北京工业大学信息学部-软件学院 2022-2023 学年第一学期

《嵌入式微处理器结构与应用》

CPU 设计实验报告

班级 200803

学号 20080309

姓名 常兴阳

1. CPU 设计规格

本次实验中所设计的 CPU 具有以下规格：

1. 一个全功能的 CPU，可以在 FPGA 上综合和仿真；
2. 保证操作正确性的前提下拥有尽量简单的结构；
3. 实现此设计所需的 Verilog 源代码数量较小
4. 本 CPU 可以用汇编语言进行编程，由于没时间编写针对于 C 语言转汇编的编译器，故现不支持 C 语言编程
5. 使用精简完整的指令集，包括中止(HALT)，压栈立即数(PUSHI)，按址压栈(PUSH)，弹出栈(POP)，立即跳转(JMP)，栈顶值为 0 跳转(JZ)，栈顶值为 1 跳转(JNZ)，外部输入(IN)，输出外部(OUT)，操作(OP) 一共 10 条指令;
6. 有一个至少 16 位的体系结构；
7. 支持通过指定方式输入/输出；
8. 支持一般的条件语句操作，通过 JMP、JZ、JNZ 实现；
9. 采用简洁的堆栈体系结构，除了 PC、IR、Outbuf 以外甚至没有其他寄存器。

2. 体系结构

1. 在本 CPU 中，信息通过 16 位宽的数据总线 dbus 进行传输；
2. 通过内部程序储存器 RAM0 进行程序及数据的存储，通过地址总线 abus 进行数据传输；
3. 通过 RAM0 存储程序、数据的存储、堆栈 stack0 提供快速读写数据寄存、数字/逻辑计算单元 alu0 进行一元或二元运算、输出寄存器 obuf0 输出最终执行结果 以及连接此些的数据总线和地址总线实现整体的逻辑运算；
4. 通过控制器 state0 控制系统内部运行流程，通过指令计数器 pc0、指令寄存器 ir0、stack0、alu、RAM0、obuf0 实现了指令具体化为操作并指导 CPU 运行，在此期间 abus、dbus 实现将指令和操作在原件之间传输。

3. 指令集设计

为实现简单但完整的结构，该 CPU 的指令集包括以下 10 条：中止(HALT)，压栈立即数(PUSHI)，按址压栈(PUSH)，弹出栈(POP)，立即跳转(JMP)，栈顶值为 0 跳转(JZ)，栈顶值为 1 跳转(JNZ)，外部输入(IN)，输出外部(OUT)，操作(OP)；为完成结构简单但较大范围的取值，支持的数据类型仅为 16 位字。该 CPU 使用到 3 各寄存器：负责指令计数定位的 pc0 寄存器；负责存储当前执行指令的 ir0 寄存器；记录输出结果的 obuf0 寄存器。下见表格：

助记符	15	14	13	12	11-0	十六进制	功能描述
HALT	0	0	0	0	X（任意）	0XXX	终止处理器操作

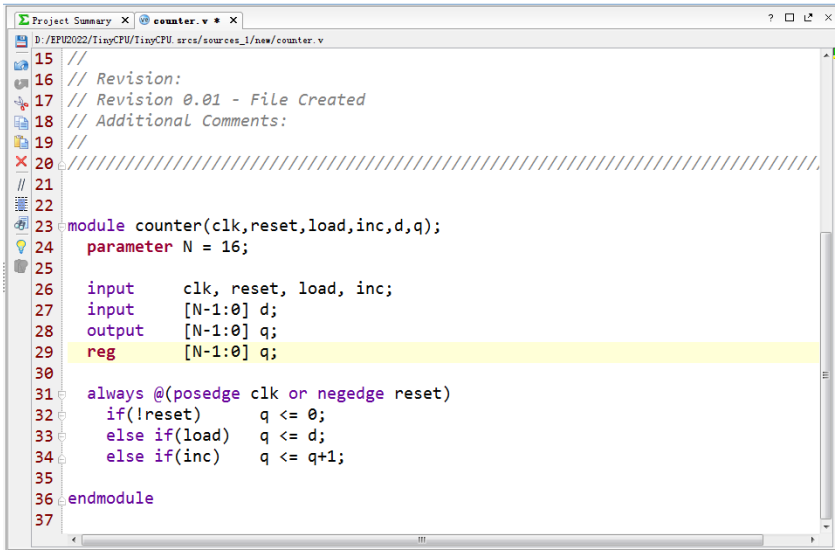
PUSH I	0	0	0	1	A(无 符号数)	1000+I	把立即数压入堆栈
PUSH A	0	0	1	0	A	2000+A	检索 ram0 中存储位置 A 的内容并把它压入
POP A	0	0	1	1	A	3000+A	从栈顶弹出数据并把它保存在 RAM 地址 A
JMP A	0	1	0	0	A	4000+A	总是跳转到地址 A 执行下一条指令；
JZ A	0	1	0	1	A	5000+A	当栈顶弹出的数据项是 0 时，跳转到地址 A
JNZ A	0	1	1	0	A	6000+A	当栈顶弹出的数据非 0 时，跳转到地址 A，
IN	1	1	0	1	X	D000	读取输入端口并把值压入堆栈
OUT	1	1	1	0	X	E000	弹出栈顶值并把它锁存到输出缓冲中
OP f	1	1	1	1	X f	F000+f	一类指令，使 ALU 执行 f 中的编码所要求的

助记符	4	3	2	1	0	十六进制	栈顶内容
ADD	0	0	0	0	0	F000	next+top
SUB	0	0	0	0	1	F001	next-top
MUL	0	0	0	1	0	F002	next*top
SHL	0	0	0	1	1	F003	next>>top
SHR	0	0	1	0	0	F004	next<<top
BAND	0	0	1	0	1	F005	next&top
BOR	0	0	1	1	0	F006	next|top
BXOR	0	0	1	1	1	F007	next^top

AND	0	1	0	0	0	F008	next&&top
OR	0	1	0	0	1	F009	next||top
EQ	0	1	0	1	0	F00A	next==top
NE	0	1	0	1	1	F00B	next!=top
GE	0	1	1	0	0	F00C	next>=top
LE	0	1	1	0	1	F00D	next<=top
GT	0	1	1	1	0	F00E	next>top
LT	0	1	1	1	1	F00F	next<top
NEG	1	0	0	0	0	F010	-top
BNOT	1	0	0	0	1	F011	~top
NOT	1	0	0	1	0	F012	！top

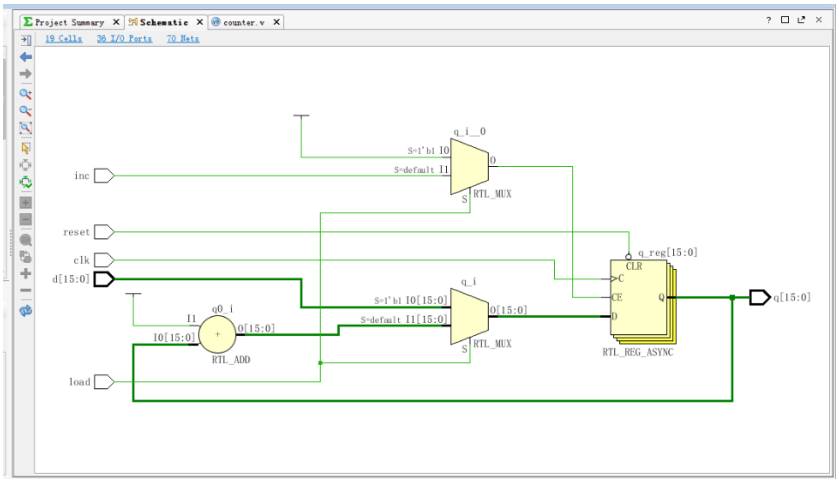
4. CPU 实现

4.1 Counter 模块

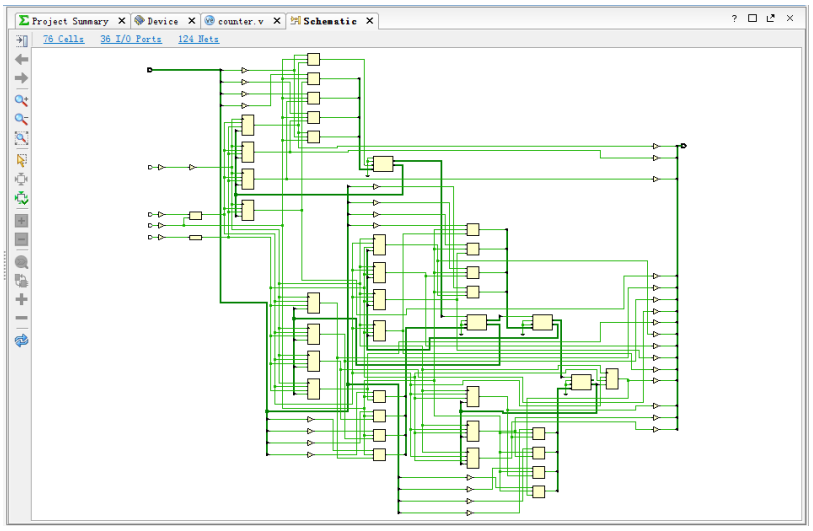


```
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21
22
23 module counter(clk,reset,load,inc,d,q);
24   parameter N = 16;
25
26   input      clk, reset, load, inc;
27   input      [N-1:0] d;
28   output     [N-1:0] q;
29   reg        [N-1:0] q;
30
31   always @(posedge clk or negedge reset)
32     if(!reset)      q <= 0;
33     else if(load)   q <= d;
34     else if(inc)    q <= q+1;
35
36 endmodule
37
```

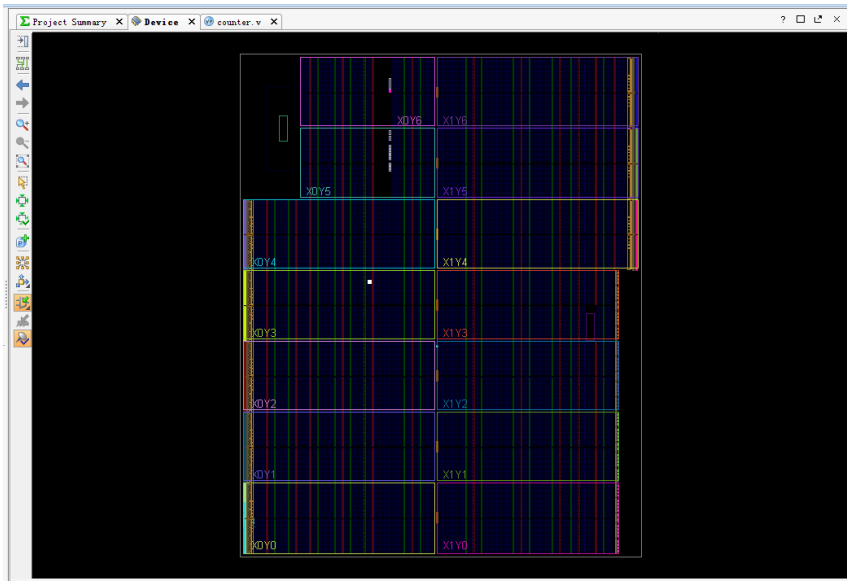
Counter 模块实现所用 Verilog HDL 描述



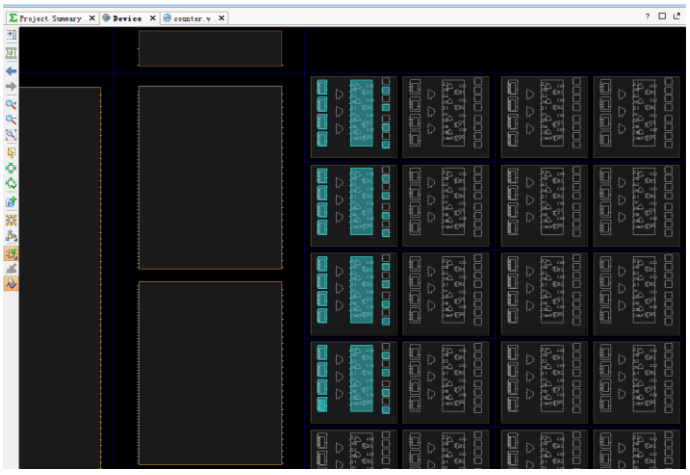
根据上述代码得到的 RTL 级连接结构



综合设计后的网表结构和内部映射关系



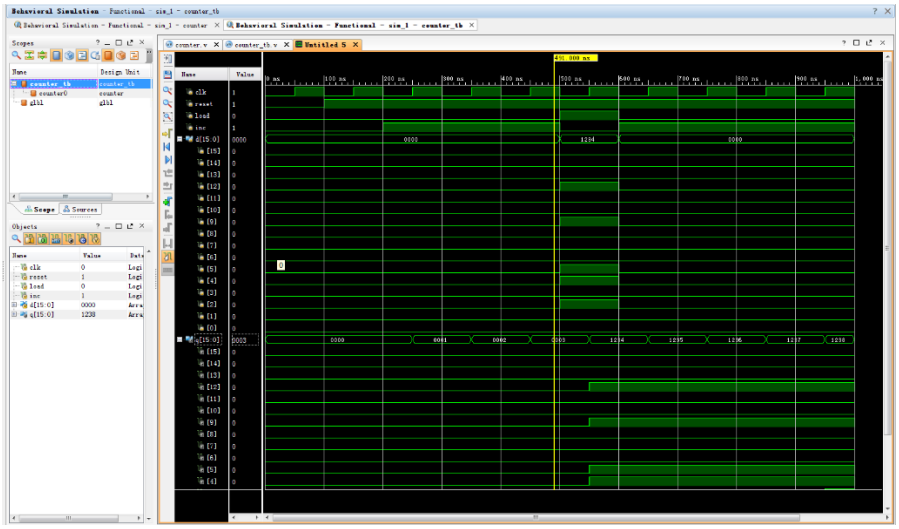
器件设计实现的结构图



结构图部分细节

```
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

`timescale 1ns/1ps

module counter_tb;
  reg   clk, reset, load, inc;
  reg   [15:0] d;
  wire  [15:0] q;
  counter counter0(.clk(clk), .reset(reset), .load(load), .inc(inc), .d(d), .q(q));

  initial begin
    clk = 0;
    forever
      #50 clk = ~clk;
  end

  initial begin
    reset = 0; load = 0; inc = 0; d = 16'h0000;
    #100  reset = 1;
    #100  inc = 1;
    #300  inc = 0; load = 1; d = 16'h1234;
    #100  inc = 1; load = 0; d = 16'h0000;
    #500  reset = 0;
  end
endmodule
```

仿真测试所用代码



行为级仿真和测试运行结果

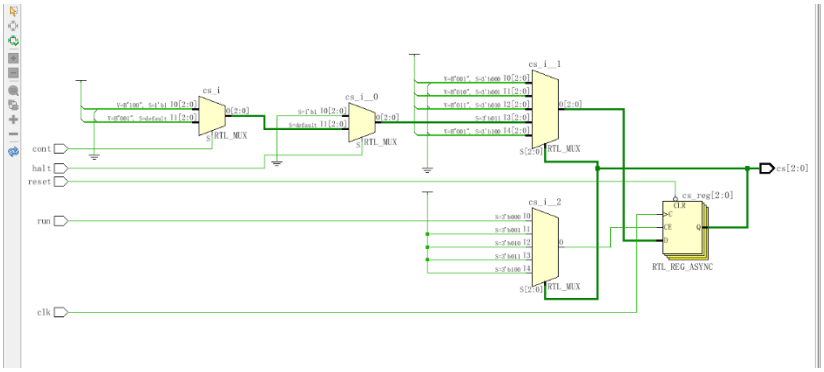
4.2 state 模块

```
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
`include "defs.v"

module state(clk, reset, run, cont, halt, cs);

    input   clk, reset, run, cont, halt;
    output  [2:0] cs;
    reg     [2:0] cs;

    always @(posedge clk or negedge reset)
        if(!reset)                  cs <= `IDLE;    //manual reset
        else
            case(cs)
                `IDLE:  if(run)     cs <= `FETCHA;
                `FETCHA:            cs <= `FETCHB;
                `FETCHB:            cs <= `EXECA;
                `EXECA: if(halt)    cs <= `IDLE;
                   else if(cont)    cs <= `EXECB;
                   else             cs <= `FETCHA;
                `EXECB:             cs <= `FETCHA;
                default:            cs <= 3'bxxx;
            endcase
endmodule
```

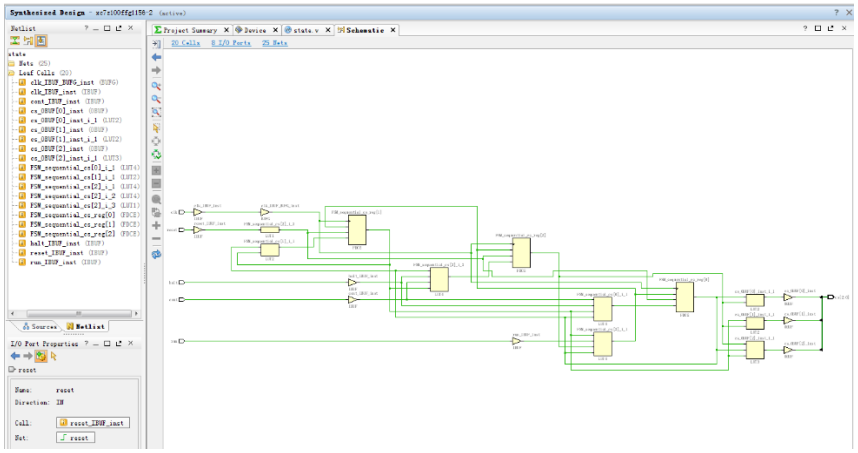
State 模块实现所用 Verilog HDL 描述

```
`define IDLE    3'b000      //start
`define FETCHA  3'b001      //fetch-start
`define FETCHB  3'b010      //fetch-end
`define EXECA   3'b011      //excute-command
`define EXECB   3'b100      //excute=last
```

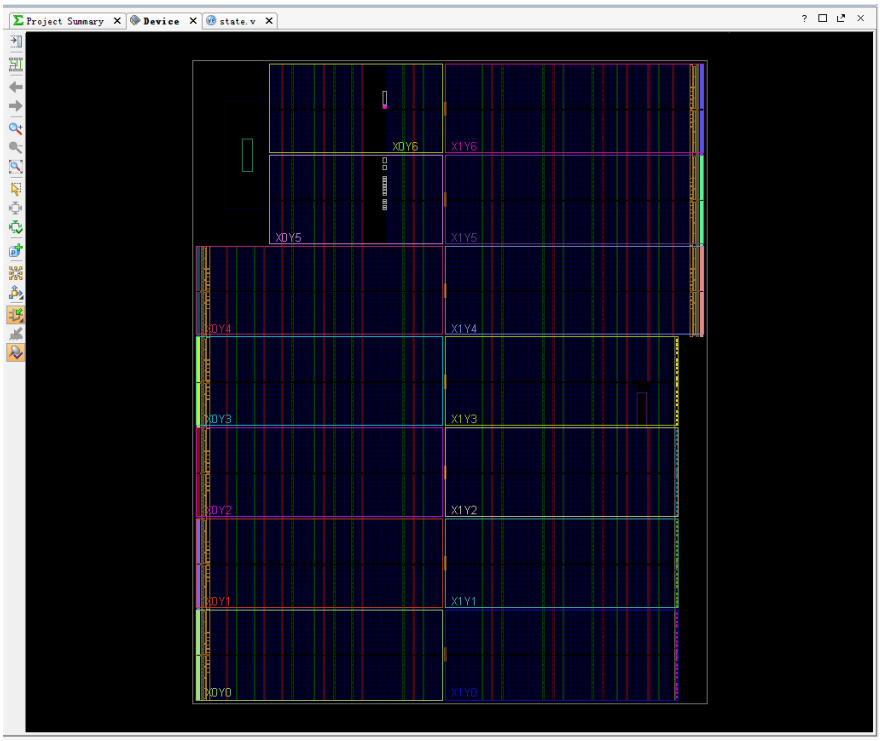
State 模块所用助记码定义及文件目录结构



根据上述代码得到的 RTL 级连接结构



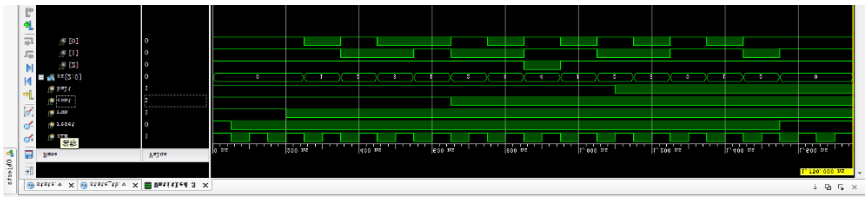
综合设计后的网表结构和内部映射关系



器件设计实现的结构图

```
22
23  module state_tb;
24      reg    clk, reset, run, cont, halt;
25      wire   [2:0] cs;
26      state state0(.clk(clk), .reset(reset), .run(run), .cont(cont), .halt(halt),
27
28      initial begin
29          clk = 0;
30          forever
31              #50 clk = ~clk;
32      end
33
34      initial begin
35          reset = 0; run = 0; cont = 0; halt = 0;
36          #50  reset = 1;
37          #150  run = 1;
38          #450  cont = 1;
39          #450  halt = 1;
40          #450  reset = 0;
41      end
42  endmodule
43
```

仿真测试所用代码

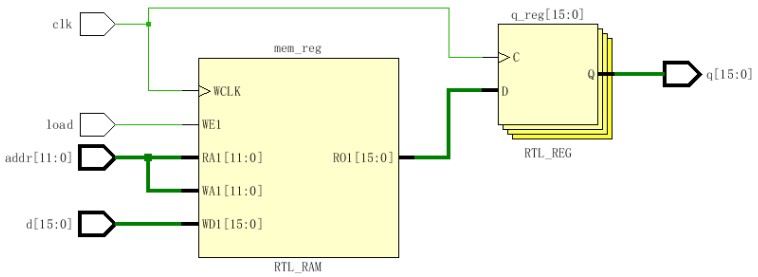


行为级仿真和测试运行结果

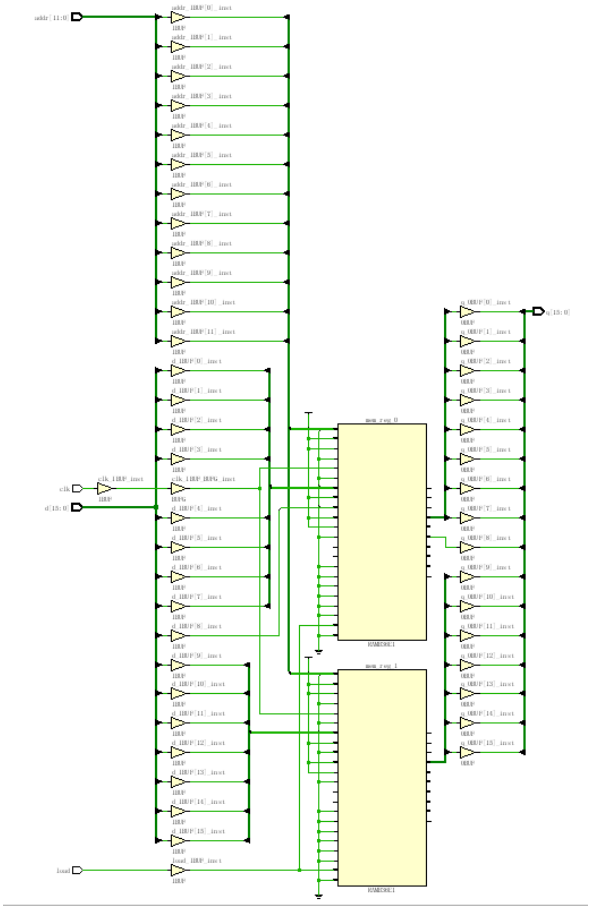
4.3 ram 模块

```
23  module ram(clk ,load, addr, d, q);
24      parameter DWIDTH=16, AWIDTH=12, WORDS=4096;
25
26      input   clk, load;
27      input   [AWIDTH-1:0] addr;
28      input   [DWIDTH-1:0] d;
29      output  [DWIDTH-1:0] q;
30      reg     [DWIDTH-1:0] q;
31      reg     [DWIDTH-1:0] mem [WORDS-1:0];
32
33      always @(posedge clk)
34      begin
35          if(load)    mem[addr] <= d;     //write
36          q <= mem[addr];                 //output
37      end
38
39      integer i;
40      initial begin                       //set initial value
41          for(i = 0; i < WORDS; i = i+1)
42              mem[i] = 0;
43          mem[12'h000] = 16'hF000;
44          mem[12'h001] = 16'hF001;
45          mem[12'h002] = 16'hF002;
46          mem[12'h003] = 16'hF003;
47          mem[12'h004] = 16'hF004;
48          mem[12'h005] = 16'hF005;
49          mem[12'h006] = 16'hF006;
50          mem[12'h007] = 16'hF007;
51          mem[12'h008] = 16'hF008;
52          mem[12'h009] = 16'hF009;
53          mem[12'h00A] = 16'hF00A;
54          mem[12'h00B] = 16'hF00B;
55          mem[12'h00C] = 16'hF00C;
56          mem[12'h00D] = 16'hF00D;
57          mem[12'h00E] = 16'hF00E;
58          mem[12'h00F] = 16'hF00F;
59          mem[12'h010] = 16'hF010;
60      end
61  endmodule
62
```

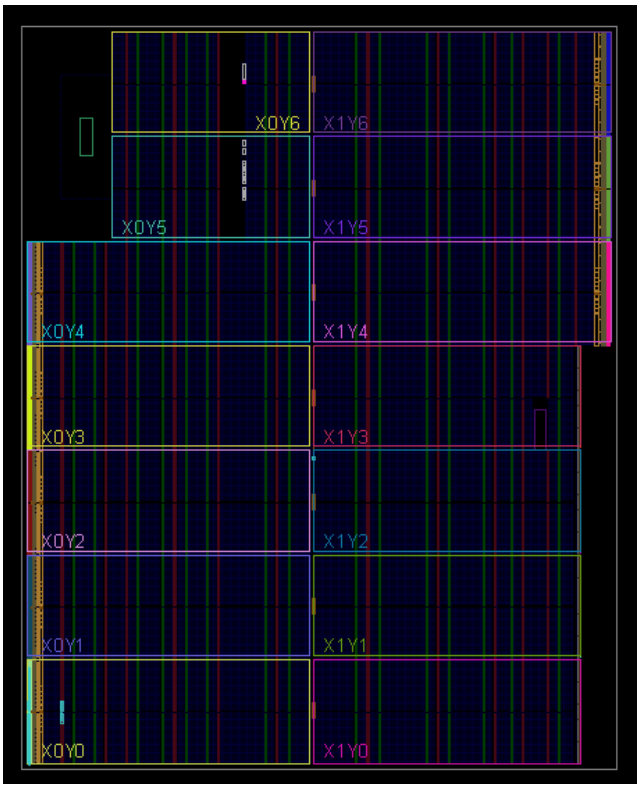
ram 模块实现所用 Verilog HDL 描述



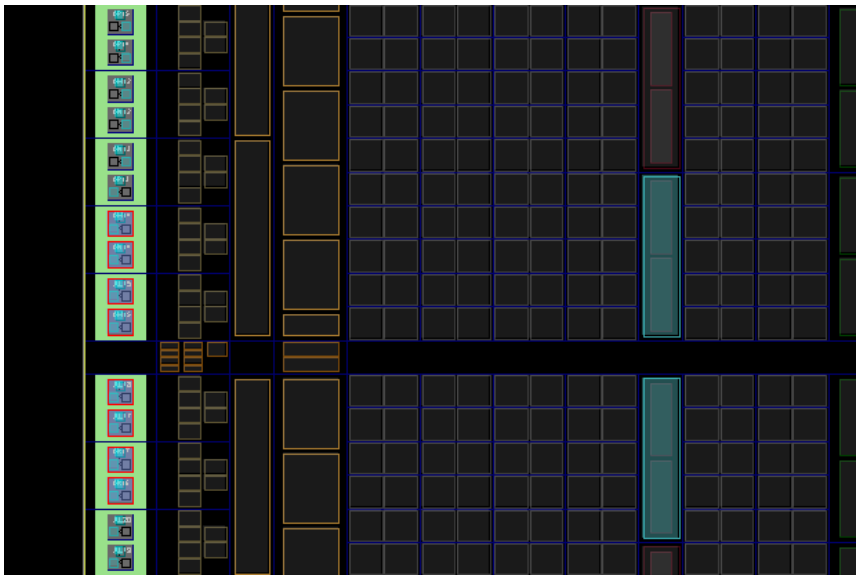
根据上述代码得到的 RTL 级连接结构



综合设计后的网表结构和内部映射关系



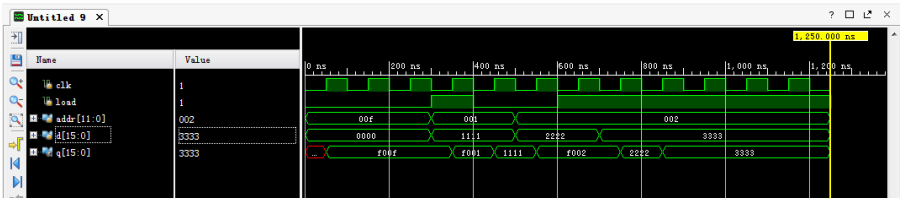
器件设计实现的结构图



结构图部分细节

```
module ram_tb;
    reg   clk,load;
    reg   [11:0] addr;
    reg   [15:0] d;
    wire  [15:0] q;
    ram ram0(.clk(clk), .load(load), .addr(addr), .d(d), .q(q));

    initial begin
        clk = 0;
        forever
            #50 clk=~clk;
    end

    initial begin
            load = 0;   addr = 12'h00F;     d = 16'h0000;
        #100
        #200 load = 1;   addr = 12'h001;     d = 16'h1111;
        #100 load = 0;
        #100             addr = 12'h002;     d = 16'h2222;
        #100 load = 1;
        #100 load = 1;   addr = 12'h002;     d = 16'h3333;
    end
endmodule
```

仿真测试所用代码

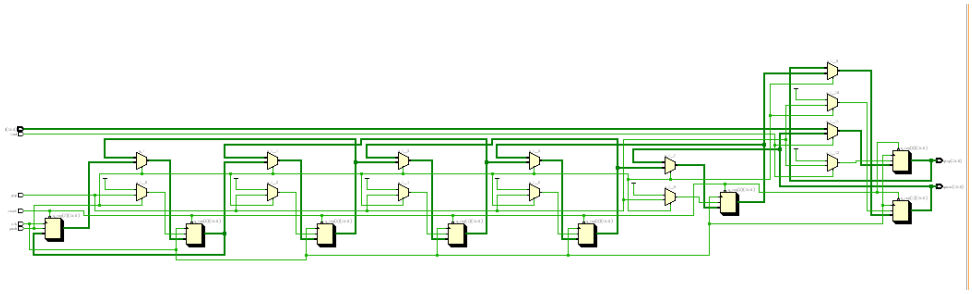


行为级仿真和测试运行结果

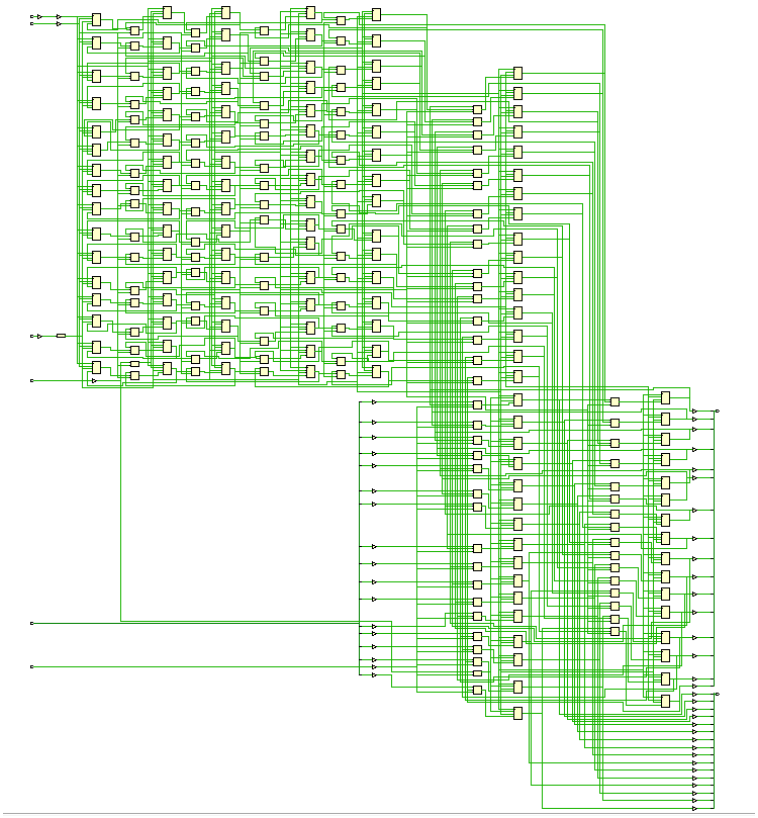
4.4 stack 模块

```
23  module stack(clk, reset, load, push, pop, d, qtop, qnext);
24      parameter N = 8;                            //stack size
25
26      input   clk, reset, load, push, pop;
27      input   [15:0] d;
28      output  [15:0] qtop, qnext;
29      reg     [15:0] q [0:N-1];
30
31      assign qtop = q[0];                         //stack top
32      assign qnext = q[1];                        //stack second
33
34      always @(posedge clk or negedge reset)  //top data read&write
35          if(!reset)      q[0] <= 0;              //clear
36          else if(load)   q[0] <= d;              //load data
37          else if(pop)    q[0] <= q[1];           //pop_front  move data
38
39      integer i;
40      always @(posedge clk or negedge reset)  //internal process
41          for(i = 1; i < N-1; i = i + 1)
42              if(!reset)      q[i] <= 0;          //clear all data
43              else if(push)   q[i] <= q[i-1];     //push_front
44              else if(pop)    q[i] <= q[i+1];     //pop_front
45
46      always @(posedge clk or negedge reset)  //edge process
47          if(!reset)      q[N-1] <= 0;            //clear
48          else if(push)   q[N-1] <= q[N-2];       //push_front
49          //else if(pop)    q[N-1] <= 0;            //*pop_front
50  endmodule
51
```

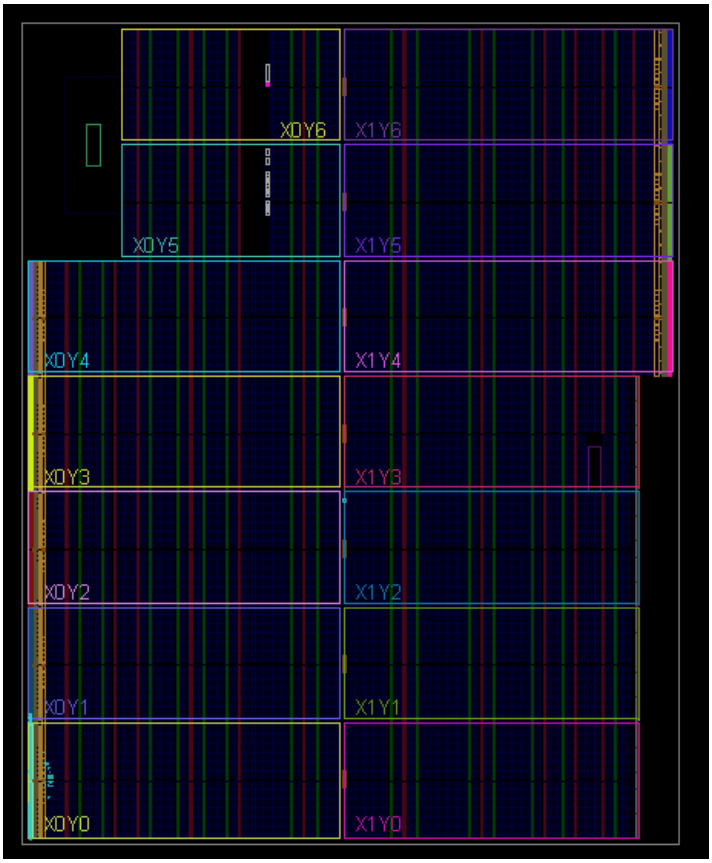
stack 模块实现所用 Verilog HDL 描述



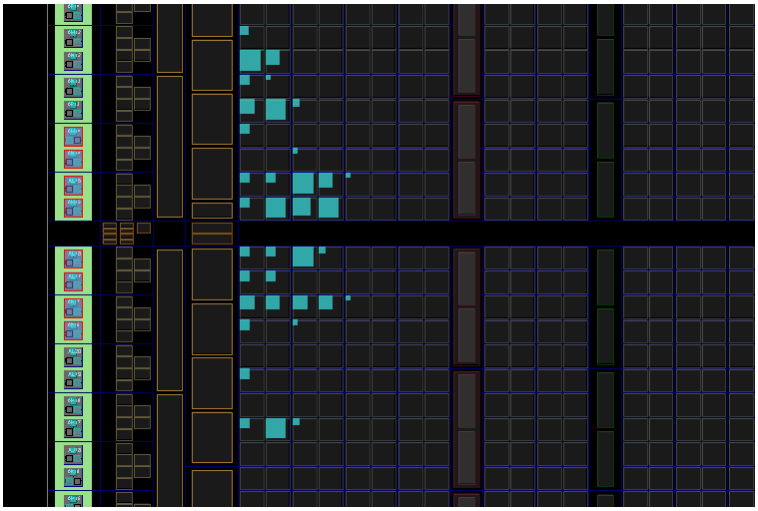
根据上述代码得到的 RTL 级连接结构



综合设计后的网表结构和内部映射关系



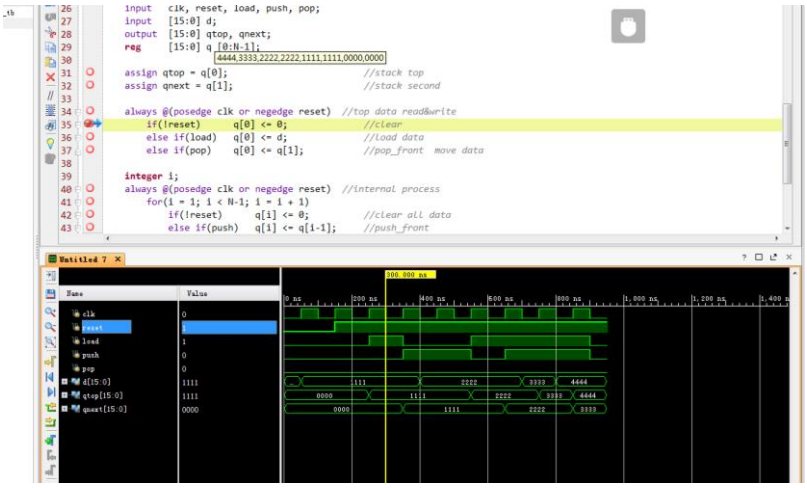
器件设计实现的结构图



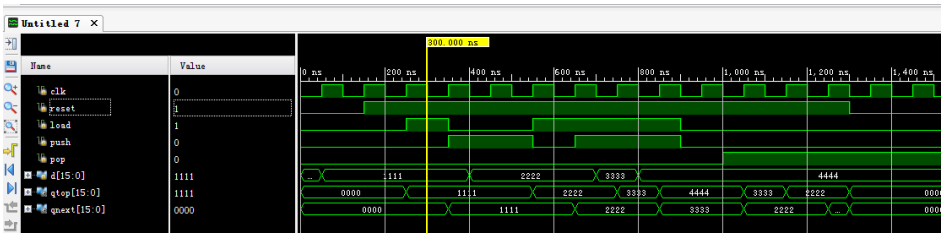
结构图部分细节

```
module stack_tb;
    reg     clk, reset, load, push, pop;
    reg     [15:0]  d;
    wire    [15:0] qtop, qnext;
    stack stack0(.clk(clk), .reset(reset), .load(load), .push(push), .pop(pop), .d(d), .qtop(qtop), .qnext(qnext));

    initial begin
        clk = 0;
        forever
            #50 clk = ~clk;
    end

    initial begin
              reset = 0;    load = 0;   push = 0;   pop = 0;
              d = 16'h0000;

        #50   d = 16'h1111;
        #100  reset = 1;
        #100                load = 1;
        #100                load = 0;   push = 1;
        #50   d = 16'h2222;
        #150                load = 1;   push = 0;
        #100                load = 1;   push = 1;
        #50   d = 16'h3333;
        #100  d = 16'h4444;
        #100                load = 0;   push = 0;
        #100                                    pop = 1;
        #300  reset = 0;
    end

endmodule
```

仿真测试所用代码



仿真过程中时刻元件运行情况



行为级仿真和测试运行结果

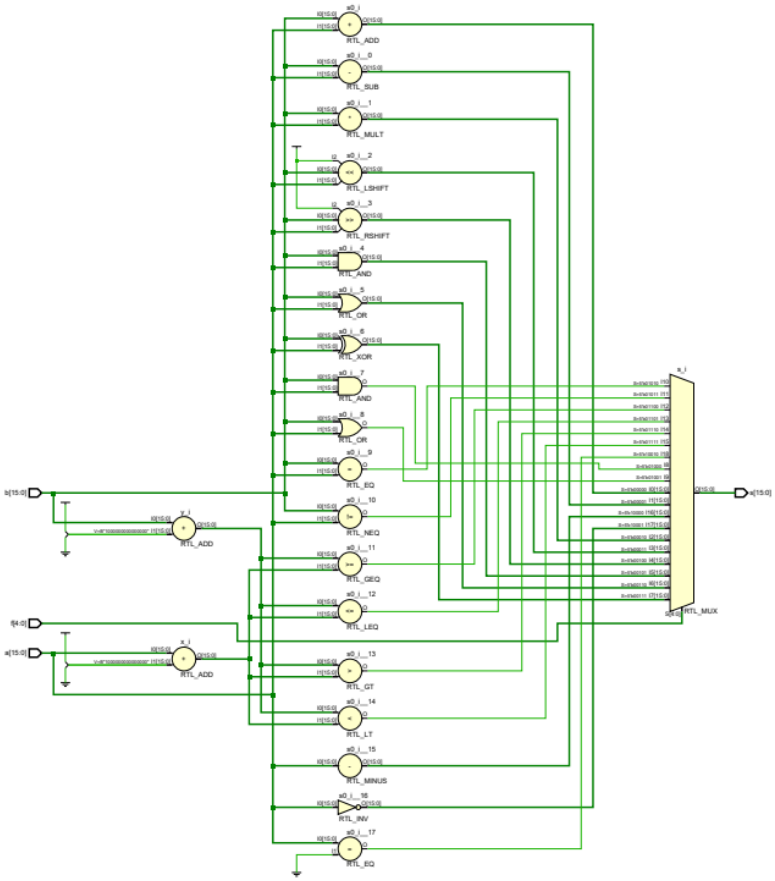
4.5 alu 模块

```
23  `include "defs.v"
24
25  module alu(a, b, f, s);
26
27      input   [15:0] a, b;
28      input   [ 4:0] f;
29      output  [15:0] s;
30      reg     [15:0] s;
31      wire    [15:0] x, y;
32
33      assign x = a + 16'h8000;
34      assign y = b + 16'h8000;
35
36      always @(a or b or x or y or f)
37          case(f)
38              `ADD :  s = b + a;
39              `SUB :  s = b - a;
40              `MUL :  s = b * a;
41              `SHL :  s = b << a;
42              `SHR :  s = b >> a;
43              `BAND : s = b & a;
44              `BOR :  s = b | a;
45              `BXOR : s = b ^ a;
46              `AND :  s = b && a;
47              `OR :   s = b || a;
48              `EQ :   s = b == a;
49              `NE :   s = b != a;
50              `GE :   s = y >= x;
51              `LE :   s = y <= x;
52              `GT :   s = y > x;
53              `LT :   s = y < x;
54              `NEG :  s = -a;
55              `BNOT : s = ~a;
56              `NOT :  s = !a;
57              default : s = 16'hxxxx;
58      endcase
59  endmodule
```

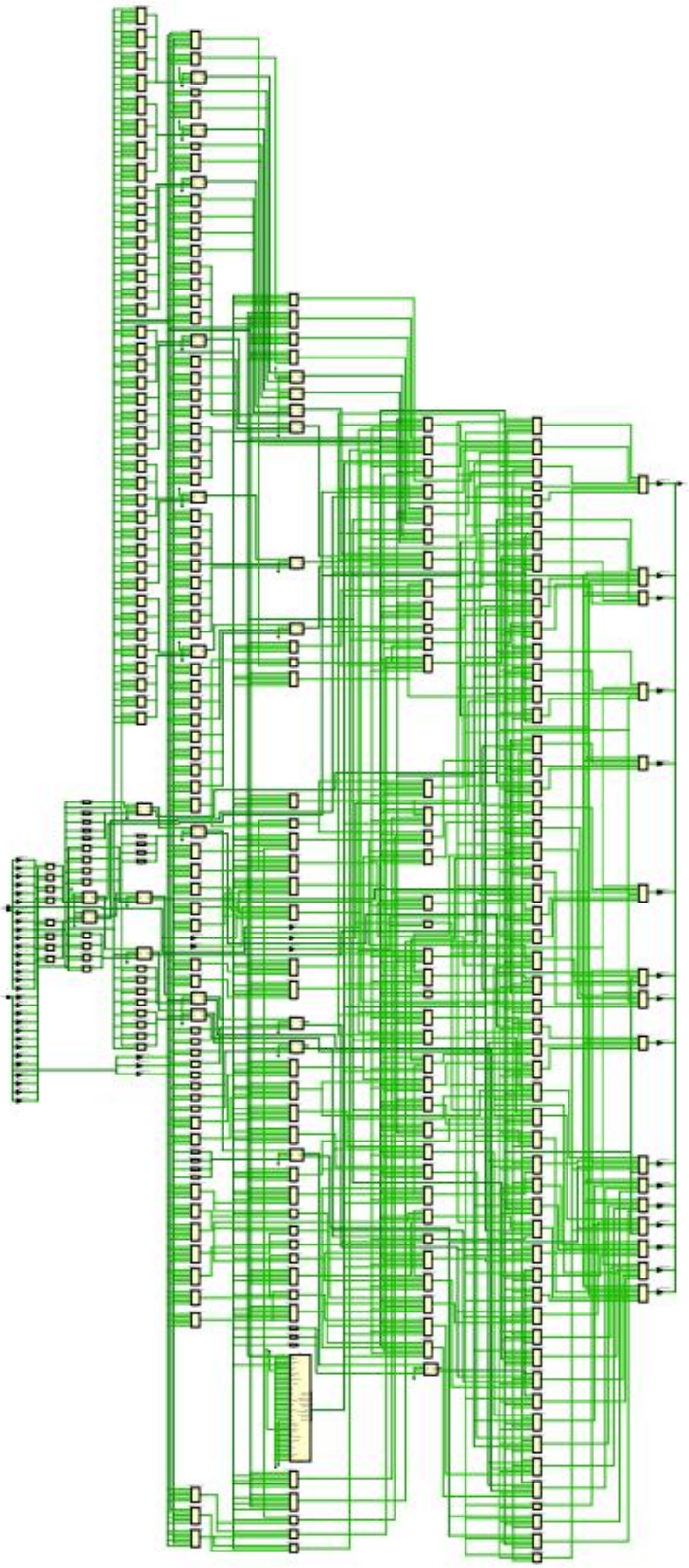
alu 模块实现所用 Verilog HDL 描述

```
// detailed OP function
`define ADD     5'b00000    // 16'hF000
`define SUB     5'b00001    // 16'hF001
`define MUL     5'b00010    // 16'hF002
`define SHL     5'b00011    // 16'hF003
`define SHR     5'b00100    // 16'hF004
`define BAND    5'b00101    // 16'hF005
`define BOR     5'b00110    // 16'hF006
`define BXOR    5'b00111    // 16'hF007
`define AND     5'b01000    // 16'hF008
`define OR      5'b01001    // 16'hF009
`define EQ      5'b01010    // 16'hF00A
`define NE      5'b01011    // 16'hF00B
`define GE      5'b01100    // 16'hF00C
`define LE      5'b01101    // 16'hF00D
`define GT      5'b01110    // 16'hF00E
`define LT      5'b01111    // 16'hF00F
`define NEG     5'b10000    // 16'hF010
`define BNOT    5'b10001    // 16'hF011
`define NOT     5'b10010    // 16'hF012
```

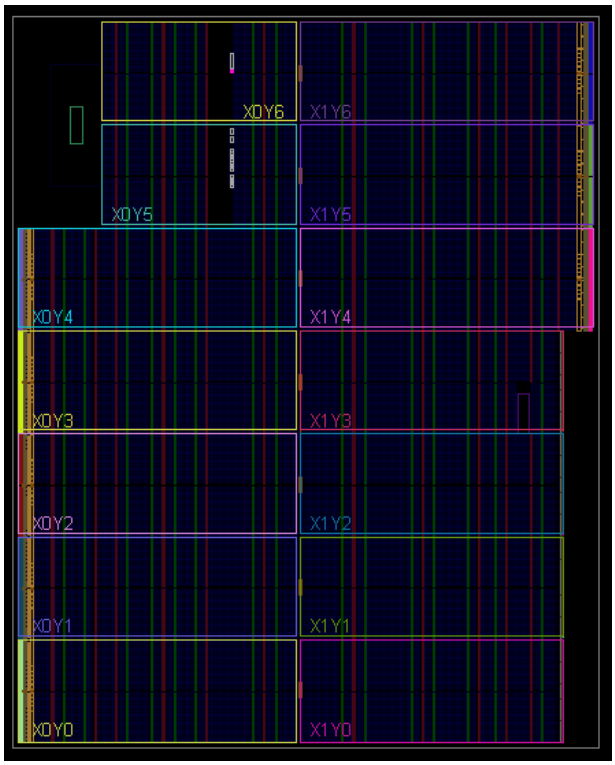
alu 模块所使用助记符的定义



根据上述代码得到的 RTL 级连接结构



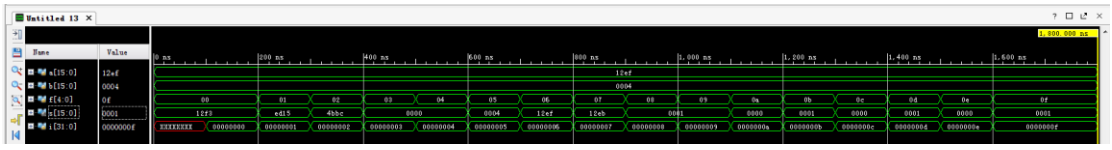
综合设计后的网表结构和内部映射关系



器件设计实现的结构图

```
22
23  module alu_tb;
24      reg     [15:0]  a,b;
25      reg     [ 4:0]  f;
26      wire    [15:0]  s;
27      integer i;
28      alu alu0(.a(a), .b(b), .f(f), .s(s));
29
30      initial begin
31          a = 16'h12EF;   b = 16'h0004;
32          f =  5'b0000;
33
34          # 100
35          for(i = 0; i < 15; i = i + 1)
36              #100    f <= f + 1'b1;
37      end
38  endmodule
```

仿真测试所用代码



行为级仿真和测试运行结果

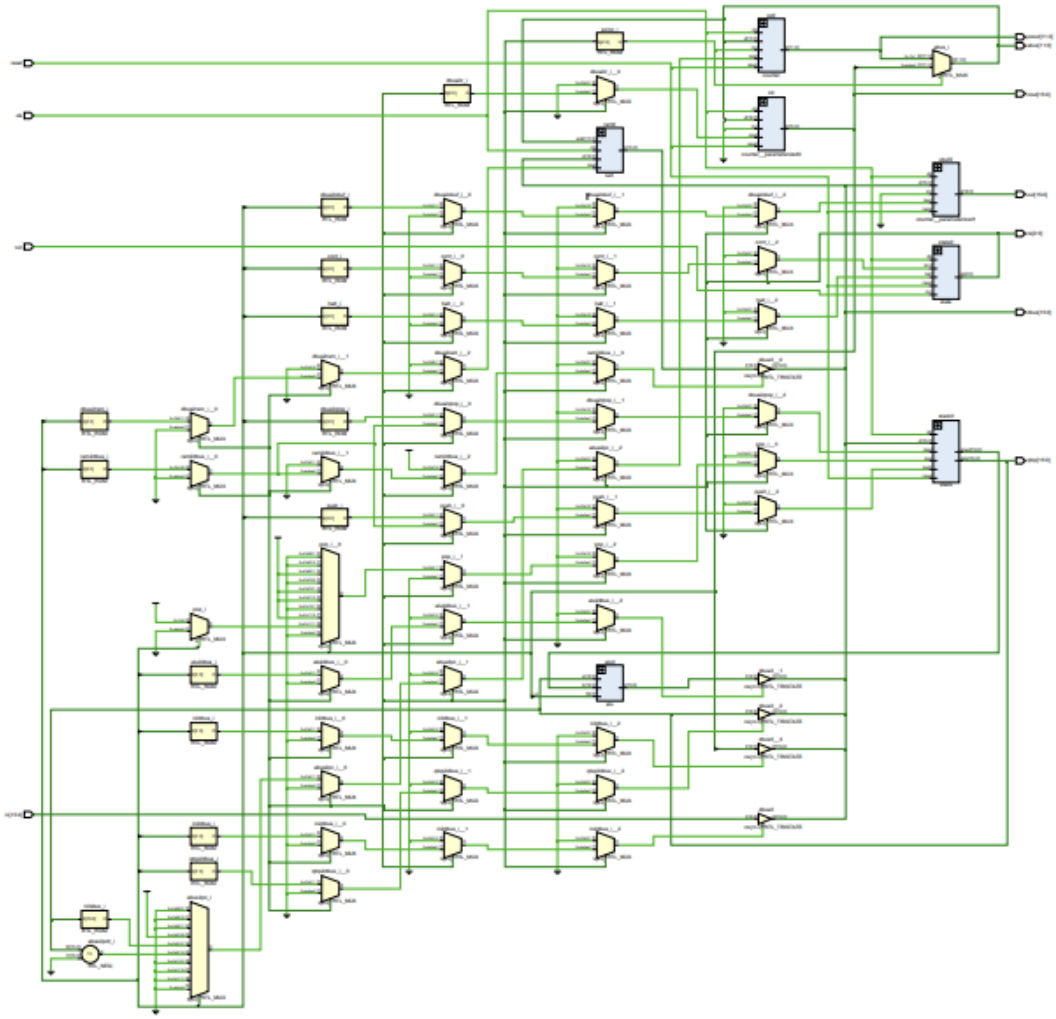
4.6综合测试设计 tinycpu

```
23  `include "defs.v"
24
25  module tinycpu(clk, reset, run, in, cs, pcout, irout, qtop, abus, dbus, out);
26
27      input   clk, reset, run;
28      input   [15:0] in;
29      output  [ 2:0] cs;
30      output  [15:0] irout, qtop, dbus, out;
31      output  [11:0] pcout, abus;
32      wire    [15:0] qnext, ramout, aluout;
33      reg     [11:0] abus;
34      reg     halt,       cont,       pcinc,      push,       pop,
35              abus2pc,    dbus2ir,    dbus2qtop,  dbus2ram,   dbus2obuf,  pc2abus,
36              ir2abus,    ir2dbus,    qtop2dbus,  alu2dbus,   ram2dbus,   in2dbus;
37
38      counter #(12) pc0(
39          .clk(clk),      .reset(reset),  .load(abus2pc),
40          .inc(pcinc),    .d(abus),       .q(pcout));
41      counter #(16) ir0(
42          .clk(clk),      .reset(reset),  .load(dbus2ir),
43          .inc(0),        .d(dbus),       .q(irout));
44      state state0(
45          .clk(clk),      .reset(reset),  .run(run),
46          .cont(cont),    .halt(halt),    .cs(cs));
47      stack stack0(
48          .clk(clk),      .reset(reset),  .load(dbus2qtop),
49          .push(push),    .pop(pop),      .d(dbus),
50          .qtop(qtop),    .qnext(qnext));
51      alu alu0(
52          .a(qtop),       .b(qnext),      .f(irout[4:0]),
53          .s(aluout));
54      ram #(16, 12, 4096) ram0(
55          .clk(clk),      .load(dbus2ram), .addr(abus[11:0]),
56          .d(dbus),       .q(ramout));
57      counter #(16) obuf0(
58          .clk(clk),      .reset(reset),  .load(dbus2obuf),
59          .inc(0),        .d(dbus),       .q(out));
60
61      // Bus Operation logic
62      always @(pc2abus or ir2abus or pcout or irout)
63          if(pc2abus)         abus <= pcout;          // PC -> ABus
64          else if(ir2abus)    abus <= irout[11:0];    // IR -> ABus
65          else                abus <= 12'hxxx;
66
67      assign dbus = ir2dbus   ? {{4{irout[11]}}, irout[11:0]} : 16'hzzzz; // up 4 bits extension
68      assign dbus = qtop2dbus ? qtop   : 16'hzzzz;
69      assign dbus = alu2dbus  ? aluout : 16'hzzzz;
70      assign dbus = ram2dbus  ? ramout : 16'hzzzz;
71      assign dbus = in2dbus   ? in     : 16'hzzzz;
```

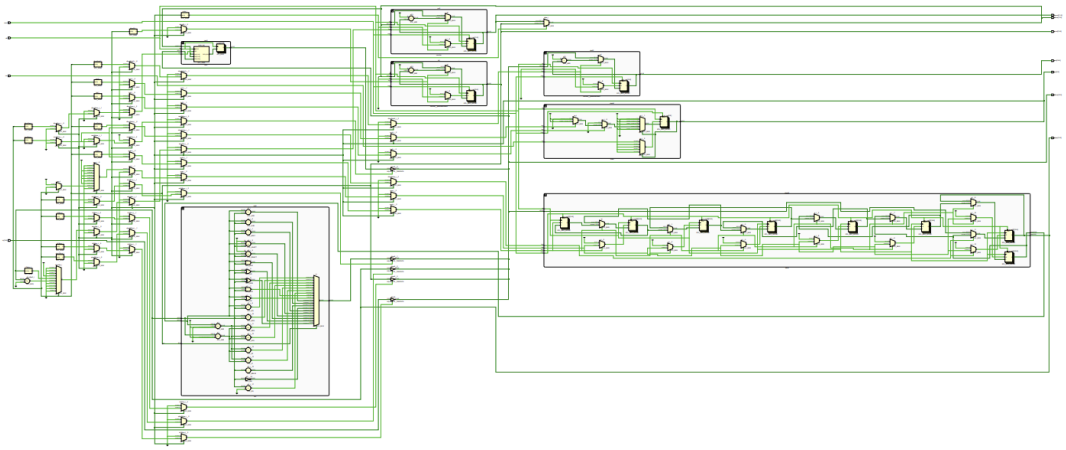
```
73      always @(cs or irout or qtop)
74      begin
75          halt = 0;       pcinc = 0;      push = 0;       pop = 0;    cont = 0;
76          abus2pc = 0;    dbus2ir = 0;    dbus2qtop = 0;  dbus2ram = 0;   dbus2obuf = 0;  pc2abus = 0;
77          ir2abus = 0;    ir2dbus = 0;    qtop2dbus = 0;  alu2dbus = 0;   ram2dbus = 0;   in2dbus = 0;
78
79          if(cs == `FETCHA)       // command in PC output to ABus
80          begin
81              pcinc = 1;
82              pc2abus = 1;
83          end
84          else if(cs == `FETCHB)
85          begin
86              ram2dbus = 1;       // RAM <- ABus  RAM -> DBus
87              dbus2ir = 1;        // DBus -> IR
88          end
89          else if(cs == `EXECA)
90              case(irout[15:12])  // Choose command to be executed
91                  `PUSHI:             // push integer into stack
92                  begin
93                      ir2dbus = 1;    // IR -> DBus
94                      dbus2qtop = 1;  // DBus -> StackTop
95                      push = 1;       // stack push
96                  end
97                  `PUSH:              // push into stack from address
98                  begin
99                      ir2abus = 1;    // IR -> ABus
100                     cont = 1;       // extra execute
101                 end
102                 `POP:               // pop from stack
103                 begin
104                     ir2abus = 1;    // IR -> ABus
105                     qtop2dbus = 1;  // StackTop -> DBus
106                     dbus2ram = 1;   // DBus -> RAM
107                     pop = 1;        // stack pop
108                 end
109                 `JMP:               // jump to address
110                 begin
111                     ir2abus = 1;
112                     abus2pc = 1;
113                 end
114                 `JZ:                // jump if false
115                 begin
116                     if(qtop==0)
117                     begin
118                         ir2abus = 1;
119                         abus2pc = 1;
120                     end
121                     pop = 1;
122                 end
123                 `JNZ:               // jump if true
124                 begin
125                     if(qtop != 0)
126                     begin
127                         ir2abus = 1;
128                         abus2pc = 1;
129                     end
130                     pop = 1;
131                 end
132                 `IN:                // load from IN port
133                 begin
134                     in2dbus = 1;
135                     dbus2qtop = 1;
136                     push = 1;
137                 end
138                 `OUT:               // output to OUT port
139                 begin
140                     qtop2dbus = 1;
141                     dbus2obuf = 1;
142                     pop = 1;
143                 end
```

```
144                 `OP:                // calculate
145                 begin
146                     alu2dbus = 1;   // ALU -> DBus
147                     dbus2qtop = 1;  // DBus -> StackTop
148                     if(irout[4]==0) pop = 1;    // mono/binary op stack pop
149                 end
150                 default:
151                     halt = 1;
152             endcase
153         else if(cs == `EXECB)
154         begin
155             if(irout[15:12]==`PUSH)
156             begin
157                 ram2dbus = 1;       // RAM -> DBus
158                 dbus2qtop = 1;      // DBus -> StackTop
159                 push = 1;           // stack push
160             end
161         end
162     end
163 endmodule
```

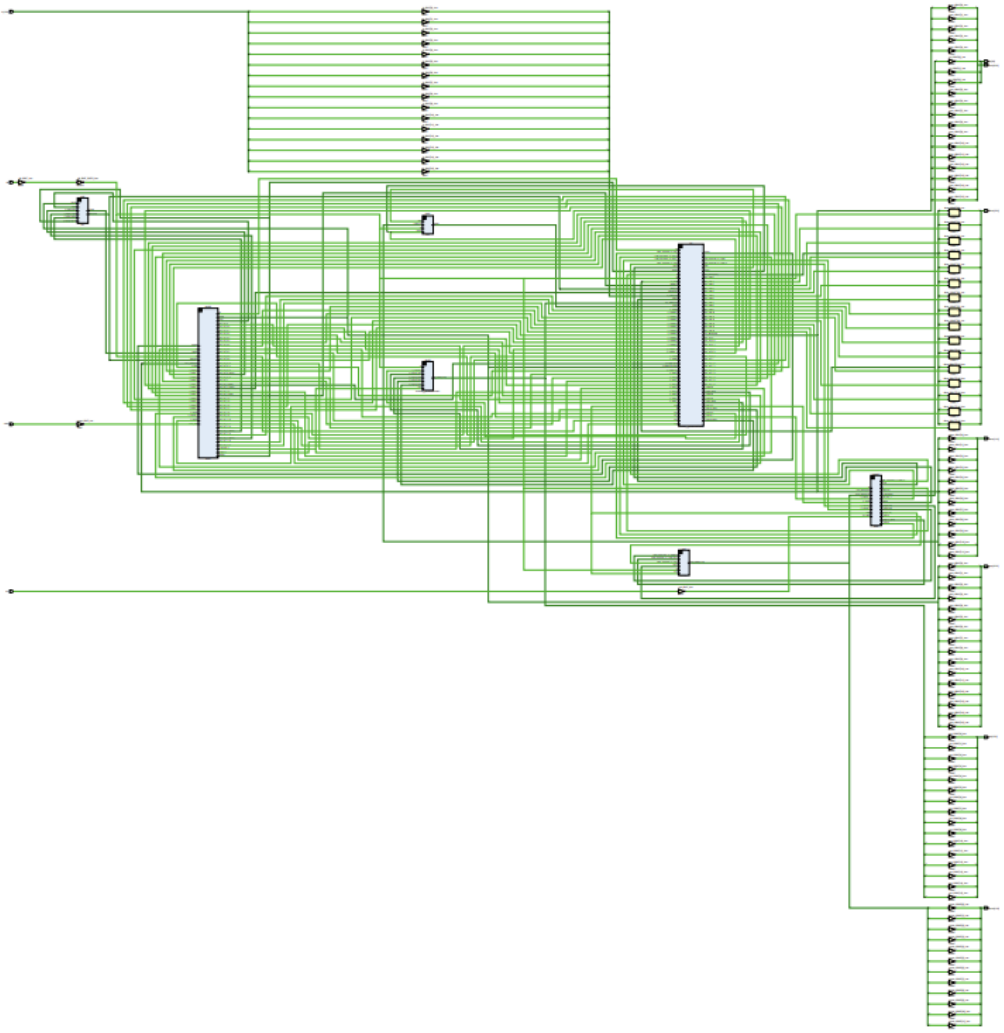
实现所用 Verilog HDL 描述



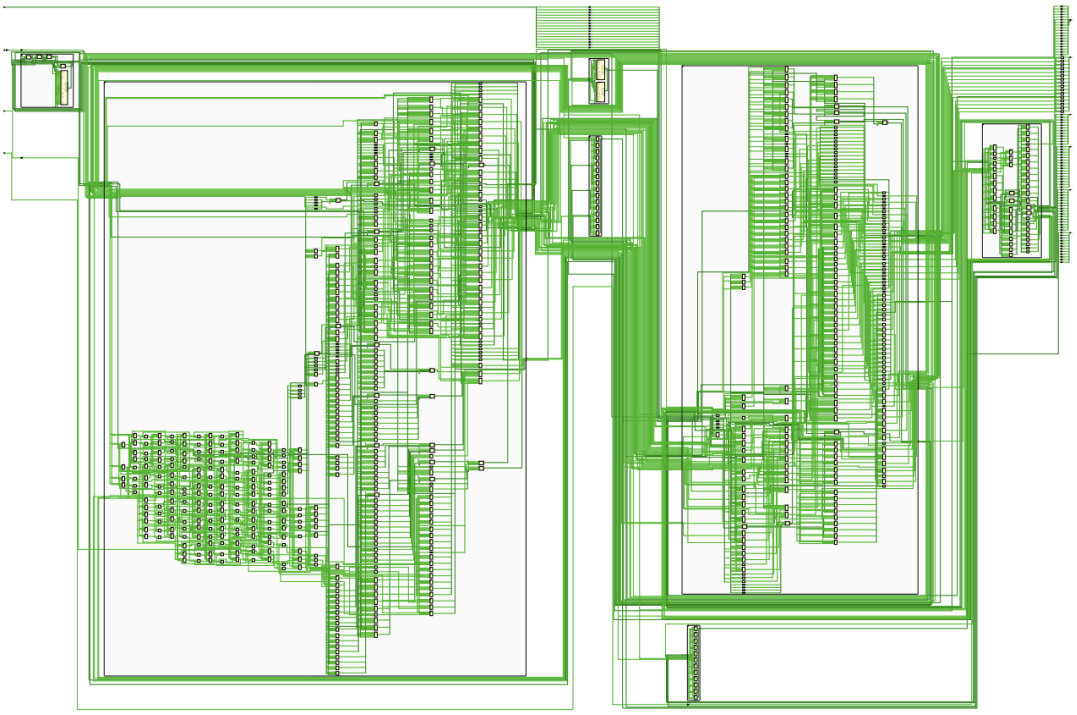
根据上述代码得到的 RTL 级连接结构



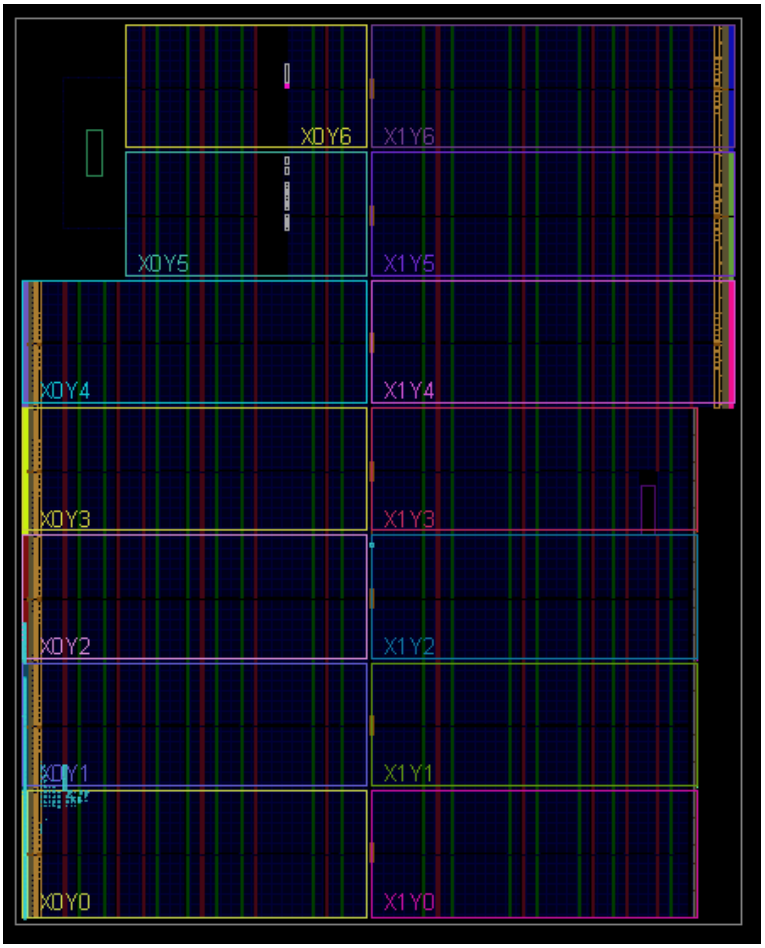
展开图



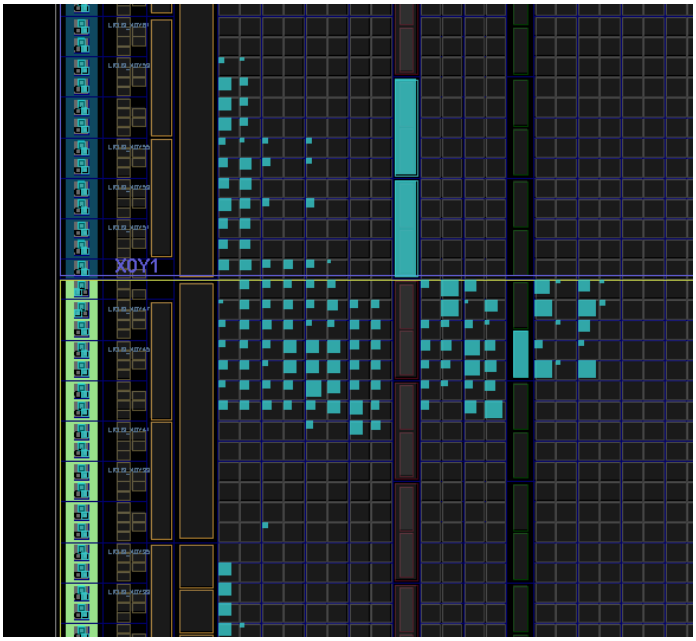
综合设计后的网表结构和内部映射关系



展开图



器件设计实现的结构图



结构图部分细节

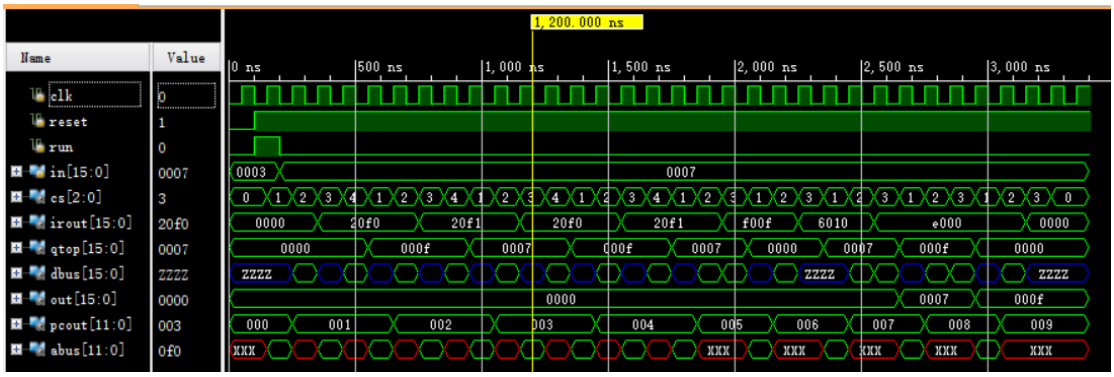
sim_1 > new > Tinycpu_tb_subtract.v

```
1   `timescale 1ns / 1ps
2   module tinycpu_tb;
3
4       reg     clk, reset, run;
5       reg     [15:0]  in;
6       wire    [2:0]   cs;
7       wire    [15:0]  irout, qtop, dbus, out;
8       wire    [11:0]  pcout, abus;
9
10  tinycpu tinycpu0(
11      .clk(clk),      .reset(reset),  .run(run),
12      .in(in),        .cs(cs),        .pcout(pcout),
13      .irout(irout),  .qtop(qtop),    .abus(abus),
14      .dbus(dbus),    .out(out));
15
16      initial begin
17          clk=0;
18          forever #50 clk = ~clk;
19      end
20
21      initial begin
22          $dumpfile("tinycpu.vcd");   // saving path
23          $dumpvars(0, tinycpu_tb);   // saving variable and depth
24          // where variable set
25          //      run=0;  in=3;   reset=0;
26          // #100     run=1;          reset=1;
27          // #100     run=0;  in=7;
28
29                  run=0;  reset=0;
30          #100    run=1;  reset=1;
31          #9000   run=0;
32
33          #9500 $finish;              // saving finish
34      end
35
36  endmodule
```

仿真测试所用代码

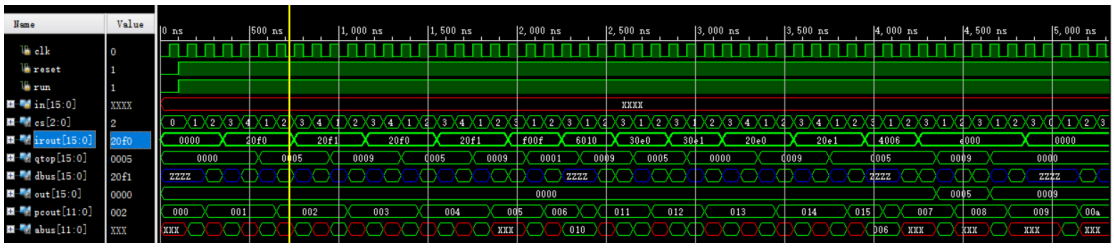
```
module ram(clk ,load, addr, d, q);
    parameter DWIDTH=16, AWIDTH=12, WORDS=4096;

    input   clk, load;
    input   [AWIDTH-1:0] addr;
    input   [DWIDTH-1:0] d;
    output  [DWIDTH-1:0] q;
    reg     [DWIDTH-1:0] q;
    reg     [DWIDTH-1:0] mem [WORDS-1:0];

    always @(posedge clk)
    begin
        if(load)    mem[addr] <= d;     //write
        q <= mem[addr];                 //output
    end

    integer i;
    initial begin                       //set initial value
        for(i = 0; i < WORDS; i = i+1)
            mem[i] = 0;
        // mem[12'h000] = 16'hD000;    // IN
        // mem[12'h001] = 16'h2005;    // PUSH from RAM ADDRESS 5
        // mem[12'h002] = 16'hF001;    // SUB
        // mem[12'h003] = 16'hE000;    // OUT
        // mem[12'h004] = 16'h0000;    // HALT
        // mem[12'h005] = 16'h0003;    // cnst:3

        // stack pop used var(s) after operation
        mem[12'h000] = 16'h20F0;    // load var1 from RAM for comparion
        mem[12'h001] = 16'h20F1;    // load var2 from RAM for comparion
        mem[12'h002] = 16'h20F0;    // load var1 from RAM for swap
        mem[12'h003] = 16'h20F1;    // load var2 from RAM for swap
        mem[12'h004] = 16'hF00F;    // OP-LT
        mem[12'h005] = 16'h6010;    // JNZ false then swap
        //mem[12'h006] = 16'h3000;    // no need to pop comparison result

        // output
        mem[12'h006] = 16'hE000;    // pop smaller
        mem[12'h007] = 16'hE000;    // pop bigger
        mem[12'h008] = 16'h0000;    // HALT

        // swap
        //mem[12'h010] = 16'h3000;    // no need to pop comparison result
        mem[12'h010] = 16'h30E0;    //  bigger temporarily save to RAM
        mem[12'h011] = 16'h30E1;    // smaller temporarily save to RAM
        mem[12'h012] = 16'h20E0;    //  bigger deeper
        mem[12'h013] = 16'h20E1;    // smaller shalllow
        mem[12'h014] = 16'h4006;    // return

        // variable for sort
        mem[12'h0F0] = 16'h000F;    // cnst: 5
        mem[12'h0F1] = 16'h0007;    // cnst: 9
    end
endmodule

```

该仿真中 RAM 赋值



当内存中取值为 F 和 7 时排序行为级仿真和测试运行结果

out 在最后按从小到大顺序输出了 7、F



当内存中取值为 5 和 9 时排序行为级仿真和测试运行结果

out 在最后按从小到大顺序输出了 5、9

在此集成 CPU 的程序编写上，完成了一个简单的两个数字比大小并按从小到大的顺序输出的功能，重点麻烦在交换，需要明晰 alu 在进行二元计算后会自动 pop 出所使用的数据；了解 JNZ 跳转指令执行时同时会 pop 出所使用数据。了解栈内数据存储情况后便可利用内存进行数据交换。

5. 总结与讨论

本次实验中根据教师指点和研究学习，顺利完成了一个简易的 CPU，并用其成功运行了烧写在 RAM 中的数字排序输出功能。在此过程中深入了解了每一个简单的模块是如何数据交互、相互协作完成复杂任务的，花了很多时间学习理解，但是看到寥寥几行 Verilog 代码就能转变成庞大错综复杂的硬件结构，我感到震撼，但在现有芯片中显示所涉及使用部分时却仅微小部分被点亮，说明所有我做一切，仅仅是蹒跚学步，还是入门中的入门，同时意识到处理器制作的复杂之处；最终 CPU 运行产生的波形图如我所愿完成简单任务时觉得依然内心欢喜，颇有满足感，所作一切努力没有浪费，感谢您给予的这次实验机会，让我初步理解了处理器的基础结构、功能、运行流程。

在此过程中，经过思考我发现了一些设计不足/不满意的地方，比如：堆栈模块中并未定义空栈时 pop 边界行为，经过思考，我认为是在逻辑简化上做的取舍，边界判断会导致硬件的复杂和不必要的繁冗简陋，此些应当在软件编程时注意并避免，硬件资源珍贵，应尽量舍去非必须功能。此外在编写 RAM 程序时发现寄存器过少，逻辑运算结果于 stack 内原地完成，导致多个数据处理繁琐麻烦，需借用 RAM 暂存，无论是存储还是读取都会拖慢运行速度，以后应在此方面改进。综上，实验顺利完成，感谢您的讲授。