

一种基于多级流水线加法器的累加电路设计研究

袁松^a, 唐敬友^b, 刘莉^a

(西南科技大学 a. 理学院; b. 国防学院, 四川 绵阳 621010)

摘要:专用硬件电路常用来实现加速, 以提升科学计算速度。在科学计算中, 多个数据的累加是常见运算。在设计硬件累加器时, 容易出现流水线阻塞问题。提出将数据依据流水线级次分成两类模块, 不同类型的模块采用不同的累加方式。基于多级流水线加法器, 在 FPGA 上实现了多个数据的累加。该设计消耗资源少, 流水线利用率高, 控制相对简单, 尤其是在数据规模很大时, 优势尤其明显。

关键词:硬件加速; FPGA; 多级流水线; 累加器

中图分类号: TN79⁺1

文献标志码: A

引言

随着集成电路工艺的进步, FPGA 的性能得到了显著的提升, 它为用户提供了更多的资源和更高的速度, FPGA 作为一种硬件可重构器件, 为现代数字系统设计提供一个良好的开发平台。由于 FPGA 硬件资源增加, 其处理浮点数的功能逐渐增强, 越来越多的国内外研究者将其用于科学计算以提高计算的速度。矩阵运算是科学计算的重要组成部分, 近年来 FPGA 常用于实现矩阵运算加速^[1-9]。

在矩阵运算中, 经常遇到多个数的累加。比如, 矩阵乘积、线性方程组的数值解等都会有累加运算, 因此累加电路的设计十分重要。对此, 大多数设计均在浮点数加法器的输出端引入反馈到输入端, 再加上其它附加电路(多路选择器, 缓存等)来实现累加运算。由于浮点数加法器采用了多级流水线(一般为 7~14 级), 有可能出现流水线阻塞。若设计不合理, 会大大增加缓存的使用量及控制器的设计难度, 最终降低设计的性能, 故实现速度与资源消耗的折中, 是累加器模块设计的难点。

因此, 累加电路的设计研究对于实现高速科学计算有重要的意义。

1 问题的数学定义

给定 N 个数据串, 分别用 A, B, C, \dots 表示; 每个串含有 m ($m \geq 2$) 个元素, 每个元素分别用下标 $1, 2, 3, \dots, m$ 标识, 例如, 串 A 中的元素就表示为 $A_1, A_2, A_3, \dots, A_m$ 。若用 R_i ($i=1, 2, 3, \dots, N$) 表示串内各元素的累加之和, 要实现 m 个元素相加, 则该累加方式的数学定义为

$$R_1 = \sum_{i=1}^m A_i, R_2 = \sum_{i=1}^m B_i, R_3 = \sum_{i=1}^m C_i \dots$$

2 相关研究现状

早在 1983 年, Lionel M. Ni 等^[10]就提出了基于多级流水线硬件的累加结构, 对单串及多串数据的累加进行了一般性的讨论, 提出了通用的硬件结构及算法。

Ling Zhuo 等^[11-13]对矩阵运算的硬件加速进行了全面系统的研究。他们在文献[11]中提出了能实现任意规模数据累加的硬件结构, 而不会使流水线暂停。

Qingzeng Song 等^[14]使用一个加法器, 四个 buffer 和四个多路选择器实现了对单串数据的累加。

3 本文方案设计

以前的设计大多让数据串中每个元素顺序进入累加器, 为了使流水线不发生阻塞, 必须使用大量的缓存

(buffer)。各串元素及串内元素是相互独立的,不存在先后顺序,可以将其交换,这为充分利用流水线提供了可能。为了充分利用流水线而又不消耗大量的缓存资源,本设计采用该方式与另一方式相结合的策略。为了定量描述该设计,引入下列符号: k ——加法器的流水线级次; T_c ——时钟周期; T_p ——加法器的流水线周期,很明显 $T_p = kT_c$ 。

3.1 累加器的原理图

本设计的数据通路如图 1 所示,采用的模块有:一个加法器,一个缓存,两个 3 选 1 多路选择器。

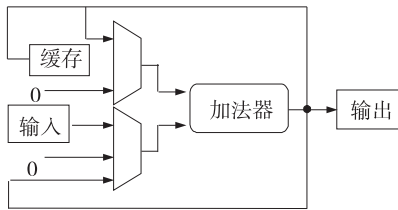


图 1 累加器数据通路

如表 1 所示,将 N 个数据串以 k (这里以 7 级为例) 为单位分割为模块 1、模块 2、模块 n ...模块 $[N/7]$ (中括号表示取整)、剩余模块 (当 $N/7$ 的余数不为零时就会产生剩余模块)。

表 1 数据串分块方式 ($k=7$)

	A_1	A_2	...	A_m
第 1 模块	B_1	B_2	...	B_m

	G_1	G_2	...	G_m
第 2 模块	H_1	H_2	...	H_m

第 n 模块			...	
剩余模块	α_1	α_2	...	α_m
	β_1	β_2	...	β_m

3.1.1 前 $[N/7]$ 个模块累加方式

对于前 $[N/7]$ 个模块,数据将采用纵向方式进入累加器,以模块 1 为例,其数据进入累加器的顺序是 $A_1B_1C_1D_1E_1F_1G_1, A_2B_2 \dots G_2, \dots, A_mB_m \dots G_m$, 其余模块依此类推,通过交换元素的顺序,刚好实现了各串累的元素相加。这种模式使流水线的利用率达到了 100%。完成这些模块累加所需时间 t_i 为

$$t_i = [N/7]mT_p \quad (1)$$

3.1.2 剩余模块的处理

剩余模块的处理较为复杂,对于剩余模块中的数据可采用两种方式累加,即纵向方式和组合方式。

(1) 纵向方式

该方式就是延续前 $[N/7]$ 个模块中的累加方式。因为剩余模块中的串数少于 k ($k=7$), 所以为了采用纵向累加的方式,必须将该模块添 0 补足为 7 串,这样就

会引入 $0+0$ 的无效的操作,这些无效操作占据部分流水线,使得流水线的利用率降低。但是当 $\text{rem}(N/7)$ (rem 表示取余数) 较大,而 m 较小时,无效操作的数量相比于有效操作还是较少,进行无效的操作是值得的。这时累加剩余模块需要的时间 t_{r1} 为,

$$t_{r1} = (m+1)T_p \quad (2)$$

而此累加方式引入的无效操作时间 t_{ne1} 为,

$$t_{ne1} = m(7 - \text{rem}(N/7))T_c \quad (3)$$

(2) 组合方式

当 m 很大时,剩余模块若采用纵向累加方式会带来大量的无效操作,这时可以采用组合的累加方式来达到较高的流水线利用率,分两个阶段完成:

(1) 合并阶段。将 m 个元素累加为 k 个部分和的阶段称为合并阶段,即是让每个数据串顺序进入累加器实现合并。以剩余模块的 α 串为例,各个流水线周期进入累加器的数据为表 2 所示。由第 p ($p \geq 2$) 个周期中的通项表达式可得:当 $k=7$ 时,经过 $([m/7] + \text{rem}(m/7)/7)T_p$ 时间后, α 串中 m 个数据完全进入加法器,再经过一个 T_p 时间,就会产生 7 个部分和。

(2) 部分和累加阶段。将前 6 个部分和送入 buffer,再用纵向式中纵向添 0 的方式完成累加。

表 2 合并阶段各个流水线周期进入加法器输入端的数据

流水线周期	数据
1	$\alpha_1, 0$
第 1 流水线周期	$2 \quad \alpha_2, 0$
...	...
k	$\alpha_k, 0$
第 2 流水线周期	$1 \quad \alpha_1, \alpha_{k+1}$
...	$2 \quad \alpha_2, \alpha_{k+2}$
...	...
k	α_k, α_{2k}
第 3 流水线周期	$1 \quad \alpha_1 + \alpha_{k+1}, \alpha_{2k+1}$
...	$2 \quad \alpha_2 + \alpha_{k+2}, \alpha_{2k+2}$
...	...
k	$\alpha_k + \alpha_{2k}, \alpha_{3k}$
第 p 流水线周期	$1 \quad \alpha_1 + \alpha_{k+1} + \alpha_{2k+1} + \dots + \alpha_{(p-2) \cdot k+1}, \alpha_{(p-1) \cdot k+1}$
...	$2 \quad \alpha_2 + \alpha_{k+2} + \alpha_{2k+2} + \dots + \alpha_{(p-2) \cdot k+2}, \alpha_{(p-1) \cdot k+2}$
...	...
k	$\alpha_k + \alpha_{2k} + \alpha_{3k} + \dots + \alpha_{(p-1) \cdot k}, \alpha_{p \cdot k}$

以此方式完成剩余模块所需时间 t_{r2} 为:

$$t_{r2} = \text{rem}(N/7)(7[m/7] + \text{rem}(m/7) + 48)T_c \quad (4)$$

该种方式的无效操作时间 t_{ne2} 为

$$t_{ne2} = \text{rem}(N/7)(6T_c + T_p) \quad (5)$$

(3) 剩余模块累加方式的选择

究竟采用那种方式累加剩余模块取决于 t_{r1} 与 t_{r2} 的大小,为了便于比较,现作出(2)式、(4)式的函数图象,如图 2 所示。图中小球标记的为 t_{r1} ,其余六条为 t_{r2}

(分别是 $\text{rem}(N/7) = 1, 2, 3, 4, 5, 6$ 的情形, 图例中 t_2 连字符后的数就表示 $\text{rem}(N/7)$)。

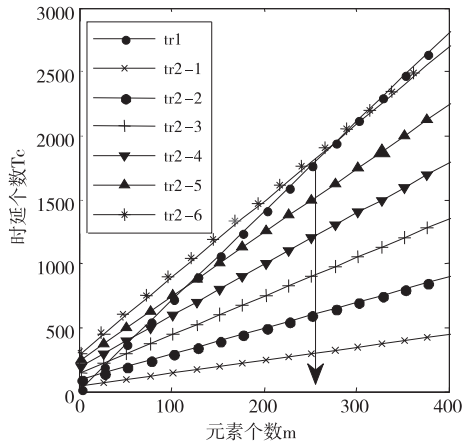


图2 两种方式累加剩余模块的时延比较

除了 t_{r1} , 其余均为折线, 但为简便计, 七条曲线均近似视为直线。在近似情况下, 取 $m = 280$ 为分界点(图中箭头所指)。当 $m < 280$ 时, t_{r1} 有时大于 t_{r2} , 有时小于 t_{r2} , 这视具体 m, N 的数值而定, 这时定义一个时间比值表达式: $r_i = t_{r1}/t_{r2}$, 若 $t_{r1}/t_{r2} \leq 1, r_i = 1$, 否则 $r_i = 0$ 。将时间比值做成一个 280 行 \times 6 列的二维表, 实际应用时可将该表固化于芯片中, 以此决定剩余模块的累加方式——若 $r_i = 1$, 采用横向方式; 若 $r_i = 0$, 采用组合方式。为便于存储及控制, 只取 2^8 (256) 行数据; 而当 $m > 256$ 时, 剩余模块采用横向累加的方式。

3.2 控制器设计

该累加器由调用它的电路模块控制, 具体而言就是主模块控制该累加器中两个多路选择器的地址端来实现控制。控制器的设计采用状态机描述, 由上面的分析可知, 电路存在的状态最多有 5 个, 依次是空闲等待(1 个状态), 小数部分累加(3 个状态, 收敛阶段 1 个状态, 部分和累加 2 个状态), 整数模块累加(1 个状态)。外部控制器在不同的状态有不同的输出控制字, 控制两个多路选择器的地址端来完成累加。

4 性能评估

4.1 流水线的利用率

在处理剩余模块时引入的无效操作会使流水线暂停, 因此有必要估算流水线的利用率。定义该设计流水线利用率为 $e = \text{有效操作时间} / \text{总时间} = 1 - \text{无效操作} / \text{总时间}$ 。由式(1)~式(5)易得出流水线的利用率, 图3是 matlab 计算流水线利用率的结果。

可以看出, 在 $m < 150$, 且 $N < 28$ 的一些区域流水线的利用率不高 ($\leq 80\%$), 这有两方面的原因: 一是数据

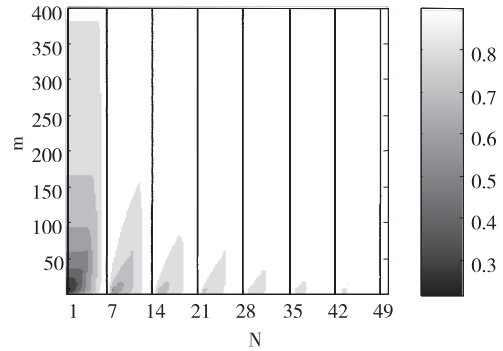


图3 流水线利用率($k=7$)

规模过小, 发挥不了流水线的优势。二是在这些区域有很多无效操作, 为的是不引入复杂的控制器设计。随着 M, n 的增大, 流水线的利用率将达到了 90% 以上 ($N > 37, m > 380$ 的区域), 尤其是当 N 是流水线级数 7 的整数倍时, 利用率达到了 100% (图中竖线)。图3只绘制了 $N[1, 50], m[2, 400]$ 区间的图像, 在这之外的区间, 利用率均达到了 90% 以上, 故未画出。

4.2 时延比较

文献[11]提供的设计不会暂停流水线, 它总共消耗的时间为 $Mn + 2k^2$, 这在时延方面是较好的设计。为方便与其比较, 定义一个比值: $t_compare = \text{本设计的时延} / \text{文献[11]中设计的时延}$ 。 $t_compare$ 随 M, n 变化的函数图象为图4所示, 本设计的时延与不会超过文献[5]中设计时延的 1.2 倍, 并且达到 1.2 倍的区域(图中白色区域)是很小的。故本设计达到了较好的时延效果。

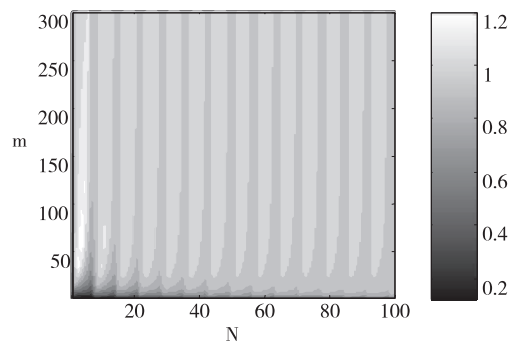


图4 本设计与文献[11]中设计的时延比较

同样, 在 $m > 300, N > 380$ 的区域, $t_compare$ 不会大于 1.2, 故未画出图像。

4.3 硬件资源消耗

本设计的实现平台为 Altera 公司生产的 Cyclone IEP2C35F672C6 芯片, 通过了 quartusII 仿真。由于采用的平台不同, 资源消耗有差异, 下面只列出了独立于具体平台的资源比较结果。

本设计: 加法器(k 级流水线)1 个; 缓存规模为 $k-1$ 。
文献[10]: 加法器(k 级流水线)1 个; 缓存规模为 0

$(N=k), N-k (N>k), \lfloor k/N \rfloor \cdot N - k (N < k)$ 。

文献[11];加法器(k 级流水线)1个;缓存规模为 $2k^2$

可以看出,本设计消耗的资源少于已有的设计,并且只与流水线级次有关,与问题的规模无关,不会随着问题的规模增大而增加。

5 结束语

本设计采用一个加法器, $k-1$ 个缓存,两个3选1多路选择器,实现了多串数据的累加。该设计资源消耗较少,控制相对简单。在数据规模较小时,流水线利用率不高,随着数据规模的增大,流水线利用率大大增加(大于90%)。该设计时延最多是已有设计的1.2倍。因此,对于有大规模数据累加的场合,比如大型矩阵乘积、大型线性方程组求解等,该设计的优势能得到充分发挥。

参考文献:

- [1] Ling Zhuo. High-performance linear algebra on reconfigurable computing system[D]. US, SC: University of Southern California, 2007.
- [2] Prasenjit Biswas, Pramod P Udupa. Accelerating Numerical Linear Algebra Kernels on a Scalable Run Time Reconfigurable Platform[C]. 2010 IEEE Annual Symposium on VLSI, 2010: 161-166.
- [3] 邬贵明. FPGA 矩阵计算并行算法与结构[D]. 长沙:国防科技大学, 2011.
- [4] 安婧. 基于 FPGA 的高速矩阵运算算法研究[D]. 太原:中北大学, 2009.
- [5] 郭磊. 矩阵运算的硬件加速技术研究[D]. 长沙:国防科学技术大学, 2010.
- [6] 林皓. 基于 FPGA 的矩阵运算实现[D]. 南京:南京理工大学, 2007.
- [7] 袁俊瑜, 杜正聪, 祝俊. 基于近似核 FFT 快速测频算法的 FPGA 实现[J]. 四川理工学院学报:自然科学版, 2011, 24(4): 456-458.
- [8] Falguni Gandhi. A novel algorithm for fixed-point and floating-point matrix multiplication on a FPGA[D]. Texas A&M University-Kingsville, 2006.
- [9] Yamini Yadav. Reconfigurable matrix multiplication[D]. Texas A&M University-Kingsville, 2005.
- [10] Lionel M Ni, Kai Hwang. Vector reduction methods for arithmetic pipelines[C]. Proceedings of the 6th International Symposium on Computer Arithmetic, 1983: 144-150.
- [11] Ling Zhuo, Viktor K Prasanna. High-performance and area-efficient reduction circuits on FPGAs[C]. Proceedings of the 17th International Symposium on Computer Architecture and High Performance Computing, 2005: 1-8.
- [12] Ling Zhuo, Viktor K Prasanna. High-performance designs for linear algebra operations on reconfigurable hardware [C]. IEEE Transactions on Computers, 2008, 57 (8): 1057-1071.
- [13] Ling Zhuo, Viktor K Prasanna. Scalable hybrid designs for linear algebra on reconfigurable computing systems [C]. IEEE Transactions on Computers, 2008, 57 (12): 1661-1675.
- [14] Song Qingzeng, Gu Junhua. Design and implementation of an FPGA-based high-performance improved vector-reduction method[C]. 2011 International Conference on Electronics and Optoelectronics (ICEOE 2011). 2011: 52-55.

Research on a Kind of Accumulator Basing on Multilevel Pipeline Adder

YUAN Song^a, TANG Jing-you^b, LIU Li^a

(a. School of Science; b. School of National Defense, Southwest University of Science and Technology, Mianyang 621010, China)

Abstract: Purpose-designed circuits can accelerate the speed of scientific calculation. Multiple data accumulation is a common operation in scientific calculation. It is easy to meet pipeline data hazards during designing the hardware accumulator. Our design is dividing those data into two kinds of modules according to pipeline level, and different modules using different accumulation methods. Based on a multilevel pipeline adder, this design is implemented on a FPGA. It has less hardware resources and higher pipeline utilization, and the control is relatively simple. Especially for large-scale data, its advantages can be fully taken on.

Key words: hardware acceleration; FPGA; multilevel pipeline; accumulator