一种高速的条件跳转指令硬件实现

朱俊华
(上海交通大学 微电子学院,上海 200030)

摘 要:条件跳转指令是处理器常用的一种指令,循环是条件跳转指令应用的主要领域之一。条件跳转指令是否能够高效实现,制约了循环的效率。而循环寻址的引入又进一步增加了该类指令实现的复杂性。针对一种广泛应用于循环的条件跳转指令的特性,讨论了该特性的布尔代数的数学性质,基于该数学特性提出一种高速的条件判断逻辑结构,并针对循环寻址的特性对该结构进行扩展。实验结果表明该结构具有较高的效率以及结构上的稳定性。

关键词:条件跳转;零检测;循环寻址;并行结构
中图分类号:TP302 **文献标识码**:A **文章编号**:1673-629X(2008)07-0005-03

A Fast Hardware Implementation of Conditional Branch Instruction

ZHU Jun-hua
(School of Micro-electronics, Shanghai Jiaotong University, Shanghai 200030, China)

Abstract: Conditional branch instruction is a common instruction of processor, and mainly applied on the implementation of loop executions. The implementation of this instruction greatly impacts on the efficiency of the loop executions. The introduction of circular addressing further complicates the design. Discusses the boolean mathematic property of conditional branch instruction according to their features in loop applications. It also presents a novel hi h-speed architecture, extending it to accommodate circular addressing feature. The experiment shows the great improvement and stabilization of this architecture.

Key words: conditional branch; zero checking; circular addressing; parallel structure

0 引 言

数字信号处理芯片(DSP)是针对数字信号处理应用开发的进行数字信号处理的芯片,在数字信号处理领域比一般的通用处理器有更高的效能,在更低功耗、更高效率方面都要优于通用处理器。而条件跳转指令无论是在DSP中还是在通用处理器中都有广泛的应用。能否高效执行此类指令制约着处理器的性能。

在DSP中存在一种特殊的条件跳转指令,该指令以判断地址指针是否为零作为跳转条件,而地址指针的给出往往需要经过地址运算。该指令被大量应用于与循环相关的DSP应用程序中[1]。由于需要对地址进行运算,加上DSP访存方式的复杂性,如果采用常规的方法加以实现,将制约处理器频率的提升。

文中讨论常规方法实现该指令所带来的问题,讨论此类指令的特性,在此基础上提出一种高效的结构,并针对循环寻址的应用对该结构扩展。实验结果表明,该结构取得性能上的较大提升。

收稿日期:2007-10-14
基金项目:国家863计划资助项目(2003AAIZ)
作者简介:朱俊华(1982-),男,上海人,硕士研究生,主要研究方向为计算机体系结构。

1 设计思路

条件跳转指令中地址指针的运算主要是将一个基址加上或减去一个偏移量,结果作为新的地址。而在处理器中减法能够规约为加法运算[2]。因此对于此类特殊的基于地址指针是否为零的条件跳转指令,其跳转结果取决于两个地址操作数相加的结果是否为零。

1.1 常规电路

常规方法判断两个数相加后的值是否为零,实现时,基本采用在加法器的输出部分加上一个收缩阵列,以tsmc 0.18μm工艺提供的标准单元,位宽16 bit为例,该收缩阵列由4层构成,第一层是8个或非门(NOR2x4)的标准单元,以加法器的16个输出,两个一组作为其输入。第二层是4个与非门(NAND2x2),以第一层的8个输出,两个一组作为其输入。第三层由2个或非门(NOR2x2)构成,以第二层4个输出,两个一组作为输入。最后一层是一个与非门(NAND2x1),输出0检测的结果[3]。

在不带wire load model的情况下,延时为0.29ns。

该结构已经充分利用了标准单元的逻辑特性与时延特性,以树状结构收缩出结果,是采用该标准单元库的最优结构。但是这种方法,需要等待加法器的输出结果。一个 16bit 的 Brent – Kung 结构的加法器[4],在不带 wire load model 的情况下,延时为 0.94ns,加上前面的一个 wallace 结构[5]的 4 – 2 压缩器(该压缩器用于支持 4 个源操作数的地址运算,将 4 个操作数压缩为 2 个),一组数据选通阵列以及程序控制单元的处理逻辑时延,成为整个流水线中一条关键路径(critical path)。

而如果将操作分两级完成,需要再加入一个加法器专门完成 0 检测所需的加法运算以及相应的一组数据锁存(flip flop),同时该指令的完成时间将延长一个周期,无论是效率还是功耗都无法满足要求。

1.2 运算性质

如何将零检测与加法部分同时进行,在地址运算的过程中就判断出结果是否为零,成为提高速度的关键。从分析 $A + B = C$ 这个公式入手,这里假设 A, B, C 都为两位,分别为 a_1a_2, b_1b_2, c_1c_2。那么令 C 为零的(A,B) 组合如下表:

	A	B
1	00	00
2	01	11
3	10	10
4	11	01

这里可以发现对于 C 中的高位,如 c_2,是否为零,除了需要判断当前参加运算的 a_2, b_2 的结果是否为零(如行 1,3),还要等待低位的进位(如行 2,4)。而低位的进位则成为限制判断速度的关键路径。消除对低位结果的依赖,就消除了这条关键路径。

通过对 $A + B = 0$ 进行如下的变换:

$$A + B = 0 \tag{1}$$

$$\Leftrightarrow -A - B = 0 \tag{2}$$

$$\Leftrightarrow \overline{A} + 1 + \overline{B} + 1 = 0 \tag{3}$$

$$\Leftrightarrow \overline{A} + 1 + \overline{B} = 11 \tag{4}$$

判断 $A + B$ 是否为 0,就转换为判断 $\overline{A} + 1 + \overline{B}$ 是否为全 1。这里先讨论 $\overline{A} + \overline{B}$ 是否为 11 的情况。与直接判断 $A + B = 0$ 相比,判断 $\overline{A} + \overline{B} = 11$ 的优势在于,c_i 位为 1 这个结果是否受到 c_{i-1} 的影响,是否需要等待 c_{i-1} 的运算结果。如果在 c_{i-1} 运算前发现 $a_{i-1} + b_{i-1}$ 不会出现进位,而在 c_{i-1} 运算后产生了进位,此时 c_{i-1} 一定为 0,而这个进位位是从 $c_{i-2}..0$ 各位传递上来的,此时反推,必然有一位 c_j 的运算结果为 0,且该位没有得到低位的进位。这样 $\overline{A} + \overline{B}$ 一定不等于全 1。若在 c_i 运算前 $a_{i-1} + b_{i-1}$ 产生进位,那么无论运算后 c_{i-2} 是否产生进位,都不会出现 c_{i-1} 向 c_i 的进位。根据这一性质,就消除了对低位结果的依赖。

于是根据半加器的特性,令 $G = \overline{A}\ \&\ \overline{B}$, $P = \overline{A} \wedge \overline{B}$。

$$\overline{A} + \overline{B} = G + (P << 1) \tag{5}$$

判断$(\overline{A} + \overline{B})_i$ 是否为 1 只需要判断 $G_i \wedge (P << 1)_i$ 的结果是否为 1。

1.3 改进电路

现在考虑 $\overline{A} + 1 + \overline{B} = 11$ 这个最终式。可以对(5)式进行扩展,得到如下表达式:

$$\overline{A} + \overline{B} + 1 = G + (P << 1) \mid 1 \tag{6}$$

也就是说,令 $L = (P << 1)$,经过扩展,L_0 由 0 变为 1,也就是 L_{-1} 永远向 L_0 进位。同样利用(5)的算法判断出 $\overline{A} + 1 + \overline{B}$ 是否为全 1。16bit 的判零逻辑示意图见图 1,该逻辑与加法器处于并行的位置。图中,A, B 为进行地址运算的两个操作数,Z 为判零结果。

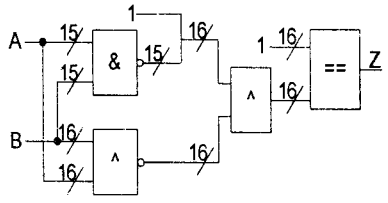

图 1 零检测逻辑图

2 逻辑改进

由于地址处理方式的复杂性,该判零逻辑还需要对循环寻址进行处理。循环寻址是指程序员确定高位地址与缓冲大小,将预定值写入特定的基址与大小寄存器,在地址运算时,硬件将对该缓冲以循环队列的方式对待。即当地址上溢或下溢出缓冲大小后,程序员无须对指针边界进行判断处理,转而由硬件自动完成指针调整。例如程序员定义基址为 0xFF00h,缓冲的大小为 16 个字,假设存储器以字编址,那么缓冲的地址上界为 0xFF0Fh。此时若一条指令的地址计算结果为 0xFF10h,那么硬件将自动调整该地址为 0xFF00h,这样就节省了一条条件语句加上一条算术逻辑指令的开销,极大提高了循环队列的执行效率。

然而由于缓冲大小不是固定的,因此地址运算单元的计算结果分为两个部分,一部分是基址部分,另一部分是缓冲内的偏移。地址运算单元的结果需要根据缓冲大小进行调整。如果按照前面的常规方法,则在关键路径上,又增加了一层 2 选 1 的选通逻辑。这进一步恶化了判零逻辑上的时延。因此需要对这一部分逻辑进行优化。可以对上面的判零逻辑进行扩展。首先根据缓冲大小生成两组掩码。这两组掩码为互反关

系,例如,缓冲大小为2,地址位为8 bit,因此生成两组掩码分别为

00000011　　(7)

11111100　　(8)

其中掩码(8)用于抽取基址部分,将抽取出的基址利用普通的判零电路进行结果判零。而掩码(7)作用于偏移部分,偏移部分的运算是现在的关键路径,因此应用改进的判零电路,将掩码(7)则作用于P与G。

普通判零电路与改进判零电路见图2、图3。

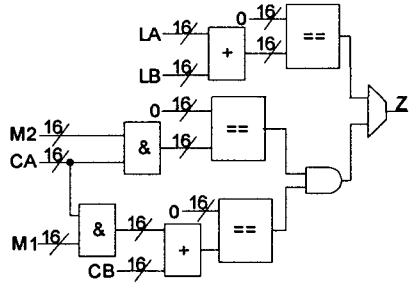

图2　普通判零电路逻辑图

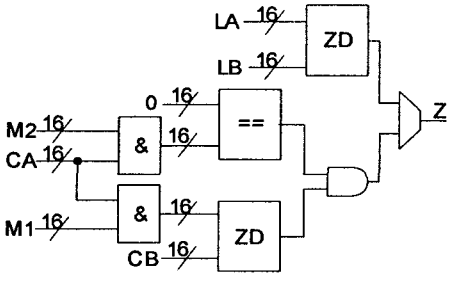

图3　改进判零电路逻辑图

图3中的ZD就是图1中的零检测电路。图2,3中的LA,LB分别指进行线性寻址时需要运算的两个地址操作数,CA,CB为循环寻址时需要运算的两个地址操作数,M1,M2为两个掩码。

3　实验结果

表1是两种判零电路在不同位宽下的时延比较结果。判零电路使用VerilogHDL语言编写的100%可综合的代码建模,Synopsys公司VCS验证通过,综合工具为Synopsys公司的Design Compiler[6],标准单元库为TSMC 0.18μm 1P6M Generic[7],环境条件为Worst Case(125°C,1.62V)。

表1　时延比较结果表(ns)

位宽(bit)	常规电路	改进电路
16	1.23	0.7
24	1.41	0.71
32	1.45	0.74
48	1.70	0.83
64	1.81	0.83

从表1中,可以看到,改进电路的时延随位宽的增加只有微量的延长,这是因为判1部分层数由于位宽的加深而增长,而增长量与位宽成logN的关系。常规电路,16位与64位的时延相差0.58ns,而改进电路,两者仅有0.13ns的差别。而常规电路的加法器如果采用非并行前缀[4]结构,时延将进一步恶化。

图4是效率提升结果,横坐标为位宽,纵坐标为百分比。从图4中可以发现位宽越大,改进电路效率提升越明显,在超过32位的地址位宽部分,改进电路有一倍以上的性能提升。

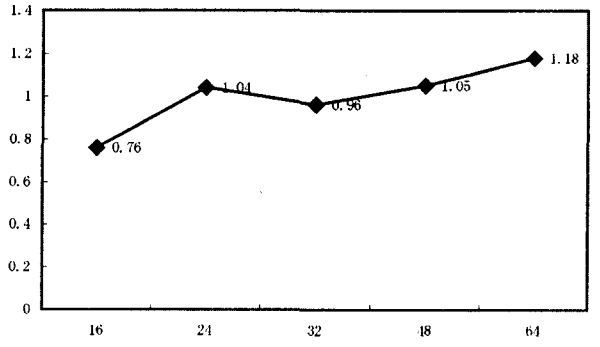

图4　效率提升结果

4　结束语

提出一种应用于条件跳转指令的条件判断电路的快速实现,相对于普通的电路实现,取得较高的性能提升。从结构的讨论上,可以发现,该判零电路具有非常规整的结构,非常利于手工版图设计。而手工版图设计能够进一步提升该逻辑的速度,降低面积的开销。

参考文献:

[1] Araujo G, Sudarsanam A. Instruction Set Design and Optimizations for Address Computation in DSP Architectures [C]//Proceedings of the 9th International Symposium on System Synthesis table of contents 105. La Jolla: [s. n.], 1996.

[2] Gebotys C H. A Minimum - Cost Circulation Approach to DSP Address - Code Generation[M]. China: Pearson Education, 2004.

[3] Rabaey J M. Digital Integrated Circuits: A Design Perspective [M]. China: Pearson Education, 2004.

[4] Beaumont - Smith A. Parallel Prefix Adder Design[C]//15th IEEE Symposium on Computer Arithmetic. USA: [s. n.], 2001.

[5] Araujo G, Ottoni G, Cintra M H. Global array reference allocation[M]//Trans. on Design Automation of Electronic Systems. USA: ACM, 2002.

[6] Synopsys Design Compiler User Guide[S]. Synopsys Etc. USA, 2004.

[7] Synopsys TSMC CL018G(0.18um Generic process)datasheet [S]. Synopsys Etc. USA, 2003.