實用數位系統設計

作業一

1-1 Combinational 8-bit adder (RTL)

(一)設計原理：

直接的想法是直接使用8-bit ripple carry adder，為了符合Add_ctrl為1 時，是加法器，反之為減法器，所以將Add_ctrl的輸入值接上NOT gate後，再接上電路。

延遲時間的設計是假設NOT gate為1ns、XOR gate為4ns，而8-bit加法器為34ns，因為算式只有一行，所以設計成為最長路徑的延遲時間。

(二)結果分析：

case1: 加法模式且沒有發生overflow

從0ns開始至50ns，在39ns算出結果。

case2: 減法模式且沒有發生overflow

從50ns開始至100ns，在89ns算出結果。

case3: 加法模式且發生overflow

從100ns開始至150ns，在134ns時，Cout會變化，而在139ns時，算出Sum。

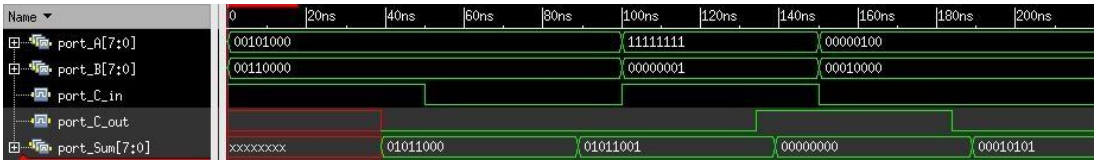
case4: 減法模式且發生overflow

從150ns開始至200ns，在184ns時，Cout會變化，而在189ns時，算出Sum。結論:當發生overflow且為加法模式時，Cout為1，而在減法模式時，Cout

為0。

最長路徑為B[0]->Sum[7]，最長延遲時間為39ns。

```
ncsim> run
case 1:
                  0:A=8'b00101000 B=8'b00110000 Add_ctrl=1'b'1 Sum=8'bxxxxxxxx Cout=1'b'x
                 39:A=8'b00101000 B=8'b00110000 Add_ctrl=1'b'1 Sum=8'b01011000 Cout=1'b'0
correct!!
case 2:
                 50:A=8'b00101000 B=8'b00110000 Add_ctrl=1'b'0 Sum=8'b01011000 Cout=1'b'0
                 89:A=8'b00101000 B=8'b00110000 Add_ctrl=1'b'0 Sum=8'b01011001 Cout=1'b'0
correct!!
case 3:
                100:A=8'b11111111 B=8'b00000001 Add_ctrl=1'b'1 Sum=8'b01011001 Cout=1'b'0
                134:A=8'b11111111 B=8'b00000001 Add_ctrl=1'b'1 Sum=8'b01011001 Cout=1'b'1
                139:A=8'b11111111 B=8'b00000001 Add_ctrl=1'b'1 Sum=8'b00000000 Cout=1'b'1
correct!!
case 4:
                150:A=8'b00000100 B=8'b00010000 Add_ctrl=1'b'0 Sum=8'b00000000 Cout=1'b'1
                184:A=8'b00000100 B=8'b00010000 Add_ctrl=1'b'0 Sum=8'b00000000 Cout=1'b'0
                189:A=8'b00000100 B=8'b00010000 Add_ctrl=1'b'0 Sum=8'b00010101 Cout=1'b'0
correct!!
Simulation complete via $finish(1) at time 230 NS + 0
./HW1RTL-tb.v:38   #30 $finish;
```



1-1 Combinational 8-bit adder (gate level)

(一)設計原理：

直接的想法是直接使用8-bit ripple carry adder，為了符合Add_ctrl為1 時，是加法器，反之為減法器，所以將Add_ctrl的輸入值接上NOT gate後，再接上電路，而電路由半加器組成全加器後，再串成八位元加法器。

延遲時間的設計是假設NOTgate為1ns、ORgate為2ns、ANDgate為2ns、XOR

gate為4ns。

(二)結果分析：

case1: 加法模式且沒有發生overflow

從0ns開始至50ns，在25ns算出結果。

case2: 減法模式且沒有發生overflow

從50ns開始至100ns，在75ns算出結果。

case3: 加法模式且發生overflow，為最長路徑的驗證。

從100ns開始至150ns，在138ns時，Cout會算出結果，而在139ns時，算出Sum。Sum會不斷的變化是因為上一位的carry_out會不斷觸發下一位的計算，導致計算時間最長。

case4: 減法模式且發生overflow

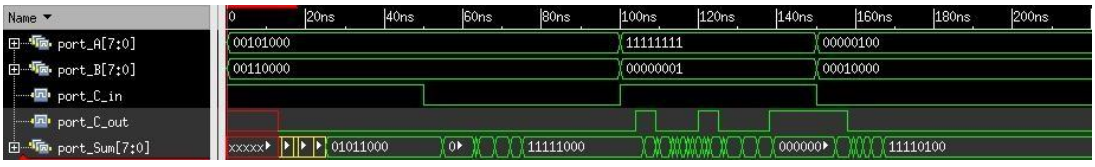
從150ns開始至200ns，在158ns時，Cout會變化，而在189ns時，算出Sum。結論:當發生overflow且為加法模式時，Cout為1，而在減法模式時，Cout

為0。

當第八個全加器的carry_in值輸入之後，等待4ns即會算出Cout，而Sum 要再等待1ns才會輸出。

最長路徑為B[0]->Sum[7]，最長延遲時間為39ns。

```
ncsim> run
case 1:
                  0:A=8'b00101000 B=8'b00110000 Add_ctrl=1'b1 Sum=8'bxxxxxxxx Cout=1'bx
                 13:A=8'b00101000 B=8'b00110000 Add_ctrl=1'b1 Sum=8'bxxxxx000 Cout=1'b0
                 14:A=8'b00101000 B=8'b00110000 Add_ctrl=1'b1 Sum=8'bx1xxx000 Cout=1'b0
                 17:A=8'b00101000 B=8'b00110000 Add_ctrl=1'b1 Sum=8'b01xxx000 Cout=1'b0
                 18:A=8'b00101000 B=8'b00110000 Add_ctrl=1'b1 Sum=8'b01xx1000 Cout=1'b0
                 22:A=8'b00101000 B=8'b00110000 Add_ctrl=1'b1 Sum=8'b01x11000 Cout=1'b0
                 25:A=8'b00101000 B=8'b00110000 Add_ctrl=1'b1 Sum=8'b01011000 Cout=1'b0
correct!!
case 2:
                 50:A=8'b00101000 B=8'b00110000 Add_ctrl=1'b0 Sum=8'b01011000 Cout=1'b0
                 55:A=8'b00101000 B=8'b00110000 Add_ctrl=1'b0 Sum=8'b01011001 Cout=1'b0
                 63:A=8'b00101000 B=8'b00110000 Add_ctrl=1'b0 Sum=8'b11011111 Cout=1'b0
                 64:A=8'b00101000 B=8'b00110000 Add_ctrl=1'b0 Sum=8'b11110110 Cout=1'b0
                 68:A=8'b00101000 B=8'b00110000 Add_ctrl=1'b0 Sum=8'b11110100 Cout=1'b0
                 72:A=8'b00101000 B=8'b00110000 Add_ctrl=1'b0 Sum=8'b11110000 Cout=1'b0
                 75:A=8'b00101000 B=8'b00110000 Add_ctrl=1'b0 Sum=8'b11111000 Cout=1'b0
correct!!
case3:
                100:A=8'b11111111 B=8'b00000001 Add_ctrl=1'b1 Sum=8'b11111000 Cout=1'b0
                104:A=8'b11111111 B=8'b00000001 Add_ctrl=1'b1 Sum=8'b11111000 Cout=1'b1
                106:A=8'b11111111 B=8'b00000001 Add_ctrl=1'b1 Sum=8'b11111001 Cout=1'b1
                109:A=8'b11111111 B=8'b00000001 Add_ctrl=1'b1 Sum=8'b10101000 Cout=1'b0
                110:A=8'b11111111 B=8'b00000001 Add_ctrl=1'b1 Sum=8'b10101110 Cout=1'b0
                113:A=8'b11111111 B=8'b00000001 Add_ctrl=1'b1 Sum=8'b10001110 Cout=1'b0
                114:A=8'b11111111 B=8'b00000001 Add_ctrl=1'b1 Sum=8'b11011110 Cout=1'b0
                115:A=8'b11111111 B=8'b00000001 Add_ctrl=1'b1 Sum=8'b11011100 Cout=1'b0
                117:A=8'b11111111 B=8'b00000001 Add_ctrl=1'b1 Sum=8'b10011100 Cout=1'b0
                118:A=8'b11111111 B=8'b00000001 Add_ctrl=1'b1 Sum=8'b10111100 Cout=1'b0
                119:A=8'b11111111 B=8'b00000001 Add_ctrl=1'b1 Sum=8'b10111000 Cout=1'b0
                120:A=8'b11111111 B=8'b00000001 Add_ctrl=1'b1 Sum=8'b10111000 Cout=1'b1
                121:A=8'b11111111 B=8'b00000001 Add_ctrl=1'b1 Sum=8'b00111000 Cout=1'b1
                122:A=8'b11111111 B=8'b00000001 Add_ctrl=1'b1 Sum=8'b01111000 Cout=1'b1
                123:A=8'b11111111 B=8'b00000001 Add_ctrl=1'b1 Sum=8'b01110000 Cout=1'b1
                125:A=8'b11111111 B=8'b00000001 Add_ctrl=1'b1 Sum=8'b01110000 Cout=1'b0
                126:A=8'b11111111 B=8'b00000001 Add_ctrl=1'b1 Sum=8'b11110000 Cout=1'b0
                127:A=8'b11111111 B=8'b00000001 Add_ctrl=1'b1 Sum=8'b11100000 Cout=1'b0
                131:A=8'b11111111 B=8'b00000001 Add_ctrl=1'b1 Sum=8'b11000000 Cout=1'b0
                135:A=8'b11111111 B=8'b00000001 Add_ctrl=1'b1 Sum=8'b10000000 Cout=1'b0
                138:A=8'b11111111 B=8'b00000001 Add_ctrl=1'b1 Sum=8'b10000000 Cout=1'b1
                139:A=8'b11111111 B=8'b00000001 Add_ctrl=1'b1 Sum=8'b00000000 Cout=1'b1
correct!!
case 4:
                150:A=8'b00000100 B=8'b00010000 Add_ctrl=1'b0 Sum=8'b00000000 Cout=1'b1
                155:A=8'b00000100 B=8'b00010000 Add_ctrl=1'b0 Sum=8'b00000001 Cout=1'b1
                158:A=8'b00000100 B=8'b00010000 Add_ctrl=1'b0 Sum=8'b00000001 Cout=1'b0
                159:A=8'b00000100 B=8'b00010000 Add_ctrl=1'b0 Sum=8'b11111001 Cout=1'b0
                160:A=8'b00000100 B=8'b00010000 Add_ctrl=1'b0 Sum=8'b11111000 Cout=1'b0
                162:A=8'b00000100 B=8'b00010000 Add_ctrl=1'b0 Sum=8'b11101000 Cout=1'b0
                164:A=8'b00000100 B=8'b00010000 Add_ctrl=1'b0 Sum=8'b11100000 Cout=1'b0
                167:A=8'b00000100 B=8'b00010000 Add_ctrl=1'b0 Sum=8'b11110100 Cout=1'b0
correct!!
Simulation complete via $finish(1) at time 230 NS + 0
./HW1CBC-tb.v:38   #30 $finish;
```



1-2 Voting circuitdesign:

(一)設計原理：

先將每個相同位元的輸入值相加後，再互相比較相同位元的數量，假如其中一個位元的數量等於三，則第一個輸出值為此票且第二個輸出值為 0，而當發生兩個位元的數量同時等於二時，則第一個輸出值和第二個輸出值為兩張不同的輸入值 。

(二)結果分析：

case1:時間為0ns~5ns。

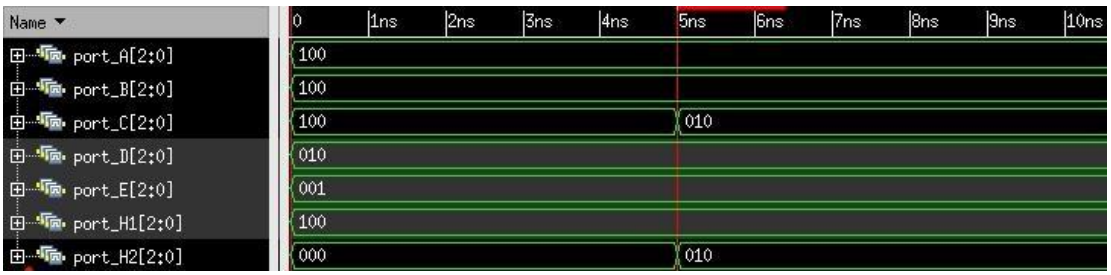
輸入分別為port_A:100、port_B:100、port_C:100、port_D:010、port_E:001，則第一個輸出(port_H1)為100，而第二輸出(port_H2)為000。

case2: 時間為5ns~10ns。

輸入分別為port_A:100、port_B:100、port_C:100、port_D:010、port_E:001，則第一個輸出(port_H1)為100，而第二個輸出(port_H2)為010。

結論: 設計與驗證結果相符。

```
ncsim> run
case 1:
                    0:A=3'b100 B=3'b100 C=3'b100 D=3'b010 E=3'b001 H1=3'b100 H2=3'b000
correct!!
case 2:
                    5:A=3'b100 B=3'b100 C=3'b010 D=3'b010 E=3'b001 H1=3'b100 H2=3'b010
correct!!
Simulation complete via $finish(1) at time 15 NS + 0
./HW1-2VCD-tb.v:36   #5 $finish;
```



1-2 Middle circuit design:

(一)設計原理：

將三個不同的值互相比較後，則可以得到中間值並且輸出，若發生兩個或三個輸入值相同時，則利用加法器與比較器找出相同的輸入值並且輸出。

(二)結果分析：

case1: 時間為0ns~5ns。

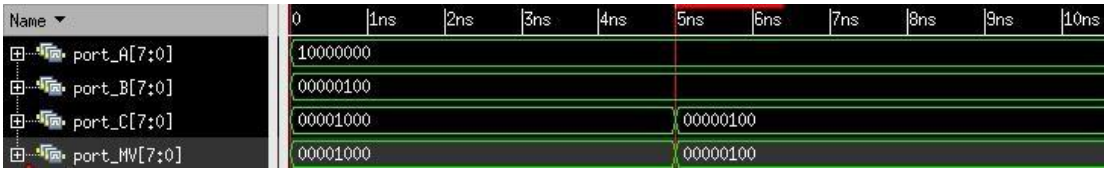
輸入分別為port_A:10000000、port_B:00000100、port_C:00001000，則第一個輸出(port_H1)為00001000。

case2: 時間為5ns~10ns。

輸入分別為port_A:10000000、port_B:00000100、port_C:00000100，則第一個輸出(port_H1)為00000100。

結論: 設計與驗證結果相符。

```
ncsim> run
case 1:
                0:A=8'b10000000 B=8'b00000100 C=8'b00001000 MV=8'b00001000
correct!!
case 2:
                5:A=8'b10000000 B=8'b00000100 C=8'b00000100 MV=8'b00000100
correct!!
Simulation complete via $finish(1) at time 15 NS + 0
./HW1-2MCD-tb.v:28    #5 $finish;
```



心得與討論

我覺得這次的實作中，最好玩的是利用 gate-level寫出 8位元加減法器了，原因在於可以設定每一個邏輯閘的延遲時間後，而模擬結果可以依照這些參數不斷的運算，並且得到輸出的每個變化；相較之下，RTL 似乎就只能以最長路徑來設定延遲時間，這樣的設定太過草率，不夠嚴謹，最後，有一個小遺憾就是未能算出 vote circuit 和 middle circuit的最長路徑與延遲時間。