Lab06 Design Consideration

Tz-Hsi, Lin

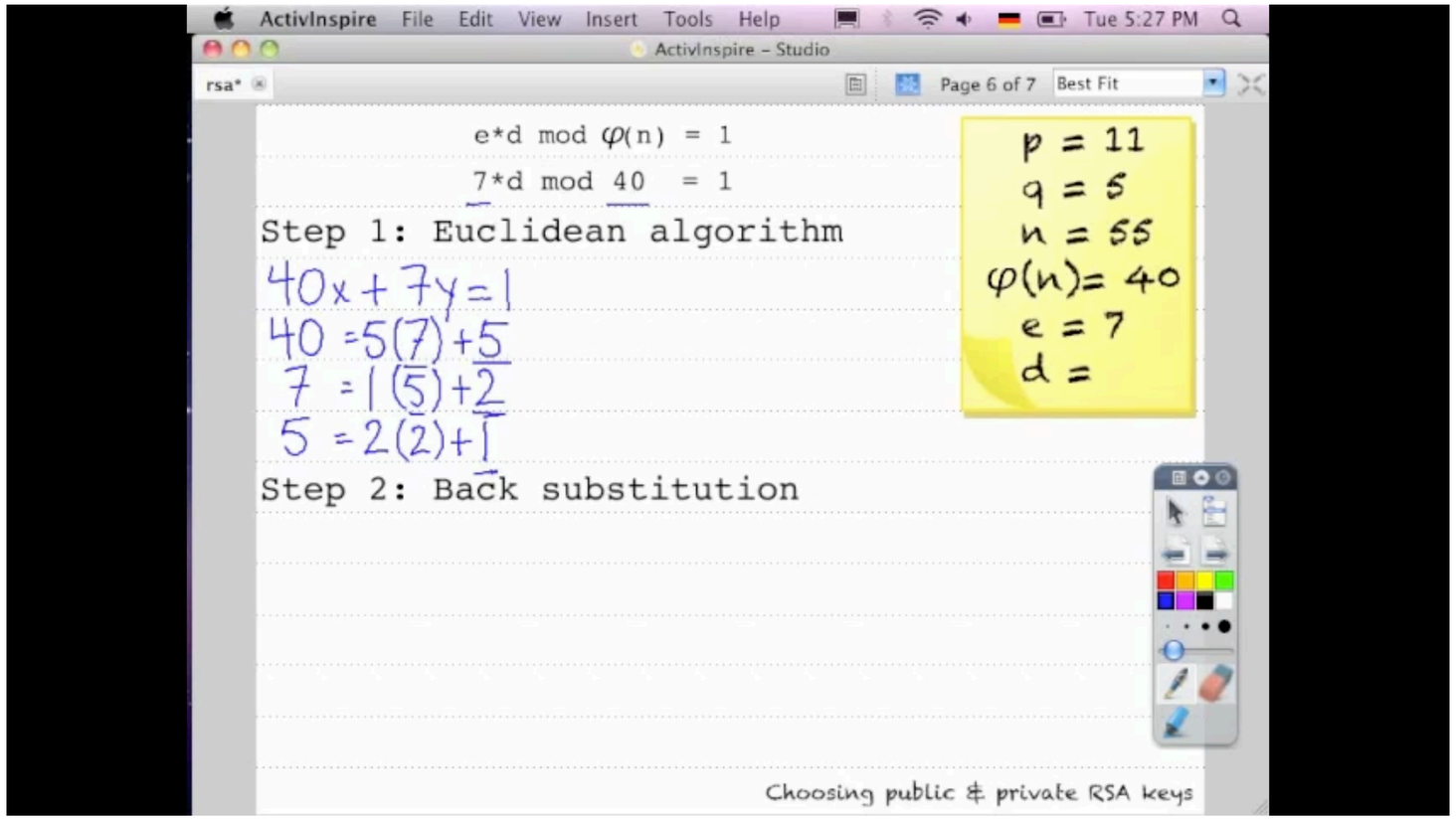
2022/04/27

Agenda

- Soft IP
- Top Design

Soft IP

Algorithm - Extended Euclidean Algorithm



Ref[1]: https://youtu.be/kYasb426Yjk
Ref[2]: https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm

Soft IP

Algorithm - Extended Euclidean Algorithm

A B r_i s_i A t_i B

- $\gcd(240, 46) = 2, -9 \times 240 + 47 \times 46 = 2$

index i	quotient q_{i-1}	Remainder r_i	s_i	t_i
0		240	1	0
1		46	0	1
2	$240 \div 46 = 5$	$240 - 5 \times 46 = 10$	$1 - 5 \times 0 = 1$	$0 - 5 \times 1 = -5$
3	$46 \div 10 = 4$	$46 - 4 \times 10 = 6$	$0 - 4 \times 1 = -4$	$1 - 4 \times -5 = 21$
4	$10 \div 6 = 1$	$10 - 1 \times 6 = 4$	$1 - 1 \times -4 = 5$	$-5 - 1 \times 21 = -26$
5	$6 \div 4 = 1$	$6 - 1 \times 4 = 2$	$-4 - 1 \times 5 = -9$	$21 - 1 \times -26 = 47$
6	$4 \div 2 = 2$	$4 - 2 \times 2 = 0$	$5 - 2 \times -9 = 23$	$-26 - 2 \times 47 = -120$

Ref[1]: https://youtu.be/kYasb426Yjk
Ref[2]: https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm

Soft IP

Algorithm - Extended Euclidean Algorithm

- Hence, there will be two parts, the computation and the decision.

index i	quotient q_{i-1}	Remainder r_i	s_i	t_i
0		240	1	0
1		46	0	1
2	240 ÷ 46 = 5	240 – 5 × 46 = 10	1 – 5 × 0 = 1	0 – 5 × 1 = –5
3	46 ÷ 10 = 4	46 – 4 × 10 = 6	0 – 4 × 1 = –4	1 – 4 × –5 = 21
4	10 ÷ 6 = 1	10 – 1 × 6 = 4	1 – 1 × –4 = 5	–5 – 1 × 21 = –26
5	6 ÷ 4 = 1	6 – 1 × 4 = 2	–4 – 1 × 5 = –9	21 – 1 × –26 = 47
6	4 ÷ 2 = 2	4 – 2 × 2 = 0	5 – 2 × –9 = 23	–26 – 2 × 47 = –120

Ref[1]: https://youtu.be/kYasb426Yjk
Ref[2]: https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm

Soft IP

Algorithm - Extended Euclidean Algorithm

- Computation
  1. calculation of (p-1), (q-1), and phi
  2. Iteratively find the remainder and the quotient for six times
- Decision
  1. When the remainder is 1, the t is the result

Ref[1]: https://youtu.be/kYasb426Yjk
Ref[2]: https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm

Soft IP

Algorithm - Extended Euclidean Algorithm

- The detailed information can be found in the reference.

```
else if (level_idx == 3) begin: if_lv_3
    wire [WIDTH + 1: 0] r, q;
    assign q = gen_level[2].if_lv_2.phi / gen_level[1].if_lv_1.e;
    assign r = gen_level[2].if_lv_2.phi - q * gen_level[1].if_lv_1.e;

    wire signed [WIDTH + 2: 0] t;
    assign t = -q;
end
else if (level_idx == 4) begin: if_lv_4
    wire [WIDTH + 1: 0] r, q;
    assign q = gen_level[1].if_lv_1.e / gen_level[3].if_lv_3.r;
    assign r = gen_level[1].if_lv_1.e - q * gen_level[3].if_lv_3.r;

    wire signed [WIDTH + 2: 0] t;
    assign t = 1 - gen_level[3].if_lv_3.t * q;
end
else if (level_idx == 5) begin: if_lv_5
    wire [WIDTH + 1: 0] r, q;
    assign q = gen_level[3].if_lv_3.r / gen_level[4].if_lv_4.r;
    assign r = gen_level[3].if_lv_3.r - q * gen_level[4].if_lv_4.r;

    wire signed [WIDTH + 2: 0] t;
    assign t = gen_level[3].if_lv_3.t - gen_level[4].if_lv_4.t * q;
end
```

```
generate
    for (level_idx2 = WIDTH; level_idx2 != WIDTH + 1; level_idx2 = level_idx2 + 1) begin
        if (level_idx2 == 3) begin…
        end
        else if (level_idx2 == 4) begin
            always @(*) begin
                if (gen_level[3].if_lv_3.r == 1)
                    tempD = gen_level[3].if_lv_3.t;
                else if (gen_level[4].if_lv_4.r == 1)
                    tempD = gen_level[4].if_lv_4.t;
                else if (gen_level[5].if_lv_5.r == 1)
                    tempD = gen_level[5].if_lv_5.t;
                else if (gen_level[6].if_lv_6.r == 1)
                    tempD = gen_level[6].if_lv_6.t;
                else if (gen_level[7].if_lv_7.r == 1)
                    tempD = gen_level[7].if_lv_7.t;
                else if (gen_level[8].if_lv_8.r == 1)
                    tempD = gen_level[8].if_lv_8.t;
                else
                    tempD = 0;
            end
        end
    end
endgenerate
```

Ref[1]: https://youtu.be/kYasb426Yjk
Ref[2]: https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm

Top Design

Algorithm - Modular Exponentiation

```
function modular_pow(base, exponent, modulus) is
    if modulus = 1 then
        return 0
    Assert :: (modulus - 1) * (modulus - 1) does not overflow base
    result := 1
    base := base mod modulus
    while exponent > 0 do
        if (exponent mod 2 == 1) then
            result := (result * base) mod modulus
        exponent := exponent >> 1
        base := (base * base) mod modulus
    return result
```

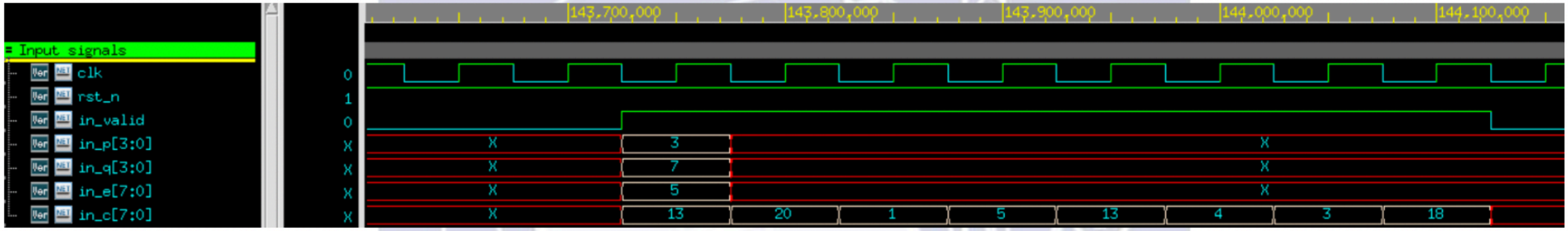
Ref:https://en.wikipedia.org/wiki/Modular_exponentiation

Top Design

Algorithm - Modular Exponentiation

- Find the “e” when it starts to input.
- Start doing the modular exponentiation by pipelining, storing them is unnecessary. -> smaller latency

2. 8 cycles for input the information of each round(pattern)

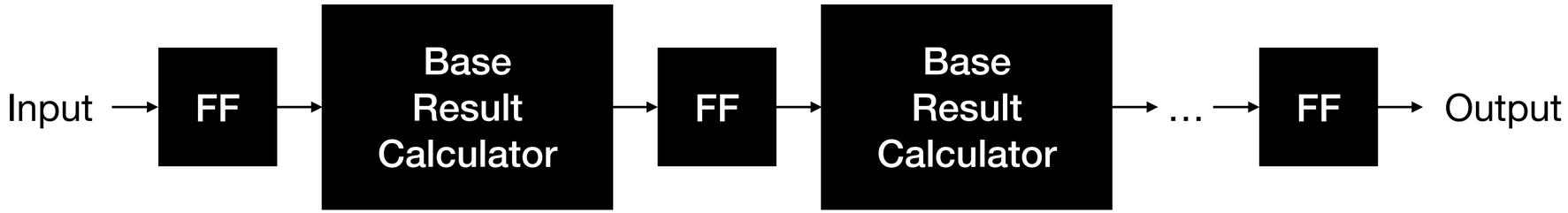


Ref:https://en.wikipedia.org/wiki/Modular_exponentiation

Top Design

Algorithm - Modular Exponentiation

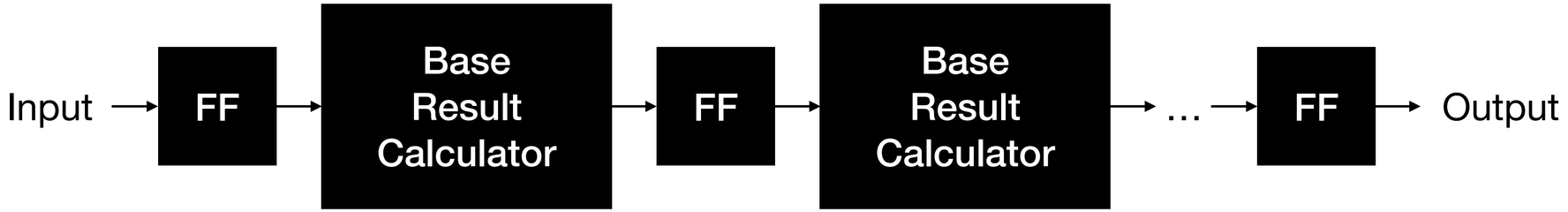
- Find the “e” when it starts to input.
- Start doing the modular exponentiation by pipelining, storing them is unnecessary. -> smaller latency
- The stages needed are the same as the amount of the input.



Ref:https://en.wikipedia.org/wiki/Modular_exponentiation

Top Design

Algorithm - Modular Exponentiation




```
//================================================================
    assign base[0]      = (parameter_d)
                        ? (in_c_ff * in_c_ff) % parameter_n
                        : (in_c_ff) % parameter_n;
    assign result[0]    = (parameter_d & parameter_d[0])
                        ? (in_c_ff) % parameter_n
                        : 1;
//================================================================
    assign base[1]      = (parameter_d[6: 1])
                        ? (base_ff[0] * base_ff[0]) % parameter_n
                        : base_ff[0];
    assign result[1]    = (parameter_d[6: 1] & parameter_d[1])
                        ? (result_ff[0] * base_ff[0]) % parameter_n
                        : result_ff[0];
//================================================================
    assign base[2]      = (parameter_d[6: 2])
                        ? (base_ff[1] * base_ff[1]) % parameter_n
                        : base_ff[1];
    assign result[2]    = (parameter_d[6: 2] & parameter_d[2])
                        ? (result_ff[1] * base_ff[1]) % parameter_n
                        : result_ff[1];
```

```
    always @(posedge clk or negedge rst_n) begin
        if (~rst_n)
            for (i = 0; i != 7; i = i + 1)
                result_ff[i] <= 8'b0;
        else
            for (i = 0; i != 7; i = i + 1)
                result_ff[i] <= result[i];
    end
    always @(posedge clk or negedge rst_n) begin
        if (~rst_n)
            for (i = 0; i != 7; i = i + 1)
                base_ff[i] <= 8'b0;
        else
            for (i = 0; i != 7; i = i + 1)
                base_ff[i] <= base[i];
    end
```

```
    always @(posedge clk or negedge rst_n) begin
        if (~rst_n) begin
            out_valid <= 1'b0;  out_m <= 8'b0;
        end
        else if (NS == OUTPUT) begin
            out_valid <= 1'b1;  out_m <= result_ff[6];
        end
        else begin
            out_valid <= 1'b0;  out_m <= 8'b0;
        end
    end
```

Ref:https://en.wikipedia.org/wiki/Modular_exponentiation

Tz-Hsi Lin presented

hsicc.ee10@nycu.edu.tw

2022/04/18