1 - (3)

Carry ripple adder的critical path應當是從LSB full adder的carry-in至MSB full adder的carry-out。在一個full adder中，ci到co經過一層AND和一層OR，共2ns delay。

```
module FA (
      input a,     // Clock
      input b, // Clock Enable
      input ci,  // Asynchronous reset active low
      output sum,
      output co
);
      wire w0, w1, w2;

      //sum = a ^ b ^ carry-in
      xor #1 U0(sum, a, b, ci);
      //carry-out = (a && b) || (b && carry-in) || (carry-in && a)
      and   #1 U1(w0, a, b);
      and   #1 U2(w1, b, ci);
      and   #1 U3(w2, ci, a);
      or #1 U4(co, w0, w1, w2);

endmodule
```

所以理論上8-bit carry ripple adder的clock cycle會是16ns。用助教給的測資模擬後發現clock cycle只需要9ns就夠了，推測是因為測資沒有worst case (aka. 8'hFF + 8'h1)的關係，更改測資後的最小clock cycle為16ns，和理論相同。

2 - (3)

一個MUX的critical path delay是3ns (NOT + AND + OR)。

```
module mux (x,a,b,sel);
input       a,b,sel;
output      x;
wire sel_i,w1,w2;

not #1 n0(sel_i,sel);
and #1 a1(w1,a,sel_i); //0
and #1 a2(w2,b,sel); //1
or #1 o1(x,w1,w2);


endmodule
```

而一層"level"中8個MUX之間是彼此平行運算的，所以一層的delay理論上也是3ns。Barrel shifter總共有3層所以理論上delay總和應為9ns。但是實測模擬minimum clock cycle為7ns，可能和測資沒有考慮到worst case有關。

3 - (2)

Adder: delay 16ns; shifter: delay 9ns; mode mux: delay 2.5ns. 因為adder和shifter是彼此平行的，所以理論上worst case delay會是18.5ns。但最後實測出來的模擬結果其minimum clock cycle是9.5 (= 7 + 2.5) ns，推測是因為原始測資使ASU的critical path發生在barrel shifter身上而非adder。

3 - (3)

我主要更改的架構還是adder。Carry ripple adder的minimum clock cycle是O(n)，因為carry的運算要通過所有的gate之後才能得到正確答案。改用carry look-ahead adder的話carry運

算時間是constant，當half adder算出propagate與generate後，每個bit的carry只需要經過兩個gate delay (AND + OR)就能得到。新的adder使用第一大題的測資後，minimum clock cycle的確大幅縮短為4 ns (half adder: 2ns, carry: 2ns)。

```
module adder_gate(x, y, carry, out);
input [7:0] x, y;
output carry;
output [7:0] out;

/*Write your code here*/
      wire w0, w1, w2, w3, w4, w5, w6, w7, w8, w9, w10, w11, w12, w13, w14, w15,
w16, w17, w18, w19, w20, w21, w22, w23, w24, w25, w26, w27, w28;
      wire p0, p1, p2, p3, p4, p5, p6, p7;
      wire g0, g1, g2, g3, g4, g5, g6, g7;
      wire c2, c3, c4, c5, c6, c7;

      //carry look-ahead adder: http://www.eecs.umich.edu/courses/eecs370/
eecs370.w20/resources/materials/17-FastAdders-ch06aplus.pdf
      half_adder HA0(.a(x[0]), .b(y[0]), .cin(0), .p(p0), .g(g0), .sum(out[0]));
//c0 = 0
      //and #1 AND0(w0, p0, 0); //w0 == 0
      //or #1 OR0(c1, g0, w0); //c1 == g0
      half_adder
HA1(.a(x[1]), .b(y[1]), .cin(g0), .p(p1), .g(g1), .sum(out[1])); //c1 = g0 +
p0*c0 = g0
      and #1 AND1(w1, p1, g0);
      or #1 OR1(c2, g1, w1);
      half_adder
HA2(.a(x[2]), .b(y[2]), .cin(c2), .p(p2), .g(g2), .sum(out[2])); //c2 = g1 +
p1*c1 = g1 + p1*g0
      and #1 AND2(w2, p2, g1);
      and #1 AND3(w3, p2, p1, g0);
      or #1 OR2(c3, g2, w2, w3);
      half_adder
HA3(.a(x[3]), .b(y[3]), .cin(c3), .p(p3), .g(g3), .sum(out[3])); //c3 = g2 +
p2*c2 = g2 + p2*g1 + p2*p1*g0
      and #1 AND4(w4, p3, g2);
      and #1 AND5(w5, p3, p2, g1);
      and #1 AND6(w6, p3, p2, p1, g0);
      or #1 OR3(c4, g3, w4, w5, w6);
      half_adder
HA4(.a(x[4]), .b(y[4]), .cin(c4), .p(p4), .g(g4), .sum(out[4])); //c4 = g3 +
p3*c3 = g3 + p3*g2 + p3*p2*g1 + p3*p2*p1*g0
      and #1 AND7(w7, p4, g3);
      and #1 AND8(w8, p4, p3, g2);
      and #1 AND9(w9, p4, p3, p2, g1);
      and #1 AND10(w10, p4, p3, p2, p1, g0);
      or #1 OR4(c5, g4, w7, w8, w9, w10);
      half_adder
HA5(.a(x[5]), .b(y[5]), .cin(c5), .p(p5), .g(g5), .sum(out[5])); //c5 = g4 +
p4*c4 = g4 + p4*g3 + p4*p3*g2 + p4*p3*p2*g1 + p4*p3*p2*p1*g0
      and #1 AND11(w11, p5, g4);
      and #1 AND12(w12, p5, p4, g3);
      and #1 AND13(w13, p5, p4, p3, g2);
      and #1 AND14(w14, p5, p4, p3, p2, g1);
      and #1 AND15(w15, p5, p4, p3, p2, p1, g0);
      or #1 OR5(c6, g5, w11, w12, w13, w14, w15);
      half_adder
HA6(.a(x[6]), .b(y[6]), .cin(c6), .p(p6), .g(g6), .sum(out[6])); //c6 = g5 +
```

```
p5*c5 = g5 + p5*g4 + p5*p4*g3 + p5*p4*p3*g2 + p5*p4*p3*p2*g1 +
p5*p4*p3*p2*p1*g0
      and #1 AND16(w16, p6, g5);
      and #1 AND17(w17, p6, p5, g4);
      and #1 AND18(w18, p6, p5, p4, g3);
      and #1 AND19(w19, p6, p5, p4, p3, g2);
      and #1 AND20(w20, p6, p5, p4, p3, p2, g1);
      and #1 AND21(w21, p6, p5, p4, p3, p2, p1, g0);
      or #1 OR6(c7, g6, w16, w17, w18, w19, w20, w21);
      half_adder
HA7(.a(x[7]), .b(y[7]), .cin(c7), .p(p7), .g(g7), .sum(out[7])); //c7 = g6 +
p6*c6 = g6 + p6*g5 + p6*p5*g4 + p6*p5*p4*g3 + p6*p5*p4*p3*g2 +
p6*p5*p4*p3*p2*g1 + p6*p5*p4*p3*p2*p1*g0
      //carry = g7 + p7*c7 = g7 + p7*g6 + p7*p6*g5 + p7*p6*p5*g4 +
p7*p6*p5*p4*g3 + p7*p6*p5*p4*p3*g2 + p7*p6*p5*p4*p3*p2*g1 +
p7*p6*p5*p4*p3*p2*p1*g0
      and #1 AND22(w22, p7, g6);
      and #1 AND23(w23, p7, p6, g5);
      and #1 AND24(w24, p7, p6, p5, g4);
      and #1 AND25(w25, p7, p6, p5, p4, g3);
      and #1 AND26(w26, p7, p6, p5, p4, p3, g2);
      and #1 AND27(w27, p7, p6, p5, p4, p3, p2, g1);
      and #1 AND28(w28, p7, p6, p5, p4, p3, p2, p1, g0);
      or #1 OR7(carry, g7, w22, w23, w24, w25, w26, w27, w28);

/*End of code*/

endmodule

module half_adder (
      input a,
      input b,
      input cin,
      output p, //propagate
      output g, //generate
      output sum
);

      xor #1 U0(p, a, b);
      and #1 U1(g, a, b);
      xor #1 U2(sum, cin, p);

endmodule
```

Barrel shifter的優化有上網查了一下但沒什麼人有做出相關的設計，而且要做level reduction的話，一個MUX會變很大，同樣不太能達到優化的效果，所以我還是決定維持第二大題shifter的架構。若在助教給的測資加入adder的worst case，clock cycle reduction應該就會很有感 ((16+2.5)ns -> (7+2.5)ns)，但是在上一小題的模擬結果顯示critical path在barrel shifter上 (7ns)，所以在這個情況下就算優化了adder，所需要的minimum clock cycle依然不會有變動 (9.5ns)。

3 - (4)

假設有個夠大的register P可以存乘積，當n-bit乘數B的LSB是1的時候，則把n-bit被乘數A加至P (P+A*1)，否則加0 (do nothing)，然後再把乘數B和P right barrel shift 1-bit。重複進行上述步驟n-1次即可在P得到unsigned乘法的乘積。

			a_3	a_2	a_1	a_0	← *Multiplicand*
			b_3	b_2	b_1	b_0	← *Multiplier*
		X	a_3b_0	a_2b_0	a_1b_0	a_0b_0	
		a_3b_1	a_2b_1	a_1b_1	a_0b_1	0	*Partial*
	a_3b_2	a_2b_2	a_1b_2	a_0b_2	0	0	*products*
a_3b_3	a_2b_3	a_1b_3	a_0b_3	0	0	0	
			. . .		$a_1b_0+a_0b_1$	a_0b_0	← *Product*