第七次實驗報告

題目:7-1~7-6

姓名:羅名志

學號:0813228

繳交日期:2022/4/27

一、7-1

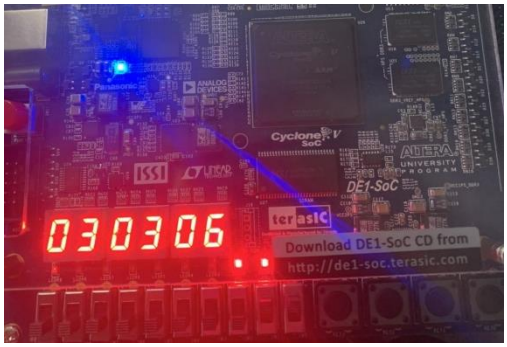
(1)實驗程式碼:

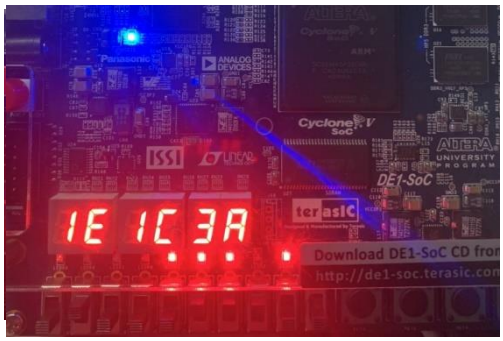
```
module labseven1(KEY,SW,LEDR,HEX5,HEX4,HEX3,HEX2,HEX1,HEX0);
	input [1:0]KEY;
	input [8:0]SW;
	output [8:0]LEDR;
	output [6:0]HEX0;
	output [6:0]HEX1;
	output [6:0]HEX2;
	output [6:0]HEX3;
	output [6:0]HEX4;
	output [6:0]HEX5;
	reg [7:0] a;
	reg [7:0] b;
	wire [7:0]c;
	wire [7:0]s;

	always @(negedge KEY[1],negedge KEY[0]) begin
		if (KEY[0]==0) begin
			a<=0;
			b<=0;
		end

		else   begin
			if(SW[8]==1) begin
				b<=SW[7:0];
			end
			else begin
				a<=SW[7:0];
			end
		end
	end

	hexto7segment s1(a[7:4],HEX5);
	hexto7segment s2(a[3:0],HEX4);
	hexto7segment s3(b[7:4],HEX3);
	hexto7segment s4(b[3:0],HEX2);
```

```
        full_adder f1(a[0], b[0], 0, c[0], s[0]);
        full_adder f2(a[1], b[1], c[0], c[1], s[1]);
        full_adder f3(a[2], b[2], c[1], c[2], s[2]);
        full_adder f4(a[3], b[3], c[2], c[3], s[3]);
        full_adder f5(a[4], b[4], c[3], c[4], s[4]);
        full_adder f6(a[5], b[5], c[4], c[5], s[5]);
        full_adder f7(a[6], b[6], c[5], c[6], s[6]);
        full_adder f8(a[7], b[7], c[6], c[7], s[7]);

        assign LEDR[7:0]=s[7:0];
        assign LEDR[8]=c[7]^c[6];
        hexto7segment s5(s[7:4],HEX1);
        hexto7segment s6(s[3:0],HEX0);

endmodule

module full_adder(a,b,cin,cout,sout);
input a;
input b;
input cin;
output cout;
output sout;
wire C1, C2, S1;

process(a,b,C1,S1);
process(S1,cin,C2,sout);
assign cout=C1|C2;

endmodule

module process(x,y,C,S);
input x,y;
output C, S;

assign C=(x&y);
assign S=(~x&y)|(x&~y);

endmodule
```

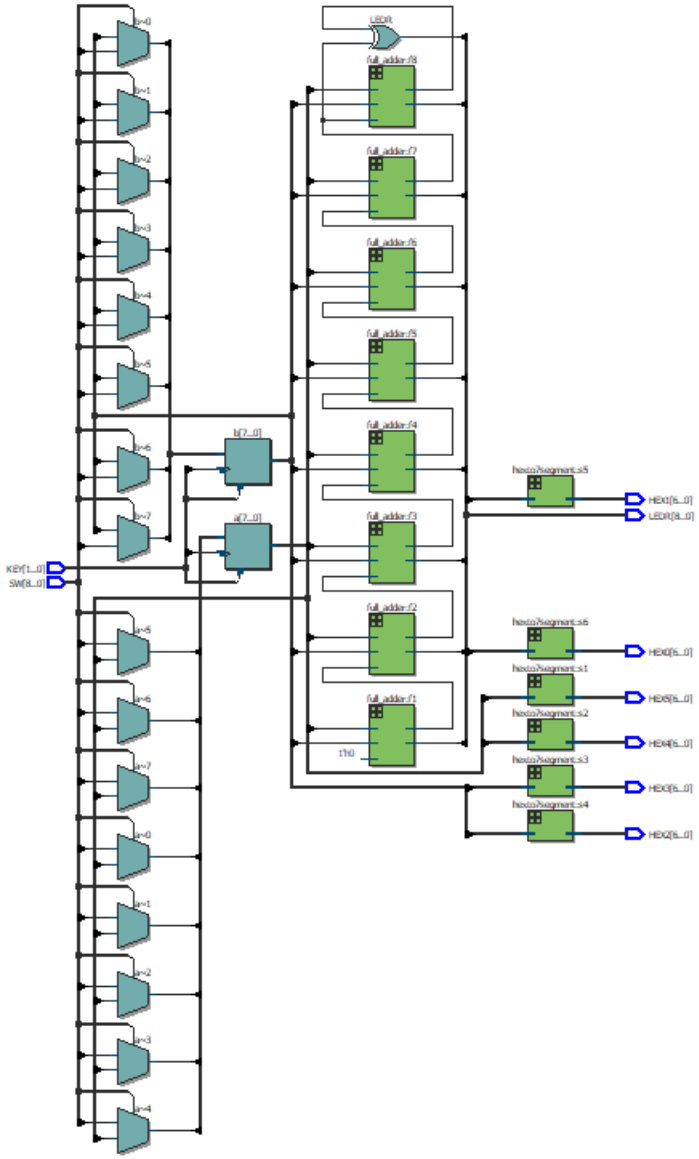
```
module hexto7segment (
    input [3:0] iDIG,
    output reg [6:0] oSEG
        );

  always@(iDIG) begin
    case(iDIG)
        4'h1: oSEG = 7'b1111001;
       4'h2: oSEG = 7'b0100100;    // ---t----          4'h2: oSEG = 7'b0100100;    // |
|
        4'h3: oSEG = 7'b0110000;    // lt       rt
        4'h4: oSEG = 7'b0011001;    // |           |
        4'h5: oSEG = 7'b0010010;    // ---m----
        4'h6: oSEG = 7'b0000010;    // |           |
        4'h7: oSEG = 7'b1111000;    // lb       rb
        4'h8: oSEG = 7'b0000000;    // |           |
        4'h9: oSEG = 7'b0011000;    // ---b----
        4'ha: oSEG = 7'b0001000;
        4'hb: oSEG = 7'b0000011;
        4'hc: oSEG = 7'b1000110;
        4'hd: oSEG = 7'b0100001;
        4'he: oSEG = 7'b0000110;
        4'hf: oSEG = 7'b0001110;
        4'h0: oSEG = 7'b1000000;
    endcase
  end

endmodule
```

(2)實驗結果:





(3)RTL 布局:



(4)問題與討論:

這個 part 算是這次 lab 最不複雜的部份，主要是沿用上次的加法器並加長到 8 bit 長度，然後搭配 KEY 呈現 reset 跟設定 A、B 值的功能。

二、7-2

(1)實驗程式碼:

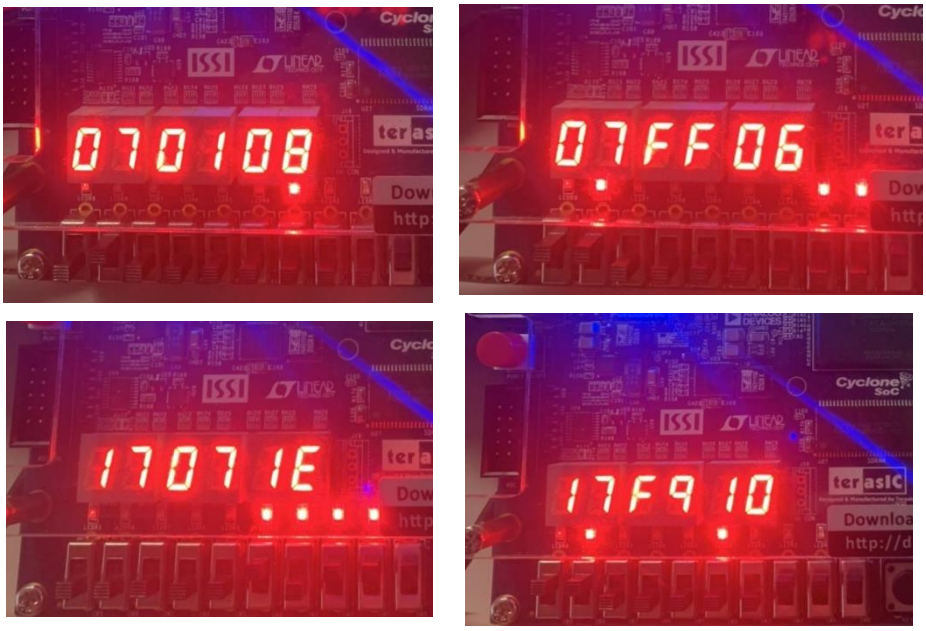
```
module labseven2(KEY,SW,LEDR,HEX5,HEX4,HEX3,HEX2,HEX1,HEX0);
	input [1:0]KEY;
	input [9:0]SW;
	output [8:0]LEDR;
	output [6:0]HEX0;
	output [6:0]HEX1;
	output [6:0]HEX2;
	output [6:0]HEX3;
	output [6:0]HEX4;
	output [6:0]HEX5;
	reg [7:0] a;
	reg [7:0] b;
	wire [7:0]c;
	wire [7:0]s;

	always @(negedge KEY[1],negedge KEY[0]) begin
		if (KEY[0]==0) begin
			a<=0;
			b<=0;
		end

		else    begin
			if(SW[9]==0) begin
				if(SW[8]==1) begin
					b<=SW[7:0];
				end
				else begin
					a<=SW[7:0];
				end
			end
			else begin
				if(SW[8]==0)begin
					a<=SW[7:0];
				end
				else begin
```

```
                    b[7]=~SW[7];
                    b[6]=~SW[6];
                    b[5]=~SW[5];
                    b[4]=~SW[4];
                    b[3]=~SW[3];
                    b[2]=~SW[2];
                    b[1]=~SW[1];
                    b[0]=~SW[0];
                    b<=b+1;
                end
            end
        end
    end

    hexto7segment s1(a[7:4],HEX5);
    hexto7segment s2(a[3:0],HEX4);
    hexto7segment s3(b[7:4],HEX3);
    hexto7segment s4(b[3:0],HEX2);
    full_adder f1(a[0], b[0], 0, c[0], s[0]);
    full_adder f2(a[1], b[1], c[0], c[1], s[1]);
    full_adder f3(a[2], b[2], c[1], c[2], s[2]);
    full_adder f4(a[3], b[3], c[2], c[3], s[3]);
    full_adder f5(a[4], b[4], c[3], c[4], s[4]);
    full_adder f6(a[5], b[5], c[4], c[5], s[5]);
    full_adder f7(a[6], b[6], c[5], c[6], s[6]);
    full_adder f8(a[7], b[7], c[6], c[7], s[7]);

    assign LEDR[7:0]=s[7:0];
    assign LEDR[8]=c[7];
    hexto7segment s5(s[7:4],HEX1);
    hexto7segment s6(s[3:0],HEX0);

endmodule

module full_adder(a,b,cin,cout,sout);
input a;
input b;
input cin;
```

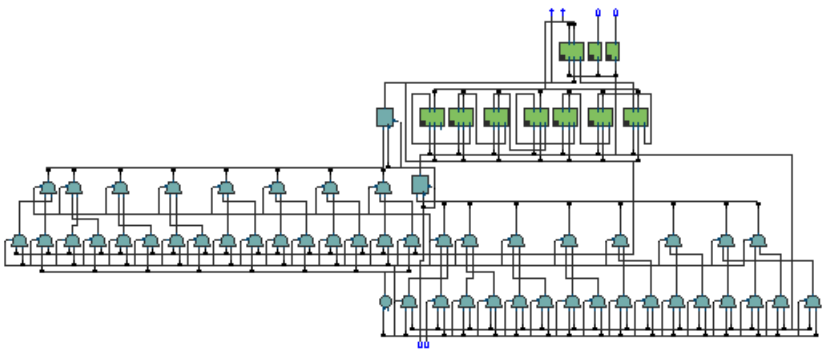
```
output cout;
output sout;
wire C1, C2, S1;

process(a,b,C1,S1);
process(S1,cin,C2,sout);
assign cout=C1|C2;

endmodule

module process(x,y,C,S);
input x,y;
output C, S;

assign C=(x&y);
assign S=(~x&y)|(x&~y);

endmodule


module hexto7segment (
    input [3:0] iDIG,
    output reg [6:0] oSEG
        );

  always@(iDIG) begin
    case(iDIG)
        4'h1: oSEG = 7'b1111001;
      4'h2: oSEG = 7'b0100100;    // ---t----          4'h2: oSEG = 7'b0100100;    // |
|
        4'h3: oSEG = 7'b0110000;    // lt        rt
        4'h4: oSEG = 7'b0011001;    // |            |
        4'h5: oSEG = 7'b0010010;    // ---m----
        4'h6: oSEG = 7'b0000010;    // |            |
        4'h7: oSEG = 7'b1111000;    // lb        rb
        4'h8: oSEG = 7'b0000000;    // |            |
        4'h9: oSEG = 7'b0011000;    // ---b----
        4'ha: oSEG = 7'b0001000;
```

```
        4'hb: oSEG = 7'b0000011;
        4'hc: oSEG = 7'b1000110;
        4'hd: oSEG = 7'b0100001;
        4'he: oSEG = 7'b0000110;
        4'hf: oSEG = 7'b0001110;
        4'h0: oSEG = 7'b1000000;
    endcase
  end

endmodule
```

(2)實驗結果:



(3) RTL 布局



(4) 問題與討論

整體跟第一 part 一樣，但當 SW[9]=1 時，將 B 的值設定成其補數加上 1，最後再與 A 相加，完成減法的操作。

三、7-3

(1) 實驗程式碼:

主程式:

```
module labseven3(KEY,SW,LEDR,HEX5,HEX4,HEX3,HEX2,HEX1,HEX0);
	input [1:0]KEY;
	input [8:0]SW;
	output [8:0]LEDR;
	output [6:0]HEX0;
	output [6:0]HEX1;
	output [6:0]HEX2;
	output [6:0]HEX3;
	output [6:0]HEX4;
	output [6:0]HEX5;
	reg [7:0] a;
	reg [7:0] b;
	wire [0:0]c;
	wire [7:0]s;

	always @(negedge KEY[1],negedge KEY[0]) begin
		if (KEY[0]==0) begin
			a<=0;
			b<=0;
		end

		else   begin
			if(SW[8]==1) begin
				b<=SW[7:0];
			end
			else begin
				a<=SW[7:0];
			end
		end
	end

	hexto7segment s1(a[7:4],HEX5);
	hexto7segment s2(a[3:0],HEX4);
	hexto7segment s3(b[7:4],HEX3);
```

```
        hexto7segment s4(b[3:0],HEX2);
        test(a[7:0],b[7:0],c,s[7:0]);

        assign LEDR[7:0]=s[7:0];
        assign LEDR[8]=c;
        hexto7segment s5(s[7:4],HEX1);
        hexto7segment s6(s[3:0],HEX0);


endmodule


module hexto7segment (
    input [3:0] iDIG,
    output reg [6:0] oSEG
        );

  always@(iDIG) begin
    case(iDIG)
        4'h1: oSEG = 7'b1111001;
      4'h2: oSEG = 7'b0100100;    // ---t----          4'h2: oSEG = 7'b0100100;    // |
|
        4'h3: oSEG = 7'b0110000;    // lt        rt
        4'h4: oSEG = 7'b0011001;    // |           |
        4'h5: oSEG = 7'b0010010;    // ---m----
        4'h6: oSEG = 7'b0000010;    // |           |
        4'h7: oSEG = 7'b1111000;    // lb        rb
        4'h8: oSEG = 7'b0000000;    // |           |
        4'h9: oSEG = 7'b0011000;    // ---b----
        4'ha: oSEG = 7'b0001000;
        4'hb: oSEG = 7'b0000011;
        4'hc: oSEG = 7'b1000110;
        4'hd: oSEG = 7'b0100001;
        4'he: oSEG = 7'b0000110;
        4'hf: oSEG = 7'b0001110;
        4'h0: oSEG = 7'b1000000;
    endcase
  end
```

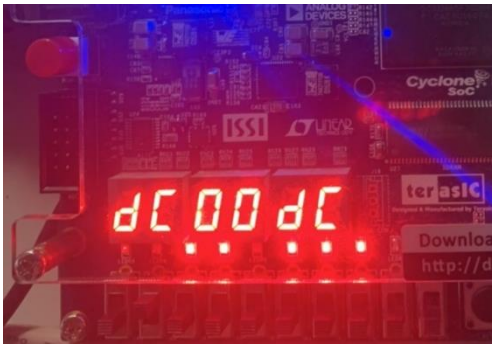
```
endmodule
```

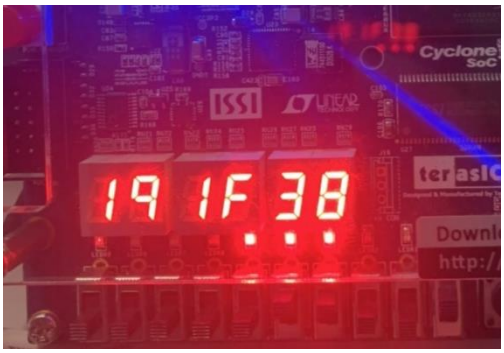
MegaWizard:

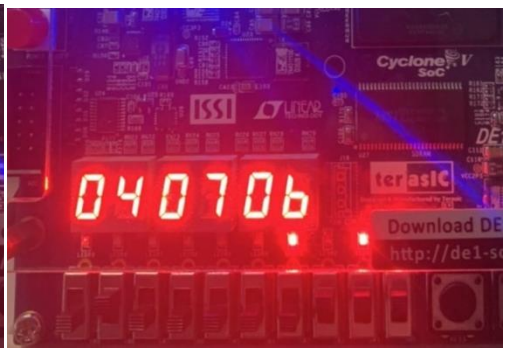
```
module test (
	dataa,
	datab,
	overflow,
	result);

	input	[7:0]	dataa;
	input	[7:0]	datab;
	output		overflow;
	output	[7:0]	result;

	wire	sub_wire0;
	wire [7:0] sub_wire1;
	wire	overflow = sub_wire0;
	wire [7:0] result = sub_wire1[7:0];

	lpm_add_sub	LPM_ADD_SUB_component (
				.dataa (dataa),
				.datab (datab),
				.overflow (sub_wire0),
				.result (sub_wire1)
				// synopsys translate_off
				,
				.aclr (),
				.add_sub (),
				.cin (),
				.clken (),
				.clock (),
				.cout ()
				// synopsys translate_on
				);
	defparam
		LPM_ADD_SUB_component.lpm_direction = "ADD",
```

```
        LPM_ADD_SUB_component.lpm_hint =
"ONE_INPUT_IS_CONSTANT=NO,CIN_USED=NO",
        LPM_ADD_SUB_component.lpm_representation = "SIGNED",
        LPM_ADD_SUB_component.lpm_type = "LPM_ADD_SUB",
        LPM_ADD_SUB_component.lpm_width = 8;


endmodule
```

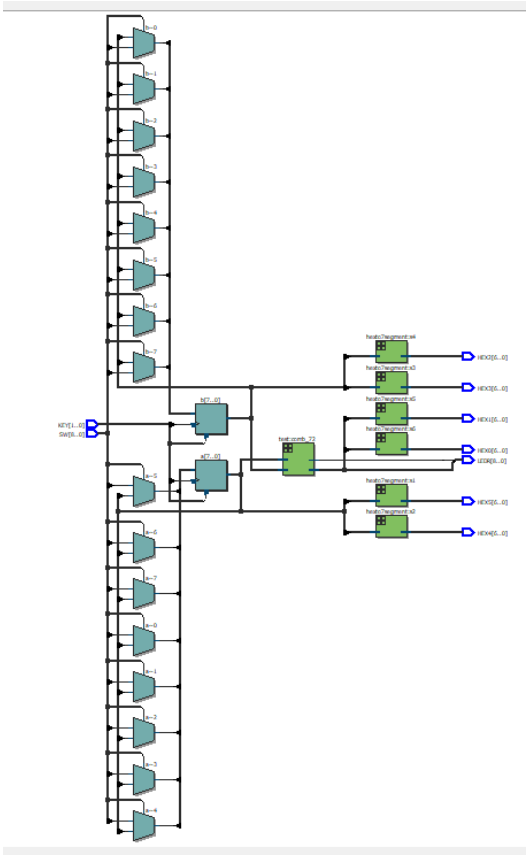
(2)實驗結果:





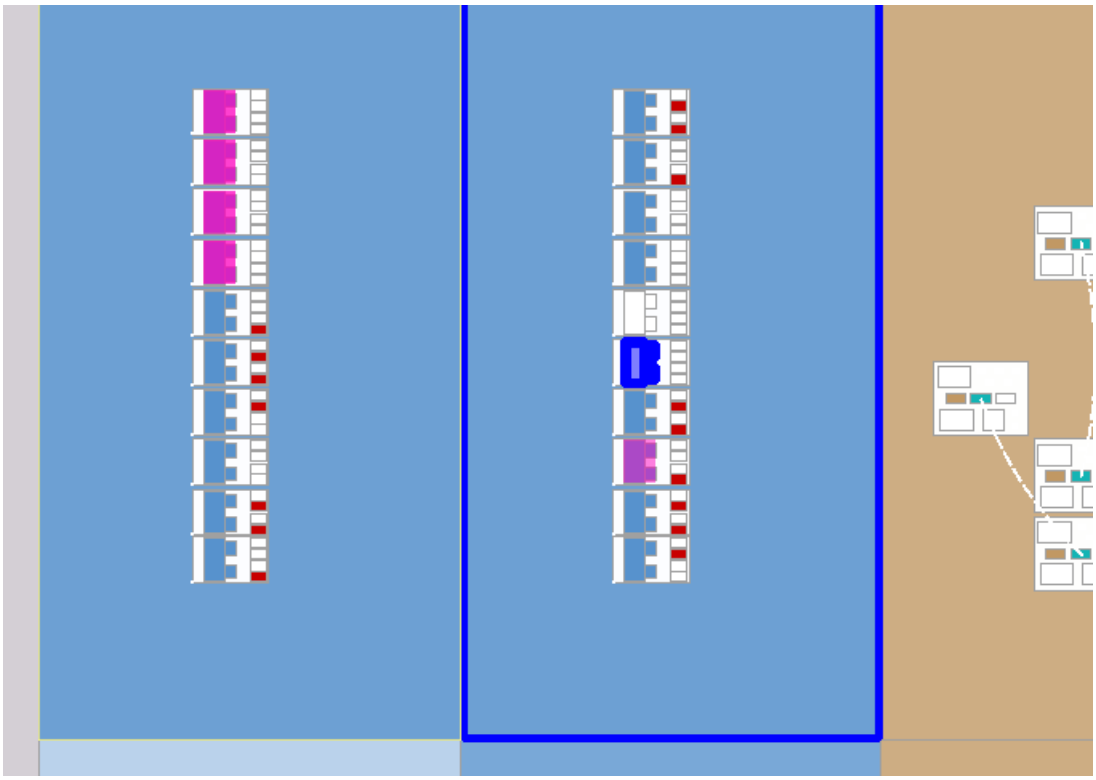


(3)RTL 布局:



(4)問題與討論:

利用 MegaWizard 的 LPM adder 直接替代第一 part 的功能，完成兩個 8bit 的加法，另外，照講義上的做法也學會如何利用 chip planner 觀察其邏輯閘皆法。



四、7-4

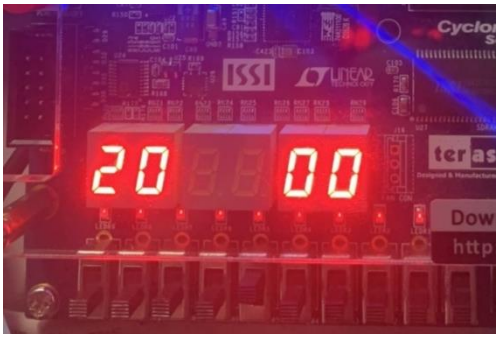
(1)實驗程式碼:

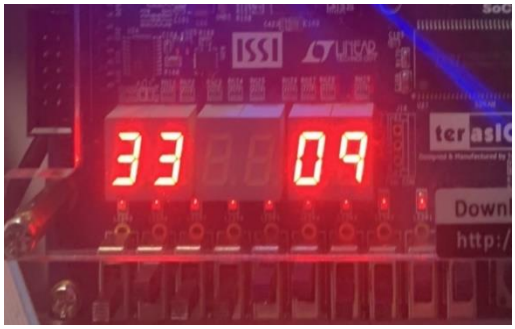
```
module labseven4(SW,HEX5,HEX4,HEX1,HEX0);
	input [8:0]SW;
	output [6:0]HEX0;
	output [6:0]HEX1;

	output [6:0]HEX4;
	output [6:0]HEX5;

	wire [10:0]c;
	wire [7:0]s;
	wire [23:0]w;

	hexto7segment s1(SW[7:4],HEX5);
	hexto7segment s2(SW[3:0],HEX4);

	and(w[0],SW[0],SW[4]);
	and(w[1],SW[0],SW[5]);
	and(w[2],SW[1],SW[4]);
	and(w[3],SW[0],SW[6]);
	and(w[4],SW[1],SW[5]);
	and(w[5],SW[2],SW[4]);
	and(w[6],SW[0],SW[7]);
	and(w[7],SW[1],SW[6]);
	and(w[8],SW[2],SW[5]);
	and(w[9],SW[3],SW[4]);
	and(w[10],SW[1],SW[7]);
	and(w[11],SW[2],SW[6]);
	and(w[12],SW[3],SW[5]);
	and(w[13],SW[2],SW[7]);
	and(w[14],SW[3],SW[6]);
	and(w[15],SW[3],SW[7]);

	assign s[0]=w[0];
```

```
        full_adder f2(w[1], w[2], 0, c[1], s[1]);

        full_adder f3(w[3], w[4], c[1], c[2], w[16]);
        full_adder f4(w[6], w[7], c[2], c[3], w[17]);
        full_adder f5(0, w[10], c[3], w[18], w[19]);
        full_adder f6(w[16], w[5], 0, c[4], s[2]);

        full_adder f7(w[17], w[8], c[4], c[5], w[20]);
        full_adder f8(w[19], w[11], c[5], c[6], w[21]);
        full_adder f9(w[18], w[13], c[6], w[22], w[23]);
        full_adder f10(w[20], w[9], 0, c[7], s[3]);

        full_adder f11(w[21], w[12], c[7], c[8], s[4]);
        full_adder f12(w[23], w[14], c[8], c[9], s[5]);
        full_adder f13(w[22], w[15], c[9], s[7], s[6]);

        hexto7segment s5(s[7:4],HEX1);
        hexto7segment s6(s[3:0],HEX0);

endmodule

module full_adder(a,b,cin,cout,sout);
input a;
input b;
input cin;
output cout;
output sout;
wire C1, C2, S1;

process(a,b,C1,S1);
process(S1,cin,C2,sout);
assign cout=C1|C2;

endmodule

module process(x,y,C,S);
input x,y;
output C, S;
```

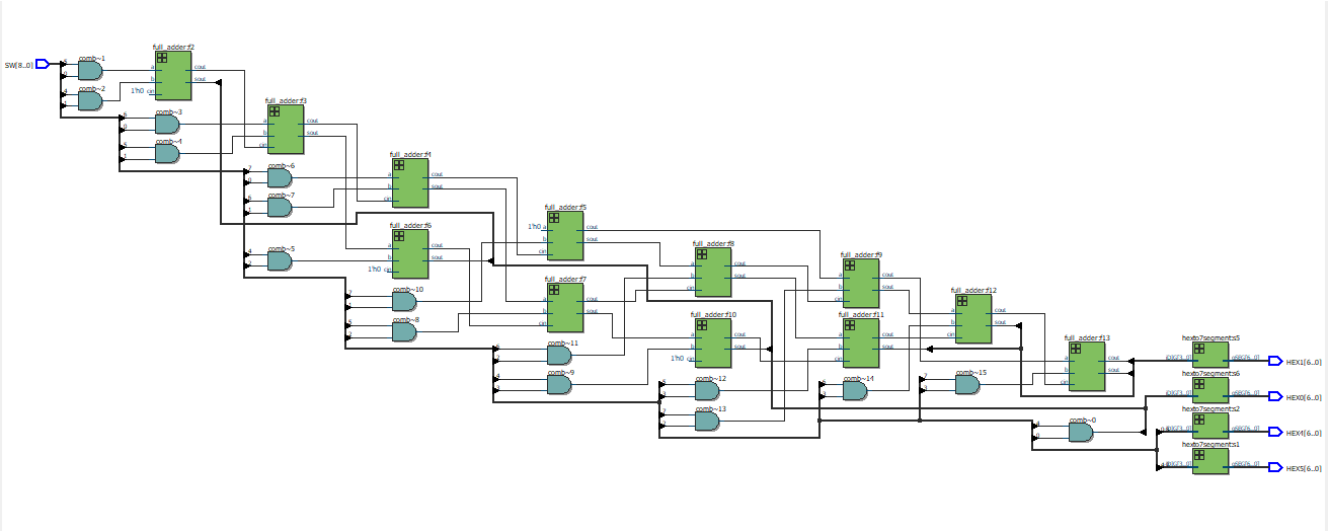
```
assign C=(x&y);
assign S=(~x&y)|(x&~y);

endmodule


module hexto7segment (
    input [3:0] iDIG,
    output reg [6:0] oSEG
        );

  always@(iDIG) begin
    case(iDIG)
        4'h1: oSEG = 7'b1111001;
      4'h2: oSEG = 7'b0100100;    // ---t----          4'h2: oSEG = 7'b0100100;    // |
|
        4'h3: oSEG = 7'b0110000;    // lt        rt
        4'h4: oSEG = 7'b0011001;    // |            |
        4'h5: oSEG = 7'b0010010;    // ---m----
        4'h6: oSEG = 7'b0000010;    // |            |
        4'h7: oSEG = 7'b1111000;    // lb        rb
        4'h8: oSEG = 7'b0000000;    // |            |
        4'h9: oSEG = 7'b0011000;    // ---b----
        4'ha: oSEG = 7'b0001000;
        4'hb: oSEG = 7'b0000011;
        4'hc: oSEG = 7'b1000110;
        4'hd: oSEG = 7'b0100001;
        4'he: oSEG = 7'b0000110;
        4'hf: oSEG = 7'b0001110;
        4'h0: oSEG = 7'b1000000;
    endcase
  end

endmodule
```

(2)實驗結果:





(3)RTL 布局



(4)問題與討論:

本次 lab 最需要思考的部分，同時還要考慮到進位、overflow 等問題，原本是直接將各個 output 單獨處理，結果產生了許 bug，最後使用 and 先將各組和運算出，再由小至大循序處理各個 output 值。

五、7-5
(1)實驗程式碼:

```
module labseven5(SW,HEX5,HEX4,HEX3,HEX2,HEX1,HEX0);
	input [8:0]SW;
	output [6:0]HEX0;
	output [6:0]HEX1;
	output [6:0]HEX2;
	output [6:0]HEX3;
	output [6:0]HEX4;
	output [6:0]HEX5;
	reg [7:0]A,B,C;
	wire [15:0]ans;
	wire [7:0] w1,w2,w3,w4,w5,w6,w7,w8,w9;
	wire c1,c2,c3;
	always@ (SW[8]) begin
		if(SW[8]==1) begin
			A<=SW[7:0];
			C<=SW[7:0];
		end
		else begin
			B<=SW[7:0];
			C<=SW[7:0];
		end
	end

	m4 mm1(A[3:0],B[3:0],w1[7:0]);
	m4 mm2(A[3:0],B[7:4],w2[7:0]);
	m4 mm3(A[7:4],B[3:0],w3[7:0]);
	m4 mm4(A[7:4],B[7:4],w4[7:0]);

	assign ans[3:0]=w1[3:0];

	full_8 fff1(w2[7:0],w3[7:0],0,c1,w5[7:0]);
	assign w6[3:0]=w1[7:4];
	assign w6[7:4]={4'b0000};
	full_8 fff2(w6[7:0],w5[7:0],0,c2,w7[7:0]);
	assign ans[7:4]=w7[3:0];
```

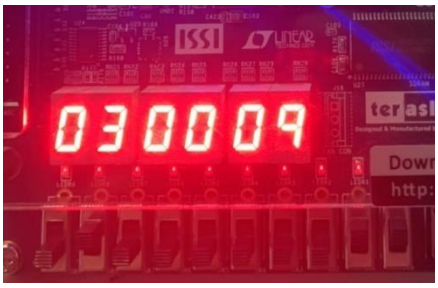
```
        assign w8[3:0]=w7[7:4];
        assign w8[7:5]={3'b0000};
        assign w8[4]=c1;
        full_8 fff3(w8[7:0],w4[7:0],0,c3,w9[7:0]);
        assign ans[15:8]=w9[7:0];

        hexto7segment s1(C[7:4],HEX5);
        hexto7segment s2(C[3:0],HEX4);
        hexto7segment s3(ans[15:12],HEX3);
        hexto7segment s4(ans[11:8],HEX2);
        hexto7segment s5(ans[7:4],HEX1);
        hexto7segment s6(ans[3:0],HEX0);

endmodule

module m4(a,b,s);
        input [3:0]a;
        input [3:0]b;
        output [7:0]s;
        wire [10:0] c;
        wire [23:0]w;
        and(w[0],a[0],b[0]);
        and(w[1],a[0],b[1]);
        and(w[2],a[1],b[0]);
        and(w[3],a[0],b[2]);
        and(w[4],a[1],b[1]);
        and(w[5],a[2],b[0]);
        and(w[6],a[0],b[3]);
        and(w[7],a[1],b[2]);
        and(w[8],a[2],b[1]);
        and(w[9],a[3],b[0]);
        and(w[10],a[1],b[3]);
        and(w[11],a[2],b[2]);
        and(w[12],a[3],b[1]);
        and(w[13],a[2],b[3]);
        and(w[14],a[3],b[2]);
        and(w[15],a[3],b[3]);
```

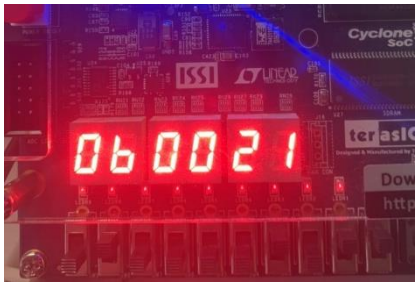
```
        assign s[0]=w[0];

        full_adder f2(w[1], w[2], 0, c[1], s[1]);

        full_adder f3(w[3], w[4], c[1], c[2], w[16]);
        full_adder f4(w[6], w[7], c[2], c[3], w[17]);
        full_adder f5(0, w[10], c[3], w[18], w[19]);
        full_adder f6(w[16], w[5], 0, c[4], s[2]);

        full_adder f7(w[17], w[8], c[4], c[5], w[20]);
        full_adder f8(w[19], w[11], c[5], c[6], w[21]);
        full_adder f9(w[18], w[13], c[6], w[22], w[23]);
        full_adder f10(w[20], w[9], 0, c[7], s[3]);

        full_adder f11(w[21], w[12], c[7], c[8], s[4]);
        full_adder f12(w[23], w[14], c[8], c[9], s[5]);
        full_adder f13(w[22], w[15], c[9], s[7], s[6]);
endmodule

module full_adder(a,b,cin,cout,sout);
input a;
input b;
input cin;
output cout;
output sout;
wire C1, C2, S1;

process(a,b,C1,S1);
process(S1,cin,C2,sout);
assign cout=C1|C2;

endmodule

module process(x,y,C,S);
input x,y;
output C, S;
```

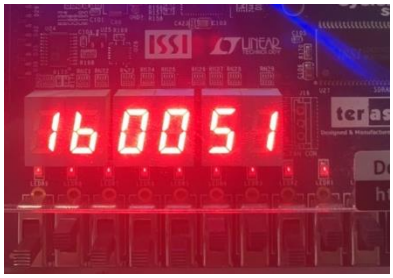
```
assign C=(x&y);
assign S=(~x&y)|(x&~y);

endmodule


module full_8(a,b,cin,cout,sout);
	input [7:0]a,b;
	input cin;
	output cout;
	output [7:0]sout;

	wire[7:0]c;
	full_adder ff1(a[0],b[0],cin,c[0],sout[0]);
	full_adder ff2(a[1],b[1],c[0],c[1],sout[1]);
	full_adder ff3(a[2],b[2],c[1],c[2],sout[2]);
	full_adder ff4(a[3],b[3],c[2],c[3],sout[3]);
	full_adder ff5(a[4],b[4],c[3],c[4],sout[4]);
	full_adder ff6(a[5],b[5],c[4],c[5],sout[5]);
	full_adder ff7(a[6],b[6],c[5],c[6],sout[6]);
	full_adder ff8(a[7],b[7],c[6],c[7],sout[7]);
	assign cout=c[7];
endmodule

module hexto7segment (
    input [3:0] iDIG,
    output reg [6:0] oSEG
      );

 always@(iDIG) begin
    case(iDIG)
       4'h1: oSEG = 7'b1111001;
      4'h2: oSEG = 7'b0100100;    // ---t----          4'h2: oSEG = 7'b0100100;    // |
|
       4'h3: oSEG = 7'b0110000;    // lt        rt
       4'h4: oSEG = 7'b0011001;    // |            |
       4'h5: oSEG = 7'b0010010;    // ---m----
       4'h6: oSEG = 7'b0000010;    // |            |
```

```
        4'h7: oSEG = 7'b1111000;    // lb       rb
        4'h8: oSEG = 7'b0000000;    // |          |
        4'h9: oSEG = 7'b0011000;    // ---b----
        4'ha: oSEG = 7'b0001000;
        4'hb: oSEG = 7'b0000011;
        4'hc: oSEG = 7'b1000110;
        4'hd: oSEG = 7'b0100001;
        4'he: oSEG = 7'b0000110;
        4'hf: oSEG = 7'b0001110;
        4'h0: oSEG = 7'b1000000;
    endcase
 end

endmodule
```

(2)實驗結果:

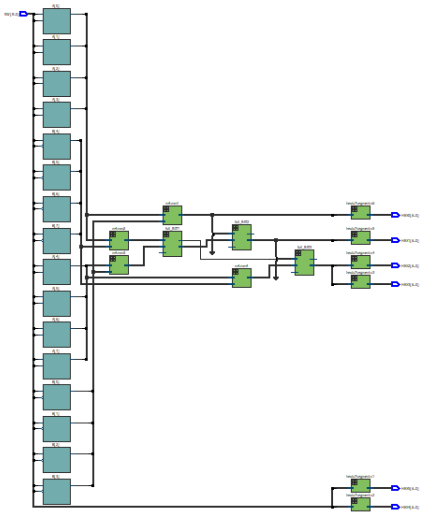






//A=3, B=3 or 11 or 27

(3)RTL 布局:



(4)問題與討論:

非常複雜且困難且難以理解，整體上延續上部分乘法器，但變成 8 bit，因此要由 4 個 4 bit input 彼此相乘並考慮 overflow，非常複雜不過學到更多如何將大問題拆成小問題的技巧。

六、7-6

(1)實驗程式碼:

主程式碼:

```
module labseven6(SW,KEY,HEX3,HEX2,HEX1,HEX0,LEDR);
	input [9:0] SW;
	input [3:0] KEY;
	output [6:0] HEX3, HEX2, HEX1, HEX0;
	output [9:9] LEDR;

	wire cout;
	reg [7:0] a,b,c,d;
	wire [15:0] ab,cd,out;

	reg [15:0] ans;

	always@(posedge KEY[1], negedge KEY[0]) begin
	if(KEY[0]==0) begin
		a<=0;
		b<=0;
		c<=0;
		d<=0;
	end
	else begin
	if(SW[9] == 1) begin
		if(SW[8] == 0) begin
			if( KEY[2] == 0)
				a[7:0] <= SW[7:0];
			else
				b[7:0] <= SW[7:0];
		end
		else begin
			if(KEY[2] == 1)
				c[7:0] <= SW[7:0];
			else
				d[7:0] <= SW[7:0];
		end
		end
```

```
        end
        end


        always@(KEY[3]) begin
        if(SW[8]==0 && KEY[3]==1) begin
                ans[15:8]=a[7:0];
                ans[7:0]=b[7:0];
        end
        else if(SW[8]==1 && KEY[3]==1) begin
                ans[15:8]=c[7:0];
                ans[7:0]=d[7:0];
        end
        else if(KEY[3] == 0) begin
                ans[15:0] = out;
        end
        end
        mul(a,b,ab);
        mul(c,d,cd);
        add (ab,cd,cout,out);

        hexto7segment h1(ans[15:12],HEX3);
        hexto7segment h2(ans[11:8],HEX2);
        hexto7segment h3(ans[7:4],HEX1);
        hexto7segment h4(ans[3:0],HEX0);
        assign LEDR[9] = cout;
endmodule


module hexto7segment (
    input [3:0] iDIG,
    output reg [6:0] oSEG
       );

  always@(iDIG) begin
    case(iDIG)
      4'h1: oSEG = 7'b1111001;
     4'h2: oSEG = 7'b0100100;    // ---t----          4'h2: oSEG = 7'b0100100;    // |
```

```
|
        4'h3: oSEG = 7'b0110000;    // lt       rt
        4'h4: oSEG = 7'b0011001;    // |          |
        4'h5: oSEG = 7'b0010010;    // ---m----
        4'h6: oSEG = 7'b0000010;    // |          |
        4'h7: oSEG = 7'b1111000;    // lb       rb
        4'h8: oSEG = 7'b0000000;    // |          |
        4'h9: oSEG = 7'b0011000;    // ---b----
        4'ha: oSEG = 7'b0001000;
        4'hb: oSEG = 7'b0000011;
        4'hc: oSEG = 7'b1000110;
        4'hd: oSEG = 7'b0100001;
        4'he: oSEG = 7'b0000110;
        4'hf: oSEG = 7'b0001110;
        4'h0: oSEG = 7'b1000000;
    endcase
 end

endmodule
```

MegaWizard:

```
module mul (
        dataa,
        datab,
        result);

        input       [7:0]    dataa;
        input       [7:0]    datab;
        output      [15:0]    result;

        wire [15:0] sub_wire0;
        wire [15:0] result = sub_wire0[15:0];

        lpm_multlpm_mult_component (
                        .dataa (dataa),
                        .datab (datab),
```

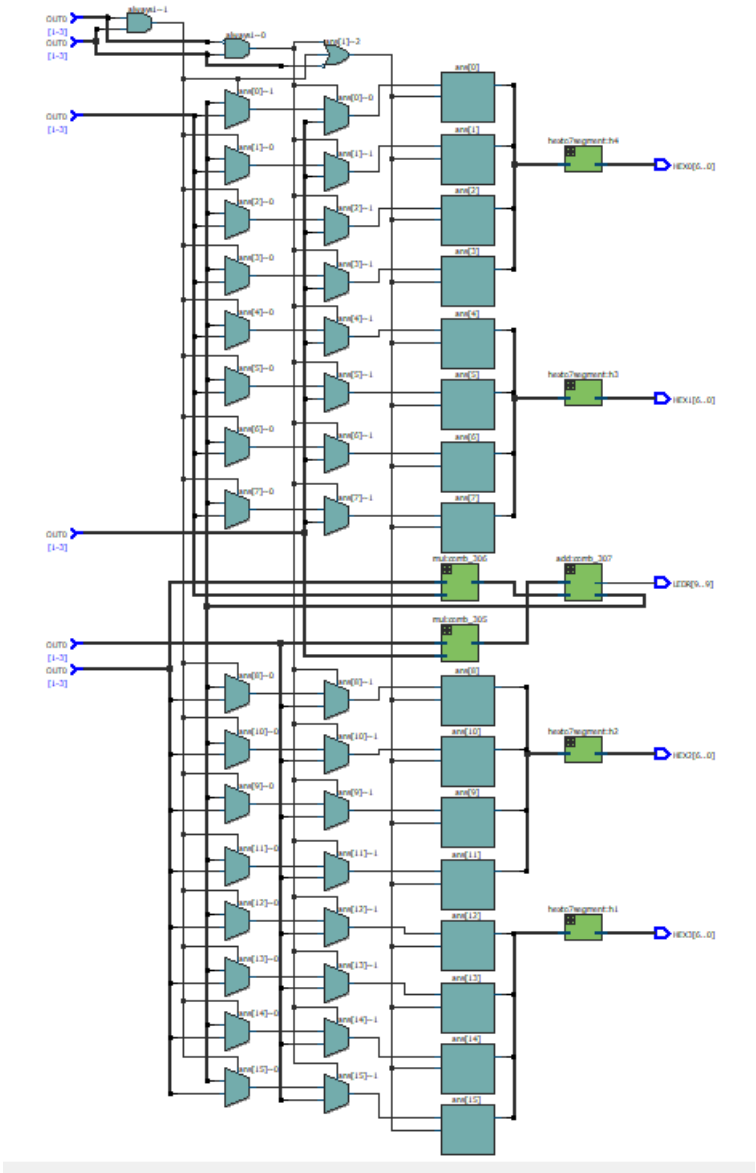
```
                .result (sub_wire0),
                .aclr (1'b0),
                .clken (1'b1),
                .clock (1'b0),
                .sum (1'b0));
    defparam
        lpm_mult_component.lpm_hint = "MAXIMIZE_SPEED=5",
        lpm_mult_component.lpm_representation = "UNSIGNED",
        lpm_mult_component.lpm_type = "LPM_MULT",
        lpm_mult_component.lpm_widtha = 8,
        lpm_mult_component.lpm_widthb = 8,
        lpm_mult_component.lpm_widthp = 16;


endmodule

module add (
    dataa,
    datab,
    overflow,
    result);

    input    [15:0]  dataa;
    input    [15:0]  datab;
    output      overflow;
    output   [15:0]  result;

    wire    sub_wire0;
    wire [15:0] sub_wire1;
    wire    overflow = sub_wire0;
    wire [15:0] result = sub_wire1[15:0];

    lpm_add_sub  LPM_ADD_SUB_component (
                .dataa (dataa),
                .datab (datab),
                .overflow (sub_wire0),
                .result (sub_wire1)
                // synopsys translate_off
```

```
                        ,
                        .aclr (),
                        .add_sub (),
                        .cin (),
                        .clken (),
                        .clock (),
                        .cout ()
                        // synopsys translate_on
                        );
    defparam
        LPM_ADD_SUB_component.lpm_direction = "ADD",
        LPM_ADD_SUB_component.lpm_hint =
"ONE_INPUT_IS_CONSTANT=NO,CIN_USED=NO",
        LPM_ADD_SUB_component.lpm_representation = "UNSIGNED",
        LPM_ADD_SUB_component.lpm_type = "LPM_ADD_SUB",
        LPM_ADD_SUB_component.lpm_width = 16;


endmodule
```

(2)實驗結果:

//A=3, B=3, C=5, D=4, S=29

(3)RTL 布局



(4)問題與討論:

作法上就是把 A、B 和 C、D 利用乘法器運算後相加，但因為設定 4 個參數時搭配 KEY 有點複雜，所以耗時許久，但整體上算簡單易懂。