一种适合 FPGA 实现的开平方算法

林志谋, 卢贵主*
(厦门大学电子工程系, 福建 厦门 361005)

摘要: 随着嵌入式等实时系统技术的发展, 越来越多的数字信号处理要求将开平方等运算用硬件实现. 本文介绍了一种改进的不恢复余数的开平方算法, 分析了它的一些优点, 用 VHDL 语言在 Xilinx 公司的 FPGA 器件上实现了该算法; 分别设计了被开方数是 16 位和 8 位的电路, 并进行了比较. 实验结果表明, 与其他算法比较, 改进的不恢复余数的开平方算法具有输出延时小和占用资源少的优点, 同时随着被开方数的增加, 最高频率下降幅度不大, 具有较好的实用价值.

关键词: 开平方; FPGA; VHDL
中图分类号: TN 914 文献标识码: A 文章编号: 0438-0479(2006)02-0199-03

目前常见的开平方算法主要有实函数近似法、Newton-Raphson 算法和逐位循环算法. 实函数近似法无法满足精度要求, 适用范围有限; Newton-Raphson 算法迭代次数的多少仅由初始值与被开方数的误差决定, 与被开方数大小无关; 逐位循环算法比 Newton-Raphson 算法节省芯片资源, 它可以分为恢复余数算法和不恢复余数算法[1], 后者由于无须恢复部分余数, 没有反馈调整单元, 便于进行流水线设计, 比前者占用资源少, 输出时延小. 在硬件实现的时候, 我们关心的是算法能够具有更快的数据处理速度和需要更少的芯片面积, 目前很多开平方算法往往很难满足上述要求. 随着半导体技术的飞速发展, FPGA 芯片的规模越来越大, 速度也越来越快, 非常适合实现开平方运算. 本文介绍了一种改进的不恢复余数的开平方算法, 很适合用 FPGA 器件实现, 它使得含有较多开平方运算的系统实现 SOC(System On Chip) 成为可能.

1 改进的不恢复余数的开平方算法

我们知道, 不恢复余数算法有一个缺点是, 有时某一步的运算可能存在部分余数都变为零, 继续计算将导致错误, 故需要另外的辅助电路判别处理, 这样使得电路设计变得比较复杂.

本文在设计电路时采用的是改进的不恢复余数算法[2], 它在原有不恢复余数算法的基础上, 保持了流水线设计的特点, 每一步算得的根都是确定的, 不再需要进行恢复; 避免了例外情况的存在, 也不需要辅助电路. 该算法将上一步的部分余数经过适当变换作为被加(减) 数, 将前面得到的根经过适当的组合作为加(减) 数, 然后它们之间进行补码运算就可以得出一个新的部分余数. 平方根通过判断部分余数的正负得出. 为便于说明, 假设 $A = 0a_1a_2 \cdots a_n$ 为被开方数, n 代表被开方数的位数, $Q = q_1q_2 \cdots q_n$ 为计算出的平方根, $R(k)$ 为第 k 步运算时初始的部分余数, $D(k)$ 为第 k 步运算时的加(减) 数. 具体步骤如下:

(1) 初始化 $R(0) = 0a_1a_2$, $D(1) = 001$, 则 $R(1) = R(0) - D(0)$, $q(1)$ 取 $R(1)$ 的最高位.

(2) 去掉 $R(k)$ 的最高位, 再从被开方数补入两位, 形成被减数, 减数为 $D(k) = q_1q_2 \cdots q_k\bar{q}_k1$.

(3) 如果 $q(k) = 1$, 被减数减去减数就得到 $R(k+1)$, 取 $R(k+1)$ 的最高位, 求反就得到 $q(k+1)$.

(4) 如果 $q(k) = 0$, 被减数加上减数就得到 $R(k+1)$, 取 $R(k+1)$ 的最高位, 求反就得到 $q(k+1)$.

(5) 重复(2)~(4), 直到求出所有平方根为止.

整个算法的流程图如图 1 所示.

2 算法的设计和实现

本文分别设计了被开方数是 8 位和 16 位的电路, 现只以后者为例说明, data 为 16 位无符号的被开方数输入, 声明为 std_logic_vector(15 downto 0) 的数据类型, 其中设前 8 位为整数部分, 后 8 位为小数部分; qout 为 8 位无符号的平方根输出, 声明为 std_logic_vector(7 downto 0), 其中设前 4 位为整数部分, 后 4 位为小数部分; en 为 1 位的使能控制信号, 声明为 std_logic 类型, 低电平有效; done 表示使能计算时, 得出第 1 个计算结果的信号, 声明为 std_logic 类型.

收稿日期: 2005-09-05
基金项目: 厦门大学预研基金(Y07014) 资助
作者简介: 林志谋(1978-), 男, 硕士研究生.
通讯作者: rsguizhu@xmu.edu.cn

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

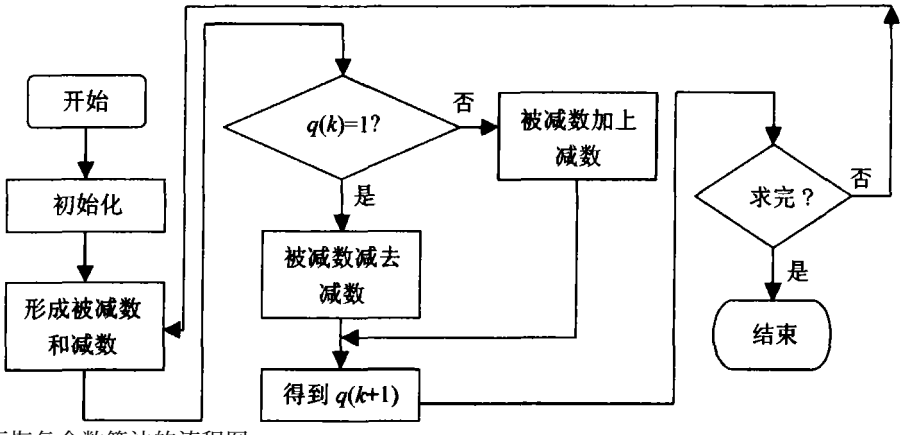

图 1 改进的不恢复余数算法的流程图

Fig. 1 Revised Non-restoring square root algorithm flow chart

该设计主要由若干个功能模块和若干个 D 锁存器组成[2]. 模块的功能大部分是相同的, 只是处理的位数不同而已, 它们的功能是先根据给出的 $q(k)$ 判断这一步是作加法还是作减法运算, 然后再根据 $D(k)$ 等计算出下一步的部分余数 $R(k+1)$, 并给出下一步的根 $q(k+1)$. D 锁存器的功能主要是为了同步, 并使计算结果在同一时刻出现, 它们由同一个时钟控制.

整个电路的设计采用模块化的设计思想[3], 采用 VHDL 语言描述. 由于电路比较复杂, 如果顶层电路采用 Xilinx ECS 的电路图输入方式, 则设计的效率不高, 而且众多的信号线之间的连接容易出错, 而顶层电路采用结构描述方式, 模块间类似的结构使得用 VHDL 语言描述显得十分简洁, 便于优化设计. 这里采用结构描述方式, 把功能模块作为低层次的设计模块, 在构造体中用 COMPONENT 语句指明在该电路中已经生成的功能模块, 用 PORT MAP 语句将生成的功能模块的端口与所设计的各模块的端口联系起来, 并用 VHDL 语言描述出输入输出的 D 锁存器, 以及它们与各模块之间的连接关系.

3 结果的分析与比较

本文采用 Xilinx 公司的 FPGA 器件, 它们具有丰富的锁存器资源, 完全可以满足设计对锁存器的需求. 整个设计在 ISE6 下编译完成, 使用 ModelSim 工具进行功能仿真和时序仿真, 最后得到的时序仿真波形如图 2 所示. 算得的平方根结果在 9 个时钟周期后开始输出, 而且以后每个时钟输出一个运算结果.

输入被开方数 240(11110000.00000000), 从仿真波形可以看出, 结果为 15.437 5(1111.0111), 同样地, 输入被开方数 150.289 1(10010110.01001010), 得到平方根为 12.25(1100.0100).

为了便于说明该算法的优点, 下面对它和其他算法的性能进行比较, 并对它在不同的输入被开方位数和不同的器件上实现的性能也进行了比较:

(1) 在相同器件和相同精度的条件下, 对不同的开平方算法的性能进行比较: 采用 Xilinx 公司的 Spartan II 系列的 XC2S150 器件, 输入均为 16 位的被开方数, 对本算法和参考文献[4] 采用的逐次逼近算法[4](它属于 Newton-Raphson 算法) 的结果进行比较, 结果见表 1.

表 1 不同算法性能比较

Tab. 1 Compare of different algorithm

算法类型	占用 CLB /%	需要的时钟周期	精度
改进的不恢复余数算法	7	9	0.0625
逐次逼近算法	10	31	0.0625

本文的算法性能指标较好, 在相同精度的条件下,

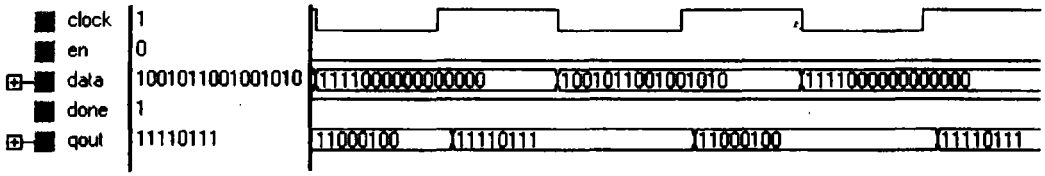

图 2 仿真结果的部分波形

Fig. 2 Waveform of simulation result

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

占用资源比采用逐次逼近算法少了 3%, 完成一次平方根运算需要的周期明显减少, 它只需要 9 个时钟周期就可以输出第 1 个运算结果, 以后每个时钟输出 1 个运算结果. 而逐次逼近算法由于需要反馈调整, 需要 31 个时钟周期才可以完成 1 次开平方运算.

(2) 在不同的输入被开方位数和不同的器件上实现的条件下, 对该算法的性能进行比较, 它们的综合结果见表 2.

表 2　改进的不恢复余数开平方算法性能指标

Tab. 2　Performance of revised non-restoring square root algorithm

项目/指标	Spartan II XC2S150		Virtex2 XC2V500	
	占用 CLB/%	最高频率 /MHz	占用 CLB/%	最高频率 /MHz
被开方位数 8 位	2	143	1	268
被开方位数 16 位	7	134	4	252

从表 2 可以看出, 在相同器件的条件下, 随着被开方位数的增加, 占用的资源增加了, 这主要是由于为了提高速率, 使用了流水线设计, 增加了许多锁存器的缘故, 但在同等条件下与参考文献[4] 所采用的算法比较, 所占用资源仍然较少. 最高频率虽然有所下降, 但电路处理的被开方位数却增加了一倍, 相对而言, 最高频率下降幅度并不大, 分别为 6.2% 和 6.0%. 以被开方位数为 16 且使用 Spartan II 的情况为例, 流水线的设计使电路每秒可以获得 134 M 个运算结果, 可以满足大部分实时系统的要求. 同许多算法一样, 要使处理被开方数的位数增多或精度更好, 一般要以较多的资源和较慢的处理速度为代价. 根据设计电路的特点可以看出, 被开方数每增加 2 位, 输入输出的锁存器数量增加了一倍, 模块内的加法器等资源也相应增加了, 输出延时增加了一个时钟周期. 假设被开方数为 $2N$ 位, 平方根为 N 位, 我们可以得出第 1 个结果的输出延迟为 $N+1$ 个时钟周期(N 为大于 0 的偶数).

在被开方位数相同的条件下, 以被开方位数均为 8 的情况为例, 在 Spartan II 系列的 XC2S150 器件上, CLB 占用 2%, 最高频率为 143 MHz; 在 Virtex2 的 XC2V500 器件上, CLB 占用 1%, 最高频率为 268 MHz. 后者占用的资源减少了, 而最高频率却提高了近两倍, 处理数据的速度显然更快, 被开方位数为 16 的情况与此类似. Virtex2 系列和 Spartan II 系列结构相同, 但是前者比后者有更大的容量和更高的性能. 随着 FPGA 产品的规模和速度不断提高, 含有大量开平方运算的系统在单芯片上实现已经不再是梦想.

4　结　论

本文介绍了一种改进的不恢复余数的开平方算法, 并用 VHDL 语言在 Xilinx 公司的 FPGA 器件实现了该算法, 与其他算法比较, 它具有占用资源少, 处理速度快的优点. 随着 FPGA 产品的规模和速度不断提高, 可以很容易在单芯片上实现含有较多开平方运算的系统.

参考文献:

[1] Li Yamin, Chu Wanming. Parallel-array implementations of a non-restoring square root algorithm[C]// 1997 IEEE International Conference, 1997: 690-695.

[2] 于翔涛. 1. 高速硬件开平方电路的 ASIC 设计; 2. 电流反馈型运算放大器的工作原理与电路实现(部分章节)[D]. 天津: 南开大学, 2000.

[3] 侯伯亨, 顾新. VHDL 硬件描述语言与数字逻辑电路设计[M]. 西安: 西安电子科技大学出版社, 2000.

[4] 王艳梅, 王同杰, 郑成文. 用 VHDL 实现的开方运算[J]. 沈阳工业学院学报, 2004, 23(1): 3.

A Square Root Algorithm Based on FPGA

LIN Zhi-mou, LU Gui-zhu*

(Dept. of Electronic Engineering, Xiamen Univ., Xiamen 361005, China)

Abstract: With the development of embedded real time systems, more and more application areas need hardware implementation of square root algorithm. This paper introduced a revised non-restoring square root algorithm, discussed it's advantages and implementation way on FPGA. Different circuit was designed separately when radicand is 16 bits and 8 bits, respectively. The experiment shows that the algorithm introduced here has many advantages in terms on practical performance, such as output delay and speed. In the mean time with the increasing of the number of radicand, the maximum frequency only decreased a few.

Key words: square root; FPGA; VHDL

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net