FPGA Final Project:
Elliptic Curve Cryptography

Team memebers:

F64096114 郭家佑

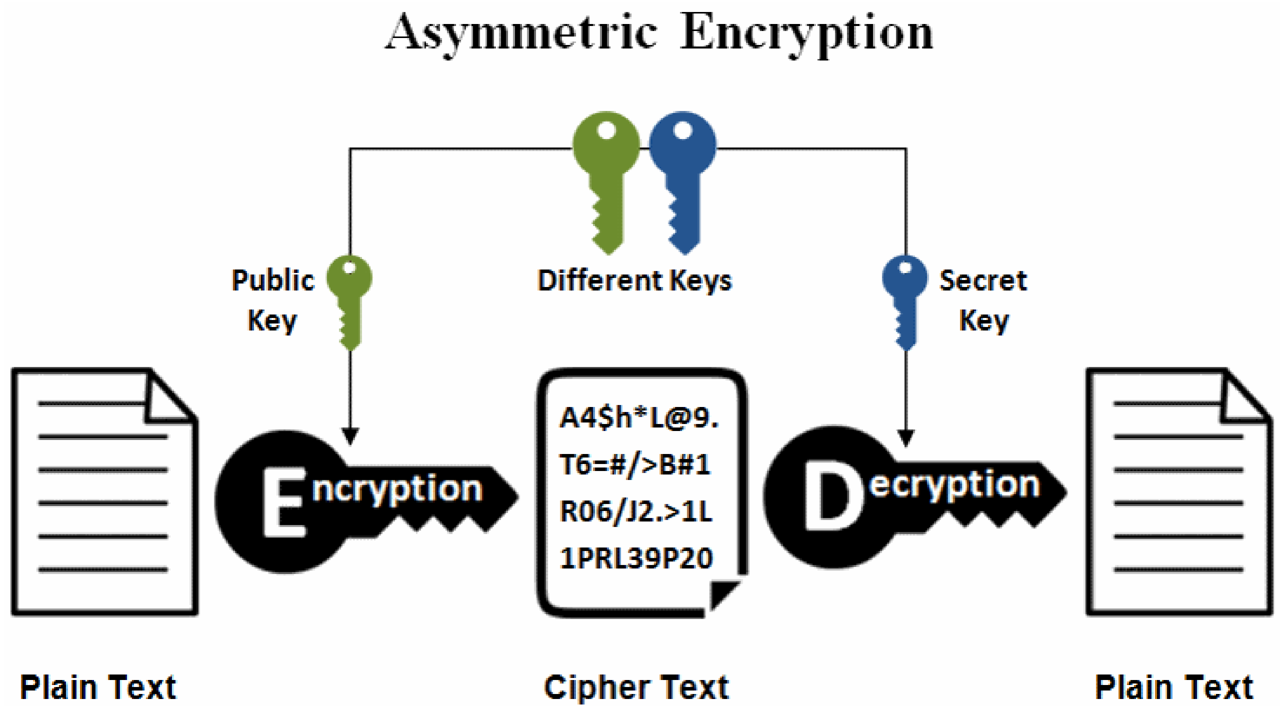
E24096213 蔡奇佑

E14071025 趙泓瑞

Outline

- Introduction to ECC
- Algorithm & Hardware Architecture
- Design features
- Results
- Problems to occur

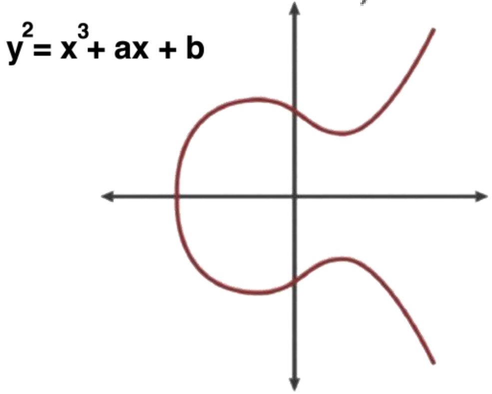
Introduction to ECC



Introduction to ECC

- 橢圓曲線密碼學（英語：Elliptic Curve Cryptography，縮寫：ECC）是一種基於橢圓曲線數學的公開密鑰**非對稱加密演算法**。
- 其特色為**安全性能更高**，160位ECC 和 1024位RSA、DSA有相同的安全強度。
- **處理速度更快**，在計算速度上，ECC比RSA、DSA快得多。
- **儲存空間更小**
- 橢圓曲線是由以下形式的方程式定義的平面曲線(Weierstrass方程式)：

$y^2 = x^3$**+ax+b 其中a和b是實數。**



Introduction to ECC - point addition

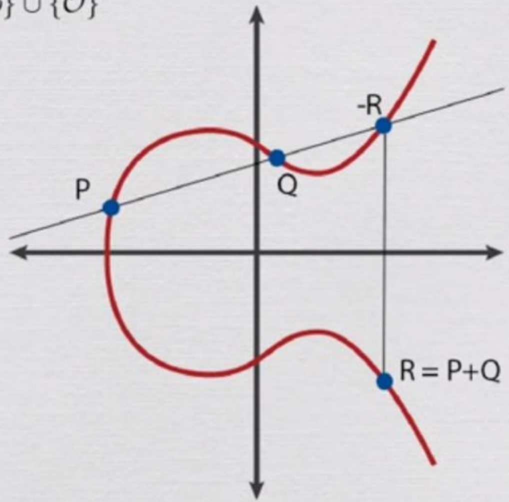
Group Operations

+ ADDITION

point at infinity

Given two points in the set $E = \{(x, y) \mid y^2 = x^3 + ax + b\} \cup \{\mathcal{O}\}$

P+Q=?



Algebraically

$$s = \frac{y_P - y_Q}{x_P - x_Q}$$

$$x_R = s^2 - (x_P + x_Q)$$

$$y_R = s(x_P - x_R) - y_P$$

Introduction to ECC - point doubling

Point Doubling

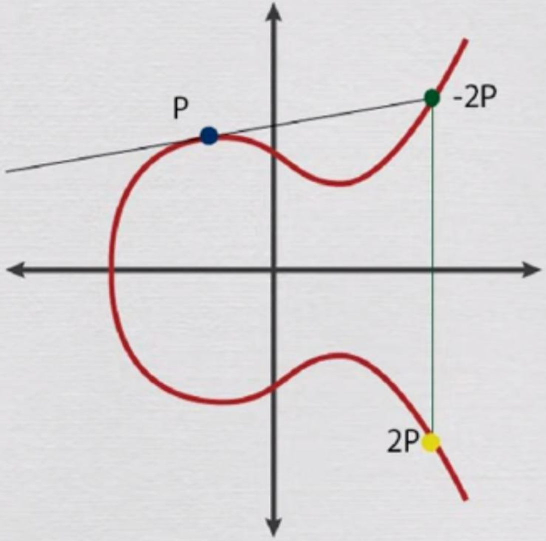
$$P + P = R = 2P$$

Algebraically

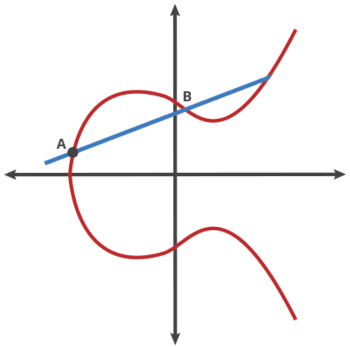
$$s = \frac{3x_P^2 + a}{2y_P}$$

$$x_R = s^2 - 2x_P$$

$$y_R = s(x_P - x_R) - y_P$$



Introduction to ECC



$$G = (5,1)$$
$$2G = (6,3)$$
$$3G = (10,6)$$
$$4G = (3,1)$$
$$5G = (9,16)$$
$$6G = (16,13)$$
$$7G = (0,6)$$
$$8G = (13,7)$$
$$9G = (7,6)$$
$$10G = (7,11)$$
$$11G = (13,10)$$
$$12G = (0,11)$$
$$13G = (16,4)$$
$$14G = (9,1)$$
$$15G = (3,16)$$
$$16G = (10,11)$$
$$17G = (6,14)$$
$$18G = (5,16)$$
$$19G = \mathcal{O}$$

Bob

Bob picks

$\beta = 9$

Computes

$B = 9G = (7,6)$

Receives

$A = (10,6)$

Computes

$\beta A = 9A = 9(3G) = 27G = 8G = (13,7)$

Eve

$y^2 \equiv x^3 + 2x + 2 \pmod{17}$

$G = (5,1)$

$n = 19$

$A = (10,6)$

$B = (7,6)$

?

Alice

Alice picks

$\alpha = 3$

Computes

$A = 3G = (10,6)$

Receives

$B = (7,6)$

Computes

$\alpha B = 3B = 3(9G) = 27G = 8G = (13,7)$

Outline

- Introduction to ECC
- Algorithm & Hardware Architecture
- Design features
- Results
- Problems to occur

Algorithm – modulus (version1: use pre-division and shift)

- It replace division operation with multiply and shift
- However it would error when input is close to its bandwidth limit (128bits) since error rate.

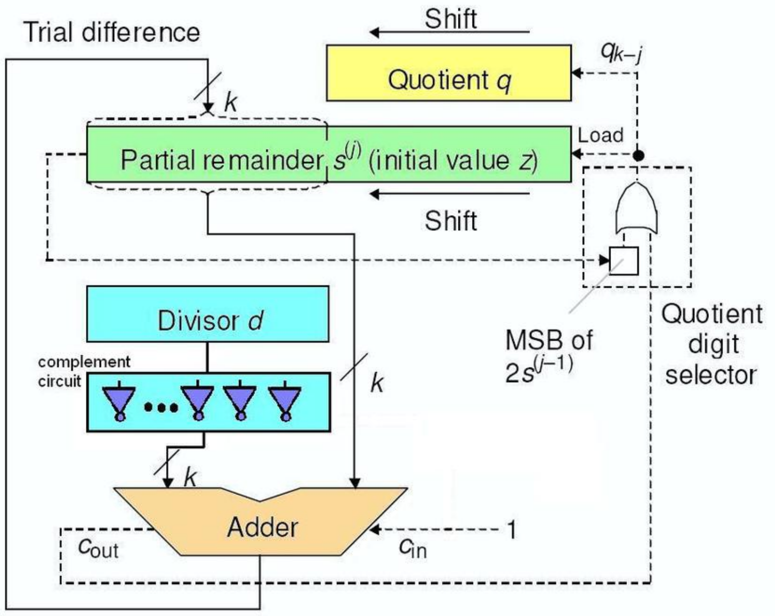
a % q = $a - \left\lfloor \frac{a}{q} \right\rfloor \cdot q = a - \left\lfloor \frac{a \times \frac{2^k}{q}}{2^k} \right\rfloor \cdot q$

$$e = \frac{1}{q} - \frac{q}{2^k} \Rightarrow ae < 1$$

Algorithm – modulus (sol2 :直式除法)

- We use sequential divider instead
- It cost less clk cycle (66/2) than previous version (64*2)

```
modulus algorithm ( calculate x = a % p )
input a (128bits)
output x (64bits)
p = 64'b1001100010011101010011100100001111110101011111111100111101000101;
//decimal 10997031918897188677

mod (a) {
    x = a[ 127  :  64 ]
    for ( i = 0 to 63 ) {
        if ( x > p )
            x = x - p
        if ( i != 0 )
            x =  (x << 1) + a[ 63 - i ]
    }
    return x
}
```



Algorithm – inv_mod (費馬小定理 & 模冪)

- 費馬小定理

費馬小定理是數論中的一個定理：假如 a 是一個整數，p 是一個素數，那麼

$$a^p \equiv a \pmod{p}$$

如果 a 不是 p 的倍數，這個定理也可以寫成

$$a^{p-1} \equiv 1 \pmod{p}$$

這個書寫方式更加常用。（符號的應用請參見模運算。）

- 模冪定理

模冪（英語：modular exponentiation）是一種對模進行的冪運算，在計算機科學，尤其是公開密鑰加密方面有一定用途。

模冪運算是指求整數b的e次方b^e被正整數m所除得到的餘數c的過程，可用數學符號表示為$c = b^e \bmod m$。由c的定義可得$0 \le c < m$。

例如，給定$b = 5$，$e = 3$和$m = 13$，$5^3 = 125$被13除得的餘數$c = 8$。

Algorithm – inv_mod (費馬小定理 & 模冪)

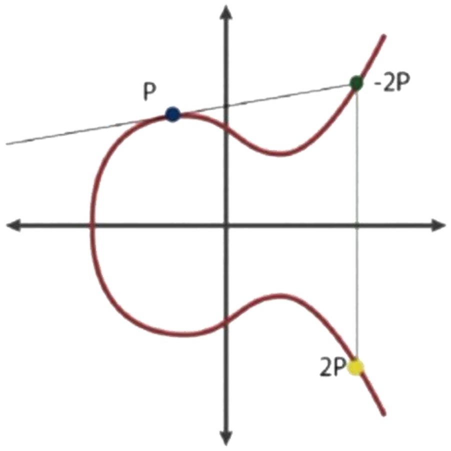
```
inv_mod (calculate x = (1 / a) % p )

input a (64bit)
output x (64bit)

inv_mod(a) {

    // use Fermat's little theorem -> x = ( a^(p-2) ) % p
    exp = p-2
    x = 1
    t_exp = a

    // calculate exponent
    for (i = 0 to 63) begin
        if (exp[i] == 1) x = multiply(x, t_exp)
        t_exp = multiply(t_exp, t_exp)
    end

    return x
}
```

Algorithm – mul64x64

```
mul64x64 (calculate p = a x b)

input a (64bit)
input b (64bit)
output p (128bit)

sc_mul(a, b) {

    tmp[7:0][0:15] is a 8*16 array initialize to 0

    for (i = 0 to 3) begin
        base_addr_tmp = i*4;
        base_addr_b = i*16;
        mul_b = b[ base_addr_b+16 : base_addr_b ]
        tmp[base_addr_tmp  ] += mul_b * a[15:0 ]
        tmp[base_addr_tmp+1] += mul_b * a[31:16]
        tmp[base_addr_tmp+2] += mul_b * a[47:32]
        tmp[base_addr_tmp+3] += mul_b * a[63:48]
    end

    x = tmp[7:0]

    return x
}
```

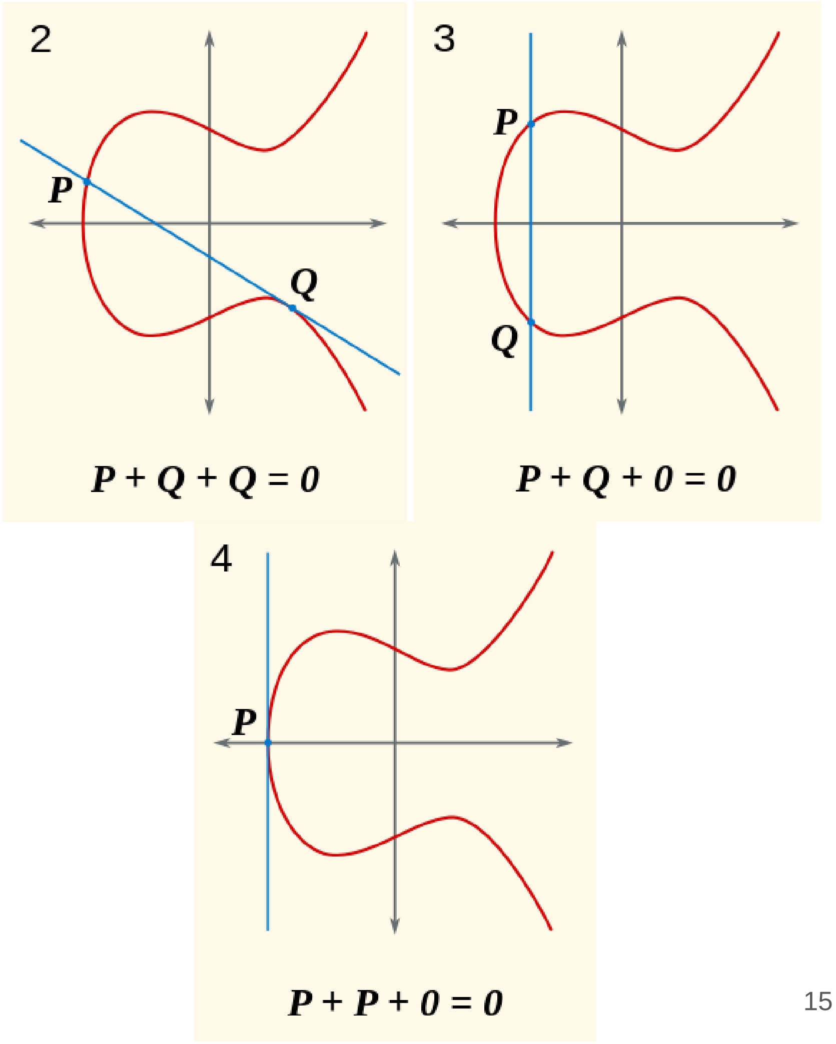
Algorithm – double

```
double  ( calculate T = 2R )

input R (129bit)
output T (129bit)
a = 128'd36284492833867293 67

double ( point R ){
    if(R[128] == 1){
        T = {1'b1 , 128'b0}
        return T
    }
    else {
        Ry = R [127 : 64];
        Rx = R [63 : 0];
        S1 = mod( mul64x64( mul64x64(3,Rx) , Rx ) + a ) //S1 = (3*Rx*Rx+a)%p
        S2 = inv_mod( mul64x64( 2 , Ry ))               //S2 =  inv(2Ry)
        S = mod( mul64x64( S1 , S2 ) )                  //S = (S1*S2)%p
        Tx = mod( mul64x64( S , S ) - Rx - Rx )         //Tx = (S*S -2*Rx)%p
        Ty = mod( mul64x64( S , Rx-Tx ) - Ry )          //Ty = (S*(Rx-Tx)-Ry)%p
        T = {1'b0 , Ty , Tx };
        return T
    }
}
```



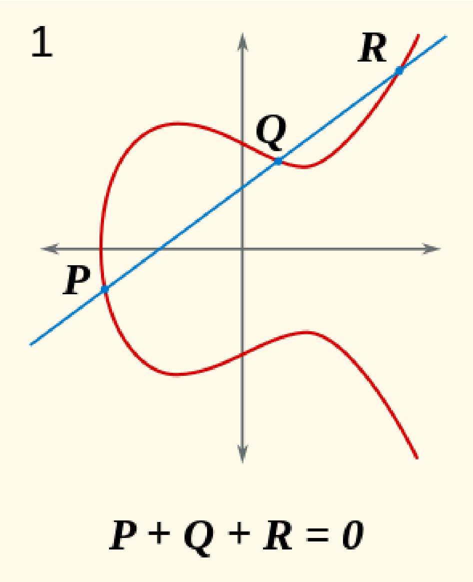
Algorithm – add&sub

```
add  ( calculate T = P+Q )
sub  ( calculate T = P-Q )

input P (129bit)
input Q (129bit)
output T (129bit)
op == 0 -->add
op == 1 -->sub

add&sub ( point P , point Q ){
    if (P[128] == 1)&&(Q[128] == 1) {
        T = {1'b1 , 128'b0}
        return T
    }
    else if(P[128] == 0)&&(Q[128] == 1)&&(op == 0){
        T = {1'b0 , P [127 : 64] , P [127 : 64]}
        return T
    }
    else if(P[128] == 0)&&(Q[128] == 1)&&(op == 1){
        T = {1'b0 , -P [127 : 64] , P [127 : 64]}
        return T
    }
    else if(P[128] == 1)&&(Q[128] == 0)&&(op == 0){
        T = {1'b0 , Q [127 : 64] , Q [127 : 64]}
        return T
    }
    else if(P[128] == 1)&&(Q[128] == 0)&&(op == 1){
        T = {1'b0 , -Q [127 : 64] , Q [127 : 64]}
        return T
    }
```



Algorithm – add&sub

```
    else {
        Px = P [63 : 0]
        Py = P [127 : 64]
        Qx = Q [63 : 0]
        if (op == 1)
            Qy = - Q [127 : 64]
        else
            Qy = + Q [127 : 64]
        S1 = mod( Py - Qy )                              //S1 = (Py - Qy)%p
        S2 = inv_mod( Px - Qx )                          //S2 =  inv(Px - Qx)
        S = mod( mul64x64( S1 , S2 ) )                   //S = (S1*S2)%p
        Tx = mod( mul64x64( S , S ) - Px - Qx )          //Tx = (S*S -Px - Qx)%p
        Ty = mod( mul64x64( S , Rx-Tx ) - Ry )           //Ty = (S*(Rx-Tx)-Ry)%p
        T = { 1'b0 , Ty , Tx }
        return T
    }
}
```



Algorithm double vs. add&sub

```
    else {
        Ry = R [127 : 64];
        Rx = R [63 : 0];
        S1 = mod( mul64x64( mul64x64(3,Rx) , Rx ) + a ) //S1 = (3*Rx*Rx+a)%p
        S2 = inv_mod( mul64x64( 2 , Ry ))               //S2 =  inv(2Ry)
        S = mod( mul64x64( S1 , S2 ) )                  //S = (S1*S2)%p
        Tx = mod( mul64x64( S , S ) - Rx - Rx )         //Tx = (S*S -2*Rx)%p
        Ty = mod( mul64x64( S , Rx-Tx ) - Ry )          //Ty = (S*(Rx-Tx)-Ry)%p
        T = {1'b0 , Ty , Tx };
        return T
    }
}

    else {
        Px = P [63 : 0]
        Py = P [127 : 64]
        Qx = Q [63 : 0]
        if (op == 1)
            Qy = - Q [127 : 64]
        else
            Qy = + Q [127 : 64]
        S1 = mod( Py - Qy )                         //S1 = (Py - Qy)%p
        S2 = inv_mod( Px - Qx )                     //S2 =  inv(Px - Qx)
        S = mod( mul64x64( S1 , S2 ) )              //S = (S1*S2)%p
        Tx = mod( mul64x64( S , S ) - Px - Qx )     //Tx = (S*S -Px - Qx)%p
        Ty = mod( mul64x64( S , Rx-Tx ) - Ry )      //Ty = (S*(Rx-Tx)-Ry)%p
        T = { 1'b0 , Ty , Tx }
        return T
    }
}
```

Point addition

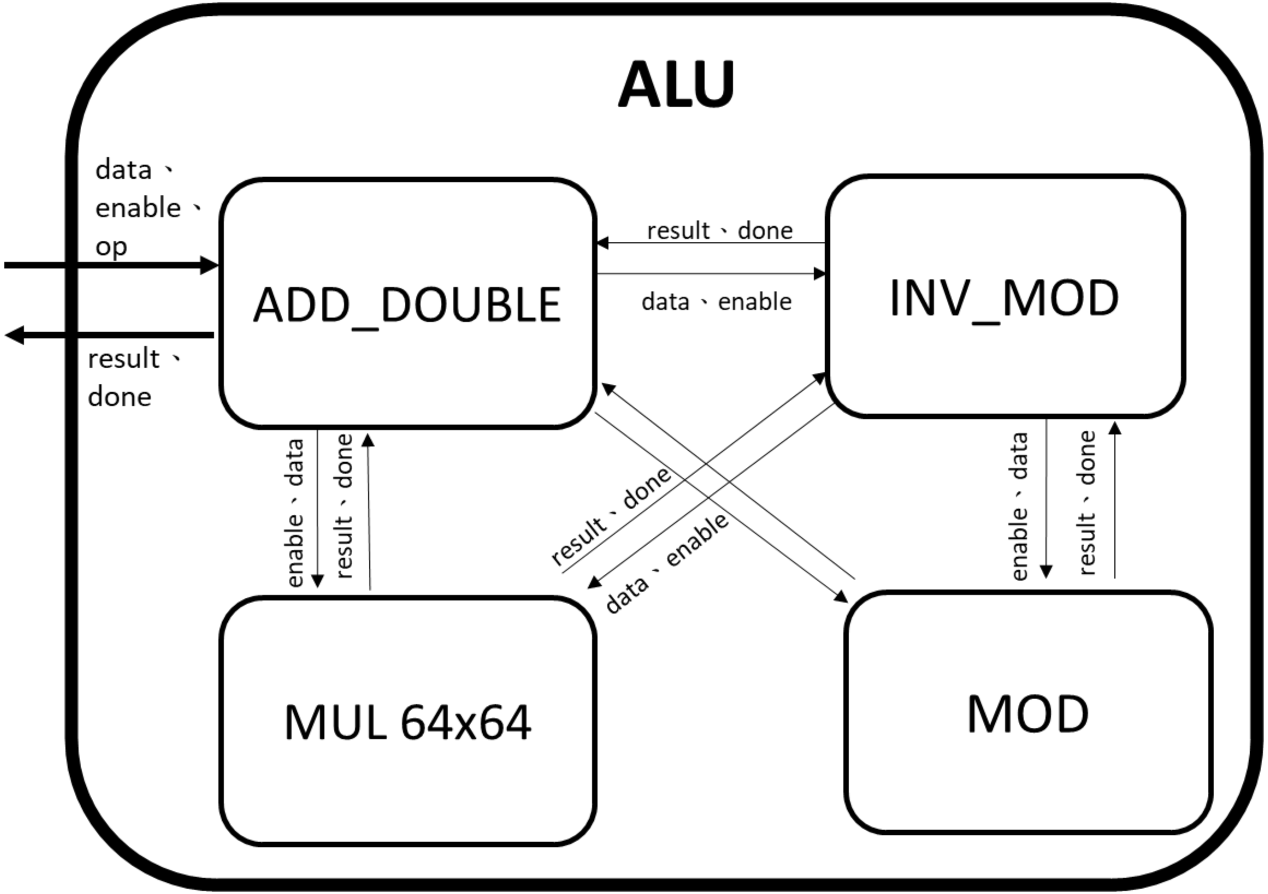
Point subtraction

Point double

Resource sharing for 3 functions in one architecture.

→ Control unit of ALU

ALU Hardware Architecture



Introduction to ECC – scalar multiplication

$11_{(10)} = 1011_{(2)}$ (initial R = infinity point)

$R = 11_{(10)}\, P = 1011_{(2)}\, P$

$R = 2^3\, P + 2^1\, P \; + 2^0\, P$

$R = \; 2\, (2^2\, P + 2^0\, P) + P$

$R = \; 2\, (\, 2\, (2\, P) + 2\, P) + P$

$1x2^3$	$R = 2\, R + 1\, P = P$
$0x2^2$	$R = 2\, R + 0\, P = 2^1\, P$
$1x2^1$	$R = 2\, R + 1\, P = 2^2\, P + 2^0\, P$
$1x2^0$	$R = 2\, R + 1\, P = 2^3\, P + 2^1\, P + 2^0\, P$

Algorithm – scalar multiplicaiton

```
scalar multiplicaiton (calculate R = a x P)

input a (64bit) // number
ipuut P (1+64+64bit) // point
output R (1+64+64bit) // point

sc_mul(a, P) {

    R = infinity point
    for (i = 63 to 0) begin
        R = double(R)
        if(a[i] == 1)
            R = add(R, P)
    end

    return R
}
```

Algorithm – generate key

```
p = 10997031918897188677
a = 3628449283386729367
b = 4889270915382004880
Gx = 3124469192170877657
Gy = 4370601445727723733
G = { Gy , Gx }
k = srand(0,p)

////////////Generate key/////////////

Deliver k from software by AXI to memory
K = scalar_mul(k , G)
Store K in memory
Deliver K to software by AXI

```

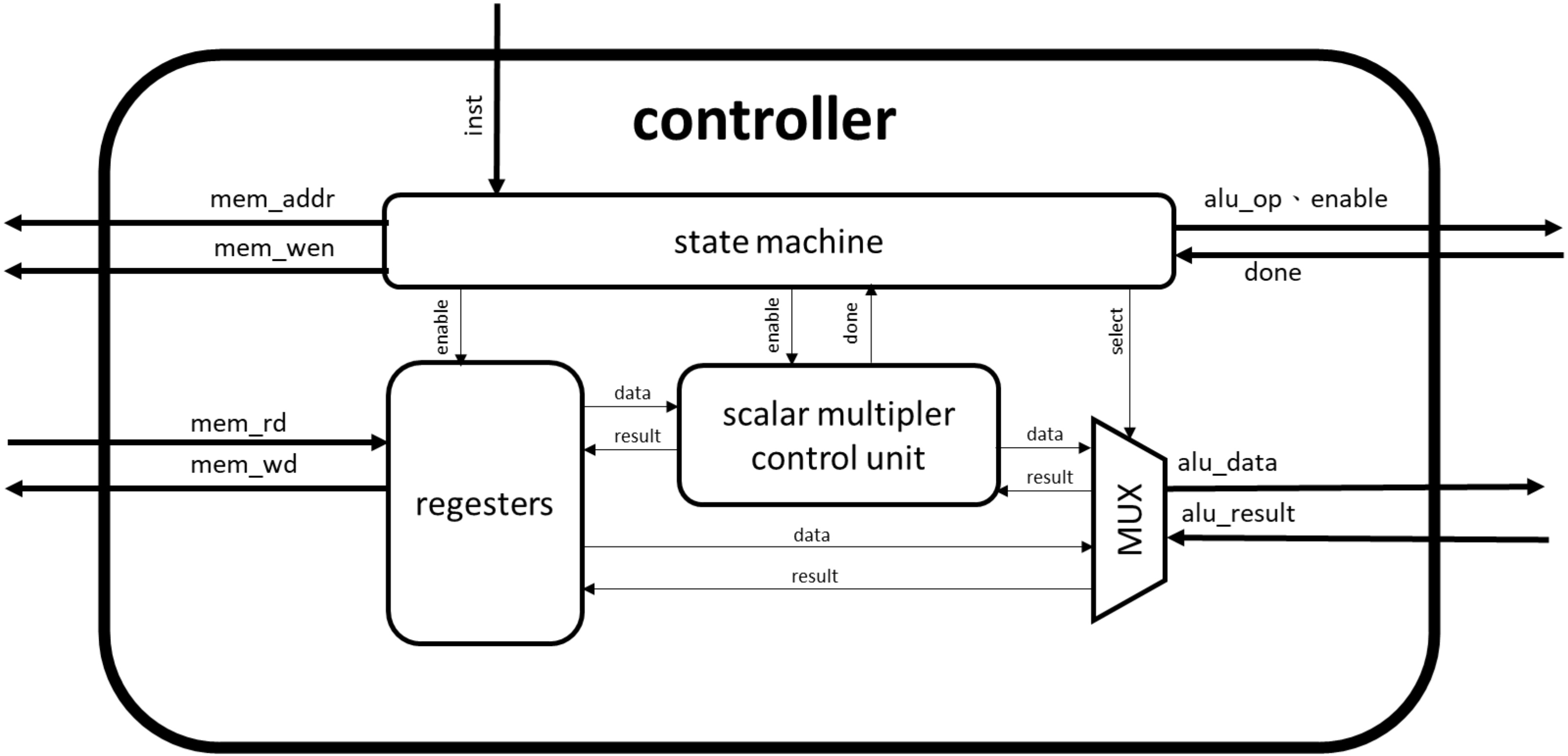
Algorithm – Encryption

```
p = 10997031918897188677
a = 3628449283386729367
b = 4889270915382004880
Gx = 3124469192170877657
Gy = 4370601445727723733
G = { Gy , Gx }
k = srand(0,p)

///////////////ENC/////////////////////

Deliver r,m from software by AXI to memory
C1 = scalar_mul(r , G)
C2 = add( m , scalar_mul(r , K) )  //C2 = m+rk
Store C1,C2 in memory & Deliver C1,C2 to software by AXI

```

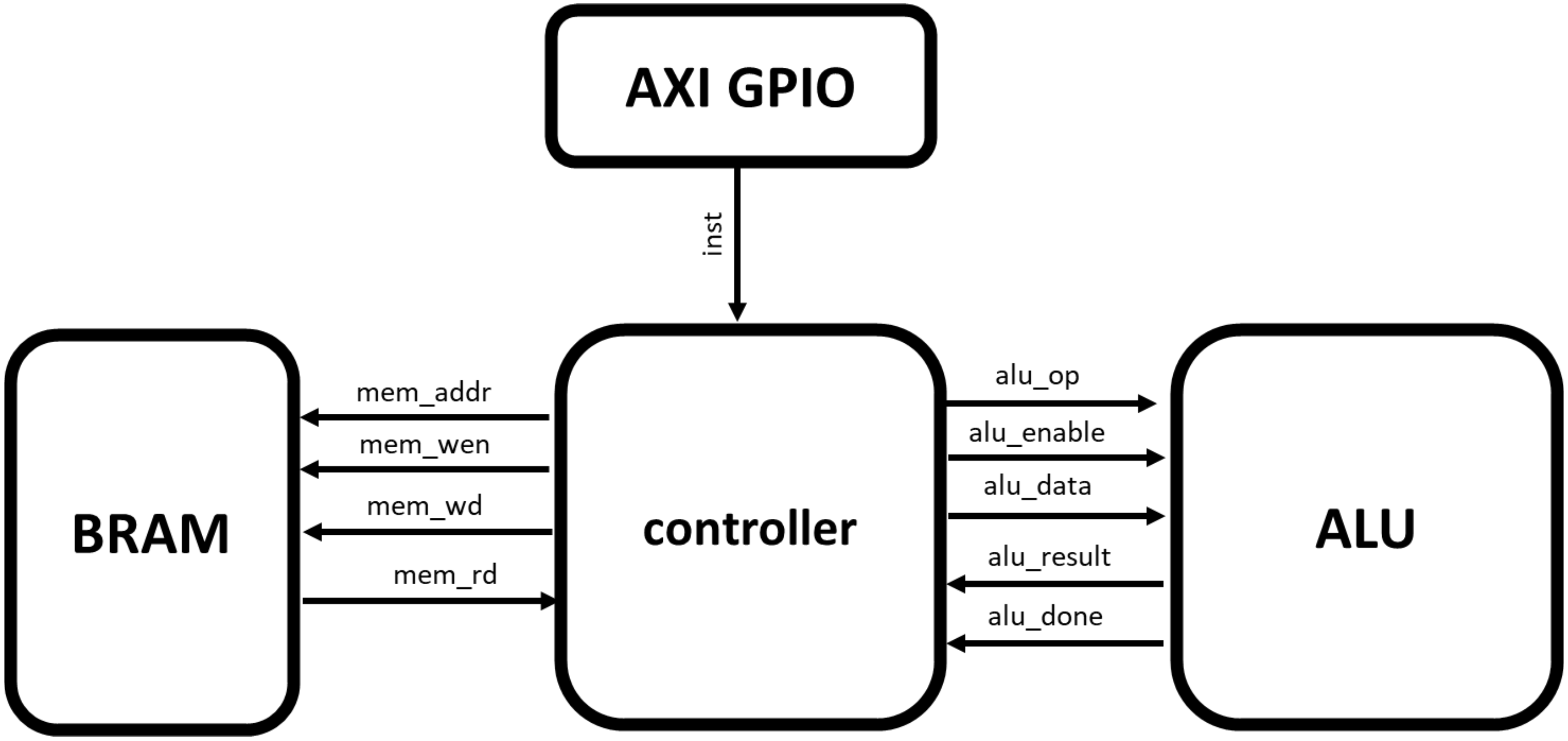
Algorithm – Decryption

```
p = 10997031918897188677
a = 3628449283386729367
b = 4889270915382004880
Gx = 3124469192170877657
Gy = 4370601445727723733
G = { Gy , Gx }
k = srand(0,p)

///////////////DEC//////////////////

Deliver C1,C2 from software by AXI to memory
M1 = scalar_mul(k,C1)
m = C2 - M1
Store m in memory & Deliver m to software by AXI

// k is private key
// K is public key

```

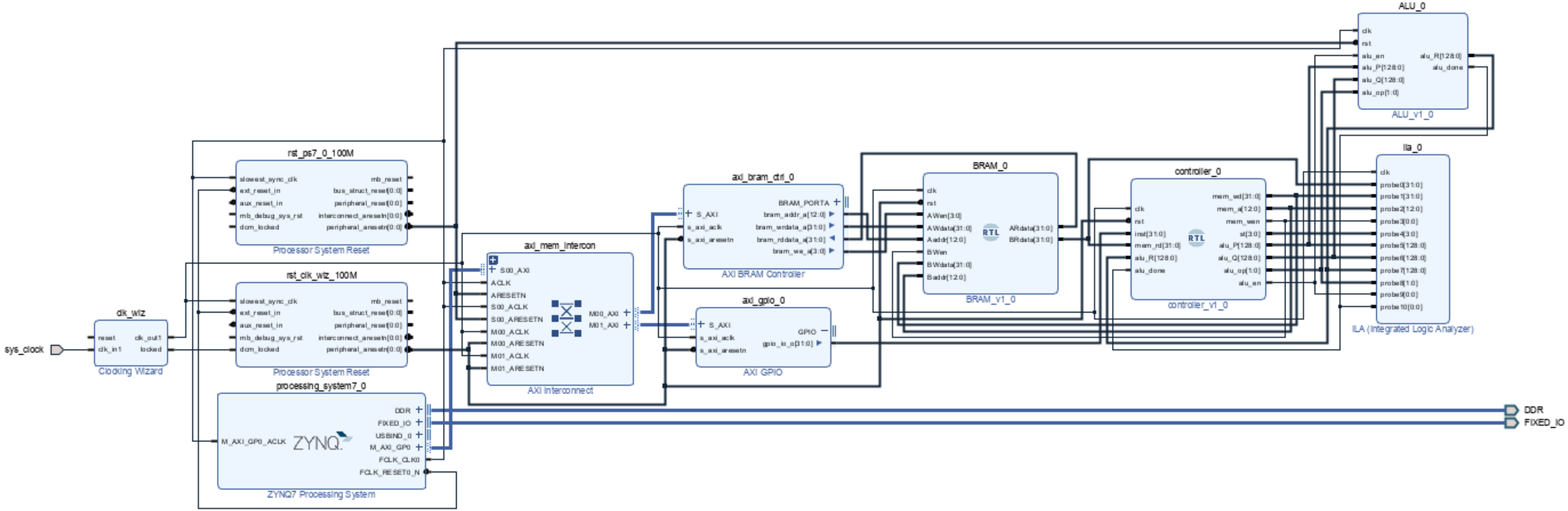
Controller Hardware Architecture



ECC System Hardware Architecture



Block diagram

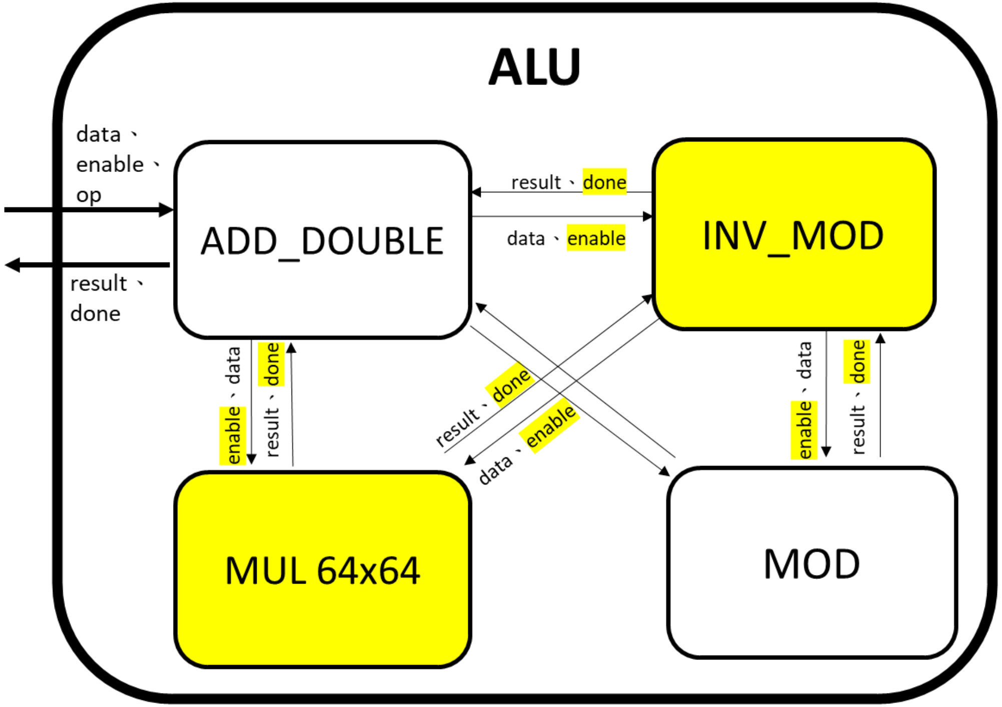


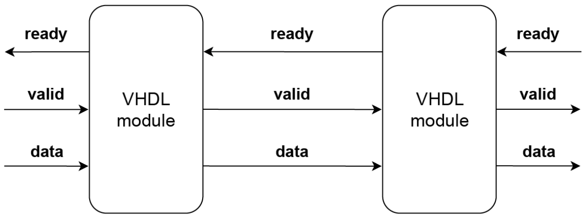
Outline

- Introduction to ECC
- Algorithm & Hardware Architecture
- Design features
- Result
- Problems to occur

Resource sharing

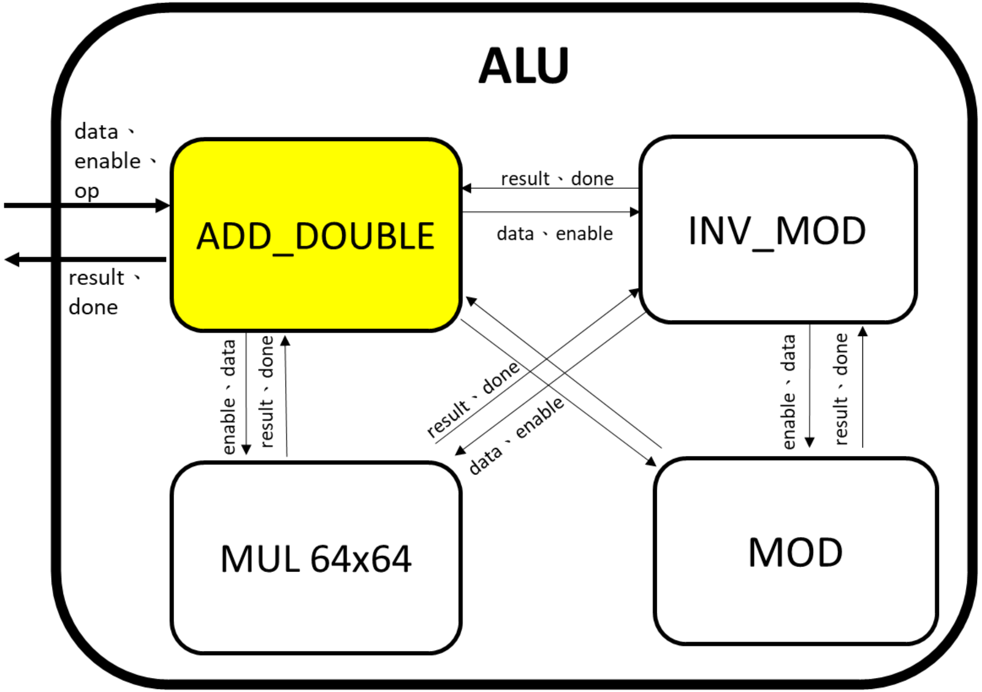
- Use handshake and stall to share hardware

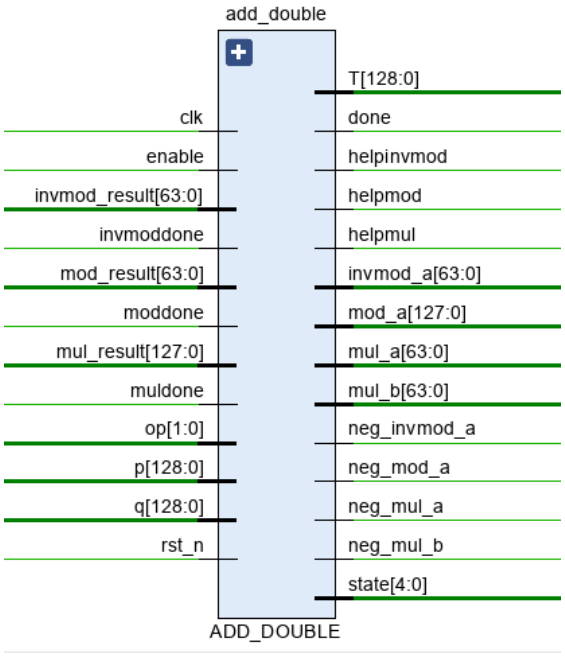




Resource sharing

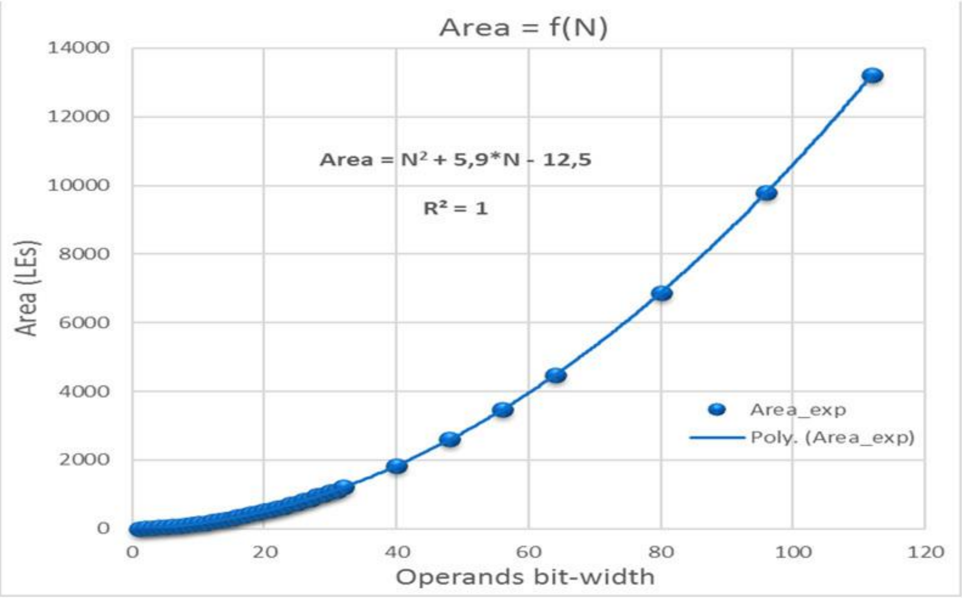
- Merge the similar function hardware to one module

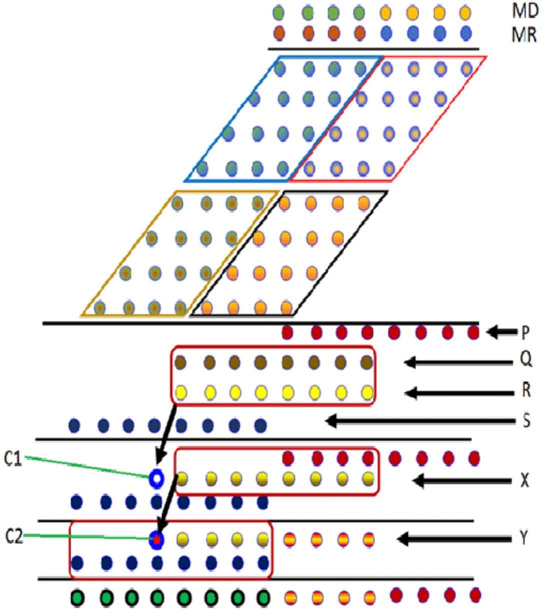




Resource sharing

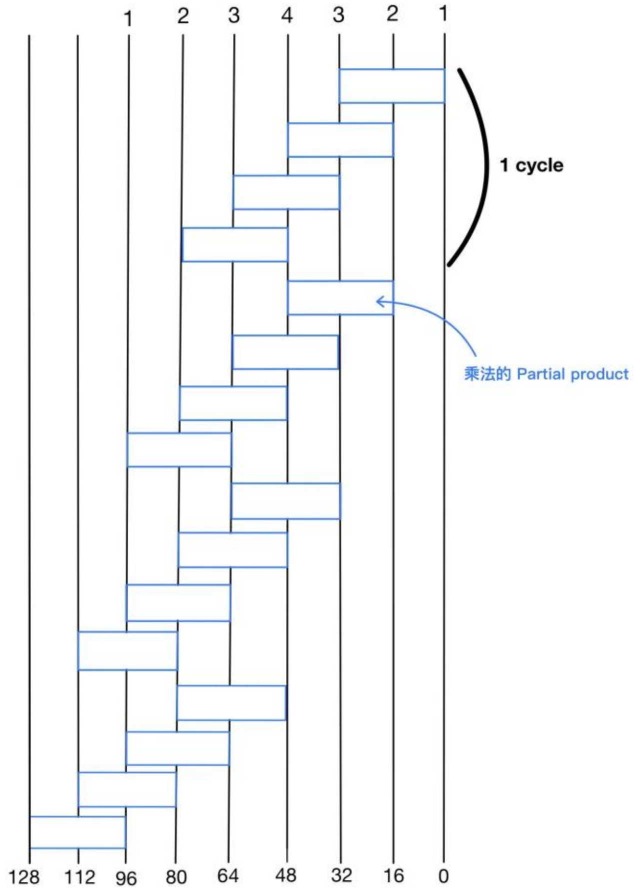
- Use **wallace tree** multiplier and sequential divider to solve large number calculation and share its sub-multiplier





parallelism of small DSP multiplier

- With split 64*64 multiplier into four 16*16 multiplier run four stage rather than four 32*32 multiplier run one stage
- Reduce critical path & period
- Can use DSP module in FPGA rather than LUT
- Area decreases significantly



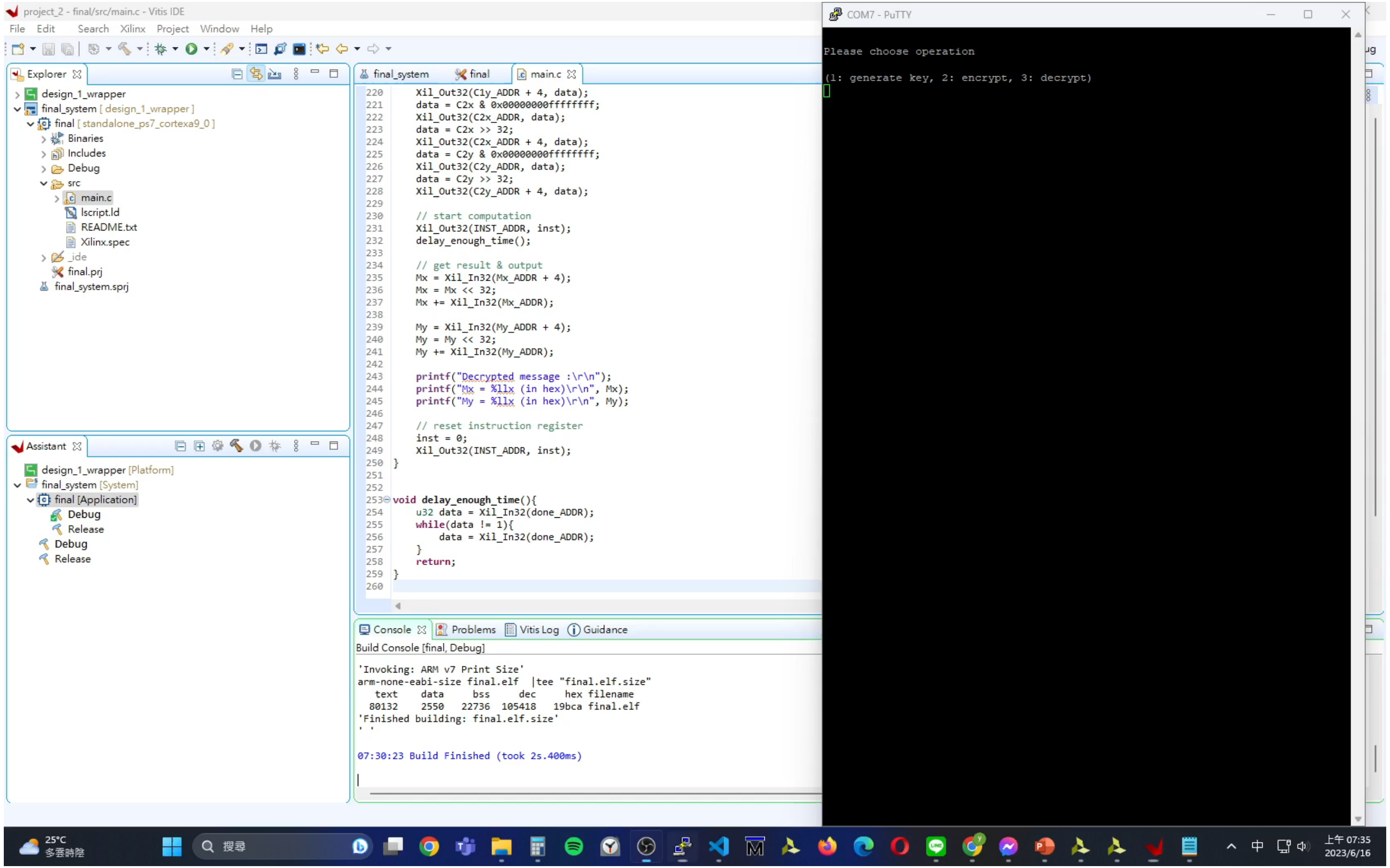
Outline

- Introduction to ECC
- Algorithm & Hardware Architecture
- Design features
- Results
- Problems to occur

Result

- 軟體執行(透過VITIS IDE 與硬體互動)
    1. 先將ECC的參數存入BRAM的指定位置
    2. 使用者輸入inst mode = 1 (generate key)利用私鑰k產生公鑰K
    3. 使用者輸入inst mode = 2 (encrypt) 並輸入公鑰、message(M)、random number來加密
    4. 使用者輸入inst mode = 3 (decrypt) 並輸入私鑰、被加密文件C1、C2來解密

Demo



project_2 - final/src/main.c - Vitis IDE
File Edit Search Xilinx Project Window Help
Explorer
design_1_wrapper
final_system [design_1_wrapper]
final [standalone_ps7_cortexa9_0]
Binaries
Includes
Debug
src
main.c
lscript.ld
README.txt
Xilinx.spec
_ide
final.prj
final_system.sprj
Assistant
design_1_wrapper [Platform]
final_system [System]
final [Application]
Debug
Release
Debug
Release
final_system
final
main.c
Xil_Out32(C1y_ADDR + 4, data);
data = C2x & 0x00000000ffffffff;
Xil_Out32(C2x_ADDR, data);
data = C2x >> 32;
Xil_Out32(C2x_ADDR + 4, data);
data = C2y & 0x00000000ffffffff;
Xil_Out32(C2y_ADDR, data);
data = C2y >> 32;
Xil_Out32(C2y_ADDR + 4, data);
// start computation
Xil_Out32(INST_ADDR, inst);
delay_enough_time();
// get result & output
Mx = Xil_In32(Mx_ADDR + 4);
Mx = Mx << 32;
Mx += Xil_In32(Mx_ADDR);
My = Xil_In32(My_ADDR + 4);
My = My << 32;
My += Xil_In32(My_ADDR);
printf("Decrypted message :\r\n");
printf("Mx = %llx (in hex)\r\n", Mx);
printf("My = %llx (in hex)\r\n", My);
// reset instruction register
inst = 0;
Xil_Out32(INST_ADDR, inst);
}
void delay_enough_time(){
u32 data = Xil_In32(done_ADDR);
while(data != 1){
data = Xil_In32(done_ADDR);
}
return;
}
Console Problems Vitis Log Guidance
Build Console [final, Debug]
'Invoking: ARM v7 Print Size'
arm-none-eabi-size final.elf |tee "final.elf.size"
text data bss dec hex filename
80132 2550 22736 105418 19bca final.elf
'Finished building: final.elf.size'
07:30:23 Build Finished (took 2s.400ms)
COM7 - PuTTY
Please choose operation
(1: generate key, 2: encrypt, 3: decrypt)
25°C
多雲時陰
搜尋
上午 07:35
2023/6/16

Result

- With something encryption and decryption error
- But with correct ECC arithmetic operation (ALU module)

```
 Loading work.INV_MOD
 Refreshing C:/Users/oppol/Desktop/FPGA/final_project/final/src/work.ADD_DOUBLE
 Loading work.ADD_DOUBLE
SIM 82> run -all
             109960 00000000000000010000000000000005 * 00000000000000010000000000000005  outcome:      3,       6, op=2
             219280 00000000000000010000000000000005 * 00000000000000030000000000000006  outcome:      6,      10, op=0
             328600 00000000000000010000000000000005 * 0000000000000006000000000000000a  outcome:      1,       3, op=0
             437920 00000000000000010000000000000005 * 00000000000000010000000000000003  outcome:     16,       9, op=0
             547240 00000000000000010000000000000005 * 00000000000000010000000000000009  outcome:     13,      16, op=0
             656560 00000000000000010000000000000005 * 000000000000000d0000000000000010  outcome:      6,       0, op=0
             765880 00000000000000010000000000000005 * 00000000000000060000000000000000  outcome:      7,      13, op=0
             875200 00000000000000010000000000000005 * 0000000000000007000000000000000d  outcome:      6,       7, op=0
             984520 00000000000000010000000000000005 * 00000000000000060000000000000007  outcome:     11,       7, op=0
            1093840 00000000000000010000000000000005 * 000000000000000b0000000000000007  outcome:     10,      13, op=0
            1203160 00000000000000010000000000000005 * 000000000000000a000000000000000d  outcome:     11,       0, op=0
            1312480 00000000000000010000000000000005 * 000000000000000b0000000000000000  outcome:      4,      16, op=0
            1421800 00000000000000010000000000000005 * 00000000000000040000000000000010  outcome:      1,       9, op=0
            1531120 00000000000000010000000000000005 * 00000000000000010000000000000009  outcome:     16,       3, op=0
            1640440 00000000000000010000000000000005 * 00000000000000010000000000000003  outcome:     11,      10, op=0
            1749760 00000000000000010000000000000005 * 000000000000000b000000000000000a  outcome:     14,       6, op=0
            1859080 00000000000000010000000000000005 * 000000000000000e0000000000000006  outcome:     16,       5, op=0
            1968400 00000000000000010000000000000005 * 00000000000000010000000000000005  outcome:     16,       7, op=0
 ** Note: $finish    : C:/Users/oppol/Desktop/FPGA/final_project/final/src/alu_tb.v(161)
    Time: 1968400 ns  Iteration: 3  Instance: /alu tb
```

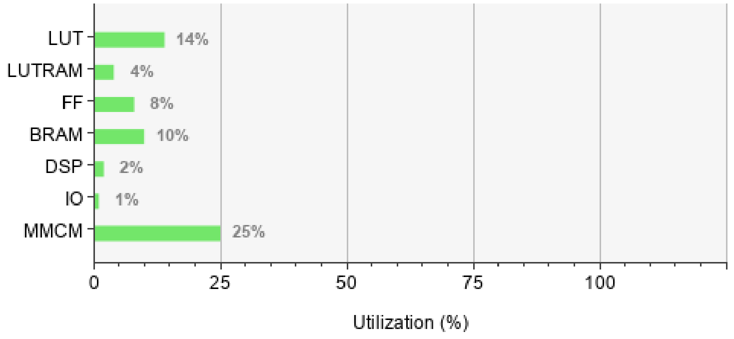
$$E: \quad y^2 \equiv x^3 + 2x + 2 \pmod{17}$$

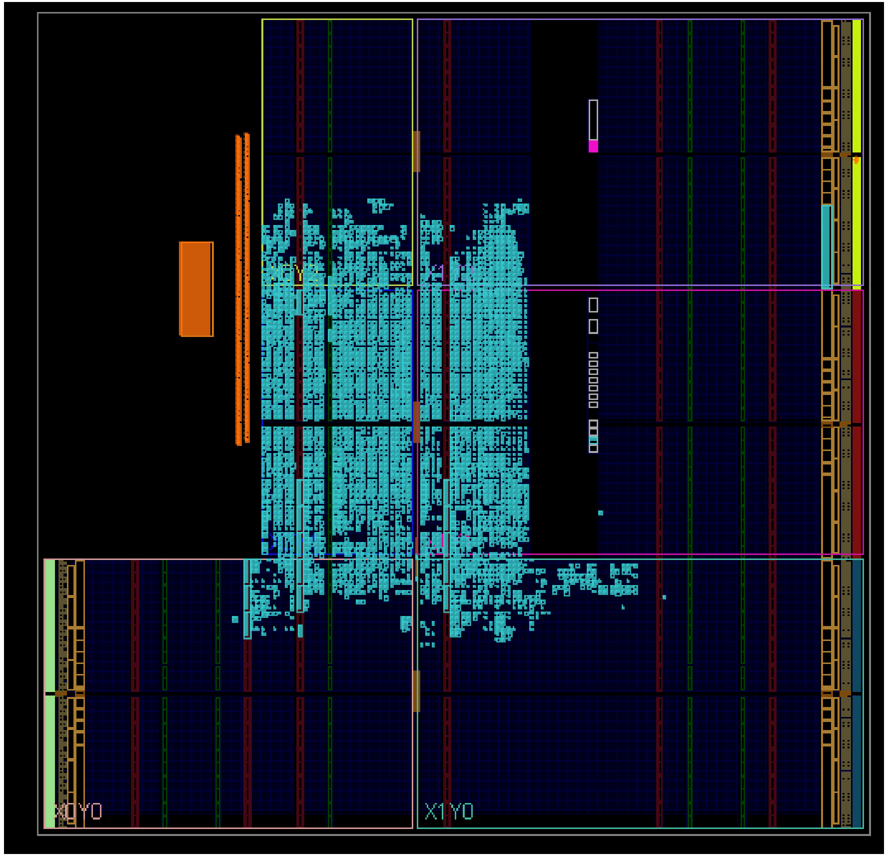
$$\begin{aligned}
G &= (5,1) & 11G &= (13,10) \\
2G &= (6,3) & 12G &= (0,11) \\
3G &= (10,6) & 13G &= (16,4) \\
4G &= (3,1) & 14G &= (9,1) \\
5G &= (9,16) & 15G &= (3,16) \\
6G &= (16,13) & 16G &= (10,11) \\
7G &= (0,6) & 17G &= (6,14) \\
8G &= (13,7) & 18G &= (5,16) \\
9G &= (7,6) & 19G &= \mathcal{O} \\
10G &= (7,11) & &
\end{aligned}$$

因此 n = 19 , h = 1

Utilization

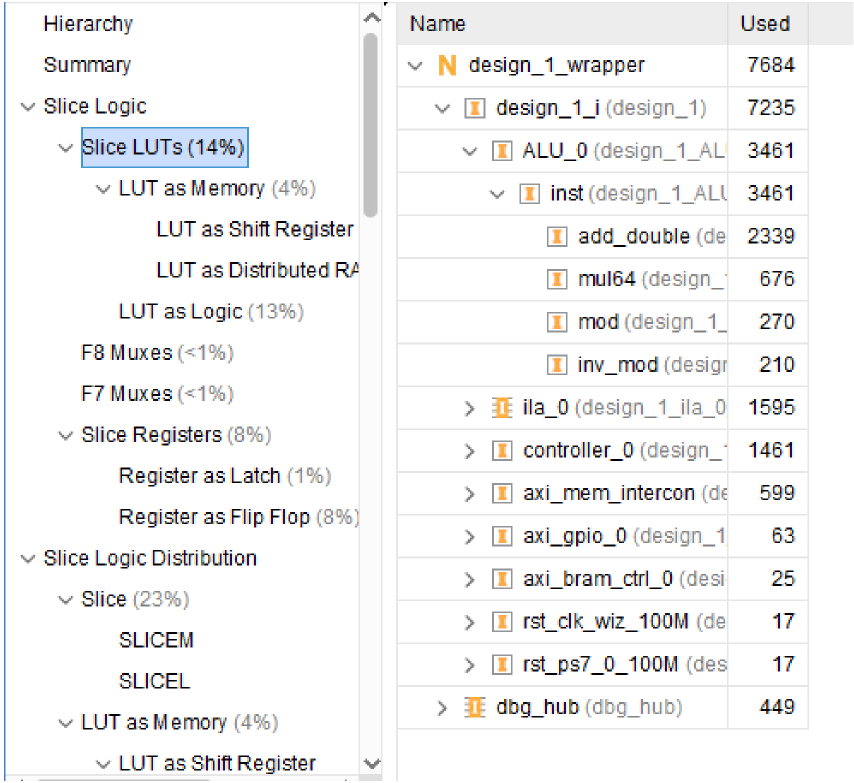
Resource	Utilization	Available	Utilization %
LUT	7684	53200	14.44
LUTRAM	658	17400	3.78
FF	8465	106400	7.96
BRAM	14.50	140	10.36
DSP	4	220	1.82
IO	1	125	0.80
MMCM	1	4	25.00





LUT Utilization

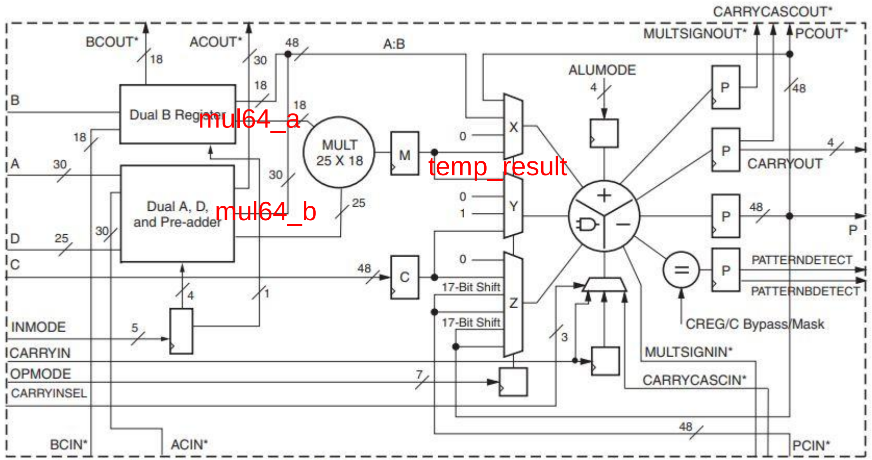
- LUT utilization is dominated by add_double module
- Multiply is replaced by DSP module in mul64
- Mod area is dominated only two 128bits subtraction



DSP Utilization

- With split 64*64 multiplier into four 16*16 multiplier run four stage

Name	Slice LUTs (53200)	Slice Registers (106400)	F7 Muxes (26600)	F8 Muxes (13300)	Slice (13300)	LUT as Logic (53200)	LUT as Memory (17400)	Block RAM Tile (140)	DSPs (220)
design_1_wrapper	7684	8465	54	1	2995	7026	658	14.5	4
design_1_i (design_1)	7235	7724	54	1	2765	6601	634	14.5	4
ALU_0 (design_1_AL	3461	1584	0	0	1005	3461	0	0	4
inst (design_1_ALU	3461	1584	0	0	1005	3461	0	0	4
mul64 (design_	676	645	0	0	237	676	0	0	4
mod (design_1	270	200	0	0	248	270	0	0	0



*These signals are dedicated routing paths internal to the DSP48E1 column. They are not accessible via fabric routing resources.

UG369_c1_01_052109

Figure 2-1: **7 Series FPGA DSP48E1 Slice**

Outline

- Introduction to ECC
- Algorithm & Hardware Architecture
- Results
- Design features
- Problems to occur

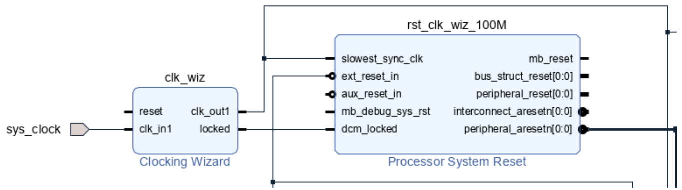
Setup time violation

- (Sol)
  - 將乘法器和除法器一個stage的critical path再縮短
  - 使用clock wizard將clk period提高至slack>0

ports | Design Runs | Power | DRC | Methodology | Timing

Intra-Clock Paths - clk_fpga_0 - Setup

Name	Slack	Levels	High F...	From	To	Total Delay	Logic Delay	Net Delay
Path 1	-488.278	658	282	design_1_i/ALU_0/inst/add_double/px_reg[7]/C	design_1_i/ALU_0/inst/mod/dand5_reg[127]_P/D	498.201	191.773	306.429
Path 2	-487.902	658	282	design_1_i/ALU_0/inst/add_double/px_reg[7]/C	design_1_i/ALU_0/inst/mod/dand5_reg[127]_C/D	497.821	191.773	306.048
Path 3	-475.845	642	282	design_1_i/ALU_0/inst/add_double/px_reg[7]/C	design_1_i/ALU_0/inst/mod/dand5_reg[126]_P/D	485.734	187.410	298.324
Path 4	-475.735	642	282	design_1_i/ALU_0/inst/add_double/px_reg[7]/C	design_1_i/ALU_0/inst/mod/dand5_reg[126]_C/D	485.745	187.410	298.336
Path 5	-474.538	643	282	design_1_i/ALU_0/inst/add_double/px_reg[7]/C	design_1_i/ALU_0/inst/mod/dand5_reg[125]_C/D	484.503	187.198	297.305
Path 6	-473.837	642	282	design_1_i/ALU_0/inst/add_double/px_reg[7]/C	design_1_i/ALU_0/inst/mod/dand5_reg[125]_P/D	483.805	187.074	296.732
Path 7	-466.120	630	282	design_1_i/ALU_0/inst/add_double/qy_reg[0]/C	design_1_i/ALU_0/inst/mod/dand5_reg[124]_P/D	475.635	183.663	291.972
Path 8	-464.413	628	282	design_1_i/ALU_0/inst/add_double/px_reg[7]/C	design_1_i/ALU_0/inst/mod/dand5_reg[124]_C/D	474.375	183.380	290.995
Path 9	-462.862	627	282	design_1_i/ALU_0/inst/add_double/qy_reg[0]/C	design_1_i/ALU_0/inst/mod/dand5_reg[123]_P/D	472.643	182.863	289.780
Path 10	-462.420	625	282	design_1_i/ALU_0/inst/add_double/px_reg[7]/C	design_1_i/ALU_0/inst/mod/dand5_reg[123]_C/D	472.430	182.606	289.825



Name	Waveform	Period (ns)	Frequency (MHz)
clk_fpga_0	{0.000 5.000}	10.000	100.000
dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_E	{0.000 16.500}	33.000	30.303
sys_clock	{0.000 4.000}	8.000	125.000
clk_out1_design_1_clk_wiz_1	{0.000 10.000}	20.000	50.000
clkfbout_design_1_clk_wiz_1	{0.000 4.000}	8.000	125.000

Tools with verification

- large number calculator (* & mod)

https://www.calculator.net/big-number-calculator.html

- inverse modulo calculator

https://keisan.casio.com/exec/system/15901266097609

- ecc calculator

http://www.christelbach.com/eccalculator.aspx

Reference

- Hardware design and implementation of ECC based crypto processor for low-area-applications on FPGA https://ieeexplore.ieee.org/abstract/document/8279005?fbclid=IwAR3y2Ey7g9STRfPIBXAQByA_J2NVTw2d140kOitqDdDAnMINuXb0hFFYOU4
- **非對稱式加密演算法 - 橢圓曲線密碼學 Elliptic Curve Cryptography , ECC (觀念篇)** **https://ithelp.ithome.com.tw/articles/10251031**
- How can I best check these Elliptic Curve parameters are valid? https://stackoverflow.com/questions/22270485/how-can-i-best-check-these-elliptic-curve-parameters-are-valid
- Elliptic Curve Cryptography (ECC) by Christof Paar https://www.youtube.com/watch?v=zTt4gvuQ6sY&t=4805s

Thanks for listening