262	12.9
Our Result (Virtex 4 FPGA)	XC4VLX80	163	8070	14265	147	9.7
	XC4V200	163	8095	14507	132	10.7
	XC4VLX100	233	13620	23147	154	12.5
Our Result (Virtex 5 FPGA)	XC5VLX85t	163	3446	10176	167	8.6
	XC5VSX240	163	3513	10195	148	9.5
	XC5VLX85t	233	5644	18097	156	12.3

1. uses 4 field multipliers; 2. uses 3 field multipliers; 3. uses 2 field multipliers

Conclusions

- We show the implementation of a high-speed elliptic curve crypto-processor for FPGA platforms
- The use of highly optimized finite field primitives, efficient utilization of FPGA primitives, help reduce the area, which in turn make routing easier
- A theoretically designed pipline strategy provides ideal pipelining of the design to increase clock frequncy
- Efficient scheduling with data forwarding machanisms reduce clock cycle requirement
- All these result in one of the fastest elliptic curve implementation on FPGAs
- Additionally, area required is significantly less compared to other high-speed designs

Thank You for your Attention