Lab2_Team32_Report

組別：Team32

組長：蘇勇誠（108062373）

組員：張晏瑄（108062273）

The Circuits of Basic Question 1

使用 NAND gates 來實作 AND, OR, NOR, XOR, XNOR, NOT，之後再用 8-to-1 的 MUX 串起來。

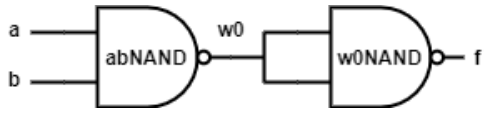
而 8-to-1 的 MUX 又可由 2 個 4-to-1 的 MUX 和 1 個 2-to-1 的 MUX 組成，4-to-1 的 MUX 則可用 3 個 2-to-1 的 MUX 組合而成。

使用 NAND 實作 AND, NOT, OR, NOR, XOR, XNOR

- AND

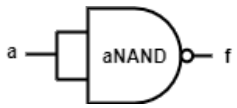
  $w0 = \overline{a \cdot b}$

  $f = \overline{w0 \cdot w0} = \overline{\overline{a \cdot b}} = a \cdot b$



- NOT

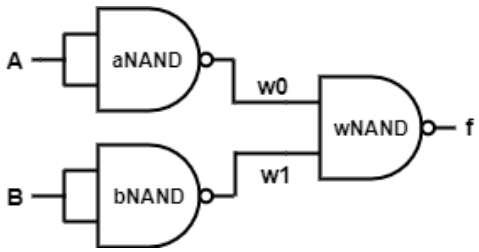
  $f = \overline{a \cdot a} = \overline{a}$



- OR

  $w0 = \overline{A}\ , w1 = \overline{B}$

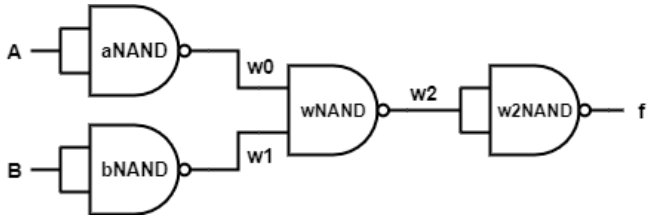
  $f = \overline{\overline{A} \cdot \overline{B}} = \overline{\overline{A}} + \overline{\overline{B}} = A + B$



- NOR

  $w0 = \overline{A \cdot A} = \overline{A}\ , w1 = \overline{B \cdot B} = \overline{B}\ , w2 = \overline{w0 \cdot w1}$

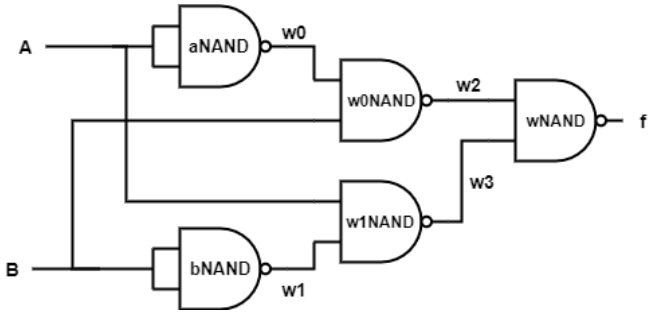
  $f = \overline{w2} = \overline{\overline{\overline{A} \cdot \overline{B}}} = \overline{\overline{\overline{A}} + \overline{\overline{B}}} = \overline{A + B}$



- XOR

$w0 = \overline{A}\,, w1 = \overline{B}\,, w2 = \overline{\overline{A} \cdot B}\,, w3 = \overline{\overline{B} \cdot A}$

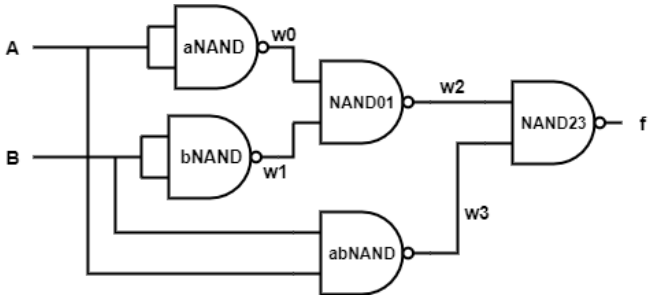
$f = \overline{w2 \cdot w3} = \overline{\overline{\overline{A} \cdot B} \cdot \overline{\overline{B} \cdot A}} = \overline{\overline{\overline{A} \cdot B}} + \overline{\overline{\overline{B} \cdot A}} = \overline{A} \cdot B + A \cdot \overline{B}$



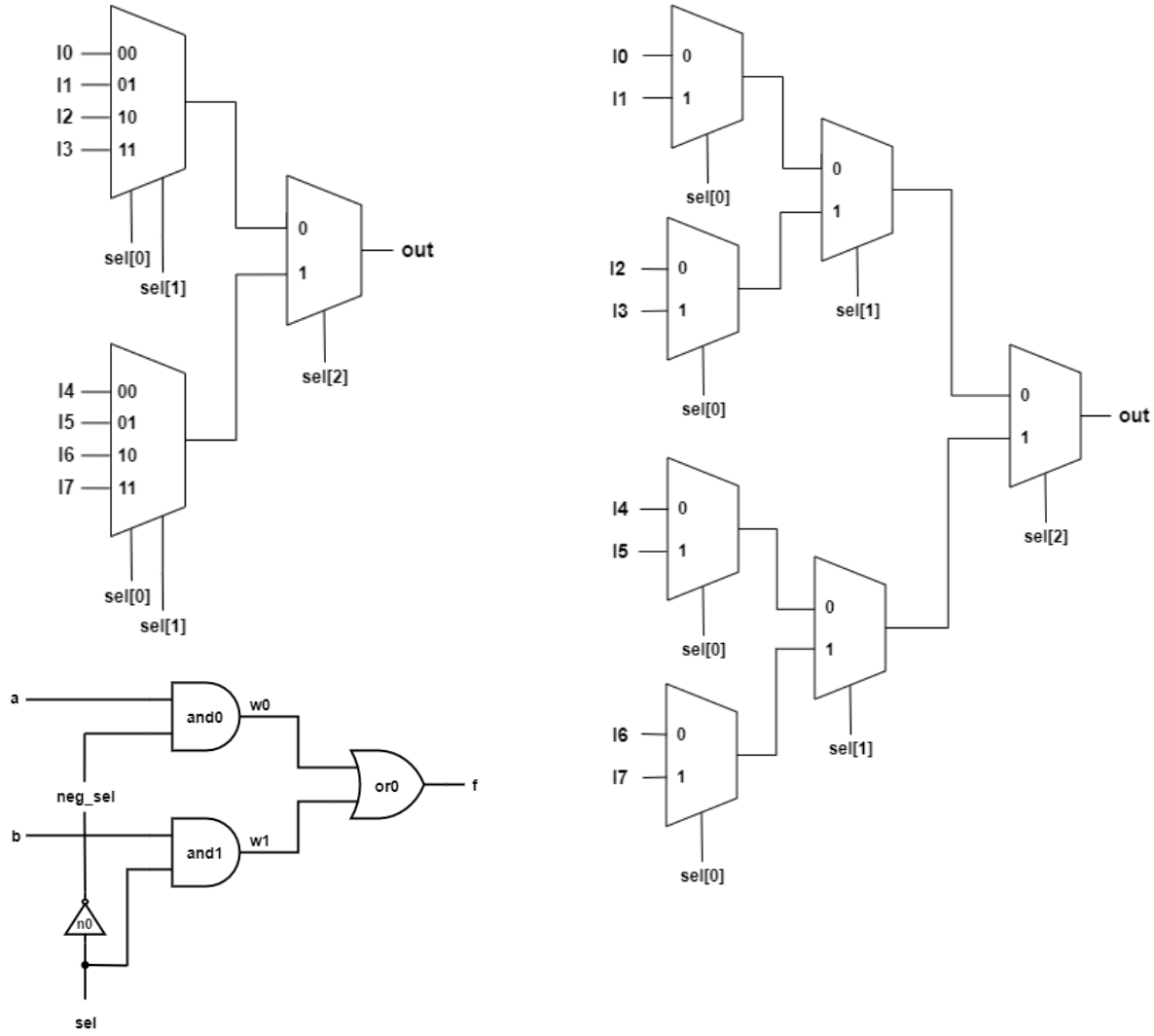
- XNOR

$w0 = \overline{A}\,, w1 = \overline{B}\,, w2 = \overline{\overline{A} \cdot \overline{B}}\,, w3 = \overline{A \cdot B}$

$f = \overline{w2 \cdot w3} = \overline{\overline{\overline{A} \cdot \overline{B}} \cdot \overline{A \cdot B}} = \overline{\overline{\overline{A} \cdot \overline{B}}} + \overline{\overline{A \cdot B}} = \overline{A} \cdot \overline{B} + A \cdot B$



8x1 MUX



The Difference between Basic Question 3 Adders

Full Adder 有三個輸入端、兩個輸出端，三個輸入分別為 A 與 B，A和B 兩 input 代表要相加的兩個 1bits訊號， 第三個輸入 Cin 代表前一個加法器的 Carry Out ，輸出 Sum 為三個 input 的和，而輸出 Cout 為進位（此加法器的Carry Out）。

Half Adder 有二個輸入端、兩個輸出端。 A 與 B 兩 input 代表要相加的兩個 1 bits訊號，輸出 Sum 為 A和B 之和，輸出 Cout 則為進位（此加法器的Carry Out）。

以下是兩加法器差別細項，分別比較 Truth Table、Boolean 函數、邏輯電路：

1. Truth Table

- Full Adder
- Half Adder

Cin	A	B	Cout	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

A	B	Cout	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

2. Boolean Function

- Full Adder

$$Cout = \overline{Cin}AB + Cin\overline{A}B + CinA\overline{B} + CinAB$$
$$= CinA + CinB + AB$$

$$Sum = \overline{Cin}\ \overline{A}B + \overline{Cin}A\overline{B} + Cin\overline{A}\ \overline{B} + CinAB$$
$$= Cin \bigoplus A \bigoplus B$$

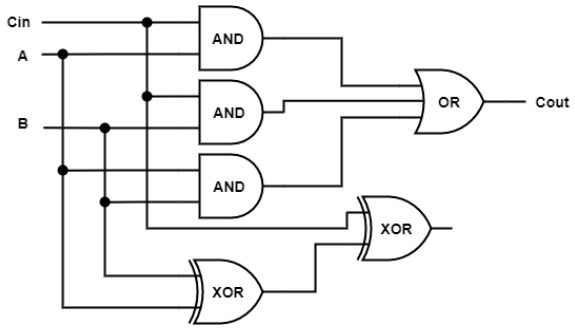
- Half Adder

$$Cout = A \cdot B$$

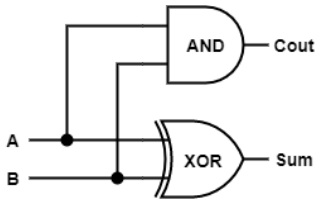
$$Sum = \overline{A}B + A\overline{B} = A \bigoplus B$$

3. Logic Circuit

- Full Adder



- Half Adder



8-bit Ripple-Carry Adder (RCA)

Design

Ripple carry adder 由 8 個 1-bit Full Adder 組成，故先設計 1-bit Full Adder，然後再把各個 `cin` `cout` 串起來。

Detail

1. 將 8 個 Full Adder 設計後，將 `cin` `cout` 串起來。

```
FullAdder fa0(.a(a[0]), .b(b[0]), .cin(cin), .sum(sum[0]), .cout(out[0]));
FullAdder fa1(.a(a[1]), .b(b[1]), .cin(out[0]), .sum(sum[1]), .cout(out[1]));
FullAdder fa2(.a(a[2]), .b(b[2]), .cin(out[1]), .sum(sum[2]), .cout(out[2]));
FullAdder fa3(.a(a[3]), .b(b[3]), .cin(out[2]), .sum(sum[3]), .cout(out[3]));
FullAdder fa4(.a(a[4]), .b(b[4]), .cin(out[3]), .sum(sum[4]), .cout(out[4]));
FullAdder fa5(.a(a[5]), .b(b[5]), .cin(out[4]), .sum(sum[5]), .cout(out[5]));
FullAdder fa6(.a(a[6]), .b(b[6]), .cin(out[5]), .sum(sum[6]), .cout(out[6]));
FullAdder fa7(.a(a[7]), .b(b[7]), .cin(out[6]), .sum(sum[7]), .cout(cout));
```

2. 1-bit Full Adder，使用 Basic Question 3 的 Full Adder

```
// sum
XOR xor0(.a(a), .b(b), .out(w0));
XOR xor1(.a(w0), .b(cin), .out(sum));

// cout
nand nand0(w1, a, b);
nand nand1(w2, a, cin);
nand nand2(w3, b, cin);
nand nand3(cout, w1, w2, w3);
```

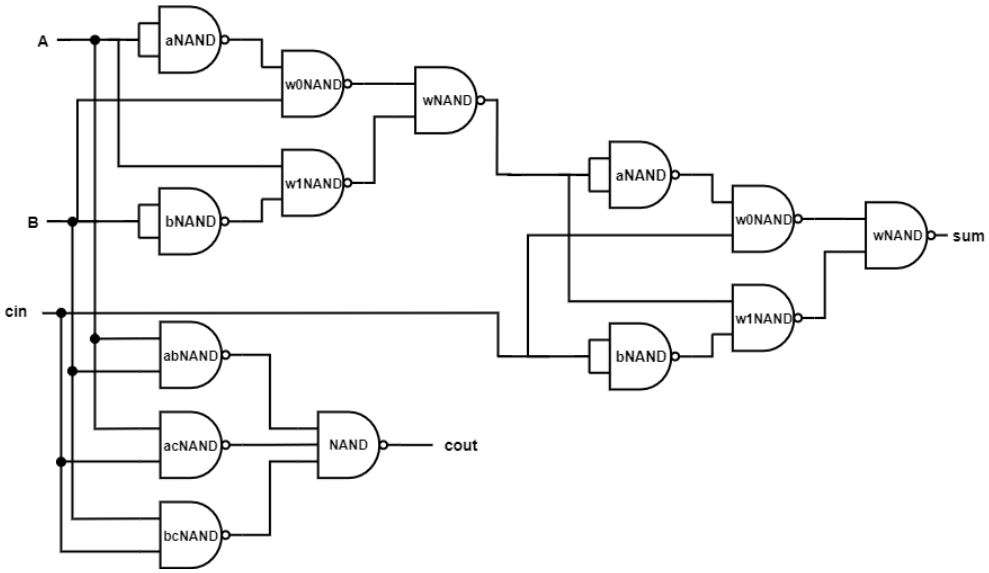
3. 使用 NAND Gates 實作 XOR

$$w0 = \overline{A}\,, w1 = \overline{B}\,, w2 = \overline{\overline{A} \cdot B}\,, w3 = \overline{\overline{B} \cdot A}$$

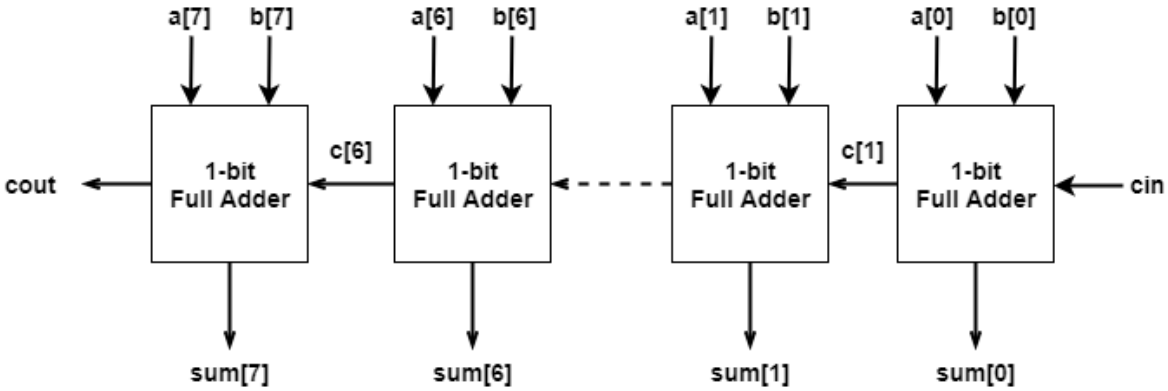
$$f = \overline{w2 \cdot w3} = \overline{\overline{\overline{A} \cdot B} \cdot \overline{\overline{B} \cdot A}} = \overline{\overline{A} \cdot B} + \overline{\overline{B} \cdot A} = \overline{A} \cdot B + A \cdot \overline{B}$$

```
nand nand0(w0, a, a), nand1(w1, b, b);
nand nand2(w2, w0, b), nand3(w3, a, w1);
nand nand4(out, w2, w3);
```

Gate Level Diagram



1-bit Full Adder using only NAND gates



8-bit Ripple Carry Adder (RCA)

Testbench

考慮 `cin = 0` 及 `cin = 1` 的情況，然後比對設計的 module 所算是否和電腦算的相等，若有不一致的情況 if 條件會滿足，就能看到錯誤發生在哪個位置

```
repeat (2 ** 16) begin
    #1 {a, b} = {a, b} + 1'b1;
  #1
  if(a + b + cin != {cout, sum})
    $display("error occurs at a=%b b=%b cin=%b cout=%b sum=%b", a, b, cin, cout,sum);
  end
```

Decode and Execute

Design

先利用 Universal Gate 設計 And、Or、Not、Nand、Nor、Xor、Xnor Gates。

再設計 Design Adder、Substractor、Bitwise-And、Bitwise-Or、Circular Left Shift、Arithmetic Right Shift、Compare Equality、Grater Than 的 module。

最後將上述8個 module 的 Output Port 連接到 8-to-1MUX，並利用 selection 訊號選擇欲使用的功能。

Detail

一、Universal Gate

Universal Gate的Boolean Function：f=a & (~b)

假設以下logic date的input均為in1和in2

1. not：a = 1, b = in ⇒ f = 1 & (~in) = ~in
2. and：a = in1, b = ~in2 ⇒ f = in1 & (1 & ~(1 & ~(~in2))) = in1 & in2
3. or：f = (1 and ~((1 & ~in1) & (~in2))) = ~((~in1) & (~in2)) = ~~(in1 | in2) = in1 | in2
4. nand：f = (1 & ~(in1 & (1 & ~(1 & ~(~in2))) = ~(in1 & in2)
5. nor：f = ~((1 & ~in1) & (~in2)) = (~in1) & (~in2) = ~(in1 | in2)
6. xor：(in1 & ~in2) + (~in1 & in2) (使用Universal_Gate組成的and、or、not實作)
7. xnor：(in1 & in2) + (~in1 & ~in2) (使用Universal_Gate組成的and、or、not 實作)

二、8個module

1. Adder：將Q1 design的8 bits Ripple Carry Adder改為4 bits Ripple Carry Adder。
2. Substractor：

   由於A-B=A+(2's complement of B)=A+(1's complement of B)+1'b1，所以我們先將B取1's complement，再利用4 bit Ripple Carry Adder將A、B的1's complement 、carry in = 1'b1相加，即可達到substractor的功能。
3. Bitwise And：利用verilog提供的語法，再搭配使用nand gate組合成的and gate即可達到Bitwise and的功能。
4. Bitwise Or：利用verilog提供的語法，再搭配使用nand gate組合成的or gate即可達到Bitwise and的功能。
5. RS Circular Left Shift：利用fan_out概念實作（詳細實作方式見Gate Level Diagram）
6. RT Arithmetic Right Shift：利用fan_out概念實作（詳細實作方式見Gate Level Diagram）

7. Compare Equality

   假設x和y為兩個1bit的數，若x和y相等，則 x xnor y == 1。

   假設A[3:0] B[3:0]，A == B的條件為：

   A[3] == B[3], A[2] == B[2], A[1] == B[1], A[0] == B[0]

   將以上條件寫成Boolean Function為：

$$f = (A[3] xnor B[3]) \cdot (A[2] xnor B[2]) \cdot (A[1] xnor B[1]) \cdot (A[0] xnor B[0])$$

8. Grater Than：

   假設A[3:0] B[3:0]，A>B的條件有以下4種case。

   Case 1: A[3] == 1, B[3] == 0

   Case 2: A[3] == B[3], A[2] == 1, B[2] == 0

   Case 3: A[3] == B[3], A[2] == B[2], A[1] == 1, B[1] == 0

   Case 4: A [3] == B[3], A[2] == B[2], A[1] == B[1], A[0] == 1, B[0] == 0

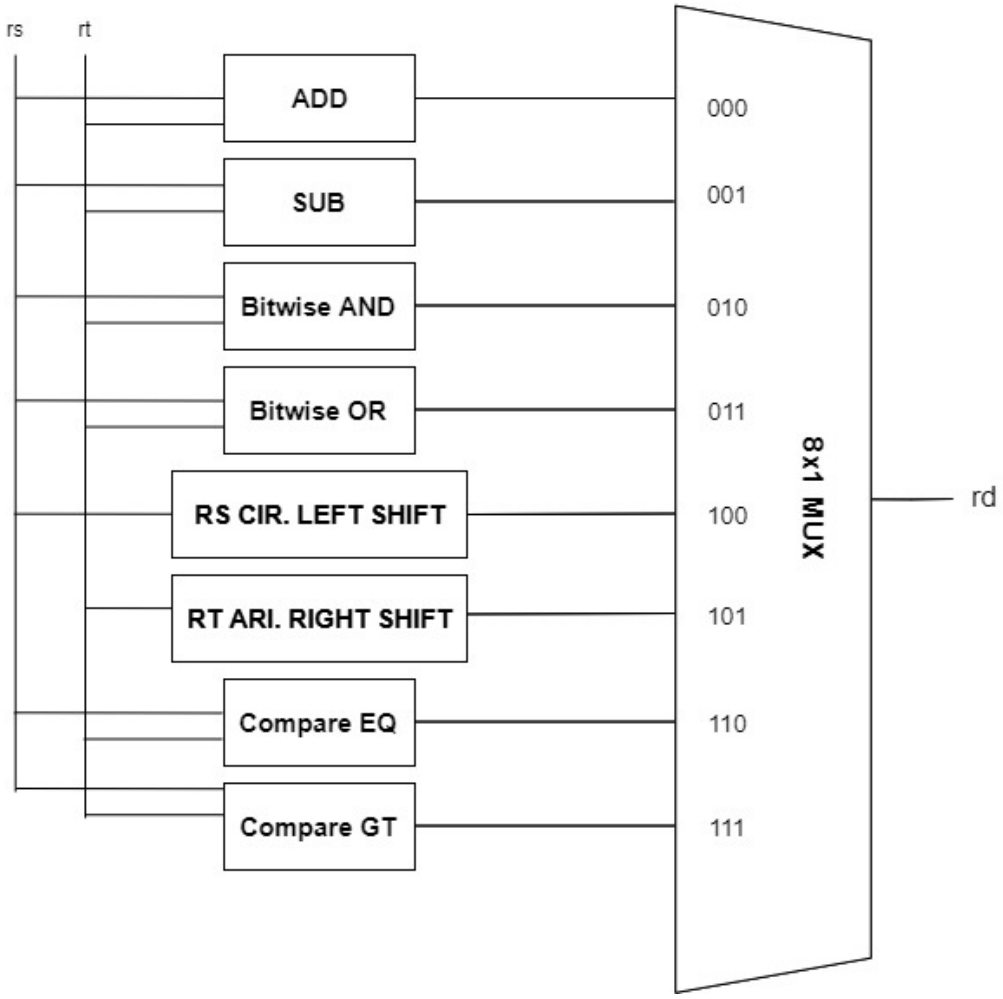
   將以上四種case寫成Boolean Function為

$$f = A[3] \cdot \overline{B[3]} + A[3] \cdot B[3] \cdot A[2] \cdot \overline{B[2]} + A[3] \cdot B[3] \cdot A[2] \cdot B[2] \cdot A[1] \cdot \overline{B[1]} + A[3] \cdot B[3$$

   再將上述boolean function轉為Gate Level

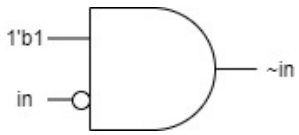
Gate Level Diagram

一、Top Module

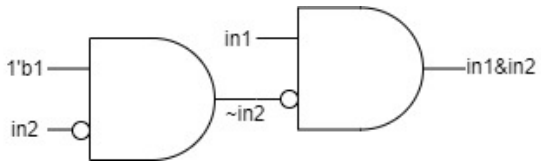


二、Universal Gate

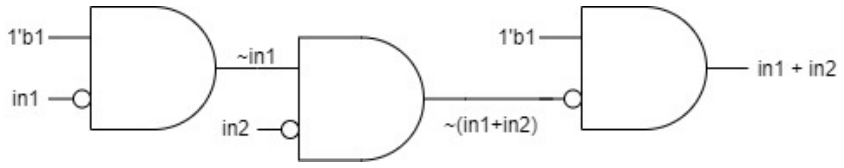
- not



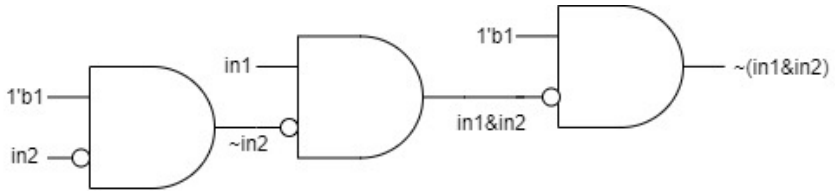
- and



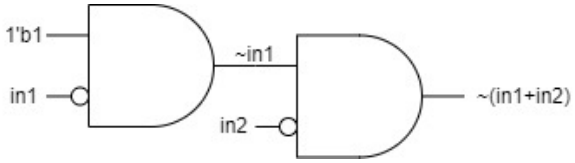
- or



- nand

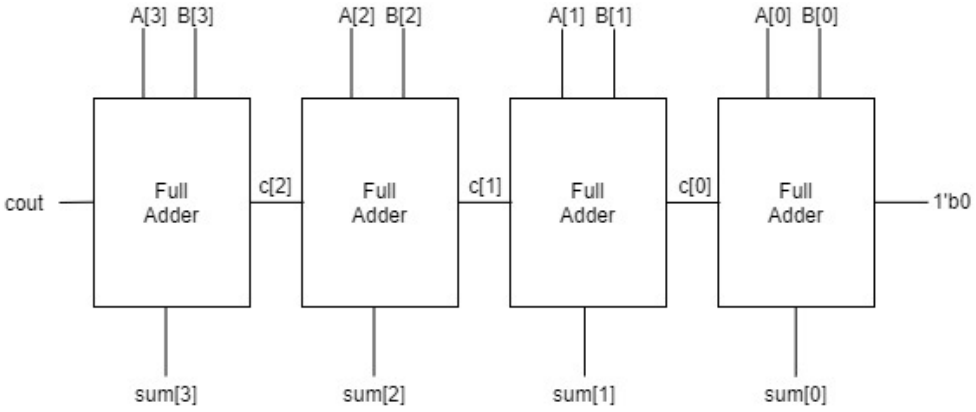


- nor

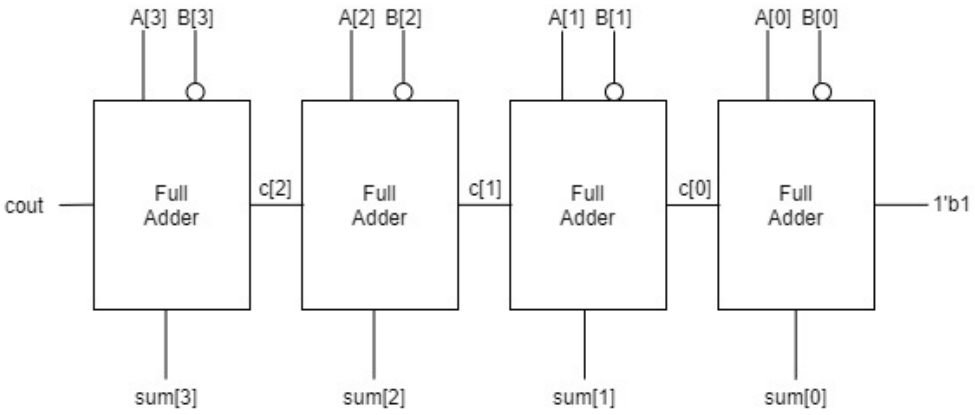


三、Module

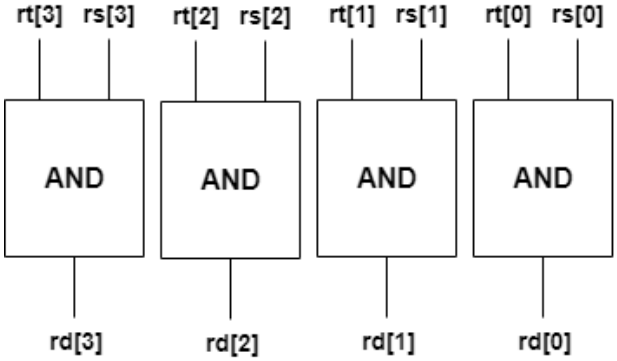
1. Adder：A + B == {cout, sum}



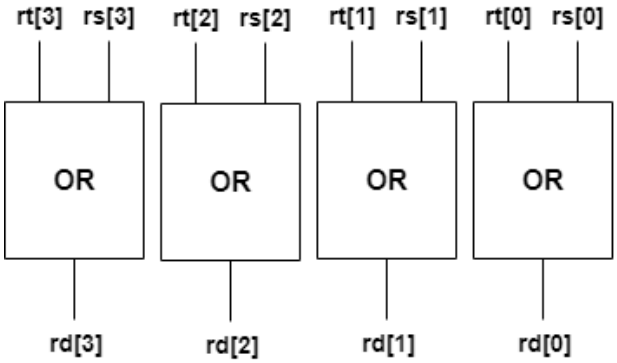
2. Substractor



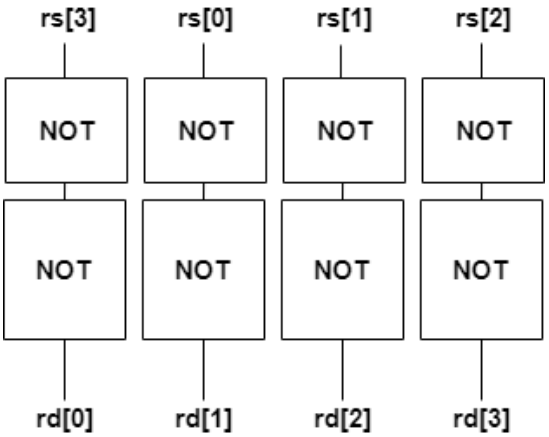
3. Bitwise And



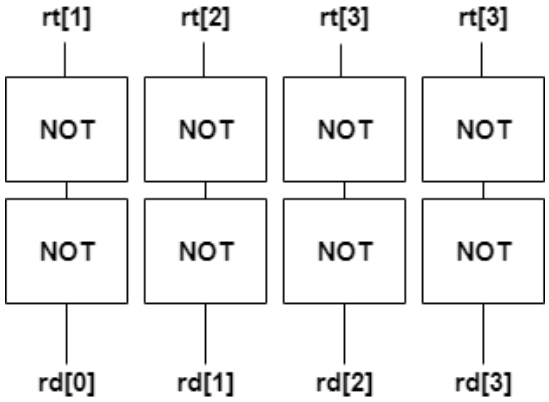
4. Bitwise Or



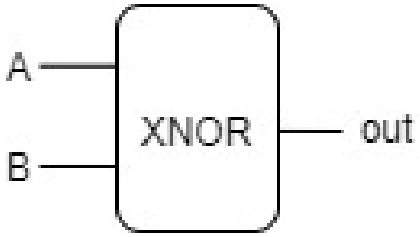
5. RS Circular Left Shift



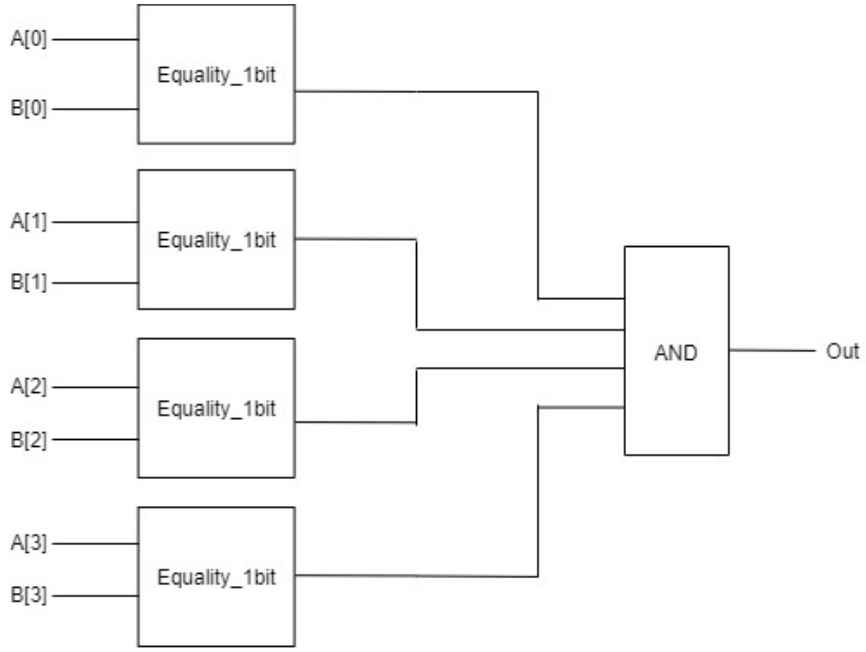
6. RT Arithmetic Right Shift



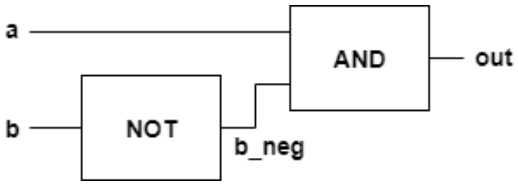
7. Compare Equality_1bit（out = 1 , if a == b）



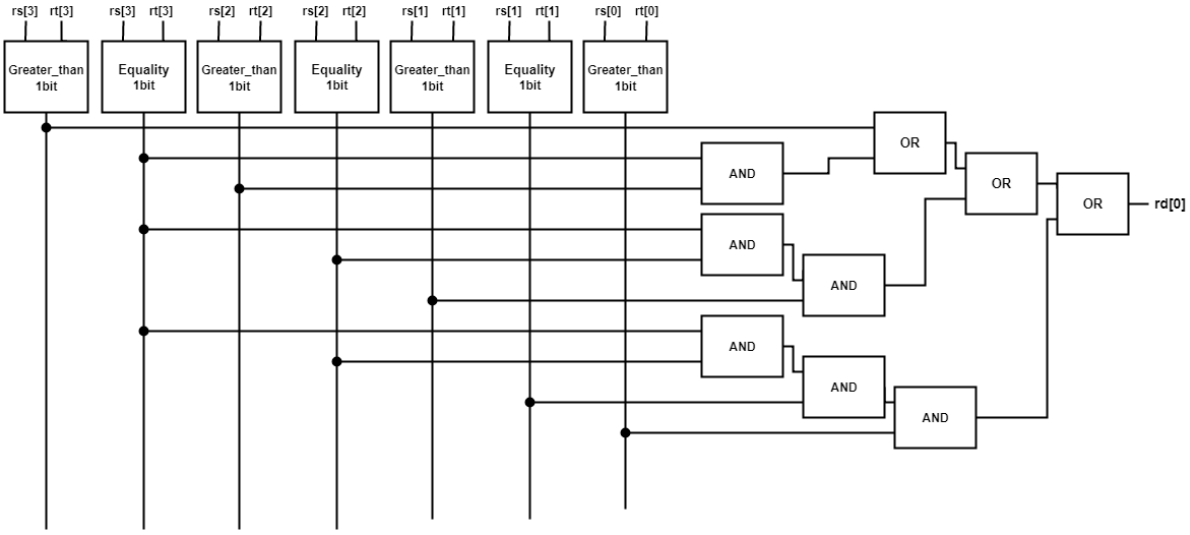
8. Compare Equality_4bits



9. Grater Than_1bit（out = 1 , if a > b)



10. Grater Than_4bits



Testbench

考慮所有可能的情況，然後比對設計的 module 所算是否和電腦算的相等，若有不一致的情況 if 條件會滿足，就使用display輸出error，即可快速debug。由於sel有3個bit且rs和rt共8個bit，故本題須考慮的情$8 * 2^8$種。

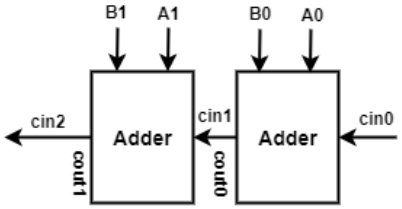
```
initial begin
    repeat(8)begin
        repeat(2 ** 8)begin
            #1 {rs, rt} = {rs, rt} + 4'b0001;
            #1
            //$display("%b %b %b",rs, rt, rd);
            case (sel)
                3'b000:
                    if(rs + rt != rd)begin
                        $display("error occurs at add rs=%b rt=%b rd=%b", rs, rt, rd);
                    end
                3'b001:
                    if(rs - rt != rd)begin
                        $display("error occurs at sub rs=%b rt=%b rd=%b", rs, rt, rd);
                    end
                3'b010:
                    if((rs & rt) != rd)begin
                        $display("error occurs at bitwise_and");
                        $display("rs=%b rt=%b rd=%b rs&rd=%b", rs, rt, rd, rs&rd);
                    end
                3'b011:
                    if((rs | rt) != rd)begin
                        $display("error occurs at bitwise_or");
                        $display("rs=%b rt=%b rd=%b rs|rd=%b", rs, rt, rd, rs|rd);
                    end
                3'b100:
                    if(rd != {rs[2:0], rs[3]})begin
                        $display("error occurs at rs L_shift rs=%b rd=%b", rs, rd);
                    end
                3'b101:
                    if(rd != {rt[3], rt[3:1]})begin
                        $display("error occurs at rt R_shift rt=%b rd=%b", rt, rd);
                    end
                3'b110:
                    if(rd != {3'b111, rs == rt})begin
                        $display("error occurs at equal rs=%b rt=%b rd=%b", rs, rt, rd);
                    end
                3'b111:
                    if(rd != {3'b101, rs > rt})begin
                        $display("error occurs at grater_than rs=%b rt=%b rd=%b", rs, rt, rd);
                    end
            endcase
        end
        sel = sel + 3'b001;
    end
    #1 $finish;
end
```

8-bit Carry-Lookahead Adder (CLA)

Design

要設計 8-bit 的 Carry Lookahead Adder (CLA)，需先設計三個 modules。1-bit Full Adder, 4-bit CLA, 2-bit CLA。因 CLA 的特性為 Adder 的 cin 並不會太依賴前一個 Adder 的 cout，所以須藉由 p (propagate), g (generate) 得知 cin 之值。

Propagate, Generate



$Cout = (A * B) + (A * Cin) + (B * Cin)$

- $cin_2 = cout_1 = (A_1 * B_1) + (A_1 * cin_1) + (B_1 * cin_1)$
- $cin_1 = cout_0 = (A_0 * B_0) + (A_0 * cin_0) + (B_0 * cin_0)$

將 cin1 帶入 cin2 並作化簡

$cin_2 = (A_1 * B_1) + (A_1 * A_0 * B_0) + (A_1 * A_0 * cin_0) + (A_1 * B_0 * cin_0) + (B_1 * A_0 * B_0) + (B_1 * A_0 * cin_0) + (B_1 * B_0 * cin_0)$
$= (A_1 * B_1) + (A_1 + B_1) * (A_0 * B_0) + (A_1 + B_1) * (A_0 + b_0) * cin_0$

經過化簡後的式子有以下特性：有 AB 做 AND 的運算、有 AB 做 OR 的運算、還有一個方程式跟 cin 會有關係。

- 定義 p, g

  $g_i = A_i * B_i,\ p_i = A_i + B_i$

  Generate (g): 某一個 bit i，若 A 和 B 皆為 1 的話，可保證一定會產生 cout

  Propagate (p): 當 p_i 為 1 時，若第 i 個 bit 的 cin 為 1，再加上 A_i 和 B_i 其中至少有個為 1，則也會產生 cout。而因 cin 為 1，且有將此 cin 再作為 cout 往下級傳遞出去，因此叫做 propagate 傳遞

由 p, g 定義我們可以得到

$cin_1 = g_0 + (p_0 * cin_0)$
$cin_2 = g_1 + (p_1 * g_0) + (p_1 * p_0 * cin_0)$
$cin_3 = g_2 + (p_2 * g_1) + (p_2 * p_1 * g_0) + (p_2 * p_1 * p_0 * cin_0)$
$cin_4 = g_3 + (p_3 * g_2) + (p_3 * p_2 * g_1) + (p_3 * p_2 * p_1 * g_0) + (p_3 * p_2 * p_1 * p_0 * cin_0)$

Detail

因隨著 bit 數越多，要直接得出下一級的 cin 的式子也會越來越長，因此若直接設計第 8 個 bit 的 cin，效能可能會變得很糟糕，進而失去 CLA 的優點，因此我們將 8-bit CLA 拆成兩個 4-bit CLA 及一個 2-bit CLA，第一個 4-bit CLA 產生其最高位元的 p, g 後，可傳入 2-bit CLA，進而得出下個位元的 cin。接著再將結果傳入第二個 4-bit CLA，再按照第一個 CLA 的做法，算出最後的 cout。

- 1-bit Full Adder

  將之前實作過的 Full Adder 改造，XOR 則用 NAND Gates 實作，輸出 sum_i, p_i, g_i。與之前 Full Adder 差別為不需要算出 cout 為何，因 cout 之後可由 p, g 得知

  $$sum_i = A_i \bigoplus B_i \bigoplus cin_i$$
  $$p_i = A_i + B_i$$
  $$g_i = A_i * B_i$$

- 4-bit Carry Look Ahead

  於 4-bit CLA 內平行算出各個 Adder 的傳至下一個 adder 的 cin 為何，於上方已有詳細說明推導過程

  $$cin_1 = g_0 + (p_0 * cin_0)$$
  $$cin_2 = g_1 + (p_1 * g_0) + (p_1 * p_0 * cin_0)$$
  $$cin_3 = g_2 + (p_2 * g_1) + (p_2 * p_1 * g_0) + (p_2 * p_1 * p_0 * cin_0)$$
  $$cin_4 = g_3 + (p_3 * g_2) + (p_3 * p_2 * g_1) + (p_3 * p_2 * p_1 * g_0) + (p_3 * p_2 * p_1 * p_0 * cin_0)$$

- 2-bit Carry Look Ahead

  先定義 PG (propagate group) 及 GG (generate group)：

  $$PG = P_0 \cdot P_1 \cdot P_2 \cdot P_3$$
  $$GG = G_3 + G_2 \cdot P_3 + G_1 \cdot P_3 \cdot P_2 + G_0 \cdot P_3 \cdot P_2 \cdot P_1$$

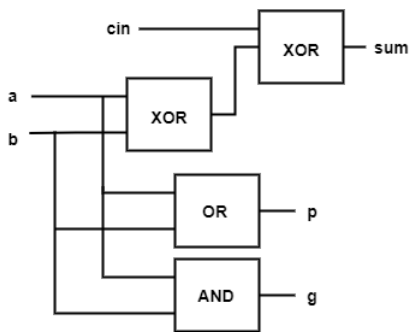
  PG, GG 可為該特定四位元組產生 cout，由下列式子可以看出 CG 與 cin_4 相等：

  $$CG = GG + (PG \cdot cin)$$

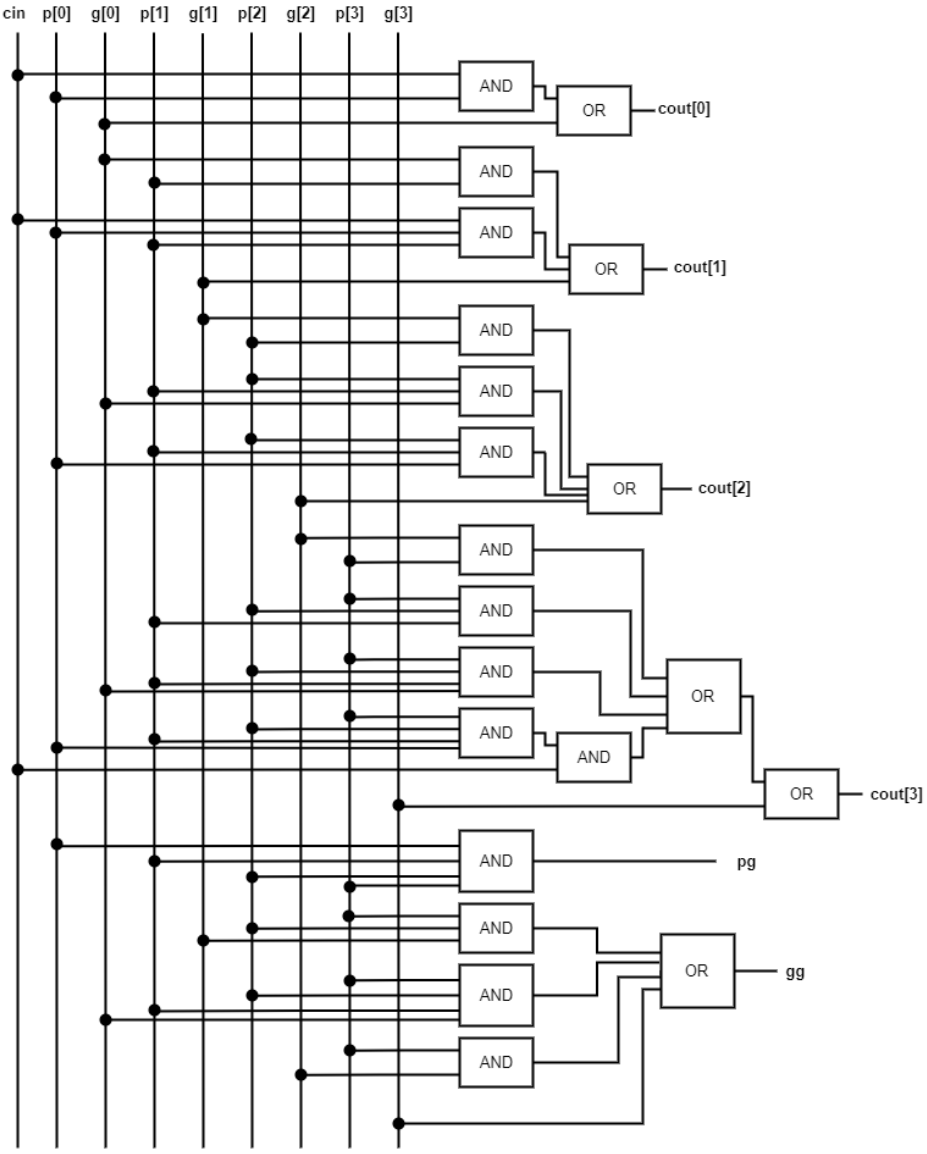
  故可以藉由傳入兩個 4-bit CLA 的 PG, GG 來產生 cin_4 及 cin_8

Gate Level Diagram

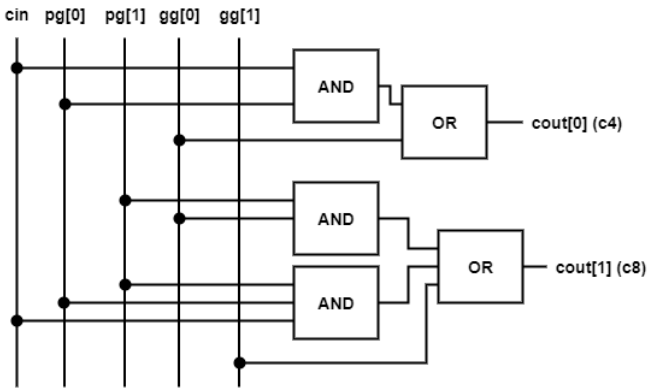
- 1-bit Full Adder



- 4-bit Carry Look Ahead



- 2-bit Carry Look Ahead



Testbench

如同 RCA 的 Testbench 考慮 `cin = 0` 及 `cin = 1` 的情況，然後比對設計的 module 所算是否和電腦算的相等，若有不一致的情況 if 條件會滿足，即可看到錯誤發生在哪個位置

```
repeat (2 ** 16) begin
    #1 {a, b} = {a, b} + 1'b1;
  #1
  if(a + b + cin != {cout, sum})
    $display("error occurs at a=%b b=%b cin=%b cout=%b sum=%b", a, b, cin, cout, sum);
  end
```

The Benefits of CLA

- Ripple Carry Adder (RCA) 的問題

  因 RCA 是由多個 FullAdder 串接而成，而每個 FullAdder 的 cin 都需要等前一個的 cout 傳進來，故要實施高位元的加法時，整個運算時間將會拉長。

- 解決：Carry-Lookahead Adder (CLA)

  CLA 讓每個 bit 彼此之間 carry dependence 能夠移除，每個 Adder 要做加法時，都不需要等待前一個 Adder 做完其運算，也就是減少了 propagation delay ，每個 Adder 可以直接知道 carry bits 然後做運算

4-bit Multiplier

Design

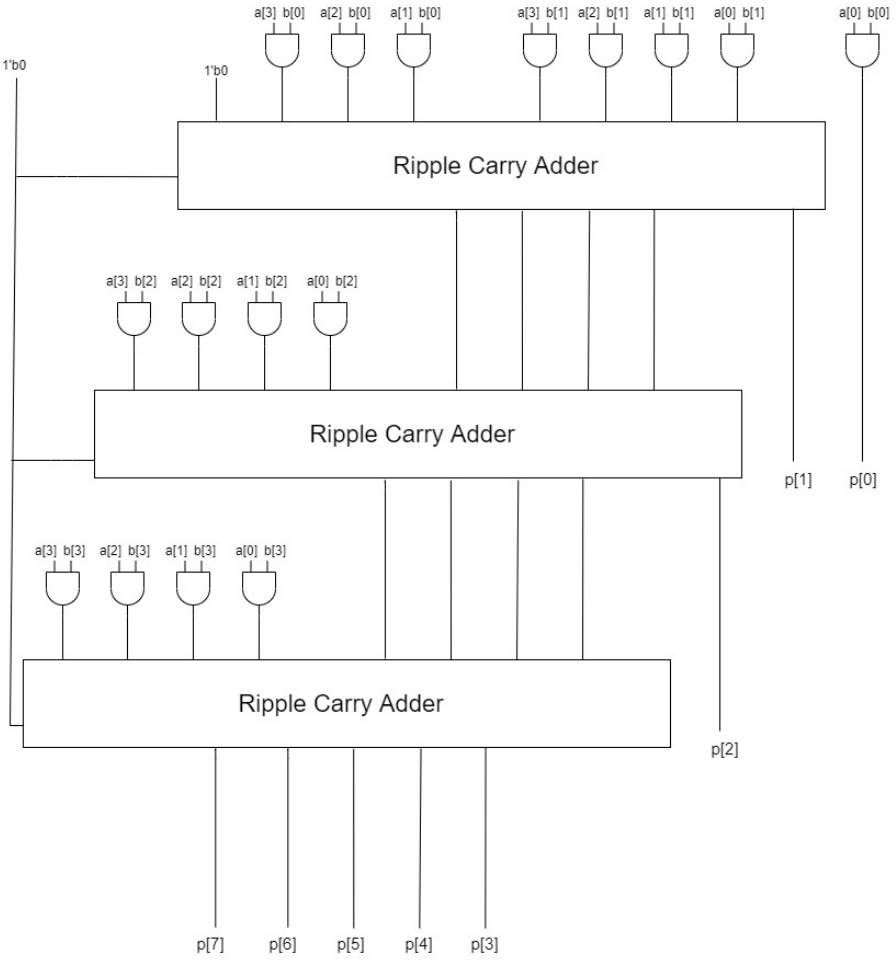
利用直式乘法，將兩個 4 bits 的數相乘，再將每個bit相乘後的結果相加。

Detail

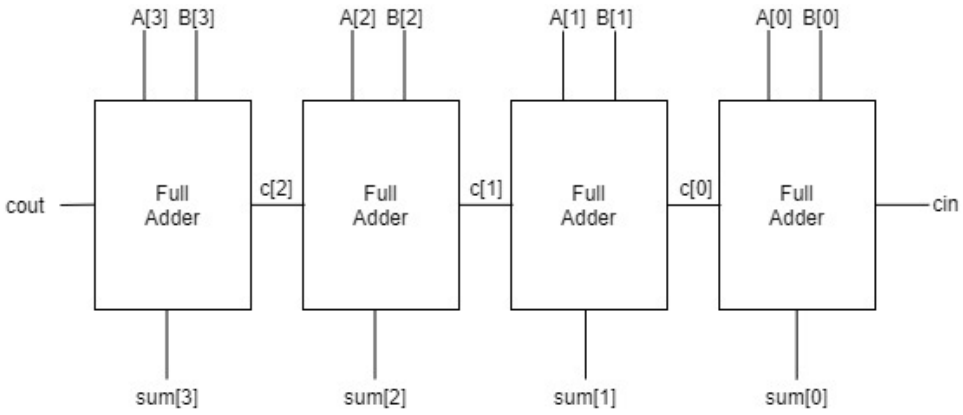
1. Logic Gate：沿用Basic Q1 利用nand gate design的各種Logic Gate（AND、OR、NOR、NOT、NAND、XOR、XNOR）。
2. 直式乘法中的兩個1bit數相乘，可利用 and Gate 實作。
3. 最後，將所有相乘後的結果相加，則可用Q1的Ripple Carry Adder實作。

Gate Level Diagram

1. Multiplier架構圖（以下And Gate均為示意圖，module中的AND Gate均以Nand Gate實作）



2. Ripple Carry Adder



Testbench

考慮所有可能的情況，然後比對設計的 module 所算是否和電腦算的相等，若有不一致的情況 if 條件會滿足，就使用display輸出error。

```
initial begin
    repeat (2 ** 8) begin
        #1 {a, b} = {a, b} + 4'b0001;
        #1
        tmp = a * b;
        if(tmp !== p)
            $display("error occurs at sub a=%d b=%d p=%d a*b=%d", a, b, p, a*b);
    end
    #1 $finish;
end
```

Exhausted Testbench

Design

利用 Verilog 中的 loop 窮舉所有數值，然後比對設計的 module 所算是否和電腦算的相等，若有不一致的情況 if 條件會滿足，就將error訊號設為1'b1。由於cin有2種可能且a和b一共8個bit，故本題須考慮的情形有 $2 * 2^8$ 種。最後，當窮舉完所有可能發生的數值後，就將done訊號設定為 1'b1。

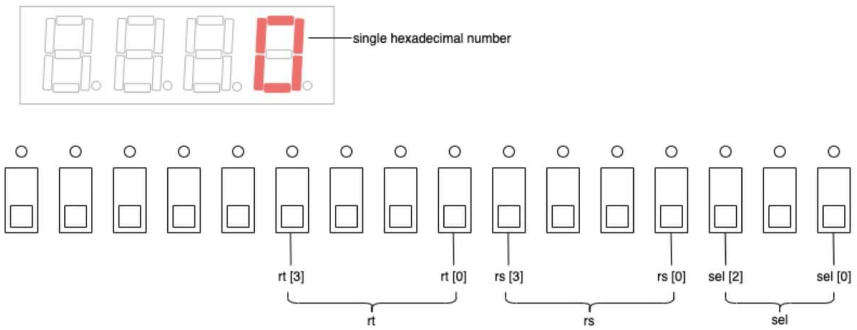
Detail

```
initial begin
  repeat(2 ** 8) begin
```

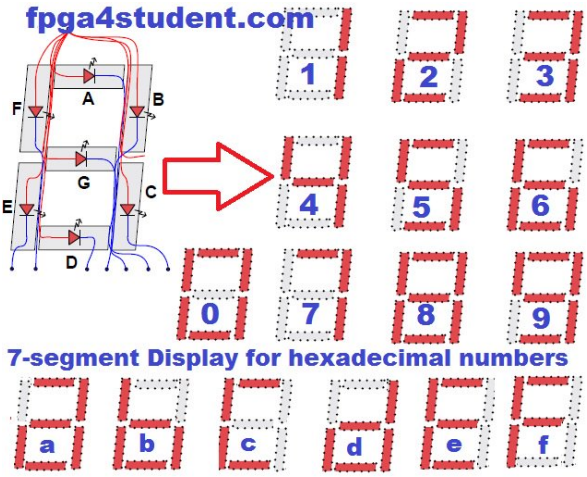
```
        #1
        if(a + b + cin != {cout, sum}) begin
           error = 1'b1;
        end
        else begin
            error = 1'b0;
        end
        #4
        {a, b} = {a, b} + 4'b0001;
    end
    #5 done = 1'b1;
    #5 $finish;
end
```

Decode and Execute FPGA

Design



依據簡報， `SW[2:0]` 表 `sel`，`SW[6:3]` 表 `rs`，`SW[10:7]` 表 `rt` ，對應於 FPGA 的 pin 如上圖所示，決定要做的運算及 input 後即可開啟對應開關，接著經過 Decode and Execute 完後會 `rd` 會被賦予值，根據 `rd` 得到的值來設計 7-segment display，0 表示亮燈，1 表示不亮燈，而 DP (dot) 都要設成不亮燈

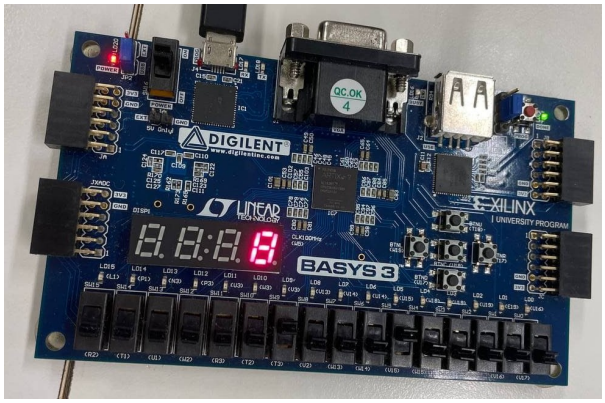


其中，因若 B 和 D 為大寫，則分別會和 8 和 0 重複，故改為小寫

Detail

```
module FPGA_Display (SW, control, seg);
input [10:0] SW;
output [3:0] control;
output [7:0] seg;

wire [3:0] rd;
reg [7:0] seg;

assign control = 4'b1110;

Decode_And_Execute d0(SW[6:3], SW[10:7], SW[2:0], rd);

always @(*) begin

    case (rd)
        4'h0: seg = 8'b00000011;
        4'h1: seg = 8'b10011111;
        4'h2: seg = 8'b00100101;
        4'h3: seg = 8'b00001101;
        4'h4: seg = 8'b10011001;
        4'h5: seg = 8'b01001001;
        4'h6: seg = 8'b01000001;
        4'h7: seg = 8'b00011111;
        4'h8: seg = 8'b00000001;
        4'h9: seg = 8'b00001001;
        4'hA: seg = 8'b00010001;
        4'hB: seg = 8'b11000001;
        4'hC: seg = 8'b01100011;
        4'hD: seg = 8'b10000101;
        4'hE: seg = 8'b01100001;
        default : seg = 8'b01110001;
    endcase

end

endmodule
```

- 測試

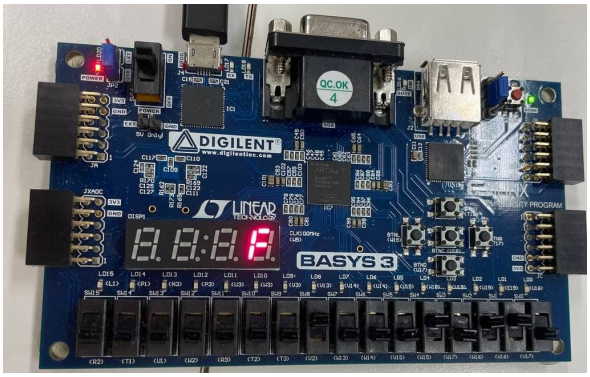


OP_code: `000` → `ADD`

rs: `0100`

rt: `0100`

rd: `8`



OP_code: `110` → `COMPARE EQ`

rs: `0010`

rt: `0010`

rd: `F`

Learning

- 取名的重要性，且助教於課堂中有特別提醒此問題，跟組員有商量一個取 gates 及 wire 的標準
- 熟悉用 Universal Gate 去 implement 各個邏輯閘 (NOT, NOR, AND, OR, XOR, XNOR)
- 瞭解全/半加法器的差異及 RCA 的缺點及改良方式 → CLA
- 由於這次 module 更加複雜，再次體會到 module 設計的重要性

分工

- 蘇勇誠（108062373）

  Decode and Execute

  8-bit Carry-Lookahead Adder (CLA)

  4-bit Multiplier

  Exhausted Testbench

  Decode and Execute FPGA

- 張晏瑄（108062273）

  The Circuits of Basic Question 1

  The Difference between Basic Question 3 Adder

  8-bit Ripple-Carry Adder (RCA)

8-bit Carry-Lookahead Adder (CLA)