Project3 Logisim完成单周期CPU开发

一、总体设计

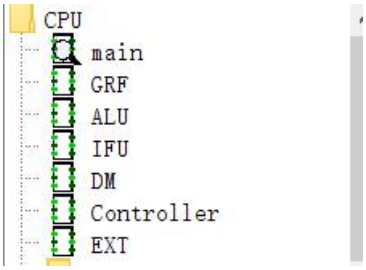

图 1 顶层模块

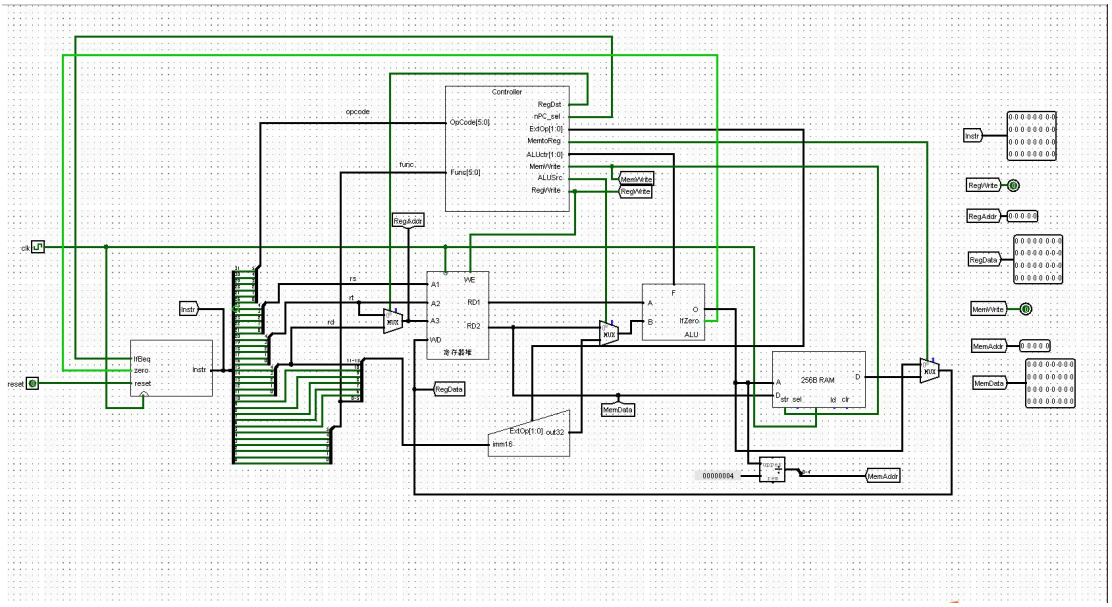

图 2 顶层电路图

二、模块定义

1、IFU

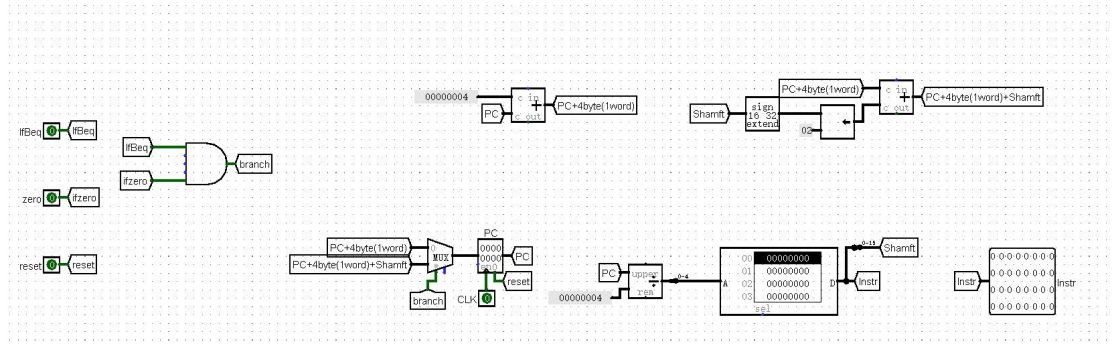

图 3 IFU 电路图

（1）基本描述

IFU 主要作用是完成取指令功能。IFU 内部包含 PC、IM 以及其他相关逻辑操作。IFU 除了能执行顺序取指令外，还能根据 BEQ 指令的执行情况决定 PC 接下来的操作是顺序取指令还是转移取指令。

（2）模块接口

表 1 IFU 模块接口

信号名	方向	描述
IfBeq	I	判断当前指令是否为 beq 指令的标志 1：当前指令为 beq 指令 0：当前指令非 beq 指令
Zero	I	判断 ALU 计算结果是否为 0 的标志 1：计算结果为 0 0：计算结果非 0
clk	I	时钟信号
reset	I	判断 PC 是否复位的信号 1：复位 0：无效
Instr	O	32 位 MIPS 指令

（3）功能定义

表 2 IFU 功能定义

序号	功能名称	功能描述
1	复位	当复位信号有效时 PC 设置成 0x0000_0000
2	取指令	根据 PC 指定的地址从 IM 中取出指令
3	计算下一条指令地址	*PC 取地址为 4 字节即一个字，所以 PC 的低 2 位地址可以省略 如果当前 Zero 为 1 且 IfBeq 为 1，则 PC←PC+sign_ext 否则，PC←PC+1

2、GRF

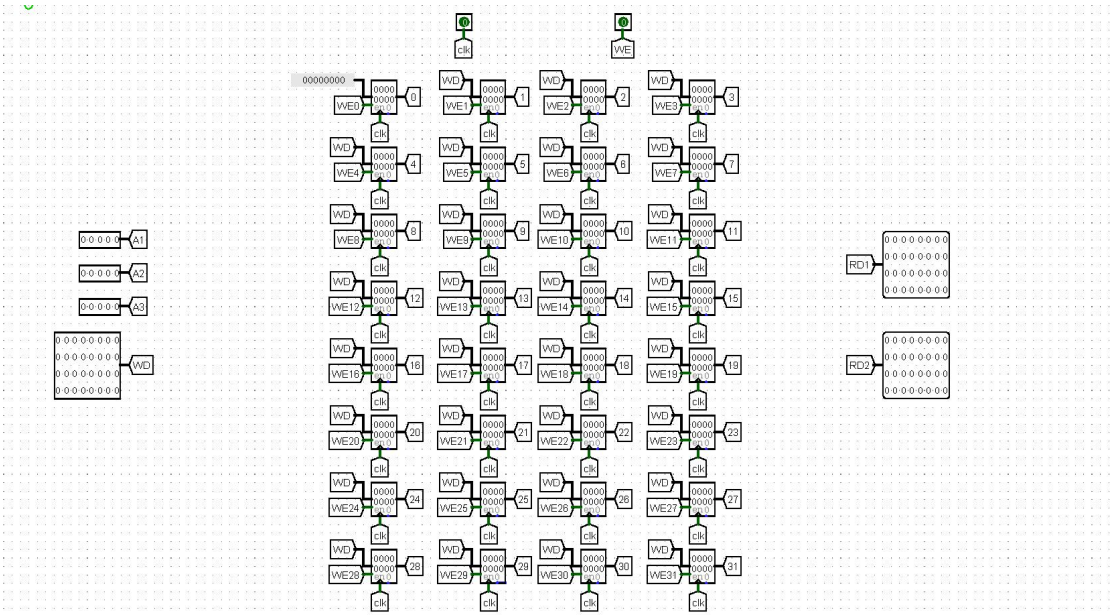

图 4.1 GRF 电路图

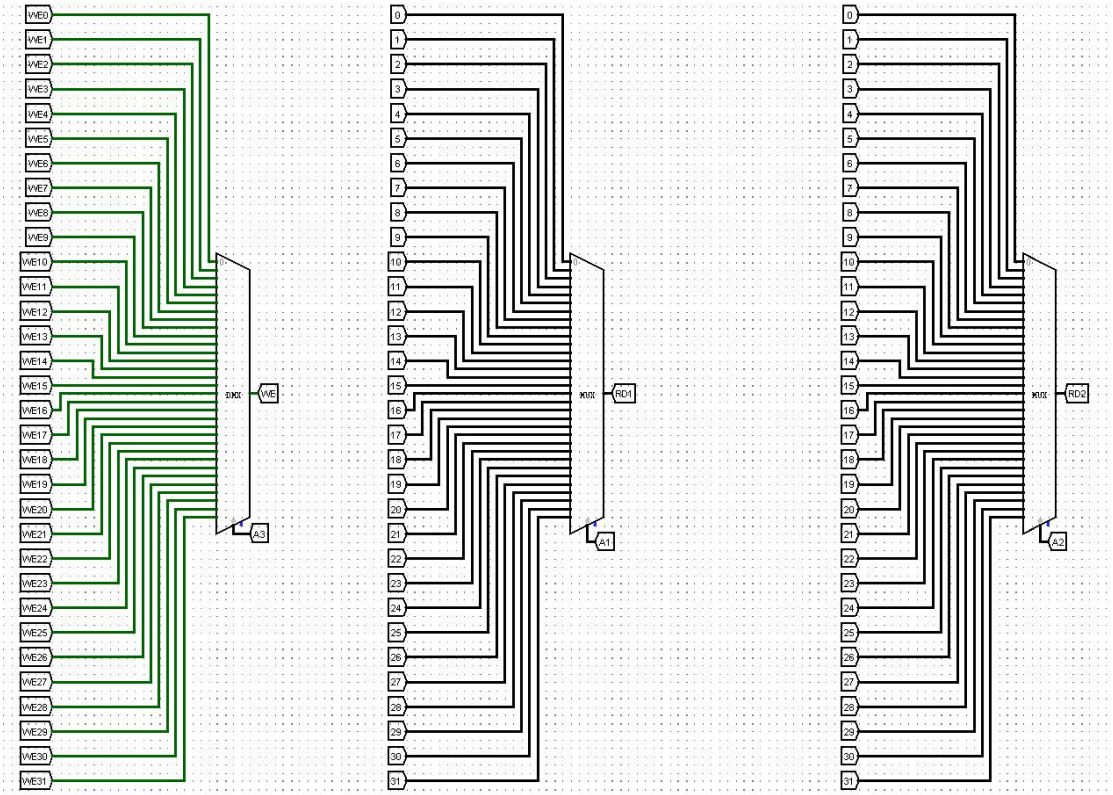

图 4.2 GRF 电路图

（1）基本描述

GRF 的作用主要是提供相应的寄存器以供指令完成相应的操作。GRF 内部包含 32 个 32 位寄存器，分别对应 0-31 号寄存器，其中 0 号寄存器的读取结果均为 0，还包含选择读写寄存器的逻辑操作电路。

（2）模块接口

表 3 GRF 模块接口

信号名	方向	描述
clk	I	时钟信号
WE	I	写使能信号 1：可向 GRF 中写入数据 0：不可向 GRF 中写入数据
A1	I	5 位地址输入信号，指定 32 个寄存器中的一个，将其中存储的数据读出到 RD1
A2	I	5 位地址输入信号，指定 32 个寄存器中的一个，将其中存储的数据读出到 RD2
A3	I	5 位地址输入信号，指定 32 个寄存器中的一个作为写入的目标寄存器
WD	I	32 位数据输入信号
RD1	O	输出 A1 指定的寄存器中的 32 位数据
RD2	O	输出 A2 指定的寄存器中的 32 位数据

（3）功能定义

表 4 GRF 功能定义

序号	功能名称	描述
1	读寄存器	读出 A1、A2 地址对应寄存器中所存储的数据到 RD1、RD2
2	写寄存器	当 WE 信号有效且时钟上升沿到来时，将 WD 写入 A3 所对应的寄存器中

3、ALU

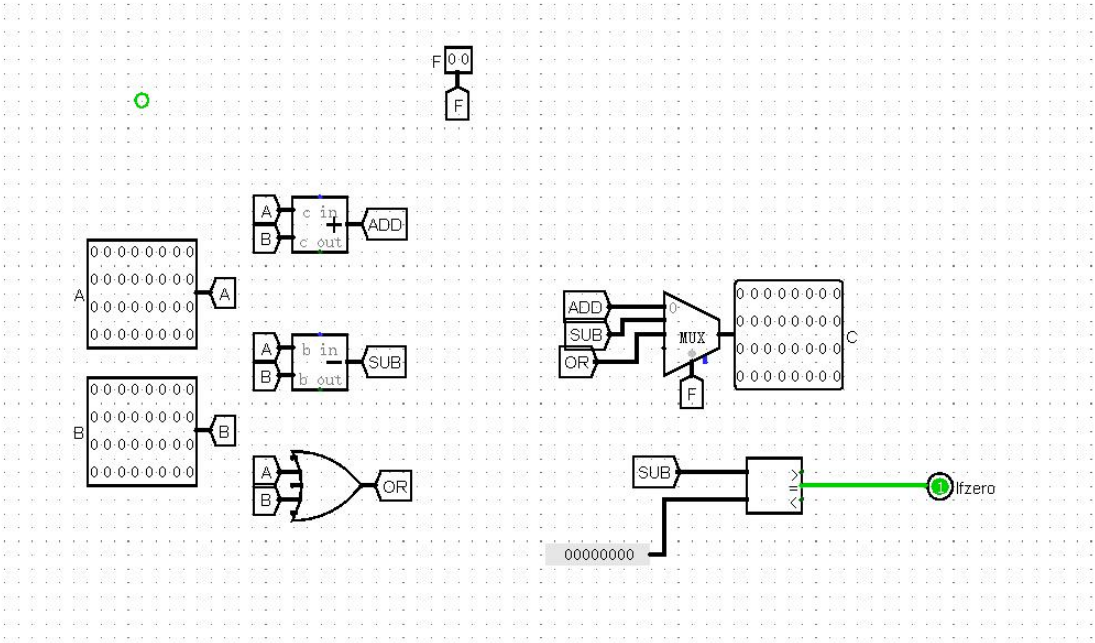

图 5 ALU 电路图

（1）基本描述

ALU 的主要作用是完成算术逻辑指令或者其他指令所需要的算术逻辑操作。ALU 内部包含加法、减法、与或非等基本的算数逻辑部件。

（2）模块接口

表 5 ALU 模块接口

信号名	方向	描述
A	I	32 位输入信号，参与 ALU 计算的第一个值
B	I	32 位输入信号，参与 ALU 计算的第二个值
F	I	2 位输入信号，ALU 的功能选择信号： 00：ALU 进行加法运算 01：ALU 进行减法运算 10：ALU 进行或运算
C	O	32 位输出信号，ALU 的计算结果
Ifzero	O	通过减运算输出的值和常数 0 作比较判断输入两值是否相等的信号

（3）功能定义

表 6 ALU 功能定义

序号	功能名称	描述
1	加运算	C = A + B
2	减运算	C = A - B
3	或运算	C = A | B

4、EXT

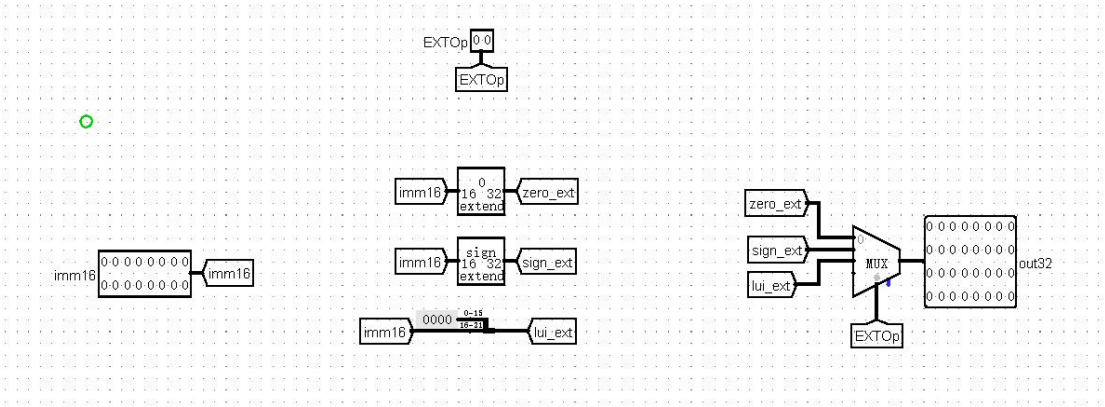

图 6 EXT 电路图

（1）基本描述

EXT 的主要作用是完成对输入到其中的 16 位数据的符号扩展、零扩展以及将输入的 16 位数据加载到高位等操作。其内部包含与完成符号扩展、零扩展以及讲输入的 16 位数据加载到高位等操作的相关逻辑部件。

（2）模块接口

表 7 EXT 模块接口

信号名	方向	描述
imm16	I	输入 EXT 内部需要被扩展的 16 位数据
EXTOp	I	输入数据进行扩展的方式的选择信号： 00：将 imm16 进行零扩展到 32 位 01：将 imm16 进行符号扩展到 32 位 10：将 imm16 加载到高位，低位补 0
out32	O	imm16 进行扩展后的数据输出

（3）功能定义

表 8 EXT 功能定义

序号	功能名称	描述
1	零扩展	将 imm16 进行高位补 0 扩展到 32 位
2	符号扩展	将 imm16 进行符号扩展到 32 位
3	加载到高位	将 imm16 加载到高位，低位补 0

5、DM

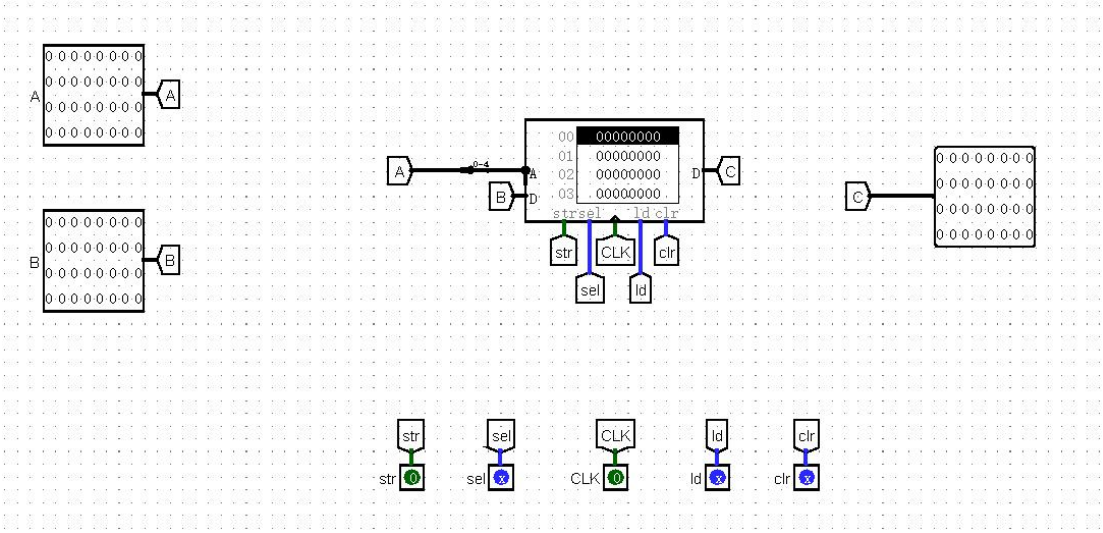

图 7 DM 电路图

（1）基本描述

DM 的主要作用是作为 CPU 执行程序时的临时数据存储媒介。内部包含存储芯片及相关逻辑通路。

（2）模块接口

表 9 DM 模块接口

信号名	方向	描述
A	I	32 位输入信号，操作存储器的地址
B	I	32 位输入信号，为写入数据的输入
CLK	I	时钟信号
str	I	读写控制信号 1：写信号 0：读信号
C	O	32 位输出信号，输出存储器操作地址输入对应的数据

（3）功能定义

表 10 DM 功能定义

序号	功能名称	描述
1	读	根据输入的存储器地址读出数据
2	写	根据输入的存储器地址，写入输入的数据

三、Controller 设计

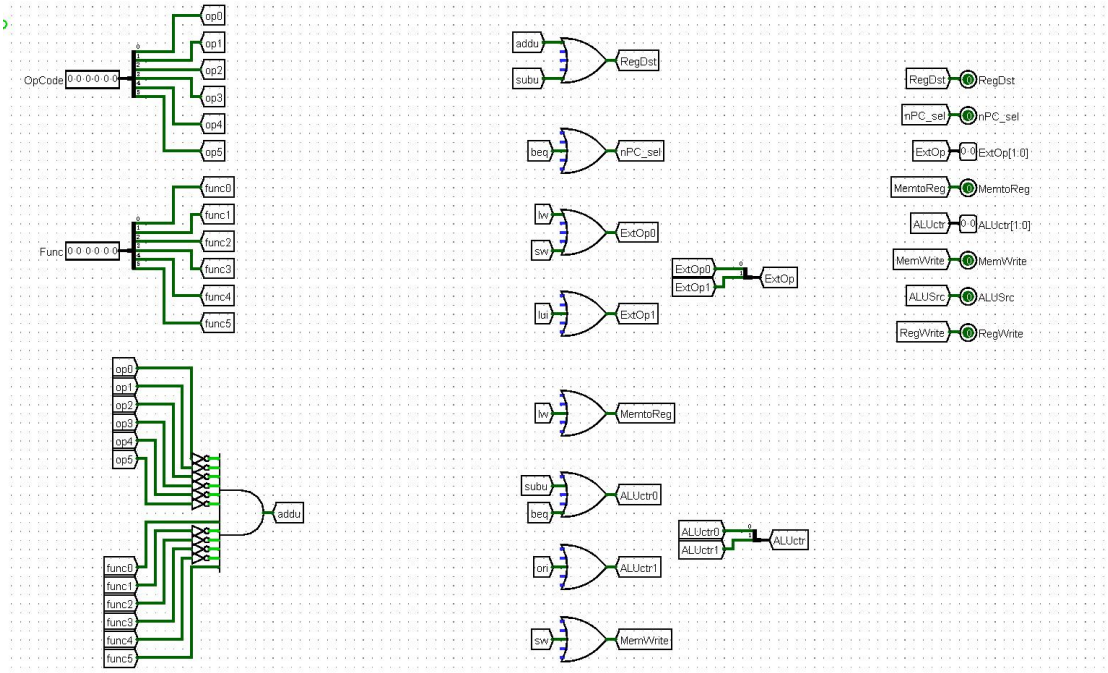

图 8.1 Controller 电路图

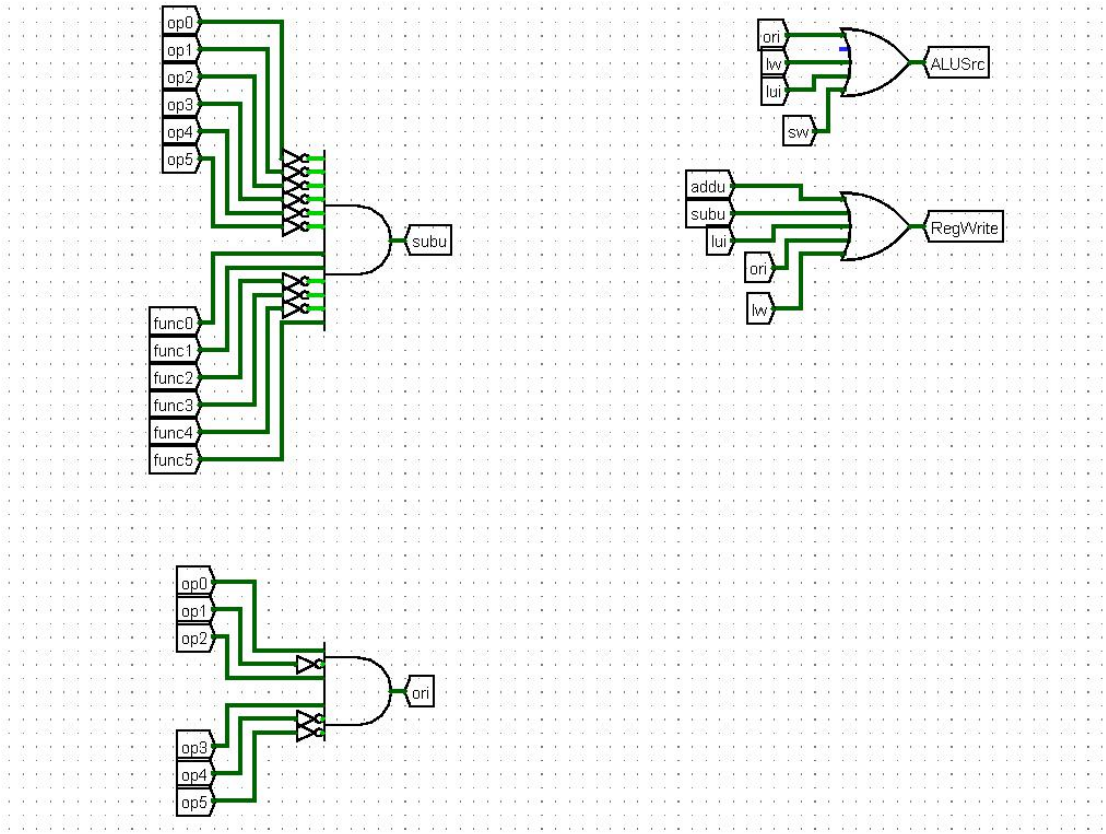

图 8.2 Controller 电路图

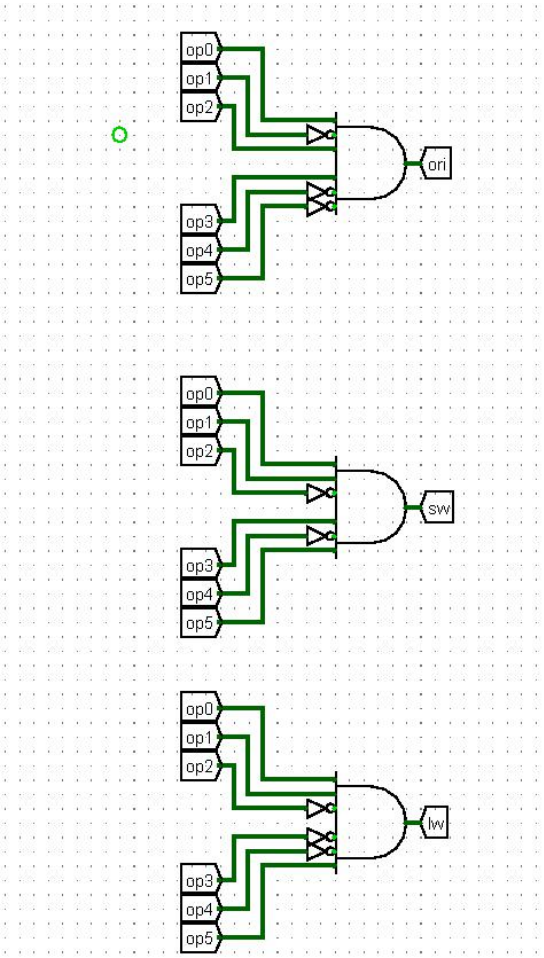

图 8.3 Controller 电路图

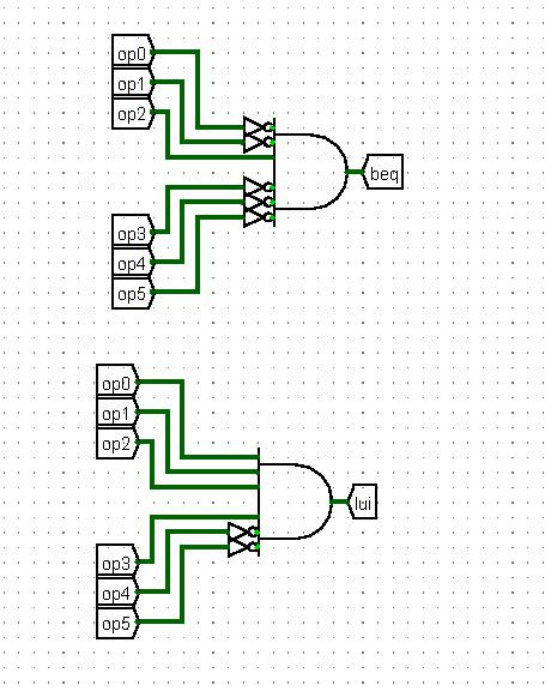

图 8.4 Controller 电路图

（1）基本描述

控制器的主要作用是译码，即将每一条机器指令中包含的信息，转化为给 CPU 各部分的控制信号。其内部主要包含与或门阵列，与逻辑部分的功能是识别，将输入的机器码识别为相应的指令。或逻辑部分的功能是生成，根据输入的指令不同，产生不同的控制信号。

（2）模块接口

表 11 Controller 模块接口

信号名	方向	描述
OpCode	I	机器指令的操作码部分
Func	I	机器指令的函数码部分
RegDst	O	GRF 写地址控制
nPC_sel	O	BEQ 指令标志
ExtOp[1:0]	O	EXT 扩展方式控制标志
MemtoReg	O	DM 输出数据控制标志
ALUctr[1:0]	O	ALU 运算控制标志
MemWrite	O	DM 写入数据控制标志
ALUSrc	O	ALU 操作数控制标志
RegWrite	O	GRF 写入数据控制标志

（3）单周期真值表

表 12 Controller 单周期真值表

Instruction	addu	subu	ori	lw	sw	beq	lui
Func	100001	100011	N/A				
Op	000000	000000	001101	100011	101011	000100	001111
RegDst	1	1	0	0	X	X	0
nPC_sel	0	0	0	0	0	1	0
MemtoReg	0	0	0	1	X	X	0
MemWrite	0	0	0	0	1	0	0
ALUSrc	0	0	1	1	1	0	1
RegWrite	1	1	1	1	0	0	1
ExtOp[1:0]	X	X	0	1	1	X	2
ALUctr[1:0]	0	1	2	0	0	1	X

四、测试 CPU

```
lui $s1,0x0008              #lui 测试程序要实现：立即数 0x0008 加载到 s1
                            #（17）寄存器高位

ori $s0,$zero,0x00000004    #ori 测试程序要实现：zero（0）寄存器中的数据与
                            #立即数 0x00000004 进行或运算，结果存储在 s0
                            #(16)寄存器中

ori $t0,$zero,0x00003000    #ori 测试程序要实现：zero（0）寄存器中的数据与
                            #立即数 0x00003000 进行或运算，结果存储在 t0(8)
                            #寄存器中

sw $t0,0($s0)               #sw 测试程序要实现：把 t0(8)寄存器中的数据
                            #（1word），存储到 s0(16)的值再加上偏移量 0 所
                            #指向的 RAM 中

sw $t0,4($s0)               #sw 测试程序要实现：把 t0(8)寄存器中的数据
                            #（1word），存储到 s0(16)的值再加上偏移量 4 所
                            #指向的 RAM 中

loop:
lw $s2,0($s0)               #lw 测试程序要实现：把 s0(16)寄存器中的值再加上
                            #偏移量 0 所指向的 RAM 中的数据取出来并存到
                            #s2(18)寄存器中

addu $s2,$s2,$s1            #add 测试程序要实现：s1(17)寄存器中的值加上
                            #s2(18)寄存器中的值后将结果存到 s2(18)寄存器中

nop                         #nop 测试程序不用实现任何指令，默认 PC <- PC+4
```

```
sw $s2,0($s0)          #sw 测试程序要实现：把 s2(18)寄存器中的数据
                       #（1word），存储到 s0(16)的值再加上偏移量 0 所指
                       #向的 RAM 中

beq $t0,$s2,loop       #beq 测试程序要实现：判断 t0(8)的值和 s2(18)的值
                       #是否相等，相等跳转 loop

lw $s2,4($s0)          #lw 测试程序要实现：把 s0(16)寄存器中的值再加
                       #上偏移量 4 所指向的 RAM 中的数据取出来并存到
                       #s2(18)寄存器中

subu $s2,$s1,$s2       #subu 测试程序要实现：s1(17)的值减去 s2(18)
                       #的值后将结果存到 s2（18）中

sw $s2,4($s0)          #sw 测试程序要实现：把 s2(18)寄存器中的数据
                       #（1word），存储到 s0(16)的值再加上偏移量 4 所
                       #指向的 RAM 中
```

机器码：

```
3c110008  34100004  34083000  ae080000  ae080004  8e120000
02519021  00000000  ae120000  1112fffb  8e120004  02329023
ae120004
```

MARS 模拟结果：

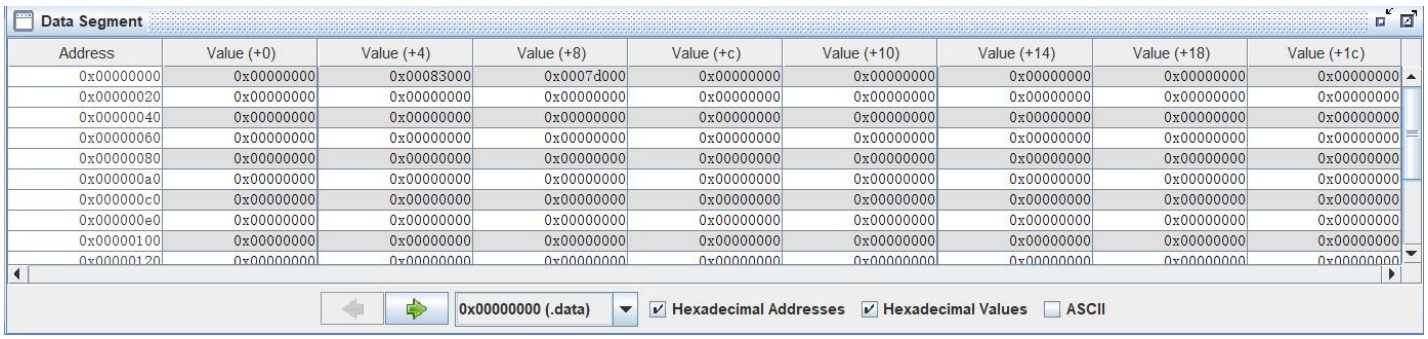

图 9 测试程序 MARS 模拟结果

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00003000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000004
$s1	17	0x00080000
$s2	18	0x0007d000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00003034
hi		0x00000000
lo		0x00000000

图 10 测试程序 MARS 模拟结果

Logisim:

DM：

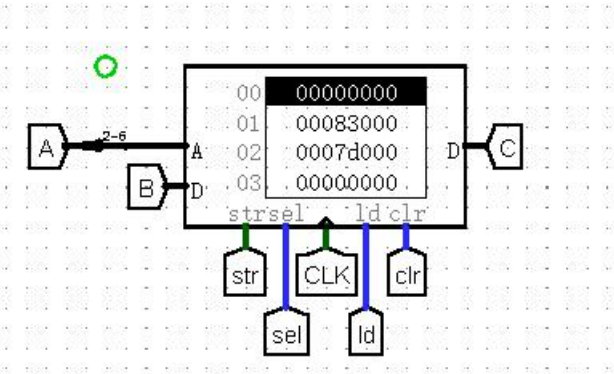

图 11 测试程序 Logisim 模拟结果之 DM

GRF：

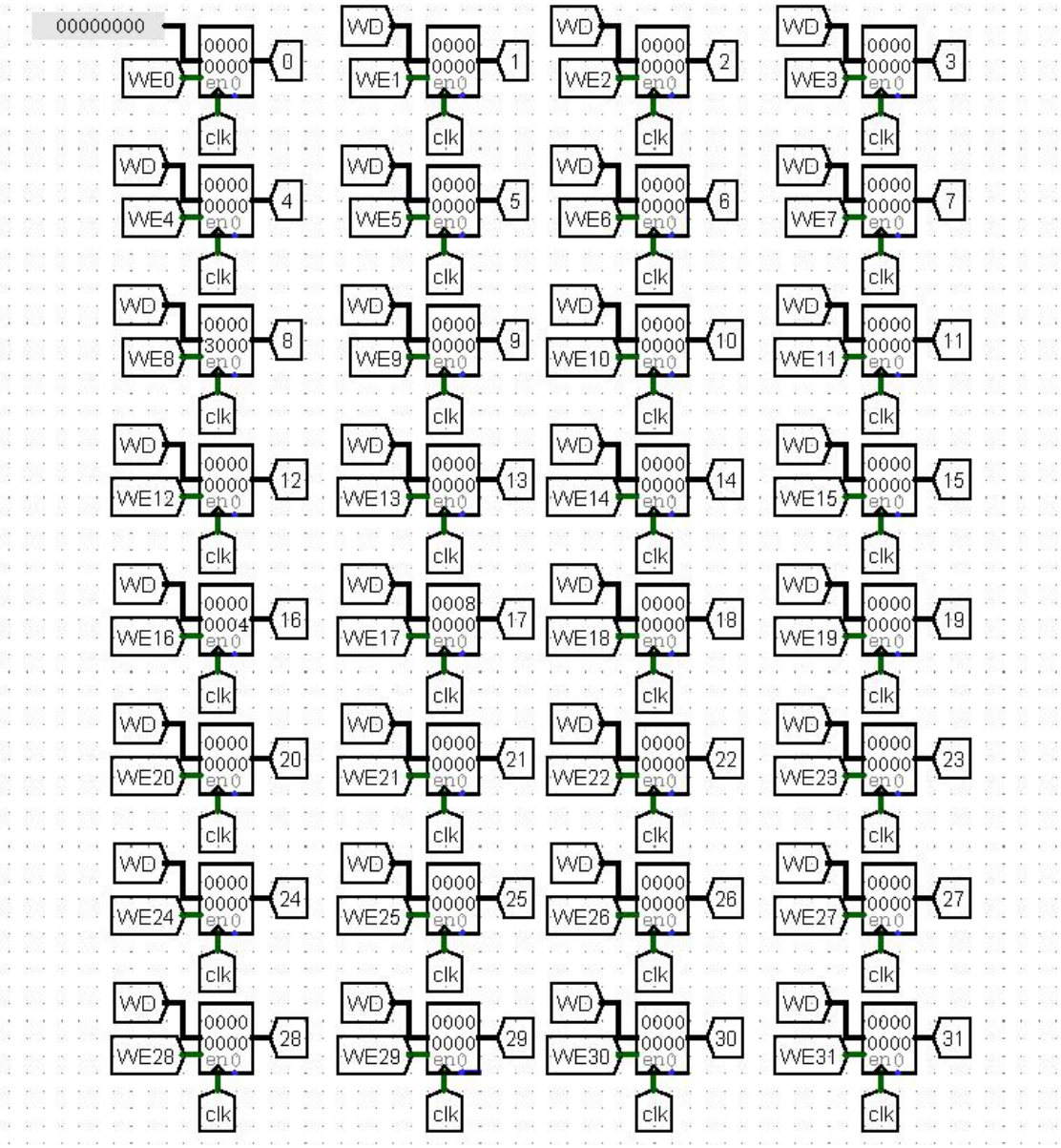

图 12 测试程序 Logisim 模拟结果之 GRF

五、思考题

1、若 PC（程序计数器）位数为 30 位，试分析其与 32 位 PC 的优劣。

在计算下一条指令地址时，32 位 PC 可以直接加 4，而 30 位 PC 需要左移两位才能加 4，即通常情况下 30 位 PC 可以省略低两位而直接加 1。

2、现在我们的模块中 IM 使用 ROM， DM 使用 RAM， GRF 使用寄存器，这种做法合理吗？ 请给出分析，若有改进意见也请一并给出。

合理。ROM 是只读存储器，作为指令存储器来存储指令可以避免电路运行时对程 序进行修改，更好的保持程序的稳定性；RAM 可读可写，且可以满足 DM 用来向 内存中存储数据，需要足够的存储空间而不需要特别快的访存速度的要求；GRF 用寄存器来实现的话可以满足其需要非常快的存储速度的需求。

3、结合上文给出的样例真值表，给出 RegDst，ALUSrc，MemtoReg，RegWrite，nPC_Sel, ExtOp 与 op 和 func 有关的布尔表达式（表达式中只能使用“与、或、非”3 种基本逻辑运算。）

$$\mathrm{Re}\,gDst = \overline{O_5}\,\overline{O_4}\,\overline{O_3}\,\overline{O_2}\,\overline{O_1}\,\overline{O_0} f_5 \overline{f_4}\,\overline{f_3}\,\overline{f_2} f_0$$

$$ALUSrc = \overline{O_5}\,\overline{O_4} O_3 O_2 O_0 + O_5 \overline{O_4}\,\overline{O_2} O_1 O_0$$

$$Memto\,\mathrm{Re}\,g = O_5 \overline{O_4}\,\overline{O_3}\,\overline{O_2} O_1 O_0$$

$$\mathrm{Re}\,gWrite = \overline{O_5}\,\overline{O_4}\,\overline{O_3}\,\overline{O_2}\,\overline{O_1}\,\overline{O_0} f_5 \overline{f_4}\,\overline{f_3}\,\overline{f_2} f_0 + \overline{O_5}\,\overline{O_4} O_3 O_2 \overline{O_1} O_0 + O_5 \overline{O_4}\,\overline{O_3}\,\overline{O_2} O_1 O_0 + \overline{O_5}\,\overline{O_4} O_3 O_2 O_1 O_0$$

$$nPC\_Sel = \overline{O_5}\,\overline{O_4}\,\overline{O_3} O_2 \overline{O_1}\,\overline{O_0}$$

$$ExtOp0 = O_5 \overline{O_4}\,\overline{O_2} O_1 O_0$$

$$ExtOp1 = \overline{O_5}\,\overline{O_4} O_3 O_2 O_1 O_0$$

4、充分利用真值表中的 X 可以将以上控制信号化简为最简单的表达式，请给出化简后的形式。

将 X 按照方便化简的原则当做 0 或 1，则得最简式如下：

$$\mathrm{Re}\,gDst = \overline{O_5}\,\overline{O_4}\,\overline{O_3}\,\overline{O_2}\,\overline{O_1}\,\overline{O_0} f_5 \overline{f_4}\,\overline{f_3}\,\overline{f_2} f_0$$

$$ALUSrc = \overline{O_5}\,\overline{O_4} O_3 O_2 O_0 + O_5 \overline{O_4}\,\overline{O_2} O_1 O_0$$

$$Memto\,\mathrm{Re}\,g = O_5 \overline{O_4}\,\overline{O_2} O_1 O_0$$

$$\mathrm{Re}\,gWrite = \overline{O_5}\,\overline{O_4}\,\overline{O_3}\,\overline{O_2}\,\overline{O_1}\,\overline{O_0} f_5 \overline{f_4}\,\overline{f_3}\,\overline{f_2} f_0 + \overline{O_5}\,\overline{O_4} O_3 O_2 \overline{O_1} O_0 + O_5 \overline{O_4}\,\overline{O_3}\,\overline{O_2} O_1 O_0 + \overline{O_5}\,\overline{O_4} O_3 O_2 O_1 O_0$$

$$nPC\_Sel = \overline{O_5}\,\overline{O_4}\,\overline{O_3} O_2 \overline{O_1}\,\overline{O_0}$$

$$ExtOp0 = O_5 \overline{O_4}\,\overline{O_2} O_1 O_0$$

$$ExtOp1 = \overline{O_5}\,\overline{O_4} O_3 O_2 O_1 O_0$$

5、事实上，实现 nop 空指令，我们并不需要将它加入控制信号真值表，为什么？请给出你的理由。

nop 指令本身的意义就是对电路中的元件不进行任何操作，且其机器码为 0x00000000，所以不管从实际意义和机器码来看，其存在与否都对电路的工作没有影响。

6、前文提到，“可能需要手工修改指令码中的数据偏移”，但实际上只需再增加一个 DM 片选信号,就可以解决这个问题。请阅读相关资料并设计一个 DM 改造方案使得无需手工修改数据偏移。

前文提到的可能需要手工修改指令码中的数据偏移，是因为 MARS 中指令存储器的起始地址和数据存储器的起始地址是不重叠的，如果 MARS 中设置指令存储器的起始地址为 0，则 MARS 中的数据存储器的起始地址就有了偏移量如 0x00003000,但是此次设置的 DM 中的 RAM 很小，地址位宽只有 5 位，即在此次试验中，我们只截取了地址输入信号的[6:2]位进行工作，所以当偏移量比实验用的 RAM 容量要大时，不会产生取错数的影响。

但若 DM 容量很大，需要多片 RAM 来实现 DM 时，则需要添加片选信号来解决手工修改指令码中的数据偏移的问题。例如，若设置 MARS 数据段地址范围为 0x30000000-0x3fffffff，即需要 256MB×32 容量的 DM，其存储器地址线为 28 位，而 Logisim 中，RAM 的地址位最多为 24 位，即每片 RAM 的容量最大为 16MB×32，所以需要 16 片 RAM 去实现满足此例的 DM，那么只需要将输入 DM 的地址信号的[31:28]位与 0x3 比较，若相等则表明需存取的数据存储器范围为

0x30000000-0x3fffffff 之间，再取输入 DM 的地址信号的[27:24]位作为这个区域 16 片的片选信号即可。

7、除了编写程序进行测试外，还有一种验证 CPU 设计正确性的办法——形式验证。 形式验证的含义是根据某个或某些形式规范或属性，使用数学的方法证明其正确性或非正确性。请搜索“形式验证（Formal Verification)”了解相关内容后，简要阐述相比与测试，形式验证的优劣。

形式验证的优点：

1、测试者不必考虑如何获得测试向量，因为形式验证是借用数学上的方法将待验证电路和功能描述或参考设计直接进行比较的。

2、有效地克服了测试验证的覆盖面不全的缺点，因为形式验证是对指定描述的所有可能的情况进行验证，覆盖率 100%，而不仅是对其中的部分进行多次试验。

3、形式验证可以进行的验证层次广，可以进行从系统级到门级的验证，而且验证时间短，有利于尽早、尽快地发现并改正电路设计中存在的错误，利于缩短设计时间和周期。

形式验证的缺点：

形式验证到目前为止仍不能有效地验证电路的时延和功耗等性能。