

2024/XX/XX

# 實驗十一

姓名：000      學號：000000000

班級：000

E-mail：000000000

## 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

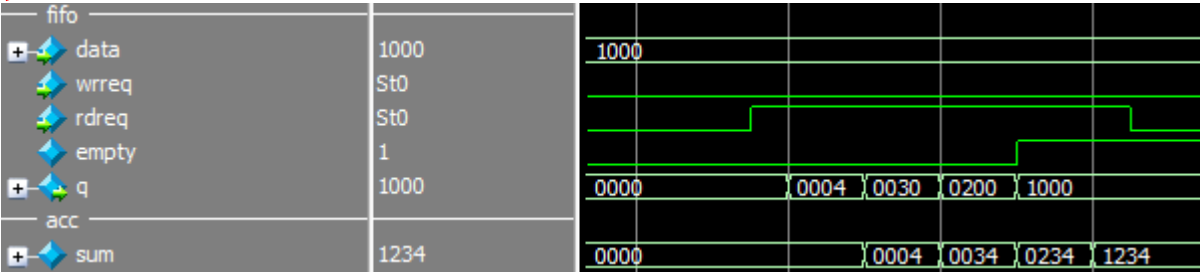
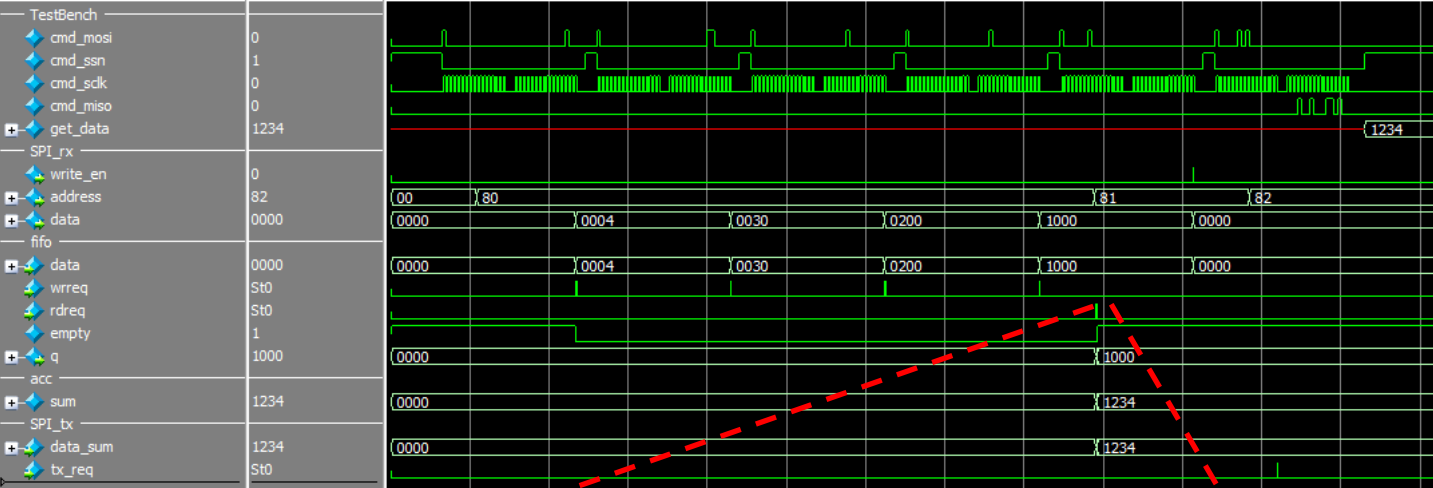
## SPI Slave

### ■ 實驗說明：

1. 將資料透過 SPI 寫進 FIFO 後，輸入一個指令將 FIFO 內所有的值加總，再輸入另一個指令將加總的值透過 SPI 輸出。
2. 指令格式:
  - 8000\_xxxx：將 xxxx 輸入至 FIFO 內儲存
  - 8100\_0000：將 FIFO 內的所有值加總
  - 8280\_0000：讀出加總完的值
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

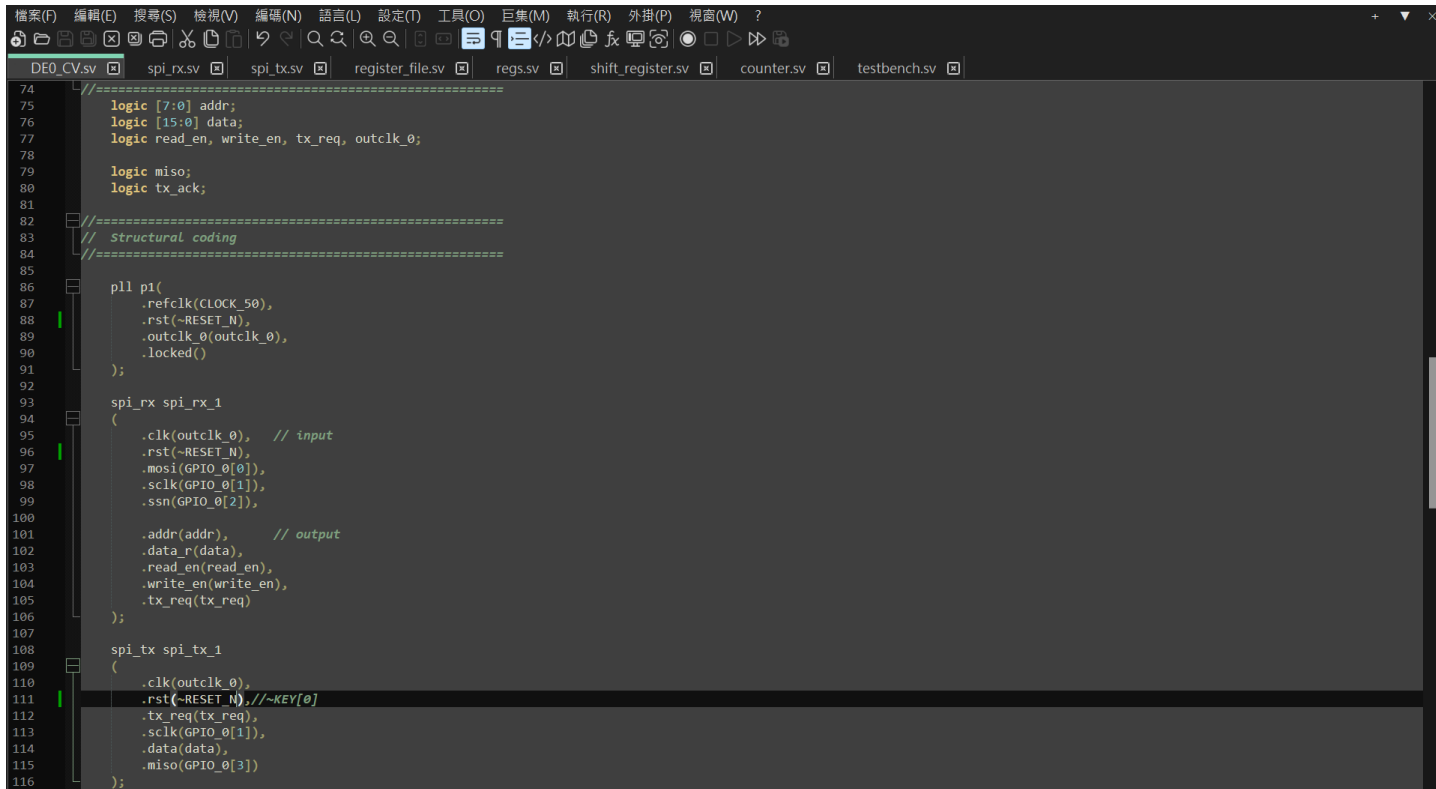
- 務必擷取到 get\_data 的波型



## ■ 系統架構程式碼與程式碼說明

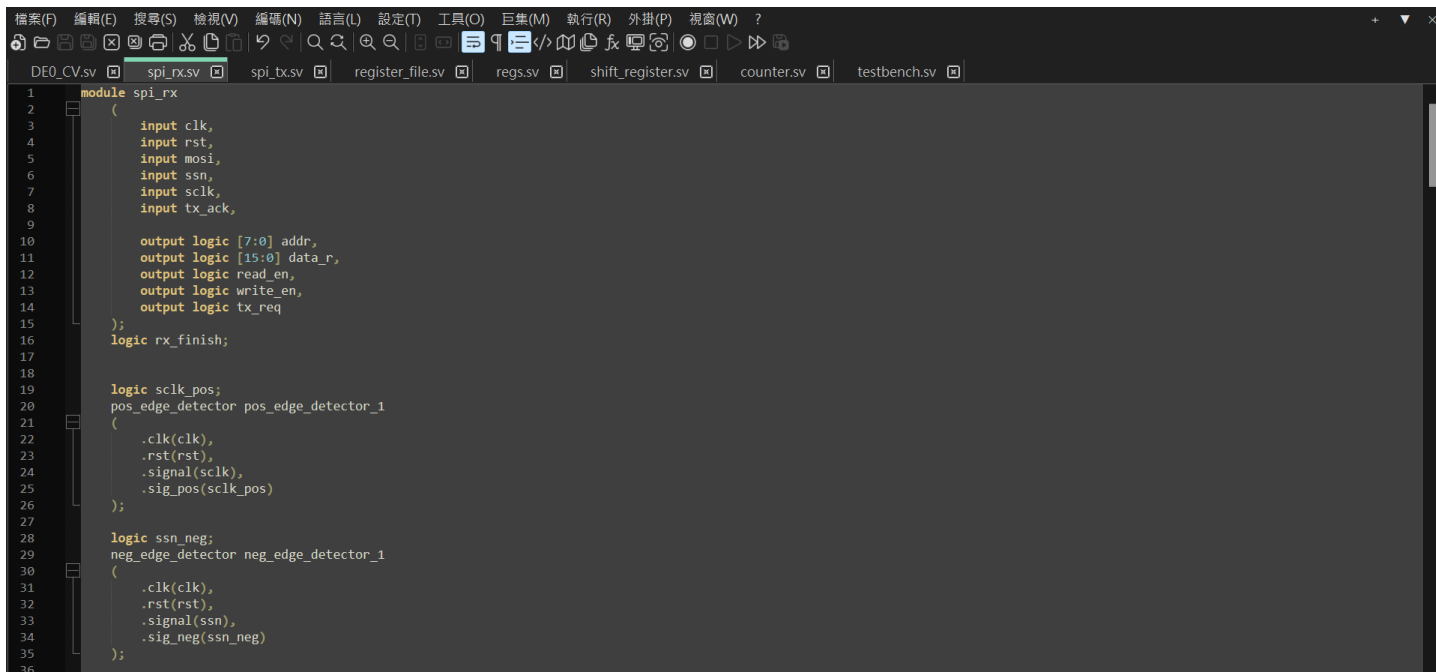
截圖請善用 win+shift+S

DE0\_CV:



```
74 //=====
75 logic [7:0] addr;
76 logic [15:0] data;
77 logic read_en, write_en, tx_req, outclk_0;
78
79 logic miso;
80 logic tx_ack;
81
82 //=====
83 // Structural coding
84 //=====
85
86 pll p1(
87     .refclk(CLOCK_50),
88     .rst(~RESET_N),
89     .outclk_0(outclk_0),
90     .locked()
91 );
92
93 spi_rx spi_rx_1
94 (
95     .clk(outclk_0), // input
96     .rst(~RESET_N),
97     .mosi(GPIO_0[0]),
98     .sclk(GPIO_0[1]),
99     .ssn(GPIO_0[2]),
100
101     .addr(addr), // output
102     .data_r(data),
103     .read_en(read_en),
104     .write_en(write_en),
105     .tx_req(tx_req)
106 );
107
108 spi_tx spi_tx_1
109 (
110     .clk(outclk_0),
111     .rst(~RESET_N), //~KEY[0]
112     .tx_req(tx_req),
113     .sclk(GPIO_0[1]),
114     .data(data),
115     .miso(GPIO_0[3])
116 );
```

Rx:主要更改 rx，左邊有綠色的線是這次有修改的



```
1 module spi_rx
2 (
3     input clk,
4     input rst,
5     input mosi,
6     input ssn,
7     input sclk,
8     input tx_ack,
9
10     output logic [7:0] addr,
11     output logic [15:0] data_r,
12     output logic read_en,
13     output logic write_en,
14     output logic tx_req
15 );
16 logic rx_finish;
17
18
19 logic sclk_pos;
20 pos_edge_detector pos_edge_detector_1
21 (
22     .clk(clk),
23     .rst(rst),
24     .signal(sclk),
25     .sig_pos(sclk_pos)
26 );
27
28 logic ssn_neg;
29 neg_edge_detector neg_edge_detector_1
30 (
31     .clk(clk),
32     .rst(rst),
33     .signal(ssn),
34     .sig_neg(ssn_neg)
35 );
36
```

```

37 logic rst shift_regs;
38 logic [15:0] shift_data;
39 shift_register shift_register_1
40 (
41     .rst(rst),
42     .clk(clk),
43     .sclk_pos(sclk_pos), // 每次 sclk 正緣觸發時
44     .mosi(mosi), // 就把一個 bit 的資料寫入移位暫存器
45     .shift_data(shift_data)
46 );
47
48 logic rst_data_cnt;
49 logic load_data_cnt;
50 logic [15:0] rcv_data_cnt;
51 counter counter_1
52 (
53     .clk(clk),
54     .rst(rst_data_cnt),
55     .load(sclk_pos), // 每次 sclk 正緣觸發 bit 寫入移位暫存器時, 計數現在傳了幾個 bit
56     .cnt(rcv_data_cnt)
57 );
58
59
60
61 logic load_addr;
62 logic load_cmd;
63 logic load_data;
64 //Logic [7:0] address;
65 logic command;
66 //Logic [15:0] data;
67 logic [15:0] data;
68 regs regs_1
69 (
70     .clk(clk),
71     .rst(rst),
72     .load_addr(load_addr),
73     .load_cmd(load_cmd),
74     .load_data(load_data),
75     .shift_data(shift_data),
76
77     .address(addr),
78     .command(command),
79     .data(data)
80 );
81
82 logic [15:0] data_reg;
83 register_file register_file_1
84 (
85     .clk(clk),
86     .rst(rst),
87     .write_en(write_en),
88     .read_en(read_en),
89     .addr(addr),
90     .data(data),
91     .data_r(data_reg)
92 );
93
94 logic wrreq;
95 logic rdreq;
96 logic wrreq_neg;
97 logic rdreq_neg;
98 logic almost_empty;
99 logic [15:0] read_data_fifo;
100 logic [15:0] usedw;
101 logic [15:0] write_data_neg;
102
103 always_ff @(negedge clk)
104 begin
105     if(rst) begin
106         write_data_neg <= 0;
107         wrreq_neg <= 0;
108         rdreq_neg <= 0;
109     end
110     else begin
111         write_data_neg <= data;
112         wrreq_neg <= wrreq;
113         rdreq_neg <= rdreq;
114     end
115 end
116
117 fifo fifo_1
118 (
119     .clock(clk),
120     .data(write_data_neg),
121     .rdreq(rdreq_neg),
122     .sclr(rst),
123     .wrreq(wrreq_neg),
124     .almost_empty(almost_empty),
125     .empty(empty),
126     .full(full),
127     .q(read_data_fifo),
128     .usedw(usedw)
129 );
130
131 //sum
132 logic [15:0] sum;
133
134 always_ff @(negedge clk)
135 begin
136     if(rst) begin
137         sum <= 0;
138     end
139     if(rdreq) begin
140         sum <= sum + read_data_fifo;
141     end
142 end
143
144 assign data_r = addr[7] ? sum : data_reg;
145

```

```

146 //wrreq_cnt
147 //用來判斷fifo中有幾個數，類似可控制初始值的empty
148 logic [15:0] wrreq_cnt;
149 always_ff @(negedge clk)
150 begin
151     if(rst) begin
152         wrreq_cnt <= 1;
153     end
154     else if(rdreq) begin
155         wrreq_cnt <= wrreq_cnt - 1;
156     end
157     else if(wrreq) begin
158         wrreq_cnt <= wrreq_cnt + 1;
159     end
160 end
161
162 typedef enum {
163     INIT,
164     START_SPI_RX,
165     RECEIVE_ADDRESS,
166     DUMMY,
167     CHECK_COMMAND,
168     TX_REQ,
169     FINISH,
170     RECEIVE_DATA,
171     WRITE
172 } state_t;
173
174 state_t ps, ns;
175
176
177 always_ff @(posedge clk)
178 if(rst) ps <= INIT;
179 else ps <= ns;
180
181 always_comb begin
182     rst_shift_regs = 0;
183     rst_data_cnt = 0;
184     load_data_cnt = 0;
185     rst_shift_regs = 0;
186     load_data_cnt = 0;
187
188     load_addr = 0;
189     load_cmd = 0;
190     load_data = 0;
191
192     write_en = 0;
193     rx_finish = 0;
194
195     read_en = 0;
196     tx_req = 0;
197
198     wrreq = 0;
199     rdreq = 0;
200
201     ns = ps;
202     case(ps)
203     INIT:
204         begin
205             ns = START_SPI_RX;
206         end
207
208     START_SPI_RX:
209         begin
210             if(ssn_neg)
211                 begin
212                     rst_data_cnt = 1;
213                     rst_shift_regs = 1;
214                     ns = RECEIVE_ADDRESS;
215                 end
216             end
217
218     RECEIVE_ADDRESS:
219         begin
220             if(rcv_data_cnt >= 8) begin
221                 load_addr = 1;
222                 ns = DUMMY;
223             end
224             load_data_cnt = 1;
225         end
226
227     DUMMY:
228         begin
229             if(rcv_data_cnt >= 16) begin
230                 load_cmd = 1;
231                 rst_data_cnt = 1;
232                 ns = CHECK_COMMAND;
233             end
234             load_data_cnt = 1;
235         end
236
237     CHECK_COMMAND:
238         begin
239             if(command) begin //read
240
241                 //if(addr[7]) rdreq = 1;
242                 //else read_en = 1;
243                 read_en = 1;
244                 ns = TX_REQ;
245             end
246             else //write
247                 if(addr == 8'h81) begin //在write中遇到addr為81時 139行
248                     rdreq = 1; //要求從fifo中讀取值 ==> if(rdreq) begin
249                     ns = DUMMY2; //做一個dummy等待加一次 sum <= sum + read_data_fifo;
250                     // end
251                 end
252                 else begin
253                     ns = RECEIVE_DATA;
254                 end
255             end
256
257     DUMMY2:
258         begin
259             if(wrreq_cnt>0) begin //wrreq_cnt 146行
260                 ns = CHECK_COMMAND;
261             end
262             else begin
263                 ns = FINISH;
264             end
265         end
266     end

```

```
267 TX_REQ:
268     begin
269         tx_req = 1;
270         ns = FINISH;
271     end
272
273 RECEIVE_DATA:
274     begin
275         if(rcv_data_cnt >= 16) begin
276             load_data = 1;
277             rst_data_cnt = 1;
278             ns = WRITE;
279         end
280         load_data_cnt = 1;
281     end
282
283 WRITE:
284     begin
285         if(addr == 8'h80) wrreq = 1;
286         else write_en = 1;
287     end
288     ns = FINISH;
289
290
291
292
293 FINISH:
294     begin
295         rx_finish = 1;
296         ns = INIT;
297     end
298
299 endcase
300 end
301
302 endmodule
```

Tx:

```
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) ?
DE0_CV.sv spi_rx.sv spi_tx.sv register_file.sv regs.sv shift_register.sv counter.sv testbench.sv

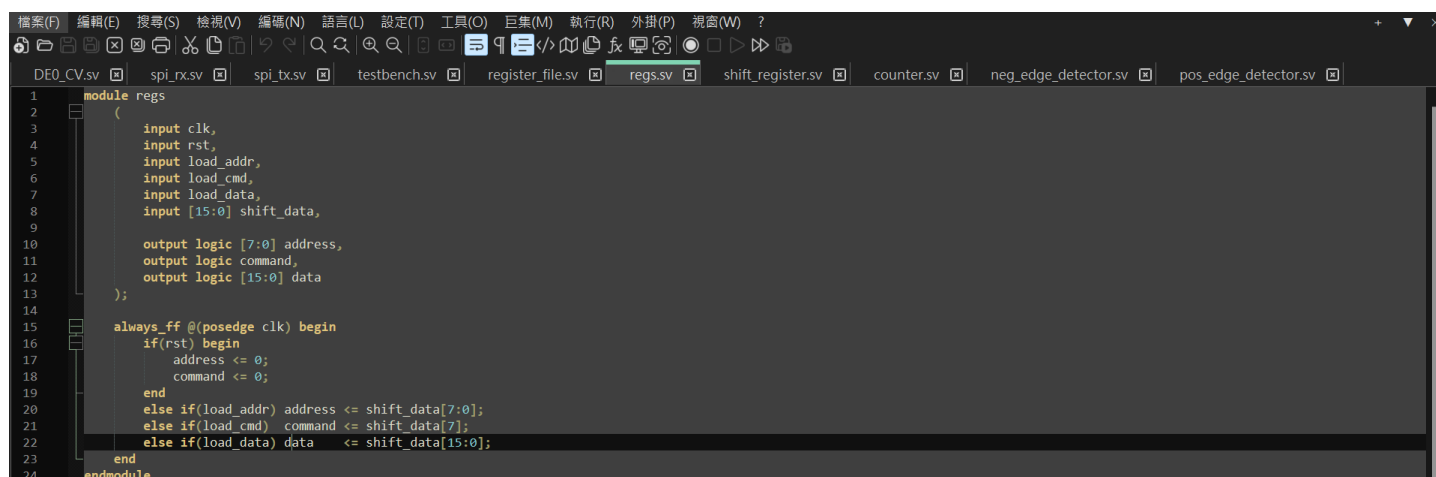
1 module spi_tx
2 (
3     input clk,
4     input rst,
5     input tx_req,
6     input sclk,
7     input logic [15:0] data,
8     output logic miso
9 );
10
11 logic sclk_neg;
12 neg_edge_detector neg_edge_detector_1
13 (
14     .clk(clk),
15     .rst(rst),
16     .signal(sclk),
17     .sig_neg(sclk_neg)
18 );
19
20 logic rst_data_cnt;
21 logic load_data_cnt;
22 logic [15:0] rcv_data_cnt;
23 counter counter_1
24 (
25     .clk(clk),
26     .rst(rst_data_cnt),
27     .load(sclk_neg), // 每次 sclk 正緣觸發 bit 寫入移位寄存器時, 計數現在傳了幾個 bit
28     .cnt(rcv_data_cnt)
29 );
30
31 // shift register
32 logic load_shift_data;
33 logic [15:0] shift_data;
34 always_ff @(posedge clk) begin
35     if(rst) shift_data <= 0;
36     else if(load_shift_data) shift_data <= data;
37     else if(sclk_neg) {miso, shift_data} <= {shift_data, 1'b0};
38 end
39
40
41
42 typedef enum {
43     INIT,
44     START_SPI_TX,
45     SEND_DATA,
46     FINISH
47 } state_t;
48
49 state_t ps, ns;
50
51 always_ff @(posedge clk)
52 if(rst) ps <= INIT;
53 else ps <= ns;
54
55 always_comb begin
56     rst_data_cnt = 0;
57     load_shift_data = 0;
58     ns = ps;
59     case(ps)
60     INIT:
61         begin
62             ns = START_SPI_TX;
63         end
64     START_SPI_TX:
65         begin
66             rst_data_cnt = 1;
67             if(tx_req) begin
68                 load_shift_data = 1;
69                 ns = SEND_DATA;
70             end
71         end
72     SEND_DATA:
73         begin
74             if(rcv_data_cnt >= 16) begin
75                 rst_data_cnt = 1;
76                 ns = FINISH;
77             end
78         end
79     FINISH: begin
80         ns = INIT;
81     end
82     endcase
83 end
84
85 endmodule
```



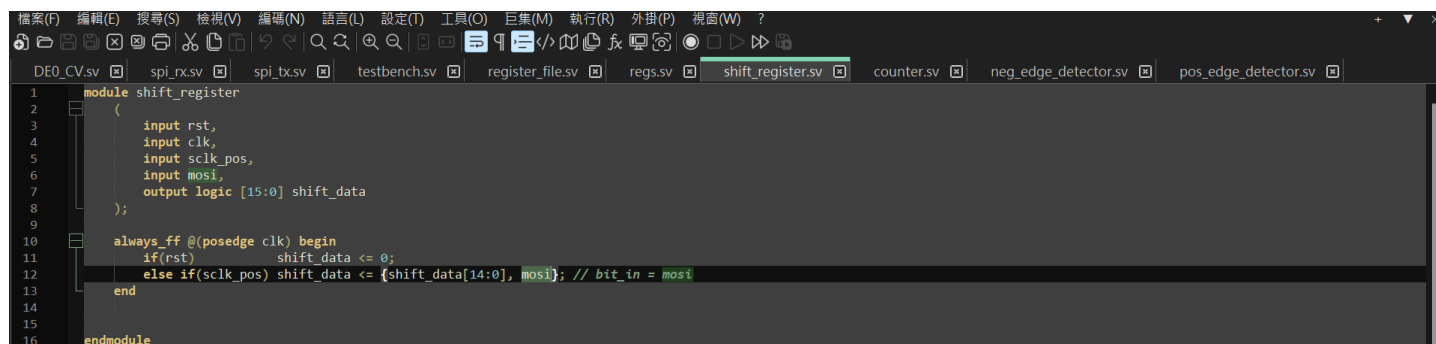
## 其他:



```
1 module register_file
2 (
3     input clk,
4     input rst,
5     input write_en,
6     input read_en,
7     input [7:0] addr,
8     input [15:0] data,
9     output logic [15:0] data_r
10 );
11
12 logic [15:0] reg_file [255:0];
13
14 always_ff @(posedge clk) begin
15     if(write_en) reg_file[addr] <= data;
16     if(read_en) data_r <= reg_file[addr];
17 end
18
19 endmodule
```



```
1 module regs
2 (
3     input clk,
4     input rst,
5     input load_addr,
6     input load_cmd,
7     input load_data,
8     input [15:0] shift_data,
9
10     output logic [7:0] address,
11     output logic command,
12     output logic [15:0] data
13 );
14
15 always_ff @(posedge clk) begin
16     if(rst) begin
17         address <= 0;
18         command <= 0;
19     end
20     else if(load_addr) address <= shift_data[7:0];
21     else if(load_cmd) command <= shift_data[7];
22     else if(load_data) data <= shift_data[15:0];
23 end
24 endmodule
```



```
1 module shift_register
2 (
3     input rst,
4     input clk,
5     input sclk_pos,
6     input mosi,
7     output logic [15:0] shift_data
8 );
9
10 always_ff @(posedge clk) begin
11     if(rst) shift_data <= 0;
12     else if(sclk_pos) shift_data <= {shift_data[14:0], mosi}; // bit_in = mosi
13 end
14
15
16 endmodule
```



```
1 module counter
2 (
3     input clk,
4     input rst,
5     input load,
6     output logic [15:0] cnt
7 );
8
9 always_ff @(posedge clk)
10     if (rst) cnt <= 0;
11     else if (load) cnt <= cnt + 1;
12
13 endmodule
```

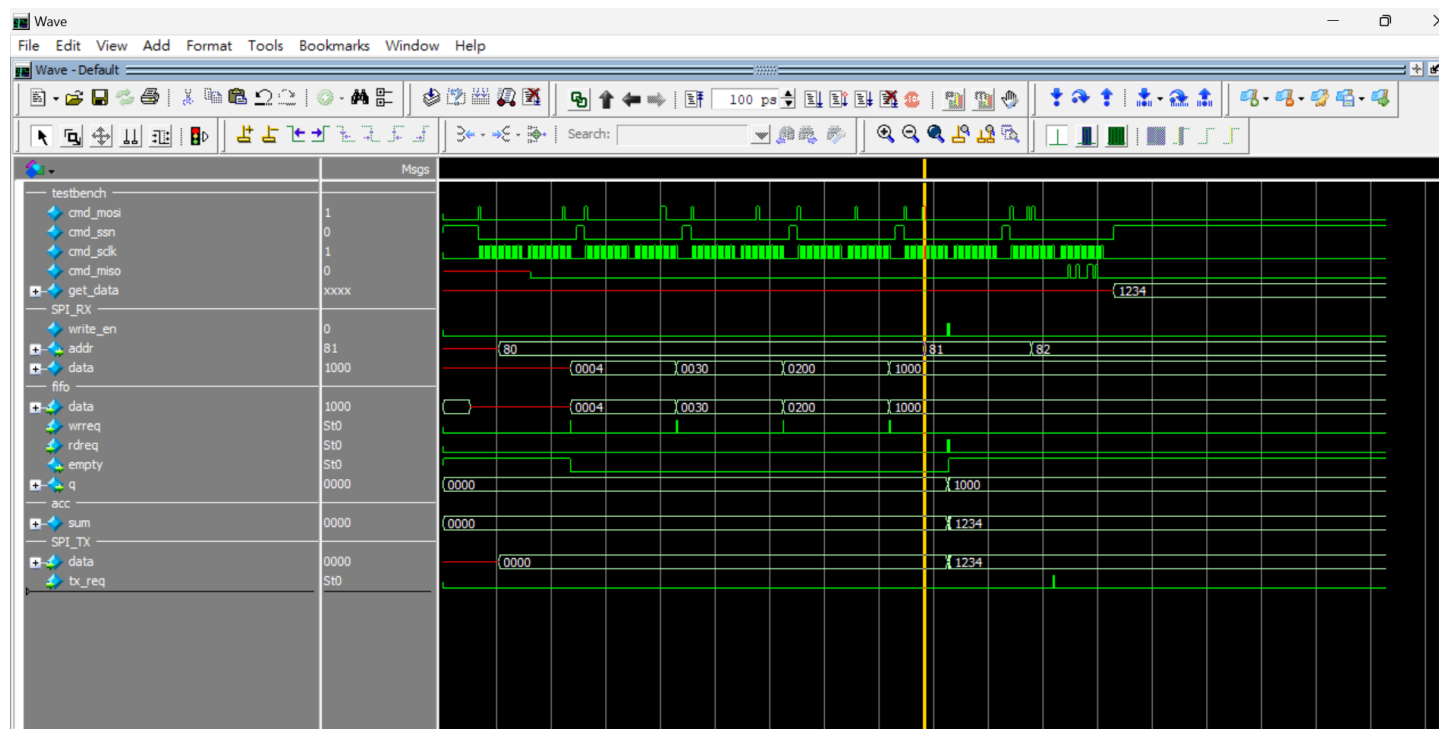
```
檔案(F) 編輯(E) 搜尋(S) 檢視(V) 編碼(N) 語言(L) 設定(T) 工具(O) 巨集(M) 執行(R) 外掛(P) 視窗(W) ?
DE0_CV.sv spi_rx.sv spi_tx.sv testbench.sv register_file.sv regs.sv shift_register.sv counter.sv neg_edge_detector.sv pos_edge_detector.sv

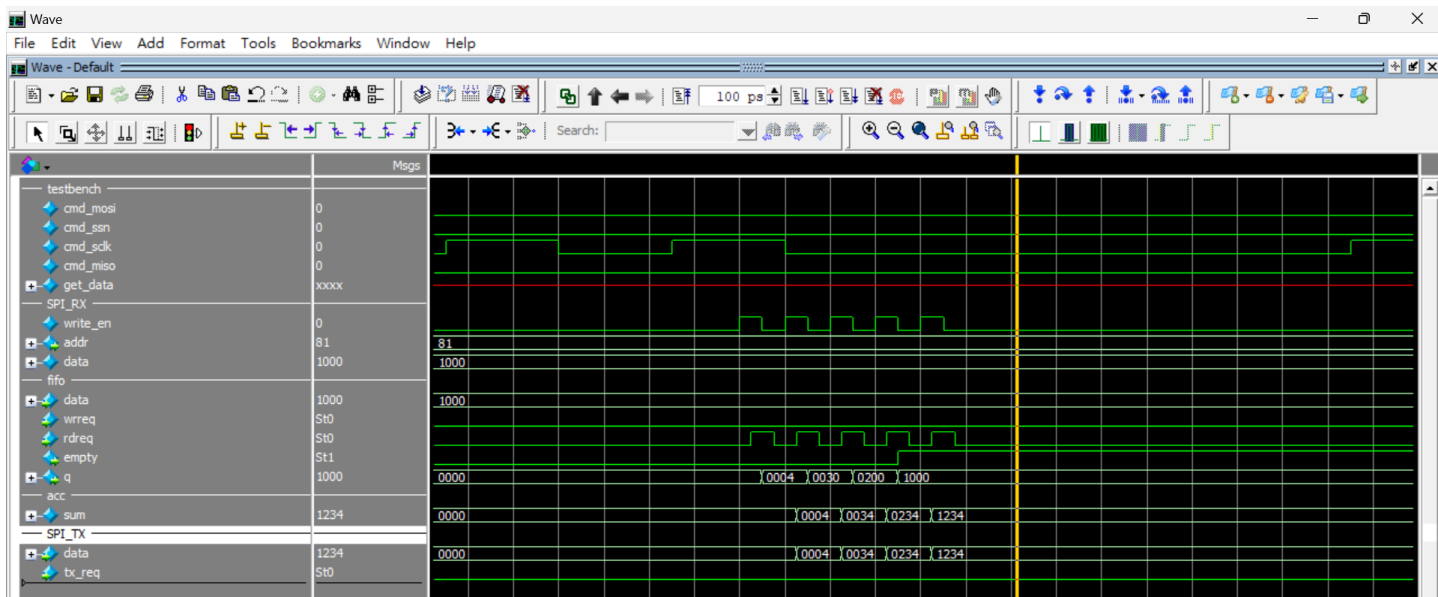
1 module neg_edge_detector
2 (
3     input clk,
4     input rst,
5     input signal,
6     output logic sig_neg
7 );
8
9     logic d_signal;
10    logic s_signal;
11
12    always_ff @(posedge clk) begin
13        if(rst) begin
14            s_signal <= 1'b1;
15            d_signal <= 1'b1;
16            sig_neg <= 1'b0;
17        end
18        else begin
19            {d_signal, s_signal} <= {s_signal, signal};
20            sig_neg <= d_signal & ~s_signal;
21        end
22    end
23
24 endmodule
```

```
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) ?
DE0_CV.sv spi_rx.sv spi_tx.sv testbench.sv register_file.sv regs.sv shift_register.sv counter.sv neg_edge_detector.sv pos_edge_detector.sv

1 module pos_edge_detector
2 (
3     input clk,
4     input rst,
5     input signal,
6     output logic sig_pos
7 );
8
9     logic d_signal;
10    logic s_signal;
11
12    always_ff @(posedge clk) begin
13        if(rst) begin
14            s_signal <= 1'b1;
15            d_signal <= 1'b1;
16            sig_pos <= 1'b0;
17        end
18        else begin
19            {d_signal, s_signal} <= {s_signal, signal};
20            sig_pos <= ~d_signal & s_signal;
21        end
22    end
23
24 endmodule
```

## ■ 模擬結果與結果說明：





## ■ 結論與心得：

這次作業大致上是基於上次 fifo 來更改，調整完地址判斷後，在相應的地址判斷中做要做的事就好：

1.改 8' h80 需求:當地址為 80 時，寫入資料進入 fifo(wrreq)

```

283 WRITE:
284 begin
285
286     if(addr == 8'h80) wrreq = 1;    // if(addr[7]) wrreq = 1;
287     else write_en = 1;
288
289     ns = FINISH;
290
291 end

```

2.當指令為寫入時(command == 0)，做 sum 的累加，並用一個 dummy 的時間去等跑一次累加，command 為 1 時，由 addr[7]做地址為 82 的讀出判斷就好

```

143 assign data_r = addr[7] ? sum : data_reg;
144
145

```

```

237 CHECK_COMMAND:
238 begin
239     if(command) begin //read
240
241         //if(addr[7]) rdreq = 1;
242         //else read_en = 1;
243         read_en = 1;
244         ns = TX_REQ;
245     end
246     else //write
247         if(addr == 8'h81) begin //在write中遇到addr為81時 139行
248             rdreq = 1; //要求從fifo中讀取值 =====> if(rdreq) begin
249             ns = DUMMY2; //做一個dummy等待加一次 sum <= sum + read_data_fifo;
250             // end
251         end
252         else begin
253             ns = RECEIVE_DATA;
254         end
255     end
256
257 DUMMY2:
258 begin
259     if(wrreq_cnt>0) begin //wrreq_cnt 146行
260         ns = CHECK_COMMAND;
261     end
262     else begin
263         ns = FINISH;
264     end
265 end
266

```