16 位嵌入式 RISC 微处理器设计

雷少波，黄 民
（北京信息科技大学 机电工程学院，北京 100192）

摘 要：设计了一款具有 4 级流水线结构的 16 位 RISC 嵌入式微处理器。针对转移指令，未采用惯用的延迟转移技术，而是通过在取指阶段增加相应的硬件结构实现了无延迟转移。采用内部前推技术解决了指令执行过程中的数据相关。同时通过设置相应的硬件堆栈实现了对中断嵌套和调用嵌套的支持。整体系统结构采用 Verilog HDL 语言设计，指令系统较完善。在软件平台上的仿真验证初步表明了本设计的正确性。

关键词：微处理器；流水线；精简指令集；现场可编程门阵列；嵌入式系统

中图分类号：TP368.1 **文献标识码**：A **文章编号**：1674-7720(2013)07-0013-03

Design of 16 bit embedded RISC microprocessor

Lei Shaobo, Huang Min
(School of Mechanical & Electrical Engineering, Beijing Information Science & Technology University, Beijing 100192, China)

Abstract: Designed a four pipeline structure of the 16-bit RISC embedded microprocessor. For the branch instruction, did not adopt the delayed branch technology which is commonly used, but by adding the appropriate hardware structures to achieve branch without delay. Using the internal forwarding technology to solve data hazard during the instruction execution process. The microprocessor supported interrupts nesting, calls nesting by adding hardware stacks. The system was programmed by Verilog HDL, and has a relatively complete instruction system. Finally, simulation on the software platform indicated the correctness of the design preliminarily.

Key words: microprocessor; pipeline; RISC; FPGA; embedded system

随着微电子及 EDA 技术的高速发展，可编程逻辑器件的研发与应用也取得了长足的进步。其中，基于现场可编程门阵列(FPGA)的片上系统(System on Chip)在嵌入式系统中得到了广泛的应用。与传统的专用集成电路(ASIC)相比，其具有开发周期短、设计简单灵活等特点。

本文描述了基于 FPGA 的 16 bit 嵌入式微处理器的结构设计。该处理器采用程序存储器与数据存储器分离的哈佛型结构，采用精简指令集(RISC)，指令面向寄存器操作，加快了运行速度，简化了控制逻辑。

1 系统结构设计

该处理器执行指令时按照取指(IF)、译码(ID)、执行(EX)和结果保存(WR)4 个阶段依次进行。由于采用 4 级流水线结构，每个时钟周期能完成一条指令的执行。

1.1 指令系统

该处理器采用 RISC 型指令，设置了数据传送、算术逻辑运算和程序控制 3 大类共 21 条指令，指令格式固定，每条指令长度为 32 bit。

1.2 硬件结构

该处理器共有 4 级流水线，因此可将硬件基本划分为取指、译码、执行和结果保存 4 部分。

1.2.1 取指

取指部分结构如图 1 所示。与参考文献[1]中提及的处理器取指部分相比，其增加了 PC 选择生成器 G_PC，通过它实现了无延迟的程序转移指令。

G_PC 是本部分的核心，其部分 Verilog HDL 实现代码如下：

```
……
input  inta, z;                    //inta 为中断请求信号
                                   //z 为译码部分回送的信号，用于 jz、jnz 指令
input[4:0]p_op_code;               //指令操作码，即 inst[28:24]
output  status_up, status_down, epc_up, epc_down;
```

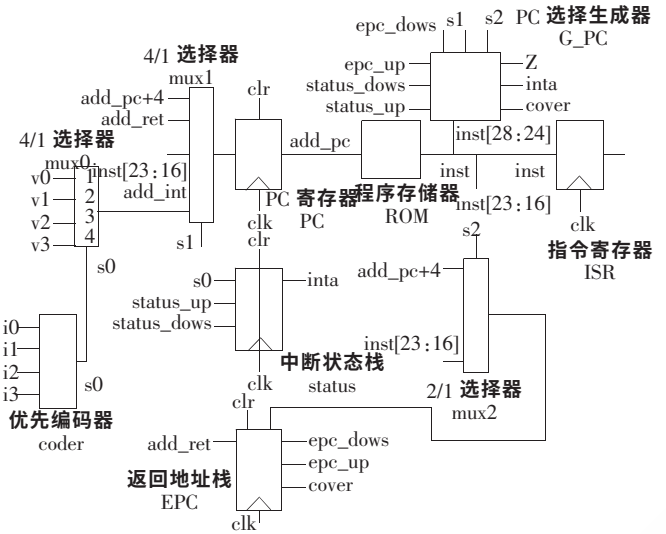

图 1 处理器取指部分结构

```
                    //对 status 和 EPC 进行出入栈控制
output  s2;   //对将要压入 EPC 中的返回地址进行选择
output  cover; //控制 EPC 顶部单元数据是否被 mux2 送
                                        来的数据覆盖
output[1:0]s1;    //s1 对下条指令的取指地址进行选择
reg  status_up, status_down, epc_up, epc_down, s2;
reg[1:0]  s1;
always  begin
case(p_op_code)
                    //根据指令的操作码来产生信号的输出
  5'b00000:begin    //此操作码对应 add  ra, rb, rc 指令
  if(inta)begin
       //如果有符合优先级条件的中断请求则 inta 为 1
          status_down=1;
                    //控制 status 在下个时钟沿将 s0 压栈
          status_up=0;
          epc_down=1;
         //控制 EPC 下个时钟沿将相应的返回地址压栈
          epc_up=0;
          s1=3;
           //选择中断入口地址 add_int 作下条指令地址
          s2=0;
       //选择当前指令地址加 4 作为返回地址压入 EPC
cover=0;                    //EPC 顶部数据不被覆盖
    end
  else  begin               //无符合优先级条件的中断请求
          status_down=0;
          status_up=0;
          epc_down=0;
          epc_up=0;
          s1=0;    //下条指令地址为当前指令地址加 4
          s2=1'bx;          //s2 为无关信号,0 或 1 均可
          cover=0;
    end
  end
  ……
  //其他指令类似,根据操作码及是否有中断请求来生成
相应的输出信号
  endcase
  end
```

优先编码器 coder 则从硬件电路上实现了中断源的优先级,即 3 号中断优先级最高,然后依次递减(3 号中断的优先级值为 4,2 号中断的优先级值为 3,1 号中断为 2,0 号中断为 1)。coder 的输出信号 s0 既代表了中断的优先级,又起到了选择中断入口地址的作用。

取指部分的具体工作流程如下。

(1) 假设此时指令地址 add_pc 在 ROM 中对应的指令 inst 为 sub r0,r1,r2, 此类语句不会使下一条指令地址发生转移。同时假设没有中断信号产生,于是当下一个时钟上升沿到达时, 由 G_PC 生成的控制信号 s1 将 add_pc+4 选通送入 PC 寄存器中, 即取出物理地址相邻的下一条指令(加 4 是因为一条指令有 32 bit, 共 4 B)。

假设 add_pc 在 ROM 中对应的指令 inst 为 sub r0, r1, r2, 且中断状态栈 status 顶部单元数据为 0。此时有中断信号 0 和中断信号 2 产生, 优先编码器 coder 生成的 s0 为 2 号中断的优先级 3 (大于 status 顶部数据 0),于是中断请求信号 inta 有效。当下一个时钟上升沿到达时,G_PC 生成的 s2 信号和 epc_down 信号将 add_pc+4 压入返回地址栈 EPC 中,s0 和由 G_PC 产生的信号 s1 将 2 号中断的入口地址 v2 送入 PC 寄存器中(0 号中断被忽略掉),同时将 s0 的值 3(即 2 号中断的优先级)压入中断状态栈 status 顶部。

(2) 假设此时 add_pc 在 ROM 中对应的指令 inst 为绝对跳转 jump 100, 即程序转移到地址 100 处开始执行,且没有中断信号产生。则当下一个时钟上升沿来临时,由 G_PC 产生的 s1 等控制信号将 inst[23:16](此时为 100)送入 PC 寄存器中开始执行。

假设此时 add_pc 在 ROM 中对应的指令 inst 为绝对跳转 jump 100,但此时有中断信号 1 产生(且假设此时 status 顶部值小于 2),则 inta 为 1。于是 G_PC 产生的控制信号将 inst[23:16](即 100)压入 EPC 顶,将中断 1 的优先级(即 s0 的值 2)压入 status 顶部,将中断 1 的入口地址送入 PC 寄存器中,开始响应中断。如果之后又产生优先级大于 2 的中断, 则将相关数据压栈后响应中断,若产生中断的优先级小于或等于 2,则被忽略不执行。

条件转移 jz、jnz 与 jump 指令类似,只是当译码阶段产生的 Z 信号为 1 时,jz 跳转,Z 为 0 时 jnz 跳转。通常比较指令 comp 后面紧跟条件转移指令来实现程序转移控制。

（3）假设此时 add_pc 在 ROM 中对应的指令 inst 为调用指令 call 100，执行此条指令时忽略所有中断请求信号，将 add_pc+4 压入 EPC 中后，将 inst [23:16]（即 100）送入 PC 寄存器中开始执行调用程序。

（4）假设此时 add_pc 在 ROM 中对应的指令 inst 为中断返回指令 int_ret，且没有高优先级的中断产生，则 EPC 顶部的数据 add_ret 送入 PC 寄存器中，同时，EPC 和 status 中的顶部数据弹出，其余数据依次上移一位。

假设此时 add_pc 在 ROM 中对应的指令 inst 为中断返回指令 int_ret，但有优先级比 status 顶部单元数据高的中断信号产生，则 G_PC 产生的 cover 信号有效，status 顶部数据被新的中断优先级值所覆盖，EPC 维持不变，同时响应新中断。

（5）调用返回指令 call_ret 与中断返回指令 int_ret 类似，不过执行时 status 中的数据不弹出。

1.2.2 译码

译码部分结构如图 2 所示。

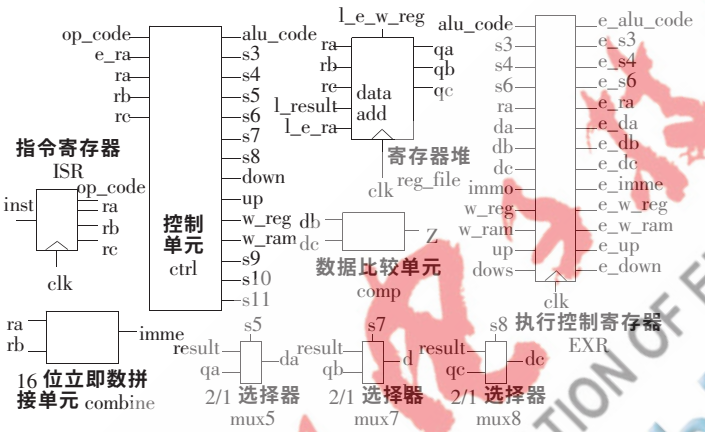

图 2 处理器译码部分结构

译码部分核心是控制单元 ctrl。为了解决流水线的数据相关，采用了内部前推的方法，将执行部分产生的数据 result 回送至本部分。将比较单元 comp 产生的信号 Z 送至译码部分，使得条件跳转指令（jz、jnz）在取指阶段就能实现，从而实现无延迟跳转。

本部分中的寄存器堆 reg_file 在时钟下降沿且写信号 l_e_w_reg 有效时执行写数据操作（实现结果保存 WR 这一部分的功能）。ctrl 产生的一部分控制信号通过执行控制寄存器 EXR 送至下一级使用。

1.2.3 执行

执行部分的结构如图 3 所示。此部分核心是算术逻辑运算单元 ALU，前面译码部分的 ctrl 产生的运算控制码 alu_code 指定运算操作，运算结果 c_out 送入 5/1 选择器 mux6。关于 mux6 的其余 4 路数据对应的指令为：dout 对应 load ra，imme 即将数据存储器中指定地址单元 M[imme]的数据送入 ra 号寄存器中；ds 对应 pop ra，即将数据堆栈 stake 栈顶的数据弹入 ra 号寄存器中；e_imme 对应指令 val ra，imme 送立即数 imme 入 ra 号寄存器；

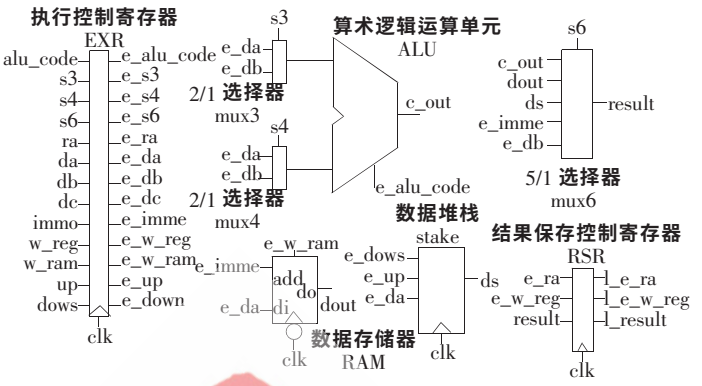

图 3 处理器执行部分结构

e_db 对应 mov ra，rb 即将 rb 号寄存器中的数据送入 ra 号寄存器中。

1.2.4 结果保存

这里的结果保存是针对目的地址为寄存器的指令，如算术逻辑运算指令、寄存器之间的数据传输指令等。工作流程即将图 3 中 RSR 的 l_e_ra、l_e_w_reg、l_result 信号送入图 2 中的 reg_file。当时钟下降沿到达且 l_e_w_reg 有效时，数据 l_result 被写入 l_e_ra 号寄存器中。

2 仿真验证

为了验证该设计，利用 Altera 公司的 Quartus II 软件进行仿真验证。仿真时设计在取地址为 32 的指令时出现 0 号中断，在取地址为 52 的指令时出现 1 号中断。实际执行时，1 号中断嵌套在 0 号中断之中。对应中断的入口地址分别为 48、68。仿真结果如图 4 所示。

图 4 仿真结果

图 4 中 a0~a6 分别显示的是 r0~r6 号寄存器中的数据；v0、v1 分别是 0 号中断、1 号中断的入口地址；pc_num 是处在取指阶段的指令的地址；ints 是中断输入信号，ints=1、ints=2 分别表示外设请求 1 号中断、外设请求 2 号中断。从图 4 中可以看出：

（1）几乎每条处在取指阶段的指令都要经过 3 个时钟上升沿和一个时钟下降沿后才执行完毕。这是因为指令取指完成后，还要经过译码、执行和结果保存 3 个阶段，并且结果保存是在时钟下降沿完成的。但由于是流水线结构，故等效于每一个时钟执行一条指令。

（2）当取到地址为 pc_num=32 的指令时，中断信号 1 产生，于是下条指令的取指地址为 1 号中断入口地址

48。执行 1 号中断的子程序到 52 时,中断信号 2 产生,于是响应 2 号中断,下条指令地址为 2 号中断入口地址 68(2 号中断子程序就一条返回指令)。

(3)当执行地址为 24 的 jump 0 指令时,下条指令地址为 0,实现了无延迟转移。

综上所述,经初步验证,该设计能实现 4 级流水线结构,并具备中断及其嵌套、无延迟转移等功能。

本文设计了一种基于 FPGA 的 16 bit 嵌入式 RISC 微处理器。该处理器主要特点是通过增加硬件结构实现了对转移指令的无延迟实现以及对中断及调用指令的支持。在下一步工作中将优化结构设计,增加外围设备,逐步构成一个高性能的单片系统。

参考文献

[1] 李亚民.计算机原理与设计[M].北京:清华大学出版社,2011.

[2] 郑纬民,汤志忠.计算机系统结构[M].北京:清华大学出版社,1998.

[3] 夏宇闻.Verilog 数字系统设计教程[M].北京:北京航空航天大学出版社,2008.

[4] 于洋,肖铁军,丁伟.面向教学的 16 位 CISC 微处理器的设计[J].计算机工程与设计,2010,31(16):3584-3587.

[5] 张英武, 袁国顺.32 位嵌入式 RISC 处理器的设计与实现[J].微电子学与计算机,2008,25(6):14-17.

[6] 曾舒婷,杨志家.高性能 PLC 专用指令集处理器设计与仿真[J].微电子学与计算机,2011,28(7):76-81.

(收稿日期:2012-10-15)

作者简介:

雷少波,男,1987 年生,硕士研究生,主要研究方向:机电一体化技术。