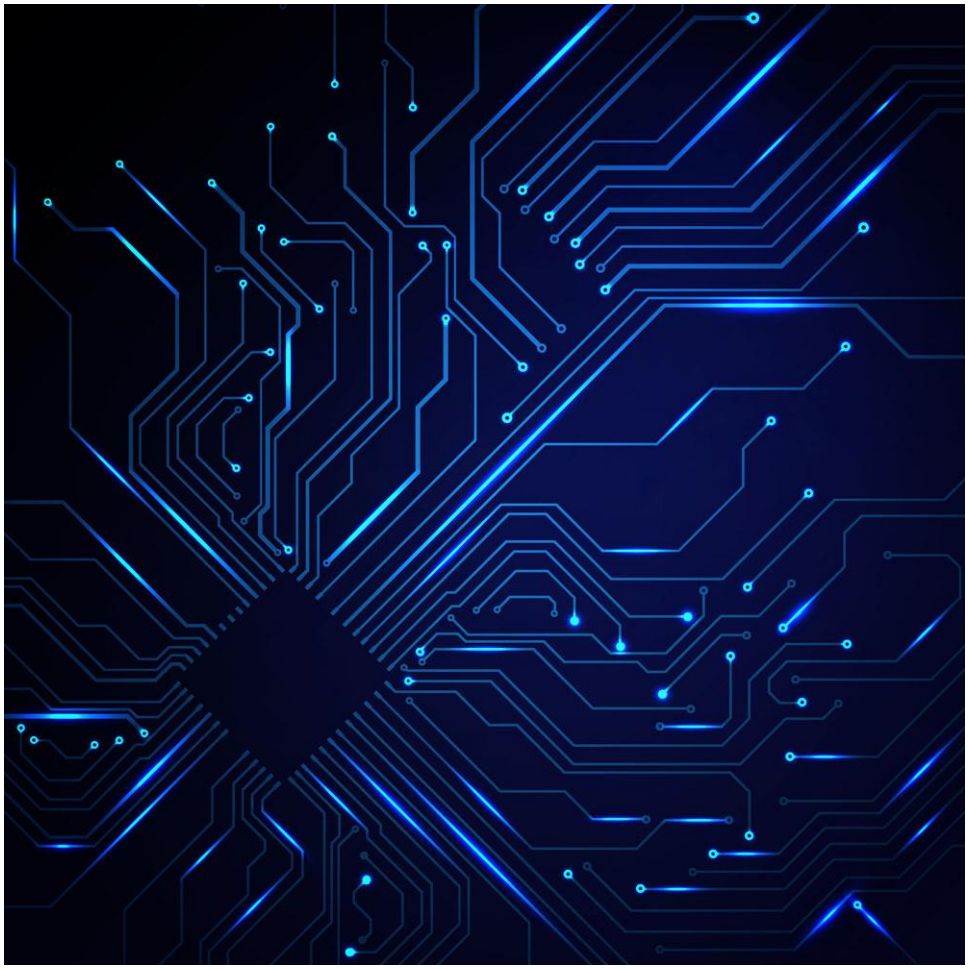


2024/XX/XX

實驗八

姓名：黃文祺　　學號：01057013

班級：資工 3A

E-mail：OOOOOOOO

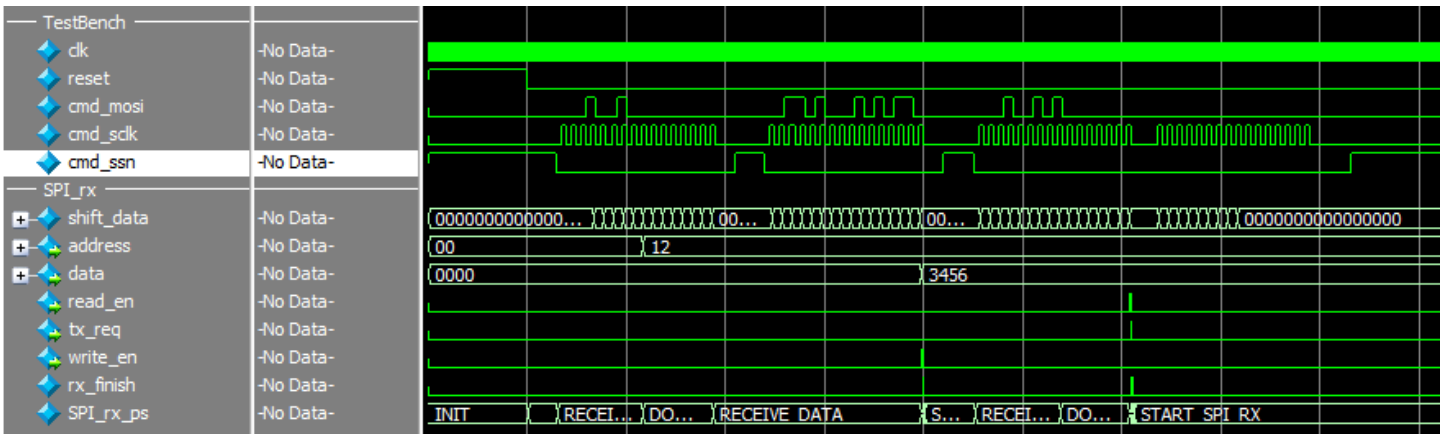
注意

1. 繳交時一律轉 PDF 檔
2. 繳交期限為
   隔週三上午九點
3. 一人繳交一份
4. 檔名：學號_HW?.pdf
   檔名請按照作業檔名格式進行填寫
   未依照格式不予批改

SPI Slave Reciever

- 實驗說明：
  1. 寫出 SPI Slave Reciever，並顯示其模擬圖。
  2. testbench.sv 會呼叫 DE0_CV.sv，請在 DE0_CV.sv 中完成程式碼撰寫。
  3. DE0_CV.sv 使用接腳：
     - CLOCK_50：50MHz 的 clk 訊號。
     - RESET_N：系統 reset，為 0 時重置系統。
     - GPIO_0[0]：傳訊號進 SPI Slave 的 mosi。
     - GPIO_0[1]：傳訊號進 SPI Slave 的 sclk。
     - GPIO_0[2]：傳訊號進 SPI Slave 的 ssn。
     - GPIO_0[3]：接收 SPI Slave 的 miso 訊號。
  4. SPI_rx.sv 輸入:
     - clk (請將 DE0_CV 的 CLOCK_50，用 PLL 升至 100MHz)
     - mosi
     - sclk(testbench 已設定為 10MHz)
     - ssn
     - reset
  5. 輸出:
     - address [7:0]
     - data [15:0]
     - read_en
     - write_en
     - tx_req

■ 波型圖參考


TestBench
clk
reset
cmd_mosi
cmd_sclk
cmd_ssn
SPI_rx
shift_data
address
data
read_en
tx_req
write_en
rx_finish
SPI_rx_ps
-No Data-
00000000000000...
00...
00...
00000000000000000
00
12
0000
3456
INIT
RECEI...
DO...
RECEIVE DATA
S...
RECEI...
DO...
START SPI RX

■ 系統架構程式碼與程式碼說明

截圖請善用 win+shift+S

SPI_rx.sv:

C:\Users\user\Desktop\DE0_CV_SPI\design\SPI_rx.sv - Notepad++

檔案(F) 編輯(E) 搜尋(S) 檢視(V) 編碼(N) 語言(L) 設定(T) 工具(O) 巨集(M) 執行(R) 外掛(P) 視窗(W) ?

SPI_rx.sv

```
module SPI_rx(
    input               mosi,
    input               sclk,           //10MHz
    input               ssn,
    input               clk,            //100MHz
    input               rst,            //reset=1時, 系統重製

    output logic [7:0]  address,        //8 bits address (0-255)
    output logic [15:0] data,           //16 bits data
    output logic        read_en,
    output logic        write_en,
    output logic        tx_req
    );

    logic s_signal_ssn;
    logic d_signal_ssn;
    logic s_signal_sclk;
    logic d_signal_sclk;
    logic ssn_negedge;
    logic sclk_posedge;

    logic cnt_rst;
    logic command;
    logic addr_load;
    logic command_load;
    logic pkg_complete;
    logic rx_finish;

    logic [15:0] shift_data;
    logic [31:0] receive_bit_counter;
    logic [15:0] read_data;
    logic [15:0] write_data;

    logic [15:0] reg_file [255:0];

    typedef enum {INIT, START, RECEIVE_ADDR,RECEIVE_COMMAND, PKG_COMPLETE, CHK_COMMAND, RECEIVE_DATA, TX_REQ, WRITE, FINISH
    } rx_state;

    rx_state SPI_rx_ps, SPI_rx_ns;

//ssn negedge detector
always_ff @(posedge clk or posedge rst) begin
    if(rst) begin
        s_signal_ssn <= 1;
        d_signal_ssn <= 1;
        ssn_negedge  <= 0;
    end
    else begin
        {d_signal_ssn, s_signal_ssn} <= {s_signal_ssn, ssn};
        ssn_negedge <= ~s_signal_ssn & d_signal_ssn;
    end
end

//sclk posedge detector
always_ff @(posedge clk or posedge rst) begin
    if(rst) begin
        s_signal_sclk <= 1;     //
        d_signal_sclk <= 1;
        sclk_posedge  <= 0;
    end
    else begin
        {d_signal_sclk, s_signal_sclk} <= {s_signal_sclk, sclk};
        sclk_posedge <= s_signal_sclk & ~d_signal_sclk;
    end
end

//left shift register
always_ff@ (posedge clk or posedge rst) begin
    if  (rst|ssn) begin
        shift_data[15:0] <= 0;
    end
    else if(sclk_posedge) begin
        shift_data[15:0] <= {shift_data[14:0], mosi};
    end
end

//receive package shift counter
always_ff@ (posedge clk or posedge rst) begin
    if(rst | cnt_rst| ssn) begin
        receive_bit_counter[31:0] <= 0;
    end
    else if(sclk_posedge) begin
        receive_bit_counter <= receive_bit_counter + 1;
    end
end
```

```
87    //receive addr of package
88    always_ff@ (posedge clk or posedge rst) begin
89        if(rst) begin
90            address <= 8'h00;
91        end
92        else if(addr_load) begin
93            address <= shift_data[7:0];
94        end
95    end
96
97    //receive command of package
98    always_ff@ (posedge clk or posedge rst) begin
99        if(rst) begin
100           command <= 1'bx;
101       end
102       else if(command_load) begin
103           command <= shift_data[0];
104       end
105   end
106
107   //access data in register file
108   always_ff@ (posedge clk or posedge rst) begin
109       if(rst) begin
110           write_data <= 16'h0000;
111           read_data  <= 16'h0000;
112           data       <= 16'h0000;
113       end
114       else if(write_en) begin
115           write_data        <= shift_data[15:0];
116           reg_file[address] <= shift_data[15:0];
117           data  <= shift_data[15:0];
118       end
119       else if(read_en) begin
120           read_data <= reg_file[address];
121           //data    <= reg_file[address];
122       end
123   end
124
125   //fsm
126   always_ff@ (posedge clk or posedge rst) begin
127       if(rst|ssn) begin
128           SPI_rx_ps <= INIT;
129       end
130       else begin
131           SPI_rx_ps <= SPI_rx_ns;
132       end
133   end
134
135   //controller
136   always_comb begin
137       read_en      = 0;
138       write_en     = 0;
139       tx_req       = 0;
140       cnt_rst      = 0;
141       addr_load    = 0;
142       command_load = 0;
143       pkg_complete = 0;
144       rx_finish    = 0;
145
146       SPI_rx_ns = SPI_rx_ps;
147
148       case(SPI_rx_ps)
149           INIT: begin
150               SPI_rx_ns = START;
151           end
152
153           START: begin
154               if(ssn_negedge) begin
155                   cnt_rst = 1;
156                   SPI_rx_ns   = RECEIVE_ADDR;
157               end
158           end
159
160           RECEIVE_ADDR: begin
161               if(receive_bit_counter == 8) begin
162                   addr_load = 1;
163               end
164               if(receive_bit_counter >= 8) begin
165                   SPI_rx_ns      = RECEIVE_COMMAND;
166               end
167           end
168
169           RECEIVE_COMMAND: begin
170               if(receive_bit_counter == 9)  begin
171                   command_load = 1;
172               end
173               if(receive_bit_counter >= 16) begin
174                   SPI_rx_ns = PKG_COMPLETE;
175               end
176           end
177
```

```
        PKG_COMPLETE: begin
            cnt_rst      = 1;
            pkg_complete = 1;
            SPI_rx_ns        = CHK_COMMAND;
        end

        CHK_COMMAND: begin
            if(command) begin
                SPI_rx_ns = TX_REQ;
            end
            else begin
                SPI_rx_ns = RECEIVE_DATA;
            end
        end

        RECEIVE_DATA: begin
            if(receive_bit_counter >= 16) begin
                SPI_rx_ns = WRITE;
            end
        end

        TX_REQ: begin
            tx_req  = 1;
            read_en = 1;
            SPI_rx_ns  = FINISH;
        end

        WRITE: begin
            write_en = 1;
            SPI_rx_ns    = FINISH;
        end

        FINISH: begin
            rx_finish = 1;
            SPI_rx_ns     = INIT;
        end

    endcase
end
endmodule
```

DE0_CV.sv: 使用接腳

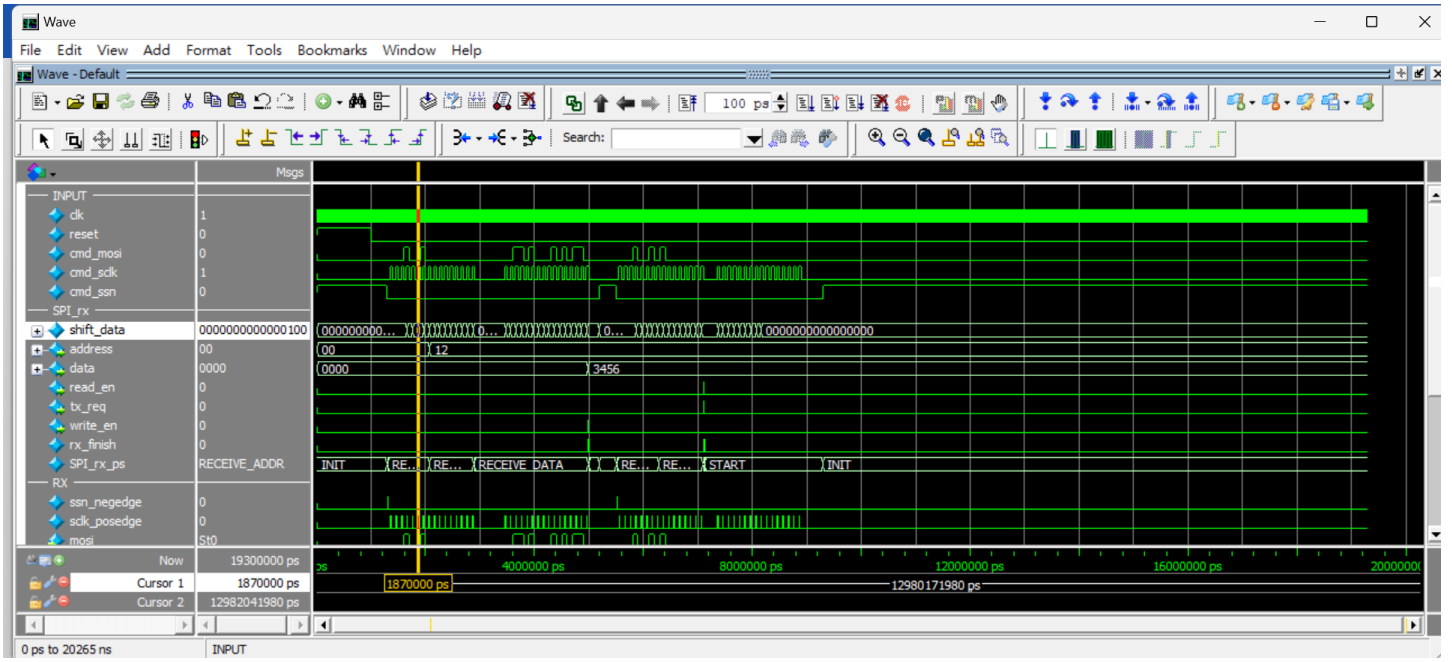
C:\Users\user\Desktop\DE0_CV_SPI\design\DE0_CV.sv - Notepad++

檔案(F) 編輯(E) 搜尋(S) 檢視(V) 編碼(N) 語言(L) 設定(T) 工具(O) 巨集(M) 執行(R) 外掛(P) 視窗(W) ?

SPI_rx.sv | DE0_CV.sv

```
    //  Structural coding
    //=======================================================


    assign mosi             =   GPIO_0[0];
    assign sclk             =   GPIO_0[1];
    assign ssn              =   GPIO_0[2];
    assign GPIO_0[3]        =   miso;
    assign LEDR         =   data_debug;

    /*請使用PLL IP
        輸入：CLOCK_50
        輸出：CLK_100MHz
    */
    pll pll(
        .refclk(CLOCK_50),
        .rst(~RESET_N),
        .outclk_0(CLK_100MHz),
        .locked()
    );

    SPI SPI(
        //------output----------------------------------
        .miso               (miso),
        .data_debug         (data_debug),

        //------input-----------------------------------
        .mosi               (mosi),
        .sclk               (sclk),
        .ssn                (ssn),
        .clk                (CLK_100MHz),
        .reset              (~RESET_N)
);

```

■ 模擬結果與結果說明：



■ 結論與心得：

這次作業跟之前 rs232 最不一樣的地方是同步與非同步的差異，然後還有複習了之前 pll 的頻率改變，上次上課遲到了一個半小時，有些東西沒聽到，回來才跟同學一起討論，討論前自己看 ppt，有些東西自己看真的看不太懂，尤其是 ssn 和 sclk，不過後來理解後大概就 ok 了，