FPGA DESIGN

Final project
Report

	姓名	學號
組員 1	郭家佑	F64096114
組員 2	蔡奇佑	E24096213
組員 3	趙泓瑞	E14071025

- ECC encryption introduction

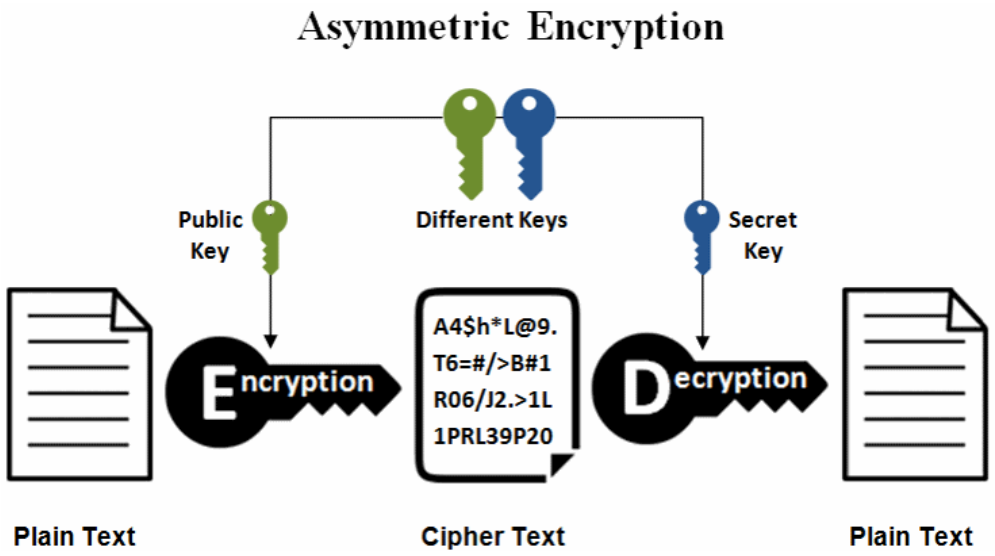
ECC(Elliptic Curve Cryptography) 是一種非對稱密碼學。對非對稱加密算法利用公鑰(Public Key)對明文(Plain text)進行加密產生密文(Cipher text)，再利用私鑰(Private Key)對密文進行解密，產生明文。

其優點有:

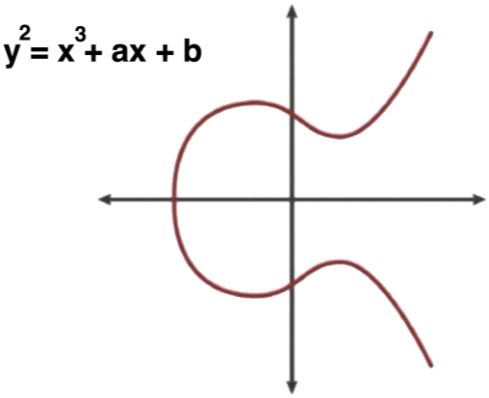
1. **處理速度更快**，在計算速度上，ECC 比 RSA、DSA 快得多

2. **儲存空間更小**

3. **安全性能更高**，160 位 ECC 和 1024 位 RSA、DSA 有相同的安全強度



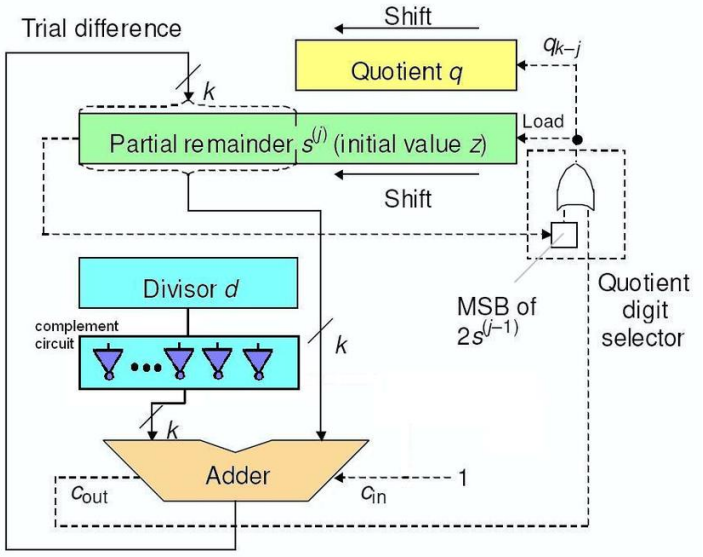
而 ECC 是一種基於橢圓取線上取利散點進行運算的密碼學，橢圓取縣的方程式表現如下：



在我們的加解密過程中，會需要用的橢圓取線上的點加法與標量乘法運算。

- Hardware Design
  - Mod

Modulus(以下稱呼 Mod)利用長除法，從 MSB 開始的 64 位元與 P 做比較，如果取出的 partial remainder 比較大，則做相減，相反則不用。存回 partial remainder 後向左位移一位。依此類推。由於我們將上述運算做兩極的疊接後，可以減半 mod p 一次所需要的 cycles。

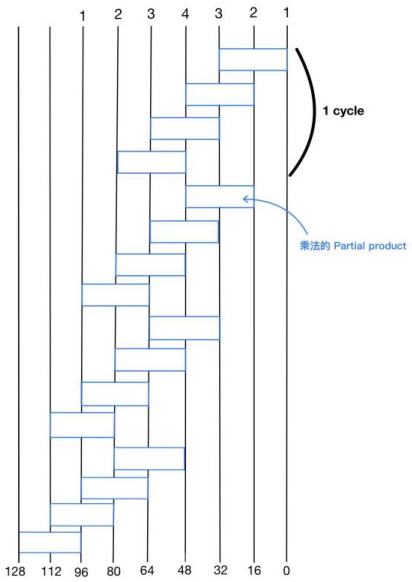


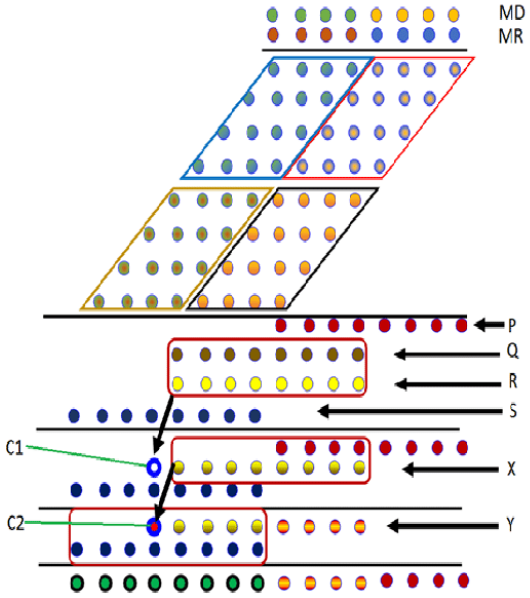
但經過老師的教導後發現除法器是最沒有效率的做法，原因是其有著嚴重的前後相依性，因此使用底下 pre-devision + shift 有著更好的效率(乘法可用平行處理加速)

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

  - Mul64*64

Mul64*64，將分別兩個 64bits 的 input 做 16bit*16bit 的乘法，每一 cycle 將乘數的 16bits 乘以被乘數 64bits 做 partial product 的疊加，4 個 cycle 即可完成 64 位元的兩數乘法。





但經過老師的教導後發現其實做法可以是先將 64*64 先拆成 4 個 32*32，再將 32*32 拆成 16*16 平行化處理，不用一次 64 拆成 16

■ Inv_mod

利用費馬小定理求 mod p 時 $a^{p-1}= 1$ 的性質，因為我們需要 1/a，所以兩側同除以 a 得 Inv_mod 的標準式$a^{p-2}= 1/a$。利用模冪的特性以輪到當下的 exp bit 做迭代 a 的二的指數次方疊乘，即完成。

```
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
}
```

■ Add_double

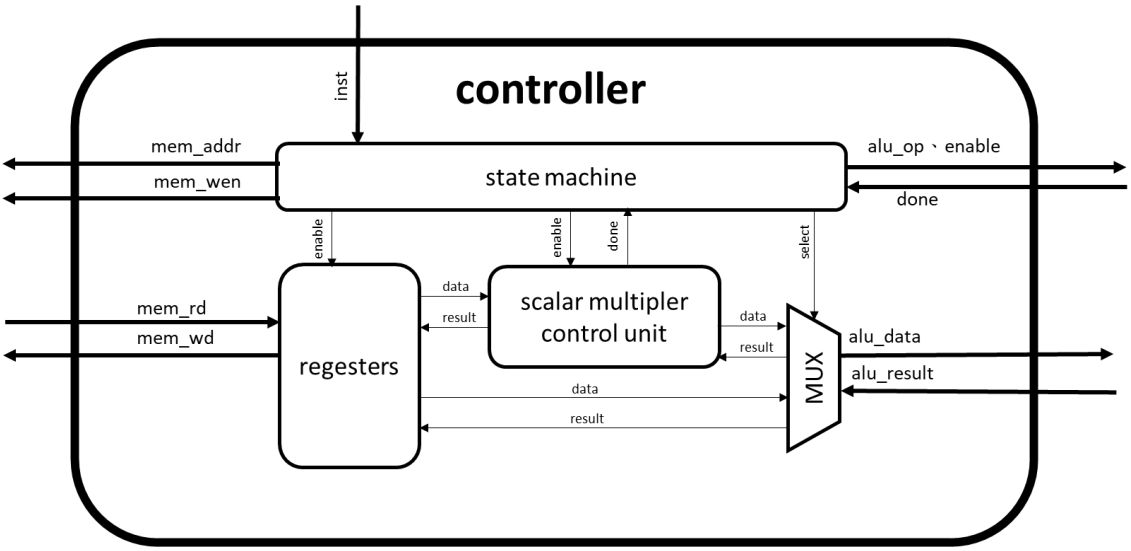
因演算法過程發現 add subtraction double 的方法極為相似，所以先判對輸入資料是否為無限。倘若其中一方為有限，則需判斷是否為被減數，是的話其 y 軸的資料要取負號。不是的話取輸出有限值。若兩者皆為有限，則依據演算法求得其斜率，並藉由與 Inv_mod、Mul64*64 和 Mod 換算出當下的運算的結果。

```
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
}
```

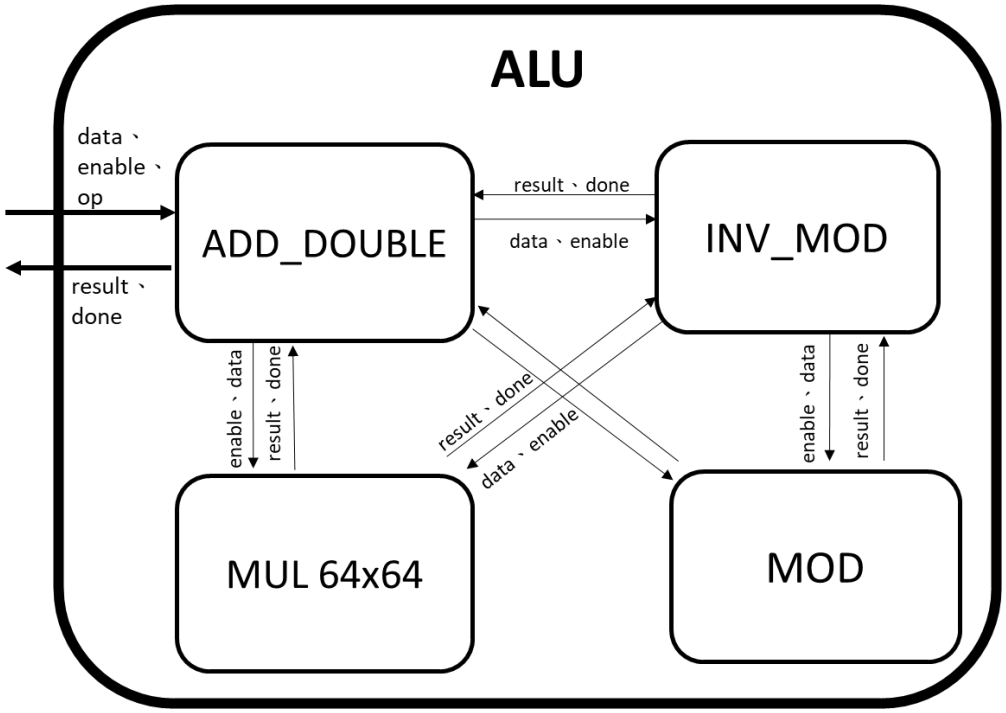
■ Controller

Controller 作為 generation key 和加解密的三大功能的重要電路，藉由 axi 與軟體端要密鑰 k 到記憶體，與預設的標準點做矢向量計算後即為公鑰 K。將公鑰傳回軟體端，即完成 generation key。將 r 跟 m 從軟體端傳入硬體後，將 r 跟標準點做 scalar multiplication 後暫存 C1。以及將 r 跟公鑰 K 做 scalar multiplication 後跟 m 做點加法後暫存 C2。最後將兩者交回給軟體端即為加密。如果我們將剛剛兩者從軟體端要回來後密鑰 k 與 C1 做 scalar multiplication 後存暫存 M1，C2-M1 後存回軟體端即為解密。

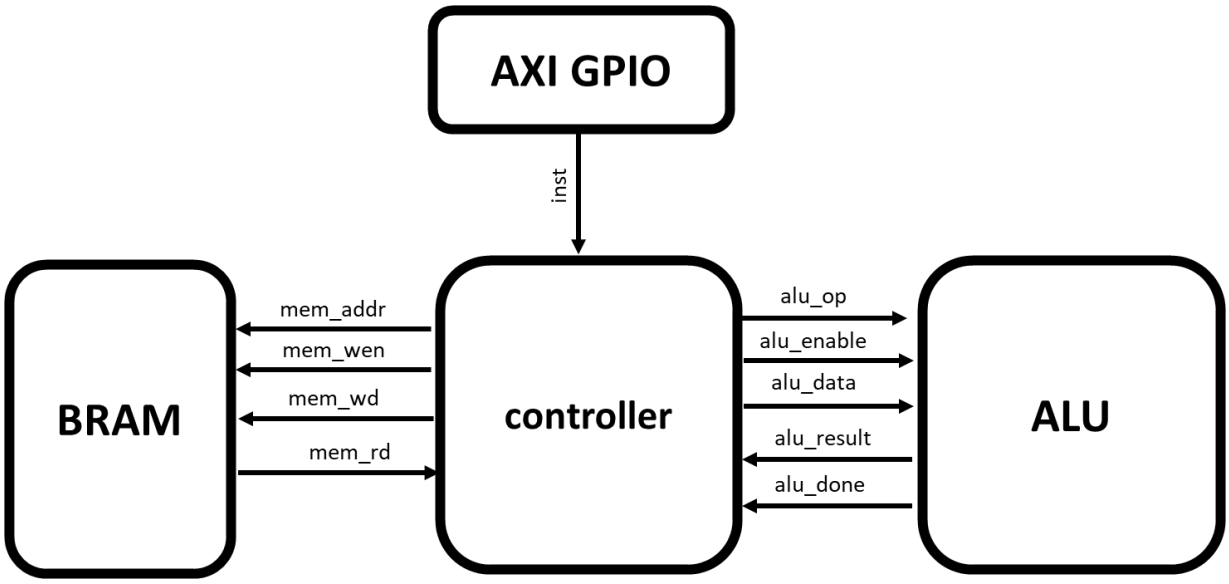
- Hardware Architecture
  - Submodule
    - Controller



    - ALU



  - Top module

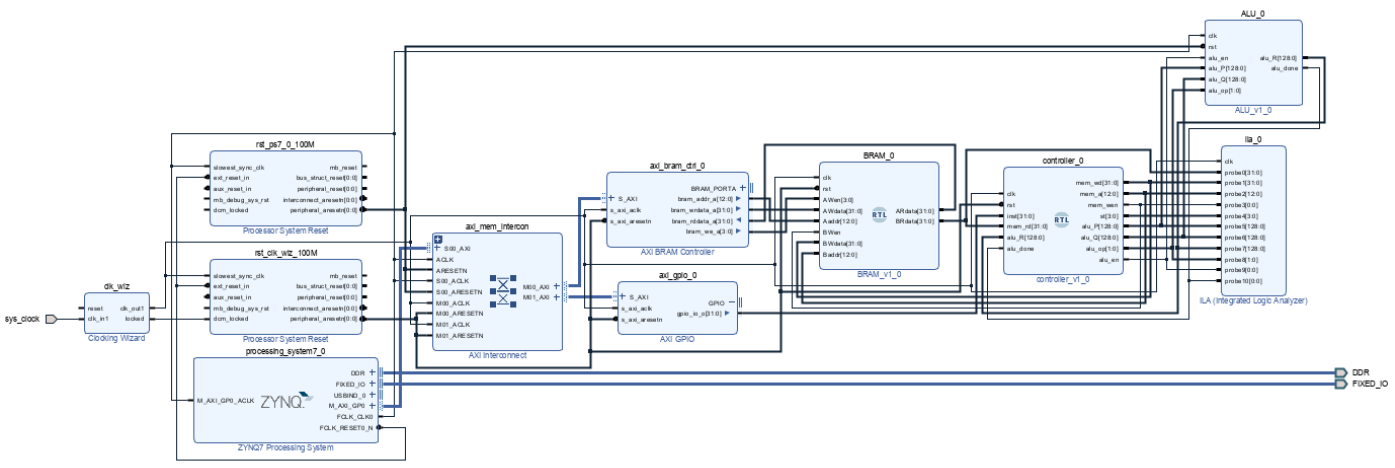


PS 端透透過 GPIO 將指令傳送給 controller，而 controller 會根據指令去運作。當指令為 1 時，會利用私鑰 k 產生公鑰 K；當指令為 2 時，會用輸入進來的公鑰、message(M)、random number 來加密，輸出一組密文存回 Bram；當指令為 3 時，會將輸入進來的私鑰、被加密文件 C1、C2 解密回 M 並存回 Bram 裡面。

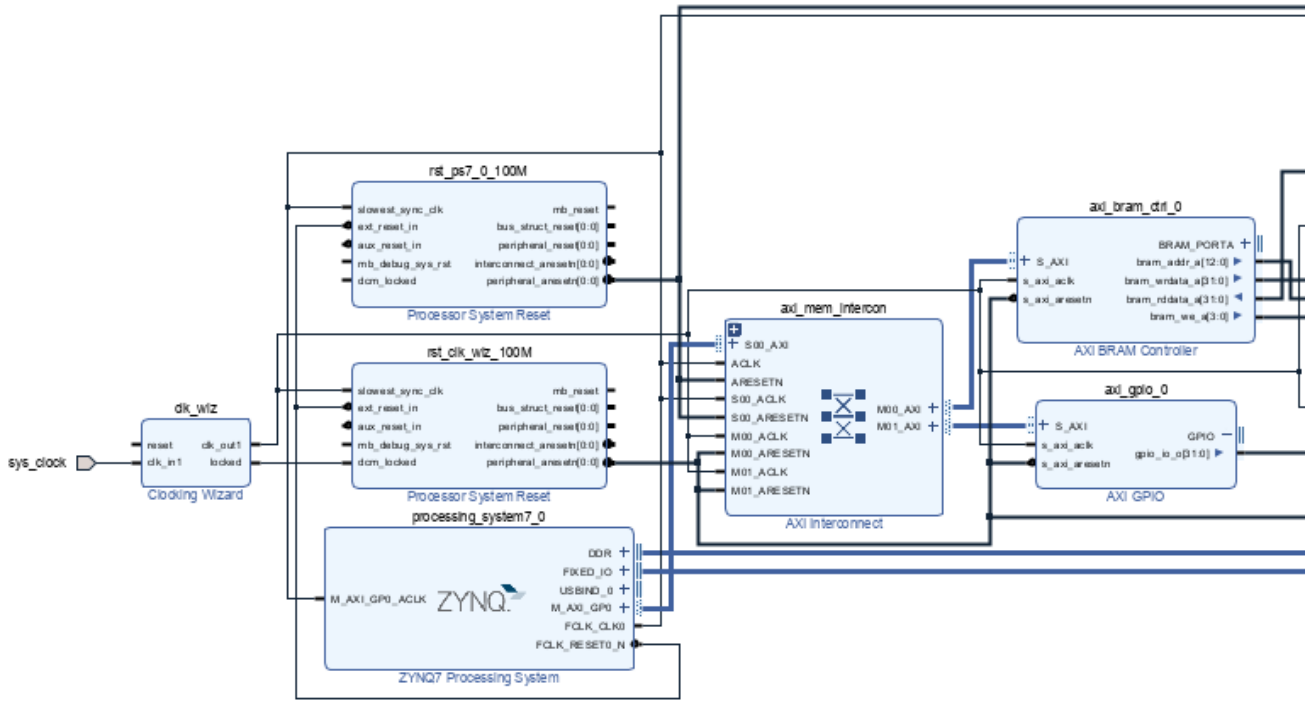
- Block Design

  1. 使用 clk_wizard 去產生不同的頻率來解決 setup-time violation。
  2. 使用 ILA IP 來 debug

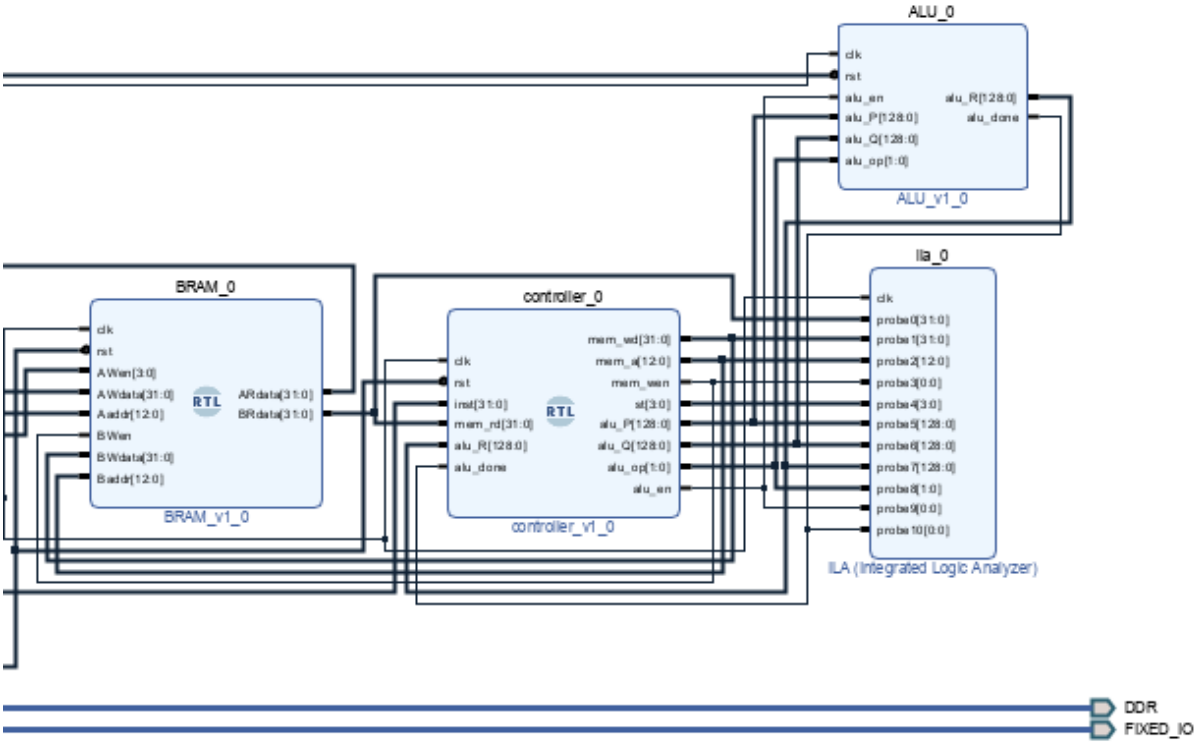
全



左

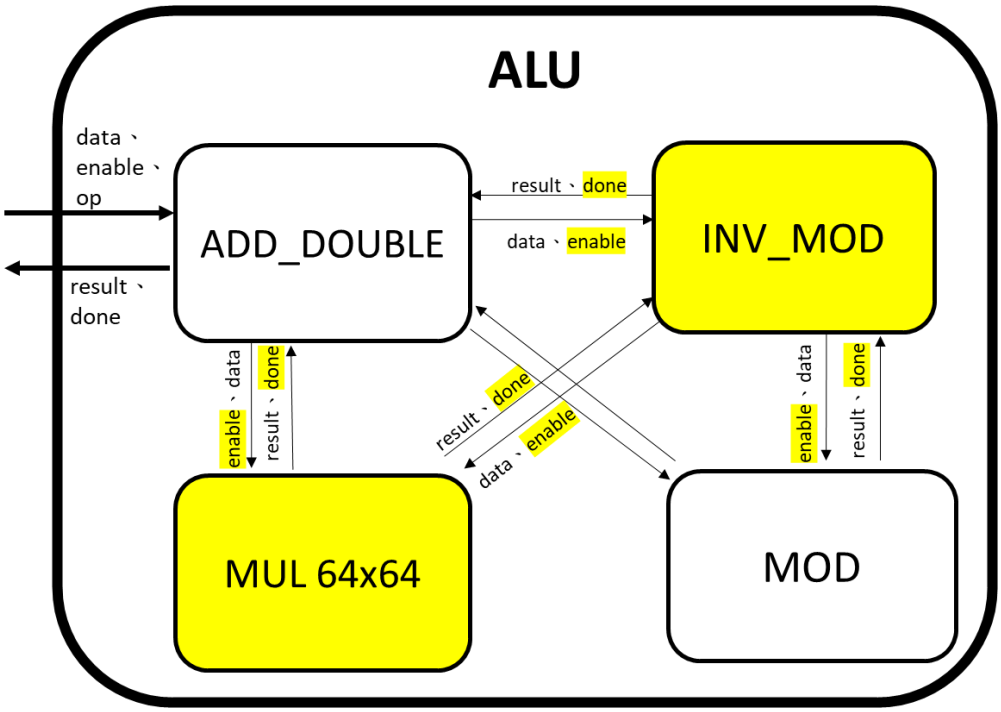

rst_ps7_0_100M
Processor System Reset
rst_clk_wiz_100M
Processor System Reset
clk_wiz
Clocking Wizard
sys_clock
processing_system7_0
ZYNQ7 Processing System
axi_mem_intercon
AXI Interconnect
axi_bram_ctrl_0
AXI BRAM Controller
axi_gpio_0
AXI GPIO

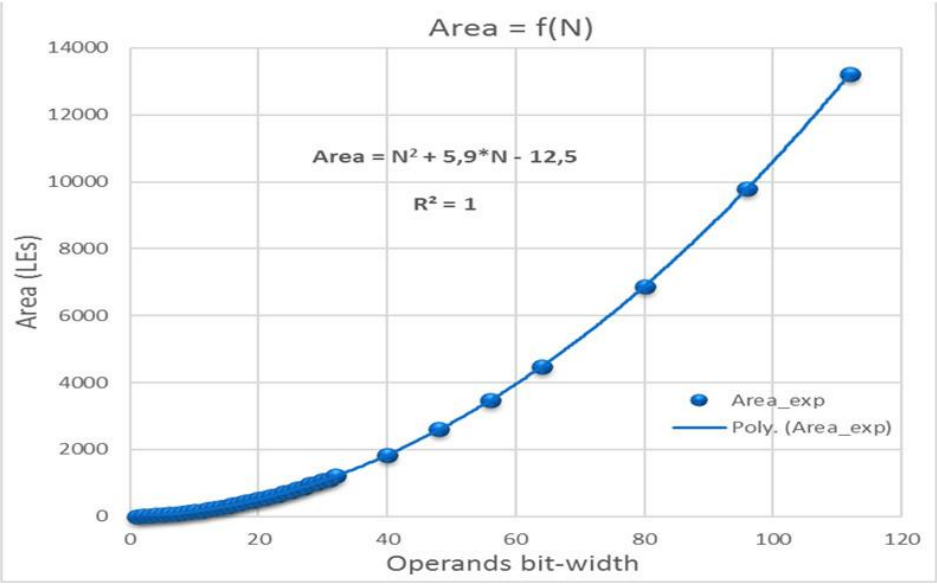
右


ALU_0
ALU_v1_0
BRAM_0
BRAM_v1_0
controller_0
controller_v1_0
ila_0
ILA (Integrated Logic Analyzer)
DDR
FIXED_IO

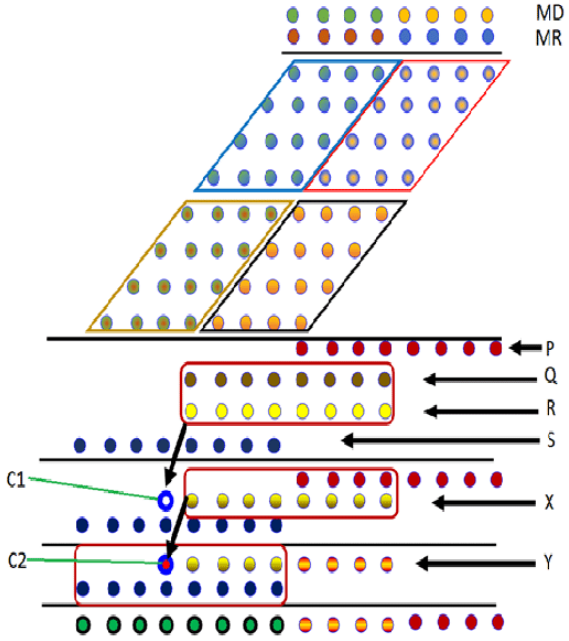
- Design feature

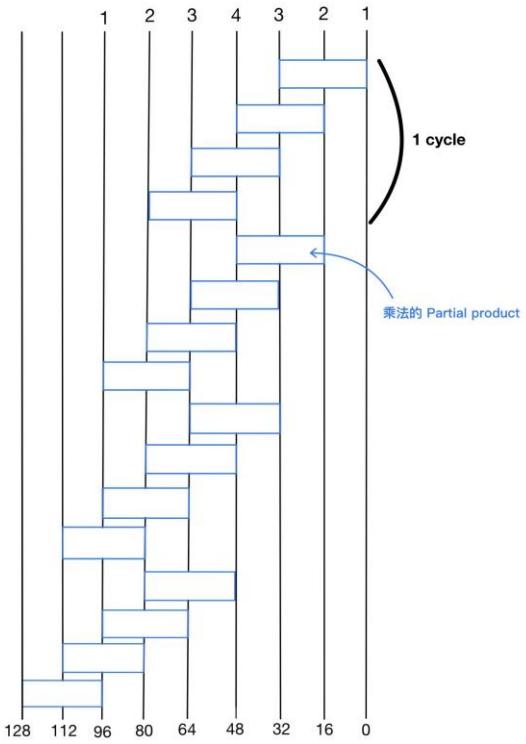
在我們的系統架構中，我們在 ALU 進行的設計上採用了一些 resource sharing 的架構設計。ALU 的內部架構如下圖所示，由於橢圓曲線的點加法、點減法、點加倍運算流程皆類似，因此我們將三種功能合併成一個 ADD_DOUBLE Ccontrol Unit。此外，由於 ADD_DOUBLE 與 inverse mod 皆會使用到乘法器與 modulus unut，因此我們利用 stall 與 handshake 的機制來共用硬體。





在我們的乘法器中，由於乘法器面積與乘法 bit 數量成二次多項事關係，且因考慮到 DSP module 的輸入限制，因此我們利用 wallace tree multiplier 將 64bit 乘法運算拆分成 16 個 16bit 的乘法運算(如下圖左所示，示意圖僅供參考，非實際架構)，再將乘法器複製 4 份，每一個 cycle 皆計算 4 組乘法，再利用 4 個 cycle 將 partial product 進行加總，最後利用 8 個 cycle 的出 64bit 的乘法結果(如下圖右)。





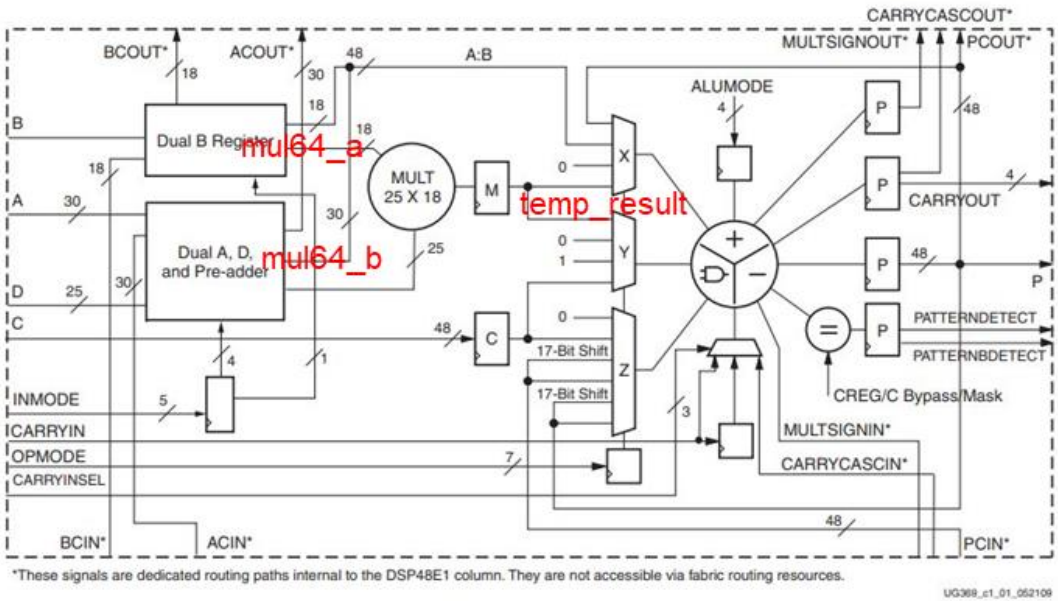


Figure 2-1: **7 Series FPGA DSP48E1 Slice**

- Simulation result (use VITIS IDE to sumulate)

1. generate key

```
Please choose operation

(1: generate key, 2: encrypt, 3: decrypt)
mode is 1

Please input private key k : (in hex)
 k is : 66

Public Key :
Kx = 5f4c2e935e437212 (in hex)
Ky = 64774be9e77b27bc (in hex)
```

2. encryption

```
Please choose operation

(1: generate key, 2: encrypt, 3: decrypt)
mode is 2

Please input x position of public key Kx : (in hex)
Kx is 5f4c2e935e437212

Please input y position of public key Ky : (in hex)
Ky is 64774be9e77b27bc

Please input x position of message Mx : (in hex)
Mx is 1a0a

Please input y position of message My : (in hex)
My is 8f5c356536c9c638

Please input a random number r for encryption : (in hex)
 r is 5
Encrypted data :
C1x = 586ce3a67803a04d (in hex)
C1y = 5bf5cf683216ee70 (in hex)
C2x = 58269077f58dab89 (in hex)
C2y = 4ed1cc5e6f278f73 (in hex)
```

3. decryption

```
Please choose operation

(1: generate key, 2: encrypt, 3: decrypt)
mode is 3

Please input private key k : (in hex)
k is 66

Please input encrypted data C1x : (in hex)
C1x is 586ce3a67803a04d

Please input encrypted data C1y : (in hex)
C1y is 5bf5cf683216ee70

Please input encrypted data C2x : (in hex)
C2x is 58269077f58dab89

Please input encrypted data C2y : (in hex)
C2y is 4ed1cc5e6f278f73
Decrypted message :
Mx = bcf2249014e0131 (in hex)
My = 6767caa0dba7aad2 (in hex)
```

4.correction of ECC arithmetic operation (testbench is in the src folder)

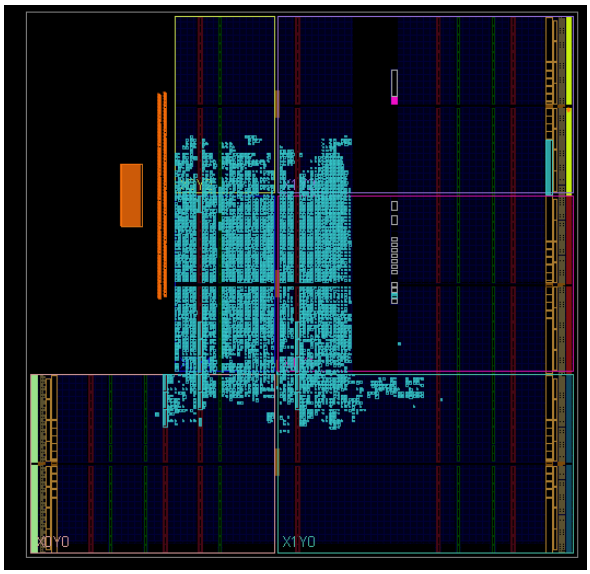
```
 Loading work.INV_MOD
 Refreshing C:/Users/oppo1/Desktop/FPGA/final_project/final/src/work.ADD_DOUBLE
 Loading work.ADD_DOUBLE
SIM 82> run -all
              109960 0000000000000001000000000000005 * 0000000000000001000000000000005  outcome:                3,                6, op=2
              219280 0000000000000001000000000000005 * 0000000000000003000000000000006  outcome:                6,               10, op=0
              328600 0000000000000001000000000000005 * 000000000000000600000000000000a  outcome:                1,                3, op=0
              437920 0000000000000001000000000000005 * 0000000000000001000000000000003  outcome:               16,                9, op=0
              547240 0000000000000001000000000000005 * 0000000000000001000000000000009  outcome:               13,               16, op=0
              656560 0000000000000001000000000000005 * 000000000000000d000000000000010  outcome:                6,                0, op=0
              765880 0000000000000001000000000000005 * 000000000000000600000000000000000  outcome:                7,               13, op=0
              875200 0000000000000001000000000000005 * 000000000000000700000000000000d  outcome:                6,                7, op=0
              984520 0000000000000001000000000000005 * 000000000000000600000000000000007  outcome:               11,                7, op=0
             1093840 0000000000000001000000000000005 * 000000000000000b000000000000007  outcome:               10,               13, op=0
             1203160 0000000000000001000000000000005 * 000000000000000a000000000000000d  outcome:               11,                0, op=0
             1312480 0000000000000001000000000000005 * 000000000000000b000000000000000  outcome:                4,               16, op=0
             1421800 0000000000000001000000000000005 * 0000000000000004000000000000010  outcome:                1,                9, op=0
             1531120 0000000000000001000000000000005 * 0000000000000001000000000000009  outcome:               16,                3, op=0
             1640440 0000000000000001000000000000005 * 0000000000000001000000000000003  outcome:               11,               10, op=0
             1749760 0000000000000001000000000000005 * 000000000000000b000000000000000a  outcome:               14,                6, op=0
             1859080 0000000000000001000000000000005 * 000000000000000e000000000000006  outcome:               16,                5, op=0
             1968400 0000000000000001000000000000005 * 0000000000000001000000000000005  outcome:               16,                7, op=0
** Note: $finish    : C:/Users/oppo1/Desktop/FPGA/final_project/final/src/alu_tb.v(161)
   Time: 1968400 ns  Iteration: 3  Instance: /alu_tb
```

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

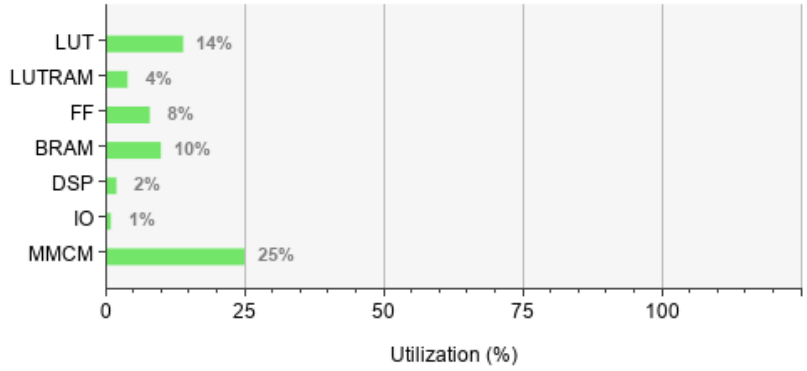
$$\begin{aligned}
G &= (5,1) & 11G &= (13,10)\\
2G &= (6,3) & 12G &= (0,11)\\
3G &= (10,6) & 13G &= (16,4)\\
4G &= (3,1) & 14G &= (9,1)\\
5G &= (9,16) & 15G &= (3,16)\\
6G &= (16,13) & 16G &= (10,11)\\
7G &= (0,6) & 17G &= (6,14)\\
8G &= (13,7) & 18G &= (5,16)\\
9G &= (7,6) & 19G &= \mathcal{O}\\
10G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

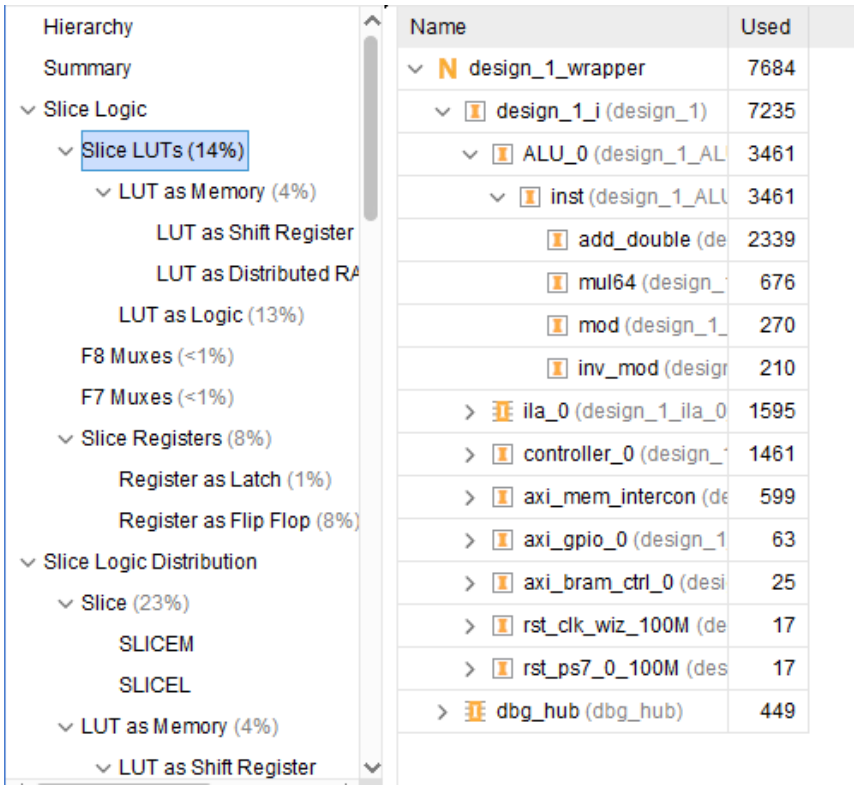
因此 n = 19, h = 1

- 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





硬體的資源每有預期的超高使用量，分析可能的原因為我們將硬體切的太細導致只有唯一一塊硬體可以用，但大幅增加硬體時間。應該適當調整硬體 resource share 的比例來達到最好的 time * area 值。