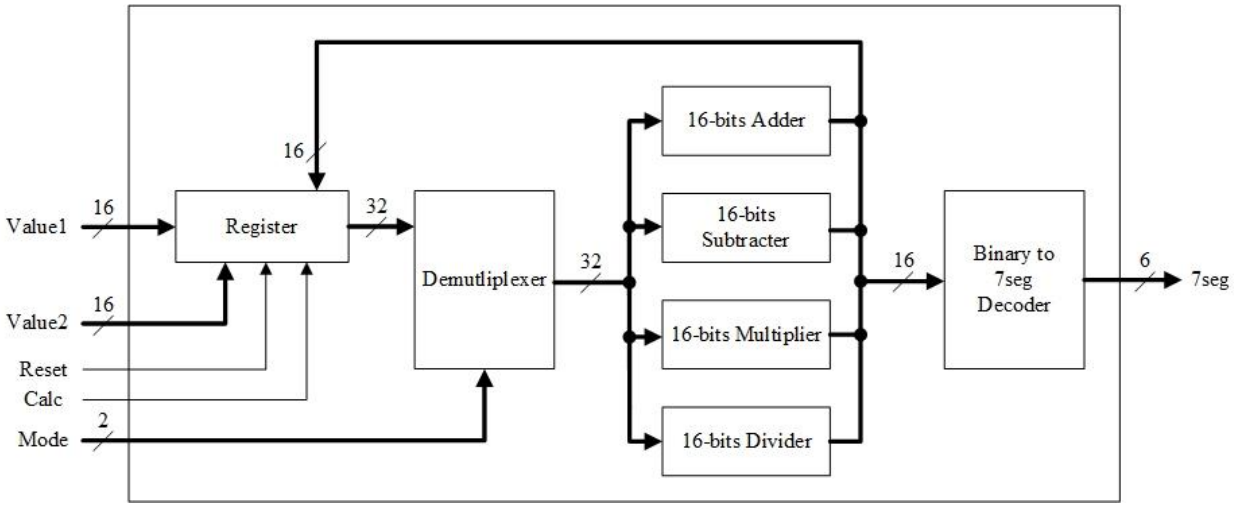
16 位元多功能計算機

1、 摘要

在實驗平台中，設計具有加、減、乘、除功能的 16 位元無符號數計算機，計算結果將呈現於 7 段顯示器中。

2、 方塊圖



3、 電路結構或 HDL

- 加法減法電路

在一般的加法器電路可以分為漣波進位加法器(Ripple-carry adder)、前瞻進位加法器(Carry Look-Ahead Adder)、串列加法器(Serial Adder)，而其各自的延遲時間如下表所示

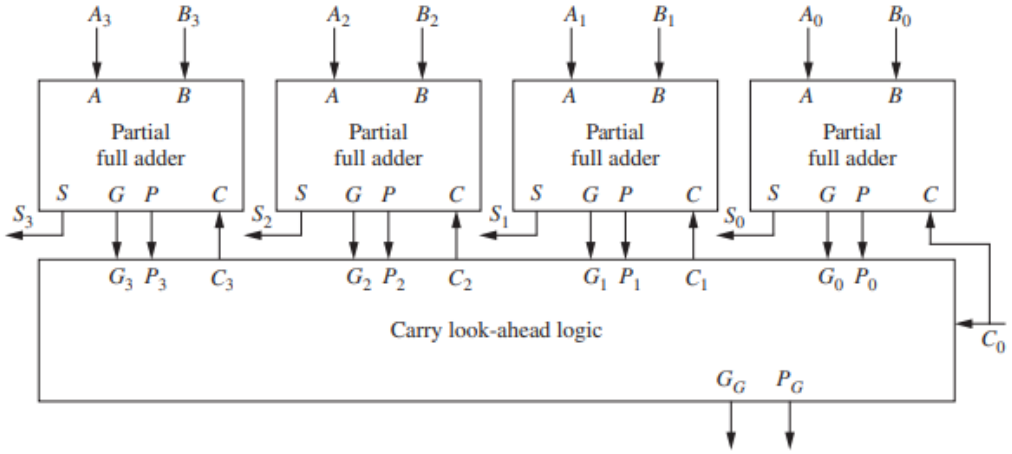
Adder size	Ripple-carry adder delay	CLA delay	Serial adder delay
4 bit	8 t_g	5-6 t_g	16 t_g
16 bit	32 t_g	7-8 t_g	64 t_g
32 bit	64 t_g	9-10 t_g	128 t_g
64 bit	128 t_g	9-10 t_g	256 t_g

在這邊已知我們的電路為 16 位元的加(減)法，考量延遲時間以及合成完的電路大小後，選擇使用前瞻進位加法器。

前瞻進位加法器是在全加器的基礎上再加上前瞻進位邏輯，透過直接讀取輸入的各位元數值，來直接得到原本須透過全加器的進位輸出來的到的進位，進一步縮短電路延遲。已知全加器進位輸出的布林代數如下

$$C_{i+1} = A_i \oplus B_i \oplus C_i = A_iB_i + (A_i \oplus B_i)C_i$$
$$= G_i + P_iC_i$$

所以我們可以透過這個關係先組成 4 位元的前瞻進位加法器



全加器 (Verilog Code)

```
module GPFullAdder (X, Y, Cin, G, P, Sum);

        input X, Y, Cin;
        output G, P, Sum;

        wire P_int;

        assign G = X & Y;
        assign P = P_int;
        assign P_int = X ^ Y;
        assign Sum = P_int ^ Cin;

endmodule
```

前瞻進位邏輯 (Verilog Code)

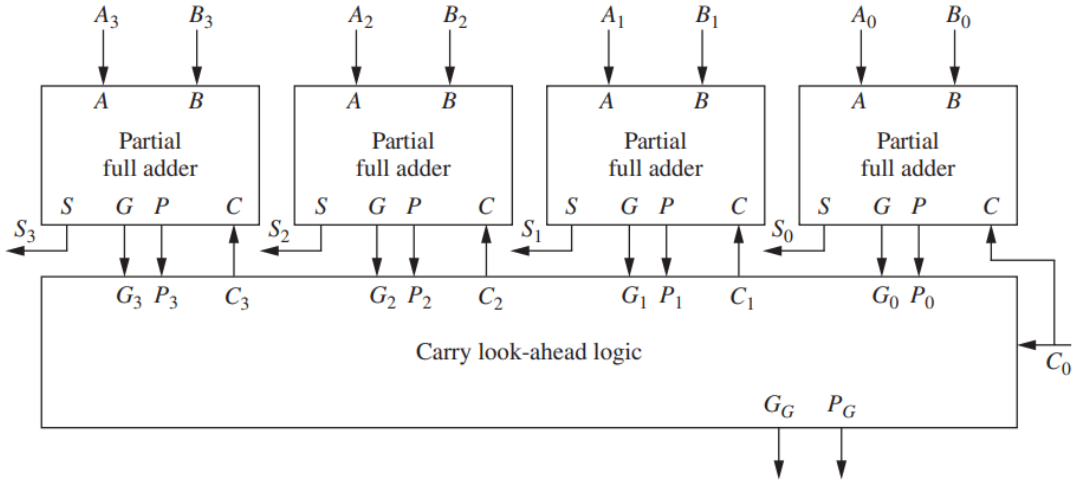
```
module CLALogic (G, P, Ci, C, Co, PG, GG);

        input           [3:0]   G;
        input           [3:0]   P;
        input                   Ci;
        output          [3:1]   C;
        output                  Co;
        output                  PG;
        output                  GG;

        wire GG_int, PG_int;

        assign C[1] = G[0] | (P[0] & Ci);
        assign C[2] = G[1] | (P[1] & G[0]) | (P[1] & P[0] & Ci);
        assign C[3] = G[2] | (P[2] & G[1]) | (P[2] & P[1] &
G[0]) | (P[2] & P[1] & P[0] & Ci);
```

```
        assign Co = GG_int | (PG_int & Ci);
        assign PG_int = P[3] & P[2] & P[1] & P[0];
        assign GG_int = G[3] | (P[3] & G[2]) | (P[3] & P[2] &
G[1]) | (P[3] & P[2] & P[1] & G[0]);
        assign PG = PG_int;
        assign GG = GG_int;

endmodule
```

4 位元前瞻進位加法器 (Verilog Code)

```
module CarryLookAheadAdder4 (A, B, Ci, S, PG, GG);
        input   [3:0] A;
        input   [3:0] B;
        input                   Ci;
        output  [3:0] S;
        output                  PG; // group propagate
        output                  GG; // group generate

        wire            [3:0]   G;
        wire            [3:0]   P;
        wire            [3:1]   C;

        CLALogic CarryLogic(G, P, Ci, C, Co, PG, GG);
        GPFullAdder FA0(A[0], B[0], Ci, G[0], P[0], S[0]);
        GPFullAdder FA1(A[1], B[1], C[1], G[1], P[1], S[1]);
        GPFullAdder FA2(A[2], B[2], C[2], G[2], P[2], S[2]);
        GPFullAdder FA3(A[3], B[3], C[3], G[3], P[3], S[3]);

endmodule
```

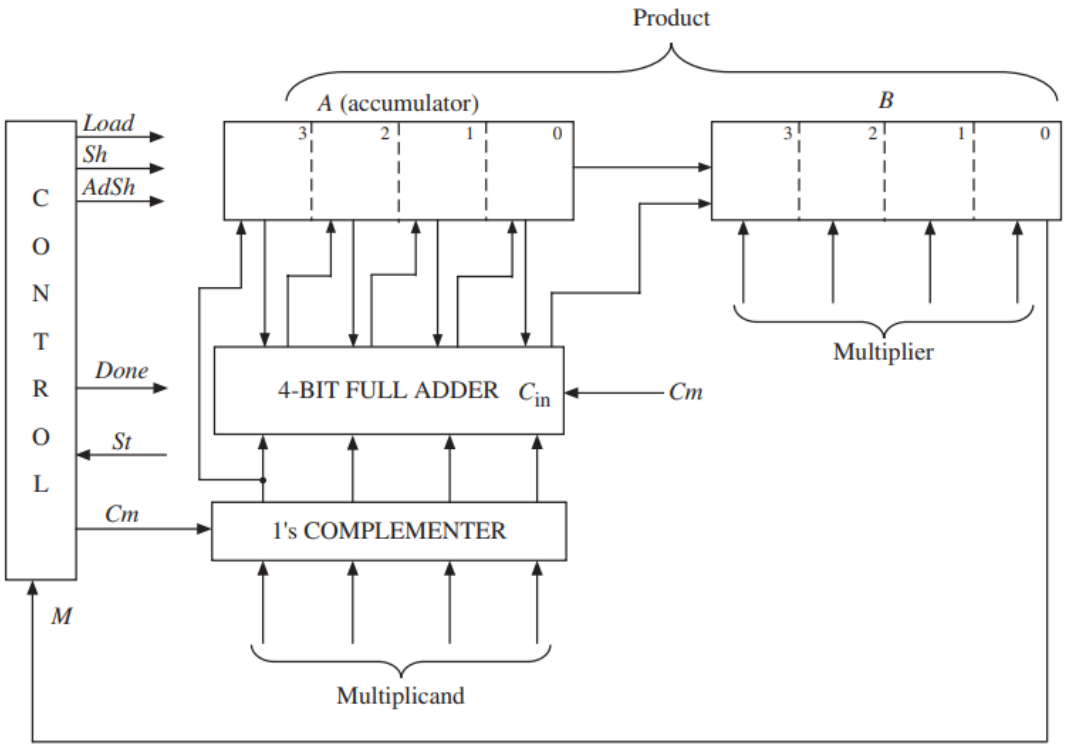
接下來就可以再將其組合成 16 位元的前瞻進位加法器



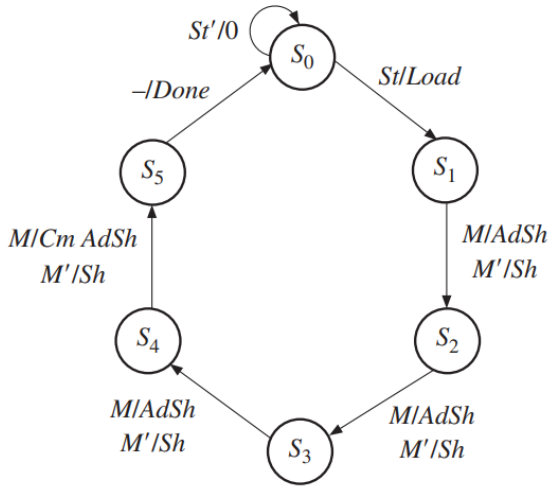
16 位元前瞻進位加法器 (Verilog Code)

```
module CLA16(A, B, Ci, S, Co, PG, GG);

        input   [15:0] A;
        input   [15:0] B;
        input                   Ci;
        output  [15:0] S;
        output                  Co;
        output                  PG; // group propagate
        output                  GG; // group generate

        wire            [3:0]   G;
        wire            [3:0]   P;
        wire            [3:1]   C;

        CLALogic BlockCarryLogic(G, P, Ci, C, Co, PG, GG);
        CarryLookAheadAdder4 CLA0 (A[3:0], B[3:0], Ci, S[3:0],
P[0], G[0]);
        CarryLookAheadAdder4 CLA1 (A[7:4], B[7:4], C[1], S[7:4],
P[1], G[1]);
        CarryLookAheadAdder4 CLA2 (A[11:8], B[11:8], C[2], S[11:8],
P[2], G[2]);
        CarryLookAheadAdder4 CLA3 (A[15:12], B[15:12], C[3],
S[15:12], P[3], G[3]);

endmodule
```

- 乘法電路



我們以四位元的例子進行解說，相乘兩個 4 位元分數（包括符號）。使用 4 位元加法器，因此和的符號不會因為進位到符號位而丟失。控制電路的 M 輸入是乘數的當前活動位元。控制信號 Sh 使累加器以符號擴展的方式向右移一位。Ad 使加法器的輸出載入到累加器的左側 4 位元中。由於我們在進行二補數加法，因此丟棄了加法器最後一位元的進位輸出。Cm 使被乘數（Mcand）在進入加法器輸入之前進行補數（一補數）。Cm 還連接到加法器的進位輸入，因此當 Cm = 1 時，加法器將 1 加上一補數的 Mcand 加到累加器中，這相當於加上 Mcand 的二補數。M 是符號位，如果 M = 1，則被乘數的補數被加到累加器中。這個狀態下加法和移位操作必須在兩個獨立的時鐘時間內完成。我們可以通過將從加法器輸出的導線右移一位來加快乘法器的操作，使得加法器輸出在加載到累加器時已經右移一位。通過這種排列，加法和移位操作可以在同一個時鐘時間內完成，產生了下圖中的控制狀態圖。在本專題中，將 4 位元的運算增加至 16 位元。



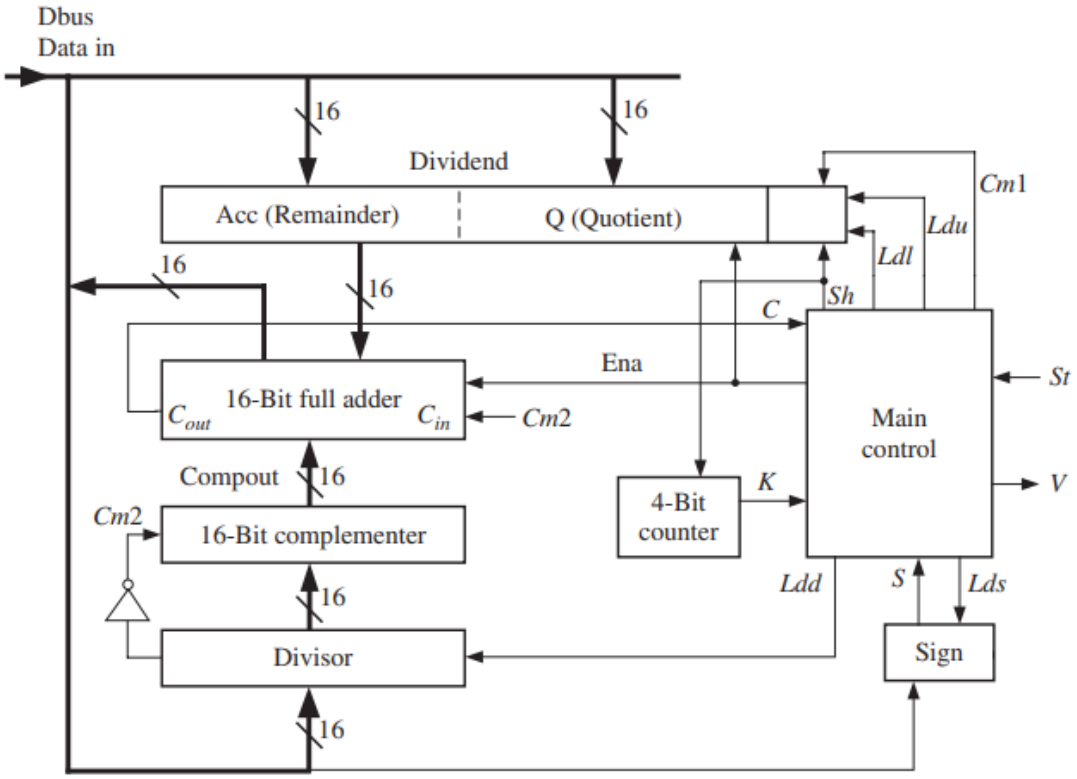
16 位元乘法器 (Verilog Code)

```
module Mult16 (CLK, St, Mplier, Mcand, CalcFinish, Product, Done);
        input CLK;
        input St;
        input[15:0] Mplier;
        input[15:0] Mcand;
        input CalcFinish;
        output[31:0] Product;
        output Done;

        reg Done = 0;
        reg[5:0] State = 0;
        reg[5:0] Nextstate = 0;
        reg[15:0] A = 0;
        reg[15:0] B = 0;
```

```
        wire[15:0] compout;
        wire[15:0] addout;
        reg AdSh = 0;
        reg Sh = 0;
        reg Load = 0;
        reg Cm = 0;

        always @(State, St, B[0]) begin
                Load = 1'b0; AdSh = 1'b0; Sh = 1'b0; Cm = 1'b0;
Done = 1'b0; Nextstate = 1'b0;
                case (State)
                        0 : begin
                                if (St == 1'b1) begin
                                        Load = 1'b1 ;
                                        Nextstate = 1 ;
                                end
                                else begin
                                        Load = 1'b0 ;
                                        Nextstate = 0 ;
                                end
                        end
                        1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
13, 14, 15 : begin
                                if (B[0] == 1'b1) begin
                                        AdSh = 1'b1 ;
                                end
                                else begin
                                        Sh = 1'b1 ;
                                end
                                Nextstate = State + 1 ;
                        end
                        16 : begin
                                if (B[0] == 1'b1) begin
                                        Cm = 1'b1 ;
                                        AdSh = 1'b1 ;
                                end
                                else begin
                                        Sh = 1'b1 ;
                                end
                                Nextstate = 17 ;
                        end
                        17 : begin
                                Done = 1'b1 ;
                                Nextstate = 0 ;
                                if (!CalcFinish) begin
                                        Nextstate = 17 ;
                                end
                        end
                        default : begin
                                Done = 1'b0 ;
                                Nextstate = 0 ;
                        end
```

```
                endcase
        end

        assign compout = (Cm == 1'b1) ? ~Mcand : Mcand ;
        assign addout = A + compout + Cm ;

        always @(posedge CLK) begin
                if (Load == 1'b1) begin
                        A <= 0 ;
                        B <= Mplier ;
                end
                if (AdSh == 1'b1) begin
                        A <= {compout[15], addout[15:1]} ;
                        B <= {addout[0], B[15:1]} ;
                end
                if (Sh == 1'b1) begin
                        A <= {A[15], A[15:1]} ;
                        B <= {A[0], B[15:1]} ;
                end
                State <= Nextstate ;
        end

        assign Product = {A[14:0], B} ;
endmodule
```

- 除法電路



執行符號數除法的步驟如下：

1. 從匯流排加載被除數的上半部分，並將被除數的符號複製到符號觸發器。
2. 從匯流排加載被除數的下半部分。
3. 從匯流排加載除數。
4. 如果被除數為負數，則對其進行補數。
5. 如果存在溢出條件，則進入完成狀態。
6. 否則通過一系列的移位和減法來進行除法。
7. 當除法完成時，如果有必要，對商進行補數並進入完成狀態。

控制電路分為兩部分：主要控制部分，它決定移位和減法的順序；以及計數器部分，它計算移位的次數。當發生 15 次移位時，計數器輸出信號 K = 1。

16 位元除法器 (Verilog Code)

```
module Div16 (CLK, St, Dividend_in, Divisor_in, Quotient, Remainder,
V, CalcFinish, Rdy);
        input CLK;
        input St;
        input CalcFinish;
        input[15:0] Dividend_in;
        input[15:0] Divisor_in;
        output[15:0] Quotient;
        output[15:0] Remainder;
        output V; // Overflow
        output reg Rdy; // Ready Signal

        reg                             V = 0;
        reg     [2:0]   State;
        reg     [3:0]   Count;
        reg                             Sign;
        wire                            C, Cm2;
        reg     [15:0]  Divisor;
        wire    [15:0]  Sum;
        wire    [15:0]  Compout;
        reg     [31:0]  Dividend;

        assign Cm2 = ~Divisor[15] ;
        assign Compout = (Cm2 == 1'b0) ? Divisor : ~Divisor ;
        assign Sum = Dividend[31:16] + Compout + Cm2;
        assign C = ~Sum[15] ;
        assign Quotient = Dividend[15:0] ;
        assign Remainder = Dividend[31:16];

        initial begin
                State = 0;
        end
```

```
        always @(posedge CLK) begin
                case (State)
                        0 : begin
                                Rdy = 0;
                                if (St == 1'b1) begin
                                        Dividend[31:0] <= {16'b0,
Dividend_in} ;
                                        Sign <= Dividend_in[15] ;
                                        V <= 1'b0 ;
                                        State <= 1 ;
                                        Count <= 4'b0000 ;
                                end
                        end
                        1 : begin
                                Divisor <= Divisor_in ;
                                if (Sign == 1'b1) begin
                                        Dividend <= ~Dividend + 1 ;
                                end
                                State <= 2 ;
                        end
                        2 : begin
                                Dividend <= {Dividend[30:0], 1'b0} ;
                                Count <= Count + 1 ;
                                State <= 3 ;
                        end
                        3 : begin
                                if (C == 1'b1) begin
                                        V <= 1'b1 ;
                                        State <= 0 ;
                                end
                                else begin
                                        Dividend <= {Dividend[30:0],
1'b0} ;
                                        Count <= Count + 1 ;
                                        State <= 4 ;
                                end
                        end
                        4 : begin
                                if (C == 1'b1) begin
                                        Dividend[31:16] <= Sum ;
                                        Dividend[0] <= 1'b1 ;
                                end
                                else begin
                                        Dividend <= {Dividend[30:0],
1'b0} ;
                                        if (Count == 15) begin
                                                State <= 5 ;
                                        end
                                        Count <= Count + 1 ;
                                end
                        end
```

```
                        5 : begin
                                State <= 6 ;
                                if (C == 1'b1) begin
                                        Dividend[31:16] <= Sum ;
                                        Dividend[0] <= 1'b1 ;
                                        State <= 5 ;
                                end
                                else if ((Sign ^ Divisor[15]) ==
1'b1) begin
                                        Dividend[15:0] <=
~Dividend[15:0] + 1 ;
                                        if(Sign == 1) begin
                                                Dividend [31:16] <=
~Dividend[31:16] + 1;
                                        end
                                        Rdy = 1;
                                end
                                else begin
                                        if(Sign && Divisor[15]) begin
                                                Dividend [31:16] <=
~Dividend[31:16] + 1;
                                        end
                                        Rdy = 1;
                                end
                        end
                        6 : begin // Wait
                                State <= 0 ;
                                if (!CalcFinish) begin
                                        State <= 6 ;
                                end
                        end
                endcase
        end
endmodule
```

- 系統時序

當系統一開始時，七段顯示器會顯示指波開關所對應的十進位數字，並等待第一個數字輸入完成並案下運算的符號鍵。等待手鬆開第二個按鍵後，等待第二個數字輸入完成並案下運算鍵或復原鍵。若按下復原鍵，則回到輸入第一個數字那一步重新輸入。若按下運算鍵，則會給出開始運算的訊號(St = 1)。在進入下個時序時，會先將開始運算的訊號歸零(St = 0)，並等待乘法與除法電路回傳完成運算的訊號(Done = 1 & Ryd = 1)，完成後才將結果傳送至七段顯示器上。若要重新運算則按下復原鍵即可開始新一輪的運算。

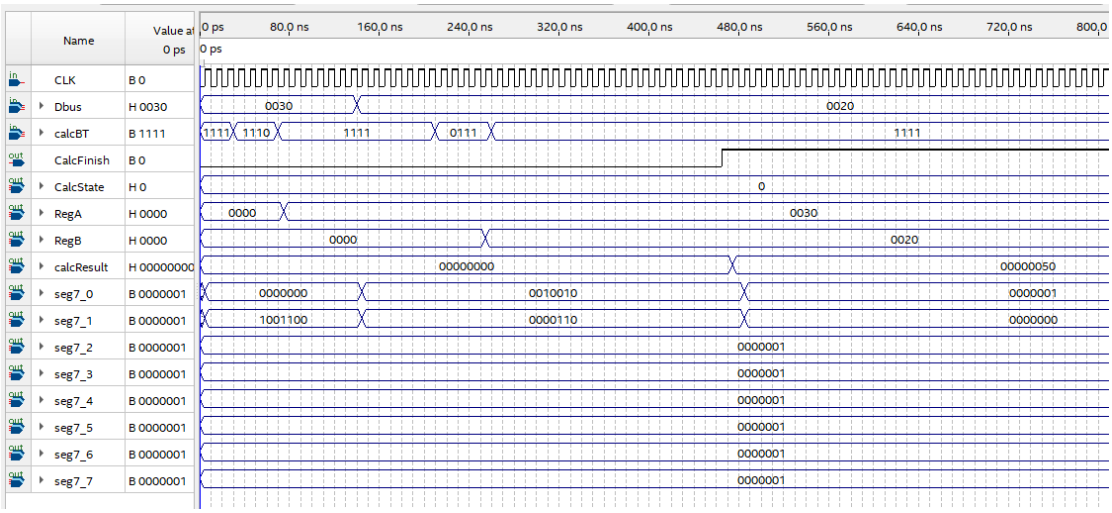
系統時序 (Verilog Code)

```
always @(posedge CLK) begin
                case (State)
                        0 : begin
                                bcd_out = bcd_in;
                                CalcFinish = 0;
                                if (calcBT_De != 4'b1111) begin
                                        case (calcBT_De)
                                                4'b1110 : CalcState
= 2'b00; // Add
                                                4'b1101 : CalcState
= 2'b01; // Sub
                                                4'b1011 : CalcState
= 2'b10; // Mul
                                                4'b0111 : CalcState
= 2'b11; // Div
                                        endcase
                                        RegA = Dbus;
                                        State = 1;
                                end
                        end
                        1 : begin
                                if (calcBT_De == 4'b1111) begin
                                        State = 2;
                                end
                        end
                        2 : begin
                                bcd_out = bcd_in;
                                if (calcBT_De == 4'b0111) begin
                                        RegB = Dbus;
                                        St = 1;
                                        State = 3;
                                end
                                else if (calcBT_De == 4'b1011) begin
// Reset
                                        State = 7;
                                end
                        end
                        3 : begin
                                CalcFinish = Done & Rdy;
                                St = 0;
                                State = 4;
                                if (!CalcFinish) begin
                                        State = 3 ;
                                end
                        end
                        4 : begin
                                case (CalcState)
                                        2'b00 : calcResult = {Co_A,
Out_A}; // Add
                                        2'b01 : calcResult = Out_S;
// Sub
```

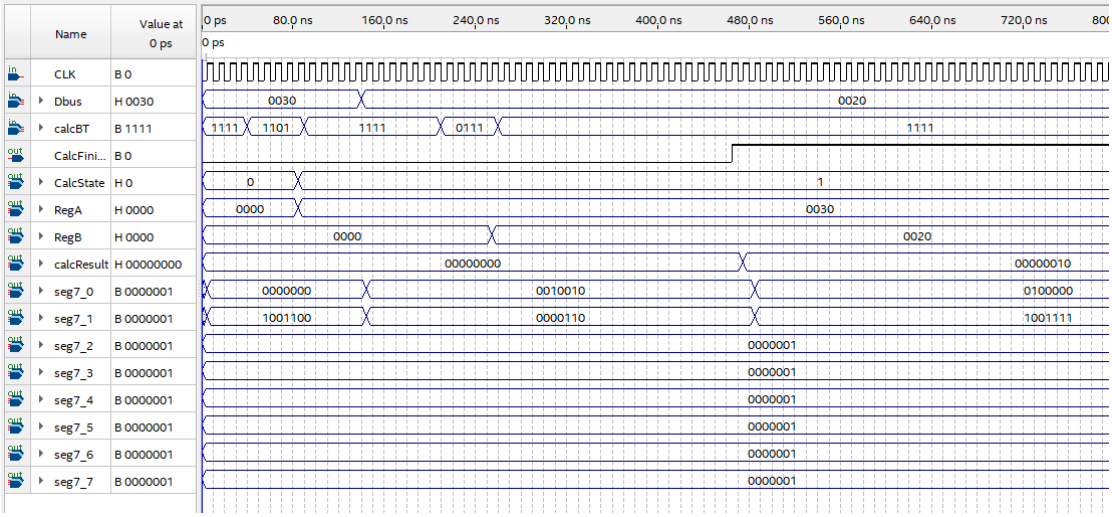
```
                                        2'b10 : calcResult =
Product; // Mul
                                        2'b11 : calcResult =
Quotient; // Div
                                endcase
                                State = 5;
                        end
                        5 : begin
                                bcd_out = bcd_R;
                                State = 6;
                        end
                        6 : begin
                                if (calcBT_De != 4'b1111) begin
                                        case (calcBT_De)
                                                4'b1011 : begin //
Reset
                                                        State = 7;
                                                end
                                        endcase
                                end
                        end
                        7 : begin
                                if (calcBT_De == 4'b1111) begin
                                        State = 0;
                                end
                        end
                endcase
        end
```

4、 波形模擬

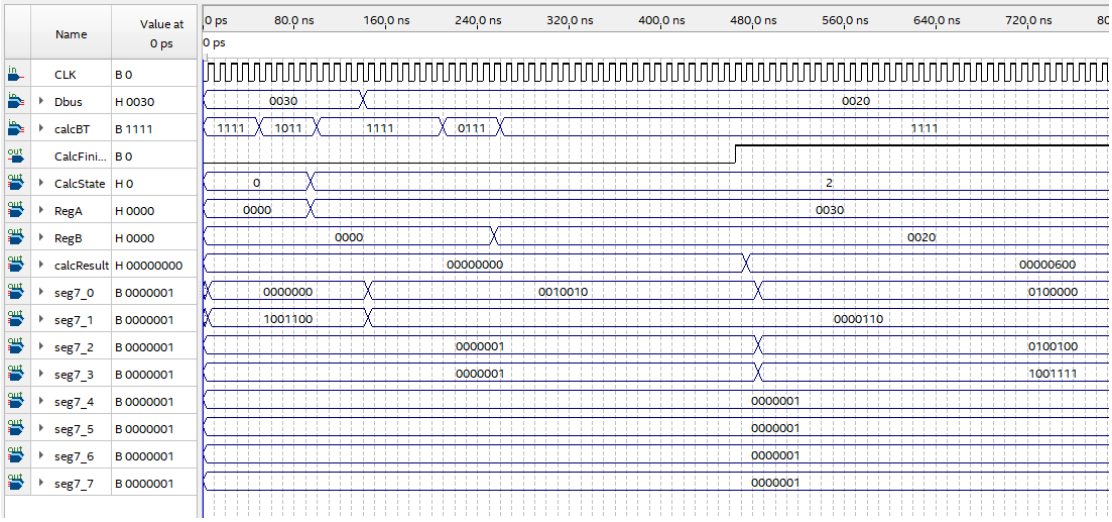
- 加法



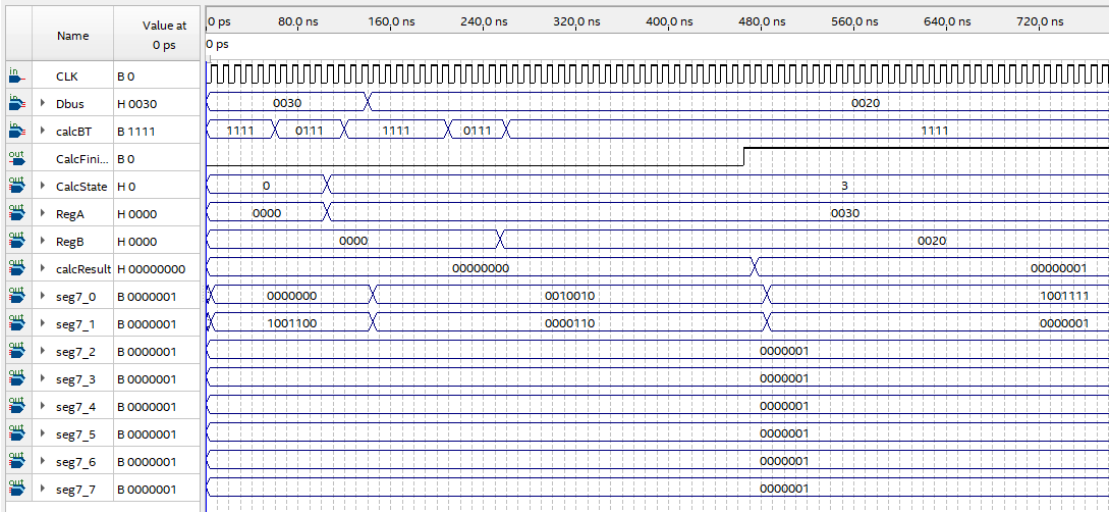
- 减法


Name
Value at 0 ps
0 ps
80.0 ns
160.0 ns
240.0 ns
320.0 ns
400.0 ns
480.0 ns
560.0 ns
640.0 ns
720.0 ns
CLK B 0
Dbus H 0030
calcBT B 1111
CalcFini... B 0
CalcState H 0
RegA H 0000
RegB H 0000
calcResult H 00000000
seg7_0 B 0000001
seg7_1 B 0000001
seg7_2 B 0000001
seg7_3 B 0000001
seg7_4 B 0000001
seg7_5 B 0000001
seg7_6 B 0000001
seg7_7 B 0000001

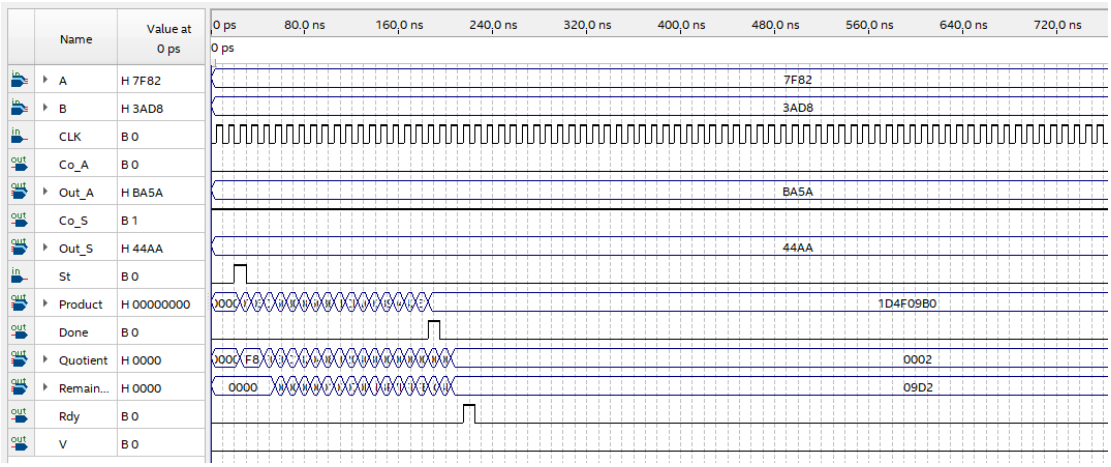
- 乘法


Name
Value at 0 ps
0 ps
80.0 ns
160.0 ns
240.0 ns
320.0 ns
400.0 ns
480.0 ns
560.0 ns
640.0 ns
720.0 ns
CLK B 0
Dbus H 0030
calcBT B 1111
CalcFini... B 0
CalcState H 0
RegA H 0000
RegB H 0000
calcResult H 00000000
seg7_0 B 0000001
seg7_1 B 0000001
seg7_2 B 0000001
seg7_3 B 0000001
seg7_4 B 0000001
seg7_5 B 0000001
seg7_6 B 0000001
seg7_7 B 0000001

- 除法


Name
Value at 0 ps
0 ps
80.0 ns
160.0 ns
240.0 ns
320.0 ns
400.0 ns
480.0 ns
560.0 ns
640.0 ns
720.0 ns
CLK B 0
Dbus H 0030
calcBT B 1111
CalcFini... B 0
CalcState H 0
RegA H 0000
RegB H 0000
calcResult H 00000000
seg7_0 B 0000001
seg7_1 B 0000001
seg7_2 B 0000001
seg7_3 B 0000001
seg7_4 B 0000001
seg7_5 B 0000001
seg7_6 B 0000001
seg7_7 B 0000001

- 內部運算電路



5、 成果

影片網址：

https://youtu.be/hf9ig_Ci3AM

6、 結論

這次我們完成各種運算電路的設計後，再加上整體的控制時序電路後，放到開發版上能正常運作，是一個非常難忘得經驗。希望透過這次的課程及專題，可以更加確定我對未來研究所及工作的志向。老師說寫 Verilog 不單單只是一行行的程式，而是還要考慮實際電路的特性，我是完全體會到了，若以後的混和電路設計環境中需要再用上，一定會更加熟悉。