Application Acceleration with High-Level-Synthesis

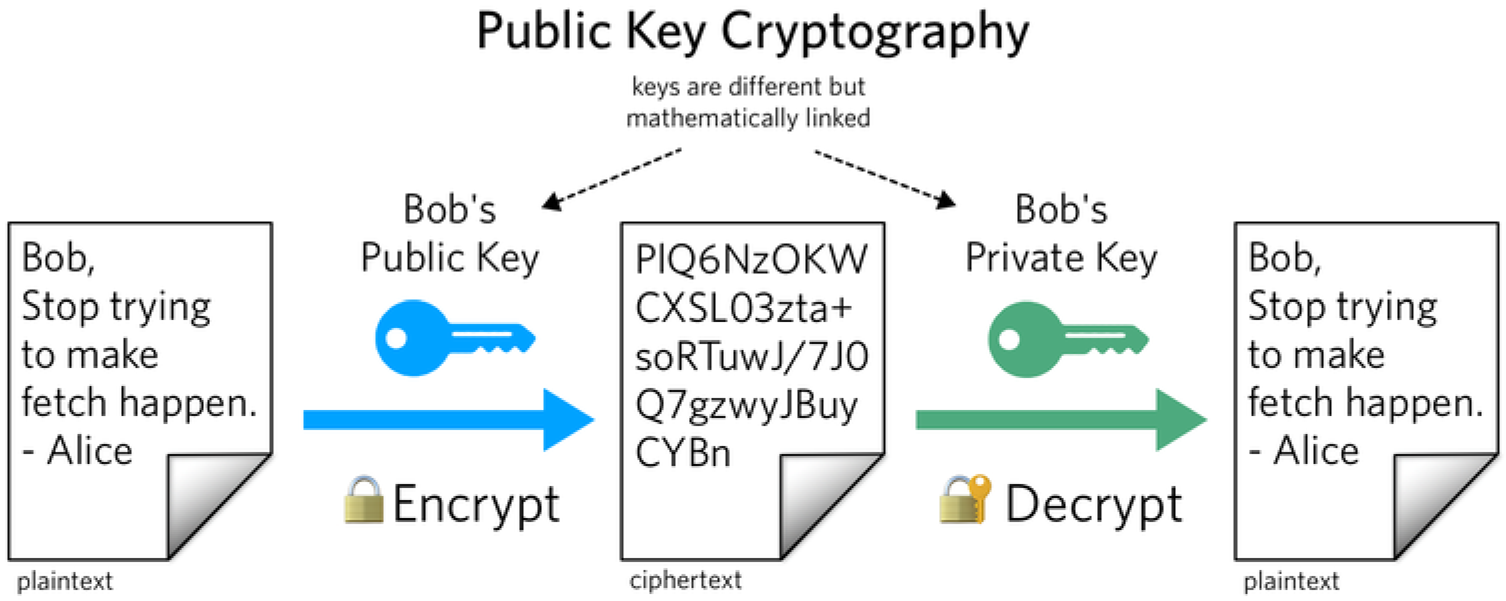
Dec. 29, 2022

# HLS Final Project - Dilithium

**Presenter: Liang-Hsin Lin, Yu-Cheng Lin**

*Graduate Institute of Electronics Engineering, National Taiwan University*

# Cryptography


Public Key Cryptography
keys are different but mathematically linked
Bob's Public Key
Bob's Private Key
Bob, Stop trying to make fetch happen. - Alice
plaintext
Encrypt
PIQ6NzOKW CXSL03zta+ soRTuwJ/7J0 Q7gzwyJBuy CYBn
ciphertext
Decrypt
Bob, Stop trying to make fetch happen. - Alice
plaintext

- **Encrypting with a public key**
  - Sending messages only the intended recipient can read
- **Signing with your private key**
  - Verifying that you're the one who sent a message

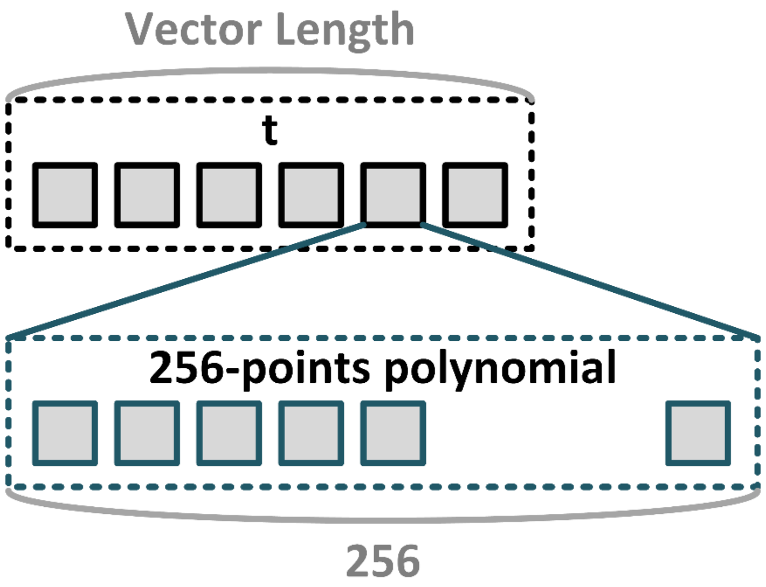
# Post-Quantum Cryptography

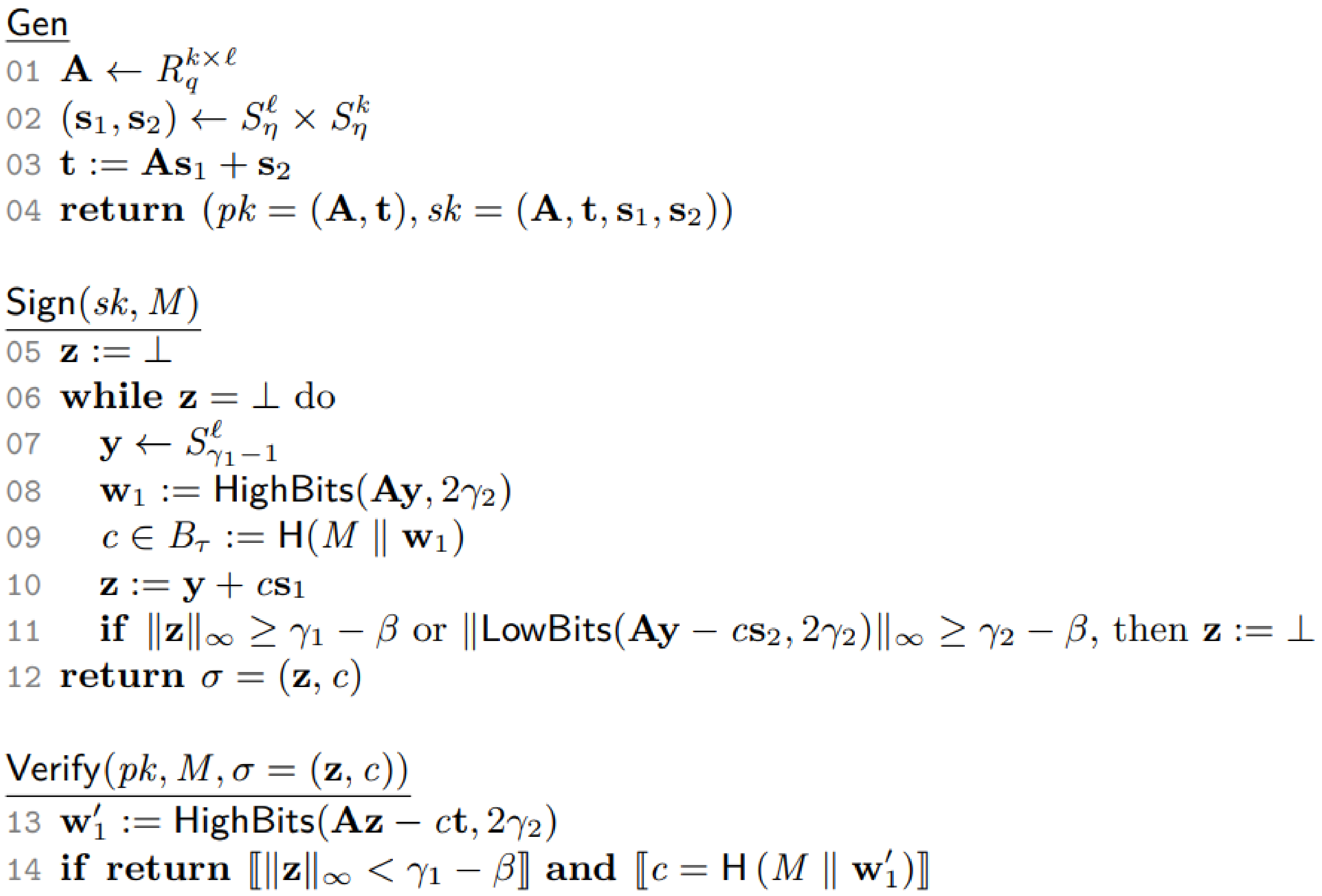
- Current popular algorithms could be easily solved on a sufficiently powerful quantum computer running Shor's algorithm
- Post-quantum cryptography is secure against a cryptanalytic attack by a quantum computer
- Computation complexity is high
  - Need efficient algorithm and dedicated hardware codesign
- Hardware acceleration design reference in the past few years
  - ISSCC’22, VLSI’22, ISSCC’23

➔ **Fast architecture search using HLS**

# PQC Algorithm: Dilithium

- Basic component:
  - Polynomial ring
  - Degree of 256



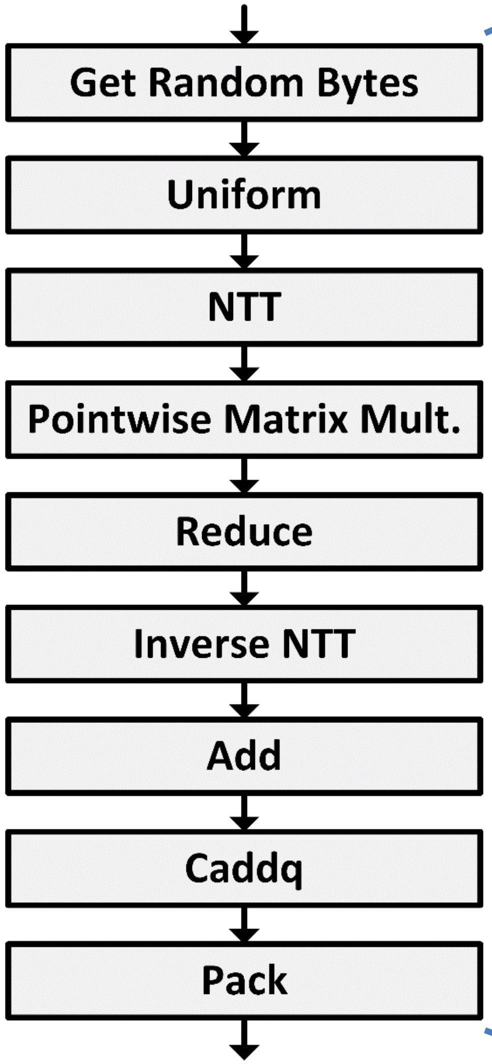


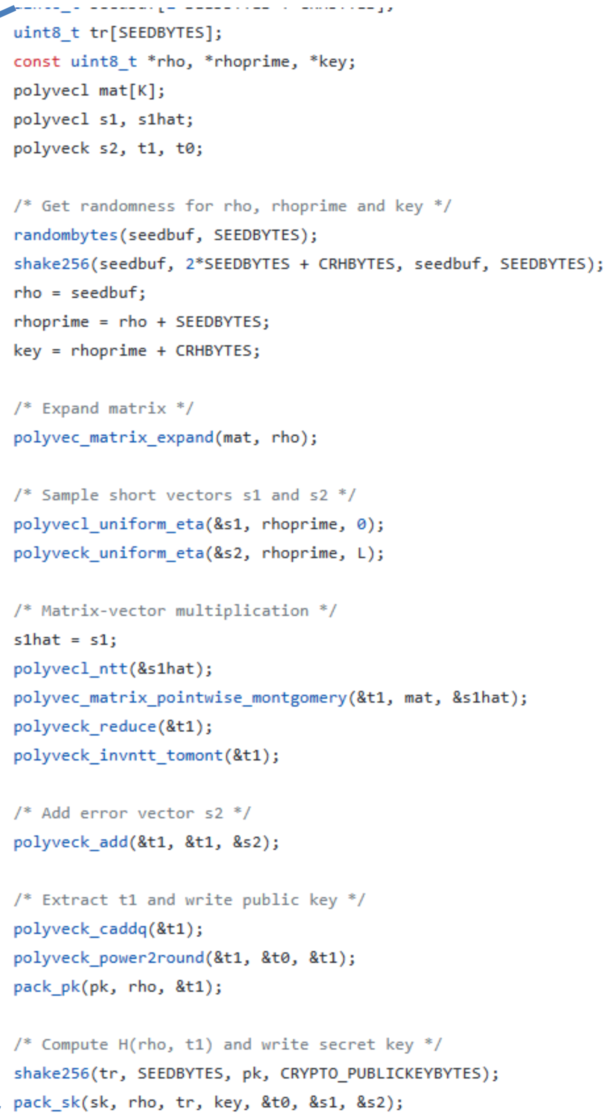
```
Gen
$\mathbf{A} \leftarrow R_q^{k \times \ell}$
$(\mathbf{s}_1, \mathbf{s}_2) \leftarrow S_\eta^\ell \times S_\eta^k$
$\mathbf{t} := \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$
$\textbf{return } (pk = (\mathbf{A}, \mathbf{t}), sk = (\mathbf{A}, \mathbf{t}, \mathbf{s}_1, \mathbf{s}_2))$

Sign$(sk, M)$
$\mathbf{z} := \perp$
$\textbf{while } \mathbf{z} = \perp \text{ do}$
    $\mathbf{y} \leftarrow S_{\gamma_1 - 1}^\ell$
    $\mathbf{w}_1 := \mathsf{HighBits}(\mathbf{A}\mathbf{y}, 2\gamma_2)$
    $c \in B_\tau := \mathsf{H}(M \parallel \mathbf{w}_1)$
    $\mathbf{z} := \mathbf{y} + c\mathbf{s}_1$
    $\textbf{if } \|\mathbf{z}\|_\infty \geq \gamma_1 - \beta \text{ or } \|\mathsf{LowBits}(\mathbf{A}\mathbf{y} - c\mathbf{s}_2, 2\gamma_2)\|_\infty \geq \gamma_2 - \beta, \text{ then } \mathbf{z} := \perp$
$\textbf{return } \sigma = (\mathbf{z}, c)$

Verify$(pk, M, \sigma = (\mathbf{z}, c))$
$\mathbf{w}_1' := \mathsf{HighBits}(\mathbf{A}\mathbf{z} - c\mathbf{t}, 2\gamma_2)$
$\textbf{if return } [\![\|\mathbf{z}\|_\infty < \gamma_1 - \beta]\!] \textbf{ and } [\![c = \mathsf{H}(M \parallel \mathbf{w}_1')]\!]$
```

# Algorithm Workflow

Key generation for example

- Uniform Sampling
- Element-wise Operations
  - NTT/Inverse NTT
  - Addition/Subtraction
  - Reduce
  - Caddq
- Pack



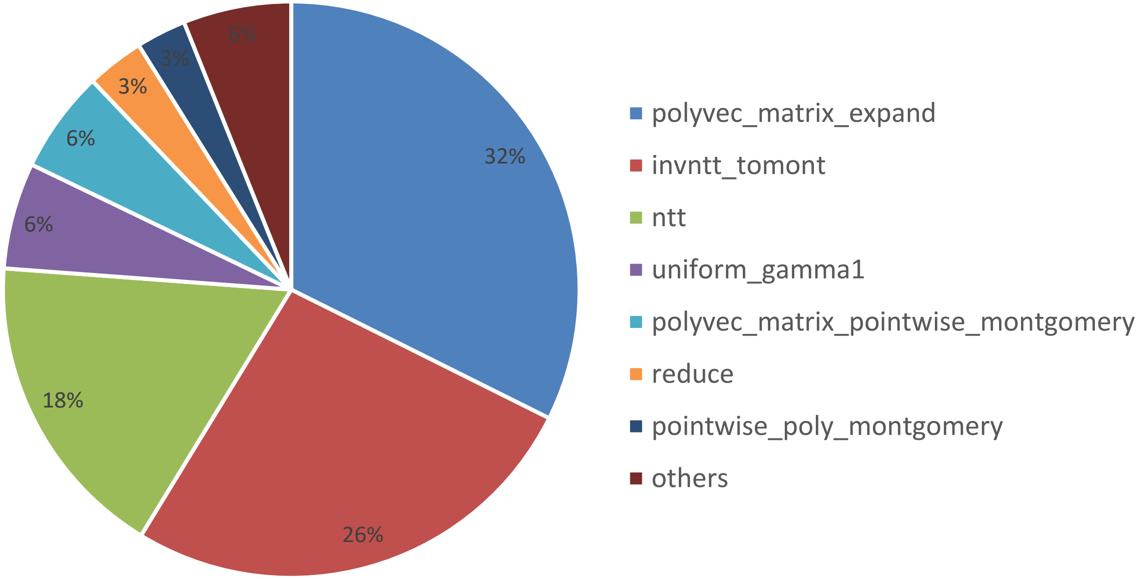


```
uint8_t tr[SEEDBYTES];
const uint8_t *rho, *rhoprime, *key;
polyvecl mat[K];
polyvecl s1, s1hat;
polyveck s2, t1, t0;

/* Get randomness for rho, rhoprime and key */
randombytes(seedbuf, SEEDBYTES);
shake256(seedbuf, 2*SEEDBYTES + CRHBYTES, seedbuf, SEEDBYTES);
rho = seedbuf;
rhoprime = rho + SEEDBYTES;
key = rhoprime + CRHBYTES;

/* Expand matrix */
polyvec_matrix_expand(mat, rho);

/* Sample short vectors s1 and s2 */
polyvecl_uniform_eta(&s1, rhoprime, 0);
polyveck_uniform_eta(&s2, rhoprime, L);

/* Matrix-vector multiplication */
s1hat = s1;
polyvecl_ntt(&s1hat);
polyvec_matrix_pointwise_montgomery(&t1, mat, &s1hat);
polyveck_reduce(&t1);
polyveck_invntt_tomont(&t1);

/* Add error vector s2 */
polyveck_add(&t1, &t1, &s2);

/* Extract t1 and write public key */
polyveck_caddq(&t1);
polyveck_power2round(&t1, &t0, &t1);
pack_pk(pk, rho, &t1);

/* Compute H(rho, t1) and write secret key */
shake256(tr, SEEDBYTES, pk, CRYPTO_PUBLICKEYBYTES);
pack_sk(sk, rho, tr, key, &t0, &s1, &s2);
```

# Algorithm Analysis – CPU Profiling

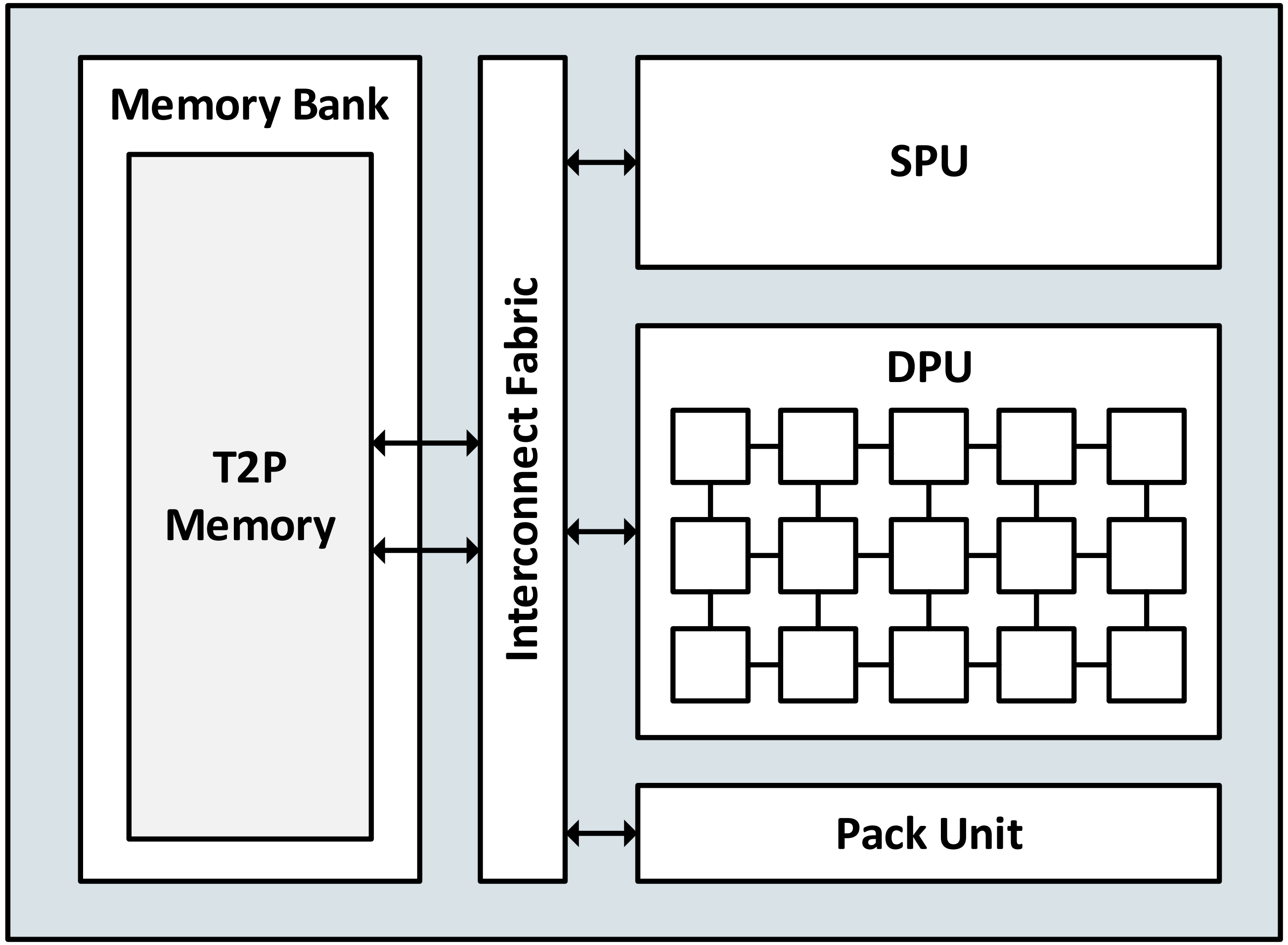
- Run on Intel Core i7-9700
- Computation bottleneck:
  - Uniform sampling
  - Matrix multiplication
  - Number theoretic transform (NTT)
  - Inverse NTT

➔ **accounts for 82% execution time**

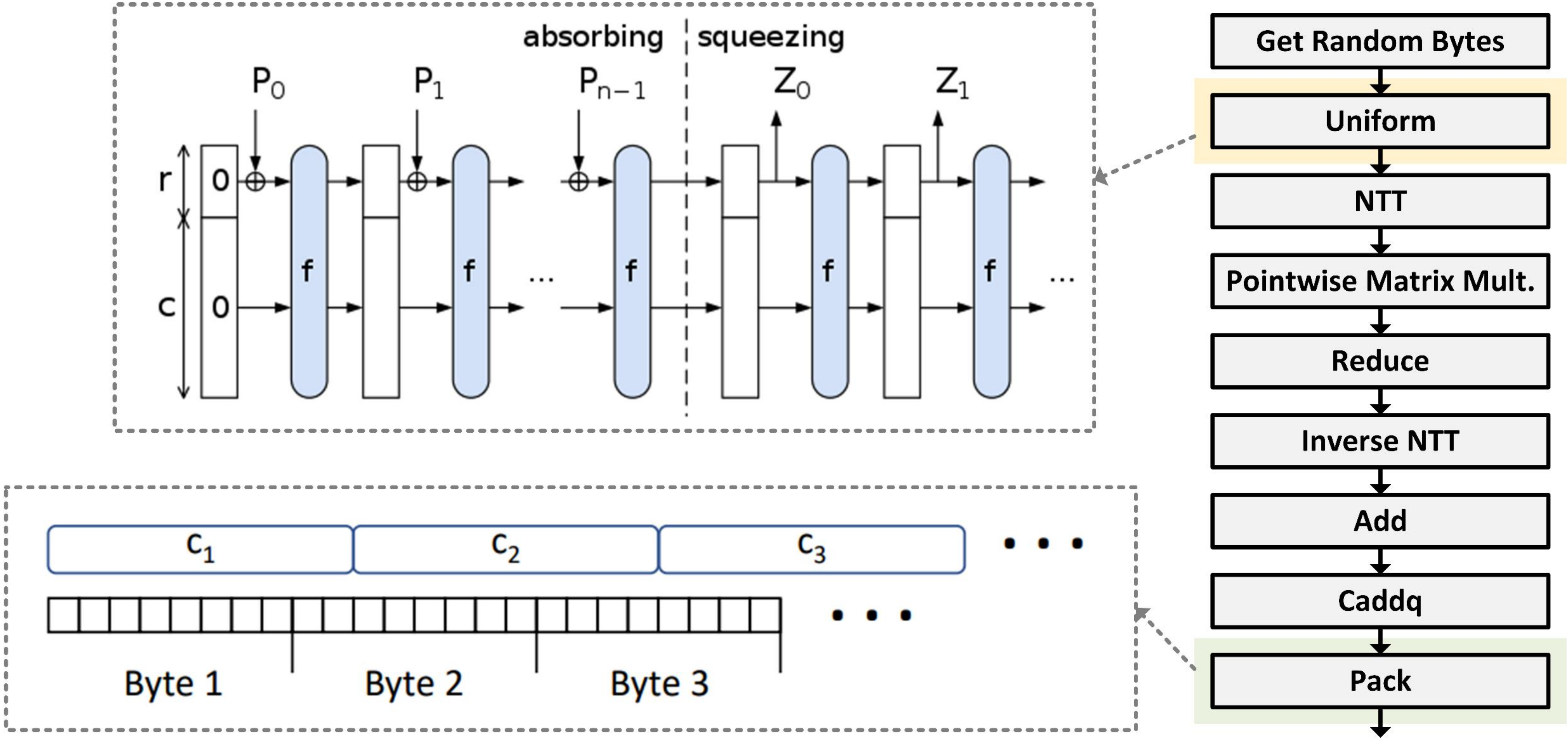


# Proposed Architecture

- Fully-Integrated architecture supporting operations for Dilithium



# SPU and Pack Unit



Get Random Bytes

Uniform

NTT

Pointwise Matrix Mult.

Reduce

Inverse NTT

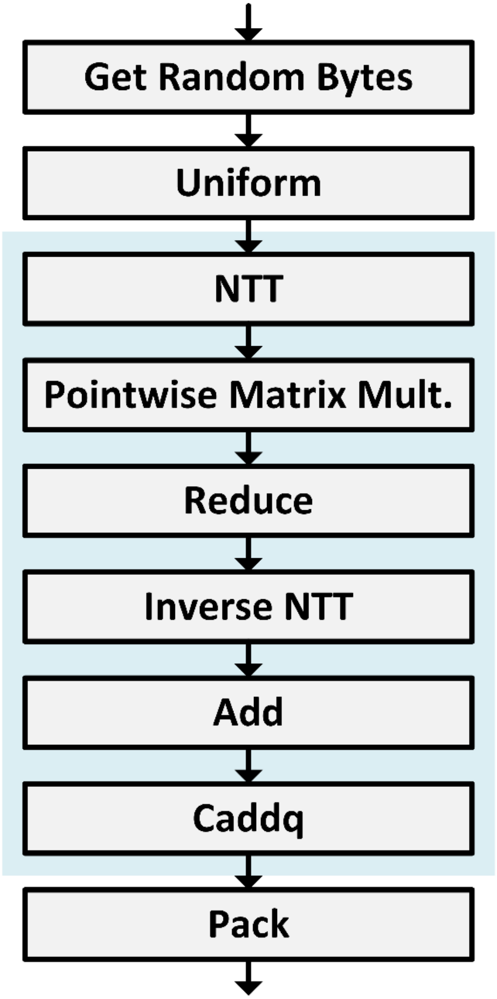
Add

Caddq

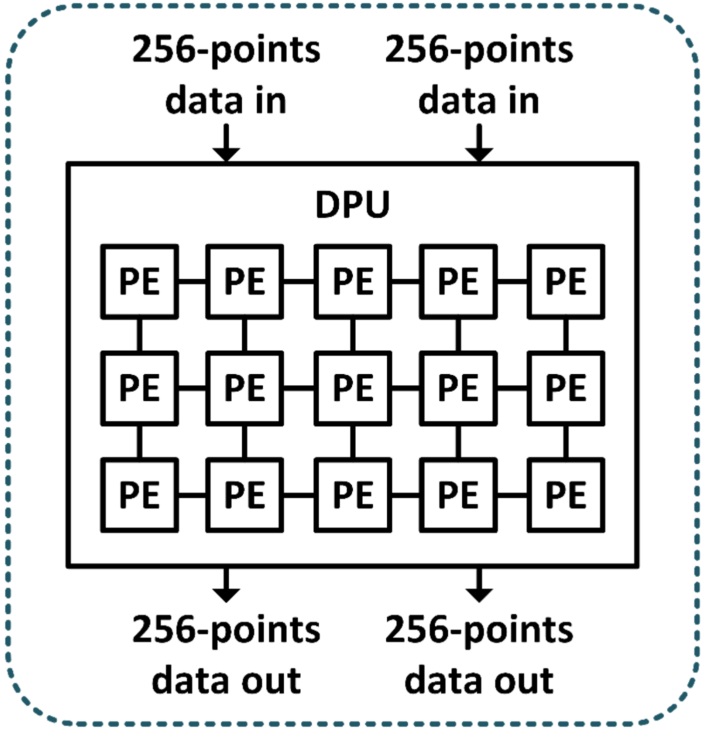
Pack

# DPU

- Support operations for element-wise data computation
- 256-points data is computed in a parallel manner
- The DPU is composed of 256 PEs
- Each processing element (PE) supports different operation
- Hardware resource sharing across similar operations



Element-wise Computing Scheme

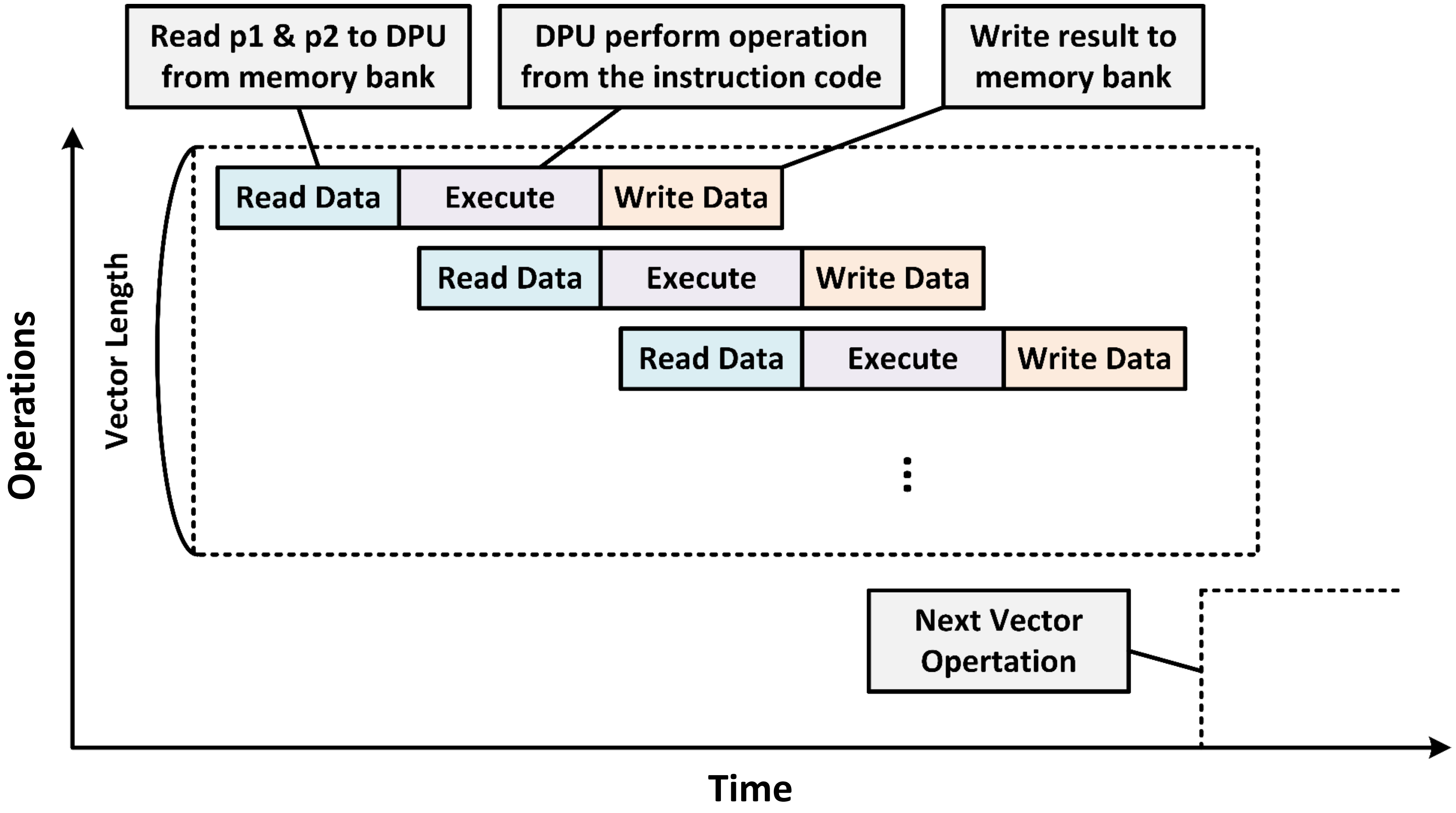


# Supported PE Operations

- Addition
- Subtraction
- Multiplication
- CADDQ: Add Q if the data is smaller than 0
- Reduce: Reduce the range of data to 0~Q
- Pass
- Power2Round: Decompose a data into MSBs and LSBs
- NTT
- Inverse NTT

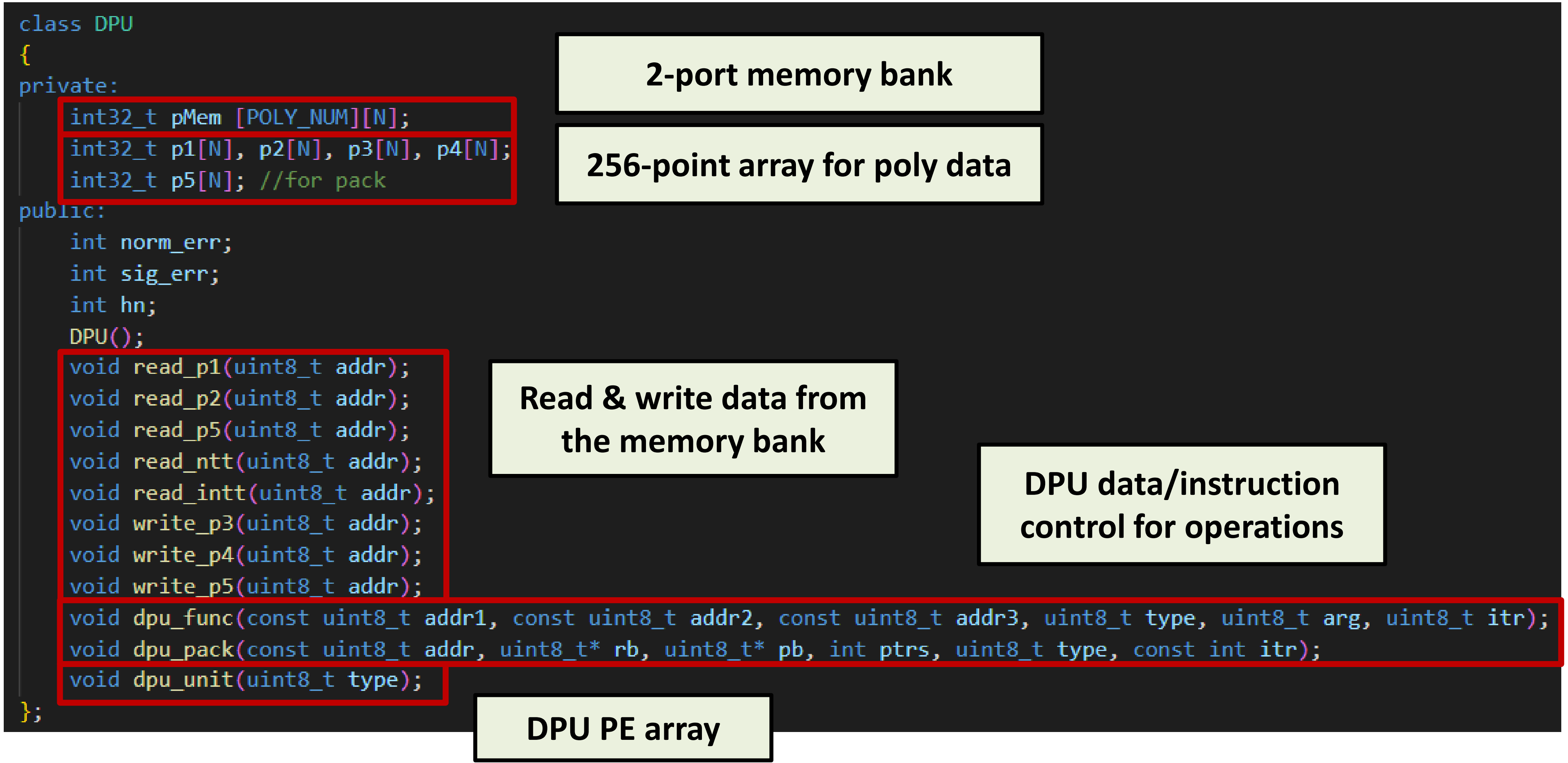
# DPU Operation Flow

- Pipelined operation for DPU unit
  - Fully utilize all 256 PEs in DPU



# Code Implementation – DPU unit

## Code Structure for DPU Class



```
class DPU
{
private:
    int32_t pMem [POLY_NUM][N];
    int32_t p1[N], p2[N], p3[N], p4[N];
    int32_t p5[N]; //for pack
public:
    int norm_err;
    int sig_err;
    int hn;
    DPU();
    void read_p1(uint8_t addr);
    void read_p2(uint8_t addr);
    void read_p5(uint8_t addr);
    void read_ntt(uint8_t addr);
    void read_intt(uint8_t addr);
    void write_p3(uint8_t addr);
    void write_p4(uint8_t addr);
    void write_p5(uint8_t addr);
    void dpu_func(const uint8_t addr1, const uint8_t addr2, const uint8_t addr3, uint8_t type, uint8_t arg, uint8_t itr);
    void dpu_pack(const uint8_t addr, uint8_t* rb, uint8_t* pb, int ptrs, uint8_t type, const int itr);
    void dpu_unit(uint8_t type);
};
```

2-port memory bank

256-point array for poly data

Read & write data from the memory bank

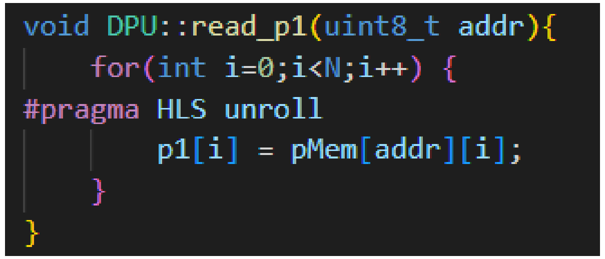
DPU data/instruction control for operations

DPU PE array

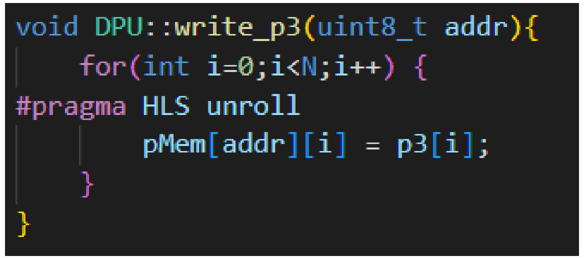
# Code Implementation – DPU unit

## Code Structure for DPU Modules

- Unroll loops to facilitate parallel processing

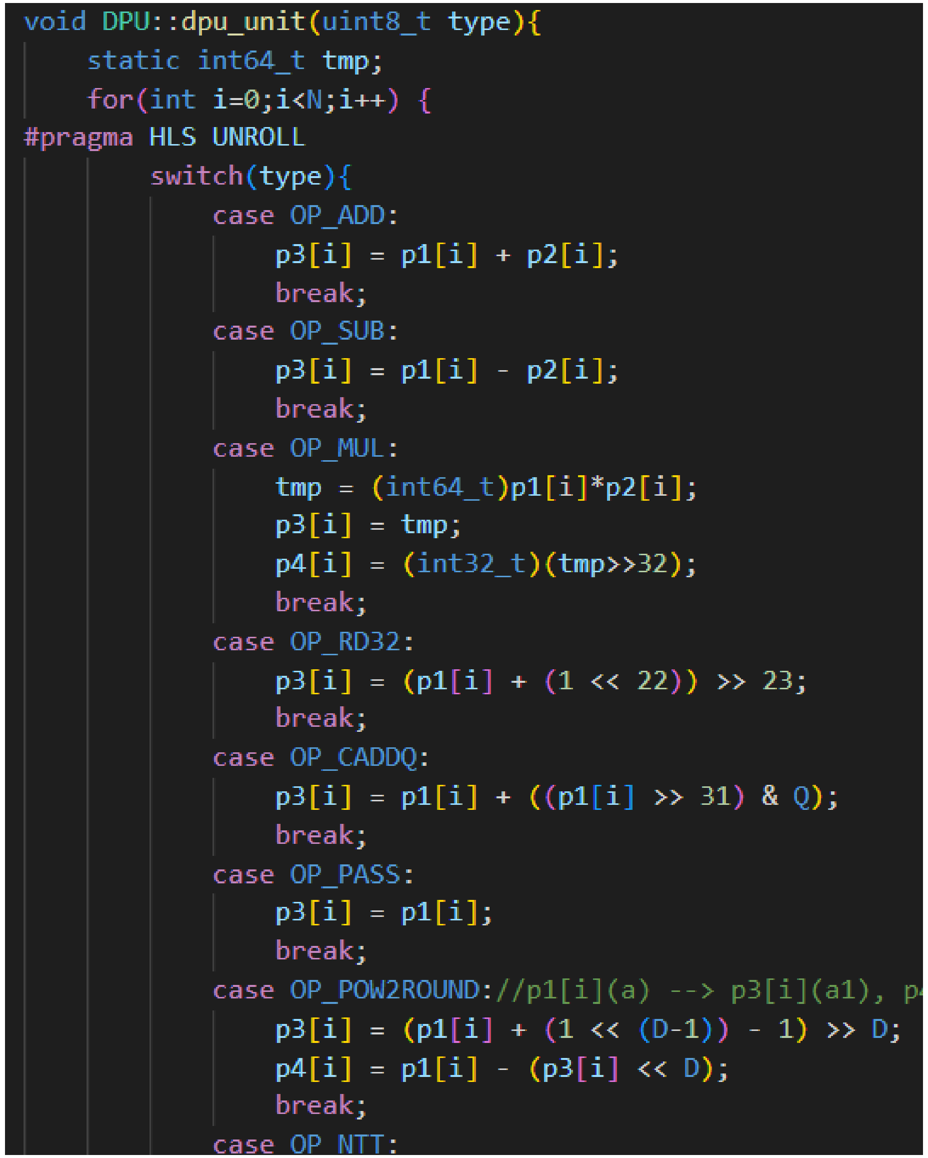


```
void DPU::read_p1(uint8_t addr){
    for(int i=0;i<N;i++) {
#pragma HLS unroll
        p1[i] = pMem[addr][i];
    }
}
```



```
void DPU::write_p3(uint8_t addr){
    for(int i=0;i<N;i++) {
#pragma HLS unroll
        pMem[addr][i] = p3[i];
    }
}
```

- Integrate all micro-operations in a single function
- DPU read two inputs and generate two outputs
  - p1 and p2 in
  - p3 and p4 out



```
void DPU::dpu_unit(uint8_t type){
    static int64_t tmp;
    for(int i=0;i<N;i++) {
#pragma HLS UNROLL
        switch(type){
            case OP_ADD:
                p3[i] = p1[i] + p2[i];
                break;
            case OP_SUB:
                p3[i] = p1[i] - p2[i];
                break;
            case OP_MUL:
                tmp = (int64_t)p1[i]*p2[i];
                p3[i] = tmp;
                p4[i] = (int32_t)(tmp>>32);
                break;
            case OP_RD32:
                p3[i] = (p1[i] + (1 << 22)) >> 23;
                break;
            case OP_CADDQ:
                p3[i] = p1[i] + ((p1[i] >> 31) & Q);
                break;
            case OP_PASS:
                p3[i] = p1[i];
                break;
            case OP_POW2ROUND://p1[i](a) --> p3[i](a1), p
                p3[i] = (p1[i] + (1 << (D-1)) - 1) >> D;
                p4[i] = p1[i] - (p3[i] << D);
                break;
            case OP_NTT:
```

# Code Implementation – DPU unit

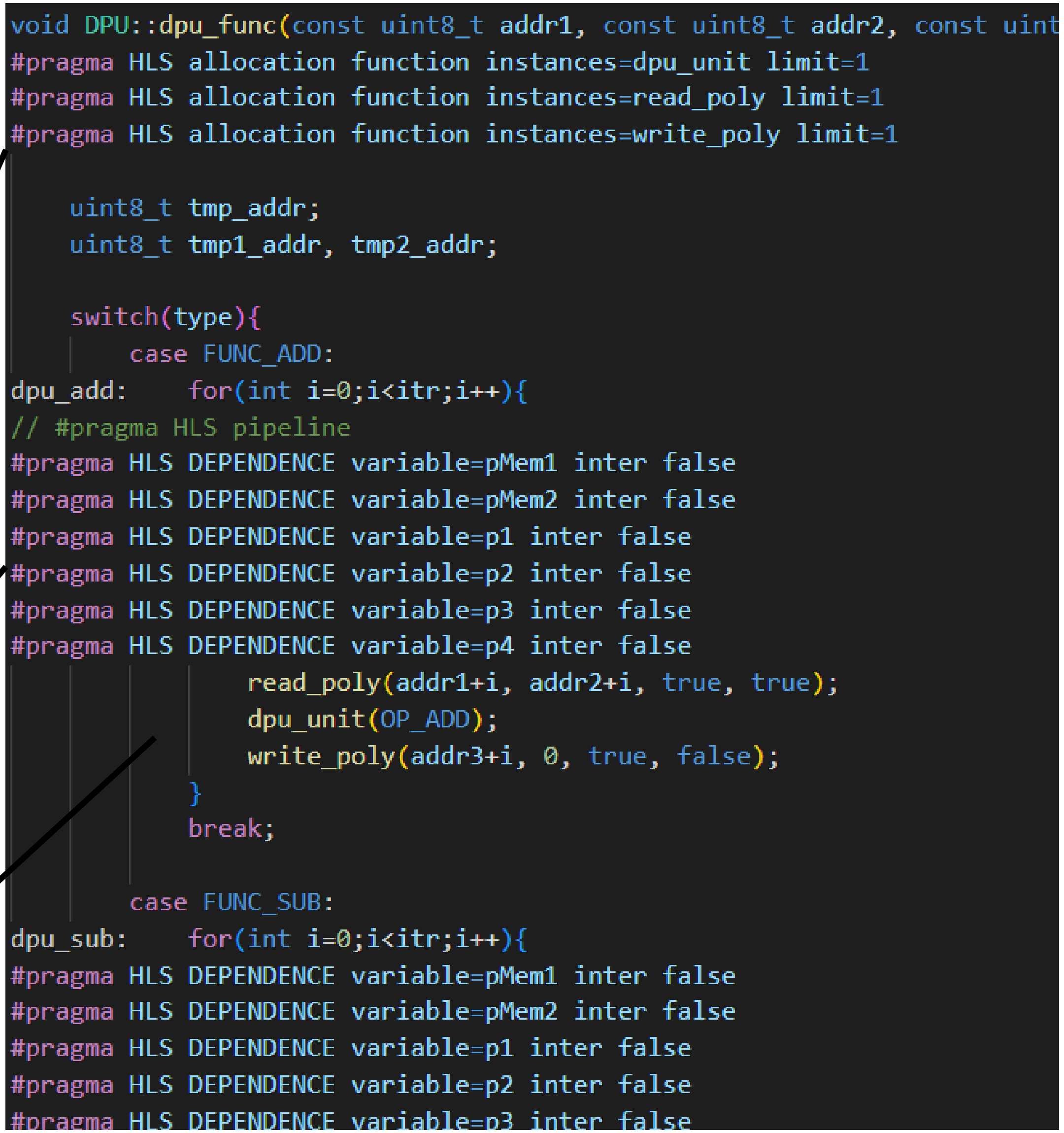
## Code Structure for DPU Operations

- For each function, an array of poly can be processed independently
- No data dependency between each array element

**Prevent the modules from over-duplicating**

**Set false dependency for loop pipelining**

**3 operations (Read, Operation, Write) for each for loop**

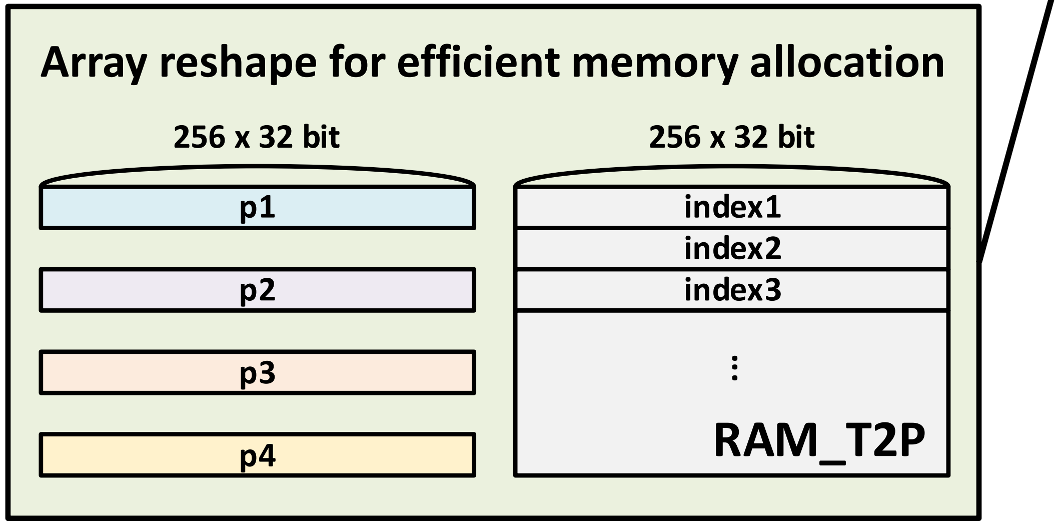


```
void DPU::dpu_func(const uint8_t addr1, const uint8_t addr2, const uint
#pragma HLS allocation function instances=dpu_unit limit=1
#pragma HLS allocation function instances=read_poly limit=1
#pragma HLS allocation function instances=write_poly limit=1

    uint8_t tmp_addr;
    uint8_t tmp1_addr, tmp2_addr;

    switch(type){
        case FUNC_ADD:
dpu_add:    for(int i=0;i<itr;i++){
// #pragma HLS pipeline
#pragma HLS DEPENDENCE variable=pMem1 inter false
#pragma HLS DEPENDENCE variable=pMem2 inter false
#pragma HLS DEPENDENCE variable=p1 inter false
#pragma HLS DEPENDENCE variable=p2 inter false
#pragma HLS DEPENDENCE variable=p3 inter false
#pragma HLS DEPENDENCE variable=p4 inter false
                read_poly(addr1+i, addr2+i, true, true);
                dpu_unit(OP_ADD);
                write_poly(addr3+i, 0, true, false);
            }
            break;

        case FUNC_SUB:
dpu_sub:    for(int i=0;i<itr;i++){
#pragma HLS DEPENDENCE variable=pMem1 inter false
#pragma HLS DEPENDENCE variable=pMem2 inter false
#pragma HLS DEPENDENCE variable=p1 inter false
#pragma HLS DEPENDENCE variable=p2 inter false
#pragma HLS DEPENDENCE variable=p3 inter false
```

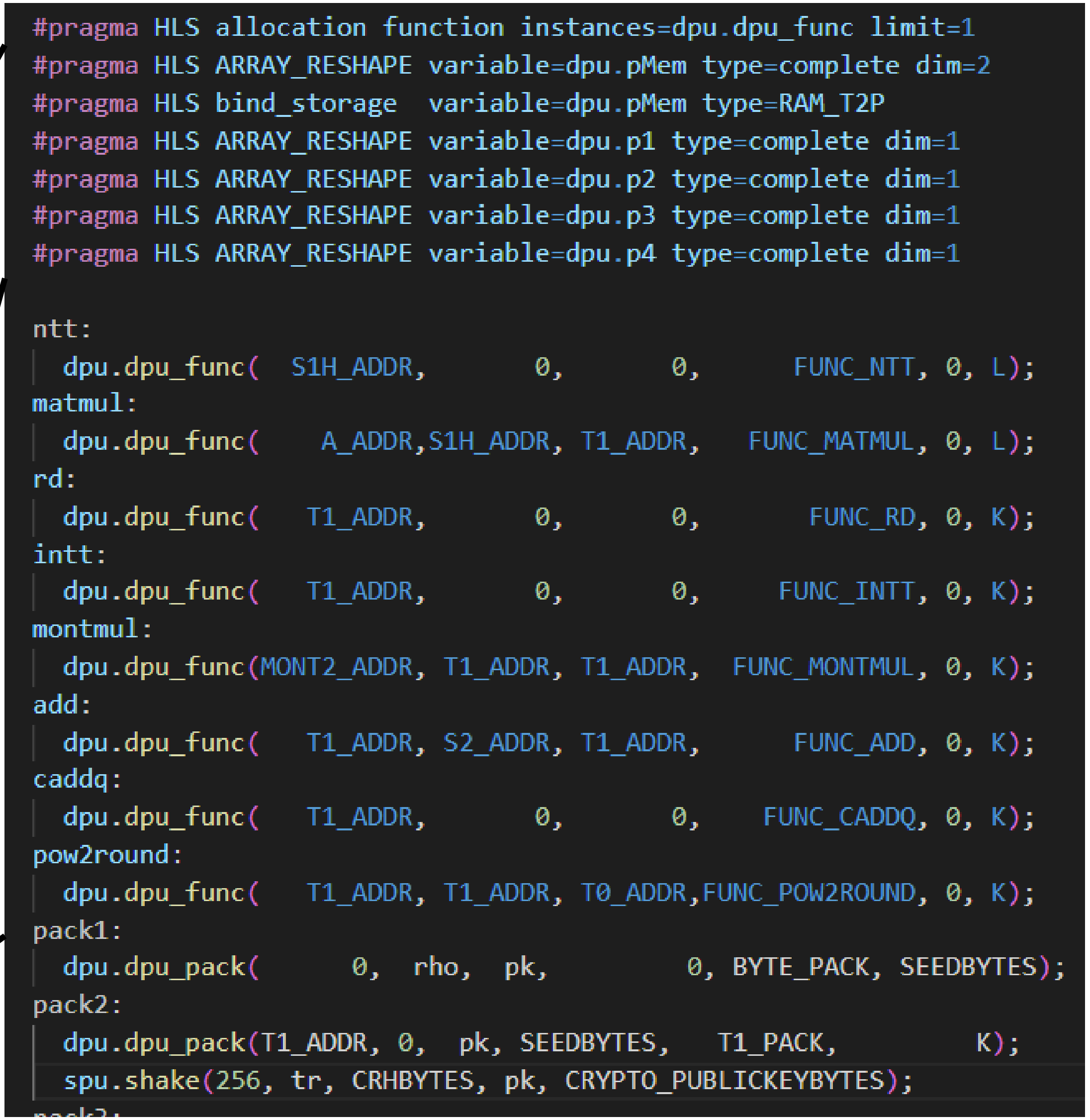
# Code Implementation – DPU unit

## Code Structure for Top Level Function

Limiting the number of DPU modules (to 1)

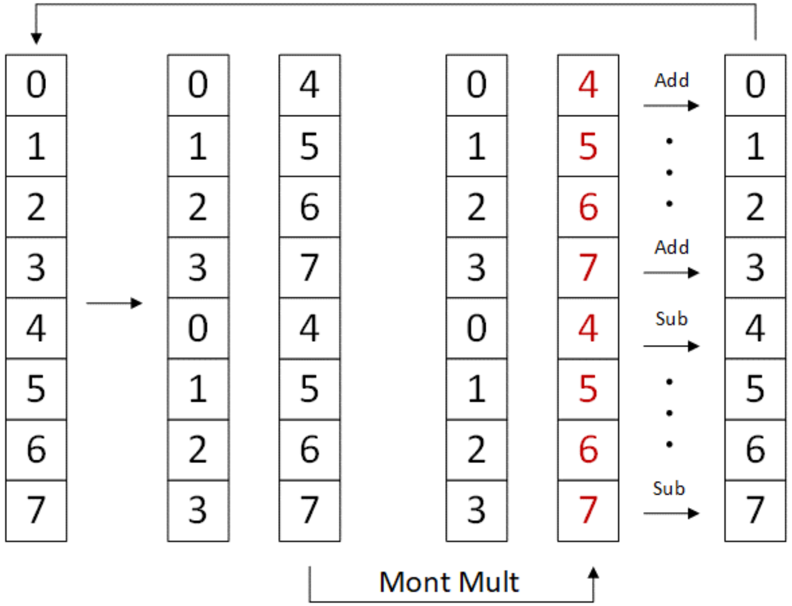

Array reshape for efficient memory allocation

Call DPU function for each operations (Instruction code pass-in)

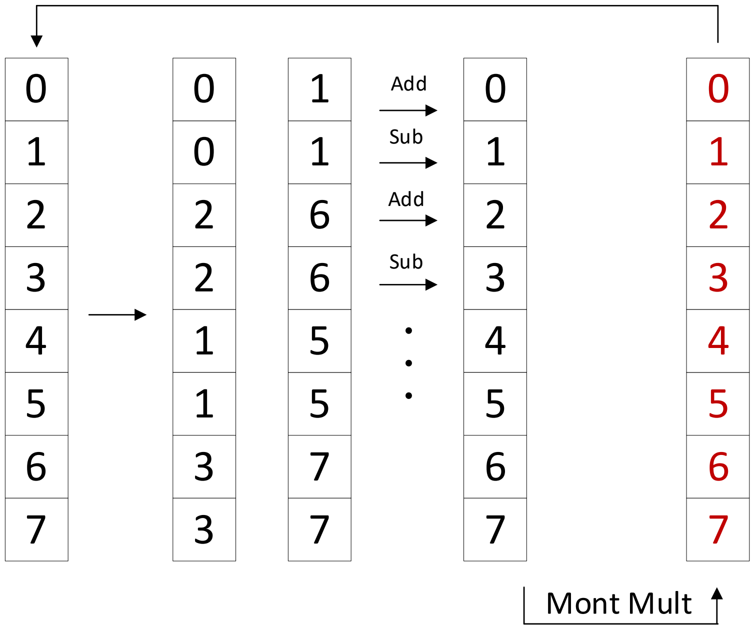

```
#pragma HLS allocation function instances=dpu.dpu_func limit=1
#pragma HLS ARRAY_RESHAPE variable=dpu.pMem type=complete dim=2
#pragma HLS bind_storage  variable=dpu.pMem type=RAM_T2P
#pragma HLS ARRAY_RESHAPE variable=dpu.p1 type=complete dim=1
#pragma HLS ARRAY_RESHAPE variable=dpu.p2 type=complete dim=1
#pragma HLS ARRAY_RESHAPE variable=dpu.p3 type=complete dim=1
#pragma HLS ARRAY_RESHAPE variable=dpu.p4 type=complete dim=1

ntt:
  dpu.dpu_func(  S1H_ADDR,       0,       0,      FUNC_NTT, 0, L);
matmul:
  dpu.dpu_func(    A_ADDR,S1H_ADDR, T1_ADDR,   FUNC_MATMUL, 0, L);
rd:
  dpu.dpu_func(   T1_ADDR,       0,       0,       FUNC_RD, 0, K);
intt:
  dpu.dpu_func(   T1_ADDR,       0,       0,     FUNC_INTT, 0, K);
montmul:
  dpu.dpu_func(MONT2_ADDR, T1_ADDR, T1_ADDR,  FUNC_MONTMUL, 0, K);
add:
  dpu.dpu_func(   T1_ADDR, S2_ADDR, T1_ADDR,      FUNC_ADD, 0, K);
caddq:
  dpu.dpu_func(   T1_ADDR,       0,       0,    FUNC_CADDQ, 0, K);
pow2round:
  dpu.dpu_func(   T1_ADDR, T1_ADDR, T0_ADDR,FUNC_POW2ROUND, 0, K);
pack1:
  dpu.dpu_pack(      0,  rho,  pk,         0, BYTE_PACK, SEEDBYTES);
pack2:
  dpu.dpu_pack(T1_ADDR, 0,  pk, SEEDBYTES,   T1_PACK,         K);
  spu.shake(256, tr, CRHBYTES, pk, CRYPTO_PUBLICKEYBYTES);
pack3:
```

# Code Implementation – NTT and Inverse NTT

- NTT
  - Step1: permutation
  - Step3: vector multiplication
  - Step2: vector add/sub
  - Step4: go to step1

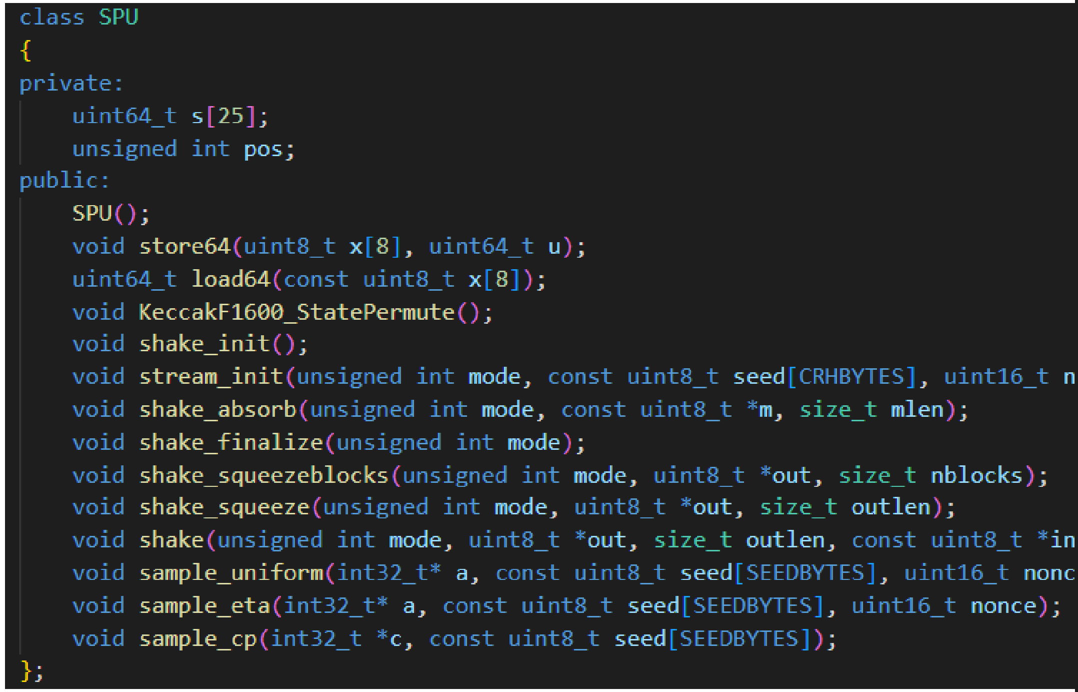


- Inverse NTT
  - Step1: permutation
  - Step2: vector add/sub
  - Step3: vector multiplication
  - Step4: go to step1

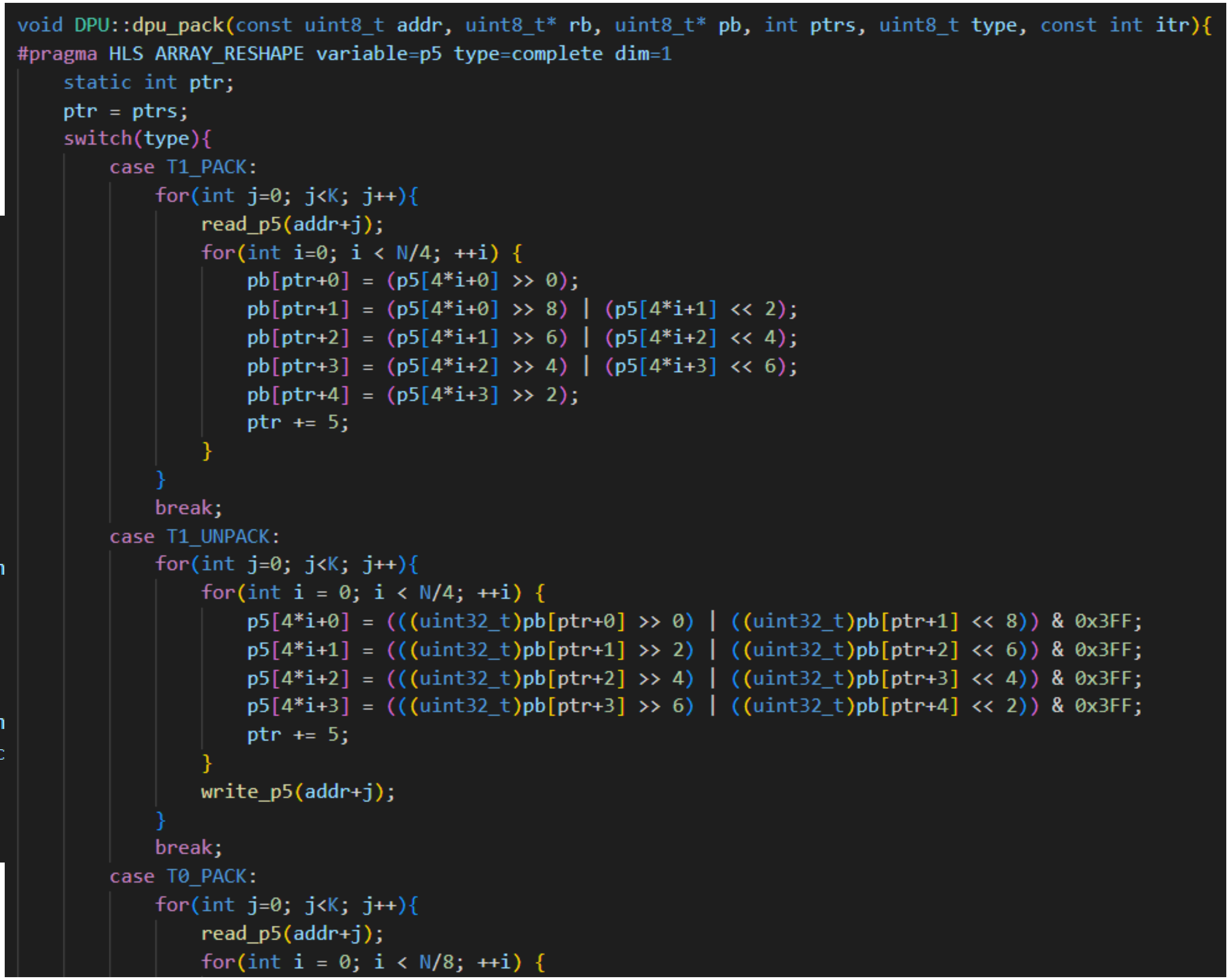


# Code Implementation – SPU unit and Pack unit

- SPU & pack unit



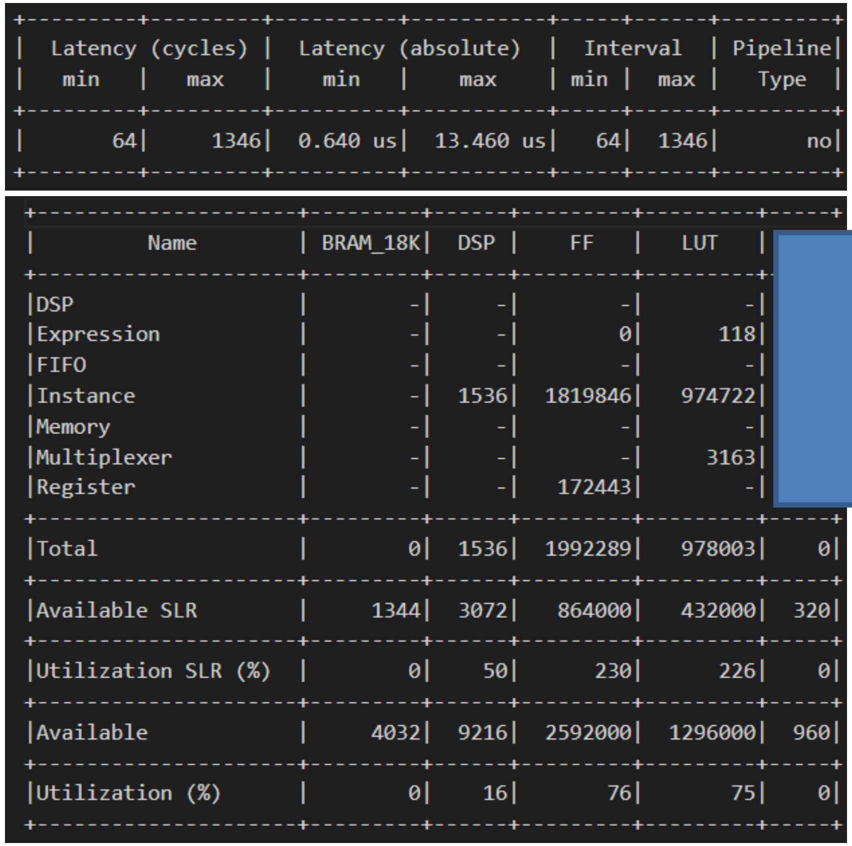
```
class SPU
{
private:
    uint64_t s[25];
    unsigned int pos;
public:
    SPU();
    void store64(uint8_t x[8], uint64_t u);
    uint64_t load64(const uint8_t x[8]);
    void KeccakF1600_StatePermute();
    void shake_init();
    void stream_init(unsigned int mode, const uint8_t seed[CRHBYTES], uint16_t n
    void shake_absorb(unsigned int mode, const uint8_t *m, size_t mlen);
    void shake_finalize(unsigned int mode);
    void shake_squeezeblocks(unsigned int mode, uint8_t *out, size_t nblocks);
    void shake_squeeze(unsigned int mode, uint8_t *out, size_t outlen);
    void shake(unsigned int mode, uint8_t *out, size_t outlen, const uint8_t *in
    void sample_uniform(int32_t* a, const uint8_t seed[SEEDBYTES], uint16_t nonc
    void sample_eta(int32_t* a, const uint8_t seed[SEEDBYTES], uint16_t nonce);
    void sample_cp(int32_t *c, const uint8_t seed[SEEDBYTES]);
};
```



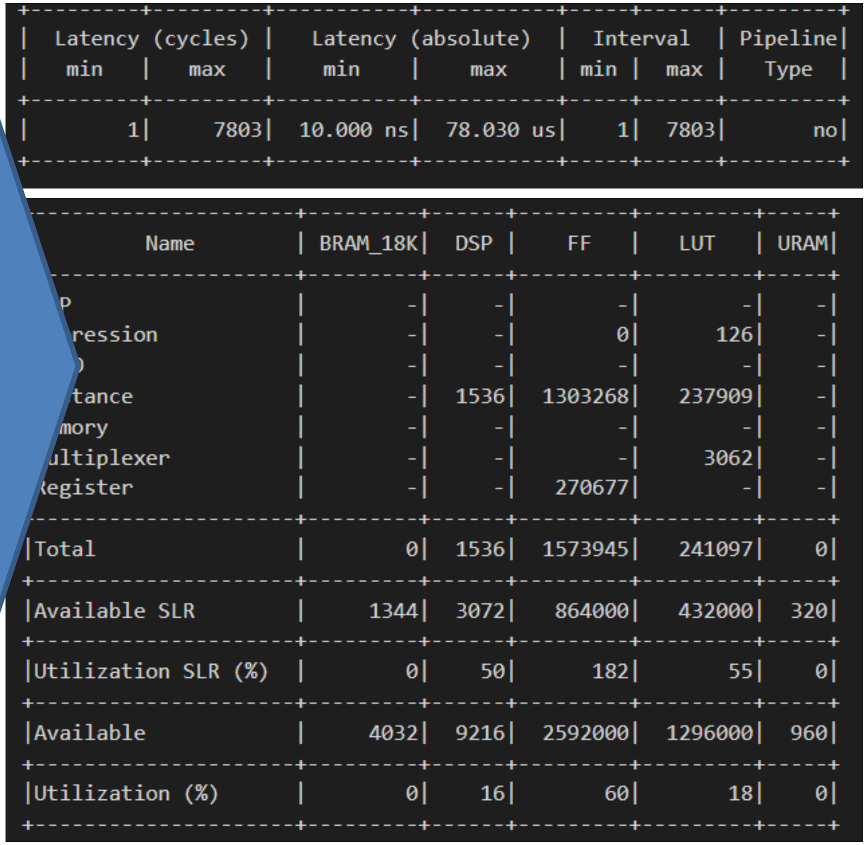
```
void DPU::dpu_pack(const uint8_t addr, uint8_t* rb, uint8_t* pb, int ptrs, uint8_t type, const int itr){
#pragma HLS ARRAY_RESHAPE variable=p5 type=complete dim=1
    static int ptr;
    ptr = ptrs;
    switch(type){
        case T1_PACK:
            for(int j=0; j<K; j++){
                read_p5(addr+j);
                for(int i=0; i < N/4; ++i) {
                    pb[ptr+0] = (p5[4*i+0] >> 0);
                    pb[ptr+1] = (p5[4*i+0] >> 8) | (p5[4*i+1] << 2);
                    pb[ptr+2] = (p5[4*i+1] >> 6) | (p5[4*i+2] << 4);
                    pb[ptr+3] = (p5[4*i+2] >> 4) | (p5[4*i+3] << 6);
                    pb[ptr+4] = (p5[4*i+3] >> 2);
                    ptr += 5;
                }
            }
            break;
        case T1_UNPACK:
            for(int j=0; j<K; j++){
                for(int i = 0; i < N/4; ++i) {
                    p5[4*i+0] = (((uint32_t)pb[ptr+0] >> 0) | ((uint32_t)pb[ptr+1] << 8)) & 0x3FF;
                    p5[4*i+1] = (((uint32_t)pb[ptr+1] >> 2) | ((uint32_t)pb[ptr+2] << 6)) & 0x3FF;
                    p5[4*i+2] = (((uint32_t)pb[ptr+2] >> 4) | ((uint32_t)pb[ptr+3] << 4)) & 0x3FF;
                    p5[4*i+3] = (((uint32_t)pb[ptr+3] >> 6) | ((uint32_t)pb[ptr+4] << 2)) & 0x3FF;
                    ptr += 5;
                }
                write_p5(addr+j);
            }
            break;
        case T0_PACK:
            for(int j=0; j<K; j++){
                read_p5(addr+j);
                for(int i = 0; i < N/8; ++i) {
```

# HLS Observations

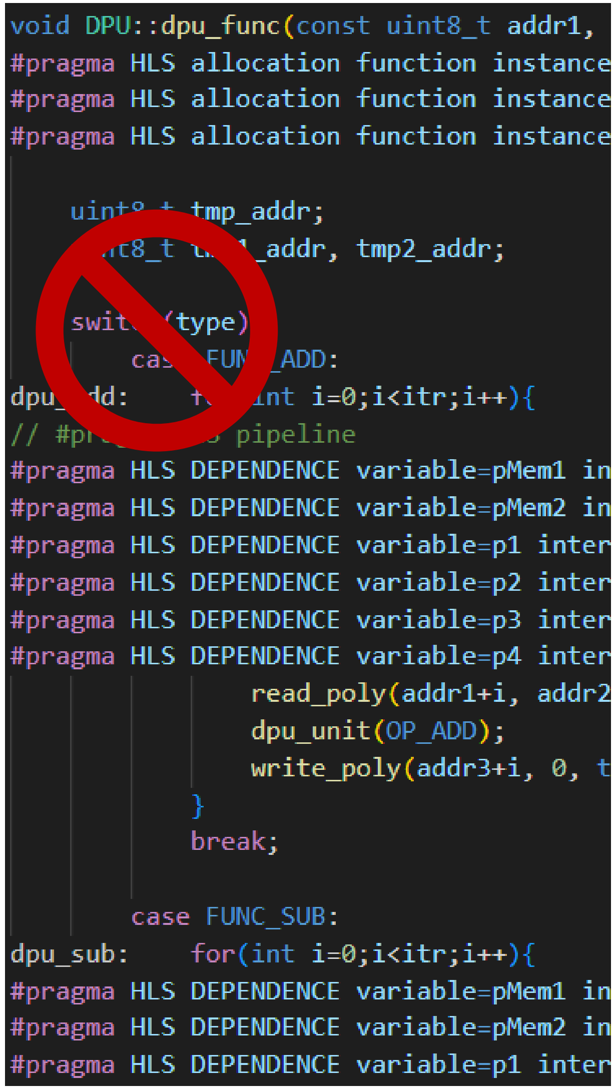
1. DPU Unit high latency and recourse usage
   – Use “if-else” instead of “switch-case”



```
+---------+---------+----------+----------+-----+-----+---------+
|  Latency (cycles) |  Latency (absolute) |  Interval | Pipeline|
|   min   |   max   |    min   |    max   | min | max |   Type  |
+---------+---------+----------+----------+-----+-----+---------+
|       64|     1346| 0.640 us | 13.460 us|   64| 1346|       no|
+---------+---------+----------+----------+-----+-----+---------+

+---------------------+---------+------+---------+---------+
|         Name        | BRAM_18K| DSP  |    FF   |   LUT   |
+---------------------+---------+------+---------+---------+
|DSP                  |        -|     -|        -|        -|
|Expression           |        -|     -|        0|      118|
|FIFO                 |        -|     -|        -|        -|
|Instance             |        -|  1536|  1819846|   974722|
|Memory               |        -|     -|        -|        -|
|Multiplexer          |        -|     -|        -|     3163|
|Register             |        -|     -|   172443|        -|
+---------------------+---------+------+---------+---------+-----+
|Total                |        0|  1536|  1992289|   978003|    0|
+---------------------+---------+------+---------+---------+-----+
|Available SLR        |     1344|  3072|   864000|   432000|  320|
+---------------------+---------+------+---------+---------+-----+
|Utilization SLR (%)  |        0|    50|      230|      226|    0|
+---------------------+---------+------+---------+---------+-----+
|Available            |     4032|  9216|  2592000|  1296000|  960|
+---------------------+---------+------+---------+---------+-----+
|Utilization (%)      |        0|    16|       76|       75|    0|
+---------------------+---------+------+---------+---------+-----+
```



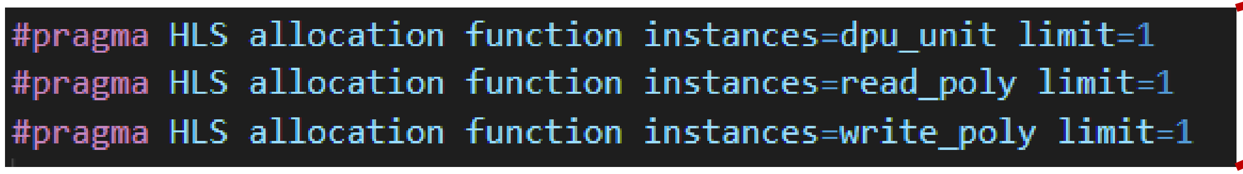
```
+---------+---------+----------+----------+-----+-----+---------+
|  Latency (cycles) |  Latency (absolute) |  Interval | Pipeline|
|   min   |   max   |    min   |    max   | min | max |   Type  |
+---------+---------+----------+----------+-----+-----+---------+
|        1|     7803| 10.000 ns| 78.030 us|    1| 7803|       no|
+---------+---------+----------+----------+-----+-----+---------+

+---------------------+---------+------+---------+---------+-----+
|         Name        | BRAM_18K| DSP  |    FF   |   LUT   | URAM|
+---------------------+---------+------+---------+---------+-----+
 P                    |        -|     -|        -|        -|    -|
 ression              |        -|     -|        0|      126|    -|
 )                    |        -|     -|        -|        -|    -|
 tance                |        -|  1536|  1303268|   237909|    -|
 mory                 |        -|     -|        -|        -|    -|
 ltiplexer            |        -|     -|        -|     3062|    -|
 egister              |        -|     -|   270677|        -|    -|
+---------------------+---------+------+---------+---------+-----+
|Total                |        0|  1536|  1573945|   241097|    0|
+---------------------+---------+------+---------+---------+-----+
|Available SLR        |     1344|  3072|   864000|   432000|  320|
+---------------------+---------+------+---------+---------+-----+
|Utilization SLR (%)  |        0|    50|      182|       55|    0|
+---------------------+---------+------+---------+---------+-----+
|Available            |     4032|  9216|  2592000|  1296000|  960|
+---------------------+---------+------+---------+---------+-----+
|Utilization (%)      |        0|    16|       60|       18|    0|
+---------------------+---------+------+---------+---------+-----+
```



```
void DPU::dpu_func(const uint8_t addr1,
#pragma HLS allocation function instance
#pragma HLS allocation function instance
#pragma HLS allocation function instance

    uint8_t tmp_addr;
     nt8_t t  1_addr, tmp2_addr;

    swit  (type)
        cas  FUN _ADD:
dpu_ dd:    f   int i=0;i<itr;i++){
// #pr     pipeline
#pragma HLS DEPENDENCE variable=pMem1 in
#pragma HLS DEPENDENCE variable=pMem2 in
#pragma HLS DEPENDENCE variable=p1 inter
#pragma HLS DEPENDENCE variable=p2 inter
#pragma HLS DEPENDENCE variable=p3 inter
#pragma HLS DEPENDENCE variable=p4 inter
                read_poly(addr1+i, addr2
                dpu_unit(OP_ADD);
                write_poly(addr3+i, 0, t
            }
            break;

        case FUNC_SUB:
dpu_sub:    for(int i=0;i<itr;i++){
#pragma HLS DEPENDENCE variable=pMem1 in
#pragma HLS DEPENDENCE variable=pMem2 in
#pragma HLS DEPENDENCE variable=p1 inter
```

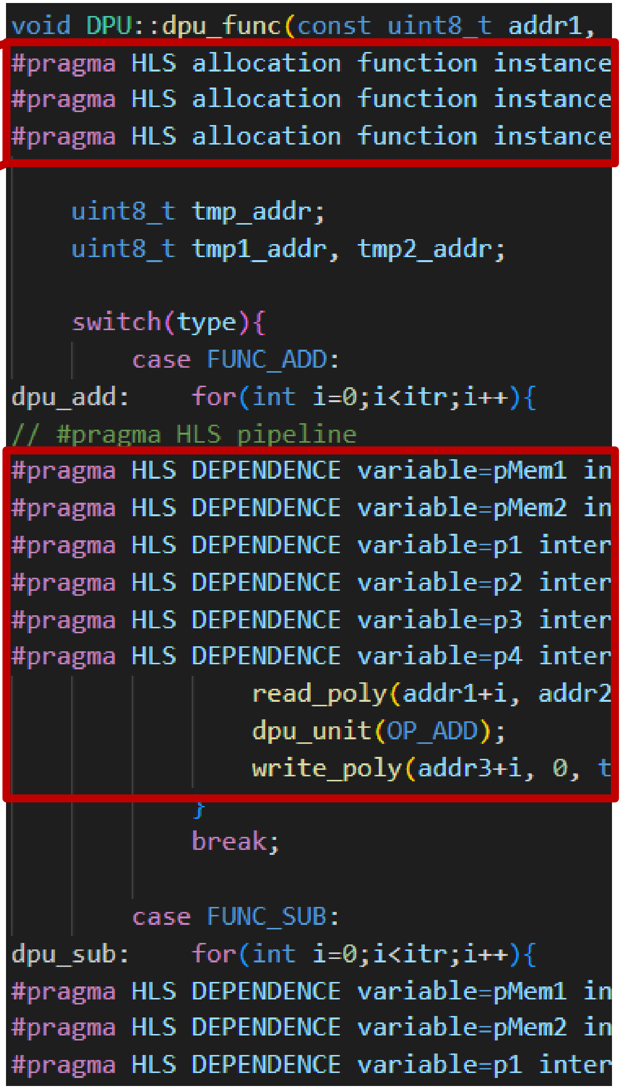
# HLS Observations

2. Use ALLOCATION pragma to restrict instance number



```
#pragma HLS allocation function instances=dpu_unit limit=1
#pragma HLS allocation function instances=read_poly limit=1
#pragma HLS allocation function instances=write_poly limit=1
```

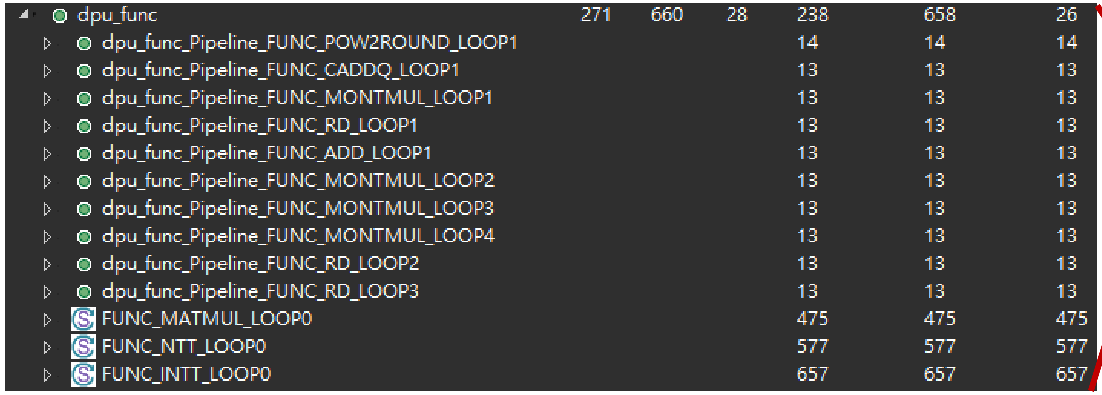
3. Inline read/write function to set false dependency
   – Vitis HLS cannot set false dependency for member in function
   – Add inline pragma in read/write function



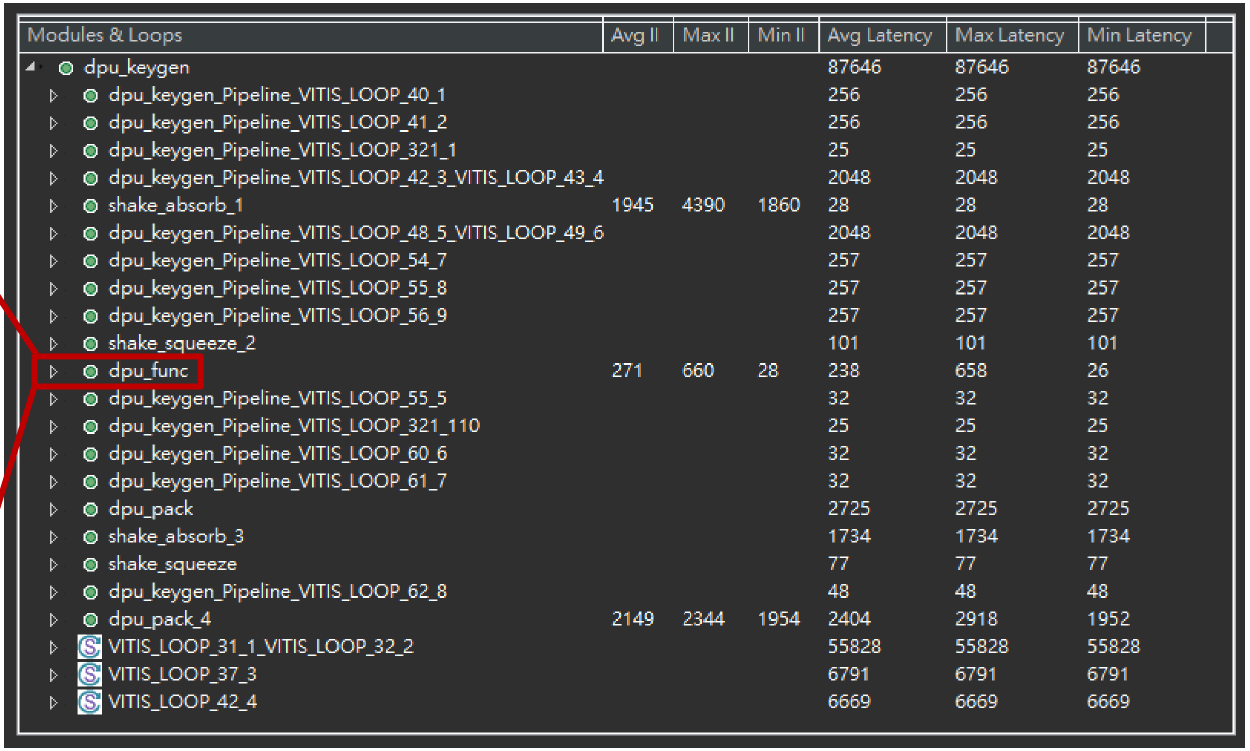
```
void DPU::dpu_func(const uint8_t addr1,
#pragma HLS allocation function instance
#pragma HLS allocation function instance
#pragma HLS allocation function instance

    uint8_t tmp_addr;
    uint8_t tmp1_addr, tmp2_addr;

    switch(type){
        case FUNC_ADD:
dpu_add:    for(int i=0;i<itr;i++){
// #pragma HLS pipeline
#pragma HLS DEPENDENCE variable=pMem1 in
#pragma HLS DEPENDENCE variable=pMem2 in
#pragma HLS DEPENDENCE variable=p1 inter
#pragma HLS DEPENDENCE variable=p2 inter
#pragma HLS DEPENDENCE variable=p3 inter
#pragma HLS DEPENDENCE variable=p4 inter
                read_poly(addr1+i, addr2
                dpu_unit(OP_ADD);
                write_poly(addr3+i, 0, t
            }
            break;

        case FUNC_SUB:
dpu_sub:    for(int i=0;i<itr;i++){
#pragma HLS DEPENDENCE variable=pMem1 in
#pragma HLS DEPENDENCE variable=pMem2 in
#pragma HLS DEPENDENCE variable=p1 inter
```

# HLS Implementation Result

- Total execution time: 87646
  - – SPU and Pack Unit is the bottleneck



| | | | | | | |
|---|---|---|---|---|---|---|
| dpu_func | 271 | 660 | 28 | 238 | 658 | 26 |
| dpu_func_Pipeline_FUNC_POW2ROUND_LOOP1 | | | | 14 | 14 | 14 |
| dpu_func_Pipeline_FUNC_CADDQ_LOOP1 | | | | 13 | 13 | 13 |
| dpu_func_Pipeline_FUNC_MONTMUL_LOOP1 | | | | 13 | 13 | 13 |
| dpu_func_Pipeline_FUNC_RD_LOOP1 | | | | 13 | 13 | 13 |
| dpu_func_Pipeline_FUNC_ADD_LOOP1 | | | | 13 | 13 | 13 |
| dpu_func_Pipeline_FUNC_MONTMUL_LOOP2 | | | | 13 | 13 | 13 |
| dpu_func_Pipeline_FUNC_MONTMUL_LOOP3 | | | | 13 | 13 | 13 |
| dpu_func_Pipeline_FUNC_MONTMUL_LOOP4 | | | | 13 | 13 | 13 |
| dpu_func_Pipeline_FUNC_RD_LOOP2 | | | | 13 | 13 | 13 |
| dpu_func_Pipeline_FUNC_RD_LOOP3 | | | | 13 | 13 | 13 |
| FUNC_MATMUL_LOOP0 | | | | 475 | 475 | 475 |
| FUNC_NTT_LOOP0 | | | | 577 | 577 | 577 |
| FUNC_INTT_LOOP0 | | | | 657 | 657 | 657 |



| Modules & Loops | Avg II | Max II | Min II | Avg Latency | Max Latency | Min Latency |
|---|---|---|---|---|---|---|
| dpu_keygen | | | | 87646 | 87646 | 87646 |
| dpu_keygen_Pipeline_VITIS_LOOP_40_1 | | | | 256 | 256 | 256 |
| dpu_keygen_Pipeline_VITIS_LOOP_41_2 | | | | 256 | 256 | 256 |
| dpu_keygen_Pipeline_VITIS_LOOP_321_1 | | | | 25 | 25 | 25 |
| dpu_keygen_Pipeline_VITIS_LOOP_42_3_VITIS_LOOP_43_4 | | | | 2048 | 2048 | 2048 |
| shake_absorb_1 | 1945 | 4390 | 1860 | 28 | 28 | 28 |
| dpu_keygen_Pipeline_VITIS_LOOP_48_5_VITIS_LOOP_49_6 | | | | 2048 | 2048 | 2048 |
| dpu_keygen_Pipeline_VITIS_LOOP_54_7 | | | | 257 | 257 | 257 |
| dpu_keygen_Pipeline_VITIS_LOOP_55_8 | | | | 257 | 257 | 257 |
| dpu_keygen_Pipeline_VITIS_LOOP_56_9 | | | | 257 | 257 | 257 |
| shake_squeeze_2 | | | | 101 | 101 | 101 |
| dpu_func | 271 | 660 | 28 | 238 | 658 | 26 |
| dpu_keygen_Pipeline_VITIS_LOOP_55_5 | | | | 32 | 32 | 32 |
| dpu_keygen_Pipeline_VITIS_LOOP_321_110 | | | | 25 | 25 | 25 |
| dpu_keygen_Pipeline_VITIS_LOOP_60_6 | | | | 32 | 32 | 32 |
| dpu_keygen_Pipeline_VITIS_LOOP_61_7 | | | | 32 | 32 | 32 |
| dpu_pack | | | | 2725 | 2725 | 2725 |
| shake_absorb_3 | | | | 1734 | 1734 | 1734 |
| shake_squeeze | | | | 77 | 77 | 77 |
| dpu_keygen_Pipeline_VITIS_LOOP_62_8 | | | | 48 | 48 | 48 |
| dpu_pack_4 | 2149 | 2344 | 1954 | 2404 | 2918 | 1952 |
| VITIS_LOOP_31_1_VITIS_LOOP_32_2 | | | | 55828 | 55828 | 55828 |
| VITIS_LOOP_37_3 | | | | 6791 | 6791 | 6791 |
| VITIS_LOOP_42_4 | | | | 6669 | 6669 | 6669 |

# Design Comparison

- Compared with high-end CPU & state-of-the-art FPGA architecture
  - CPU: Intel Core i7-9700
  - FPGA Implementation: ICFPT’21[3]

**Latency (cycle)**

| | CPU | This Work |
|---|---|---|
| **Total Key Generation** | **446475** | **87646** |

| | ICFPT’21[3] | This Work |
|---|---|---|
| **NTT** | **1831** | **577** |
| **Matrix Multiplication** | **1754** | **475** |
| **Inverse NTT** | **1475** | **670** |

[3] L. Beckwith *et al.*, *ICFPT,* 2021.

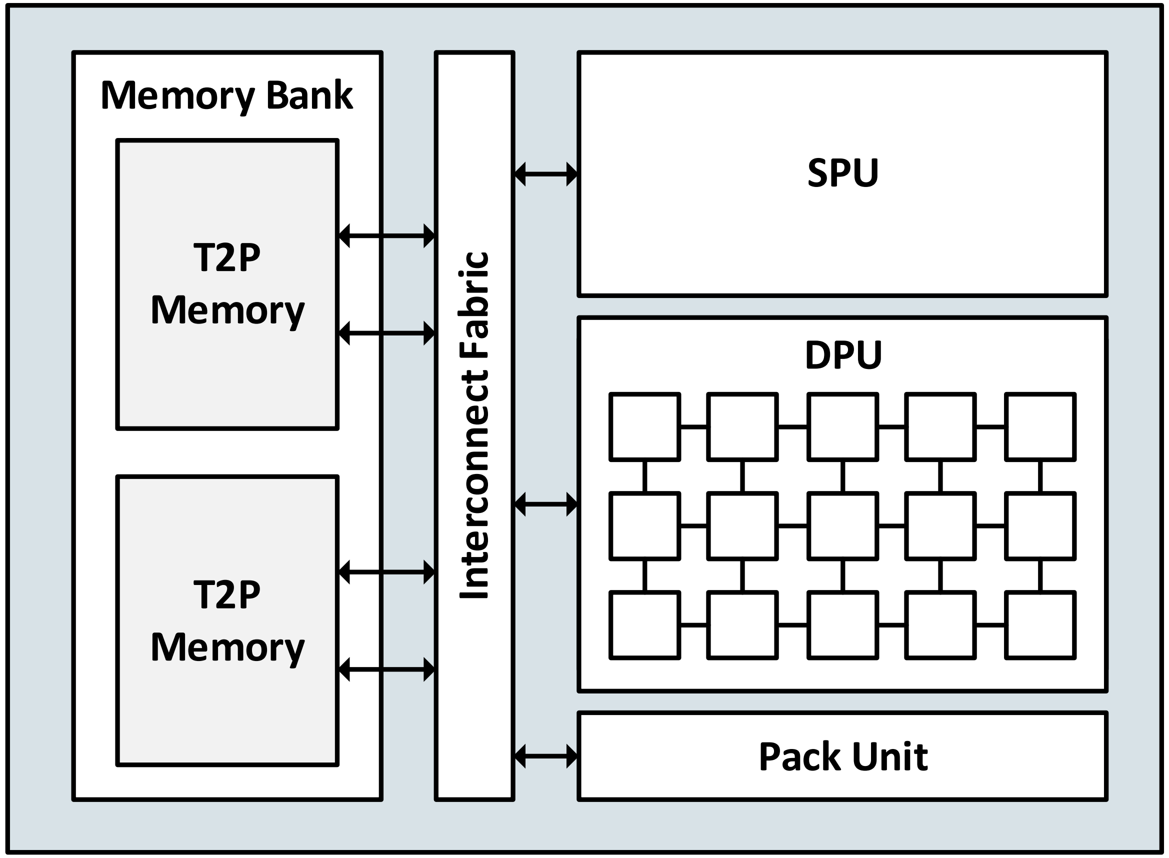
# Architecture Exploration

- Read p1 and read p2 takes two cycle
  – Additional data idle time can increase buffer usage
  – Same situation happens for memory write access

- **2 memory banks**
  – Complicated data access pattern for different operations
  – How to allocate data?

➔ **Algorithm breakdown & analysis**

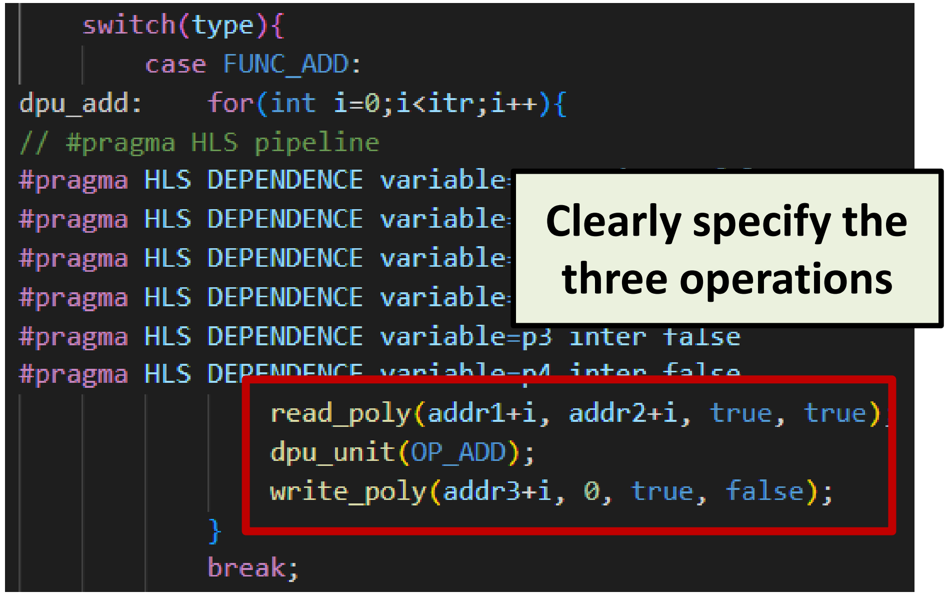
➔ **Intermediate product reallocation**



# Architecture Exploration

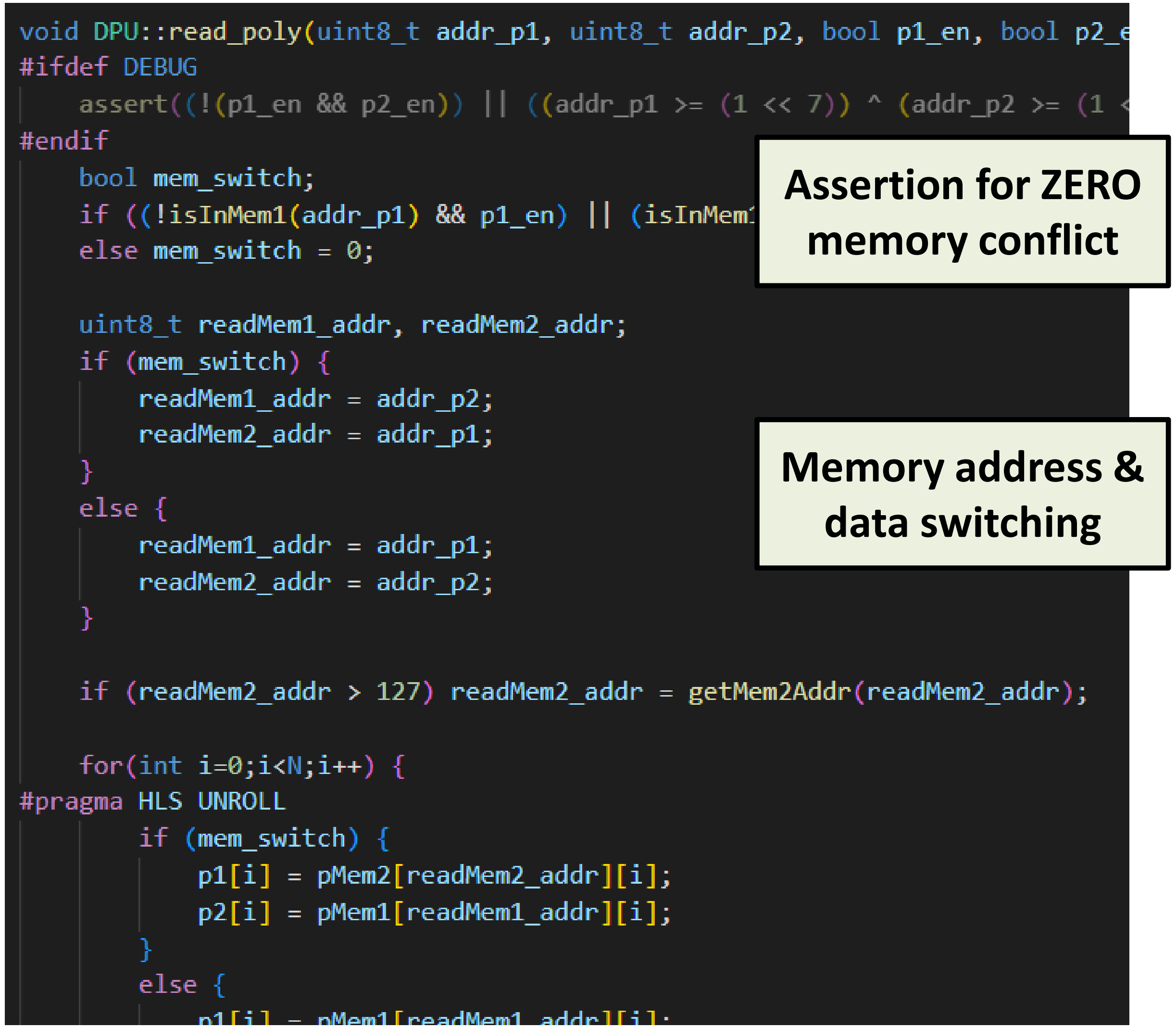
## 2-memory-bank implementation

- Increase memory bandwidth via efficient allocation of variable
  - Enhance pipeline efficiency
  - Read & write in one cycle



```
    switch(type){
        case FUNC_ADD:
dpu_add:    for(int i=0;i<itr;i++){
// #pragma HLS pipeline
#pragma HLS DEPENDENCE variable=
#pragma HLS DEPENDENCE variable=
#pragma HLS DEPENDENCE variable=
#pragma HLS DEPENDENCE variable=
#pragma HLS DEPENDENCE variable=p3 inter false
#pragma HLS DEPENDENCE variable=p4 inter false
                read_poly(addr1+i, addr2+i, true, true)
                dpu_unit(OP_ADD);
                write_poly(addr3+i, 0, true, false);
            }
            break;
```

Clearly specify the three operations



```
void DPU::read_poly(uint8_t addr_p1, uint8_t addr_p2, bool p1_en, bool p2_e
#ifdef DEBUG
    assert((!(p1_en && p2_en)) || ((addr_p1 >= (1 << 7)) ^ (addr_p2 >= (1 <
#endif
    bool mem_switch;
    if ((!isInMem1(addr_p1) && p1_en) || (isInMem1
    else mem_switch = 0;

    uint8_t readMem1_addr, readMem2_addr;
    if (mem_switch) {
        readMem1_addr = addr_p2;
        readMem2_addr = addr_p1;
    }
    else {
        readMem1_addr = addr_p1;
        readMem2_addr = addr_p2;
    }

    if (readMem2_addr > 127) readMem2_addr = getMem2Addr(readMem2_addr);

    for(int i=0;i<N;i++) {
#pragma HLS UNROLL
        if (mem_switch) {
            p1[i] = pMem2[readMem2_addr][i];
            p2[i] = pMem1[readMem1_addr][i];
        }
        else {
            p1[i] = pMem1[readMem1_addr][i];
```

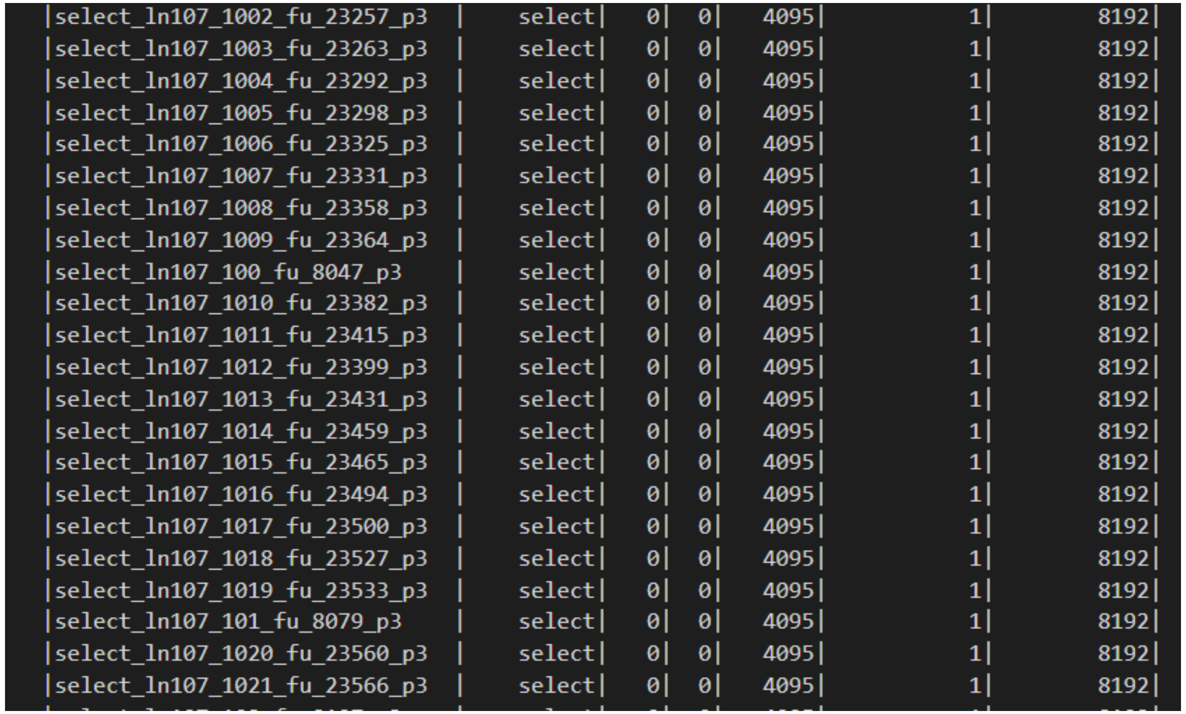
Assertion for ZERO memory conflict

Memory address & data switching

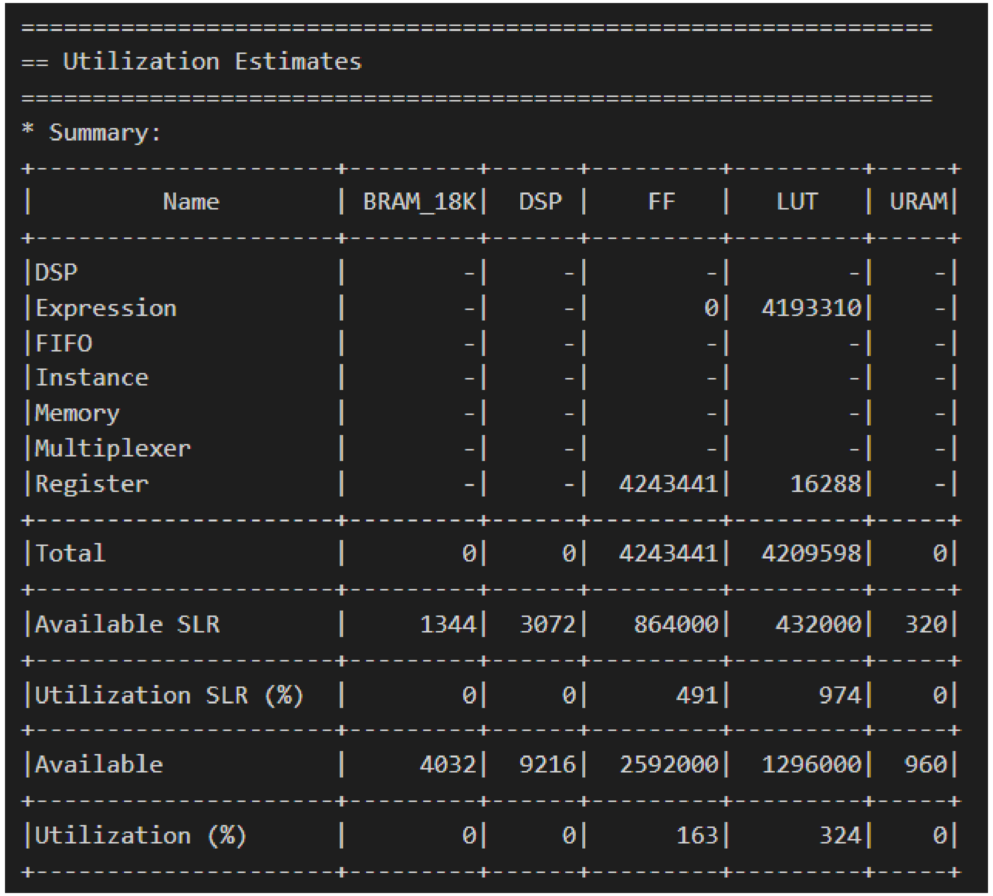
# Architecture Exploration

## Encountered Problem

- Excessively large resource usage for data switching in read module
- Generate numerous expressions using LUT



```
|select_ln107_1002_fu_23257_p3  |    select|  0|  0|  4095|          1|        8192|
|select_ln107_1003_fu_23263_p3  |    select|  0|  0|  4095|          1|        8192|
|select_ln107_1004_fu_23292_p3  |    select|  0|  0|  4095|          1|        8192|
|select_ln107_1005_fu_23298_p3  |    select|  0|  0|  4095|          1|        8192|
|select_ln107_1006_fu_23325_p3  |    select|  0|  0|  4095|          1|        8192|
|select_ln107_1007_fu_23331_p3  |    select|  0|  0|  4095|          1|        8192|
|select_ln107_1008_fu_23358_p3  |    select|  0|  0|  4095|          1|        8192|
|select_ln107_1009_fu_23364_p3  |    select|  0|  0|  4095|          1|        8192|
|select_ln107_100_fu_8047_p3    |    select|  0|  0|  4095|          1|        8192|
|select_ln107_1010_fu_23382_p3  |    select|  0|  0|  4095|          1|        8192|
|select_ln107_1011_fu_23415_p3  |    select|  0|  0|  4095|          1|        8192|
|select_ln107_1012_fu_23399_p3  |    select|  0|  0|  4095|          1|        8192|
|select_ln107_1013_fu_23431_p3  |    select|  0|  0|  4095|          1|        8192|
|select_ln107_1014_fu_23459_p3  |    select|  0|  0|  4095|          1|        8192|
|select_ln107_1015_fu_23465_p3  |    select|  0|  0|  4095|          1|        8192|
|select_ln107_1016_fu_23494_p3  |    select|  0|  0|  4095|          1|        8192|
|select_ln107_1017_fu_23500_p3  |    select|  0|  0|  4095|          1|        8192|
|select_ln107_1018_fu_23527_p3  |    select|  0|  0|  4095|          1|        8192|
|select_ln107_1019_fu_23533_p3  |    select|  0|  0|  4095|          1|        8192|
|select_ln107_101_fu_8079_p3    |    select|  0|  0|  4095|          1|        8192|
|select_ln107_1020_fu_23560_p3  |    select|  0|  0|  4095|          1|        8192|
|select_ln107_1021_fu_23566_p3  |    select|  0|  0|  4095|          1|        8192|
```



```
================================================================
== Utilization Estimates
================================================================
* Summary:
+---------------------+---------+------+---------+---------+-----+
|         Name        | BRAM_18K|  DSP |    FF   |   LUT   | URAM|
+---------------------+---------+------+---------+---------+-----+
|DSP                  |        -|     -|        -|        -|    -|
|Expression           |        -|     -|        0|  4193310|    -|
|FIFO                 |        -|     -|        -|        -|    -|
|Instance             |        -|     -|        -|        -|    -|
|Memory               |        -|     -|        -|        -|    -|
|Multiplexer          |        -|     -|        -|        -|    -|
|Register             |        -|     -|  4243441|    16288|    -|
+---------------------+---------+------+---------+---------+-----+
|Total                |        0|     0|  4243441|  4209598|    0|
+---------------------+---------+------+---------+---------+-----+
|Available SLR        |     1344|  3072|   864000|   432000|  320|
+---------------------+---------+------+---------+---------+-----+
|Utilization SLR (%)  |        0|     0|      491|      974|    0|
+---------------------+---------+------+---------+---------+-----+
|Available            |     4032|  9216|  2592000|  1296000|  960|
+---------------------+---------+------+---------+---------+-----+
|Utilization (%)      |        0|     0|      163|      324|    0|
+---------------------+---------+------+---------+---------+-----+
```

# Summary & Conclusions

- **Fully-Integrated architecture for Dilithium using high level synthesis**
  - Approximately good as state-of-the-art design

**Difficulties**

- Rewriting the C code takes time
  - 2000+ lines code rewrite
  - 2-3 weeks labor work for 2 people

**What’s next? (If possible)**

- Finish two-memory bank design
- Manual resource sharing in the DPU PE array
- Optimization on SPU and Pack Unit

# References

1. NIST Post-Quantum Cryptography Standardization: https://csrc.nist.gov/Projects/post-quantum-cryptography
2. Dilithium: https://pq-crystals.org/dilithium/index.shtml
3. L. Beckwith, D. T. Nguyen and K. Gaj, "High-Performance Hardware Implementation of CRYSTALS-Dilithium," *2021 International Conference on Field-Programmable Technology (ICFPT)*, pp. 1-10, 2021.