Design and Evaluation of

POST-QUANTUM CRYPTOGRAPHY

Accelerator on FPGA

PROGRESS PRESENTATION

Members
Pakin Panawattanakul 6580043
Nitchayanin Thamkunanon 6580081
Panupong Sangaphunchai 6580587

Table of Contents

Current Progress

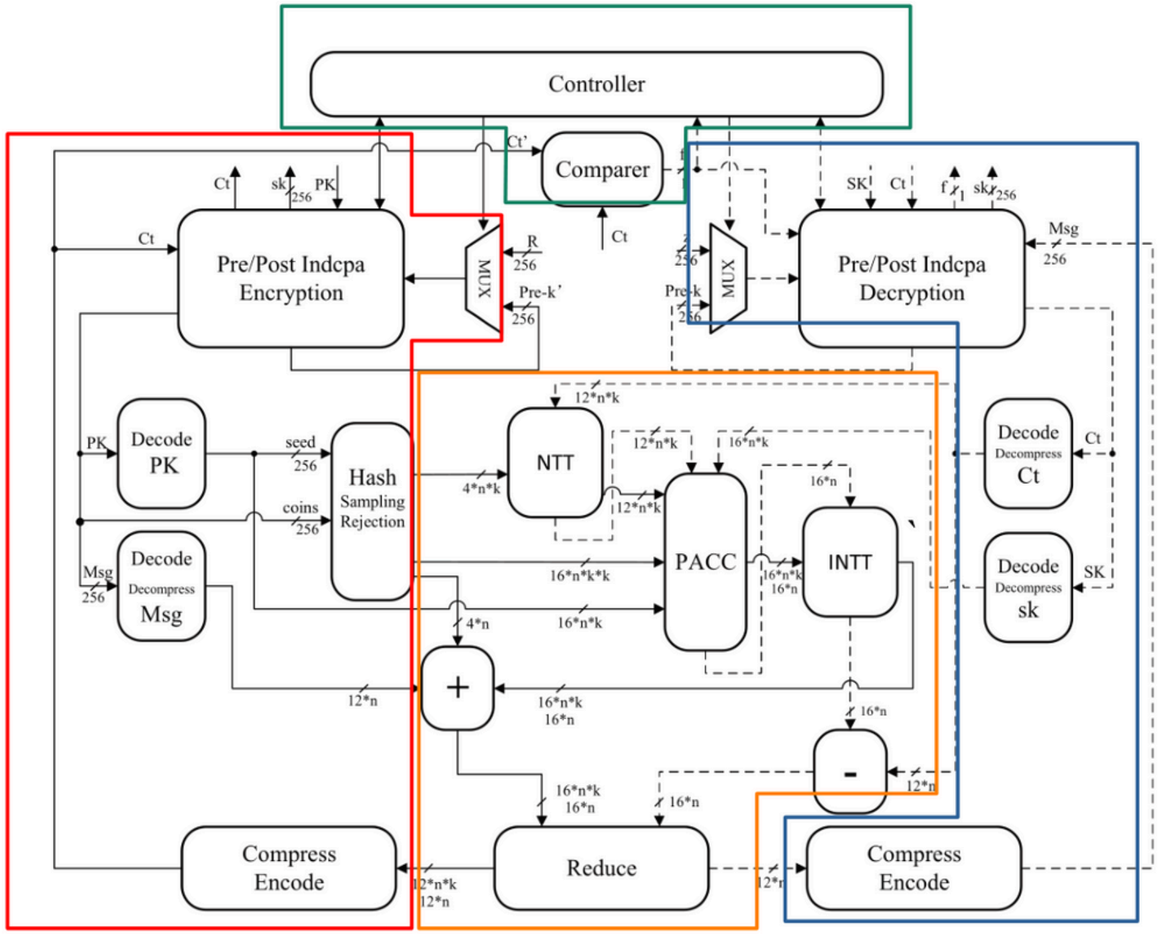


Fig. 3 Overall Kyber encapsulation and decapsulation architecture.

Modules sections

1) Control

2) Encapsulation

3) Decapsulation

4) Main computation

Note Article does not include detail implementation of each module

Current Progress

Encapsulation	Decapsulation	Main Computation
Decode PK	Pre/Post Indcpa Decryption	Addition
Decode msg	Decode Decompress SK	PACC
Hash (SHAKE+CBD+Rejection)	Decode Decompress Ct	INTT
Compress Encode	Compress Encode	NTT
Pre/Post Indcpa Encryption		Reduce
		Subtraction

Green = Done
Blue = In Progress
Red = Will start soon

Designing Philosophy

- Understanding official reference code or Documentations
- Implement in system verilog
- Verify the correctness using simulation compare to reference code

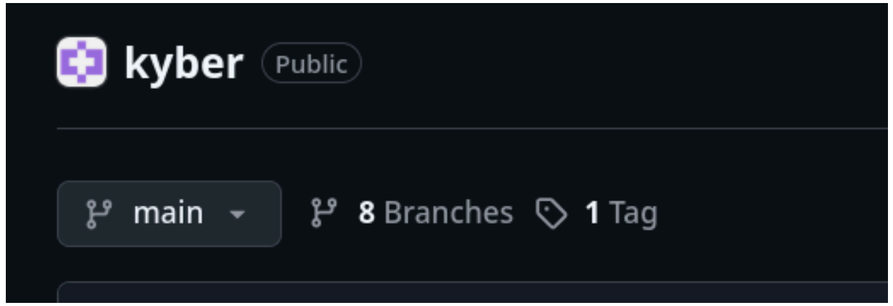
Reference codes/Documents

- Kyber github
- FIPS202 (Hash)
- FIPS203 (Key Encapsulation Mechanism)
- Kyber Specification Round 3 (Core Kyber design)


FIPS 203

Federal Information Processing Standards Publication

Module-Lattice-Based
Key-Encapsulation Mechanism Standard



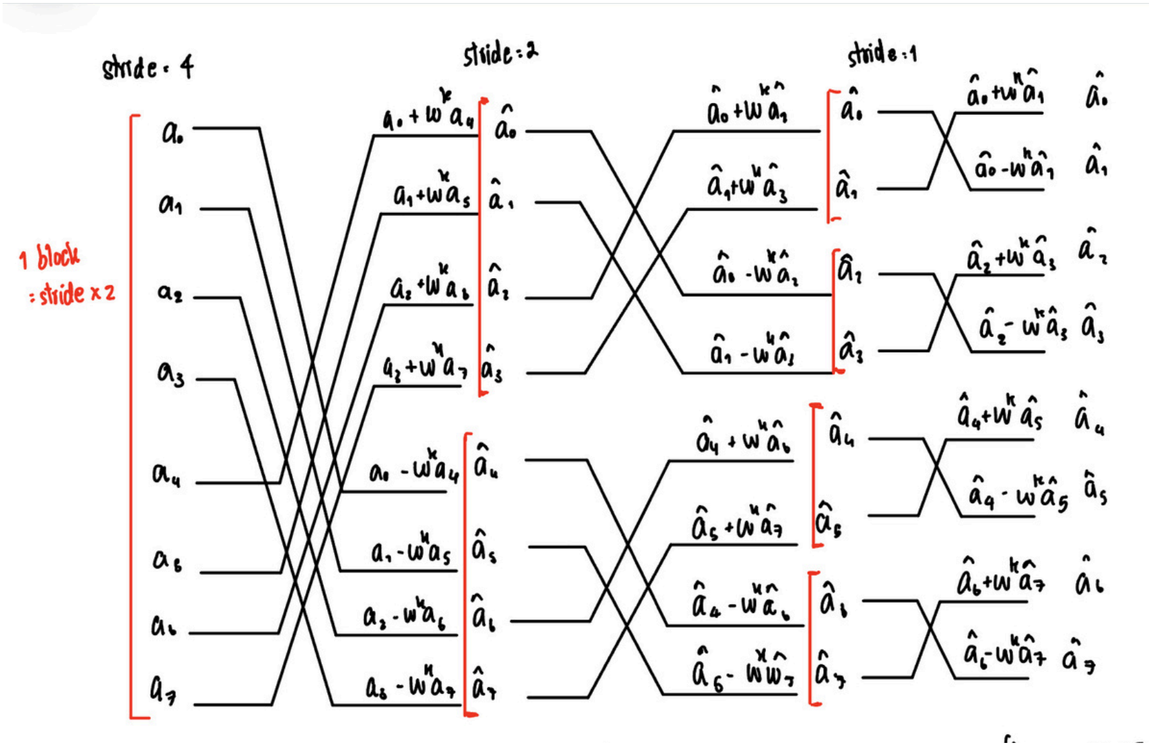
- Translating C code -> Verilog
- Similar loop structure to generate wire
- Add to MUX to reuse the same combinational circuit

```
reg [15:0] buf_in[0:255];      When an unpacked dimension range is zero-based ([0:N-1
wire [15:0] a[0:6][0:127];     When an unpacked dimension range is zero-based ([0:N
wire [15:0] b[0:6][0:127];     When an unpacked dimension range is zero-based ([0:N
// generate a,b for muxes
genvar s, start, j;
generate
  for (s = 0; s < 7; s = s + 1) begin : g_stride
    for (start = 0; start < 256; start = start + 2 * (128 >> s)) begin : outer     A
      for (j = start; j < start + (128 >> s); j = j + 1) begin : inner     All gener
        localparam int idx = (start / (2 * (128 >> s))) * (128 >> s) + (j - start);
        assign a[s][idx] = buf_in[j];
        assign b[s][idx] = buf_in[j+(128>>s)];
      end
    end
  end
endgenerate
```

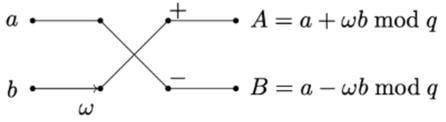
```
/*************************************************
* Name:        ntt
*
* Description: Inplace number-theoretic transform (NTT) in Rq.
*              input is in standard order, output is in bitreversed order
*
* Arguments:   - int16_t r[256]: pointer to input/output vector of elements of Zq
**************************************************/
void ntt(int16_t r[256]) {
  unsigned int len, start, j, k;
  int16_t t, zeta;

  k = 1;
  for(len = 128; len >= 2; len >>= 1) {
    for(start = 0; start < 256; start = j + len) {
      zeta = zetas[k++];
      for(j = start; j < start + len; j++) {
        t = fqmul(zeta, r[j + len]);
        r[j + len] = r[j] - t;
        r[j] = r[j] + t;
      }
    }
  }
}
```

```
  // add them to muxes
  genvar i;
  generate
    for (i = 0; i < 128; i = i + 1) begin     All generate block stateme
      ntt_mux7 mux_a (
          .in0(a[0][i]),
          .in1(a[1][i]),
          .in2(a[2][i]),
          .in3(a[3][i]),
          .in4(a[4][i]),
          .in5(a[5][i]),
          .in6(a[6][i]),
          .out(mux_out_a[i]),
          .sel(stage)
      );
      ntt_mux7 muxb (
          .in0(b[0][i]),
          .in1(b[1][i]),
          .in2(b[2][i]),
          .in3(b[3][i]),
          .in4(b[4][i]),
          .in5(b[5][i]),
          .in6(b[6][i]),
          .out(mux_out_b[i]),
          .sel(stage)
      );
    end
  endgenerate
```

NTT Circuit Design



- This kind of building block or pattern is called **Cooley–Tukey butterfly**



(a) Cooley-Tukey butterfly

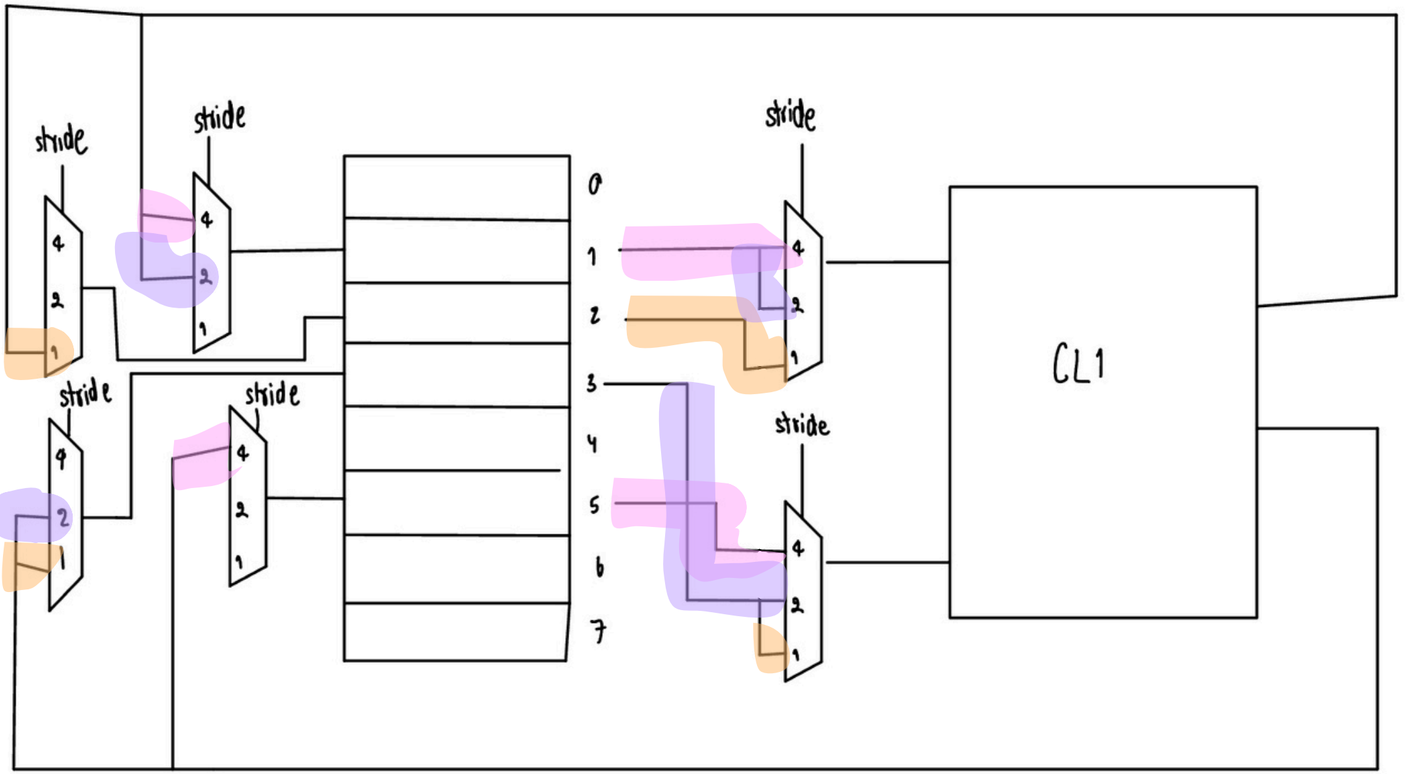
- Recursion is iterative grouping of elements in particular order
- The grouping is called Cooley–Tukey butterfly

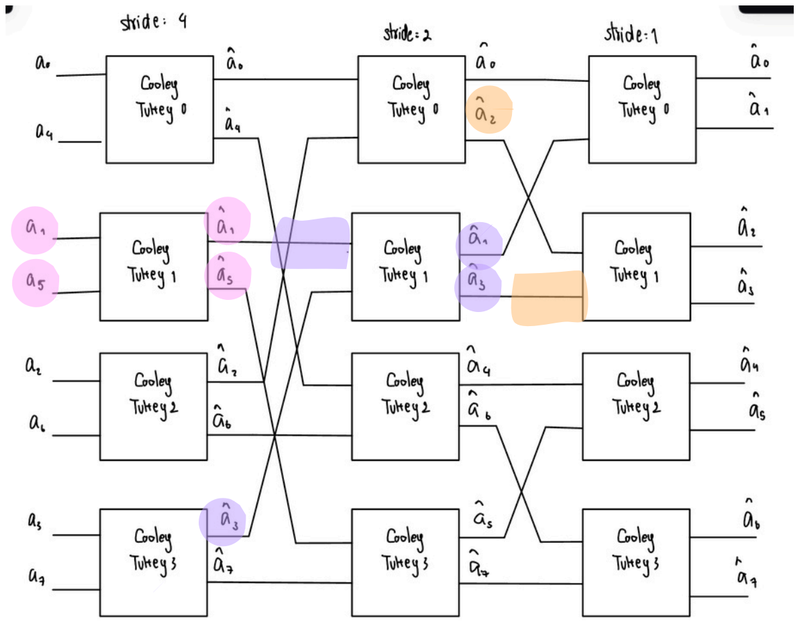
NTT

- **NTT is a particular flavor of Discrete Fourier Transform (DFT) over a finite field.**
- **used for element-wise multiplication. O(nlogn)**
- **(P·Q)(x_i)=P(x)_i * Q(x)_i.**

```
void ntt(int16_t r[256]) {
  unsigned int len, start, j, k;
  int16_t t, zeta;

  k = 1;
  for(len = 128; len >= 2; len >>= 1) {
    for(start = 0; start < 256; start = j + len) {
      zeta = zetas[k++];
      for(j = start; j < start + len; j++) {
        t = fqmul(zeta, r[j + len]);
        r[j + len] = r[j] - t;
        r[j] = r[j] + t;
      }
    }
  }
}
```

NTT Circuit Design





- 2 Mux per Cooley Tukey
- 1 Mux per Buffer
- 128 Cooley Tukey = 248 Mux
- 256 Buffer = 256 Mux
- 504 Mux
- 128 Cooley Tukey Modules

Future Works

- Continue designing each module seperately
- Combine each module into a working system
- Revised Model (with prof. Tanaka)
- Implementing with a real FPGA hardware (IF POSSIBLE)

Thank you