Compact Ring LWE Cryptoprocessor

CHES 2014

Sujoy Sinha Roy[1], Frederik Vercauteren[1], Nele Mentens[1], Donald Donglong Chen[2] and Ingrid Verbauwhede[1]

[1]ESAT/COSIC and iMinds, KU Leuven

[2]Electronic Engineering, City University of Hong Kong

Outline

- Introduction to the ring-LWE problem and an encryption scheme
- Our optimization techniques
- Architecture of the ring-LWE Cryptoprocessor
- Results
- Conclusions

Modern Public Key Schemes

- Modern public-key schemes
  - RSA ← difficulty of factoring problem
  - DSA/ECDSA ← difficulty of Discrete Logarithm problem
- Intractable using classical computers
- Threat
  - Quantum computers destroy RSA, DSA and ECDSA

The Ring-LWE Problem

- Defined over a ring $R_q = \mathbb{Z}_q[\mathrm{x}]/\langle f \rangle$
  - A polynomial $a \in R_q$ is chosen uniformly
  - The secret $s \in R_q$ is a fixed polynomial
  - An error polynomial e is sampled from χ_σ
  - Compute $t = as + e \in R_q$
  - The ring-LWE distribution on $R_q \times R_q$ consists of tuples (a, t)
- Search ring-LWE problem: given many $(\mathbf{a}_i, t_i)$ samples, find secret s

Ring-LWE : Encryption Scheme

The Encryption Scheme : Key Generation

- Key Generation :
  - Choose two polynomials $r_1, r_2 \leftarrow \chi_\sigma$
  - Compute $p = r_1 - a \cdot r_2$
  - The secret key is r_2
  - The public key is (a, p)

The Encryption Scheme : Encryption/Decryption

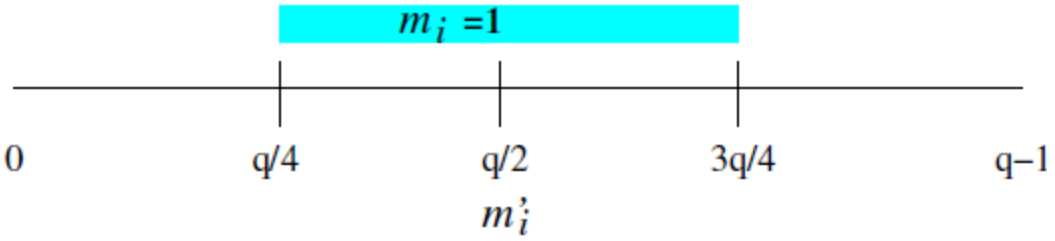
- Encryption
  - Input message *m* → encoded to polynomial $\bar{m}$: $\begin{matrix} 0 \to 0 \\ 1 \to (q-1)/2 \end{matrix}$
  - Choose $e_1, e_2, e_3 \in R_q$ with coefficients from $\mathcal{X}_\sigma$
  - Ciphertext

$$c_1 = a \cdot e_1 + e_2 \in R_q$$
$$c_2 = p \cdot e_1 + e_3 + \bar{m} \in R_q$$

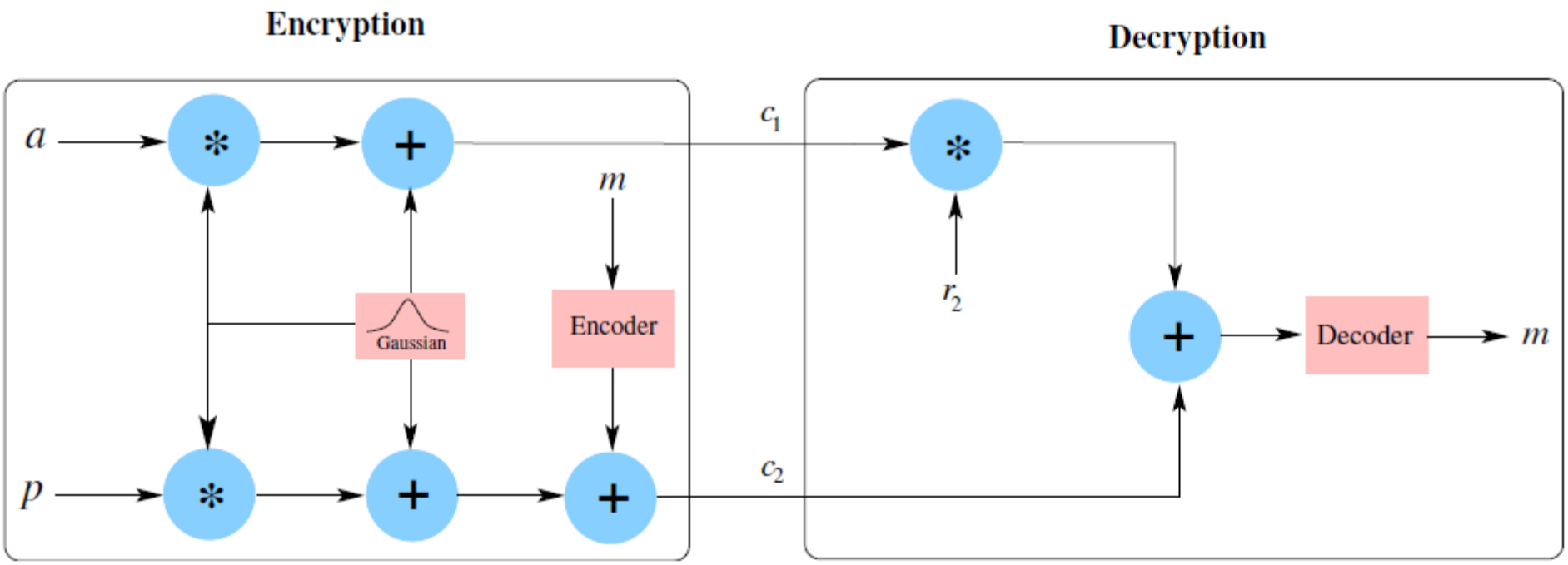
- Decryption

$$m' = c_1 \cdot r_2 + c_2 \in R_q$$

- Decoding compares the decrypted message coefficients



LWE Cryptosystem : Block Level Diagram



Polynomials have 256 or 512 coefficients and each coefficient is 13 or 14 bit

Standard deviation is small (less than 5)

Roadmap to Implementation

- Ring-LWE based encryption
  - Primitives
    - Discrete Gaussian Sampler
    - Polynomial Multiplier
  - Full cryptosystem
- Discrete Gaussian sampler architecture ---- (Presented in SAC 2013)
  - Knuth-Yao random walk

High Precision Discrete Gaussian Sampling on FPGAs, SAC 2013

Polynomial Multiplication

Polynomial Multiplication Algorithms

- Schoolbook multiplication : complexity n^2
- Karatsuba multiplication : complexity $n^{1.585}$
- FFT based multiplication : **complexity (*n log n*)**
- Number Theoretic Transform (**NTT**) is a generalization of FFT
  - $e^{-\frac{2\pi i}{n}} \leftrightarrow$ *n*-th primitive root of unity in $\mathbb{Z}_q$
  - Involves integer arithmetic modulo *q*

Polynomial Multiplication : NTT

- Polynomial multiplication : $c(x) = a(x)b(x)$
  - $2n$ point NTT : $c = NTT^{-1}_{\omega_{2n}}\left(NTT_{\omega_{2n}}(a) * NTT_{\omega_{2n}}(b)\right)$

Polynomial Multiplication : NTT

- Polynomial multiplication : $c(x) = a(x)b(x)$
  - $2n$ point NTT : $c = NTT^{-1}_{\omega_{2n}}\left(NTT_{\omega_{2n}}(a) * NTT_{\omega_{2n}}(b)\right)$
- Special optimization : $c(x) = a(x)b(x) \bmod f(x)$ where $f = x^n + 1$
  - **Negative-wrapped convolution** : using n point NTT
    - n is a power of two and prime $q \equiv 1 \mod 2n$
    - Scaling $a'(x) = a(\omega_{2n} \cdot x)$ and $b'(x) = b(\omega_{2n} \cdot x)$
    - $c' = NTT^{-1}_{\omega_n}(NTT_{\omega_n}(a') * NTT_{\omega_n}(b'))$
    - Final scaling is required to compute *c(x)* from $c'(x) = c(\omega_{2n} \cdot x)$

The basic NTT Algorithm

Iterative NTT

Input: Polynomial $a(x) \in \mathbb{Z}_q[\mathbf{x}]$ of degree $n-1$ and n-th primitive root $\omega_n \in \mathbb{Z}_q$ of unity
Output: Polynomial $A(x) \in \mathbb{Z}_q[\mathbf{x}] = \mathrm{NTT}(a)$

```
1  begin
2      A ← BitReverse(a);
3      for m = 2 to n by m = 2m do
4          ω_m ← ω_n^{n/m} ;
5          ω ← 1 ;
6          for j = 0 to m/2 − 1 do
7              for k = 0 to n − 1 by m do
8                  t ← ω · A[k + j + m/2] ;
9                  u ← A[k + j] ;
10                 A[k + j] ← u + t ;
11                 A[k + j + m/2] ← u − t ;
12             end
13             ω ← ω · ω_m ;
14         end
15     end
16 end
```

The NTT Algorithm

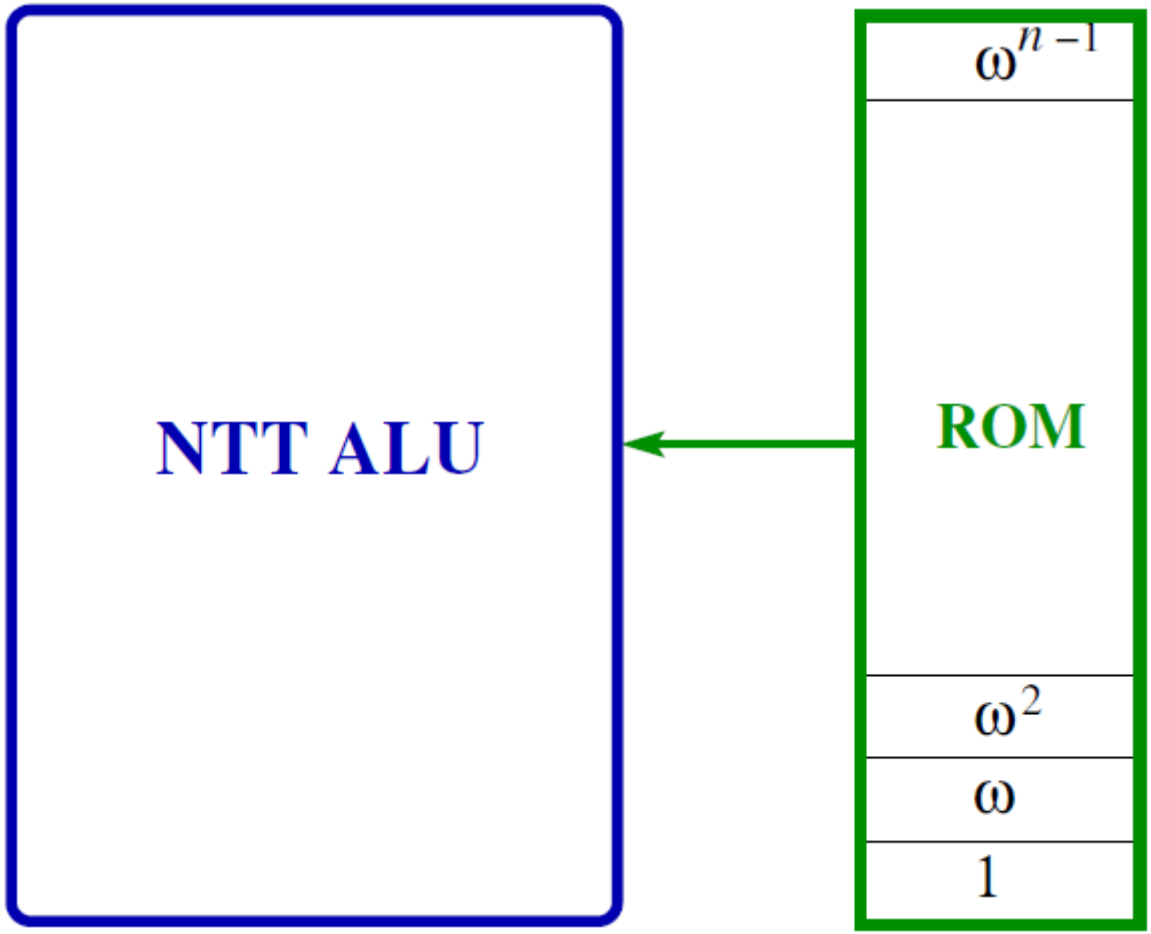
Iterative NTT

Input: Polynomial $a(x) \in \mathbb{Z}_q[\mathrm{x}]$ of degree $n-1$ and n-th primitive root $\omega_n \in \mathbb{Z}_q$ of unity
Output: Polynomial $A(x) \in \mathbb{Z}_q[\mathrm{x}] = \mathrm{NTT}(a)$

```
1  begin
2      A ← BitReverse(a);
3      for m = 2 to n by m = 2m do
4          ω_m ← ω_n^{n/m} ;
5          ω ← 1 ;
6          for j = 0 to m/2 − 1 do
7              for k = 0 to n − 1 by m do
8                  t ← ω · A[k + j + m/2] ;
9                  u ← A[k + j] ;
10                 A[k + j] ← u + t ;
11                 A[k + j + m/2] ← u − t ;
12             end
13             ω ← ω · ω_m ;
14         end
15     end
16 end
```

Fixed Computation Cost

NTT Core

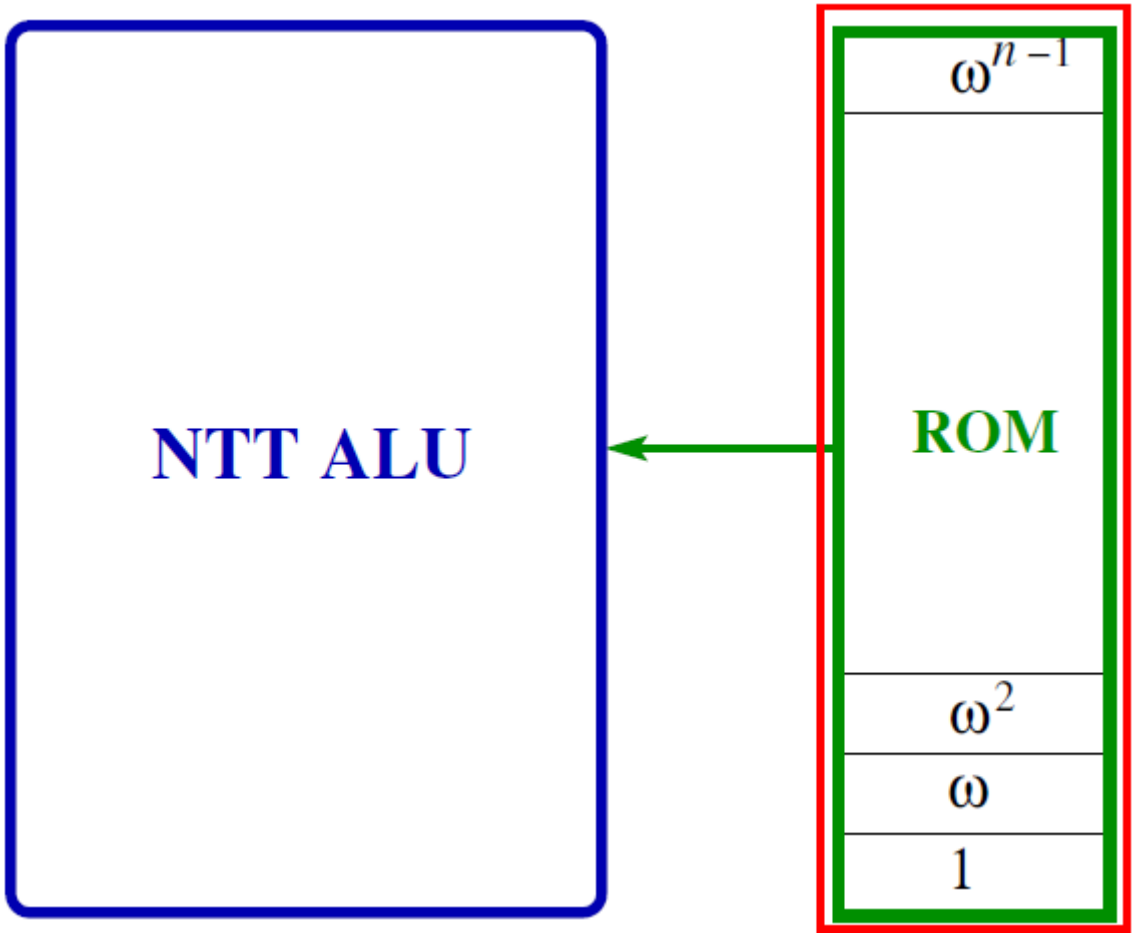


Pöppelmann et al. Latincrypt 2012

Our Target : Compact Architecture

- **Minimize Memory Requirement**

Optimization in Area



This increases computation cost !

Computational Optimizations

Optimization in Computation: Step 1

Iterative NTT

```
1  begin
2      A ← BitReverse(a);
3      for m = 2 to n by m = 2m do
4          ω_m ← ω_N^{N/m} ;
5          for k = 0 to n − 1 by m do
6              ω ← 1 ;
7              for j = 0 to m/2 − 1 do
8                  t ← ω · A[k + j + m/2] ;
9                  u ← A[k + j] ;
10                 A[k + j] ← u + t ;
11                 A[k + j + m/2] ← u − t ;
12                 ω ← ω · ω_m ;
13             end
14         end
15     end
16 end
```

Pöppelmann et al. Latincrypt 2012

Aysu et al. HOST 2013

Iterative NTT

```
1  begin
2      A ← BitReverse(a);
3      for m = 2 to n by m = 2m do
4          ω_m ← ω_N^{N/m} ;
5          ω ← 1 ;
6          for j = 0 to m/2 − 1 do
7              for k = 0 to n − 1 by m do
8                  t ← ω · A[k + j + m/2] ;
9                  u ← A[k + j] ;
10                 A[k + j] ← u + t ;
11                 A[k + j + m/2] ← u − t ;
12             end
13             ω ← ω · ω_m ;
14         end
15     end
16 end
```

Our

Reduction in fixed computation overhead

Optimization in Computation: Step 2

- Forward negative wrapped NTT requires **pre-scaling** of the input polynomials

$$\bar{a} = (a_0, \psi a_1, ..., \psi^{n-1} a_{n-1}) \quad \bar{b} = (b_0, \psi b_1, ..., \psi^{n-1} b_{n-1}) \text{ where } \psi = \omega_{2n}$$

- Our implementation is **free** from this pre-scaling

Reduction in the pre-scaling overhead

Optimization in Computation: Step 2

Iterative NTT

Input: Polynomial $a(x) \in \mathbb{Z}_q[\mathrm{x}]$ of degree $n-1$ and n-th primitive root $\omega_n \in \mathbb{Z}_q$ of unity
Output: Polynomial $A(x) \in \mathbb{Z}_q[\mathrm{x}] = \mathrm{NTT}(a)$

```
1  begin
2      A ← BitReverse(a);
3      for m = 2 to n by m = 2m do
4          ω_m ← ω_n^{n/m} ;
5          ω ← 1 ;
6          for j = 0 to m/2 − 1 do
7              for k = 0 to n − 1 by m do
8                  t ← ω · A[k + j + m/2] ;
9                  u ← A[k + j] ;
10                 A[k + j] ← u + t ;
11                 A[k + j + m/2] ← u − t ;
12             end
13             ω ← ω · ω_m ;
14         end
15     end
16 end
```

Optimization in Computation: Step 2

Iterative NTT

Input: Polynomial $a(x) \in \mathbb{Z}_q[\mathbf{x}]$ of degree $n-1$ and n-th primitive root $\omega_n \in \mathbb{Z}_q$ of unity
Output: Polynomial $A(x) \in \mathbb{Z}_q[\mathbf{x}] = \mathrm{NTT}(a)$

```
1  begin
2      A ← BitReverse(a);
3      for m = 2 to n by m = 2m do
4          ω_m ← ω_n^{n/m} ;
5          ω ← ω_{2m} ;
6          for j = 0 to m/2 − 1 do
7              for k = 0 to n − 1 by m do
8                  t ← ω · A[k + j + m/2] ;
9                  u ← A[k + j] ;
10                 A[k + j] ← u + t ;
11                 A[k + j + m/2] ← u − t ;
12             end
13             ω ← ω · ω_m ;
14         end
15     end
16 end
```

Reduction in pre-scaling overhead

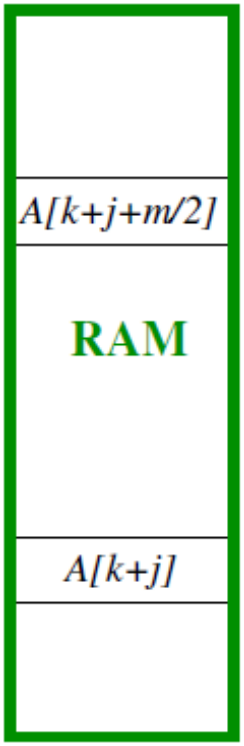
Optimization Step 3 : Memory Access Scheme

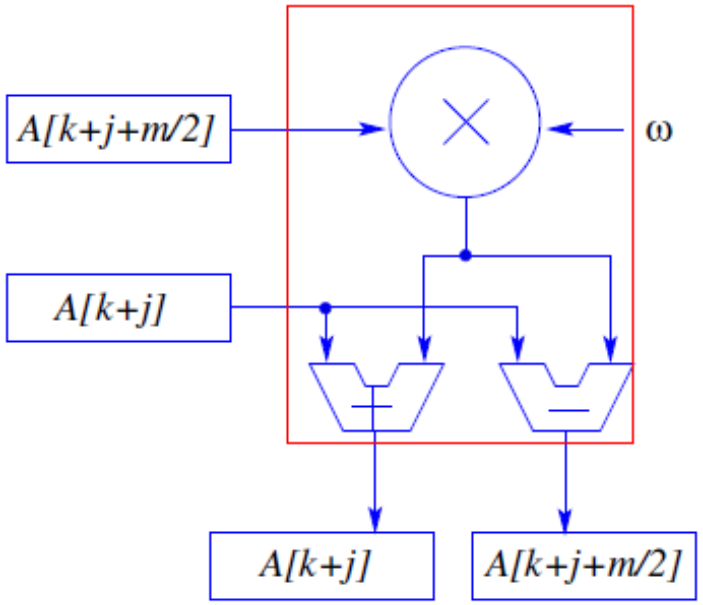
Memory Access : Motivation 1

Iterative NTT

```
1  begin
2      A ← BitReverse(a);
3      for m = 2 to n by m = 2m do
4          ω_m ← ω_N^{N/m} ;
5          ω ← ω_{2m}
6          for j = 0 to m/2 − 1 do
7              for k = 0 to n − 1 by m do
8                  t ← ω · A[k + j + m/2] ;
9                  u ← A[k + j] ;
10                 A[k + j] ← u + t ;
11                 A[k + j + m/2] ← u − t ;
12             end
13             ω ← ω · ω_m ;
14         end
15     end
16 end
```

- Two coefficients are accessed from RAM
- One arithmetic operation is performed
- Two new coefficients are written in RAM

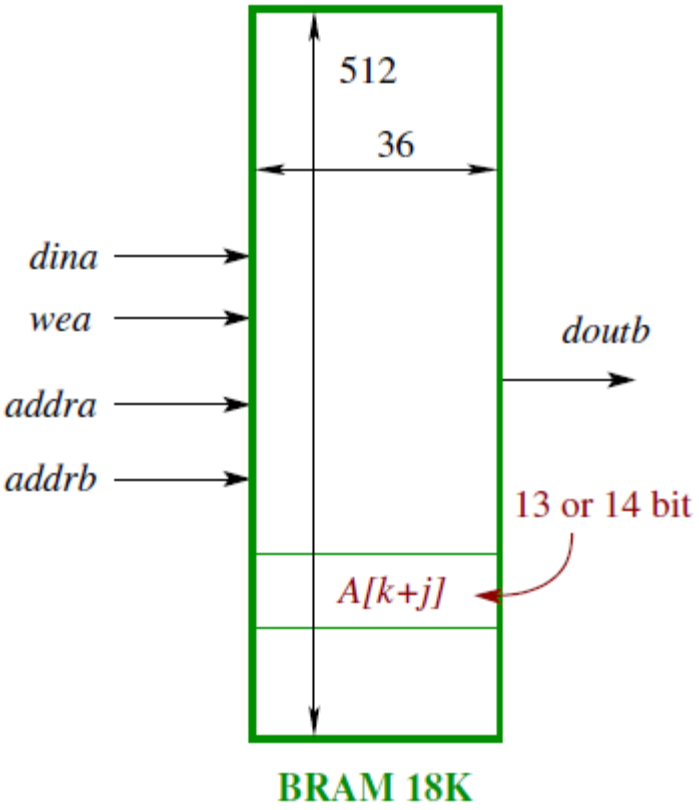




Arithmetic blocks remains idle

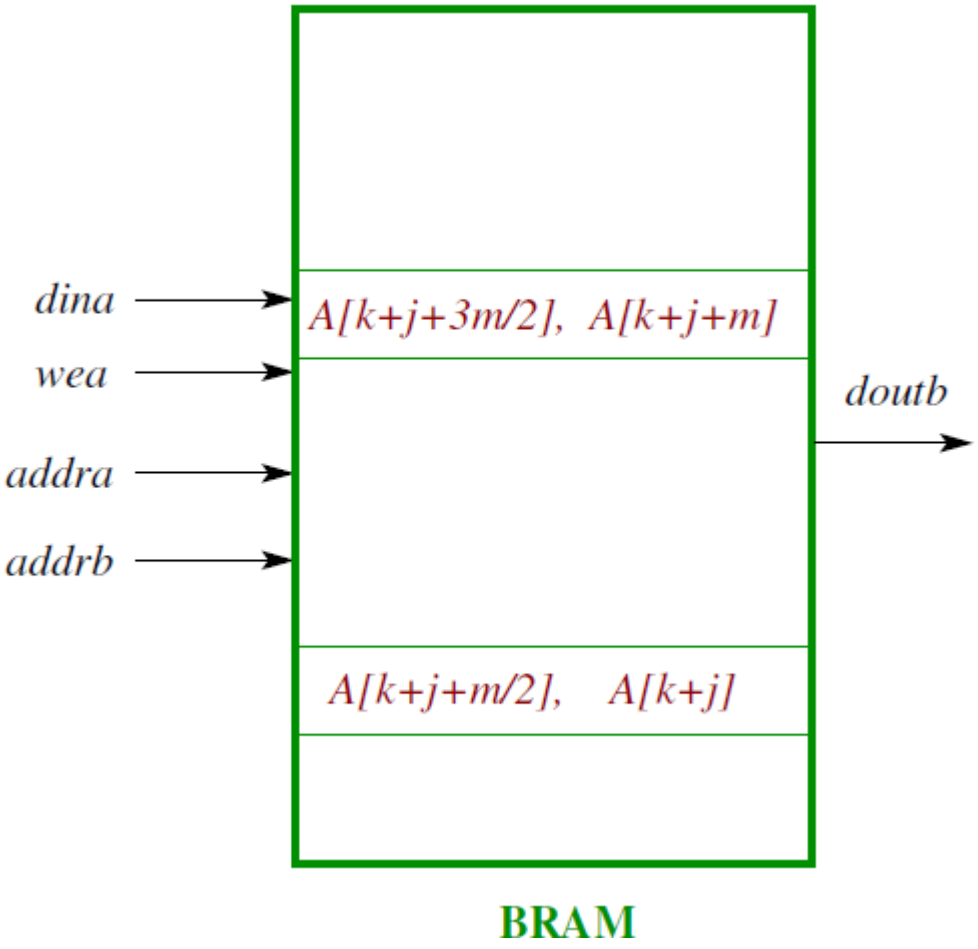
Memory Access : Motivation 2

- Xilinx BRAM Slice of 18Kb
  - Width of words : 36 bits
  - Depth of RAM : 512
- Polynomials in LWE
  - LWE256 : q= 7681 (13 bit)
  - LWE512 : q= 12289 (14 bit)
- Two coefficients can be stored in one word



Our memory efficient NTT …

Optimization : Memory Access



In this scheme

- Two coefficients are in one word
- Two pairs are processed together

Optimization : Memory Access

Iterative NTT : Memory Efficient Version

Input: Polynomial $a(x) \in \mathbb{Z}_q[\mathbf{x}]$ of degree $n-1$ and n-th primitive root $\omega_n \in \mathbb{Z}_q$ of unity
Output: Polynomial $A(x) \in \mathbb{Z}_q[\mathbf{x}] = \mathrm{NTT}(a)$

1 **begin**
2 $A \leftarrow BitReverse(a)$; /* Coefficients are stored in the memory as proper pairs */
3 **for** $m = 2$ *to* $n/2$ *by* $m = 2m$ **do**
4 $\omega_m \leftarrow m$-th primitiveroot(1) ;
5 $\omega \leftarrow squareroot(\omega_m)$ or 1 /* Depending on forward or backward NTT */ ;
6 **for** $j = 0$ *to* $m/2 - 1$ **do**
7 **for** $k = 0$ *to* $n/2 - 1$ *by* m **do**
8 $(t_1, u_1) \leftarrow (A[k + j + m/2], A[k + j])$ /* From MEMORY[k+j] */ ;
9 $(t_2, u_2) \leftarrow (A[k + m + j + m/2], A[k + m + j])$ /* MEMORY[k+j+m/2]
10 $t_1 \leftarrow \omega \cdot t_1$;
11 $t_2 \leftarrow \omega \cdot t_2$;
12 $(A[k + j + m/2], A[k + j]) \leftarrow (u_1 - t_1, u_1 + t_1)$;
13 $(A[k + m + j + m/2], A[k + m + j]) \leftarrow (u_2 - t_2, u_2 + t_2)$;
14 $MEMORY[k + j] \leftarrow (A[k + j + m], A[k + j])$;
15 $MEMORY[k + j + m/2] \leftarrow (A[k + j + 3m/2], A[k + j + m/2])$;
16 **end**
17 $\omega \leftarrow \omega \cdot \omega_n$;
18 **end**
19 **end**

No idle cycles

BRAM efficiency

Optimization : Ring-LWE Encryption Scheme

Pöppelmann, SAC 2013

Encryption

$$\tilde{e_1} \leftarrow NTT(e_1)$$
$$c_1 \leftarrow INTT(\tilde{a} * \tilde{e_1}) + \tilde{e_2}$$
$$c_2 \leftarrow INTT(\tilde{p} * \tilde{e_1}) + e_3 + \bar{m}$$

Decryption

$$\tilde{c_1} \leftarrow NTT(c_1)$$
$$m' \leftarrow INTT(\tilde{c_1} * \tilde{r_2}) + c_2)$$

#NTT = 5

Our Scheme

Encryption

$$\tilde{e_1} \leftarrow NTT(e_1); \quad \tilde{e_2} \leftarrow NTT(e_2)$$
$$(\tilde{c_1}, \tilde{c_2}) \leftarrow (\tilde{a} * \tilde{e_1} + \tilde{e_2};\ \tilde{p} * \tilde{e_1} + NTT(e_3 + \bar{m}))$$

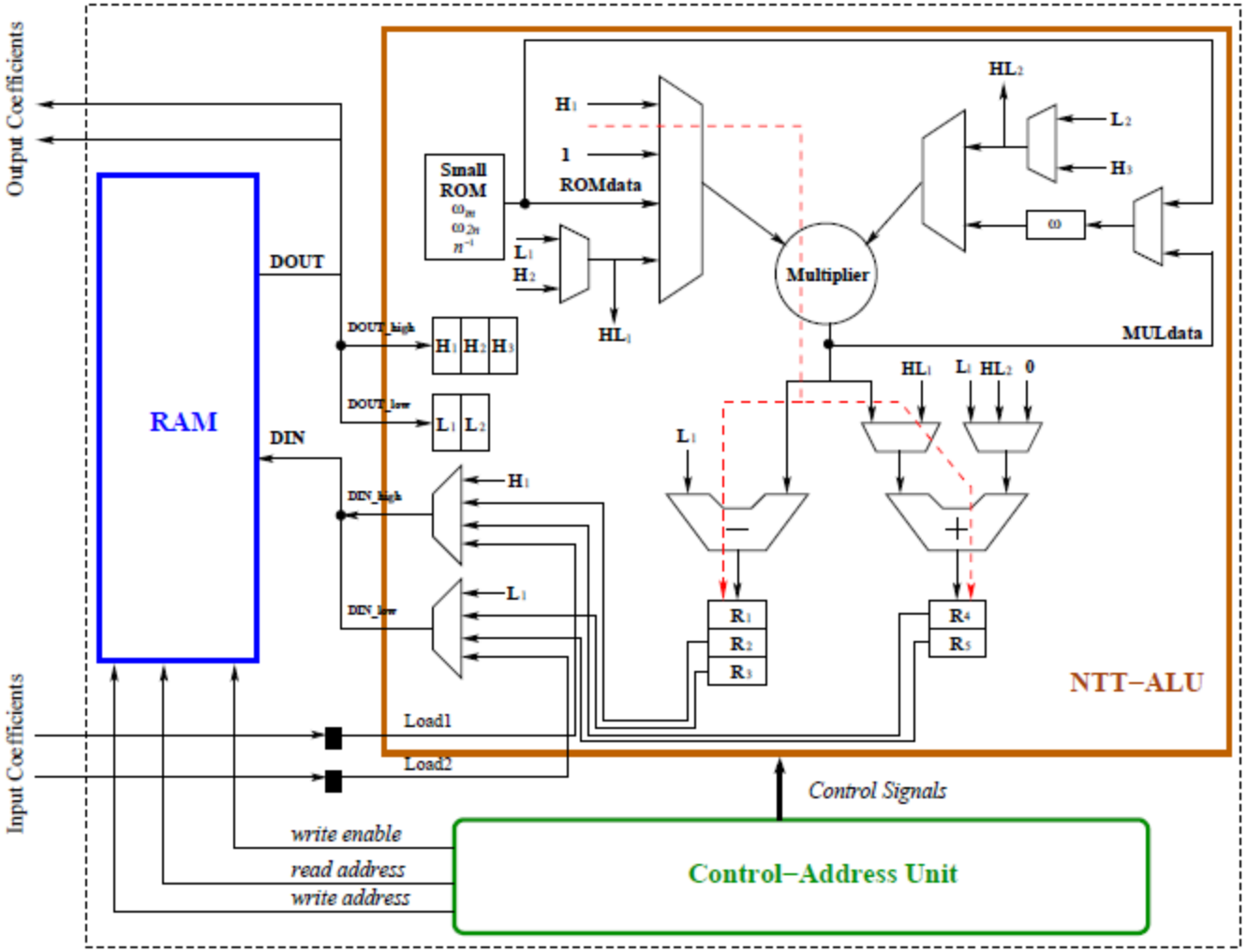
Decryption

$$m' = INTT(\tilde{c_1} * \tilde{r_2} + \tilde{c_2})$$

#NTT = 4

The NTT Core

The NTT Core


Output Coefficients
RAM
DOUT
DIN
Input Coefficients
Small ROM
ROMdata
Multiplier
MULdata
DOUT_high
DOUT_low
DIN_high
DIN_low
Load1
Load2
write enable
read address
write address
Control Signals
Control–Address Unit
NTT–ALU

Memory Efficient NTT

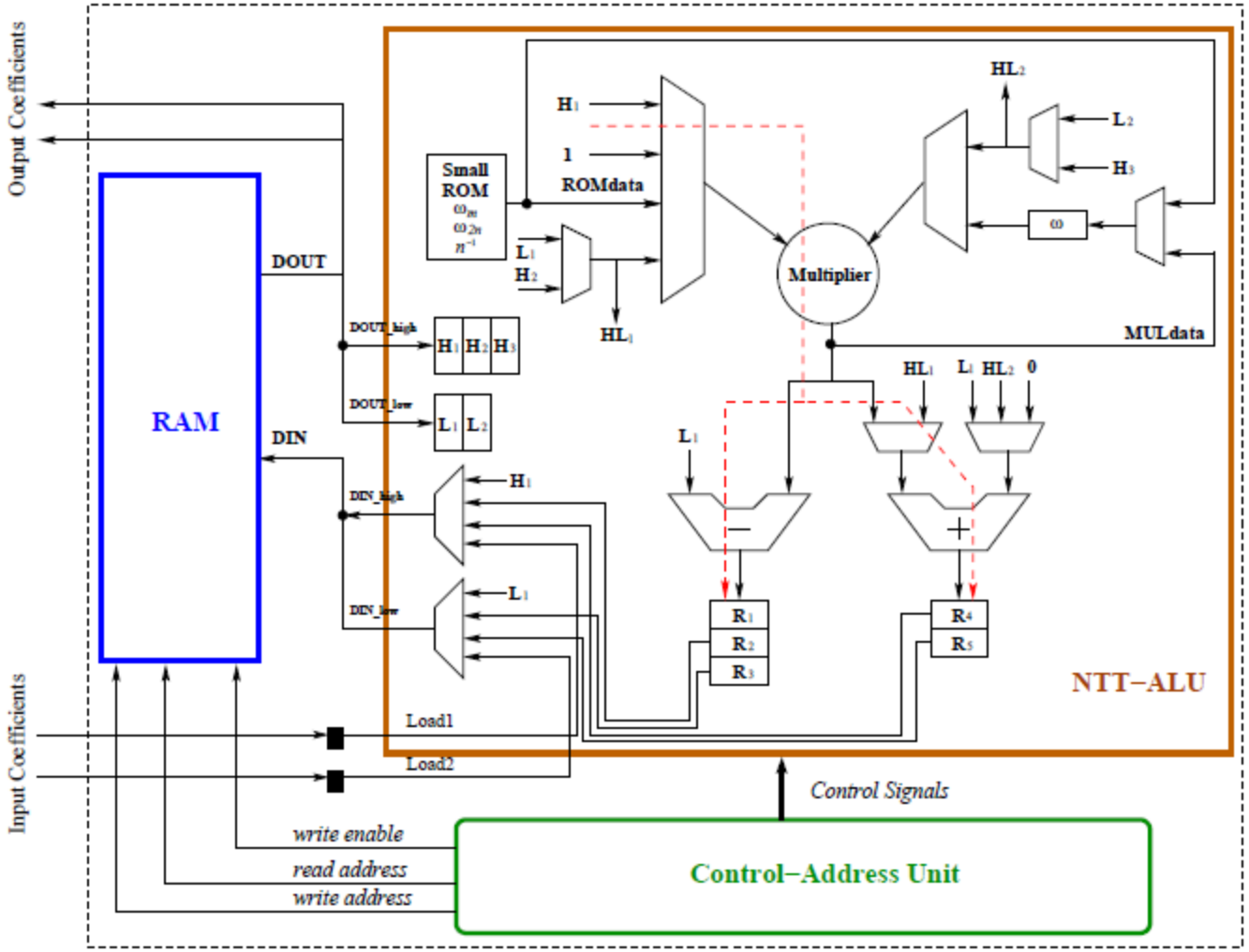
Iterative NTT : Memory Efficient Version

Input: Polynomial $a(x) \in \mathbb{Z}_q[\mathbf{x}]$ of degree $n-1$ and n-th primitive root $\omega_n \in \mathbb{Z}_q$ of unity
Output: Polynomial $A(x) \in \mathbb{Z}_q[\mathbf{x}] = \mathrm{NTT}(a)$

1 **begin**
2 $A \leftarrow BitReverse(a)$; /* Coefficients are stored in the memory as proper pairs */
3 **for** $m = 2$ *to* $n/2$ *by* $m = 2m$ **do**
4 $\omega_m \leftarrow m$-th primitiveroot(1) ;
5 $\omega \leftarrow squareroot(\omega_m)$ or 1 /* Depending on forward or backward NTT */ ;
6 **for** $j = 0$ *to* $m/2 - 1$ **do**
7 **for** $k = 0$ *to* $n/2 - 1$ *by* m **do**
8 $(t_1, u_1) \leftarrow (A[k + j + m/2], A[k + j])$ /* From MEMORY[k+j] */ ;
9 $(t_2, u_2) \leftarrow (A[k + m + j + m/2], A[k + m + j])$ /* MEMORY[k+j+m/2]
10 $t_1 \leftarrow \omega \cdot t_1$;
11 $t_2 \leftarrow \omega \cdot t_2$;
12 $(A[k + j + m/2], A[k + j]) \leftarrow (u_1 - t_1, u_1 + t_1)$;
13 $(A[k + m + j + m/2], A[k + m + j]) \leftarrow (u_2 - t_2, u_2 + t_2)$;
14 $MEMORY[k + j] \leftarrow (A[k + j + m], A[k + j])$;
15 $MEMORY[k + j + m/2] \leftarrow (A[k + j + 3m/2], A[k + j + m/2])$;
16 **end**
17 $\omega \leftarrow \omega \cdot \omega_n$;
18 **end**
19 **end**

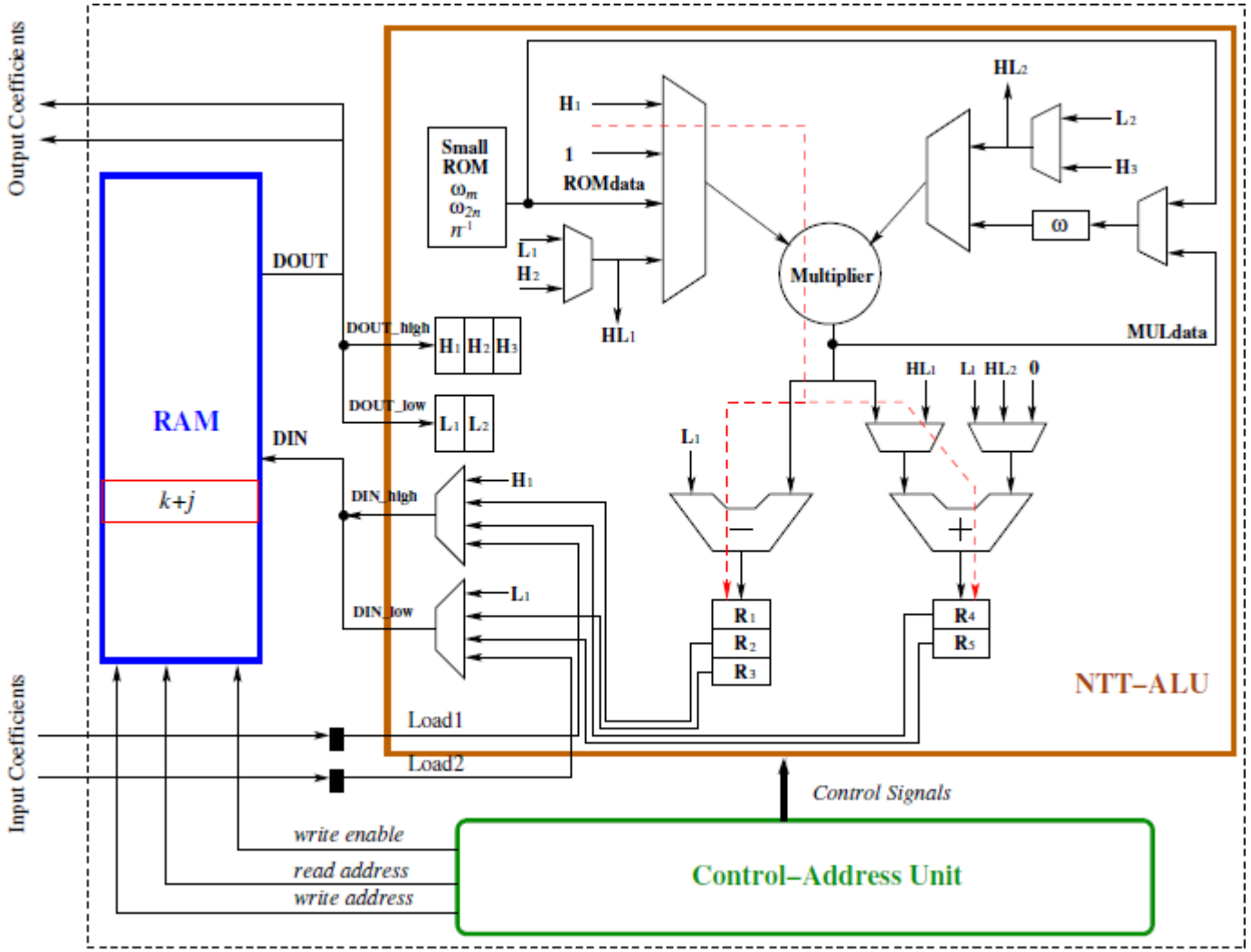
The NTT Cor

$$(H_1, L_1) \leftarrow (A[k+j+m/2], A[k+j]) \text{ /* From MEMORY[k+j] */ ;}$$
$$(R_1, R_4) \leftarrow (L_1 - \omega \cdot H_1, L_1 + \omega \cdot H_1) ;$$
$$(R_2, R_5) \leftarrow (R_1, R_4) ;$$
$$(H_1, L_1) \leftarrow (A[k+j+3m/2], A[k+j+m]) \text{ /* From MEMORY[k+j+m/2] */ ;}$$
$$(R_1, R_4) \leftarrow (L_1 - \omega \cdot H_1, L_1 + \omega \cdot H_1) ;$$
$$MEMORY[k+j] \leftarrow (R_4, R_5) ;$$
$$MEMORY[k+j+m/2] \leftarrow (R_2, R_3) ;$$



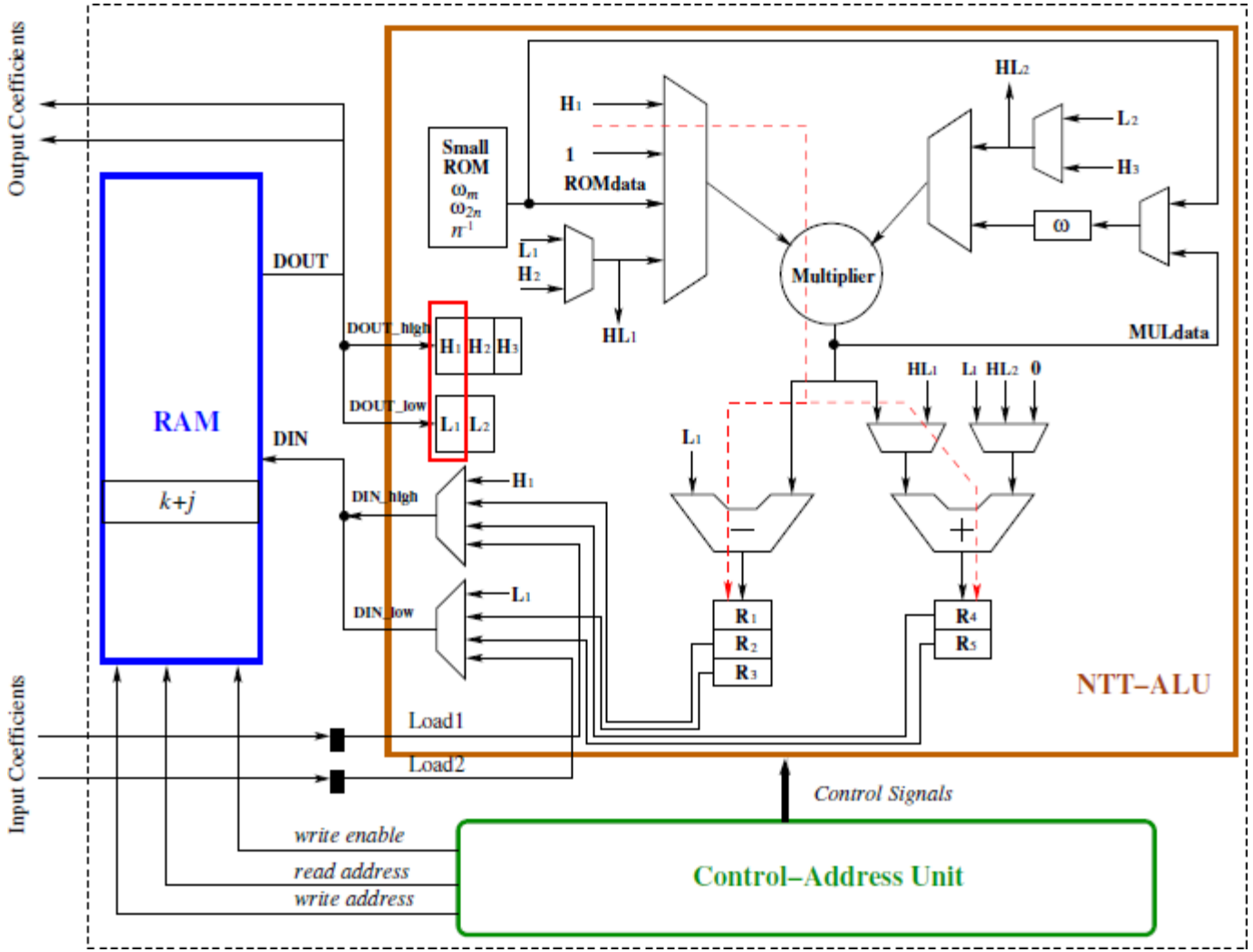
The NTT Cor

$$(H_1, L_1) \leftarrow (A[k+j+m/2], A[k+j]) \text{ /* From MEMORY[k+j] */ ;}$$
$$(R_1, R_4) \leftarrow (L_1 - \omega \cdot H_1, L_1 + \omega \cdot H_1) ;$$
$$(R_2, R_5) \leftarrow (R_1, R_4) ;$$
$$(H_1, L_1) \leftarrow (A[k+j+3m/2], A[k+j+m]) \text{ /* From MEMORY[k+j+m/2] */ ;}$$
$$(R_1, R_4) \leftarrow (L_1 - \omega \cdot H_1, L_1 + \omega \cdot H_1) ;$$
$$MEMORY[k+j] \leftarrow (R_4, R_5) ;$$
$$MEMORY[k+j+m/2] \leftarrow (R_2, R_3) ;$$



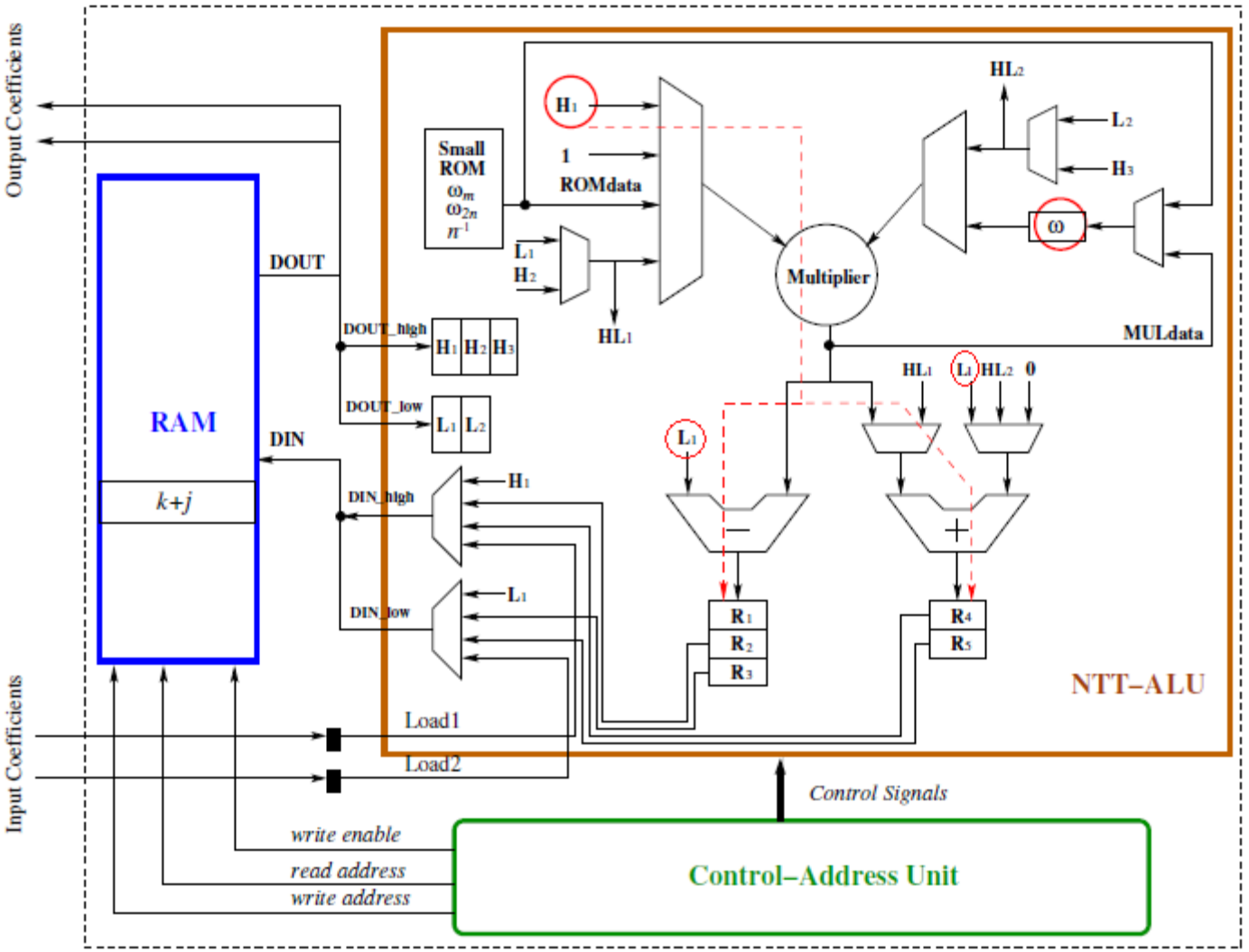
The NTT Cor

$$(H_1, L_1) \leftarrow (A[k+j+m/2], A[k+j]) \text{ /* From MEMORY[k+j] */ ;}$$
$$(R_1, R_4) \leftarrow (L_1 - \omega \cdot H_1, L_1 + \omega \cdot H_1) ;$$
$$(R_2, R_5) \leftarrow (R_1, R_4) ;$$
$$(H_1, L_1) \leftarrow (A[k+j+3m/2], A[k+j+m]) \text{ /* From MEMORY[k+j+m/2] */ ;}$$
$$(R_1, R_4) \leftarrow (L_1 - \omega \cdot H_1, L_1 + \omega \cdot H_1) ;$$
$$MEMORY[k+j] \leftarrow (R_4, R_5) ;$$
$$MEMORY[k+j+m/2] \leftarrow (R_2, R_3) ;$$



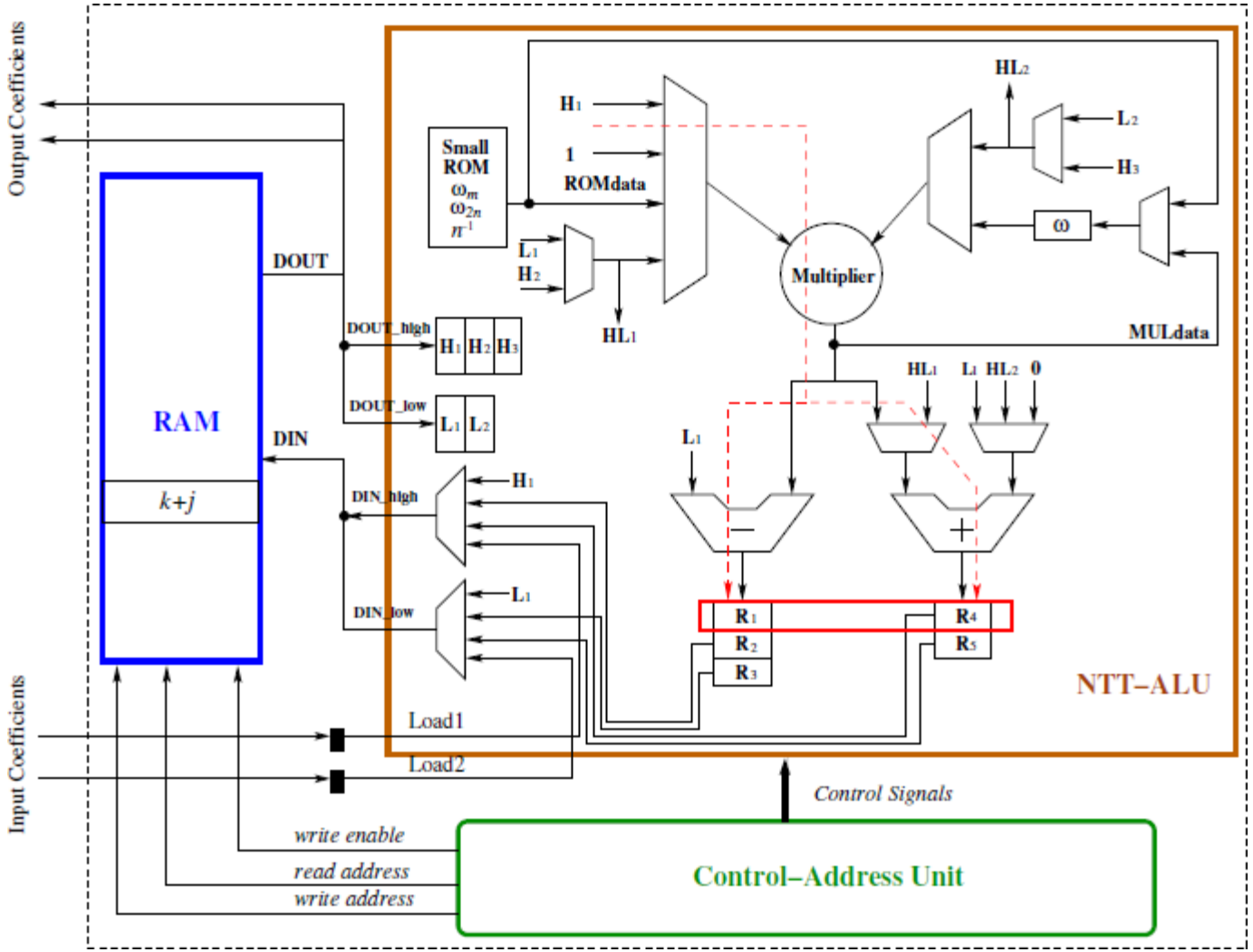
The NTT Cor

$$(H_1, L_1) \leftarrow (A[k+j+m/2], A[k+j])\ /*\ \text{From MEMORY[k+j]}\ */\ ;$$
$$(R_1, R_4) \leftarrow (L_1 - \omega \cdot H_1, L_1 + \omega \cdot H_1)\ ;$$
$$(R_2, R_5) \leftarrow (R_1, R_4)\ ;$$
$$(H_1, L_1) \leftarrow (A[k+j+3m/2], A[k+j+m])\ /*\ \text{From MEMORY[k+j+m/2]}\ */\ ;$$
$$(R_1, R_4) \leftarrow (L_1 - \omega \cdot H_1, L_1 + \omega \cdot H_1)\ ;$$
$$MEMORY\,[k+j] \leftarrow (R_4, R_5)\ ;$$
$$MEMORY\,[k+j+m/2] \leftarrow (R_2, R_3)\ ;$$



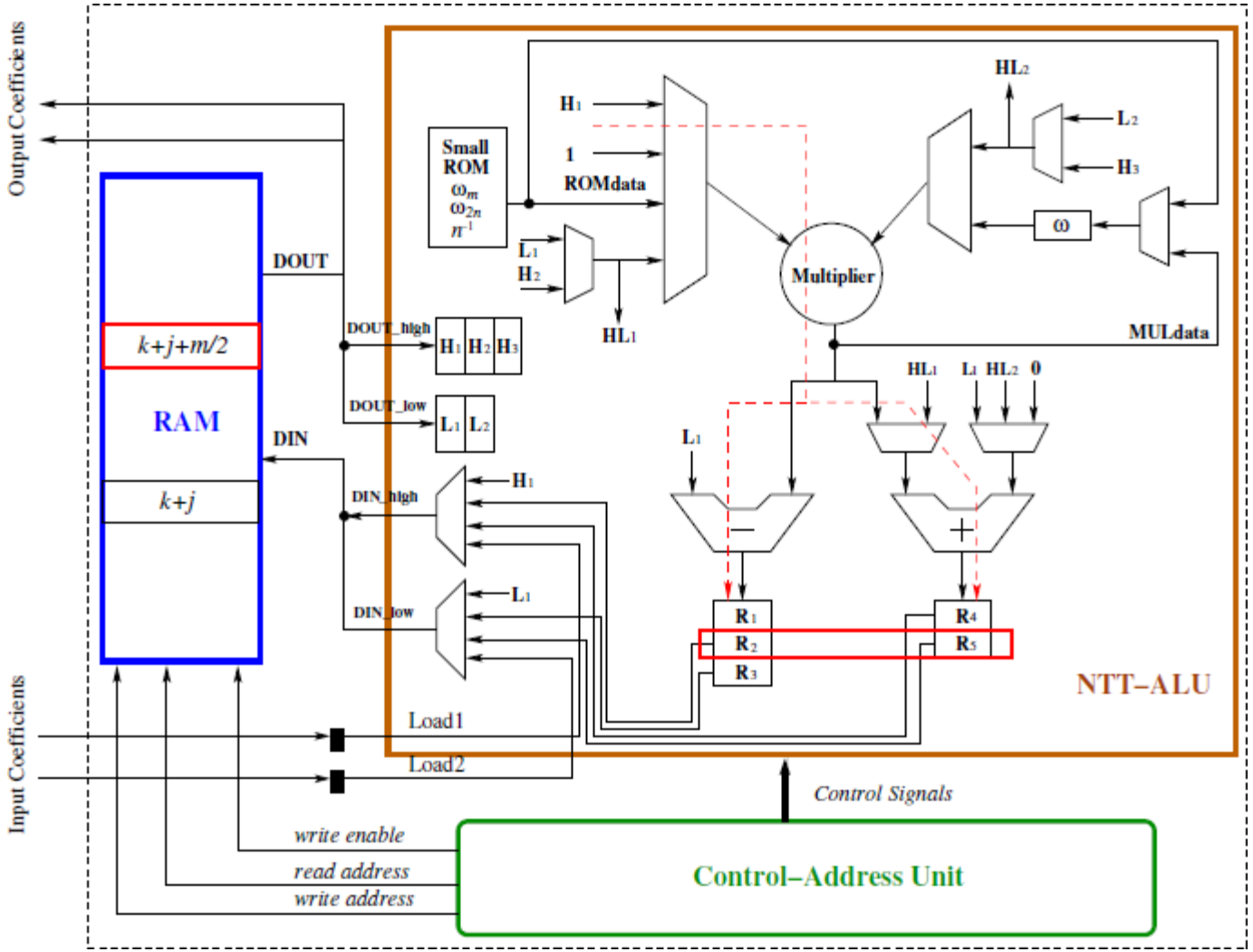
The NTT Cor

$$(H_1, L_1) \leftarrow (A[k+j+m/2], A[k+j]) \text{ /* From MEMORY[k+j] */ ;}$$
$$(R_1, R_4) \leftarrow (L_1 - \omega \cdot H_1, L_1 + \omega \cdot H_1) ;$$
$$(R_2, R_5) \leftarrow (R_1, R_4) ;$$
$$(H_1, L_1) \leftarrow (A[k+j+3m/2], A[k+j+m]) \text{ /* From MEMORY[k+j+m/2] */ ;}$$
$$(R_1, R_4) \leftarrow (L_1 - \omega \cdot H_1, L_1 + \omega \cdot H_1) ;$$
$$MEMORY[k+j] \leftarrow (R_4, R_5) ;$$
$$MEMORY[k+j+m/2] \leftarrow (R_2, R_3) ;$$



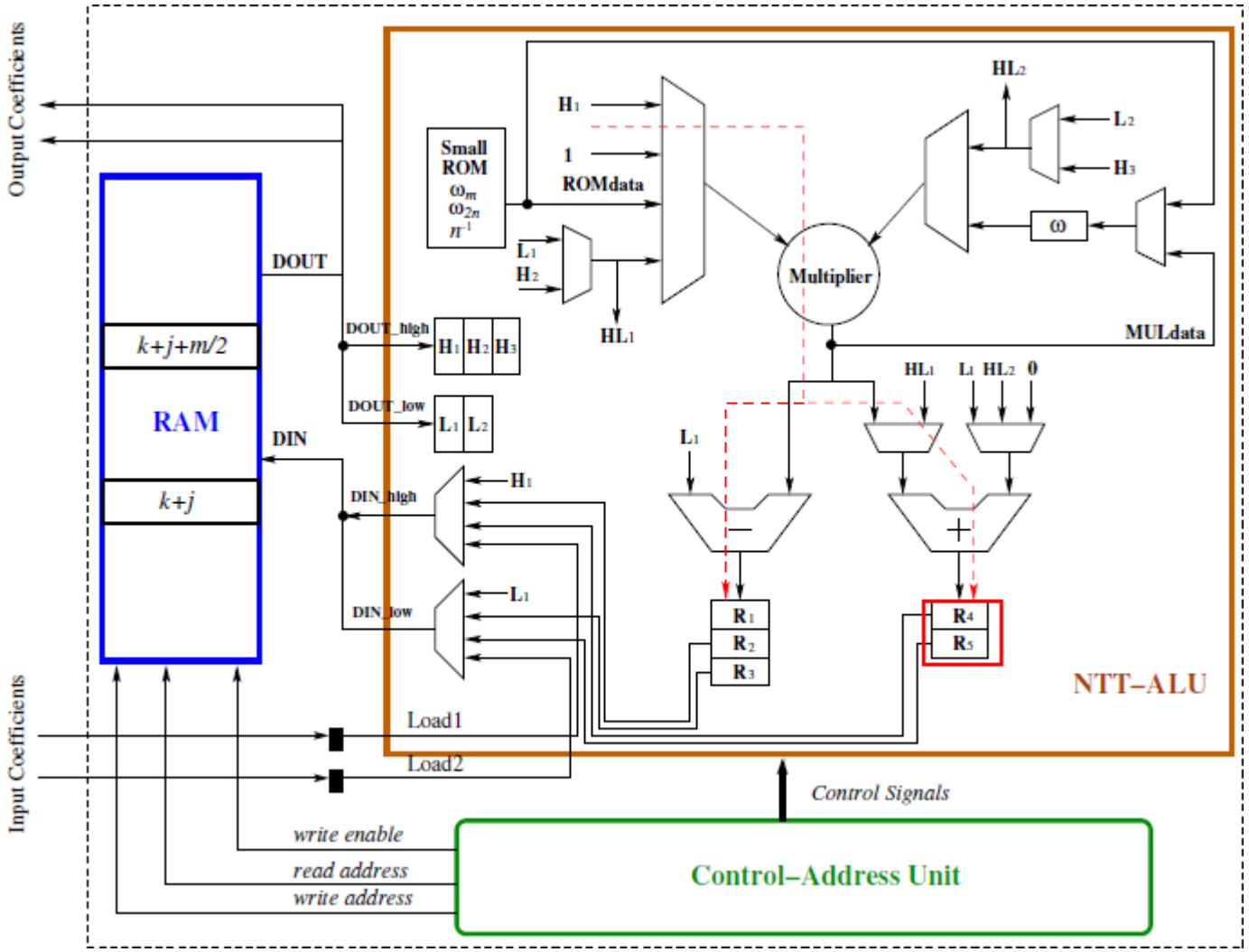
The NTT Cor

$$(H_1, L_1) \leftarrow (A[k+j+m/2], A[k+j]) \text{ /* From MEMORY[k+j] */ ;}$$
$$(R_1, R_4) \leftarrow (L_1 - \omega \cdot H_1, L_1 + \omega \cdot H_1) ;$$
$$(R_2, R_5) \leftarrow (R_1, R_4) ;$$
$$(H_1, L_1) \leftarrow (A[k+j+3m/2], A[k+j+m]) \text{ /* From MEMORY[k+j+m/2] */ ;}$$
$$(R_1, R_4) \leftarrow (L_1 - \omega \cdot H_1, L_1 + \omega \cdot H_1) ;$$
$$MEMORY[k+j] \leftarrow (R_4, R_5) ;$$
$$MEMORY[k+j+m/2] \leftarrow (R_2, R_3) ;$$



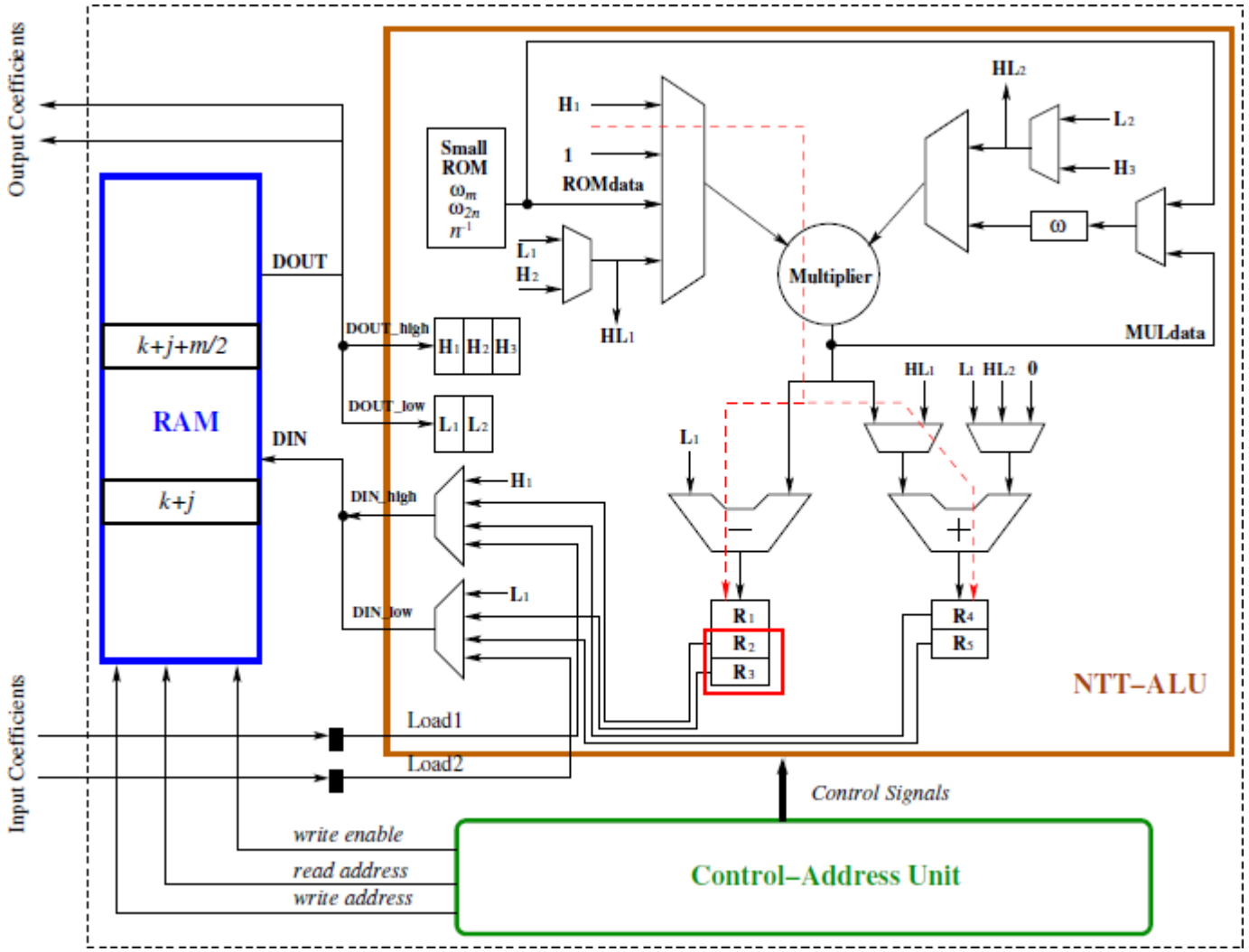
The NTT Cor

$$(H_1, L_1) \leftarrow (A[k+j+m/2], A[k+j]) \text{ /* From MEMORY[k+j] */ ;}$$
$$(R_1, R_4) \leftarrow (L_1 - \omega \cdot H_1, L_1 + \omega \cdot H_1) ;$$
$$(R_2, R_5) \leftarrow (R_1, R_4) ;$$
$$(H_1, L_1) \leftarrow (A[k+j+3m/2], A[k+j+m]) \text{ /* From MEMORY[k+j+m/2] */ ;}$$
$$(R_1, R_4) \leftarrow (L_1 - \omega \cdot H_1, L_1 + \omega \cdot H_1) ;$$
$$MEMORY[k+j] \leftarrow (R_4, R_5) ;$$
$$MEMORY[k+j+m/2] \leftarrow (R_2, R_3) ;$$

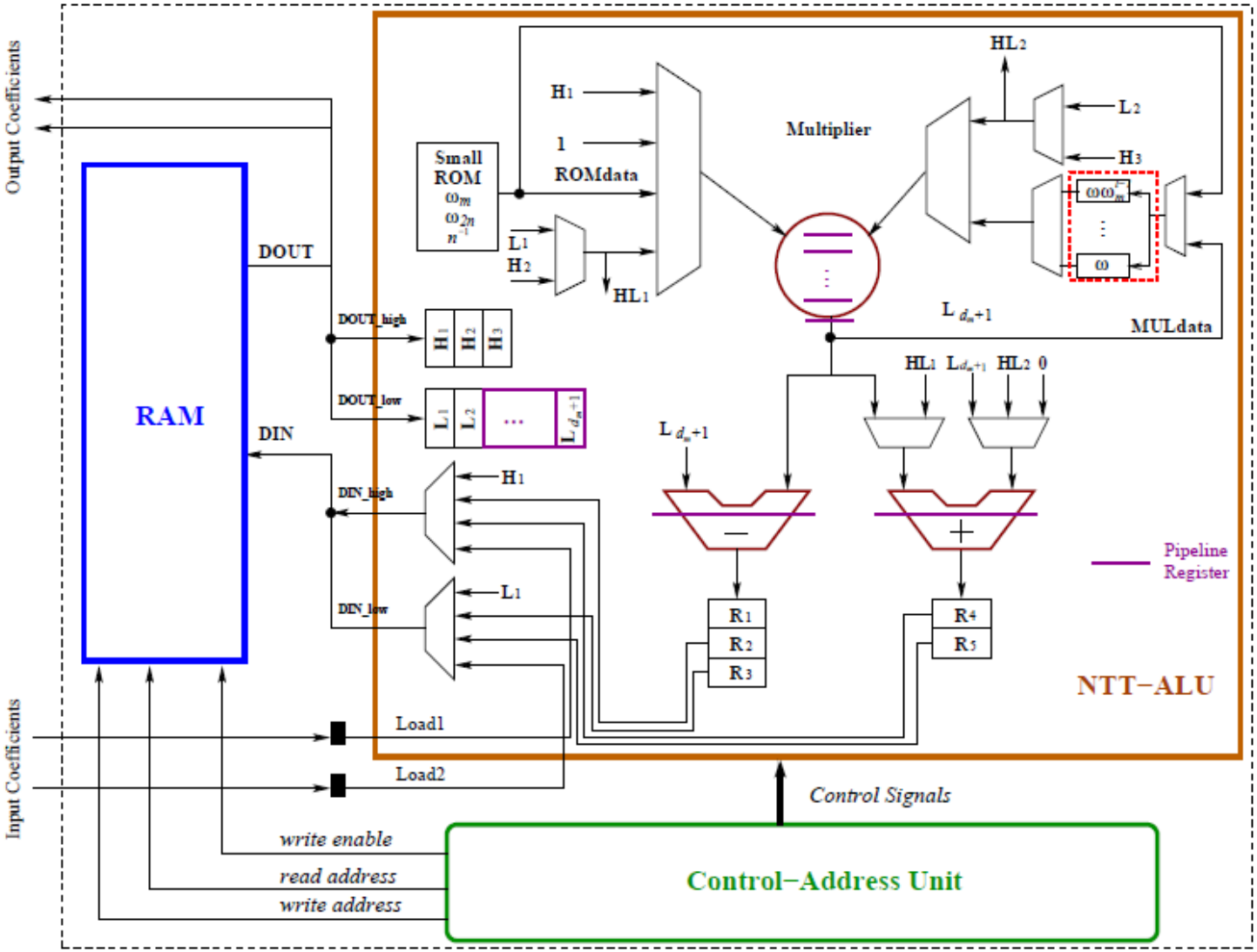


The NTT Cor

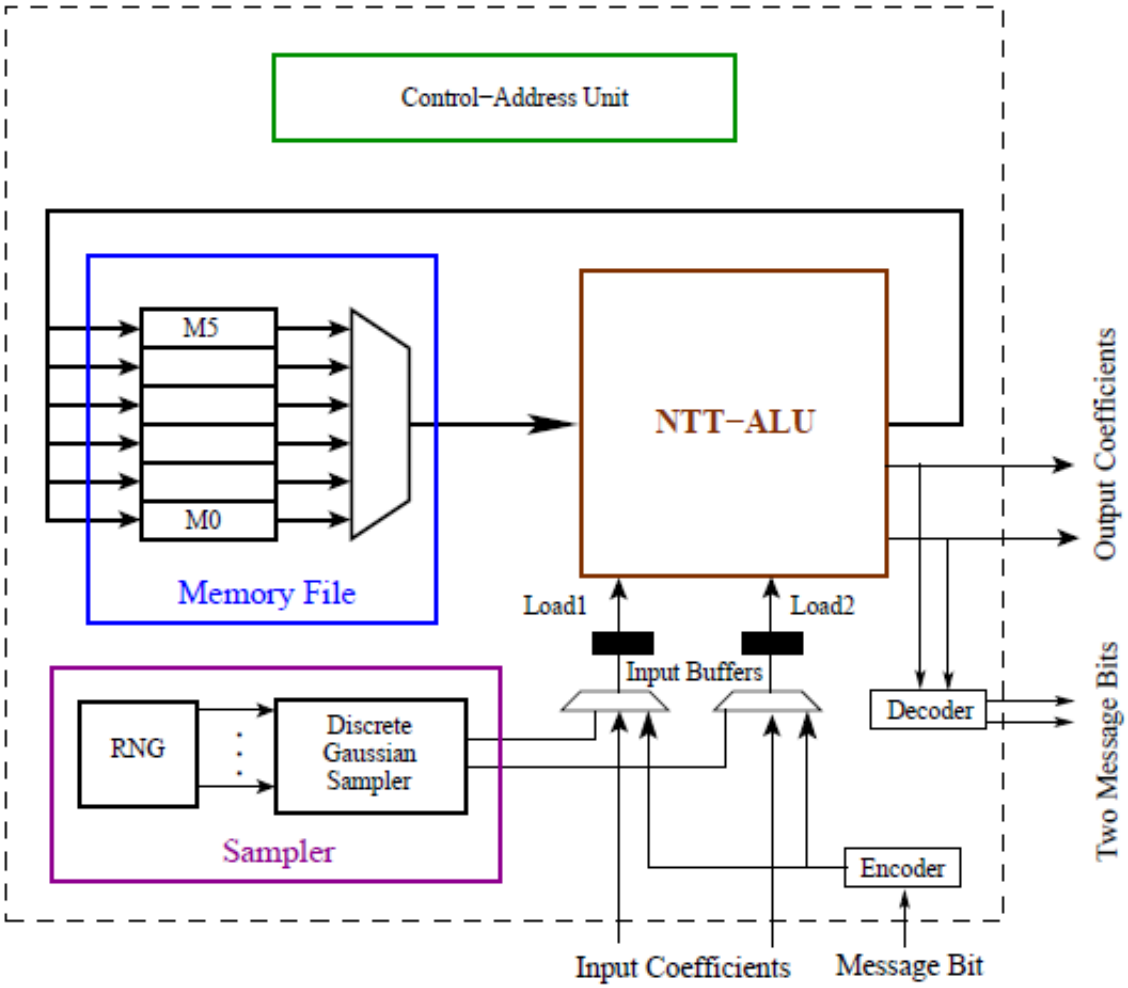
$$(H_1, L_1) \leftarrow (A[k+j+m/2], A[k+j]) \text{ /* From MEMORY[k+j] */ ;}$$
$$(R_1, R_4) \leftarrow (L_1 - \omega \cdot H_1, L_1 + \omega \cdot H_1) ;$$
$$(R_2, R_5) \leftarrow (R_1, R_4) ;$$
$$(H_1, L_1) \leftarrow (A[k+j+3m/2], A[k+j+m]) \text{ /* From MEMORY[k+j+m/2] */ ;}$$
$$(R_1, R_4) \leftarrow (L_1 - \omega \cdot H_1, L_1 + \omega \cdot H_1) ;$$
$$MEMORY[k+j] \leftarrow (R_4, R_5) ;$$
$$MEMORY[k+j+m/2] \leftarrow (R_2, R_3) ;$$



The NTT Core : Pipeline



The Ring-LWE Cryptoprocessor



Instruction Set

LOAD

ENCODE-LOAD

GAUSSIAN-LOAD

FNTT

INTT

ADD

CMULT

REARRANGE

READ

Results

Our Ring-LWE Cryptoprocessor : Results on Virtex 6

Parameters	Device	LUTs/FFs/ DSPs/BRAM18	Freq (MHz)	Cycles/Time(μs)	
				Encryption	Decryption
(256,7681,11.32)	V6LX75T	1349/860/1/2	313	6.3k/20.1	2.8k/9.1
(512,12289,12.18)		1536/953/1/3	278	13.3k/47.9	5.8k/21

Comparison with previous Ring-LWE Implementation

	Parameters	Device	LUTs/FFs/ DSPs/BRAM18	Freq (MHz)	Cycles/Time(μs) Encryption	Decryption
1	(256,7681,11.32)	V6LX75T	1349/860/1/2	313	6.3k/20.1	2.8k/9.1
	(512,12289,12.18)		1536/953/1/3	278	13.3k/47.9	5.8k/21
2	(256,7681,11.32)	V6LX75T	4549/3624/1/12	262	6.8k/26.2	4.4k/16.8
	(512,12289,12.18)	V6LX75T	5595/4760/1/14	251	13.7k/54.8	8.8k/35.4

1 RLWE, Our Implementation

2 RLWE, Pöppelmann et al. SAC 2013

Comparison with ECC (ECIES)

	Parameters	Device	LUTs/FFs/ DSPs/BRAM18	Freq (MHz)	Cycles/Time(μs) Encryption	Decryption
1	(256,7681,11.32)	V6LX75T	1349/860/1/2	313	6.3k/20.1	2.8k/9.1
2	Binary-233	V5LX85T	18097/-/5644/0	156	≈3.8k/24.6	≈1.9k/12.3

1 RLWE, Our Implementation

2 ECC, Rebeiro et al. CHES 2012

Conclusions and Future Work

Conclusions

- Hardware implementation of an instruction-set ring-LWE processor
  - Optimizations in the NTT
  - Architecture level accelerations
  - Best area-time performance

Future Work

- Lattice based signature scheme and fully-homomorphic encryption
  - Discrete Gaussian sampling
  - Larger polynomial multiplier

Thank You

Backup Slides

Comparison with NTRU

Implementation Algorithm	Parameters	Device	LUTs/FFs/ DSPs/BRAM18	Freq (MHz)	Cycles/Time(μs) Encryption	Decryption
RLWE	(256,7681,11.32)	V6LX75T	1349/860/1/2	313	6.3k/20.1	2.8k/9.1
	(512,12289,12.18)		1536/953/1/3	278	13.3k/47.9	5.8k/21
NTRU (ICM-09) Kamal et al.	NTRU-251	XCV1600E	27292/5160/14352/0	62.3	-/1.54	-/1.41

Comparison with McEliece

	Implementation Algorithm	Security	Device	LUTs/FFs/ DSPs/BRAM18	Freq (MHz)	Cycles/Time(μs) Encryption	Decryption
1	RLWE	≈128	Virtex 6	1349/860/1/2	313	6.3k/20.1	2.8k/9.1
2	McEliece	≈128	Virtex 6	≈ 20k/-/-/488	254	1.2k/4.7	233k/920

1 RLWE, Our Implementation

2 McEliece Cryptosystem by Ghosh et al. IEEE TC, 2014

All Comparisons

Implementation Algorithm	Parameters	Device	LUTs/FFs/ DSPs/BRAM18	Freq (MHz)	Cycles/Time(μs) Encryption	Decryption
RLWE	(256,7681,11.32)	V6LX75T	1349/860/1/2	313	6.3k/20.1	2.8k/9.1
	(512,12289,12.18)		1536/953/1/3	278	13.3k/47.9	5.8k/21
RLWE (*SAC-13*)	(256,7681,11.32)	V6LX75T	4549/3624/1/12	262	6.8k/26.2	4.4k/16.8
Pöppelmann et al.	(512,12289,12.18)	V6LX75T	5595/4760/1/14	251	13.7k/54.8	8.8k/35.4
RLWE (*ISCAS-14*)	(256,4096,8.35)	S6LX9	317/238/95/1	144	136k/946	-
Pöppelmann et al.			112/87/32/1	189	-	66k/351
ECC (CHES-12)	Binary-233	V5LX85T	18097/-/5644/0	156	≈3.8k/24.6	≈1.9k/12.3
Rebeiro et al.						
NTRU (ICM-09)	NTRU-251	XCV1600E	27292/5160/14352/0	62.3	-/1.54	-/1.41
Kamal et al.						