Joël Cathébras
Alexandre Carbon
Renaud Sirdey
Nicolas Ventroux

Peter Milder

DATA FLOW ORIENTED HARDWARE DESIGN OF RNS-BASED POLYNOMIAL MULTIPLICATION FOR SHE ACCELERATION

Conference on Cryptographic Hardware and Embedded Systems 2018 | Amsterdam, The Netherlands | 09-10-18

IMPLEMENTATION PROBLEMATIC FOR RLWE-BASED LEVELED-FHE SCHEMES

- **Handling polynomial of $\boldsymbol{R} = \mathbb{Z}[X]/(F(X))$ and $\boldsymbol{R}_q = \boldsymbol{R}/q\boldsymbol{R}$:**
  - Modulus q ~ several hundred of bits
  - deg(F) ~ several thousand

Impact ⇦ Security, Multiplicative depth

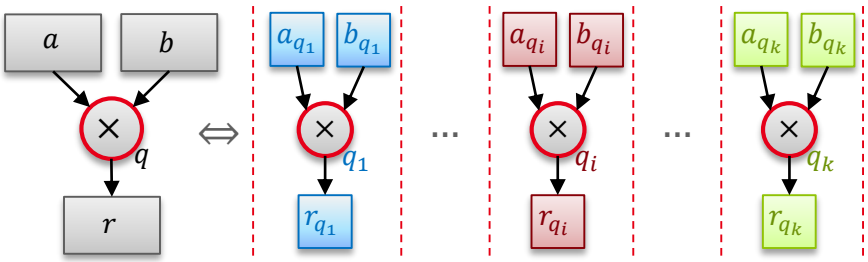
IMPLEMENTATION PROBLEMATIC FOR RLWE-BASED LEVELED-FHE SCHEMES

- **Handling polynomial of $\boldsymbol{R} = \mathbb{Z}[X]/(F(X))$ and $\boldsymbol{R}_q = \boldsymbol{R}/q\boldsymbol{R}$:**
  - Modulus q ~ several hundred of bits
  - $\deg(F)$ ~ several thousand

Impact ⇐ Security, Multiplicative depth

- **Residue Number System:**

$$q = \prod_{i=1}^{k} q_i$$



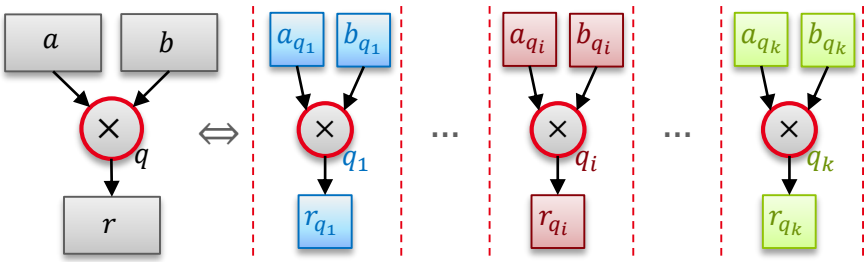
IMPLEMENTATION PROBLEMATIC FOR RLWE-BASED LEVELED-FHE SCHEMES

- **Handling polynomial of $\boldsymbol{R} = \mathbb{Z}[X]/(F(X))$ and $\boldsymbol{R}_q = \boldsymbol{R}/q\boldsymbol{R}$:**
  - Modulus q ~ several hundred of bits
  - $\deg(F)$ ~ several thousand

Impact ⇐ Security, Multiplicative depth

- **Residue Number System:**

$$q = \prod_{i=1}^{k} q_i$$



- **Bajard et al. in 2016, further simplified by Halevi et al. in 2018 :**
  - RNS compatible $\mathrm{FV.Dec}_{RNS}$ and $\mathrm{FV.Mult\&Relin}_{RNS}$.
  - New $\boldsymbol{rlk}_{RNS}$: pair of $k \times k$–matrices with elements in R_{q_i} for i in $1, \ldots, k$.
  - Performance bottleneck: Residue Polynomial Multiplication (R_{q_i}'s products)

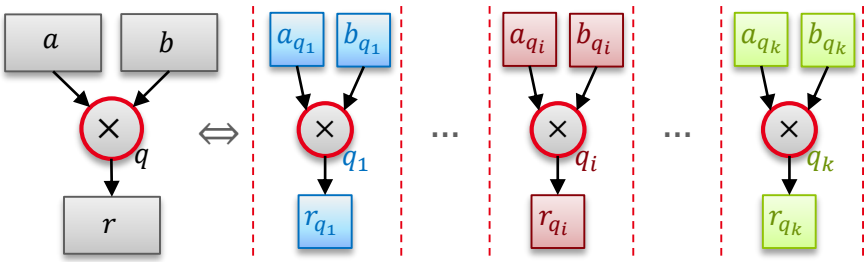
IMPLEMENTATION PROBLEMATIC FOR RLWE-BASED LEVELED-FHE SCHEMES

- **Handling polynomial of $\boldsymbol{R} = \mathbb{Z}[X]/(F(X))$ and $\boldsymbol{R}_q = \boldsymbol{R}/q\boldsymbol{R}$:**
  - Modulus q ~ several hundred of bits
  - $\deg(F)$ ~ several thousand

Impact ⇐ Security, Multiplicative depth

- **Residue Number System:**

$$q = \prod_{i=1}^{k} q_i$$



- **Bajard et al. in 2016, further simplified by Halevi et al. in 2018 :**
  - RNS compatible $\mathrm{FV.Dec}_{RNS}$ and $\mathrm{FV.Mult\&Relin}_{RNS}$.
  - New $\boldsymbol{rlk}_{RNS}$: pair of $k \times k$–matrices with elements in R_{q_i} for i in $1, \dots, k$.
  - Performance bottleneck: Residue Polynomial Multiplication (R_{q_i}'s products)

- **Negative Wrapped Convolution over $R_{q_i} = \mathbb{Z}_{q_i}[X]/(F(X))$:**
  - No polynomial modular reduction.
  - Restrict the choice of $F(X) = X^n + 1$ with n a power of 2.
  - Restrict the choice of q_i: $q_i \equiv 1 \bmod 2n$.
  - $2nk$ precomputed values: $(\psi_i^j)_{0 \le j < 2n}$, where ψ_i a n-th primitive root of -1 in $\mathbb{Z}_{q_i}^*$.

RELATED WORKS (HARDWARE ACCELERATION)

- **Migliore et al. 2018: Karatsuba rather than NWC (no RNS)**
  - Finer choice of $F(X)$ allowing batching of binary messages.
  - Asymptotic complexity in $O(n^{1,585})$ Vs $O(n \log n)$: turning point ($n = 6144$, $\log_2 q = 512$).

  Not sufficient to target large multiplicative depth.

- **Öztürk et al. 2015: RNS and NTT approach for LTV scheme (no NWC)**
  - Memory-access iterative NTT.
  - External pre-computation of NTT twiddle factors.

  Use communication bandwidth for non-payload data.

- **Cousins et al. 2017: RNS and NTT approach for LTV scheme**
  - Dataflow oriented pipelined NTT.
  - Local storage of all twiddle factors at compile time.

  Storage cost in O(kn), dependent of RNS basis size.

- **Sinha Roy et al. 2015: RNS and NTT (no NWC) approach for RLWE-based scheme**
  - Memory-access iterative NTT.
  - Local storage of a subset of the twiddle factors, and computation on-the-fly of the others.

  Better storage in O($k \log n$), but still dependent of RNS basis size.

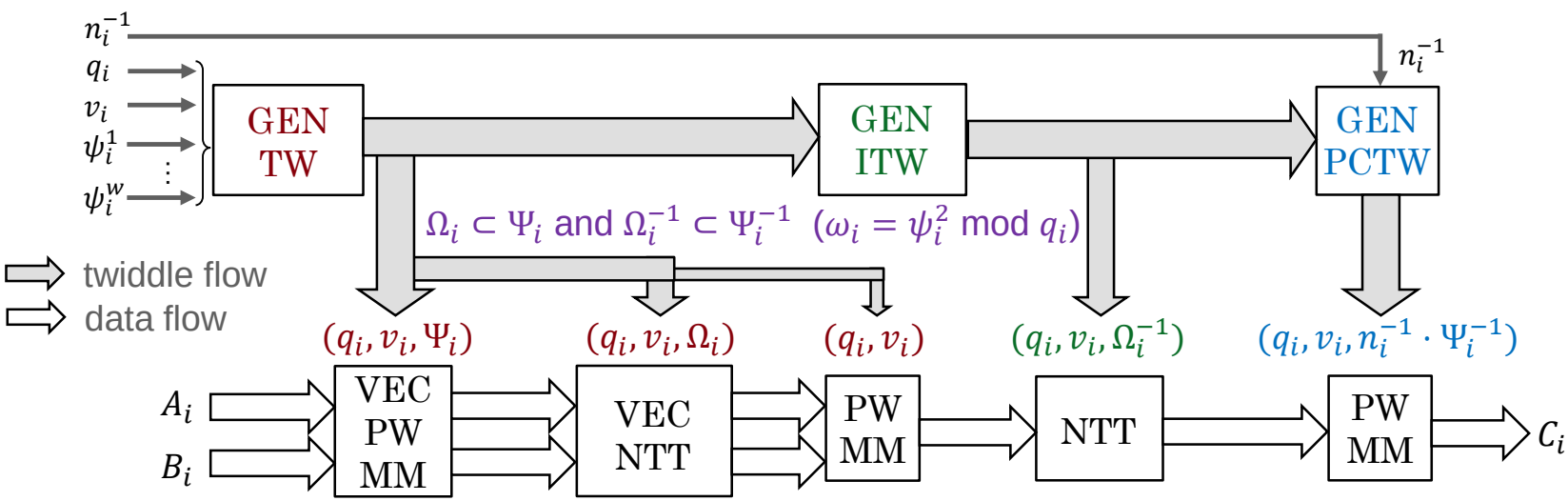
Dataflow oriented NWC with on-the-fly computation of twiddle factors

NWC ARCHITECTURE PRINCIPLE

- **Architecture principle:**

  One NWC over $\boldsymbol{R} \Leftrightarrow O(k)$ smaller NWC over the $\boldsymbol{R}_{q_i}$'s : $C_i = \mathrm{NWC}_i(A_i, B_i)$

- **Required values for NWC_i:**
  - ψ_i: a n-th primitive root of -1 over $\mathbb{Z}_{q_i}^* \Rightarrow \omega_i = \psi_i^2$ mod q_i is a n-th primitive root of 1 over $\mathbb{Z}_{q_i}^*$



NWC ARCHITECTURE PRINCIPLE

- **Architecture principle:**

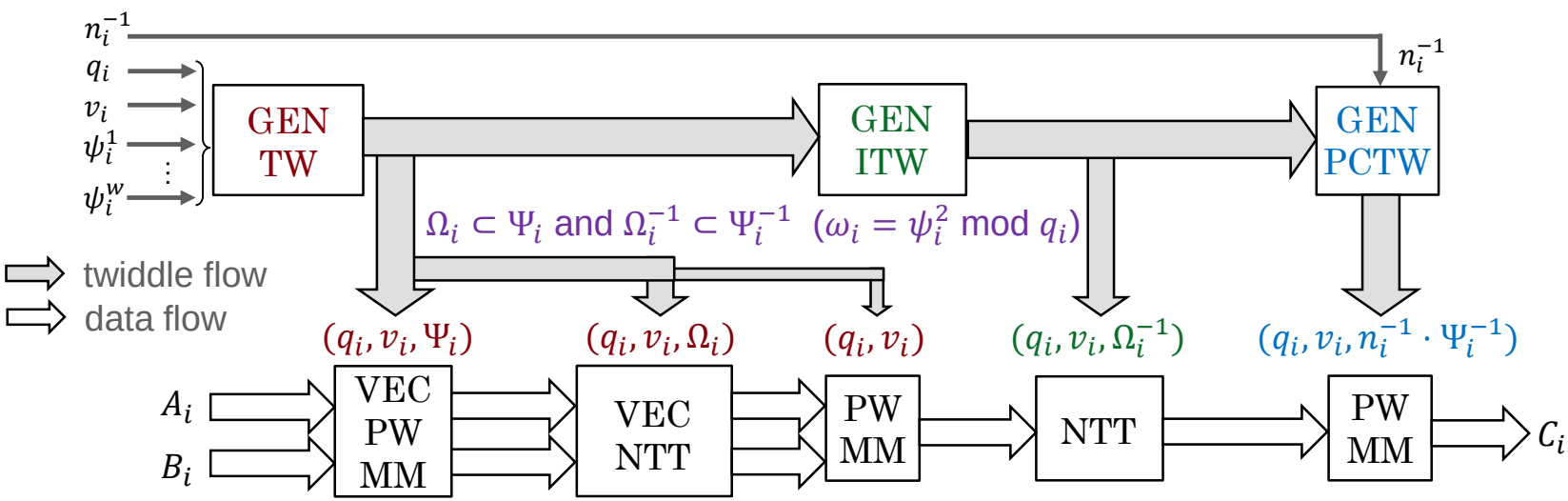
  One NWC over $\boldsymbol{R} \Leftrightarrow O(k)$ smaller NWC over the $\boldsymbol{R}_{q_i}$'s : $C_i = \mathrm{NWC}_i(A_i, B_i)$

- **Required values for NWC_i:**
  - ψ_i: a n-th primitive root of -1 over $\mathbb{Z}_{q_i}^* \Rightarrow \omega_i = \psi_i^2$ mod q_i is a n-th primitive root of 1 over $\mathbb{Z}_{q_i}^*$

$O(w)$ seeds $\ll O(n)$ twiddles

Generation of $\Psi_i = \left(\psi_i^j\right)_{j=0}^{n-1}$.

One set every $T = \frac{n}{w}$ cycles.



NWC ARCHITECTURE PRINCIPLE

- **Architecture principle:**

  One NWC over $\boldsymbol{R} \Leftrightarrow O(k)$ smaller NWC over the $\boldsymbol{R}_{q_i}$'s : $C_i = \mathrm{NWC}_i(A_i, B_i)$

- **Required values for NWC_i:**
  - ψ_i: a n-th primitive root of -1 over $\mathbb{Z}_{q_i}^* \Rightarrow \omega_i = \psi_i^2$ mod q_i is a n-th primitive root of 1 over $\mathbb{Z}_{q_i}^*$

$O(w)$ seeds $\ll O(n)$ twiddles

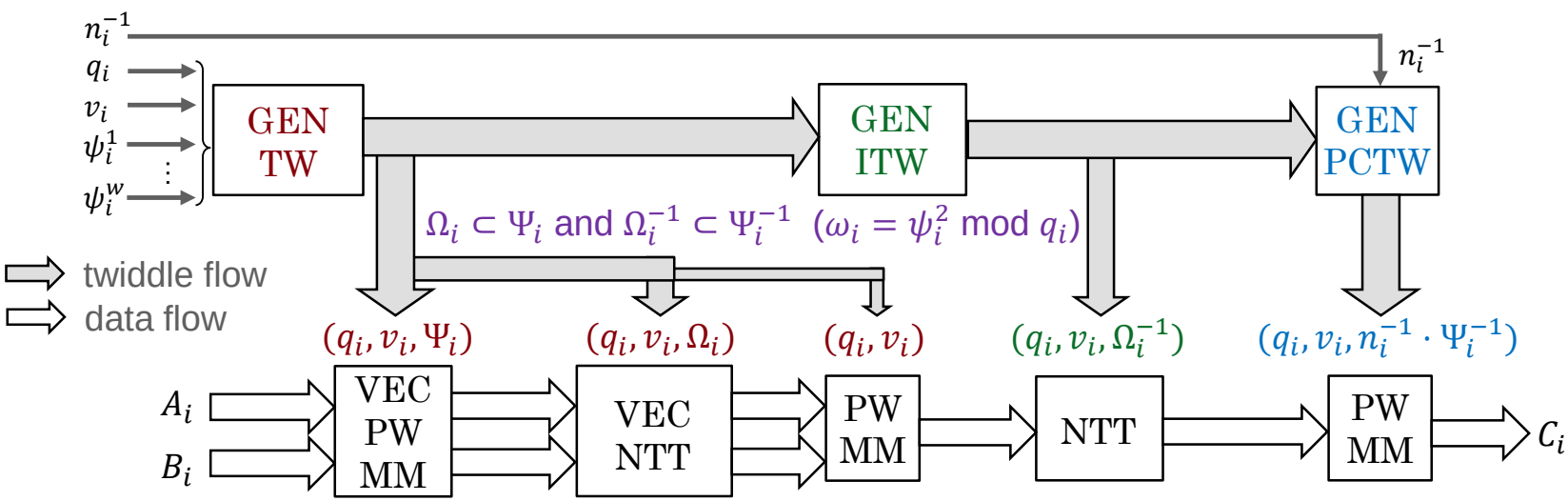
Generation of $\Psi_i = \left(\psi_i^j\right)_{j=0}^{n-1}$.

One set every $T = \frac{n}{w}$ cycles.

Computation of $\Psi_i^{-1} = \left(\psi_i^{-j}\right)_{j=0}^{n-1}$

$\Psi_i^{-1} = \mathrm{Reorder}(q_i - \Psi_i)$

$(q_i - \psi_i^j = \psi_i^{-(n-j)} \bmod q_i)$



NWC ARCHITECTURE PRINCIPLE

- **Architecture principle:**

  One NWC over $\boldsymbol{R} \Leftrightarrow O(k)$ smaller NWC over the $\boldsymbol{R}_{q_i}$'s : $C_i = \mathrm{NWC}_i(A_i, B_i)$

- **Required values for NWC_i:**
  - ψ_i: a n-th primitive root of -1 over $\mathbb{Z}_{q_i}^* \Rightarrow \omega_i = \psi_i^2$ mod q_i is a n-th primitive root of 1 over $\mathbb{Z}_{q_i}^*$

$O(w)$ seeds $\ll O(n)$ twiddles

Generation of $\Psi_i = \left(\psi_i^j\right)_{j=0}^{n-1}$.

One set every $T = \frac{n}{w}$ cycles.

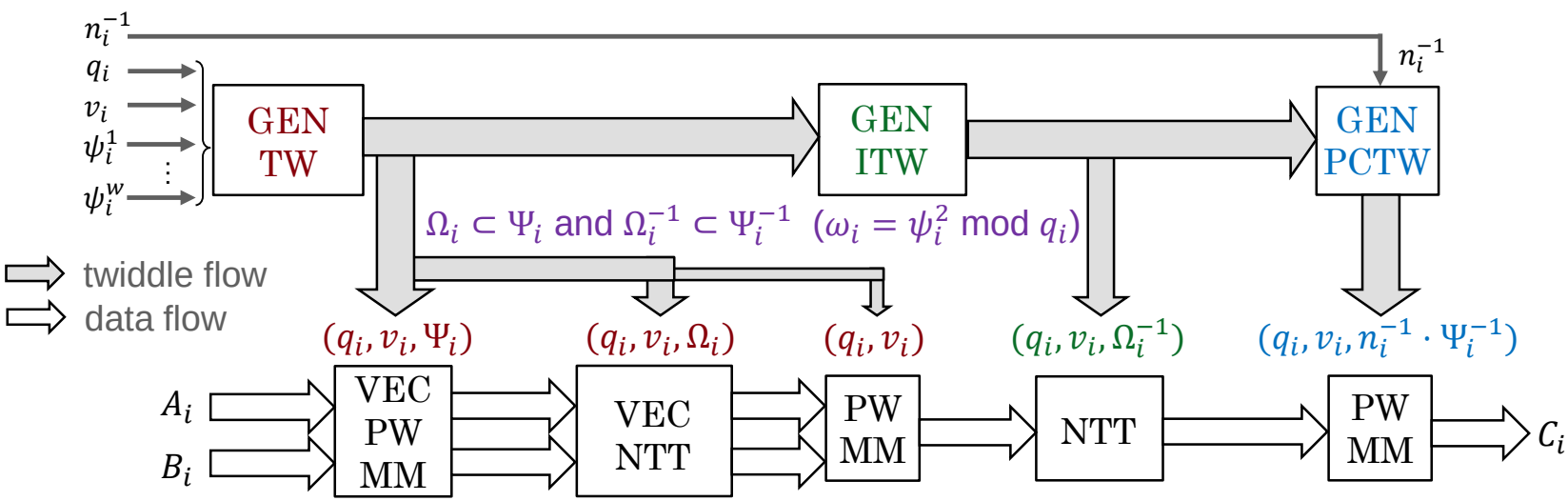
Computation of $\Psi_i^{-1} = \left(\psi_i^{-j}\right)_{j=0}^{n-1}$

$\Psi_i^{-1} = \mathrm{Reorder}(q_i - \Psi_i)$

$(q_i - \psi_i^j = \psi_i^{-(n-j)} \bmod q_i)$

Scale Ψ_i^{-1} by n_i^{-1}

$(n_i^{-1} = n^{-1} \bmod q_i)$



NWC ARCHITECTURE PRINCIPLE

- **Architecture principle:**

  One NWC over $\boldsymbol{R} \Leftrightarrow O(k)$ smaller NWC over the $\boldsymbol{R}_{q_i}$'s : $C_i = \mathrm{NWC}_i(A_i, B_i)$

- **Required values for NWC_i:**
  - ψ_i: a n-th primitive root of -1 over $\mathbb{Z}_{q_i}^* \Rightarrow \omega_i = \psi_i^2$ mod q_i is a n-th primitive root of 1 over $\mathbb{Z}_{q_i}^*$

$O(w)$ seeds $\ll O(n)$ twiddles

Generation of $\Psi_i = \left(\psi_i^j\right)_{j=0}^{n-1}$.

One set every $T = \frac{n}{w}$ cycles.

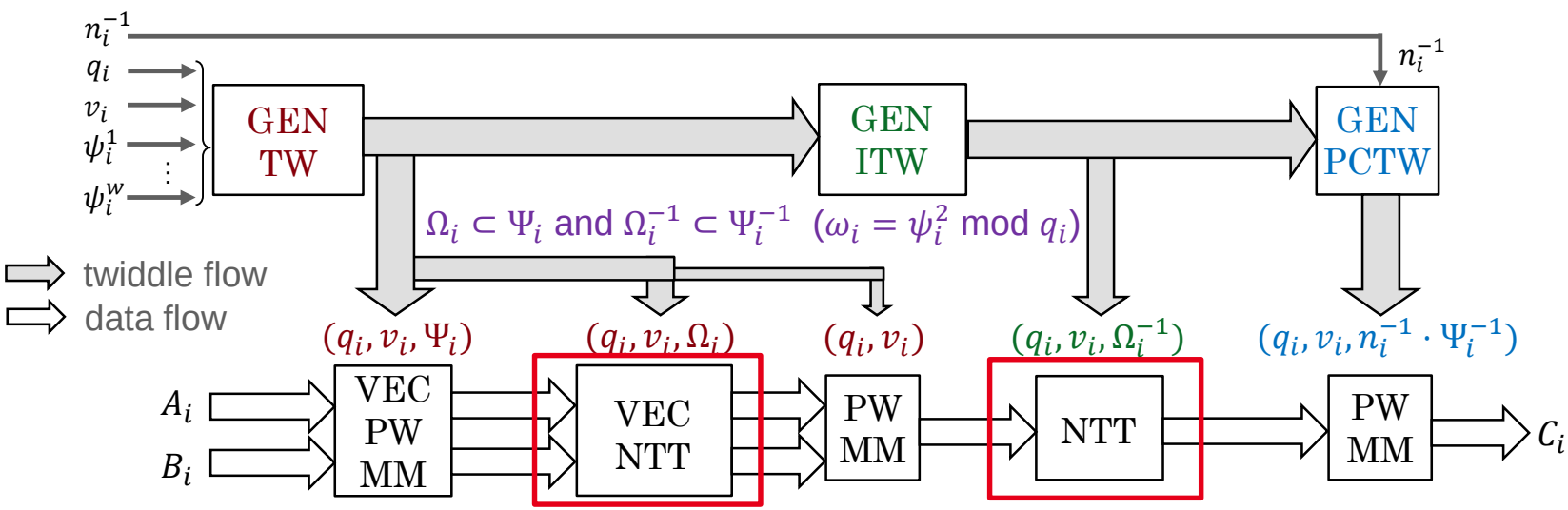
Computation of $\Psi_i^{-1} = \left(\psi_i^{-j}\right)_{j=0}^{n-1}$

$\Psi_i^{-1} = \mathrm{Reorder}(q_i - \Psi_i)$

$(q_i - \psi_i^j = \psi_i^{-(n-j)} \bmod q_i)$

Scale Ψ_i^{-1} by n_i^{-1}

$(n_i^{-1} = n^{-1} \bmod q_i)$



AUTOMATIC GENERATION OF MULTI FIELD NTT DESIGN (1)

- **SPIRAL tool: DFT hardware generator.**
  - Design space exploration.

AUTOMATIC GENERATION OF MULTI FIELD NTT DESIGN (1)

- **SPIRAL tool: DFT hardware generator.**
  - Design space exploration.
  - Complex arithmetic ⇒ $\mathbb{Z}_{q_i}$ modular arithmetic.

$q_i \longleftarrow$ NFLlib prime selection.

Barrett modular reduction.

$$\left(v_i = \left\lfloor \frac{2^{2(s+2)}}{q_i} \right\rfloor \bmod 2^{s+2}\right)$$

AUTOMATIC GENERATION OF MULTI FIELD NTT DESIGN (1)

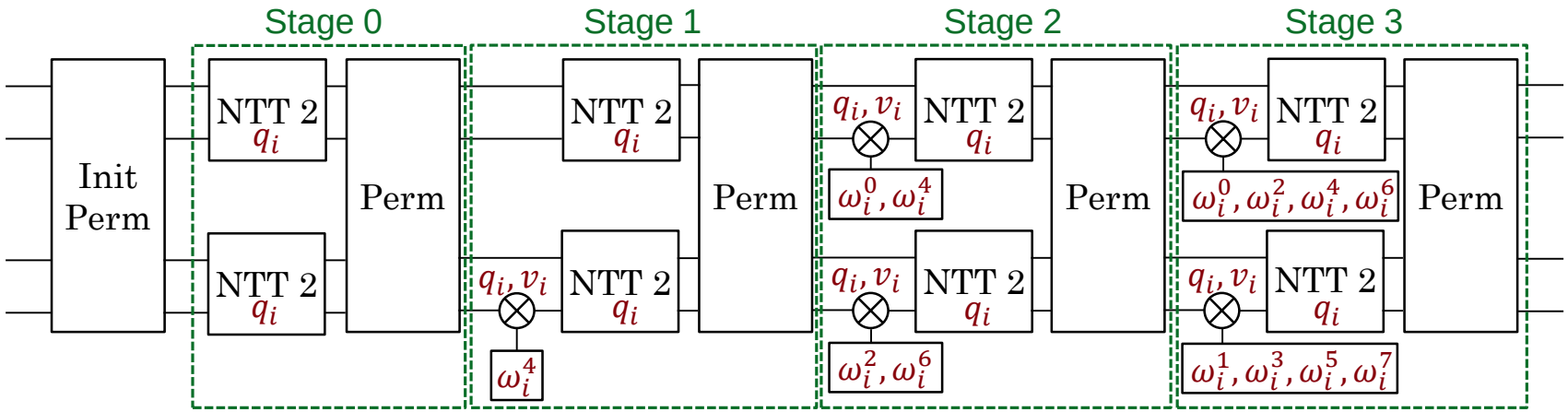
- **SPIRAL tool: DFT hardware generator.**
  - Design space exploration.
  - Complex arithmetic ⇒ $\mathbb{Z}_{q_i}$ modular arithmetic.
  - Modifying twiddle factor handling.

$q_i \longleftarrow$ NFLlib prime selection.

Barrett modular reduction.

$$\left(v_i = \left\lfloor \frac{2^{2(s+2)}}{q_i} \right\rfloor \bmod 2^{s+2}\right)$$

Example of NTT data path ($r = 2$, $n = 16$, $w = 4$):



Characteristics:

- $L = (\log_r n)$ stages.
- w words per cycles.
- One transform every $T = \frac{n}{w}$ cycles.

AUTOMATIC GENERATION OF MULTI FIELD NTT DESIGN (1)

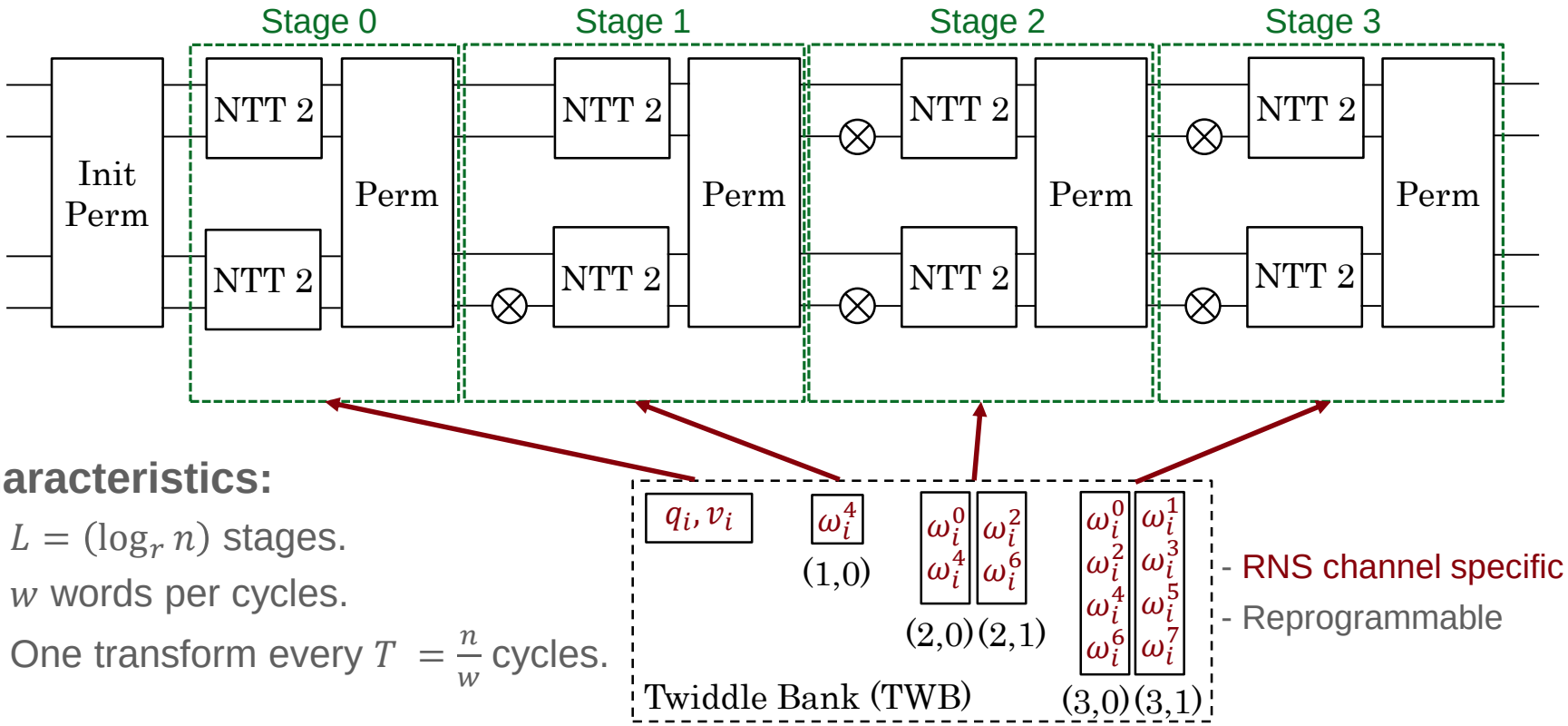
- **SPIRAL tool: DFT hardware generator.**
  - Design space exploration.
  - Complex arithmetic ⇒ $\mathbb{Z}_{q_i}$ modular arithmetic.
  - Modifying twiddle factor handling.

$q_i \longleftarrow$ NFLlib prime selection.

Barrett modular reduction.

$$\left(v_i = \left\lfloor \frac{2^{2(s+2)}}{q_i} \right\rfloor \bmod 2^{s+2}\right)$$

Example of NTT data path ($r = 2$, $n = 16$, $w = 4$):

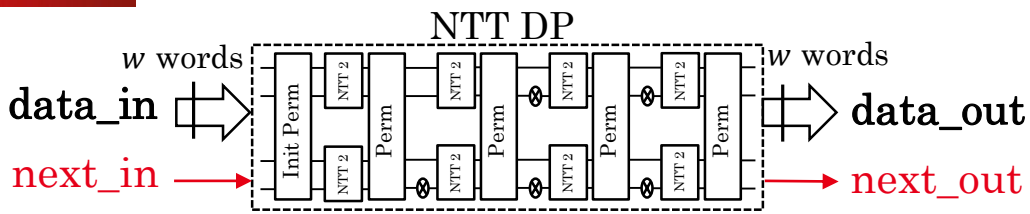


- RNS channel specific
- Reprogrammable

Characteristics:

- $L = (\log_r n)$ stages.
- w words per cycles.
- One transform every $T = \frac{n}{w}$ cycles.

AUTOMATIC GENERATION OF MULTI FIELD NTT DESIGN (2)



- twiddle flow
- read addresses
- write addresses
- write enables

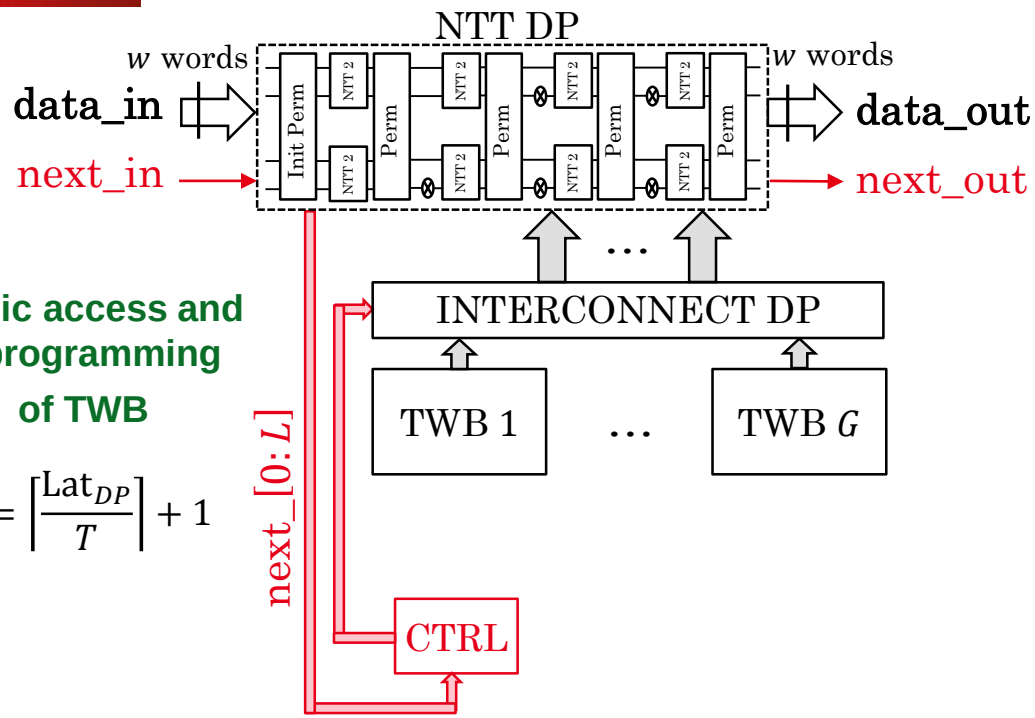
l : stage index

t : way index (in $0, \dots, \frac{w}{2} - 1$)

Cyclic access and reprogramming of TWB

$$G = \left\lceil \frac{\mathrm{Lat}_{DP}}{T} \right\rceil + 1$$

AUTOMATIC GENERATION OF MULTI FIELD NTT DESIGN (2)



Cyclic access and reprogramming of TWB

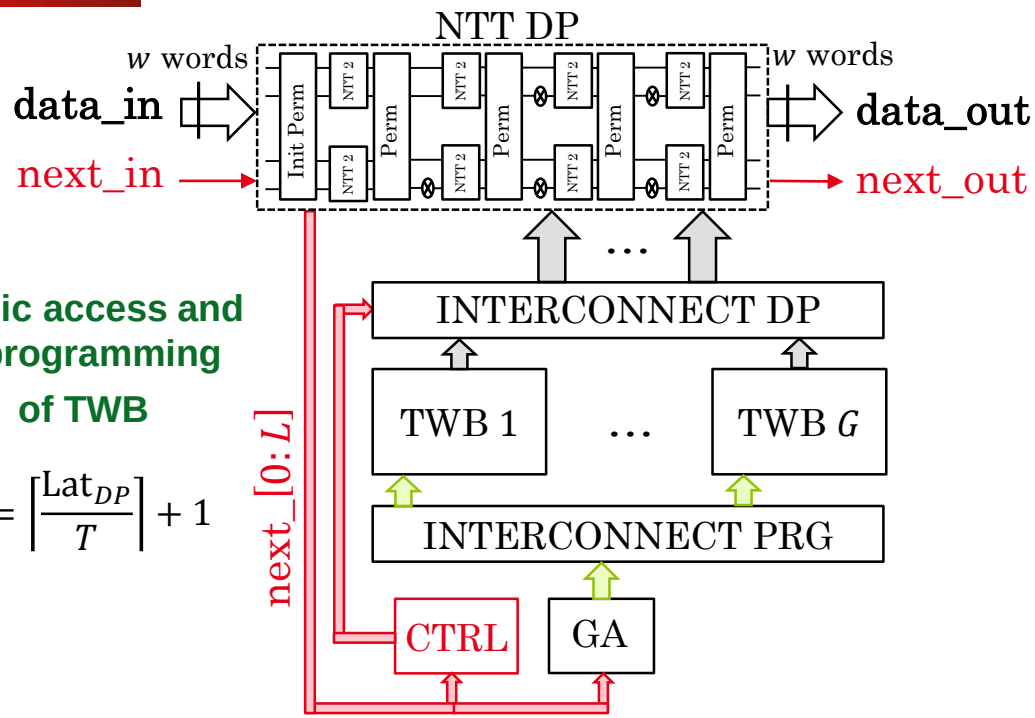
$$G = \left\lceil \frac{\mathrm{Lat}_{DP}}{T} \right\rceil + 1$$

- twiddle flow
- read addresses
- write addresses
- write enables

l : stage index

t : way index (in $0, \ldots, \frac{w}{2} - 1$)

AUTOMATIC GENERATION OF MULTI FIELD NTT DESIGN (2)



Cyclic access and reprogramming of TWB

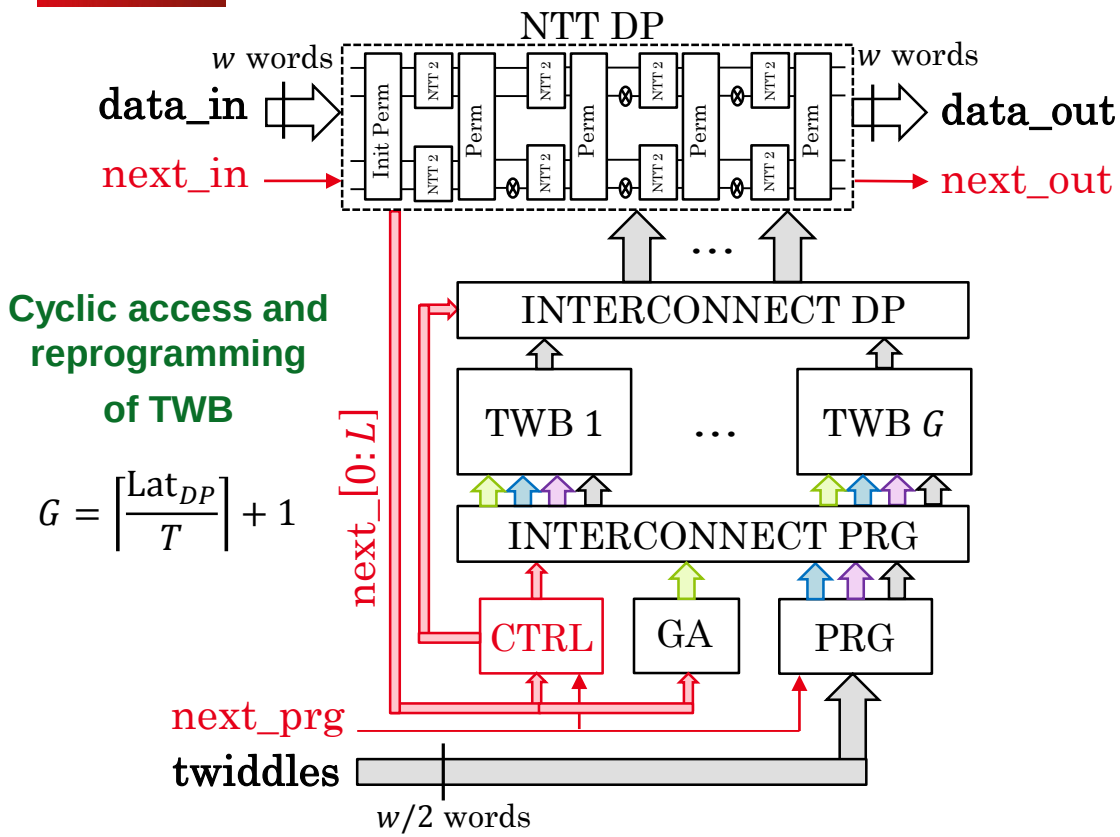
$$G = \left\lceil \frac{\mathrm{Lat}_{DP}}{T} \right\rceil + 1$$

- twiddle flow
- read addresses
- write addresses
- write enables

l : stage index

t : way index (in $0, \ldots, \frac{w}{2} - 1$)

AUTOMATIC GENERATION OF MULTI FIELD NTT DESIGN (2)



Cyclic access and reprogramming of TWB

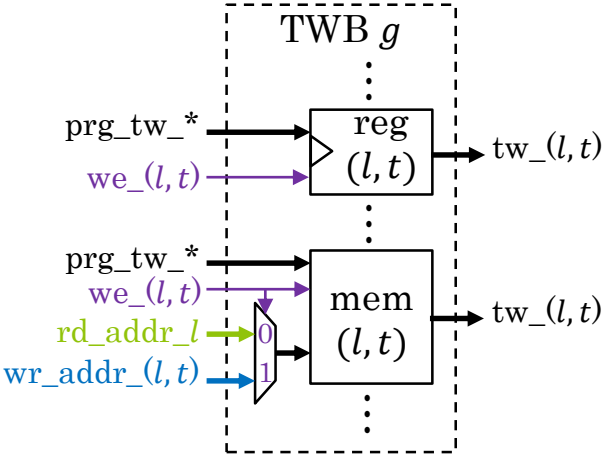
$$G = \left\lceil \frac{\mathrm{Lat}_{DP}}{T} \right\rceil + 1$$

- twiddle flow
- read addresses
- write addresses
- write enables

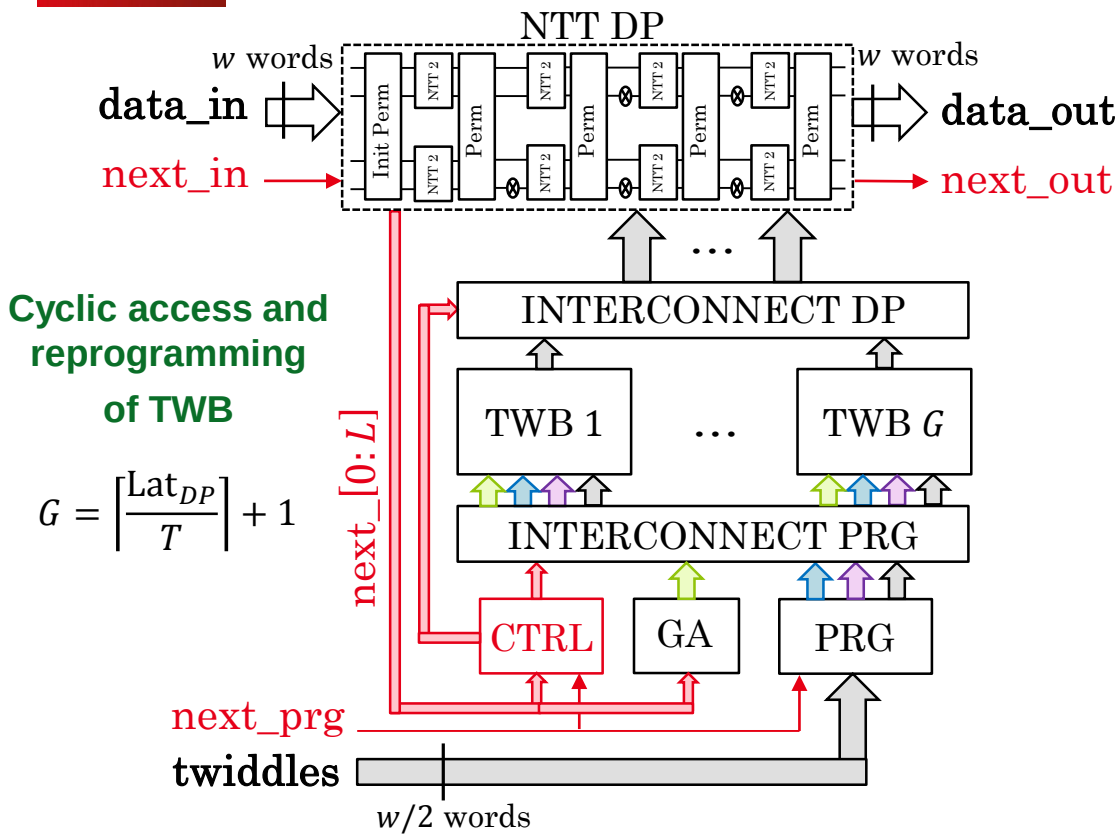
l : stage index

t : way index (in $0, \ldots, \frac{w}{2} - 1$)

- **Reprogramming a TWB:**



AUTOMATIC GENERATION OF MULTI FIELD NTT DESIGN (2)



Cyclic access and reprogramming of TWB

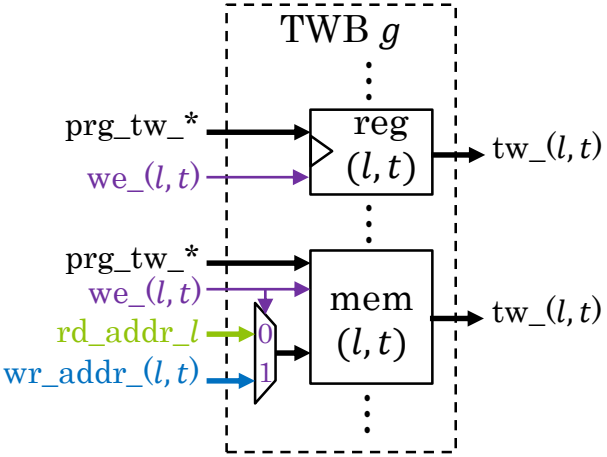
$$G = \left\lceil \frac{\mathrm{Lat}_{DP}}{T} \right\rceil + 1$$

- twiddle flow
- read addresses
- write addresses
- write enables

l : stage index

t : way index (in $0, \ldots, \frac{w}{2} - 1$)

- **Reprogramming a TWB:**



- **Example of reprogram counters ($r = 2$, $n = 16$, $w = 4$):**

twiddles ⇒ $w/2$ words per cycles

Select from the flow ⇒ Update $\mathrm{we\_}(l,t)$ and $\mathrm{wr\_addr\_}(l,t)$

prg_tw_0 ⟶ $\omega_i^0, \omega_i^2, \omega_i^4, \omega_i^6$

prg_tw_1 ⟶ $\omega_i^1, \omega_i^3, \omega_i^5, \omega_i^7$

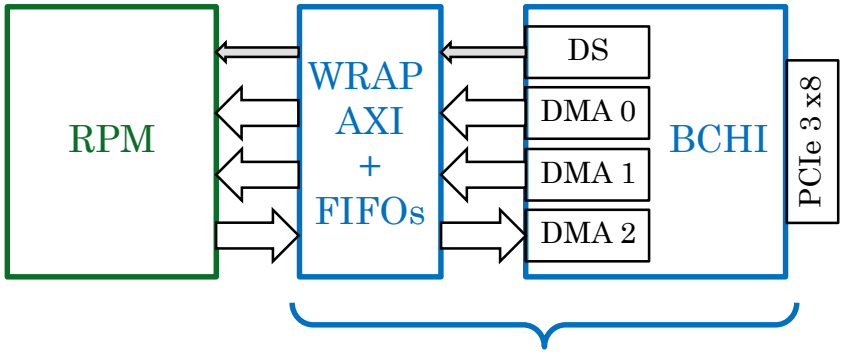
Counter for mem(2,1) : offset 1, step 2, index 0 $\boxed{\omega_i^2, \omega_i^6}$

Counter for mem(3,1) : offset 0, step 1, index 1 $\boxed{\omega_i^1, \omega_i^3, \omega_i^5, \omega_i^7}$

RPM CHARACTERIZATION
PROOF-OF-CONCEPT INTEGRATION (1)

- **Preliminary integration:**
  - Alpha-Data ADM-PCIE 7v3.
  - Xilinx Virtex 7: XC7VX690T-2-FFG1157C.
  - PCIe Gen3, 8 lanes.
  - Vivado 2016.3: placed and routed.

$f_{RPM} = 200$ MHz



$n = 2^{12}$, $\log q_i = 30$, $w = 2$

Ressources		RPM					BCHI
type	available	total	NTT	MM	GTW	Others	& WRAP
LUT	432,368	**54,188**	41,964	5,198	5,906	1,120	27,775
LUTRAM	173,992	**14,402**	10,710	2,056	1,550	86	5,425
FF	864,736	**66,444**	50,961	6,755	7,761	967	39,614
BRAM	1,470	**208**	147	0	21	40	153
DSP	3,600	**517**	363	88	66	0	48
IO	600	**0**	0	0	0	0	59
Pcie	3	**0**	0	0	0	0	1

LUT	12.5%
LUTRAM	8.3%
FF	7.7%
BRAM	**14.1%**
DSP	**14.4%**

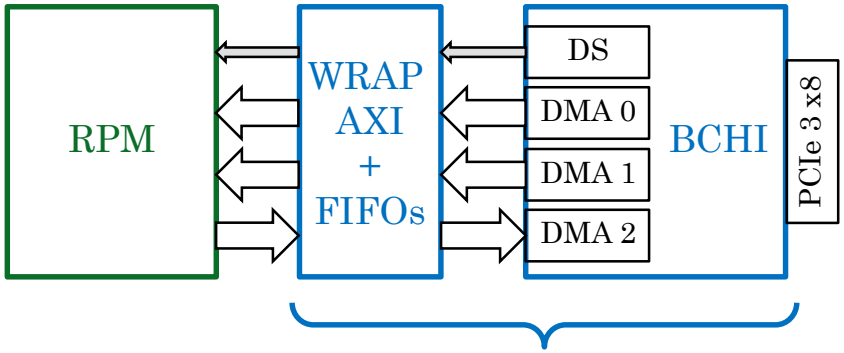
LUT	6.4%
LUTRAM	3.1%
FF	4.6%
BRAM	10.4%
DSP	1.3%

Test PCIe Ok!

RPM CHARACTERIZATION
PROOF-OF-CONCEPT INTEGRATION (1)

- **Preliminary integration:**
  - Alpha-Data ADM-PCIE 7v3.
  - Xilinx Virtex 7: XC7VX690T-2-FFG1157C.
  - PCIe Gen3, 8 lanes.
  - Vivado 2016.3: placed and routed.

$f_{RPM} = 200$ MHz



$n = 2^{12}$, $\log q_i = 30$, $w = 2$

Ressources		RPM					BCHI
type	available	total	NTT	MM	GTW	Others	& WRAP
LUT	432,368	**54,188**	41,964	5,198	5,906	1,120	27,775
LUTRAM	173,992	**14,402**	10,710	2,056	1,550	86	5,425
FF	864,736	**66,444**	50,961	6,755	7,761	967	39,614
BRAM	1,470	**208**	147	0	21	40	153
DSP	3,600	**517**	363	88	66	0	48
IO	600	**0**	0	0	0	0	59
Pcie	3	**0**	0	0	0	0	1

LUT	12.5%
LUTRAM	8.3%
FF	7.7%
BRAM	**14.1%**
DSP	**14.4%**

LUT	6.4%
LUTRAM	3.1%
FF	4.6%
BRAM	10.4%
DSP	1.3%

Test PCIe Ok!

RPM more constraining resources:

- BRAM slices
- DSP slices
- PCIe bandwidth

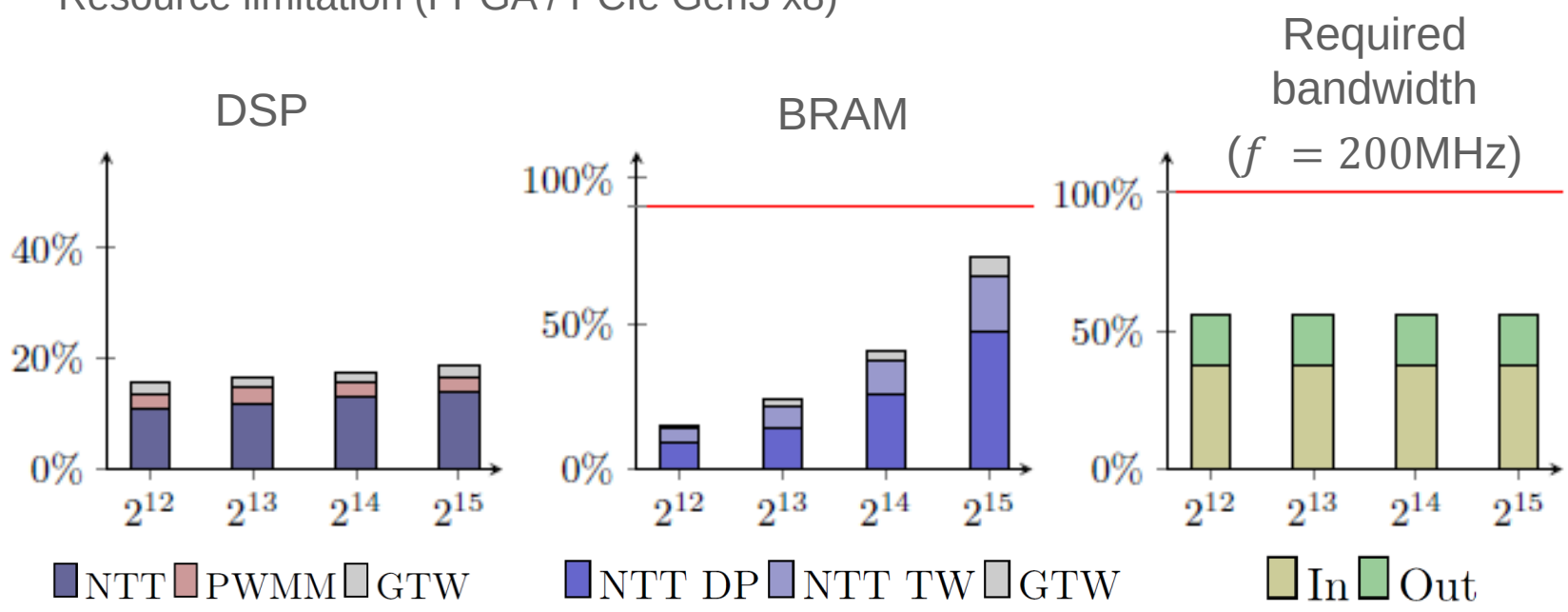
How does RPM scale in SHE context?

RPM CHARACTERIZATION PROJECTIONS (1)

- **Impact of the polynomial degree** n ($w = 2$ and $\log_2 q_i = 30$)**:**

Xilinx Virtex 7: XC7VX690T-2-FFG1157C

— Resource limitation (FPGA / PCIe Gen3 x8)



Slight increase in DSP utilization.

BRAM is restrictive for $n > 2^{15}$ ([58-65]% for NTT permutations)

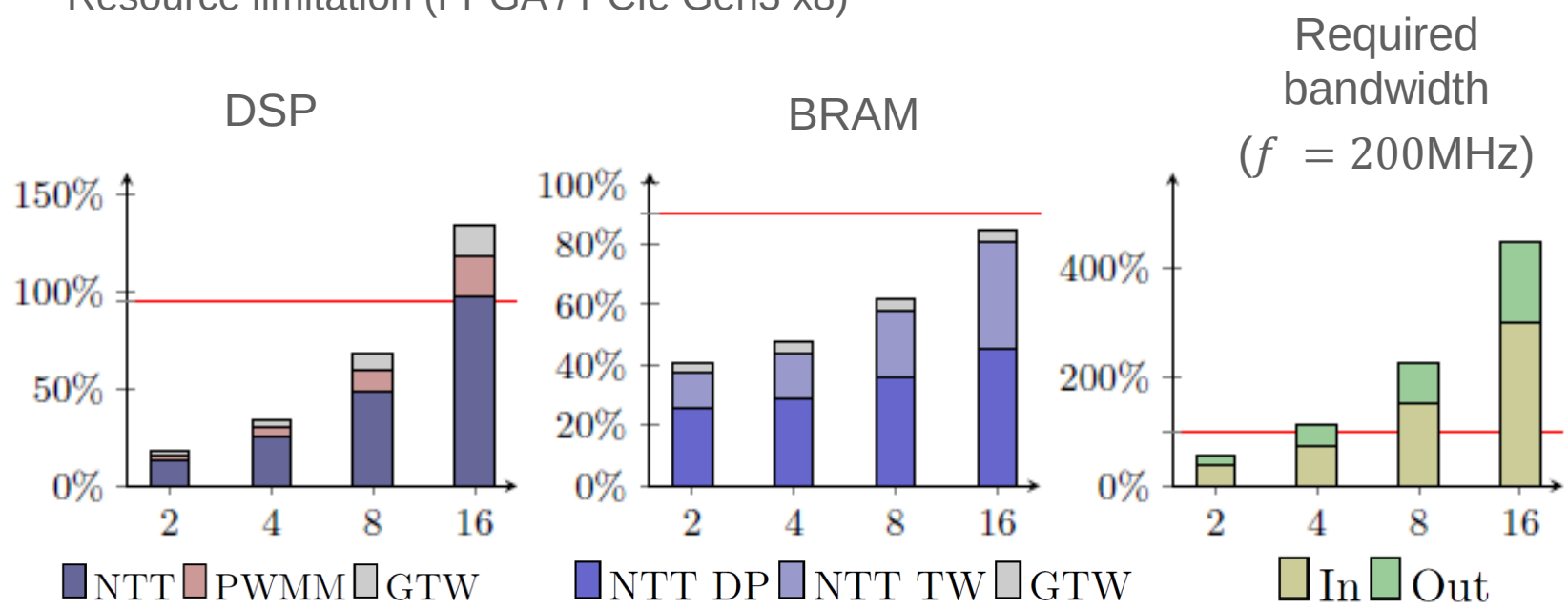
Required bandwidth is acheivable

RPM CHARACTERIZATION PROJECTIONS (2)

- **Impact of the streaming width** w ($n = 2^{14}$ and $\log_2 q_i = 30$):

Xilinx Virtex 7: XC7VX690T-2-FFG1157C

— Resource limitation (FPGA / PCIe Gen3 x8)



Great increase in DSP utilization.

Increase of BRAM utilization.

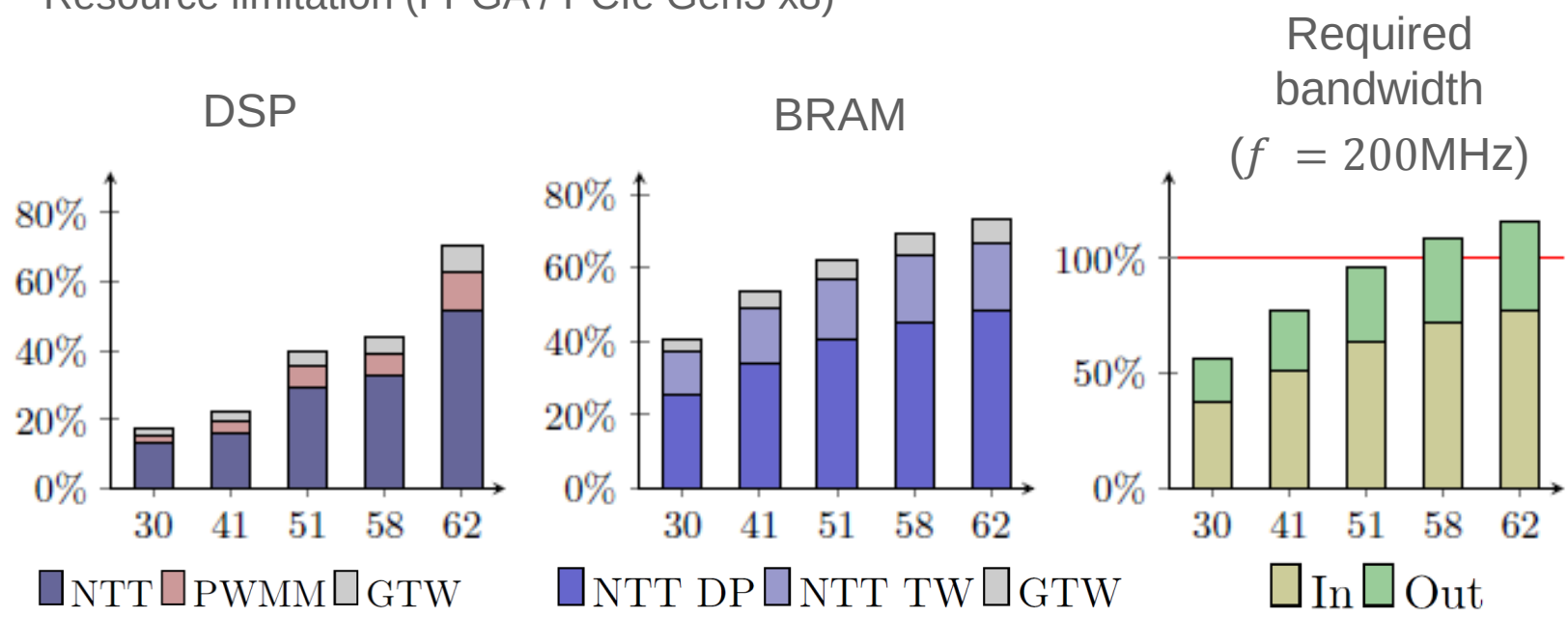
Required bandwidth is prohibitive

RPM CHARACTERIZATION PROJECTIONS (3)

- **Impact of the RNS prime size** $\log_2 q_i$ ($n = 2^{14}$ and $w = 2$)**:**

Xilinx Virtex 7: XC7VX690T-2-FFG1157C

— Resource limitation (FPGA / PCIe Gen3 x8)



Balanced impact on DSP and BRAM utilization.

Required Bandwidth may become restrictive.

PERFORMANCE PROJECTIONS: FV-RNS APPLICATION

- **Raw performances:**

RPM / s $\frac{f_{RPM}}{\frac{n}{w}}$

Required bandwidth $\sim f_{RPM} 3w \log_2 q_i$

- **Performance projection @200MHz:**

With respect to timing from [HPS18] ($\lambda > 128$)

Parameters						RPM	Mul.RPM		Relin.RPM		Total	BW
L	n	S_q	s	k	w	1/ms	$3(k+k')$	ms (**su**)	$2k^2$	ms (**su**)	ms (su)	GB/s
1	2^{12}	94		4		97.7	27	0.3 (**37.3**)	32	0.3 (**5.1**)	6.3 (2.8)	4.5
5	2^{13}	141		5		48.8	33	0.7 (**44.7**)	50	1.0 (**6.9**)	18.2 (3.0)	4.5
10	2^{14}	235	30	8	2	24.4	51	2.1 (**49.9**)	128	5.2 (**7.2**)	63 (3.1)	4.5
20	2^{14}	376		13			81	3.3 (**48.4**)	338	13.8 (**6.3**)	119.3 (2.9)	4.5
30	2^{15}	564		19		12.2	117	9.6 (**53**)	722	59.1 (**7.6**)	442.6 (3.0)	4.5
20	2^{14}	376	30	13	2	24.4	81	3.3 (**48.4**)	338	13.8 (**6.3**)	119.3 (2.9)	4.5
					4	48.8		1.7 (**96.7**)		6.9 (**12.6**)	110.7 (3.2)	9
					8	97.7		0.8 (**193.5**)		3.5 (**25.1**)	106.4 (3.3)	18
					16	195.3		0.4 (**387**)		1.7 (**50.3**)	104.2 (3.4)	36
20	2^{14}	376	30	13	2	24.4	81	3.3 (**48.4**)	338	13.8 (**6.3**)	119.3 (2.9)	4.5
			41	10			63	2.6 (**62.2**)	200	8.2 (**10.6**)	112.9 (3.1)	6.15
			51	8			51	2.1 (**76.8**)	128	5.2 (**16.6**)	109.4 (3.2)	7.65
			58	7			45	1.8 (**87.1**)	98	4.0 (**21.7**)	108 (3.2)	8.7
			62	7								9.3

PERFORMANCE PROJECTIONS: FV-RNS APPLICATION

- **Raw performances:**

RPM / s $\frac{f_{RPM}}{\frac{n}{w}}$

Required bandwidth $\sim f_{RPM} 3w \log_2 q_i$

- **Performance projection @200MHz:**

With respect to timing from [HPS18] ($\lambda > 128$)

Parameters						RPM	Mul.RPM		Relin.RPM		Total	BW
L	n	S_q	s	k	w	1/ms	$3(k+k')$	ms (**su**)	$2k^2$	ms (**su**)	ms (su)	GB/s
1	2^{12}	94		4		97.7	27	0.3 (**37.3**)	32	0.3 (**5.1**)	6.3 (2.8)	4.5
5	2^{13}	141		5		48.8	33	0.7 (**44.7**)	50	1.0 (**6.9**)	18.2 (3.0)	4.5
10	2^{14}	235	30	8	2	24.4	51	2.1 (**49.9**)	128	5.2 (**7.2**)	63 (3.1)	4.5
20	2^{14}	376		13			81	3.3 (**48.4**)	338	13.8 (**6.3**)	119.3 (2.9)	4.5
30	2^{15}	564		19		12.2	117	9.6 (**53**)	722	59.1 (**7.6**)	442.6 (3.0)	4.5
					2	24.4		3.3 (**48.4**)		13.8 (**6.3**)	119.3 (2.9)	4.5
20	2^{14}	376	30	13	4	48.8	81	1.7 (**96.7**)	338	6.9 (**12.6**)	110.7 (3.2)	9
					8	97.7		0.8 (**193.5**)		3.5 (**25.1**)	106.4 (3.3)	18
					16	195.3		0.4 (**387**)		1.7 (**50.3**)	104.2 (3.4)	36
			30	13			81	3.3 (**48.4**)	338	13.8 (**6.3**)	119.3 (2.9)	4.5
			41	10			63	2.6 (**62.2**)	200	8.2 (**10.6**)	112.9 (3.1)	6.15
20	2^{14}	376	51	8	2	24.4	51	2.1 (**76.8**)	128	5.2 (**16.6**)	109.4 (3.2)	7.65
			58	7			45	1.8 (**87.1**)	98	4.0 (**21.7**)	108 (3.2)	8.7
			62	7								9.3

Scalability w.r.t. multiplicative depth:

- Speedup (su) is scalable.
- Realistic bandwidth usage.
- Timing after RPM speedup:
  - Basis ext. & Scaling: [77-86] %
  - RPMs: [9-16] %
- RPM Vs NTT implementation?

PERFORMANCE PROJECTIONS: FV-RNS APPLICATION

- **Raw performances:**

RPM / s $\frac{f_{RPM}}{\frac{n}{w}}$

Required bandwidth $\sim f_{RPM} 3w \log_2 q_i$

- **Performance projection @200MHz:**

With respect to timing from [HPS18] ($\lambda > 128$)

Parameters						RPM	Mul.RPM		Relin.RPM		Total	BW
L	n	S_q	s	k	w	1/ms	$3(k+k')$	ms (**su**)	$2k^2$	ms (**su**)	ms (su)	GB/s
1	2^{12}	94		4		97.7	27	0.3 (**37.3**)	32	0.3 (**5.1**)	6.3 (2.8)	4.5
5	2^{13}	141		5		48.8	33	0.7 (**44.7**)	50	1.0 (**6.9**)	18.2 (3.0)	4.5
10	2^{14}	235	30	8	2	24.4	51	2.1 (**49.9**)	128	5.2 (**7.2**)	63 (3.1)	4.5
20	2^{14}	376		13			81	3.3 (**48.4**)	338	13.8 (**6.3**)	119.3 (2.9)	4.5
30	2^{15}	564		19		12.2	117	9.6 (**53**)	722	59.1 (**7.6**)	442.6 (3.0)	4.5
20	2^{14}	376	30	13	2	24.4	81	3.3 (**48.4**)	338	13.8 (**6.3**)	119.3 (2.9)	4.5
					4	48.8		1.7 (**96.7**)		6.9 (**12.6**)	110.7 (3.2)	9
					8	97.7		0.8 (**193.5**)		3.5 (**25.1**)	106.4 (3.3)	18
					16	195.3		0.4 (**387**)		1.7 (**50.3**)	104.2 (3.4)	36
20	2^{14}	376	30	13	2	24.4	81	3.3 (**48.4**)	338	13.8 (**6.3**)	119.3 (2.9)	4.5
			41	10			63	2.6 (**62.2**)	200	8.2 (**10.6**)	112.9 (3.1)	6.15
			51	8			51	2.1 (**76.8**)	128	5.2 (**16.6**)	109.4 (3.2)	7.65
			58	7			45	1.8 (**87.1**)	98	4.0 (**21.7**)	108 (3.2)	8.7
			62	7								9.3

Scalability w.r.t. multiplicative depth:

- Speedup (su) is scalable.
- Realistic bandwidth usage.
- Timing after RPM speedup:
  - Basis ext. & Scaling: [77-86] %
  - RPMs: [9-16] %
- RPM Vs NTT implementation?

Increasing parallelism:

- Greatly improves speedup.
- Bandwidth and DSPs may be quickly restrictive.

PERFORMANCE PROJECTIONS: FV-RNS APPLICATION

- **Raw performances:**

RPM / s $\frac{f_{RPM}}{\frac{n}{w}}$

Required bandwidth $\sim f_{RPM} 3w \log_2 q_i$

- **Performance projection @200MHz:**

With respect to timing from [HPS18] ($\lambda > 128$)

Parameters						RPM	Mul.RPM		Relin.RPM		Total	BW
L	n	S_q	s	k	w	1/ms	$3(k+k')$	ms (**su**)	$2k^2$	ms (**su**)	ms (su)	GB/s
1	2^{12}	94		4		97.7	27	0.3 (**37.3**)	32	0.3 (**5.1**)	6.3 (2.8)	4.5
5	2^{13}	141		5		48.8	33	0.7 (**44.7**)	50	1.0 (**6.9**)	18.2 (3.0)	4.5
10	2^{14}	235	30	8	2	24.4	51	2.1 (**49.9**)	128	5.2 (**7.2**)	63 (3.1)	4.5
20	2^{14}	376		13			81	3.3 (**48.4**)	338	13.8 (**6.3**)	119.3 (2.9)	4.5
30	2^{15}	564		19		12.2	117	9.6 (**53**)	722	59.1 (**7.6**)	442.6 (3.0)	4.5
					2	24.4		3.3 (**48.4**)		13.8 (**6.3**)	119.3 (2.9)	4.5
20	2^{14}	376	30	13	4	48.8	81	1.7 (**96.7**)	338	6.9 (**12.6**)	110.7 (3.2)	9
					8	97.7		0.8 (**193.5**)		3.5 (**25.1**)	106.4 (3.3)	18
					16	195.3		0.4 (**387**)		1.7 (**50.3**)	104.2 (3.4)	36
			30	13			81	3.3 (**48.4**)	338	13.8 (**6.3**)	119.3 (2.9)	4.5
			41	10			63	2.6 (**62.2**)	200	8.2 (**10.6**)	112.9 (3.1)	6.15
20	2^{14}	376	51	8	2	24.4	51	2.1 (**76.8**)	128	5.2 (**16.6**)	109.4 (3.2)	7.65
			58	7			45	1.8 (**87.1**)	98	4.0 (**21.7**)	108 (3.2)	8.7
			62	7								9.3

Scalability w.r.t. multiplicative depth:

- Speedup (su) is scalable.
- Realistic bandwidth usage.
- Timing after RPM speedup:
  - Basis ext. & Scaling: [77-86] %
  - RPMs: [9-16] %
- RPM Vs NTT implementation?

Increasing parallelism:

- Greatly improves speedup.
- Bandwidth and DSPs are quickly restrictive.

Increasing prime size:

- Slightly improves speedup.
- Balanced cost on DSP and BRAM usage.
- Bandwidth may be restrictive.

CONCLUSION & PERSPECTIVES

- **Hardware implementation for SHE should be flexible:**
  - Refinement of parameter range still in progress.
  - Multiplicative depth has significant impact on both n and $\log_2 q$.

- **Our response:**
  - Dataflow RNS-based NWC with on-the-fly generation of twiddles.
  - Exploiting DSP knowledge on DFT implementation.
  - Minimize the impact of $\log_2 q$ on hardware design.

- **Research perspectives:**
  - NTT Vs RPM?
  - Proper system integration
  - Design space exploration with SPIRAL

- **Application perspectives:**
  - Hybrid architecture for SHE acceleration

Thanks!

Questions?

Centre de Saclay
Nano-Innov PC 172 - 91191 Gif sur Yvette Cedex

INTRODUCTION : HOMOMORPHIC ENCRYPTION

- **Decryption function is an homomorphism:**

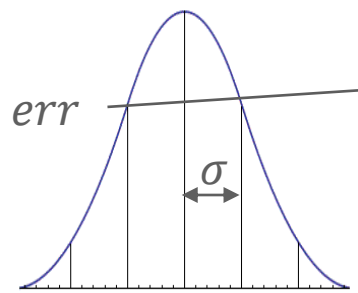
c_1, c_2 two ciphertexts such that $c_1 = \mathrm{Enc}(m_1)$ and $c_2 = \mathrm{Enc}(m_2)$ $\Rightarrow$ $\begin{cases} \mathrm{Dec}(c_1) \circ \mathrm{Dec}(c_2) = \mathrm{Dec}(c_1 \circledcirc c_2) \\ m_1 \circ m_2 \Leftrightarrow c_1 \circledcirc c_2 \end{cases}$

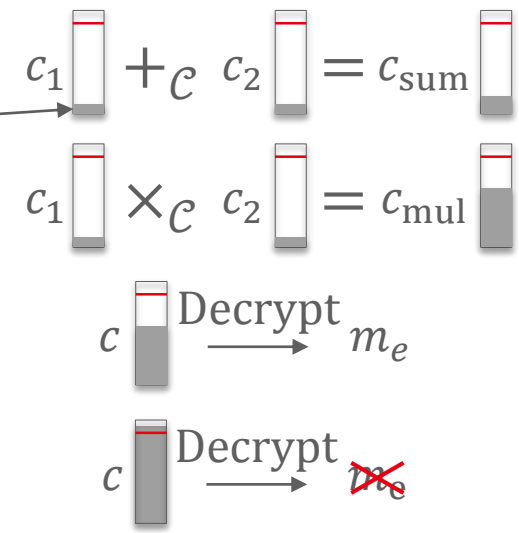
- **Semantic security : noise in ciphertexts**

$m \in \mathcal{M}$ message space

$m_e \in \mathcal{H}$ cleartext space

$c \in \mathcal{C}$ ciphertext space





$m \xrightarrow{\mathrm{Encode}} m_e \xrightarrow{\mathrm{Encrypt}} c$

$c \xrightarrow{\mathrm{Decrypt}} m_e \xrightarrow{\mathrm{Decode}} m$

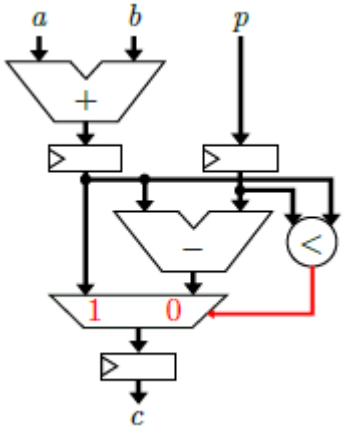
Error distribution χ_{err}

Usually $\chi_{err} = N(0, \sigma^2)$

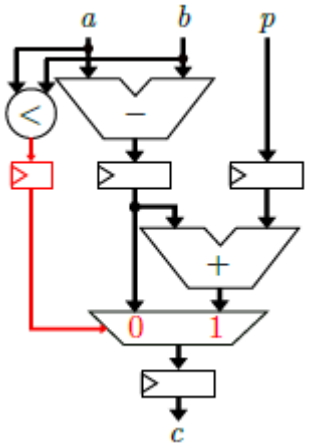
Homomorphic encryption has to be secure … <u>and correct</u> !

MODULAR ARITHMETIC

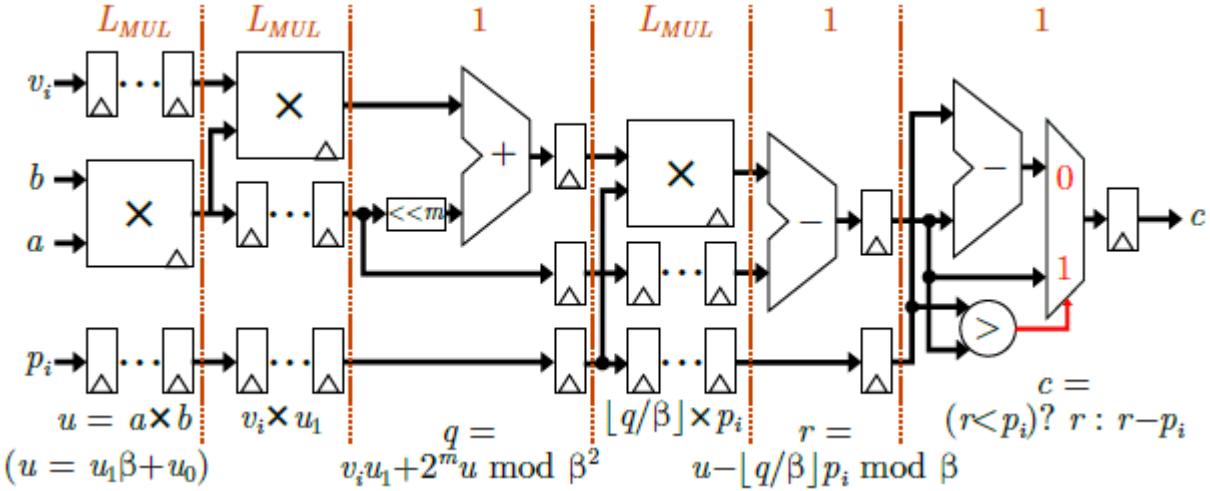
- **Modular Addition:**



- **Modular Subtraction:**

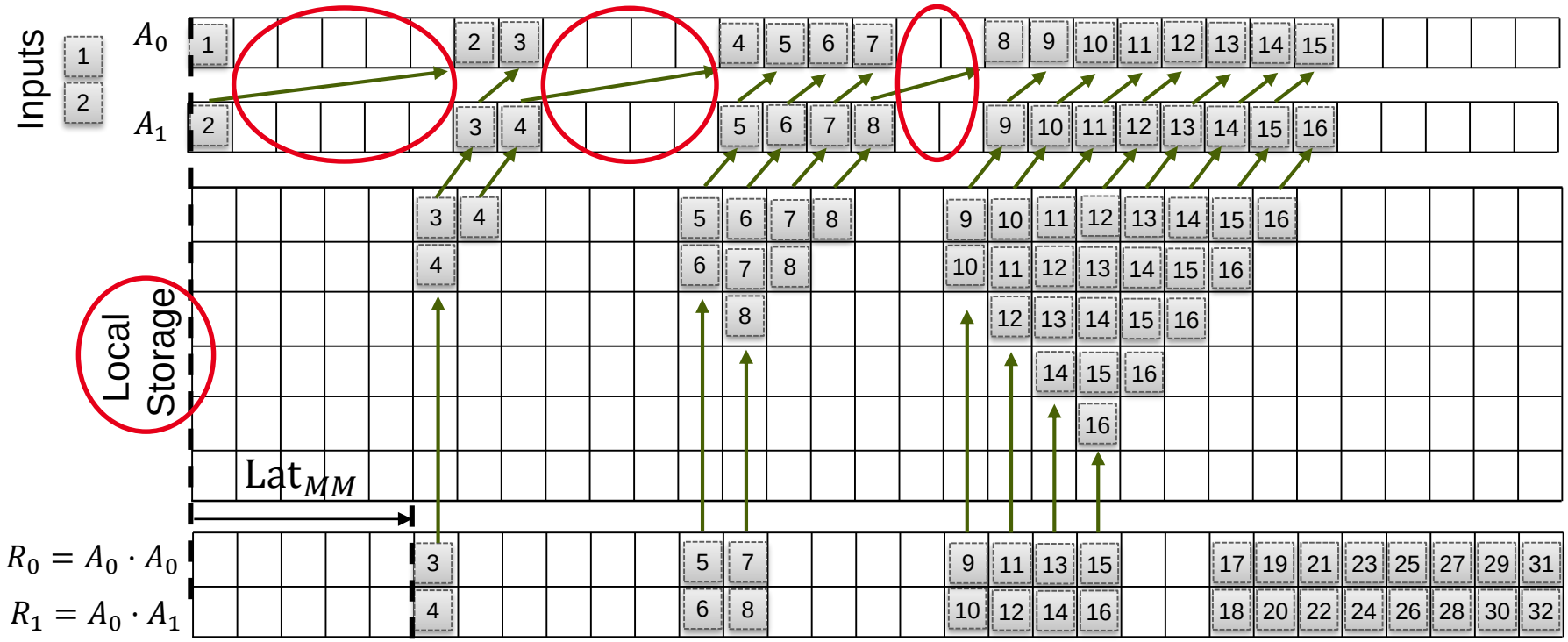


- **Modular Multiplication (NFLlib):**



GENERATION OF TWIDDLE FACTORS (1)

- **Problematic of twiddle generation:**
  - Data dependencies.
  - Modular multiplication latency.
  - Required throughput $T = \frac{n}{w}$.
- **Example of recurrence relation:**
  - $\psi^{2k} = \psi^k \cdot \psi^k$ and $\psi^{2k+1} = \psi^k \cdot \psi^{k+1}$
  - Intermediate storage in $O\left(\frac{n}{4}\right)$
  - Compute "at the earliest"
- **Example of Ψ generation ($n = 32$, $w = 2$):**



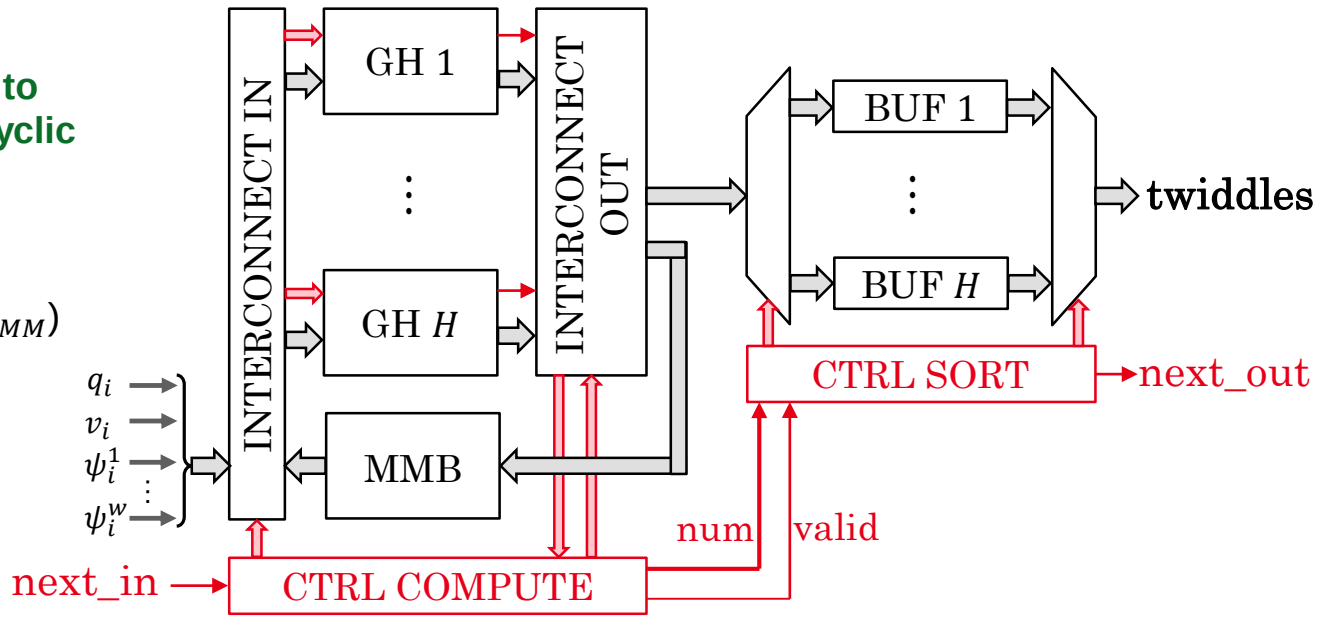
GENERATION OF TWIDDLE FACTORS (2)

- **Data flow twiddle generation:**

Sequential access to MMB (w MMs) with cyclic priority order

$$H = \left\lceil \frac{\mathrm{Lat}_{GEN}}{T} \right\rceil + 1$$

$(H = 3 \text{ when } T \gg \mathrm{Lat}_{MM})$



- **Minimize Generation Handler local storage:**

$$\mathrm{bunch}_t \begin{cases} \psi^{t+1} \\ \psi^{t+2} \\ \vdots \\ \psi^{t+w} \end{cases}$$

$$\text{twiddle set} \approx (\mathrm{bunch}_t)_{t=0}^{T-1}$$

$$f_j = \psi^{jw}$$

$$\mathrm{bunch}_{t_{next}} = f_j \cdot \mathrm{bunch}_{t_{last}}$$

$$(t_{next} = j + t_{last})$$

j is upper bounded by design parameter only

$\Downarrow$

Local storage independent of n!