Lab 2 寄存器堆与存储器及其应用

姓名:傅申 学号: PB20000051 实验日期: 2022-3-15

1 实验题目

寄存器堆与存储器及其应用

2 实验目的

- 掌握寄存器堆 (Register File) 和存储器的功能, 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

3 实验平台

- Xilinx Vivado v2019.1
- Microsoft Visual Studio Code
- FPGAOL

4 实验过程

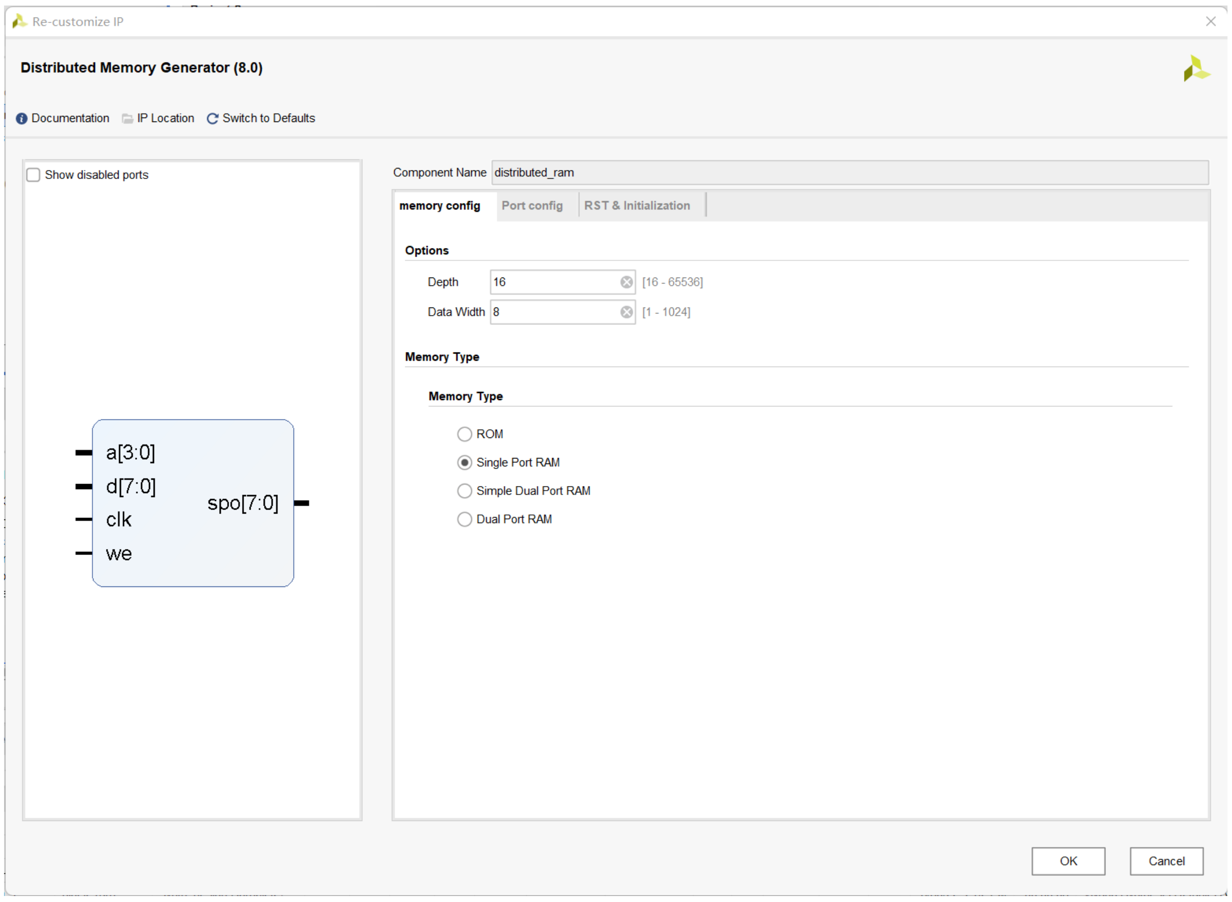
4.1 寄存器堆

寄存器堆的 Verilog 代码如下

```verilog
// 32 * WIDTH Register File
module register_file #(
    parameter WIDTH = 32
) (
    input  clk,              // Clock (posedge)
    input  [4:0] ra0,        // Read address 0
    input  [4:0] ra1,        // Read address 1
    input  [4:0] wa,         // Write address
    input  we,               // Write enable
    input  [WIDTH-1:0] wd,   // Write data
    output [WIDTH-1:0] rd0,  // Read data 0
    output [WIDTH-1:0] rd1   // Read data 1
);
    reg [WIDTH-1:0] regfile [0:31];
    assign rd0 = regfile[ra0];
    assign rd1 = regfile[ra1];
    always @(posedge clk) begin
        if (we) regfile[wa] <= wd;
    end
endmodule  //register_file
```

4.2 RAM 存储器

4.2.1 分布式 16×8 位单端口 RAM


Re-customize IP

Distributed Memory Generator (8.0)

Documentation IP Location Switch to Defaults

Show disabled ports

Component Name distributed_ram

memory config | Port config | RST & Initialization

Options

Depth 16 [16 - 65536]

Data Width 8 [1 - 1024]

Memory Type

Memory Type

ROM

[✓ Single Port RAM]

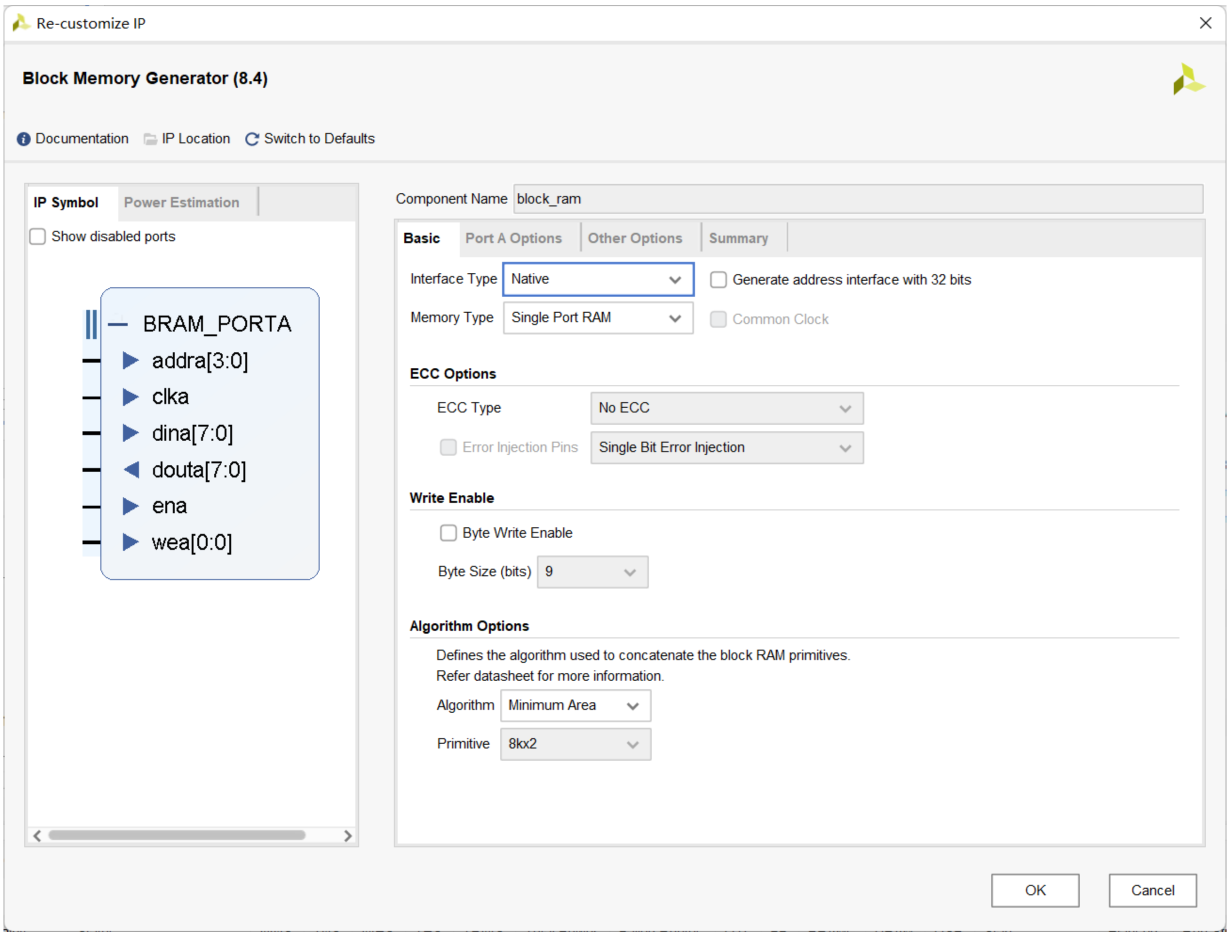
Simple Dual Port RAM

Dual Port RAM

其中 coe 文件如下

```
memory_initialization_radix=16;
memory_initialization_vector=0F 1E 2D 3C 4B 5A 69 78
                             87 96 A5 B4 C3 D2 E1 F0;
```

4.2.2 块式 16×8 位单端口 RAM



Show disabled ports [ ]

Component Name: block_ram

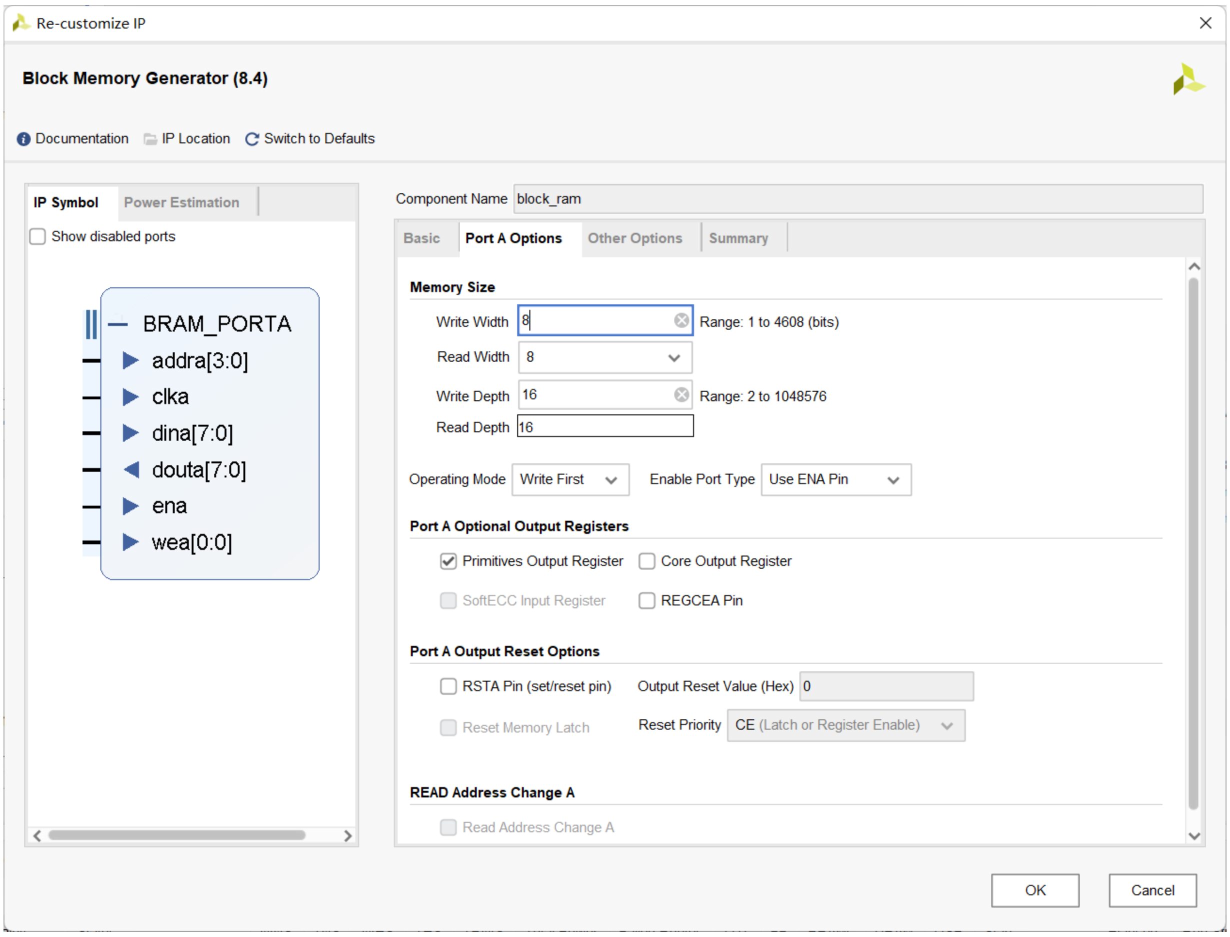
Interface Type: Native
Generate address interface with 32 bits [ ]
Memory Type: Single Port RAM
Common Clock [ ]

ECC Type: No ECC
Error Injection Pins [ ]: Single Bit Error Injection

Byte Write Enable [ ]
Byte Size (bits): 9

Defines the algorithm used to concatenate the block RAM primitives.
Refer datasheet for more information.
Algorithm: Minimum Area
Primitive: 8kx2

它的数据通路如下,


Re-customize IP

Block Memory Generator (8.4)

Documentation IP Location Switch to Defaults

IP Symbol | Power Estimation

Show disabled ports

Component Name: block_ram

Basic | Port A Options | Other Options | Summary

Memory Size

Write Width: 8 — Range: 1 to 4608 (bits)
Read Width: 8
Write Depth: 16 — Range: 2 to 1048576
Read Depth: 16

Operating Mode: Write First Enable Port Type: Use ENA Pin

Port A Optional Output Registers

[✓] Primitives Output Register [ ] Core Output Register
[ ] SoftECC Input Register [ ] REGCEA Pin

Port A Output Reset Options

[ ] RSTA Pin (set/reset pin) Output Reset Value (Hex): 0
[ ] Reset Memory Latch Reset Priority: CE (Latch or Register Enable)

READ Address Change A

[ ] Read Address Change A

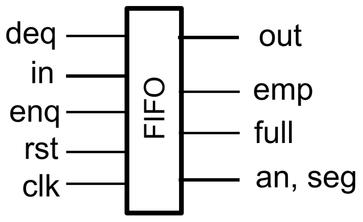
OK Cancel

其中 coe 文件如下

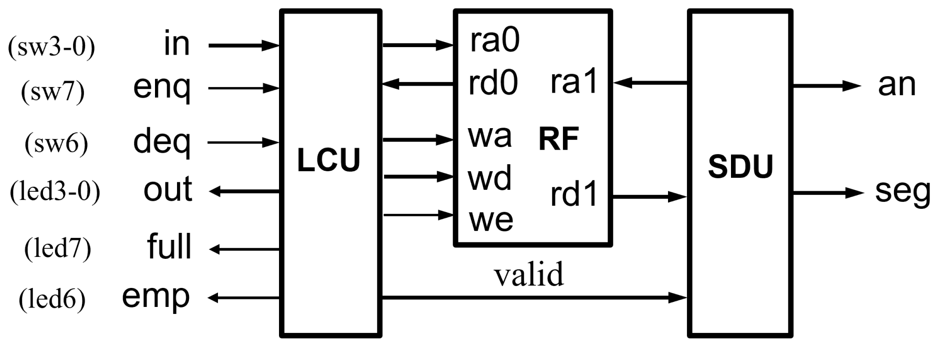
```
memory_initialization_radix=16;
memory_initialization_vector=0F 1E 2D 3C 4B 5A 69 78
                             87 96 A5 B4 C3 D2 E1 F0;
```

4.3 利用寄存器堆实现 FIFO 队列

FIFO 队列的输入输出如下图



它的数据通路如下, 其中 enq 与 deq 信号都由 SEDG 模块两级同步并取边缘后输入到 LCU 模块.



* 省略了clk（100MHz）和 rst（button）

在 Vivado 项目中, 设计文件层次如下

```
fifo (FIFO.v)
├── RF:       reg_file          (RegisterFile.v)
├── SEDG_enq: sig_edge          (SignalEdge.v)
├── SEDG_deq: sig_edge          (SignalEdge.v)
├── LCU:      list_control_unit (ListControlUnit.v)
└── SDU:      segplay_unit      (SegDisplayUnit.v)
```

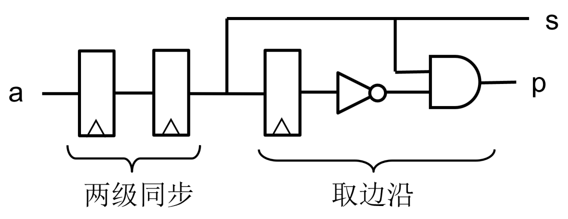
4.3.1 寄存器堆模块

该模块实现了一个 8×16 位寄存器堆.

```
// A 8*16 register file
module reg_file (
    input  clk,         // clock (posedge)
    input  [2:0] ra0,   // read address 0
    input  [2:0] ra1,   // read address 1
    input  we,          // write enable
    input  [2:0] wa,    // write address
    input  [3:0] wd,    // write data
    output [3:0] rd0,   // read data 0
    output [3:0] rd1    // read data 1
);
    reg [3:0] regfile [0:7];
    assign rd0 = regfile[ra0];
    assign rd1 = regfile[ra1];
    always @(posedge clk) begin
        if (we) regfile[wa] = wd;
    end
endmodule  //reg_file
```

4.3.2 取边缘模块

该模块按照下面的电路对信号进行两级同步并取边缘.



```verilog
module sig_edge(
    input  clk, // clock (100 MHz, posedge)
    input  a,   // input signal
    output s,   // sychronized signal
    output p    // signal edge
);
    reg a_reg_0;
    reg a_reg_1;
    reg a_reg_2;
    always @(posedge clk) begin
        a_reg_0 <= a;
        a_reg_1 <= a_reg_0;
        a_reg_2 <= a_reg_1;
    end
    assign s = a_reg_1;
    assign p = a_reg_1 & ~a_reg_2;
endmodule
```

4.3.3 列控制单元模块

该模块处理输入信号, 对寄存器堆进行写入, 并向显示单元发送 valid 信号. 由于 enq 与 deq 在经过取边缘后每次有效的时间都是一个周期, 并且进入三个状态的条件与当前状态, 所以这里没有用到状态机.

在该模块中, 我定义了两个寄存器变量: 头指针 head 与尾指针 tail, 两者都指向下一次入/出队的数据下标, 即初始值均为 0. 在入队后, 尾指针 tail 自增, 在出队后, 头指针 head 自增. 对于判断队列是否为空 (满), 这里只需要判断 valid 的每一位是否都是 0 (1). 模块的具体代码如下

```verilog
module list_control_unit(
    input              clk,          // clock (100 MHz, posedge)
    input              rst,          // sychronous reset (active high)
    input        [3:0] in,
    input              enq,          // enqueue edge
    input              deq,          // dequeue edge
    input        [3:0] rd,           // read data
    output             full,
    output             emp,          // empty
    output reg   [3:0] out,          // dequqed data
    output       [2:0] ra,           // read address
    output             we,           // write enable
    output       [2:0] wa,           // write address
    output       [3:0] wd,           // write data
    output reg   [7:0] valid
);
    reg [2:0] head;  // pointer to head
    reg [2:0] tail;  // pointer to tail

    assign full = &valid;
    assign emp  = ~(|valid);

    assign ra = head;
    assign we = enq & ~full & ~rst;
    assign wa = tail;
    assign wd = in;

    always @(posedge clk) begin
        if (rst) begin
            valid <= 8'h00;
            head  <= 3'h0;
            tail  <= 3'h0;
            out   <= 3'h0;
        end
        else if(enq & ~full) begin
            valid[tail] <= 1'b1;
            tail        <= tail + 3'h1;
        end
        else if(deq & ~emp) begin
            valid[head] <= 1'b0;
            head        <= head + 3'h1;
            out         <= rd;
        end
    end
endmodule
```

4.3.4 数码管显示单元模块

在显示单元中, 输入的时钟信号是 100MHz 的, 对于数码管来说频率过快, 因此这里对其降频到 400Hz.

我使用了一个 18 位的模 250000 计数器, 在每个 100MHz 的时钟上升沿进行计数, 并在进位 (计数器值大于等于 249999) 时对输出的地址 (输出到寄存器堆的 **ra0**) 进行加一, 实现对寄存器堆 400Hz 的扫描. 同时, 在计数器值为 1 时会输出一个脉冲, 用于控制数码管信号的输出.

若队列为空, 则显示单元会输出信号使得最低位数码管显示 0, 否则会使得有效位的数码管上显示该位的值.

具体的模块代码如下

```verilog
module segplay_unit(
    input               clk_100mhz,
    input         [3:0] data,
    input         [7:0] valid,
    output reg    [2:0] addr,
    output        [2:0] segplay_an,
    output        [3:0] segplay_data
);
```

```
    // Counter: slow the clock down to 400Hz
    wire clk_400hz;
    reg [17:0] clk_cnt;
    assign clk_400hz = ~(|clk_cnt);     // clk_400hz = (clk_cnt == 0)
    always @(posedge clk_100mhz) begin
        if (clk_cnt >= 18'h3D08F) begin // clk_cnt >= 249999
            clk_cnt <= 18'h00000;
            addr <= addr + 3'b001;
        end else
            clk_cnt <= clk_cnt + 18'h00001;
    end
    // Generate segplay output
    reg [2:0] segplay_an_reg;
    reg [3:0] segplay_data_reg;
    always @(posedge clk_100mhz) begin
        if (clk_400hz && valid[addr]) begin
            segplay_an_reg <= addr;
            segplay_data_reg <= data;
        end
    end
    assign segplay_data = (|valid) ? segplay_data_reg : 4'h0;
    assign segplay_an = (|valid) ? segplay_an_reg : 3'h0;
endmodule
```

4.3.5 顶层模块 FIFO

顶层模块主要做连线任务, 这里不多赘述.

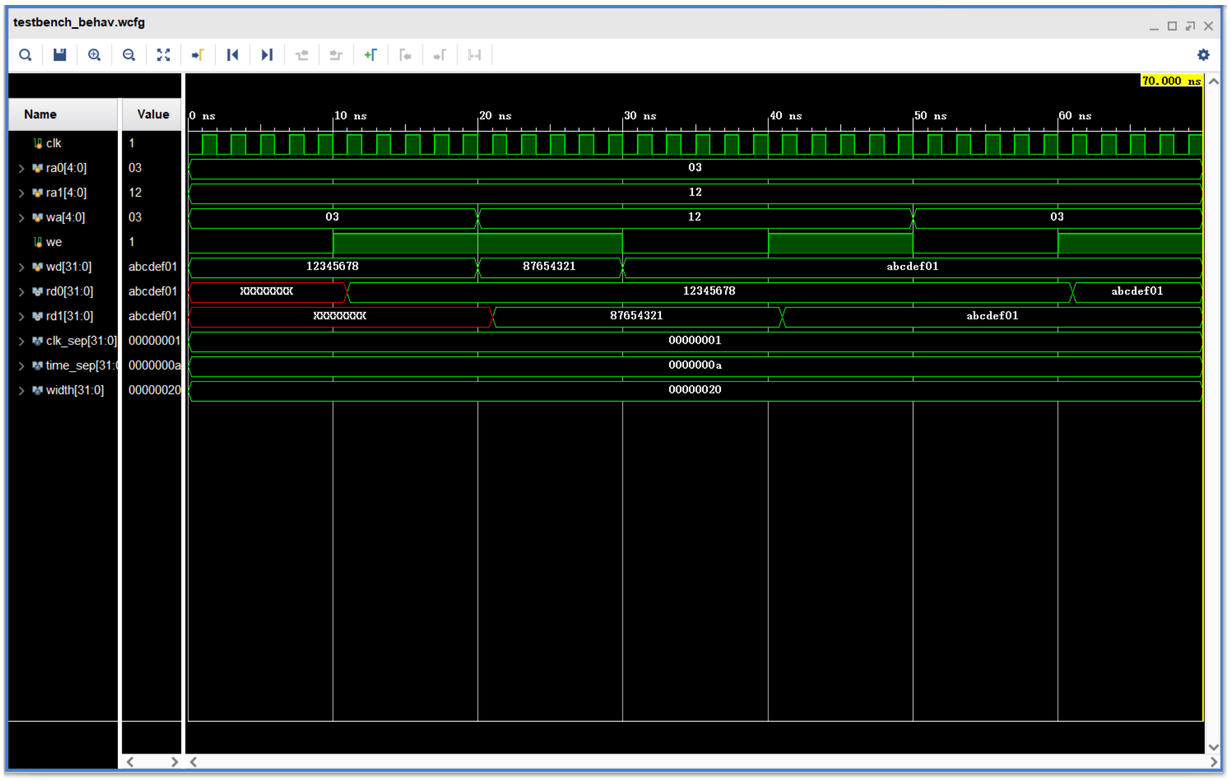
```
module fifo(
    input           clk,    // clock (100 MHz, posedge)
    input           rst,    // sychronous reset (active high)
    input           enq,    // enqueue (active high)
    input   [3:0]   in,     // enqueue data
    input           deq,    // dequeue (active high)
    output  [3:0]   out,    // dequeue data
    output          full,   // queue full
    output          emp,    // queue empty
    output  [2:0]   an,     // segment display selection
    output  [3:0]   seg     // segment display data
);
    // wires
    wire        enq_edge;
    wire        deq_edge;
    wire        we;
    wire [2:0]  ra0, ra1, wa;
    wire [3:0]  rd0, rd1, wd;
    wire [7:0]  valid;
    // datapath
    reg_file RF(
        .clk(clk),
        .ra0(ra0),
        .ra1(ra1),
        .we (we),
        .wa (wa),
        .wd (wd),
```

```verilog
        .rd0(rd0),
        .rd1(rd1)
    );

    sig_edge SEDG_enq(
        .clk(clk),
        .a  (enq),
        .p  (enq_edge)
    );
    sig_edge SEDG_deq(
        .clk(clk),
        .a  (deq),
        .p  (deq_edge)
    );

    list_control_unit LCU(
        .clk  (clk),
        .rst  (rst),
        .in   (in),
        .enq  (enq_edge),
        .deq  (deq_edge),
        .rd   (rd0),
        .full (full),
        .emp  (emp),
        .out  (out),
        .ra   (ra0),
        .we   (we),
        .wa   (wa),
        .wd   (wd),
        .valid(valid)
    );

    segplay_unit SDU(
        .clk_100mhz  (clk),
        .data        (rd1),
        .valid       (valid),
        .addr        (ra1),
        .segplay_an  (an),
        .segplay_data(seg)
    );
endmodule
```

5 实验结果

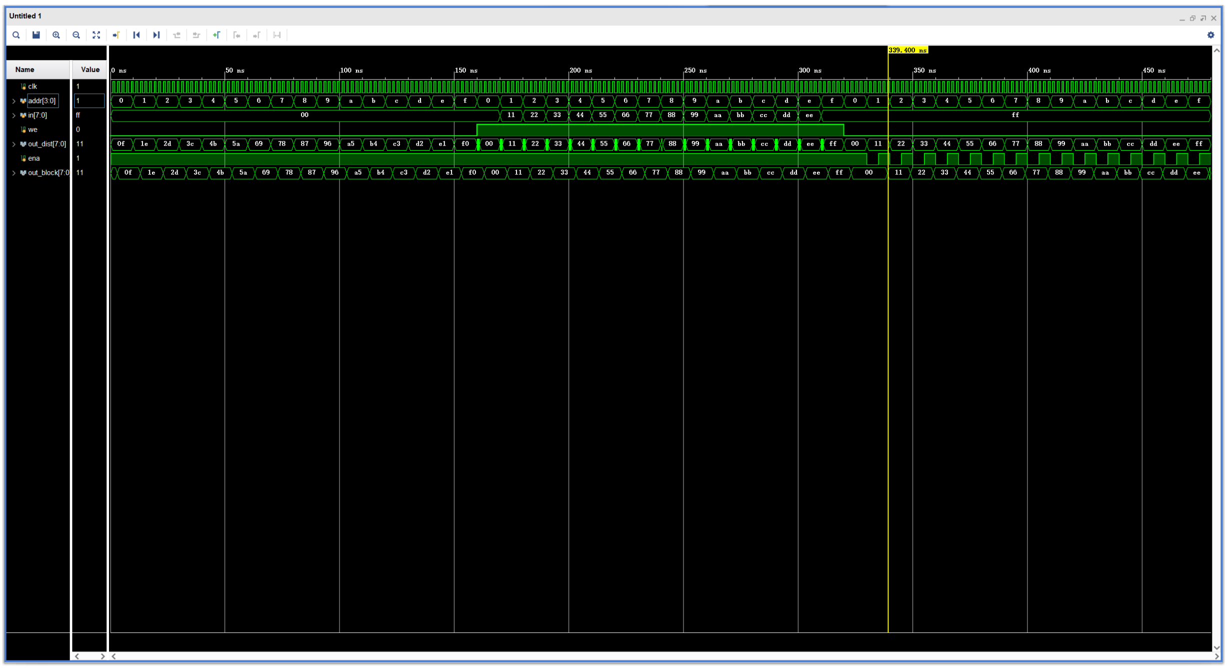
5.1 寄存器堆

针对寄存器堆的仿真文件, 仿真结果如下:



5.2 RAM 存储器

针对RAM存储器的仿真文件, 仿真结果如下:

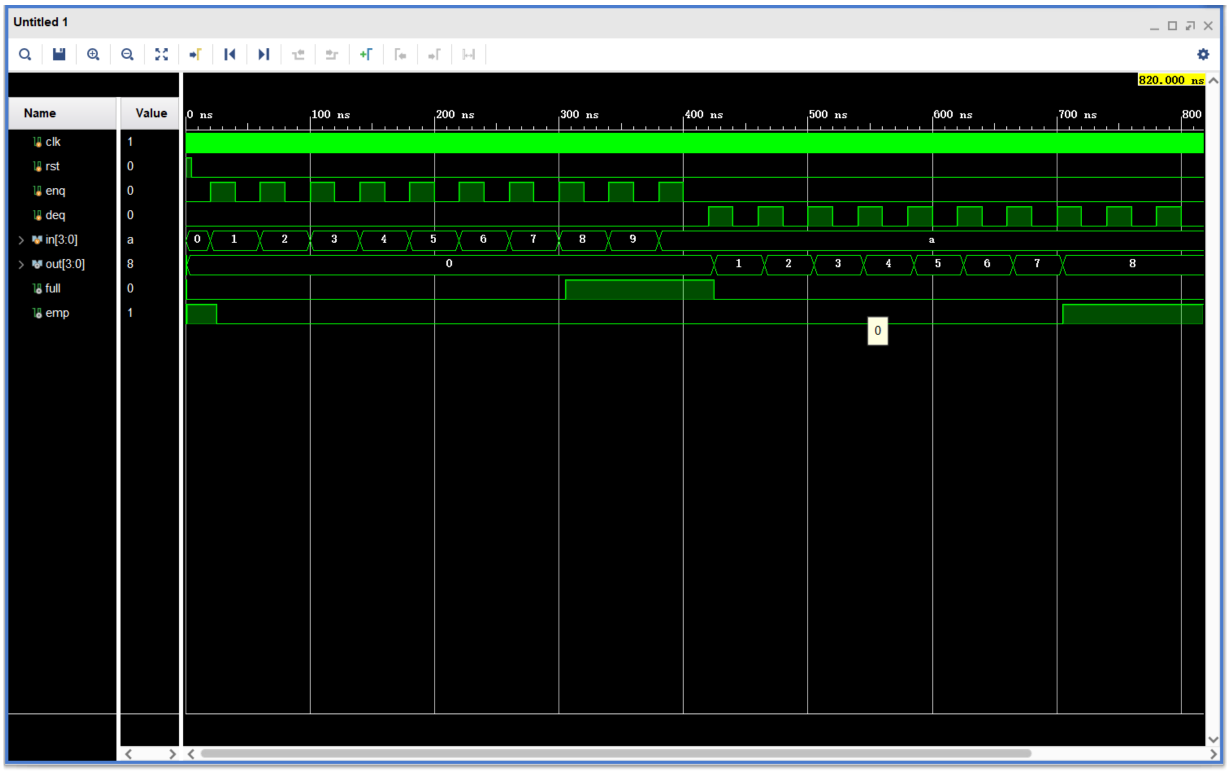


可以看到块存储器的输出要比输入慢, 并且 ena 信号可以控制块存储器的输出.

5.3 FIFO 队列

5.3.1 仿真

针对FIFO队列的仿真文件, 仿真结果如下:

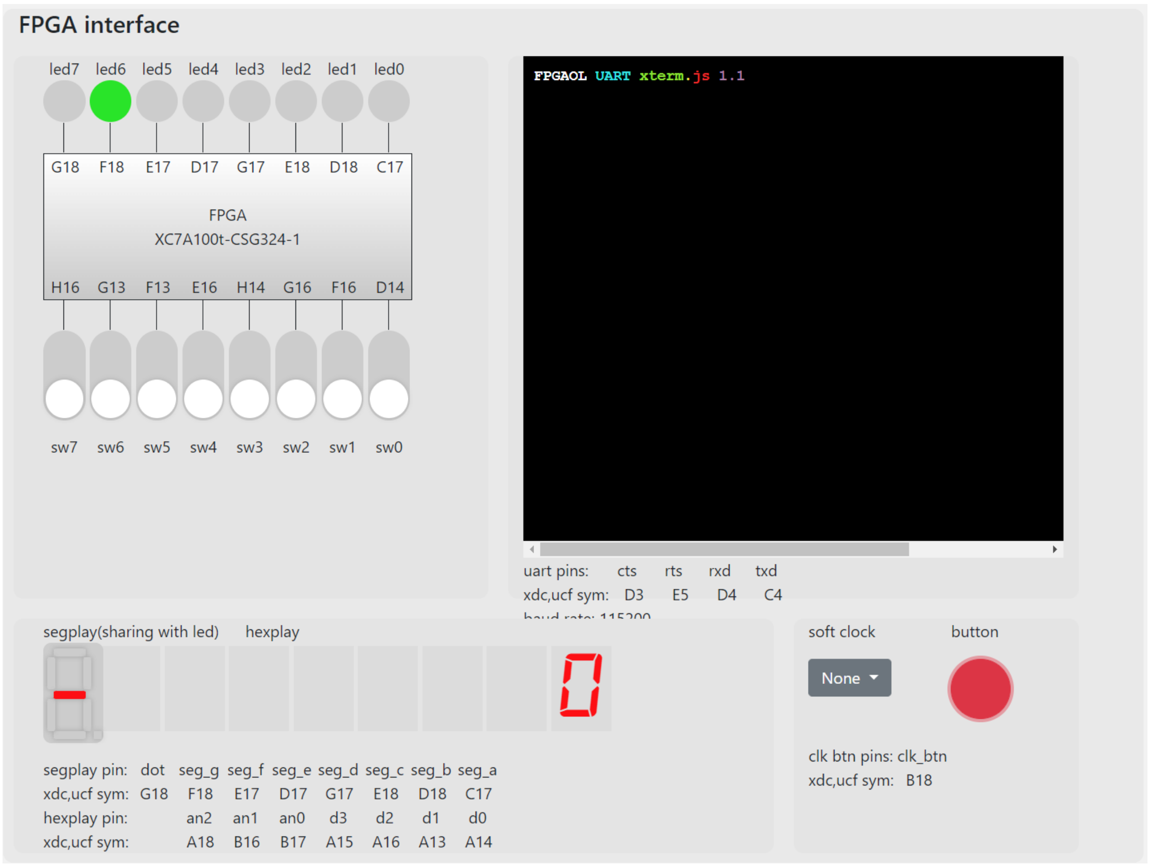


其中数码管显示信号没有进行仿真.

5.3.2 下载测试

因为在检查中已经演示过了, 这里只展示几个特殊的场景. 或者在这里查看视频.

- 队空/初始状态


FPGA interface
led7 led6 led5 led4 led3 led2 led1 led0
G18 F18 E17 D17 G17 E18 D18 C17
FPGA
XC7A100t-CSG324-1
H16 G13 F13 E16 H14 G16 F16 D14
sw7 sw6 sw5 sw4 sw3 sw2 sw1 sw0
segplay(sharing with led)
hexplay
soft clock
button
None
0

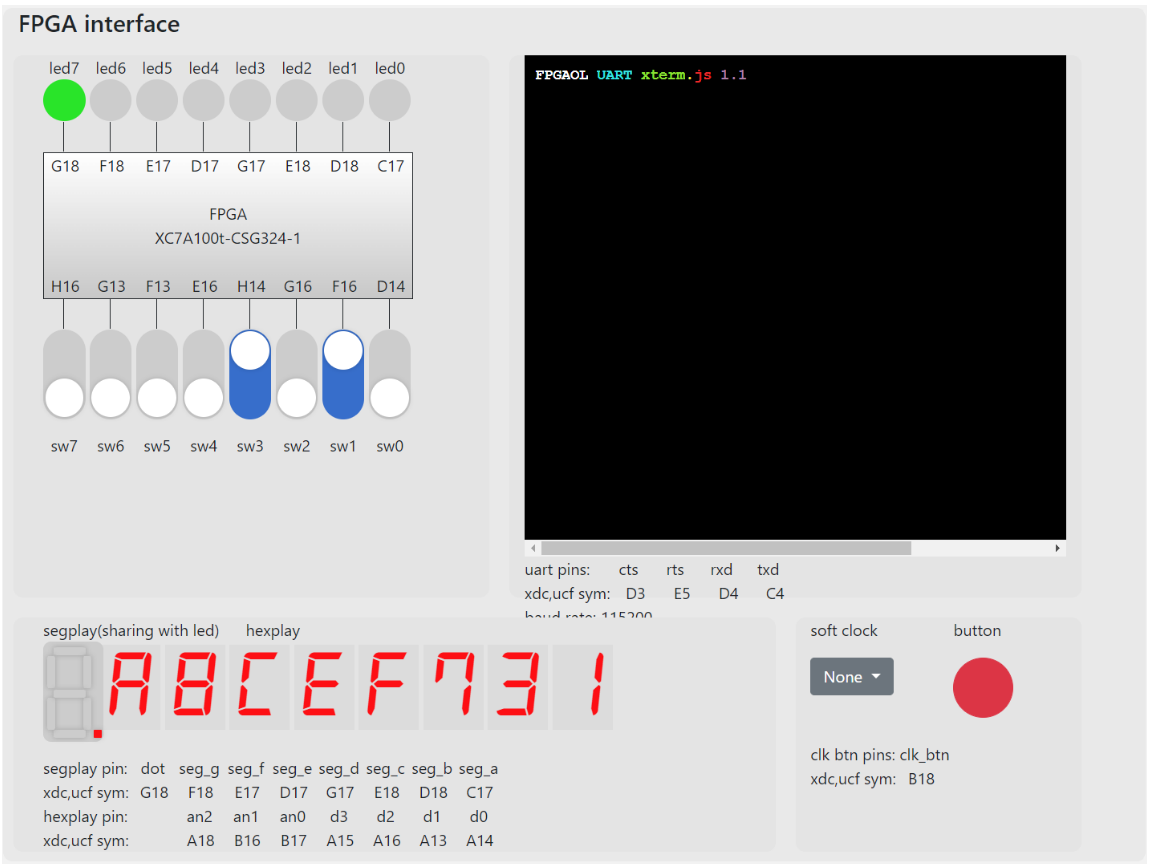
```
FPGAOL UART xterm.js 1.1
```

uart pins: cts rts rxd txd
xdc,ucf sym: D3 E5 D4 C4

segplay pin:	dot	seg_g	seg_f	seg_e	seg_d	seg_c	seg_b	seg_a
xdc,ucf sym:	G18	F18	E17	D17	G17	E18	D18	C17
hexplay pin:		an2	an1	an0	d3	d2	d1	d0
xdc,ucf sym:		A18	B16	B17	A15	A16	A13	A14

clk btn pins: clk_btn
xdc,ucf sym: B18

- 队满


FPGA interface
led7 led6 led5 led4 led3 led2 led1 led0
G18 F18 E17 D17 G17 E18 D18 C17
FPGA
XC7A100t-CSG324-1
H16 G13 F13 E16 H14 G16 F16 D14
sw7 sw6 sw5 sw4 sw3 sw2 sw1 sw0
segplay(sharing with led)
hexplay
soft clock
button
None
ABCEF731

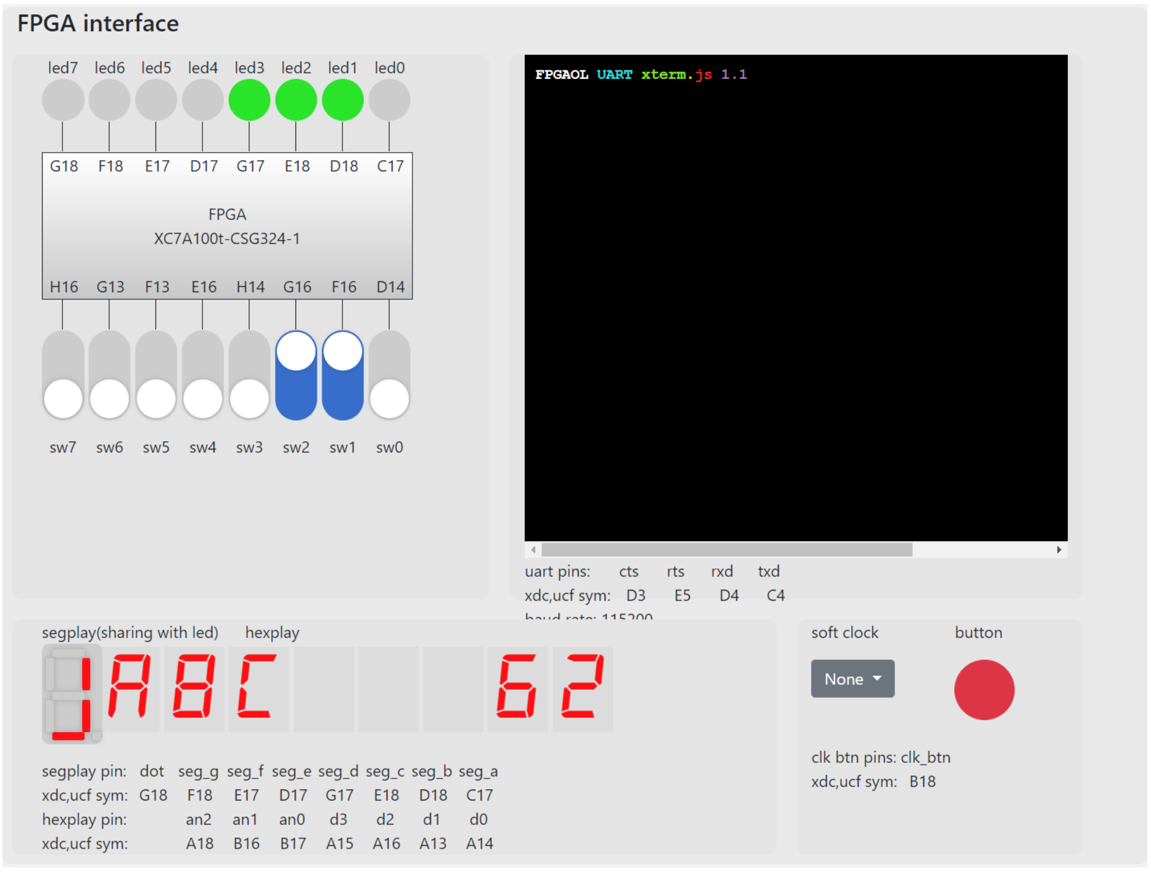
```
FPGAOL UART xterm.js 1.1
```

uart pins: cts rts rxd txd
xdc,ucf sym: D3 E5 D4 C4

segplay pin:	dot	seg_g	seg_f	seg_e	seg_d	seg_c	seg_b	seg_a
xdc,ucf sym:	G18	F18	E17	D17	G17	E18	D18	C17
hexplay pin:		an2	an1	an0	d3	d2	d1	d0
xdc,ucf sym:		A18	B16	B17	A15	A16	A13	A14

clk btn pins: clk_btn
xdc,ucf sym: B18

- 循环队列展示


FPGA interface
led7 led6 led5 led4 led3 led2 led1 led0
G18 F18 E17 D17 G17 E18 D18 C17
FPGA
XC7A100t-CSG324-1
H16 G13 F13 E16 H14 G16 F16 D14
sw7 sw6 sw5 sw4 sw3 sw2 sw1 sw0
segplay(sharing with led)
hexplay
soft clock
button
None

```
FPGAOL UART xterm.js 1.1
```

uart pins:	cts	rts	rxd	txd
xdc,ucf sym:	D3	E5	D4	C4

baud rate: 115200

segplay pin:	dot	seg_g	seg_f	seg_e	seg_d	seg_c	seg_b	seg_a
xdc,ucf sym:	G18	F18	E17	D17	G17	E18	D18	C17
hexplay pin:		an2	an1	an0	d3	d2	d1	d0
xdc,ucf sym:		A18	B16	B17	A15	A16	A13	A14

clk btn pins:	clk_btn
xdc,ucf sym:	B18

6　心得体会

实验难度适中.

7 附录: 仿真文件

7.1 寄存器堆仿真文件

寄存器堆仿真结果

```
`timescale 1ns / 1ps
module testbench();
    parameter clk_sep   = 1;
    parameter time_sep  = 10;
    parameter width     = 32;
    reg       clk;
    reg       [4:0] ra0;
    reg       [4:0] ra1;
    reg       [4:0] wa;
    reg       we;
    reg       [width-1:0] wd;
    wire      [width-1:0] rd0;
    wire      [width-1:0] rd1;
    register_file regfile(
        .clk(clk),
        .ra0(ra0),
        .ra1(ra1),
        .wa(wa),
        .we(we),
        .wd(wd),
        .rd0(rd0),
        .rd1(rd1)
    );
    initial begin
        clk = 0;
        ra0 = 5'h03;
        ra1 = 5'h12;
        forever #clk_sep clk = ~clk;
    end
    initial begin
        we = 1'b0;
        wa = 5'h03;
        wd = 32'h12345678;
        #time_sep
        we = 1'b1;
        #time_sep
        wa = 5'h12;
        wd = 32'h87654321;
        #time_sep
        we = 1'b0;
        wd = 32'habcdef01;
        #time_sep
        we = 1'b1;
        #time_sep
        we = 1'b0;
        wa = 5'h03;
        #time_sep
        we = 1'b1;
        #time_sep
        $finish;
    end
endmodule
```

寄存器堆仿真结果

7.2 RAM存储器仿真文件

RAM存储器仿真结果

```
module tb();
    // shared input signals
    reg         clk;
    reg  [3:0]  addr;
    reg  [7:0]  in;
    reg         we;
    initial begin
        clk <= 1'b0;
        forever
            #1 clk <= ~clk;
    end
    initial begin
        addr <= 4'h0;
        in   <= 8'h00;
        we   <= 1'b0;
        #10 addr <= 4'h1;
        #10 addr <= 4'h2;
        #10 addr <= 4'h3;
        #10 addr <= 4'h4;
        #10 addr <= 4'h5;
        #10 addr <= 4'h6;
        #10 addr <= 4'h7;
        #10 addr <= 4'h8;
        #10 addr <= 4'h9;
        #10 addr <= 4'hA;
        #10 addr <= 4'hB;
        #10 addr <= 4'hC;
        #10 addr <= 4'hD;
        #10 addr <= 4'hE;
        #10 addr <= 4'hF;
        #10 addr <= 4'h0;
        in   <= 8'h00;
        we   <= 1'b1;
        #10 addr <= 4'h1;
        in   <= 8'h11;
        #10 addr <= 4'h2;
        in   <= 8'h22;
        #10 addr <= 4'h3;
        in   <= 8'h33;
        #10 addr <= 4'h4;
        in   <= 8'h44;
        #10 addr <= 4'h5;
        in   <= 8'h55;
        #10 addr <= 4'h6;
```

```verilog
        in   <= 8'h66;
        #10 addr <= 4'h7;
        in   <= 8'h77;
        #10 addr <= 4'h8;
        in   <= 8'h88;
        #10 addr <= 4'h9;
        in   <= 8'h99;
        #10 addr <= 4'hA;
        in   <= 8'hAA;
        #10 addr <= 4'hB;
        in   <= 8'hBB;
        #10 addr <= 4'hC;
        in   <= 8'hCC;
        #10 addr <= 4'hD;
        in   <= 8'hDD;
        #10 addr <= 4'hE;
        in   <= 8'hEE;
        #10 addr <= 4'hF;
        in   <= 8'hFF;
        #10 addr <= 4'h0;
        we   <= 1'b0;
        #10 addr <= 4'h1;
        #10 addr <= 4'h2;
        #10 addr <= 4'h3;
        #10 addr <= 4'h4;
        #10 addr <= 4'h5;
        #10 addr <= 4'h6;
        #10 addr <= 4'h7;
        #10 addr <= 4'h8;
        #10 addr <= 4'h9;
        #10 addr <= 4'hA;
        #10 addr <= 4'hB;
        #10 addr <= 4'hC;
        #10 addr <= 4'hD;
        #10 addr <= 4'hE;
        #10 addr <= 4'hF;
        #10 $finish;
    end
    // block memory
    reg         ena;
    wire [7:0]  out_block;
    initial begin
        ena <= 1'b1;
        #330 ena <= 1'b0;
        forever
            #5 ena <= ~ena;
    end
    block_ram       test_block(
        .clka(clk),
        .addra(addr),
        .dina(in),
        .douta(out_block),
        .ena(ena),
        .wea(we)
    );
    // distributed memory
    wire [7:0]  out_dist;
    distributed_ram test_dist(
        .clk(clk),
        .a(addr),
        .d(in),
        .we(we),
        .spo(out_dist)
    );
endmodule
```

RAM 存储器仿真结果

7.3 FIFO队列仿真文件

FIFO队列仿真结果

```verilog
`timescale 1ns / 1ps
module tb();
    reg         clk;
    reg         rst;
    reg         enq;
    reg         deq;
    reg  [3:0]  in;
    wire [3:0]  out;
    wire        full;
    wire        emp;

    fifo        test(
        .clk(clk),
        .rst(rst),
        .enq(enq),
        .deq(deq),
        .in(in),
        .out(out),
        .full(full),
        .emp(emp)
    );

    initial begin
        clk <= 1'b0;
        forever
            #1 clk <= ~clk;
    end

    initial begin
        rst <= 1'b1;
        #5  rst <= 1'b0;
    end

    initial begin
        enq <= 1'b0;
        deq <= 1'b0;
        in  <= 4'h0;
        #20 enq <= 1'b1;    // 1st enqueue
        in  <= 4'h1;
        #20 enq <= 1'b0;
        #20 enq <= 1'b1;    // 2nd enqueue
        in  <= 4'h2;
        #20 enq <= 1'b0;
        #20 enq <= 1'b1;    // 3rd enqueue
        in  <= 4'h3;
        #20 enq <= 1'b0;
        #20 enq <= 1'b1;    // 4th enqueue
        in  <= 4'h4;
        #20 enq <= 1'b0;
        #20 enq <= 1'b1;    // 5th enqueue
        in  <= 4'h5;
        #20 enq <= 1'b0;
        #20 enq <= 1'b1;    // 6th enqueue
        in  <= 4'h6;
        #20 enq <= 1'b0;
        #20 enq <= 1'b1;    // 7th enqueue
        in  <= 4'h7;
        #20 enq <= 1'b0;
        #20 enq <= 1'b1;    // 8th enqueue
        in  <= 4'h8;
        #20 enq <= 1'b0;
        #20 enq <= 1'b1;    // 9th enqueue (invalid)
        in  <= 4'h9;
        #20 enq <= 1'b0;
        #20 enq <= 1'b1;    // 10th enqueue (invalid)
        in  <= 4'hA;
        #20 enq <= 1'b0;
        #20 deq <= 1'b1;    // 1st dequeue
        #20 deq <= 1'b0;
        #20 deq <= 1'b1;    // 2nd dequeue
        #20 deq <= 1'b0;
        #20 deq <= 1'b1;    // 3rd dequeue
        #20 deq <= 1'b0;
        #20 deq <= 1'b1;    // 4th dequeue
        #20 deq <= 1'b0;
        #20 deq <= 1'b1;    // 5th dequeue
        #20 deq <= 1'b0;
        #20 deq <= 1'b1;    // 6th dequeue
        #20 deq <= 1'b0;
        #20 deq <= 1'b1;    // 7th dequeue
        #20 deq <= 1'b0;
        #20 deq <= 1'b1;    // 8th dequeue
        #20 deq <= 1'b0;
        #20 deq <= 1'b1;    // 9th dequeue (invalid)
        #20 deq <= 1'b0;
        #20 deq <= 1'b1;    // 10th dequeue (invalid)
        #20 deq <= 1'b0;
        #20 $finish;
    end
endmodule
```

FIFO 队列仿真结果