

《计算机组成原理》 实验报告



实验题目：寄存器堆与队列

学生姓名：王志强

学生学号：PB18051049

完成日期：2020.05.04

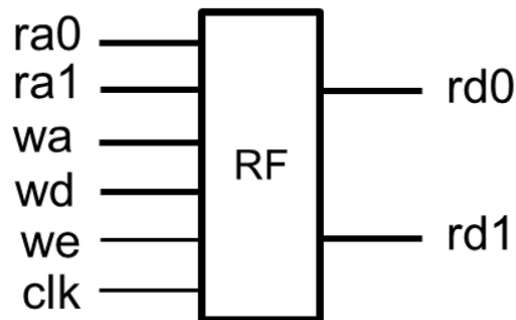
一、实验目标

- 掌握寄存器堆（Register File）和存储器（Memory）的功能、时序及其应用；
- 熟练掌握数据通路和控制器的设计和描述方法。

二、实验内容

1. 寄存器堆

设计参数化的寄存器堆，其逻辑符号如下图所示。该寄存器堆含有32个寄存器（r0 ~ r31，其中r0的内容恒定为零），寄存器的位宽由参数WIDTH指定，具有2个异步读端口和1个同步写端口。



- 根据端口和功能要求，Verilog代码实现如下：

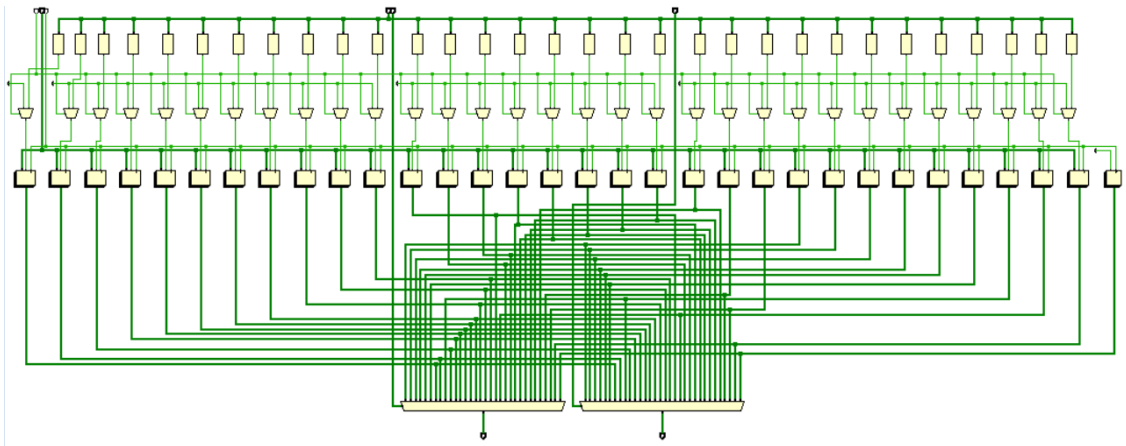
```
1  module register_file    //32xWIDTH寄存器堆
2      #(parameter WIDTH = 32) //数据宽度
3      (
4          input clk,        //时钟，上升沿有效
5          input [4:0] ra0,   //读端口0地址
6          output [WIDTH-1:0] rd0, //读端口0数据
7          input [4:0] ra1,   //读端口1地址
8          output [WIDTH-1:0] rd1, //读端口1数据
9          input [4:0] wa,    //写端口地址
10         input we,         //写使能，高电平有效
11         input [WIDTH-1:0] wd //写端口数据
12     );
13
14     parameter NUM = 32;    //寄存器数量
15     reg [WIDTH-1:0] REG_FILE [0:NUM-1]; //寄存器堆
16     integer i;
17
18     initial //仿真初始化
19     for(i = 0; i < NUM; i = i + 1)
20         REG_FILE[i] = 0;
21     //异步读端口
22     assign rd0 = REG_FILE[ra0];
23     assign rd1 = REG_FILE[ra1];
24     //写入端口
25     always @(posedge clk)
26     begin
27         if(we)
28             REG_FILE[wa] <= wd;
29         REG_FILE[0] <= 0; //R0恒为0
```

```

30     end
31 endmodule

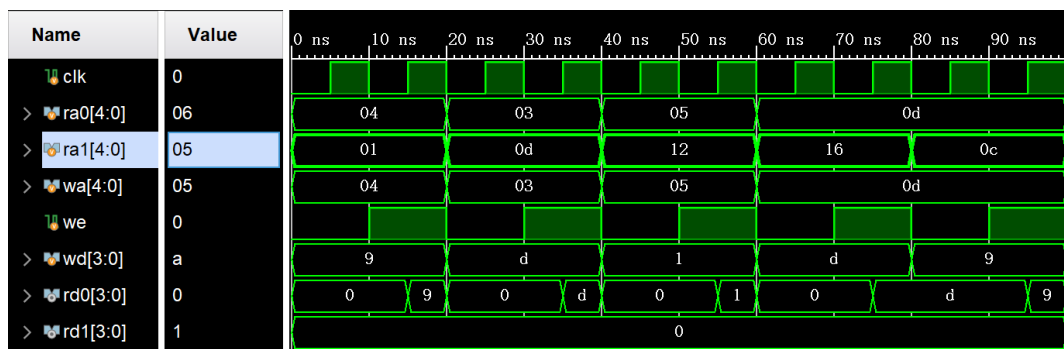
```

- RTL ANALYSIS-Schematic:

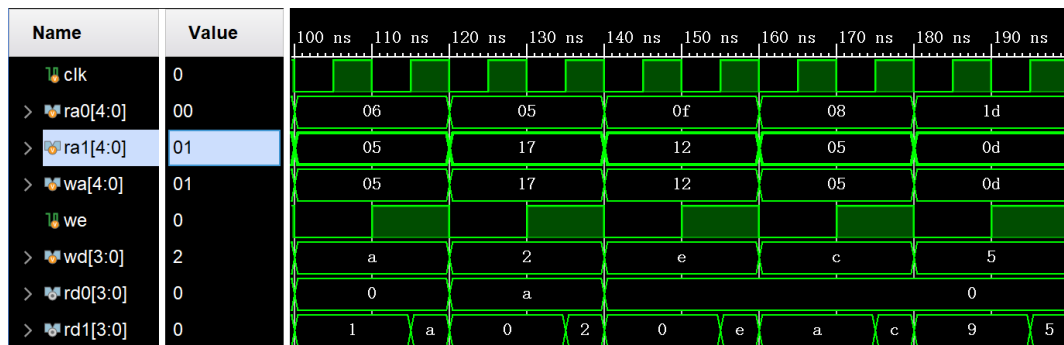


- Vivado仿真如下:

- wa=ra0:



- wa=ra1:



- test_bench(tb_register_file.v):

```

1  module tb_register_file;
2
3  // register_file Parameters
4  parameter PERIOD = 10;
5  parameter WIDTH = 4;
6
7  // register_file Inputs
8  reg    clk                = 0 ;
9  reg [4:0] ra0             = 0 ;
10 reg [4:0] ra1             = 0 ;
11 reg [4:0] wa              = 0 ;
12 reg    we                = 0 ;
13 reg [WIDTH-1:0] wd        = 0 ;
14

```

```

15 // register_file Outputs
16 wire [WIDTH-1:0] rd0          ;
17 wire [WIDTH-1:0] rd1          ;
18
19
20 initial
21 begin
22     forever #(PERIOD/2) clk=~clk;
23 end
24
25 initial
26 begin
27     forever #PERIOD we=~we; //write enable
28 end
29
30 register_file #(4) u_register_file (
31     .clk          ( clk          ),
32     .ra0          ( ra0 [4:0]    ),
33     .ra1          ( ra1 [4:0]    ),
34     .wa          ( wa  [4:0]    ),
35     .we          ( we            ),
36     .wd          ( wd  [WIDTH-1:0] ),
37
38     .rd0          ( rd0 [WIDTH-1:0] ),
39     .rd1          ( rd1 [WIDTH-1:0] )
40 );
41 //10组测试
42 initial
43 begin
44     repeat(5) begin
45         ra0 = $random%32;
46         ra1 = $random%32;
47         wa = ra0;
48         wd = $random%16;
49         #(PERIOD*2);
50     end
51
52     repeat(5)
53     begin
54         ra0 = $random%32;
55         ra1 = $random%32;
56         wa = ra1;
57         wd = $random%16;
58         #(PERIOD*2);
59     end
60     $finish;
61 end
62 endmodule

```

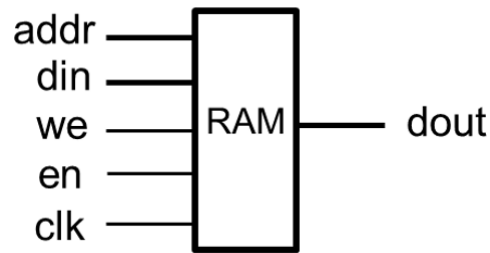
- **FPGA开发板测试如下：**

- 返校后进行

2、存储器：

存储器与寄存器堆的功能类似，都是用于存储信息，只是存储器的容量更大，配置方式更多，例如ROM/RAM、单端口/简单双端口/真正双端口、分布式/块式等方式。设计存储器可以通过行为方式描述，也可以通过IP例化方式实现。

例如，设计一容量为16 x 8位（即深度DEPTH：16，宽度WIDTH：8）的单端口RAM，其逻辑符号如图所示。用行为方式描述的Verilog代码如下：（请补充代码中空缺的参数）



```
1  module ram_16x8(           //16x8位单端口RAM
2      input  clk,           //时钟（上升沿有效）
3      input  en, we,        //使能，写使能
4      input  [ 3:0 ] addr,  //地址
5      input  [ 2:0 ] din,    //输入数据
6      output [ 2:0 ] dout   //输出数据
7  );
8
9      reg [ 3:0 ] addr_reg;
10     reg [ 2:0 ] mem[ 0:3 ];
11
12     //初始化RAM的内容
13     initial
14         $readmemh("初始化数据文件名", mem);
15
16     assign dout = mem[addr_reg];
17
18     always@(posedge clk) begin
19         if(en) begin
20             addr_reg <= addr;
21             if(we)
22                 mem[addr] <= din;
23         end
24     end
25 endmodule
26
```

- 分布式存储器例化

- 分布式存储器例化界面—存储器配置：

Re-customize IP

Distributed Memory Generator (8.0)

[Documentation](#)
[IP Location](#)
[Switch to Defaults](#)

☒ Show disabled ports

a[3:0]

d[7:0]

dpra[3:0]

clk

we

i_ce

qspo_ce

qdpo_ce

qdpo_clk

qspo_rst

qdpo_rst

qspo_srst

qdpo_srst

spo[7:0]

dpo[7:0]

qspo[7:0]

qdpo[7:0]

Component Name:

memory config | Port config | RST & Initialization

Options

Depth: [16 - 65536]

Data Width: [1 - 1024]

Memory Type

Memory Type

☐ ROM
☒ Single Port RAM
☐ Simple Dual Port RAM
☐ Dual Port RAM

OK Cancel

○ 分布式存储器例化界面—端口配置：

Re-customize IP

Distributed Memory Generator (8.0)

[Documentation](#)
[IP Location](#)
[Switch to Defaults](#)

☒ Show disabled ports

a[3:0]

d[7:0]

dpra[3:0]

clk

we

i_ce

qspo_ce

qdpo_ce

qdpo_clk

qspo_rst

qdpo_rst

qspo_srst

qdpo_srst

spo[7:0]

dpo[7:0]

qspo[7:0]

qdpo[7:0]

Component Name:

memory config | **Port config** | RST & Initialization

Input Options

Input Options

☒ Non Registered ☐ Registered

☐ Input Clock Enable ☐ Qualify WE with I_CE

Dual Port Address

Dual Port Address

☒ Non Registered ☐ Registered

Output Options

Output Options

☒ Non Registered ☐ Registered ☐ Both

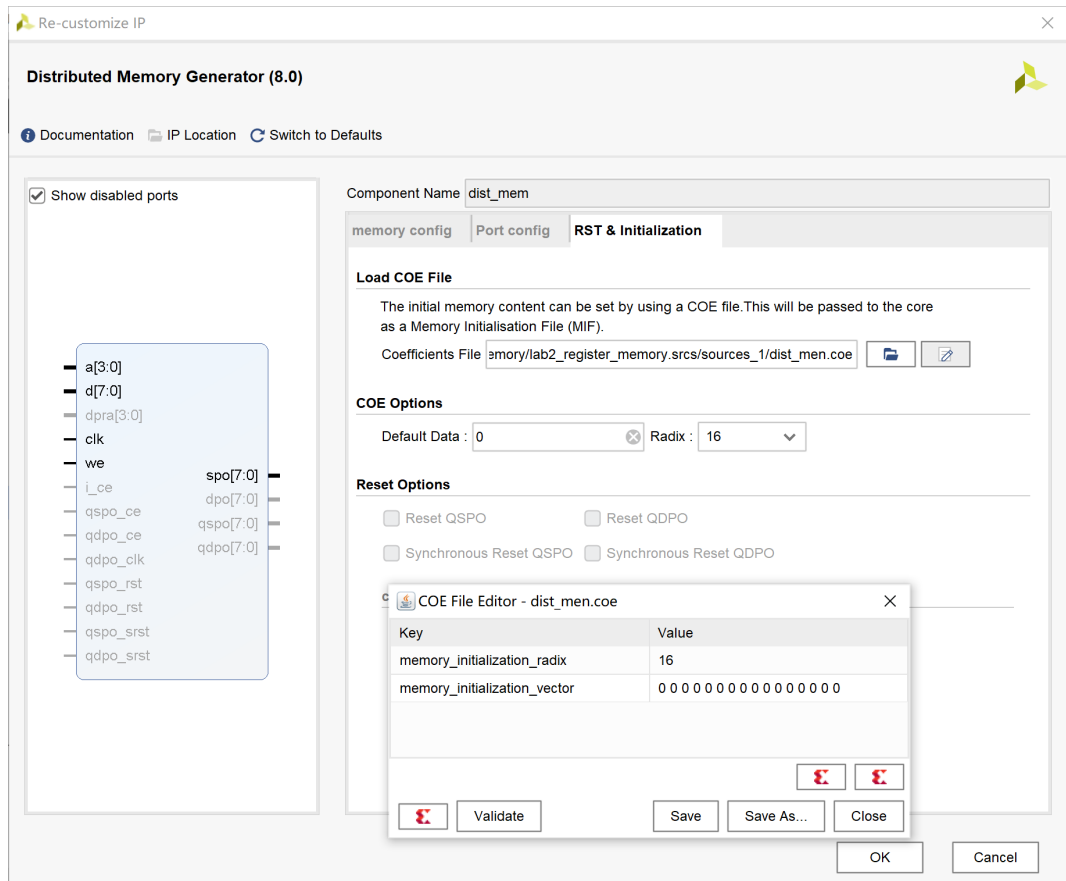
☐ Common Output CLK ☐ Single Port Output CE
☐ Common Output CE ☐ Dual Port Output CE

Pipelining Options

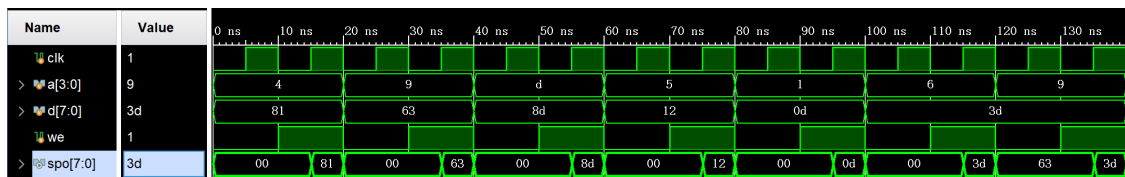
Pipeline Stages:

OK Cancel

○ 分布式存储器例化界面—复位和初始化：



• 分布式存储器仿真波形



test_bench(tb_dist_mem.v):

```

1  module tb_dist_mem;
2
3  // dist_mem Parameters
4  parameter PERIOD = 10;
5
6  // dist_mem Inputs
7  reg    clk                                = 0 ;
8  reg    [3:0] a                            = 0 ;
9  reg    [7:0] d                            = 0 ;
10 reg    we                                = 0 ;
11
12 // dist_mem Outputs
13 wire   [7:0] spo                            ;
14
15 initial
16 begin
17     forever #(PERIOD/2) clk=~clk;
18 end
19
20 initial
21 begin
22     forever #PERIOD we=~we; //write enable
23 end
24

```

```

25 dist_mem u_dist_mem (
26     .clk          ( clk          ),
27     .a             ( a      [3:0] ),
28     .d             ( d      [7:0] ),
29     .we            ( we           ),
30     .spo           ( spo   [7:0] )
31 );
32
33 initial
34 begin
35     repeat(3) begin
36         a  = $random%16;
37         d  = $random%256;
38         #(PERIOD*2);
39     end
40
41     repeat(3)
42     begin
43         a  = $random%16;
44         d  = $random%256;
45         #(PERIOD*2);
46     end
47     a = 9;
48     #(PERIOD*2);
49     $finish;
50 end
51 endmodule

```

- 块式存储器例化

- 块式存储器例化界面—基本：

Re-customize IP

Block Memory Generator (8.4)

Documentation IP Location Switch to Defaults

IP Symbol Power Estimation

☒ Show disabled ports

Component Name: blk_mem

Basic Port A Options Other Options Summary

Interface Type: Native ☐ Generate address interface with 32 bits

Memory Type: Single Port RAM ☐ Common Clock

ECC Options

ECC Type: No ECC

☐ Error Injection Pins: Single Bit Error Injection

Write Enable

☐ Byte Write Enable

Byte Size (bits): 9

Algorithm Options

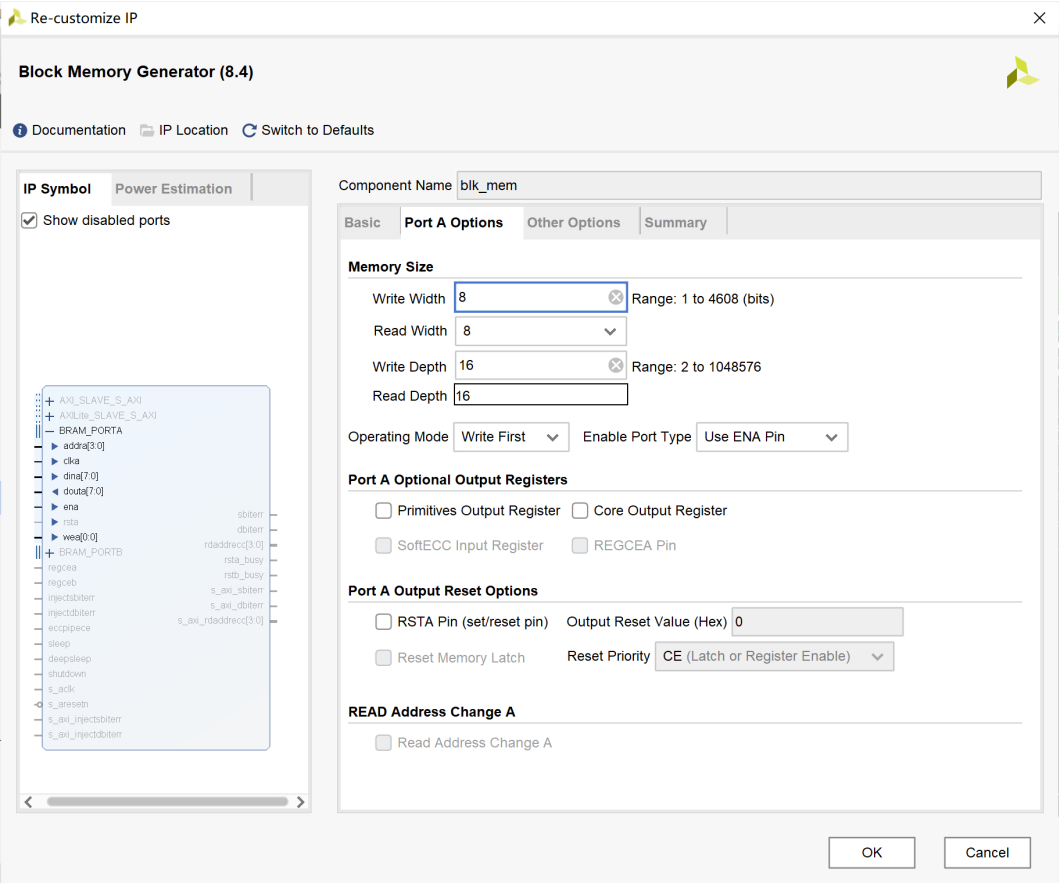
Defines the algorithm used to concatenate the block RAM primitives. Refer datasheet for more information.

Algorithm: Minimum Area

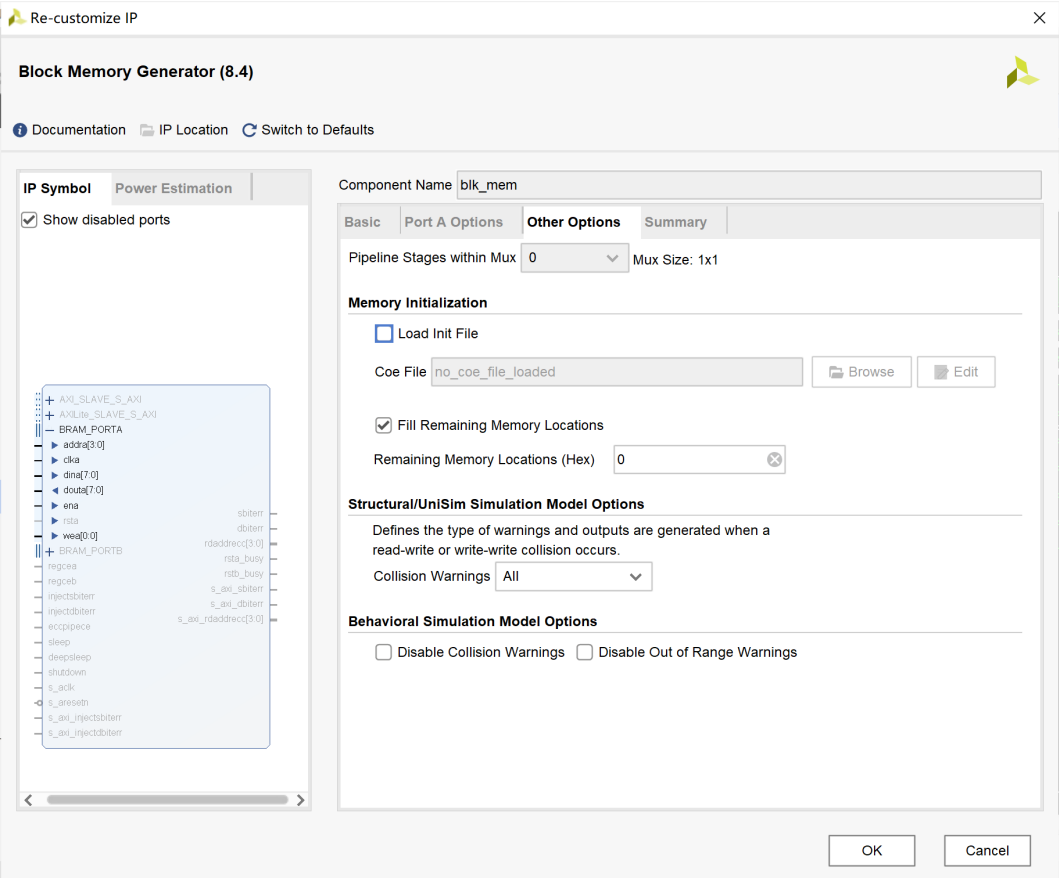
Primitive: 8kx2

OK Cancel

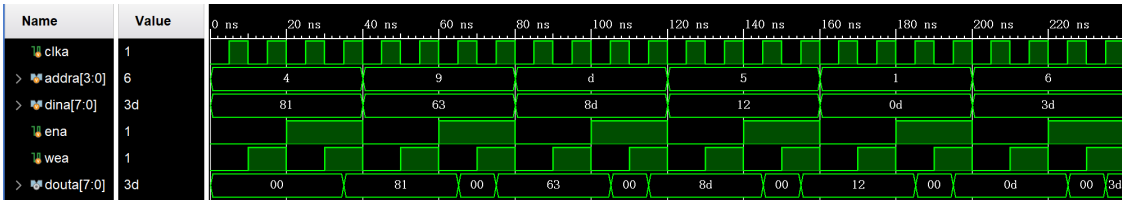
- 块式存储器例化界面—端口选项：



o 块式存储器例化界面—其他选项：



• 块式存储器仿真波形



test_bench(tb_blk_mem.v):

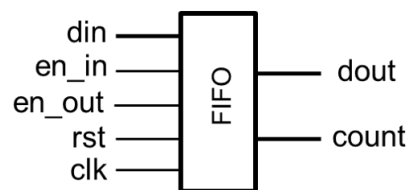
```
1  module tb_blk_mem;
2  //和分布式存储器仿真文件基本相同
3  //blk_mem Parameters
4  parameter PERIOD = 10;
5
6  // blk_mem Inputs
7  reg  clka                                = 0 ;
8  reg  [3:0]  addra                        = 0 ;
9  reg  [7:0]  dina                        = 0 ;
10 reg  ena                                = 0 ;
11 reg  wea                                = 0 ;
12 // blk_mem outputs
13 wire [7:0] douta                        ;
14
15 initial
16 begin
17     forever #(PERIOD/2)  clka=~clka;
18 end
19
20 initial
21 begin
22     forever #(PERIOD) wea=~wea; //write enable
23 end
24
25 initial
26 begin
27     forever #(PERIOD*2) ena=~ena; //top enable
28 end
29
30 blk_mem  u_blk_mem (
31     .clka                ( clka                ),
32     .addra                ( addra    [3:0]      ),
33     .dina                 ( dina    [7:0]       ),
34     .wea                  ( wea                  ),
35     .ena                  ( ena                  ),
36     .douta                ( douta    [7:0]      )
37 );
38
39 initial
40 begin
41     repeat(3) begin
42         addra = $random%16;
43         dina  = $random%256;
44         #(PERIOD*4);
45     end
46
47     repeat(3)
48     begin
49         addra = $random%16;
50         dina  = $random%256;
51         #(PERIOD*4);
52     end
53     $finish;
54 end
55 endmodule
```

- 分布式单端口RAM和块式单端口RAM对比

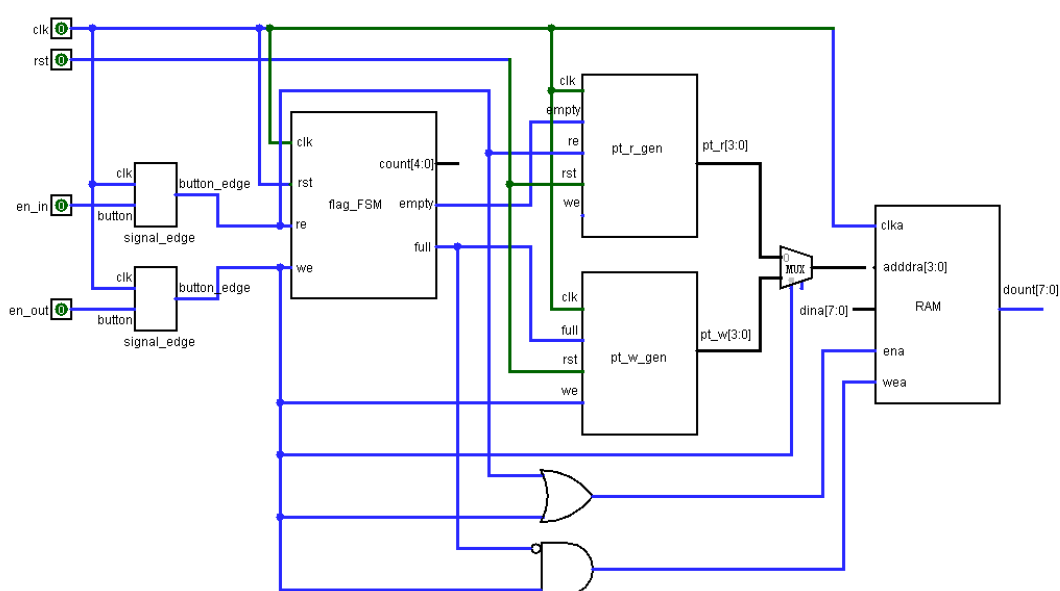
- block RAM增设总使能信号ena, dist RAM只有写使能信号we
- block RAM输出需要时钟控制, ena有效情况下在时钟上升沿输出, 为同步读取RAM
- dist RAM异步读取, 给出地址即可输出数据
- block RAM存储空间较大, 而dist RAM浪费LUT资源

3、先进先出(FIFO)队列

利用例化的存储器IP (16 x 8位块式的单端口RAM) 和适当的逻辑电路, 设计实现数据宽度为8位、最大长度为16的FIFO队列, 其逻辑符号如图-9所示。入队列使能 (en_in) 有效时, 将输入数据 (din) 加入队尾; 出队列使能 (en_out) 有效时, 将队列头数据输出 (dout)。队列数据计数 (count) 指示队列中有效数据个数。当队列满 (count = 16) 时不能执行入队操作, 队列空 (count = 0) 时不能进行出队操作。在入对使能信号的一次有效持续期间, 仅允许最多入队一个数据, 出队操作类似。

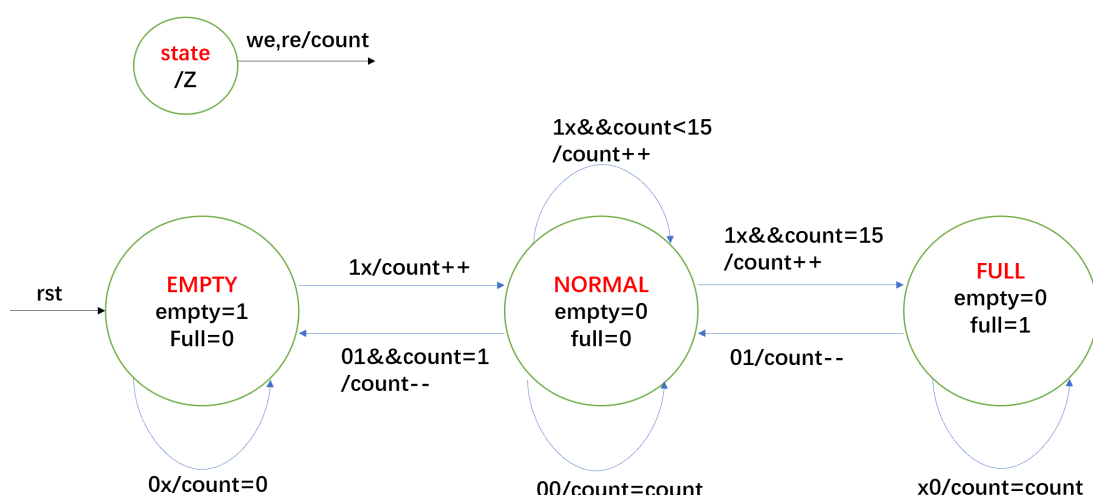


- 根据端口和功能要求, 数据通路如下图所示:

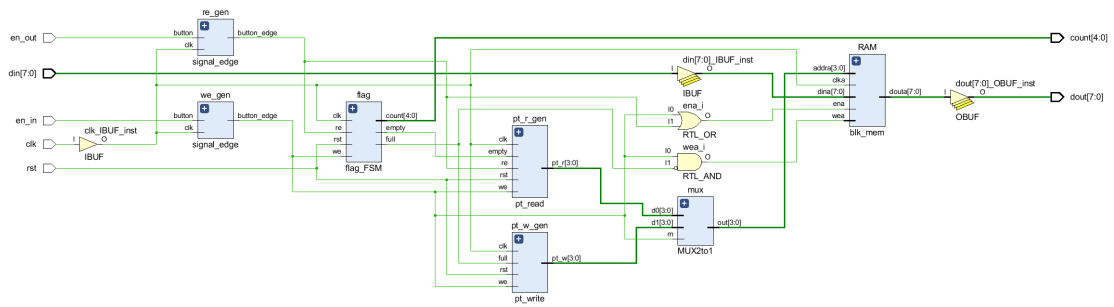


- 控制器状态转移图(write优先级高于read)

count共有16个值, 理论上可以设置16个状态, 但简单起见, 也可设置三个有效状态: EMPTY、NORMAL和FULL, 状态输出为full、empty标志以及count值。



• RTL ANALYSIS-Schematic



• FIFO具体实现

◦ 顶层模块(fifo.v):

```

1  module fifo(
2      input  clk, rst,          //时钟（上升沿有效）、异步复位（高电平有效）
3      input  [7:0] din,        //入队列数据
4      input  en_in,            //入队列使能，高电平有效
5      input  en_out,           //出队列使能，高电平有效
6      output [7:0] dout,       //出队列数据
7      output [4:0] count       //队列数据计数
8  );
9
10     wire re,we; //边沿信号
11     wire ena;   //总使能
12     wire wea;   //写使能
13     wire empty,full; //队列标志
14     wire [3:0] pt_r,pt_w; //队列指针
15     wire [3:0] addr; //RAM地址
16
17     //结构化描述数据通路
18     signal_edge re_gen(.clk(clk),.button(en_out)
19     ,.button_edge(re));
20
21     signal_edge we_gen(.clk(clk),.button(en_in),
22     .button_edge(we));
23
24     //读写地址生成、选择
25     pt_read pt_r_gen(.clk(clk),.rst(rst),.re(re),.we(we),
26     .empty(empty),.pt_r(pt_r));
27
28     pt_write pt_w_gen (.clk(clk),.rst(rst),.we(we),.full (full
29     ),.pt_w(pt_w));
30
31     MUX2to1 mux(.m(we),.d0(pt_r),.d1(pt_w),.out(addr)); //地址选择
32
33     //状态机
34     flag_FSM flag(.clk(clk),.rst(rst),
35     we(we),.re(re),.full(full),.empty(empty),.count(count));
36
37     //使能信号
38     assign wea = we&(~full);
39     assign ena = we|re;
40
41     //访存
42     blk_mem RAM(.clka(clk),.addra(addr),.dina(din),
43     .ena(ena),.wea(wea),.douta(dout));
44 
```

○ 信号取边沿模块:

```

1 //上学期数字电路实验方法
2 module signal_edge(
3     input clk,
4     input button,
5     output button_edge
6 );
7
8     reg button_r1,button_r2;
9
10    always@(posedge clk)
11        button_r1 <= button;
12    always@(posedge clk)
13        button_r2 <= button_r1;
14
15    assign button_edge = button_r1 & (~button_r2);
16
17 endmodule

```

○ 状态机&信号生成(write优先级高于read):

```

1 module flag_FSM
2     #(parameter MAX_LEN = 16)
3     (
4         input clk,rst,
5         input we,re,
6         output reg full,empty, //Moore型
7         output reg [4:0] count //Mealy型
8     );
9
10    parameter EMPTY = 2'b00;
11    parameter NORMAL = 2'b01;
12    parameter FULL = 2'b10;
13    parameter OTHER = 2'b11;
14
15    reg [1:0] current_state,next_state; //states
16
17    always @(posedge clk,posedge rst)
18    begin
19        if(rst)
20            current_state <= EMPTY;
21        else
22            current_state <= next_state;
23    end
24    //设置优先级, write高于read
25    always @(*)
26    begin
27        case(current_state)
28        EMPTY:
29            begin
30                casex({we,re})
31                    2'b0x:begin next_state = EMPTY ; count = 0; end //reamin
32                    2'b1x:begin next_state = NORMAL; count = 1; end //PUSH

```

```

33         endcase
34     end
35     NORMAL:
36         begin
37             casex({we,re})
38                 2'b00:begin next_state = NORMAL; count = count;end
//remain
39                 2'b01: //POP
40                     begin
41                         count = count-1;
42                         if(count == 0)
43                             next_state = EMPTY; //empty
44                         else
45                             next_state = NORMAL;
46                         end
47                 2'b1x: //PUSH
48                     begin
49                         count = count+1;
50                         if(count == MAX_LEN)
51                             next_state = FULL; //full
52                         else
53                             next_state = NORMAL;
54                         end
55                     endcase
56                 end
57     FULL:
58         begin
59             casex({we,re})
60                 2'b00,2'b1x:begin next_state = FULL; count = count;
end//remain
61                 2'b01:begin next_state =NORMAL;count = count-1; end //POP
62             endcase
63         end
64     OTHER:begin next_state = EMPTY; count = 0; end //完全赋值
65     endcase
66     end
67     //组合输出
68     always @(*)
69     begin
70         case(current_state) //flag
71             EMPTY,OTHER:{empty,full} = 2'b10;
72             NORMAL:{empty,full} = 2'b00;
73             FULL:{empty,full} = 2'b01;
74         endcase
75     end
76
77 endmodule

```

◦ 队头指针生成:

```

1 module pt_read
2     #(parameter MAX_LEN = 16)
3     (
4         input clk,rst,
5         input empty,
6         input re,    //read enable
7         input we,    //write enable

```

```

8      output reg [3:0] pt_r
9      );
10
11     always @(posedge clk,posedge rst)
12     begin
13         if(rst)
14             pt_r <= 4'b0000;    //复位
15         else if({empty,re,we}==3'b010 ) //POP条件
16             pt_r <= (pt_r+1)%MAX_LEN;
17         else
18             pt_r <= pt_r;
19     end
20
21 endmodule

```

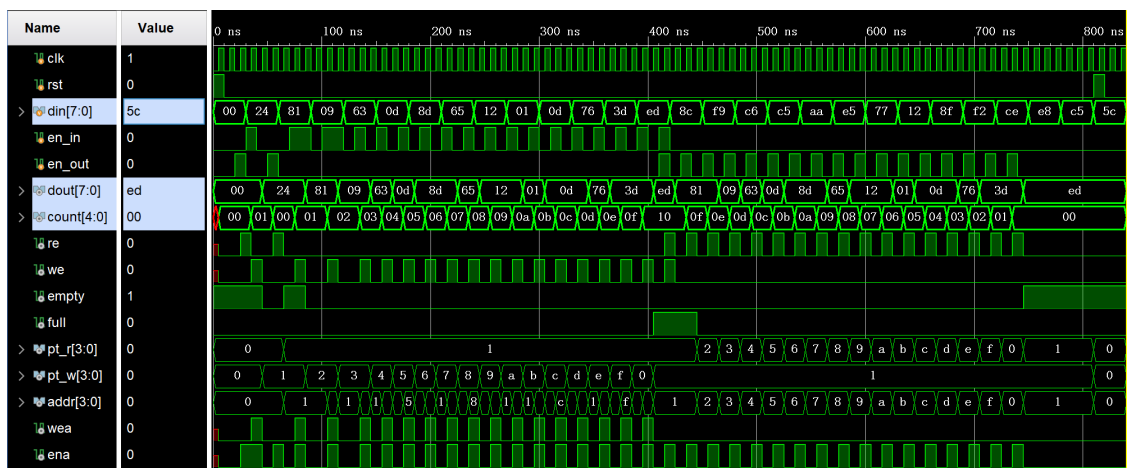
- 队尾指针生成:

```

1  module pt_write
2      #(parameter MAX_LEN = 16)
3      (
4          input clk,rst,
5          input full,
6          input we,
7          output reg [3:0] pt_w
8      );
9
10     always @(posedge clk,posedge rst)
11     begin
12         if(rst)
13             pt_w <= 4'b0000;
14         else if(full==0 && we==1)
15             pt_w <= (pt_w+1)%MAX_LEN;
16         else
17             pt_w <= pt_w;
18     end
19
20 endmodule

```

- Vivado仿真如下:



test_bench(tb_fifo.v):

```

1  module tb_fifo;
2

```

```

3 // fifo Parameters
4 parameter PERIOD = 10;
5
6 // fifo Inputs
7 reg clk = 0 ;
8 reg rst = 1 ;
9 reg [7:0] din = 0 ;
10 reg en_in = 0 ;
11 reg en_out = 0 ;
12
13 // fifo Outputs
14 wire [7:0] dout ;
15 wire [4:0] count ;
16
17 initial
18 begin
19     forever #(PERIOD/2) clk=~clk;
20 end
21
22 initial
23 begin
24     forever
25     #(3*PERIOD) din = $random%256;
26 end
27
28
29 fifo u_fifo (
30     .clk ( clk ),
31     .rst ( rst ),
32     .din ( din [7:0] ),
33     .en_in ( en_in ),
34     .en_out ( en_out ),
35
36     .dout ( dout [7:0] ),
37     .count ( count [4:0] )
38 );
39
40 initial
41 begin
42     #(PERIOD) {en_in,en_out}=2'b00;
43     #(PERIOD) {en_in,en_out}=2'b01; //POP
44     #(PERIOD) {en_in,en_out}=2'b10; //PUSH
45     #(PERIOD) {en_in,en_out}=2'b00; //remain
46     #(PERIOD) {en_in,en_out}=2'b01; //POP
47     #(PERIOD) {en_in,en_out}=2'b00; //remain
48
49     repeat(2)
50     begin
51         #(PERIOD) {en_in,en_out}=2'b10; //PUSH
52         #(PERIOD*2) {en_in,en_out}=2'b00; //remain
53     end
54
55     repeat(14)
56     begin
57         #(PERIOD) {en_in,en_out}=2'b10; //PUSH
58         #(PERIOD) {en_in,en_out}=2'b00; //remain
59     end
60     #(PERIOD) {en_in,en_out}=2'b11; //PUSH, full

```



```

61      #(PERIOD)    {en_in,en_out}=2'b00;    //remain
62
63      repeat(16)
64      begin
65          #(PERIOD)    {en_in,en_out}=2'b01;    //POP
66          #(PERIOD)    {en_in,en_out}=2'b00;    //remain
67      end
68
69      #(PERIOD)    rst=1;    //reset
70      #(PERIOD*10)    $finish;
71  end
72  endmodule

```

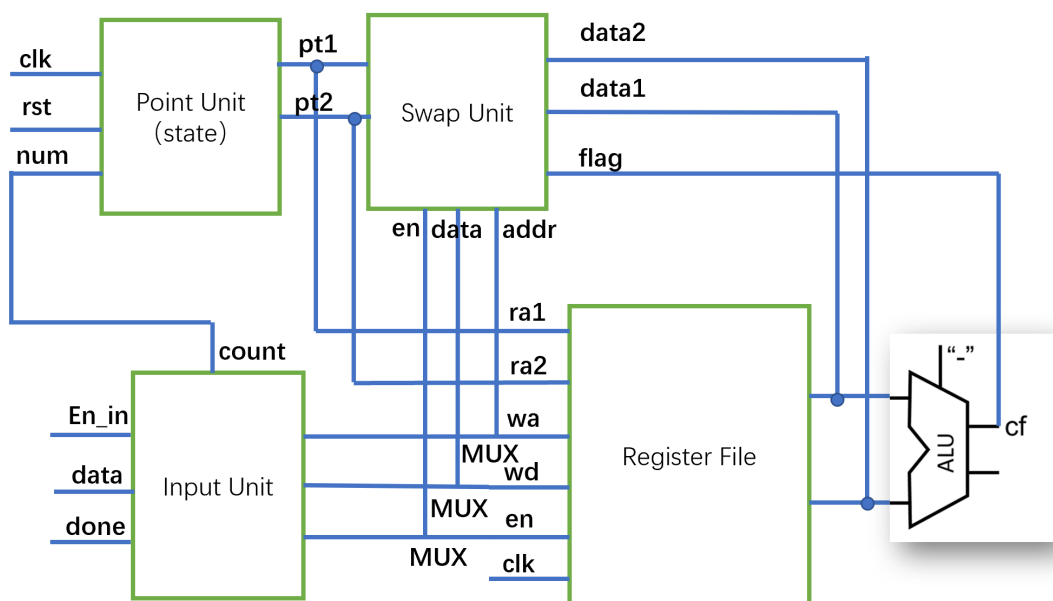
- FPGA开发板测试：
 - 返校后进行

三、思考题

Q：如何利用寄存器堆和适当电路设计实现可变个数的数据排序电路？

A：输入数据存入寄存器堆中，利用指针进行排序

- 数据通路：



- 部件功能表：

部件	功能
Input Unit	数据输入，写到寄存器堆，按下done表示完成输入
Point Unit	状态控制，输出正在比较的两数据的地址
Swap Unit	交换控制，根据信号值和指针地址对寄存器堆修改
Register File	寄存器堆，存储排序的数据
ALU	减法操作，根据借位比较两数大小并反馈到Swap Unit