

2024/XX/XX

實驗七

姓名：黃文祺 學號：01057013

班級：資工 3A

E-mail：OOOOOOOOO

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

一、RS-232 Reciever(一)

■ 實驗說明：

1. 寫出 RS-232 Reciever，並顯示其模擬圖。

2. 輸入:

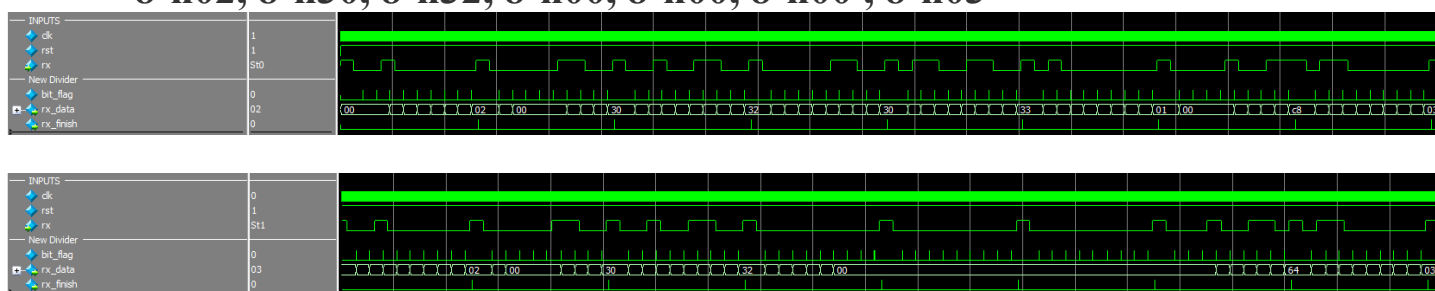
- clk : 50M
- rst
- rx : 接收到資料(1-bit)

3. 輸出:

- rx_data[7:0] : 將收到的資料解析、組合然後輸出。
- rx_finish : 標示完整接收到 1 個 byte 的 trigger。

4. 測資:

- 8'h02, 8'h30, 8'h32, 8'h30, 8'h33, 8'h01, 8'h03
- 8'h02, 8'h30, 8'h32, 8'h00, 8'h00, 8'h00, 8'h03

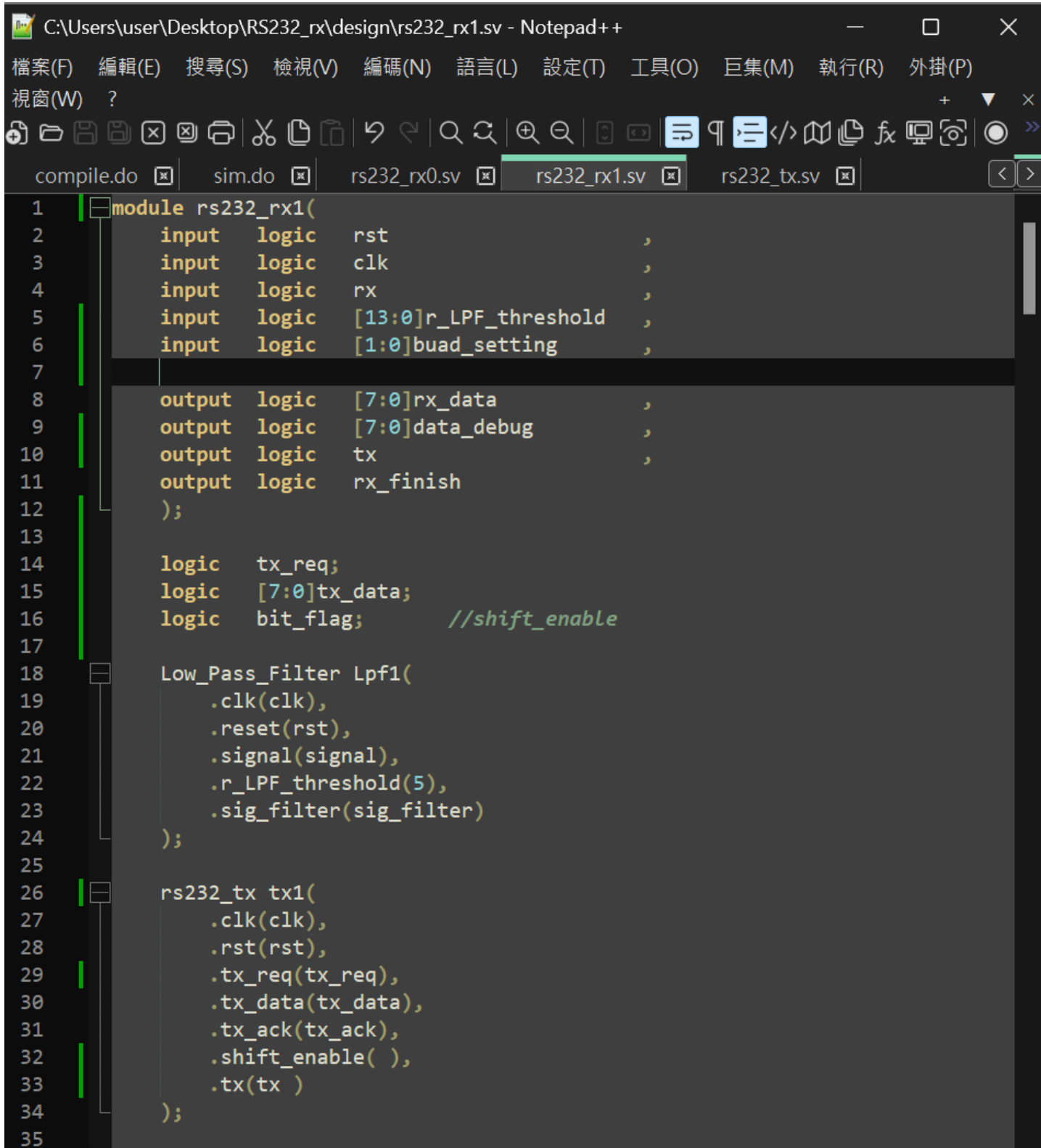


■ 系統架構程式碼與程式碼說明

截圖請善用 win+shift+S

(作業 1 和作業 2 的 rs232_rx 都用同一份)

(rs232_rx1.sv)



```
1 module rs232_rx1(
2     input logic rst,
3     input logic clk,
4     input logic rx,
5     input logic [13:0]r_LPF_threshold,
6     input logic [1:0]buad_setting,
7
8     output logic [7:0]rx_data,
9     output logic [7:0]data_debug,
10    output logic tx,
11    output logic rx_finish
12 );
13
14    logic tx_req;
15    logic [7:0]tx_data;
16    logic bit_flag; //shift_enable
17
18    Low_Pass_Filter Lpf1(
19        .clk(clk),
20        .reset(rst),
21        .signal(signal),
22        .r_LPF_threshold(5),
23        .sig_filter(sig_filter)
24    );
25
26    rs232_tx tx1(
27        .clk(clk),
28        .rst(rst),
29        .tx_req(tx_req),
30        .tx_data(tx_data),
31        .tx_ack(tx_ack),
32        .shift_enable( ),
33        .tx(tx)
34    );
35
```

```

36     typedef enum    {START,RX_START,NUM_BITS,COMPLETE,RECEIVE,RW_REG_F,
TX_REQ_1,TX_ACK_1,TX_REQ_2,TX_ACK_2,TX_REQ_3,TX_ACK_3,TX_REQ_4,
TX_ACK_4} state_Rx_t;
37     state_Rx_t  ps , ns ;    //Tx_ps, Tx_ns;
38
39     logic    [7:0]head    ;
40     logic    [7:0]addr1   ;
41     logic    [7:0]addr2   ;
42     logic    [7:0]data1   ;
43     logic    [7:0]data2   ;
44     logic    [7:0]r_w     ;
45     logic    [7:0]tail    ;
46     logic    shift        ;
47
48     logic    rst_shift    ;
49
50     integer  Baud_Rate;
51     integer  Baud_Rate_Half;
52     integer  baud_cnt;
53     logic    enable_baud_cnt;
54     logic    rst_baud_cnt;
55
56     assign  Baud_Rate      = 50000000/38400; //    50M/38400
57     assign  Baud_Rate_Half = Baud_Rate>>1;
58
59     integer  bit_cnt;
60     logic    enable_bit_cnt;
61     logic    rst_bit_cnt;
62
63     logic    s_signal;
64     logic    d_signal;
65     logic    rx_neg  ;
66
67     logic    [7:0] reg_file [255:0];
68
69
70     logic    [7:0] addr;          //register file
71     logic    [7:0] data;
72     logic    [7:0] data_r;
73
74     assign  data      = {data1[3:0],data2[3:0]};
75     assign  addr      = {addr1[3:0],addr2[3:0]};
76
77
78     logic    [7:0] tx_data_temp[3:0];
79     logic    [1:0] tx_idx;
80     logic    rst_tx_idx;
81     logic    inc_tx_idx;
82

```

```
83     assign tx_data_temp[0] = 8'h02;
84     assign tx_data_temp[1] = {4'h3,data_r[7:4]};
85     assign tx_data_temp[2] = {4'h3,data_r[3:0]};
86     assign tx_data_temp[3] = 8'h03;
```

```
87
88
89     logic [3:0] pkg_cnt;
90     logic rst_pkg_cnt;
91     logic pkg_ready;
92     logic write;
```

```
93
94     // BAUD_CNT
95     always_ff @(posedge clk) begin
96         if(rst_baud_cnt)
97             baud_cnt <= 0;
98         else if(enable_baud_cnt)
99             baud_cnt <= baud_cnt + 1;
100     end
```

```
101
102     // bit_cnt
103     always_ff @(posedge clk) begin
104         if(rst_bit_cnt)
105             bit_cnt <= 0;
106         else if(enable_bit_cnt)
107             bit_cnt <= bit_cnt + 1;
108     end
109
```

```
110     // package_counter
111     always_ff @(posedge clk) begin
112         if(rst) begin
113             pkg_cnt <= 0;
114         end
115         if(rst_pkg_cnt) begin
116             pkg_cnt <= 0;
117         end
118         else if(shift) begin
119             pkg_cnt <= pkg_cnt + 1;
120         end
121     end
```

```
122
123     // shift_register
124     always_ff @(posedge clk) begin
125         if(rst) begin
126             rx_data <= 0;
127         end
128
129         if(bit_flag ) begin//bit_cnt
130             rx_data[7] <= rx;
131             rx_data=rx_data>>1;
132         end
133     end
134
```

```

135      // package shift_register
136      always_ff @(posedge clk) begin
137          if(rst_shift) begin
138              rx_data = 0;
139              head    = 0;
140              addr1   = 0;
141              addr2   = 0;
142              data1   = 0;
143              data2   = 0;
144              r_w     = 0;
145              tail    = 0;
146          end
147          if(shift ) begin//bit_cnt
148              head    <= addr1;
149              addr1   <= addr2;
150              addr2   <= data1;
151              data1   <= data2;
152              data2   <= r_w;
153              r_w     <= tail;
154              tail    <= rx_data;
155          end
156      end
157
158
159      //negedge detect
160      always_ff @(posedge clk) begin
161          if (rst) begin
162              s_signal    <= 1;
163              d_signal    <= 1;
164              rx_neg      <= 0;
165          end
166          else begin
167              {d_signal, s_signal} <= {s_signal, rx};
168              rx_neg <= ~s_signal & d_signal;
169          end
170      end
171
172      //register file
173      always_ff @(posedge clk) begin
174          if(write) reg_file[addr] <= data;
175      end
176
177      assign data_r = reg_file[addr];
178

```

```

179      //tx_idx
180      always_ff @(posedge clk) begin
181          if (rst | rst_tx_idx) begin
182              tx_idx = 0;
183          end
184
185          tx_data <= tx_data_temp[tx_idx];
186
187          if (inc_tx_idx) begin
188              tx_idx = tx_idx + 1;
189          end
190      end
191
192      // fsm count_clk
193      always_ff @(posedge clk) begin
194          if(rst)
195              ps <= START;
196          else
197              ps <= ns;
198      end
199
200      always_comb begin
201          rst_baud_cnt = 0;
202          enable_baud_cnt = 0;
203
204          bit_flag = 0;
205          rx_finish = 0;
206
207          enable_bit_cnt = 0;
208          rst_bit_cnt = 0;
209
210          rst_pkg_cnt = 0;
211          pkg_ready = 0;
212          write = 0;
213
214          tx_req = 0;
215          inc_tx_idx = 0;
216          rst_tx_idx = 0;
217
218          shift = 0;
219          rst_shift = 0;
220
221          ns = ps;
222

```

```

223 case(ps)
224
225     START: begin
226         rst_shift=1;
227         rst_bit_cnt=1;
228         rst_tx_idx    = 1;
229         ns    = RX_START;
230
231     end
232
233     RX_START: begin
234
235         if(rx_neg)
236             ns = NUM_BITS;
237         else
238             ns = RX_START;
239         end
240
241     NUM_BITS: begin
242         rst_baud_cnt=1;
243         if(bit_cnt>8) ns=COMPLETE;
244         else          ns=RECEIVE;
245     end
246
247     RECEIVE: begin
248
249         enable_baud_cnt=1;
250         if (baud_cnt==(Baud_Rate_Half))    begin
251             bit_flag =1;
252         end
253
254         if(baud_cnt>=Baud_Rate) begin
255             enable_bit_cnt=1;
256             ns=NUM_BITS;
257         end
258         else    ns=RECEIVE;
259     end
260
261     COMPLETE: begin
262         rst_bit_cnt =1;
263         rx_finish    =1;
264         shift        =1;    //rx_finish
265         if(pkg_cnt+1>=7)begin
266             pkg_ready    =1;
267             rst_pkg_cnt =1;
268             ns    = RW_REG_F;
269         end
270         else begin
271             ns    = RX_START;
272         end
273     end
274

```



```

275 RW_REG_F: begin
276     rst_tx_idx =1;
277     if(r_w[0]) begin
278         write=1;
279         ns = RX_START;
280     end
281     else ns = TX_REQ_1;
282 end
283
284 TX_REQ_1: begin
285     tx_req =1;
286     //inc_tx_idx =1;
287     ns = TX_ACK_1;
288 end
289
290 TX_ACK_1: begin
291     if(tx_ack) ns = TX_REQ_2;
292     else ns = TX_ACK_1;
293 end
294
295 TX_REQ_2: begin
296     tx_req =1;
297     inc_tx_idx =1;
298     ns = TX_ACK_2;
299 end
300
301 TX_ACK_2: begin
302     if(tx_ack) ns = TX_REQ_3;
303     else ns = TX_ACK_2;
304 end
305
306 TX_REQ_3: begin
307     tx_req =1;
308     inc_tx_idx =1;
309     ns = TX_ACK_3;
310 end
311
312 TX_ACK_3: begin
313     if(tx_ack) ns = TX_REQ_4;
314     else ns = TX_ACK_3;
315 end
316
317 TX_REQ_4: begin
318     tx_req =1;
319     inc_tx_idx =1;
320     ns = TX_ACK_4;
321 end
322
323 TX_ACK_4: begin
324     if(tx_ack) ns = START;
325     else ns = TX_ACK_4;
326 end
327 endcase
328 end
329 endmodule

```

(rs232_tx.sv)

```
C:\Users\user\Desktop\RS232_rx\design\rs232_tx.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) ?  
+ - x  
compile.do [x] sim.do [x] rs232_rx0.sv [x] rs232_rx1.sv [x] rs232_tx.sv [x]  
1 module rs232_tx(  
2     input logic rst ,  
3     input logic clk ,  
4     input logic tx_req ,  
5     input logic [7:0]tx_data ,  
6     output logic tx_ack ,  
7     output logic shift_enable ,  
8     output logic tx  
9 );  
10  
11 typedef enum {IDLE,CHK_REQ,LD_TX_DATA,CHK_BIT_CNT,TRANS,COMP}  
state_Tx_t;  
state_Tx_t Tx_ps, Tx_ns;  
12  
13  
14 logic load_tx_data;  
15  
16 integer Baud_rate;  
17 integer baud_cnt;  
18 logic enable_baud_cnt;  
19 logic rst_baud_cnt;  
20  
21 integer bit_cnt;  
22 logic enable_bit_cnt;  
23 logic rst_bit_cnt;  
24  
25  
26 logic [9:0]temp_data ;  
27  
28 assign Baud_rate = 50000000/38400; // 50M/38400  
29  
30  
31 // BAUD_CNT  
32 always_ff @(posedge clk) begin  
33     if(rst_baud_cnt)  
34         baud_cnt <= 0;  
35     else if(enable_baud_cnt)  
36         baud_cnt <= baud_cnt + 1;  
37 end  
38  
39 // bit_cnt  
40 always_ff @(posedge clk) begin  
41     if(rst_bit_cnt)  
42         bit_cnt <= 0;  
43     else if(enable_bit_cnt)  
44         bit_cnt <= bit_cnt + 1;  
45 end
```

```

46
47
48     // shift_register
49     always_ff @(posedge clk) begin
50         if(rst) begin                //rst
51             tx <= 1;
52             temp_data <= 10'b0;
53         end
54
55         if(shift_enable)              //shift => 8bits
56             tx <= {temp_data[bit_cnt-1]}; //each tx 1 bit
57
58         if(load_tx_data)              //start => 1bit
59             tx <= 0;
60
61         if(load_tx_data)              //stop  => 1bit
62             temp_data <= {2'b11,tx_data[7:0]};
63
64     end
65
66     // fsm count_clk
67     always_ff @(posedge clk) begin
68         if(rst)
69             Tx_ps <= IDLE;
70         else
71             Tx_ps <= Tx_ns;
72     end
73

```

```

74     always_comb begin
75         rst_baud_cnt    = 0;
76         enable_baud_cnt = 0;
77
78         shift_enable     = 0;
79         tx_ack           = 0;
80         load_tx_data     = 0;
81
82         enable_bit_cnt   = 0;
83         rst_bit_cnt      = 0;
84
85         Tx_ns            = Tx_ps;
86
87         case(Tx_ps)
88
89             IDLE: begin
90                 rst_bit_cnt=1;
91                 rst_baud_cnt=1;
92                 Tx_ns    = CHK_REQ;
93             end
94         end
95

```

```

96     CHK_REQ: begin
97
98         if(tx_req)
99             Tx_ns = LD_TX_DATA;
100        else
101            Tx_ns = CHK_REQ;
102        end
103
104     LD_TX_DATA: begin
105         load_tx_data=1;
106         Tx_ns=CHK_BIT_CNT;
107     end
108
109     CHK_BIT_CNT: begin
110         enable_bit_cnt=1;
111         if(bit_cnt>=10) begin
112             rst_bit_cnt=1;
113
114             Tx_ns=COMP;
115         end
116         else
117             Tx_ns=TRANS;
118         end
119
120     TRANS: begin
121         enable_baud_cnt=1;
122         if(baud_cnt>=Baud_rate) begin
123             shift_enable=1;
124             rst_baud_cnt=1;
125             Tx_ns=CHK_BIT_CNT;
126         end
127         else
128             Tx_ns=TRANS;
129         end
130     end

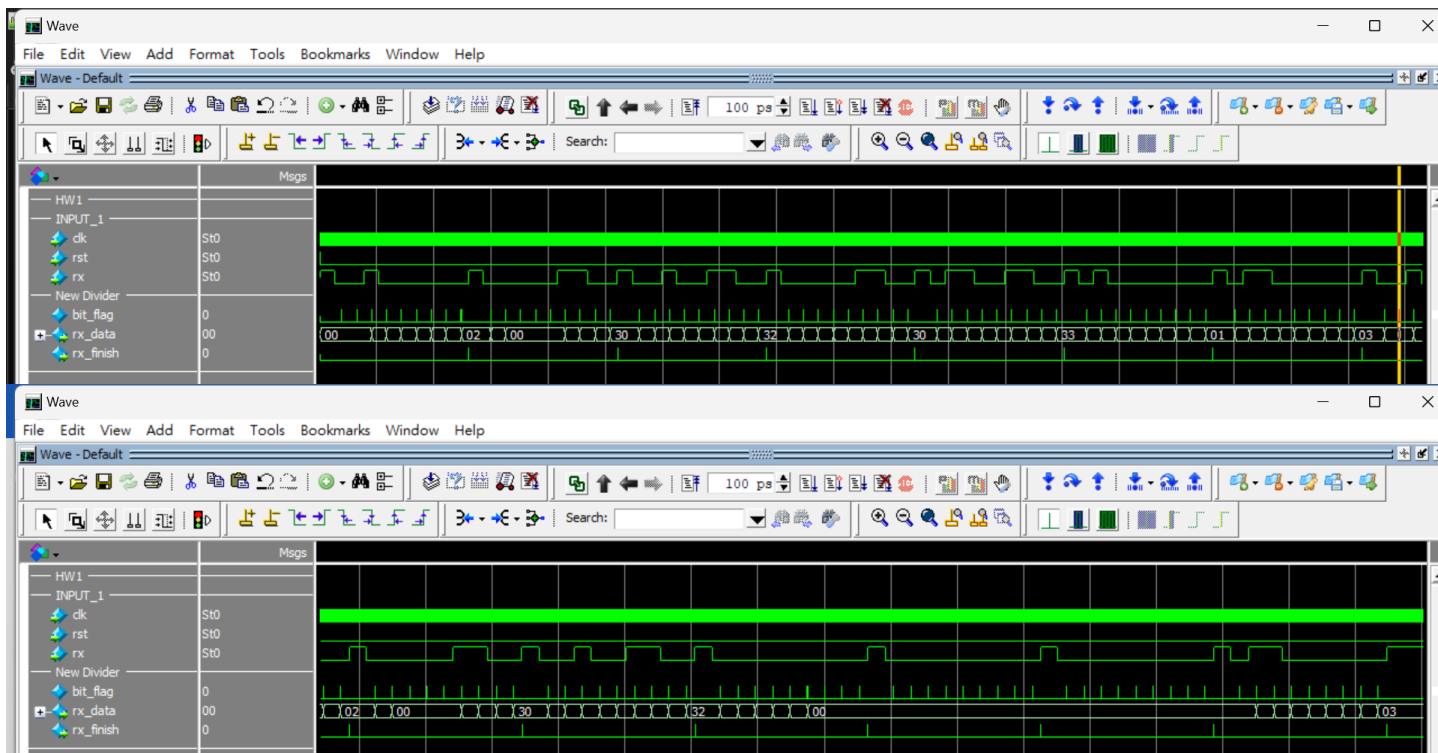
```

```

131     COMP: begin
132         tx_ack=1;
133         if(tx_req==0)
134             Tx_ns=CHK_REQ;
135         else
136             Tx_ns=COMP;
137         end
138
139     endcase
140 end
141
142 endmodule

```

■ 模擬結果與結果說明： (HW1)

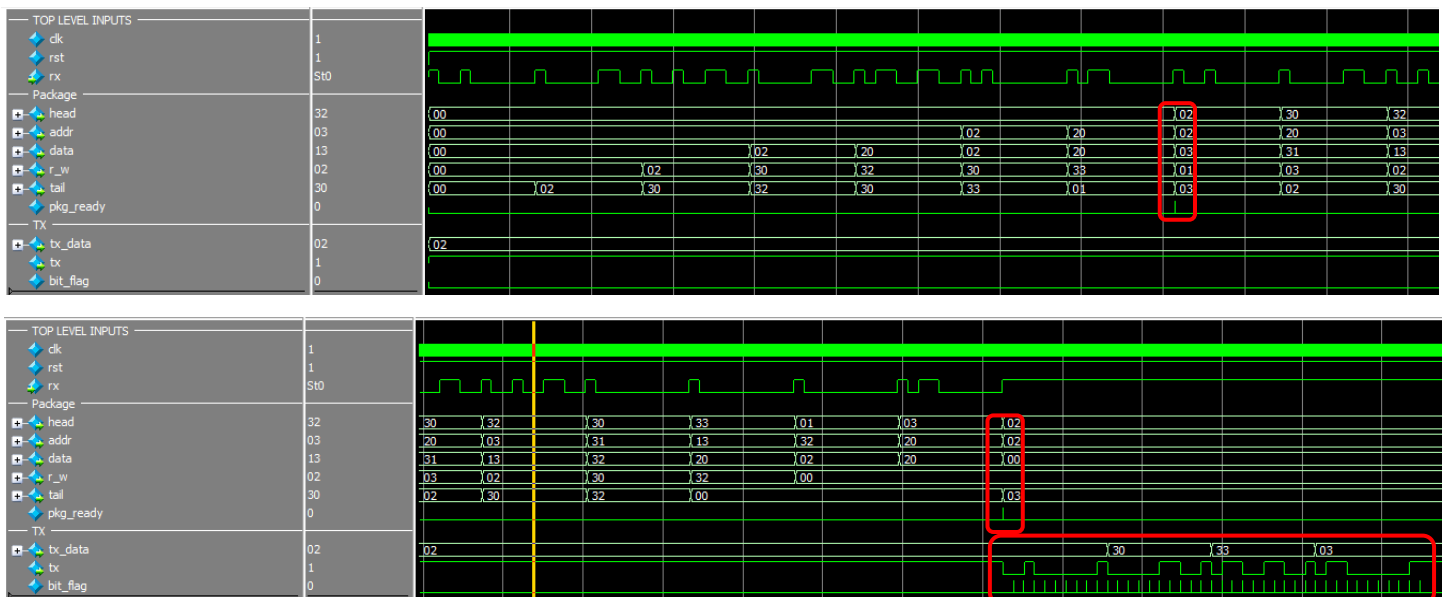


(沒有做 check sum，所以沒有 c8 跟 64)

二、RS-232 Reciever(二)

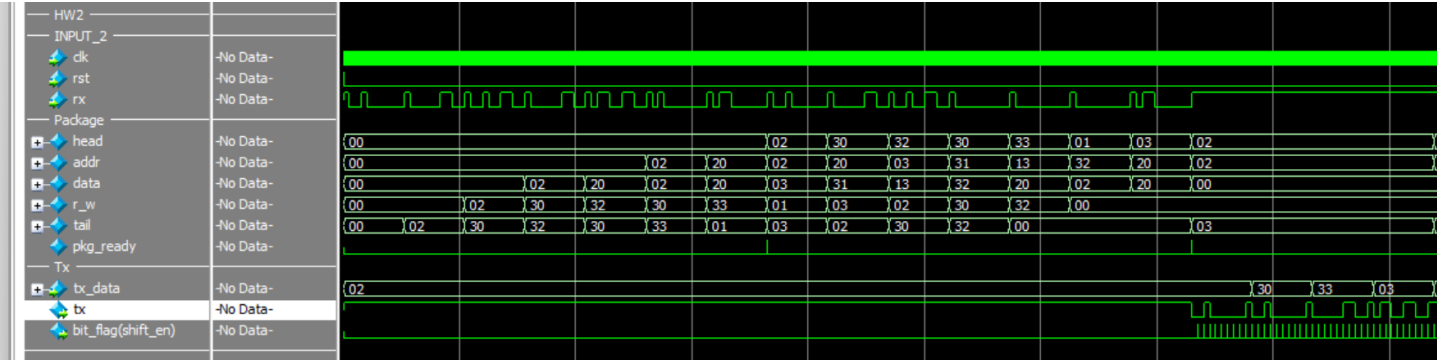
■ 實驗說明:

1. 承上題，接收並解析整個 package 資料，依格式將資料分類輸出，並結合 RS-232 Transmitter 將資料依照 package 格式回傳。
2. 輸入:
 - clk : 50M
 - rst
 - rx : 接收到資料(1-bit)
3. 輸出
 - head[7:0] : 8'h02
 - addr[7:0] : 解析完的真實 address
 - data[7:0] : 解析完的真實 data
 - r_w[7:0] : 讀取(8'h00)或是寫入(8'h01)
 - tail[7:0] : 8'h03
 - pkg_ready : 標示完整接收到 1 個 package 的 trigger。
 - tx_data[7:0] : 要給 transmitter 回傳的資料。
 - tx : 傳出去的資料(1-bit)
4. 測資:
 - 8'h02, 8'h30, 8'h32, 8'h30, 8'h33, 8'h01, 8'h03
 - 8'h02, 8'h30, 8'h32, 8'h00, 8'h00, 8'h00, 8'h03



■ 模擬結果與結果說明：

(HW2)



■ 結論與心得：

這次的作業比較難，就算少了一個 check sum，還是要花很多時間理解並完成，而且要用到的不只是這次的 rx，還結合了上次作業的 tx，像是 handshaking 的感覺，照著流程圖慢慢理解後，也發現了一些流程圖的小問題，並試著自己做改正。

程式碼有附上 Rx 各個方塊圖的標題，會比較好比對，Low_Pass_Filter 和之前用的是同一個程式碼，

Baud Rate 為 38400(bit/s)

Baud_Rate_Half 為其一半

(附上全部的波形圖)

