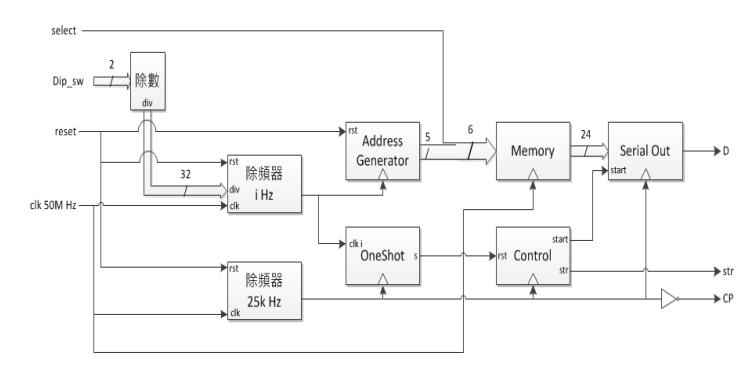
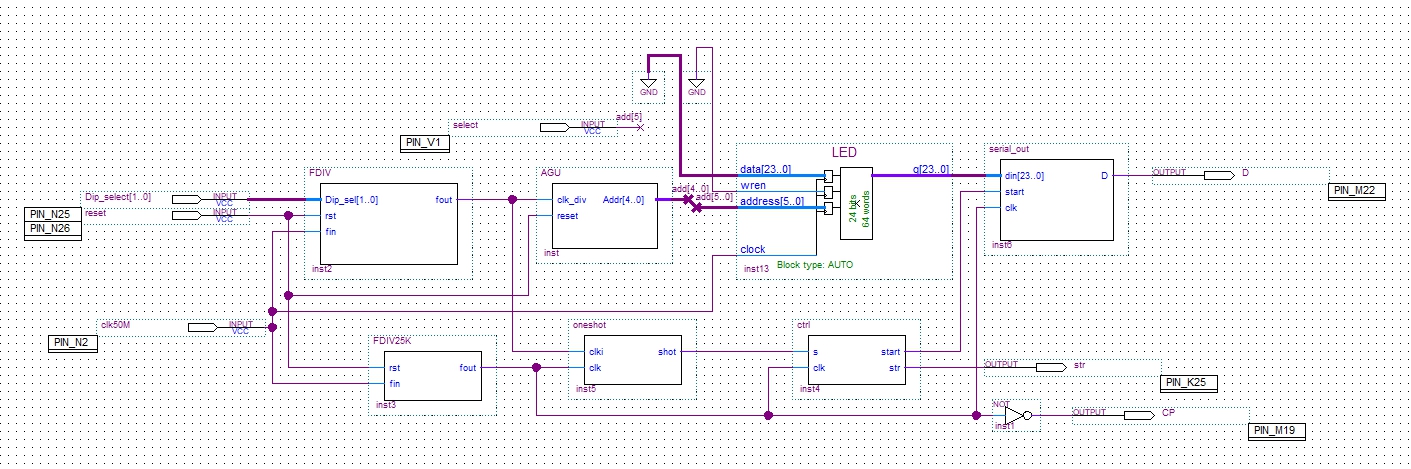
**數位實驗5：彩色 LED 燈控制**

**第十四組**

**許銘森B063011058**

**整體架構圖**





**各模組的Verilog code和波形模擬**

1.除頻器 50M（有除數表）：

module FDIV(Dip\_sel,rst,fin,fout);

input [1:0] Dip\_sel;

input fin,rst;

output fout;

wire [31:0] \_DIVN;

reg [31:0] count,DIVN;

reg fout;

always @(Dip\_sel)

case(Dip\_sel)

除數表

2'b00: DIVN=32'd25000000;

2'b01: DIVN=32'd12500000;

2'b10: DIVN=32'd8333333;

2'b11: DIVN=32'd6250000;

endcase

assign \_DIVN = DIVN >>1;

always @(posedge fin)

if(count>=DIVN-1)

count=0;

else

count = count +1;

always @ (posedge fin or posedge rst)

if(rst)

fout=0;

else if(count>=\_DIVN)

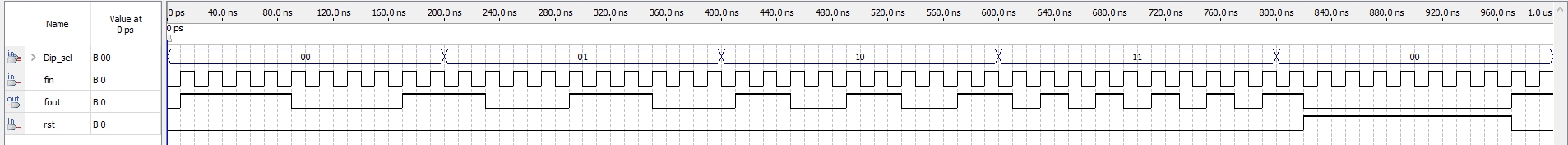
fout=0;

else

fout=1'd1;

endmodule

**註解:**將版子上的 50M Hz 轉成 2Hz、4Hz、6Hz 及 8Hz，為切換 memory 所儲存之圖騰的速度。



解釋：fin為輸入頻率，我設了50M與實驗一樣，我把DIVN分別設為8，4，6，2，。隨著Dip\_sel增加，DIVN越小，輸出頻率越高，模組有正常運作。

2.除頻器 25KHz：

module FDIV25K(rst,fin,fout);

input fin,rst;

output fout;

wire [31:0] \_DIVN,DIVN;

reg [31:0] count;

reg fout;

assign DIVN=2000;

assign \_DIVN = DIVN >>1;

always @(posedge fin)

if(count>=DIVN-1)

count=0;

else

count = count +1;

always @ (posedge fin or posedge rst)

if(rst)

fout=0;

else if(count>=\_DIVN)

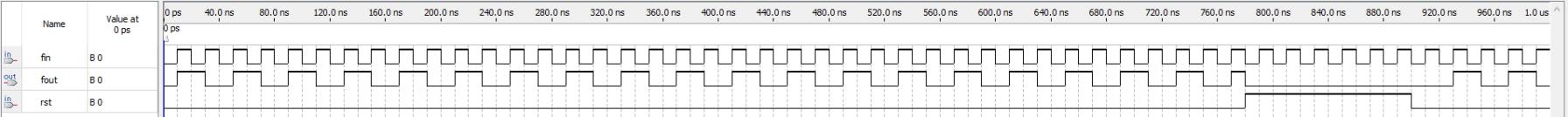
fout=0;

else

fout=1'd1;

endmodule

**註解:**將版子上的 50M Hz 轉成 25kHz，以供 IC 使用。



解釋：fin為輸入頻率，我設了50M與實驗一樣，把DIVN設為2，rst為1時，fout停止輸出，模組有正常運作。

1. Address Generator：

module AGU(clk\_div,reset,Addr);

input clk\_div,reset;

output [4:0]Addr;

reg [4:0]Addr;

always@(posedge clk\_div or posedge reset)

begin

if(reset)

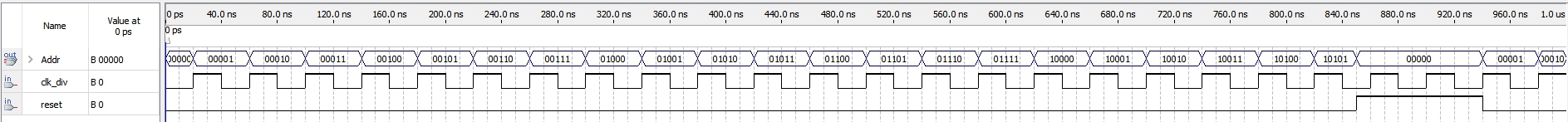
Addr <=5'd0;

else

Addr <=Addr+5'd1;

end

Endmodule

**註解:**產生 memory 所需的 5bits 位址值，當 rst 拉起時， 位址歸零，使 memroy 可以從頭開始輸出資料。

解釋：當 rst 拉起時， 位址歸零，使 memroy 從頭開始輸出資料。模組有正常運作。

1. **Serial Out：**

module serial\_out(din,start,clk,D);

input start,clk;

input [23:0] din;

reg [23:0] store;

output wire D;

always @(posedge clk or posedge start)

begin

if(start)

store = din;

else

begin

進行輸出移位

store[23] = store[22];

store[22] = store[21];

store[21] = store[20];

store[20] = store[19];

store[19] = store[18];

store[18] = store[17];

store[17] = store[16];

store[16] = store[15];

store[15] = store[14];

store[14] = store[13];

store[13] = store[12];

store[12] = store[11];

store[11] = store[10];

store[10] = store[9];

store[9] = store[8];

store[8] = store[7];

store[7] = store[6];

store[6] = store[5];

store[5] = store[4];

store[4] = store[3];

store[3] = store[2];

store[2] = store[1];

store[1] = store[0];

end

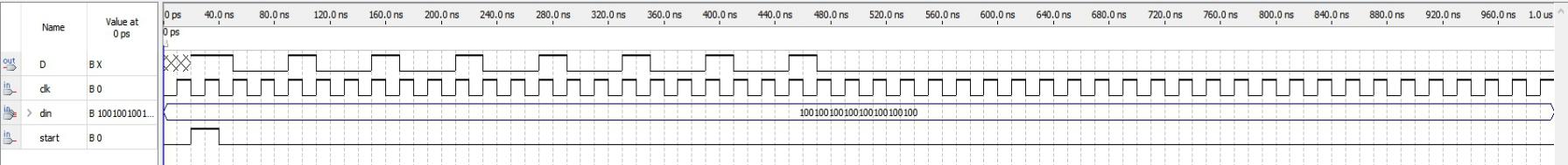
end

assign D = store[23];

endmodule

**註解:**輸入 start = 1 時，將輸入之 24bit 值存入暫存器內，並在

start = 0 之後每一個 clock 正緣時由暫存器之最高位元依序送出至 D。



解釋：輸入 start = 1 時，將din（100100100100100100100100）之 24bit 值存入store內，並在 start = 0 之後每一個 clock 正緣時由store之最高位元依序 送出至 D，依序為100，100，100，100，100，100，100，100。

1. Oneshot：

module oneshot(clki,clk,shot);

input clki,clk;

output shot;

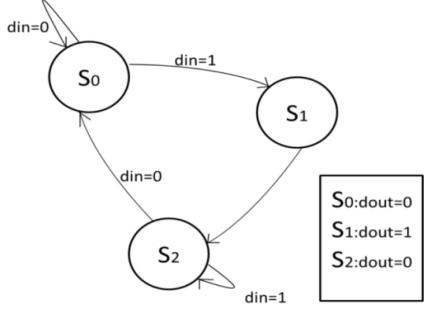
reg shot;

reg [1:0]cs,ns;

always@(posedge clk)

cs<=ns;

always@(cs)

case(cs)

2'd0:shot<=1'b0;

每個state要進行的變化

2'd1:shot<=1'b1;

2'd2:shot<=1'b0;

default:shot<=1'b0;

endcase

always@(cs)

case(cs)

state 變化條件

2'd0:ns<=(clki)?2'd1:2'd0;

2'd1:ns<=2'd2;

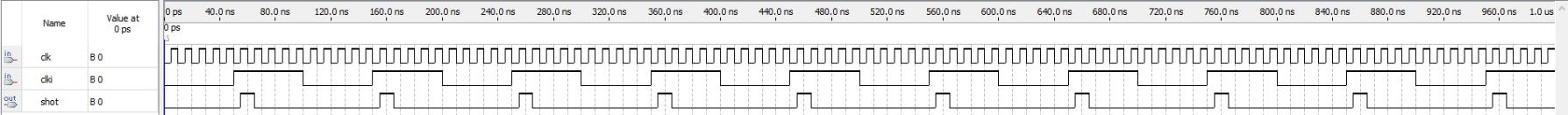
2'd2:ns<=(clki)?2'd2:2'd0;

default:ns<=2'd0;

endcase

endmodule

**註解:**利用 state machine 設計，做出能在較長的輸入訊號正緣觸發後，產生一個長度 1clk(25kHz)之輸出訊號。



解釋：在較長的輸入cklki訊號正緣觸發後，產生一個長度 1clk之輸出訊號。

1. **Control：**

module ctrl(s,clk,start,str);

input s,clk;

output start,str;

reg[4:0] count;

reg [2:0]state,nstate;

always@(posedge clk)

state<=nstate;

always@(state)

case(state)

2'd0:nstate=(s)?2'd1:2'd0;

2'd1:nstate=2'd2;

2'd2:nstate=(count==5'd25)?2'd0:2'd2;

endcase

always@(posedge clk)

begin

if(state==2'd2)

count=count+1;

else

count=5'd0;

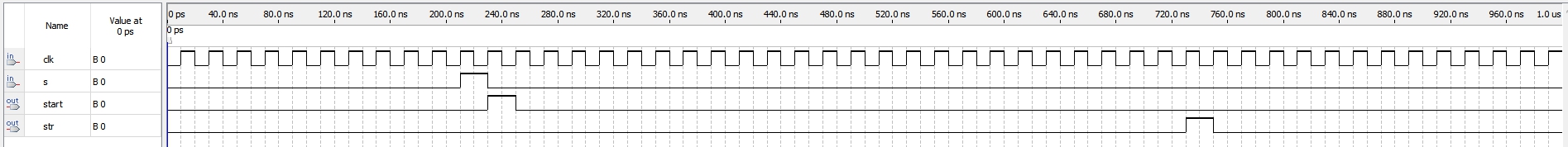
end

assign start=(state==2'd1)?1'd1:1'd0;

assign str=(count==5'd24)?1'd1:1'd0;

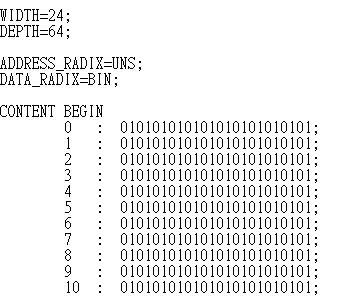
endmodule

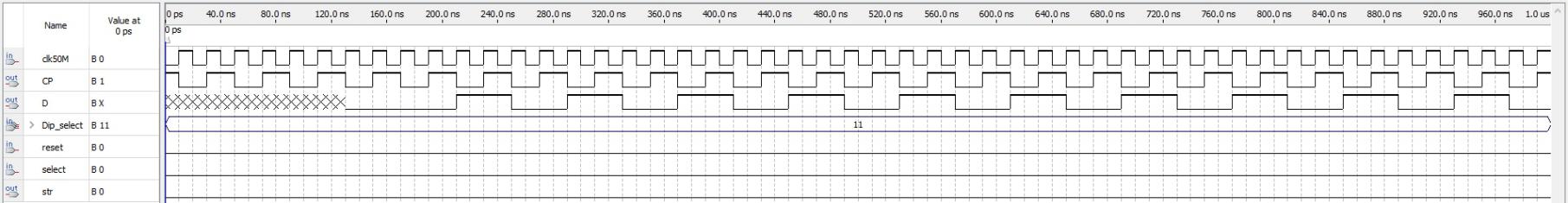
**註解:**依照 oneshot 輸出(s)產生兩個訊號，在 s 拉起時，輸出長度為一個 clk 的 start 訊號，並在 start 結束之後的第 24 個 clk 時，輸出長度為一個 clk 的 str 訊 號。



**解釋：**在 s 拉起時，輸出長度為一個 clk 的 start 訊號，

並在 start 結束之後的第 24 個 clk 時，輸出長度為一個 clk 的 str 訊 號。

**7.BDF檔案**

****

**實驗心得**

這次實驗code和上次一樣，只是把模組位元數改一改。有利用到基本的除頻器和上次學到的模組ONESHOT及CONTROL。用到了state machine的觀念。這個觀念對於寫CONTROL模組十分便利，把狀態圖畫出來，便能清晰地把我們想要的功能計劃出來。便利的另一原因是可以改變數達到想要的效果，以這次實驗為例：

assign str=(count==5'd24)?1'd1:1'd0;

在 start 結束之後的第 24 個 clk 時，輸出長度為一個 clk 的 str 訊號。

變數改一改便可以調整str訊號的輸出時候。

還有，這次也需要事先把電路焊接出來，因爲上次的經驗，電路的表現有改善。但也會因爲焊接不穩定，有LED的顔色沒有亮，導致輸出結果未如理想。總括而言，這次實驗很好玩，亮法可以調整，code也會打了，焊接也穩定。