10910EECS207001 Logic Design Lab

Lab 2: Advanced Gate-Level Verilog

TEAM 5

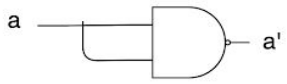
組長：108062211 黃子瑋

組員：108062213 顏浩昀

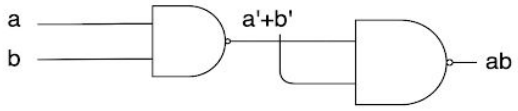
2020.10.08

Basic Question 1 (NAND_impl)

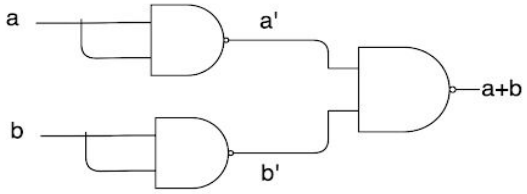
NAND_NOT



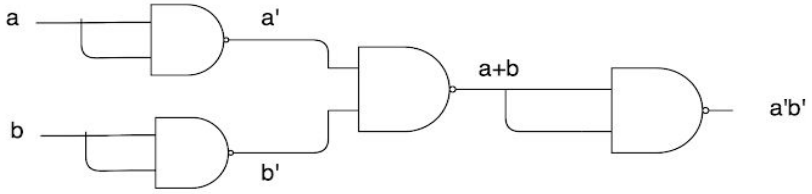
NAND_AND



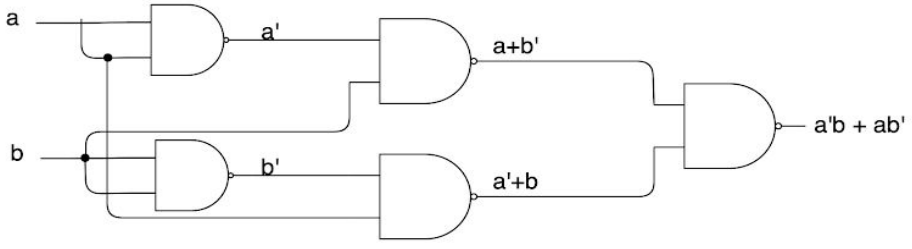
NAND_OR



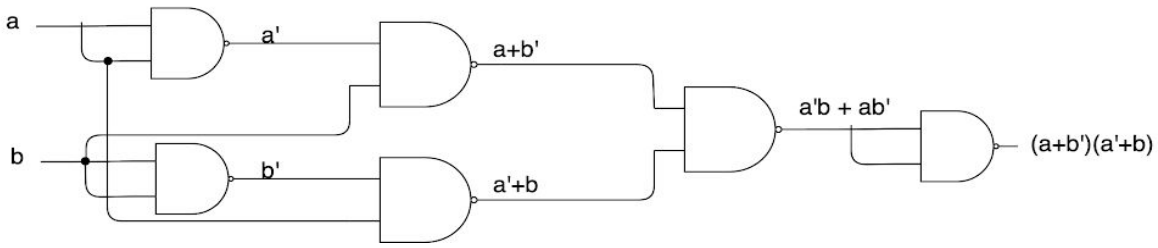
NAND_NOR



NAND_XOR

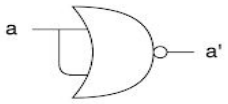


NAND_XNOR

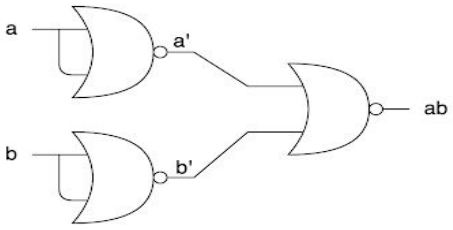


Basic Question 2 (NOR_impl)

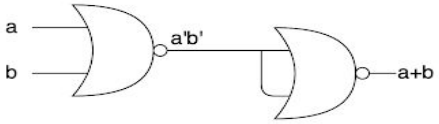
NOR_NOT



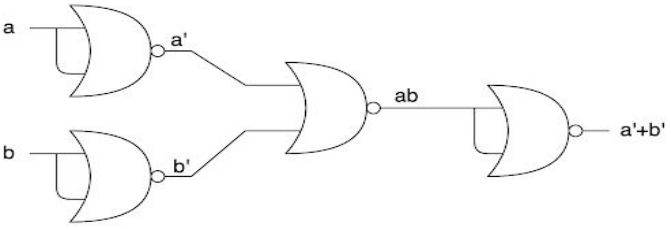
NOR_AND



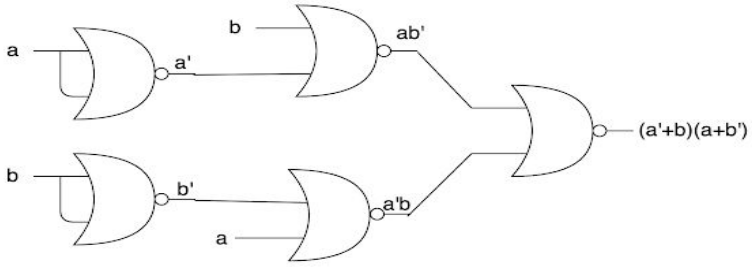
NOR_OR



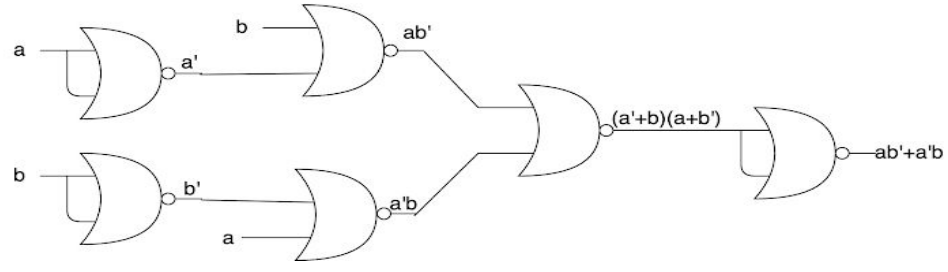
NOR_NAND



NOR_XNOR



NOR_XOR



Binary Code to Grey Code

I. Logic Design Diagram

(i) K-map

a. dout[0] = din[1] ⊕ din[0]

	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

b. dout[1] = din[2] ⊕ din[1]

	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	1	1	0	0
10	0	0	1	1

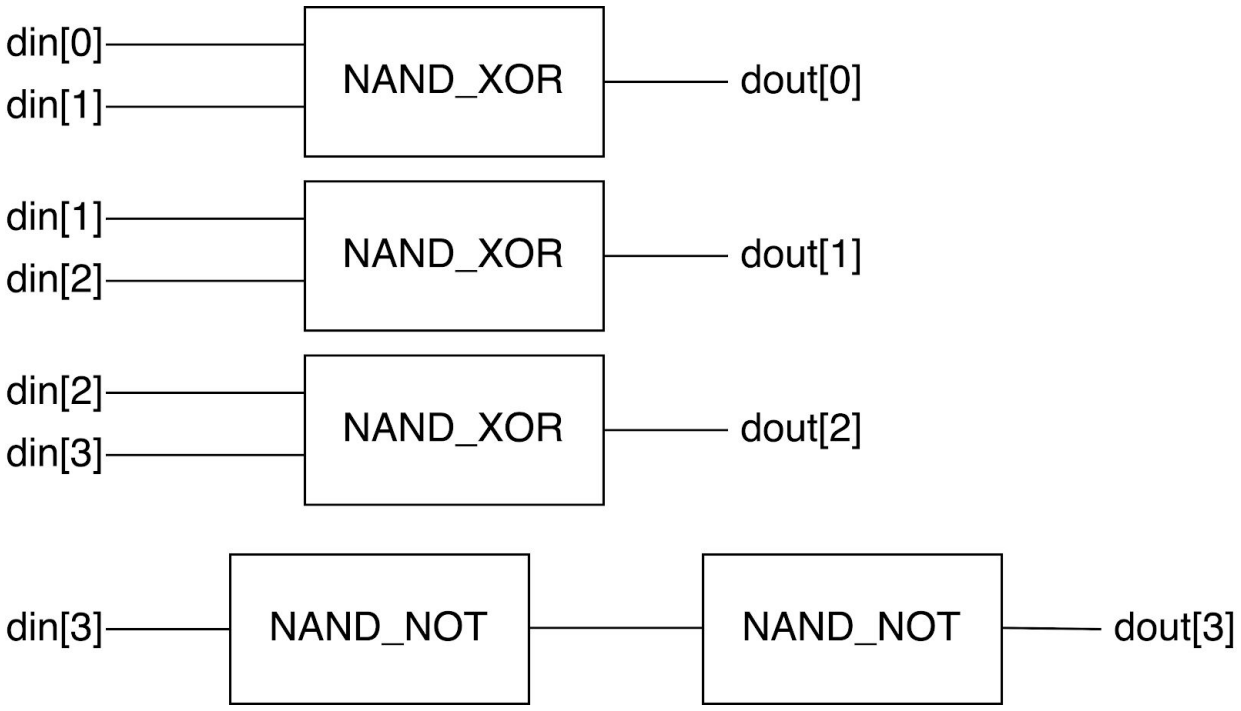
c. dout[2] = din[3] ⊕ din[2]

	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

d. dout[3] = din[3]

	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

(ii) 完整電路設計圖



II. Explaination

我們使用K-map找出Grey Code(dout)與Binary Code(din)各個bit的關係，並利用Basic Question做的NAND_implement，將各關係透過一系列的nand gate完成。

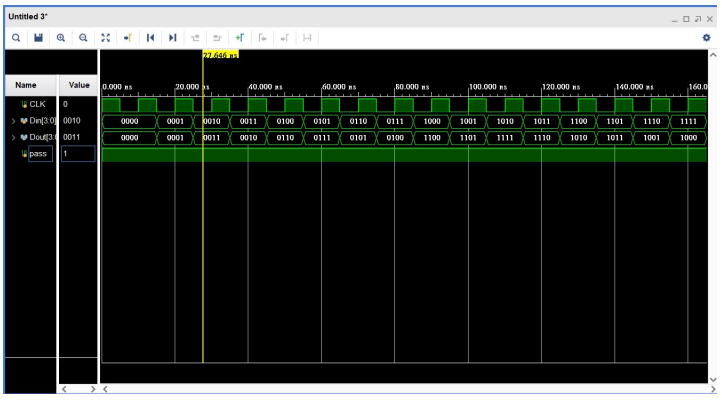
III. Testbench

窮舉所有的Binary code，將通過module得出的dout，與題目所述的Grey code相比較，若有不同則write error messenge。此外，我們在testbench當中設計一條pass，將其初始為1。如果在某一筆測資中有出現錯誤使pass = 0。因此，如果在waveform當中，pass一直維持於1，則代表所有轉換都是正確的；反之，若出現pass為0，代表轉換有誤。這樣可以快速的確認是否有任何bug出現。

```
initial begin
    Din = 4'b0;
    repeat (2**4) begin
        @(posedge CLK)
            Test;
        @(negedge CLK)
            Din = Din + 1'b1;
    end
    $finish;
end
```

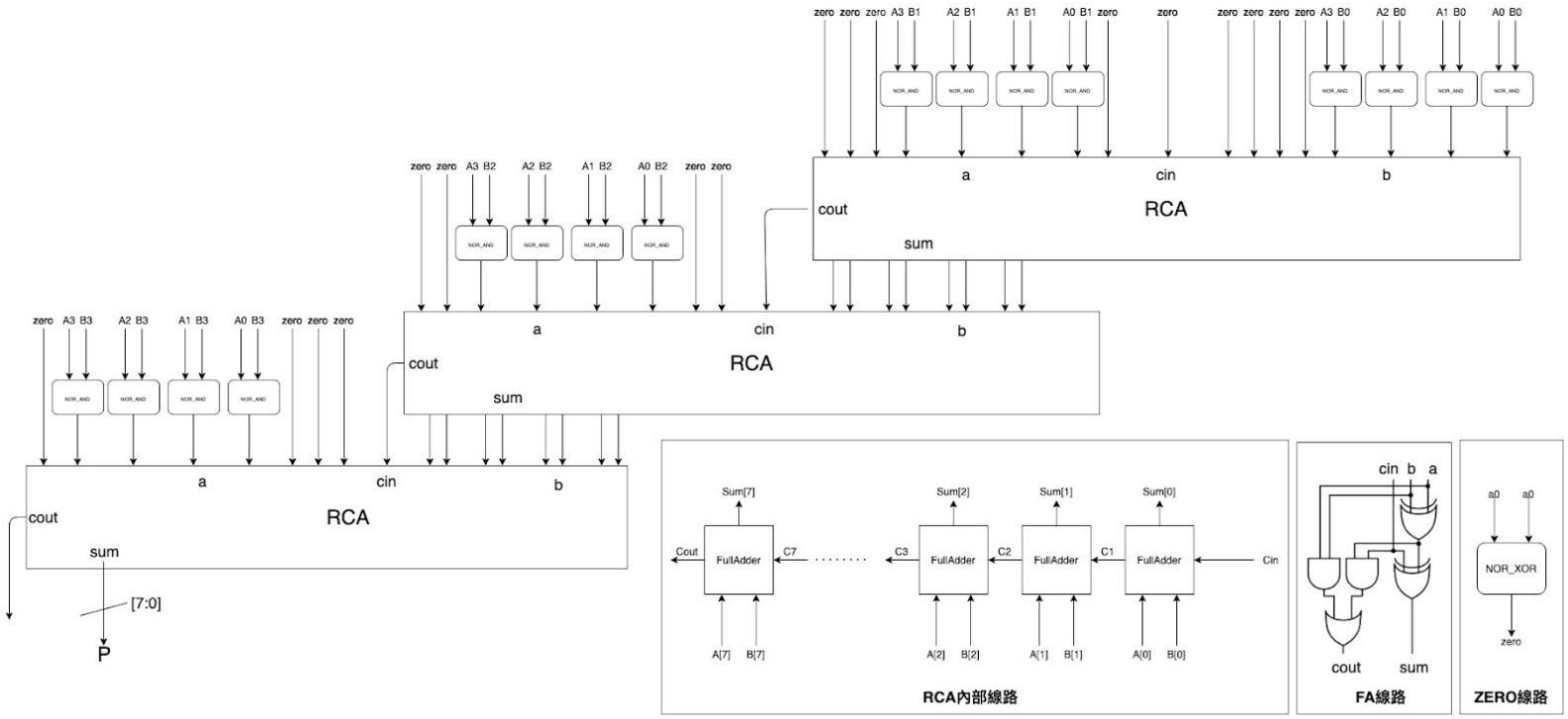
```
task Test;
begin

    if(Din === 4'b0000) begin
        if(Dout !== 4'b0000) begin
            Wrong;
        end
    end
```

```
task Wrong;
begin
    pass =0;
    $display("[ERROR]");
    $write("Binary:%4b",Din);
    $write("Grey:%4b",Dout);
    $display;
end
endtask
```



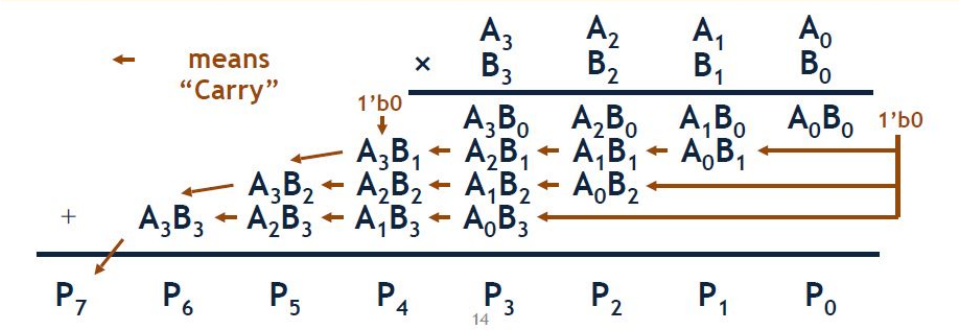
Multiplier

I. Logic Design Diagram



II. Explaination

概念是將乘法以AND和加法的方式呈現，先將B的每一個bit和A做AND，得到四個數字，再將數字依照位數做左移，再補到八位數，其他位數則以補0的方式實現，例如B1的結果為 {**zero,zero,zero,A3B1,A2B1,A1B1,A0B1,zero**}，B2的結果為 {**zero,zero,A3B2,A2B2,A1B2,A0B2,zero,zero**}，以此類推，得到四個8bit數字，再將之以RCA銜接起來，最後做三次加法即得出乘法的結果。



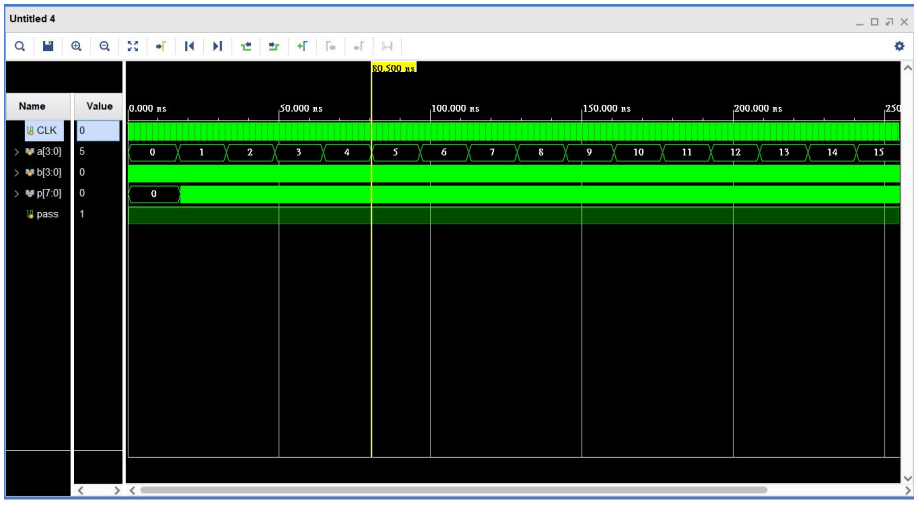
III. Testbench

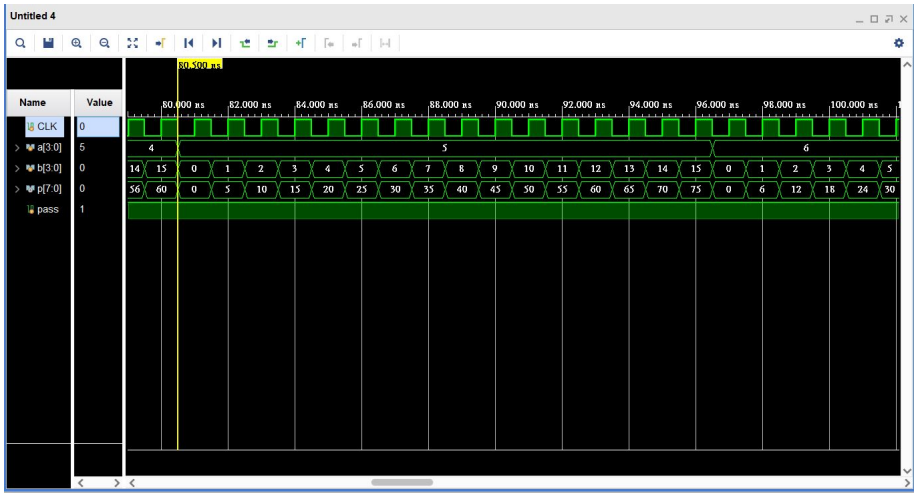
窮舉所有的a,b，並且將通過module的乘法結果p與a*b做比較，若有不同則write error messenge，並將pass改為0。

```
initial begin
    {a, b} = 8'b0;
    repeat(2**8)begin
    @ (posedge CLK)
      Test;
    @ (negedge CLK)
      {a, b} ={a, b} + 1'b1;
    end
    $finish;
end
```

```
task Test;
begin

  if(p !== (a*b))begin
    pass=0;
    $display("[ERROR]");
    $write("A:%d",a);
    $write("B:%d",b);
    $write("A * B = %d",a*b);
    $write("multiplier_answer:%d",p);
    $display;
  end
  else begin
    pass =1;
  end
end
endtask
```

下方兩張waveform可見此次pass皆為1，並擷取部分測資結果。

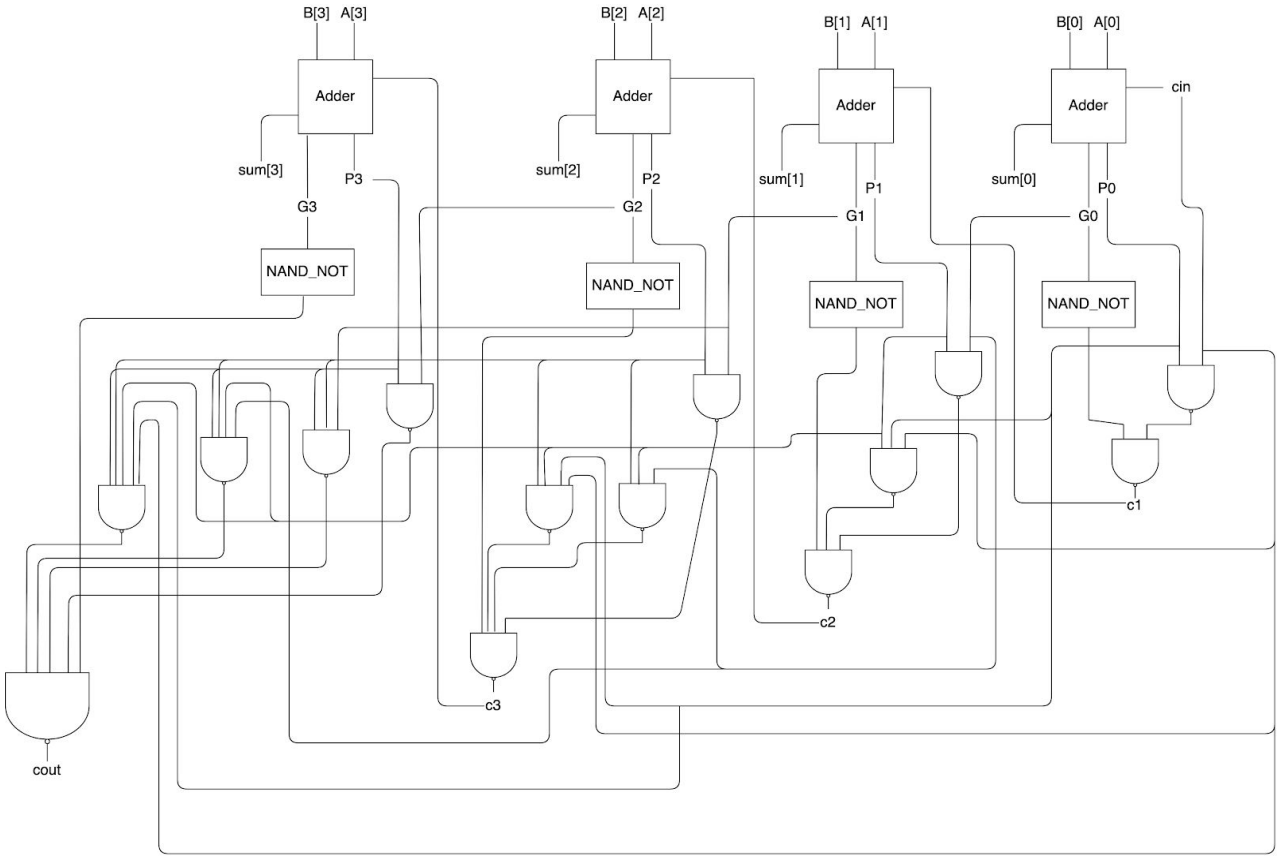




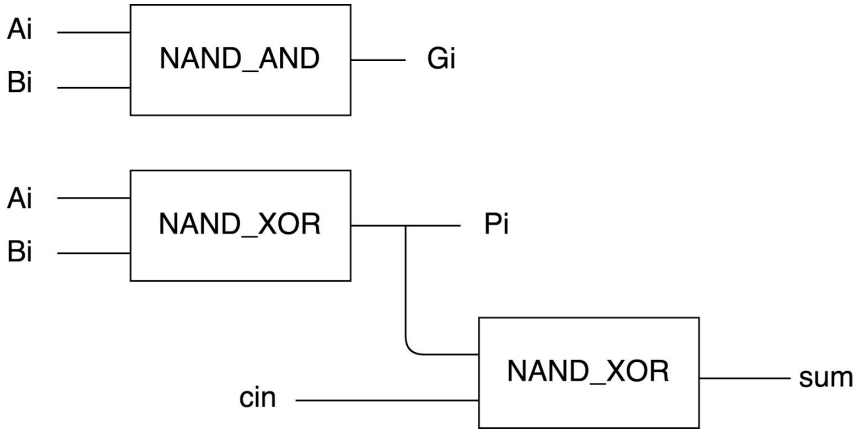
4-bits Carry-Lookahead Adder

I. Logic Design Diagram

(i)完整電路設計圖



(ii)Adder(FullAdder)電路設計



II. Explaination

(i) Advantage of CLA：

CLA的特點是將Cout以最初的input訊號(A,B,Cin)表示。與Ripple Carry Adder(RCA)相比，我們不需要等待前一個FullAdder(FA)計算出Ci，再將Ci作為下一個FA的cin訊號。使用CLA可以更快速的得到cout的結果。依維基百科的舉例說明，在一個16bits的加法器中，CLA需要的等待時間是8 gates delay，而RCA需要47 gates delay，由此可見CLA在運算時間上是更快的。

(ii) How do we implement CLA：

使用FullAdder(FA)做出G,P,Sum(做法如上圖)。並將各階段的P_i, G_i 及初始的Cin，透過多個nand做出Cout(可詳見下圖)。以最後的Cout為例，利用P,G及Cin可做出Cout = $G_3+P_3G_2+P_3P_2G_1+P_3P_2P_1G_0+P_3P_2P_1P_0Cin$。

```
//c1
NAND_NOT notG0(ng0, g0);
nand C1a(c1a, p0, cin);
nand C1b(c1, ng0, c1a);
//c2
NAND_NOT notG1(ng1, g1);
nand C2a(c2a, p1, g0);
nand C2b(c2b, cin, p0, p1);
nand C2c(c2, ng1, c2a, c2b);
//c3
NAND_NOT notG2(ng2, g2);
nand C3a(c3a, p2, g1);
nand C3b(c3b, p2, p1, g0);
nand C3c(c3c, p2, p1, p0, cin);
nand C3d(c3, ng2, c3a, c3b, c3c);
//c4
NAND_NOT notG3(ng3, g3);
nand C4a(c4a,p3, g2);
nand C4b(c4b, p3, p2, g1);
nand C4c(c4c, p3, p2, p1, g0);
nand C4d(c4d, p3, p2, p1, p0, cin);
nand C4e(cout, ng3, c4a, c4b, c4c, c4d);
```

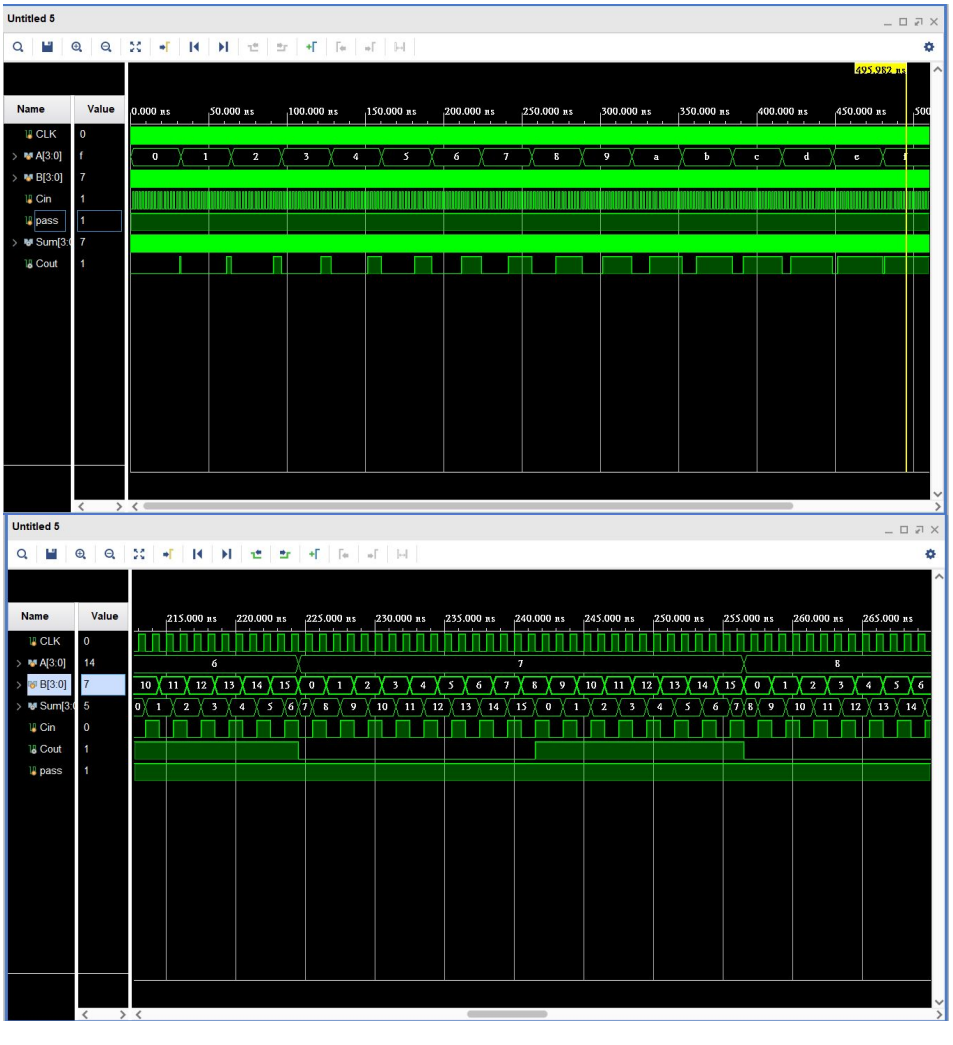
III. Testbench

窮舉所有A,B,Cin，並比較通過CLA的{Cout,Sum}與A+B+Cin，若有不同則write error messenge並使pass = 0，若相同則pass = 1，可以得知每一筆測資結果是否正確。

```
initial begin
  {A, B, Cin} = 9'b0;

  repeat (2 ** 9) begin
    @ (posedge CLK)
        Test;
    @ (negedge CLK)
        {A, B, Cin} = {A, B, Cin} + 1'b1;
  end

  $finish;
end
```

```
task Test;
begin
    if({Cout, Sum} !== (A + B + Cin))begin
        pass=0;
        $display("[ERROR]");
        $write("A:%d",A);
        $write("B:%d",B);
        $write("Cin:%d",Cin);
        $write("Cout:%d",Cout);
        $write("Sum:%d",Sum);
        $display;
    end
    else begin
        pass=1;
    end
end
endtask
```

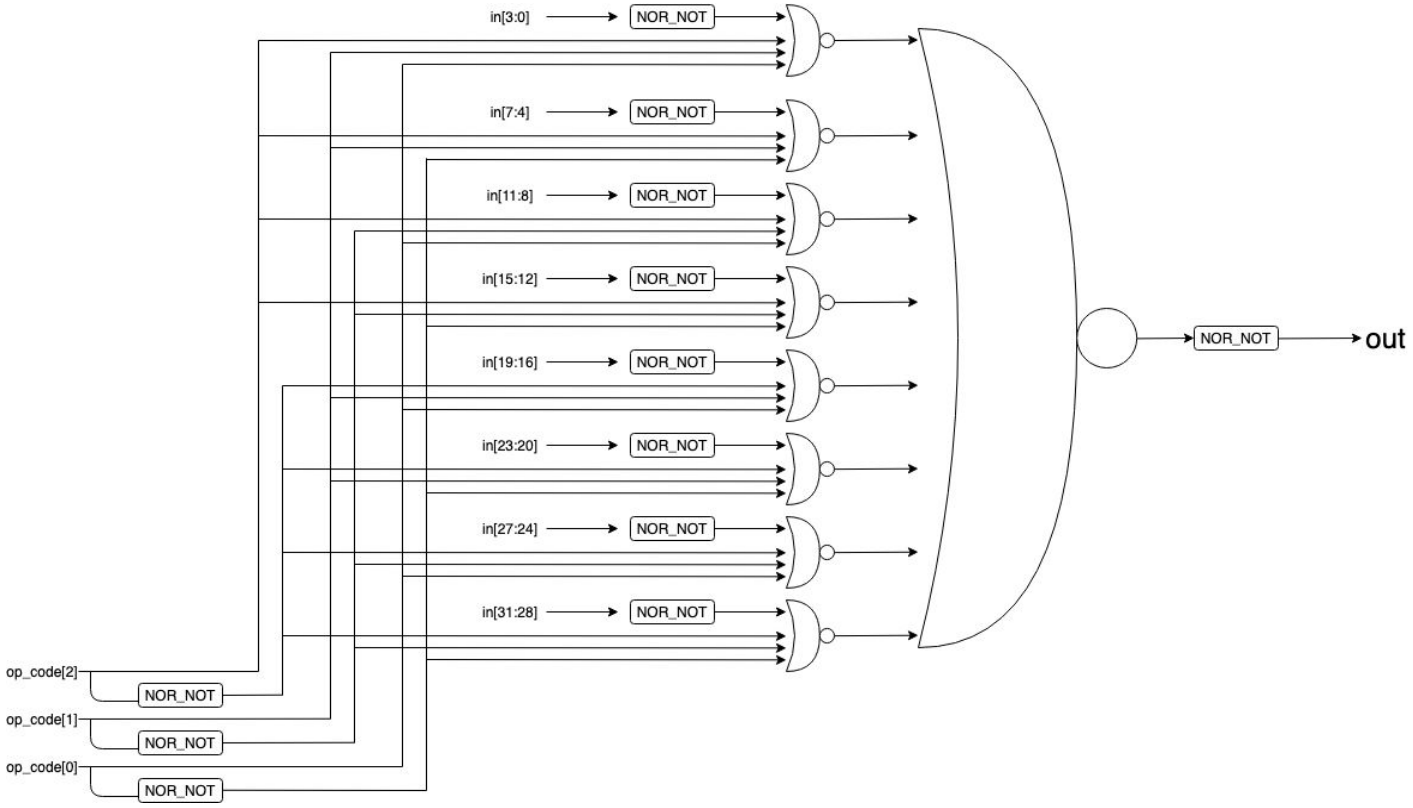
下方兩張waveform表示pass全數為1，並擷取部分測資結果。



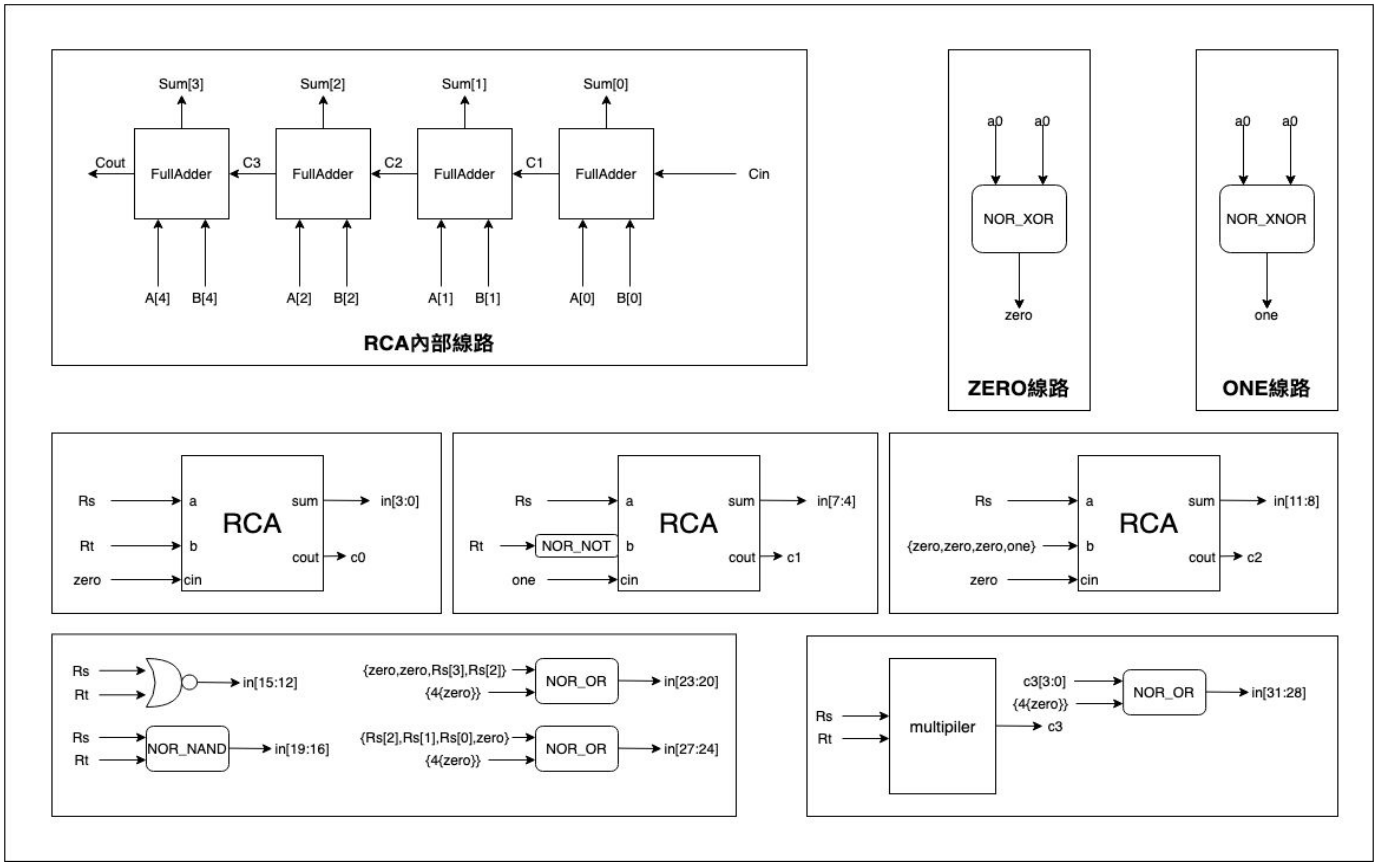
Decoding and Execution

I. Logic Design Diagram

(i)完整電路設計圖



(ii)各附件電路設計圖



II. Explaination

概念是使用Op_code控制8way MUX選擇需要的功能，圖一為使用NOR實作MUX方法，因為每一路訊號輸入是4bit，8通道就需要使用32bit，而MUX原先的AND-OR線路，在使用NOR實現時就需要新增許多NOR_NOT去輔助完成。而第一到第三個功能接是加法器的運用，較為特別的是減法需要透過將其中一個input接上NOT再將cin設定為1，就可以實現2'complement的實作，BITWISE運算則是直接使用相對應的功能完成，最後shifter的功能則是透過補0的方式實現，例如>>2的功能就是將輸出的第3、2、1、0個bit接上0、0、rs[3]、rs[2]實現，最後乘法器則是使用第二題的module實現。

```
//1
RippleCarryAdder_4bits R0 (rs,rt,zero,c0, in[3:0] );
//2
RippleCarryAdder_4bits R1 (rs,nrt,one,c1, in[7:4] );
//3
RippleCarryAdder_4bits R2 (rs,{ {3{zero}},one },zero,c2, in[11:8] );
//4
nor  nor0 [3:0](in[15:12],rs,rt);
//5
NOR_NAND N3 (rs[0],rt[0],in[16]);
NOR_NAND N4 (rs[1],rt[1],in[17]);
NOR_NAND N5 (rs[2],rt[2],in[18]);
NOR_NAND N6 (rs[3],rt[3],in[19]);
//6
NOR_OR_4bits N7 ({4{zero}}, {zero,zero,rs[3],rs[2]} ,in[23:20]);
//7
NOR_OR_4bits N8 ({4{zero}}, {rs[2],rs[1],rs[0],zero} ,in[27:24]);
//8
Multiplier M0 (rs,rt,c3);
NOR_OR_4bits N9 ({4{zero}}, c3[3:0],in[31:28]);
```

III. Testbench

因為保證ADD、SUB、INC、RS_MUL_2、MUL不會overflow，因此先設定Rt的值，然後為了避免Rs-Rt overflow，設定Rs的值與Rt相等，然後在Rt+Rs 及 Rt*Rs 及 Rs*2都不會overflow的狀況下將Rs的值逐一跑過(Rs = Rs + 1'b1)，並在每一個固定Rt,Rs值跑過所有的op_code。

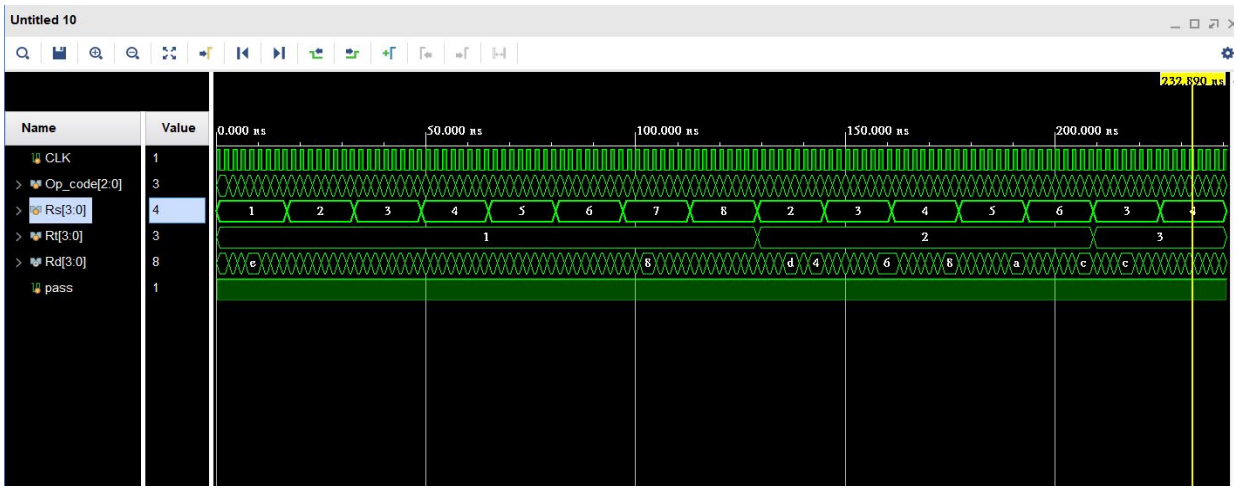
```
always #5 CLK = ~CLK;

initial begin
    Rt = 4'b1;
    Rs = 4'b1;
    Op_code = 3'b0;
    repeat (2**3) begin
        repeat (2**3) begin
            @ (posedge CLK)
                Test;
            @ (negedge CLK)
                Op_code = Op_code + 1'b1;
        end
        Rs = Rs + 1'b1;
    end
#5
    Rt = 4'b0010;
    Rs = 4'b0010;
    Op_code = 3'b0;
    repeat (2**2+1) begin
        repeat (2**3) begin
            @ (posedge CLK)
                Test;
            @ (negedge CLK)
                Op_code = Op_code + 1'b1;
        end
        Rs = Rs + 1'b1;
    end
#5
```

```
task Test;
begin

    if(Op_code === 000) begin
        if(Rd !== (Rs + Rt)) begin
            Wrong;
        end
    end
    if(Op_code === 001) begin
        if(Rd !== (Rs - Rt)) begin
            Wrong;
        end
    end
    if(Op_code === 010) begin
        if(Rd !== (Rs + 1'b1)) begin
            Wrong;
        end
    end
```

```
task Wrong;
begin
pass=0;
$display("[ERROR]");
$write("Op_code:%3b",Op_code);
$write("Rs:%4b",Rs);
$write("Rt:%4b",Rt);
$write("Rd:%4b",Rd);
$display;
```

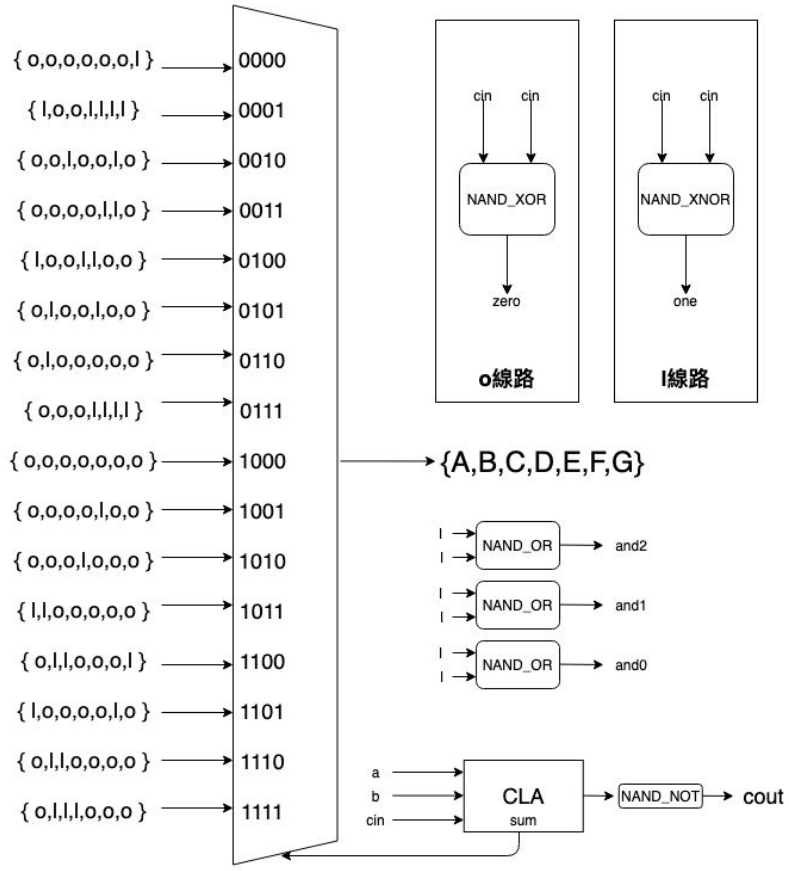
最後比較經過operation的各個輸出Rd 是否跟測資直接做運算相等，若有不相等則write error messenge，並使pass = 0 (詳見右上圖)。

下圖是waveform結果。



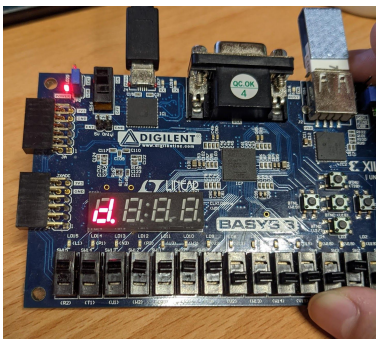
FPGA_DEMO

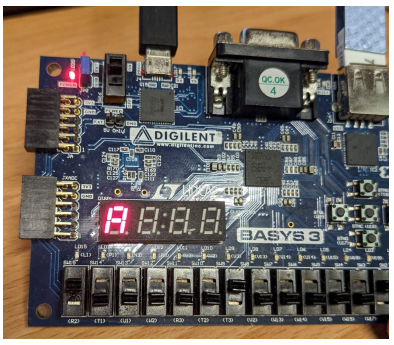
I. Logic Design Diagram

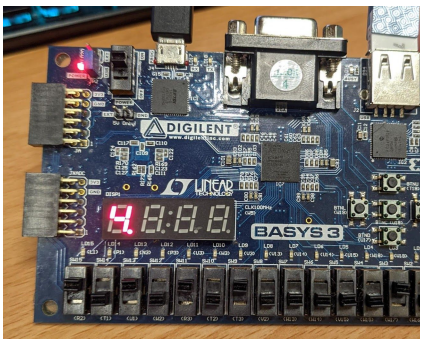


II. Explaination

在CLA的基礎上外包一個16way MUX，以CLA輸出的SUM訊號作為控制，決定要輸出哪一種七段顯示器的訊號組合，再將原先的cout跟不用亮起的的訊號以o、I訊號處理，即可達成預期成果。

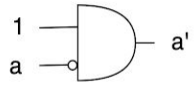




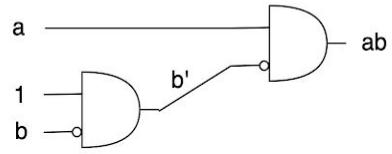


Other Universal Logic Gate

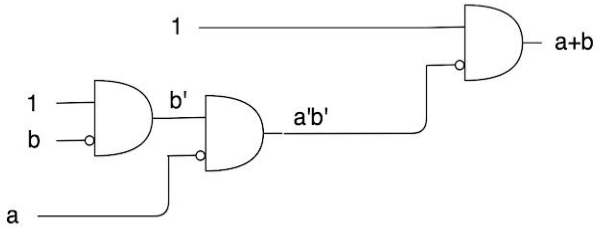
NOT



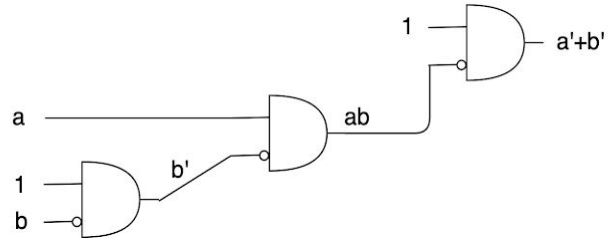
AND



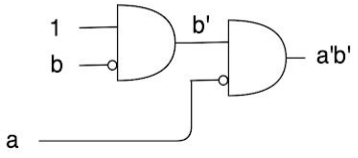
OR



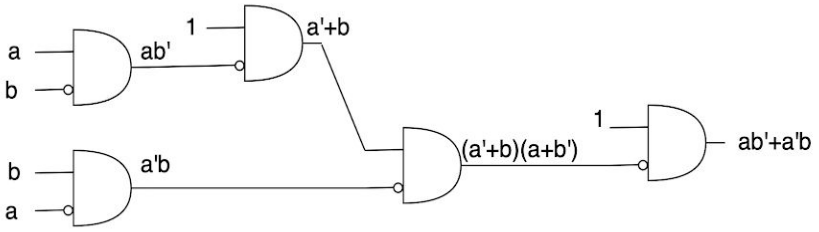
NAND



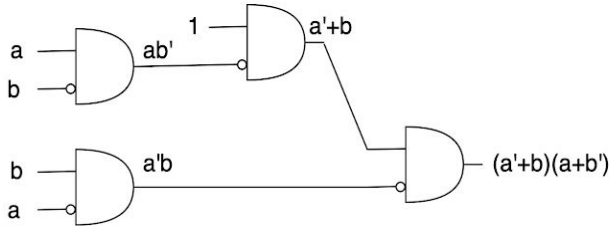
NOR



XOR



XNOR



What are we learned from this Lab

這次lab中，有許多題需要實現多功能，也因此需要使用許多的MUX實作，而因為只能使用gate-level的緣故，若是MUX的輸入選擇跟輸入bit數上升，MUX內部複雜度就會急速上升因此這次使用許多{}實作bit相接的功能實現，更顯得先畫出design diagram的重要性，並且大幅地增進了自己verilog的coding的能力。而這次題目也指定使用universal logic gates去實作，也讓我們更加有機會理解universal logic gates該如何去使用，例如如何將原先設計的module去轉換成universal logic gates的形式，但是這次在做作業時有許多不清楚的地方，最後都需要去討論區得到解答，或是有時討論區會新增多的作業需求，也養成了我們時常上去確認的習慣，也算是另外一種另類的成長吧。

Cooperation

108062211 黃子瑋：

Multiplier電路圖及實作、Decoding and Execution電路圖及實作、FPGA電路圖及實作、各題simulation、Testbench Debug。

108062213 顏浩昀：

Binary_to_Grey電路圖及實作、CLA電路圖及實作、Basic Question及Other universal Logic Gate電路圖、各題Testbench。