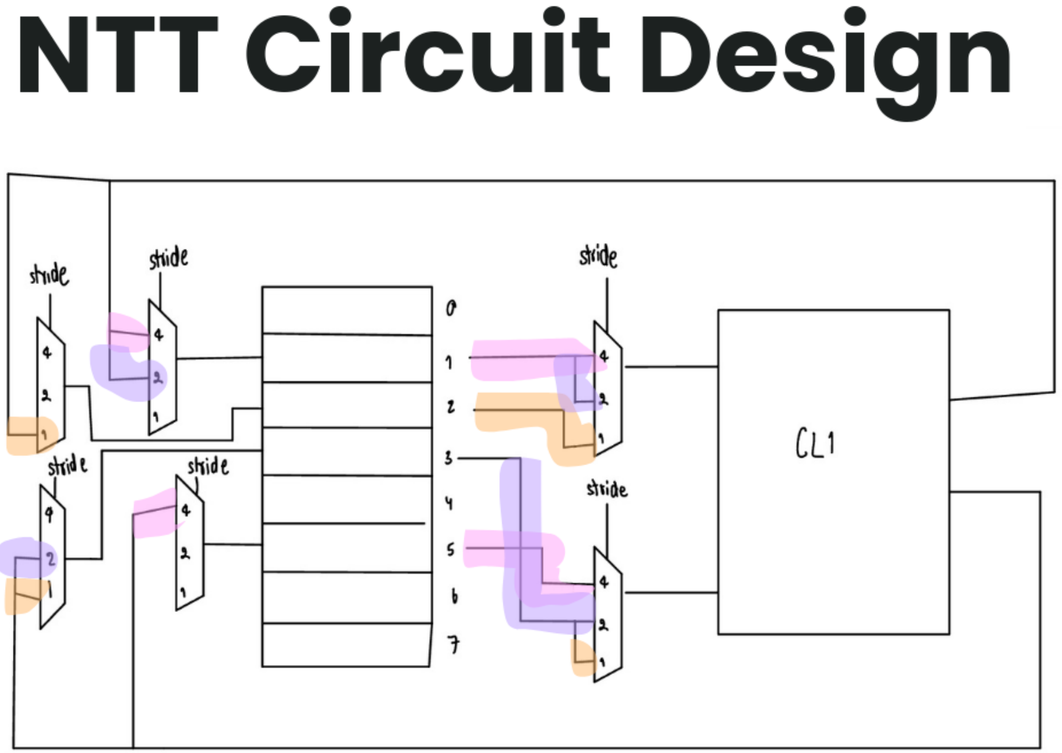
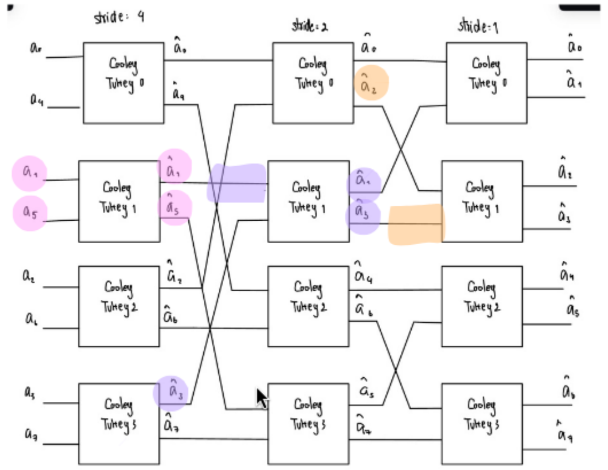
Kyber modules design progress

By Pakin Panawattanakul
19/01/2026

Presentation Outline

Previous NTT module design



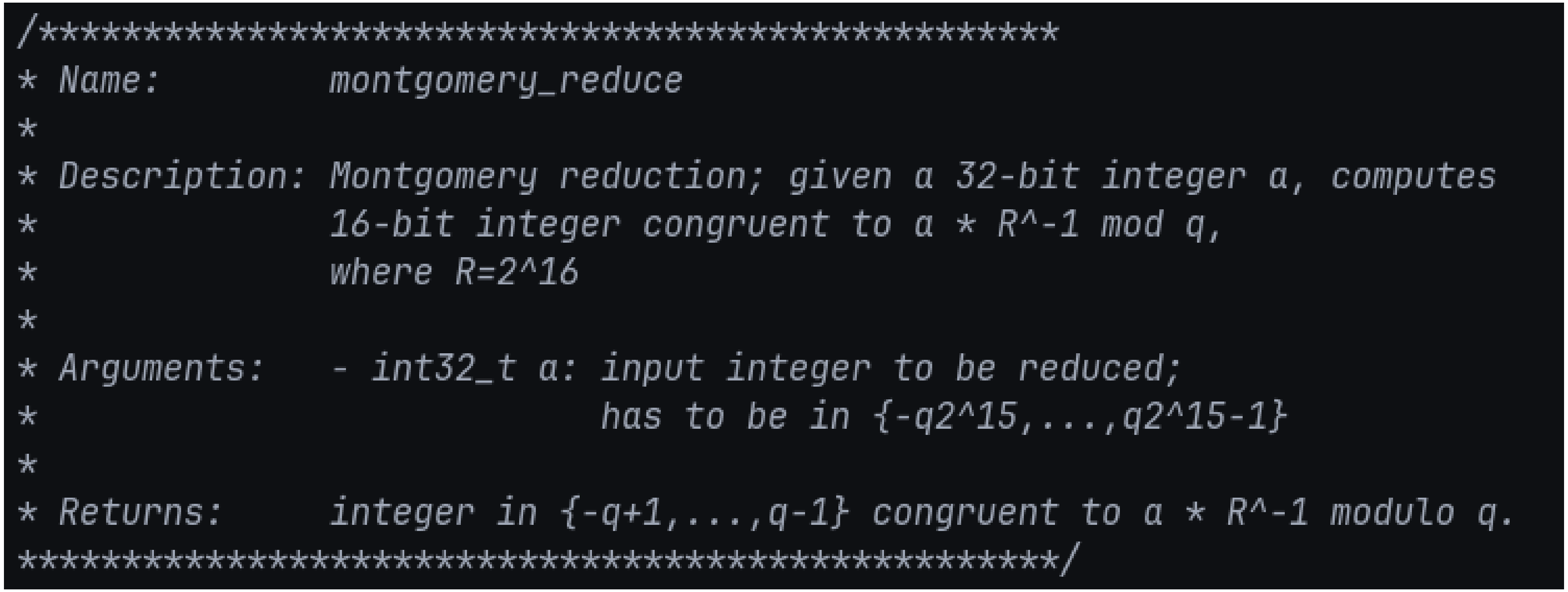


- 2 Muxes per Cooley Tukey
- 1 Mux per address in buffer
- 128 Cooley Tukey = 256 Muxes
- 256 Address = 256 Mux
- 512 Muxes

Modules

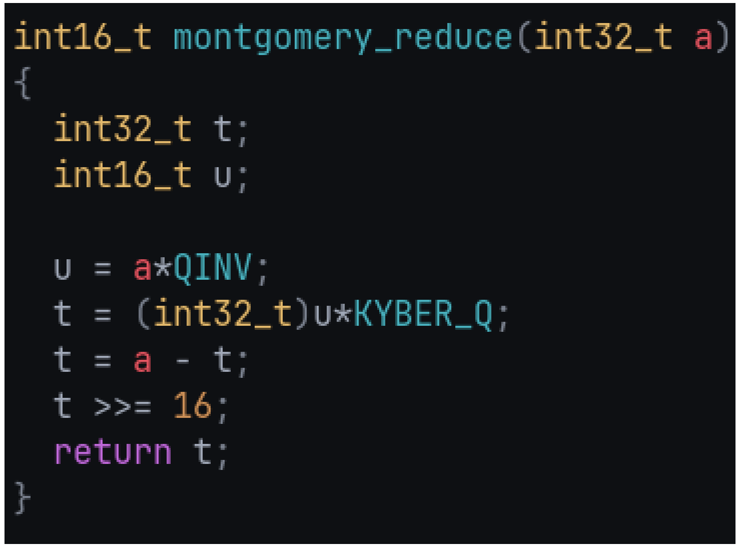
- Montgomery reduce
- fqmul
- Cooley tukey
- NTT

Montgomery Reduce



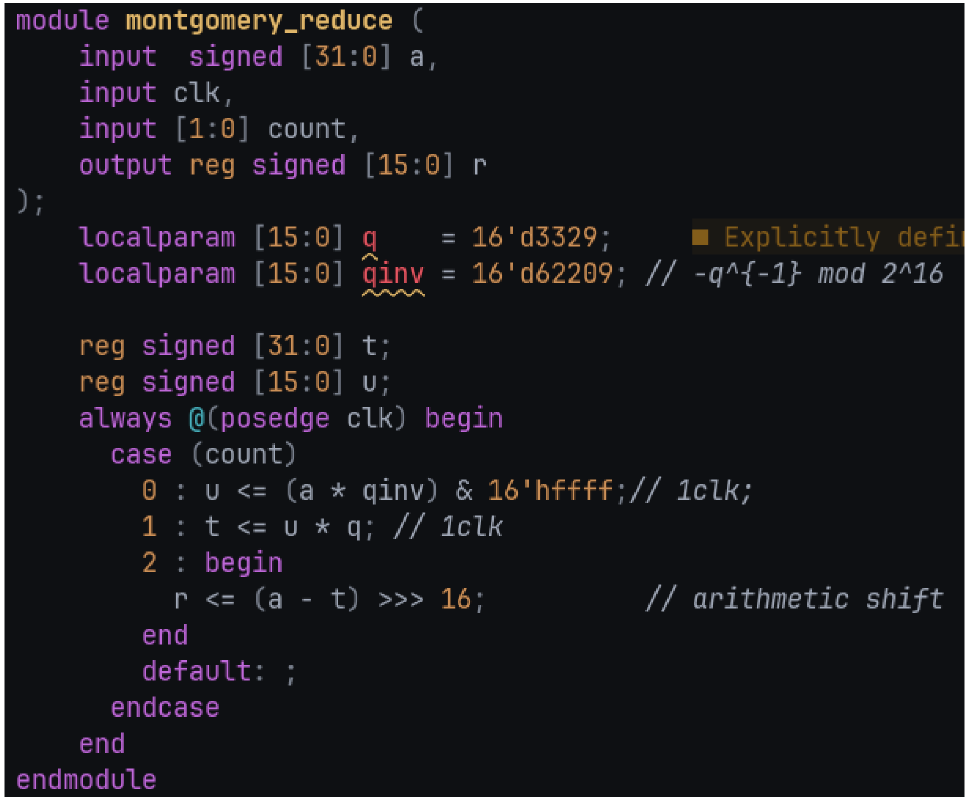
```
/**************************************************
* Name:        montgomery_reduce
*
* Description: Montgomery reduction; given a 32-bit integer a, computes
*              16-bit integer congruent to a * R^-1 mod q,
*              where R=2^16
*
* Arguments:   - int32_t a: input integer to be reduced;
*                           has to be in {-q2^15,...,q2^15-1}
*
* Returns:     integer in {-q+1,...,q-1} congruent to a * R^-1 modulo q.
**************************************************/
```

Montgomery reduce



```c
int16_t montgomery_reduce(int32_t a)
{
  int32_t t;
  int16_t u;

  u = a*QINV;
  t = (int32_t)u*KYBER_Q;
  t = a - t;
  t >>= 16;
  return t;
}
```

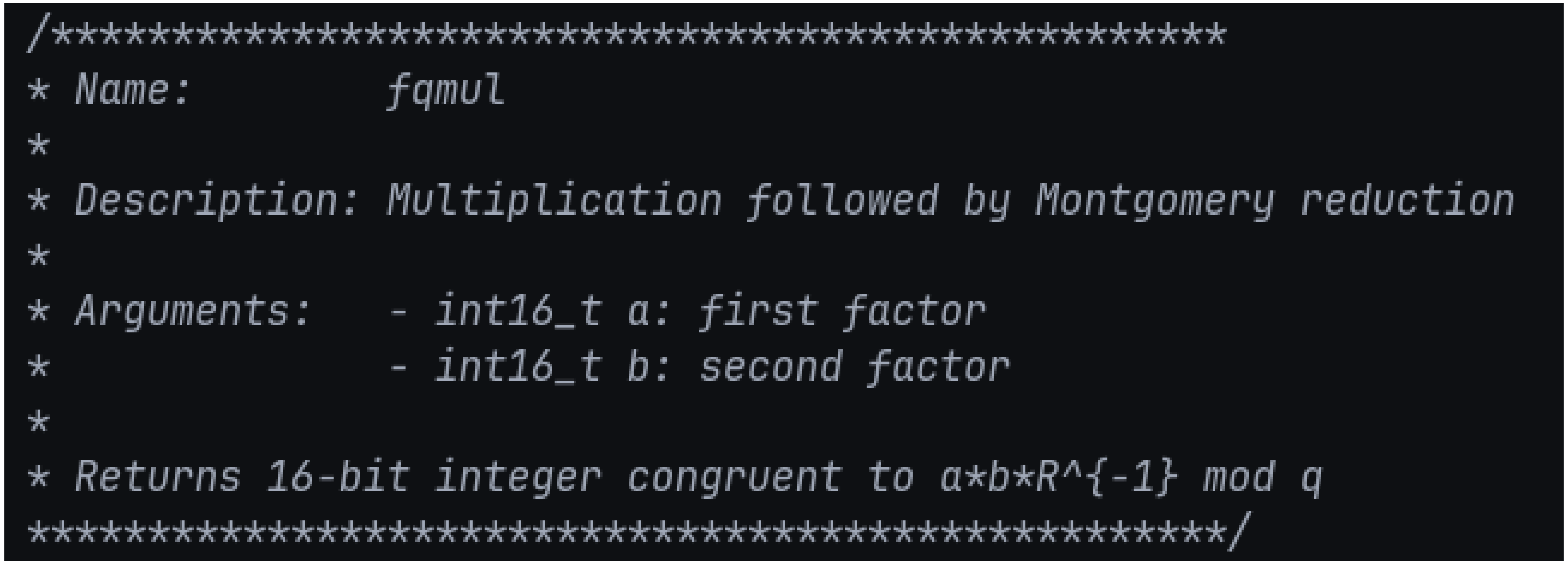
C



```verilog
module montgomery_reduce (
    input  signed [31:0] a,
    input clk,
    input [1:0] count,
    output reg signed [15:0] r
);
    localparam [15:0] q    = 16'd3329;      ■ Explicitly defi
    localparam [15:0] qinv = 16'd62209; // -q^{-1} mod 2^16

    reg signed [31:0] t;
    reg signed [15:0] u;
    always @(posedge clk) begin
      case (count)
        0 : u <= (a * qinv) & 16'hffff;// 1clk;
        1 : t <= u * q; // 1clk
        2 : begin
          r <= (a - t) >>> 16;          // arithmetic shift
        end
        default: ;
      endcase
    end
endmodule
```

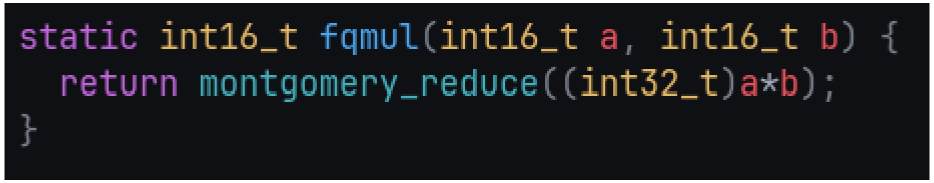
System Verilog

fqmul



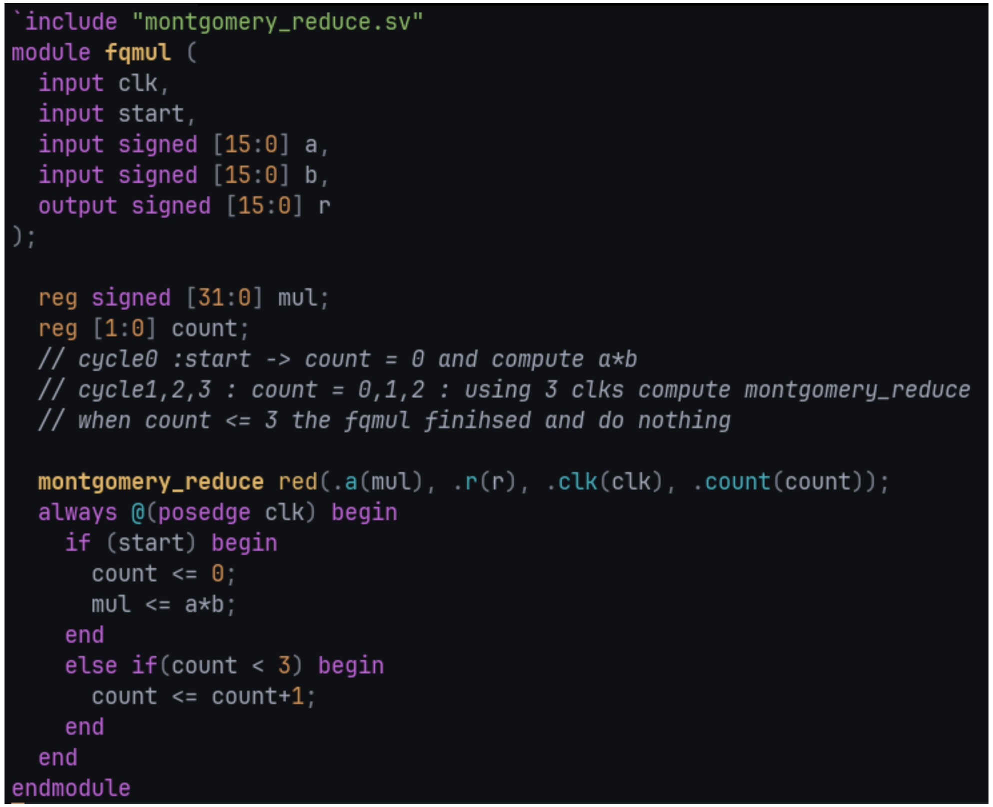
```
/*************************************************
* Name:        fqmul
*
* Description: Multiplication followed by Montgomery reduction
*
* Arguments:   - int16_t a: first factor
*              - int16_t b: second factor
*
* Returns 16-bit integer congruent to a*b*R^{-1} mod q
**************************************************/
```

fqmul



```c
static int16_t fqmul(int16_t a, int16_t b) {
  return montgomery_reduce((int32_t)a*b);
}
```

C

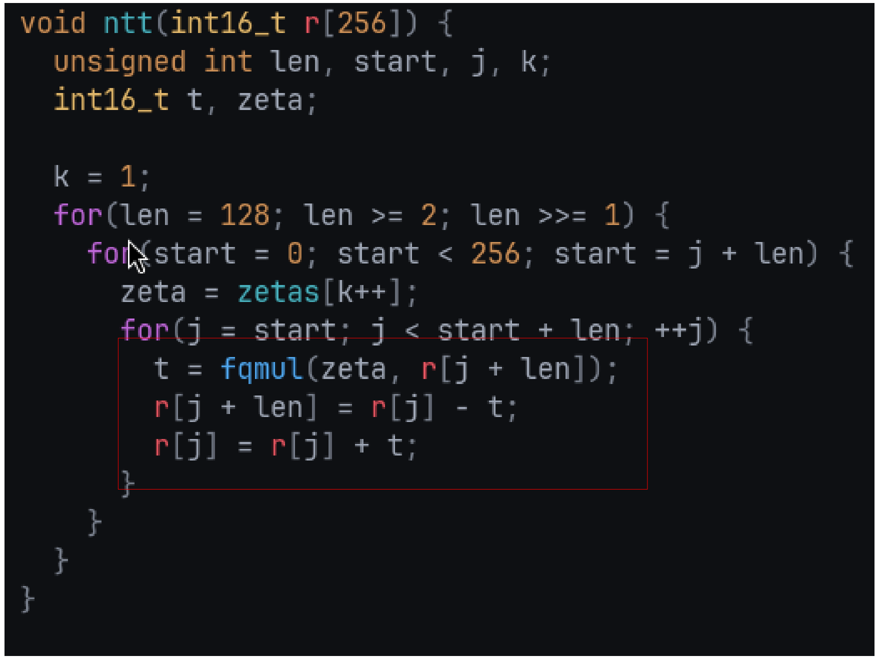


```systemverilog
`include "montgomery_reduce.sv"
module fqmul (
  input clk,
  input start,
  input signed [15:0] a,
  input signed [15:0] b,
  output signed [15:0] r
);

  reg signed [31:0] mul;
  reg [1:0] count;
  // cycle0 :start -> count = 0 and compute a*b
  // cycle1,2,3 : count = 0,1,2 : using 3 clks compute montgomery_reduce
  // when count <= 3 the fqmul finihsed and do nothing

  montgomery_reduce red(.a(mul), .r(r), .clk(clk), .count(count));
  always @(posedge clk) begin
    if (start) begin
      count <= 0;
      mul <= a*b;
    end
    else if(count < 3) begin
      count <= count+1;
    end
  end
endmodule
```

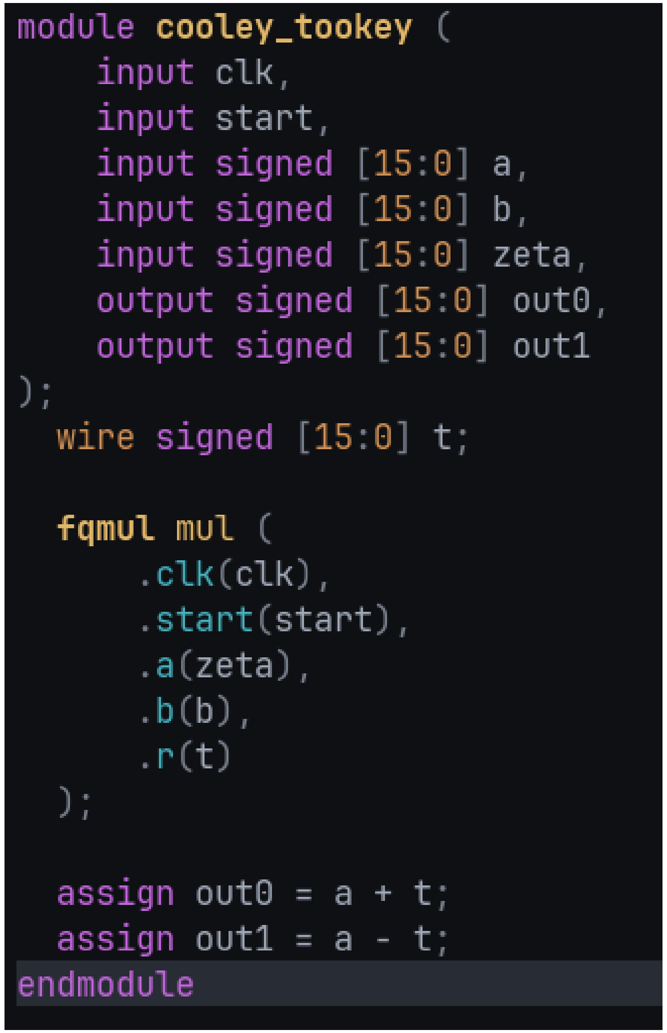
System Verilog

cooley_tookey

- no direct C implementation for cooley tookey function
- Imitate what each NTT loop do



```c
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;
      }
    }
  }
}
```



```systemverilog
module cooley_tookey (
    input clk,
    input start,
    input signed [15:0] a,
    input signed [15:0] b,
    input signed [15:0] zeta,
    output signed [15:0] out0,
    output signed [15:0] out1
);
  wire signed [15:0] t;

  fqmul mul (
      .clk(clk),
      .start(start),
      .a(zeta),
      .b(b),
      .r(t)
  );

  assign out0 = a + t;
  assign out1 = a - t;
endmodule
```

System Verilog

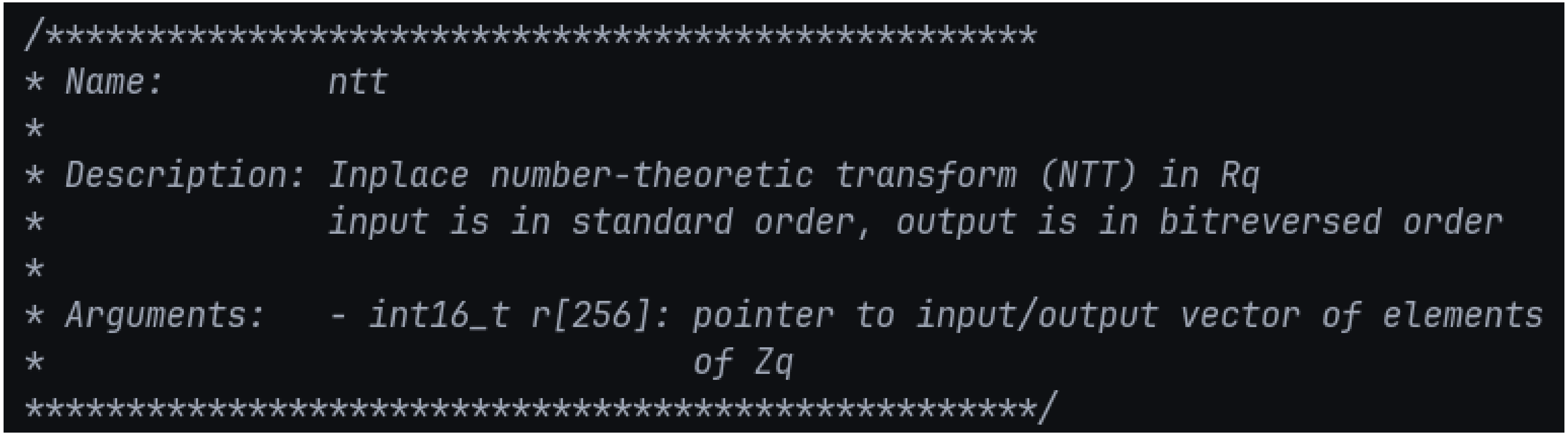
NTT

Top module for converting polynomial ring into NTT form

New Design

- Using 8 cooley tukey units
- 2 buffers
- Finite state machine

NTT description

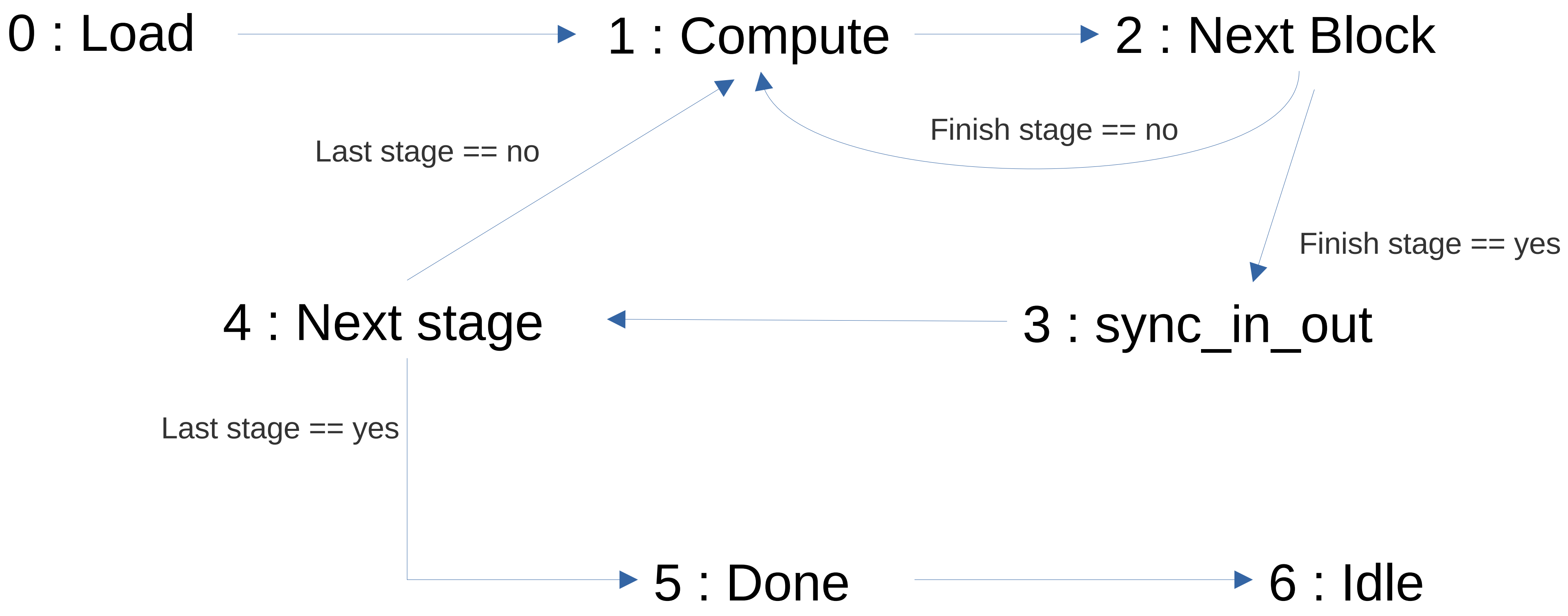


```
/*************************************************
* Name:        ntt
*
* Description: Inplace number-theoretic transform (NTT) in Rq
*              input is in standard order, output is in bitreversed order
*
* Arguments:   - int16_t r[256]: pointer to input/output vector of elements
*                                of Zq
**************************************************/
```

Finite state machine

No.	Name	Description
0	load	Initial state : load input to buffer_in, set initial variable Go to **compute(1)**
1	compute	Compute CooleyTookey until next block is detected Go to **next_blk(2)**
2	next_blk	Set variable for next block If the current stage complete => go to **sync_in_out(3)** else => go back to **compute(1)**
3	sync_in_out	Save data from output buffer to input buffer go to **next_stage(4)**
4	next_stage	Set variable for next stage If (this is the last stage) : go to **done(5)** Else : go back to **compute(1)**
5	done	Save value from output buffer to output, go to **idle(6)**
6	idle	Do nothing waiting for next enable signal

Finite state machine

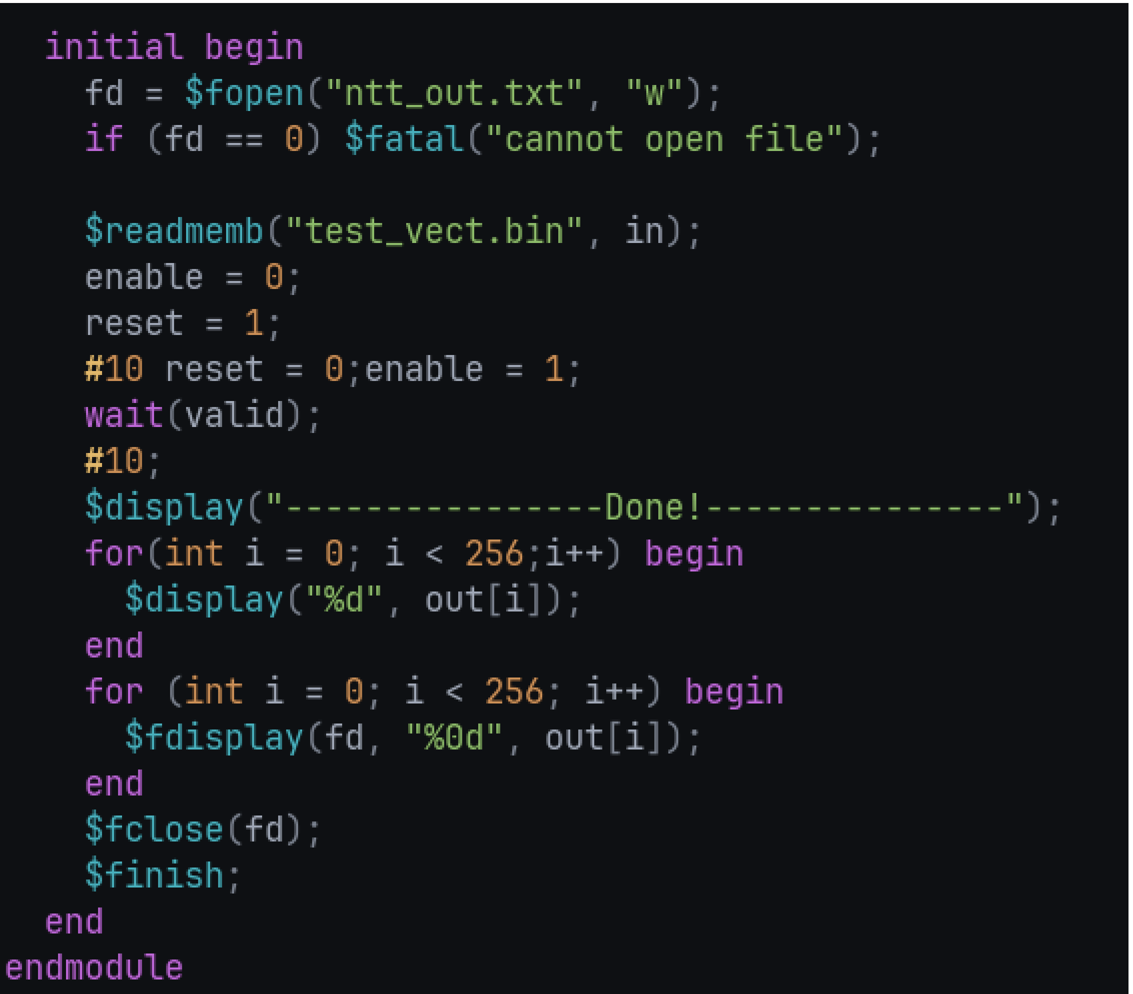


Result & Verifications

- Save RTL output to files
- C test code compare output from RTL to C-reference output

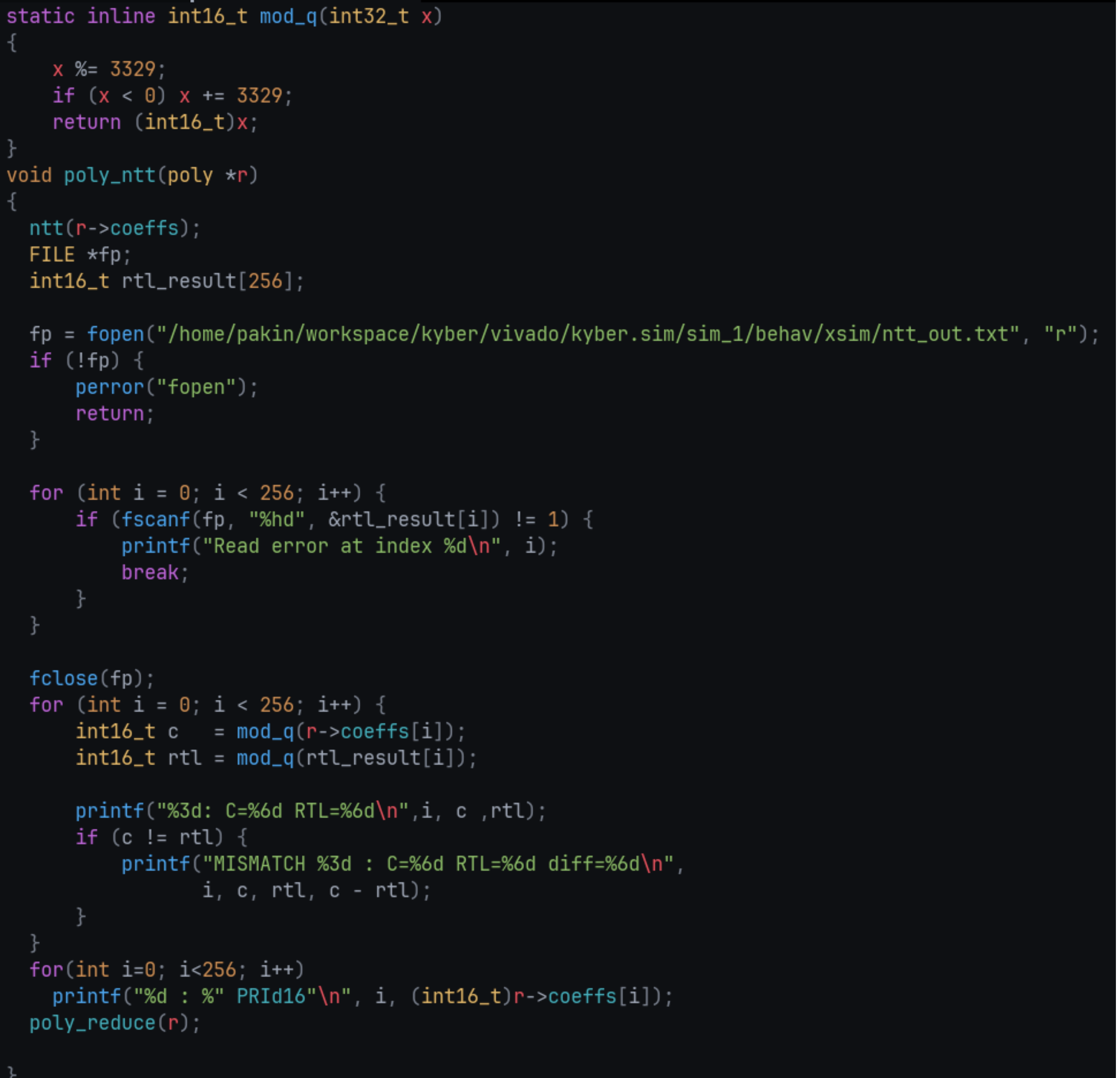
Modules other than NTT are verified by testbench but we will skipped in this presentation

NTT Testbench



```
  initial begin
    fd = $fopen("ntt_out.txt", "w");
    if (fd == 0) $fatal("cannot open file");

    $readmemb("test_vect.bin", in);
    enable = 0;
    reset = 1;
    #10 reset = 0;enable = 1;
    wait(valid);
    #10;
    $display("---------------Done!---------------");
    for(int i = 0; i < 256;i++) begin
      $display("%d", out[i]);
    end
    for (int i = 0; i < 256; i++) begin
      $fdisplay(fd, "%0d", out[i]);
    end
    $fclose(fd);
    $finish;
  end
endmodule
```

- Load input from file
- Enable & reset signal
- Write the result to file



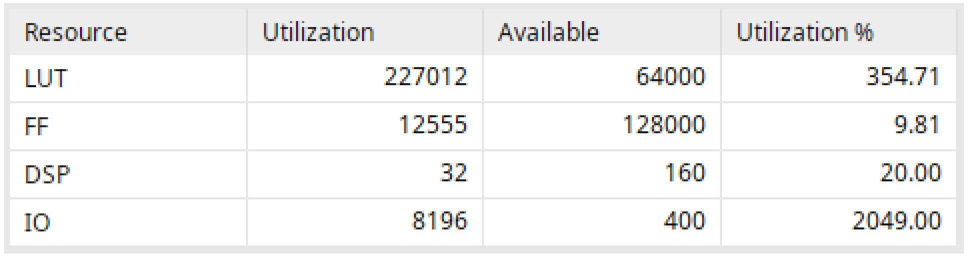
```c
static inline int16_t mod_q(int32_t x)
{
    x %= 3329;
    if (x < 0) x += 3329;
    return (int16_t)x;
}
void poly_ntt(poly *r)
{
  ntt(r->coeffs);
  FILE *fp;
  int16_t rtl_result[256];

  fp = fopen("/home/pakin/workspace/kyber/vivado/kyber.sim/sim_1/behav/xsim/ntt_out.txt", "r");
  if (!fp) {
      perror("fopen");
      return;
  }

  for (int i = 0; i < 256; i++) {
      if (fscanf(fp, "%hd", &rtl_result[i]) != 1) {
          printf("Read error at index %d\n", i);
          break;
      }
  }

  fclose(fp);
  for (int i = 0; i < 256; i++) {
      int16_t c   = mod_q(r->coeffs[i]);
      int16_t rtl = mod_q(rtl_result[i]);

      printf("%3d: C=%6d RTL=%6d\n",i, c ,rtl);
      if (c != rtl) {
          printf("MISMATCH %3d : C=%6d RTL=%6d diff=%6d\n",
                 i, c, rtl, c - rtl);
      }
  }
  for(int i=0; i<256; i++)
    printf("%d : %" PRId16"\n", i, (int16_t)r->coeffs[i]);
  poly_reduce(r);

}
```

C testcode

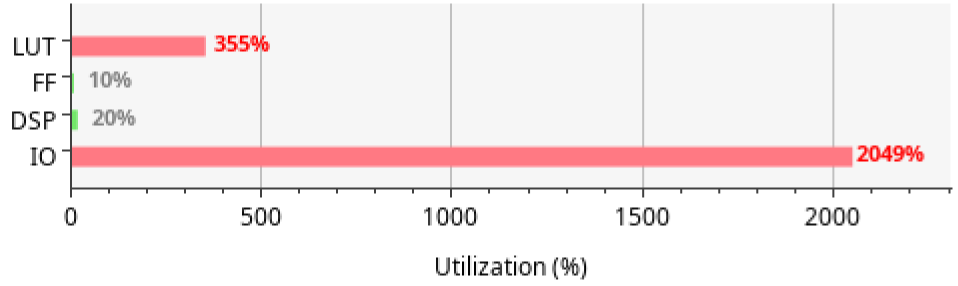
- Read RTL output file
- Mod output with 3329
- Compare with output from C code

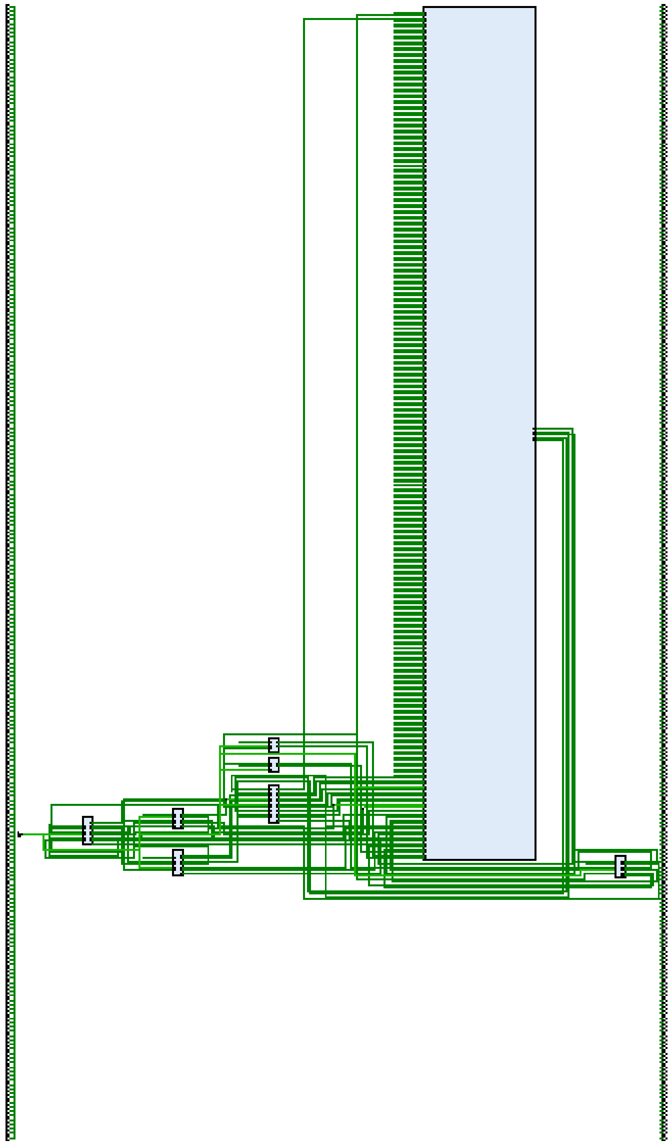
Problems

- Correct behavioral simulation output but ...
- Resource utilzation is extremely high
  - Main Problem : LUT
  - I/O : Not a real problem



Resource	Utilization	Available	Utilization %
LUT	227012	64000	354.71
FF	12555	128000	9.81
DSP	32	160	20.00
IO	8196	400	2049.00





Data flow design

- Abnomally large cooley_tukey block
- I/O are very big : My guess is : not a problem

  since the real implementation will be internal

  BRAM or wire or reg, not I/O port

Future work

- Deisign other module in encapsulations (Main focus)
- Improve NTT module : reduce resources usage
- Goal : Finish all encapsualtion modules before end of

  Jan 2026