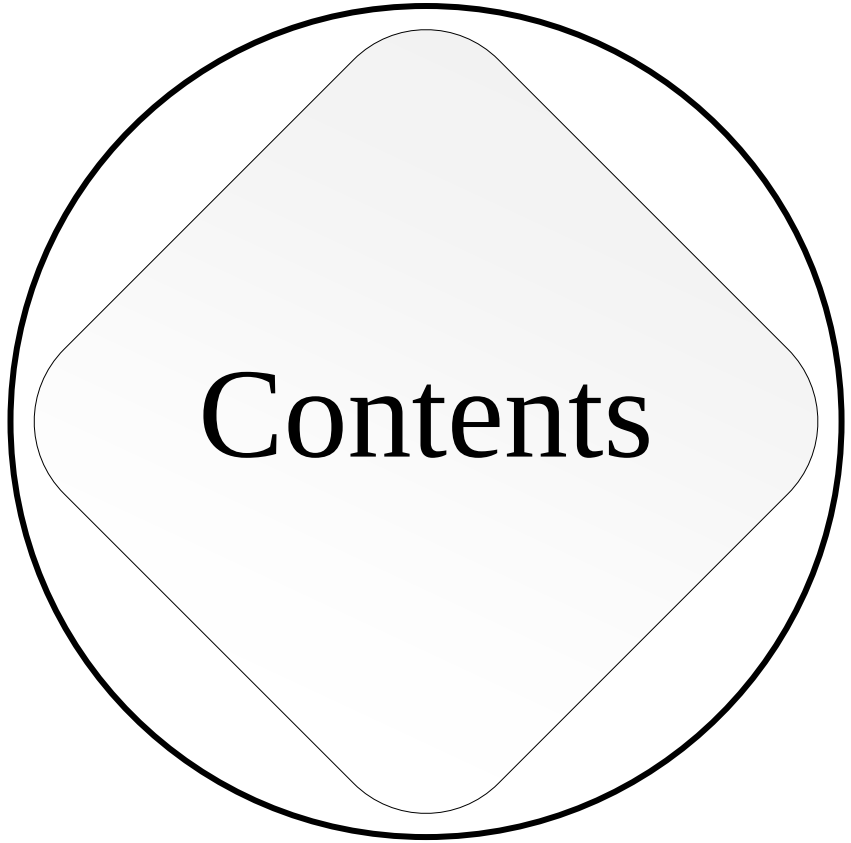
Design and Implementation of a Hardware Accelerator IP for Post-Quantum Cryptography ML-DSA Compatible with the AXI-4 Interface

學生：蘇柏丞

指導教授：林銘波



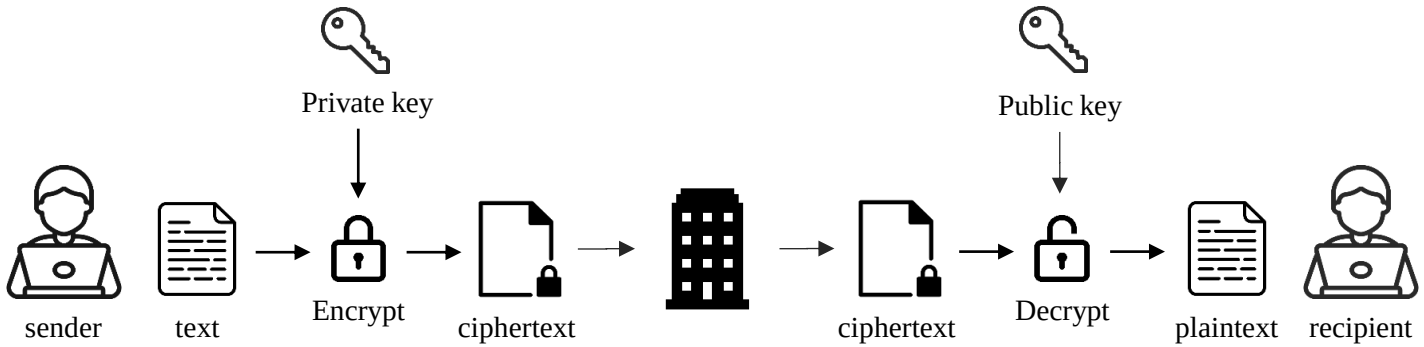
Introduction

Background

- ✓ Shor's algorithm, combined with a powerful quantum computer, will possibly break RSA and ECC.
- ✓ Initiated by NIST in 2016, the post-quantum cryptography standardization process, , it finalized the selection of **ML-DSA** as one of the encryption methods.
- ✓ Previously known as CRYSTAL-DILITHIUM

ML-DSA

- ✓ Defines method for generating digital signatures
- ✓ Based on the worst-case hardness of module lattice problems, it has potential resistance against both quantum and classical attacks.
- ✓ Advantages include fast arithmetic operations, efficient encryption, and compact signatures.
- ✓ Uses uniformly sampled high-entropy Gaussian-distributed secrets to generate random keys.
- ✓ The core security challenges of ML-DSA include MLWE problem and tMSIS problem



Fiat-Shamir with Aborts

1. Commitment：
   - The signer generates a random vector y ∈ R_q^{ℓ}
   - The commitment value is $w = Ay$
   - w is rounded to obtain w_1
2. Challenge：
   - The challenge c is generated by hashing w_1 and the message representative μ
3. Response：
   - The response $z = y + S_1 \cdot c$(where S_1 is part of the private key)
   - Use rejection sampling to check whether z meets specific coefficient bounds
4. Hint Calculation：
   - To enable the verifier to reconstruct w_1 from z and the compressed public value t_1
   - hint h ∈ R_q^{k}
5. Signature Composition：
   - The final signature consists of three parts: the rounded commitment w_1, the response z, and the hint h
6. Second Stage of Rejection Sampling：
   - To ensure the correctness of the signature, a second stage of rejection sampling must be performed

MLWE (module learning with errors)

Setup:

1. Modulus q=7.
2. Matrix A is of size 2×2, with elements selected randomly.
3. Secret vectors $s1$ and $s2$ are both of size 2×1.
4. Values for A, $s1$, $s2$:

$$A = \begin{bmatrix} 3 & 4 \\ 1 & 5 \end{bmatrix}, \quad s_1 = \begin{bmatrix} 2 \\ 3 \end{bmatrix}, \quad s_2 = \begin{bmatrix} 1 \\ 4 \end{bmatrix}$$

Calculation Steps:

1. Calculate $As1$:

$$As_1 = \begin{bmatrix} 3 & 4 \\ 1 & 5 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 3 \cdot 2 + 4 \cdot 3 \\ 1 \cdot 2 + 5 \cdot 3 \end{bmatrix} = \begin{bmatrix} 6 + 12 \\ 2 + 15 \end{bmatrix} = \begin{bmatrix} 18 \\ 17 \end{bmatrix}$$

2. Add the secret vector $s2$ to the result and take modulus q:

$$t = As_1 + s_2 = \begin{bmatrix} 18 \\ 17 \end{bmatrix} + \begin{bmatrix} 1 \\ 4 \end{bmatrix} = \begin{bmatrix} 19 \\ 21 \end{bmatrix}$$

$$t = \begin{bmatrix} 19 \mod 7 \\ 21 \mod 7 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$$

3. The public data is the matrix A and the result vector t:

$$A = \begin{bmatrix} 3 & 4 \\ 1 & 5 \end{bmatrix}, \quad t = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$$

MSIS (module shortest integer solution)

Setup:

1. Modulus $q = 7$
2. Matrix A is of size 3×2, with elements selected randomly.
3. Values for A:

$$A = \begin{bmatrix} 3 & 4 \\ 1 & 5 \\ 6 & 2 \end{bmatrix}$$

Goal:

Find vectors z and u such that $Az + u = 0 \bmod q$.

Attempt to Solve:

1. Assume a vector z, u:

$$z = \begin{bmatrix} 2 \\ -1 \end{bmatrix}, \quad u = \begin{bmatrix} -1 \\ 3 \\ -5 \end{bmatrix}$$

2. Calculate $Az + u$ and take modulus q:

$$Az = \begin{bmatrix} 3 & 4 \\ 1 & 5 \\ 6 & 2 \end{bmatrix} \begin{bmatrix} 2 \\ -1 \end{bmatrix} = \begin{bmatrix} 3 \cdot 2 + 4 \cdot (-1) \\ 1 \cdot 2 + 5 \cdot (-1) \\ 6 \cdot 2 + 2 \cdot (-1) \end{bmatrix} = \begin{bmatrix} 6 - 4 \\ 2 - 5 \\ 12 - 2 \end{bmatrix} = \begin{bmatrix} 2 \\ -3 \\ 10 \end{bmatrix}$$

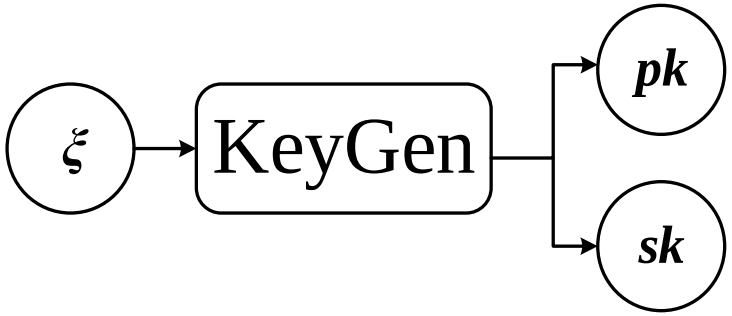
$$Az + u = \begin{bmatrix} 2 \\ -3 \\ 10 \end{bmatrix} + \begin{bmatrix} -1 \\ 3 \\ -5 \end{bmatrix} = \begin{bmatrix} 2 + (-1) \\ -3 + 3 \\ 10 + (-5) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 5 \end{bmatrix}$$

$$Az + u \mod 7 = \begin{bmatrix} 1 \mod 7 \\ 0 \mod 7 \\ 5 \mod 7 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 5 \end{bmatrix}$$

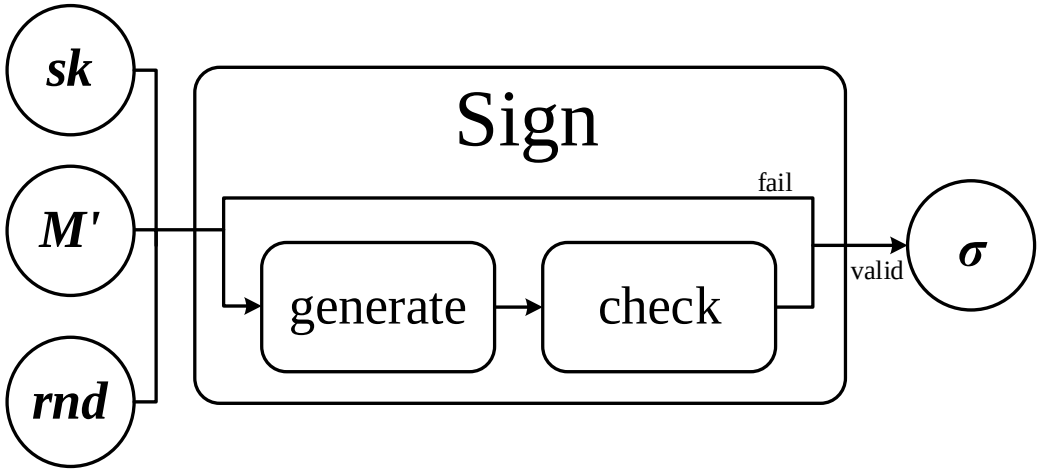
Algorithms

Algorithm

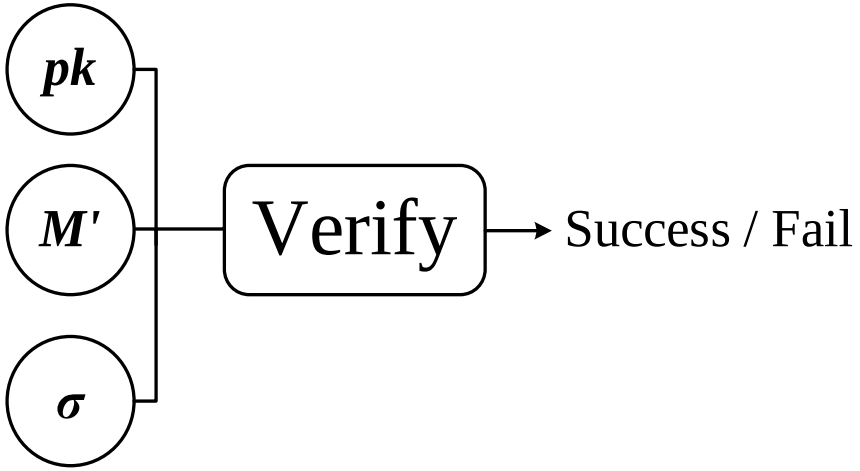
1. Key generation (KeyGen)



2. Signature generation (Sign)



3. Signature verification (Verify)



Symbols

- ξ : random seed
- ***p***k : public key
- ***sk*** : secret key
- ***M'*** : hash message
- ***rnd*** : random number
- σ : signature

Key Generation

Algorithm 6 ML-DSA.KeyGen_internal(ξ)

Generates a public-private key pair from a seed.

Input: Seed $\xi \in \mathbb{B}^{32}$

Output: Public key $pk \in \mathbb{B}^{32+32k(\text{bitlen }(q-1)-d)}$
and private key $sk \in \mathbb{B}^{32+32+64+32\cdot((\ell+k)\cdot\text{bitlen }(2\eta)+dk)}$

1: $(\rho, \rho', K) \in \mathbb{B}^{32} \times \mathbb{B}^{64} \times \mathbb{B}^{32} \leftarrow$ H(ξ||IntegerToBytes(k, 1)||IntegerToBytes(ℓ, 1), 128)
2: ▷ expand se
3: $\hat{\mathbf{A}} \leftarrow$ ExpandA(ρ) ▷ $\mathbf{A}$ is generated and stored in NTT representation as
4: $(\mathbf{s}_1, \mathbf{s}_2) \leftarrow$ ExpandS(ρ')
5: $\mathbf{t} \leftarrow \text{NTT}^{-1}(\hat{\mathbf{A}} \circ \text{NTT}(\mathbf{s}_1)) + \mathbf{s}_2$ ▷ compute $\mathbf{t} = \mathbf{A}\mathbf{s}_1 +$
6: $(\mathbf{t}_1, \mathbf{t}_0) \leftarrow$ Power2Round($\mathbf{t}$) ▷ compres
7: ▷ PowerTwoRound is applied componentwise (see explanatory text in Section 7
8: $pk \leftarrow$ pkEncode($\rho, \mathbf{t}_1$)
9: $tr \leftarrow$ H(pk, 64)
10: $sk \leftarrow$ skEncode($\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0$) ▷ K and tr are for use in signi
11: **return** (pk, sk)

```python
def KeyGen(xi):
    H_xi = SHAKE_256(xi,1024)
    print(H_xi)
    p = H_xi[:32]
    p_prime = H_xi[32:96]
    K = H_xi[96:128]
    A_hat = ExpandA(p)
    s1, s2 = ExpandS(p_prime)
    s1Hat = [NTT(s) for s in s1]
    s1Hat = np.array(s1Hat)
    A_NTT_s1 = NTT_dot(A_hat,s1Hat)
    aHat_mul_s1Hat = [NTT_inv(s) for s in A_NTT_s1]
    t = []
    for i in range(ML_DSA["k"]):
        d = []
        for k in range(256):
            sum = aHat_mul_s1Hat[i][k] + s2[i][k]
            d.append(sum)
        t.append(d)
    t1 = []
    t0 = []
    for ti in range (ML_DSA["k"]):
        ta1 = []
        ta0 = []
        for tp in range(256):
            t1_temp,t0_temp = Power2Round(t[ti][tp])
            ta1.append(t1_temp)
            ta0.append(t0_temp)
        t1.append(ta1)
        t0.append(ta0)
    pk = pk_encode(p, t1)
    tr = SHAKE_256(pk,512)
    sk = sk_encode(p, K, tr, s1, s2, t0)
    return pk, sk
```

Key Generation

```
seed: 6CAE2E9C2CF64D2686C31C2118E0F24A47DD46DB85590910AAC9DF4C1B854E44

rho: C8BEADEDC6DBA5BF3BECA52C67CEAFB4F3EBF84190B2CFA6BCA132883129A28B

rhoPrime: 11779B16A7054953860C14796F63018C9EFD3957CC53A12AF727A5AFC64507445D9EA5E19B6403B3DD3ABAD9B1DAD1146E9C64410E372E7A6D9973F0D04D9632

k: B149C045A55EADA0C519069A8EE0602FBEDA8D2EDFEA09CAE01D542D47DCBA1E

aHat: [[[4518441, 4610216, 2805006, 6522567, 958931, 2266298, 7298857, 6160680, 4376220, 5886423, 2456656, 7246256, 4825911, 4337879, 2286865, 4
[1518172, 2060152, 4749985, 6513620, 2245042, 7549147, 2532897, 6922184, 1547706, 7925910, 4641118, 6372818, 5442868, 3048857, 7986176, 21420, 5
[5580016, 7782159, 4916820, 3492846, 1528232, 8008932, 7778144, 980016, 3083229, 8050068, 4533047, 3121986, 1216278, 1788935, 5913428, 2162915,
[3867698, 3883645, 7640217, 1653450, 7082472, 1447081, 7250588, 6581285, 294533, 5402653, 604135, 2911419, 6568667, 5301208, 4153480, 7717253, 3
[[1279303, 2532728, 3723926, 1729839, 3554515, 7192021, 7349548, 488026, 4908512, 4753212, 2935848, 582517, 8226312, 3995094, 3902326, 5741747,
[2588787, 7264972, 1949825, 6006983, 4106024, 65365, 8042118, 4118970, 7298493, 5150193, 3503187, 6298208, 7082413, 6628507, 875436, 2772530, 14
[7214090, 1245002, 5091873, 3288262, 5791684, 3803755, 1182560, 491901, 8125913, 8076680, 5245769, 261418, 5214617, 1778846, 4876381, 7795651, 7
[1502785, 581000, 1879879, 3156914, 1881, 5520763, 5935759, 6693937, 3320379, 537813, 3615546, 5159640, 8378114, 7826275, 4223748, 1036709, 4188
[[2404918, 276932, 3882934, 6309816, 7054, 1227527, 6032464, 1468902, 1006551, 7960608, 2274509, 6217106, 2692912, 3723609, 7365367, 479793, 827
[693190, 1361324, 7727759, 1970984, 6574841, 5428942, 6405128, 7678800, 803027, 5292092, 7678200, 2171904, 4578474, 116086, 5949644, 7854469, 44
[7312216, 779896, 2063100, 2626307, 113765, 2660404, 5929719, 639671, 4486125, 7505161, 3557068, 3961934, 3889306, 5903614, 4669780, 3123630, 41
[5053121, 7113161, 8075856, 4167528, 3962210, 5505083, 2796737, 4967776, 2306280, 5546792, 2245077, 1129294, 1964533, 3418665, 3511436, 8207089,
[[8083583, 3829710, 4605090, 5594754, 3627807, 5380754, 6806165, 770598, 3986640, 7635515, 5405099, 2939507, 6176056, 5874875, 3050712, 2456835,
[5225355, 3636085, 6264034, 4804566, 1436962, 4576464, 7345998, 2774594, 1298527, 6241183, 6452112, 187476, 4626517, 6625557, 6117743, 6996883,
[1465241, 4597311, 4033004, 7584645, 4594230, 4330242, 6022842, 5220659, 1647018, 7693321, 6223896, 8022657, 5312843, 5162426, 1117933, 5704909,
[8338083, 4172559, 2550928, 1858116, 1603331, 4131505, 2410053, 6945245, 898089, 3000517, 836782, 3521873, 334161, 4235527, 2384101, 1220958, 38
```

Key Generation

```
s1: [[-2, -1, 1, 2, -2, -1, 2, 2, -1, 1, 1, 0, -2, 2, 0, 2, -2, 0, 2, 2, -1, 2, -1, 2, 0, -2, -2, 0, 0, -2, 2, -2, 2, -2, 1, 1
[1, 1, 0, 0, 0, 2, 0, 0, 1, 1, 2, 0, -1, -1, 0, 0, -1, 1, 1, 0, 2, -1, -1, 0, 0, 0, 2, 1, -2, 2, -2, -1, 1, -2, -1, 1, -1, 2,
[0, -1, -2, 0, 2, -1, 1, -1, -2, -1, 1, -2, 2, 2, 0, 2, -1, -1, 1, -2, 0, 0, 2, 0, 0, 0, 2, -2, 2, 2, -1, 2, 2, 0, 2, 0, -1, 1
[0, -2, 2, 2, 2, -2, 2, 0, 1, 1, -2, -2, 2, 2, -1, -1, -2, 1, 1, -2, -2, 0, -2, -1, 1, -2, 1, -1, 2, -1, 2, -1, 0, -2, -2, 2,

s2: [[1, 1, -1, 0, 1, -1, 2, -2, 1, 0, 0, 1, -2, -1, 2, -1, 2, 1, 1, 0, 2, 1, -1, -1, 0, 2, -2, 0, 2, 2, 2, -2, -2, -2, -1, -2
[2, 0, 1, 1, 2, -1, 0, 0, -1, 1, -1, 0, -2, 0, -1, 2, 2, 1, 0, -2, 2, -1, -1, 1, 2, -2, 1, -2, 1, 0, 0, 2, -2, -2, 0, -2, 1, -
[0, -2, -1, -2, 1, 2, 1, 2, 0, -2, 1, 2, 1, -2, -1, -2, 0, -2, 1, 0, 0, 0, -1, 2, 0, -1, 0, -2, -2, -1, 1, 2, 0, 1, -1, 1, 2,
[1, 2, 1, 1, 1, -1, -2, 0, 2, 1, -1, 2, -1, 1, 0, -2, 2, 1, -1, -2, -1, 0, -1, -2, 2, -1, -1, -1, -1, 2, -1, 1, 0, 1, 1, 1, 1,

s1Hat: [[6579390, 3234202, 5760413, 813693, 7870206, 2714807, 5107675, 3985485, 7446642, 7802351, 141, 5569695, 7400683, 44564
[1777819, 261206, 3793527, 4091808, 8075935, 8319015, 1591393, 6418054, 2659780, 5318519, 4711574, 7434797, 1779310, 4891453,
[4210846, 2632563, 1462415, 7922880, 6998343, 2905191, 1706001, 4830423, 8168731, 3073535, 1021728, 3391631, 7847474, 4202026,
[3403971, 3716226, 2350306, 311129, 1392253, 5521860, 2432006, 1589053, 715014, 3344243, 3872748, 7139941, 933479, 6536172, 70

aHat * s1Hat: [[7173756, 4463163, 7813712, 8016531, 3997849, 5162484, 7557753, 5209556, 2455766, 1538558, 5954781, 7567856, 82
[5284259, 3885376, 1368620, 132205, 6879018, 685232, 7336444, 6249654, 2777641, 2709738, 3850162, 6826710, 7020678, 3892065, 1
[1828613, 8016403, 5983422, 6981535, 5591372, 3243001, 2083753, 755060, 6994817, 236727, 7259592, 3082803, 4788195, 4417006, 1
[7860225, 8258854, 8004522, 1814955, 6846388, 7533577, 2341537, 5866449, 3366727, 6720900, 7291455, 4012098, 6941796, 2314989,

NTTInverse(aHat * s1Hat): [[4089385, 3243627, 2997576, 1860759, 7743501, 7853441, 1170077, 1195218, 7888106, 665458, 5751129,
[3424695, 3851902, 7946663, 7319124, 3293286, 4224957, 4060028, 3286208, 60159, 2504816, 5758015, 5804699, 749986, 7462904, 53
[1285947, 630930, 2012121, 7066228, 3129344, 6394749, 6593383, 4387907, 887463, 812692, 603020, 4377173, 4103483, 1156382, 536
[2744379, 6885345, 1807923, 6069656, 7723085, 1276462, 7935274, 1842025, 7671994, 1471837, 2361166, 5712830, 6416006, 4256155,
```

Key Generation

```
t: [[4089386, 3243628, 2997575, 1860759, 7743502, 7853440, 1170079, 1195216, 7888107, 665458, 5751129, 5154175, 7545299, 4808039, 4175100, 72
[3424697, 3851902, 7946664, 7319125, 3293288, 4224956, 4060028, 3286208, 60158, 2504817, 5758014, 5804699, 749984, 7462904, 5351145, 2632037,
[1285947, 630928, 2012120, 7066226, 3129345, 6394751, 6593384, 4387909, 887463, 812690, 603021, 4377175, 4103484, 1156380, 5369497, 8071446,
[2744380, 6885347, 1807924, 6069657, 7723086, 1276461, 7935272, 1842025, 7671996, 1471838, 2361165, 5712832, 6416005, 4256156, 2086655, 60419

t0: [[1578, -404, -697, 1175, 2062, -2688, -1377, -816, -789, 1906, 345, 1407, 467, -665, -2820, 3878, -2013, -413, -675, -103, -1522, -3652,
[441, 1662, 424, 3669, 104, -2116, -3204, 1216, 2814, -1935, -962, -3429, -3680, -8, 1769, 2405, 2469, -3357, 330, 1151, 2380, -3509, 3340, -
[-197, 144, -3112, -3470, 1, -3201, -1176, -3003, 2727, 1682, -3187, 2647, -708, 1308, 3737, 2326, 4077, -2508, -3615, 3899, 1482, 3316, -100
[60, 4067, -2508, -615, -1970, -1491, -2776, -1175, -3908, -2722, 1869, 3008, 1669, -3684, -2305, -3718, 2776, 1940, -1385, -1043, -1351, 295

t1: [[499, 396, 366, 227, 945, 959, 143, 146, 963, 81, 702, 629, 921, 587, 510, 885, 898, 534, 598, 982, 588, 490, 744, 917, 674, 991, 336, 7
[418, 470, 970, 893, 402, 516, 496, 401, 7, 306, 703, 709, 92, 911, 653, 321, 348, 75, 64, 856, 862, 617, 913, 204, 889, 0, 782, 232, 931, 17
[157, 77, 246, 863, 382, 781, 805, 536, 108, 99, 74, 534, 501, 141, 655, 985, 278, 516, 644, 224, 738, 406, 824, 666, 267, 674, 162, 664, 205
[335, 840, 221, 741, 943, 156, 969, 225, 937, 180, 288, 697, 783, 520, 255, 738, 958, 592, 923, 83, 807, 741, 144, 553, 864, 236, 361, 328, 5

tr: 75A821E4FF2B52A3AB3DDD0C77C3A9F96FCC9BE360C2B75C97D7F9DEC97D1BDDE028D36C4FE18093AF6C5794AD19F9FA090C19A76F05A7F3B930B11792A13A7A
pk: C8BEADEDC6DBA5BF3BECA52C67CEAFB4F3EBF84190B2CFA6BCA132883129A28BF331E6D638B1FFFE8824C347E16B9D992FE95FDD825B68A5F54CAA876EE5A27E0FD5B5A89
sk: C8BEADEDC6DBA5BF3BECA52C67CEAFB4F3EBF84190B2CFA6BCA132883129A28BB149C045A55EADA0C519069A8EE0602FBEDA8D2EDFEA09CAE01D542D47DCBA1E75A821E4F
```

Signing

Algorithm 7 ML-DSA.Sign_internal(sk, M', rnd)

Deterministic algorithm to generate a signature for a formatted message M'.

Input: Private key $sk \in \mathbb{B}^{32+32+64+32\cdot((\ell+k)\cdot \text{bitlen }(2\eta)+dk)}$, formatted message $M' \in \{0,1\}^*$, and per message randomness or dummy variable $rnd \in \mathbb{B}^{32}$.
Output: Signature $\sigma \in \mathbb{B}^{\lambda/4+\ell\cdot 32\cdot(1+\text{bitlen }(\gamma_1-1))+\omega+k}$.

1: $(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0) \leftarrow \text{skDecode}(sk)$
2: $\hat{\mathbf{s}}_1 \leftarrow \text{NTT}(\mathbf{s}_1)$
3: $\hat{\mathbf{s}}_2 \leftarrow \text{NTT}(\mathbf{s}_2)$
4: $\hat{\mathbf{t}}_0 \leftarrow \text{NTT}(\mathbf{t}_0)$
5: $\hat{\mathbf{A}} \leftarrow \text{ExpandA}(\rho)$ ▷ $\mathbf{A}$ is generated and stored in NTT representation as $\hat{\mathbf{A}}$
6: $\mu \leftarrow \text{H}(\text{BytesToBits}(tr)||M', 64)$ ▷ message representative that may optionally be computed in a different cryptographic module
7: $\rho'' \leftarrow \text{H}(K||rnd||\mu, 64)$ ▷ compute private random seed
8: $\kappa \leftarrow 0$ ▷ initialize counter κ
9: $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
10: **while** $(\mathbf{z}, \mathbf{h}) = \perp$ **do** ▷ rejection sampling loop
11: $\mathbf{y} \in R_q^\ell \leftarrow \text{ExpandMask}(\rho'', \kappa)$
12: $\mathbf{w} \leftarrow \text{NTT}^{-1}(\hat{\mathbf{A}} \circ \text{NTT}(\mathbf{y}))$
13: $\mathbf{w}_1 \leftarrow \text{HighBits}(\mathbf{w})$ ▷ signer's commitment
14: ▷ HighBits is applied componentwise (see explanatory text in Section 7.4)
15: $\tilde{c} \leftarrow \text{H}(\mu||\text{w1Encode}(\mathbf{w}_1), \lambda/4)$ ▷ commitment hash
16: $c \in R_q \leftarrow \text{SampleInBall}(\tilde{c})$ ▷ verifier's challenge
17: $\hat{c} \leftarrow \text{NTT}(c)$

```python
# 算法 2 ML-DSA.Sign(sk,M)
def Sign(sk,M,rnd):
    (p,K,tr,s1,s2,t0) = sk_decode(sk)
    s1_hat = [NTT(si) for si in s1]
    s2_hat = [NTT(si) for si in s2]
    t0_hat = [NTT(ti) for ti in t0]
    A_hat = ExpandA(p)
    u = tr + M
    u = SHAKE_256(u,512)
    p_prime = K + rnd + u
    p_prime = SHAKE_256(p_prime,512)
    ka = 0
    z = None
    h = None
    while z == None and h == None:
        y = ExpandMask(p_prime,ka)
        y_hat = [NTT(yi) for yi in y]
        w = NTT_dot(A_hat,y_hat)
        w = [NTT_inv(wi) for wi in w]
        w1 = [HighBits(w1i) for w1i in w]
        w1 = w1Encode(w1)
        c_tilde = u + w1
        c_tilde = SHAKE_256(c_tilde,2*ML_DSA["lamda"])
        c = SampleInBall(c_tilde)
        print(c)
        c_hat = NTT(c)
```

Signing

```
message: DA5FBC7F78116BC8537E8E522DFBB6F68710FC36AE5AF0ACE5CFA3BA4FEBF6C86D966A44C2CF53CCC4FF5B9CC4E6403CFF3C23B4F292AAC01E35A21AB11F0707726E88048DD05887448B8

sk: 5AE5192442A0894AC775D84419BB7EDE9E8143A6E21CD709A47B58545EAEC7F59EAD08050900CBCF4093F9327CAD67B7600D9F8425084B6905BB6550F426F58B27B47C434BC194D466D83C14BF

rnd: 0000000000000000000000000000000000000000000000000000000000000000

rho: 5AE5192442A0894AC775D84419BB7EDE9E8143A6E21CD709A47B58545EAEC7F5

k: 9EAD08050900CBCF4093F9327CAD67B7600D9F8425084B6905BB6550F426F58B

tr: 27B47C434BC194D466D83C14BF5ABEA249E05287EC80D1B56BD0824E4EB84CE1CD4CCD0DBD1BDE59106AE10516E1150E841B03E759DFDD930F6C7BAC85753045

mu: A12A0CBDCEC8AEF5EC6AD85326CA82938BBEA6C371CA19083DC23BED8053940A544CA299C0CF8AF2DD7BDA21A995598DB01944BAEC75E78D9774652CB8CB3250

aHat: [[[6040986, 2350664, 1214746, 7632339, 2508550, 633397, 2663794, 7047179, 821075, 758878, 8358712, 1928899, 6060553, 760254, 7369556, 7140499, 5751989,
[7400627, 6870200, 1419581, 6282757, 2134937, 2354926, 8261572, 5849282, 5519182, 7542763, 5975707, 3266136, 3059571, 5539098, 5795874, 1179544, 4284364, 1201
[4853538, 5915247, 6459939, 623769, 866340, 1744787, 2650216, 1046178, 4635104, 1794943, 5661288, 8122745, 799868, 2108125, 3052542, 6100564, 2865788, 6089437
[3916585, 1206507, 7616051, 1490935, 7681477, 4390106, 2603657, 5730813, 7464249, 7237698, 3578248, 568065, 4982191, 6213955, 6384033, 3672923, 4717541, 33648
[[943959, 7939384, 3714210, 3779061, 3532517, 1733564, 3701928, 5174646, 8297884, 6102742, 4457757, 3421267, 6110238, 2334728, 340452, 5776822, 6474300, 31303
[5477485, 2611553, 1259502, 2429692, 2155400, 5304211, 108616, 7182226, 5890307, 6712102, 4890109, 3285941, 5670628, 4516456, 1881767, 1532264, 5785651, 36436
[7915225, 4356971, 5032330, 4384725, 7883036, 4028534, 8055029, 267544, 7542538, 6772860, 7559022, 5734317, 3926646, 4942056, 5690476, 685077, 2090582, 722504
[967844, 2541863, 1434278, 7791641, 6290694, 477058, 2117736, 5389582, 6432849, 2186234, 5524702, 1879624, 5388089, 3042210, 159604, 4293019, 3327343, 3827274
[[1250521, 4914661, 3956311, 304150, 1808420, 6170884, 4896688, 7569431, 6368685, 5599167, 6795675, 4292470, 2230677, 7633465, 2348413, 940116, 4197286, 51816
[7723786, 1986439, 5167384, 5129199, 5986250, 5843946, 2363491, 5431901, 5836706, 1548222, 5312317, 2608992, 4175032, 5499437, 2380595, 1418505, 1681347, 3361
[3975431, 4948822, 763025, 65789, 4414616, 6756589, 7422492, 4171291, 3197578, 4422295, 1518670, 8127993, 6088136, 2061394, 7026758, 3450526, 327492, 8192319,
[2372094, 5397346, 7224234, 4647690, 1380723, 8020559, 7654057, 3062121, 7027898, 2226550, 4801860, 7819970, 1451714, 5809247, 6771665, 7292397, 5633880, 5755
[[1286566, 1892857, 7600279, 2094394, 3830016, 1410580, 2187758, 1207715, 983845, 6450122, 5787642, 4170645, 5728402, 3347029, 4856373, 5635836, 4279081, 5025
[4597438, 1858914, 2048909, 149414, 4881059, 5550767, 3233540, 7590486, 6645948, 2271790, 4936024, 7436776, 1975151, 3365797, 8055158, 6369121, 5167870, 41332
[6415545, 5183767, 3793280, 3140822, 3501151, 4562966, 7366328, 5744523, 4336994, 5186210, 3928108, 1420493, 3675722, 7522454, 7842773, 6099531, 3645181, 3572
[7965062, 607644, 2309032, 2362594, 8260421, 7879312, 110959, 2350996, 702703, 7741031, 539149, 5773106, 95166, 3022158, 3317906, 1923911, 6167338, 3046735, 2
```

Signing

```
s1Hat: [[3780472, 8001483, 7484040, 4059895, 7647343, 3434819, 902292, 6727194, 2342222, 4671283, 2590107, 8356189, 3994258, 2362867, 101548
[7490143, 2416677, 6018412, 4886851, 4661549, 951995, 1217523, 200620, 4047296, 5752223, 5204483, 629955, 7868040, 5657793, 2519186, 7379324
[674988, 7034336, 4666570, 6284494, 4350353, 7495817, 2574693, 7634530, 6913604, 560773, 7663665, 2035513, 3454030, 33489, 5804374, 2874174,
[4406566, 5954670, 7130704, 1917294, 3030143, 7617234, 1177455, 3376554, 5350376, 3264614, 7417136, 2215991, 224662, 7250601, 4335013, 67614

s2Hat: [[3629686, 4323385, 1136763, 1892621, 396580, 4015673, 7506969, 413433, 3429836, 5013117, 5164433, 3162170, 3109721, 7122983, 2862134
[1691622, 622196, 3664387, 1888517, 6954472, 1664504, 4698479, 6040992, 3462971, 8143638, 1775563, 4512773, 4656917, 1689760, 1077062, 87430
[1613043, 4987276, 7287169, 6380375, 224343, 5098268, 1770771, 294873, 858281, 2279584, 6969794, 4732503, 6092736, 370529, 5329783, 1832892,
[1302014, 5489441, 37061, 4413680, 1025518, 6704923, 6704959, 5549932, 2780863, 1049739, 364807, 716450, 5247856, 63141, 3739196, 7655028, 4

t0Hat: [[610973, 5626726, 3451877, 7487350, 6292492, 6530640, 1903374, 7333475, 6327440, 7331451, 1406104, 1022202, 8154710, 6372860, 169334
[6938612, 1405233, 2812565, 2424669, 85312, 2801824, 4388851, 4016848, 956731, 3276110, 1543239, 2157804, 7522223, 3984456, 673385, 3865973,
[317716, 7952704, 5819693, 7775640, 6498344, 692925, 296983, 3947939, 2170031, 2731242, 991907, 4337667, 7009311, 5014777, 447037, 586519, 4
[1570404, 3374127, 6203829, 3403877, 8157306, 6508111, 4782724, 2219282, 4184276, 4852824, 3605792, 1342862, 1287197, 8372404, 5693597, 3661

rhoPrime: A9A9F85C9C20C2B207E25FDC800A2E59F41FB874B7A5DB2CA80C3CD2F6FD1DD6D8F945C176A75CCDEE86D21C830C4164C2386D38968EC0F58ECAA2E6C5193BC7
```

Signing

```
y: [[-54016, 57151, 7319, -1221, -121367, -8268, 75887, -124779, 12220, -96893, 110461, -24630, 87493, -2004, 23040, 73278, 103663, -34055, -40061, 11565, 13203, 99130, -
[-29248, 56187, -92953, -5383, 29593, 10080, -55123, 90644, -91145, -64817, 2370, -92112, 22362, 55467, 61858, 104855, -3696, -32105, -94001, 98242, -33037, -85356, 11368
[-13691, 89927, -101188, -38894, -4441, -25708, 23843, -97024, 13762, 25626, 127025, 116145, 89972, -76945, -127740, 72856, -24798, -101731, 120929, -4575, 97409, -31376,
[122411, -114569, 126096, -15602, 60730, -85979, -92801, -96588, -90323, 39522, 9459, 48319, -114916, 32791, 59968, -89000, 124900, 35073, -80370, -73021, 24611, -74589,

NTT(y): [[3035491, 2809772, 6725498, 567655, 5447090, 6430497, 588219, 6597274, 7259775, 1642189, 5313552, 2601666, 2944974, 6295297, 3581905, 6874192, 210561, 5960622, 8
[1186428, 6627972, 5529664, 451697, 4448725, 4380951, 4173214, 3118588, 751528, 5748926, 6051368, 6781590, 953583, 8107394, 4197755, 1182276, 4062055, 2844100, 4363510, 3
[3532585, 4924697, 1037304, 5507705, 7307075, 5989549, 8042933, 4279280, 4889520, 6020792, 857658, 7123262, 1696421, 841316, 2868754, 3895707, 5482347, 5322601, 7095490,
[5891005, 3311385, 5414291, 4580860, 8340878, 5520282, 6364105, 7045576, 7815361, 810465, 7786775, 3706582, 6136129, 7748501, 1457512, 4498603, 7106653, 243797, 2818566,

aHat * NTT(y): [[6144420, 4965121, 7555555, 3905658, 6043149, 7189003, 8075652, 2203037, 2783350, 3674236, 2430017, 5393384, 4941164, 4296081, 6148283, 7660585, 2308491,
[3947975, 3650816, 8224516, 4685363, 3926342, 144489, 3132631, 907258, 6824604, 5556910, 2181192, 7208281, 4569563, 3179171, 1144616, 5741696, 1144035, 2523799, 8272894,
[4570944, 627762, 5756578, 4675500, 836083, 5598067, 5641604, 5537951, 5805924, 1180617, 3380506, 4345096, 5582818, 5533245, 1968659, 8240317, 2532568, 7335666, 7945304,
[2205174, 538673, 4505981, 4879551, 3213359, 1731252, 2120126, 3363074, 3771584, 5588762, 3902236, 1437007, 3268368, 897382, 6649813, 3210041, 351148, 6592297, 4212276, 1

w = NTTInverse(aHat * NTT(y)): [[4968407, 4718179, 462388, 1119871, 7817329, 3820487, 6327751, 947481, 6639094, 4489022, 8186388, 1912045, 4956263, 7739800, 7404478, 8379
[3630454, 5624929, 2246792, 132343, 4322521, 4071699, 2673323, 4672357, 5664089, 578958, 6719380, 5325833, 3539196, 1512492, 8373427, 51420, 3989611, 8346944, 3406196, 61
[4180087, 6172391, 4813829, 1863491, 1606438, 784736, 7811099, 2734334, 1234685, 3971486, 6958897, 3641168, 448494, 7052068, 206469, 5585204, 2415087, 2675390, 323449, 48
[3075011, 1782742, 3609275, 2285451, 5721724, 1975868, 4541321, 7349816, 1822499, 3892614, 2365141, 6626613, 7806763, 5523919, 1101957, 5963548, 2099901, 3623760, 7831180

w1: [[26, 25, 2, 6, 41, 20, 33, 5, 35, 24, 43, 10, 26, 41, 39, 0, 34, 37, 1, 37, 20, 5, 15, 0, 42, 18, 29, 32, 31, 24, 10, 40, 18, 13, 17, 32, 11, 22, 40, 43, 22, 14, 24,
[19, 30, 12, 1, 23, 21, 14, 25, 30, 3, 35, 28, 19, 8, 0, 0, 21, 0, 18, 32, 11, 5, 41, 4, 24, 3, 26, 18, 19, 3, 39, 38, 2, 14, 12, 19, 23, 39, 40, 34, 28, 11, 8, 5, 21, 35
[22, 32, 25, 10, 8, 4, 41, 14, 6, 21, 37, 19, 2, 37, 1, 29, 13, 14, 2, 25, 10, 29, 42, 16, 21, 35, 42, 21, 41, 17, 27, 10, 13, 41, 3, 33, 3, 41, 17, 27, 11, 10, 14, 26, 3
[16, 9, 19, 12, 30, 10, 24, 39, 10, 20, 12, 35, 41, 29, 6, 31, 11, 19, 41, 24, 22, 12, 16, 5, 24, 0, 5, 42, 2, 20, 2, 10, 32, 5, 30, 20, 2, 2, 20, 4, 10, 37, 14, 24, 6, 6

w1Encode: 5A261829151623B62A5A7A0262199454F100AAD4811FA6A05213818B85AE96833959781E65B126E056866349665AD05020795E81443552C8308B9115CE7AA998A86EA8018CDAC43817B18D94E800E8C1

cTilde: 3202542EF1E239D32BE1BCE5AE4AC8052D578899D653E368E11BC11C5480BA06

c: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 1,

cHat: [3919627, 1297980, 1398134, 6081972, 7171056, 414117, 5281780, 4348975, 7918931, 7431142, 1215943, 8115251, 2501891, 2979933, 1056051, 4629893, 4230848, 5517032, 50
```

Signing

18: $\langle\langle c\mathbf{s}_1\rangle\rangle \leftarrow \text{NTT}^{-1}(\hat{c} \circ \hat{\mathbf{s}}_1)$
19: $\langle\langle c\mathbf{s}_2\rangle\rangle \leftarrow \text{NTT}^{-1}(\hat{c} \circ \hat{\mathbf{s}}_2)$
20: $\mathbf{z} \leftarrow \mathbf{y} + \langle\langle c\mathbf{s}_1\rangle\rangle$ ▷ signer's response
21: $\mathbf{r}_0 \leftarrow \text{LowBits}(\mathbf{w} - \langle\langle c\mathbf{s}_2\rangle\rangle)$
22: ▷ LowBits is applied componentwise (see explanatory text in Section 7.4)
23: **if** $||\mathbf{z}||_\infty \geq \gamma_1 - \beta$ **or** $||\mathbf{r}_0||_\infty \geq \gamma_2 - \beta$ **then** $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$ ▷ validity checks
24: **else**
25: $\langle\langle c\mathbf{t}_0\rangle\rangle \leftarrow \text{NTT}^{-1}(\hat{c} \circ \hat{\mathbf{t}}_0)$
26: $\mathbf{h} \leftarrow \text{MakeHint}(-\langle\langle c\mathbf{t}_0\rangle\rangle, \mathbf{w} - \langle\langle c\mathbf{s}_2\rangle\rangle + \langle\langle c\mathbf{t}_0\rangle\rangle)$ ▷ Signer's hint
27: ▷ MakeHint is applied componentwise (see explanatory text in Section 7.4)
28: **if** $||\langle\langle c\mathbf{t}_0\rangle\rangle||_\infty \geq \gamma_2$ **or** the number of 1's in $\mathbf{h}$ is greater than ω, **then** $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
29: **end if**
30: **end if**
31: $\kappa \leftarrow \kappa + \ell$ ▷ increment counter
32: **end while**
33: $\sigma \leftarrow \text{sigEncode}(\tilde{c}, \mathbf{z} \bmod^{\pm} q, \mathbf{h})$
34: **return** σ

```
100         cs1 = NTT_dot_l(s1_hat,c_hat)
101         cs1 = [NTT_inv(csi) for csi in cs1]
102         cs2 = NTT_dot_k(s2_hat,c_hat)
103         cs2 = [NTT_inv(csi) for csi in cs2]
104         z = array_plus_l(y,cs1)
105         temp = array_minus_k(w,cs2)
106         r0 = [LowBits(w1i) for w1i in temp]
107         if (infinity_norm(z) >= ML_DSA["gamma_1"] - ML_DSA["beta"] or
108             infinity_norm(r0) >= ML_DSA["gamma_2"] - ML_DSA["beta"]):
109             z = None
110             h = None
111         else:
112             ct0 = NTT_dot_k(t0_hat,c_hat)
113             ct0 = [NTT_inv(cti) for cti in ct0]
114             zero_array = [[0]*256] * ML_DSA["k"]
115             w_minus_cs2 = array_minus_k(w,cs2)
116             w_minus_cs2_pluse_ct0 = array_plus_k(w_minus_cs2,ct0)
117             minus_ct0 = array_minus_k(zero_array,ct0)
118             h,true_num = MakeHint(minus_ct0, w_minus_cs2_pluse_ct0)
119             if (infinity_norm(c_tilde) >= ML_DSA["gamma_2"] or
120                 true_num > ML_DSA["omega"]):
121                 z = None
122                 h = None
123         ka = ka + ML_DSA["l"]
124     z_mod = []
125     for i in range(ML_DSA["l"]):
126         z_temp = []
127         for j in range(256):
128             z_temp.append(mod_pm(z[i][j]))
129         z_mod.append(z_temp)
130     Sigma = sigEncode(c_tilde,z_mod,h)
131     return Sigma
```

Signing

```
cHat: [3919627, 1297980, 1398134, 6081972, 7171056, 414117, 5281780, 4348975, 7918931, 7431142, 1215943, 8115251, 2501891, 2979933, 1056051, 4629893, 4230848, 5517032, 50

cs1: [[2, 8380399, 8380416, 8380402, 8380407, 0, 1, 8380415, 6, 8380415, 9, 8380408, 0, 24, 7, 4, 5, 2, 8380406, 8380412, 6, 8, 3, 8380414, 8, 8380408, 8380414, 13, 83804
[8380407, 12, 8380404, 0, 8380416, 8380413, 1, 8380403, 20, 8380416, 9, 1, 4, 8380414, 20, 8, 8380406, 13, 8380413, 8, 8380402, 8380410, 3, 8380414, 8380403, 5, 8380410,
[8, 8380413, 8380415, 3, 2, 2, 8380413, 8380403, 19, 8380400, 1, 10, 8380403, 8380410, 9, 6, 7, 8380408, 4, 27, 8380399, 5, 1, 8380406, 21, 8380410, 8380406, 9, 9, 838040
[8380411, 18, 8380409, 8380408, 8380405, 8380410, 8380402, 8380402, 8380406, 3, 8380408, 8380407, 0, 8380410, 3, 22, 8380407, 8, 10, 2, 4, 5, 9, 6, 13, 16, 0, 2, 10, 8380

cs2: [[7, 13, 8380415, 8380414, 8380406, 8380415, 5, 8380415, 8380415, 18, 8380405, 8380409, 8380415, 8380392, 5, 0, 2, 8380410, 8380407, 8, 4, 1, 11, 12, 8380406, 10, 83
[4, 0, 8380413, 1, 7, 1, 8380414, 1, 8380413, 8380404, 8380414, 3, 8380412, 1, 0, 8380407, 8, 2, 6, 4, 4, 8380407, 8380410, 8380411, 9, 6, 11, 0, 8380415, 8380405, 838040
[8380401, 8380402, 8380411, 4, 4, 8380413, 5, 9, 0, 2, 8380413, 8380412, 8380415, 9, 3, 8380415, 8380394, 7, 3, 8380413, 8380406, 8380416, 6, 8380409, 8380414, 9, 13, 12,
[8380416, 8380411, 8380408, 8, 10, 9, 8380413, 8380407, 8380406, 8380411, 0, 26, 9, 5, 8380414, 0, 11, 1, 8380406, 3, 8380406, 8380411, 8380413, 8380402, 8380415, 8380415

z: [[8326403, 57133, 7318, 8379181, 8259040, 8372149, 75888, 8255636, 12226, 8283522, 110470, 8355778, 87493, 8378437, 23047, 73282, 103668, 8346364, 8340345, 11560, 1320
[8351159, 56199, 8287451, 8375034, 29592, 10076, 8325295, 90630, 8289292, 8315599, 2379, 8288306, 22366, 55464, 61878, 104863, 8376710, 8348325, 8286412, 98250, 8347365,
[8366734, 89923, 8279227, 8341526, 8375978, 8354711, 23839, 8283379, 13781, 25609, 127026, 116155, 89958, 8303465, 8252686, 72862, 8355626, 8278677, 120933, 8375869, 9739
[122405, 8265866, 126088, 8364806, 60718, 8294431, 8287601, 8283814, 8290083, 39525, 9450, 48309, 8265501, 32784, 59971, 8291439, 124890, 35081, 8300057, 8307398, 24615,

||z||: 130985, ||z|| check passed

r0: [[16336, -43434, 81462, -22910, 8316, 11209, 42434, -4837, -27144, -82132, -3552, 7413, 4201, -69199, -23623, -591, 50423, 27202, -43436, -94230, 1484, -49886, -81705
[11634, -88991, -38772, -58122, -58158, 71954, 6830, -89244, -49827, 7579, 53143, -7162, -79615, -11221, -6990, 51430, -10141, -33475, -22162, 60211, 49437, -7706, -15665
[-10105, 77558, 52235, -41153, 82722, 22884, 2070, 67829, 91901, -28260, -88267, 22357, 67568, 4891, 16002, 61750, -60922, 8887, -57482, 50502, -44950, 61868, 67963, -880
[27588, 68572, -9532, -125, 7794, 71219, -29811, -78270, -82130, 83340, 79573, -39653, -2270, 458, -40824, 59164, 4786, 4943, 22167, 77284, 21272, 9424, 75396, -65313, 26

||r0||95002, ||r0|| check passed

cHat * t0Hat: [[6685568, 3625903, 5012222, 7942920, 1902157, 4674810, 4210018, 6254152, 5621555, 4432127, 3161400, 2991916, 3326774, 463020, 6499212, 7545109, 1598396, 89
[4777632, 90958, 8065217, 584878, 6688472, 5835341, 8057752, 7925126, 4307379, 1526365, 6347656, 5095628, 5273048, 3594180, 237683, 4235200, 2183933, 4287717, 2230703, 57
[6626149, 6184842, 4559639, 3664809, 1188987, 6544145, 2698182, 8118771, 4144600, 140242, 1139078, 2045111, 6709581, 8048885, 8220443, 5311540, 2355211, 658576, 853096, 3
[6774059, 2102179, 2377584, 6403672, 5836339, 2437038, 3875612, 1383305, 5749753, 439798, 4497464, 2573238, 8325184, 5985321, 5879206, 7991253, 3954615, 4532081, 7378153,
```

Signing

```
ct0: [[8373033, 8364566, 3534, 8346680, 8373079, 8358277, 11564, 24947, 21052, 4504, 8380387, 8379636, 8372277, 8368576, 8371438, 8379841, 13012, 8369219, 8377262, 18148, 14402, 14008, 1570,
[8350538, 8376421, 8363, 14916, 8356087, 15252, 8372461, 8363779, 32127, 8368940, 29451, 8372748, 4924, 18150, 5450, 8368354, 8373227, 8369870, 21516, 31989, 8354226, 24283, 5733, 8363948, 8
[8352700, 9310, 7029, 3298, 8354288, 8368751, 8376483, 8376560, 8375645, 8161, 14353, 8378176, 17744, 2312, 8356471, 8370054, 8378039, 8380024, 19885, 8342, 18543, 2468, 8371682, 8373492, 11
[23885, 9067, 13870, 8374717, 8375006, 8363100, 8354061, 3065, 8372238, 8374739, 8353195, 8372532, 8367737, 8369738, 4557, 8378531, 22363, 8375058, 8369175, 8359640, 8369253, 8373150, 12381,

-ct0: [[7384, 15851, 8376883, 33737, 7338, 22140, 8368853, 8355470, 8359365, 8375913, 30, 781, 8140, 11841, 8979, 576, 8367405, 11198, 3155, 8362269, 8366015, 8366409, 8378847, 22163, 835620
[29879, 3996, 8372054, 8365501, 24330, 8365165, 7956, 16638, 8348290, 11477, 8350966, 7669, 8375493, 8362267, 8374967, 12063, 7190, 10547, 8358901, 8348428, 26191, 8356134, 8374684, 16469, 1
[27717, 8371107, 8373388, 8377119, 26129, 11666, 3934, 3857, 4772, 8372256, 8366064, 2241, 8362673, 8378105, 23946, 10363, 2378, 393, 8360532, 8372075, 8361874, 8377949, 8735, 6925, 8369294,
[8356532, 8371350, 8366547, 5700, 5411, 17317, 26356, 8377352, 8179, 5678, 27222, 7885, 12680, 10679, 8375860, 1886, 8358054, 5359, 11242, 20777, 11164, 7267, 8368036, 1879, 8354153, 8376689

w - cs2 + ct0: [[4961016, 4702315, 465924, 1086137, 7810002, 3798349, 6339310, 972430, 6660148, 4493508, 8186370, 1911272, 4948125, 7727984, 7395494, 8379250, 6539211, 7063172, 143873, 69710
[3600571, 5620933, 2255159, 147258, 4298184, 4086950, 2665370, 4655718, 5696220, 567494, 6748834, 5318161, 3544125, 1530641, 8378877, 39367, 3982413, 8336395, 3427706, 6187048, 2118350, 9688
[4152386, 6181716, 4820864, 1866785, 1580305, 773074, 7807160, 2730468, 1229913, 3979645, 6973254, 3638932, 466240, 7054371, 182520, 5574843, 2412732, 2674990, 343331, 4820444, 1878233, 5587
[3098897, 1791815, 3623154, 2279743, 5716303, 1958542, 4514969, 7352891, 1814331, 3886942, 2337919, 6618702, 7794074, 5513235, 1106517, 5961662, 2122253, 3618400, 7819949, 4627643, 4200316,

h: [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

||h||: 62, Hint check passed

||ct0||: 46263, ||ct0|| check passed

signature: 3202542EF1E239D32BE1BCE5AE4AC8052D578899D653E368E11BC11C5480BA06FED24E83A4361E358121DA338108794DDBF93ED0FDE9AD07C50F983BAAF01E985F9A6F380C6B14148AC829B67467CCF9F2A16D2594DB895BBE6
```

Verification

Algorithm 8 ML-DSA.Verify_internal(pk, M', σ)

Internal function to verify a signature σ for a formatted message M'.

Input: Public key $pk \in \mathbb{B}^{32+32k(\text{bitlen }(q-1)-d)}$ and message $M' \in \{0,1\}^*$

Input: Signature $\sigma \in \mathbb{B}^{\lambda/4+\ell\cdot 32\cdot(1+\text{bitlen }(\gamma_1-1))+\omega+k}$.

Output: Boolean

1: $(\rho, \mathbf{t}_1) \leftarrow$ pkDecode(pk)
2: $(\tilde{c}, \mathbf{z}, \mathbf{h}) \leftarrow$ sigDecode(σ) ▷ signer's commitment hash $\tilde{c}$,
3: **if** $\mathbf{h} = \perp$ **then return** false ▷ hint was
4: **end if**
5: $\hat{\mathbf{A}} \leftarrow$ ExpandA(ρ) ▷ $\mathbf{A}$ is generated and stored in NT
6: $tr \leftarrow$ H$(pk, 64)$
7: $\mu \leftarrow$ (H(BytesToBits$(tr)||M', 64)$) ▷ message representative computed in a different cryptographic module
8: $c \in R_q \leftarrow$ SampleInBall$(\tilde{c})$ ▷ compute ver
9: $\mathbf{w}'_{\text{Approx}} \leftarrow \text{NTT}^{-1}(\hat{\mathbf{A}} \circ \text{NTT}(\mathbf{z}) - \text{NTT}(c) \circ \text{NTT}(\mathbf{t}_1 \cdot 2^d))$ ▷ w
10: $\mathbf{w}'_1 \leftarrow$ UseHint$(\mathbf{h}, \mathbf{w}'_{\text{Approx}})$ ▷ reconstruction c
11: ▷ UseHint is applied componentwise (see explanat
12: $\tilde{c}' \leftarrow$ H$(\mu||\text{w1Encode}(\mathbf{w}'_1), \lambda/4)$ ▷ hash
13: **return** $[[\, ||\mathbf{z}||_\infty < \gamma_1 - \beta]]$ **and** $[[\tilde{c} = \tilde{c}']]$

```python
def Ver(pk,M,signature):
    rho,t1 = pk_decode(pk)
    c_tilde,z,h = sigDecode(signature)
    if h == None:
        return False
    A_hat = ExpandA(rho)
    tr = SHAKE_256(pk,512)
    mu = SHAKE_256(tr + M,512)
    c = SampleInBall(c_tilde)
    z_hat = [NTT(zi) for zi in z]
    Ah_d_zh = NTT_dot(A_hat,z_hat)
    c_hat = NTT(c)
    t1_hat = [NTT(ti) for ti in t1]
    for i in range(ML_DSA["k"]):
        for j in range(256):
            t1_hat[i][j] = (t1_hat[i][j] * (2**ML_DSA["d"])) % ML_DSA["q"]
    ch_d_t12d = NTT_dot_k(t1_hat,c_hat)
    w_prime_approx = array_minus_k(Ah_d_zh,ch_d_t12d)
    w_prime_approx = [NTT_inv(wi) for wi in w_prime_approx]
    w1_prime = []
    for i in range(ML_DSA["k"]):
        w1_prime_temp = []
        for j in range(256):
            w1_prime_temp.append(UseHint(h[i][j],w_prime_approx[i][j]))
        w1_prime.append(w1_prime_temp)
    w1En = w1Encode(w1_prime)
    c_prime_tilde = SHAKE_256(mu + w1En,2 * ML_DSA["lamda"])
    return ((infinity_norm(z) < (ML_DSA["gamma_1"] - ML_DSA["beta"]))
            and (c_prime_tilde == c_tilde))
```

Verification

```
pk: 5B003CBFAF3E5166A85F8A45B9C1A4533FF216FB226CFEB83A81A20EE6E97E540FE2E3C6E44262A8C344330126E881551371383EA34EA2ADEDAD1185908B34905B09FC1E1304B

signature: E98901A3F79293983D935DCF3A4DC9BA8966F70CB2991E6E1E5942643D37A1523FA43A15CC894A81285C4BD0E5063267D317BD1EA3E3A2F0AEA6BFAADF074926F5E522

message: DBAEDE95F7793725C9DB980AE6544EB2E2C4FC165C28A12B6EE675764F020C01C048BD0DC8064612E4B6858FB6871F71D104ECC4AA0FB27B9B79D1D95EF34E1072743826

rho: 5B003CBFAF3E5166A85F8A45B9C1A4533FF216FB226CFEB83A81A20EE6E97E54

t1: [[527, 248, 110, 915, 578, 536, 58, 275, 307, 384, 642, 519, 853, 68, 903, 248, 675, 147, 730, 950, 429, 324, 264, 558, 52, 740, 149, 1008, 7
[162, 758, 880, 2, 918, 485, 1002, 459, 413, 240, 274, 123, 617, 23, 6, 147, 509, 332, 925, 319, 92, 399, 890, 426, 787, 237, 1013, 921, 320, 971
[97, 407, 842, 605, 22, 954, 860, 435, 687, 58, 207, 829, 619, 992, 996, 166, 348, 149, 380, 603, 938, 617, 713, 292, 196, 868, 867, 168, 693, 64
[70, 437, 1, 712, 755, 398, 792, 710, 14, 928, 914, 735, 452, 631, 849, 605, 778, 740, 192, 716, 899, 120, 633, 161, 718, 775, 658, 565, 689, 38,

cTilde: E98901A3F79293983D935DCF3A4DC9BA8966F70CB2991E6E1E5942643D37A152

z: [[-42047, -66894, 87908, 89595, 111780, 18060, -29472, 106675, -73405, 83736, -61194, -65178, -122794, 28095, -24402, 110453, -15, -102450, -2
[123647, -70330, -93442, 60681, -86285, -51894, 49194, -44950, 57114, -17199, -58463, -71488, 4364, 53766, -64258, -25477, 59878, 47821, 109235,
[57016, 59853, 117938, 25787, 49505, 116013, -108741, -48313, 71670, 29923, -97360, 116408, 79617, -129957, -42369, 78665, 25762, -8610, -119751,
[-40359, -101301, 17776, -86009, -105279, 96534, -108073, -94116, -86484, -119434, 80204, -39970, -122636, 107842, 51703, 11226, 129241, -4976, 6

h: [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,

Proper number of hints provided. Provided: 63, expected: <=80
||z||: 130971, ||z|| check passed
```

Verification

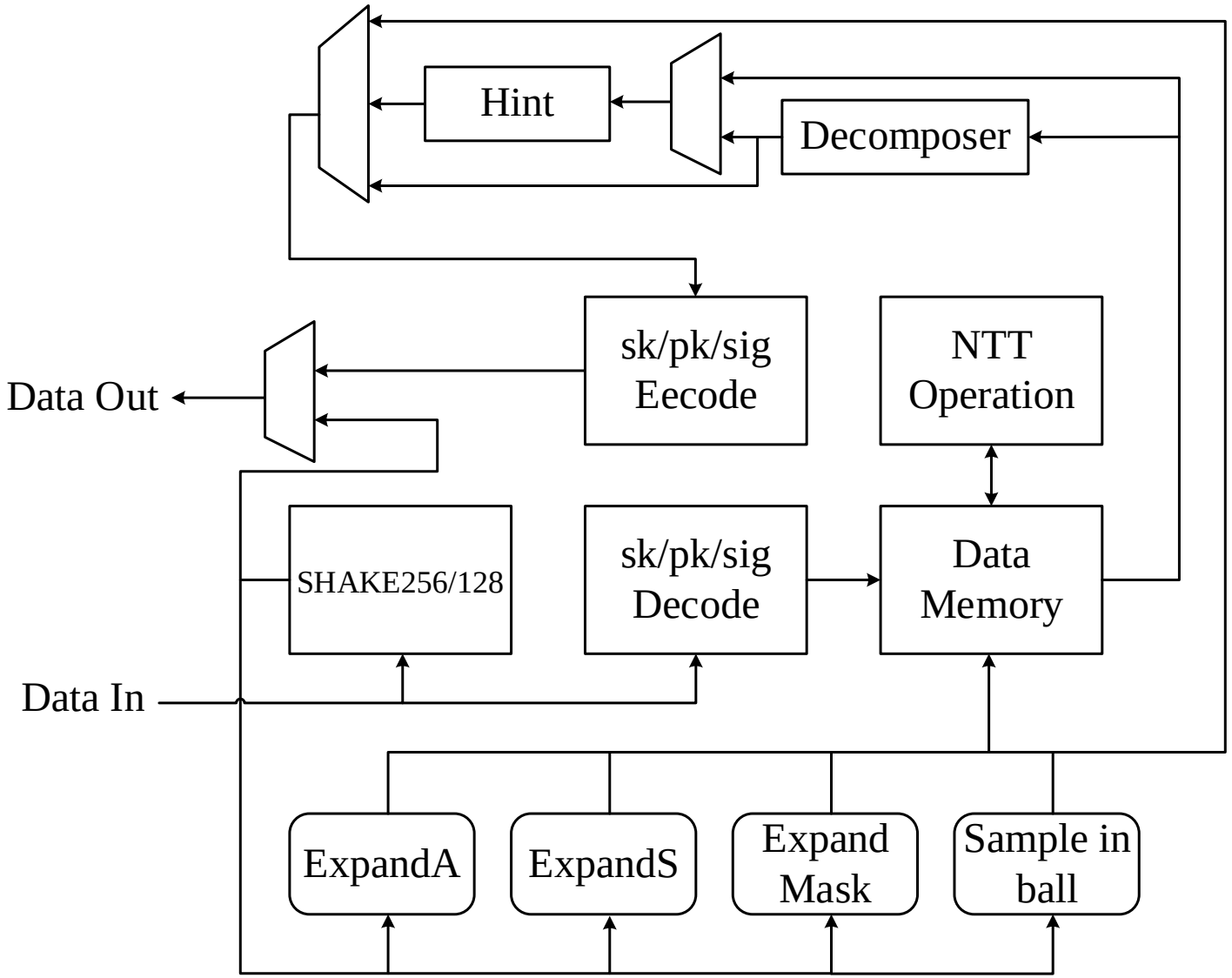
```
aHat: [[[2758750, 2429991, 2498833, 2173545, 8204600, 6778139, 1183134, 8086046, 462450, 3640781, 4260818, 97547, 7105425, 8135912, 7715776, 7341256, 5370857, 7268194, 27561
[8332077, 8005590, 7055174, 5233073, 1846710, 2730459, 2948961, 2191187, 6818952, 2605391, 5681870, 2710406, 112916, 7898262, 2127584, 3443826, 658735, 8338831, 6225691, 343
[7009549, 7600667, 4775169, 2160667, 3600757, 7091565, 7827571, 8288451, 1920413, 3246206, 4121206, 3881033, 1984606, 2323597, 6349627, 7370082, 4190371, 6857049, 4568562, 2
[872863, 5061949, 3640109, 7504018, 2680244, 1574485, 3742144, 7401748, 3343495, 4529782, 1796863, 2509379, 5903433, 5714205, 6234191, 3890754, 2361813, 1152933, 73849, 1713
[[2718495, 8283619, 1425067, 6373195, 4688230, 1587563, 766062, 2029025, 2510679, 6479173, 2954143, 1256883, 2265905, 910481, 2381083, 7433215, 5978644, 486739, 7878474, 395
[6654982, 539205, 6901414, 4306406, 4323334, 3921318, 7110591, 5152702, 3609791, 3010945, 4371365, 2139674, 5260311, 1197618, 6908818, 4549945, 7765268, 6757965, 6592512, 78
[1486140, 8149136, 3503405, 4506612, 3862405, 4828041, 3654089, 486418, 3552090, 4332863, 6321532, 5126500, 7984196, 3193681, 7249164, 2182493, 8140482, 5327361, 125662, 633
[1115279, 4220009, 8014196, 7256285, 6397743, 1824649, 2765309, 6428064, 6452502, 7926922, 5878030, 3571021, 5157849, 5077780, 4627997, 5592870, 3854561, 454813, 7267527, 61
[[6950338, 2115386, 6928349, 327570, 3902349, 298487, 5782285, 8201882, 4150826, 2132911, 5243966, 927144, 8320127, 425757, 1624704, 3722774, 1209442, 1612492, 8342370, 2664
[5852529, 919415, 5479291, 3781435, 6097879, 4997421, 7817984, 7656906, 5267745, 7345667, 6158925, 3188692, 2897107, 6262476, 2670056, 358741, 1579438, 854392, 5075932, 4042
[2689114, 309431, 5207970, 6047783, 4509433, 4055700, 7448908, 5426334, 5933344, 3849382, 6877084, 7636054, 4124963, 7221349, 5703021, 7101008, 5309984, 225313, 4813692, 265
[552833, 4429991, 8205740, 6871326, 6292004, 8174831, 1618170, 2844306, 3473762, 4973316, 3529343, 8149057, 5022386, 3969157, 20732, 4153278, 742133, 5162184, 4443546, 14279
[[4472096, 7236352, 4221145, 8347497, 4249314, 5913183, 8153235, 479277, 224134, 6261561, 3243881, 154304, 5821916, 6075114, 1111272, 7489574, 5320869, 5511034, 1234444, 449
[7352069, 7918539, 188906, 427145, 441022, 2537168, 6789776, 6358894, 2292569, 3987766, 7604470, 6499864, 8032114, 3174361, 2240912, 6694442, 222061, 2750671, 6652327, 81831
[8217507, 6872628, 6383570, 1673212, 8052859, 3354171, 2792420, 2904433, 2750832, 2213338, 6161318, 6768156, 4669374, 6370607, 7989007, 2396125, 3604793, 3874993, 4938805, 4
[4663440, 1941853, 5367071, 5014854, 4576328, 1084628, 472335, 5208558, 1100699, 6255907, 5606440, 1861394, 4814860, 2651636, 2945762, 7614198, 506607, 3481866, 333773, 5085
trCandidate: A9CBDF7C927F677A561B66AB5BCD6E1F267279AFCCA1576595E6BA6E4D1B225C95123962C9523C7276AF324071DFC205F88140E81C44199FE92964744959CE99
muCandidate: BA55E8D45110BBFC1C9A23DDB43B93AC4DC6EB68780D5A1CE0FDF50895E1AB0E9F1EBCD34FE2E952B9C1BB8DE721E892A741C1C055190AB43563BF73479AB937

c: [0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, -1, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

NTT(z): [[6959427, 6929453, 6113881, 4981143, 3545954, 6943164, 1647500, 2087619, 7374489, 5817057, 7760820, 6107494, 6733654, 3414907, 3084396, 6976092, 4524195, 1134448, 5
[1231705, 7495344, 2673873, 4296704, 5139723, 2188749, 3314440, 134207, 4164366, 1010512, 4356688, 247311, 5471383, 5312157, 3555880, 5253388, 8256502, 5169005, 4910956, 823
[4766893, 1072408, 4924305, 4440602, 5910429, 173222, 5081017, 3446850, 3377581, 738231, 972604, 7542340, 1934563, 6243789, 1833524, 7196375, 2121851, 3643719, 1938134, 5635
[4026856, 2807882, 1088349, 8124353, 276002, 2293700, 5377309, 2376131, 1418632, 5987554, 6752925, 7556578, 4355976, 3355939, 2487045, 6128779, 2551426, 3447835, 3449874, 56

aHat * NTT(z): [[8205148, 3682531, 6246060, 749942, 3179582, 6639911, 7079666, 4195088, 375904, 7358218, 7510083, 2890955, 2299967, 775515, 1123241, 4864387, 280980, 4284786
[198555, 3875383, 5473293, 1983473, 2018570, 7978287, 665150, 363524, 8164354, 6028711, 1364403, 4740059, 1911360, 7290277, 4965247, 4023333, 1408979, 7088204, 4279402, 8206
[4351705, 4966420, 3517333, 2886633, 3386198, 103045, 7302249, 56215, 7750504, 1780529, 6679151, 179266, 5238553, 5902511, 6542327, 5102217, 4161061, 6982302, 1335822, 30963
[3210973, 8123259, 4756226, 3262637, 1745097, 1913357, 5436248, 5333631, 41958, 3876465, 1247662, 6621239, 7970832, 4908562, 3270501, 595571, 8048017, 361522, 2507026, 10650

NTT(t1): [[5436534, 311483, 2358223, 1202317, 7183104, 878370, 7569206, 7369810, 3782687, 2912922, 4980011, 6733999, 6136647, 8007920, 4194504, 7002838, 4687700, 7823600, 11
[3763656, 1959721, 2047761, 7444383, 545239, 5638797, 7894902, 3442018, 1074138, 4235419, 2951792, 7501614, 3471012, 6896109, 3721834, 4434209, 1391206, 854334, 840506, 2947
[4795173, 1191984, 4600702, 2008942, 4768908, 709219, 6925263, 1124933, 1206531, 5896368, 4542004, 4450041, 6083689, 1515801, 7670763, 6133828, 3670136, 6276840, 101311, 781
[3242704, 3240698, 3667167, 7522279, 6853824, 2381178, 703273, 198737, 2468163, 5161041, 6060504, 5781074, 254902, 5675382, 5133585, 8276894, 5890375, 4732925, 534433, 80563
```

Verification

```
NTT(t1) * 2^d: [[2550590, 4021968, 1701631, 2390889, 5080211, 5209254, 230169, 959452, 5370255, 3609825, 380156, 5015114, 5671058, 7356781, 1667068,
[315809, 5535877, 6043695, 91027, 8216044, 166520, 3359195, 5288668, 8281063, 1626068, 3577019, 8004444, 8155840, 533931, 1307082, 4312850, 7772849,
[3042737, 1547123, 2215535, 6494293, 5770699, 2293067, 4711823, 5372853, 3390309, 6703485, 7425705, 8302339, 7620806, 6044215, 2523830, 7719061, 5198
[6689695, 7017377, 6017536, 1303367, 6112725, 5379817, 3865937, 2252606, 5625492, 44107, 2058460, 821741, 1433351, 6556245, 1395814, 6742118, 7891331

NTT(c): [6740534, 4045296, 2602173, 4057786, 3828614, 2205945, 8167143, 6593076, 3152783, 8270939, 5589630, 286975, 2981496, 6078868, 5695322, 482104

NTT(c) * (NTT(t1) * 2^d): [[5324147, 7803716, 2455124, 465449, 6859752, 7840043, 3419480, 72744, 7270387, 7283536, 1228177, 1426655, 7968572, 3683039
[3199419, 1831771, 7504950, 1206947, 6645593, 3523456, 4514483, 4681694, 1624444, 6525827, 4323945, 3017200, 3130274, 2468093, 3873057, 4396976, 6136
[1123697, 8024472, 6427826, 3687539, 7960317, 1884200, 3224802, 7766844, 5152293, 5746694, 3481117, 2801008, 5394943, 7829698, 328196, 6482414, 57723
[724829, 3485727, 3008317, 3761766, 3053529, 4855029, 3478807, 3537413, 1363367, 6754463, 2162978, 2569512, 3667282, 8314185, 5278827, 3359864, 29003

wPrimeApprox: [[7922246, 5265226, 5506507, 4646550, 2602037, 8056124, 4540058, 3683346, 947069, 5492323, 5072451, 2036902, 3275644, 360403, 1342966,
[4250175, 1134865, 2368947, 127674, 6593998, 5500651, 3684706, 2949331, 3638645, 5847886, 6124713, 2457891, 7691497, 4701413, 5167273, 4131020, 68017
[5225532, 3184459, 3593619, 145438, 169419, 1461937, 7782223, 5828487, 8182468, 623885, 3445292, 1969565, 3946194, 4361076, 3330933, 8225361, 6162009
[2027829, 2889169, 1412751, 1398701, 4746044, 3334015, 7728514, 1145068, 2677181, 5837222, 1559867, 4891268, 1196552, 1846487, 7882602, 128082, 32647

w1Prime: [[42, 28, 29, 24, 14, 42, 24, 19, 5, 29, 27, 11, 17, 2, 7, 28, 40, 7, 12, 36, 2, 11, 11, 35, 6, 7, 43, 22, 13, 6, 19, 5, 34, 2, 31, 40, 38,
[22, 6, 12, 1, 35, 29, 19, 16, 19, 31, 32, 13, 40, 25, 27, 22, 36, 41, 4, 30, 13, 30, 25, 18, 25, 41, 9, 13, 40, 14, 9, 9, 10, 29, 41, 36, 24, 29, 11
[28, 17, 19, 1, 1, 8, 41, 31, 43, 3, 18, 10, 21, 23, 17, 43, 32, 42, 8, 42, 9, 22, 27, 37, 21, 0, 42, 26, 0, 34, 9, 42, 22, 36, 18, 38, 23, 32, 35, 1
[11, 15, 7, 7, 25, 18, 41, 6, 14, 30, 8, 26, 6, 10, 41, 1, 17, 22, 33, 14, 4, 41, 12, 27, 6, 2, 25, 16, 20, 0, 13, 14, 12, 18, 22, 30, 1, 10, 7, 24,

w1EncodeTemp: 2AD7618E8A4D45B72D917070E8C190C2B28CC6B15A8D3115A2F0A1E6512111183D6700AD82A88D40E01D1156944E59880DB74C2BD4845A529194F170D2946CC1124A010

cTilde: E98901A3F79293983D935DCF3A4DC9BA8966F70CB2991E6E1E5942643D37A152

cTildePrime: E98901A3F79293983D935DCF3A4DC9BA8966F70CB2991E6E1E5942643D37A152

cTilde == cTildePrime, signature verified
```

Architecture

Block Diagram



NTT

NTT/INTT

- ✓ NTT is a variant of the Fast Fourier Transform (FFT) based on a finite field.
- ✓ It transforms polynomial multiplication into “pointwise multiplication," accelerating polynomial multiplication operations.
- ✓ Cooley-Tukey decomposition is used for NTT and Gentleman-Sande decomposition for INTT in a butterfly architecture.
- ✓ It adopts a Radix-2 Multi-path Delay Commutator (MDC) FFT, using 8 Butterfly Units (BU) to process data with N = 256.

NTT - Mathematical Derivation

- ✓ The Number Theoretic Transform (NTT) is defined as:

$$\hat{a}_j = \sum_{i=0}^{n-1} \psi^{2ij} a_i \mod q$$

Note:

$$\psi^{k+2n} = \psi^k$$
$$\psi^{k+n} = -\psi^k$$

- ✓ Using the Cooley-Tukey decomposition:

$$\hat{a}_j = \sum_{i=0}^{n/2-1} \psi^{4ij+2i} a_{2i} + \sum_{i=0}^{n/2-1} \psi^{4ij+2j+2i+1} a_{2i+1} \mod q$$

$$= \sum_{i=0}^{n/2-1} \psi^{4ij+2i} a_{2i} + \psi^{2j+1} \sum_{i=0}^{n/2-1} \psi^{4ij+2i} a_{2i+1} \mod q$$

- ✓ Define :

$$A_j = \sum_{i=0}^{n/2-1} \psi^{4ij+2i} a_{2i}, \quad B_j = \sum_{i=0}^{n/2-1} \psi^{4ij+2i} a_{2i+1}$$

- ✓ Thus, the transformed coefficients are:

$$\hat{a}_j = A_j + \psi^{2j+1} B_j \mod q$$
$$\hat{a}_{j+n/2} = A_j - \psi^{2j+1} B_j \mod q$$

INTT - Mathematical Derivation

- ✓ The Inverse Number Theoretic Transform (INTT) is given by:

$$\mathbf{a}_i = \sum_{j=0}^{n-1} \psi^{-(2i+1)j} \hat{a}_j \mod q$$

Note:

$$\psi^{k+2n} = \psi^k$$
$$\psi^{k+n} = -\psi^k$$

- ✓ Using the Gentleman-Sande decomposition:

$$\mathbf{a}_i = \left[\sum_{j=0}^{n/2-1} \psi^{-(2i+1)j} \hat{a}_j + \sum_{j=0}^{n/2-1} \psi^{-(2i+1)(j+n/2)} \hat{a}_{j+n/2} \right] \mod q$$

$$= \psi^{-i} \left[\sum_{j=0}^{n/2-1} \psi^{-2ij} \hat{a}_j + \sum_{j=0}^{n/2-1} \psi^{-2i(j+n/2)} \hat{a}_{j+n/2} \right] \mod q$$

- ✓ Define :

$$A_i = \sum_{j=0}^{n/2-1} \hat{a}_j \psi^{-4ij}, \quad B_i = \sum_{j=0}^{n/2-1} \hat{a}_{j+n/2} \psi^{-4ij}$$

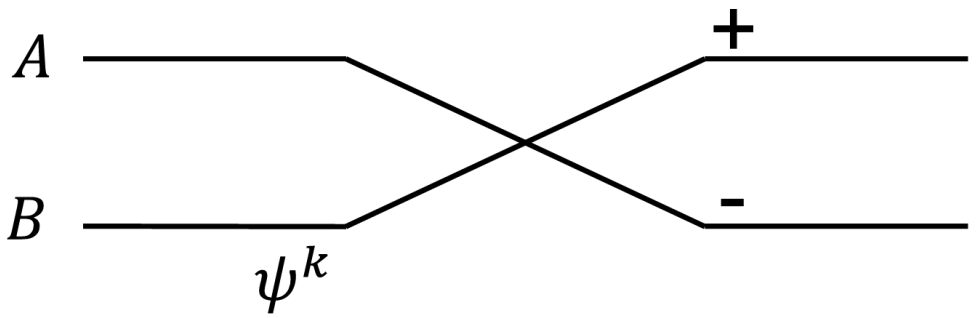
- ✓ Thus, the inverse transform coefficients are:

$$\mathbf{a}_{2i} = (A_i + B_i)\psi^{-2i} \mod q$$
$$\mathbf{a}_{2i+1} = (A_i - B_i)\psi^{-2i} \mod q$$

NTT/INTT - Butterfly diagram

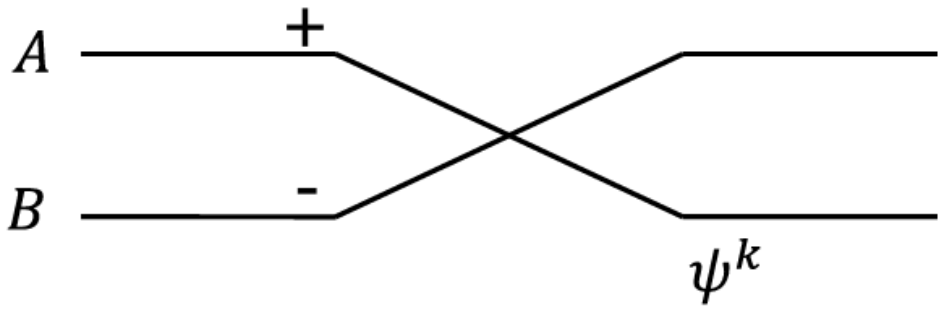
- ✓ Fundamental in FFT/NTT computations, is given by:

$$A = A + \psi^k B$$
$$B = A - \psi^k B$$

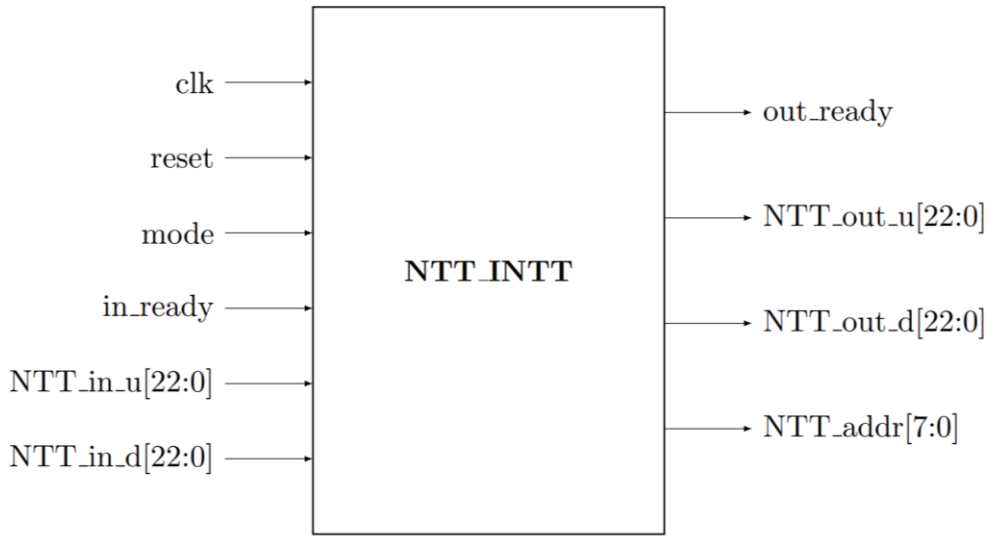


- ✓ Fundamental in FFT/NTT computations, is given by:

$$A = A + B$$
$$B = (A - B)\psi^{-k}$$



NTT_INTT - Module Pin Function Definition

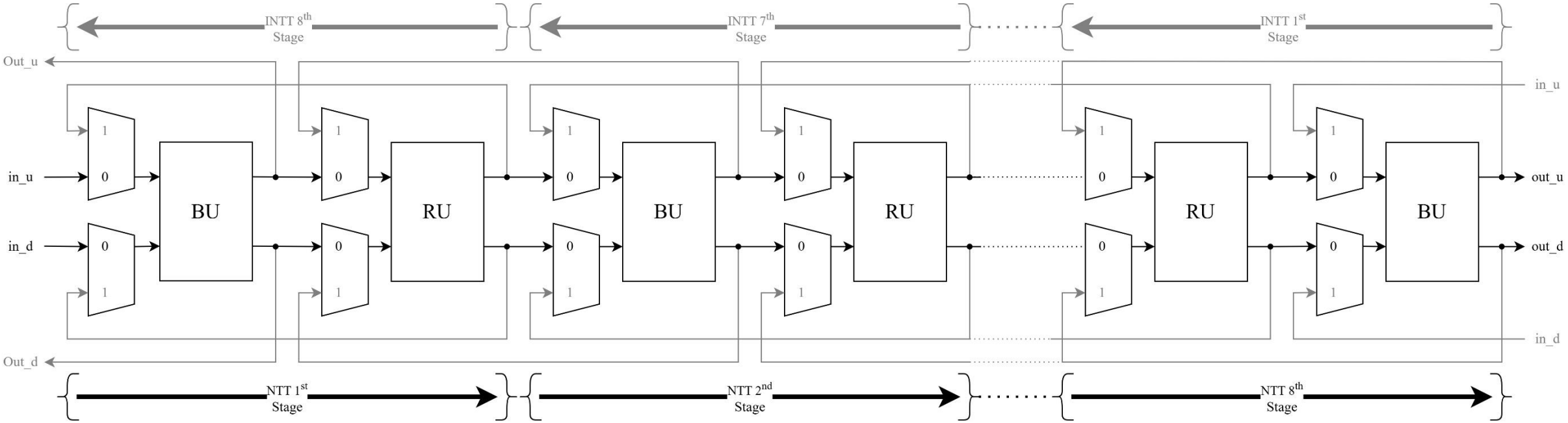


Pin name	I/O	Bit width	Function
clk	I	1	系統時脈
reset	I	1	系統同步重置信號
mode	I	1	0 : NTT mode / 1 : INTT mode
in_ready	I	1	由待轉換的MEM輸入資料準備好的指示信號
NTT_in_u	I	23	由待轉換的MEM輸入data
NTT_in_d	I	23	由待轉換的MEM輸入data
out_ready	O	1	轉換完成資料開始輸出的指示信號
NTT_out_u	O	23	轉換完成資料輸出data至指定MEM
NTT_out_d	O	23	轉換完成資料輸出data至指定MEM
NTT_addr	O	8	指定輸出的MEM位址

NTT_INTT - Module Task Assignment Table

Pin name	I/O	Bit length	Function
clk	I	1	系統時脈
reset	I	1	系統同步重置信號
mode	I	1	0 : NTT mode / 1 : INTT mode
in_ready	I	1	由待轉換的MEM輸入資料準備好的指示信號
NTT_in_u	I	23	由待轉換的MEM輸入data
NTT_in_d	I	23	由待轉換的MEM輸入data
out_ready	O	1	轉換完成資料開始輸出的指示信號
NTT_out_u	O	23	轉換完成資料輸出data至指定MEM
NTT_out_d	O	23	轉換完成資料輸出data至指定MEM
NTT_addr	O	8	指定輸出的MEM位址

NTT_INTT – Block Diagram

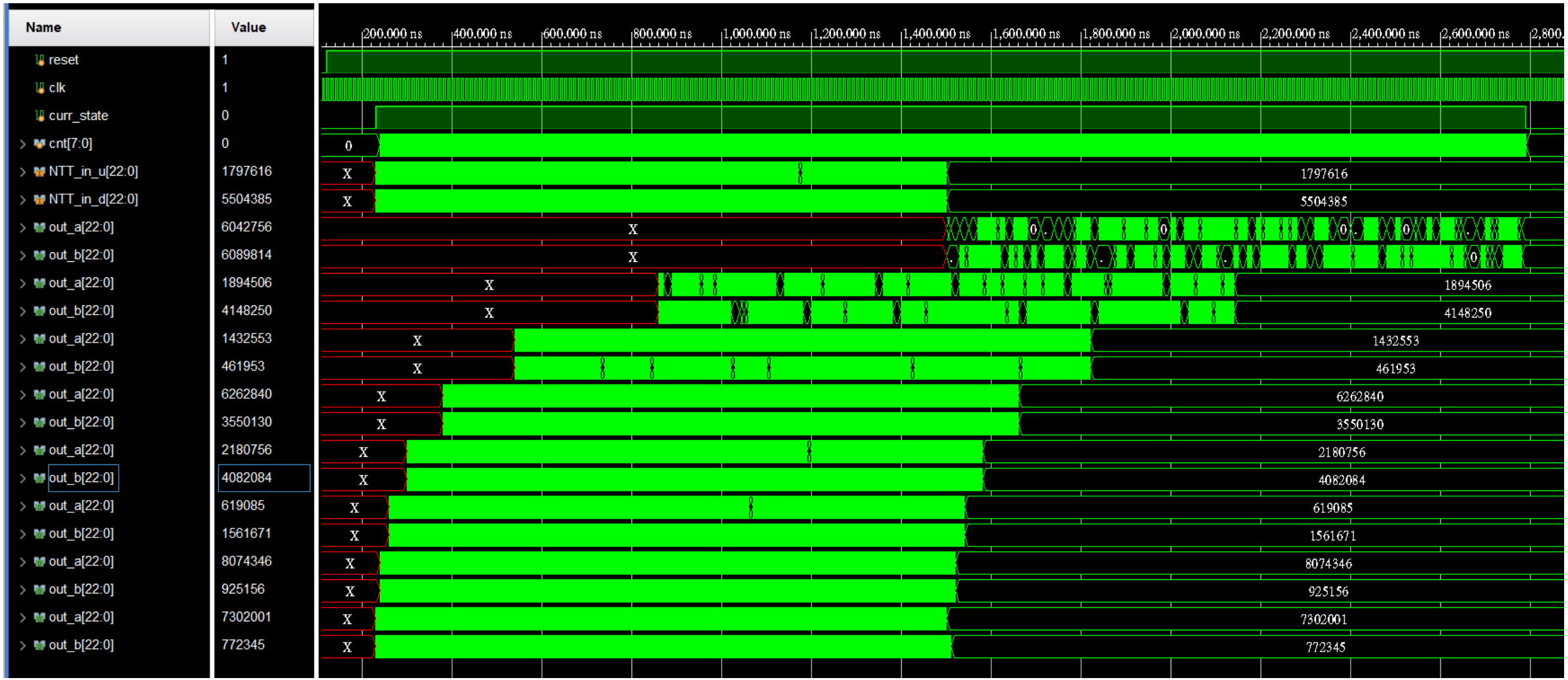


NTT – Timing Diagram

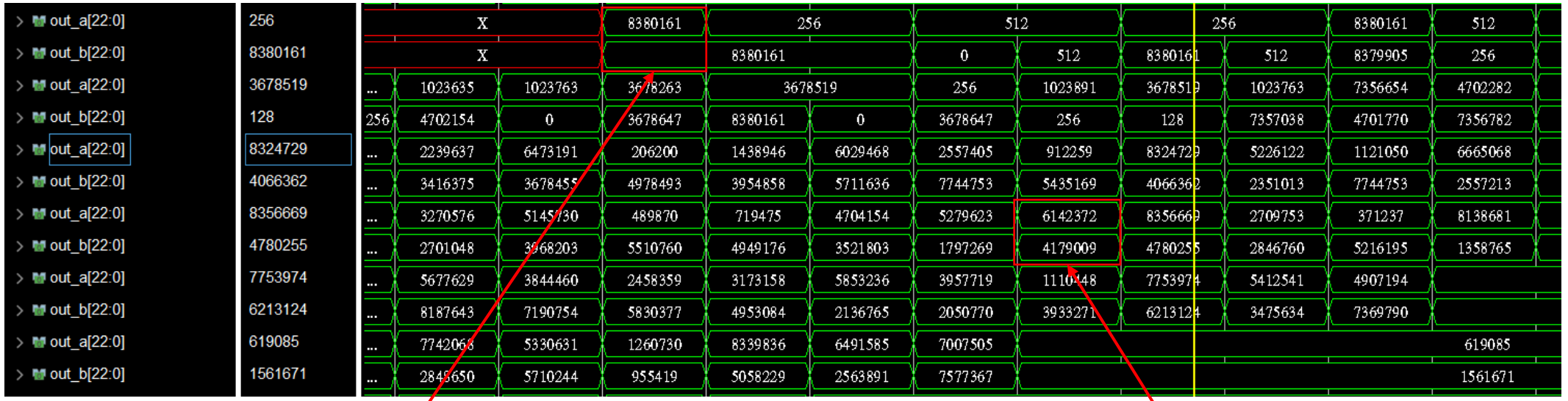
NTT - – Timing Diagram

5157967	8144313	5793821	5202208	1153275	2846100	3957586	3724921
1713881	8208803	3339230	2925445	2424097	7769886	614766	1691784
3162035	5357188	5215824	1368896	7067290	6755632	7118893	6953724
7669833	3774935	832905	2065634	2409715	899857	7254363	6880217
6641394	5558247	6436092	3879842	1575198	476033	3656372	1474475
4903989	1994723	4288883	7519296	4222356	5823871	2956723	193673
0	0	5223614	7662194	6608599	4509610	5031747	4300736
0	0	6917442	3547934	2905824	1225444	5940663	7276535
0	0	534788	7373421	1439604	627207	6967820	2051400
0	0	6170123	1867066	3095744	2358723	3793081	7466544
0	0	0	0	7483179	7841209	2168270	7466561
0	0	0	0	5524427	1571441	7132662	7833696
0	0	0	0	0	0	1797616	5504385
0	0	0	0	0	0	0	0

INTT – Timing Diagram



INTT – Timing Diagram



121	0	0	1023635	7356782	4258239	7992318	5864397	4518333
122	0	0	8380289	7356526	4010802	523968	607845	2328505
123	0	0	7356526	1023635	3940535	387971	3023220	4851779
124	0	0	1023763	256	8103694	7468350	2822564	2451780
125	0	0	1023635	4702154	2239637	3416375	3270576	2701048
126	0	0	1023763	0	6473191	3678455	5145730	3968203
127	8380161	8380161	3678263	3678647	206200	4978493	489870	5510760
128	256	8380161	3678519	8380161	1438946	3954858	719475	4949176
129	256	8380161	3678519	0	6029468	5711636	4704154	3521803
130	512	0	256	3678647	2557405	7744753	5279623	1797269
131	512	512	1023891	256	912259	5435169	6142372	4179009

Modular Reduction

- ✓ For a 46-bit value s, modular reduction is performed recursively by exploiting the relation :

$$2^{23} \equiv 2^{13} - 1 \; mod \; q$$

- ✓ The reduction ensures that the result falls within the interval $(-q, 2q)$ allowing for adjustments by adding q if negative or subtracting q if positive.
- ✓ After performing necessary additions or subtractions with q of the reduced result, the final output is determined by selecting the non-negative value.
- ✓ Each BU module contains one, and there are eight BU modules, requiring a total of eight modular reduction modules.

Modular Rduction - Mathematical Derivation

$$s[45:0] \equiv 2^{23}s[45:23] + s[22:0] \equiv 2^{13}s[45:23] - s[45:23] + s[22:0]$$

$$\equiv 2^{23}s[45:33] + 2^{13}s[32:23] - s[45:23] + z$$

$$\equiv 2^{13}\,(s[45:33] + s[32:23]) - (s[45:33] + s[45:23]) + z$$

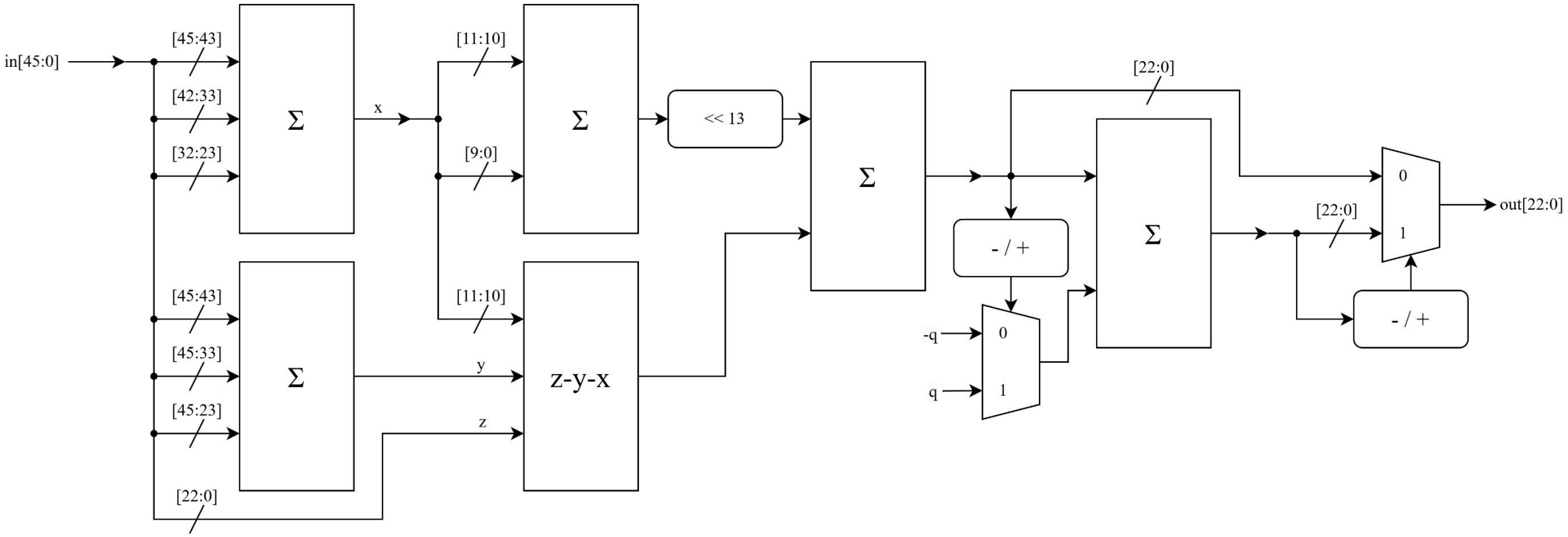
$$\equiv 2^{23}s[45:43] + 2^{13}\,(s[42:33] + s[32:23]) - (s[45:33] + s[45:23]) + z$$

$$\equiv 2^{13}\,(s[45:43] + s[42:33] + s[32:23]) - (s[45:43] + s[45:33] + s[45:23]) + z$$

$$\equiv 2^{13}x - y + z \equiv 2^{23}x[11:10] + 2^{13}x[9:0] - y + z$$

$$\equiv 2^{13}\,(x[11:10] + x[9:0]) - (y + x[11:10]) + z \pmod q$$

Modular Rduction – Block Diagram



Modular Reduction – Timing Diagram

Name	Value	10.000 ns	15.000 ns	20.000 ns	25.000 ns	30.000 ns	35.000 ns	40.000 ns	45.000 ns	50.000 ns	55.000 ns
in[45:0]	98765432	0		70368744177663		123456789		8380416		987654321	
out[22:0]	7145532	0		49144		6130951		8380416		7145532	
in[45:0]	00003ade	000000000000		3fffffffffff		0000075bcd15		0000007fe000		00003ade68b1	
out[22:0]	6d083c	000000		00bff8		5d8d07		7fe000		6d083c	
x[11:0]	075	000		805		00e		000		075	
y[23:0]	000075	000000		802005		00000e		000000		000075	
z[22:0]	5e68b1	000000		7fffff		5bcd15		7fe000		5e68b1	
d[11:0]	075	000		007		00e		000		075	
e[26:0]	05e683c	0000000		7ffdff8		05bcd07		07fe000		05e683c	
f[23:0]	6d083c	000000		00bff8		5d8d07		7fe000		6d083c	
adjust[23:0]	801fff	801fff									
g[23:0]	ed283b	801fff		80dff7		ddad06		ffffff		ed283b	

BU

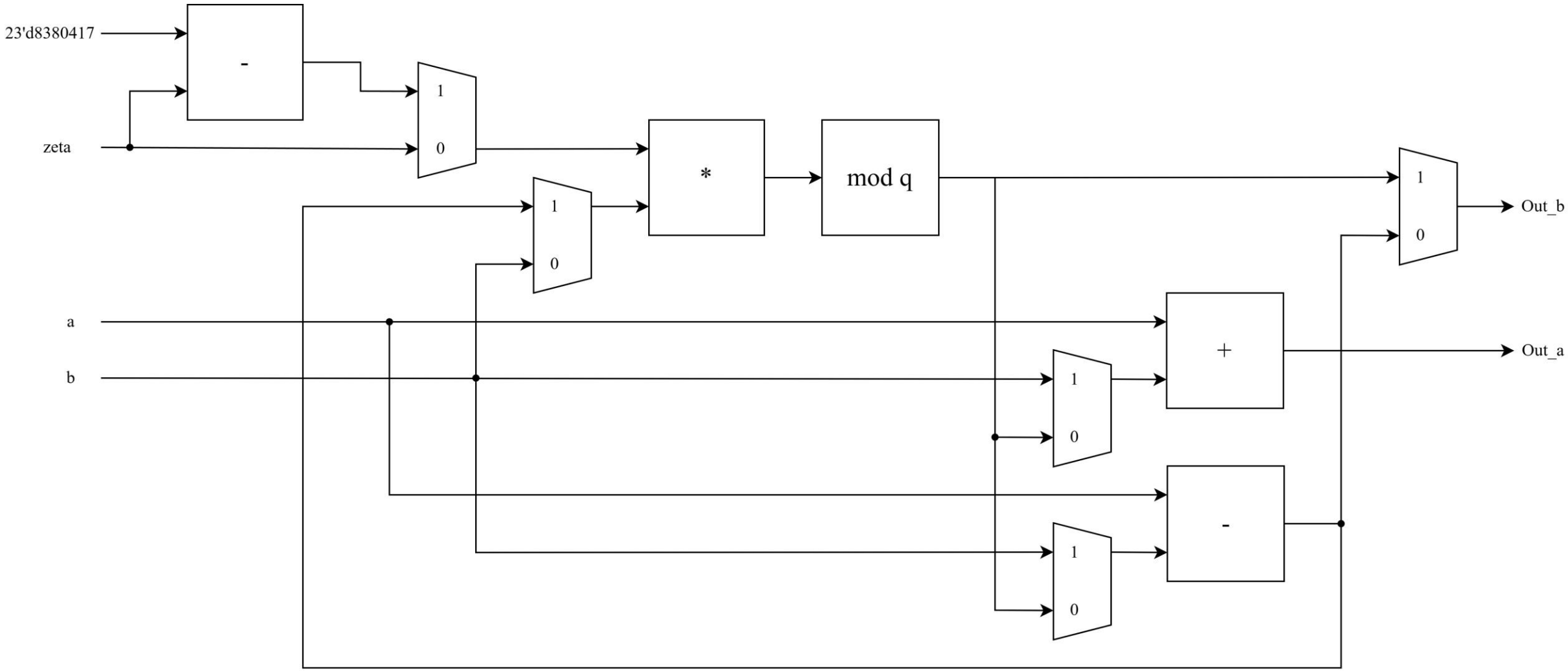
- ✓ The Butterfly Unit is formed based on the symmetry and parity properties in NTT and INTT computations.
- ✓ For a pair of input numbers a and b, together with a corresponding twiddle factor w, the butterfly operation in a finite field (modulo p) proceeds as follows:

$$x=(a+w\times b) \bmod q$$

$$y=(a-w\times b) \bmod q$$

- ✓ The structures of NTT and INTT are similar, but the twiddle factors used in INTT are the modular inverses of those in NTT. A normalization factor must also be applied at the end.
- ✓ Eight BU modules are used in the NTT/INTT of the thesis.

BU – Block Diagram



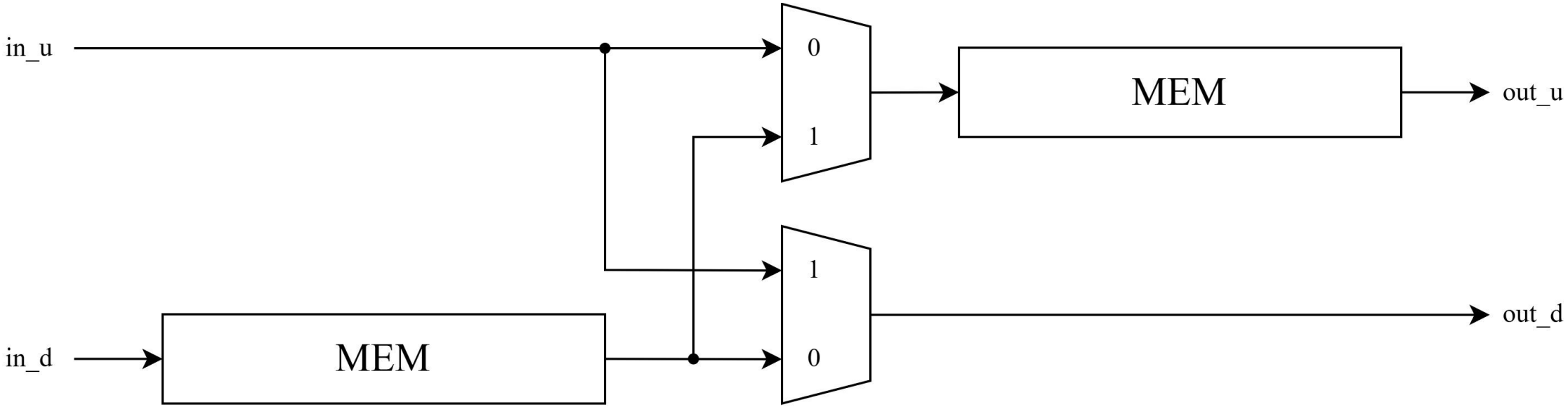
RU

- ✓ To match the butterfly structure, the output of each stage's BU is reordered accordingly.
- ✓ There are a total of 7 RUi, where $1 \leq i \leq 7$, in our implemented NTT and INTT.
- ✓ In NTT/INTT, the MEM depth of each RUi Stage is different:

$$\text{NTT} : \text{MEM\_Depth\_i} = 2 \times ((8 - i) - 1)$$

$$\text{INTT} : \text{MEM\_Depth\_i} = 2 \times (i - 1)$$

RUi – Block Diagram



SHA3

Security strength of the SHA-3 function

✓ Characteristics of the hash function:

Variable Size ⊕ **Collision Resistant** ⊕ **Preimage Resistant** ⊕ **Second Preimage Resistant** ⊕ **Pseudo-Randomness**

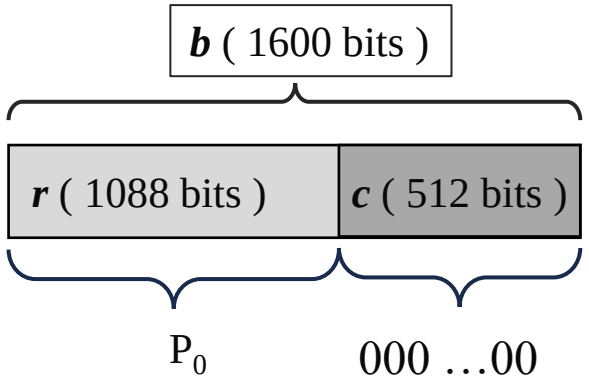
✓ Security strengths of the hash function:

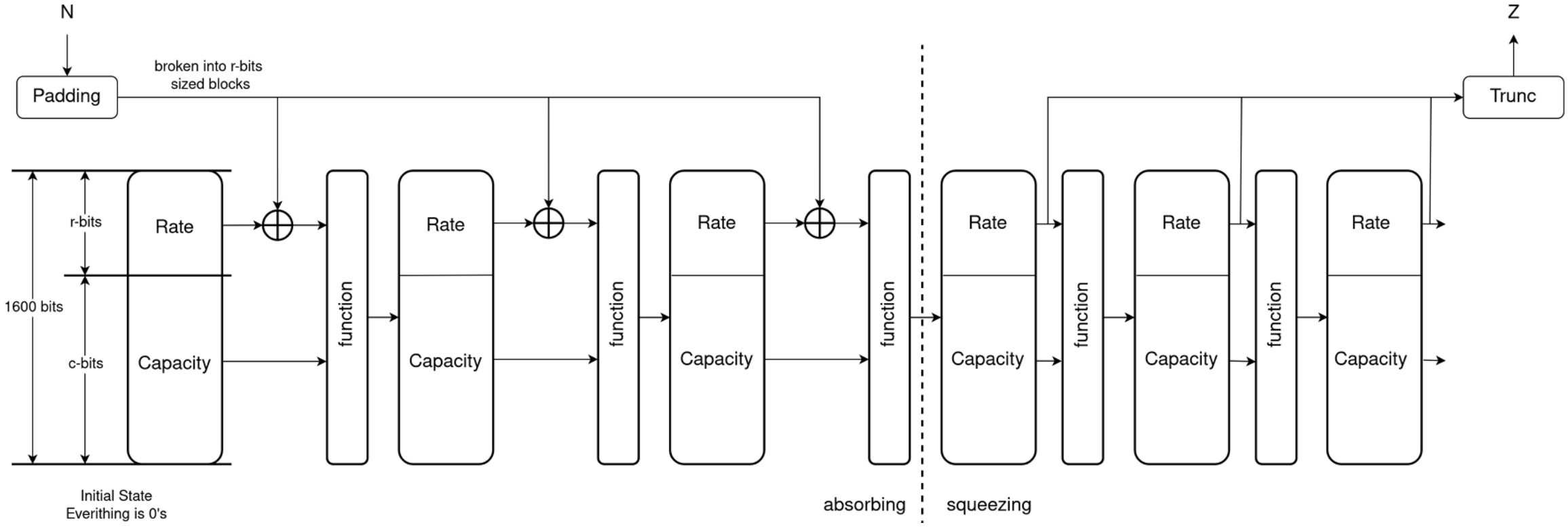
Function	SHA3 Primitive	Output Size	Security Strengths in Bits		
			Collision	Preimage	2nd Preimage
Cryptographic Hash Function	SHA3-224	224	112	224	224
	SHA3-256	256	128	256	256
	SHA3-384	384	192	384	384
	SHA3-512	512	256	512	512
Extendable-Output Function	SHAKE128	d	min(d/2, 128)	≥ min(d, 128)	min(d, 128)
	SHAKE256	d	min(d/2, 256)	≥ min(d, 256)	min(d, 256)

Sponge Construction

- ✓ Sponge Construction: This structure is composed of absorbing phases and squeezing phases
- ✓ Padding: The padding algorithm (pad10*1)
- ✓ Function: The internal function used to process each input block include θ, ρ, π, χ, and ι

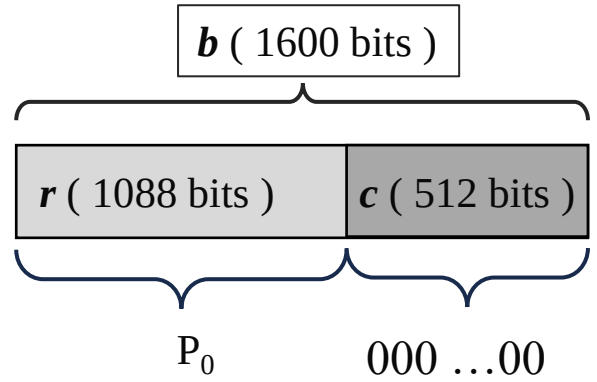
Ex: SHAKE-256





KECCAK-p

Ex: SHAKE-256



- ✓ For *b*=1600, the KECCAK family is referred to as KECCAK[c]:

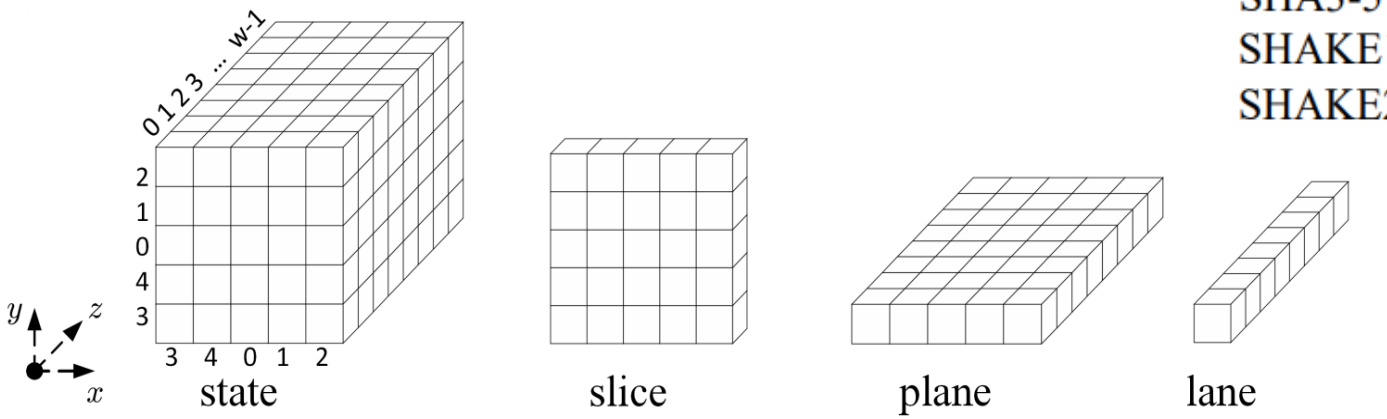
  KECCAK[c]=SPONGE[KECCAK−p[1600, 24], pad10∗1, 1600−c]

- ✓ Given an input bit string N and an output length d:

  KECCAK[c](M, d)=SPONGE[KECCAK−p[1600, 24], pad10∗1, 1600−c](M, d)

b	25	50	100	200	400	800	1600
w	1	2	4	8	16	32	64
l	0	1	2	3	4	5	6

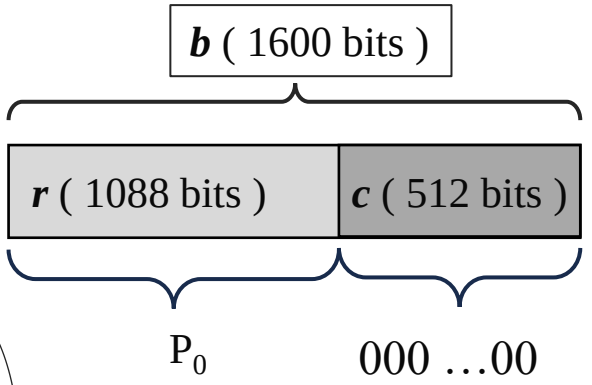
SHA3-224(*M*) = KECCAK[448] (*M* || 01, 224);
SHA3-256(*M*) = KECCAK[512] (*M* || 01, 256);
SHA3-384(*M*) = KECCAK[768] (*M* || 01, 384);
SHA3-512(*M*) = KECCAK[1024] (*M* || 01, 512).
SHAKE128(*M*, *d*) = KECCAK[256] (*M* || 1111, *d*),
SHAKE256(*M*, *d*) = KECCAK[512] (*M* || 1111, *d*).



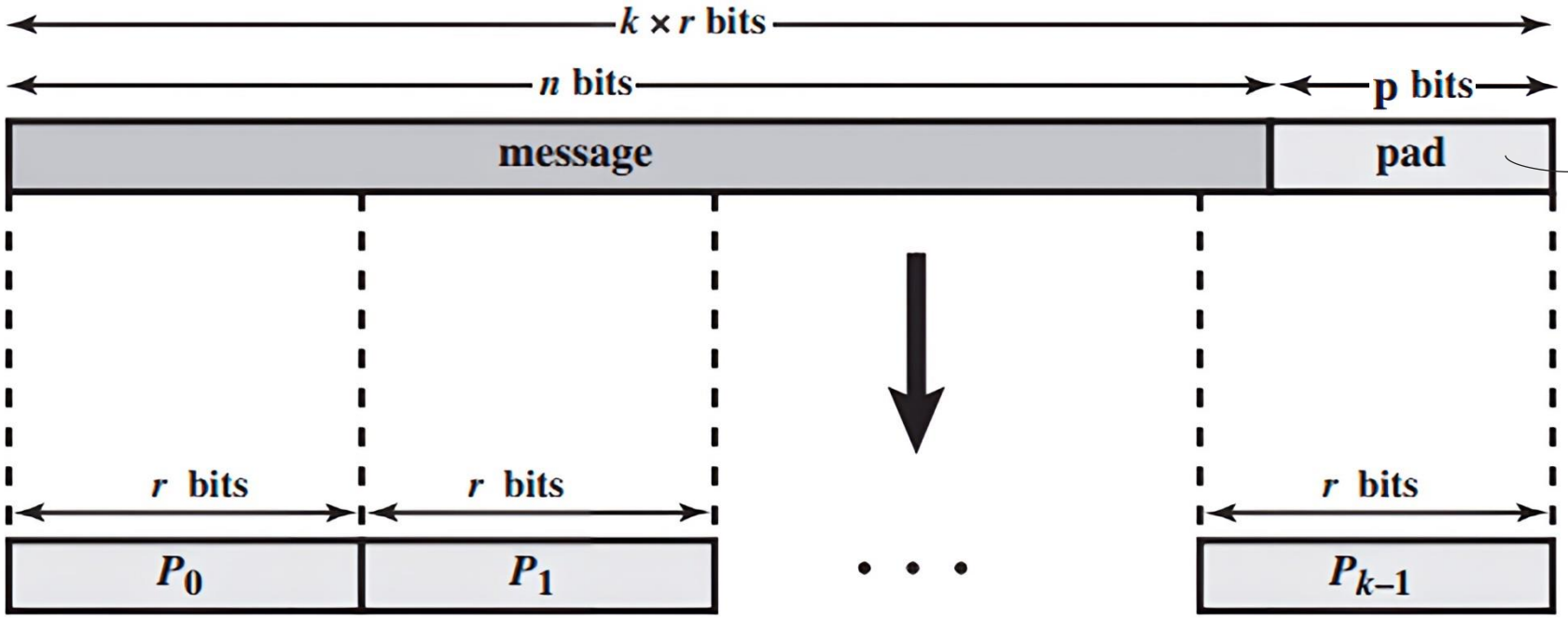
Padding

- ✓ **Simple padding :** The filling method is to pad10(0*)
- ✓ **Multi-rate padding :** The filling method is to pad10(0*)1
- ✓ The padding rule for KECCAK uses **multi-rate padding**

Ex: SHAKE-256



100…0001



Ex: SHAKE-256

r = 1088 bits

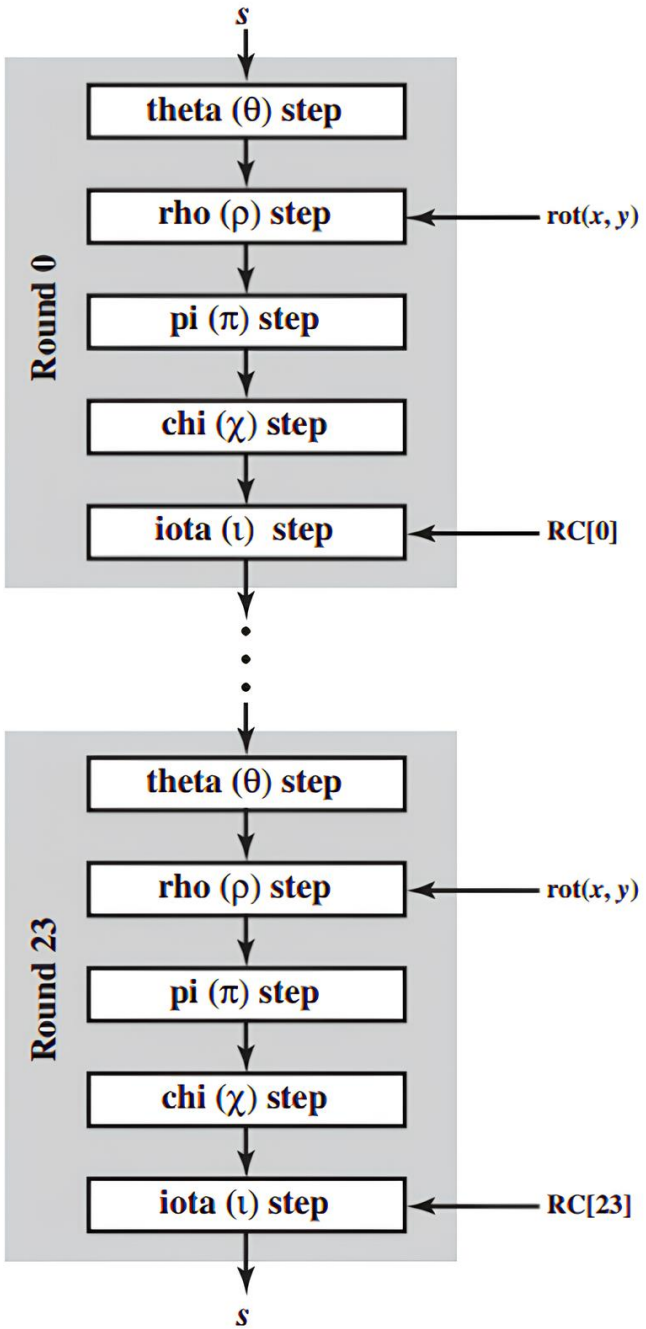
n = 10496 bits

k = ⌈10496 / 1088⌉ = 10

p = 10496 mod 1088 = 704

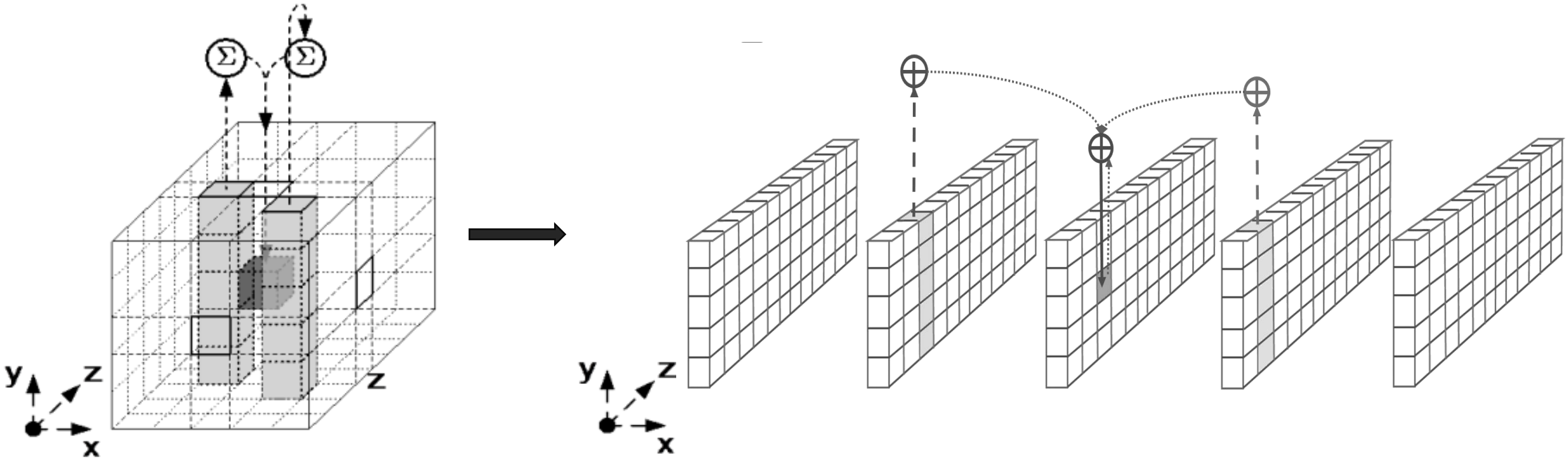
Function

Function	Type
θ	Substitution
ρ	Permutation
π	Permutation
χ	Substitution
ι	Substitution



Function - Theta **θ**

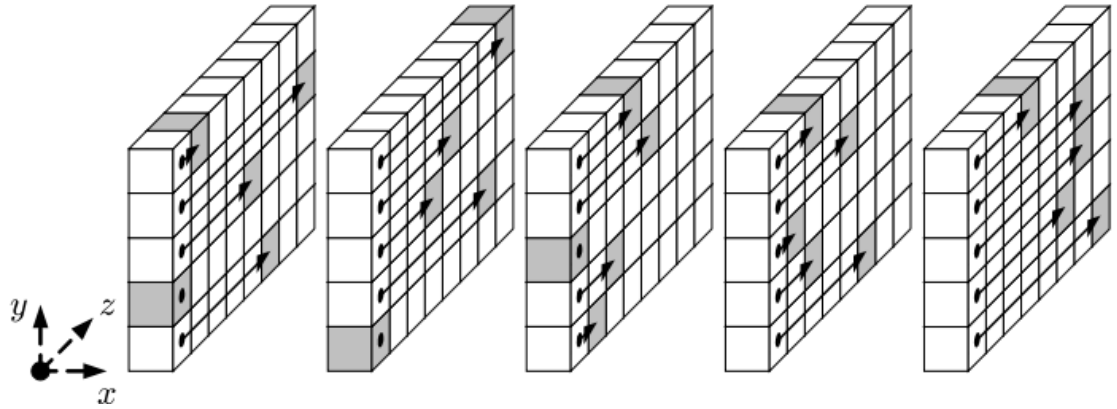
- ✓ $C[x] = A[x, 0] \oplus A[x, 1] \oplus A[x, 2] \oplus A[x, 3] \oplus A[x, 4] \quad \forall\ x \text{ in } 0...4$
- ✓ $D[x] = C[x - 1] \oplus ROT(C[x + 1], 1) \quad \forall\ x \text{ in } 0...4$
- ✓ $A[x, y] = A[x, y] \oplus D[x] \quad \forall\ (x, y) \text{ in } (0...4, 0...4)$

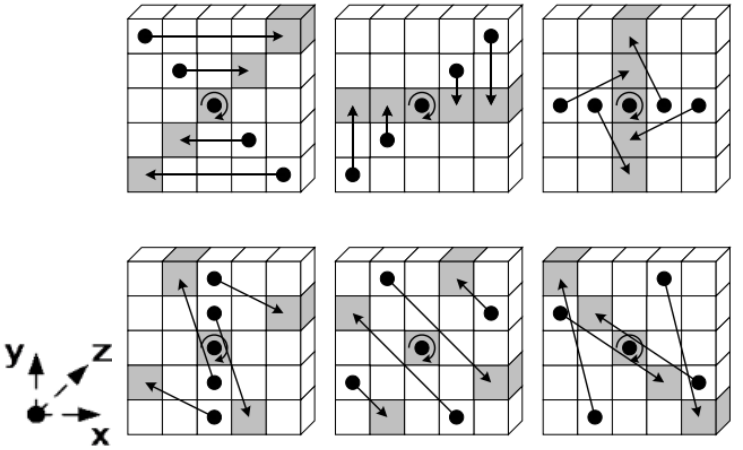


Function - Rho ρ and Pi π

✓ B[y, 2x + 3y] = ROT(A[x, y], r[x, y])

∀ (x, y) in (0...4, 0...4)

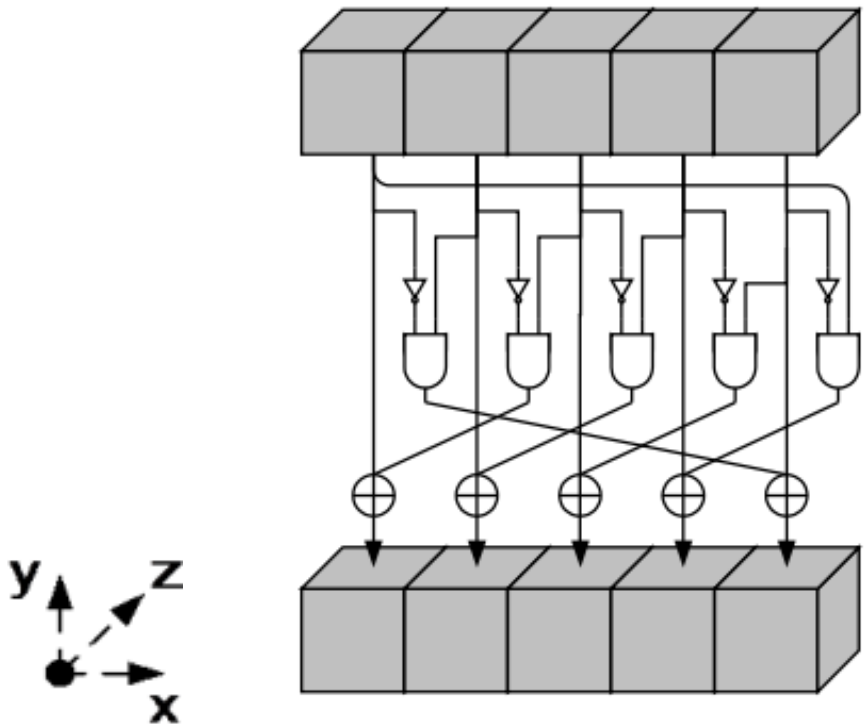




	***x* = 3**	***x* = 4**	***x* = 0**	***x* = 1**	***x* = 2**
***y* = 2**	25	39	3	10	43
***y* = 1**	55	20	36	44	6
***y* = 0**	28	27	0	1	62
***y* = 4**	56	14	18	2	61
***y* = 3**	21	8	41	45	15

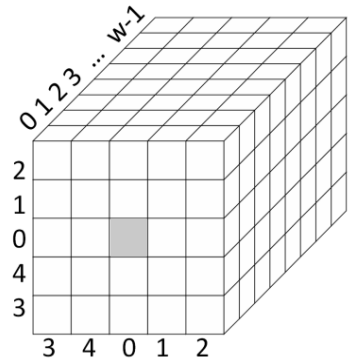
Function - Chi χ

- ✓ $A[x, y] = B[x, y] \oplus ((\text{NOT } B[x + 1, y]) \text{ AND } B[x + 2, y]), \quad \forall (x, y) \text{ in } (0...4, 0...4)$



Function - Iota ι

- ✓ The round constant for each round is generated by a linear feedback shift register (LFSR)
- ✓ A[0, 0] = A[0, 0] $\oplus$ RC
- ✓ The bits corresponding to 63, 31, 15, 7, 3, 1, and 0 are extracted



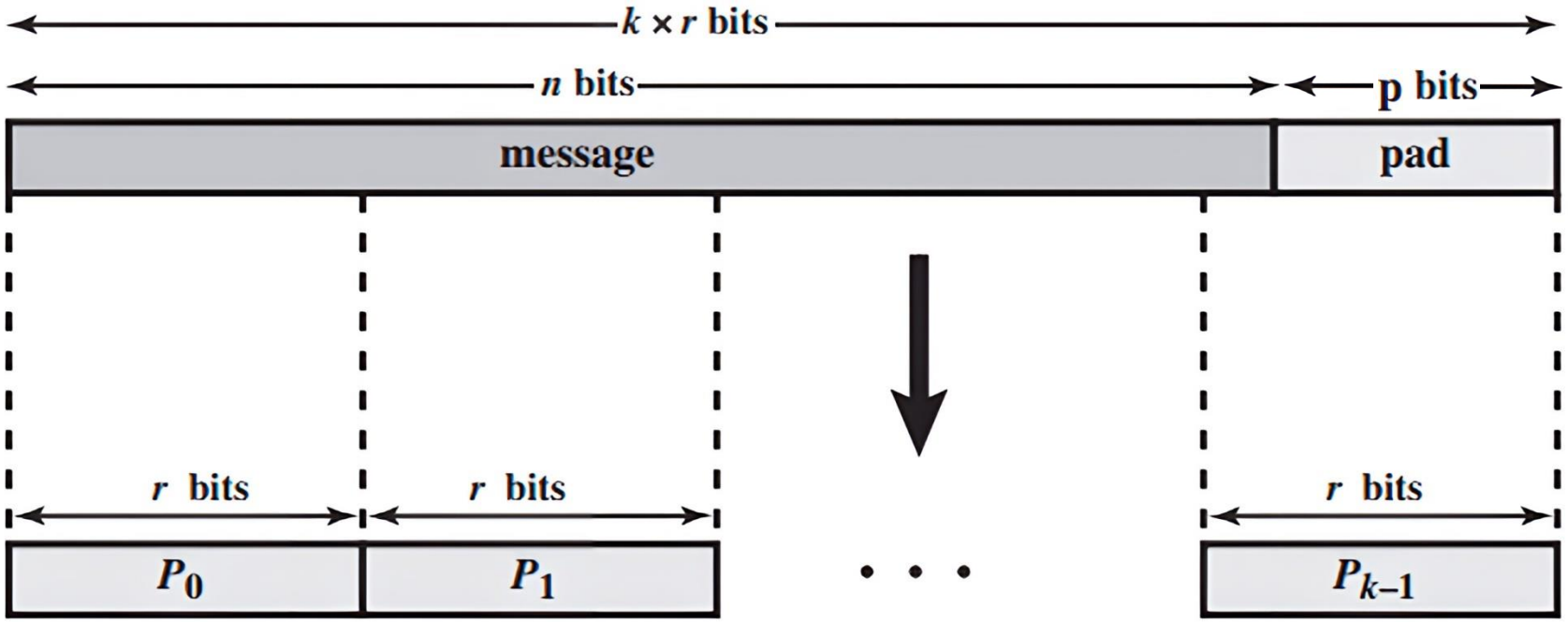
Round	Constant	Round	Constant
0	0000_0000_0000_0001	12	0000_0000_8000_808B
1	0000_0000_0000_8082	13	8000_0000_0000_008B
2	8000_0000_0000_808A	14	8000_0000_0000_8089
3	8000_0000_8000_0000	15	8000_0000_0000_8003
4	0000_0000_0000_808B	16	8000_0000_0000_8002
5	0000_0000_8000_0001	17	8000_0000_0000_0080
6	8000_0000_8000_8081	18	0000_0000_0000_800A
7	8000_0000_0000_8009	19	8000_0000_8000_000A
8	0000_0000_0000_008A	20	8000_0000_8000_8081
9	0000_0000_0000_0088	21	8000_0000_0000_8080
10	0000_0000_8000_8009	22	0000_0000_8000_0001
11	0000_0000_8000_000A	23	8000_0000_8000_8008

Ex.

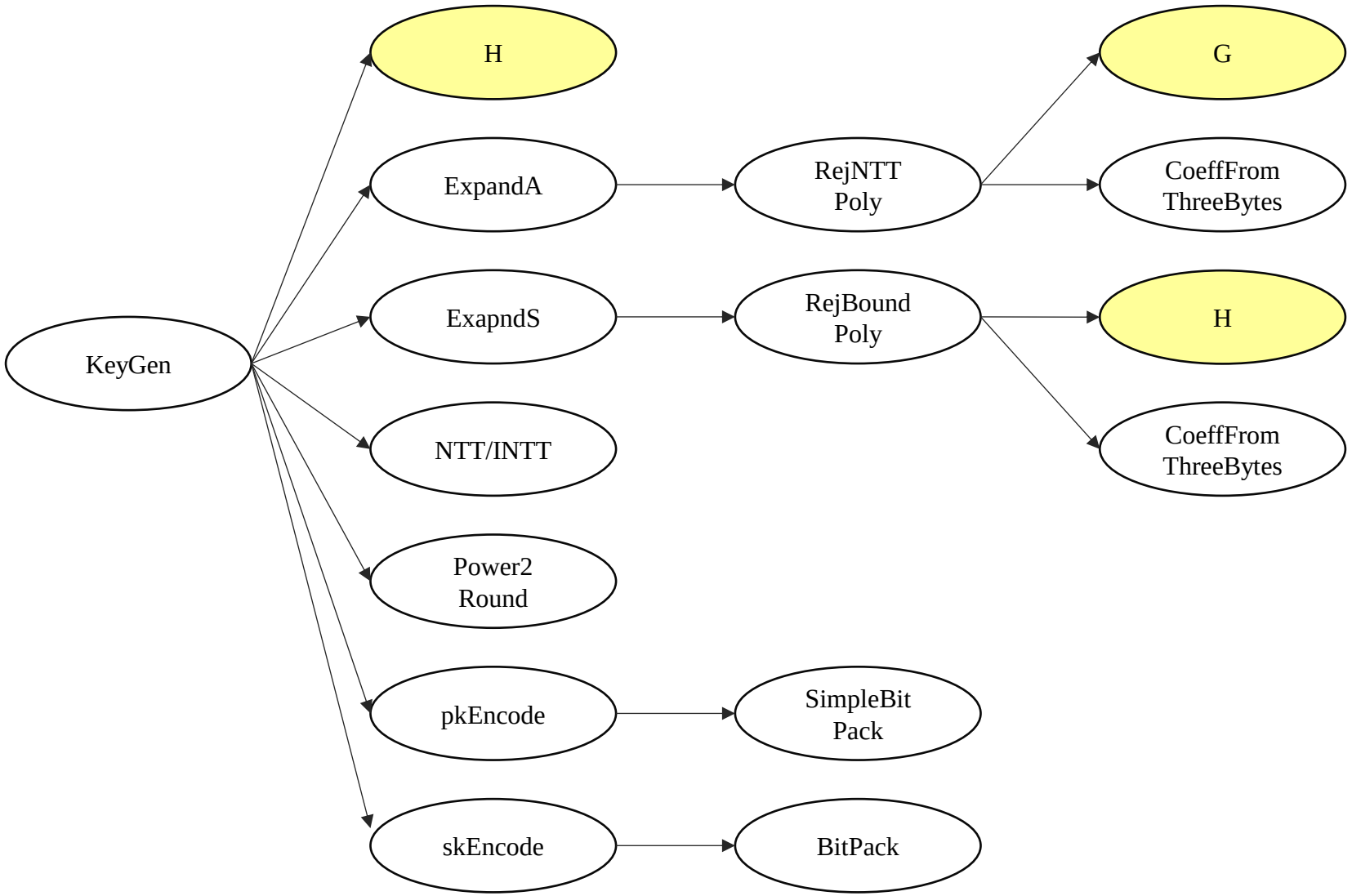
rc[0] = i[0] | i[4] | i[5] | i[6] | i[7] | i[10] |
i[12] | i[13] | i[14] | i[15] | i[20] | i[22];

KECCAK-p PERMUTATIONS

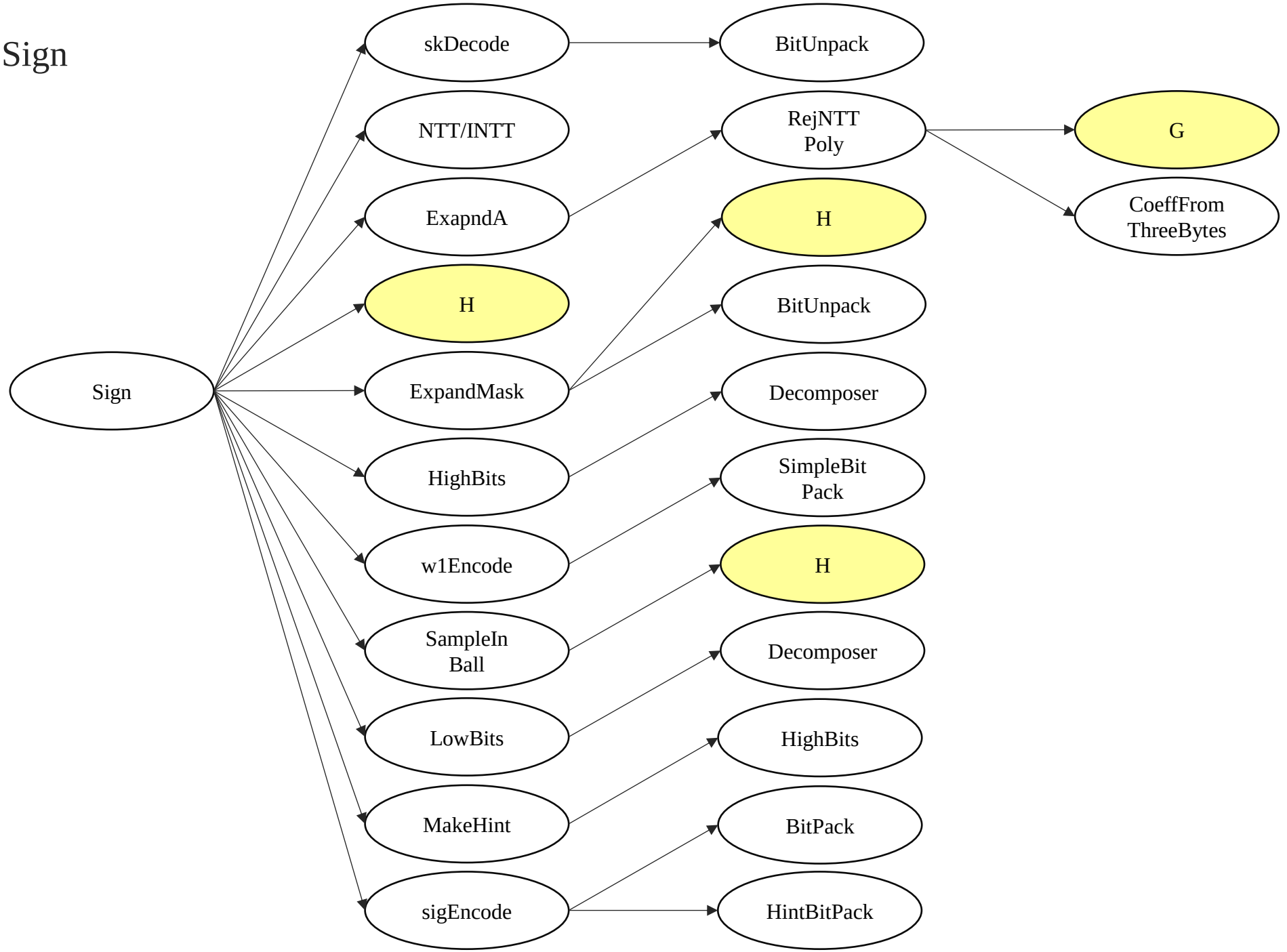
SHA3 primitives	MLDSA primitives	Padding	Size in bits		
			r	c	output length
SHAKE128	G	M || 0x1f || 0x00 … || 0x80	1344	256	unlimited
SHAKE256	H	M || 0x1f || 0x00 … || 0x80	1088	512	unlimited



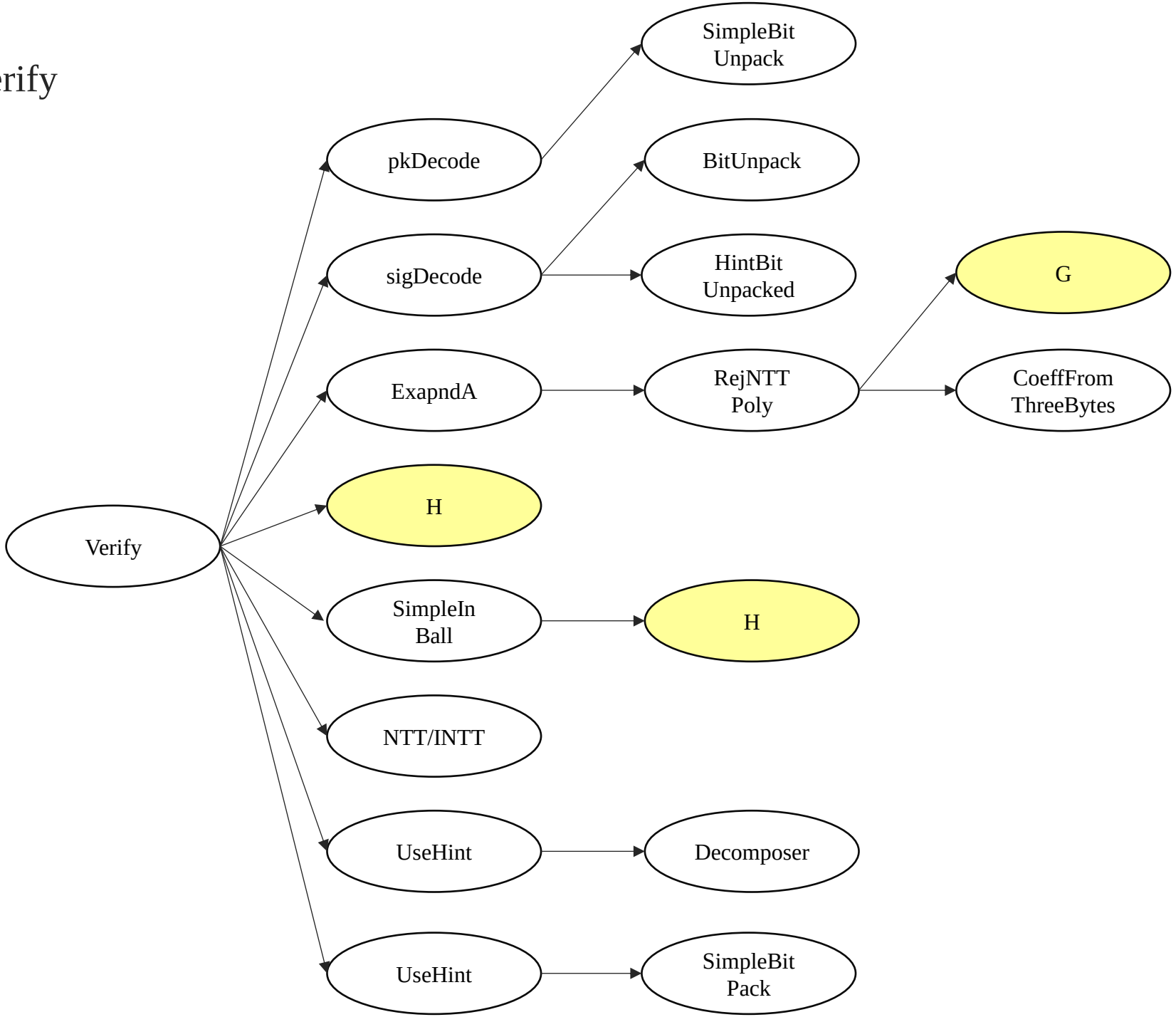
KeyGen


KeyGen
H
ExpandA
ExapndS
NTT/INTT
Power2
Round
pkEncode
skEncode
RejNTT
Poly
RejBound
Poly
SimpleBit
Pack
BitPack
G
CoeffFrom
ThreeBytes
H
CoeffFrom
ThreeBytes

Sign


Sign
skDecode
NTT/INTT
ExapndA
H
ExpandMask
HighBits
w1Encode
SampleIn Ball
LowBits
MakeHint
sigEncode
BitUnpack
RejNTT Poly
H
BitUnpack
Decomposer
SimpleBit Pack
H
Decomposer
HighBits
BitPack
HintBitPack
G
CoeffFrom ThreeBytes

Verify


Verify
pkDecode
sigDecode
ExapndA
H
SimpleIn
Ball
NTT/INTT
UseHint
UseHint
SimpleBit
Unpack
BitUnpack
HintBit
Unpacked
RejNTT
Poly
H
G
CoeffFrom
ThreeBytes
Decomposer
SimpleBit
Pack

MLDSA SHA3

Main algorithm	SHA3 primitives	Input Size	Output Size
KeyGen	H_SeedGen	(32 + 2)*8 = 272	128*8 = 1024
	G_ExapndA	34*8 = 272	894*8 = 7152
	H_ExpandS	(64 + 2) * 8 = 528	481 * 8 =3848
	H_tr	1312 * 8 = 10496	64 * 8 = 512
Sign	G_ExapndA	34*8 = 272	894*8 = 7152
	H_μ	(64 +(2+n+m))*8	64 * 8 =512
	H_ρ''	(32 + 32 + 64) *8 = 1024	64 * 8 =512
	H_ExpandMask	(64 + 2)*8 = 528	32 * 18 * 8 = 4608
	H_C_tilde	(64 + 768)*8 = 6656	128 / 4 * 8 = 256
	H_SampleInball	128 / 4 * 8 = 256	221 * 8 = 1768
Verify	G_ExapndA	34*8 = 272	894*8 = 7152
	H_tr	1312 * 8 = 10496	64 * 8 = 512
	H_μ	(64 + 283)*8 = 2776	64 * 8 =512
	H_SampleInball	128 / 4 * 8 = 256	221 * 8 = 1768

MLDSA SHA3

SHA3 primitives	Input Size	Output Size	mode	is_last cycle $\lceil \text{Input Size}/64 \rceil + 1$	byte_num $\frac{\text{Input Size mod } 64}{8}$	Squeeze time $\left\lceil \frac{Output\ Size}{mode\ size} \right\rceil - 1$
H_SeedGen	272	1024	1	5	2	0
G_ExapndA	272	7152	0	5	2	6
H_ExpandS	528	3848	1	9	2	4
H_tr	10496	512	1	164	0	0
H_μ	2776	512	1	44	3	0
H_ρ''	1024	512	1	17	0	0
H_ExpandMask	528	4608	1	9	2	4
H_C_tilde	6656	256	1	105	0	0
H_SampleInball	256	1768	0	5	0	1

MLDSA SHA3

SHA3 primitives	Primitives sign	MLDSA_mode	Stage	mode	Input side		Output side		
H(ξ+x04+x04)	0	KeyGen	1	H	MLDSA_in_1	seed_gen_index	Rho MEM	Rho_prime MEM	Kata MEM
Gen(s1)	1	KeyGen	2	H	Rho_prime MEM	s1_gen_index	s1 MEM		
Gen(s2)	2	KeyGen	3	H	Rho_prime MEM	s2_gen_index	s2 MEM		
Gen(A)	3	KeyGen	4	G	Rho MEM	A_gen_index	A MEM		
H(ρ || t1) = tr	4	KeyGen	7	H	Rho MEM	t1 MEM	tr MEM		
H(tr || M')_1	5	SignGen	1	H	MLDSA_in_1		u MEM		
H(K || rnd || u)	6	SignGen	2	H	MLDSA_in_1		p" MEM		
Gen(y)	7	SignGen	3	H	p" MEM	y_gen_index	y MEM		
Gen(A)	3	SignGen	4,5	G	Rho MEM		A MEM		
H(μ || $\mathbf{w}$1)	8	SignGen	9	H	u MEM	w MEM	c_tilde MEM		
Gen(c)_1	9	SignGen	10	H	c_tilde MEM		c MEM		
Gen(y)	7	SignGen	15	H	p" MEM	y_gen_index	y MEM		
Gen(c)_2	10	SignVer	1	H	MLDSA_in_1		c MEM		
Gen(A)	3	SignVer	2,3	G	Rho MEM	A_gen_index	A MEM		
H(pk)	11	SignVer	4	H	MLDSA_in_1		tr MEM		
H(tr || M')_2	12	SignVer	5	H	tr MEM	MLDSA_in_1	u MEM		
H(μ || $\mathbf{w}$1)	8	SignVer	8	H	u MEM	w MEM	c_tilde MEM		

MLDSA SHA3

SHA3 primitives	Primitives sign	mode	Input side		Output side			sha_byte_num	i_last cycle times
H(ξ+x04+x04)	0	H	MLDSA_in_1	seed_gen_index	Rho MEM	Rho_prime MEM	Kata MEM	010	5
Gen(s1)	1	H	Rho_prime MEM	s1_gen_index	s1 MEM			010	9
Gen(s2)	2	H	Rho_prime MEM	s2_gen_index	s2 MEM			010	9
Gen(A)	3	G	Rho MEM	A_gen_index	A MEM			010	5
H(pk)_1	4	H	Rho MEM	t1 MEM	tr MEM			000	165
H($tr \parallel M'$)_1	5	H	MLDSA_in_1		u MEM			(64+2+n+m) mod 8	$\left\lceil \frac{64+2+n+m}{8} \right\rceil$
H($K \parallel rnd \parallel$ u)	6	H	MLDSA_in_1		p" MEM			000	17
Gen(y)	7	H	p" MEM	y_gen_index	y MEM			010	9
H($\mu \parallel \mathbf{w}1$)	8	H	u MEM	w1 MEM	c_tilde MEM			000	105
Gen(c)_1	9	H	c_tilde MEM		c MEM			000	5
Gen(c)_2	10	H	MLDSA_in_1		c MEM			000	5
H(pk)_2	11	H	MLDSA_in_1		tr MEM			000	165
H($tr \parallel M'$)_2	12	H	tr MEM	MLDSA_in_1	u MEM			011	44

MLDSA SHA3

SHA3 primitives	Primitives sign	mode	Input side		Output side			sha_byte_num	i_last cycle times
H(ξ+x04+x04)	0	H	MLDSA_in_1	seed_gen_index	Rho MEM	Rho_prime MEM	Kata MEM	010	5
Gen(A)	1	G	Rho MEM	A_gen_index	A MEM			010	5
Gen(s1)	2	H	Rho_prime MEM	s1_gen_index	s1 MEM			010	9
Gen(s2)	3	H	Rho_prime MEM	s2_gen_index	s2 MEM			010	9
Gen(y)	4	H	p" MEM	y_gen_index	y MEM			010	9
H($\mu \parallel \mathbf{w}1$)	5	H	u MEM	w MEM	c_tilde MEM			000	105
H(pk)_1	6	H	Rho MEM	t1 MEM	tr MEM			000	165
H(pk)_2	7	H	MLDSA_in_1		tr MEM			000	165
Gen(c)_1	8	H	c_tilde MEM		c MEM			000	5
Gen(c)_2	9	H	MLDSA_in_1		c MEM			000	5
H($K \parallel rnd \parallel$ u)	10	H	MLDSA_in_1		p" MEM			000	17
H($tr \parallel M'$)_1	11	H	MLDSA_in_1		u MEM			(64+2+n+m) mod 8	$\left\lceil \frac{64+2+n+m}{8} \right\rceil$
H($tr \parallel M'$)_2	12	H	tr MEM	MLDSA_in_1	u MEM			(64+2+n+m) mod 8	$\left\lceil \frac{64+2+n+m}{8} \right\rceil$

MLDSA SHA3

SHA3 primitives	Primitives sign	mode	Input side		Output side			sha_byte_num	i_last cycle times
Gen_Seed	0	H	MLDSA_in_1	seed_gen_index	Rho MEM	Rho_prime MEM	Kata MEM	010	5
Gen_A	1	G	Rho MEM	A_gen_index	A MEM			010	5
Gen_s1	2	H	Rho_prime MEM	s1_gen_index	s1 MEM			010	9
Gen_s2	3	H	Rho_prime MEM	s2_gen_index	s2 MEM			010	9
Gen_y	4	H	p" MEM	y_gen_index	y MEM			010	9
H_u_w1	5	H	u MEM	w MEM	c_tilde MEM			000	105
H_pk_1	6	H	Rho MEM	t1 MEM	tr MEM			000	165
H_pk_2	7	H	MLDSA_in_1		tr MEM			000	165
Gen_c_1	8	H	c_tilde MEM		c MEM			000	5
Gen_c_2	9	H	MLDSA_in_1		c MEM			000	5
H_K_rnd_u	10	H	MLDSA_in_1		p" MEM			000	17
H_tr_M_1	11	H	MLDSA_in_1		u MEM			(64+2+n+m) mod 8	$\left\lceil \frac{64+2+n+m}{8} \right\rceil$
H_tr_M_2	12	H	tr MEM	MLDSA_in_1	u MEM			(64+2+n+m) mod 8	$\left\lceil \frac{64+2+n+m}{8} \right\rceil$

MLDSA SHA3

Data source	keccak_in_sel	mem_sel_1	mem_sel_2	index_sel	in_seed_sel
Rho MEM	0	0	X	X	X
Rho_prime MEM	0	1	X	X	X
Rho_prime_prime MEM	0	2	X	X	X
u MEM	0	3	x	X	X
w MEM	1	X	0	X	X
t1 MEM	1	X	1	X	X
c_tilde MEM	1	X	2	X	X
tr MEM	1	X	3	X	X
A_gen_index	2	X	X	0	X
s1_gen_index	2	X	X	1	X
s2_gen_index	2	X	X	2	X
y_gen_index	2	X	X	3	X
MLDSA_in_1	3	X	X	X	0
seed_gen_index	3	X	X	X	1

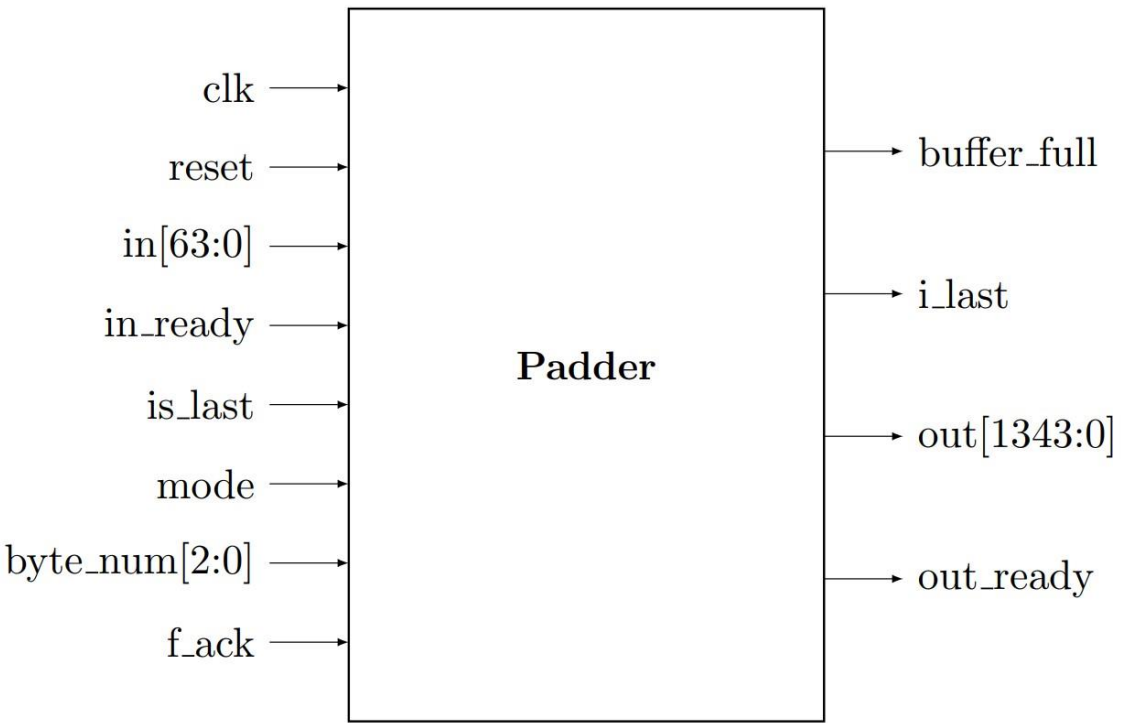
MLDSA SHA3

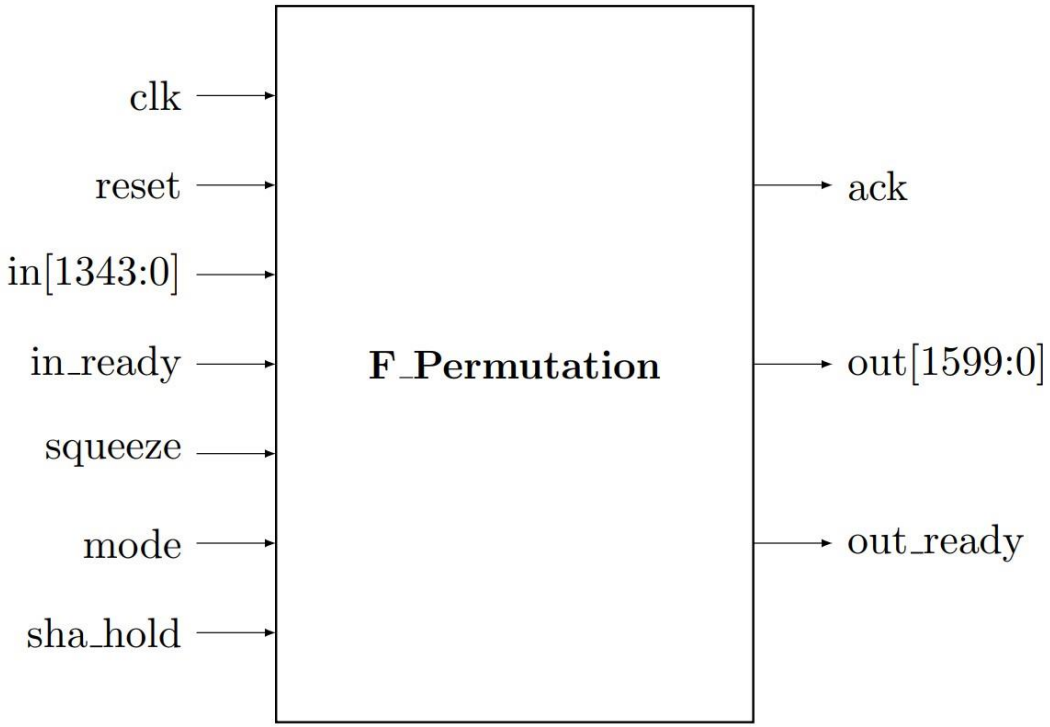
Data source	keccak_in_sel	mem_sel_1	mem_sel_2	index_sel	in_seed_sel
Seed MEM	0	0	X	X	X
u MEM	0	1	x	X	X
w MEM	1	X	0	X	X
t1 MEM	1	X	1	X	X
c_tilde MEM	1	X	2	X	X
tr MEM	1	X	3	X	X
A_gen_index	2	X	X	0	X
s1_gen_index	2	X	X	1	X
s2_gen_index	2	X	X	2	X
y_gen_index	2	X	X	3	X
MLDSA_in_1	3	X	X	X	0
seed_gen_index	3	X	X	X	1

MLDSA SHA3

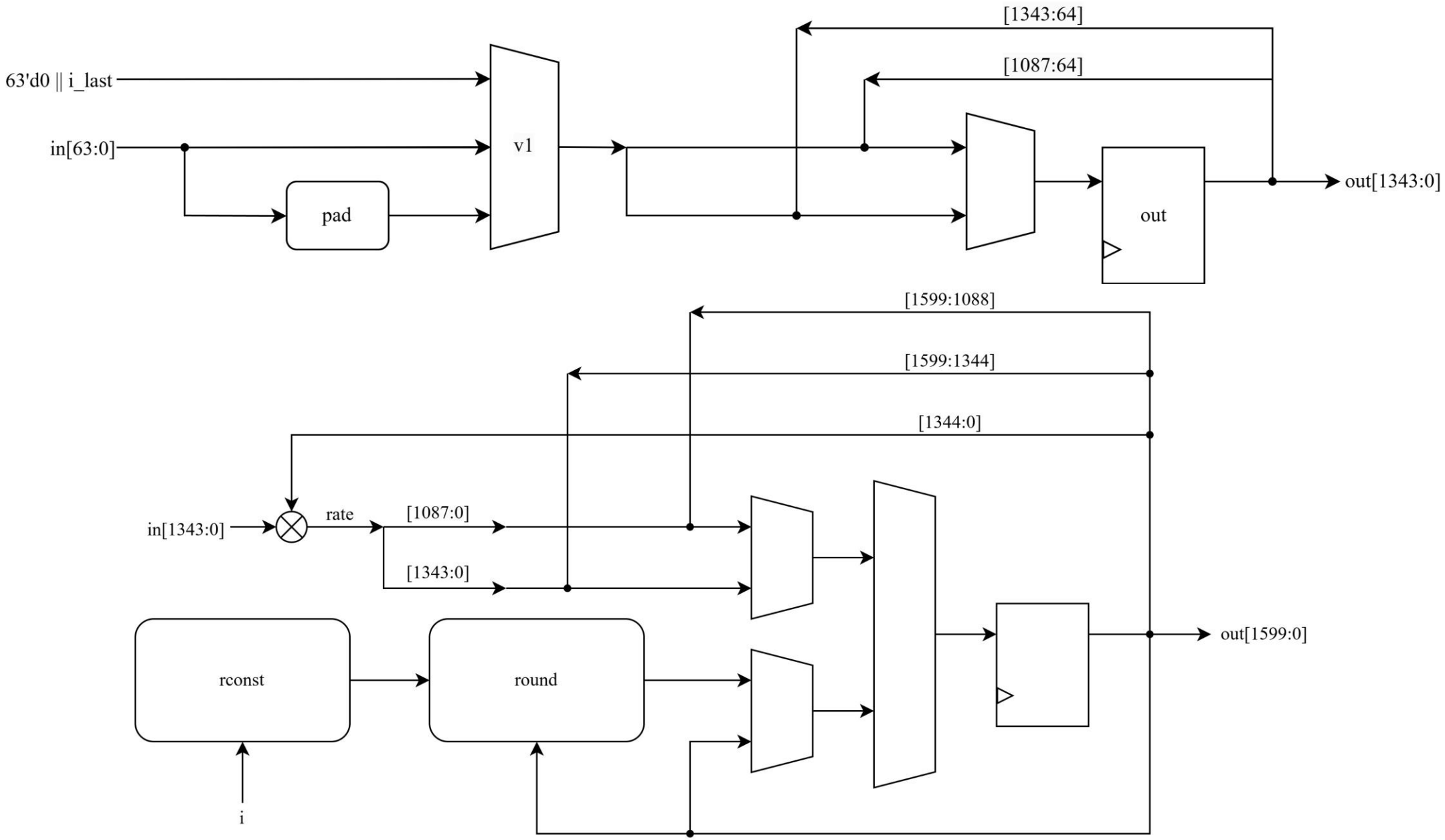
Data source	keccak_in_sel	kk_sub_sel_1	kk_sub_sel_2	kk_sub_sel_3
Seed MEM	0	0	X	X
u MEM	0	1	x	X
w MEM	0	2	X	X
t1 MEM	0	3	X	X
c_tilde MEM	1	X	0	X
tr MEM	1	X	1	X
zero_index	1	X	2	X
seed_gen_index	1	X	3	X
A_gen_index	2	X	X	0
s1_gen_index	2	X	X	1
s2_gen_index	2	X	X	2
y_gen_index	2	X	X	3
MLDSA_in_1	3	X	X	X

MLDSA SHA3 Module

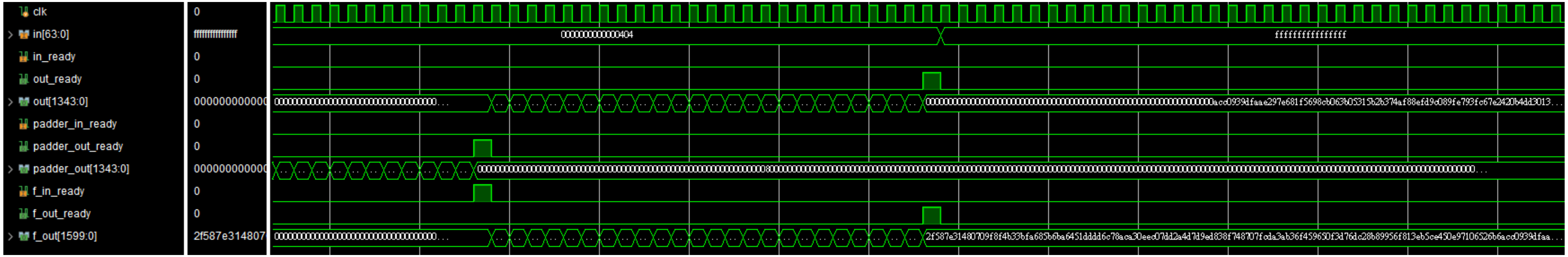




MLDSA SHA3 Block Diagram



MLDSA SHA3 Block Diagram




```
padder_out: 0000000000000000000000000000000000000000000000000000000000000000000000000000000000008000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
fout: 2f587e31480709f8f4b33bfa685b6ba6451dddd6c78aca30eec07dd2a4d7d9ed838f748707fcda3ab36f459650f3d76dc28b89956f813eb5ce450e97106526b6acc0939dfaae297e681f5698cb063b05315b2b374af88efd9c089fe793fc67e2420b4dd30138f14fa4970fd23a3bb9400920bce83750376e81f8ababf200c27668aa7a
sha_out: 0000000000000000000000000000000000000000000000000000000000000000acc0939dfaae297e681f5698cb063b05315b2b374af88efd9c089fe793fc67e2420b4dd30138f14fa4970fd23a3bb9400920bce83750376e81f8ababf200c27668aa7ae7d3f52fddccf36c0e578001f3667b89b30401f98fbb5b52697c268809d4a

         0000000000001f0404ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff.
         7ae7d3f52fddccf36c0e578001f3667b89b30401f98fbb5b52697c268809d4ac993d6fa5d4eb20869b196b33729e0724c632ec0ea29e4c2f93a312620c25de72acb56bfd0434.
         4ac993d6fa5d4eb20869b196b33729e0724c632ec0ea29e4c2f93a312620c25de72acb56bfd0434.
```

```python
### TEST "MLDSA FSM" ###
xi = 'FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF'
xi = bytes.fromhex(xi) + b'\x04\x04'
result = SHAKE_256(xi,1088)
print(Verilog_trans(result))
Rho_seed = result[0:32]
Rho_Prime_seed = result[32:96]
Kata_seed = result[96:128]
```

```
PS C:\Users\fossu\Desktop\ML_DSA_Syn\SHA3\python> python -u "c:\Users\fossu\Desktop\ML_DSA_Syn\SHA3\python\SHA_3.py"
ACC0939DFAAE297E681F5698CB063B05315B2B374AF88EFD9C089FE793FC67E2420B4DD30138F14FA4970FD23A3BB9400920BCE83750376E81F8ABABF200C27668AA7AE7D3F52FDDCCF36C0E578001F3667B89B30401F98FBB5B52697C268809D4AC
993D6FA5D4EB20869B196B33729E0724C632EC0EA29E4C2F93A312620C25DE72ACB56BFD0434
```

Data Mem

MLDSA Data Mem - SignGen

	資料存入
	資料送出

記憶體	寬度,深度	1	2	3	4-1	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
temp0	23, 1024	s1	s1	y														y	ct0	ct0
temp1	23, 1024		s2	s2		A0		A0					c^ 0-255					c^ 0-255		
temp2	23, 1024					A1		A1								z			z	
temp3	23, 1024					A2		A2							cs2^	cs2^		ct0^	ct0^	
temp4	23, 1024					A3		A3	w										w	
temp5	23, 1024			t0		t0	y^	y^				c 0-255	c 0-255	cs1^	cs1^	cs2			cs2	
temp6	23, 1024		s1^											s1^	cs1	cs1			w -cs2	w -cs2
temp7	23, 1024			s2^											s2^					
temp8	23, 1024					t0^												t0^		
temp9	23, 1024							w^	w^											
seed	64, 16	u 8-15	u 8-15								u 8-15									
			P'' 0-7	P'' 0-7	P 0-3	P 0-3					c~ 0-3	c~ 0-3								
w1_ pack	64, 96	改由encode 完生鮮直送，不存入MEM																		
r0	18, 1024	產生r0的同時計算用於判斷拒絕採樣的條件||r0||，即可部存入MEM																		

MLDSA Data Mem - SignGen

	資料存入
	資料送出

記憶體	寬度,深度	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
temp0	23, 1024	s1	s1	y																y
temp1	23, 1024		s2	s2			y_hat	y_hat	w								w	ct0_hat	ct0_hat	
temp2	23, 1024			t0		t0		w_hat	w_hat					cs1_hat	cs1_hat	cs2	cs2		ct0	ct0
temp3	23, 1024											c (0-255)	c (0-255)		cs1	cs1	w_cs2			w_cs2
temp4	23, 1024														cs2_hat	cs2_hat	HB(_w_cs2)			HB(_w_cs2)
temp5	23, 1024															z				z
temp6	23, 1024												c_hat (0-255)					c_hat (0-255)		
temp7	23, 1024					A														A
temp8	23, 1024		s1_hat																	s1_hat
temp9	23, 1024			s2_hat																s2_hat
temp10	23, 1024					t0_hat														t0_hat
seed	64, 16	u 8-15	u 8-15								u 8-15									
			p'' 0-7	p'' 0-7	p 0-3	p 0-3					c~ 0-3	c~ 0-3								
w1_pack	64, 96	改由encode 完生鮮直送，不存入MEM																		
r0	18, 1024	產生r0的同時計算用於判斷拒絕採樣的條件||r0||，即可部存入MEM																		

MLDSA Data Mem - SignGen

	資料存入
	資料送出

記憶體	寬度,深度	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
temp0	23, 1024	s1	s1	y			y									y	y			y
temp1	23, 1024		s2	s2			y_hat	y_hat	w	w							w	ct0_hat	ct0_hat	
temp2	23, 1024			t0		t0		w_hat	w_hat					cs1_hat	cs1_hat	cs2	cs2		ct0	ct0
temp3	23, 1024											c (0-255)	c (0-255)		cs1	cs1	w_cs2			w_cs2
temp4	23, 1024														cs2_hat	cs2_hat	HB(_w_cs2)			HB(_w_cs2)
temp5	23, 1024															z				z
temp6	23, 1024												c_hat (0-255)					c_hat (0-255)		
temp7	23, 4096					A		A												
temp8	23, 1024		s1_hat											s1_hat						
temp9	23, 1024			s2_hat											s2_hat					
temp10	23, 1024					t0_hat												t0_hat		
seed	64, 16	u 8-15	u 8-15								u 8-15									
			p'' 0-7	p'' 0-7	p 0-3	p 0-3					c~ 0-3	c~ 0-3								
w1_pack	64, 96	改由encode 完生鮮直送，不存入MEM																		
r0	18, 1024	產生r0的同時計算用於判斷拒絕採樣的條件||r0||，即可部存入MEM																			

MLDSA Data Mem - KeyGen

	資料存入
	資料送出

記憶體	寬度,深度	1	2	3	4	5	6	7	8	9
tempA	23, 4096				A	A				
temp0	23, 1024		s1		s1	As1_hat	As1_hat	t	t	
temp1	23, 1024			s2				s2		
temp2	23, 1024			s1_hat		s1_hat	As1	As1		
seed_ temp	64, 16	p,p',K (0-15)	p' (4-11)	p' (4-11)	p (0-3)					

MLDSA Data Mem - SignGen

	資料存入
	資料送出

記憶體	寬度,深度	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
tempA	23, 4096					A		A												
temp0	23, 1024	s1	s1	y			y									y	y			y
temp1	23, 1024		s2	s2			y_hat	y_hat	w	w							w	ct0_hat	ct0_hat	
temp2	23, 1024			t0		t0		w_hat	w_hat					cs1_hat	cs1_hat	cs2	cs2		ct0	ct0
temp3	23, 1024											c (0-255)	c (0-255)		cs1	cs1	w_cs2			w_cs2
temp4	23, 1024														cs2_hat	cs2_hat	HB(_w_cs2)			HB(_w_cs2)
temp5	23, 1024															z				z
temp6	23, 1024												c_hat (0-255)					c_hat (0-255)		
temp7	23, 1024		s1_hat											s1_hat						
temp8	23, 1024			s2_hat											s2_hat					
temp9	23, 1024					t0_hat												t0_hat		
seed_temp	64, 16	u 8-15	u 8-15								u 8-15									
			p'' 0-7	p'' 0-7	p 0-3	p 0-3					c~ 0-3	c~ 0-3								
pack_temp																				
w1_pack	64, 96	改由encode 完生鮮直送，不存入MEM																		

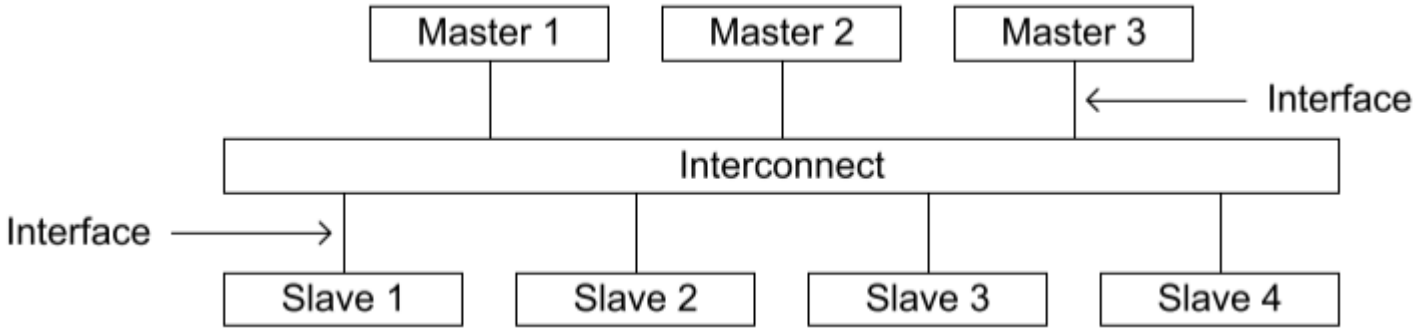
06

AXI4

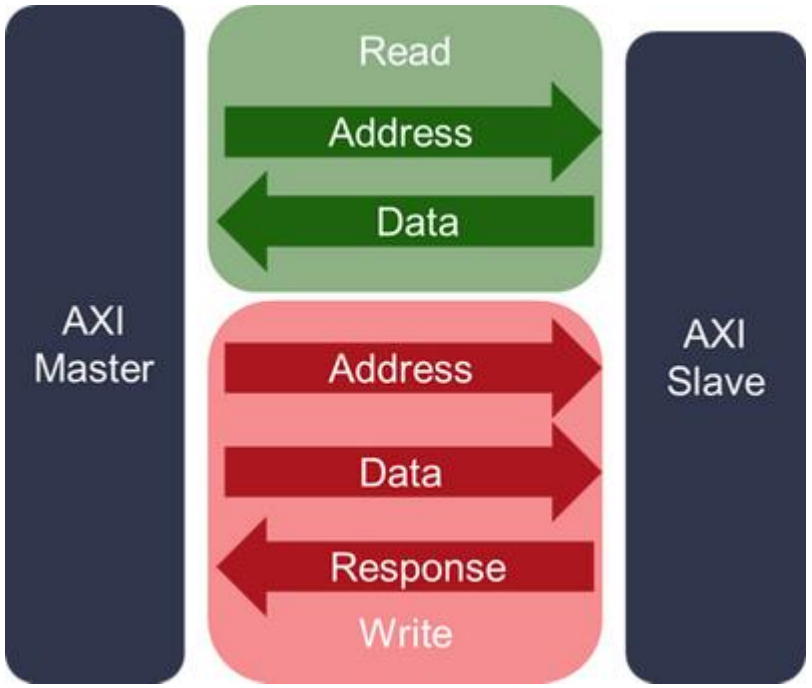
AXI (Advanced eXtensible Interface)

- ✓ AXI4 interface is divided into three types:
  1. AXI4-Full
  2. AXI4-Lite
  3. AXI4-Stream

- ✓ XILINX Application
  1. AXI Interconnect
  2. AXI SmartConnect



Channel

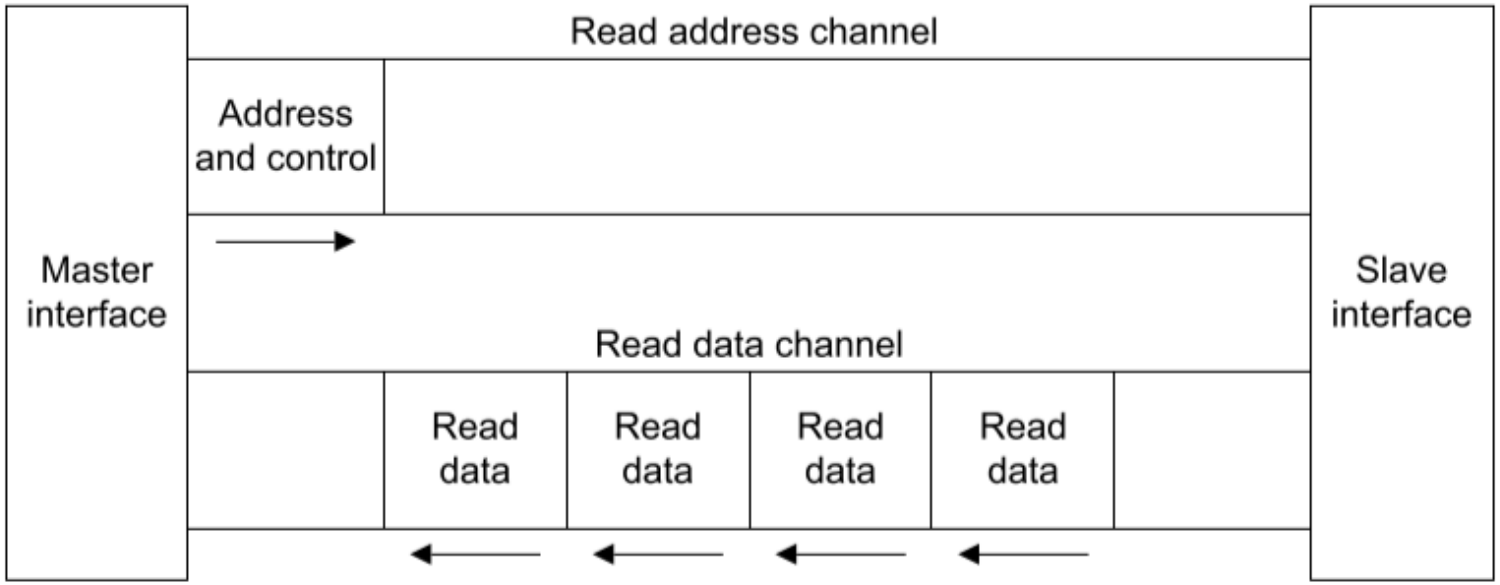


Signal / Channel	VALID	READY	LAST
Read Address (AR)	V	V	X
Read Data (R)	V	V	V
Write Address (AW)	V	V	X
Write Data (W)	V	V	V
Write Response (R)	V	V	X

Read transfer transactions

Read Address Channel Signals	**AR**	ARADDR	ARREADY	ARVALID	ARBURST	ARLEN	ARSIZE

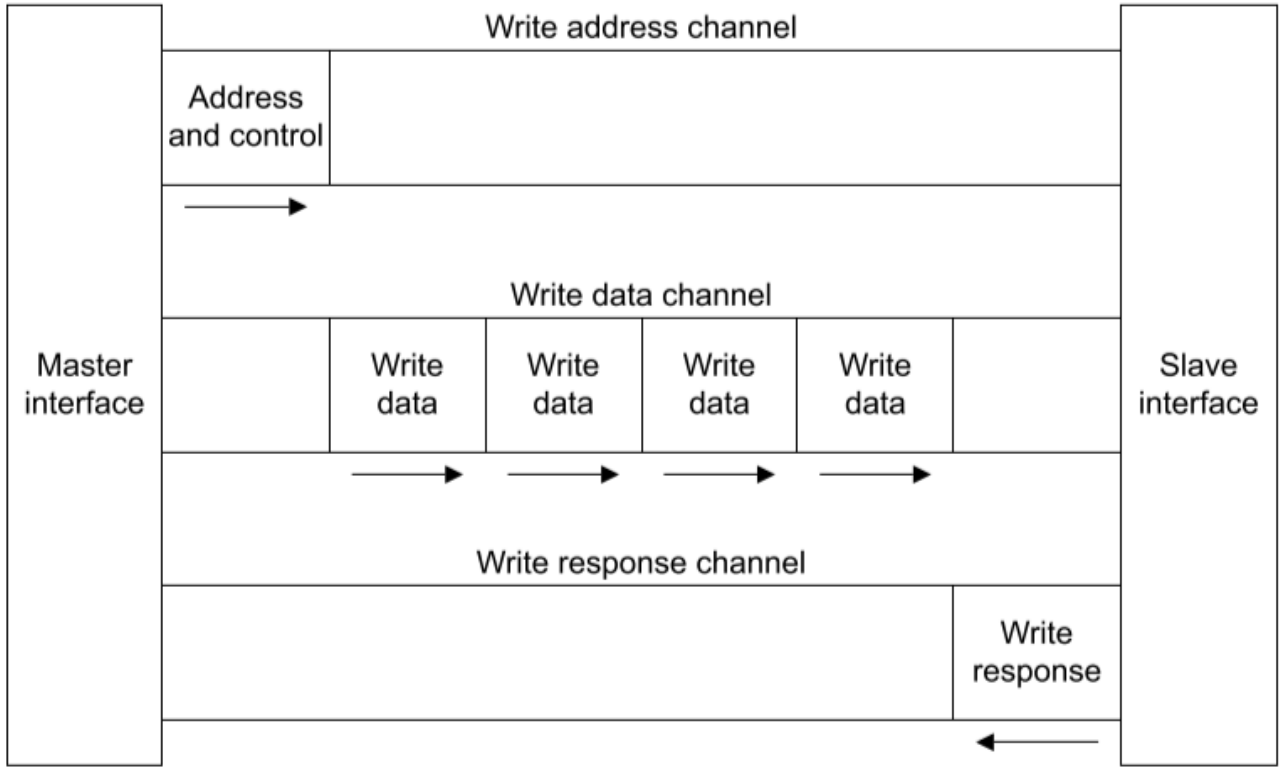
Read Data Channel Signals	**R**	RDATA	RREADY	RVALID	RRESP	RLAST



Write transfer transactions

Write Address Channel Signals	**AW**	AWADDR	AWREADY	AWVALID	AWBURST	AWLEN	AWSIZE

Write Data Channel Signals	**W**	WDATA	WREADY	WVALID	WSTRB	WLAST

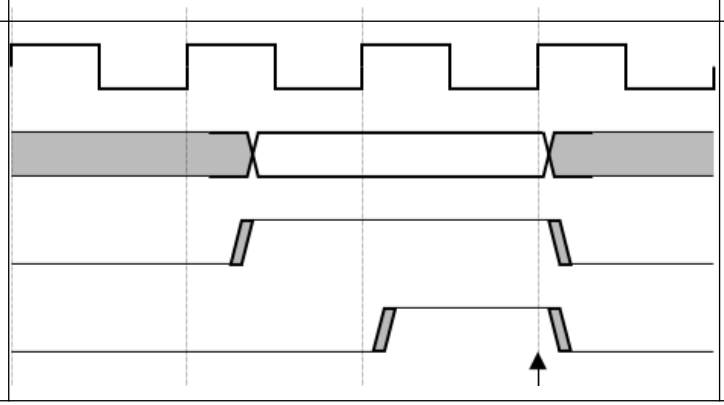
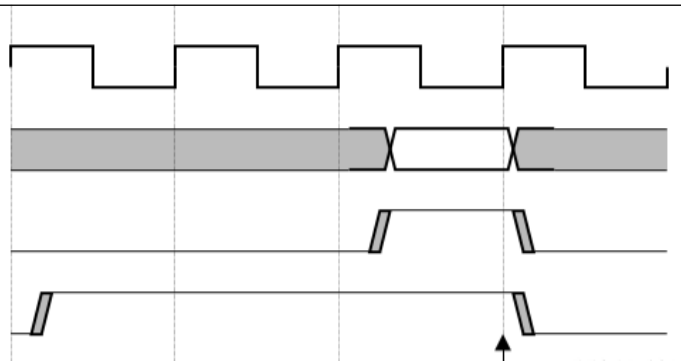
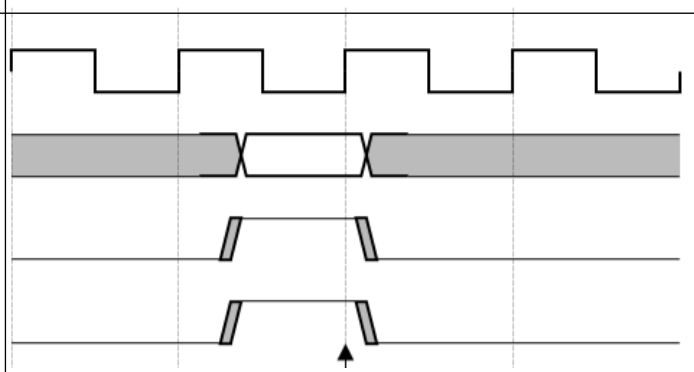


Write Response Channel Signals	**B**

BRESP	BREADY	BVALID

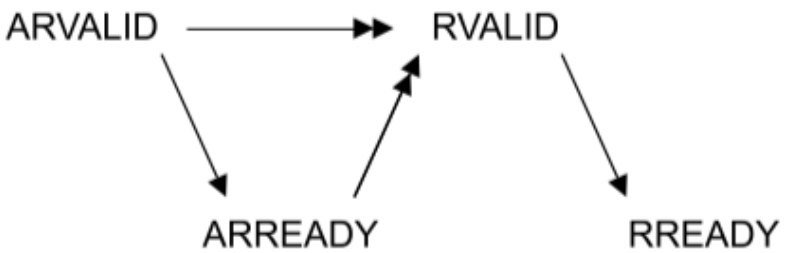
Handshake

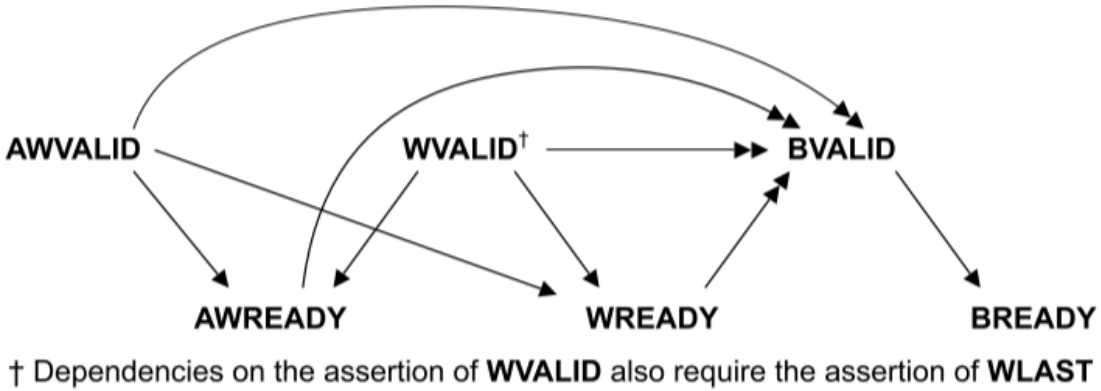
Transaction channel	Handshake pair	
Write address channel	AWVALID	AWREADY
Write data channel	WVALID	WREADY
Write response channel	BVALID	BREADY
Read address channel	ARVALID	ARREADY
Read data channel	RVALID	RREADY

Signal	Condition 1	Condition 2	Condition 3
ACLK INFORMATION TVALID TREADY			

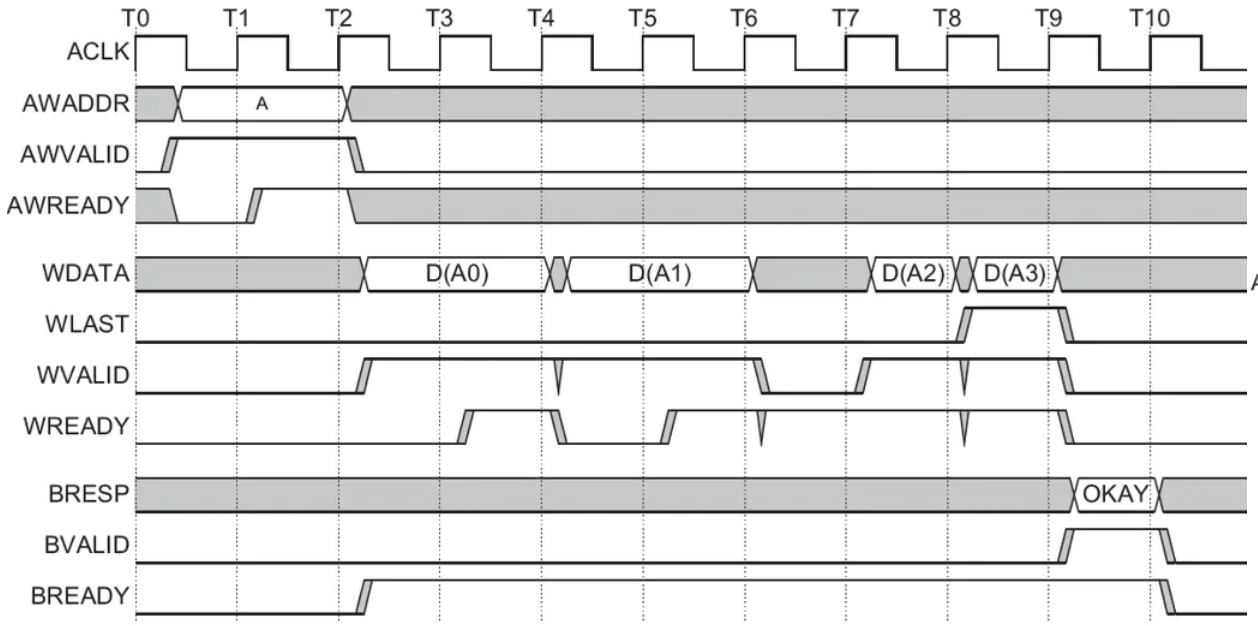
Dependence

- ✓ Write responses must occur after the completion of the last write data of the corresponding transfer
- ✓ Read data must be generated after receiving the read address signal
- ✓ Handshakes between channels must satisfy the handshake dependencies between channels
  1. The sender's VALID signal cannot rely on the READY signal to be set
  2. The receiver's READY signal can be set after detecting the VALID signal is set



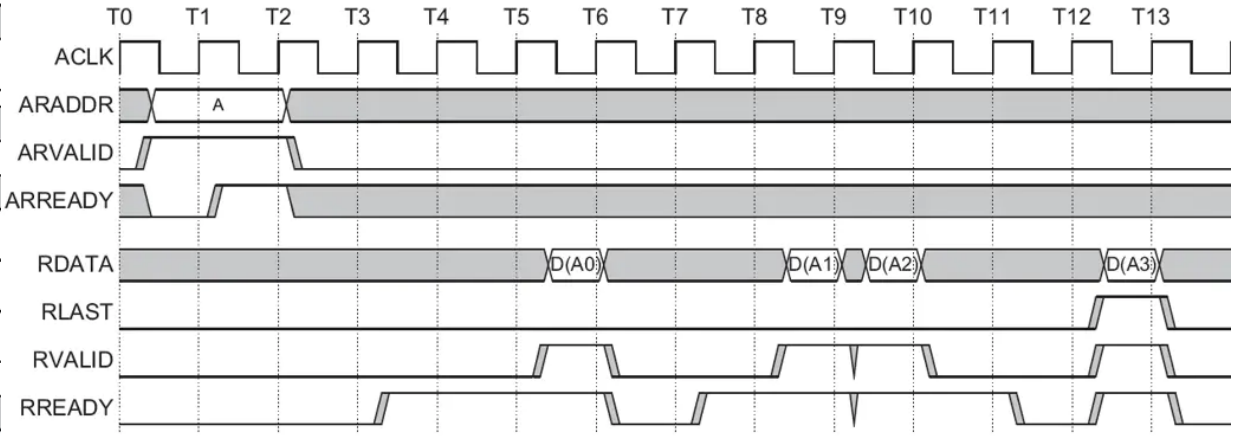


Write Burst



Source: ARM AMBA AXI Protocol v1.0: Specification

Read Burst



Source: ARM AMBA AXI Protocol v1.0: Specification

Burst

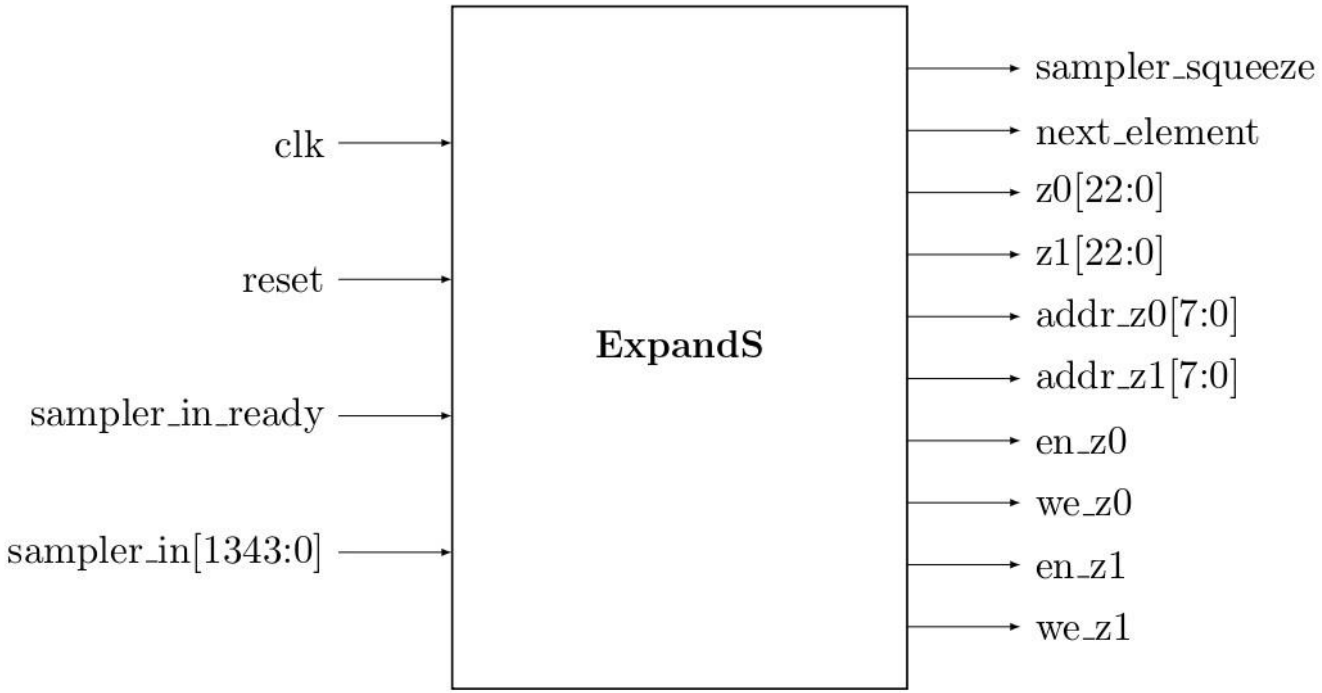
AxSIZE[2:0]	Bytes in transfer
0b000	1
0b001	2
0b010	4
0b011	8
0b100	16
0b101	32
0b110	64
0b111	128

AxBURST[1:0]	Burst type
0b00	FIXED
0b01	INCR
0b10	WRAP
0b11	Reserved

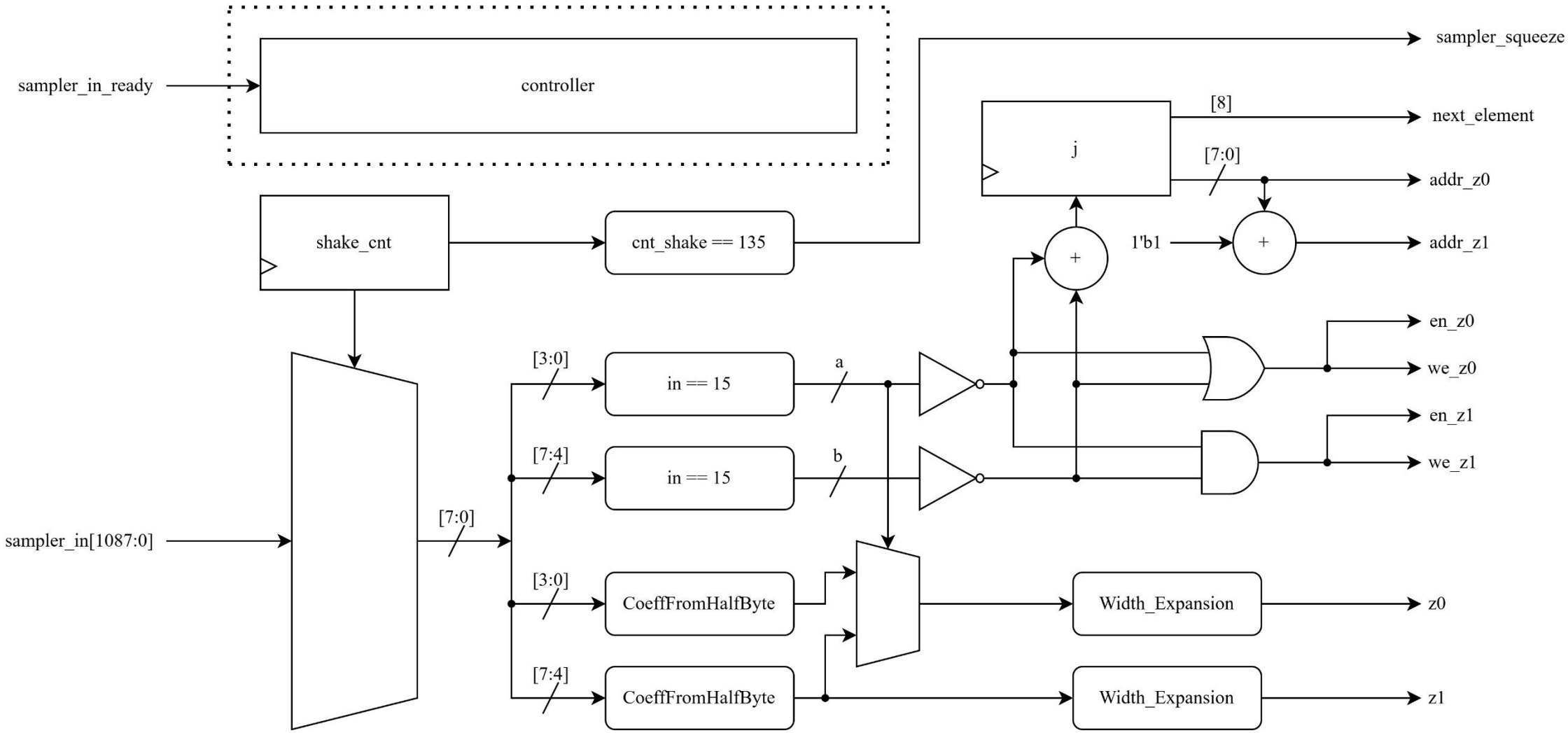
Sampler

ExpandS

- ✓ Sample the required private keys s1 and s2 from the equation t = A * s1 + s2
- ✓ The mode selection in the Sampler module is set to S_mode (0)
- ✓ The input sampled data is obtained through the H_mode (SHAKE256) in the Keccak module
- ✓ The output sampled data is stored in the s0 or s1 memory of Data_Mem, controlled by mem_sel and s_sel



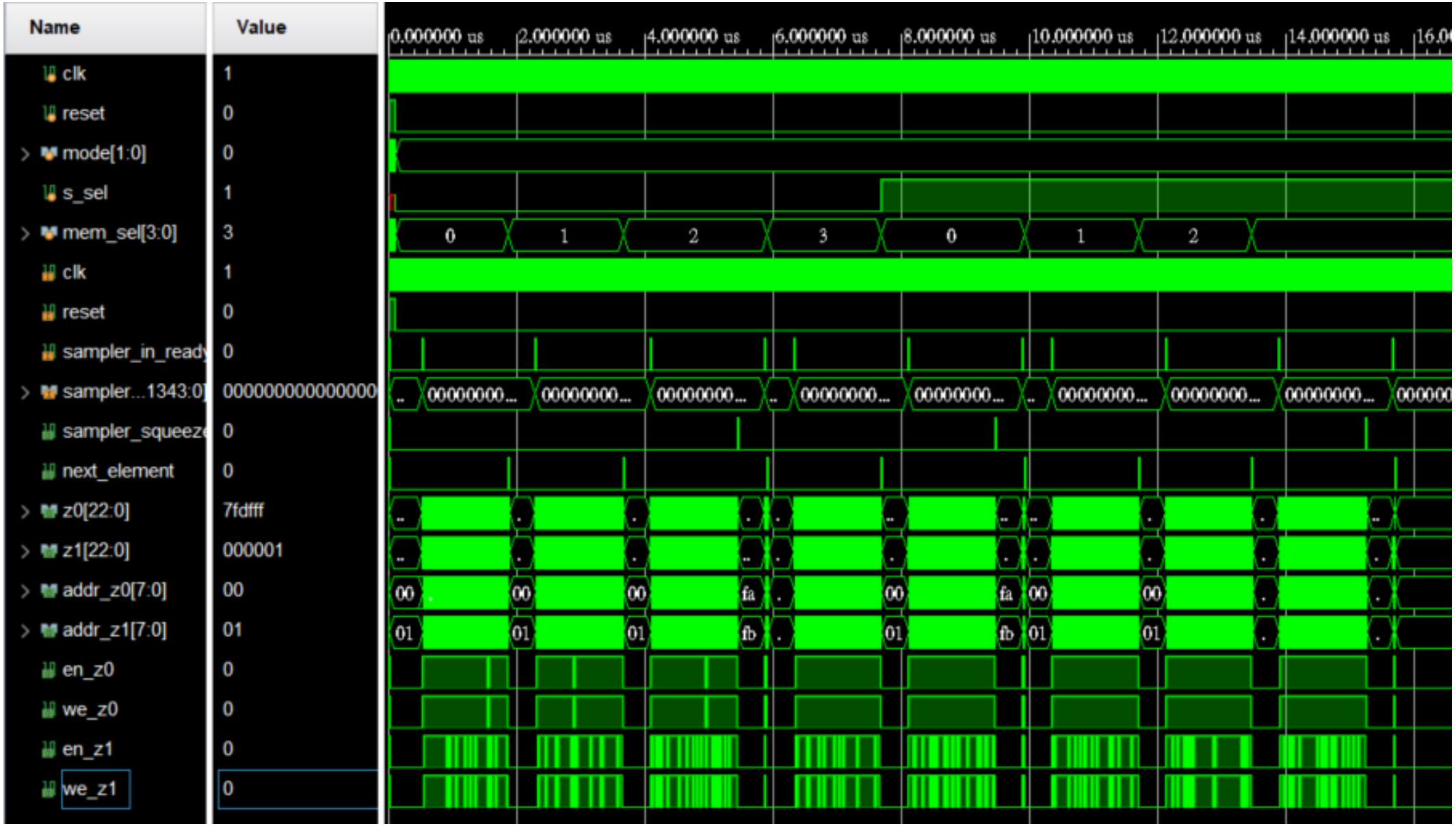
ExpandS – Block Diagram



ExpandS - Module Pin Function Definition

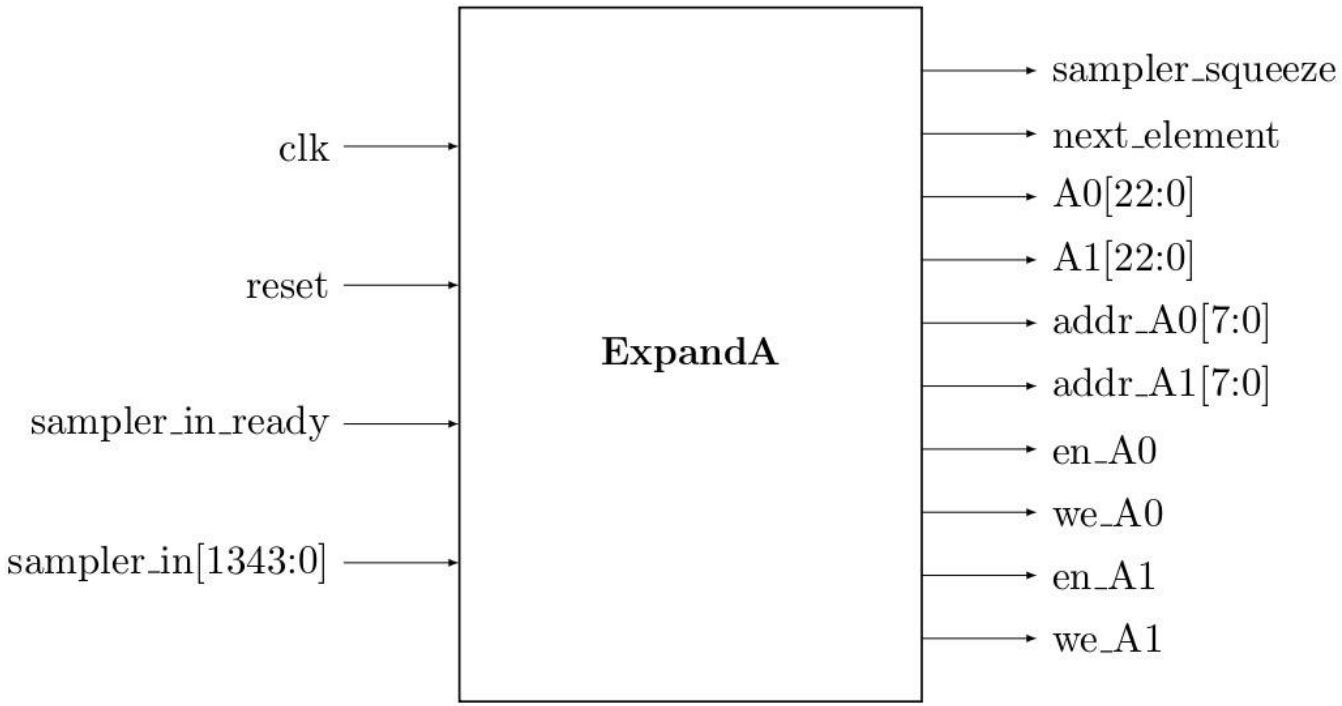
Pin name	I/O	Bit length	Function
clk	I	1	系統時脈
reset	I	1	系統同步重置信號
sampler_in_ready	I	1	Keack模組資料準備好的指示信號
sampler_in	I	1344	Keack模組的輸入資料
sampler_squeeze	O	1	和Keack模組要求squeeze一組新的輸入資料
next_element	O	1	該element取樣完畢的指示信號 (j[8])
z0	O	23	z0取樣的資料輸出通道 (連接Data_Mem中s0/s1 Mem)
z1	O	23	z1取樣的資料輸出通道 (連接Data_Mem中s0/s1 Mem)
addr_z0	O	8	z0取樣的位址輸出通道 (連接Data_Mem中s0/s1 Mem)
addr_z1	O	8	z1取樣的位址輸出通道 (連接Data_Mem中s0/s1 Mem)
en_z0	O	1	z0取樣的致能信號 (連接Data_Mem中s0/s1 Mem)
we_z0	O	1	z0取樣的寫入致能信號 (連接Data_Mem中s0/s1 Mem)
en_z1	O	1	z1取樣的致能信號 (連接Data_Mem中s0/s1 Mem)
we_z1	O	1	z1取樣的寫入致能信號 (連接Data_Mem中s0/s1 Mem)

ExpandS – Timing Diagram

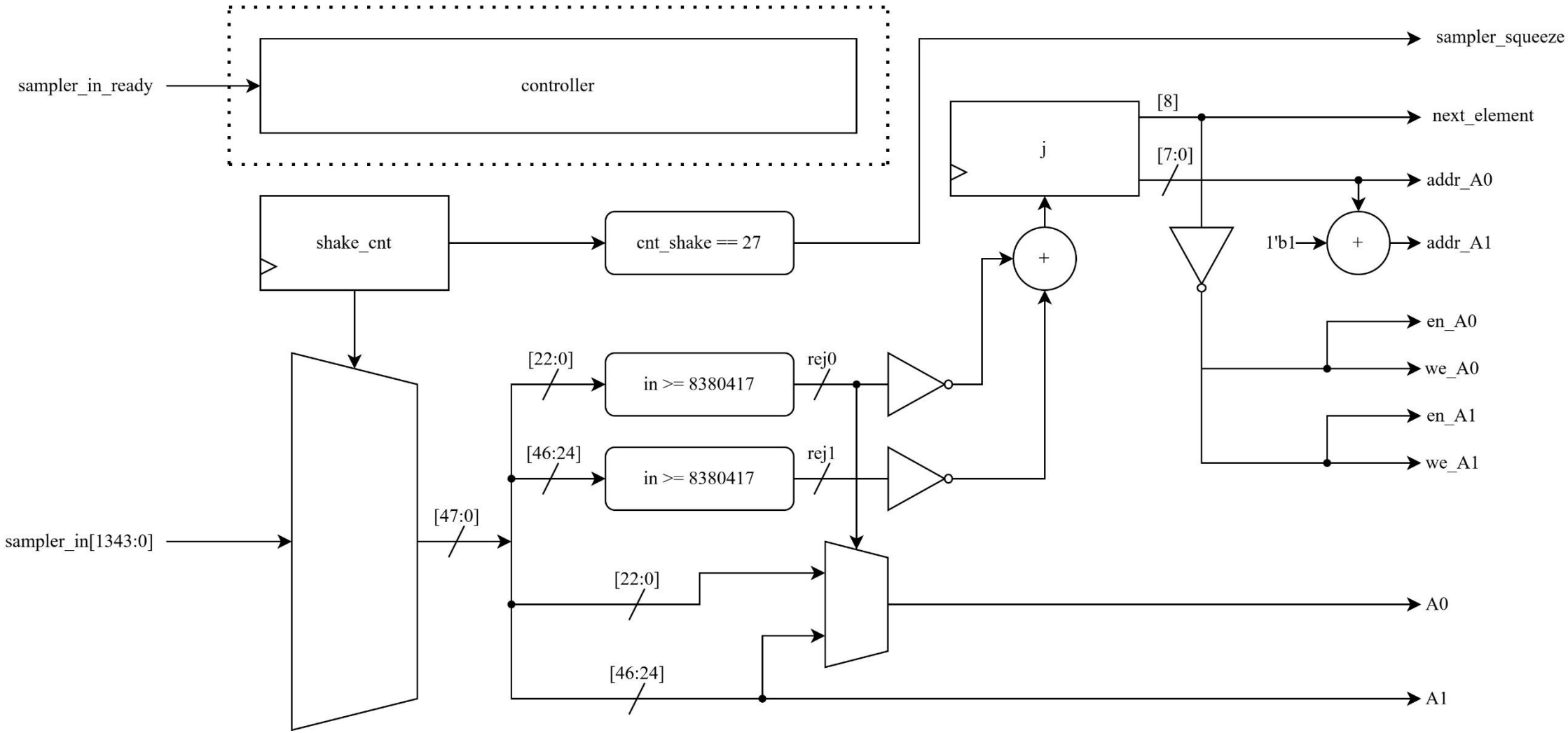


ExpandA

- ✓ Sample the required public keys A from the equation t = A * s1 + s2
- ✓ The mode selection in the Sampler module is set to A_mode (1)
- ✓ The input sampled data is obtained through the G_mode (SHAKE128) in the Keccak module
- ✓ The output sampled data is stored in the A memory of Data_Mem, controlled by mem_sel



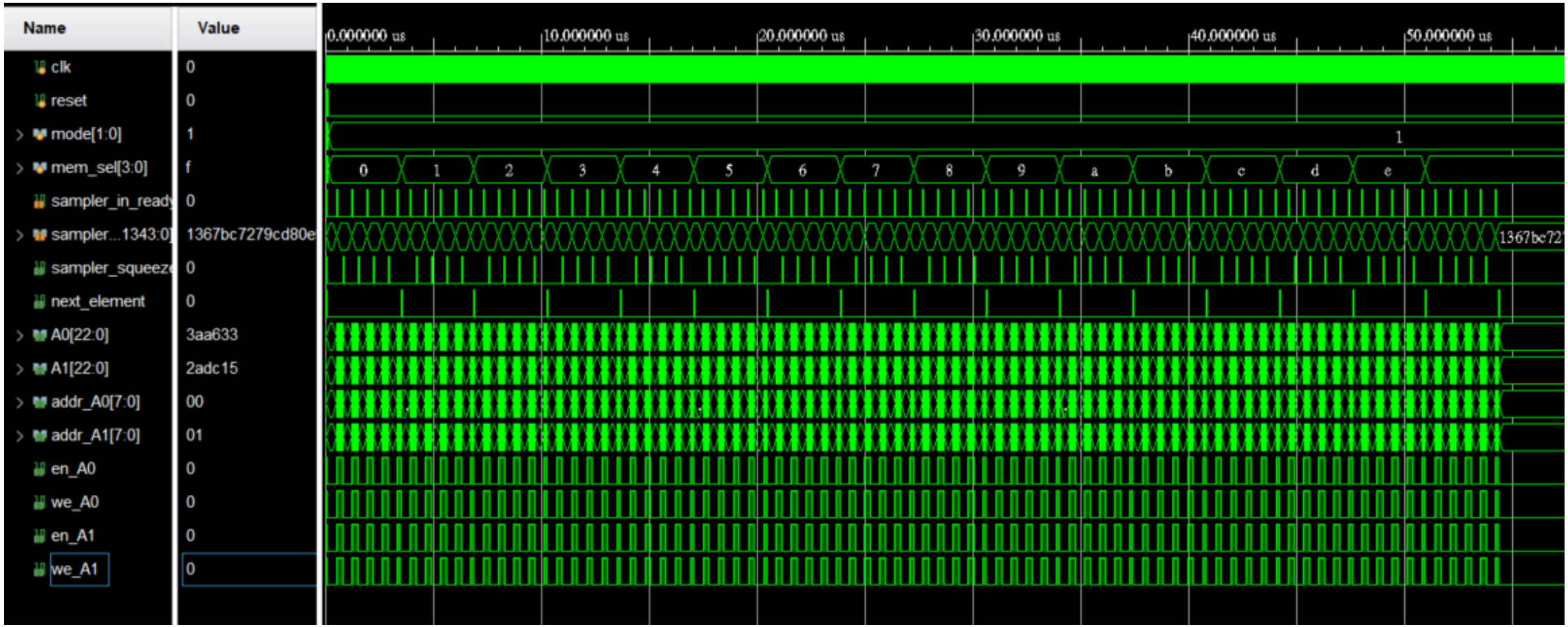
ExpandA – Block Diagram



ExpandA - Module Pin Function Definition

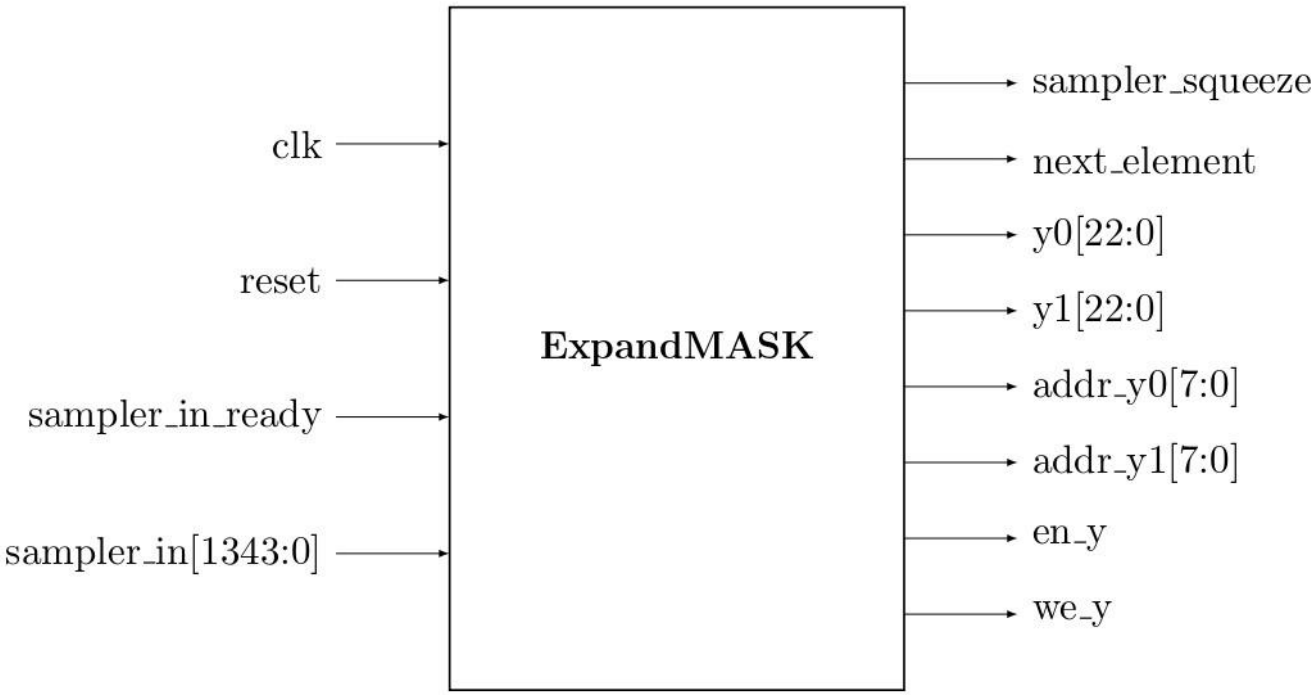
Pin name	I/O	Bit length	Function
clk	I	1	系統時脈
reset	I	1	系統同步重置信號
sampler_in_ready	I	1	Keack模組資料準備好的指示信號
sampler_in	I	1344	Keack模組的輸入資料
sampler_squeeze	O	1	和Keack模組要求squeeze一組新的輸入資料
next_element	O	1	該element取樣完畢的指示信號 (j[8])
A0	O	23	A0取樣的資料輸出通道 (連接Data_Mem中A Mem)
A1	O	23	A1取樣的資料輸出通道 (連接Data_Mem中A Mem)
addr_A0	O	8	A0取樣的位址輸出通道 (連接Data_Mem中A Mem)
addr_A1	O	8	A1取樣的位址輸出通道 (連接Data_Mem中A Mem)
en_A0	O	1	A0取樣的致能信號 (連接Data_Mem中A Mem)
we_A0	O	1	A0取樣的寫入致能信號 (連接Data_Mem中A Mem)
en_A1	O	1	A1取樣的致能信號 (連接Data_Mem中A Mem)
we_A1	O	1	A1取樣的寫入致能信號 (連接Data_Mem中A Mem)

ExpandA – Timing Diagram

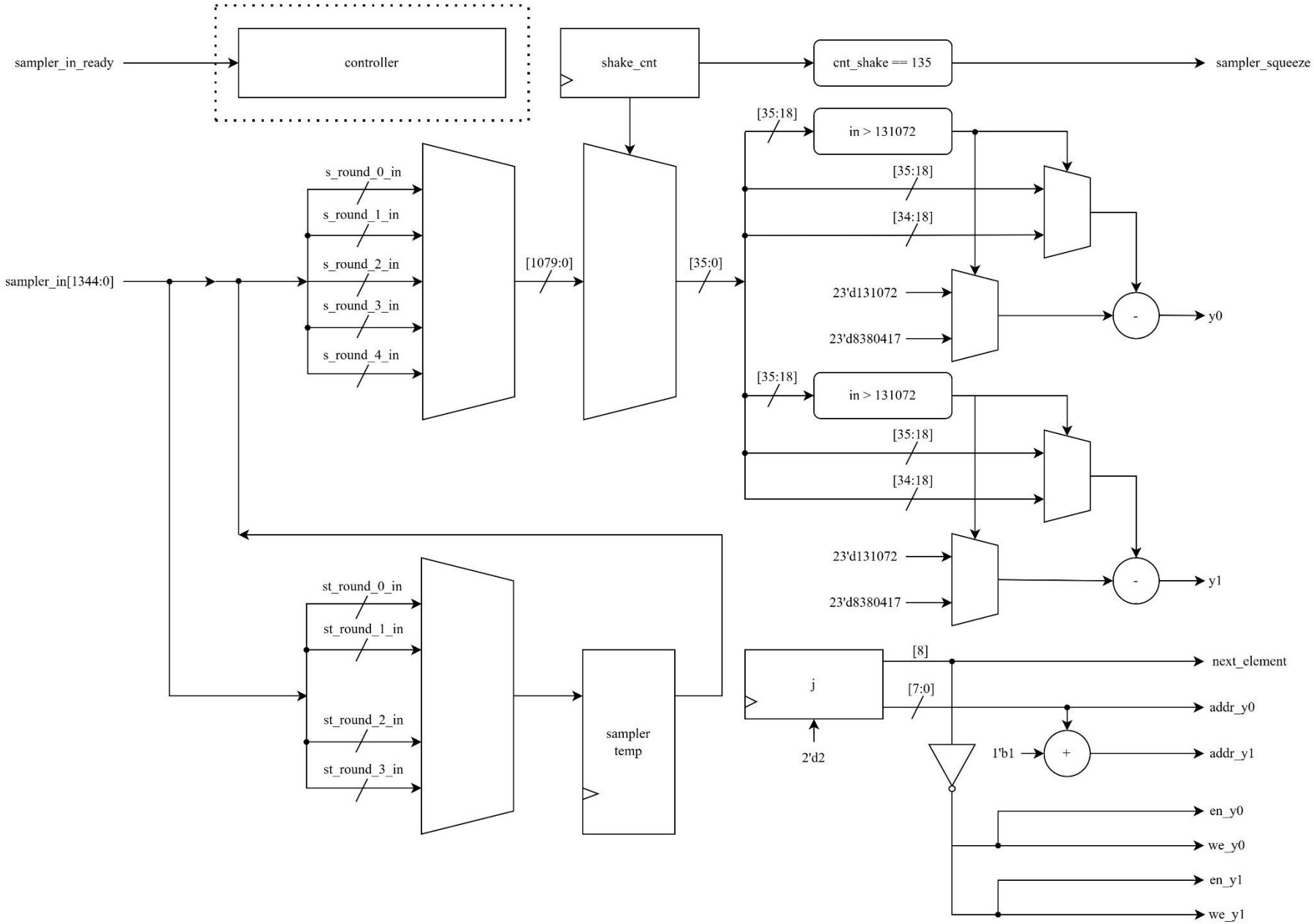


ExpandMASK

- ✓ Sample random vector y for computing the commitment value w = A * y
- ✓ The mode selection in the Sampler module is set to A_mode (2)
- ✓ The input sampled data is obtained through the H_mode (SHAKE256) in the Keccak module
- ✓ The output sampled data is stored in the y memory of Data_Mem, controlled by mem_sel



ExpandMASK – Block Diagram



Sign :

s_round_0_in :
sampler_in[1079:0]

s_round_1_in :
{sampler_in[1079 - round_0:0], sampler_temp[8-1:0]}

s_round_2_in :
{sampler_in[1079 - round_1:0], sampler_temp[16-1:0]}

s_round_3_in :
{sampler_in[1079 - round_2:0], sampler_temp[24-1:0]}

s_round_4_in :
{792'd0, sampler_in[255:0], sampler_temp[32-1:0]}:

st_round_0_in :
{{(32 - 8){1'b0}}, sampler_in[1088 - 1: 1088 - 8]}

st_round_1_in :
{{(32 - 16){1'b0}}, sampler_in[1088 - 1: 1088 - 16]}

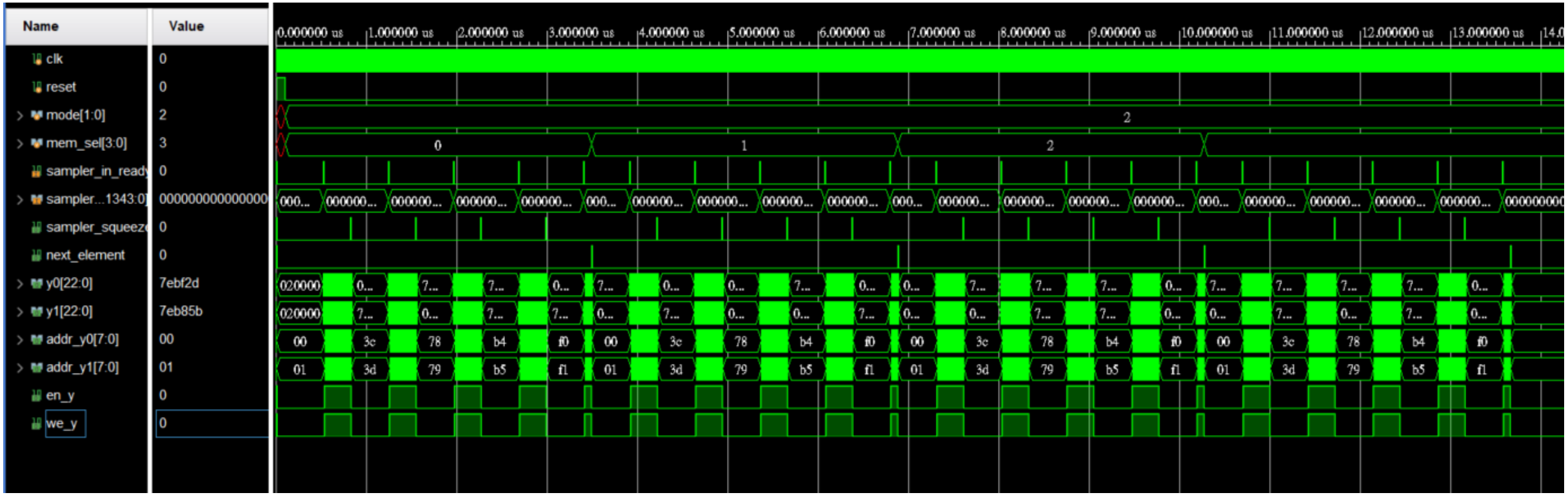
st_round_2_in :
{{(32 - 24){1'b0}}, sampler_in[1088 - 1: 1088 - 24]}

st_round_3_in :
{{(32 - 32){1'b0}}, sampler_in[1088 - 1: 1088 - 32]}

ExpandMASK - Module Pin Function Definition

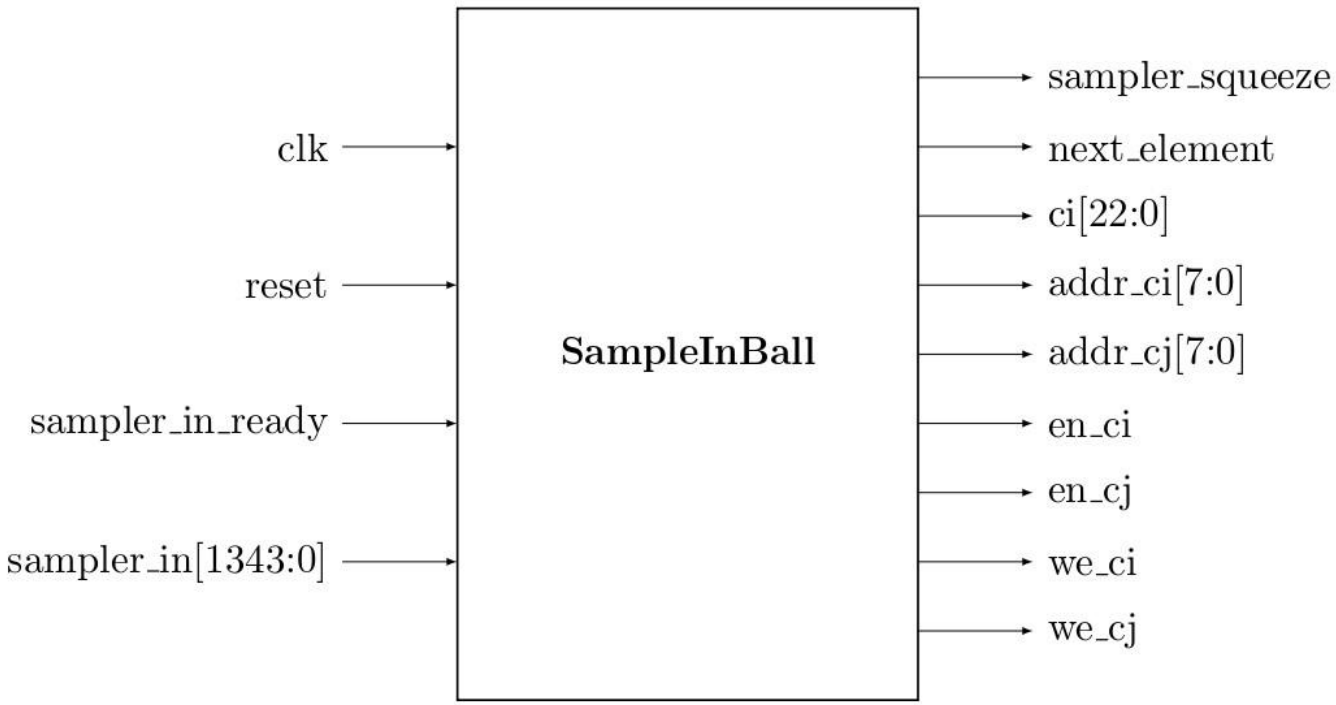
Pin name	I/O	Bit length	Function
clk	I	1	系統時脈
reset	I	1	系統同步重置信號
sampler_in_ready	I	1	Keack模組資料準備好的指示信號
sampler_in	I	1344	Keack模組的輸入資料
sampler_squeeze	O	1	和Keack模組要求squeeze一組新的輸入資料
next_element	O	1	該element取樣完畢的指示信號 (j[8])
y0	O	23	y0取樣的資料輸出通道 (連接Data_Mem中y Mem)
y1	O	23	y1取樣的資料輸出通道 (連接Data_Mem中y Mem)
addr_y0	O	8	y0取樣的位址輸出通道 (連接Data_Mem中y Mem)
addr_y1	O	8	y1取樣的位址輸出通道 (連接Data_Mem中y Mem)
en_y0	O	1	y0取樣的致能信號 (連接Data_Mem中y Mem)
we_y0	O	1	y0取樣的寫入致能信號 (連接Data_Mem中y Mem)
en_y1	O	1	y1取樣的致能信號 (連接Data_Mem中y Mem)
we_y1	O	1	y1取樣的寫入致能信號 (連接Data_Mem中y Mem)

ExpandMASK – Timing Diagram



SampleInBall

- ✓ Sample challenge cc based on the commitment $w1$ and the message to be signed, represented by μ
- ✓ The mode selection in the Sampler module is set to SIB_mode (3)
- ✓ The input sampled data is obtained through the H_mode (SHAKE256) in the Keccak module
- ✓ The output sampled data is stored in the c memory of Data_Mem, controlled by mem_sel



SampleInBall

Algorithm 29 SampleInBall(ρ)

Samples a polynomial $c \in R$ with coefficients from $\{-1, 0, 1\}$ and Hamming weight $\tau \leq 64$.

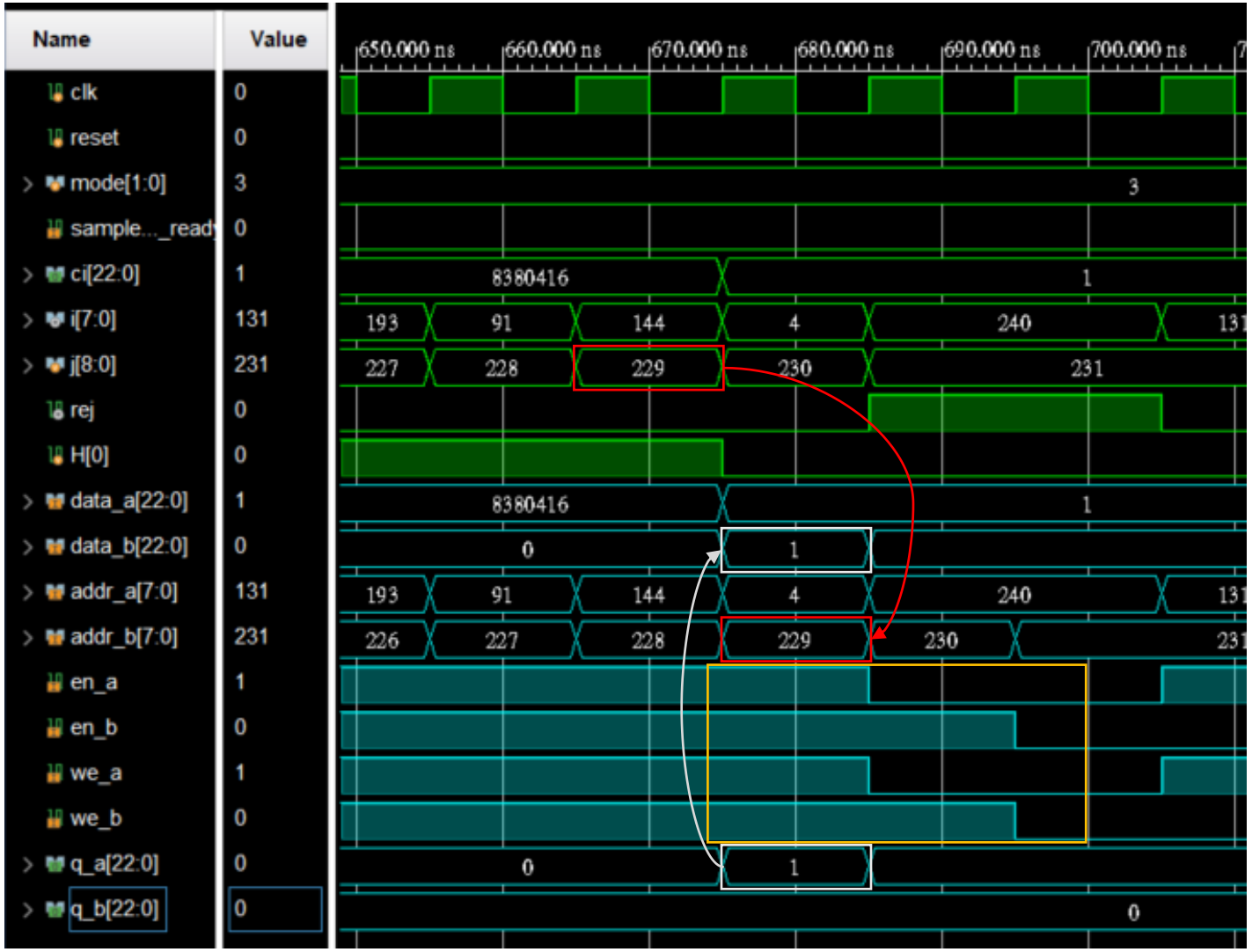
Input: A seed $\rho \in \mathbb{B}^{\lambda/4}$

Output: A polynomial c in R.

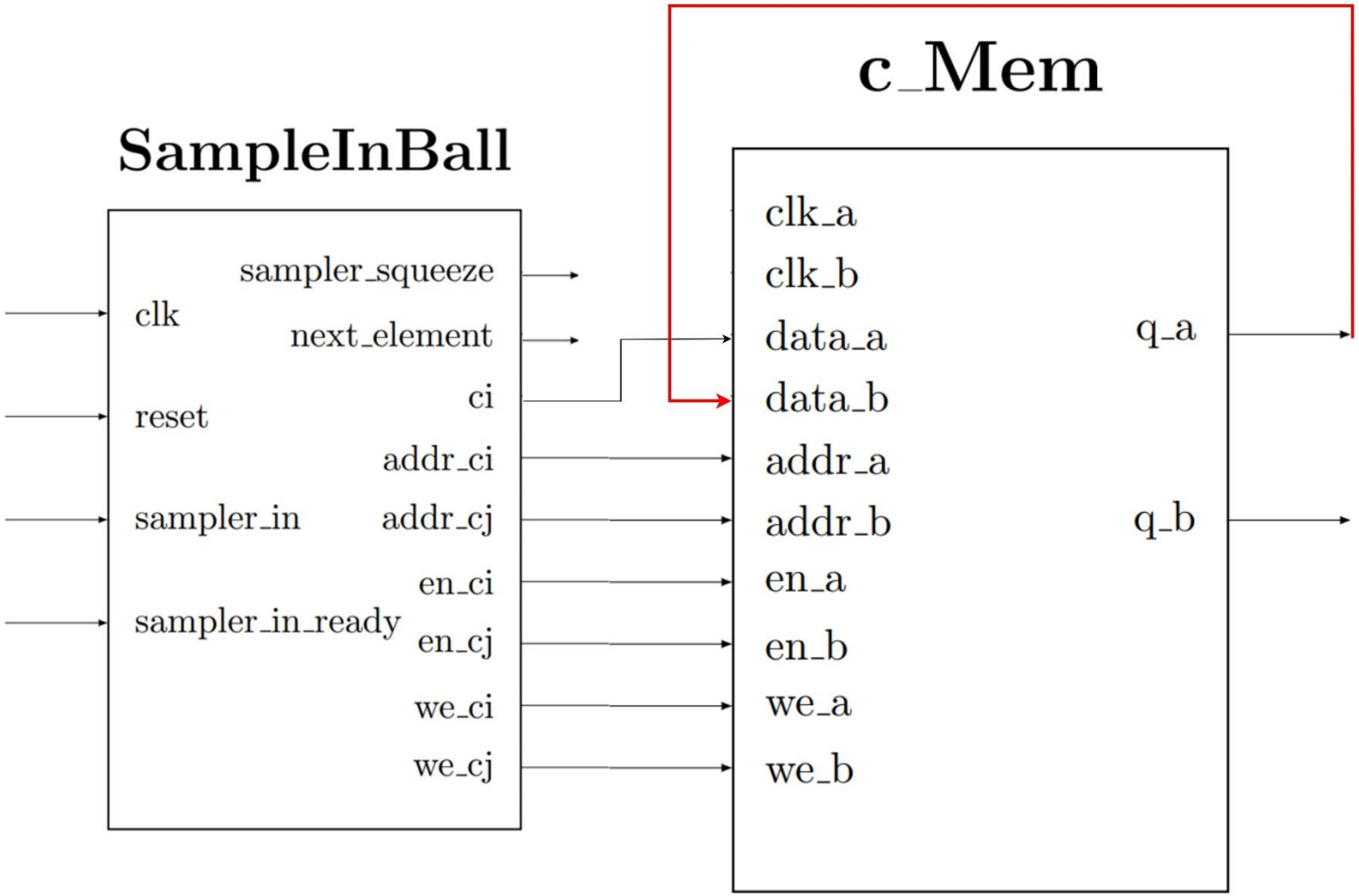
$c \leftarrow 0$
ctx $\leftarrow$ H.Init()
ctx $\leftarrow$ H.Absorb(ctx, ρ)
(ctx, s) $\leftarrow$ H.Squeeze(ctx, 8)
$h \leftarrow$ BytesToBits(s) ▷ h is a bit string of length 64
for i **from** $256 - \tau$ **to** 255 **do**
    (ctx, j) $\leftarrow$ H.Squeeze(ctx, 1)
    while $j > i$ **do** ▷ rejection sampling in $\{0, \ldots, i\}$
        (ctx, j) $\leftarrow$ H.Squeeze(ctx, 1)
    end while ▷ j is a pseudorandom byte that is $\leq i$
    $c_i \leftarrow c_j$
    $c_j \leftarrow (-1)^{h[i+\tau-256]}$
end for
return c

SampleInBall

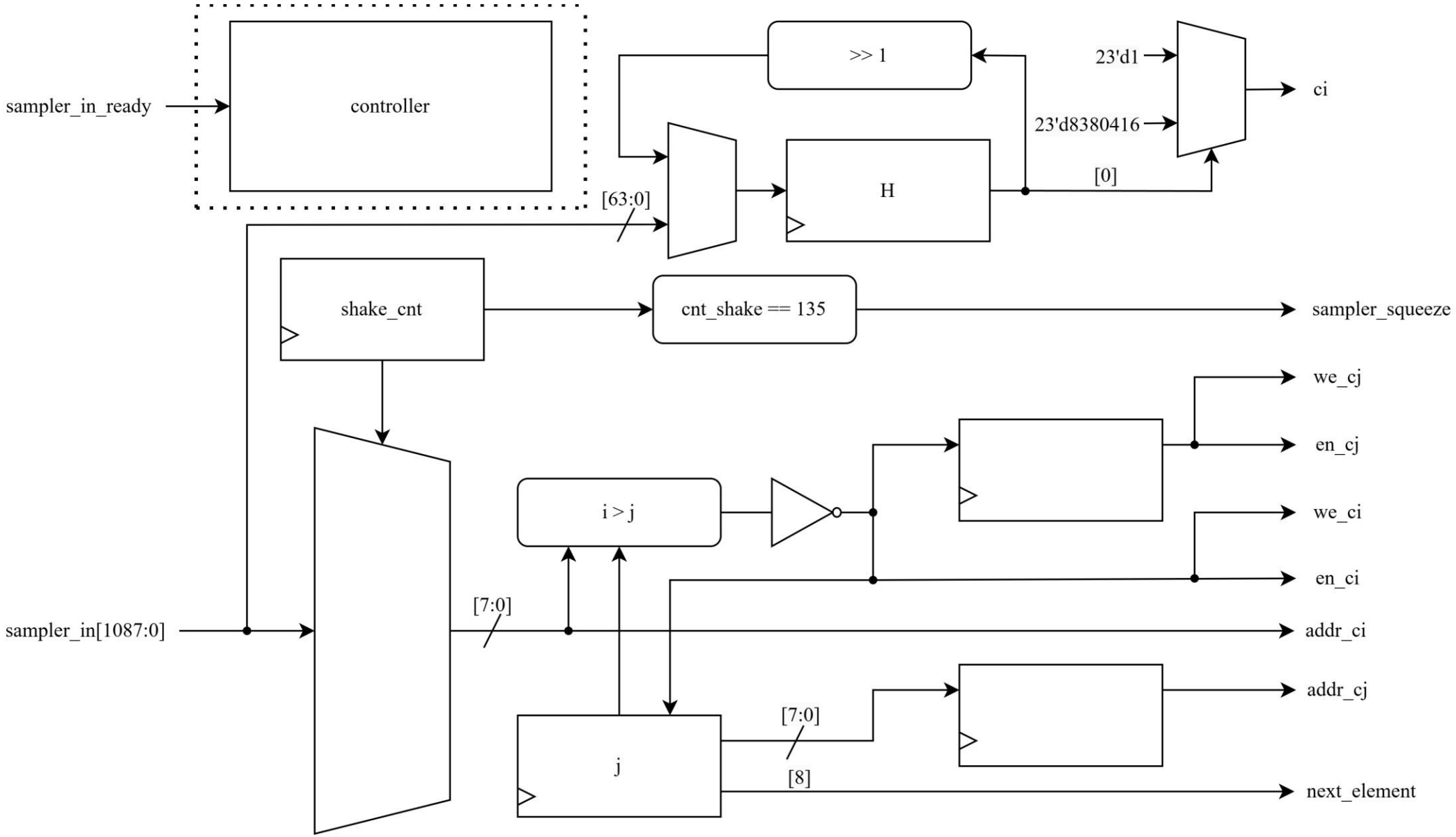
```
always @(posedge clk) begin
    if (reset)
        addr_cj <=  8'd0;
    else
        addr_cj <= j;
end

always @(posedge clk) begin
    if (reset)
        en_cj <=  1'b0;
    else
        en_cj <= en_ci;
end

assign we_cj = en_cj;
```



SampleInBall



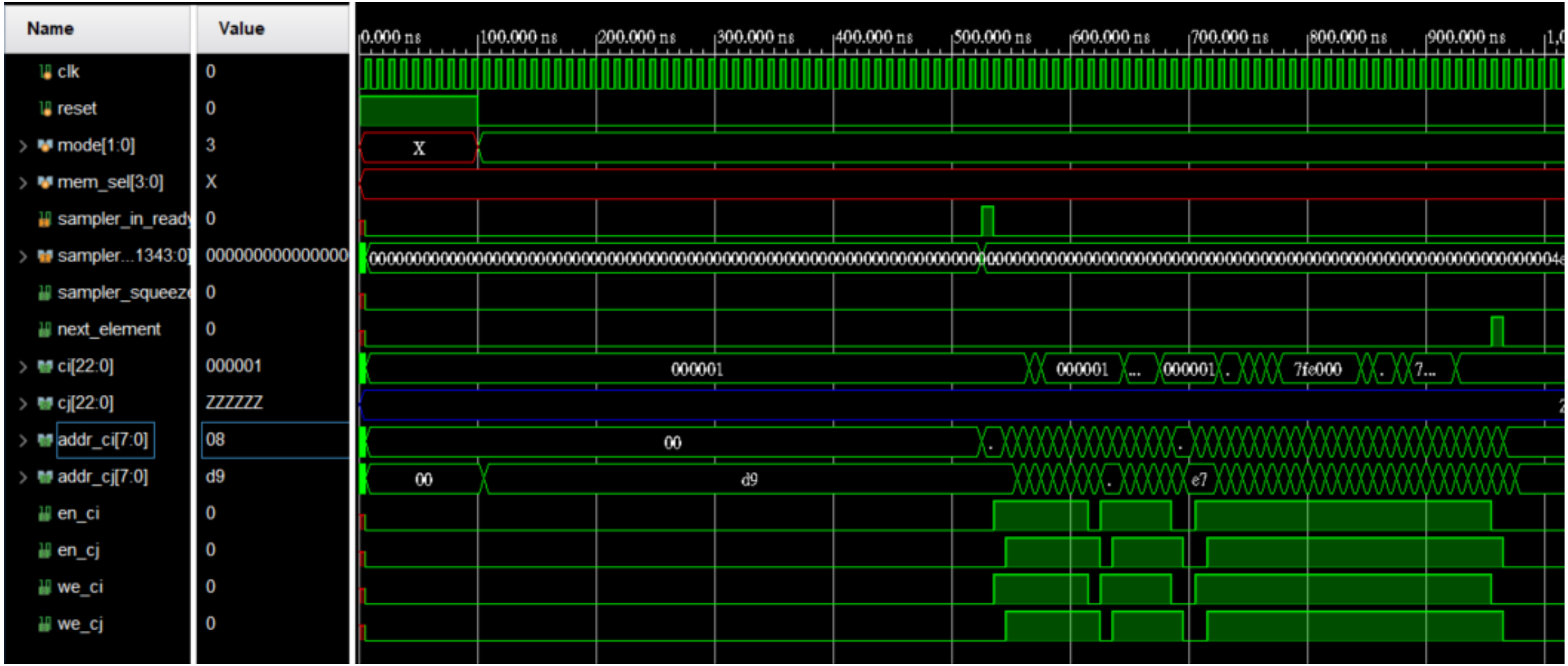
SampleInBall – Block Diagram



SampleInBall - Module Pin Function Definition

Pin name	I/O	Bit length	Function
clk	I	1	系統時脈
reset	I	1	系統同步重置信號
sampler_in_ready	I	1	Keack模組資料準備好的指示信號
sampler_in	I	1344	Keack模組的輸入資料
sampler_squeeze	O	1	和Keack模組要求squeeze一組新的輸入資料
next_element	O	1	該element取樣完畢的指示信號 (j[8])
ci	O	23	ci取樣的資料輸出通道 (連接Data_Mem中c Mem)
addr_ci	O	8	ci取樣的位址輸出通道 (連接Data_Mem中c Mem)
addr_cj	O	8	cj取樣的位址輸出通道 (連接Data_Mem中c Mem)
en_ci	O	1	ci取樣的致能信號 (連接Data_Mem中c Mem)
we_cj	O	1	ci取樣的寫入致能信號 (連接Data_Mem中c Mem)
en_ci	O	1	cj取樣的致能信號 (連接Data_Mem中c Mem)
we_cj	O	1	cj取樣的寫入致能信號 (連接Data_Mem中c Mem)

SampleInBall – Timing Diagram



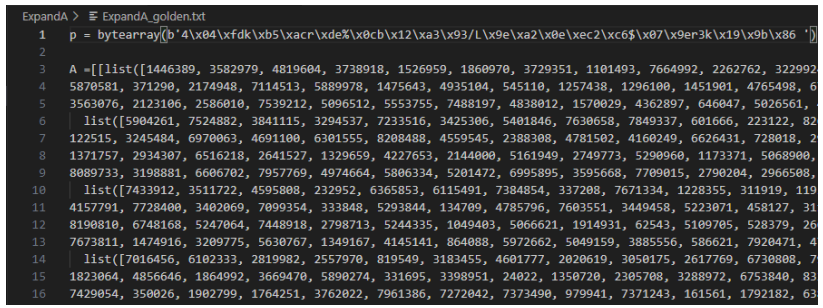
How to Use a Testing Platform

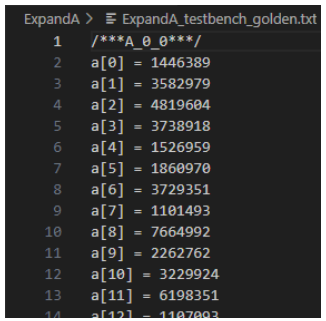
- ✓ Folder Structure :

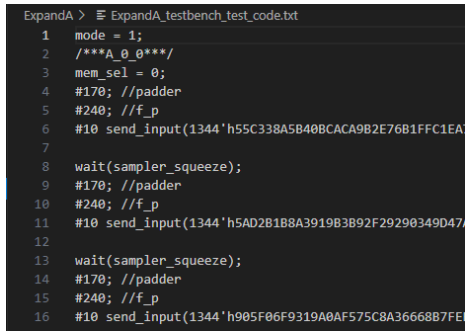
  ML_DSA_syn → Module_Test_py → module_name (ex. ExpandA, SampleInBall)

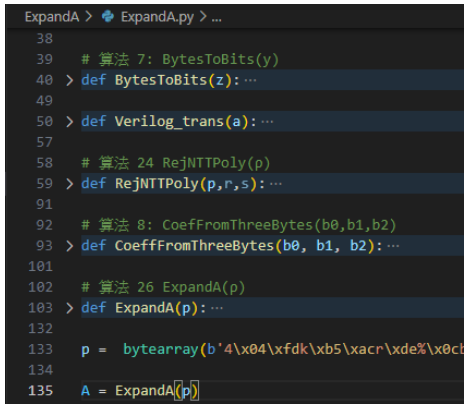
- ✓ File :

  1. module_name_golden.txt → Generate test data from ML_DSA_44_excellent_final_clean.py
  2. module_name_testbench_golden.txt → Compare with the Memory storage results in the TestBench.
  3. module_name_testbench_test_code.txt → Generate Verilog for TestBench testing from module_name.py
  4. module_name.py → Main test Python file





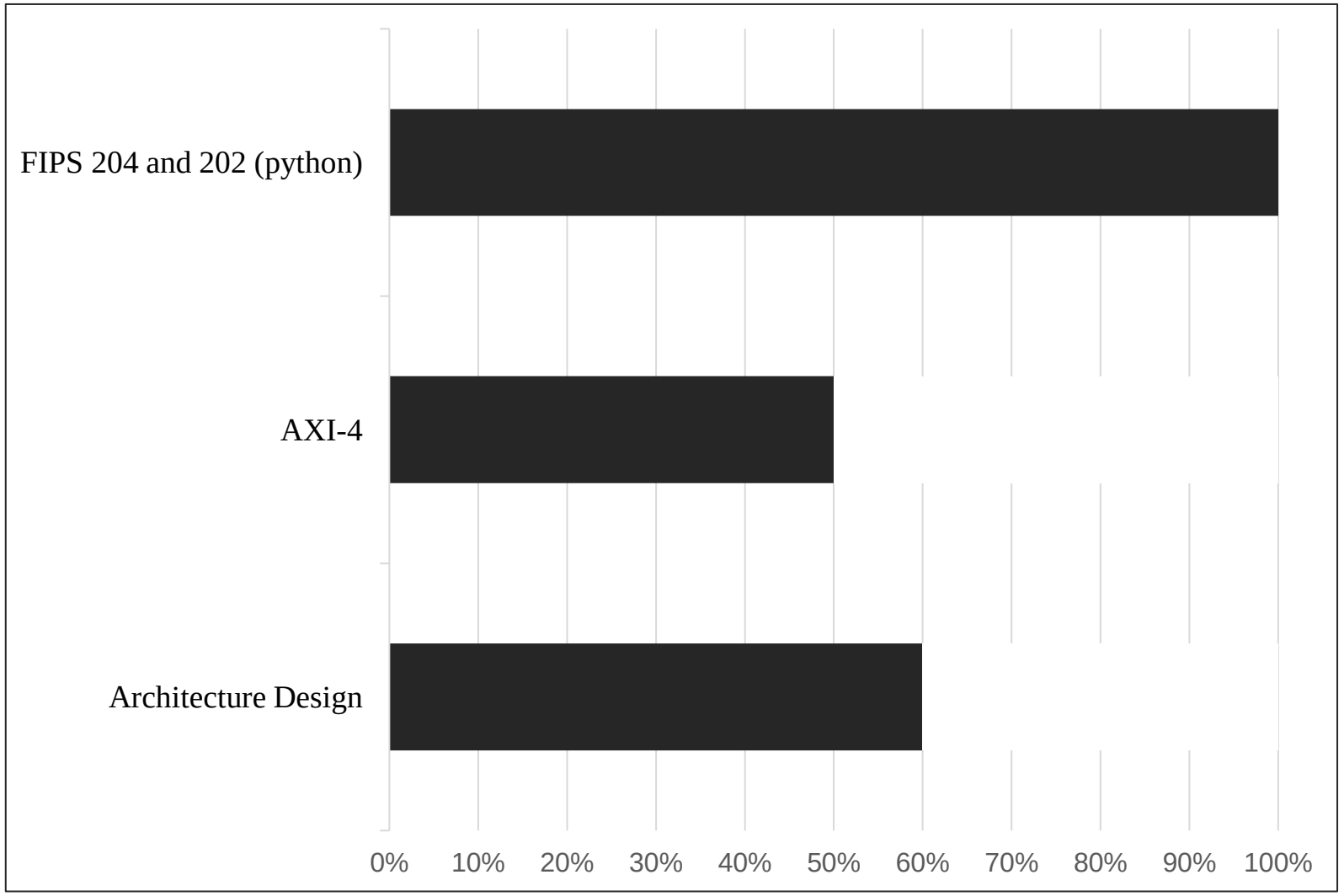




07

Current progress

Current progress



08

References

References

[1] “Module-Lattice-Based digital signature standard,” National Institute of Standards and Technology, Gaithersburg, MD, Aug. 2023, Accessed: Aug. 05, 2024. [Online]. Available: http://dx.doi.org/10.6028/nist.fips.204.ipd

[2] 蔡秉邕， Dilithium數位簽章系統的安全性估計，碩士論文，國立清華大學，2022。

[3] 黃彥霖，容錯學習密碼學的加密演算法與其實作，碩士論文，國立交通大學，2019。

[4] A. Satriawan, R. Mareta, and H. Lee, *A Complete Beginner Guide to the Number Theoretic Transform* (NTT), Dept. of Electrical and Computer Engineering, Inha University, Incheon, South Korea, Apr. 2024.

[5] K. Denisse Ortega L., and Luis J. Dominguez Perez, “Implementing crystal-dilithium on FRDM-K64,” in *Proceedings of the 2021 IEEE 12th Annual Ubiquitous Computing*, Electronics & Mobile Communication Conference (UEMCON), pp. 0178–0183, New York, NY, USA, Dec. 2021, Available: http://dx.doi.org/10.1109/uemcon53757.2021.9666622

[6] T.-H. Nguyen, B. Kieu-Do-Nguyen, C.-K. Pham, and T.-T. Hoang, “High-speed NTT accelerator for crystal-kyber and crystal-dilithium,” *IEEE Access*, vol. 12, pp. 34918–34930, Feb. 2024, Available: https://doi.org/10.1109/access.2024.3371581