Verilog单周期CPU设计文档

作者：李健健

1. CPU设计方案综述
2. 总体设计概述

使用Verilog开发一个简单的单周期CPU，总体概述如下：

1. 此CPU为32位CPU
2. 此CPU为单周期设计
3. 此CPU支持的指令集为：

{addu, subu, ori, lw, sw, beq, lui, jal, jr,nop}

1. nop机器码为0x00000000
2. addu, subu不支持溢出
3. 关键模块定义
4. IM
5. 端口说明

表1-IM端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 描述 |
| 1 | PC[31:0] | I | 时钟信号 |
| 2 | instr[31:0] | O | 指令 |

1. 功能定义

表2-IM功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 取指令 | 就是取指令 |

1. PC
2. 端口说明

表3-PC端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 描述 |
| 1 | clk | I | 时钟信号 |
| 2 | reset | I | 复位信号 |
| 3 | NPC | I | 下一条指令所在IM地址 |
| 4 | PC | O | 当前指令所在IM地址 |

1. 功能定义

表4-PC功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 存储指令的地址 | 保存当前执行指令在IM中的地址 |

1. NPC
2. 端口说明

表5-NPC端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 描述 |
| 1 | branch | I | 分支信号 |
| 2 | j | I | 跳转信号 |
| 3 | jr | I | 指令是不是jr |
| 4 | PC[31:0] | I | 当前PC值 |
| 5 | RegJump[31:0] | I | 跳转寄存器中地址值 |
| 6 | imm26[25:0] | I | 26位立即数 |
| 7 | PC4[31:0] | O | PC+4 |
| 8 | NPC[31:0] | O | 根据各种指令计算出的下一个PC值 |

1. 功能定义

表6-NPC功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 计算下一个PC的值 |  |
| 2 | 输出PC4 | 为jr指令写入寄存器做准备 |

1. GRF
2. 端口说明

表7-GRF端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 描述 |
| 1 | clk | I | 时钟信号 |
| 2 | reset | I | 异步复位信号，将32个寄存器中全部清零  1：清零  0：无效 |
| 3 | WE | I | 写使能信号  1：可向GRF中写入数据  0：不能向GRF中写入数据 |
| 4 | A1[4:0] | 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 |
| 5 | A2[4:0] | 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 |
| 6 | A3[4:0] | I | 5位地址输入信号，指定32个寄存器中的一个，作为RD的写入地址 |
| 7 | WD[31:0] | I | 32位写入数据 |
| 8 | RD1[31:0] | O | 输出A1指定的寄存器的32位数据 |
| 9 | RD2[31:0] | O | 输出A2指定的寄存器的32位数据 |

1. 功能定义

表8-GRF功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 异步复位 | reset为1时，将所有寄存器清零 |
| 2 | 读数据 | 将A1和A2地址对应的寄存器的值分别通过RD1和RD2读出 |
| 3 | 写数据 | 当WE为1且时钟上升沿来临时，将WD写入到A3对应的寄存器内部 |

1. ALU
2. 端口说明

表9-ALU端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 描述 |
| 1 | A[31:0] | I | 参与运算的第一个数 |
| 2 | B[31:0] | I | 参与运算的第二个数 |
| 3 | AluOp[2:0] | I | 决定ALU做何种操作  000：无符号加  001：无符号减  010：与  011：或  100：将B[15:0]做为res[31:16],res[15:0]=0 |
| 4 | eq | O | A与B是否相等  0：不相等  1：相等 |
| 5 | res[31:0] | O | A与B做运算后的结果 |

1. 功能定义

表10-ALU功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 加运算 | res = A + B |
| 2 | 减运算 | res = A - B |
| 3 | 与运算 | res = A & B |
| 4 | 或运算 | res = A | B |
| 5 | 加载高位运算 | res = {B[15:0], 16’h0} |

1. DM
2. 端口说明

表11-DM端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 描述 |
| 1 | clk | I | 时钟信号 |
| 2 | reset | I | 异步复位信号  0：无效  1：内存值全部清零 |
| 3 | WE | I | 写使能信号  0：禁止写入  1：允许写入 |
| 4 | MemAddr[31:0] | I | 读取或写入信号地址 |
| 5 | WD[31:0] | I | 32为写入数据 |
| 6 | RD[31:0] | O | 32位读出数据 |

1. 功能定义

表12-DM功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 异步复位 | 当reset为1时，DM中所有数据清零 |
| 2 | 写入数据 | 当WE有效时，时钟上升沿来临时，WD中数据写入A对应的DM地址中 |
| 3 | 读出数据 | RD永远读出A对应的DM地址中的值 |

1. EXT
2. 端口说明

表13-EXT端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 描述 |
| 1 | imm16[15:0] | I | 代扩展的16位信号 |
| 2 | sign | I | 无符号或符号扩展选择信号  0：无符号扩展  1：符号扩展 |
| 3 | imm32[31:0] | O | 扩展后的32位的信号 |

1. 功能定义

表14-EXT功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 无符号扩展 | 当sign为0时，将imm16无符号扩展输出 |
| 2 | 符号扩展 | 当sign为1时，将imm16符号扩展输出 |

1. Controller
2. 端口说明

表15-Controller端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 描述 |
| 1 | instr[31:0] | I | instr[31:26],6位控制信号 |
| 2 | eq | I | RegRead1和RegRead是否相等 |
| 3 | WeGrf | O | GRF写使能信号  0：禁止写入  1：允许写入 |
| 4 | WeDm | O | DM的写入信号  0：禁止写入  1：允许写入 |
| 5 | RegDst[1:0] | O | GRF写入地址选择信号  0：Rd  1：Rt |
| 6 | WhichtoReg[1:0] | O | 将何种数据写入GRF？  00：ALU计算结果  01：DM读出信号  11：upperImm |
| 7 | AluSrc | O | 参与ALU运算的第二个数，来自GRF还是imm  0：来自GRF  1：imm |
| 8 | AluOp[2:0] | O | ALU的控制信号 |
| 9 | sign | O | 是否对imm16进行符号扩展  0：不进行符号扩展  1：进行符号扩展 |
| 10 | branch | O | instr是否为beq信号  0：不是  1：是 |
| 11 | JType | O | 是不是J型指令，只有j和jal是 |
| 12 | jr | O | 是不是jr指令 |

1. 真值表

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 端口 | addu | subu | ori | lw | sw | lui | beq | jal | jr |
| op | 000000 | 000000 | 001101 | 100011 | 101011 | 001111 | 000100 | 000011 | 000000 |
| func | 100001 | 100011 |  |  |  |  |  |  | 001000 |
| WeGrf | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| WeDm | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| RegDst | 00 | 00 | 01 | 01 | 00 | 01 | 00 | 10 | 00 |
| WhichtoReg | 00 | 00 | 00 | 01 | 00 | 00 | 00 | 10 | 00 |
| AluSrc | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| AluOp | 000 | 001 | 011 | 000 | 000 | 100 | 000 | 000 | 000 |
| sign | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| branch | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| JType | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| jr | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

1. 测试方案
2. 测试代码：

.text

ori $t1, $v0, 100 # t1: 100

ori $t2, $v0, 250 # t2: 250

ori $t3, $v0, 200 # t3: 200

addu $t4, $t1, $t3

# t4 = t1 + t3 (t4 = 300)

subu $t5, $t4, $t2

# t5 = t4 - t2 (t5 = 50)

sw $t5, 4($v0)

lw $t6, 4($v0)

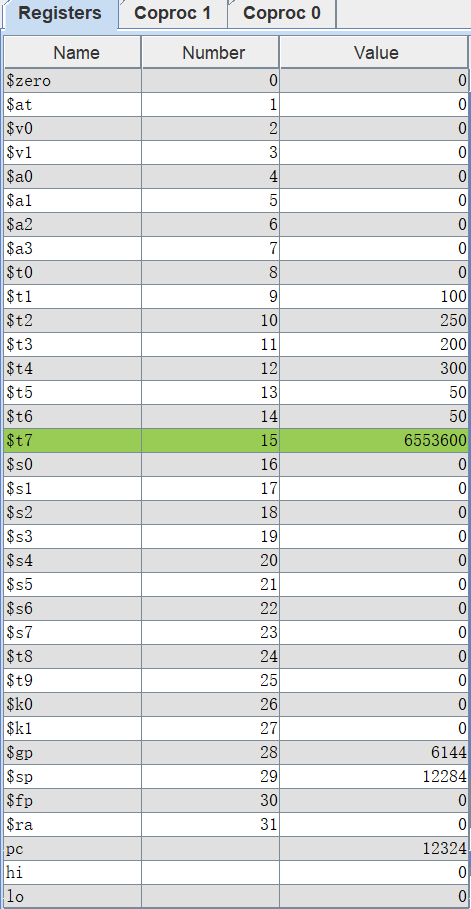
Beq:

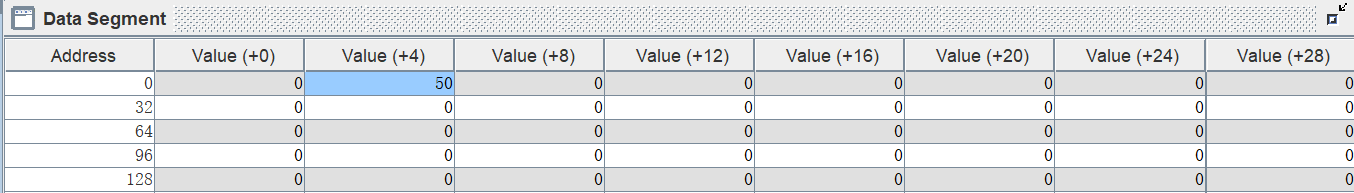
lui $t7, 100

beq $t5, $t1, Beq

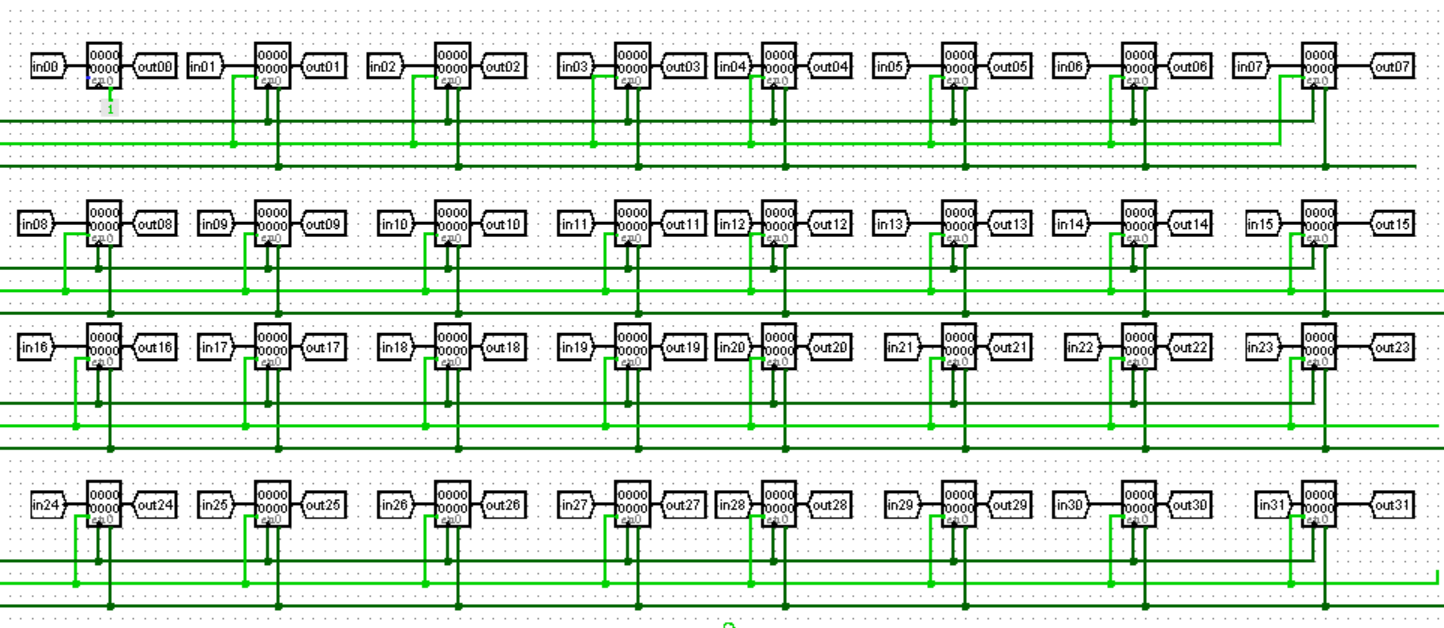
beq $t6, $t5, Beq

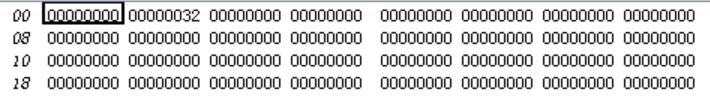
1. MARS中运行结果





1. 该CPU运行结果





1. 思考题
2. 现在我们的模块中IM使用ROM， DM使用RAM， GRF使用Register，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。

IM只需被读取，ROM只有读取功能；

DM既要进行读取，又要进行写入，但是一个周期只会进行读取和写入之一，RAM的单一地址和各一个的读写端口满足了这种要求，

当然，用寄存器也能实现DM，但是DM需要较大的空间，使用寄存器太“浪费”；

GRF需要读写，且其与ALU直接连接，需要高速地读写，故使用寄存器堆搭建合理。

抱歉，没有改进意见。

1. 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，整个CPU只执行PC<=PC+4指令，一段指令中，加与不加，执行结果没有区别。

1. 上文提到，MARS不能导出PC与DM起始地址均为0的机器码。实际上，可以通过为DM增添片选信号，来避免手工修改的麻烦，请查阅相关资料进行了解，并阐释为了解决这个问题，你最终采用的方法。

如果很不幸，寄存器中存储的DM的地址被映射在0x3000\_0000到0x3fff\_ffff间，而我们的DM起始地址是0，那么，我们可以将输入地址直接减去0x3000\_0000，再作为DM的地址输入。

假如我们不确定寄存器中的存储的DM地址的起始值，我们可以将其与0x3000\_0000比较，得到片选信号。

1. 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，简要阐述相比于测试，形式验证的优劣之处。

形式验证的优点如下：

1. 所有可能的情况进行验证，覆盖率达到了100%。
2. 形式验证的验证时间短，可以很快发现和改正电路设计中的错误，可以缩短设计周期。

形式验证的缺点如下：

1. 形式验证只能检验电路设计的正确性，却无法检验其它方面如电路能耗等的优劣。