**中国科学技术大学计算机学院**

**《数字电路实验》报告**



实验题目：\_\_\_综合实验\_\_\_\_

学生姓名：\_\_\_吴毅龙\_\_\_\_\_\_

学生学号：\_\_\_PB19111749\_\_

完成日期：\_\_2020/12/25\_\_\_

计算机实验教学中心制

2020年09月

【实验题目】

综合实验

【实验目的】

熟练掌握前面实验中的所有知识点  
熟悉几种常用通信接口的工作原理及使用  
独立完成具有一定规模的功能电路设计

【实验环境】

VLAB： vlab.ustc.edu.cn  
FPGAOL: fpgaol.ustc.edu.cn（或 Nexys4 DDR）  
Logisim  
Vivado

【实验练习】

综合实验中根据之前学习的关于数字电路知识基于Vivado开发环境，进行对ROM，单端口RAM，双端口RAM，FIFO存储结构的实现

●ROM

1. //位宽为8bit的ROM
2. module rom(
3. input       clk,         //时钟信号
4. input       cs\_n,        //片选信号
5. input [2:0] addm,        //地址信号
6. output reg [7:0] dout);  //输出数据
7. reg [7:0] rom[7:0];       //8\*8bit寄存器
9. initial
10. begin
11. rom[0] = 8'b0000\_0000;
12. rom[1] = 8'b0000\_0001;
13. rom[2] = 8'b0000\_0010;
14. rom[3] = 8'b0000\_0011;
15. rom[4] = 8'b0000\_0100;
16. rom[5] = 8'b0000\_0101;
17. rom[6] = 8'b0000\_0110;
18. rom[7] = 8'b0000\_0111;
19. end
20. always@(posedge clk)
21. begin
22. **if**(cs\_n)
23. dout <= 8'bzzzz\_zzzz;
24. **else**
25. dout <= rom[addm];
26. end
28. endmodule

●单端口RAM

1. //位宽为8bit的单端口RAM
2. module ram\_single(
3. input        clk,        //时钟信号
4. input  [2:0] addm,       //地址信号
5. input        cs\_n,       //片选信号
6. input        we\_n,       //写入使能信号
7. input  [7:0] din,        //输入数据
8. output reg [7:0] dout);  //输出数据
9. reg [7:0]raml[7:0];       //8\*8bit寄存器
11. always@(posedge clk)
12. begin
13. **if**(cs\_n)
14. dout <= 8'bzzzz\_zzzz;
15. **else**
16. begin
17. **if**(we\_n)
18. dout <= raml[addm];
19. **else**
20. raml[addm] <= din;
21. end
22. end
23. endmodule

●双端口RAM

1. //位宽为8bit的双端口RAM
2. module ram\_daul(
3. input [7:0] d,         //输入数据
4. input [2:0] addr\_in,   //写入数据地址信号
5. input [2:0] addr\_out,  //输出数据地址信号
6. input       we,        //写数据使能信号
7. input       rd,        //读数据使能信号
8. input       clk1,      //写数据时钟信号
9. input       clk2,      //读数据时钟信号
10. output reg [7:0] q);   //输出数据
11. reg [7:0] mem[7:0];    //8\*8bit寄存器
13. always@(posedge clk1)
14. begin
15. **if**(we)
16. mem[addr\_in] <= d;
17. end
18. always@(posedge clk2)
19. begin
20. **if**(rd)
21. q <= mem[addr\_out];
22. end
23. endmodule

●FIFO

1. module FIFO\_buffer(
2. input       clk,              //时钟信号
3. input       rst,              //复位信号
4. input       write\_to\_stack,   //写入信号
5. input       read\_from\_stack,  //输出信号
6. input  [7:0] Data\_in,         //写入数据
7. output [7:0] Data\_out);       //输出数据
8. wire stack\_full;
9. wire stack\_empty;
10. wire [2:0] addr\_in;           //写入数据地址信号
11. wire [2:0] addr\_out;          //输出数据地址信号
12. wire we\_sig;
13. wire rd\_sig;
15. FIFO\_control FIFO\_control(.stack\_full(stack\_full),.stack\_empty(stack\_empty),.write\_to\_stack(write\_to\_stack),
16. .read\_from\_stack(read\_from\_stack),.write\_ptr(addr\_in),.read\_ptr(addr\_out),.clk(clk),.rst(rst),.we\_sig(we\_sig),.rd\_sig(rd\_sig));
17. ram\_dub ram\_dub(.q(Data\_out),.addr\_in(addr\_in),.addr\_out(addr\_out),.d(Data\_in),.we(we\_sig),.rd(rd\_sig),
18. .clk1(clk),.clk2(clk));
19. endmodule

22. module FIFO\_control(
23. input            write\_to\_stack,
24. input            read\_from\_stack,
25. input            clk,
26. input            rst,
27. output           stack\_full,
28. output           stack\_empty,
29. output reg [2:0] read\_ptr,
30. output reg [2:0] write\_ptr,
31. output reg we\_sig,
32. output reg rd\_sig);
33. parameter stack\_width = 8;
34. parameter stack\_height = 4;
35. parameter stack\_ptr\_width = 3;
36. reg [stack\_width - 1:0]     ptr\_gap;                    //指向栈顶的指针
37. reg [stack\_width - 1:0] Data\_out;
38. reg [stack\_width - 1:0] stack[stack\_height - 1:0];
40. assign stack\_full = (ptr\_gap == stack\_height);
41. assign stack\_empty = (ptr\_gap == 0);
43. always@(posedge clk or posedge rst)
44. begin
45. **if**(rst)
46. begin
47. Data\_out <= 0;
48. read\_ptr <= 0;
49. write\_ptr <= 0;
50. ptr\_gap <= 0;
51. we\_sig <= 1;
52. rd\_sig <= 1;
53. end
54. **else** **if**(write\_to\_stack && read\_from\_stack && (!stack\_full) && (!stack\_empty))
55. begin
56. read\_ptr <= read\_ptr + 1;
57. write\_ptr <= write\_ptr + 1;
58. we\_sig <= 1;
59. rd\_sig <= 1;
60. end
61. **else** **if**((!write\_to\_stack) && read\_from\_stack && (!stack\_full) && (!stack\_empty))
62. begin
63. read\_ptr <= read\_ptr + 1;
64. ptr\_gap <= ptr\_gap + 1;
65. we\_sig <= 0;
66. rd\_sig <= 1;
67. end
68. **else** **if**(write\_to\_stack && (!read\_from\_stack) && (!stack\_full) && (!stack\_empty))
69. begin
70. write\_ptr <= write\_ptr + 1;
71. ptr\_gap <= ptr\_gap + 1;
72. we\_sig <= 1;
73. rd\_sig <= 0;
74. end
75. **else** **if**((!write\_to\_stack) && (!read\_from\_stack) && (!stack\_full) && (!stack\_empty))
76. begin
77. we\_sig <= 0;
78. rd\_sig <= 0;
79. end
80. **else** **if**(write\_to\_stack && read\_from\_stack && stack\_full && (!stack\_empty))
81. begin
82. read\_ptr <= read\_ptr + 1;
83. ptr\_gap <= ptr\_gap + 1;
84. we\_sig <= 0;
85. rd\_sig <= 1;
86. end
87. **else** **if**((!write\_to\_stack) && read\_from\_stack && stack\_full && (!stack\_empty))
88. begin
89. read\_ptr <= read\_ptr + 1;
90. ptr\_gap <= ptr\_gap + 1;
91. we\_sig <= 0;
92. rd\_sig <= 1;
93. end
94. **else** **if**(write\_to\_stack && (!read\_from\_stack) && stack\_full && (!stack\_empty))
95. begin
96. we\_sig <= 0;
97. rd\_sig <= 0;
98. end
99. **else** **if**((!write\_to\_stack) && (!read\_from\_stack) && stack\_full && (!stack\_empty))
100. begin
101. we\_sig <= 0;
102. rd\_sig <= 0;
103. end
104. **else** **if**(write\_to\_stack && read\_from\_stack && (!stack\_full) && stack\_empty)
105. begin
106. write\_ptr <= write\_ptr + 1;
107. ptr\_gap <= ptr\_gap + 1;
108. we\_sig <= 1;
109. rd\_sig <= 0;
110. end
111. **else** **if**(write\_to\_stack && (!read\_from\_stack) && (!stack\_full) && stack\_empty)
112. begin
113. write\_ptr <= write\_ptr + 1;
114. ptr\_gap <= ptr\_gap + 1;
115. we\_sig <= 1;
116. rd\_sig <= 0;
117. end
118. **else** **if**((!write\_to\_stack) && read\_from\_stack && (!stack\_full) && stack\_empty)
119. begin
120. we\_sig <= 0;
121. rd\_sig <= 0;
122. end
123. **else** **if**((!write\_to\_stack) && (!read\_from\_stack) && (!stack\_full) && stack\_empty)
124. begin
125. we\_sig <= 0;
126. rd\_sig <= 0;
127. end
128. end
129. endmodule
131. module ram\_dub(
132. input [7:0] d,         //输入数据
133. input [2:0] addr\_in,   //写入数据地址信号
134. input [2:0] addr\_out,  //输出数据地址信号
135. input       we,        //写数据使能信号
136. input       rd,        //读数据使能信号
137. input       clk1,      //写数据时钟信号
138. input       clk2,      //读数据时钟信号
139. output reg [7:0] q);   //输出数据
140. reg [7:0] mem[7:0];    //8\*8bit寄存器
142. always@(posedge clk1)
143. begin
144. **if**(we)
145. mem[addr\_in] <= d;
146. end
147. always@(posedge clk2)
148. begin
149. **if**(rd)
150. q <= mem[addr\_out];
151. end
152. endmodule

【总结与思考】

通过本次实验，我利用FPGAOL平台进行实验，并使用IP核。本次实验难度不大，根据实验说明的指导就可以完成实验操作，实验题目的难度层层递进，有基础操作的考核，也有所学知识的综合，难易结合，既有复习又有思考，让所学在实践中得以运用，加深了我对逻辑电路知识的理解。希望今后实验可以保持本次实验中详细实验指导描述的优点，辅助完成每项试验内容。