Architecture level Optimizations for Kummer based HECC on FPGAs

Gabriel GALLIN – Turku Ozlum CELIK – Arnaud TISSERAND

CNRS – IRISA – Univ. Rennes – Lab-STICC

December, 11th
Indocrypt 2017

ECC, HECC, Kummer-HECC

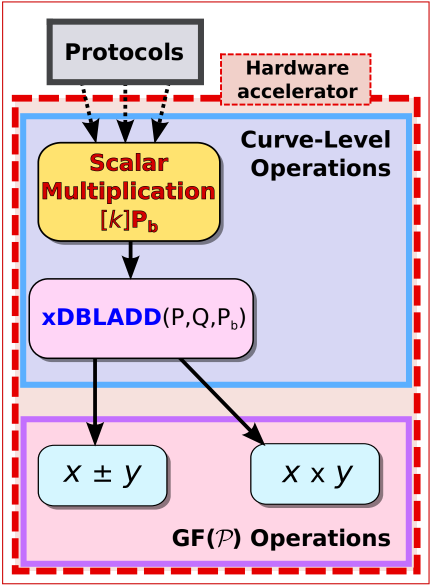
	size of $\mathtt{GF}(\mathcal{P})$ elems.	$\mathtt{ADD}$	$\mathtt{DBL}$	source
ECC	ℓ_{ECC}	$12\mathtt{M} + 2\mathtt{S}$	$7\mathtt{M} + 3\mathtt{S}$	[2]
HECC	$\ell_{\text{HECC}} \approx \frac{1}{2}\ell_{\text{ECC}}$	$40\mathtt{M} + 4\mathtt{S}$	$38\mathtt{M} + 6\mathtt{S}$	[7]
KHECC	ℓ_{HECC}	$19\mathtt{M} + 12\mathtt{S}$		[10]

Metric for algorithms efficiency: number of multiplications ($\mathtt{M}$) and squares ($\mathtt{S}$) in $\mathtt{GF}(\mathcal{P})$

Kummer-HECC (KHECC) is more efficient than ECC:

- Software implementations by Renes *et al.* at CHES 2016 [10]
- ARM Cortex M0: up to 75% clock cycles reduction for signatures
- AVR AT-mega: up to 32% cycles reduction for Diffie-Hellman

Operations Hierarchy in KHECC



- Protocols based on **scalar multiplication**
- Sequence of curve-level operation `xDBLADD`: $(\pm P, \pm Q, \pm(P-Q)) \rightarrow (\pm[2]P, \pm(P+Q))$
- Size of elements in $\mathtt{GF}(\mathcal{P})$: 128 bits
- Dedicated hyper-threaded multiplier [3]: 3 independent modular multiplications computed in parallel

Scalar Multiplication: Montgomery Ladder

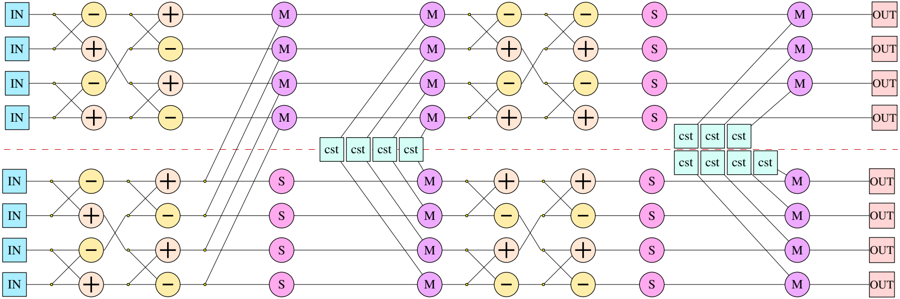
Montgomery ladder based *crypto_scalarmult* from [10]:

```
Require: $m$-bit scalar $k = \sum_{i=0}^{m-1} 2^i k_i$, point $P_b$, $cst \in \mathtt{GF}(\mathcal{P})^4$
Ensure: $V_1 = [k]P_b,\ V_2 = [k+1]P_b$
  $V_1 \leftarrow cst$
  $V_2 \leftarrow P_b$
  for $i = m-1$ downto $0$ do
    $(V_1, V_2) \leftarrow \mathtt{CSWAP}(k_i, (V_1, V_2))$
    $(V_1, V_2) \leftarrow \mathtt{xDBLADD}(V_1, V_2, P_b)$
    $(V_1, V_2) \leftarrow \mathtt{CSWAP}(k_i, (V_1, V_2))$
  end for
  return $(V_1, V_2)$
```

$\mathtt{CSWAP}(k_i, (X, Y))$ returns (X, Y) if $k_i = 0$, else (Y, X)

- Constant time, uniform operations (independent from key bits)
- CSWAP: very simple but handles secret bits (to be protected)

xDBLADD $\mathtt{GF}(\mathcal{P})$ Operation



- Some parallelism available (up to 8 $\mathtt{GF}(\mathcal{P})$ operations)
- Several possible hardware architectures can be implemented

Architectural Exploration

- Fast exploration and validation of numerous hardware architecture configurations with dedicated tools (*cf. paper*)

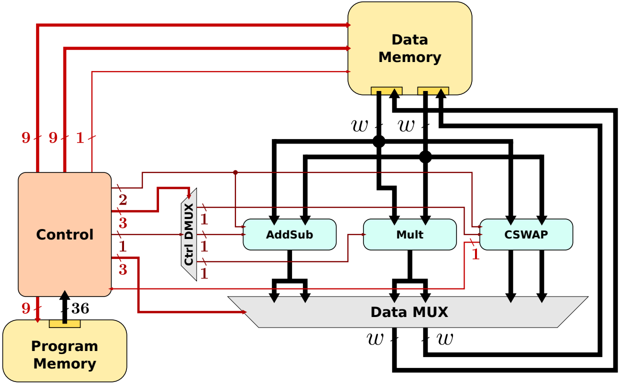
- Full implementation of 4 selected architectures

  A1: Smallest architecture
  A2: Modification of `CSWAP`
  A3: Doubled number of arithmetic units
  A4: Doubled number of units (arithmetic and `MEM`) in 2 clusters

- Width of `MEM` and interconnect to be selected: $w = 34$, 68 or 136 bits

Architecture A1: Base Solution

- Smallest accelerator: 1 `AddSub`, 1 `Mult`, 1 `MEM` and 1 `CSWAP`



FPGA	w [bit]	LUT	FF	logic slices	DSP slices	RAM blocks	freq. [MHz]	clock cycles	time [ms]
V4	**34**	**1010**	1833	1361	11	**4**	**322**	**194,614**	0.60
	68	**1750**	3050	2251	11	**5**	**305**	**186,911**	0.61
	136	**2281**	3028	1985	11	**7**	**266**	**184,337**	0.69
V5	34	757	1816	603	11	4	360	194,614	0.54
	68	1264	3033	908	11	5	360	186,911	0.52
	136	1582	3008	940	11	7	360	184,337	0.51
S6	34	1064	1770	408	11	4	278	194,614	0.70
	68	1555	2970	705	11	5	252	186,911	0.74
	136	1910	2994	747	11	7	221	184,337	0.83

- **Area increases** when w increases
- **Increased number of BRAMs** for large memories
- **Small clock cycles reduction** for larger w cancelled by **frequency drops**
- Small $w34$ more interesting for A1 architecture

Architecture A2: CSWAP Optimization

- Same architecture topology as A1:
  1 AddSub, 1 Mult, 1 MEM and 1 *modified* CSWAP

- Modified CSWAP unit implements new $\texttt{CSWAP}_{\texttt{V2}}$ operation:
  - Merged *consecutive* CSWAP *operations* of successive iterations

```
$(V_1, V_2) \leftarrow \texttt{CSWAP}_{\texttt{V2}}((0, k_{m-1}), (V_1, V_2))$
for $i = m - 1$ downto 1 do
    $(V_1, V_2) \leftarrow \texttt{xDBLADD}(V_1, V_2, P_b)$
    $(V_1, V_2) \leftarrow \texttt{CSWAP}_{\texttt{V2}}((k_i, k_{i-1}), (V_1, V_2))$
end for
```

  - Swaps $V1$ and $V2$ if $k_i \neq k_{i-1}$ (only one *xor* gate needed)

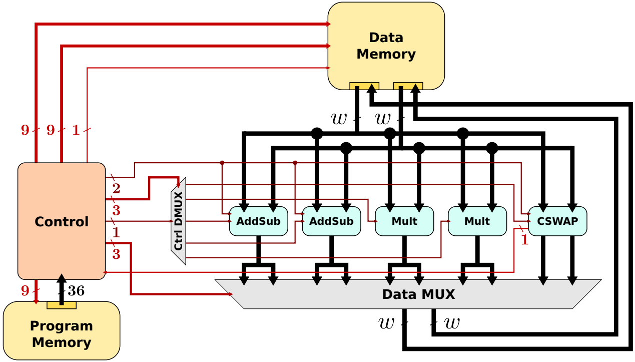
- CSWAP unit has constant time behavior

FPGA	w [bit]	LUT	FF	logic slices	DSP slices	RAM blocks	freq. [MHz]	clock cycles	time [ms]
V4	**34**	**872**	1624	1121	11	4	330	**184,374**	0.56
	68	**1556**	2637	1978	11	5	290	**183,071**	0.63
	136	**2161**	3027	2100	11	7	**327**	**183,057**	0.56
V5	34	722	1605	541	11	4	360	184,374	0.51
	68	1196	2620	840	11	5	360	183,071	0.51
	136	1419	3009	944	11	7	360	183,057	0.51
S6	34	940	1559	381	11	4	293	184,374	0.63
	68	1503	2565	553	11	5	262	183,071	0.70
	136	1890	2981	667	11	7	**283**	183,057	0.65

- Less $\texttt{CSWAP}_{\texttt{V2}}$ operations ⇒ **slightly less clock cycles** than in A1
- Simplified management of $\texttt{CSWAP}_{\texttt{V2}}$ operations
  - **Slightly higher frequencies**, with smaller variations
  - **Slightly reduced area** (LUTs and FFs)
- A2 slightly more interesting than A1 both for speed and area ($\sim 10\%$)
- Small w34 still the best configuration

Architecture A3: Large Architecture

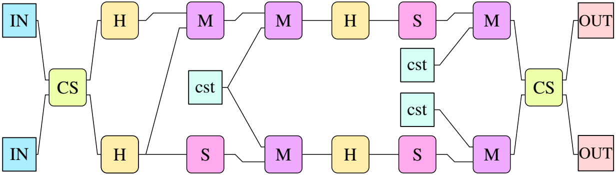
- Doubled number of $\mathtt{GF}(\mathcal{P})$ units: 2 `AddSub`, 2 `Mult`
- More $\mathtt{GF}(\mathcal{P})$ operations in parallel: up to *6 multiplications*



FPGA	w	LUT	FF	logic	DSP	RAM	freq.	clock	time
	[bit]			slices	slices	blocks	[MHz]	cycles	[ms]
V4	**34**	**1462**	2611	1783	**22**	**6**	**294**	188,218	0.64
	68	**2802**	4367	3468	**22**	**7**	282	**124,191**	0.44
	136	**3768**	5017	3660	**22**	**9**	**285**	**119,057**	0.42
V5	34	1262	2607	921	22	6	358	188,218	0.53
	68	2290	4403	1409	22	7	345	124,191	0.36
	136	2737	4978	1594	22	9	348	119,057	0.34
S6	34	1527	2503	668	22	6	**265**	188,218	0.71
	68	2421	4267	1020	22	7	**225**	124,191	0.55
	136	3007	4877	1131	22	9	**225**	119,057	0.53

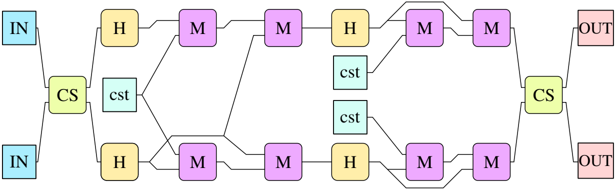
- **+60–90% LUTs**, **11 DSP slices, + 2 BRAMs** compared to A2
- **Frequency drops** on V4 ($< 13\%$) and S6 ($< 20\%$)
- **– 34–36% clock cycles for** $w68$ **and** $w136$, compared to $w34$
- 25 to 35% reduced computation time for $w136$ depending on FPGA
- A3 faster than A2, but larger → area – speed trade-offs

Architecture A4: Clustered Architecture



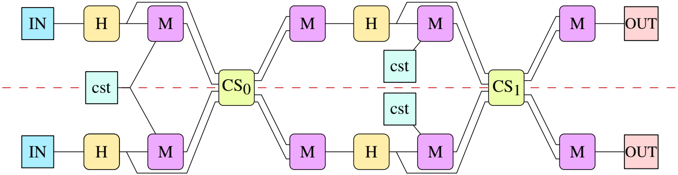
- Decomposition of `xDBLADD` into two symmetric clusters of $\mathtt{GF}(\mathcal{P})$ operations

Architecture A4: Clustered Architecture



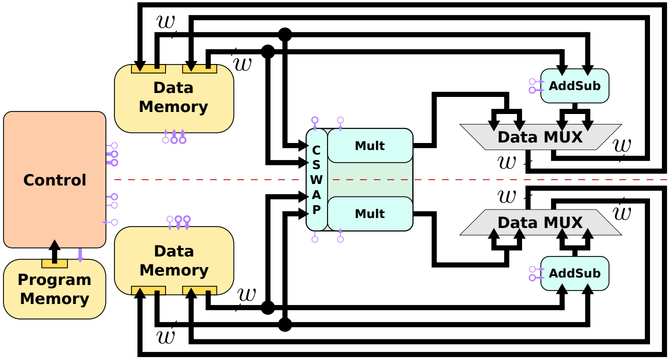
- Decomposition of `xDBLADD` into two symmetric clusters of $\mathtt{GF}(\mathcal{P})$ operations
- Modifications of `xDBLADD`:
  - Squares → multiplications
  - No impact on mathematical behavior nor on operations count

Architecture A4: Clustered Architecture



- Decomposition of `xDBLADD` into two symmetric clusters of $\mathtt{GF}(\mathcal{P})$ operations
- Modifications of `xDBLADD`:
  - Squares → multiplications
  - No impact on mathematical behavior nor on operations count
- New modification of `CSWAP`: $\mathtt{CSWAP}_{\mathtt{V3}}$
  - Replaced by two new swapping operations
  - $\mathtt{CS}_0(A, B, C, D) \rightarrow (A, B, C, B)$ if $k_i = 0$ else (C, D, A, D)
  - $\mathtt{CS}_1(A, B, C, D) \rightarrow (A, B, C, D)$ if $k_i = 0$ else (C, D, A, B)

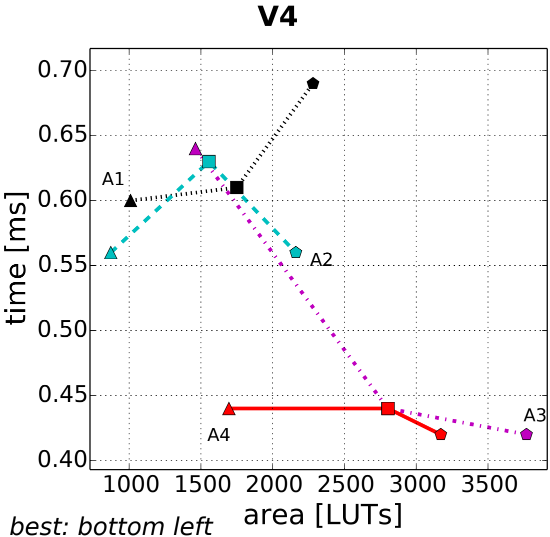
- Same number of GF($\mathcal{P}$) units as in A3: 2 AddSub, 2 Mult
- Doubled number of MEM : one for each hardware cluster
- CSWAP unit : “bridge” to exchange data between clusters
- Same control for both clusters (reduced complexity)

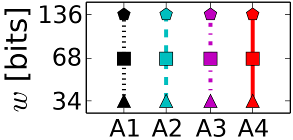


FPGA	w	LUT	FF	logic	DSP	RAM	freq.	clock	time
	[bit]			slices	slices	blocks	[MHz]	cycles	[ms]
V4	34	**1695**	**2950**	**2158**	22	**7**	324	**142,119**	0.44
	68	2804	4282	3184	22	**9**	290	128,021	0.44
	136	3171	4994	3337	22	**13**	299	125,456	0.42
V5	34	1370	2953	1013	22	7	358	142,119	0.40
	68	2095	4259	1358	22	9	337	128,021	0.38
	136	2514	4952	1589	22	13	313	125,456	0.40
S6	34	1564	2089	758	22	7	262	142,119	0.54
	68	2387	4030	1060	22	9	239	128,021	0.54
	136	3181	4786	1136	22	13	251	125,456	0.50

- **Increased area for** $w34$ compared to A3
- **Increased number of BRAMs** for additional `MEM`
- **Less clock cycles for** $w34$ ⇒ `MEM` bottleneck in small configurations
- A4 better than A3 for small configuration $w34$

Trade-offs for our Architectures A1–4

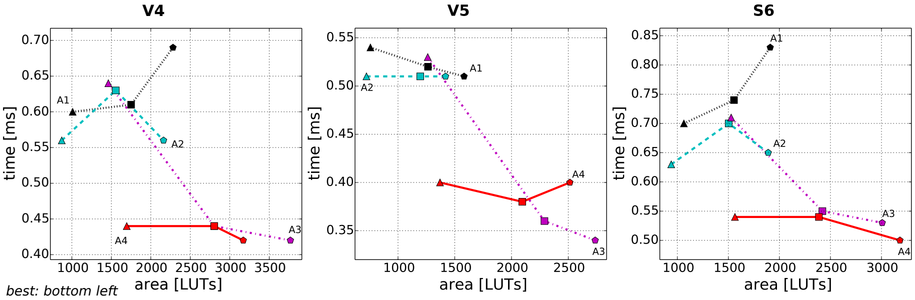




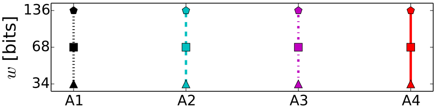
archi.	A1	A2	A3	A4
#`Mult`	1	1	2	2
#`AddSub`	1	1	2	2
#`CSWAP`	1	1	1	1
#`MEM`	1	1	1	2

best: bottom left

Trade-offs for our Architectures A1–4



best: bottom left



archi.	A1	A2	A3	A4
#Mult	1	1	2	2
#AddSub	1	1	2	2
#CSWAP	1	1	1	1
#MEM	1	1	1	2

Comparisons with ECC State-of-the-Art

year	ref.	target	$\mathcal{P}$	LUT	FF	logic slices	DSP slices	RAM blocks	freq. [MHz]	time [ms]
2008	[4]	XC4VFX12	NIST-256	2589	2028	1715	32	11	490	0.50
		XC4VFX12	NIST-256	34896	32430	24574	512	176	375	0.04
2014	[1]	XC6VFX760	NIST-256	32900	n.a.	11200	289	128	100	0.40
2012	[6]	XC4VFX12	GEN-256	n.a.	n.a.	2901	14	n.a.	227	1.09
		XC5VLX110	GEN-256	n.a.	n.a.	3657	10	n.a.	263	0.86
2013	[8]	XC4VLX100	GEN-256	5740	4876	4655	37	11	250	0.44
		XC5VLX110T	GEN-256	4177	4792	1725	37	10	291	0.38
2017	A4(w34)	XC4VLX100	GEN-128	1695	2950	2158	22	7	324	0.44
		XC5VLX110T	GEN-128	1370	2953	1013	22	7	358	0.40

Conclusion and Perspectives

- Kummer-HECC efficient alternative to ECC in hardware:
  - Halved area for same computation time
  - Scalar multiplication 40% faster for same area cost

  compared to equivalent state-of-the-art solutions for ECC

- Exploration of new architectures: topology, control, protection against SCA

- Release of VHDL codes and exploration tools under open-source license (by the end of Spring)

References I

[1] H. Alrimeih and D. Rakhmatov.
Fast and flexible hardware support for ECC over multiple standard prime fields.
IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 22(12):2661–2674, January 2014.

[2] D. J. Bernstein and T. Lange.
Explicit-formulas database.
`http://hyperelliptic.org/EFD/`.

[3] G. Gallin and A. Tisserand.
Hyper-threaded multiplier for HECC.
In *Proc. 51st Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, CA, USA, October 2017. IEEE.

[4] T. Güneysu and C. Paar.
Ultra high performance ECC over NIST primes on commercial FPGAs.
In *Proc. 10th Conf. Cryptographic Hardware and Embedded Systems (CHES)*, volume 5154 of *LNCS*, pages 62–78. Springer, August 2008.

[5] C. K. Koc, T. Acar, and B. S. Kaliski.
Analyzing and comparing Montgomery multiplication algorithms.
IEEE Micro, 16(3):26–33, June 1996.

[6] J.-Y. Lai, Y.-S. Wang, and C.-T. Huang.
High-performance architecture for elliptic curve cryptography over prime fields on FPGAs.
Interdisciplinary Information Sciences, 18(2):167–173, 2012.

[7] T. Lange.
Formulae for arithmetic on genus 2 hyperelliptic curves.
Applicable Algebra in Eng., Communication and Computing, 15(5):295–328, February 2005.

References II

[8] Y. Ma, Z. Liu, W. Pan, and J. Jing.
A high-speed elliptic curve cryptographic processor for generic curves over GF(p).
In *Proc. 20th International Workshop on Selected Areas in Cryptography (SAC)*, volume 8282 of *LNCS*, pages 421–437, Burnaby, BC, Canada, August 2013. Springer.

[9] P. L. Montgomery.
Modular multiplication without trial division.
Mathematics of Computation, 44(170):519–521, April 1985.

[10] J. Renes, P. Schwabe, B. Smith, and L. Batina.
μKummer: Efficient hyperelliptic signatures and key exchange on microcontrollers.
In B. Gierlichs and A. Y. Poschmann, editors, *Proc. 18th International Conference on Cryptographic Hardware and Embedded Systems (CHES)*, volume 9813 of *LNCS*, pages 301–320, Santa Barbara, CA, USA, August 2016. Springer.

This work is funded by **H-A-H** project

Thank you for your attention

Comparisons of A4 with ECC State-of-the-Art (%)

year	ref.	target	$\mathcal{P}$	LUT	FF	logic slices	DSP slices	RAM blocks	freq. [MHz]	time [ms]
2008	[4]	**XC4VFX12**	NIST-256	-35%	+46%	+26%	-31%	-36%	-34%	-12%
		XC4VFX12	NIST-256	-95%	-91%	-91%	-96%	-96%	-14%	+1000%
2014	[1]	**XC6VFX760**	NIST-256	-96%	.n.a.	-91%	-92%	-95%	+258%	+0%
2012	[6]	**XC4VFX12**	GEN-256	n.a.	n.a.	-26%	+57%	n.a.	+43%	-60%
		XC5VLX110	GEN-256	n.a.	n.a.	-72%	+120%	n.a.	+36%	-53%
2013	[8]	**XC4VLX100**	GEN-256	-70%	-39%	-54%	-41%	-36%	+30%	+0%
		XC5VLX110T	GEN-256	-67%	-38%	-41%	-41%	-30%	+23%	+5%
2017	A4(w34)	**XC4VLX100**	GEN-128	1695	2950	2158	22	7	324	0.44
		XC5VLX110T	GEN-128	1370	2953	1013	22	7	358	0.40

Architecture Level Modeling

- Problems when exploring solutions space:
    - Many parameters: type/number of units, communications, control, ...
    - Description in VHDL and debug of accelerators is time consuming

- Proposed solution: **hierarchical description** of accelerators
    - Allows fast exploration and validation of numerous solutions
    - Based on a library of units, fully described and implemented in VHDL
    - **CCABA** model defined for high-level description of accelerators

Units

- Multiplier `Mult` using HTMM_BRAM for multiplications and squares
- Adder-Subtractor `AddSub`
- Datapath width $w_{\text{arith}} = 34$ bits selected for `Mult` and `AddSub` after experimentations
- Swapping unit `CSWAP` with local key management and uniform behavior
- Memory `MEM` based on dual port RAMs with width w to be selected between 34, 68 or 136 bits

Accelerator Control and Interconnect

- Instantiate requiered units
- Interconnect all units
  - Based on multiplexors
  - Width to be selected: $w = 34$, 68 or 136 bits
- Control
  - Based on a tiny 36-bit instructions set architecture
  - Scalar bits managed only in `CSWAP` unit:
    control signals do not handle or depends on secret key

Most interesting FPGA implementation results

archi.	w [bit]	target	logic slices	DSP blocks	RAM blocks	freq. [MHz]	time [ms]
A2	34	V4	1121	11	4	330	0.56
A3	136		3660	22	9	285	0.42
A4	34		2158	22	7	324	0.44
A2	34	V5	541	11	4	360	0.51
A3	136		1594	22	9	348	0.34
A4	34		1013	22	7	358	0.40
A2	34	S6	381	11	4	293	0.63
A3	136		1131	22	9	225	0.53
A4	34		758	22	7	262	0.54

Instructions Set

instruc.	description
`read`	transfer operands from memory to target unit and start computation
`write`	transfer result from target unit to memory
`wait`	wait for `immediate` clock cycles
`nop`	no operation (1 clock cycle)
`jump`	change program counter (PC) to `immediate` code address
`end`	trigger the end of the scalar multiplication

4-bit opcode
3-bit unit index
2-bit operation mode
two 9-bit memory addresses
9-bit immediate value.

Memory and Internal Communication Width Configurations

config.	w [bit]	s [word]	cycle(s) / mem. op.	BRAM(s)
$w34$	34	4	4	1
$w68$	68	2	2	2
$w136$	136	1	1	4