# Hyper-Threaded Multiplier for HECC

Gabriel GALLIN and Arnaud TISSERAND

CNRS – Lab-STICC – IRISA
HAH Project

Asilomar, Oct. 2017

# Public-Key Cryptography (PKC)

- Provides cryptographic primitives such as digital signature, key exchange and specific encryption schemes
- First PKC standard: RSA
  - $\geq$ 2000-bit keys recommended today
  - Too costly for embedded applications
- Elliptic Curve Cryptography (ECC):
  - Better performances and lower cost than RSA
  - Allows more advanced schemes
- Hyper-Elliptic Curve Cryptography (HECC):
  - Evolution of ECC focusing on larger sets of curves
  - Supposed to have a smaller cost than ECC

# ECC, HECC, Kummer-HECC

| | $\mathbb{F}_{\mathcal{P}}$ elements size | ADD | DBL | source |
|---|---|---|---|---|
| ECC | $\ell_{\mathrm{ECC}}$ | $12\mathtt{M} + 2\mathtt{S}$ | $7\mathtt{M} + 3\mathtt{S}$ | [1] |
| HECC | $\ell_{\mathrm{HECC}} \approx \frac{1}{2}\ell_{\mathrm{ECC}}$ | $40\mathtt{M} + 4\mathtt{S}$ | $38\mathtt{M} + 6\mathtt{S}$ | [5] |
| Kummer | $\ell_{\mathrm{HECC}}$ | $19\mathtt{M} + 12\mathtt{S}$ | | [8] |

- ECC:
  - Size of $\mathbb{F}_{\mathcal{P}}$ elements $2\times$ larger
  - Simpler ADD and DBL operations
- HECC:
  - Smaller $\mathbb{F}_{\mathcal{P}}$
  - More operations in $\mathbb{F}_{\mathcal{P}}$ for ADD / DBL
- Kummer-HECC is more efficient than ECC [8]:
  - ARM Cortex M0: up to 75% clock cycles reduction for signatures
  - AVR AT-mega: up to 32% cycles reduction for Diffie-Hellman

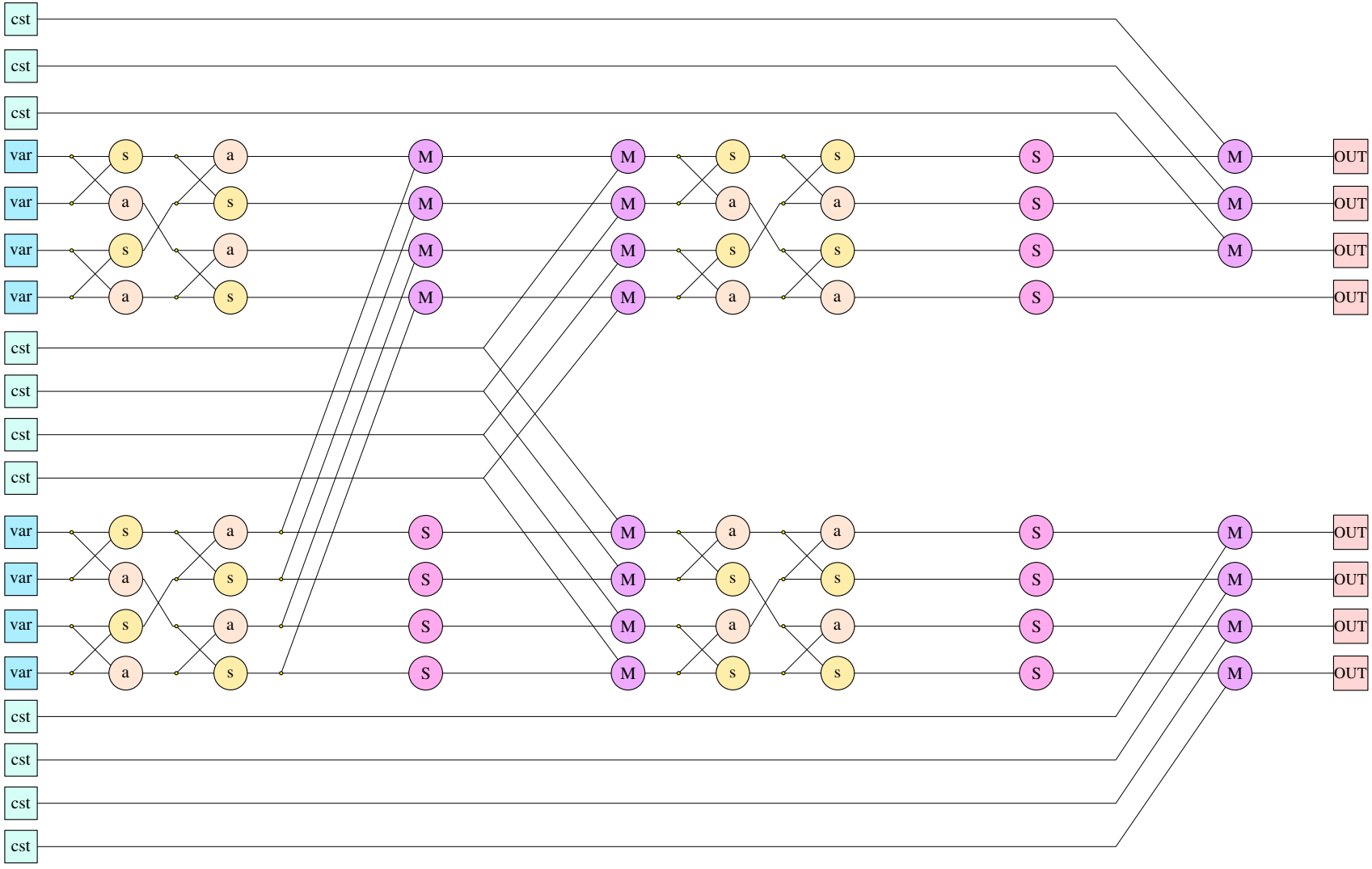
M multiplication, S square on field $\mathbb{F}_{\mathcal{P}}$

# Curve-Level Operations in Kummer

- No ADD operation but still DBL
- *Differential addition*: $\texttt{xADD}(\pm P, \pm Q, \pm(P-Q)) \to \pm(P+Q)$
- xADD and DBL can be combined:
  $\texttt{xDBLADD}(\pm P, \pm Q, \pm(P-Q)) \to (\pm[2]P, \pm(P+Q))$

For details see [8], [3] and [2]

# xDBLADD $\mathbb{F}_{\mathcal{P}}$ Operations



# Scalar Multiplication

Montgomery ladder based *crypto_scalarmult* [8]:

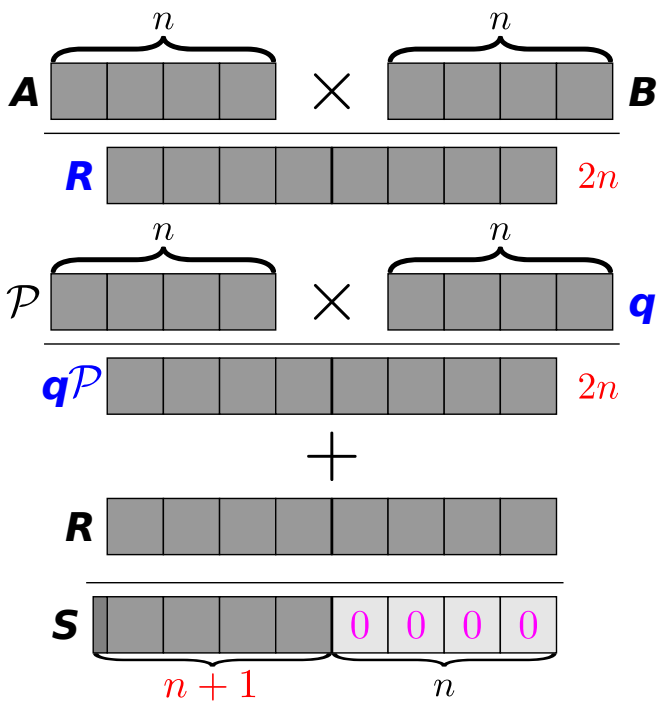
**Require:** $m$-bit scalar $k = \sum_{i=0}^{m-1} 2^i k_i$, point $P_b$, $cst \in \mathbb{F}_{\mathcal{P}}^4$
**Ensure:** $V_1 = [k]P_b$, $V_2 = [k+1]P_b$
  $V_1 \leftarrow cst$
  $V_2 \leftarrow P_b$
  **for** $i = m-1$ **downto** $0$ **do**
    $(V_1, V_2) \leftarrow \texttt{CSWAP}(k_i, (V_1, V_2))$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

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

- Constant time, uniform operations (independent from key bits)
- Some parallelism between xDBLADD internal $\mathbb{F}_{\mathcal{P}}$ operations
- CSWAP: very simple but involves secret bits (to be protected)

# Montgomery Modular Multiplication (MMM)

$$
\begin{array}{lll}
R = A \times B & & n \times n \rightarrow 2n \text{ bits} \\
q = (R \times (-\mathcal{P}^{-1})) & \mod (2^n) & n \times n \rightarrow \ \ n \text{ bits} \\
q\mathcal{P} = q \times \mathcal{P} & & n \times n \rightarrow 2n \text{ bits}
\end{array}
$$




- Objective: $A \times B \bmod \mathcal{P}$
- Proposed in [7]
- Variants are actual state-of-the-art for $\mathbb{F}_{\mathcal{P}}$ multiplication (with generic $\mathcal{P}$)
- Final reduction step discards $n$ LSBs

# Modular Multiplication: Dependencies Problem

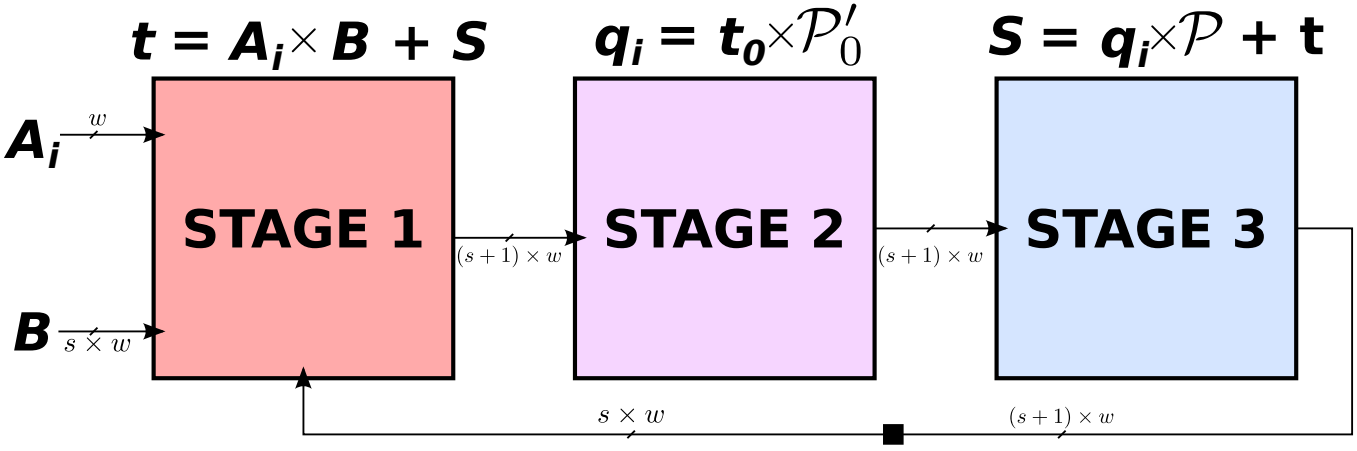
- In practice, MMM is interleaved
  - Operands are split into $s$ words of $w$ bits such that $n = s \times w$
  - Iterations over partial products and reductions on words
  - *Coarsely Integrated Operand Scanning* (CIOS) from [4]

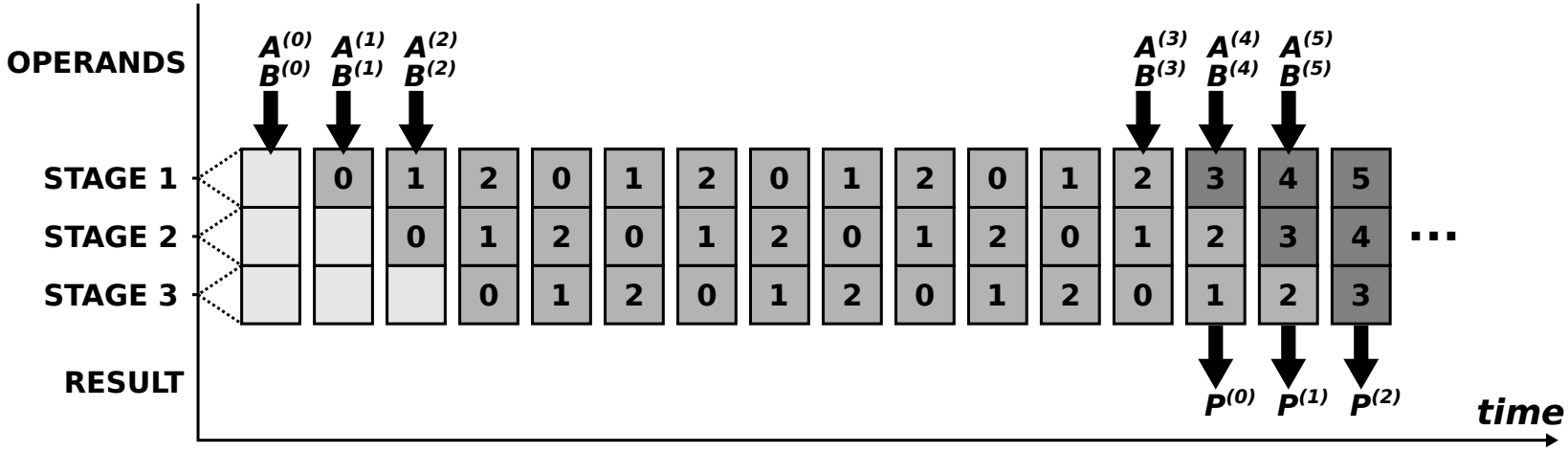
- Impact on hardware implementation
  - Dependencies $\rightarrow$ latencies between internal iterations
  - Hardware pipeline in DSP slices cannot be filled efficiently

- Proposed solution: Hyper-Threaded Modular Multiplier (HTMM)
  - Based on simple CIOS algorithm
  - Use idle stages to compute other independent MMMs in parallel

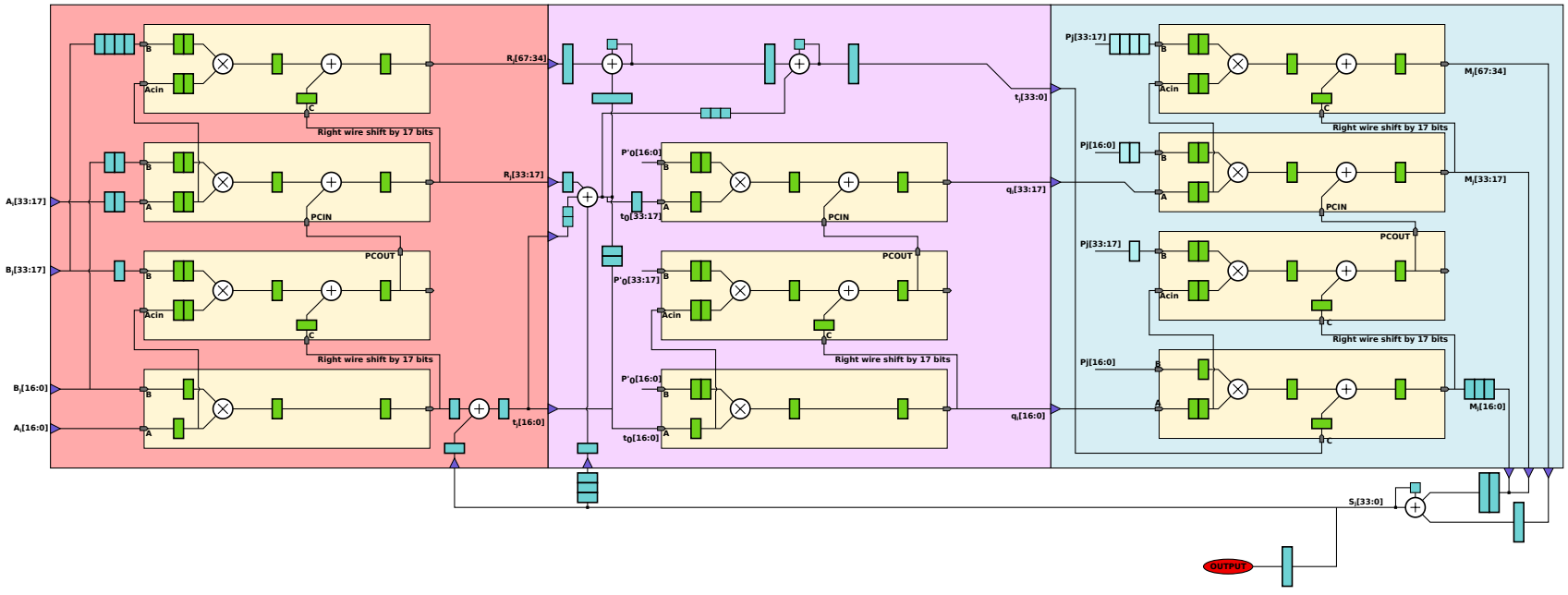
# HTMM Internal Architecture

- HTMM architecture: 3 hardware stages
  - Stages are fully pipelined (several clock cycles per stage)
  - 3 to 4 DSP slices in each stage



# HTMM Internal Architecture (details)



# HTMM Implementations

- Xilinx FPGAs
  - Virtex 4 XC4VLX100 (V4)
  - Virtex 5 XC5VLX110T (V5)
  - Spartan 6 XC6SLX75 (S6)
- Comparison with fastest MMM implementation in literature
  - Design presented in [6]
  - Implemented on the same FPGAs for fair comparison
- 2 versions of HTMM:
  - HTMM_DRAM : operands stored in FPGA slices (LUTs)
  - HTMM_BRAM : operands stored in FPGA BRAMs
- Parameters for HTMM:
  - $\mathcal{P} \rightarrow$ 128 bits
  - $w = 34$ bits, $s = 4$
  - Operands size $n = s \times w = 134$ bits

# HTMM Implementations Results

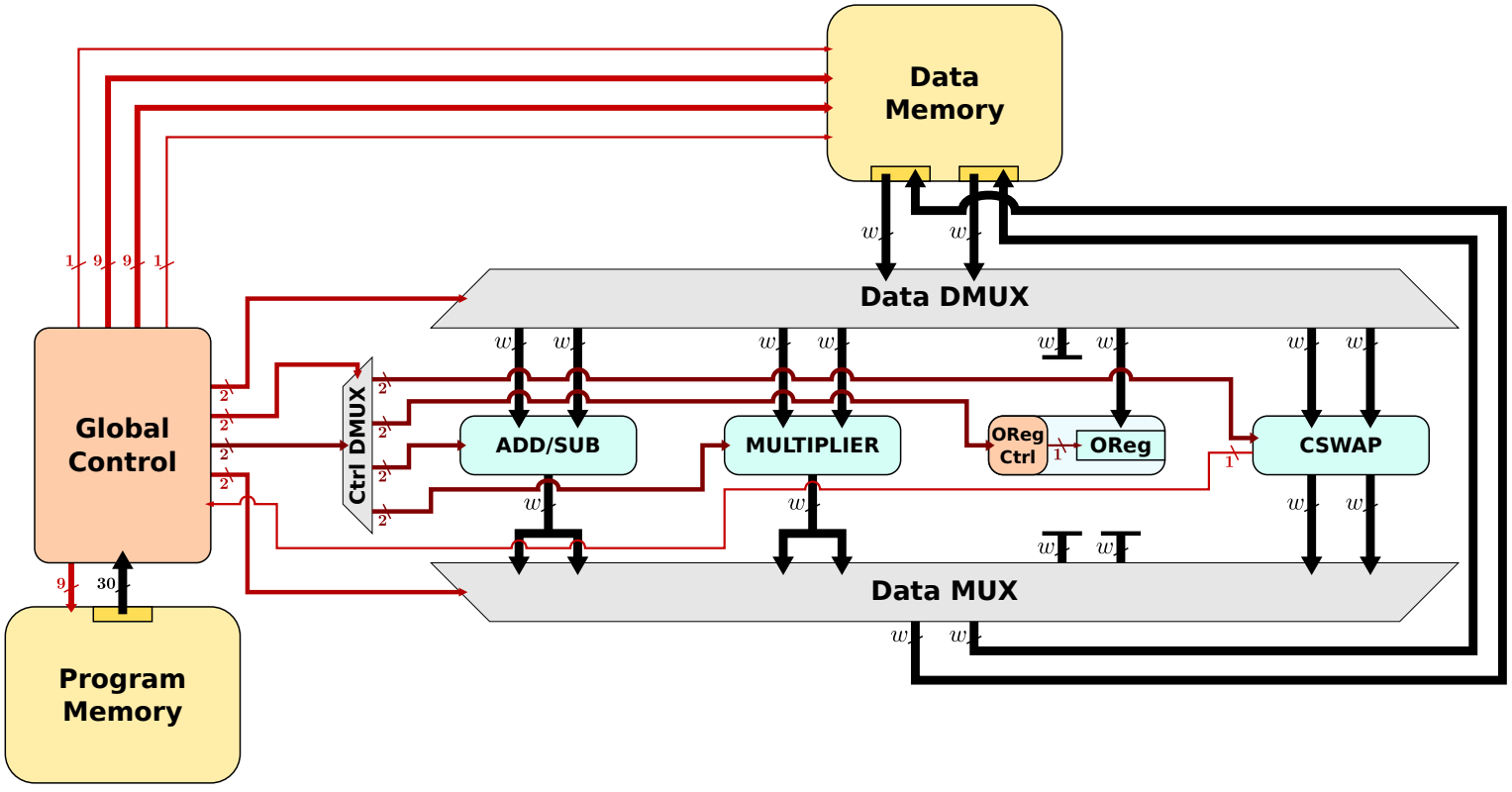
Results for 3 independent multiplications:

| Unit | FPGA | DSP | BRAM 18K/9K | FF | LUT | Slices | Freq. (MHz) | Nb. cycles | Time (ns) |
|---|---|---|---|---|---|---|---|---|---|
| [6] | V4 | 21 | 6/0 | 1311 | 1201 | 879 | 252 | | 258 |
| | V5 | 21 | 6/0 | 1310 | 1027 | 406 | 296 | 65 | 220 |
| | S6 | 21 | 0/6 | 1280 | 1600 | 540 | 210 | | 309 |
| HTMM | V4 | 11 | 0/0 | 1638 | 1128 | 1346 | 330 | | 239 |
| DRAM | V5 | 11 | 0/0 | 1616 | 652 | 517 | 400 | 79 | 198 |
| | S6 | 11 | 0/0 | 1631 | 1344 | 483 | 302 | | 261 |
| HTMM | V4 | 11 | 2/0 | 615 | 364 | 449 | 328 | | 241 |
| BRAM | V5 | 11 | 2/0 | 593 | 371 | 249 | 357 | 79 | 221 |
| | S6 | 11 | 0/2 | 587 | 359 | 180 | 304 | | 260 |

S6: -47% DSPs, -66% BRAMs, -66% slices, -15% duration

For only 1 single `M`, HTMM is less efficient (69 cycles against 25)

# Typical Architecture Model




Parameters specified at design time:

- Width $w$ and nb. words $s$ for internal communications ($s \times w = n$)
- Types and number of units

# 256b ECC vs 128b HECC (similar theoretical security)

| FPGA | Version | DSP | BRAM 18K | Slices | Freq. (MHz) | Nb. cycles | Time (ms) |
|---|---|---|---|---|---|---|---|
| V4 | ECC | 37 | 11 | 4655 | 250 | 109,297 | 0.44 |
| | H1 | 11 | 7 | 1413 | 330 | 183,051 | 0.55 |
| | H2 | 22 | 9 | 2356 | 330 | 115,211 | 0.35 |
| V5 | ECC | 37 | 10 | 1725 | 291 | 109,297 | 0.38 |
| | H1 | 11 | 7 | 873 | 360 | 183,051 | 0.51 |
| | H2 | 22 | 9 | 1542 | 360 | 115,211 | 0.32 |

Gain H1 on V5: -70% DSPs, -30% BRAMs, -49% slices, +30% duration

Gain H2 on V5: -40% DSPs, -10% BRAMs, -10% slices, -15% duration

ECC results from [6]

# Conclusions and Perspectives

- HTMM is more efficient than state of the art for 3 independent MMs
  - leads to better area / computation time trade-offs
  - more hardwired resources are active at each clock cycle

- $\mu$Kummer based HECC is an efficient alternative to ECC
  - More complex formulas but larger internal parallelism
  - Large exploration space for architectures and arithmetic

- Future works
  - Study other HTMM versions
  - Study hyper-threaded schemes impact on energy consumption
  - Study hyper-threaded schemes impact on side-channel leakage

# References


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

[2] Joppe W. Bos, Craig Costello, Huseyin Hisil, and Kristin Lauter.
Fast cryptography in genus 2.
*Journal of Cryptology*, 29(1):28–60, January 2016.

[3] Pierrick Gaudry.
Fast genus 2 arithmetic based on theta functions.
*Journal of Mathematical Cryptology*, 1(3):243–265, 2007.

[4] Çetin K. Koç, Tolga Acar, and Burton S. Kaliski, Jr.
Analyzing and comparing Montgomery multiplication algorithms.
*Micro, IEEE*, 16(3):26–33, June 1996.

[5] T. Lange.
Formulae for Arithmetic on Genus 2 Hyperelliptic Curves.
*Applicable Algebra in Engineering, Communication and Computing*, 15(5):295–328, February 2005.

[6] Yuan Ma, Zongbin Liu, Wuqiong Pan, and Jiwu 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. Springer, August 2013.

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

[8] Joost Renes, Peter Schwabe, Benjamin Smith, and Lejla Batina.
$\mu$Kummer: Efficient hyperelliptic signatures and key exchange on microcontrollers.
In *Proc. Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, volume 9813 of *LNCS*, pages 301–320. Springer, August 2016.