A compact and scalable hardware/software co-design of SIKE

Pedro Maat C. Massolino[1] Lejla Batina[1] Patrick Longa[2] Joost Renes[1]

[1]Radboud University, Nijmegen, The Netherlands
lejla@cs.ru.nl, joost.renes@nxp.com, pedro.massolino@pqshield.com

[2]Microsoft Research, USA
plonga@microsoft.com

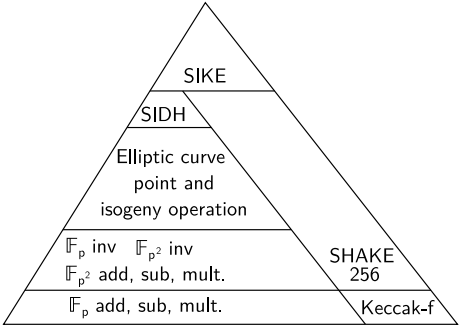
September, 2020

- Focused on the hardware architecture.
- No details on SIKE, Montgomery or more theoretical parts.
- Outline:
  1. What do we need for SIKE?
  2. Literature approach.
  3. Our approach.
  4. Results.

What do we need to make SIKE?

- Operations in $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$ with $x^2 + 1$ as primitive polynomial.
- p is a prime with size 434, 503, 610, 751, 964 bits.
- Montgomery elliptic curve operations.
- Tree traversing procedure.
- SHAKE256 (SHA-3).

How to tackle it



- Focus on fast and optimized hardware in the basis.
- Grow through hardware instructions.
- Grow even more through function calls, stacks and returns.

Our solution

<table>
<tr><td colspan="8">16 bits CPU</td></tr>
<tr><td rowspan="4">Keccak-f</td><td colspan="2">16 bits ALU</td><td colspan="2">ALU RAM</td><td rowspan="4">PROM
64x2048</td><td rowspan="4">Carmela
RAM
256x1024
(128x2048)</td><td colspan="2">Carmela</td></tr>
<tr><td rowspan="3">DSP</td><td rowspan="3">Barrel-
shifter</td><td rowspan="3">Base
RAM
16x1024</td><td rowspan="3">RD
Registers
32</td><td colspan="2">State machine controlled MACC
up to 1024 bits operands</td></tr>
<tr><td colspan="2">256 (128) MACC with 8/4 stages</td></tr>
<tr><td>256 (128) Multiplier</td><td>534 (288) Adder</td></tr>
</table>

- High level operations are solved through function calls, stacks and data arrays : CPU.
- Low level $\mathbb{F}_p$ operations are optimized on the Carmela co-processor.
- Keccak-f could be made on CPU, but in this case it is easy to integrate in the core.

- Koziel et al. different iterations:
  - Affine formulas with an optimized inversion unit [9]
  - Projective formulas, but the same architecture [7]
  - Frequency increase and results for 503, 751, 1019, 1533 bits [8]
  - Added SHAKE and give support for the SIKE operations in the SIKE proposal. [3]
- Only field arithmetic, no elliptic curve operations support, Karmakar et al. [5]
- SIDH with Montgomery multiplier using Redundant Number system, Roy and Mukhopadhyay [13].

Lets understand more

<table>
<tr><td colspan="8">16 bits CPU</td></tr>
<tr><td rowspan="4">Keccak-f</td><td colspan="2">16 bits ALU</td><td colspan="2">ALU RAM</td><td rowspan="4">PROM
64x2048</td><td rowspan="4">Carmela
RAM
256x1024
(128x2048)</td><td colspan="2">Carmela</td></tr>
<tr><td rowspan="3">DSP</td><td rowspan="3">Barrel-
shifter</td><td rowspan="3">Base
RAM
16x1024</td><td rowspan="3">RD
Registers
32</td><td colspan="2">State machine controlled MACC
up to 1024 bits operands</td></tr>
<tr><td colspan="2">256 (128) MACC with 8/4 stages</td></tr>
<tr><td>256 (128) Multiplier</td><td>534 (288) Adder</td></tr>
</table>

Let's focus on Carmela.

Carmela details

Carmela	
State machine controlled MACC up to 1024 bits operands	
256 (128) MACC with 8/4 stages	
256 (128) Multiplier	534 (288) Adder

8 $\mathbb{F}p$ mul 4 $\mathbb{F}p$ add, sub.
MACC 8 values Add/Sub 4 values

- Carmela is also a little complex.
- Lets talk each part separately.

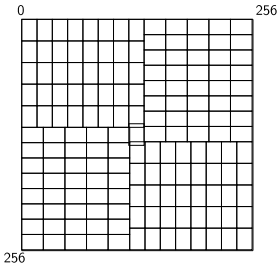
Carmela details

Carmela	
State machine controlled MACC up to 1024 bits operands	
256 (128) MACC with 8/4 stages	
256 (128) Multiplier	534 (288) Adder

8 $\mathbb{F}$p mul 4 $\mathbb{F}$p add, sub.
MACC 8 values Add/Sub 4 values

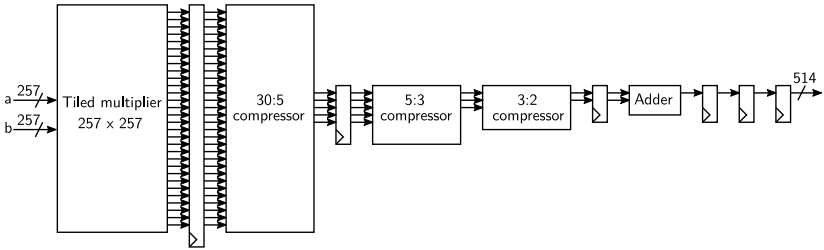
How big is the multiplier accumulator for 7 series?

- DSP48E = 18×25 signed multiplier or 17×24 unsigned.
- How much costs a 257 bits signed multiplier?
  - Square schoolbook : 256 DSP48E.
    - Create a multiplier with 16×16 unsigned and 17×17 signed.
  - Rectangular multiplier with tiling by Roy et al [12] : 161 DSP48E
    - Use the multipliers as 17×24 unsigned to construct a 256 unsigned multiplier, then a 257 signed.



Is the multiplier enough?

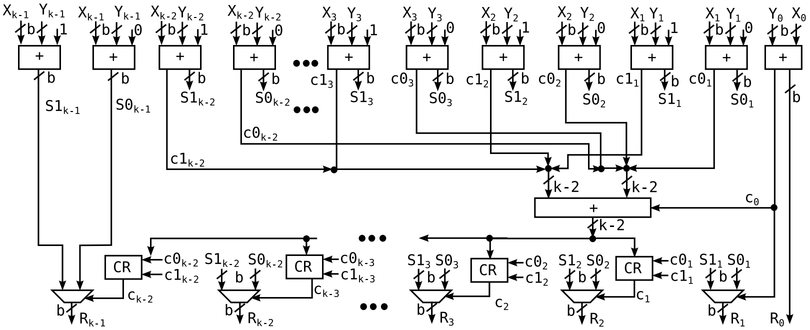
- Roy et al [12] does not fully resolve multiplication, but generates several partial products.
- How many? In 257 signed, it is 30 values.
- How to solve?
  - Compress values from 30 to 5, then compress 5 to 3 and finally 3 to 2.
  - At the end perform a final addition of the 2 values.



Carmela details

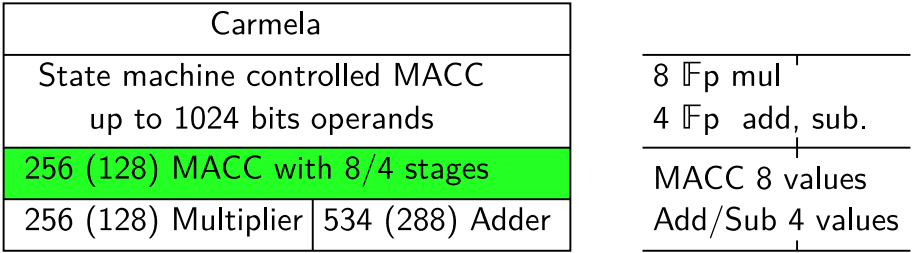
Carmela	
State machine controlled MACC up to 1024 bits operands	
256 (128) MACC with 8/4 stages	
256 (128) Multiplier	534 (288) Adder

8 $\mathbb{F}$p mul
4 $\mathbb{F}$p add, sub.
MACC 8 values
Add/Sub 4 values

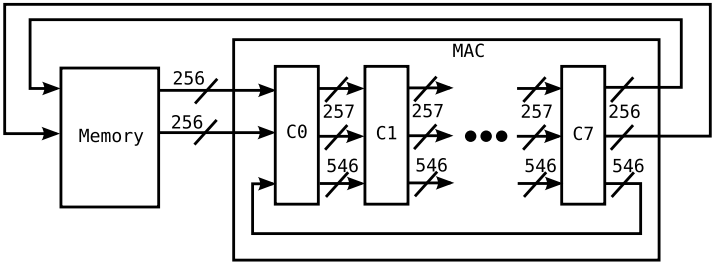


- For each bit an addition is performed with the two options of carries.
- Each addition carry out is used to solve the carry propagation with an adder of size $k-2$.
- In our case $b=2$, thus for 546 bits, $k=273$.

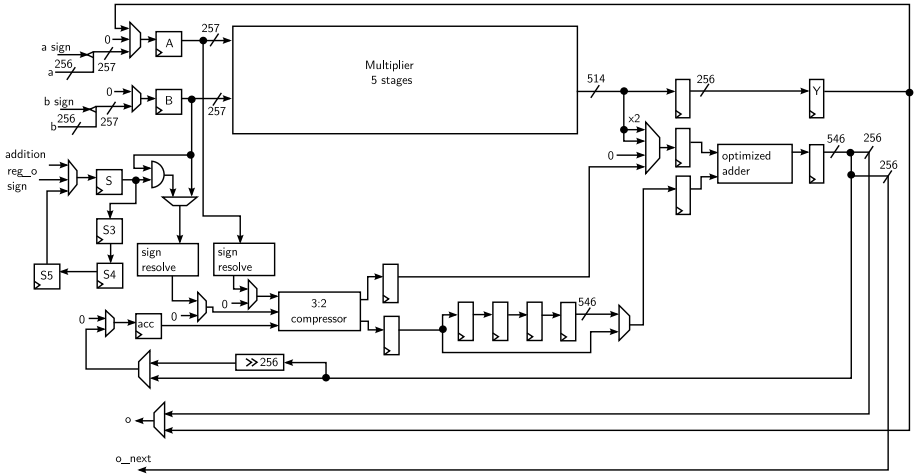
Carmela details



Proposal



- $c_i = a_i b_i + acc_i,\ i = 0,...,7$
- 8 parallel high level computations in the pipeline.
- 4 stages in case of addition/subtraction.
- Memory interface is 256 bits, and accumulator register is 546.
- Internal multiplier is 257 bits signed, thus it can behave as 256 unsigned and signed.


a sign
a
b sign
b
0
256
257
A
B
257
Multiplier
5 stages
514
256
Y
x2
0
optimized
adder
546
256
256
addition
reg_o
sign
S
S3
S4
S5
sign
resolve
sign
resolve
0
3:2
compressor
546
0
acc
0
≫256
o
o_next

Carmela details

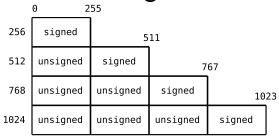
Carmela	
State machine controlled MACC up to 1024 bits operands	
256 (128) MACC with 8/4 stages	
256 (128) Multiplier	534 (288) Adder

8 $\mathbb{F}$p mul
4 $\mathbb{F}$p add, sub.
MACC 8 values
Add/Sub 4 values

Okay, but how do you perform $\mathbb{F}_p$ operations?

Given a 257 bits signed MACC, how do we proceed?

- Values can go to multiple 256 bits words:

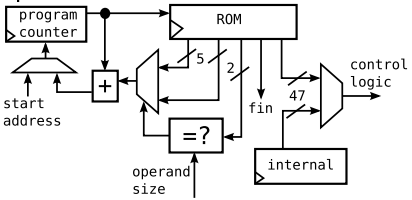


- Addition/Subtraction direct.
- Montgomery multiplication:
  - Signed representation [-p, p] instead of unsigned [0, p].
  - Product scanning (FIPS algorithm) manually unrolled for all 4 sizes.
  - 17 extra bits instead of only 2 to be able so addition/subtraction are not reduced.
- Extra operation to reduce from [-p, p] to [0, p - 1].
- Addition/Subtraction that keeps values between [-2p, 2p].

How to control all operations?

State machine + shift registers.

- Each MACC operation is a state, and for all unrolled operations we need around 300 (500) states.
- Given the states are instructions that follow a linear order, we made a special ROM based controller.



- Since all only addresses and sign change for the same 8/4 values, then we rotate addresses through registers.

Is this enough?

Our Coprocessor:

- Support $\mathbb{F}_p$ addition/subtraction and multiplication for primes up to 1022 bits.

Missing parts:

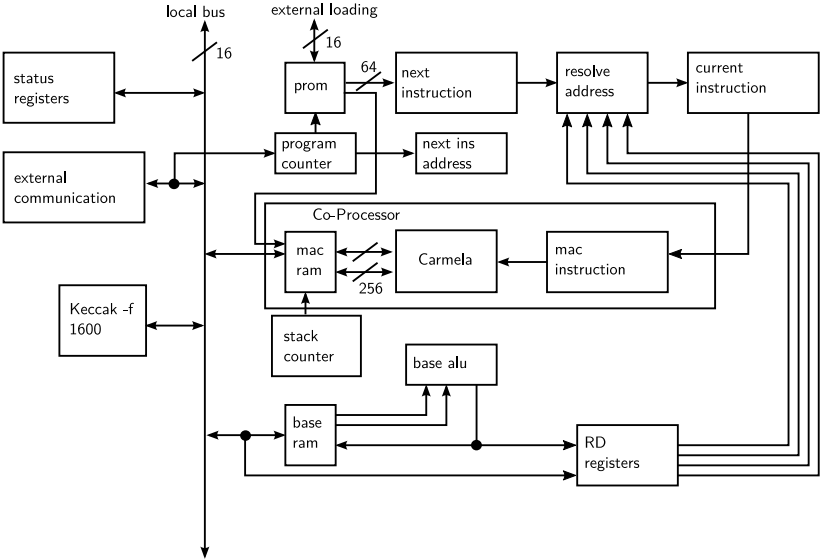
- $\mathbb{F}_p$ inversion.
- $\mathbb{F}_{p^2}$ operations.
- Montgomery elliptic curve operations.
- Tree traversing procedure.

The remainder

<table>
<tr><td colspan="7">16 bits CPU</td></tr>
<tr><td rowspan="4">Keccak-f</td><td colspan="2">16 bits ALU</td><td colspan="2">ALU RAM</td><td rowspan="4">PROM
64x2048</td><td rowspan="4">Carmela
RAM
256x1024
(128x2048)</td><td colspan="2">Carmela</td></tr>
<tr><td rowspan="3">DSP</td><td rowspan="3">Barrel-
shifter</td><td rowspan="3">Base
RAM
16x1024</td><td rowspan="3">RD
Registers
32</td><td colspan="2">State machine controlled MACC
up to 1024 bits operands</td></tr>
<tr><td colspan="2">256 (128) MACC with 8/4 stages</td></tr>
<tr><td>256 (128) Multiplier</td><td>534 (288) Adder</td></tr>
</table>

- 16 bits Harvard CPU with custom made simple ISA.
- Performs: add, sub, mul, shift, rot, logical, compare, jumps, conditional jumps, load and store, push and pop.
- It is cyclic based CPU, with pipelined address resolution.
- One memory region for main CPU values.
  - Registers share this region.
- One memory region for Carmela.

High level architecture



Results - SIKE

Reference	Par.	Slices	DSP	BRAM	Freq. (MHz)	Time (ms)	ST	DT
[7]†	p503	8 918	192	40.0	181.4	20.9	186	4.0
	p751	11 801	282	47.0	177.3	46.3	546	13.1
[8]†	p503	7 491	192	43.5	202.1	16.5	124	3.2
	p751	11 277	288	60.5	204.9	36.4	410	10.5
[13]†	p751	18 711	294	22.5	225.7	31.6	591	9.3
[3]	p751	16 756	376	56.5	198.0	33.4	560	12.6
[6]	p503	9 514	264	34.0	171.2	13.6	129	3.6
	p751	17 530	512	43.5	167.4	26.9	472	13.8
Ours (128)	pXXX	3 855	57	21.0	153.9	49.8	192	2.8 (p434)
						88.0	339	5.0 (p503)
						105.9	408	6.0 (p610)
						177.5	684	10.1 (p751)
Ours (256)	pXXX	8 131	162	39.0	141.6	24.4	198	4.0 (p434)
						49.9	405	8.1 (p503)
						52.0	423	8.4 (p610)
						61.0	496	9.9 (p751)

Results - Other Schemes

Reference	Algorithm	Slices	DSPs	BRAMs	Freq. (MHz)	Time (ms)
[11]	NewHope-Simple	1 708	2	4	125.0	2.9
	(Server/Client)	1 483	2	4	117.0	
[2]	BIKE	1 559	-	13	161.3	10.2
[4]	FrodoKEM-640	1 855	1	11	167.0	40.0
	(Encaps/Decaps)	1 992	1	16	162.0	
Ours (128)	**SIKE**	3 491	57	21	145.1	52.8 (p434) 93.4 (p503) 112.4 (p610) 188.3 (p751)
Ours (256)	**SIKE**	7 329	162	37	109.1	31.7 (p434) 64.8 (p503) 67.5 (p610) 79.2 (p751)

Results - ECC

Reference	Curve	Slices	DSPs	BRAMs	Freq. (MHz)	Time (ms)
[1]	**NIST**	1 704	20	-	225	1.5 (p256) 4.1 (p384) 9.7 (p521)
[1]	**NIST**	2 068	64	-	161	2.1 (p384) 5.0 (p521)
Ours (128)	**NIST**	3 855	57	21	153.9	1.3 (p224) 2.6 (p256) 6.2 (p384) 8.5 (p521)
Ours (256)	**NIST**	8 131	162	39	141.6	0.8 (p224) 1.6 (p256) 2.5 (p384) 4.6 (p521)

Changes between TCHES and eprint

- Added instruction for addition/subtraction with embedded reduction between [-2p, 2p].
- Fixed p503 bug.
- Increased the Montgomery multiplication optimized for SIKE primes to cover all sizes.

Thank You.

References I

D. Amiet, A. Curiger, and P. Zbinden.
Flexible FPGA-Based Architectures for Curve Point Multiplication over GF(p).
In *2016 Euromicro Conference on Digital System Design (DSD)*, pages 107–114, 2016.

N. Aragon, P. S. L. M. Barreto, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, S. Gueron, T. Güneysu, C. A. Melchor, R. Misoczki, E. Persichetti, N. Sendrier, J.-P. Tillich, V. Vasseur, , and G. Zémor.
Bit Flipping Key Encapsulation – Submission to Round 2 of NIST's Post-Quantum Cryptography Standardization Process, 2019.
Available at https://bikesuite.org.

Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, David Jao, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Geovandro Pereira, Joost Renes, Vladimir Soukharev, and David Urbanik.
Supersingular Isogeny Key Encapsulation – Submission to Round 2 of NIST's Post-Quantum Cryptography Standardization Process, 2019.
Available at https://sike.org.

James Howe, Tobias Oder, Markus Krausz, and Tim Güneysu.
Standard lattice-based key encapsulation on embedded devices.
IACR Trans. Cryptogr. Hardw. Embed. Syst., 2018(3):372–393, 2018.

Angshuman Karmakar, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede.
Efficient finite field multiplication for isogeny based post quantum cryptography.
In *Arithmetic of Finite Fields*, pages 193–207, Cham, 2016. Springer International Publishing.

References II

Brian Koziel, A.-Bon Ackie, Rami El Khatib, Reza Azarderakhsh, and Mehran Mozaffari Kermani.
SIKE'd up: Fast and secure hardware architectures for supersingular isogeny key encapsulation, 2019.
Available at https://eprint.iacr.org/2019/711.pdf.

Brian Koziel, Reza Azarderakhsh, and Mehran Mozaffari-Kermani.
Fast Hardware Architectures for Supersingular Isogeny Diffie-Hellman Key Exchange on FPGA.
In Orr Dunkelman and Somitra Kumar Sanadhya, editors, *Progress in Cryptology - INDOCRYPT 2016*, volume 10095 of *Lecture Notes in Computer Science*, pages 191–206. Springer, 2016.

Brian Koziel, Reza Azarderakhsh, and Mehran Mozaffari-Kermani.
A High-Performance and Scalable Hardware Architecture for Isogeny-Based Cryptography.
IEEE Transactions on Computers, pages 1594–1609, 2018.
https://doi.org/10.1109/TC.2018.2815605.

Brian Koziel, Reza Azarderakhsh, Mehran Mozaffari-Kermani, and David Jao.
Post-quantum cryptography on FPGA based on isogenies on elliptic curves.
IEEE Trans. on Circuits and Systems, 64-I(1):86–99, 2017.

H. D. Nguyen, B. Pasca, and T. B. Preußer.
Fpga-specific arithmetic optimizations of short-latency adders.
In *2011 21st International Conference on Field Programmable Logic and Applications*, pages 232–237, Sep. 2011.

Tobias Oder and Tim Güneysu.
Implementing the NewHope-Simple key exchange on low-cost FPGAs.
In Tanja Lange and Orr Dunkelman, editors, *Progress in Cryptology - LATINCRYPT 2017*, volume 11368 of *Lecture Notes in Computer Science*, pages 128–142. Springer, 2019.

References III

D. B. Roy, D. Mukhopadhyay, M. Izumi, and J. Takahashi.
Tile before multiplication: An efficient strategy to optimize DSP multiplier for accelerating prime field ECC for NIST curves.
In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2014.

Debapriya Basu Roy and Debdeep Mukhopadhyay.
Post Quantum ECC on FPGA Platform.
Cryptology ePrint Archive, Report 2019/568, 2019.
https://eprint.iacr.org/2019/568.

MACC - Instruction set

carmela flag	append	carmela internal	operand o		operand b				operand a			
63 62 01	61 ... 59 000	58 ... 55 type	54 Dir	53 ... 38 Mo	37 0	36 Dir	35 En	34 ... 19 Mb	18 Sign	17 Dir	16 En	15 ... 0 Ma

Carmela flag is to to tell the main processor to not execute and send to Carmela.
Carmela internal type :

- Multiplication/square no reduction (mmuld/msqud)
- Montgomery modular multiplication/square (mmulm/msqum)
- Addition/subtraction no reduction (madd_subd)
- Simple reduction (mitred)
- Simple reduction (madd_subr)

Ma, Mb, Mo : operand address.
Sign : indicate if it is addition or subtraction (Mo = Mb +/- Ma).
Dir :

- 0 - Direct access. Mem[Ma].
- 1 - Indirect access. Mem[Rd[Ma]].

High level ISA.

main flag	append	main internal	operand o		operand b				operand a			
63 62 00	61 0	60 ... 55 type	54 Dir	53 ... 38 memo	37 Sign	36 Dir	35 Cx	34 ... 19 memb	18 Sign	17 Dir	16 Cx	15 ... 0 mema

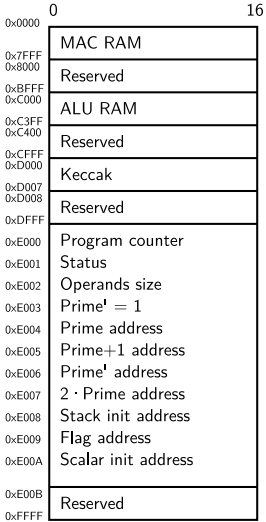
Main flag internal type :

- nop
- jump, jumpeq, jumpl, jumpeql
- push, pushf, pushm
- pop, popf, popm
- copy, copyf, copym, copya
- Iconstf, Iconstm
- call, ret, fin
- keccak_init, keccak_go
- badd, bsub, bsmul
- bshiftr, bshiftl
- brotr, brotl
- bland, blor, blxor, blnot

mema, memb, memo : operand address.
Cx : if the value in the operand is a constant.
Sign : if the operation is for signed integers or unsigned
Dir :

- 0 - Direct access. Mem[mema].
- 1 - Indirect access. Mem[Rd[mema]].

SIKE memory model



MACC memory model