CRC 校验码并行计算的 FPGA 实现

张树刚,张遂南,黄士坦
(西安微电子技术研究所,陕西 西安 710075)

摘 要:用软件实现 CRC 校验码计算很难满足高速数据通信的要求,基于硬件的实现方法中,有串行经典算法 LFSR 电路以及由软件算法推导出来的其它各种并行计算方法。以经典的 LFSR 电路为基础,研究了按字节并行计算 CRC 校验码的原理,并以常见的 CRC-16 和 CRC-CCITT 为例,用 VHDL 语言进行了可综合设计。结果表明这种实现方法在速度和占用资源方面优于常见的设计,适合在 FPGA 中实现 CRC 校验码的计算。

关键词:CRC 并行计算;CRC-CCITT;CRC-16;数据通信
中图分类号:TP311 **文献标识码**:A **文章编号**:1673-629X(2007)02-0056-03

CRC Parallel Computation Implementation on FPGA

ZHANG Shu-gang, ZHANG Sui-nan, HUANG Shi-tan
(Xi'an Micro-Electronics Institute, Xi'an 710075, China)

Abstract: CRC computing by software can hardly meet with the high speed digital system. Classic LFSR circuit and other algorithm that derived from software are generally adopting as hardware solutions. Based on classic LFSR circuit, present a byte-wise CRC algorithm and express CRC-16 and CRC-CCITT in VHDL as two examples. The method is proved faster and less resource used than the others by the result synthesized in XST.

Key words: CRC parallel computation; CRC-CCITT; CRC-16; digital communication

0 引 言

循环校验码(Cyclical Redundancy Check,缩写为 CRC)在数据通信和计算机通信中有着广泛的应用,它具有编码和解码方法简单,检错和纠错能力强等特点,可以显著地提高系统的检错能力[1]。CRC 校验码的计算一般可分为软件和硬件实现两种方法。Tenkasi V. Ramabadran 和 Sunil S. Gaitonde 总结了 5 种用于软件实现的方法,其中最常用的是按字节查表法和半字节查表法,这种方法一次处理一个字节或半个字节来生成 CRC 校验码,但它需要预先计算 512 个字节或 32 个字节的查找表,而且由于软件代码的顺序执行的特点,在对速度要求甚高的场合很难满足要求[2]。

目前,大多数 CRC 校验码的实现都是用硬件电路直接在芯片内完成的。经典的硬件实现采用 LSFR(线性反馈移位寄存器组)来完成[3],这种方法简单,但每次只能处理一位二进制数据,也很难以满足速度较高的场合。CRC 校验码的并行算法有查表法及基于查表法而导出的一些方法,但这些方法均需要存储长度较大的 CRC 余数表,并且随着并行位数的增加,余数表的长度按指数增加,其现实性亦随之大大降低,Giuseppe Campobello, Giuseppe Patane and Marco Russo 根据线性时不变系统的特性推出了用于计算 CRC 校验码计算的转换矩阵,但变换矩阵的推导方法过于烦琐[4]。还有一种按字节运算方法,它直接推导出 CRC 校验码与输入数据和生成多项式的逻辑关系,然后直接运算得出 CRC 校验码,这种方法直接、简洁,文中的 CRC 校验码并行计算也采用了这种方法[5],以 CRC-16 和 CRC-CCITT 校验码为例,用 VHDL 语言进行了可综合代码的设计,最后在 XILINX 公司的 Spartan 器件上进行了验证。

1 CRC 校验码的数学计算方法

CRC 检验的基本原理是:在一个 k 位二进制数据序列之后附加一个 r 位二进制检验码序列,构成一个总长为 $n = k + r$ 位的二进制序列,这种编码又叫(N, K)码,它可以提高整个编码系统的码距和检错能力。

例如,对 k 位二进制数据序列 $D = [d_{k-1}\ d_{k-2} \cdots d_1 d_0]$ 进行计算,得到的 r 位二进制检验码表示为 $R =$

收稿日期:2006-04-28
作者简介:张树刚(1979-),男,陕西韩城人,硕士研究生,研究方向为嵌入式高速计算机系统结构设计;黄士坦,研究员,博士生导师,研究方向为图像处理、嵌入式计算机及并行计算。

$[r_{r-1}r_{r-2}\cdots r_1r_0]$，将 R 附加到 D 之后，构成一个新的 n 位二进制序列 $M = [d_{k-1}\ d_{k-2}\ \cdots\ d_1d_0r_{r-1}r_{r-2}\ \cdots\ r_1r_0]$。其中校验码 R 是通过对数据序列 D 进行二进制除法取余式运算得到的，它被一个称为生成多项式的 $r+1$ 位二进制序列 $G = [g_rg_{r-1}\cdots g_1g_0]$ 来除，得到的余数就是 CRC 校验码。

为了方便表示，通常把一串信息表示为一个多项式，信息中的每位将作为多项式的系数。例如 11001001 表示为 $x^7 + x^6 + x^3 + 1$。这样，CRC 校验码的生成过程就可以表示如下：

$$\frac{x^rD(x)}{G(x)} = Q(x) + \frac{R(x)}{G(x)}$$

其中，$x^rD(x)$ 表示将数据序列 $D(x)$ 左移 r 位（即在 D 的末尾再增加 r 个 0），$Q(x)$ 代表这一除法所得的商，$R(x)$ 就是所需的余式。这一运算关系还可以用下式表示：

$$R(x) = \mathrm{Re}[\frac{x^rD(x)}{G(x)}]$$

其中，Re[] 表示对括号内的式子进行取余式运算。

检验码的编码计算如上所述，而检验过程则是对 M 序列直接进行除法取余式运算，即

$$R(x) = \mathrm{Re}[\frac{M(x)}{G(x)}]$$

所得到的余式 $R(x)$ 若为零则表示数据正确，否则认为发生错误。

表 1 列出了一些常见的的 CRC 资料。

表 1　常用 CRC 的相关资料

名称	生成多项式	简记式*	应用举例
CRC－16	$x^{16} + x^{15} + x^2 + 1$	8005	IBM SDLC
CRC－CCITT	$x^{16} + x^{12} + x^5 + 1$	1021	ISO HDLC，ITU X.25，V.34／V.41／V.42
CRC－32	$x^{32} + x^{26} + x^{23} + \cdots + x^2 + x + 1$	04C11DB7	ZIP，RAR，IEEE 802 LAN／FDDI，IEEE 1394

* 生成多项式的最高幂次项系数是固定的 1，在简记式中没有表示该位，如 1021 实际上是 11021。

2　并行计算的思路

经典的 CRC 校验码生成电路采用线性反馈移位寄存器 LFSR 来实现，如图 1 所示，二进制信息从 Input 端串行输入，$p_1p_2p_3\cdots p_{n-1}$ 为生成多项式 $p(x)$ 的系数，模 2 除法用异或门来实现，等到数据输入结束后，寄存器组中的内容既为余数，也就是 CRC 校验码。

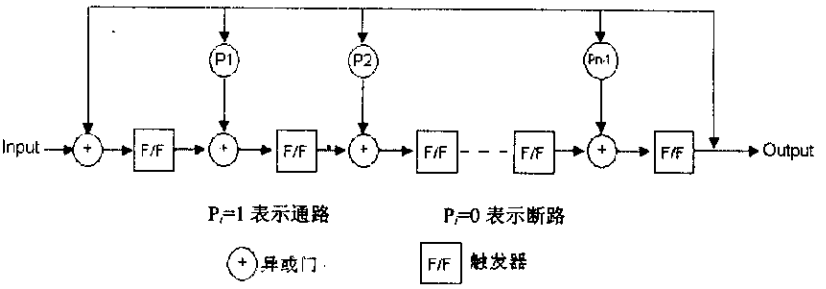

图 1　线性反馈移位寄存器 LFSR

对于 CRC－16　p_x =1000,0000,0000,0101

对于 CRC－CCITT　p_x =0001,0000,0010,0001

这种基本的串行 CRC 运算电路仅用了移位寄存器和异或门，占用的资源很少，但它运算速度的提高完全依靠于时钟的速度，相应地也增加了实现的难度，所以在现实中通常采用并行处理方法，例如，在高速的千兆以太网传输系统中往往采用 8 位或 32 位的并行处理电路以降低系统对时钟的要求。

按字节并行计算的思路是模拟串行计算电路的计算过程，从而推导出每次并行处理一个字节后的 CRC 校验码与当前输入字节和 CRC 寄存器前一状态的关系。

下面就用代入法来推导 CRC－CCITT 的逻辑计算式。图 2 中的寄存器编号是将图中的寄存器自左到右进行的编号，CRC 项为寄存器与 CRC 寄存器的对应关系，在开始运算时，把 CRC 中每一位的初始值都设为 0。以后的每一行表示当前输入一位数据后 CRC 寄存器中新的内容，CRC 的最终值为输入 D7 后各个项的因子相互异或的结果。图中的 C15 是 CRC 校验码的最高位，D0 是输入数据字节的最高位。

寄存器编号	R15	R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0
CRC	C15	C14	C13	C12	C11	C10	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
初始值	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
输入 D0	D0 C0	C15	C14	C13	C12	C11 D0 C0	C10	C9	C8	C7	C6	C5	C4 D0 C0	C3	C2	C1
输入 D1	D1 C1	D0 C0	C15	C14	C13	C12 D1 C1	C11 D0 C0	C10	C9	C8	C7	C6	C5 D1 C1	C4 D0 C0	C3	C2
输入 D2	D2 C2	D1 C1	D0 C0	C15	C14	C13 D2 C2	C12 D1 C1	C11 D0 C0	C10	C9	C8	C7	C6 D2 C2	C5 D1 C1	C4 D0 C0	C3
输入 D3	D3 C3	D2 C2	D1 C1	D0 C0	C15	C14 D3 C3	C13 D2 C2	C12 D1 C1	C11 D0 C0	C10	C9	C8	C7 D3 C3	C6 D2 C2	C5 D1 C1	C4 D0 C0
输入 D4	D4 C4 D0 C0	D3 C3	D2 C2	D1 C1	D0 C0	C15 D4 C4 D0 C0	C14 D3 C3	C13 D2 C2	C12 D1 C1	C11 D0 C0	C10	C9	C8 D4 C4 D0 C0	C7 D3 C3	C6 D2 C2	C5 D1 C1
输入 D5	D5 C5 D1 C1	D4 C4 D0 C0	D3 C3	D2 C2	D1 C1	D0 C0 D5 C5 D1 C1	C15 D4 C4 D0 C0	C14 D3 C3	C13 D2 C2	C12 D1 C1	C11 D0 C0	C10	C9 D5 C5 D1 C1	C8 D4 C4 D0 C0	C7 D3 C3	C6 D2 C2
输入 D6	D6 C6 D2 C2	D5 C5 D1 C1	D4 C4 D0 C0	D3 C3	D2 C2	D1 C1 D6 C6 D2 C2	D0 C0 D5 C5 D1 C1	C15 D4 C4 D0 C0	C14 D3 C3	C13 D2 C2	C12 D1 C1	C11 D0 C0	C10 D6 C6 D2 C2	C9 D5 C5 D1 C1	C8 D4 C4 D0 C0	C7 D3 C3
输入 D7	D7 C7 D3 C3	D6 C6 D2 C2	D5 C5 D1 C1	D4 C4 D0 C0	D3 C3	D2 C2 D7 C7 D3 C3	D1 C1 D6 C6 D2 C2	D0 C0 D5 C5 D1 C1	C15 D4 C4 D0 C0	C14 D3 C3	C13 D2 C2	C12 D1 C1	C11 D0 C0 D7 C7 D3 C3	C10 D6 C6 D2 C2	C9 D5 C5 D1 C1	C8 D4 C4 D0 C0
异或	Xor	Xor	Xor	Xor	Xor	Xor	Xor	Xor	Xor	Xor	Xor	Xor	Xor	Xor	Xor	Xor
CRC	C15	C14	C13	C12	C11	C10	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0

图 2　CRC 逻辑表达式的过程

设 $x_i = C_i \text{ xor } D_i$，则可进一步得到 CRC－CCITT

的表达式如图 3 所示,同样也可以得到 CRC－16 的表达式如图 4 所示。

因子	X3 X7	X2 X6	X1 X5	X0 X4	X3	X2 X3 X7	X1 X2 X6	X0 X1 X5	C15 X0 X4	C14 X3	C13 X2	C12 X1	C11 X0 X3 X7	C10 X2 X6	C9 X1 X5	C8 X0 X4
异或	Xor	Xor	Xor	Xor	Xor	Xor	Xor	Xor	Xor	Xor	Xor	Xor	Xor	Xor	Xor	Xor
CRC	C15	C14	C13	C12	C11	C10	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0

图 3 计算 CRC-CCITT 的逻辑关系表

因子	X7 X6 X5 X4 X3 X2 X1 X0	X6 X5 X4 X3 X2 X1 X0	X7 X6	X6 X5	X5 X4	X4 X3	X3 X2	X2 X1	C15 X1 X0	C14 X0	C13	C12	C11	C10	C9	C8 X7 X6 X5 X4 X3 X2 X1 X0
异或	Xor	Xor	Xor	Xor	Xor	Xor	Xor	Xor	Xor	Xor	Xor	Xor	Xor	Xor	Xor	Xor
CRC	C15	C14	C13	C12	C11	C10	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0

图 4 计算 CRC-16 的逻辑关系表

3 并行计算的实现

一般的 CRC 校验码生成器具有编码和解码两种功能,编码用于对输入数据计算 CRC 校验码,解码用于验证接收到的数据是否正确,其实这两个过程对于 CRC 校验码生成器是一样的,不同的是解码完成后需要判断得到 CRC 校验码是否为 0,如果是 0 则表示数据正确,如果不为 0 则表示数据有错。表 2 是要设计的 CRC 校验码生成器的端口说明。

表 2 CRC 校验码生成器的端口说明

端口	宽度	输入输出	说明
Rst_i	1	Input	CRC 生成器复位
Clk_i	1	Input	工作时钟
data_i	8	Input	输入数据
Crc_o	16	Output	输出 CRC 校验码的 16 位数据

VHDL 语言的主要实现代码如下:

```
    din <= data_i;
    creg <= crc_reg(0 to 7) xor din(0 to 7);
    crc_t(0 to 3) <= creg(4 to 7) xor creg(0 to 3);
process(clk_i,rst_i)
begin
if rst_i='1' then crc_reg <= (others => '0');
elsif rising_edge(clk_i) then
        crc_reg(15) <= crc_t(3);
        crc_reg(14) <= crc_t(2);
        crc_reg(13) <= crc_t(1);
        crc_reg(12) <= crc_t(0);
        crc_reg(11) <= creg(3);
        crc_reg(10) <= creg(2) xor crc_t(3);
        crc_reg(9) <= creg(1) xor crc_t(2);
        crc_reg(8) <= creg(0) xor crc_t(1);
        crc_reg(7) <= crc_reg(15) xor crc_t(0);
        crc_reg(6) <= crc_reg(14) xor creg(3);
        crc_reg(5) <= crc_reg(13) xor creg(2);
        crc_reg(4) <= crc_reg(12) xor creg(1);
        crc_reg(3) <= crc_reg(11) xor crc_t(3) xor creg(0);
        crc_reg(2) <= crc_reg(10) xor crc_t(2);
        crc_reg(1) <= crc_reg(9) xor crc_t(1);
        crc_reg(0) <= crc_reg(8) xor crc_t(0);
end if;
end process;
crc_o <= crc_reg;
```

最后,对 CRC－CCITT 校验码生成器进行了功能仿真,结果如图 5 所示。

CRC16 校验码的仿真波形略,由图中可以看出,在数据输入的当前周期就可得到 CRC 校验码的内容。

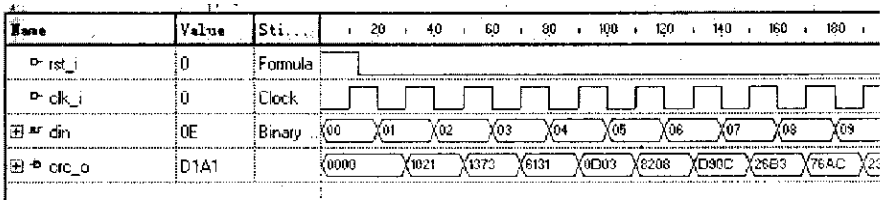

图 5 CRC－CCITT 校验码计算的功能仿真

4 结果比较

把文中的方法和其它常用方法分别从占用资源和最大频率进行了比较(见表 3),因为传统电路和查表法,以及矩阵法的硬件结构固定,综合结果与具体的生成多项式无关。而文中的代入法的逻辑运算关系会随着 CRC 校验码所采用的生成多项式不同,电路结构也不相同,所以分别给出了 CRC16 和 CRC－CCITT 的最大频率和占用资源。

表 3 各种常用方法综合结果的比较

实现途径		最大频率	占用 Slice 资源
传统电路 LFSR		1519MHz	9 Slices
查表法		867 MHz	11 Slices 和 512 字节 RAM 单元
矩阵法		617 MHz	10 Slices 和 32 个字节 RAM 单元
文中方法	CRC16	946MHz	9 Slices
	CRC－CCITT	1150MHz	9 Slices

从表中可以看出,传统硬件电路 LFSR 具有很高的速度,但它每次只能处理一位数据,且消耗的 FPGA 资源与文中的方法相当。所选用的器件为 Spartan 3 系列的 3s400,速度等级为－5,综合工具为 Xilinx ISE 6.2i 中的 XST。当然结果与器件的选择也有很大的关系。

5 结 论

文中在分析了 CRC 计算原理的基础上,用按字节

(下转第 62 页)

待数据库操作完成后更新缓冲区对象。

2.4 O/R 映射框架的优点

持久化对象访问层形成了一个封装良好的子系统以进行事务处理、数据库访问和存储。分层结构完全从物理数据模型中分离出了应用程序核心。这样允许用户任意调整数据,而不会影响到应用程序核心代码。应用程序核心使用了一个逻辑接口,它不需要知道数据库访问的细节。

(1) 灵活性。当底层数据库改变以便进行调整时,应用程序代码仍很稳定。

(2) 接口复杂度。接口最小化,因为它只是提供了应用程序核心需要的基本特性。但是,用户必须投入精力到允许处理大量需要的 SpecificPO 的产生器或模板。一旦完成产生器,定义新的 SpecificPO 则只花费非常少的时间。

(3) 访问层复杂度。持久化对象访问层的大部分类都是简单类,不存在复杂的算法,所以运行效率比较高。

(4) 性能。由于映射和软件其他层次的关系,用户因为数据库访问子系统付出了一点运行期代价。访问层以调整和缓冲简化对它进行了补偿。用户投入了大量的精力到快速处理器周期中,同时有效地利用了慢速的 I/O。

(5) 遗留应用程序的再设计。用户可以使用访问层分解现有应用程序的物理和逻辑数据模型。这对遗留应用程序再设计非常有用。首先,将数据库访问层插入代码中,然后,开始重写不同项目中的数据库和应用程序核心。

3 结束语

文中提出的持久化 OR 映射框架应用于中小型应用系统有非常好的性能,但是面对大型的面向对象技术设计的应用系统,处理更复杂对象的关系结构时,会出现大量的硬编码,整个系统设计将会显得十分庞大臃肿。如果采用 BrokerQuery 模式来联系逻辑处理层和物理处理层,它可以应用大型或者存在更复杂对象关系的面向对象应用系统中。BrokerQuery 是使用有向非循环图(DAG)描述服务,使用树匹配算法找出最佳的匹配的对象关系,操作物理层对象完成于关系数据库的交互[3]。由于树匹配算法复杂度较高,应用开销大,不适合应用于较小规模的面向对象应用系统中。

本框架对处理着重描述的是对对象的持久化的操作,对数据库的优化做的比较少,比如对海量数据的查询,如果查询的数据比较多不能将所有的对象都注册在缓冲区中,缓冲区表结构将非常庞大,不利于查询使用。可以采取对表结构的优化方案,如:Denormalization,Controlled Redundancy,Overflow Tables,这里不详细介绍可查看相关资料[3]。

参考文献:

[1] Hibernate Workgroup. Relational Persistence For Idiomatic Java Hibernate Reference Documentation 2.1.7[EB/OL]. 2005-02-16. http://www.hibernate.org/hib-docs/reference/en/html/.

[2] Martin R, Riehle D, Buschmann F. Patttern Languages of Program Design 3[M]. [s.l.]: Addison-Wesley Longman, Inc., 1998.

[3] Keller W, Coldwey J. Relational Database Access Layers-A pattern Language[C]// In Collected Papers from the PLoP96 Conferences. USA: Washington University, WUCS 97-07, 1997: 3-22.

[4] 姚淑珍,李 虎. UML 和模式应用——面向对像技术分析与设计导论[M]. 北京:机械工业出版社,2002.

[5] Gamma E, Helm R, Johnson R, et al. Design Patterns, Elements of Reusable Object-oriented Software[M]. [s.l.]: Addison-westey Pub. co., 1995.

(上接第 58 页)

并行计算的方法对 CRC-16 和 CRC-CCITT 校验码进行了设计,这种方法在数据输入的当前周期内就可以得到 CRC 校验码的结果。从综合的结果也可以看出,这种设计方式较以往的实现方式具有逻辑直接精简、速度更快,占用资源更少的优点,可以很方便地在 FPGA 中实现。

参考文献:

[1] 王新梅,肖国镇. 纠错码-原理与方法[M]. 西安:西安电子科技大学出版社,2001.

[2] 李永忠. 通用并行 CRC 计算原理及其硬件实现方法[J]. 西北民族学院学报:自然科学版,2002,23(1):33-37.

[3] Ramabadran T V, Gaitonde S S. A Tutorial on CRC Computations[J]. Micro IEEE, 1988, 8(4): 62-75.

[4] Campobello G, Patane G, Russo M. PARALLEL CRC REALIZATION[J]. IEEE TRANSACTIONS ON COMPUTERS, 2003, 52(10): 1312-1319.

[5] Perez A. Byte-Wise CRC Calculations[J]. IEEE Micro, 1983(6): 40-50.