

基于 FPGA 的 16 位堆栈处理器的设计

储昭贤, 施慧彬

(南京航空航天大学 计算机科学与技术学院, 江苏 南京 210016)

摘 要: 设计了一款面向嵌入式控制领域的 16 位堆栈处理器, 该处理器包含两个堆栈: 执行数学表达式的数据堆栈和支持子程序调用的返回堆栈, 其指令集含 35 条堆栈指令。详细给出了该堆栈处理器的体系结构及设计方法; 不仅采用简单有效的指令编码方式缩小了代码体积, 同时给出了单周期操作多个堆栈元素的解决方法。该处理器采用 FPGA 实现, 在 XC5VLX110T 芯片上的运行时钟频率最高达到 146.7 MHz。最后给出了设计的软件仿真与硬件综合结果。

关键词: 堆栈处理器; 嵌入式; FPGA

中图分类号: TP302

文献标识码: A

文章编号: 1000-7180(2012)02-0022-05

Design of FPGA-based 16-bit Stack Processor

CHU Zhao-xian, SHI Hui-bin

(College of Computer Science and Technology, Nanjing University of Aeronautics & Astronautics, Nanjing 210016, China)

Abstract: This paper introduces a 16-bit stack processor for embedded control. The stack processor has two stacks: a data stack for evaluating mathematical expression and a return stack for calling subroutine. The instruction set includes 35 stack instructions. The architecture and design of the stack processor are presented. Not only a simple and effective instruction encoding is adopted to reduce the code size, but also the method to operate multiple stack elements in a single cycle is proposed. The processor is implemented by FPGA. Finally, the results of software simulation and hardware synthesizing are presented, and it comes out that the processor is able to run up to 146.7MHz on XC5VLX110T.

Key words: Stack Processor; Embedded; FPGA

1 引言

Forth 是由 Charles H. Moore 在 1960 年代发明的一种基于堆栈、可扩展、具有简单哲学思想的计算机编程语言^[1], 特别适合于软件代码在千行数量级的中规模嵌入式系统中应用, 并且已经被国外广泛应用于天文、军事、航空航天、工业自动化、图形、仪器仪表等领域。

Forth 语言既可以被看作汇编语言又可以被看作高级语言, 它与传统语言最大的区别在于它是基于堆栈的和可扩展性。Forth 语言本质上定义了一种双堆栈体系结构。这种体系结构的主要思想是基

于两个不同的堆栈, 一个是用来执行数学表达式的数据堆栈, 另一个是用来支持子程序调用的, 即保存子程序返回地址的返回堆栈, 指令的所有操作都是针对这两个堆栈的一个或者几个栈顶元素^[2]。

Forth 语言定义的这种双堆栈体系结构清晰明了, 复杂度低, 并且其主要面向的是嵌入式控制领域。在这种背景下, 结合 FPGA 的灵活性, 本文设计并实现了一种基于 FPGA 的 16 位堆栈处理器, 相比基于 RISC 体系结构的嵌入式处理器, 具有以下几点优势:

① 很大程度上避免了处理器进行上下文切换带来的开销, 因为处理器的运行不依赖于大量的通

收稿日期: 2011-05-13; 修回日期: 2011-07-27

基金项目: 南京航空航天大学引进人才科研启动基金(S1028-042)

用寄存器;

② 处理器的寻址方式非常简单,几乎所有指令都是 0 操作数指令.这样不仅系统复杂度显著降低,速度得到提升,代码体积也大大减小;

③ 处理器在运行具有深度嵌套特征的程序时有更加明显的优势,因为具有专门的硬件堆栈来执行子程序调用与返回.

2 设计与实现

2.1 堆栈处理器体系结构概览

处理器的结构如图 1 所示,为了能够清晰地展示其结构,部分信号并未显示或直接连接.处理器包含的主要模块见表 1.

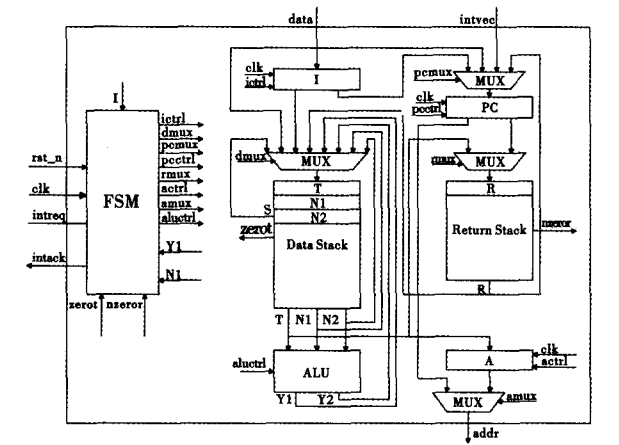


图 1 堆栈处理器结构图

表 1 堆栈处理器的主要模块

名称	描述
Data Stack	数据堆栈
Return Stack	返回堆栈
PC	程序计数器
I	指令寄存器
A	地址寄存器
ALU	算术逻辑单元
FSM	有限状态机,即控制模块

需要说明的是,表中的数据堆栈由栈顶寄存器 T、次栈顶寄存器 N1、第三栈顶寄存器 N2 以及深度为 32 的堆栈存储器构成;返回堆栈由栈顶寄存器 R 以及深度为 32 的堆栈存储器构成.

2.2 指令集

2.2.1 指令集设计

处理器实现了四种类型共 35 条指令,具体如下

表 2 所示.

表 2 堆栈处理器的指令集

指令类型	指令名称
堆栈操作	nop, swap, one, zero, rot, >r, drop, nip, r>, over, dup
数学运算	or, xor, and, +, lplus, mpp, asr, lsr, lsl, -, lmin, shld
访 存	litc, c@, c!, lit, @, !
程序转移	call, ret, jmp, jz, jnc, drine

表中大部分指令遵从了 Forth 语言的原语命名规则与功能.比如,其中的‘>r’表示将数据堆栈栈顶寄存器 T 的内容弹出到返回堆栈栈顶寄存器 R, ‘r>’则表示相反的功能.另外,‘@’表示从存储器读数据到数据堆栈,而‘!’表示将数据堆栈的内容存储到存储器中.下面分别简述各种类型指令的功能.

① 堆栈操作:主要是对数据堆栈或者返回堆栈的一个或者多个栈顶元素的操作.例如,swap 指令交换 T 与 N1 的内容;one 和 zero 指令分别将 T 的所有 bit 位置 1 和 0;drop 指令弹出 T 中内容并抛弃.所有的堆栈操作类型指令都在一个周期内完成.

② 数学运算:数学运算指令的操作数均为 T 和 N1 中内容.逻辑运算指令包括 or、xor 和 and;移位指令包括 asr、lsr 和 lsl;lplus 和 lmin 分别将数据堆栈栈顶元素加 1 和减 1.所有数学运算类型指令都在一个周期内完成.

为了节省硬件资源,堆栈处理器没有实现硬件支持的乘法器^[3]和除法器^[4],因此没有直接的乘法和除法指令,取而代之的是 mpp 和 shld 指令.这两条指令分别完成了乘法器和除法器在一个时钟周期中完成的操作.mpp 指令执行时需要同时操作寄存器 T、N1 和 N2. N1 与 N2 中分别存放初始的乘数与被乘数,T 的初始值为 0;最终乘积结果的高低 16 位分别存放于 T 与 N1. mpp 指令的执行分成两步.第一步判断 N1 最低位是否为 1;如果为 1,将 N2 与 T 中内容相加;否则不做操作.第二步将 T 做高位、N1 做低位,对其内容进行逻辑右移.这样连续执行完 16 条 mpp 指令后,T 与 N1 中存放的分别为乘积的高低 16 位.这 16 条 mpp 指令再配合其他几条指令便可以完成正常的乘法操作,完整的乘法操作耗费 19 个时钟周期.除法的实现与乘法类似,完整的除法操作耗费 21 个时钟周期.下面给出了乘法操作的程序示例,其详细解释将在 3.1 节的仿真分析中给出:

```
litc 0 //将 0 压入堆栈
mpp //第 1 条 mpp 指令
...
mpp //第 16 条 mpp 指令
rot //乘法结束后,清理掉
drop //位于 N2 的被乘数
```

③ 访存:堆栈处理器支持按字节和按字两种访存方式.指令集中的 `litc`、`c@`和 `c!` 指令为字节访存指令;而 `lit`、`@`和 `!` 为按字访存指令.这里简要说明按字访存指令的功能,按字节访存与之类似.`lit` 将一个字常量压入数据堆栈;`@`将 `T` 的内容弹出送入地址寄存器 `A`,按 `A` 中地址访存取得数据后压入数据堆栈;`!` 指令将 `T` 中的内容弹出送入地址寄存器 `A`,再将当前栈顶 `T` 的数据弹出存入 `A` 中地址.除 `litc` 指令的其他指令需要两个周期执行,`litc` 指令在设计时被优化为不需要访存,因此只需要一个周期执行.

④ 程序转移:`call` 为子程序调用指令,它将当前 `PC` 寄存器值压入返回堆栈,同时将子程序地址送入 `PC` 寄存器;`ret` 为子程序返回指令,它将返回堆栈的栈顶值弹出,送入 `PC` 寄存器;`jmp` 为无条件跳转指令;`jz` 和 `jc` 为有条件跳转指令,跳转条件分别是数据堆栈栈顶 `T` 的内容是否为 0 和 `ALU` 进位标志位是否为 0;`drjne` 将返回堆栈栈顶 `R` 的内容减 1,然后判断 `R` 中内容是否为 0,如果不为 0 则跳转.除 `call` 与 `ret` 以外的其他指令都需要先访存取得 16 位跳转地址,然后压入堆栈,因此需要两个周期执行;`ret` 与 `call` 指令只需要一个周期执行.

2.2.2 指令集编码优化

存储器是嵌入式控制系统中的一种稀缺资源,因此在设计堆栈处理器时,采用了一种简单但是有效的编码方式进行指令编码,不仅大大减小了代码体积,还在一定程度上可以降低系统的复杂度,提高系统效率.

在堆栈处理器上执行的程序一般都通过大量子程序调用进行模块化设计以减小代码尺寸,这样才能充分利用硬件支持的返回堆栈.因此,在典型的堆栈处理器程序中 25% 的时间花费在子程序调用上^[6],这就要求在实现堆栈处理器时必须高效地实现 `call` 指令.一方面,如果 `call` 指令占用的 bit 位较少,便能减小代码体积;另外一方面,如果 `call` 指令的执行能在一个周期完成,将大大加快程序的执行速度.因此指令集采用了两种不同的编码方式实现.

对于除 `call` 以外的其他指令,每条指令占 8 位

且最高位均为 0.处理器每次访存取出的 16 位数据中,高低 8 位各为一条指令.采用这种设计方案后,一方面,每次访存可以取出两条指令.这在处理器和存储器之间形成了一个处理器速度两倍于存储器速度的缓冲,可以在一定程度上避免 `Cache` 的引入,降低嵌入式系统的复杂度;另一方面,每条指令占用 8 位而不是 16 位可以大大减小代码体积.

`call` 指令采用了另外一种实现方式,如果访存取出的 16 位数据中最高位为 1,那么这 16 位不再被解释为两条指令,最高位被解释为 `call` 指令,剩下的 15 位经逻辑左移 1 位后作为子程序地址.相对于采用与其他类型指令相同的 8 位实现方式,这种实现方式使得 `call` 指令仅占 1 位,可以进一步减小代码体积.

另外,`call` 指令执行时的子程序地址不需要通过访存取得,因此其执行将只耗费一个周期.当然,这样的方案会让 `call` 指令只能调用位于偶地址的子程序,但是好的设计来源于适当的折中,这种损失相对于获得的效率和性能提升是值得的.

2.3 控制模块

控制器本质上是如图 2 所示的一个五状态的有限状态机. `InstrF` 为其初始状态,即取指令状态. `Slot0` 和 `Slot1` 为执行状态, `Slot0_M` 和 `Slot1_M` 为访存状态. `Slot0` 与 `Slot0_M` 对应于低 8 位指令地执行与访存, `Slot1` 与 `Slot1_M` 对应于高 8 位指令地执行与访存.

处理器在 `InstrF` 状态访存取出 16 位数据后进入 `Slot0` 状态,先判断数据的最高位是否为 1;若为 1,则最高位被解释为 `call` 指令,低 15 位经逻辑左移 1 位后作为子程序地址,执行子程序调用后下一状态进入 `InstrF`;若为 0,则执行低 8 位指令.如果低 8 位指令为单周期指令,下一状态进入 `Slot1` 执行高 8 位指令;如果低 8 位指令为双周期指令,下一状态进入 `Slot0_M` 进行访存,访存结束后下一状态进入 `Slot1` 执行高 8 位指令.

处理器在进入 `Slot1` 状态后,执行高 8 位指令.如果高 8 位指令为单周期指令,下一状态进入 `InstrF` 继续取指令;如果高 8 位为双周期指令,下一状态进入 `Slot1_M` 访存,访存结束后下一状态进入 `InstrF` 继续取指令.

在 `InstrF` 状态可以响应外部中断.如果此时检测到中断请求信号 `intreq` 为高电平,则给出中断响应信号 `intack`,并且将中断向量 `intvec` 送入 `PC`,下一状态依然为 `InstrF` 状态.

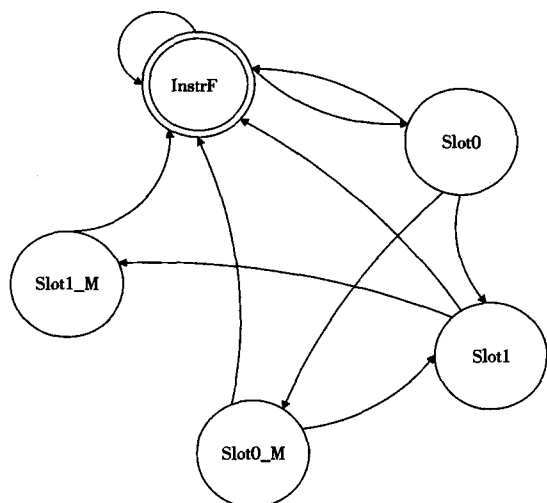


图2 有限状态机

2.4 堆栈实现

本文设计的堆栈处理器包含两个硬件支持的堆栈:数据堆栈和返回堆栈。由于所有指令都是对这两个堆栈的操作,因此堆栈的实现方式对于堆栈处理器的效率和性能是至关重要的。堆栈的实现不仅要最大程度上避免溢出,还要保证在访问时能够非常高效。

2.4.1 堆栈溢出问题

关于硬件堆栈溢出问题存在多种解决方法,主要有:请求式单元元素堆栈管理器、页式堆栈管理以及联合Cache等。其实Charles H. Moore曾经做过统计,在典型的堆栈程序执行过程中,当堆栈深度超过32时,几乎所有的程序都不会发生堆栈溢出问题。因此堆栈处理器WISC和EP32直接将堆栈深度设置为256,以求完全避免堆栈溢出问题。

相对于采用其他硬件方式来避免溢出问题,直接增加堆栈深度更加简单有效,因此本文论述的处理器直接将堆栈存储器深度设置为32(不包括栈顶寄存器在内)。这对于一般的应用已经足够,由于是采用FPGA实现,在需要的情况下可以很方便地将

堆栈深度增加。

2.4.2 指令的单周期堆栈操作

指令集中的很多指令,如swap和rot指令,需要同时操作堆栈的两个甚至三个栈顶元素。如果采用普通的堆栈存储器,想同时访问栈顶的多个元素,就需要访问堆栈多次,这样那些需要操作多个栈顶元素的指令在执行时需要耗费多个周期。这对于堆栈处理器将是极大的性能损失。

因此,为了保证所有指令的堆栈操作都能够在单周期内完成,在堆栈设计上采用了图1中所示的栈顶寄存器加堆栈存储器的方式。对于数据堆栈,由于在最多的情况下(rot指令的执行)需要同时访问栈顶的三个元素,因此设置了三个栈顶寄存器T、N1和N2。这样数据堆栈的真正栈顶元素就是T,而不是堆栈存储器的栈顶元素。返回堆栈的设计与此类似,设置了一个栈顶寄存器R。

3 仿真与综合

本文设计采用Verilog进行RTL级描述,仿真软件为ModelSim SE 6.5C,综合软件为Synplify Pro9.6.1,目标芯片为Xilinx公司的Virtex-5 XC5VLX110T。

3.1 仿真分析

图3中给出了2.2.1节中的乘法程序的仿真波形图,图中state与nextstate信号给出了有限状态机的状态转移情况;instr_exec表示当前正在执行的指令;I与PC则分别为指令寄存器与程序计数器;T、N1和N2为数据堆栈的三个栈顶寄存器;result表示ALU的运算结果。

从图中可以看出,在70~110 ns,执行了两条lit指令(指令编码为40H),这两条指令将乘数与被乘数003H与0011H压入数据堆栈。在这之后开始执行2.2.1节中给出的乘法运算程序。在120~130 ns,执行了一条litc(指令编码为50H)指令,将00H扩

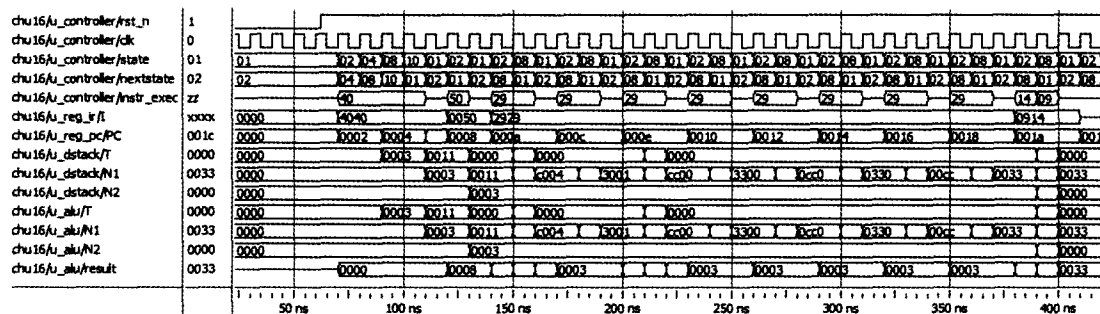


图3 堆栈处理器乘法操作仿真波形

展为 0000H 压入数据堆栈。此时,可以查看堆栈寄存器 T、N1 和 N2 里的内容分别为 0000H、0011H 和 0003H。在 140~370 ns 期间连续执行了 16 条 mpp 指令(指令编码为 29H),接着在 380~400 ns 期间又执行了 rot 指令和 drop 指令完成在 mpp 指令操作后的一些堆栈清理操作。最后,在第 400 ns 时查看数据堆栈的栈顶寄存器 T 与 N1,发现内容分别为 0000H 与 0033H,其中 0000H 为乘法结果的高 16 位,而 0033H 则为低 16 位,与预期结果一致。

3.2 综合分析

使用 Xilinx 公司的 XC5VLX110T 作为目标芯片时,片上资源利用率不超过 3%,堆栈处理器的最高频率可以达到 146.7 MHz。表 3 中给出了本设计在不同芯片上与不同文献中同类设计进行的主频性能对比。从表中数据可以看出,本设计在主频性能上与前三种同类设计相比有较明显的提升。值得注意的是,与文献[8]中的 MSL16 处理器相比较而言,本文处理器主频略有下降,这是因为 MSL16 处理器在设计中应用了两级流水线技术。本文则侧重于运用较简单的设计、较少的硬件资源以获得较高的性能:一方面,流水线技术在堆栈处理器中的应用无疑会大幅增加堆栈处理器的复杂度,这与本文的设计初衷不相符;另一方面,从性能提升率上看,流水线技术的应用所带来的频率增加并非异常显著。

表 3 主频性能对比

器件名称	其他同类设计/ MHz	本文设计/ MHz	性能提升率/ %
XC5VLX110T	—	146.7	—
Spartan-II E	25 ^[5]	55.9	123.6
FLEX10K30E	25 ^[6]	42.6	70.4
Spartan-III	50 ^[7]	73.5	47
XC4000E	33 ^[8]	30.2	-8.5

4 结束语

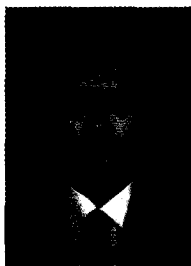
本文以双堆栈体系结构为基础,设计了一种不同于 RISC 体系结构的嵌入式堆栈处理器。该处理器结构紧凑,系统复杂度低,代码体积小。在 Xilinx

公司的 XUPV5-LX110T 开发板上实现后,其主频达到了 146.7 MHz,片内资源使用率不超过 3%。仿真与综合结果证明了设计的正确性,与同类设计相比较,主频性能有较为明显的提升。堆栈处理器较高的主频与较低的资源使用率为后期以该处理器为核心构建一个 SoPC 提供了基础。

参考文献:

- [1] Leo Brodie. Thinking forth [M]. USA: Punchy PUB, 2004: 25-26.
- [2] Shi H, Bailey C. Investigating available instruction level parallelism for stack based machine architectures [C]//Proceedings of Euromicro Symposium on Digital Systems Design. Rennes, France, 2004: 112-120.
- [3] 李楠,喻明艳. 16X16 快速乘法器的设计与实现[J]. 微电子学与计算机,2008,25(4):156-159.
- [4] 宣淑巍,李晓江,马成炎. 一种基于循环减法原理除法器的加速方法[J]. 微电子学与计算机,2009,26(12): 12-15.
- [5] Richard E Haskell. A VHDL-Forth core for FPGAs [J]. Microprocessors and Microsystems, 2004, 28 (3): 115-125.
- [6] Bernd Paysan. b16-A forth processor in an FPGA [EB/OL]. [2010-07-20]. <http://www.jwdt.com/~paysan/b16.html>.
- [7] James Bowman. The J1 forth CPU [EB/OL]. [2010-07-21]. <http://excamera.com/sphinx/fpga-j1.html>.
- [8] Leong P H W, Tsang P K, Lee T K. A FPGA based Forth microprocessor[C]//Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines. Napa Valley, California, 1998: 254-255.

作者简介:



储昭贤 男,(1988—),硕士研究生。研究方向为计算机系统结构。

施慧彬 男,(1966—),博士,副教授。研究方向为计算机系统结构、可重构计算、编译优化技术等。