Generic Hardware Implementations of AMNS Arithmetic

Louis NOYEZ[1] Fangan Yssouf DOSSO[1]
Nadia EL MRABET[1] Olivier POTIN[1]
Pascal VERON[2] Laurent-Stéphane DIDIER[2]

May 2022

International Workshop on Cryptographic Architectures Embedded in Logic Devices.

[1]Ecole des Mines de Saint-Etienne - SAS
[2]Université de Toulon - IMATH

Outline

1. The AMNS representation
   - External Reduction
   - Internal Reduction
2. Hardware FPGA implementations
   - Coefficient multiplications
   - Small multiplications
   - Modular AMNS multiplication
3. Results and conclusions
   - Comparison against state of the art
   - Perspectives

AMNS representation

The AMNS representation

The Adapted Modular Number System[1] is a **polynomial** representation of large integers.

$$a = 17414481267196955386252943367005242278 6$$

$$\downarrow$$

$$A_{\mathcal{B}} = -7580373987673 - 814425517156\ \boldsymbol{X} - 577601140430\ \boldsymbol{X}^2$$

[1] Arithmetic Operations in the Polynomial Modular Number System. Plantard et al. (2005).

The AMNS representation

The Adapted Modular Number System[1] is a **polynomial** representation of large integers.

$$a = 17414481267196955386252943367005242278 6$$

$$\downarrow$$

$$A_{\mathcal{B}} = -7580373987673 - 814425517156\ \boldsymbol{X} - 577601140430\ \boldsymbol{X}^2$$

- Let $\boldsymbol{N}$ be the number of coefficients in a polynomial.
- Let $\boldsymbol{\rho}$ be the maximum possible absolute value of coefficients.

[1] Arithmetic Operations in the Polynomial Modular Number System. Plantard et al. (2005).

Modular multiplication: classical representation

$$a = 94291709282822112 11$$

$$b = 93707181274624065 18$$

$$p = 95429155830441806 99$$

Modular multiplication: classical representation

$$
\begin{aligned}
a &= 94291709282822112 11 \\
b &= 93707181274624065 18 \\
p &= 95429155830441806 99 \\
a * b &= 88358102944595643663134126804419073298
\end{aligned}
$$

Modular multiplication: classical representation

$$
\begin{aligned}
a &= 94291709282822 11211 \\
b &= 93707181274624 06518 \\
p &= 95429155830441 80699 \\
a * b &= 88358102944595643663134126804419073298 \\
a * b \mod p &= 73372426252280 17929
\end{aligned}
$$

Modular multiplication: classical representation

$$
\begin{aligned}
a &= 94291709282822 11211 \\
b &= 93707181274624 06518 \\
p &= 95429155830441 80699 \\
a * b &= 88358102944595643663134126804419073298 \\
a * b \mod p &= 73372426252280 17929
\end{aligned}
$$

The modular multiplication and reduction using classical representation of integers can be computed efficiently using the **Montgomery Multiplication** algorithm.

Modular multiplication: AMNS representation

$$A_{\mathcal{B}} = 1427552 + 2080380\ \boldsymbol{X} + \ 390196\ \boldsymbol{X}^2$$
$$B_{\mathcal{B}} = 2938729 + 2709118\ \boldsymbol{X} + 1278900\ \boldsymbol{X}^2$$

Modular multiplication: AMNS representation

$$
\begin{aligned}
A_{\mathcal{B}} =& 1427552 + 2080380\ \boldsymbol{X} + \ 390196\ \boldsymbol{X}^2 \\
B_{\mathcal{B}} =& 2938729 + 2709118\ \boldsymbol{X} + 1278900\ \boldsymbol{X}^2 \\
A_{\mathcal{B}} * B_{\mathcal{B}} =& 4195188461408 + 9981079856156\ \boldsymbol{X} + 8608371458524\ \boldsymbol{X}^2 \\
&+ 3717684989128\ \boldsymbol{X}^3 + 499021664400\ \boldsymbol{X}^4
\end{aligned}
$$

Modular multiplication: AMNS representation

$$
\begin{aligned}
A_{\mathcal{B}} =& 1427552 + 2080380\ \boldsymbol{X} + \ 390196\ \boldsymbol{X}^2 \\
B_{\mathcal{B}} =& 2938729 + 2709118\ \boldsymbol{X} + 1278900\ \boldsymbol{X}^2 \\
A_{\mathcal{B}} * B_{\mathcal{B}} =& 4195188461408 + 9981079856156\ \boldsymbol{X} + 8608371458524\ \boldsymbol{X}^2 \\
& + 3717684989128\ \boldsymbol{X}^3 + 499021664400\ \boldsymbol{X}^4
\end{aligned}
$$

The number of coefficients must be decreased: **external reduction**.

$$11630558439664 + 10979123184956\ \boldsymbol{X} + 8608371458524\ \boldsymbol{X}^2$$

Modular multiplication: AMNS representation

$$
\begin{aligned}
A_{\mathcal{B}} =& 1427552 + 2080380\ \boldsymbol{X} + \ 390196\ \boldsymbol{X}^2 \\
B_{\mathcal{B}} =& 2938729 + 2709118\ \boldsymbol{X} + 1278900\ \boldsymbol{X}^2 \\
A_{\mathcal{B}} * B_{\mathcal{B}} =& 4195188461408 + 9981079856156\ \boldsymbol{X} + 8608371458524\ \boldsymbol{X}^2 \\
& + 3717684989128\ \boldsymbol{X}^3 + 499021664400\ \boldsymbol{X}^4
\end{aligned}
$$

The number of coefficients must be decreased: **external reduction**.

$$11630558439664 + 10979123184956\ \boldsymbol{X} + 8608371458524\ \boldsymbol{X}^2$$

The size of coefficients must be decreased: **internal reduction**

$$2159884 + 1289251\ \boldsymbol{X} + 1528170\ \boldsymbol{X}^2$$

External Reduction

- Let λ be a small integer ($\log_2 |\lambda| \leq 4$).
- Let $E = \boldsymbol{X}^N - \lambda$ be the external reduction polynomial.

External Reduction

- Let λ be a small integer ($\log_2 |\lambda| \leq 4$).
- Let $E = \boldsymbol{X}^N - \lambda$ be the external reduction polynomial.

$$
\begin{aligned}
A_{\mathcal{B}} = &\quad A_0 + A_1\ \boldsymbol{X} + A_2\ \boldsymbol{X}^2 \\
B_{\mathcal{B}} = &\quad B_0 + B_1\ \boldsymbol{X} + B_2\ \boldsymbol{X}^2 \\
A_{\mathcal{B}} * B_{\mathcal{B}} = &\quad A_0B_0 + (A_1B_0 + A_0B_1)\ \boldsymbol{X} + (A_2B_0 + A_1B_1 + A_0B_2)\ \boldsymbol{X}^2 \\
&\quad + (A_2B_1 + A_1B_2)\ \boldsymbol{X}^3 + A_2B_2\ \boldsymbol{X}^4
\end{aligned}
$$

External Reduction

- Let λ be a small integer ($\log_2 |\lambda| \leq 4$).
- Let $E = \boldsymbol{X}^N - \lambda$ be the external reduction polynomial.

$$
\begin{aligned}
A_{\mathcal{B}} = &\quad A_0 + A_1\ \boldsymbol{X} + A_2\ \boldsymbol{X}^2 \\
B_{\mathcal{B}} = &\quad B_0 + B_1\ \boldsymbol{X} + B_2\ \boldsymbol{X}^2 \\
A_{\mathcal{B}} * B_{\mathcal{B}} = &\quad A_0B_0 + (A_1B_0 + A_0B_1)\ \boldsymbol{X} + (A_2B_0 + A_1B_1 + A_0B_2)\ \boldsymbol{X}^2 \\
&\quad + (A_2B_1 + A_1B_2)\ \boldsymbol{X}^3 + A_2B_2\ \boldsymbol{X}^4
\end{aligned}
$$

$$
A_{\mathcal{B}} * B_{\mathcal{B}} \mod E = \begin{pmatrix} \lambda A_1B_2 \\ +\lambda A_2B_1 \\ + A_0B_0 \end{pmatrix} + \begin{pmatrix} \lambda A_2B_2 \\ + A_0B_1 \\ + A_1B_0 \end{pmatrix} \boldsymbol{X} + \begin{pmatrix} A_0B_2 \\ + A_1B_1 \\ + A_2B_0 \end{pmatrix} \boldsymbol{X}^2
$$

Internal Reduction

- Let ϕ be a power of 2.
- Let M and M' be the internal reduction polynomials.
  $M * M' \mod (E, \phi) \equiv -1 \mod (E, \phi)$.

Internal Reduction

- Let ϕ be a power of 2.
- Let M and M' be the internal reduction polynomials.
  $M * M' \mod (E, \phi) \equiv -1 \mod (E, \phi)$.

Algorithm 1 AMNS multiplication - Montgomery-Like algorithm

1: **procedure** $\text{AMNS\_REDINT}(A_{\mathcal{B}}, B_{\mathcal{B}})$
2: $C \leftarrow A_{\mathcal{B}} * B_{\mathcal{B}} \mod E$
3: $Q \leftarrow C * M' \mod (E, \phi)$
4: $S \leftarrow \dfrac{C + (Q * M \mod E)}{\phi}$
5: **return** S
6: **end procedure**

FPGA Implementation

Implementation

Tools:

- FPGAs[1]: re-configurable matrix of discrete elementary components.
- Flexible and allow for quick prototyping.
- Embedded arithmetic accelerators: DSP blocks feature a 25x18 bits signed multiplier and a three-input 48 bits adder.

[1] Zedboard and Artix-7 FPGA are used in this work.

Implementation

Tools:

- FPGAs[1]: re-configurable matrix of discrete elementary components.
- Flexible and allow for quick prototyping.
- Embedded arithmetic accelerators: DSP blocks feature a 25x18 bits signed multiplier and a three-input 48 bits adder.

Objective:

- Generic models for different values of the N, ϕ and λ parameters

[1] Zedboard and Artix-7 FPGA are used in this work.

Implementation

Tools:

- FPGAs[1]: re-configurable matrix of discrete elementary components.
- Flexible and allow for quick prototyping.
- Embedded arithmetic accelerators: DSP blocks feature a 25x18 bits signed multiplier and a three-input 48 bits adder.

Objective:

- Generic models for different values of the N, ϕ and λ parameters

$$A_{\mathcal{B}} * B_{\mathcal{B}} \mod E = \begin{pmatrix} \lambda A_1 B_2 \\ +\lambda A_2 B_1 \\ + A_0 B_0 \end{pmatrix} + \begin{pmatrix} \lambda A_2 B_2 \\ + A_0 B_1 \\ + A_1 B_0 \end{pmatrix} \boldsymbol{X} + \begin{pmatrix} A_0 B_2 \\ + A_1 B_1 \\ + A_2 B_0 \end{pmatrix} \boldsymbol{X}^2$$

We want to develop N resources capable of computing operations of the shape $\lambda A_1 B_2 + \lambda A_2 B_1 + A_0 B_0$ in parallel.

[1] Zedboard and Artix-7 FPGA are used in this work.

Integer Multiplication

- Partition of integers into 17 bits wide sections (18*18 bits).
- Simple and regular manipulation.
- Efficient use of our DSP blocks.
- Extra space in DSP blocks to accomodate for changes in N (48 bits accumulator).

Integer Multiplication

- Partition of integers into 17 bits wide sections (18*18 bits).
- Simple and regular manipulation.
- Efficient use of our DSP blocks.
- Extra space in DSP blocks to accomodate for changes in N (48 bits accumulator).

$$A = \underbrace{10100111}_{a_3}|\underbrace{0101011011011001}_{a_2}|\underbrace{1010101001001011}_{a_1}|\underbrace{1001100110011110}_{a_0}{}_b$$

Integer Multiplication

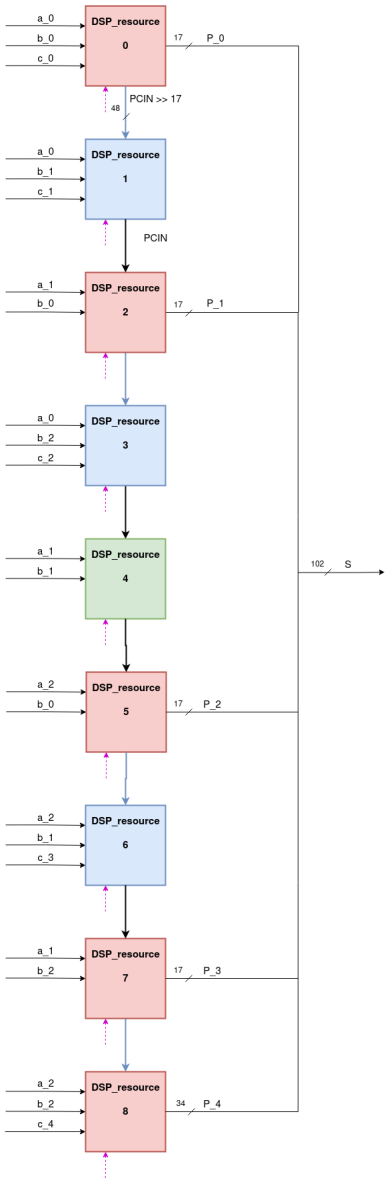
- Partition of integers into 17 bits wide sections (18*18 bits).
- Simple and regular manipulation.
- Efficient use of our DSP blocks.
- Extra space in DSP blocks to accomodate for changes in N (48 bits accumulator).

$$A = \underbrace{101001110}_{a_3}|\underbrace{0101011010110001}_{a_2}|\underbrace{1010101001001001 1}_{a_1}|\underbrace{1001100110011110}_{a_0}{}_b$$

Let K be the number of 17 bits sections required to partition coefficients.
We can represent **coefficients** as polynomials in $\boldsymbol{Y}$ (ex: $K = 3$):

$$\begin{aligned} A = &\quad a_0 + a_1\,\boldsymbol{Y} + a_2\,\boldsymbol{Y}^2 \\ B = &\quad b_0 + b_1\,\boldsymbol{Y} + b_2\,\boldsymbol{Y}^2 \\ A * B = &\quad a_0b_0 + (a_1b_0 + a_0b_1)\,\boldsymbol{Y} + (a_2b_0 + a_1b_1 + a_0b_2)\,\boldsymbol{Y}^2 \\ &+ (a_2b_1 + a_1b_2)\,\boldsymbol{Y}^3 + a_2b_2\,\boldsymbol{Y}^4 \end{aligned}$$

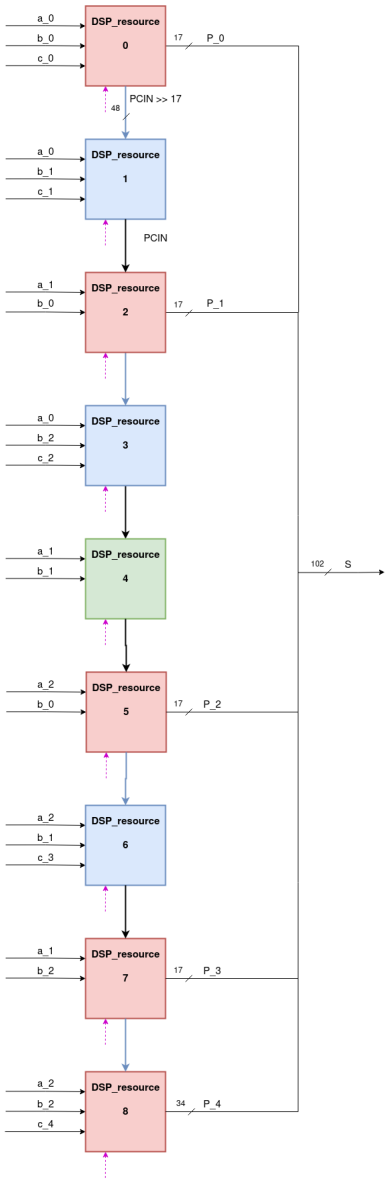
Integer multiplication - Column model



DSP blocks can be organized within a column:

- K^2 DSP blocks are used.
- All partial products are computed in parallel in a single cycle.
- Partial products must be recombined and data has to travel the entire column: high operating frequency.
- A complete integer multiplication requires K^2 cycles.

Integer multiplication - Column model



DSP blocks can be organized within a column:

- K^2 DSP blocks are used.
- All partial products are computed in parallel in a single cycle.
- Partial products must be recombined and data has to travel the entire column: high operating frequency.
- A complete integer multiplication requires K^2 cycles.

DSP_BLOCK/ operation	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
1	a_0b_0	a_0b_1	a_1b_0	a_0b_2	a_1b_1	a_2b_0	a_2b_1	a_1b_2	a_2b_2
2	-	$P_0 >> 17$	-	-	-	-	-	-	-
3	-	-	P_1	-	-	-	-	-	-
4	-	-	-	$P_2 >> 17$	-	-	-	-	-
5	-	-	-	-	P_3	-	-	-	-
6	-	-	-	-	-	P_4	-	-	-
7	-	-	-	-	-	-	$P_5 >> 17$	-	-
8	-	-	-	-	-	-	-	P_6	-
9	-	-	-	-	-	-	-	-	$P_7 >> 17$

Table 1: Scheduling of integer multiplication - Column model

Lambda Multiplication

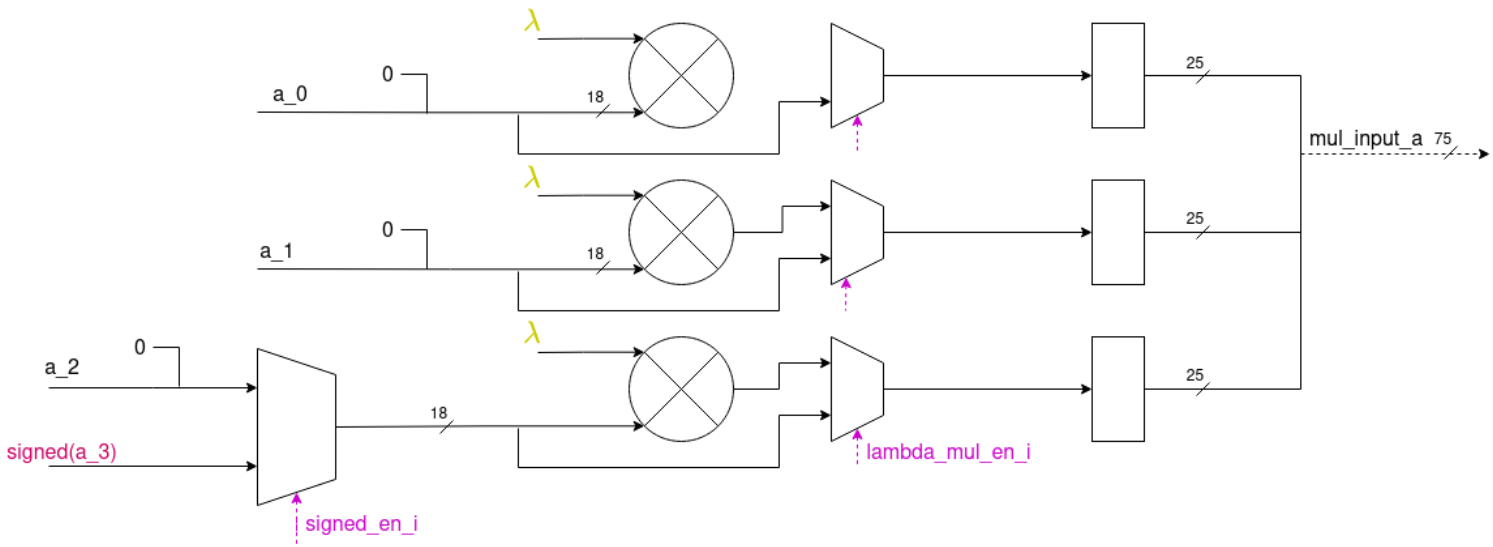
$$A_{\mathcal{B}} * B_{\mathcal{B}} \mod E = \begin{pmatrix} \lambda A_1 B_2 \\ +\lambda A_2 B_1 \\ + A_0 B_0 \end{pmatrix} + \begin{pmatrix} \lambda A_2 B_2 \\ + A_0 B_1 \\ + A_1 B_0 \end{pmatrix} \boldsymbol{X} + \begin{pmatrix} A_0 B_2 \\ + A_1 B_1 \\ + A_2 B_0 \end{pmatrix} \boldsymbol{X}^2$$

- Small multiplication ($\log_2(|\lambda|) \leq 4$).
- Extra space in the 25 bits multiplicative input of DSP blocks. We are currently using 18*18 bits signed multiplications.
- 17-bits sections can be directly multiplied by λ before being fed to DSP blocks: $\lambda A = \lambda a_0 + \lambda a_1 \boldsymbol{Y} + \lambda a_2 \boldsymbol{Y}^2$

Lambda Multiplication

$$A_{\mathcal{B}} * B_{\mathcal{B}} \mod E = \begin{pmatrix} \lambda A_1 B_2 \\ +\lambda A_2 B_1 \\ + A_0 B_0 \end{pmatrix} + \begin{pmatrix} \lambda A_2 B_2 \\ + A_0 B_1 \\ + A_1 B_0 \end{pmatrix} \boldsymbol{X} + \begin{pmatrix} A_0 B_2 \\ + A_1 B_1 \\ + A_2 B_0 \end{pmatrix} \boldsymbol{X}^2$$

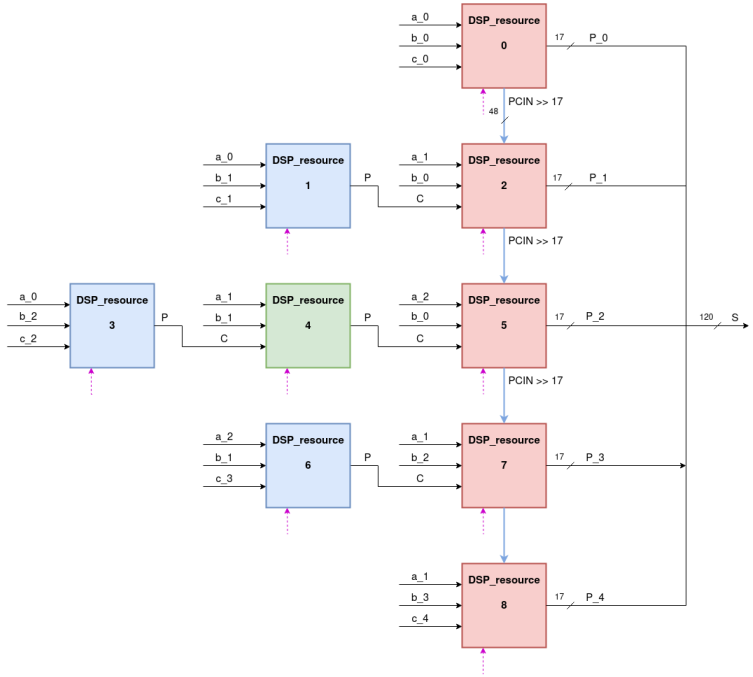
- Small multiplication ($\log_2(|\lambda|) \leq 4$).
- Extra space in the 25 bits multiplicative input of DSP blocks. We are currently using 18*18 bits signed multiplications.
- 17-bits sections can be directly multiplied by λ before being fed to DSP blocks: $\lambda A = \lambda a_0 + \lambda a_1 \boldsymbol{Y} + \lambda a_2 \boldsymbol{Y}^2$



Line Column Multiplier

DSP blocks can be organized along lines.

- K^2 DSP blocks are used.
- FPGA resource is used to route signals: frequency is lower than Column model.
- Partial products are added together simultaneously. A complete integer multiplication requires $2K - 1$ cycles.



DSP_BLOCK/ operation	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
1	a_0b_0	a_0b_1	a_1b_0	a_0b_2	a_1b_1	a_2b_0	a_2b_1	a_1b_2	a_2b_2
2	-	-	$P_0 >> 17 + P_1$	-	P_3	-	-	-	-
3	-	-	-	-	-	$P_2 >> 17 + P_4$	-	-	-
4	-	-	-	-	-	-	-	$P_5 >> 17 + P_6$	-
5	-	-	-	-	-	-	-	-	$P_7 >> 17$

Table 2: Scheduling of integer multiplication - Line Column model

Full external reduction

We can use these models to compute operations of the shape $\lambda A_1 B_2 + \lambda A_2 B_1 + A_0 B_0$.

DSP_BLOCK/ operation	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
1	$\lambda a_{20} b_{30}$	$\lambda a_{20} b_{31}$	$\lambda a_{21} b_{30}$	$\lambda a_{20} b_{32}$	$\lambda a_{21} b_{31}$	$\lambda a_{22} b_{30}$	$\lambda a_{22} b_{31}$	$\lambda a_{21} b_{32}$	$\lambda a_{22} b_{32}$
2	$\lambda a_{30} b_{20}$	$\lambda a_{30} b_{21}$	$\lambda a_{31} b_{20}$	$\lambda a_{30} b_{22}$	$\lambda a_{31} b_{21}$	$\lambda a_{32} b_{20}$	$\lambda a_{32} b_{21}$	$\lambda a_{31} b_{22}$	$\lambda a_{32} b_{22}$
3	$a_{00} b_{10}$	$a_{00} b_{11}$	$a_{01} b_{10}$	$a_{00} b_{12}$	$a_{01} b_{11}$	$a_{02} b_{10}$	$a_{02} b_{11}$	$a_{01} b_{12}$	$a_{02} b_{12}$
4	$a_{10} b_{00}$	$a_{10} b_{01}$	$a_{11} b_{00}$	$a_{10} b_{02}$	$a_{11} b_{01}$	$a_{12} b_{00}$	$a_{12} b_{01}$	$a_{11} b_{02}$	$a_{12} b_{02}$
5	-	-	$P_0 >> 17 + P_1$	-	P_3	-	-	-	-
6	-	-	-	-	-	$P_2 >> 17 + P_4$	-	-	-
7	-	-	-	-	-	-	-	$P_5 >> 17 + P_6$	-
8	-	-	-	-	-	-	-	-	$P_7 >> 17$

Table 3: Scheduling of Line Column model - $N = 4, K = 3$

Full external reduction

We can use these models to compute operations of the shape $\lambda A_1 B_2 + \lambda A_2 B_1 + A_0 B_0$.

DSP_BLOCK/ operation	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
1	$\lambda a_{20} b_{30}$	$\lambda a_{20} b_{31}$	$\lambda a_{21} b_{30}$	$\lambda a_{20} b_{32}$	$\lambda a_{21} b_{31}$	$\lambda a_{22} b_{30}$	$\lambda a_{22} b_{31}$	$\lambda a_{21} b_{32}$	$\lambda a_{22} b_{32}$
2	$\lambda a_{30} b_{20}$	$\lambda a_{30} b_{21}$	$\lambda a_{31} b_{20}$	$\lambda a_{30} b_{22}$	$\lambda a_{31} b_{21}$	$\lambda a_{32} b_{20}$	$\lambda a_{32} b_{21}$	$\lambda a_{31} b_{22}$	$\lambda a_{32} b_{22}$
3	$a_{00} b_{10}$	$a_{00} b_{11}$	$a_{01} b_{10}$	$a_{00} b_{12}$	$a_{01} b_{11}$	$a_{02} b_{10}$	$a_{02} b_{11}$	$a_{01} b_{12}$	$a_{02} b_{12}$
4	$a_{10} b_{00}$	$a_{10} b_{01}$	$a_{11} b_{00}$	$a_{10} b_{02}$	$a_{11} b_{01}$	$a_{12} b_{00}$	$a_{12} b_{01}$	$a_{11} b_{02}$	$a_{12} b_{02}$
5	-	-	$P_0 >> 17 + P_1$	-	P_3	-	-	-	-
6	-	-	-	-	-	$P_2 >> 17 + P_4$	-	-	-
7	-	-	-	-	-	-	-	$P_5 >> 17 + P_6$	-
8	-	-	-	-	-	-	-	-	$P_7 >> 17$

Table 3: Scheduling of Line Column model - $N = 4, K = 3$

Model	DSPs	Latency	Max. Freq.
Column	$N * K^2$	$\boldsymbol{N + K^2 - 1}$	+
Line Column	$N * K^2$	$\boldsymbol{N + 2K - 2}$	+
Flat Line Column	$N * (2K - 1)$	$\boldsymbol{(N + 1) * K}$	−

Table 4: Properties of external reduction resource models

Internal Reduction

- 10 additional cycles of latency are added because of data dependencies between operations, there are bubbles in the pipeline.
- The recombination step of integers is cut short during $\mod \phi$ operations since we have no need for the most significant bits.

Model	DSPs	Latency
Column	$N * K^2$	$10 + 3N + 2 * (\boldsymbol{K^2} - 1) + K * (K + 1)/2 - 1$
Line Column	$N * K^2$	$10 + 3N + 2 * (2\boldsymbol{K} - 2) + K - 1$
Flat Line Column	$N * (2K - 1)$	$10 + 3\boldsymbol{NK} + 2K + 1$

Table 5: DSP usage and cycle count - internal reduction

Results and Conclusion

Results

Model	DSPs	cycles	freq (MHz)	time (μs)	LUTs	ADP
Line Column	80	**40**	**200.00**	**0.200**	1146	4098
Flat Line Column	**35**	79	179.53	0.441	1682	4474
Column	80	64	200.00	0.320	1161	6562
Chaouch et al.[2]	120	33	200.00	**0.165**	2728	5238
Mrabet et al.[4]	**31**	33	106	0.312	870	2610
Gallin, Tisserand[3]	**9**	143	**598.00**	0.239	**634**	**672**

Table 6: Timing and usage of our models - p width: 256 bits

$$\text{ADP} = \left(\text{LUT}_{\text{used}} + \text{DSP}_{\text{used}} * \frac{\text{DSP}_{\text{total}}}{\text{LUT}_{\text{total}}}\right) * \text{time}$$

Results

Model	DSPs	cycles	freq (MHz)	time (μs)	LUTs	ADP
Line Column	80	**40**	**200.00**	**0.200**	1146	4098
Flat Line Column	**35**	79	179.53	0.441	1682	4474
Column	80	64	200.00	0.320	1161	6562
Chaouch et al.[2]	120	33	200.00	**0.165**	2728	5238
Mrabet et al.[4]	**31**	33	106	0.312	870	2610
Gallin, Tisserand[3]	**9**	143	**598.00**	0.239	**634**	**672**

Table 6: Timing and usage of our models - p width: 256 bits

$$\text{ADP} = \left(\text{LUT}_{\text{used}} + \text{DSP}_{\text{used}} * \frac{\text{DSP}_{\text{total}}}{\text{LUT}_{\text{total}}}\right) * \text{time}$$

Model	DSPs	cycles	freq (MHz)	time (μs)	LUTs	ADP
Line Column	175	51	175.00	0.292	2147	12983
Flat Line Column	63	126	165.00	0.764	2784	13766
Chaouch et al.[2]	188	33	161.76	0.204	29985	15391
Mrabet et al.[5]	60	66	106	0.624	1789	10137

Table 7: Timing and usage of our models - p width: 512 bits

Conclusion and Perspective

We have successfully developed a generic models for hardware implementations of AMNS arithmetic using a FPGA. Our models are scalable and can fit different values of the N, K and λ parameter.
Our fastest model can outperform state of the art FPGA implementations of modular multiplication/AMNS arithmetic.

Conclusion and Perspective

We have successfully developed a generic models for hardware implementations of AMNS arithmetic using a FPGA. Our models are scalable and can fit different values of the N, K and λ parameter.
Our fastest model can outperform state of the art FPGA implementations of modular multiplication/AMNS arithmetic.

Perspectives:

- Improved efficiency by using the Montgomery CIOS algorithm adapted to AMNS.
- Optimize models and choice of AMNS for a cryptographic protocol.
- Implement the SIKE post-quantum cryptography algorithm using AMNS and optimize models for SIKE prime parameters (width 434 and 503 bits respectively).

Bibliography

- Bajard, J.C., Imbert, L., Plantard, T.: Arithmetic operations in the polynomial modular number system (2005)
- Chaouch, A., Didier, L.S., Dosso, F.Y., El Mrabet, N., Bouallegue, B., Ouni, B.: Two hardware implementations for modular multiplication in the AMNS: Sequential and semi-parallel (2021)
- Gallin, G., Tisserand, A.: Generation of finely-pipelined GF(p) multipliers for flexible curve based cryptography on FPGAs (2019). https://doi.org/10.1109/TC.2019.2920352
- Mrabet, A.: Implémentation efficace de primitive cryptographique pour le couplage sur carte FPGA. Ph.D. thesis (2017)
- Mrabet, A., El-Mrabet, N., Lashermes, R., Rigaud, J.B., Bouallegue, B., Mesnager, S., Machhout, M.: High-performance elliptic curve cryptography by using the CIOS method for modular multiplication (2016)