

超大型積體電路測試

VLSI testing

Homework III



國立清華大學

系所：電子所碩一

中文姓名：李聖謙、賴家均

英文姓名：Sheng-Qian-Li、Jia-Jun-Lai

學號：111063517、111063578

授課老師：黃錫瑜

分工

111063517 李聖謙: RTL Code(SRAM, BIST_March ,BIST_Checkerboard, Mux, Top)、Test bench 、Report

111063578 賴家均: RTL Code(BIST_March, BIST_Checkerboard, Top) 、 Test bench 、 Compiler 、 Report

(a) Write the **Verilog model for this SRAM** and verify it with a simple test bench to show the functional correctness. Show the simulation waveforms produced.

SRAM RTL CODE

```
1  module sram ( clk, rst_n, address, data_in, data_out, we_n, cs_n);
2
3  input clk;
4  input rst_n;
5  input [1:0] data_in;
6  output reg [1:0] data_out;
7
8  reg [3:0] space0 [3:0];
9  reg [3:0] space1 [3:0];
10 input [3:0] address;
11 input wire we_n;
12 input wire cs_n;
13
14 always@(posedge clk) begin
15   if (!rst_n) begin
16     space0[0] <= 0;
17     space0[1] <= 0;
18     space0[2] <= 0;
19     space0[3] <= 0;
20     space1[0] <= 0;
21     space1[1] <= 0;
22     space1[2] <= 0;
23     space1[3] <= 0;
24     data_out <= 2'bx;
25   end
26   else if (we_n == 0 && cs_n == 0) begin
27     space0[address[3:2][address[1:0]]] <= data_in[0];
28     space1[address[3:2][address[1:0]]] <= data_in[1];
29     data_out[0] <= 1'bx;
30     data_out[1] <= 1'bx;
31   end
32
33   else if(we_n == 1 && cs_n == 0)begin
34     data_out[0] <= space0[address[3:2][address[1:0]]];
35     data_out[1] <= space1[address[3:2][address[1:0]]];
36   end
37
38   else if(cs_n == 1)begin
39     data_out[0] <= 1'bx;
40     data_out[1] <= 1'bx;
41   end
42   else begin
43     space0[0] <= 0;
44     space0[1] <= 0;
45     space0[2] <= 0;
46     space0[3] <= 0;
47     space1[0] <= 0;
48     space1[1] <= 0;
49     space1[2] <= 0;
50     space1[3] <= 0;
51   end
52 end
53
54 endmodule
```

SRAM test bench

```

1  `timescale 1ns/1ps
2  `define CYCLE_TIME 10
3  module SRAM_tb;
4
5  //=====
6  // I/O PORTS
7  //=====
8  reg clk;
9  reg rst_n;
10 reg cs_n_tb;
11 reg we_n_tb;
12 reg [1:0] data_in_tb;
13 reg [1:0] data_in_tb2;
14 reg [3:0] address_tb;
15 wire [1:0] data_out;
16
17 sram sram(
18   .clk(clk),
19   .rst_n(rst_n),
20   .we_n(we_n_tb),
21   .cs_n(cs_n_tb),
22   .data_in(data_in_tb),
23   .data_out(data_out),
24   .address(address_tb)
25 );
26
27 //=====
28 // PARAMETERS & VARIABLES
29 //=====
30 parameter CYCLE      = `CYCLE_TIME;
31 parameter PATNUM     = 19;
32 integer SEED        = 122;
33 integer i, j, k, l,m,o,p;
34 reg [4:0] read_iteration;
35 reg [4:0] initial_num;
36 reg [4:0] interval;
37 reg data0;
38 reg data1;
39 reg in_valid;
40 reg [31:0] pattern_data;
41 reg [31:0] input_data;
42 reg [31:0] input_data2 ;
43 reg [31:0] input_data4 ;
44 reg [3:0] p ;
45
46 //=====
47 // Clock
48 //=====
49 initial clk = 1'b0;
50 always #(CYCLE/2.0) clk = ~clk;
51
52 //=====
53 // MAIN
54 //=====
55 initial main_task;
56
57 //=====
58 // TASKS
59 //=====
60 task main_task; begin
61   reset_task;
62   $display(`n\t\t\t\t DATA IN | DATA[1] DATA[0] | DATA OUT | DATA[1] DATA[0] `);
63   $display(`\t\t\t\t-----|----- -----|----- |----- ----- `);
64
65   for(j=0; j<1; j = j + 1) begin
66     input_pattern_task;
67     write_task;
68     read_task;
69     display_task2;
70     o = o+1 ;
71   end
72
73   for(l=0; l<PATNUM; l = l + 1) begin
74     input_pattern_task;
75     write_task2;
76     // input_task;
77     read_task2;
78     o = o+1 ;
79     if(read_iteration > 0) begin
80       display_task2;
81     end
82   end
83   repeat(2) @(negedge clk);
84   $finish;
85 end endtask
86
87 //=====
88 // Reset Task
89 //=====
90 task reset_task; begin
91   clk = 0;
92   in_valid = 0;
93   rst_n = 1;
94   i = 0;
95   j = 0;
96   k = 0;
97   l = 0;
98   m = 0;
99   o = 0;
100
101  #(CYCLE/2.0) rst_n = 0;
102  #(CYCLE/2.0) rst_n = 1;
103 end endtask
104

```

```

105 //*****
106 //      Input Task
107 //*****
108 v task input_pattern_task; begin
109     $write("\n==== PATTERN %0d =====",o+1);
110     in_valid = 1;
111     pattern_data[7:0] = ${random()} % 256;
112     pattern_data[15:8] = ${random()} % 256;
113     pattern_data[23:16] = ${random()} % 256;
114     pattern_data[31:24] = ${random()} % 256;
115     repeat(1) @(posedge clk);
116     in_valid = 0;
117 end endtask
118
119 //*****
120 //      Write Task
121 //*****
122 task write_task; begin
123     we_n_tb = 0 ;
124     cs_n_tb = 0 ;
125     $write("\nWRITE STATE  cs=%d we=%d",cs_n_tb,we_n_tb) ;
126     input_data = pattern_data;
127     for(i = 0; i < 16; i = i + 1) begin
128         input_task;
129         display_task;
130     end
131     input_data4 = input_data3 ;
132     //input_data = 32'bx;
133     data_in_tb = 2'bx;
134 end endtask
135
136 task write_task2; begin
137     cs_n_tb = ${random()%2};
138     we_n_tb = 0 ;
139     $write("\nWRITE STATE  cs=%d we=%d",cs_n_tb,we_n_tb);
140     if(cs_n_tb == 0) begin
141         input_data = pattern_data;
142         input_data4 = input_data3 ;
143     end
144     else begin
145         input_data = 'bx;
146         input_data4 = 32'bx;
147     end
148     pattern_data = 32'bx;
149     for( m = 0; m < ${random(SEED)}%16 ; m = m + 1) begin
150         input_task;
151         display_task;
152         //if(cs_n_tb==0)input_data4 = input_data3 ;
153     end
154     data_in_tb = 2'bx;
155 end endtask
156
157 task input_task; begin
158     if(j == 0) address_tb = 1;
159     else address_tb = ${random(SEED+1)}%16;
160     i = address_tb;
161     if(i == 0) begin
162         if(cs_n_tb==0) input_data3[31:30] = input_data[31:30];
163         data_in_tb = input_data[31:30];
164         data1 = data_in_tb[1];
165         data0 = data_in_tb[0];
166         repeat(1) @(posedge clk);
167     end
168     else if(i == 1) begin
169         if(cs_n_tb==0)input_data3[29:28] = input_data[29:28];
170         data_in_tb = input_data[29:28];
171         data1 = data_in_tb[1];
172         data0 = data_in_tb[0];
173         repeat(1) @(posedge clk);
174     end
175     else if(i == 2) begin
176         if(cs_n_tb==0)input_data3[27:26] = input_data[27:26];
177         data_in_tb = input_data[27:26];
178         data1 = data_in_tb[1];
179         data0 = data_in_tb[0];
180         repeat(1) @(posedge clk);
181     end
182     else if(i == 3) begin
183         if(cs_n_tb==0)input_data3[25:24] = input_data[25:24];
184         data_in_tb = input_data[25:24];
185         data1 = data_in_tb[1];
186         data0 = data_in_tb[0];
187         repeat(1) @(posedge clk);
188     end
189     else if(i == 4) begin
190         if(cs_n_tb==0)input_data3[23:22] = input_data[23:22];
191         data_in_tb = input_data[23:22];
192         data1 = data_in_tb[1];
193         data0 = data_in_tb[0];
194         repeat(1) @(posedge clk);
195     end
196     else if(i == 5) begin
197         if(cs_n_tb==0)input_data3[21:20] = input_data[21:20];
198         data_in_tb = input_data[21:20];
199         data1 = data_in_tb[1];
200         data0 = data_in_tb[0];
201         repeat(1) @(posedge clk);
202     end
203 end
204
205
206
207
208
209
210
211

```

```

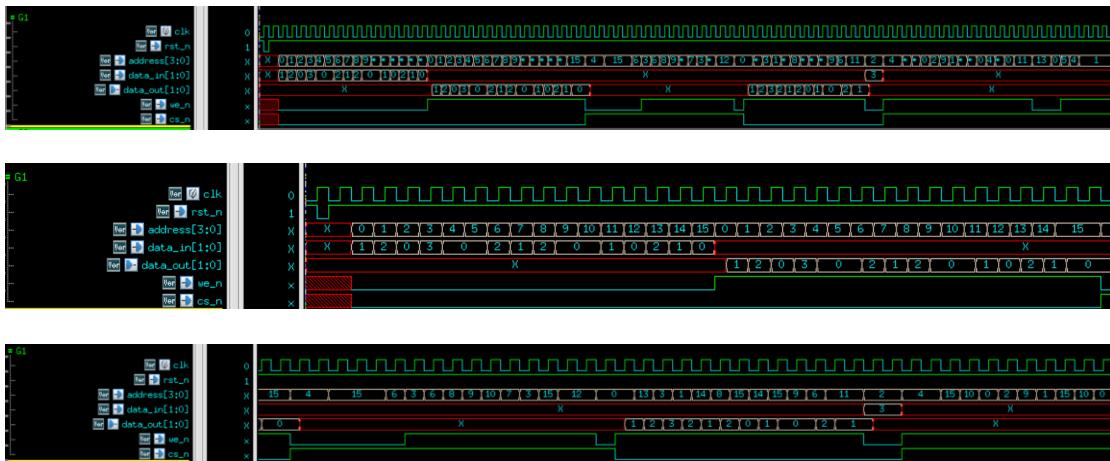
212     else if(i == 6) begin
213
214         if(cs_n_tb==0)input_data3[19:18] = input_data[19:18];
215         data_in_tb = input_data[19:18];
216         data1 = data_in_tb[1];
217         data0 = data_in_tb[0];
218         repeat(1) @ (negedge clk);
219     end
220     else if(i == 7) begin
221
222         if(cs_n_tb==0)input_data3[17:16] = input_data[17:16];
223         data_in_tb = input_data[17:16];
224         data1 = data_in_tb[1];
225         data0 = data_in_tb[0];
226         repeat(1) @ (negedge clk);
227     end
228     else if(i == 8) begin
229
230         if(cs_n_tb==0)input_data3[15:14] = input_data[15:14];
231         data_in_tb = input_data[15:14];
232         data1 = data_in_tb[1];
233         data0 = data_in_tb[0];
234         repeat(1) @ (negedge clk);
235     end
236     else if(i == 9) begin
237
238         if(cs_n_tb==0)input_data3[13:12] = input_data[13:12];
239         data_in_tb = input_data[13:12];
240         data1 = data_in_tb[1];
241         data0 = data_in_tb[0];
242         repeat(1) @ (negedge clk);
243     end
244     else if(i == 10) begin
245
246         if(cs_n_tb==0)input_data3[11:10] = input_data[11:10];
247         data_in_tb = input_data[11:10];
248         data1 = data_in_tb[1];
249         data0 = data_in_tb[0];
250         repeat(1) @ (negedge clk);
251     end
252     else if(i == 11) begin
253
254         if(cs_n_tb==0)input_data3[9:8] = input_data[9:8];
255         data_in_tb = input_data[9:8];
256         data1 = data_in_tb[1];
257         data0 = data_in_tb[0];
258         repeat(1) @ (negedge clk);
259     end
260     else if(i == 12) begin
261
262         if(cs_n_tb==0)input_data3[7:6] = input_data[7:6];
263         data_in_tb = input_data[7:6];
264         data1 = data_in_tb[1];
265         data0 = data_in_tb[0];
266         repeat(1) @ (negedge clk);
267     end
268     else if(i == 13) begin
269
270         if(cs_n_tb==0)input_data3[5:4] = input_data[5:4];
271         data_in_tb = input_data[5:4];
272         data1 = data_in_tb[1];
273         data0 = data_in_tb[0];
274         repeat(1) @ (negedge clk);
275     end
276     else if(i == 14) begin
277
278         if(cs_n_tb==0)input_data3[3:2] = input_data[3:2];
279         data_in_tb = input_data[3:2];
280         data1 = data_in_tb[1];
281         data0 = data_in_tb[0];
282         repeat(1) @ (negedge clk);
283     end
284     else if(i == 15) begin
285
286         if(cs_n_tb==0)input_data2[1:0] = input_data[1:0];
287         data_in_tb = input_data[1:0];
288         data1 = data_in_tb[1];
289         data0 = data_in_tb[0];
290         repeat(1) @ (negedge clk);
291     end
292 end endtask
293
294
295 task display_task; begin
296     $write("\n\address_tb %d:t\t\t %d\t\t %b\t\t %d\t\t %b", i, data_in_tb, data1, data0, data_out,data_out[1],data_out[0]);
297 end endtask
298 task display_task3; begin
299     $write("\n\address_tb %d:t\t\t %d\t\t %b\t\t ", k, data_in_tb2, data1, data0, );
300 end endtask
301 task display_task2; begin
302     $write("%d\t\t %b\t\t %b", data_out,data_out[1],data_out[0]);
303 end endtask
304
305 ****
306 // Read Task
307 //*****
308
309 task read_task; begin
310     $write("\nREAD STATE" );
311     data_in_tb = 2'bxx;
312     we_n_tb = 1;
313     cs_n_tb = 0;
314     for(k = 0; k < 16; k = k + 1) begin
315         if(k < 16) begin
316             address_tb = k;
317             input_data2 ;
318             if(k > 0) begin
319                 display_task2;
320
321             end
322             display_task3;
323             p = k ;
324             @(negedge clk);
325         end
326         else begin
327             address_tb = k;
328         end
329     end
330
331 end endtask
332
```

```

333
334
335 task input_data2; begin
336   k = address_tb ;
337   if(k == 0) begin
338     data_in_tb2 = input_data4[31:30];
339     data1 = data_in_tb2[1];
340     data0 = data_in_tb2[0];
341   end
342   else if(k == 1) begin
343     data_in_tb2 = input_data4[29:28];
344     data1 = data_in_tb2[1];
345     data0 = data_in_tb2[0];
346   end
347   else if(k == 2) begin
348     data_in_tb2 = input_data4[27:26];
349     data1 = data_in_tb2[1];
350     data0 = data_in_tb2[0];
351   end
352   else if(k == 3) begin
353     data_in_tb2 = input_data4[25:24];
354     data1 = data_in_tb2[1];
355     data0 = data_in_tb2[0];
356   end
357   else if(k == 4) begin
358     data_in_tb2 = input_data4[23:22];
359     data1 = data_in_tb2[1];
360     data0 = data_in_tb2[0];
361   end
362   else if(k == 5) begin
363     data_in_tb2 = input_data4[21:20];
364     data1 = data_in_tb2[1];
365     data0 = data_in_tb2[0];
366   end
367   else if(k == 6) begin
368     data_in_tb2 = input_data4[19:18];
369     data1 = data_in_tb2[1];
370     data0 = data_in_tb2[0];
371   end
372   else if(k == 7) begin
373     data_in_tb2 = input_data4[17:16];
374     data1 = data_in_tb2[1];
375     data0 = data_in_tb2[0];
376   end
377   else if(k == 8) begin
378     data_in_tb2 = input_data4[15:14];
379     data1 = data_in_tb2[1];
380     data0 = data_in_tb2[0];
381   end
382   else if(k == 9) begin
383     data_in_tb2 = input_data4[13:12];
384     data1 = data_in_tb2[1];
385     data0 = data_in_tb2[0];
386   end
387   else if(k == 10) begin
388     data_in_tb2 = input_data4[11:10];
389     data1 = data_in_tb2[1];
390     data0 = data_in_tb2[0];
391   end
392   else if(k == 11) begin
393     data_in_tb2 = input_data4[9:8];
394     data1 = data_in_tb2[1];
395     data0 = data_in_tb2[0];
396   end
397   else if(k == 12) begin
398     data_in_tb2 = input_data4[7:6];
399     data1 = data_in_tb2[1];
400     data0 = data_in_tb2[0];
401   end
402   else if(k == 13) begin
403     data_in_tb2 = input_data4[5:4];
404     data1 = data_in_tb2[1];
405     data0 = data_in_tb2[0];
406   end
407   else if(k == 14) begin
408     data_in_tb2 = input_data4[3:2];
409     data1 = data_in_tb2[1];
410     data0 = data_in_tb2[0];
411   end
412   else if(k == 15) begin
413     data_in_tb2 = input_data4[1:0];
414     data1 = data_in_tb2[1];
415     data0 = data_in_tb2[0];
416   end
417 end endtask
418
419
420 task read_task2; begin
421   cs_n_tb = ${random()}%2;
422   we_n_tb = 1;
423   read_iteration = ${random(SEED)} % 16;
424   $writeln("\nREAD STATE cs=%d we=%d read_iteration=%d",cs_n_tb,we_n_tb,read_iteration) ;
425
426
427   if (cs_n_tb == 0) input_data4 = input_data3 ;
428   else input_data4 = 32'b0;
429   for(n = 0; n < read_iteration; n = n + 1) begin
430     if(n < (read_iteration)) begin
431       address_tb = ${random(SEED)}%16;
432       input_data2 ;
433       if(n >0) begin
434         display_task2;
435
436       end
437       display_task3;
438       p = n ;
439       repeat(1) @(posedge clk);
440     end
441   end
442 end endtask
443
444 initial begin
445   // $sdf::annotate("test_syn.sdf", SRAM);
446   $fddbDumpfile("./sriram.fsdb");
447   $fddbDumpvars;
448 end
449 endmodule

```

Simulation waveforms



(CS=0, WE=0)寫入，(CS=0, WE=1)讀取，
CS=1 時不動作。

Display result

	DATA IN	DATA [1]	DATA [0]	DATA OUT	DATA [1]	DATA [0]
===== PATTERN 1 =====						
WRIRE STATE cs=0 we=0						
address_tb 0:	1	0	1	x	x	x
address_tb 1:	2	1	0	x	x	x
address_tb 2:	0	0	0	x	x	x
address_tb 3:	3	1	1	x	x	x
address_tb 4:	0	0	0	x	x	x
address_tb 5:	0	0	0	x	x	x
address_tb 6:	2	1	0	x	x	x
address_tb 7:	1	0	1	x	x	x
address_tb 8:	2	1	0	x	x	x
address_tb 9:	0	0	0	x	x	x
address_tb 10:	0	0	0	x	x	x
address_tb 11:	1	0	1	x	x	x
address_tb 12:	0	0	0	x	x	x
address_tb 13:	2	1	0	x	x	x
address_tb 14:	1	0	1	x	x	x
address_tb 15:	0	0	0	x	x	x
READ STATE						
address_tb 0:	1	0	1	1	0	1
address_tb 1:	2	1	0	2	1	0
address_tb 2:	0	0	0	0	0	0
address_tb 3:	3	1	1	3	1	1
address_tb 4:	0	0	0	0	0	0
address_tb 5:	0	0	0	0	0	0
address_tb 6:	2	1	0	2	1	0
address_tb 7:	1	0	1	1	0	1
address_tb 8:	2	1	0	2	1	0
address_tb 9:	0	0	0	0	0	0
address_tb 10:	0	0	0	0	0	0
address_tb 11:	1	0	1	1	0	1
address_tb 12:	0	0	0	0	0	0
address_tb 13:	2	1	0	2	1	0
address_tb 14:	1	0	1	1	0	1
address_tb 15:	0	0	0	0	0	0

首先 Cs=0、We=0 讓 SRAM 寫入 16 個數字，DATA IN 的個位十位數字分別記錄在 DATA IN[0]、DATA IN[1]並且顯示在工作站上，再來讓 Cs 維持 We=1 進入讀取模式，一樣讓 DATA OUT 個位十位數字分別記錄在 DATA OUT[0]、DATA OUT[1]並且也顯示在工作站上，最後進行比較確認有沒有相同。

```

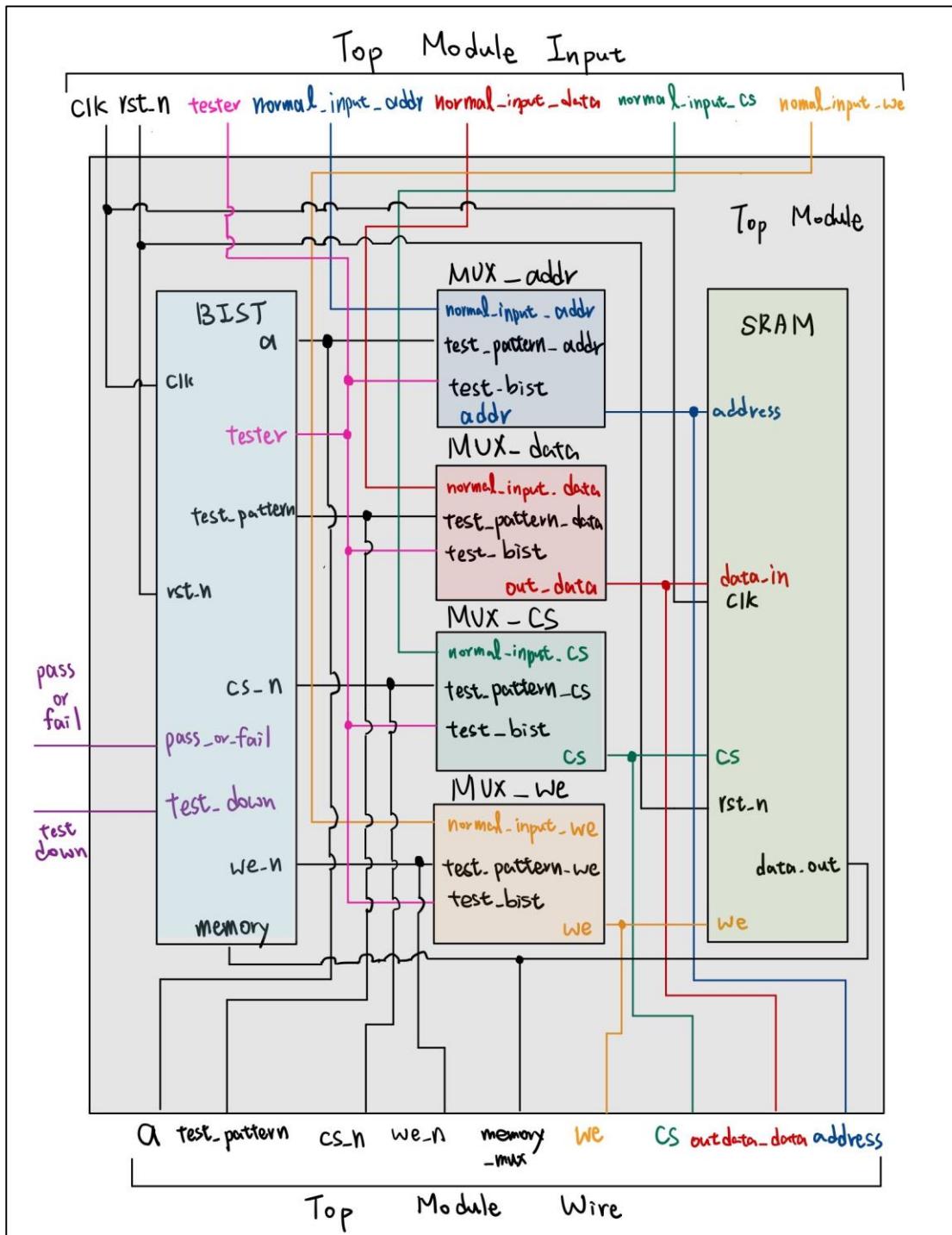
===== PATTERN 2 =====
WRIRE STATE cs=1 we=0
address_tb 4:          x      x      x      x      x      x      x
address_tb 4:          x      x      x      x      x      x      x
address_tb 15:         x      x      x      x      x      x      x
address_tb 15:         x      x      x      x      x      x      x
address_tb 15:         x      x      x      x      x      x      x
address_tb 6:          x      x      x      x      x      x      x
READ STATE cs=1 we=1 read_iteration= 9
address_tb 3:          x      x      x      x      x      x      x
address_tb 6:          x      x      x      x      x      x      x
address_tb 8:          x      x      x      x      x      x      x
address_tb 9:          x      x      x      x      x      x      x
address_tb 10:         x      x      x      x      x      x      x
address_tb 7:          x      x      x      x      x      x      x
address_tb 3:          x      x      x      x      x      x      x
address_tb 15:         x      x      x      x      x      x      x
address_tb 12:         x      x      x      x      x      x      x
===== PATTERN 3 =====
WRIRE STATE cs=1 we=0
address_tb 0:          x      x      x      x      x      x      x
READ STATE cs=0 we=1 read_iteration=12
address_tb 0:          1      0      1      1      0      1
address_tb 13:         2      1      0      2      1      0
address_tb 3:          3      1      1      3      1      1
address_tb 1:          2      1      0      2      1      0
address_tb 14:         1      0      1      1      0      1
address_tb 8:          2      1      0      2      1      0
address_tb 15:         0      0      0      0      0      0
address_tb 14:         1      0      1      1      0      1
address_tb 15:         0      0      0      0      0      0
address_tb 9:          0      0      0      0      0      0
===== PATTERN 4 =====
WRIRE STATE cs=0 we=0
address_tb 2:          3      1      1      x      x      x
address_tb 2:          3      1      1      x      x      x
READ STATE cs=1 we=1 read_iteration=15
address_tb 4:          x      x      x      x      x      x
address_tb 4:          x      x      x      x      x      x
address_tb 15:         x      x      x      x      x      x
address_tb 10:         x      x      x      x      x      x
address_tb 0:          x      x      x      x      x      x
address_tb 2:          x      x      x      x      x      x
address_tb 9:          x      x      x      x      x      x
address_tb 1:          x      x      x      x      x      x
address_tb 15:         x      x      x      x      x      x
address_tb 10:         x      x      x      x      x      x
address_tb 0:          x      x      x      x      x      x
address_tb 4:          x      x      x      x      x      x
address_tb 11:         x      x      x      x      x      x
address_tb 0:          x      x      x      x      x      x
address_tb 11:         x      x      x      x      x      x
===== PATTERN 5 =====
WRIRE STATE cs=1 we=0
address_tb 13:         x      x      x      x      x      x
address_tb 13:         x      x      x      x      x      x
address_tb 0:          x      x      x      x      x      x
READ STATE cs=1 we=1 read_iteration= 3
address_tb 5:          x      x      x      x      x      x
address_tb 4:          x      x      x      x      x      x
address_tb 1:          x      x      x      x      x      x$finish call

```

經過存入 16 數字跟讀取 16 個數字後，我們讓 Cs 跟 We 都是 random，做數次後結果顯示在工作站上，最後進行比較確認 SRAM 是否正常。

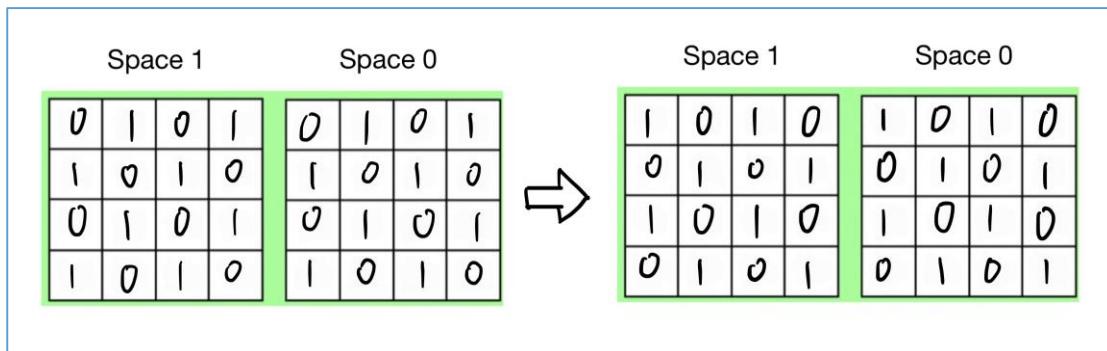
(b) Write a synthesizable Verilog code to realize the Built-In Self-Test (BIST) function for this SRAM block. Include the checkerboard test and the march test as discussed in class in your test algorithm. Replace the test bench in (a) with this BIST circuit and conduct the Verilog simulation again. Show the simulation waveforms produced.

Top module 接線圖

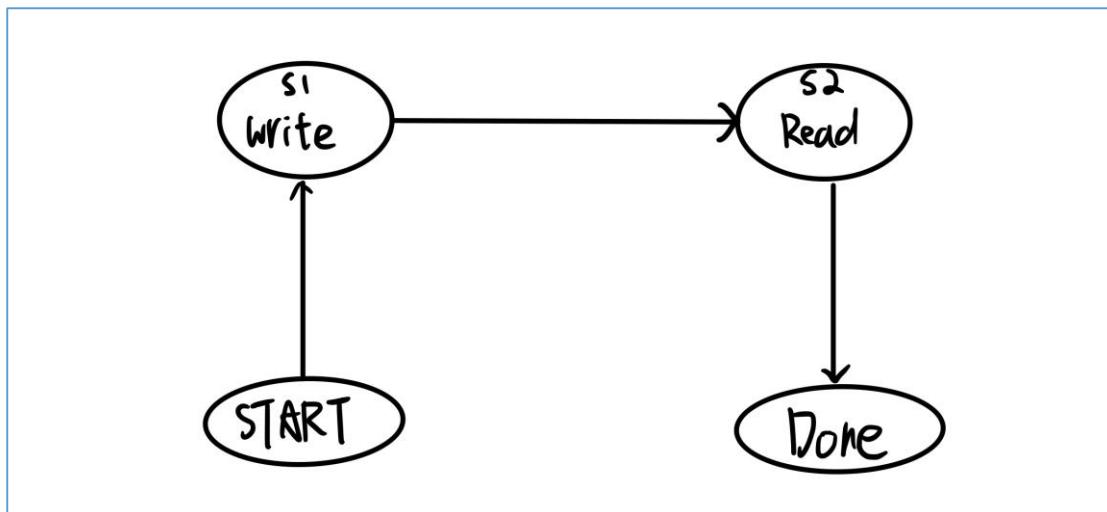


Checkerboard

做 Checkerboard 時候也會遇到同個問題是因為 SRAM 宣告的關係，所以輸入值的時候要改成輸入 3 才可以兩個 register 都填到想要的值。



Checkerboard 的 Finite State Machine



使用 4 個 states，一開始在 START 然後進 S1 寫入 0 跟 3 寫成間格狀在進入 S2，最後再做一次 S1 跟 S2 寫入 3 跟 0 然後到 Done。

RTL code Top module

```

1  `include "sram.v"
2  `include "bist_cb_mux_data.v"
3  `include "bist_cb_mux_addr.v"
4  `include "bist_cb_mux_cs.v"
5  `include "bist_cb_mux_we.v"
6  `include "BIST_checkerboard.v"
7  module cb_mux_top(
8      clk,
9      rst_n,
10
11     normal_inputs_data,
12     normal_inputs_addr,
13     normal_inputs_cs,
14     normal_inputs_we,
15     tester,
16
17     pass_or_fail,
18     test_down
19 );
20
21     input clk;
22     input rst_n;
23
24     input [1:0]normal_inputs_data;
25     input [3:0]normal_inputs_addr;
26     input normal_inputs_cs;
27     input normal_inputs_we;
28
29     input tester;
30
31     output pass_or_fail;
32     output test down;
33
34     wire [3:0] address;
35     wire [1:0] data_in;
36     wire [1:0] outdata_data;
37     wire [1:0] data_out;
38     wire [1:0] memory;
39     wire we_n, we, we_n_tb, we_n_cb;
40     wire cs_n, cs, cs_n_tb, cs_n_cb;
41
42     wire [1:0] test_pattren;
43     wire [3:0] aa;
44     wire [1:0] memory_mux;
45
46     wire [1:0] data_in_tb;
47     wire [3:0] address_tb;
48
49     wire [1:0] test_pattren_data;
50     wire [3:0] test_pattren_addr;
51     wire test_pattren_cs;
52     wire test_pattren_we;
53     wire test_bist;
54     wire [3:0]addr;
55
56     sram sram(
57         .clk(clk),
58         .rst_n(rst_n),
59         .address(address),
60         .data_in(outdata_data),
61         .we_n(we),
62         .cs_n(cs),
63         .data_out(memory_mux)
64 );
65
66 //=====
67 //      MUX
68 //=====
69 cb_mux_data cb_mux_data(
70     .normal_inputs_data(normal_inputs_data),
71     .test_pattren_data(test_pattren),
72     .test_bist(tester),
73     .outdata_data(outdata_data)
74 );
75
76 cb_mux_addr cb_mux_addr(
77     .normal_inputs_addr(normal_inputs_addr),
78     .test_pattren_addr(aa),
79     .test_bist(tester),
80     .addr(address)
81 );
82
83 cb_mux_cs cb_mux_cs(
84     .normal_inputs_cs(normal_inputs_cs),
85     .test_pattren_cs(cs_n_cb),
86     .test_bist(tester),
87     .cs(cs)
88 );
89
90 cb_mux_we cb_mux_we(
91     .normal_inputs_we(normal_inputs_we),
92     .test_pattren_we(we_n_cb),
93     .test_bist(tester),
94     .we(we)
95 );
96
97 //=====
98 //      BIST
99 //=====
100 BIST_checkerboard BIST_checkerboard(
101     .clk(clk),
102     .rst_n(rst_n),
103     .test_pattren(test_pattren),
104     .aa(aa),
105     .cs_n_cb(cs_n_cb),
106     .we_n_cb(we_n_cb),
107     .memory(memory_mux),
108     .pass_or_fail(pass_or_fail),
109     .test_down(test_down)
110 );
111
112 endmodule

```

RTL code Mux module

Mux_address

```

1  module cb_mux_addr(
2    normal_inputs_addr,
3    test_patten_addr,
4    test_bist,
5
6    addr
7  );
8
9  input [3:0] normal_inputs_addr;
10 input [3:0] test_patten_addr;
11 input test_bist;
12 output [3:0] addr;
13
14 wire [3:0] addr;
15
16 assign addr = (test_bist)? normal_inputs_addr:test_patten_addr;
17
18 endmodule

```

Mux_cs

```

1  module cb_mux_cs(
2    normal_inputs_cs,
3    test_patten_cs,
4    test_bist,
5
6    cs
7  );
8
9  input normal_inputs_cs;
10 input test_patten_cs;
11 input test_bist;
12 output cs;
13
14 wire cs;
15
16 assign cs = (test_bist)? normal_inputs_cs:test_patten_cs;
17
18 endmodule

```

Mux_data

```

1  module cb_mux_data(
2    normal_inputs_data,
3    test_patten_data,
4    test_bist,
5
6    outdata_data
7  );
8
9  input [1:0] normal_inputs_data;
10 input [1:0] test_patten_data;
11 input test_bist;
12 output [1:0] outdata_data;
13
14 wire [1:0]outdata_data;
15
16 assign outdata_data = (test_bist)? normal_inputs_data : test_patten_data;
17
18 endmodule

```

Mux_we

```

1  module cb_mux_we(
2    normal_inputs_we,
3    test_patten_we,
4    test_bist,
5
6    we
7  );
8
9  input normal_inputs_we;
10 input test_patten_we;
11 input test_bist;
12 output we;
13
14 wire we;
15
16 assign we = (test_bist)? normal_inputs_we:test_patten_we;
17
18 endmodule

```

RTL code Checkerboard module

```

1  //`include "sram.v"
2  module BIIST_checkerboard(
3    clk,
4    rst_n,
5    memory,
6
7    we_n_cb,
8    cs_n_cb,
9    aa,
10   test_patten,
11   pass_or_fail,
12   test_down
13 );
14
15   input clk;
16   input rst_n;
17   input [1:0]memory;
18
19   output we_n_cb;
20   output cs_n_cb;
21   output [3:0] aa;
22   output reg[1:0]test_patten;
23
24   output reg pass_or_fail;
25   output reg test_down;
26
27   reg[2:0] C_STATE , next_state;
28   reg we_n_cb ;
29   reg cs_n_cb ;
30
31   reg [3:0] a;
32   reg[4:0] b;
33   reg[1:0] c ;
34   reg[1:0] d ;
35   reg[3:0] count ; //比較用的
36   reg[3:0] count1 ; //S4切S5用
37   reg[3:0] address_compare ;
38   reg[3:0] address_compare_2 ;
39
40   parameter START = 2'b000;
41   parameter S1 = 2'b01;
42   parameter S2 = 2'b10;
43   parameter DONE = 2'b11;
44
45   always@(posedge clk) begin
46     if (!rst_n) begin
47       | C_STATE <= START;
48     end
49     else begin
50       | C_STATE <= next_state;
51     end
52   end
53
54   always @(*) begin
55     case(C_STATE)
56
57       START : if(a == 0) next_state = S1;
58       | else next_state = START;
59
60       S1 : if(a == 15) next_state = S2;
61       | else if (a < 16) next_state = S1;
62       | else next_state = S1;
63
64       S2 : if(a == 15 && b ==1) next_state = S1;
65       | else if(a == 15 && b ==2) next_state = DONE;
66       | else if(a < 16 ) next_state = S2;
67
68       | else next_state = DONE;
69
70       DONE : next_state = DONE;
71
72       default next_state = C_STATE;
73     endcase
74   end
75

```

```

76 //=====
77 //      a
78 //=====
79
80 always@(posedge clk) begin
81     if (!rst_n) begin
82         a <= 0;
83     end
84
85     else if(C_STATE == START)begin
86         if(next_state == S1) a <= 0;
87         else a <= a;
88     end
89
90     else if(C_STATE == S1)begin
91         a <= a + 1;
92     end
93
94     else if(next_state == S1)begin
95         if(C_STATE == S2 ) a <= 0;
96         else a <= a + 1;
97     end
98
99     else if(next_state == S2)begin
100        if(C_STATE == S1 ) a <= 0;
101        else a <= a + 1;
102    end
103    else if(next_state == DONE)begin
104        a <= a;
105    end
106
107    else a <= a;
108 end
109
110 //=====
111 //      b
112 //=====
113 always@(posedge clk) begin
114     if (!rst_n) begin
115         b <= 0;
116     end
117
118     else if (C_STATE == START) b <= 0;
119
120     else if(next_state == S1) b <= b;
121
122     else if(next_state == S2 && a==15) b <= b + 1 ;
123
124     else if(next_state == DONE) b <= b;
125
126     else b <= b;
127 end
128
129 //=====
130 //      we/cs
131 //=====
132 always@(posedge clk) begin
133     if (!rst_n) begin
134         we_n_cb <= 0;
135         cs_n_cb <= 0;
136     end
137
138     else if(C_STATE == START)begin
139         we_n_cb <= 0;
140         cs_n_cb <= 0;
141     end
142
143     else if(next_state == S1)begin
144         we_n_cb <= 0;
145         cs_n_cb <= 0;
146     end
147
148     else if(next_state == S2)begin
149         we_n_cb <= 1;
150         cs_n_cb <= 0;
151     end
152
153     else if(next_state == DONE)begin
154         cs_n_cb <= 0;
155     end
156
157 //=====
158 //      c
159 //=====
160 always@(posedge clk) begin
161     if (!rst_n) begin
162         c <= 0;
163     end
164
165
166     else if(C_STATE == START)begin
167         c <= 0;
168     end
169
170
171     else if(C_STATE == S1)begin
172         c <= c;
173     end
174
175
176     else if(C_STATE == S2)begin
177         if(next_state == S1) c <= 1;
178         else c <= c;
179     end
180
181     else if(next_state == DONE)begin
182         c <= 0;
183     end
184
185     else c <= c;
186 end

```

```

183 //=====
184 //      test_patten
185 //=====
186 always@(posedge clk) begin
187     if (!rst_n) begin
188         test_patten <= 2'b0;
189     end
190     else if(C_STATE == START)begin
191         test_patten <= 2'b0;
192     end
193     else if(next_state == 1 && a==15)begin
194         test_patten <= 2'b11;
195     end
196     else if(C_STATE == S1)begin
197         if(c == 0)begin
198             if(a == 1 || a == 7 || a==4 || a==6 || a==9 || a==12 || a==14) test_patten <= 2'b0;
199             else test_patten <= 2'b11;
200         end
201         else if(c == 1)begin
202             if(a == 0 || a == 2 || a==3 || a==5 || a==8 || a==10 || a==11 || a==13) test_patten <= 2'b0;
203             else test_patten <= 2'b11;
204         end
205         else test_patten <= test_patten;
206     end
207     else if(next_state == S2)begin
208         test_patten <= 2'bx;
209     end
210     else if(next_state == DONE)begin
211         test_patten <= test_patten;
212     end
213     else test_patten <= 0;
214 end
215
216 //=====
217 //      d
218 //=====
219
220 always@(posedge clk) begin
221     if (!rst_n) begin
222         d <= 0 ;
223     end
224     else if(C_STATE == START)begin
225         d <= 0 ;
226     end
227     else if(C_STATE == S2)begin
228         if(c == 0)begin
229             if(a == 0 || a == 2 || a==5 || a==7 || a==8 || a==10 || a==13 || a==15) d <= 2'b0;
230             else d <= 2'b11;
231         end
232         else if(c == 1)begin
233             if(a == 1 || a == 3 || a==4 || a==6 || a==9 || a==11 || a==12 || a==14) d <= 2'b0;
234             else d <= 2'b11;
235         end
236         else d <= d;
237     end
238
239     else if(next_state == DONE)begin
240         d <= d;
241     end
242     else d <= d;
243 end
244

```

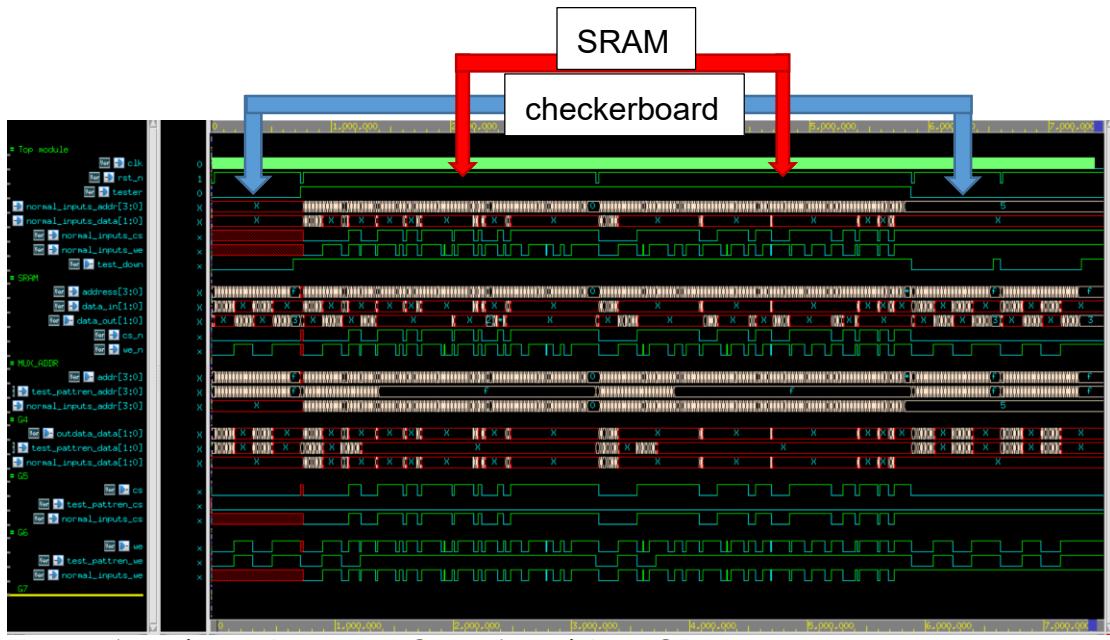
```

245 //=====
246 // address_compare
247 //=====
248 always@(posedge clk)
249 begin
250     if (!rst_n) begin
251         address_compare <= 4'bx;
252     end
253
254     else if(C_STATE == S1)begin
255         if(b ==1) address_compare <= 0;
256         else address_compare <= address_compare;
257     end
258
259     else if(C_STATE == S2)begin
260         address_compare <= a ;
261     end
262     else if(C_STATE == DONE)begin
263         address_compare <= address_compare;
264     end
265     else address_compare <= address_compare;
266 end
267 //=====
268 // address_compare_2
269 //=====
270 always@(posedge clk)
271 begin
272     if (!rst_n) begin
273         address_compare_2 <= 4'bx;
274     end
275
276     else if(C_STATE == S2)begin
277         address_compare_2 <= address_compare ;
278     end
279
280     else if(C_STATE == DONE)begin
281         address_compare_2 <= address_compare;
282     end
283
284     else address_compare_2 <= address_compare;
285 end
286
287
288 //=====
289 // pass_or_fail
290 //=====
291 always@(posedge clk) begin
292     if (!rst_n) begin
293         pass_or_fail <= 0 ;
294     end
295
296     else if(C_STATE == S1) begin
297         if(b ==1 && a==0)begin
298             if(d == memory ) pass_or_fail <= 0 ;
299             else pass_or_fail <= 1;
300         end
301         else pass_or_fail <= 0;
302     end
303
304     else if(C_STATE == S2)begin
305         if(a > 0)begin
306             if(d == memory) pass_or_fail <= 0 ;
307             else pass_or_fail <= 1;
308         end
309
310         else pass_or_fail <= 0 ;
311     end
312
313     else if(C_STATE == DONE) begin
314         if(c == 1)begin
315             if(d == memory) pass_or_fail <= 0 ;
316             else pass_or_fail <= 1;
317         end
318         else pass_or_fail <= 0;
319
320     end
321
322     else pass_or_fail <= 0 ;
323 end
324
325
326 always@ (posedge clk) begin
327     if (!rst_n) begin
328         test_down <= 0;
329     end
330     else if(C_STATE == DONE)begin
331         if(address_compare_2 == 15) test_down <= 1;
332         else test_down <= 0;
333     end
334     else test_down <= 0;
335
336
337 assign aa = a;
338
339 endmodule

```

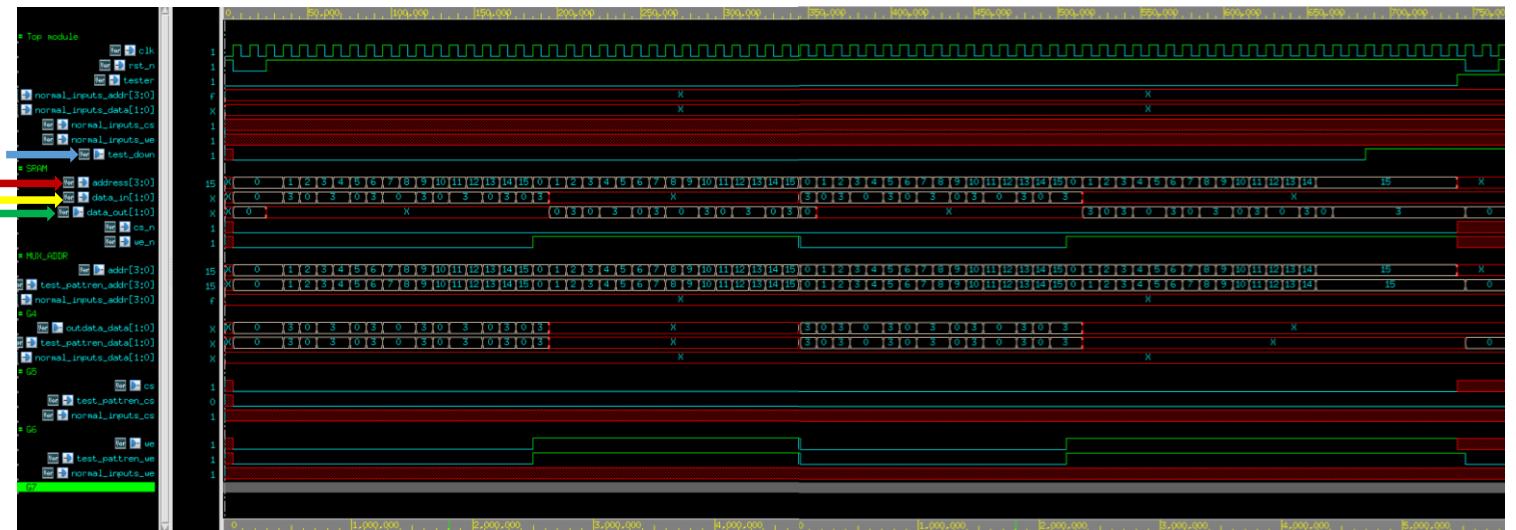
RTL SRAM module 沿用第(a)小題的 module

Checkerboard simulation waveforms



Tester 為 0 時電路執行 MARCH，為 1 時執行 SRAM。

Checkerboard 的部分



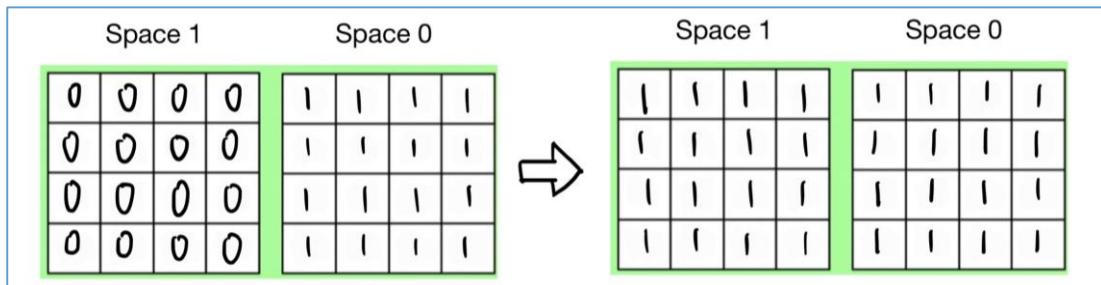
SRAM 的部分與第一題的波型相同

- Test_down: 開始測試時為 0，結束後為 1。
- Address: 輸入值對應到 SRAM 的位子。
- data_in: 輸入值給 SRAM。
- data_out: 從 SRAM 輸出的值。
- pass_or_fail: 哪個地方與預期的值不同就會顯示 1 (D 小題有圖片)
- address_compare_2: 配合 pass_or_fail 的位子，哪裡拉起表 SRAM 哪個地方有問題。(D 小題有圖片)

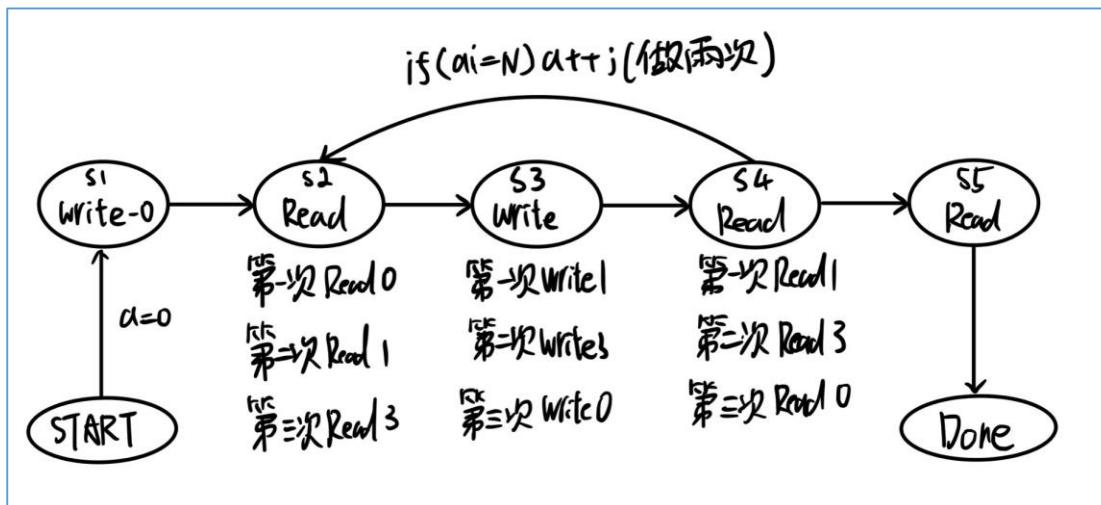
結果顯示：按造流程，輸入間格狀的 0 跟 3 紿我要的值，接著進入讀取模式並且判斷值有沒有問題，再來重新填入 3 跟 0(原本 0 的填 3, 3 的填 0)接著進入讀取模式並且判斷值有沒有問題。如有問題 pass_or_fail 會輸 1 且配合 address_compare 就可以得出 Sram 哪裡有問題。

March

在做 March 的時候，有一步驟是要把 SRAM 裡面的 bit 全部輸入一來測試，由於我們 RTL 在宣告 SRAM 的時是下面圖片的形式，宣告了兩個二維陣列，把輸入的 2 bit 的數拆成兩個數，個位數字存入 space 0 跟十位數字存入 space 1，為了讓兩個 register 都可以填滿 1，所以在寫入的階段時多做一次，把輸入改成 3 來達成目的。(如圖所示)



March 的 Finite State Machine



使用講義的方法用了 7 個 states，一開始在 START 然後進 S1 全部寫零再來跑 S2、S3 和 S4(填入 1)，跑完一次後再跑一次 S2、S3 和 S4(填入 3)然後 S5，最後當兩個 reg 都填入 1 後還要再倒回來全部填零，再跑一次 S2、S3 和 S4 然後 S5 最後到 Done 完成測試。

TOP module RTL code

```

1  `include "sram.v"
2  `include "bist_march_mux_data.v"
3  `include "bist_march_mux_addr.v"
4  `include "bist_march_mux_cs.v"
5  `include "bist_march_mux_we.v"
6  `include "BIST_MARCH.v"
7  module march_mux_top(
8    clk,
9    rst_n,
10
11   normal_inputs_data,
12   normal_inputs_addr,
13   normal_inputs_cs,
14   normal_inputs_we,
15   tester,
16
17   pass_or_fail,
18   test_down
19 );
20
21 input clk;
22 input rst_n;
23
24 input [1:0]normal_inputs_data;
25 input [3:0]normal_inputs_addr;
26 input normal_inputs_cs;
27 input normal_inputs_we;
28
29 input tester;
30
31 output pass_or_fail;
32 output test_down;
33
34 wire [3:0] address;
35 wire [1:0] data_in;
36 wire [1:0] outdata_data;
37 wire [1:0] data_out;
38 wire [1:0] memory;
39 wire we_n, we, we_n_tb, we_n_march;
40 wire cs_n, cs, cs_n_tb, cs_n_march;
41
42 wire [1:0] test_pattren;
43 wire [3:0] a;
44 wire [3:0] aa;
45 wire [1:0] memory_mux;
46
47 wire [1:0] data_in_tb;
48 wire [3:0] address_tb;
49
50 wire [1:0] test_pattren_data;
51 wire [3:0] test_pattren_addr;
52 wire test_pattren_cs;
53 wire test_pattren_we;
54 wire test_bist;
55 wire [3:0] addr;
56 wire clk_1, clk_2, rst_1, rst_2;
57
58 sram sram(
59   .clk(clk),
60   .rst_n(rst_n),
61   .address(address),
62   .data_in(outdata_data),
63   .we_n(we),
64   .cs_n(cs),
65   .data_out(memory_mux)
66 );
67
68 //=====
69 //      MUX
70 //=====
71 march_mux_data march_mux_data(
72   .normal_inputs_data(normal_inputs_data),
73   .test_pattren_data(test_pattren),
74   .test_bist(tester),
75   .outdata_data(outdata_data)
76 );
77
78 march_mux_addr march_mux_addr(
79   .normal_inputs_addr(normal_inputs_addr),
80   .test_pattren_addr(aa),
81   .test_bist(tester),
82   .addr(address)
83 );
84
85 march_mux_cs march_mux_cs(
86   .normal_inputs_cs(normal_inputs_cs),
87   .test_pattren_cs(cs_n_march),
88   .test_bist(tester),
89   .cs(cs)
90 );
91
92 march_mux_we march_mux_we(
93   .normal_inputs_we(normal_inputs_we),
94   .test_pattren_we(we_n_march),
95   .test_bist(tester),
96   .we(we)
97 );
98
99 //=====
100 //      BIST
101 //=====
102 BIST_MARCH BIST_MARCH(
103   .clk(clk),
104   .rst_n(rst_n),
105   .test_pattren(test_pattren),
106   .a(aa),
107   .cs_n_march(cs_n_march),
108   .we_n_march(we_n_march),
109   .memory(memory_mux),
110   .pass_or_fail(pass_or_fail),
111   .test_down(test_down)
112 );
113
114 assign clk_1 = (tester)? 0:clk;
115 assign rst_1 = (tester)? 1:rst_n;
116
117 endmodule

```

Mux module RTL code

Mux_address

```

1  module march_mux_addr(
2    normal_inputs_addr,
3    test_pattren_addr,
4    test_bist,
5
6    addr
7  );
8
9  input [3:0] normal_inputs_addr;
10 input [3:0] test_pattren_addr;
11 input test_bist;
12 output [3:0] addr;
13
14 wire [3:0] addr;
15
16 assign addr = (test_bist)? normal_inputs_addr:test_pattren_addr;
17
18 endmodule

```

Mux_CS

```

1  module march_mux_cs(
2    normal_inputs_cs,
3    test_pattren_cs,
4    test_bist,
5
6    cs
7  );
8
9  input normal_inputs_cs;
10 input test_pattren_cs;
11 input test_bist;
12 output cs;
13
14 wire cs;
15
16 assign cs = (test_bist)? normal_inputs_cs:test_pattren_cs;
17
18 endmodule

```

Mux_data

```

1  module march_mux_data(
2    normal_inputs_data,
3    test_pattren_data,
4    test_bist,
5
6    outdata_data
7  );
8
9  input [1:0] normal_inputs_data;
10 input [1:0] test_pattren_data;
11 input test_bist;
12 output [1:0] outdata_data;
13
14 wire [1:0]outdata_data;
15
16 assign outdata_data = (test_bist)? normal_inputs_data:test_pattren_data;
17
18 endmodule

```

Mux_we

```

1  module march_mux_we(
2    normal_inputs_we,
3    test_pattren_we,
4    test_bist,
5
6    we
7  );
8
9  input normal_inputs_we;
10 input test_pattren_we;
11 input test_bist;
12 output we;
13
14 wire we;
15
16 assign we = (test_bist)? normal_inputs_we:test_pattren_we;
17
18 endmodule

```

March module RTL code

```

1  `include "sram.v"
2  module BIST_MARCH(
3    clk,
4    rst_n,
5    test_pattren,
6    pass_or_fail ,
7    test_down
8  );
9
10 input clk;
11 input rst_n;
12 output reg[1:0]test_pattren; // 接data in
13 output reg[1:0]pass_or_fail;
14 output reg[1:0]test_down;
15
16 reg[4:0] C_STATE , next_state;
17 reg[3:0] a;
18 reg we_n ;
19 reg cs_n ;
20 reg[1:0] c ;
21 reg[1:0] d ;
22 reg[3:0] count ; //比較用的
23 reg[3:0] count1 ; //S4切S5用
24 reg[3:0] address_compare ;
25
26 wire [1:0] memory;
27
28 parameter START = 3'b000;
29 parameter S1 = 3'b001;           //寫入0
30 parameter S2 = 3'b010;           //讀addr
31 parameter S3 = 3'b011;           //
32 parameter S4 = 3'b100;
33 parameter S5 = 3'b101;
34 parameter DONE = 3'b110;
35
36 sram u1(
37   .clk(clk),
38   .rst_n(rst_n),
39   .address(a),
40   .we_n(we_n),
41   .cs_n(cs_n),
42   .data_in(test_pattren),
43   .data_out(memory)
44 );
45
46 always@(posedge clk ) begin
47   if (!rst_n) begin
48     | C_STATE <= START;
49   end
50   else begin
51     | C_STATE <= next_state;
52   end
53 end
54
55 always @(*) begin
56   case(C_STATE)
57
58   START : if(a == 0) next_state = S1;
59   | | else next_state = START;
60
61   S1 : if(a == 15) next_state = S2;
62   | | else if (a < 16) begin
63     | | | next_state = S1;
64   end
65   | | else next_state = S1;
66
67   S2 : if(a < 16) next_state = S3;
68   | | else next_state = S1;
69
70   S3 : next_state = S4;

```

```

71      S4 : if(a < 15) next_state = S2;
72      else if(a == 15) begin
73          if(count1 == 1)next_state = S5;
74          else if(count1 == 4)next_state = S5;
75          else next_state = S2;
76      end
77      else next_state = S2;
78
79      S5 : if(a == 15)
80          if(count1 == 2)next_state = S2;
81          else next_state = DONE;
82          else next_state = S5;
83
84      DONE : next_state = DONE;
85
86      default next_state = C_STATE;
87  endcase
88 end
89
90 //=====
91 //      a
92 //=====
93
94 always@ (posedge clk ) begin
95     if (!rst_n) begin
96         a <= 0;
97     end
98     else if(C_STATE == S1)begin
99         a <= a + 1;
100    end
101   else if(next_state == S2 && C_STATE == S1 )begin
102       if(C_STATE == S1 ) a <= 0;
103       else a <= a;
104   end
105   else if(C_STATE == S3)begin
106       a <= a ;
107   end
108   else if(C_STATE == S4)begin
109       if(a==15)a <= 0;
110       else a <= a + 1;
111   end
112   else if(C_STATE == S5)begin
113       a <= a + 1;
114   end
115   else a <= a;
116 end
117
118 //=====
119 //      we/cs
120 //=====
121 always@ (posedge clk ) begin
122     if (!rst_n) begin
123         we_n <= 2'b0;
124         cs_n <= 2'b0;
125         test_pattren <= 2'bx;
126     end
127     else if(C_STATE == START)begin
128         we_n <= 0;
129         cs_n <= 0;
130         test_pattren <= 0;
131     end
132     else if(next_state == S2)begin
133         we_n <= 1;
134         cs_n <= 0;
135         test_pattren <= 2'bx;
136     end
137     else if(next_state == S3)begin
138         we_n <= 0;
139         cs_n <= 0;
140         if(count1 == 0) test_pattren <= 1;
141         else if(count1 == 1) test_pattren <= 3;
142         else if(count1 == 3) test_pattren <= 1;
143         else if(count1 == 4) test_pattren <= 0;
144         else test_pattren <= 0;
145
146     end
147     else if(next_state == S4)begin
148         we_n <= 1;
149         cs_n <= 0;
150         test_pattren <= 2'bx;
151
152     end
153     else if(next_state == S5)begin
154         we_n <= 1;
155     end
156     else if(next_state == DONE)begin
157         cs_n <= 1;
158     end
159     else cs_n <= 0;
160 end
161
162 //=====
163 //      c
164 //=====
165 always@ (posedge clk ) begin
166     if (!rst_n) begin
167         c <= 0 ;
168     end
169
170     else if(C_STATE == START)begin
171         c <= 0 ;
172     end
173
174     else if(next_state == S2)begin
175         if(count1 == 0 && C_STATE == S4) begin
176             if(a == 15) c <= 1;
177             else c <= 0;
178         end
179         else if(count1 == 1) c <= 1;
180         else if (count1 == 3) begin
181             if(next_state == 4) c <= 1;
182             else if (a == 15) c <= 1;
183             else c <= 3;
184         end
185         else if (count1 == 4) c <= 1;
186         else c <= 0;
187     end
188
189     else if(next_state == S4)begin
190         c <= test_pattren;
191     end
192
193     else if(next_state == S5)begin
194         if(count1 == 2 || count1 == 1) c <= 3;
195         else c <= 0;
196     end
197
198     else if(next_state == DONE)begin
199         c <= 0;
200     end
201
202     else c <= c;
203 end
204
205 //=====
206 //      count1
207 //=====
208 always@ (posedge clk ) begin
209     if (!rst_n) begin
210         count1 <= 0 ;
211     end
212     else if(C_STATE == 4 && a == 15)begin
213         count1 <= count1 + 1 ;
214     end
215     else if(C_STATE == 5 && a == 15)count1 <= count1 + 1 ;
216     else count1 <= count1;
217 end
218
219 //=====
220 //      d
221 //=====
222 always@ (posedge clk ) begin
223     if (rst_n) begin
224         d <= 0 ;
225     end
226     else if(C_STATE == START)begin
227         d <= c ;
228     end
229     else if(next_state == S2)begin
230         d <= c;
231     end
232     else if(next_state == S3)begin
233         d <= c;
234     end
235     else if(next_state == S4)begin
236         d <= c;
237     end
238     else if(next_state == S5)begin
239         d <= c;
240     end
241     else if(next_state == DONE)begin
242         d <= c;
243     end
244     else d <= c;
245 end

```

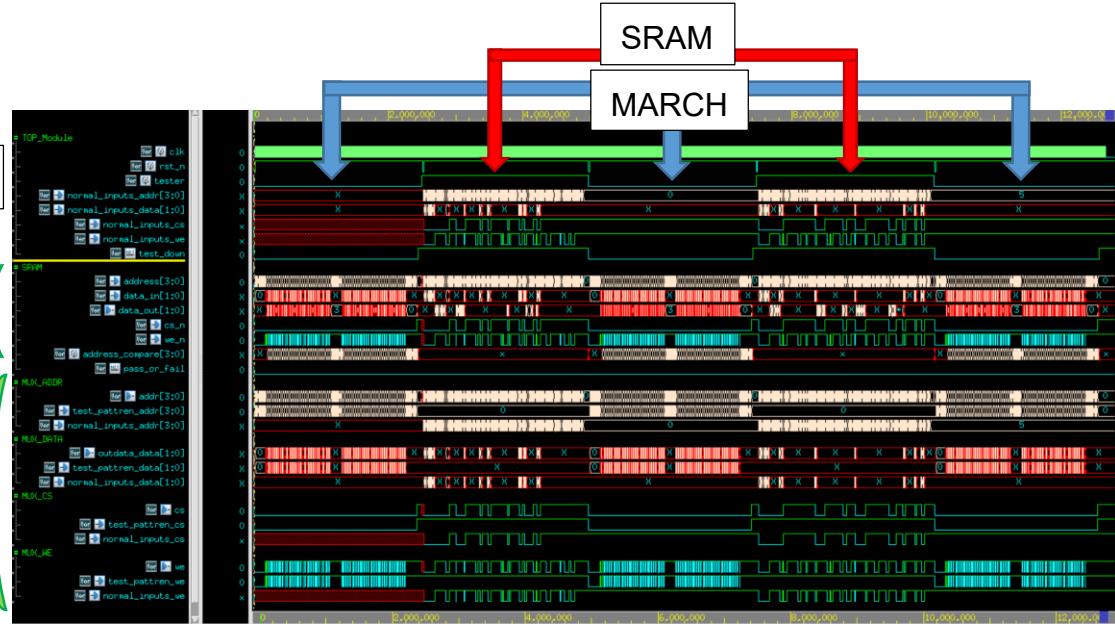
```

247 //=====
248 //      pass_or_fail
249 //=====
250 always@(posedge clk ) begin
251     if (!rst_n) begin
252         | pass_or_fail <= 1 ;
253     end
254     else if(count < 16)begin
255         if(C_STATE == S2)begin
256             if(a > 0)begin
257                 | if(d == memory) pass_or_fail <= 0 ;
258                 | else pass_or_fail <= 3;
259             end
260             else if ( count1 == 1) begin
261                 | if(d == memory) pass_or_fail <= 0 ;
262                 | else pass_or_fail <= 3;
263             end
264             else if ( count1 == 3) begin
265                 | if(d == memory) pass_or_fail <= 0 ;
266                 | else pass_or_fail <= 3;
267             end
268             else pass_or_fail <= 1 ;
269         end
270
271         else if(C_STATE == S3) begin
272             if(d == memory) pass_or_fail <= 0 ;
273             else pass_or_fail <= 3;
274         end
275
276         else if(C_STATE == S5) begin
277             if(d == memory) pass_or_fail <= 0 ;
278             else pass_or_fail <= 3;
279         end
280
281         else if(C_STATE == DONE) begin
282             if(d == memory) pass_or_fail <= 0 ;
283             else pass_or_fail <= 3;
284         end
285
286         else pass_or_fail <= 1 ;
287     end
288     else pass_or_fail <= 1 ;
289 end
290
291 //=====
292 //      count
293 //=====
294 always@(posedge clk )
295 begin
296     if (!rst_n) begin
297         | count <= 1'bx;
298     end
299     else if(C_STATE == S2)begin
300         if(a == 0) count <= 0 ;
301         else count <= count + 1 ;
302     end
303     else if(C_STATE == S5) begin
304         if(a== 0) count <= 0 ;
305         else count <= count + 1 ;
306     end
307     else if(C_STATE == DONE) count <= 1'bx;
308     else count <= count;
309 end
310
311 //=====
312 //      address_compare
313 //=====
314 always@(posedge clk )
315 begin
316     if (!rst_n) begin
317         | address_compare <= 4'bx;
318     end
319     else if(C_STATE == S3)begin
320         | address_compare <= a;
321     end
322     else if(C_STATE == S5)begin
323         | address_compare <= count;
324     end
325     else if(C_STATE == S2)begin
326         | if(count1 == 3) address_compare <= count;
327     end
328     else if(C_STATE == DONE)begin
329         | if(count1 == 6) address_compare <= count;
330     end
331     else address_compare <= address_compare;
332 end
333
334 always@(posedge clk ) begin
335     if (!rst_n) begin
336         | test_down <= 0;
337     end
338     else if(C_STATE == DONE)begin
339         | if(address_compare == 14) test_down <= 0;
340         | else test_down <= 1;
341     end
342     else test_down <= 0;
343 end
344
345 endmodule

```

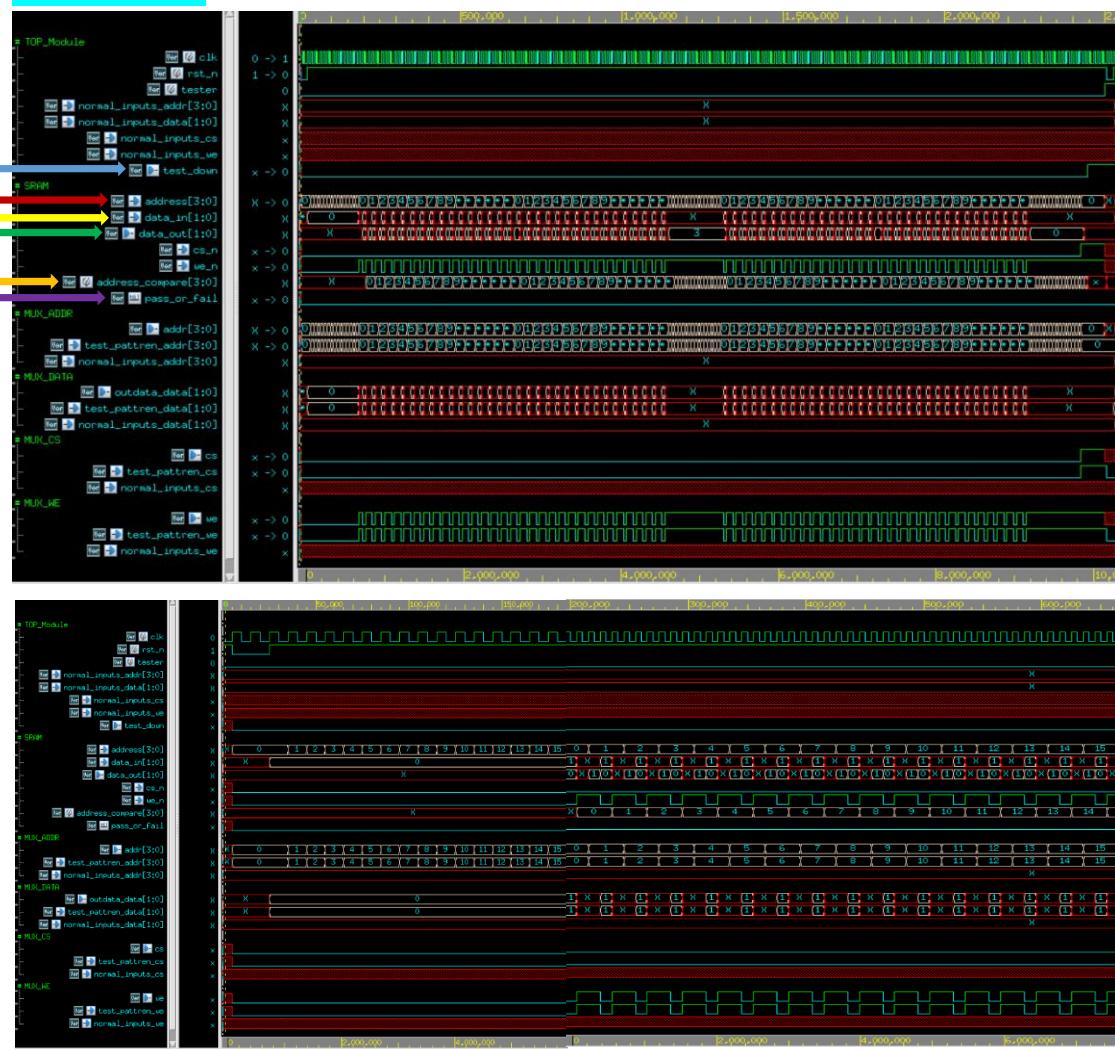
RTL SRAM module 沿用第(a)小題的 module

March simulation waveforms

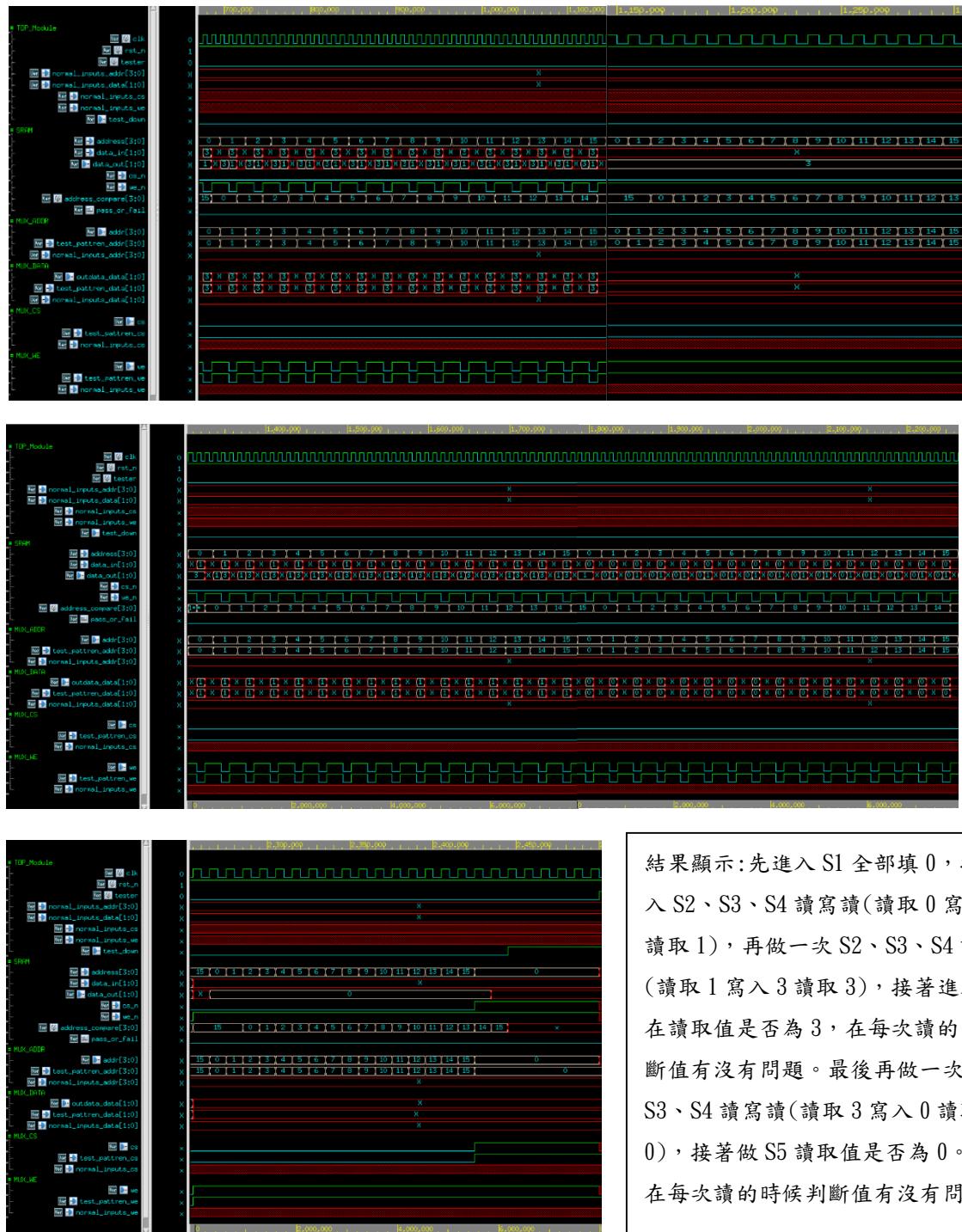


Tester 為 0 時電路執行 MARCH，為 1 時執行 SRAM。

March 的部分



底下有註
解分別對
應的作用



結果顯示:先進入 S1 全部填 0，接著進入 S2、S3、S4 讀寫讀(讀取 0 寫入 1
讀取 1)，再做一次 S2、S3、S4 讀寫讀
(讀取 1 寫入 3 讀取 3)，接著進入 S5
在讀取值是否為 3，在每次讀的時候判
斷值有沒有問題。最後再做一次 S2、
S3、S4 讀寫讀(讀取 3 寫入 0 讀取
0)，接著做 S5 讀取值是否為 0。也是
在每次讀的時候判斷值有沒有問題。

SRAM 的部分與第一題的波型相同

- **pass_or_fail**: 哪個地方與預期的值不同就會顯示 1，
- **address_compare**: 配合 **pass_or_fail** 的位子哪裡拉起表 SRAM 哪個地方有問題
- **Test_down**: 開始測試時為 0，結束後為 1。
- **Address**: 輸入值對應到 SRAM 的位子。
- **data_in**: 輸入值給 SRAM。
- **data_out**: 從 SRAM 輸出的值。

(c) Synthesize your BIST circuit using Design Compiler. Report the critical path delay of your BIST circuit.

At 100MHz

March

Point	Incr	Path
clock clk (rise edge)	0.00	0.00
clock network delay (ideal)	0.00	0.00
BIST_MARCH/count_1_reg_3_/CK (DFFQXL)	0.00	0.00 r
BIST_MARCH/count_1_reg_3/Q (DFFQXL)	0.28	0.28 f
BIST_MARCH/U72/Y (NOR2X2)	0.10	0.38 r
BIST_MARCH/U73/Y (NAND3XL)	0.17	0.55 f
BIST_MARCH/U21/Y (AOI21X1)	0.12	0.68 r
BIST_MARCH/U10/Y (INVX2)	0.04	0.72 f
BIST_MARCH/U63/Y (OAI32XL)	0.17	0.88 r
BIST_MARCH/U61/Y (OAI31X1)	0.09	0.97 f
BIST_MARCH/U60/Y (OAI21X1)	0.07	1.04 r
BIST_MARCH/U23/Y (INVX2)	0.06	1.10 f
BIST_MARCH/U12/Y (NAND2X2)	0.05	1.16 r
BIST_MARCH/U82/Y (AND3XL)	0.10	1.25 r
BIST_MARCH/U81/Y (NAND3XL)	0.16	1.41 f
BIST_MARCH/U27/Y (INVX2)	0.08	1.49 r
BIST_MARCH/U84/Y (NAND3XL)	0.09	1.58 f
BIST_MARCH/U83/Y (OAI21X1)	0.04	1.62 r
BIST_MARCH/cs_n_march1/reg/D (DFFQXL)	0.00	1.62 r
data arrival time		1.62
clock clk (rise edge)	10.00	10.00
clock network delay (ideal)	0.00	10.00
BIST_MARCH/cs_n_march1/reg/CK (DFFQXL)	0.00	10.00 r
library setup time	-0.07	9.93
data required time		9.93

data required time		9.93
data arrival time		-1.62

slack (MET)		8.31

Checkerboard

Point	Incr	Path
clock clk (rise edge)	0.00	0.00
clock network delay (ideal)	0.00	0.00
BIST_checkerboard/a_reg_1_/CK (DFFQXL)	0.00	0.00 r
BIST_checkerboard/a_reg_1/Q (DFFQXL)	0.24	0.24 r
BIST_checkerboard/U55/Y (NAND2X2)	0.08	0.32 f
BIST_checkerboard/U18/Y (NOR3X1)	0.23	0.56 r
BIST_checkerboard/U54/Y (NAND3XL)	0.13	0.68 f
BIST_checkerboard/U52/Y (OAI211XL)	0.11	0.79 r
BIST_checkerboard/U50/Y (AOI22XL)	0.10	0.89 f
BIST_checkerboard/U49/Y (AOI2BB2X2)	0.14	1.04 f
BIST_checkerboard/U53/Y (NOR3XL)	0.14	1.18 r
BIST_checkerboard/U19/Y (NOR3X1)	0.07	1.25 f
BIST_checkerboard/U57/Y (OR2X2)	0.10	1.35 f
BIST_checkerboard/U56/Y (AO22X2)	0.15	1.50 f
BIST_checkerboard/c_reg_0/D (DFFQXL)	0.00	1.50 f
data arrival time		1.50
clock clk (rise edge)	10.00	10.00
clock network delay (ideal)	0.00	10.00
BIST_checkerboard/c_reg_0/CK (DFFQXL)	0.00	10.00 r
library setup time	-0.08	9.92
data required time		9.92

data required time		9.92
data arrival time		-1.50

slack (MET)		8.42

結果顯示

March data arrival time = 1.62

Checkerboard data arrival time = 1.50

- (d) Perform gate-level simulation including your BIST circuit and the Verilog model for the SRAM block under test. Assume a clock rate of 100MHz. Does your BIST circuit report a test result of “pass”? If not, discuss why.

以下 waveform 的 pass or fail = 0 皆為 pass!!

pass or fail = 1 皆為 fail!!

Checkerboard => 皆為 pass!!



March => 皆為 pass!!



(結果顯示再 Checkerboard 跟 March 的時候 Pass_or_fail 值都是正確的所以顯示 0)

(e) Increase the clock rate gradually (i.e., from 100MHz to some higher rates) and check if at some point the test result becomes erroneous (i.e., turning from “pass” to “fail”). Discuss if you can derive the maximum operating speed of your BIST circuit in this experiment.

March

Point	Incr	Path
clock clk (rise edge)	0.00	0.00
clock network delay (ideal)	0.00	0.00
BIST_MARCH/C_STATE_reg_3/_CK (DFFHQX4)	0.00	0.00 r
BIST_MARCH/C_STATE_reg_3/_Q (DFFHQX4)	0.11	0.11 r
BIST_MARCH/U217/Y (INVX4)	0.03	0.14 f
BIST_MARCH/U22/Y (CLKAND2X8)	0.07	0.20 f
BIST_MARCH/U100/Y (NAND2BX4)	0.03	0.24 r
BIST_MARCH/U5/Y (INVX4)	0.02	0.26 f
BIST_MARCH/U47/Y (AND2X4)	0.07	0.33 f
BIST_MARCH/U62/Y (NAND2BX8)	0.07	0.40 f
BIST_MARCH/U145/Y (NAND3BX2)	0.13	0.53 f
BIST_MARCH/U143/Y (OAI32X1)	0.08	0.61 r
BIST_MARCH/U144/Y (INVX2)	0.07	0.68 f
BIST_MARCH/U113/Y (OAI2BB1X4)	0.05	0.73 r
BIST_MARCH/U109/Y (OAI222XL)	0.16	0.89 f
BIST_MARCH/address_compare_reg_1/_D (DFFXL)	0.00	0.89 f
data arrival time		0.89
clock clk (rise edge)	1.00	1.00
clock network delay (ideal)	0.00	1.00
BIST_MARCH/address_compare_reg_1/_CK (DFFXL)	0.00	1.00 r
library setup time	-0.11	0.89
data required time		0.89
data required time		0.89
data arrival time		-0.89
slack (MET)		0.00

Checkerboard

Point	Incr	Path
clock clk (rise edge)	0.00	0.00
clock network delay (ideal)	0.00	0.00
BIST_checkerboard/C_STATE_reg_1/_CK (DFFTRX1)	0.00	0.00 r
BIST_checkerboard/C_STATE_reg_1/_Q (DFFTRX1)	0.26	0.26 f
BIST_checkerboard/U63/Y (NOR3XL)	0.10	0.36 r
BIST_checkerboard/U126/Y (OAI2BB2X2)	0.10	0.47 r
BIST_checkerboard/U31/Y (CLKINVX2)	0.05	0.52 f
BIST_checkerboard/U80/Y (OR2XL)	0.14	0.66 f
BIST_checkerboard/U22/Y (NOR2XL)	0.09	0.75 r
BIST_checkerboard/U14/Y (NOR3X1)	0.07	0.81 f
BIST_checkerboard/U33/Y (OR2X4)	0.08	0.90 f
BIST_checkerboard/U32/Y (AO22X4)	0.12	1.02 f
BIST_checkerboard/c_reg_0/_D (DFFQXL)	0.00	1.02 f
data arrival time		1.02
clock clk (rise edge)	1.10	1.10
clock network delay (ideal)	0.00	1.10
BIST_checkerboard/c_reg_0/_CK (DFFQXL)	0.00	1.10 r
library setup time	-0.08	1.02
data required time		1.02
data required time		1.02
data arrival time		-1.02
slack (MET)		0.00

結果顯示在 slack 為零的時候：

MARCH 的最高運行速度 = 1/1 GHZ = **1 GHZ**

Checkerboard 的最高運行速度 = 1/1.1GHZ = **0.91 GHZ**

- (f) Try to inject a stuck-at-1 fault to bit #2 of the SRAM word #5, and report the test result of your BIST at 100MHz.

修改 : SRAM inject a stuck-at-1 fault to bit #2 of the SRAM word #5

```

43      else if(we_n == 1 && cs_n == 0)begin
44          // Fault_free
45          data_out[0] <= space0[address[3:2]][address[1:0]];
46          data_out[1] <= space1[address[3:2]][address[1:0]];
47
48          //Stuck at fault
49          /*if(address[3:2] == 1 && address[1:0] == 1)begin
50              data_out[0] <= space0[address[3:2]][address[1:0]];
51              data_out[1] <= 1'b1;
52          end
53          else begin
54              data_out[0] <= space0[address[3:2]][address[1:0]];
55              data_out[1] <= space1[address[3:2]][address[1:0]];
56          end*/
57      end

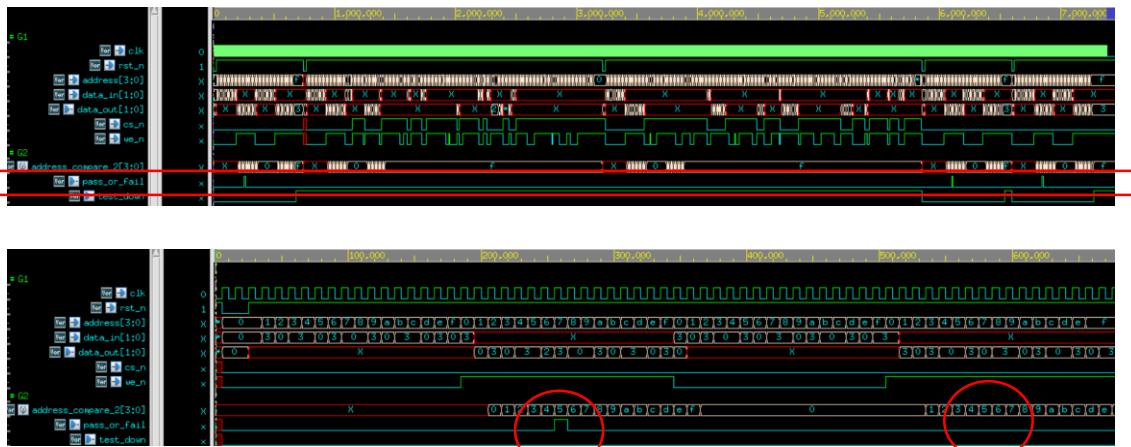
```

以下 waveform 的 pass or fail = 0 皆為 pass!!

pass or fail = 1 皆為 fail!!

Checkerboard

Pass
or
fail



我們在 SRAM 5 的 space[1] stuck-at-1，所以讓 word #5 的值只會出現 2 或 3，接著我們用 Checkerboard 來測試，由於前半段 word #5 是輸入 0 所以在讀的階段因為 word #5 的值不符合預期所以 pass_or_fail 輸入 1，但是後半段因為 word #5 是輸入 3 所以沒辦法測出這個錯誤 pass_or_fail 輸入 0。

March

Pass
or
fail



我們在 SRAM 5 的 space[1] stuck-at-1，所以讓 word #5 的值只會出現 2 或 3，接著我們用 March 來測試，由於第一次碰到 word #5 的時候分別要讀 0 跟 1 由於 word #5 的值不符所以 pass_or_fail 輸入兩次 1，第二次碰到 word #5 的時候分別要讀 1 跟 3 雖然第一個的值不符可是第二個值是對的所以沒辦法測出這個錯誤 pass_or_fail 輸入一次 1 跟一次 0，第三次碰到 word #5 的時候要讀 3 因為沒辦法測出這個錯誤 pass_or_fail 輸入 0，第四次碰到 word #5 的時候要讀 3 跟 0 第一個的值是對的沒辦法測出這個錯誤可是第二個值不符所以 pass_or_fail 輸入一次 0 跟一次 1，第五次碰到 word #5 的時候要讀 0 跟 0 由於 word #5 的值不符所以 pass_or_fail 輸入兩次 1，最後第六次碰到 word #5 的時候要讀 0 由於 word #5 的值不符所以 pass_or_fail 輸入 1。

Discussion

在寫 MARCH 跟 CK 的時後因為從 SRAM 傳出值會慢一拍，所以過程中要想辦法用 delay 或其他方法讓 SRAM 的輸出可以做比較，比較完後還要給他們正確的 address 才能讓 pass_or_fail 拉起來的時候對應到對的 address，所以才在 MARCH 跟 CK 上增加了 address compare 來區分跟原本 address 的不同。

然後在跑合成的時候發現了一個問題，就是透過公式我們可以大概知道最高的運行速度，可是在進行測試的時候當輸入最理想的值跑合成後的 testbench 都會跑不過，然後出現了(Warning: Operating condition slow set on design march_mux_top has different process, voltage and temperatures parameters than the parameters at which target library fast is characterized. Delays may be inaccurate as a result)，合理懷疑可能是製成檔案我或是其他參數造成有些微的誤差。