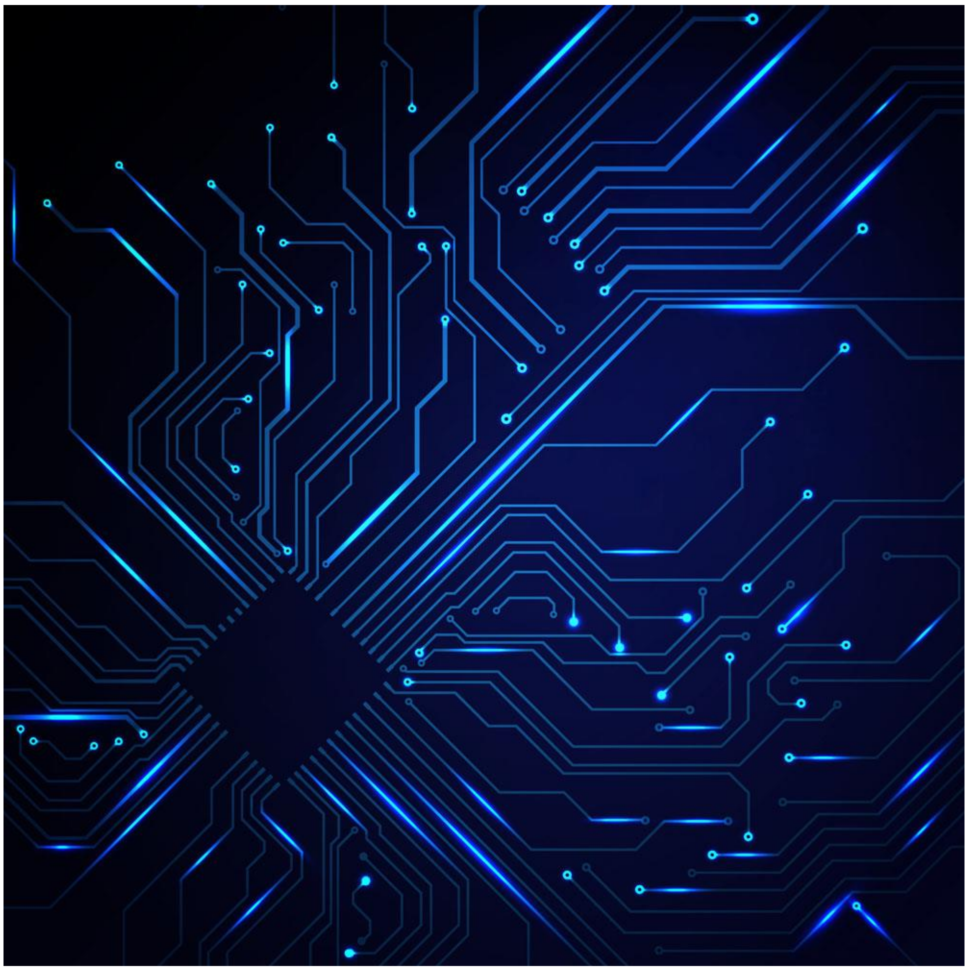


2024/XX/XX

實驗十

姓名：黃文祺　　　學號：01057013

班級：資工 3A

E-mail：OOOOOOOO

注意

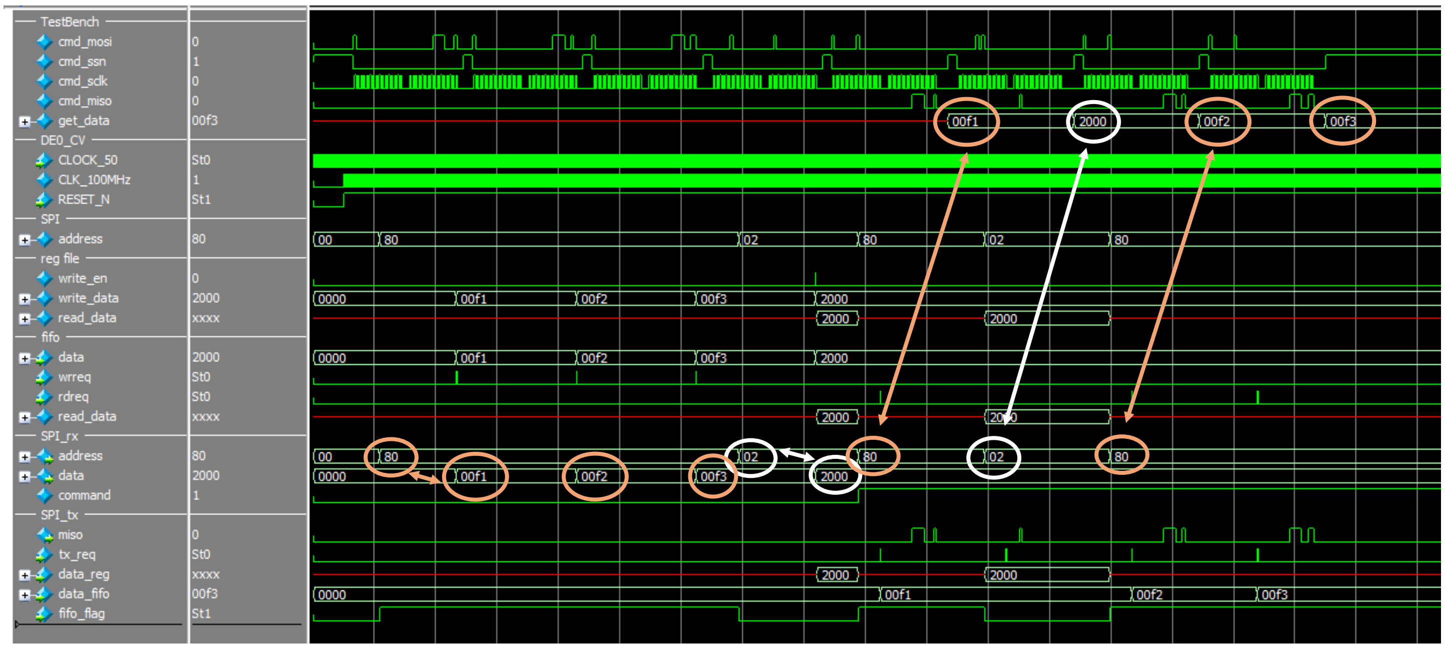
1. 繳交時一律轉 PDF 檔
2. 繳交期限為
   隔週三上午九點
3. 一人繳交一份
4. 檔名：學號_HW?.pdf
   檔名請按照作業檔名格式進行填寫
   未依照格式不予批改

SPI Slave

- ■ 實驗說明：
  1. 將 SPI Slave 結合 FIFO，並顯示其模擬圖。
  2. SPI Slave 功能說明
     - 若位址最高位為 1，由 FIFO 讀寫資料。
     - 若位址最高位為 0，由 Register File 讀寫資料。
  3. testbench.sv 會呼叫 DE0_CV.sv，請在 DE0_CV.sv 中完成程式碼撰寫。
  4. 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。
  5. SPI.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
  6. SPI.sv 輸出:
     - miso

■ 波型圖參考

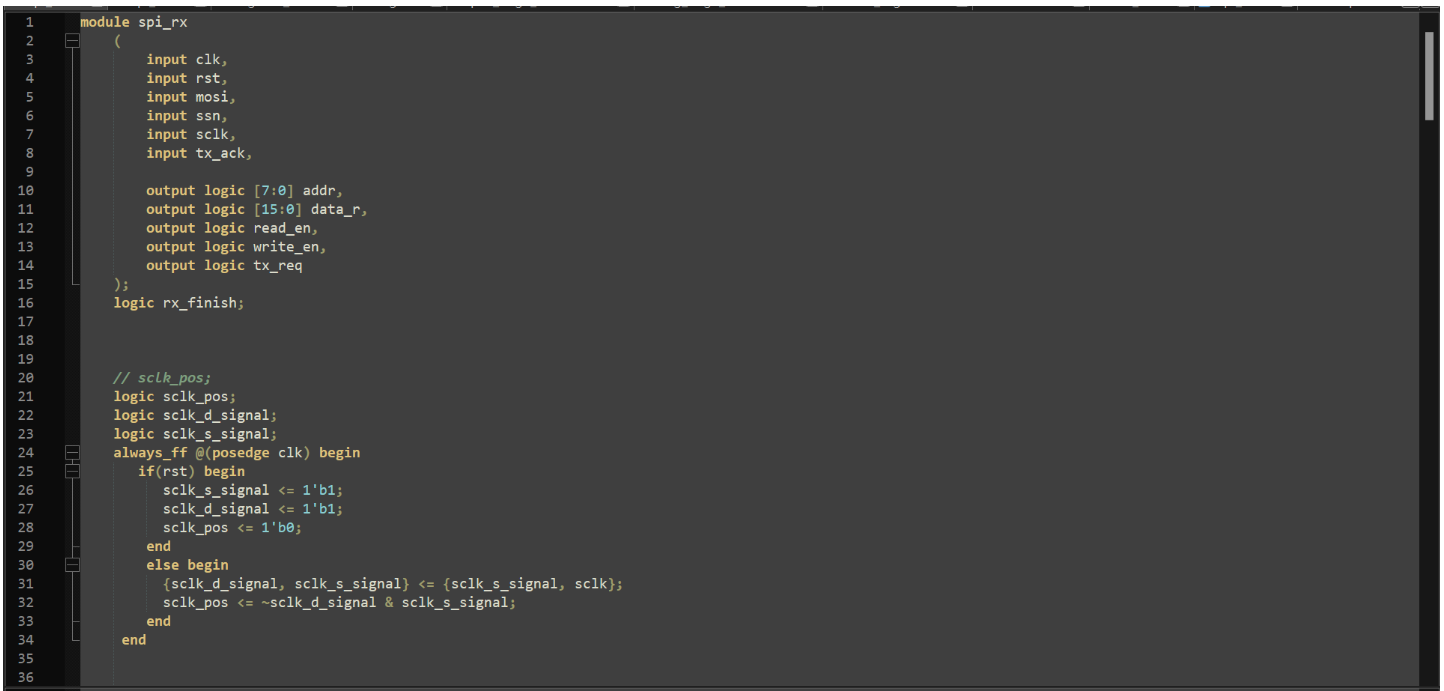
- 務必擷取到 get_data 的波型



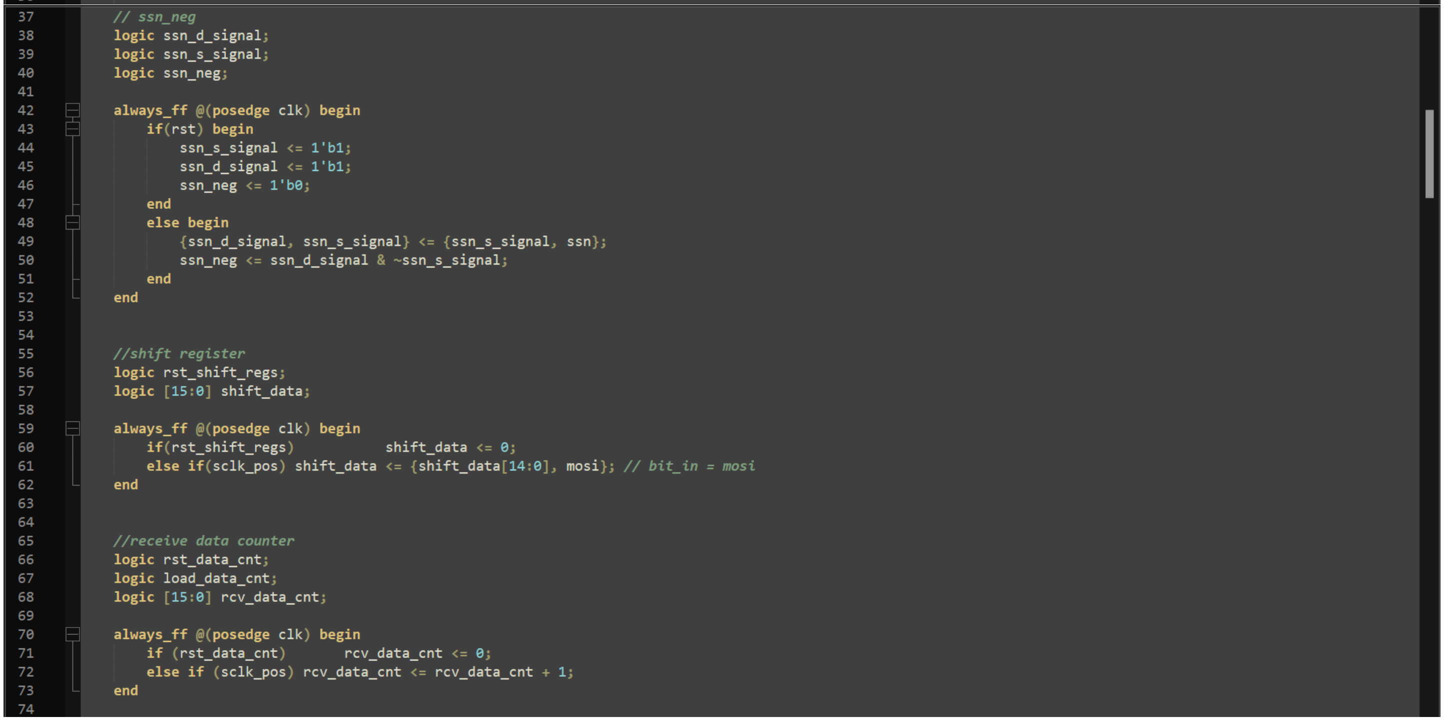
■ 系統架構程式碼與程式碼說明

截圖請善用 win+shift+S

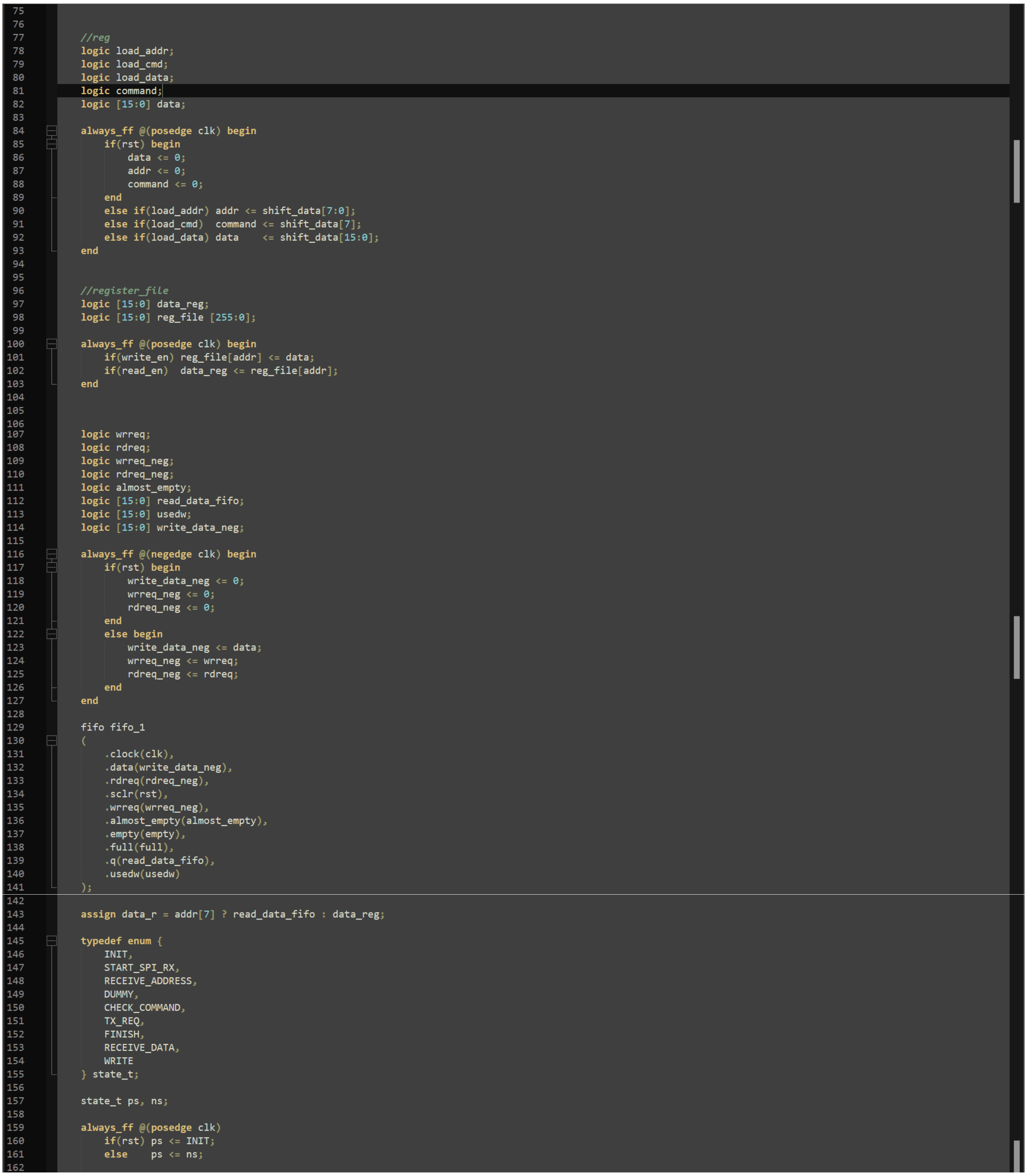
spi_rx.sv (這次主要只有改 Rx，所以就附就好)



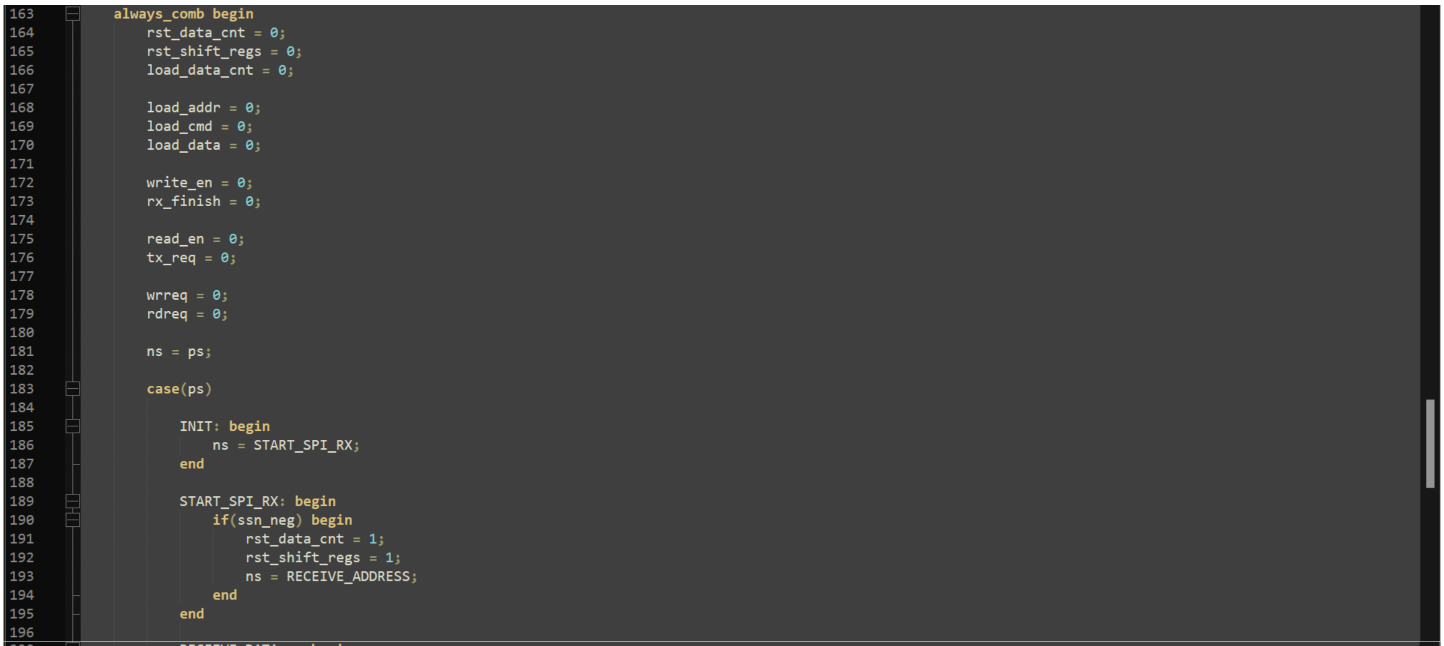
```
module spi_rx
    (
        input clk,
        input rst,
        input mosi,
        input ssn,
        input sclk,
        input tx_ack,

        output logic [7:0] addr,
        output logic [15:0] data_r,
        output logic read_en,
        output logic write_en,
        output logic tx_req
    );
    logic rx_finish;



    // sclk_pos;
    logic sclk_pos;
    logic sclk_d_signal;
    logic sclk_s_signal;
    always_ff @(posedge clk) begin
       if(rst) begin
          sclk_s_signal <= 1'b1;
          sclk_d_signal <= 1'b1;
          sclk_pos <= 1'b0;
        end
        else begin
          {sclk_d_signal, sclk_s_signal} <= {sclk_s_signal, sclk};
          sclk_pos <= ~sclk_d_signal & sclk_s_signal;
        end
     end

```



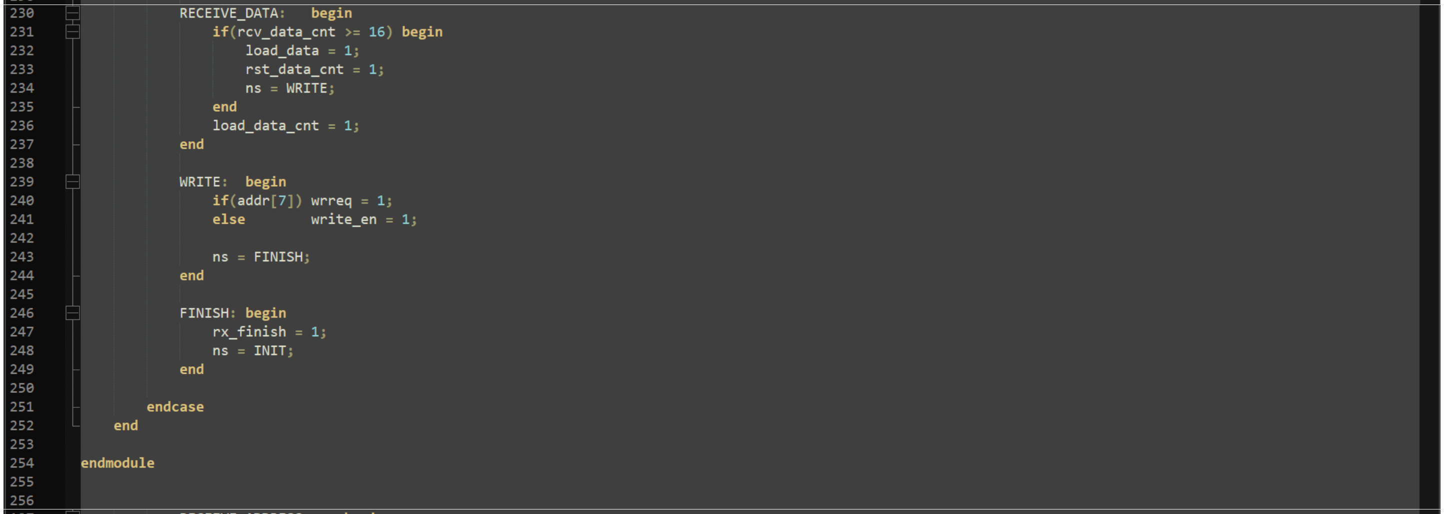
```
    // ssn_neg
    logic ssn_d_signal;
    logic ssn_s_signal;
    logic ssn_neg;

    always_ff @(posedge clk) begin
        if(rst) begin
            ssn_s_signal <= 1'b1;
            ssn_d_signal <= 1'b1;
            ssn_neg <= 1'b0;
        end
        else begin
            {ssn_d_signal, ssn_s_signal} <= {ssn_s_signal, ssn};
            ssn_neg <= ssn_d_signal & ~ssn_s_signal;
        end
    end


    //shift register
    logic rst_shift_regs;
    logic [15:0] shift_data;

    always_ff @(posedge clk) begin
        if(rst_shift_regs)          shift_data <= 0;
        else if(sclk_pos) shift_data <= {shift_data[14:0], mosi}; // bit_in = mosi
    end


    //receive data counter
    logic rst_data_cnt;
    logic load_data_cnt;
    logic [15:0] rcv_data_cnt;

    always_ff @(posedge clk) begin
        if (rst_data_cnt)       rcv_data_cnt <= 0;
        else if (sclk_pos) rcv_data_cnt <= rcv_data_cnt + 1;
    end

```



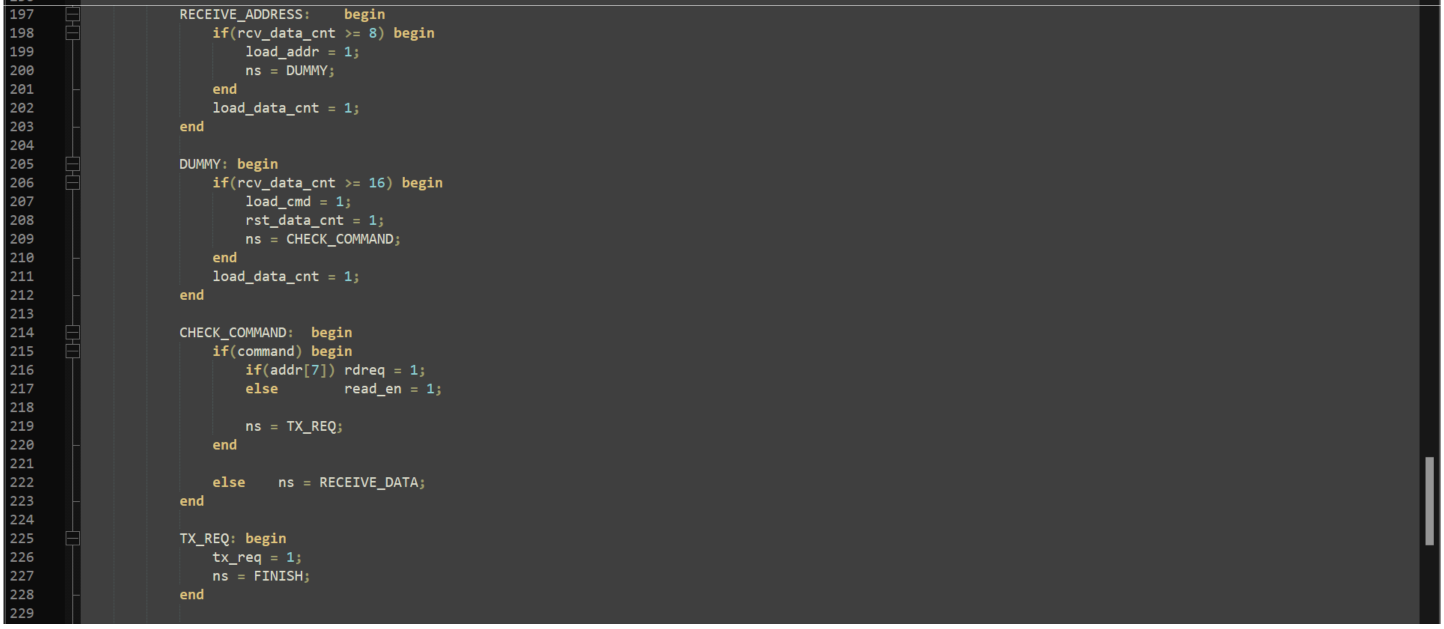
```
//reg
logic load_addr;
logic load_cmd;
logic load_data;
logic command;
logic [15:0] data;

always_ff @(posedge clk) begin
    if(rst) begin
        data <= 0;
        addr <= 0;
        command <= 0;
    end
    else if(load_addr) addr <= shift_data[7:0];
    else if(load_cmd)  command <= shift_data[7];
    else if(load_data) data    <= shift_data[15:0];
end


//register_file
logic [15:0] data_reg;
logic [15:0] reg_file [255:0];

always_ff @(posedge clk) begin
    if(write_en) reg_file[addr] <= data;
    if(read_en)  data_reg <= reg_file[addr];
end



logic wrreq;
logic rdreq;
logic wrreq_neg;
logic rdreq_neg;
logic almost_empty;
logic [15:0] read_data_fifo;
logic [15:0] usedw;
logic [15:0] write_data_neg;

always_ff @(negedge clk) begin
    if(rst) begin
        write_data_neg <= 0;
        wrreq_neg <= 0;
        rdreq_neg <= 0;
    end
    else begin
        write_data_neg <= data;
        wrreq_neg <= wrreq;
        rdreq_neg <= rdreq;
    end
end

fifo fifo_1
(
    .clock(clk),
    .data(write_data_neg),
    .rdreq(rdreq_neg),
    .sclr(rst),
    .wrreq(wrreq_neg),
    .almost_empty(almost_empty),
    .empty(empty),
    .full(full),
    .q(read_data_fifo),
    .usedw(usedw)
);

assign data_r = addr[7] ? read_data_fifo : data_reg;

typedef enum {
    INIT,
    START_SPI_RX,
    RECEIVE_ADDRESS,
    DUMMY,
    CHECK_COMMAND,
    TX_REQ,
    FINISH,
    RECEIVE_DATA,
    WRITE
} state_t;

state_t ps, ns;

always_ff @(posedge clk)
    if(rst) ps <= INIT;
    else    ps <= ns;

```



```
    always_comb begin
        rst_data_cnt = 0;
        rst_shift_regs = 0;
        load_data_cnt = 0;

        load_addr = 0;
        load_cmd = 0;
        load_data = 0;

        write_en = 0;
        rx_finish = 0;

        read_en = 0;
        tx_req = 0;

        wrreq = 0;
        rdreq = 0;

        ns = ps;

        case(ps)

            INIT: begin
                ns = START_SPI_RX;
            end

            START_SPI_RX: begin
                if(ssn_neg) begin
                    rst_data_cnt = 1;
                    rst_shift_regs = 1;
                    ns = RECEIVE_ADDRESS;
                end
            end

```

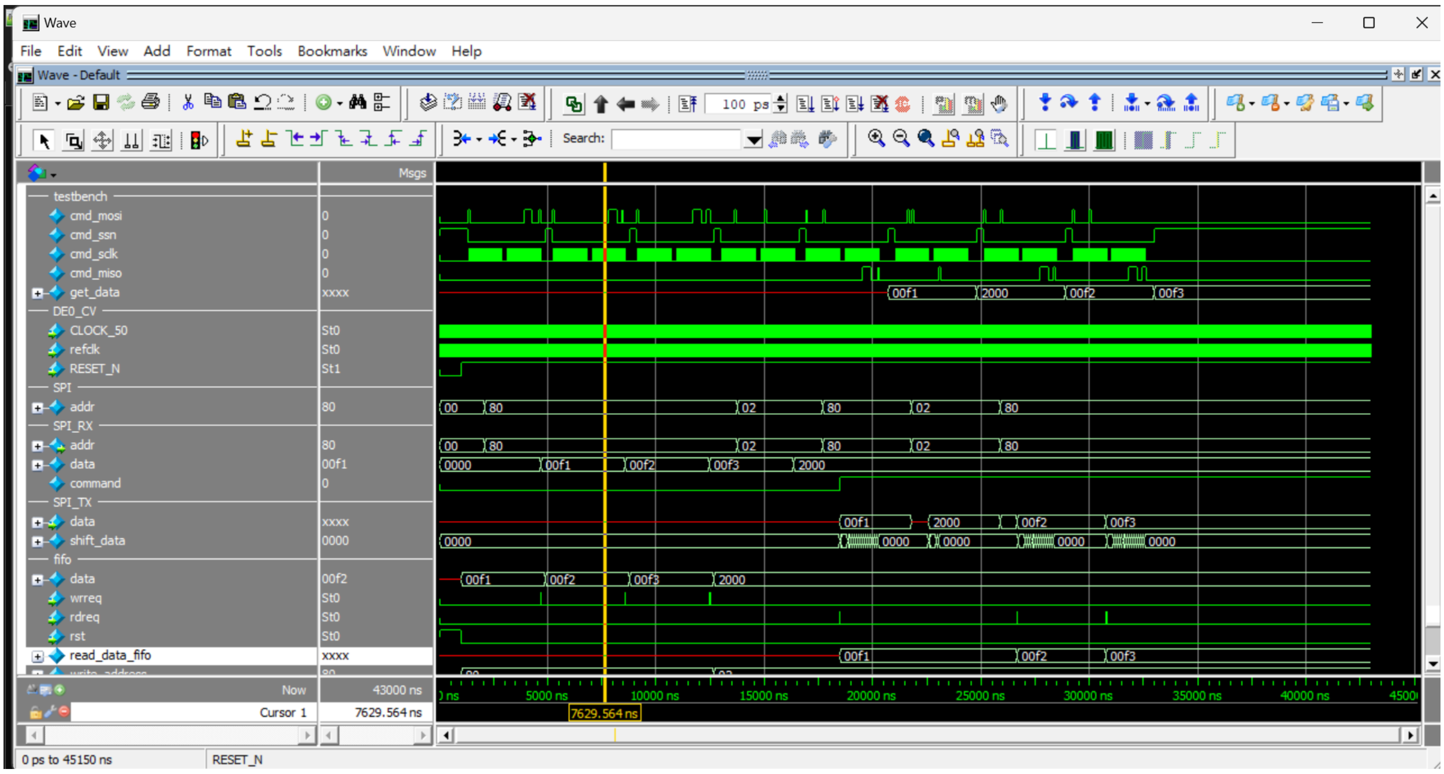


```
            RECEIVE_DATA:    begin
                if(rcv_data_cnt >= 16) begin
                    load_data = 1;
                    rst_data_cnt = 1;
                    ns = WRITE;
                end
                load_data_cnt = 1;
            end

            WRITE:   begin
                if(addr[7]) wrreq = 1;
                else        write_en = 1;

                ns = FINISH;
            end

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

        endcase
    end

endmodule


```



```
            RECEIVE_ADDRESS:    begin
                if(rcv_data_cnt >= 8) begin
                    load_addr = 1;
                    ns = DUMMY;
                end
                load_data_cnt = 1;
            end

            DUMMY: begin
                if(rcv_data_cnt >= 16) begin
                    load_cmd = 1;
                    rst_data_cnt = 1;
                    ns = CHECK_COMMAND;
                end
                load_data_cnt = 1;
            end

            CHECK_COMMAND:  begin
                if(command) begin
                    if(addr[7]) rdreq = 1;
                    else        read_en = 1;

                    ns = TX_REQ;
                end

                else    ns = RECEIVE_DATA;
            end

            TX_REQ: begin
                tx_req = 1;
                ns = FINISH;
            end

```

■ 模擬結果與結果說明：



■ 結論與心得：

這次作業一開始覺得應該只是個模組而已，應該不會太難，但沒想到在做的時候有些理解錯誤，以為要調整 register file，所以弄得亂七八糟，好險每次都會把作業備份，重新理解後再接線就沒甚麼問題了!