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

```
module sram ( clk, rst_n, address, data_in, data_out, we_n, cs_n);

input clk;
input rst_n;
input [1:0] data_in;
output reg [1:0] data_out;

reg [3:0] space0 [3:0];
reg [3:0] space1 [3:0];
input [3:0] address;
input wire we_n;
input wire cs_n;

always@(posedge clk) begin
    if (!rst_n) begin
        space0[0] <= 0;
        space0[1] <= 0;
        space0[2] <= 0;
        space0[3] <= 0;
        space1[0] <= 0;
        space1[1] <= 0;
        space1[2] <= 0;
        space1[3] <= 0;
        data_out <= 2'bx;
    end
    else if (we_n == 0 && cs_n == 0) begin
        space0[address[3:2]][address[1:0]] <= data_in[0];
        space1[address[3:2]][address[1:0]] <= data_in[1];
        data_out[0] <= 1'bx;
        data_out[1] <= 1'bx;
    end

    else if(we_n == 1 && cs_n == 0)begin
        data_out[0] <= space0[address[3:2]][address[1:0]];
        data_out[1] <= space1[address[3:2]][address[1:0]];
    end

    else if(cs_n == 1)begin
       data_out[0] <= 1'bx;
       data_out[1] <= 1'bx;
    end
    else begin
        space0[0] <= 0;
        space0[1] <= 0;
        space0[2] <= 0;
        space0[3] <= 0;
        space1[0] <= 0;
        space1[1] <= 0;
        space1[2] <= 0;
        space1[3] <= 0;
    end
end

endmodule
```

SRAM test bench

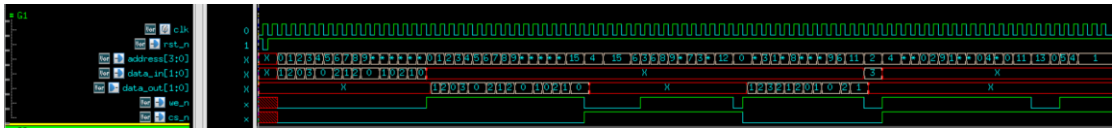
```
`timescale 1ns/1ps
`define CYCLE_TIME 10
module SRAM_tb;

//=====================================
//          I/O PORTS
//=====================================
reg clk;
reg rst_n;
reg cs_n_tb;
reg we_n_tb;
reg [1:0] data_in_tb;
reg [1:0] data_in_tb2;
reg [3:0] address_tb;
wire [1:0] data_out;

sram sram(
    .clk(clk),
    .rst_n(rst_n),
    .we_n(we_n_tb),
    .cs_n(cs_n_tb),
    .data_in(data_in_tb),
    .data_out(data_out),
    .address(address_tb)
);

//=====================================
//      PARAMETERS & VARIABLES
//=====================================
parameter CYCLE      = `CYCLE_TIME;
parameter PATNUM     = 19;
integer   SEED       = 122;
integer i, j, k, l,m,n,o;
reg [4:0] read_iteration;
reg [4:0] initial_num;
reg [4:0] interval;
reg data0;
reg data1;
reg in_valid;
reg [31:0] pattern_data;
reg [31:0] input_data;
reg [31:0] input_data3 ;
reg [31:0] input_data4 ;
reg [3:0] p ;

//=====================================
//              Clock
//=====================================
initial clk = 1'b0;
always #(CYCLE/2.0) clk = ~clk;

//=====================================
//              MAIN
//=====================================
initial main_task;

//=====================================
//              TASKS
//=====================================
task main_task; begin
    reset_task;
    $display("\n\t\t\t\t\t    DATA IN     |  DATA[1]    DATA[0]  |   DATA OUT   |  DATA[1]    DATA[0]  ");
    $display("\t\t\t\t\t    -------     |  -------    -------  |   --------   |  -------    -------  ");

    for(j=0; j<1; j = j + 1) begin
        input_pattern_task;
        write_task;
        read_task;
        display_task2;
        o = o+1 ;
    end

    for(l=0; l<PATNUM; l = l + 1) begin
        input_pattern_task;
        write_task2;
    //  input_task;
        read_task2;
        o = o+1 ;
        if(read_iteration > 0) begin
            display_task2;
            end
    end
    repeat(2) @(negedge clk);
    $finish;
end endtask

//**************************************
//      Reset Task
//**************************************
task reset_task; begin
    clk   = 0;
    in_valid = 0;
    rst_n       = 1;
    i = 0;
    j = 0;
    k = 0;
    l = 0;
    m = 0;
    o = 0;

    #(CYCLE/2.0) rst_n = 0;
    #(CYCLE/2.0) rst_n = 1;
end endtask

```

```
//**************************************
//      Input Task
//**************************************
task input_pattern_task; begin
    $write("\n\t============================================ PATTERN %0d ============================================",o+1);
    in_valid = 1;
    pattern_data[7:0] = {$random()} % 256;
    pattern_data[15:8] = {$random()} % 256;
    pattern_data[23:16] = {$random()} % 256;
    pattern_data[31:24] = {$random()} % 256;
    repeat(1) @(negedge clk);
    in_valid = 0;
end endtask

//**************************************
//      Write Task
//**************************************
task write_task; begin

    we_n_tb = 0 ;
    cs_n_tb = 0;
    $write("\nWRIRE STATE   cs=%d we=%d",cs_n_tb,we_n_tb) ;
    input_data = pattern_data;
    for(i = 0; i < 16; i = i + 1) begin
        input_task;
        display_task;
    end
    input_data4 = input_data3 ;
    //input_data = 32'bx;
    data_in_tb = 2'bx;
end endtask

task write_task2; begin

    cs_n_tb = {$random()}%2;
    we_n_tb = 0;
    $write("\nWRIRE STATE   cs=%d we=%d",cs_n_tb,we_n_tb);
    if(cs_n_tb == 0) begin
        input_data = pattern_data;
        input_data4 = input_data3 ;
    end
    else begin
    input_data = 'bx;
    input_data4 = 32'bx;
    end
    pattern_data = 32'bx;

    for( m = 0; m < {$random(SEED)}%16 ; m = m + 1) begin
            input_task;
            display_task;
            //if(cs_n_tb==0)input_data4 = input_data3 ;
    end

    data_in_tb = 2'bx;
end endtask

task input_task; begin
    if(j == 0) address_tb = i;
    else address_tb = {$random(SEED+1)}%16;
    i = address_tb;
    if(i == 0) begin
        if(cs_n_tb==0) input_data3[31:30] = input_data[31:30];
        data_in_tb = input_data[31:30];
        data1 = data_in_tb[1];
        data0 = data_in_tb[0];
        repeat(1) @(negedge clk);
    end
    else if(i == 1) begin
        if(cs_n_tb==0)input_data3[29:28] = input_data[29:28];
        data_in_tb = input_data[29:28];
        data1 = data_in_tb[1];
        data0 = data_in_tb[0];
        repeat(1) @(negedge clk);
    end
    else if(i == 2) begin

        if(cs_n_tb==0)input_data3[27:26] = input_data[27:26];
        data_in_tb = input_data[27:26];
        data1 = data_in_tb[1];
        data0 = data_in_tb[0];
        repeat(1) @(negedge clk);
    end
    else if(i == 3) begin

        if(cs_n_tb==0)input_data3[25:24] = input_data[25:24];
        data_in_tb = input_data[25:24];
        data1 = data_in_tb[1];
        data0 = data_in_tb[0];
        repeat(1) @(negedge clk);
    end
    else if(i == 4) begin

        if(cs_n_tb==0)input_data3[23:22] = input_data[23:22];
        data_in_tb = input_data[23:22];

        data1 = data_in_tb[1];
        data0 = data_in_tb[0];
        repeat(1) @(negedge clk);
    end
    else if(i == 5) begin

        if(cs_n_tb==0)input_data3[21:20] = input_data[21:20];
        data_in_tb = input_data[21:20];
        data1 = data_in_tb[1];
        data0 = data_in_tb[0];
        repeat(1) @(negedge clk);
    end
```

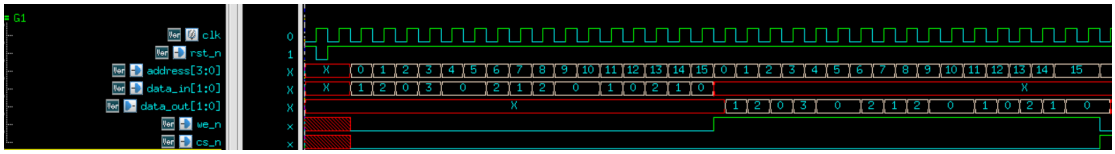
```
    else if(i == 6) begin

        if(cs_n_tb==0)input_data3[19:18] = input_data[19:18];
        data_in_tb = input_data[19:18];
        data1 = data_in_tb[1];
        data0 = data_in_tb[0];
        repeat(1) @(negedge clk);
    end
    else if(i == 7) begin

        if(cs_n_tb==0)input_data3[17:16] = input_data[17:16];
        data_in_tb = input_data[17:16];
        data1 = data_in_tb[1];
        data0 = data_in_tb[0];
        repeat(1) @(negedge clk);
    end
    else if(i == 8) begin

        if(cs_n_tb==0)input_data3[15:14] = input_data[15:14];
        data_in_tb = input_data[15:14];
        data1 = data_in_tb[1];
        data0 = data_in_tb[0];
        repeat(1) @(negedge clk);
    end
    else if(i == 9) begin

        if(cs_n_tb==0)input_data3[13:12] = input_data[13:12];
        data_in_tb = input_data[13:12];
        data1 = data_in_tb[1];
        data0 = data_in_tb[0];
        repeat(1) @(negedge clk);
    end
    else if(i == 10) begin

        if(cs_n_tb==0)input_data3[11:10] = input_data[11:10];
        data_in_tb = input_data[11:10];
        data1 = data_in_tb[1];
        data0 = data_in_tb[0];
        repeat(1) @(negedge clk);
    end
    else if(i == 11) begin

        if(cs_n_tb==0)input_data3[9:8] = input_data[9:8];
        data_in_tb = input_data[9:8];
        data1 = data_in_tb[1];
        data0 = data_in_tb[0];
        repeat(1) @(negedge clk);
    end
    else if(i == 12) begin

        if(cs_n_tb==0)input_data3[7:6] = input_data[7:6];
        data_in_tb = input_data[7:6];
        data1 = data_in_tb[1];
        data0 = data_in_tb[0];
        repeat(1) @(negedge clk);
    end
    else if(i == 13) begin

        if(cs_n_tb==0)input_data3[5:4] = input_data[5:4];
        data_in_tb = input_data[5:4];
        data1 = data_in_tb[1];
        data0 = data_in_tb[0];
        repeat(1) @(negedge clk);
    end
    else if(i == 14) begin

        if(cs_n_tb==0)input_data3[3:2] = input_data[3:2];
        data_in_tb = input_data[3:2];
        data1 = data_in_tb[1];
        data0 = data_in_tb[0];
        repeat(1) @(negedge clk);
    end
    else if(i == 15) begin

        if(cs_n_tb==0)input_data3[1:0] = input_data[1:0];
        data_in_tb = input_data[1:0];
        data1 = data_in_tb[1];
        data0 = data_in_tb[0];
        repeat(1) @(negedge clk);
    end
end endtask


task display_task; begin
    $write("\n\taddress_tb %0d:\t\t       %d        \t%b\t   %b         %d          %b          %b", i, data_in_tb, data1, data0, data_out,data_out[1],data_out[0]);
end endtask
task display_task3; begin
    $write("\n\taddress_tb %0d:\t\t       %d        \t%b\t   %b        ", k, data_in_tb2, data1, data0, );
end endtask
task display_task2; begin
    $write("%d          %b          %b", data_out,data_out[1],data_out[0]);
end endtask


//************************************
//      Read Task
//************************************
task read_task; begin
    $write("\nREAD STATE") ;
    data_in_tb = 2'bx;
    we_n_tb = 1;
    cs_n_tb = 0;
    for(k = 0; k < 16; k = k + 1) begin
        if(k < 16) begin
            address_tb = k;
            input_data2 ;
            if(k >0) begin
            display_task2;

            end
            display_task3;
            p = k ;
            @(negedge clk);
        end
        else begin
            address_tb = k;
        end
    end


end endtask
```

```
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     $write("\nREAD STATE  cs=%d we=%d read_iteration=%d",cs_n_tb,we_n_tb,read_iteration) ;
425
426
427     if (cs_n_tb == 0) input_data4 = input_data3 ;
428     else input_data4 = 32'bx ;
429     for(n = 0; n < read_iteration; n = n + 1) begin
430         if(n < (read_iteration)) begin
431             address_tb = {$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) @(negedge clk);
440         end
441     end
442 end endtask
443
444 initial begin
445     //$sdf_annotate("test_syn.sdf", SRAM);
446     $fsdbDumpfile("./sram.fsdb");
447     $fsdbDumpvars;
448 end
449 endmodule
```

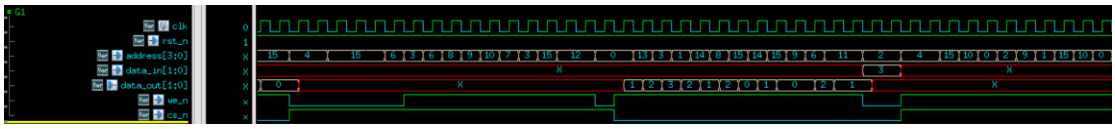
Simulation waveforms



插入 16 個數
字並且讀取



隨機插入數
字並且讀取

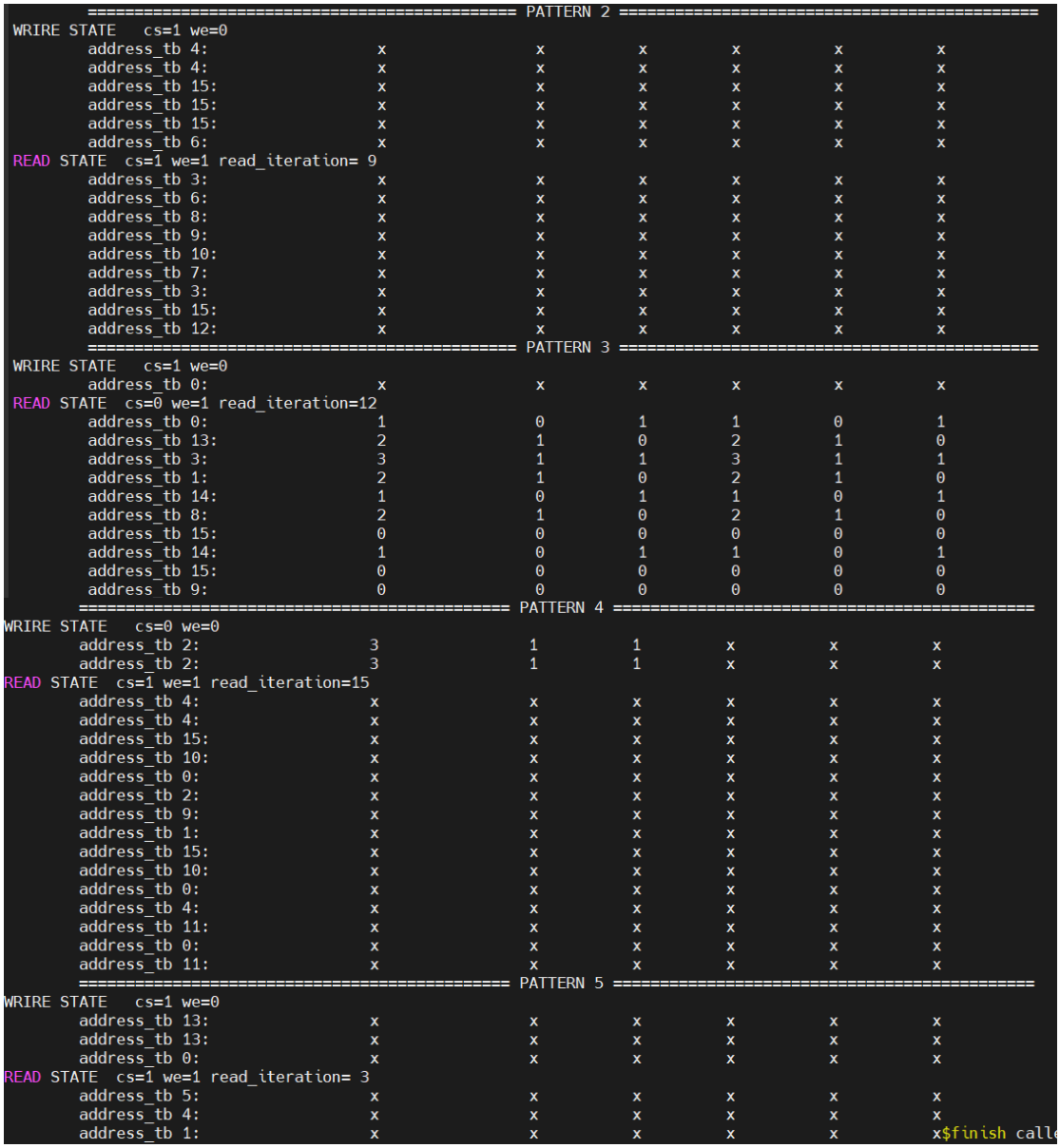


(CS=0, WE=0)寫入，(CS=0, WE=1)讀取，
CS=1 時不動作。

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
```

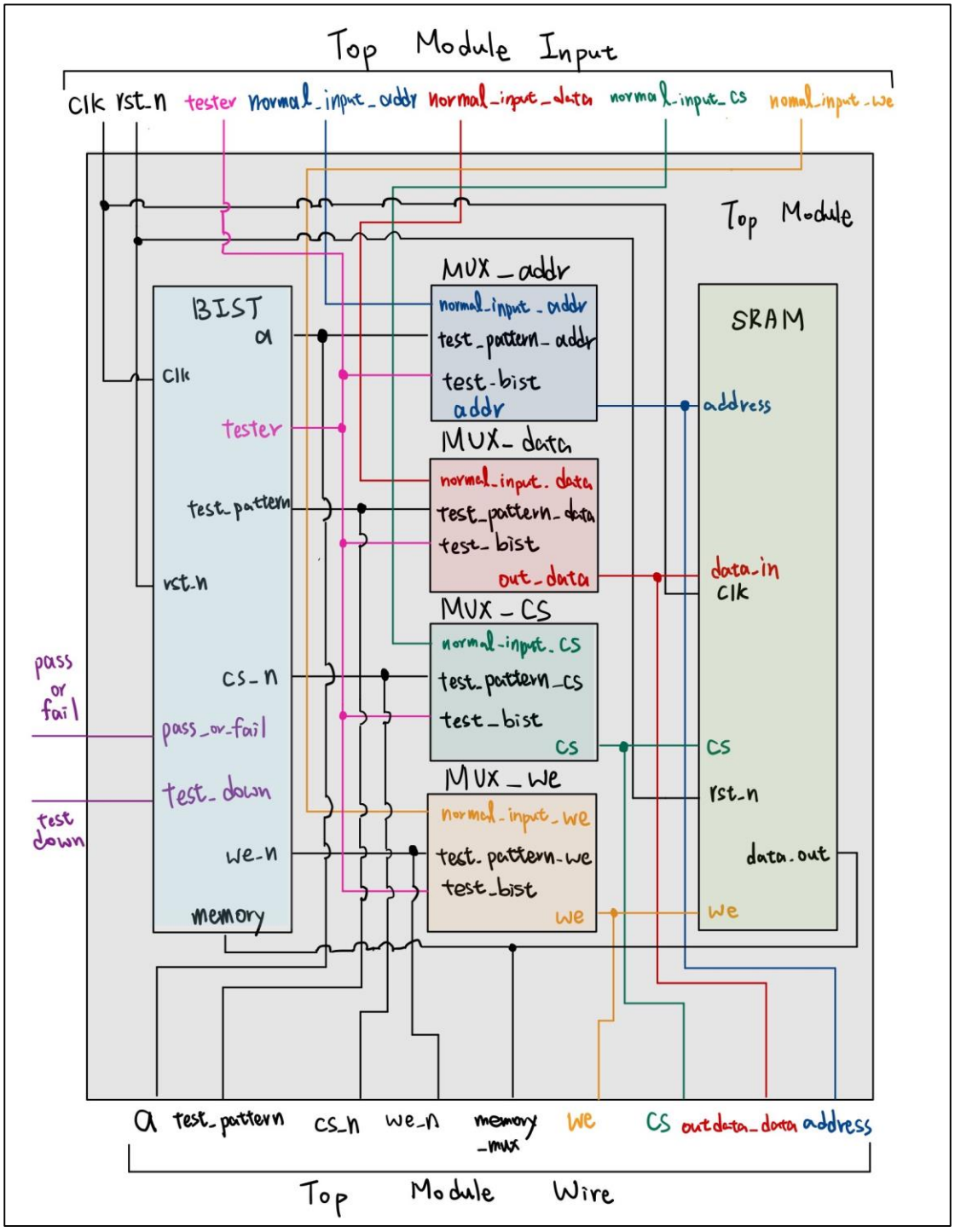
首先 Cs=0、We=0 讓 SRAM 寫入 16 個數字，DATA IN 的個位十位數字分別記錄在 DATA IN[0]、DATA IN[1]並且顯示在工作站上，再來讓 Cs 維持 We=1 進入讀取模式，一樣讓 DATA OUT 個位十位數字分別記錄在 DATA OUT[0]、DATA OUT[1]並且也顯示在工作站上，最後進行比較確認有沒有相同。



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。

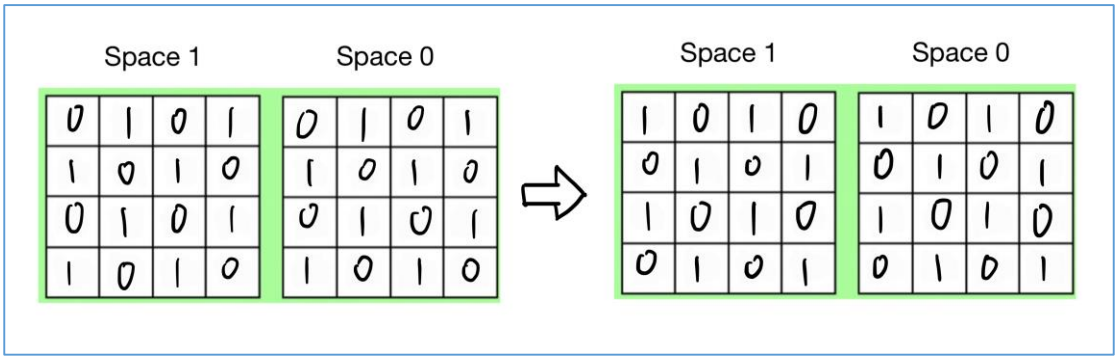
(b) Write a synthesizable Verilog code to realize the Built-In Self-Test (BIST) function for this SRAM block. Include the checkerboard test and the march test as discussed in class in your test algorithm. Replace the test bench in (a) with this BIST circuit and conduct the Verilog simulation again. Show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

Top module 接線圖

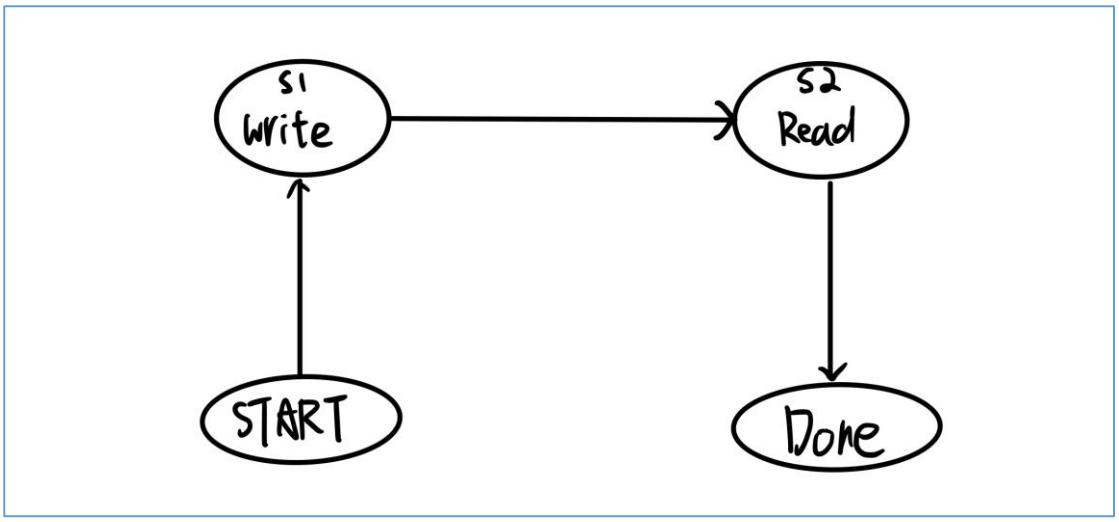


Checkerboard

做 Checkboard 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。



Checkboard 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

```v
`include "sram.v"
`include "bist_cb_mux_data.v"
`include "bist_cb_mux_addr.v"
`include "bist_cb_mux_cs.v"
`include "bist_cb_mux_we.v"
`include "BIST_checkerboard.v"
module cb_mux_top(
    clk,
    rst_n,

    normal_inputs_data,
    normal_inputs_addr,
    normal_inputs_cs,
    normal_inputs_we,
    tester,

    pass_or_fail,
    test_down
);

input clk;
input rst_n;

input [1:0]normal_inputs_data;
input [3:0]normal_inputs_addr;
input normal_inputs_cs;
input normal_inputs_we;

input tester;

output pass_or_fail;
output test_down;

wire [3:0] address;
wire [1:0] data_in;
wire [1:0] outdata_data;
wire [1:0] data_out;
wire [1:0] memory;
wire we_n, we, we_n_tb, we_n_cb;
wire cs_n, cs, cs_n_tb, cs_n_cb;

wire [1:0] test_pattren;
wire [3:0] aa;
wire [1:0] memory_mux;

wire [1:0] data_in_tb;
wire [3:0] address_tb;

wire [1:0] test_pattren_data;
wire [3:0] test_pattren_addr;
wire test_pattren_cs;
wire test_pattren_we;
wire test_bist;
wire [3:0]addr;

sram sram(
    .clk(clk),
    .rst_n(rst_n),
    .address(address),
    .data_in(outdata_data),
    .we_n(we),
    .cs_n(cs),
    .data_out(memory_mux)
);

//============================
//            MUX
//============================
cb_mux_data cb_mux_data(
    .normal_inputs_data(normal_inputs_data),
    .test_pattren_data(test_pattren),
    .test_bist(tester),
    .outdata_data(outdata_data)
);

cb_mux_addr cb_mux_addr(
    .normal_inputs_addr(normal_inputs_addr),
    .test_pattren_addr(aa),
    .test_bist(tester),
    .addr(address)
);

cb_mux_cs cb_mux_cs(
    .normal_inputs_cs(normal_inputs_cs),
    .test_pattren_cs(cs_n_cb),
    .test_bist(tester),
    .cs(cs)
);

cb_mux_we cb_mux_we(
    .normal_inputs_we(normal_inputs_we),
    .test_pattren_we(we_n_cb),
    .test_bist(tester),
    .we(we)
);

//============================
//           BIST
//============================
BIST_checkerboard BIST_checkerboard(
    .clk(clk),
    .rst_n(rst_n),
    .test_pattren(test_pattren),
    .aa(aa),
    .cs_n_cb(cs_n_cb),
    .we_n_cb(we_n_cb),
    .memory(memory_mux),
    .pass_or_fail(pass_or_fail),
    .test_down(test_down)
);

endmodule
```

RTL code Mux module

Mux_address

```
module cb_mux_addr(
    normal_inputs_addr,
    test_pattren_addr,
    test_bist,

    addr
);

input [3:0] normal_inputs_addr;
input [3:0] test_pattren_addr;
input test_bist;
output [3:0] addr;

wire [3:0] addr;

assign addr = (test_bist)? normal_inputs_addr:test_pattren_addr;

endmodule
```

Mux_cs

```
module cb_mux_cs(
    normal_inputs_cs,
    test_pattren_cs,
    test_bist,

    cs
);

input normal_inputs_cs;
input test_pattren_cs;
input test_bist;
output cs;

wire cs;

assign cs = (test_bist)? normal_inputs_cs:test_pattren_cs;

endmodule
```

Mux_data

```
module cb_mux_data(
    normal_inputs_data,
    test_pattren_data,
    test_bist,

    outdata_data
);

input [1:0] normal_inputs_data;
input [1:0] test_pattren_data;
input test_bist;
output [1:0] outdata_data;

wire [1:0]outdata_data;

assign outdata_data = (test_bist)? normal_inputs_data : test_pattren_data;

endmodule
```

Mux_we

```
module cb_mux_we(
    normal_inputs_we,
    test_pattren_we,
    test_bist,

    we
);

input normal_inputs_we;
input test_pattren_we;
input test_bist;
output we;

wire we;

assign we = (test_bist)? normal_inputs_we:test_pattren_we;

endmodule
```

RTL code Checkerboard module

```
//`include "sram.v"
module BIST_checkerboard(
    clk,
    rst_n,
    memory,

    we_n_cb,
    cs_n_cb,
    aa,
    test_pattren,
    pass_or_fail,
    test_down
);

input clk;
input rst_n;
input [1:0]memory;

output we_n_cb;
output cs_n_cb;
output [3:0] aa;
output reg[1:0]test_pattren;

output reg pass_or_fail;
output reg test_down;

reg[2:0]  C_STATE , next_state;
reg we_n_cb ;
reg cs_n_cb ;

reg [3:0] a;
reg[4:0] b;
reg[1:0] c ;
reg[1:0] d ;
reg[3:0] count ; //比較用的
reg[3:0] count1 ; //S4切S5用
reg[3:0] address_compare ;
reg[3:0] address_compare_2 ;

parameter START = 2'b00;
parameter S1 = 2'b01;
parameter S2 = 2'b10;
parameter DONE = 2'b11;

always@(posedge clk) begin
    if (!rst_n) begin
        C_STATE <= START;
    end
    else begin
        C_STATE <= next_state;
    end
end

always @(*) begin
    case(C_STATE)

        START : if(a == 0) next_state = S1;
                else next_state = START;

        S1 : if(a == 15) next_state = S2;
             else if (a < 16) next_state = S1;
             else next_state = S1;

        S2 : if(a == 15 && b ==1) next_state = S1;
             else if(a == 15 && b ==2) next_state = DONE;
             else if(a < 16 ) next_state = S2;

             else next_state = DONE;

        DONE : next_state = DONE;

        default next_state = C_STATE;
    endcase
end
```

```
//============================
//            a
//============================

always@(posedge clk) begin
    if (!rst_n) begin
        a <= 0;
    end

    else if(C_STATE == START)begin
        if(next_state == S1) a <= 0;
        else a <= a;
    end

    else if(C_STATE == S1)begin
        a <= a + 1;
    end

    else if(next_state == S1)begin
        if(C_STATE == S2 ) a <= 0;
        else  a <= a + 1;
    end

    else if(next_state == S2)begin
        if(C_STATE == S1 ) a <= 0;
        else  a <= a + 1;
    end
    else if(next_state == DONE)begin
        a <= a;
    end

    else a <= a;
end

//============================
//            b
//============================
always@(posedge clk) begin
    if (!rst_n) begin
        b <= 0;
    end

    else if (C_STATE == START) b <= 0;

    else if(next_state == S1) b <= b;

    else if(next_state == S2 && a==15) b <= b + 1 ;

    else if(next_state == DONE) b <= b;

    else b <= b;
end
```

```
//============================
//           we/cs
//============================
always@(posedge clk) begin
    if (!rst_n) begin
        we_n_cb <= 0;
        cs_n_cb <= 0;
    end
    else if(C_STATE == START)begin
        we_n_cb <= 0;
        cs_n_cb <= 0;
    end
    else if(next_state == S1)begin
        we_n_cb <= 0;
        cs_n_cb <= 0;
    end
    else if(next_state == S2)begin
        we_n_cb <= 1;
        cs_n_cb <= 0;
    end
    else if(next_state == DONE)begin
        cs_n_cb <= 0;
    end
    else cs_n_cb <= 0;
end

//============================
//            c
//============================
always@(posedge clk) begin
    if (!rst_n) begin
        c <= 0;
    end

    else if(C_STATE == START)begin
        c <= 0;
    end

    else if(C_STATE == S1)begin
        c <= c;
    end

    else if(C_STATE == S2)begin
        if(next_state == S1) c <= 1;
        else  c <= c;
    end

    else if(next_state == DONE)begin
        c <= 0;
    end

    else c <= c;
end
```

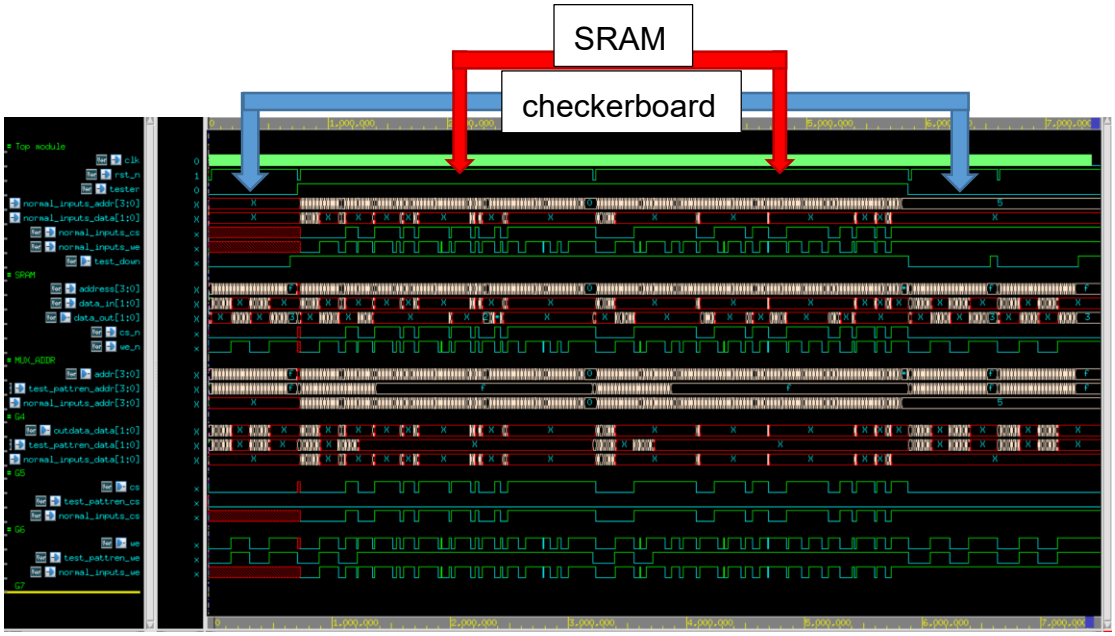
```
//============================
//        test_pattren
//============================
always@(posedge clk) begin
    if (!rst_n) begin
        test_pattren <= 2'b0;
    end
    else if(C_STATE == START)begin
        test_pattren <= 2'b0;
    end
    else if(next_state == 1 && a==15)begin
        test_pattren <= 2'b11;
    end
    else if(C_STATE == S1)begin
        if(c == 0)begin
            if(a == 1 || a == 7 || a==4 || a==6 || a==9  || a==12 || a==14) test_pattren <= 2'b0;
            else test_pattren <= 2'b11;
        end
        else if(c == 1)begin
            if(a == 0 || a == 2 || a==3 || a==5 || a==8 || a==10 || a==11 || a==13) test_pattren <= 2'b0;
            else test_pattren <= 2'b11;
        end
        else test_pattren <= test_pattren;
    end
    else if(next_state == S2)begin
        test_pattren <= 2'bx;
    end
    else if(next_state == DONE)begin
        test_pattren <= test_pattren;
    end
    else test_pattren <= 0;
end

//============================
//            d
//============================

always@(posedge clk) begin
    if (!rst_n) begin
        d <= 0 ;
    end
    else if(C_STATE == START)begin
        d <= 0 ;
    end
    else if(C_STATE == S2)begin
        if(c == 0)begin
            if(a == 0 || a == 2 || a==5 || a==7 || a==8 || a==10 || a==13 || a==15) d <= 2'b0;
            else d <= 2'b11;
        end
        else if(c == 1)begin
            if(a == 1 || a == 3 || a==4 || a==6 || a==9 || a==11 || a==12 || a==14) d <= 2'b0;
            else d <= 2'b11;
        end
        else d <= d;
    end

    else if(next_state == DONE)begin
        d <= d;
    end
    else d <= d;
end

```

```
//============================
//     address_compare
//============================
always@(posedge clk)
begin
    if (!rst_n) begin
        address_compare <= 4'bx;
    end

    else if(C_STATE == S1)begin
        if(b ==1) address_compare <= 0;
        else address_compare <= address_compare;
    end

    else if(C_STATE == S2)begin
        address_compare <= a ;
    end
    else if(C_STATE == DONE)begin
        address_compare <= address_compare;
    end
    else address_compare <= address_compare;
end

//============================
//     address_compare_2
//============================
always@(posedge clk)
begin
    if (!rst_n) begin
        address_compare_2 <= 4'bx;
    end

    else if(C_STATE == S2)begin
        address_compare_2 <= address_compare ;
    end

    else if(C_STATE == DONE)begin
        address_compare_2 <= address_compare;
    end

    else address_compare_2 <= address_compare;
end
```

```
//============================
//       pass_or_fail
//============================

always@(posedge clk) begin
    if (!rst_n) begin
        pass_or_fail <= 0 ;
    end

    else if(C_STATE == S1) begin
        if(b ==1 && a==0)begin
            if(d == memory ) pass_or_fail <= 0 ;
            else pass_or_fail <= 1;
        end
        else pass_or_fail <= 0;
    end

    else if(C_STATE == S2)begin
        if(a > 0)begin
            if(d == memory) pass_or_fail <= 0 ;
            else pass_or_fail <= 1;
        end

        else pass_or_fail <= 0 ;
    end

    else if(C_STATE == DONE) begin
        if(c == 1)begin
            if(d == memory) pass_or_fail <= 0 ;
            else pass_or_fail <= 1;
        end
        else pass_or_fail <= 0;

    end

    else pass_or_fail <= 0 ;
end

always@(posedge clk) begin
    if (!rst_n) begin
        test_down <= 0;
    end
    else if(C_STATE == DONE)begin
        if(address_compare_2 == 15) test_down <= 1;
        else test_down <= 0;
    end
    else test_down <= 0;
end

assign aa = a;

endmodule
```

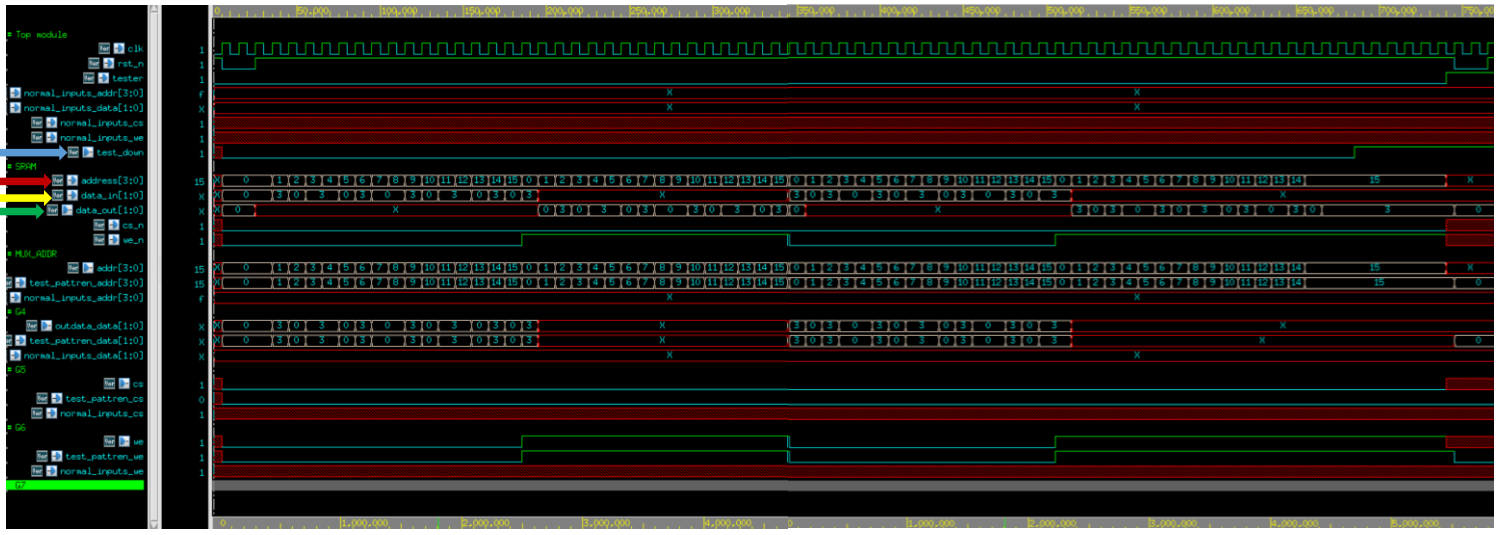
RTL SRAM module 沿用第(a)小題的 module

Checkerboard simulation waveforms



Tester 為 0 時電路執行 MARCH，為 1 時執行 SRAM。

Checkerboard 的部分



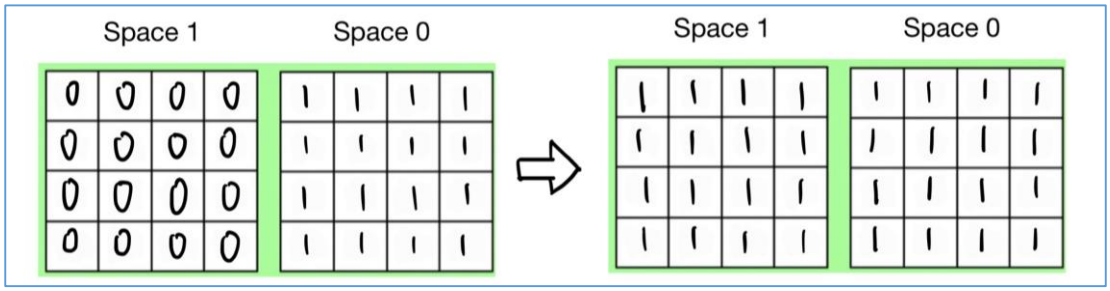
SRAM 的部分與第一題的波型相同

結果顯示:按造流程，輸入間格狀的 0 跟 3 給我要的值，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。

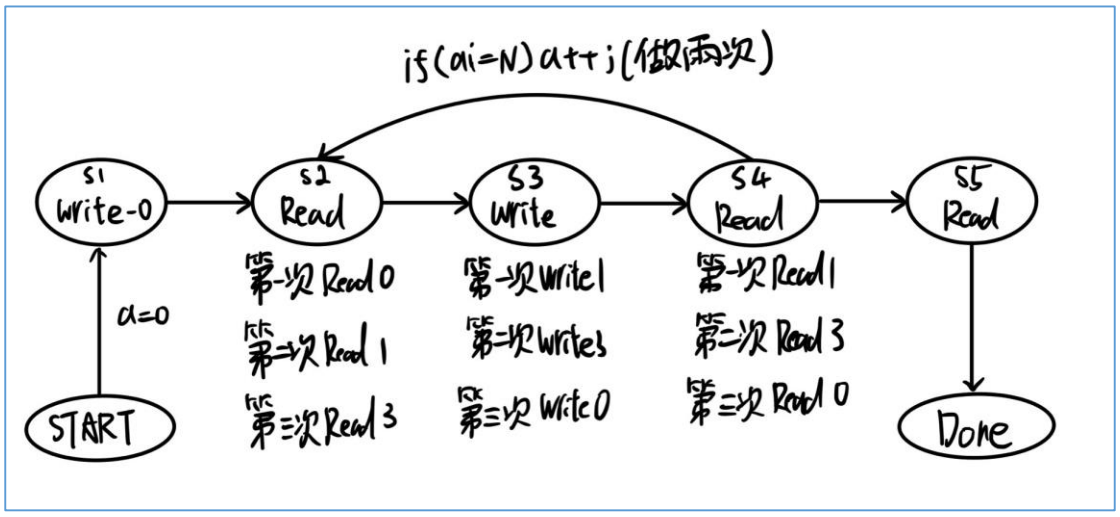
- Test_down:開始測試時為 0，結束後為 1。
- Address:輸入值對應到 SRAM 的位子。
- data_in:輸入值給 SRAM。
- data_out:從 SRAM 輸出的值。
- pass_or_fail:哪個地方與預期的值不同就會顯示 1 (D 小題有圖片)
- address_compare_2:配合 pass_or_fail 的位子，哪裡拉起表 SRAM 哪個地方有問題。(D 小題有圖片)

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

```v
`include "sram.v"
`include "bist_march_mux_data.v"
`include "bist_march_mux_addr.v"
`include "bist_march_mux_cs.v"
`include "bist_march_mux_we.v"
`include "BIST_MARCH.v"
module march_mux_top(
    clk,
    rst_n,

    normal_inputs_data,
    normal_inputs_addr,
    normal_inputs_cs,
    normal_inputs_we,
    tester,

    pass_or_fail,
    test_down
);

input clk;
input rst_n;

input [1:0]normal_inputs_data;
input [3:0]normal_inputs_addr;
input normal_inputs_cs;
input normal_inputs_we;

input tester;

output pass_or_fail;
output test_down;

wire [3:0] address;
wire [1:0] data_in;
wire [1:0] outdata_data;
wire [1:0] data_out;
wire [1:0] memory;
wire we_n, we, we_n_tb, we_n_march;
wire cs_n, cs, cs_n_tb, cs_n_march;

wire [1:0] test_pattren;
wire [3:0] a;
wire [3:0] aa;
wire [1:0] memory_mux;

wire [1:0] data_in_tb;
wire [3:0] address_tb;

wire [1:0] test_pattren_data;
wire [3:0] test_pattren_addr;
wire test_pattren_cs;
wire test_pattren_we;
wire test_bist;
wire [3:0] addr;
wire clk_1, clk_2, rst_1, rst_2;

sram sram(
    .clk(clk),
    .rst_n(rst_n),
    .address(address),
    .data_in(outdata_data),
    .we_n(we),
    .cs_n(cs),
    .data_out(memory_mux)
);

//============================
//           MUX
//============================
march_mux_data march_mux_data(
    .normal_inputs_data(normal_inputs_data),
    .test_pattren_data(test_pattren),
    .test_bist(tester),
    .outdata_data(outdata_data)
);


march_mux_addr march_mux_addr(
    .normal_inputs_addr(normal_inputs_addr),
    .test_pattren_addr(aa),
    .test_bist(tester),
    .addr(address)
);


march_mux_cs march_mux_cs(
    .normal_inputs_cs(normal_inputs_cs),
    .test_pattren_cs(cs_n_march),
    .test_bist(tester),
    .cs(cs)
);


march_mux_we march_mux_we(
    .normal_inputs_we(normal_inputs_we),
    .test_pattren_we(we_n_march),
    .test_bist(tester),
    .we(we)
);

//============================
//           BIST
//============================
BIST_MARCH BIST_MARCH(
    .clk(clk),
    .rst_n(rst_n),
    .test_pattren(test_pattren),
    .a(aa),
    .cs_n_march(cs_n_march),
    .we_n_march(we_n_march),
    .memory(memory_mux),
    .pass_or_fail(pass_or_fail),
    .test_down(test_down)
);

assign clk_1 = (tester)? 0:clk;
assign rst_1 = (tester)? 1:rst_n;

endmodule
```

Mux module RTL code

Mux_address

```
module march_mux_addr(
    normal_inputs_addr,
    test_pattren_addr,
    test_bist,

    addr
);

input [3:0] normal_inputs_addr;
input [3:0] test_pattren_addr;
input test_bist;
output [3:0] addr;

wire [3:0] addr;

assign addr = (test_bist)? normal_inputs_addr:test_pattren_addr;

endmodule
```

Mux_cs

```
module march_mux_cs(
    normal_inputs_cs,
    test_pattren_cs,
    test_bist,

    cs
);

input normal_inputs_cs;
input test_pattren_cs;
input test_bist;
output cs;

wire cs;

assign cs = (test_bist)? normal_inputs_cs:test_pattren_cs;

endmodule
```

Mux_data

```
module march_mux_data(
    normal_inputs_data,
    test_pattren_data,
    test_bist,

    outdata_data
);

input [1:0] normal_inputs_data;
input [1:0] test_pattren_data;
input test_bist;
output [1:0] outdata_data;

wire [1:0]outdata_data;

assign outdata_data = (test_bist)? normal_inputs_data:test_pattren_data;

endmodule
```

Mux_we

```
module march_mux_we(
    normal_inputs_we,
    test_pattren_we,
    test_bist,

    we
);

input normal_inputs_we;
input test_pattren_we;
input test_bist;
output we;

wire we;

assign we = (test_bist)? normal_inputs_we:test_pattren_we;

endmodule
```

March module RTL code

```
`include "sram.v"
module BIST_MARCH(
    clk,
    rst_n,
    test_pattren,
    pass_or_fail ,
    test_down
);

input clk;
input rst_n;
output reg[1:0]test_pattren; // 接data in
output reg[1:0]pass_or_fail;
output reg[1:0]test_down;

reg[4:0]  C_STATE , next_state;
reg[3:0] a;
reg we_n ;
reg cs_n ;
reg[1:0] c ;
reg[1:0] d ;
reg[3:0] count ;  //比較用的
reg[3:0] count1 ; //S4切S5用
reg[3:0] address_compare ;

wire [1:0] memory;

parameter START = 3'b000;
parameter S1 = 3'b001;        //寫入0
parameter S2 = 3'b010;        //讀addr
parameter S3 = 3'b011;        //
parameter S4 = 3'b100;
parameter S5 = 3'b101;
parameter DONE = 3'b110;

sram u1(
    .clk(clk),
    .rst_n(rst_n),
    .address(a),
    .we_n(we_n),
    .cs_n(cs_n),
    .data_in(test_pattren),
    .data_out(memory)
);

always@(posedge clk ) begin
    if (!rst_n) begin
        C_STATE <= START;
    end
    else begin
        C_STATE <= next_state;
    end
end

always @(*) begin
    case(C_STATE)

        START : if(a == 0) next_state = S1;
                else next_state = START;

        S1 : if(a == 15) next_state = S2;
             else if (a < 16) begin
                next_state = S1;
             end
             else next_state = S1;

        S2 : if(a < 16) next_state = S3;
             else next_state = S1;

        S3 : next_state = S4;
```

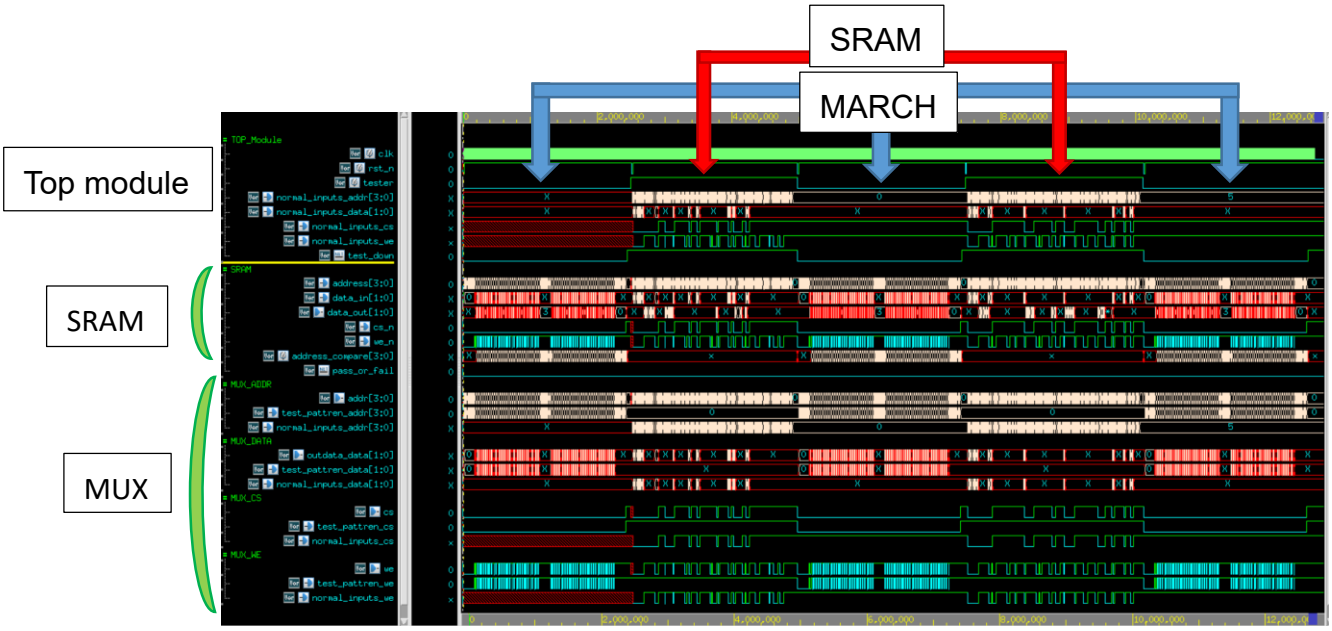
```
        S4 : if(a < 15) next_state = S2;
             else if(a == 15) begin
                if(count1 == 1)next_state = S5;
                else if(count1 == 4)next_state = S5;
                else next_state = S2;
             end
             else next_state = S2;

        S5 : if(a == 15)
                if(count1 == 2)next_state = S2;
                else next_state = DONE;
             else next_state = S5;

        DONE : next_state = DONE;

        default next_state = C_STATE;
    endcase
end

//============================
//            a
//============================
always@(posedge clk ) begin
    if (!rst_n) begin
        a <= 0;
    end
    else if(C_STATE == S1)begin
        a <= a + 1;
    end
    else if(next_state == S2 && C_STATE == S1 )begin
        if(C_STATE == S1 ) a <= 0;
        else  a <= a;
    end
    else if(C_STATE == S3)begin
        a <= a ;
    end
    else if(C_STATE == S4)begin
            if(a==15)a <= 0;
            else a <= a + 1;
    end
    else if(C_STATE == S5)begin
        a <= a + 1;
    end
    else a <= a;
end

//============================
//           we/cs
//============================
always@(posedge clk ) begin
    if (!rst_n) begin
        we_n <= 2'b0;
        cs_n <= 2'b0;
        test_pattren <= 2'bx;
    end
    else if(C_STATE == START)begin
        we_n <= 0;
        cs_n <= 0;
        test_pattren <= 0;
    end
    else if(next_state == S2)begin
        we_n <= 1;
        cs_n <= 0;
        test_pattren <= 2'bx;
    end
    else if(next_state == S3)begin
        we_n <= 0;
        cs_n <= 0;
        if(count1 == 0) test_pattren <= 1;
        else if(count1 == 1) test_pattren <= 3;
        else if(count1 == 3) test_pattren <= 1;
        else if(count1 == 4) test_pattren <= 0;
        else test_pattren <= 0;

    end
    else if(next_state == S4)begin
        we_n <= 1;
        cs_n <= 0;
        test_pattren <= 2'bx;

    end
    else if(next_state == S5)begin
        we_n <= 1;
    end
    else if(next_state == DONE)begin
        cs_n <= 1;
    end
    else cs_n <= 0;
end

//============================
//            c
//============================
always@(posedge clk ) begin
    if (!rst_n) begin
        c <= 0 ;
    end

    else if(C_STATE == START)begin
        c <= 0 ;
    end

    else if(next_state == S2)begin
        if(count1 == 0 && C_STATE == S4) begin
            if(a == 15) c <= 1;
            else c <= 0;
        end
        else if(count1 == 1)  c <= 1;
        else if (count1 == 3) begin
            if(next_state == 4) c <= 1;
            else if (a == 15) c <=1;
            else c <= 3;
        end
        else if (count1 == 4) c <= 1;
        else c <= 0;
    end

    else if(next_state == S4)begin
        c <= test_pattren;
    end

    else if(next_state == S5)begin
        if(count1 ==2 || count1 ==1) c <= 3;
        else c <= 0;
    end

    else if(next_state == DONE)begin
        c <= 0;
    end

    else c <= c;
end

//============================
//          count1
//============================
always@(posedge clk ) begin
    if (!rst_n) begin
        count1 <= 0 ;
    end
    else if(C_STATE == 4 && a == 15)begin
        count1 <= count1 + 1 ;
    end
    else if(C_STATE == 5 && a == 15)count1 <= count1 + 1 ;
    else count1 <= count1;
end

//============================
//            d
//============================
always@(posedge clk ) begin
    if (!rst_n) begin
        d <= 0 ;
    end
    else if(C_STATE == START)begin
        d <= c ;
    end
    else if(next_state == S2)begin
        d <= c;
        end
    else if(next_state == S3)begin
        d <= c;
        end
    else if(next_state == S4)begin
        d <= c;
        end
    else if(next_state == S5)begin
        d <= c;
    end
    else if(next_state == DONE)begin
        d <= c;
    end
    else d <= c;
end
```

```
//============================
//       pass_or_fail
//============================
always@(posedge clk ) begin
    if (!rst_n) begin
        pass_or_fail <= 1 ;
    end
    else if(count < 16)begin
        if(C_STATE == S2)begin
            if(a > 0)begin
                if(d == memory) pass_or_fail <= 0 ;
                else pass_or_fail <= 3;
            end
            else if ( count1 == 1) begin
                if(d == memory) pass_or_fail <= 0 ;
                else pass_or_fail <= 3;
            end
            else if ( count1 == 3) begin
                if(d == memory) pass_or_fail <= 0 ;
                else pass_or_fail <= 3;
            end
            else pass_or_fail <= 1 ;
        end

        else if(C_STATE == S3) begin
            if(d == memory) pass_or_fail <= 0 ;
            else pass_or_fail <= 3;
        end

        else if(C_STATE == S5) begin
            if(d == memory) pass_or_fail <= 0 ;
            else pass_or_fail <= 3;
        end

        else if(C_STATE == DONE) begin
            if(d == memory) pass_or_fail <= 0 ;
            else pass_or_fail <= 3;
        end

        else pass_or_fail <= 1 ;
    end
    else pass_or_fail <= 1 ;
end

//============================
//          count
//============================
always@(posedge clk )
begin
    if (!rst_n) begin
        count <= 1'bx;
    end
    else if(C_STATE == S2)begin
        if(a == 0) count <= 0  ;
        else count <= count + 1 ;
    end
    else if(C_STATE == S5) begin
        if(a== 0) count <= 0  ;
        else count <= count + 1 ;
    end
    else if(C_STATE == DONE) count <= 1'bx;
    else count <= count;
end

```

```
//============================
//     address_compare
//============================
always@(posedge clk )
begin
    if (!rst_n) begin
        address_compare <= 4'bx;
    end
    else if(C_STATE == S3)begin
        address_compare <= a;
    end
    else if(C_STATE == S5)begin
        address_compare <= count;
    end
    else if(C_STATE == S2)begin
        if(count1 == 3) address_compare <= count;
    end
    else if(C_STATE == DONE)begin
        if(count1 == 6) address_compare <= count;
    end
    else address_compare <= address_compare;
end

always@(posedge clk ) begin
    if (!rst_n) begin
        test_down <= 0;
    end
    else if(C_STATE == DONE)begin
        if(address_compare == 14) test_down <= 0;
        else test_down <= 1;
    end
    else test_down <= 0;
end

endmodule
```

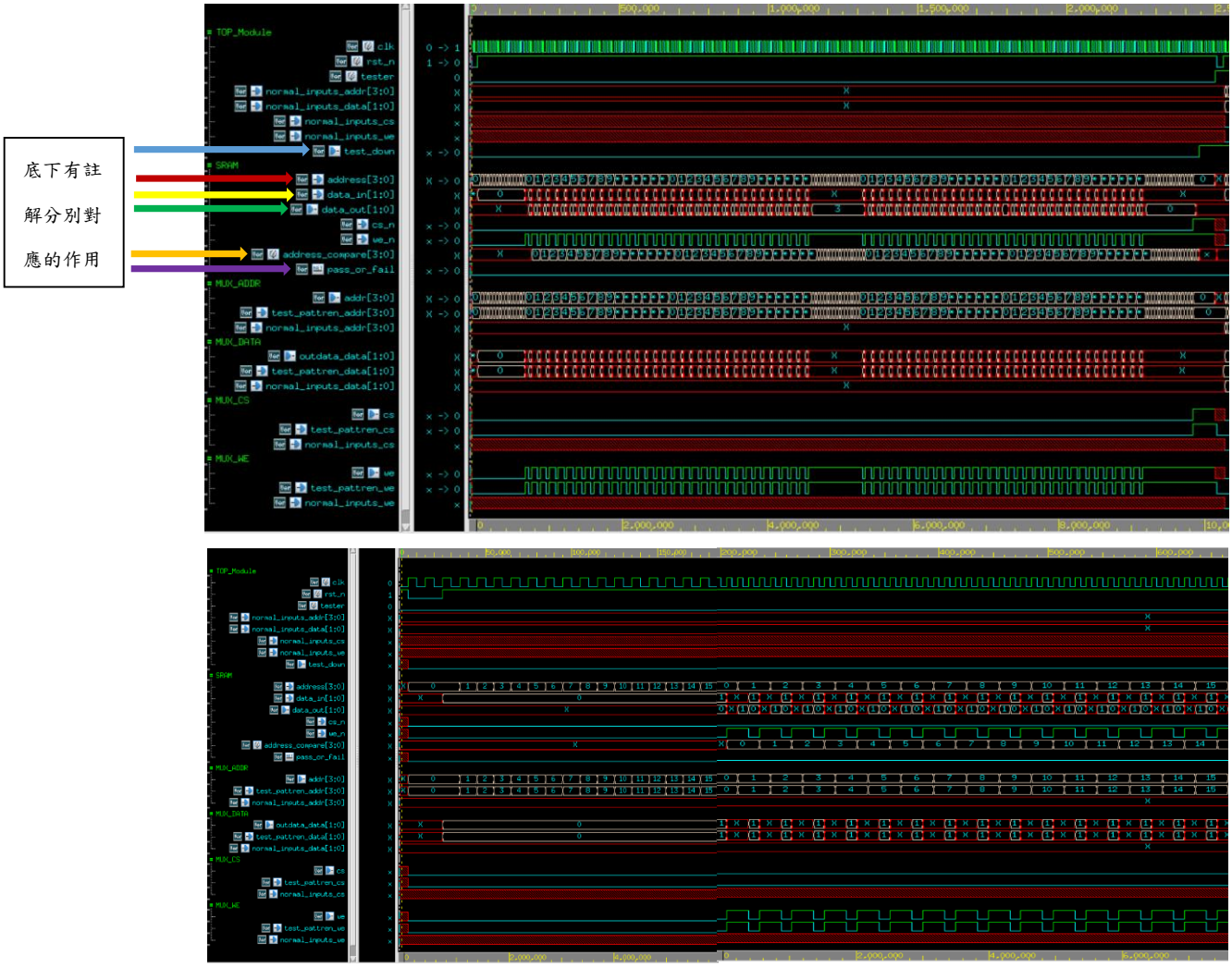
RTL SRAM module 沿用 第(a)小題的 module

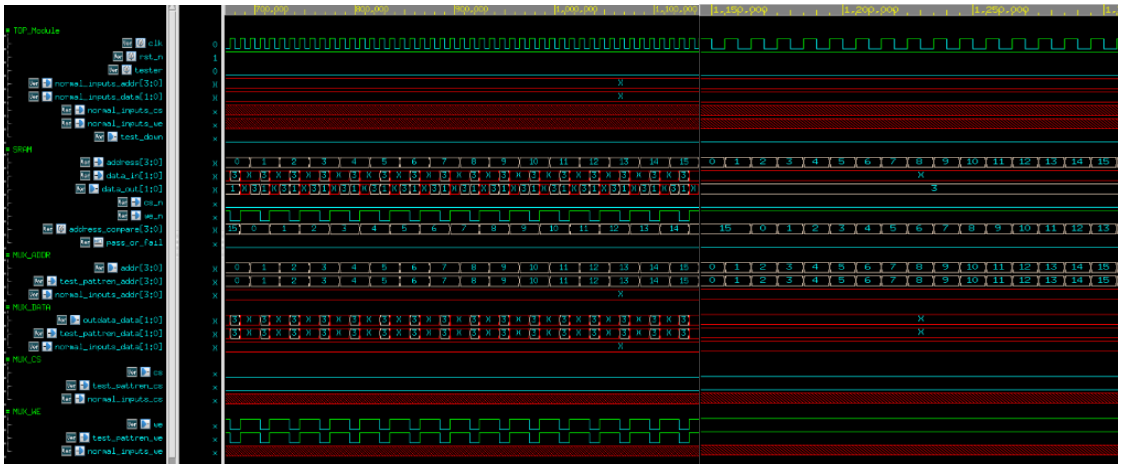
March simulation waveforms

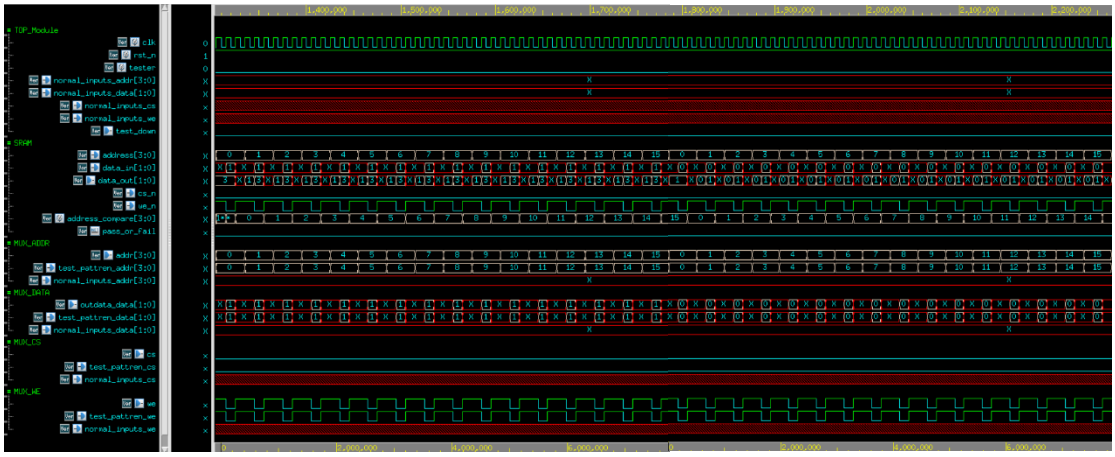


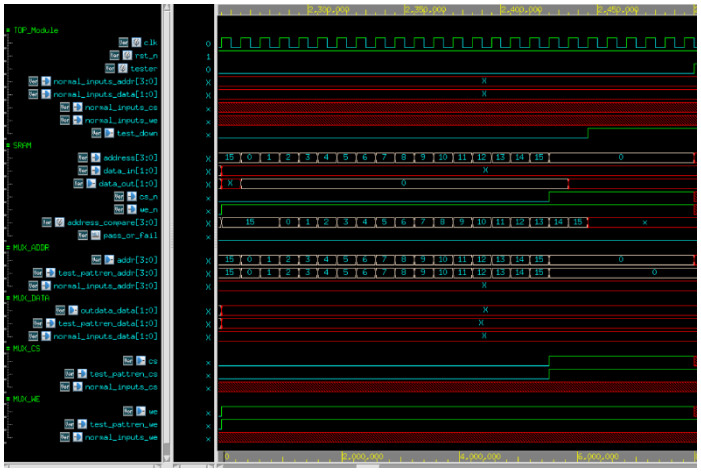
Tester 為 0 時電路執行 MARCH，為 1 時執行 SRAM。

March 的部分









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/count1_reg_3_/CK (DFFQXL)	0.00	0.00 r
BIST_MARCH/count1_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 (AND3X2)	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_march_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_march_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 (NOR3X1)	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

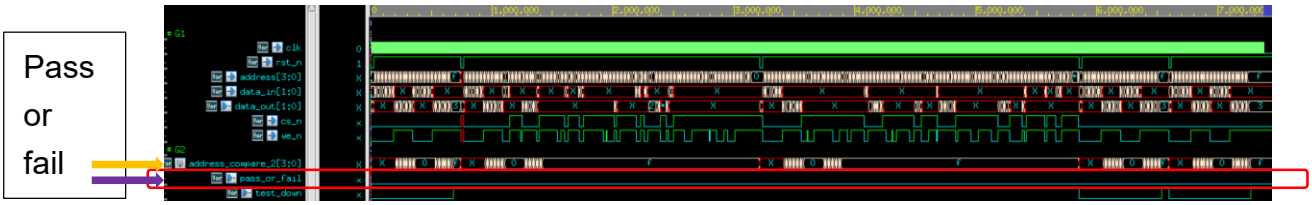
Checkerboard data arrival time = 1.50

(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 you BIST circuit report a test result of “pass”? If not, discuss why.

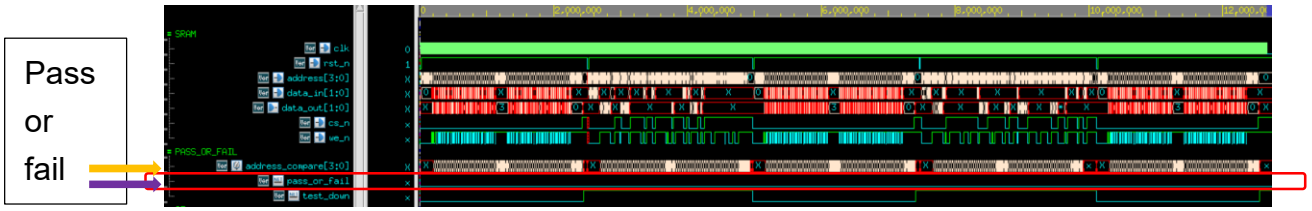
以下 waveform 的 pass or fail = 0 皆為 pass!!

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

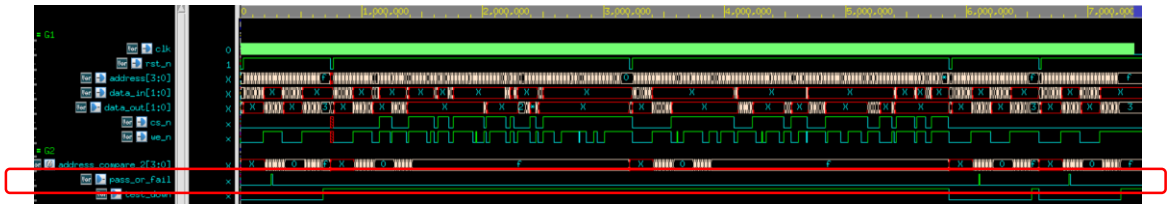
```
43      else if(we_n == 1 && cs_n == 0)begin
44          // Fault_free
45          data_out[0] <= space0[address[3:2]][address[1:0]];
46          data_out[1] <= space1[address[3:2]][address[1:0]];
47
48          //Stuck at fault
49          /*if(address[3:2] == 1 && address[1:0] == 1)begin
50              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。