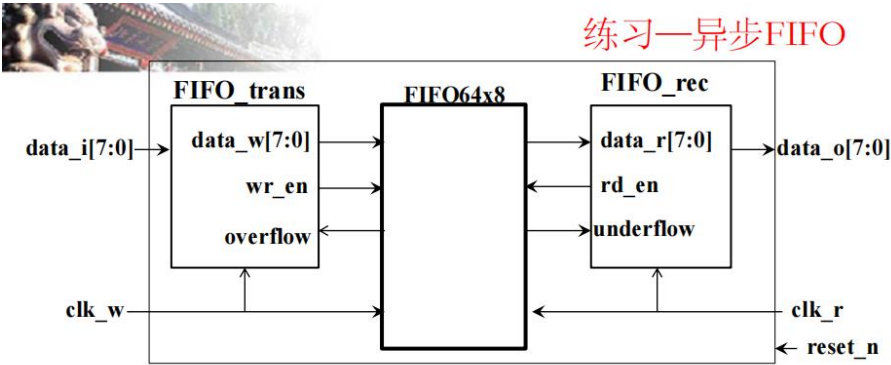


一. 异步 FIFO



练习一异步FIFO

电路receiver如图所示。其中clk\_w和clk\_r是异步时钟，其频率基本一致，但存在极小的误差。

transmitter: 接收data\_i的数据并写入FIFO64x8。若发现overflow，则找出data\_i序列中的null数据（0x00）。若发现null则丢弃，不写入FIFO，直到overflow消失。

receiver: 从FIFO中读出数据并从data\_o送出。若发现underflow，则停止读操作，向data\_o送出null数据，直到underflow消失。

1. 模块功能代码:

```
1 //异步FIFO，未考虑时钟同步，用NULL处理
2 module FIFO_asyn_64x8_NULL #(
3     //param
4     parameter DEPTH = 6,           //数据深度 (2^6)
5     parameter WIDTH = 8,           //每个数据的宽度 (8位)
6     parameter MAX_COUNT = (1 << DEPTH), //存储器中的数据个数 (2^6)
7     parameter NULL = 8'b0
8 ) (
9     //input
10    input wr_clk,                   //写时钟
11    input rd_clk,                   //读时钟
12    input wr_rstn,                  //写复位 (低电平有效)
13    input rd_rstn,                  //读复位 (低电平有效)
14    input wr_en,                    //写使能
15    input rd_en,                    //读使能
16    input [WIDTH-1:0] wr_data,      //写入的数据
17
18    //output
19    output reg wr_overflow,          //FIFO将满, 75% (影响写入, 在写时钟端判断)
20    output reg rd_underflow,        //FIFO将空, 25% (影响读出, 在读时钟端判断)
21    output reg [WIDTH-1:0] rd_data, //读出的数据
22    //output_for_test
23    output reg [DEPTH-1:0] wr_ptr,  //写指针, 指向下一个要写的位置
24    output reg [DEPTH-1:0] rd_ptr,  //读指针, 指向下一个要读的位置
25    output reg [DEPTH-1:0] count    //计数FIFO中的数据个数
26 );
27
28 /*
29 //定义读写指针
30 reg [DEPTH-1:0] wr_ptr; //写指针, 指向下一个要写的位置
31 reg [DEPTH-1:0] rd_ptr; //读指针, 指向下一个要读的位置
32 */
33
34 //定义FIFO存储器
35 reg [WIDTH-1:0] fifo_mem[MAX_COUNT-1:0]; //共有64个8bits数据的存储器
```

```

37 //写
38 always @(posedge wr_clk or negedge wr_rstn) //写控制端异步复位
39 begin
40     if(!wr_rstn) wr_ptr <= 0;
41     else if(wr_en && !wr_overflow) //写使能有效且FIFO未满时可写
42     begin
43         fifo_mem[wr_ptr] <= wr_data;
44         wr_ptr <= wr_ptr + 1'b1;
45     end
46     else if(wr_en && wr_overflow) //当overflow时找出输入数据中的NULL进行丢弃，直到overflow消失
47     begin
48         if(wr_data == NULL)
49         begin
50             wr_ptr <= wr_ptr;
51         end
52         else
53         begin
54             fifo_mem[wr_ptr] <= wr_data;
55             wr_ptr <= wr_ptr + 1'b1;
56         end
57     end
58     else wr_ptr <= wr_ptr;
59 end
60
61 //读
62 always @(posedge rd_clk or negedge rd_rstn) //读控制端异步复位
63 begin
64     if(!rd_rstn) rd_ptr <= 0;
65     else if(rd_en && !rd_underflow) //读使能有效且FIFO未空时可读
66     begin
67         rd_data <= fifo_mem[rd_ptr];
68         rd_ptr <= rd_ptr + 1'b1;
69     end
70     else if(rd_en && rd_underflow) //当underflow时停止读数据，直接送出NULL，直到underflow消失
71     begin
72         rd_data <= NULL;
73         rd_ptr <= rd_ptr;
74     end
75     else rd_ptr <= rd_ptr;
76 end
77
78 //数据个数计数
79 always @(*)
80 begin
81     if(!wr_rstn | !rd_rstn) count = 0;
82     else
83         count = wr_ptr - rd_ptr;
84 end
85
86 //更新标志位overflow和underflow
87 always @(*)
88 begin
89     wr_overflow = count[(DEPTH-1)-:2] == 2'b11; //count高两位是11时，说明数据个数>=48 (75%)
90     rd_underflow = count[(DEPTH-1)-:2] == 2'b00; //count高两位是11时，说明数据个数<16 (25%)
91 end
92
93 endmodule

```

2. 模块测试代码：先从 0 写 FIFO，直到写指针递增为 63，再将 FIFO 读空到 underflow 状态；设读时钟的频率高于写时钟；将读/写指针以及计数器输出，方便调试。

```

1 //FIFO_asyn_64x8_NULL testbench
2 `timescale 1ns/1ns
3 module FIFO_asyn_64x8_NULL_tb;
4     reg wr_clk;
5     reg rd_clk;
6     reg wr_rstn;
7     reg rd_rstn;
8     reg wr_en;
9     reg rd_en;
10    reg [7:0] wr_data;
11
12    wire wr_overflow;
13    wire rd_underflow;
14    wire [7:0] rd_data;
15
16    wire [5:0] wr_ptr;
17    wire [5:0] rd_ptr;
18    wire [5:0] count;
19
20
21 FIFO_asyn_64x8_NULL FIFO_asyn_64x8_NULL_test(
22 //input
23     .wr_clk(wr_clk),           //写时钟
24     .rd_clk(rd_clk),           //读时钟
25     .wr_rstn(wr_rstn),         //写复位（低电平有效）
26     .rd_rstn(rd_rstn),         //读复位（低电平有效）
27     .wr_en(wr_en),             //写使能
28     .rd_en(rd_en),             //读使能
29     .wr_data(wr_data),         //写入的数据
30
31 //output
32     .wr_overflow(wr_overflow),  //FIFO将满，75%（影响写入，在写时钟端判断）
33     .rd_underflow(rd_underflow), //FIFO将空，25%（影响读出，在读时钟端判断）
34     .rd_data(rd_data),         //读出的数据
35 //output_for_test
36     .wr_ptr(wr_ptr),           //写指针，指向下一个要写的位置
37     .rd_ptr(rd_ptr),           //读指针，指向下一个要读的位置
38     .count(count)              //计数FIFO中的数据个数
39 );

```

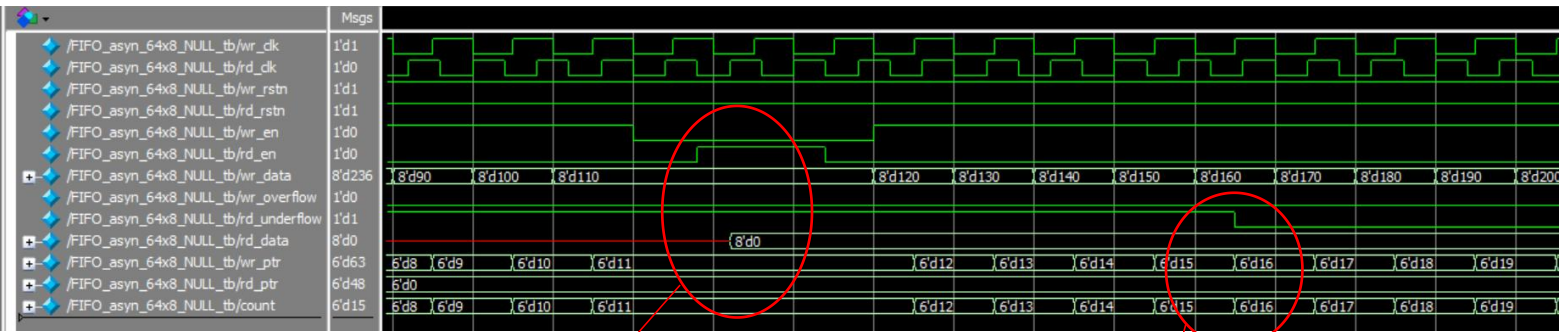


```

41 always #25 wr_clk = ~wr_clk;
42 always #20 rd_clk = ~rd_clk;
43
44 initial begin
45     wr_en = 0; rd_en = 0; wr_rstn = 1; rd_rstn = 1; wr_data = 8'b0; wr_clk = 0; rd_clk = 0;
46     #10 wr_rstn = 0; rd_rstn = 0;
47     #20 wr_rstn = 1; rd_rstn = 1;
48
49     @ (negedge wr_clk);
50     wr_data = 10;
51     wr_en = 1;
52
53     @ (negedge wr_clk);
54     wr_data = 20;
55     @ (negedge wr_clk);
56     wr_data = 30;
57     @ (negedge wr_clk);
58     wr_data = 40;
59     @ (negedge wr_clk);
60     wr_data = 50;
61     @ (negedge wr_clk);
62     wr_data = 60;
63     @ (negedge wr_clk);
64     wr_data = 70;
65     @ (negedge wr_clk);
66     wr_data = 80;
67     @ (negedge wr_clk);
68     wr_data = 90;
69     @ (negedge wr_clk);
70     wr_data = 100;
71     @ (negedge wr_clk);
72     wr_data = 110;
73
74     @ (negedge wr_clk);
75     wr_en = 0;
76
77     @ (negedge rd_clk);
78     rd_en = 1;
79     #80
80
81 @ (negedge rd_clk);
82 rd_en = 0;
83
84 @ (negedge wr_clk);
85 wr_data = 120;
86 wr_en = 1;
87 @ (negedge wr_clk);
88 wr_data = 130;
89 @ (negedge wr_clk);
90 wr_data = 140;
91 @ (negedge wr_clk);
92 wr_data = 150;
93 @ (negedge wr_clk);
94 wr_data = 160;
95 @ (negedge wr_clk);
96 wr_data = 170;
97 @ (negedge wr_clk);
98 wr_data = 180;
99 @ (negedge wr_clk);
100 wr_data = 190;
101 @ (negedge wr_clk);
102 wr_data = 200;
103 @ (negedge wr_clk);
104 wr_data = 201;
105 @ (negedge wr_clk);
106 wr_data = 202;
107 @ (negedge wr_clk);
108 wr_data = 203;
109
110 @ (negedge rd_clk);
111 rd_en = 1;
112 #40
113 @ (negedge rd_clk);
114 rd_en = 0;
115
116 @ (negedge wr_clk);
117 wr_data = 204;
118
119 @ (negedge wr_clk);
120 wr_data = 205;
121 @ (negedge wr_clk);
122 wr_data = 206;
123 @ (negedge wr_clk);
124 wr_data = 207;
125 @ (negedge wr_clk);
126 wr_data = 208;
127 @ (negedge wr_clk);
128 wr_data = 209;
129 @ (negedge wr_clk);
130 wr_data = 210;
131 @ (negedge wr_clk);
132 wr_data = 211;
133 @ (negedge wr_clk);
134 wr_data = 212;
135 @ (negedge wr_clk);
136 wr_data = 213;
137 @ (negedge wr_clk);
138 wr_data = 214;
139 @ (negedge wr_clk);
140 wr_data = 215;
141
142 @ (negedge wr_clk);
143 wr_data = 0;
144 @ (negedge wr_clk);
145 wr_data = 0;
146
147 @ (negedge wr_clk);
148 wr_data = 216;
149 @ (negedge wr_clk);
150 wr_data = 217;
151 @ (negedge wr_clk);
152 wr_data = 218;
153 @ (negedge wr_clk);
154 wr_data = 219;
155 @ (negedge wr_clk);
156 wr_data = 220;
157
158 @ (negedge wr_clk);
159 wr_data = 220;
160 @ (negedge wr_clk);
161 wr_data = 221;
162 @ (negedge wr_clk);
163 wr_data = 222;
164 @ (negedge wr_clk);
165 wr_data = 223;
166 @ (negedge wr_clk);
167 wr_data = 224;
168 @ (negedge wr_clk);
169 wr_data = 225;
170 @ (negedge wr_clk);
171 wr_data = 226;
172
173 @ (negedge wr_clk);
174 wr_data = 0;
175 @ (negedge wr_clk);
176 wr_data = 0;
177
178 @ (negedge wr_clk);
179 wr_data = 227;
180 @ (negedge wr_clk);
181 wr_data = 228;
182 @ (negedge wr_clk);
183 wr_data = 229;
184 @ (negedge wr_clk);
185 wr_data = 230;
186 @ (negedge wr_clk);
187 wr_data = 231;
188 @ (negedge wr_clk);
189 wr_data = 232;
190 @ (negedge wr_clk);
191 wr_data = 233;
192 @ (negedge wr_clk);
193 wr_data = 234;
194 @ (negedge wr_clk);
195 wr_data = 235;
196
197 @ (negedge wr_clk);
198 wr_data = 0;
199
200 @ (negedge rd_clk);
201 rd_en = 1;
202 #600;
203
204 @ (negedge rd_clk);
205 rd_en = 0;
206
207 @ (negedge wr_clk);
208 wr_en = 0;
209
210 @ (negedge wr_clk);
211 wr_data = 236;
212
213 @ (negedge rd_clk);
214 rd_en = 1;
215 #1920
216 @ (negedge rd_clk);
217 rd_en = 0;
218
219 #50 $finish;
220 end
endmodule

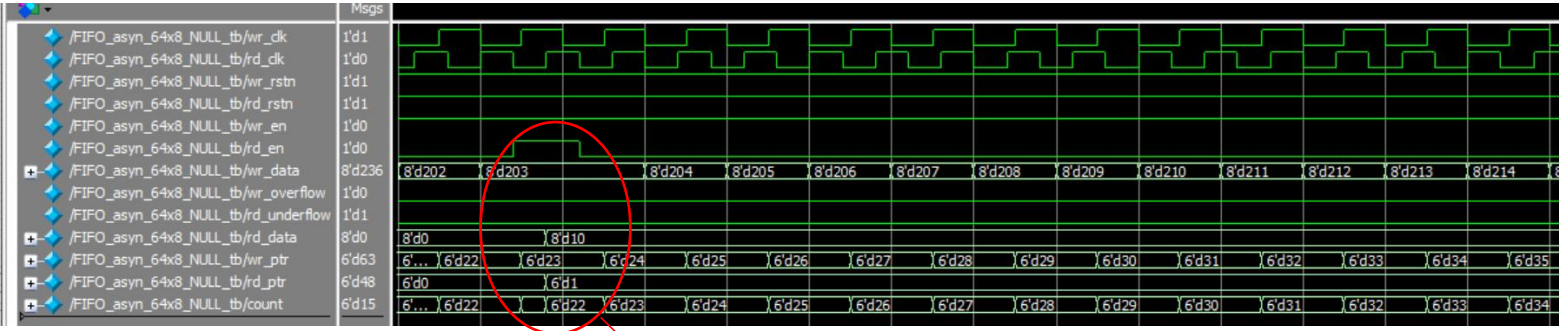
```

### 3. ModelSim 仿真结果及分析:

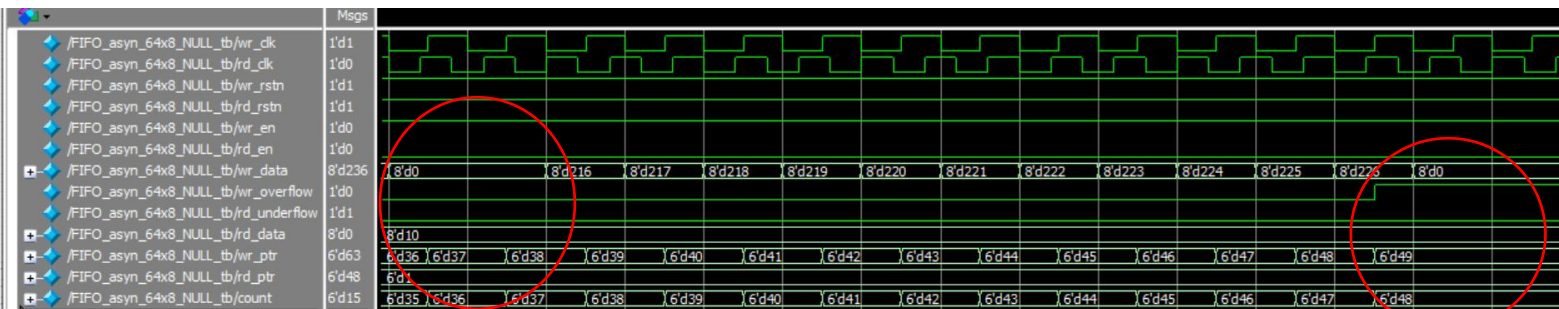


在 FIFO 中数据量为 11 时, underflow 为 1, 此时读数据读到的不是 FIFO 中的数据, 而是 NULL (0), 此时读指针也不+1

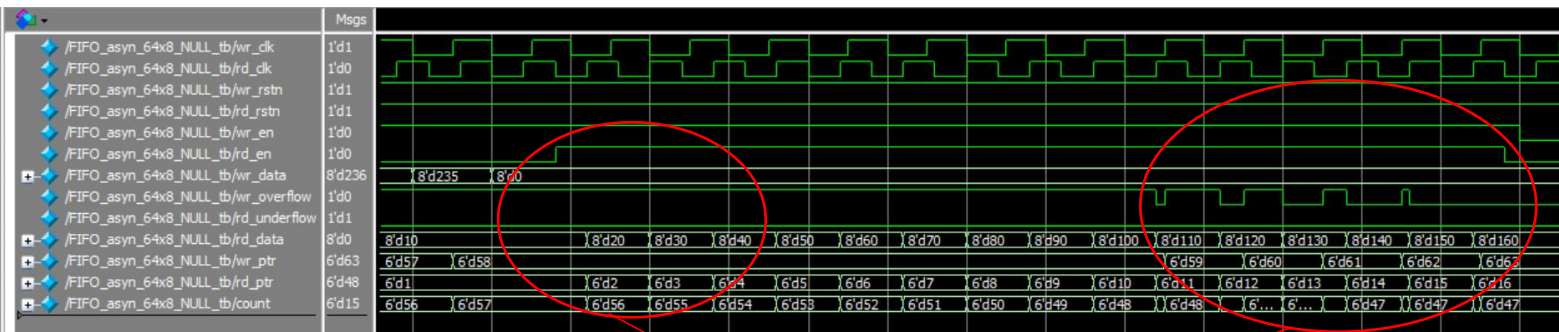
在 FIFO 中数据量为 16 时, underflow 变成 0



在 FIFO 中数据量为 22 时, underflow 为 0, 此时 rd\_data 读取了 FIFO 中的第一个数据 10, 读指针也+1。之后继续往 FIFO 中写入数据

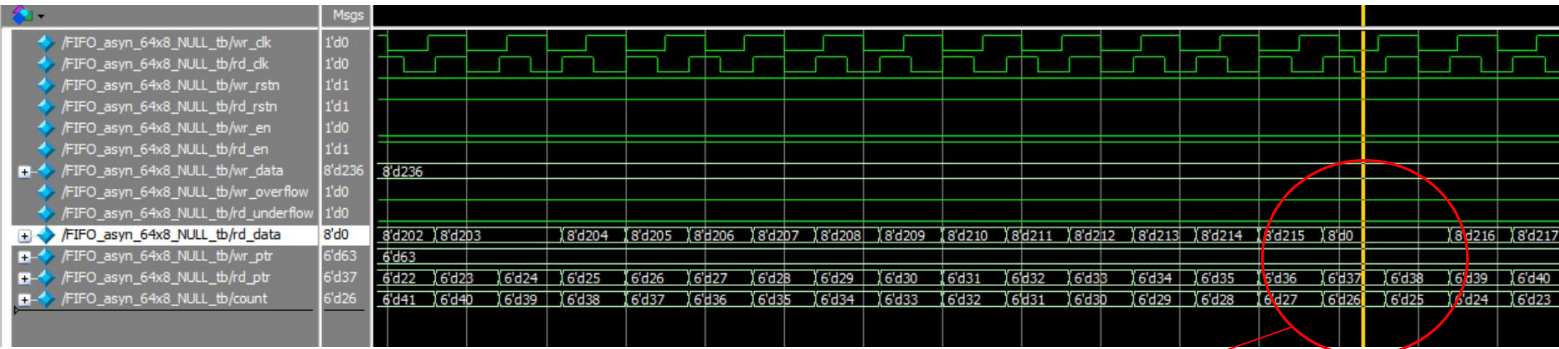


在 FIFO 中数据量为 48 时, overflow 为 1, 此时 wr\_data 中的 NULL 信号不能被写入 FIFO, 写指针不变。对比 overflow 为 0 时, wr\_data 中的 NULL 可以写入 FIFO, 写指针+1

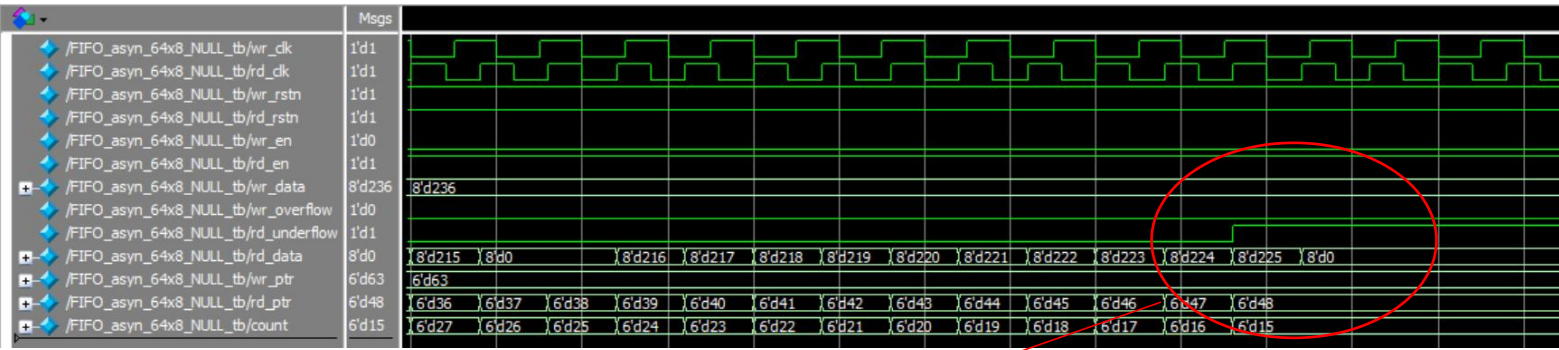


在 overflow 状态下开始连续读取数据, 不写入 (因为 wr\_data 为 NULL), FIFO 中的数据量递减 (count), 最先存入 FIFO 的数据依次读到 rd\_data。当数据量递减到 47 时, overflow 为 0, wr\_data 中的 NULL 可以写入 FIFO, 由于时钟不同步会产生毛刺, 稳定后读写同时进行, 数据量为 47 不变。





不断读出数据，不写入（写指针不变，读指针递增，计数器递减）。可以看出，读到了在 overflow 为 0 时写入 FIFO 的 NULL 数据。



当 FIFO 中的数据量为 15 时，underflow 为 1，读出的数据 rd\_data 不再是 FIFO 中的数据，而是变成 NULL，读指针不变，FIFO 中的数据量也不减少。

#### 4. 存在的问题：

标志位的产生会有不稳定状态出现，是由于时钟不同步造成的。改进时可以利用两级同步延迟的方法，进行跨时钟域同步。

二. 异步接口 SRAM 建模

试描述一个1Kx8的异步SRAM，并对其端口信号进行如右图所示的时序检查，确定信号符合时序要求。若不满足时序要求，则将输出数据置为X值。同时给出测试程序验证时序检查的正确性。

```
module SRAMSP_1Kx8 #(
    parameter width = 8,
            addr  = 10,
            depth = 1 << addr
) (
    inout wire [width-1:0] data ,
    input wire [addr-1 :0] address ,
    input wire          cs      ,
                    rd      ,
                    wr
);

reg [width-1:0] mem [depth-1:0]; // 1Kx8

assign data = (!cs && !rd) ? mem[address]
                        : 'bz;

always @(posedge wr)
    if (lcs)
        mem[address] = data;
endmodule
```

异步接口SRAM建模

The top timing diagram illustrates a read operation. It shows the address, chip select (cs), read enable (rd), and data signals. Key timing constraints are marked: address setup and hold times are less than 2 units; cs setup and hold times are less than 1 unit; rd setup and hold times are greater than 1 unit; and the data is valid for a duration greater than 1 unit. The bottom timing diagram illustrates a write operation. It shows the address, chip select (cs), write enable (wr), and data signals. Key timing constraints are marked: address setup and hold times are less than 2 units; cs setup and hold times are less than 1 unit; wr setup and hold times are greater than 1 unit; and the data is valid for a duration greater than 4 units.

1. 功能模块代码：对多个信号之间分别进行时序分析，利用 notifier。

```
1 //异步接口SRAM建模，时序检查
2 `timescale 1ns/1ns
3 module SRAMSP_1Kx8 #(
4     parameter width = 8,
5     parameter addr = 10,
6     parameter depth = (1 << addr)
7 ) (
8     inout wire [width-1:0] data,
9     input wire [addr-1:0] address,
10    input wire cs,
11    input wire rd,
12    input wire wr,
13
14    output reg re_c2a
15    //output reg re_w2c,
16    //output reg re_r2c,
17    //output reg re_w2d,
18    //output reg sk_a2c,
19    //output reg sk_c2w,
20    //output reg sk_c2r,
21    //output reg wid_data,
22    //output reg sk_w2d
23
24 );
25 reg [width-1:0] mem[depth-1:0];
26 assign data = (!cs && !rd)? mem[address] : 'bz;
27 reg [width-1:0] data_reg;
28
29 always @(negedge wr)
30     if(!cs) mem[address] <= data;
```

```

32 specify
33     $setup(posedge cs, negedge address, 2, se_c2a);
34     //$recovery(posedge wr, posedge cs, 1, re_w2c);
35     //$recovery(posedge rd, posedge cs, 1, re_r2c);
36     //$recovery(posedge wr, negedge data, 1, re_w2d);
37     //$skew(posedge address, negedge cs, 2, sk_a2c);
38     //$skew(negedge cs, negedge wr, 1, sk_c2w);
39     //$skew(negedge cs, negedge rd, 1, sk_c2r);
40     //$width(posedge data, 4, 0, wid_data);
41     //$skew(negedge rd, posedge data, 5, sk_w2d);
42 endspecify
43
44 always @(se_c2a)
45 data_reg = 8'bx;
46
47 assign data = data_reg;
48
49 endmodule

```

## 2. 测试模块代码:

```

1 `timescale 1ns/1ns
2 `include "SRAMSP_1Kx8.v"
3 module SRAMSP_1Kx8_tb;
4     wire [7:0] data;
5     reg [9:0] address;
6     reg cs;
7     reg rd;
8     reg wr;
9
10    wire re_c2a;
11    //$wire re_w2c;
12    //$wire re_w2d;
13    //$wire sk_a2c;
14    //$wire sk_c2w;
15    //$wire wid_data;
16    //$wire sk_w2d;
17    //$wire re_r2c;
18    //$wire sk_c2r;
19
20    wire clk;
21    reg en;
22    reg data0;
23
24    SRAMSP_1Kx8 SRAMSP_1Kx8_test (
25        .data(data),
26        .address(address),
27        .cs(cs),
28        .rd(rd),
29        .wr(wr),
30        .re_c2a(re_c2a)

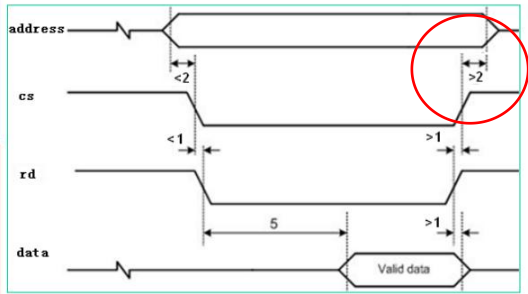
```



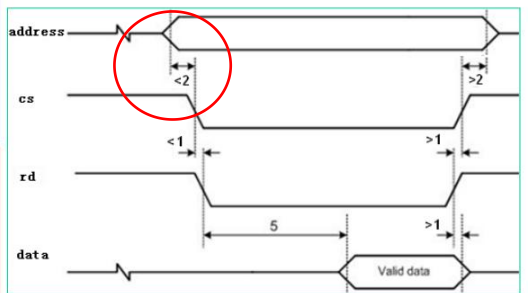
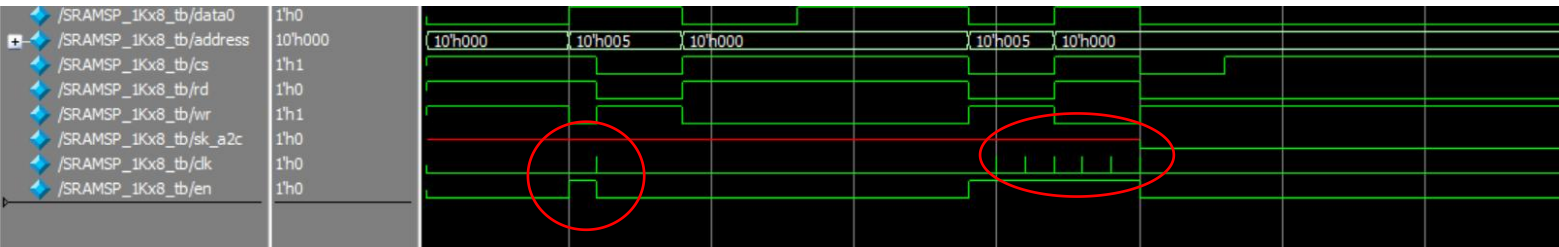
```
31 //.re_w2c(re_w2c),
32 //.re_w2d(re_w2d),
33 //.sk_a2c(sk_a2c),
34 //.sk_c2w(sk_c2w),
35 //.wid_data(wid_data),
36 //.sk_w2d(sk_w2d),
37 //.re_r2c(re_r2c),
38 //.sk_c2r(sk_c2r)
39 );
40
41 assign data = !wr? data0 : 4'bz;
42
43 and #(0.8, 0.2) (clk, en, !clk);
44
45 initial begin
46     data0 = 0; en = 0; address = 0; cs = 1; rd = 1; wr = 1;
47     #5 data0 = 1; en = 1; address = 5; cs = 1; rd = 1; wr = 0;
48     #1 data0 = 1; en = 0; address = 5; cs = 0; rd = 0; wr = 1;
49     #3 data0 = 0; en = 1; address = 0; cs = 1; rd = 1; wr = 0;
50     #4 data0 = 1; en = 0; address = 0; cs = 1; rd = 1; wr = 0;
51     #6 data0 = 0; en = 1; address = 5; cs = 0; rd = 0; wr = 1;
52     #2 data0 = 1; en = 0; address = 0; cs = 1; rd = 1; wr = 0;
53     #3 data0 = 0; en = 1; address = 0; cs = 1; rd = 0; wr = 1;
54     #20 $finish;
55 end
56
57 endmodule
```

3. ModelSim 仿真结果及分析:

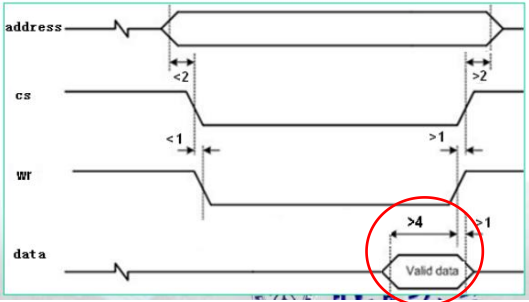
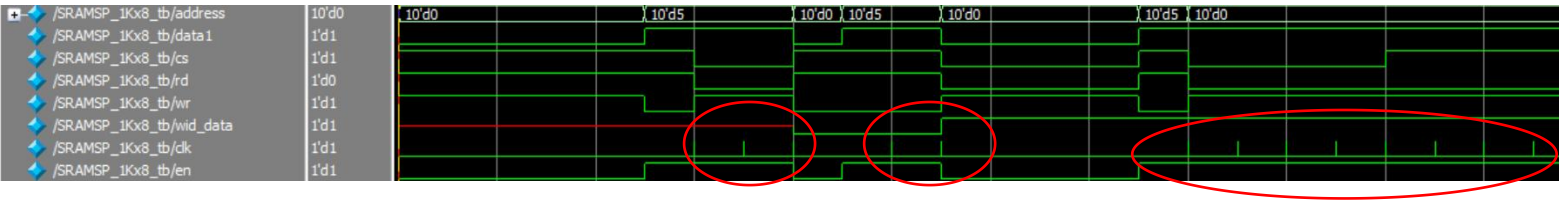
对 setup、skew、width 分别选取一例进行分析。



时差小于等于 2 时, notifier (re\_c2a) 进行翻转, 大于 2 时不翻转。



时差大于 2 时, notifier (sk\_a2c) 进行翻转, 小于 2 时不翻转。



data1 的宽度等于 2 或 3 时, 均小于 4, notifier(wid\_data) 进行翻转: data1 宽度大于 4 时不翻转。