

文章编号: 1000-5013( 2010)02-0166-04

# 低密度奇偶校验码在数字信号处理器上的实现

陈燕

( 华侨大学 信息科学与工程学院, 福建 泉州 362021 )

摘要: 研究低密度奇偶校验码( LDPC )的数字信号处理实现, 并采用 TM S320C5409DSP 芯片进行算法实现. 优化和积算法中的具体运算, 调整译码顺序, 将硬判决放入变量节点中运算, 校验和放入校验节点运算中, 避免重复寻址, 给出与现场可编程门阵列的通信方法. 在时钟频率为 20 MHz, 译码迭代次数为 10 次时, 测试得到的速率为  $20.4 \text{ kbit} \cdot \text{s}^{-1}$ .

关键词: 低密度奇偶校验码; 数字信号处理器; 译码; 校验

中图分类号: TN 911.72

文献标识码: A

Mackay 等在研究中发现, 低密度奇偶校验( LDPC )码在编译码复杂度较低的情况下, 其纠错能力具有接近甚至可能超越 Turbo 码的优点<sup>[1]</sup>. 因此, LDPC 码的研究成为当今信道编码领域最瞩目的热点. 2004 年 1 月, ETSI( 全欧广播电视组织 ) 颁布了新的卫星数字电视标准 DVB-S2, 其中使用了 LDPC 码和 BCH( Bose Chaudhuri Hocquenghem ) 码的级联来实现强大的纠错编码功能. 而且, IEEE 802.16e 移动宽带无线接入标准也将 LDPC 码纳入标准中, 成为下一代通信纠错编码的首选. 通信设备中广泛使用各种性能的数字信号处理器( Digital Signal Processor, DSP ) 器件, 将 DSP 用于 LDPC 码的解码算法中是有意义的. 本文研究低密度奇偶校验码的译码算法, 探讨采用美国 TI 公司的 TM S320C5409 芯片在数字信号处理器上的具体实现.

## 1 LDPC 码的编译码算法

### 2.1 校验矩阵的构造

分组码的构造关键在于校验矩阵, 只要给定了校验矩阵, 编码就完全确定了, 可以用之编码和译码. 采用基于块结构的 PEG 算法<sup>[2]</sup> 产生  $H$  矩阵, 校验矩阵的基本结构可以表示为

$$H = \begin{bmatrix} P_{0,0} & P_{0,1} & \cdots & P_{0,n_b-2} & P_{0,n_b-1} \\ P_{1,0} & P_{1,1} & \cdots & P_{1,n_b-2} & P_{1,n_b-1} \\ \vdots & \vdots & & \vdots & \vdots \\ P_{m_b-1,0} & P_{m_b-1,1} & \cdots & P_{m_b-1,n_b-2} & P_{m_b-1,n_b-1} \end{bmatrix} = P^H_b.$$

其中:  $P_{i,j}$  对应的是  $z \times z$  的置换矩阵( 循环右移单位阵 ) 或者是零矩阵; 校验矩阵的行数  $m = m_b \times z$ , 列数  $n = n_b \times z$ .

校验矩阵采用的基本矩阵的大小为  $14 \times 28$ , 块长度  $z$  为 36, 码率  $R$  为 0.5, 可以得到基于块结构的( 1 008, 504 ) 的 LDPC 码, 其校验节点维度的分布函数为

$$\alpha(x) = 0.28770x^5 + 0.71230x^6,$$

而变量节点维度的分布函数为

$$\lambda(x) = 0.00099 + 0.49901x + 0.17857x^2 + 0.14286x^3 + 0.17857x^6.$$

收稿日期: 2008-08-23

通信作者: 陈 燕( 1981- ), 女, 助教, 主要从事信道编码和无线通信技术的研究. E-mail: goldency@hqu.edu.cn.

基金项目: 华侨大学科研基金资助项目( 08HZR15 )

1.2 译码算法

基于 DSP 实现的 LDPC 译码器采用了 Normalized Min-Sum 算法<sup>[3]</sup>. 其中, 归一化系数  $\alpha$  为 0.75, 采用 7 bit 的均匀量化.

为了避免重复寻址, 减少指令周期数, 采用了图 1 的译码流程, 将译码架构做了调整, 改变经典的译码顺序. 其中, VNU 表示变量节点运算, CNU 表示校验节点运算. 将各码元译码输出、硬判决放入 VNU 中, 校验和运算放入 CNU 中, 初始化模块包括了变量节点运算、码元译码输出和硬判决. 各个模块在 DSP 上的具体实现过程如下.

(1) 初始化模块包含以下 3 点操作. (a) 初始化全部校验节点置信度.

$u_{0,i} = \log \frac{P(x_i = 0 | y_i)}{P(x_i = 1 | y_i)}, u_{j,i} = 0.$  (b) 变量节点运算.  $v_{i,j} = u_{0,i} +$

$\sum_{k=1, k \neq j}^{d_i^u} u_{k,i},$  由于初始化时, 传送的置信度  $u_{j,i} = 0,$  在进行和运算时, 只需要将  $u_{0,i}$  的值放入译码中间变量中, 以便于后面的校验和计算. (c) 码元译码输出硬判决.

$\hat{x}_i = \begin{cases} 0, & u_{0,i} > 0; \\ 1, & u_{0,i} < 0. \end{cases}$

(2) VNU 模块包含以下 3 点操作. (a) 求整列所有元素的和.  $Q_i =$

$u_{0,i} + \sum_{k=1}^{d_i^v} u_{k,i},$  (b) 计算全部变量节点发送置信度.  $v_{i,j} = Q_i - u_{j,i}.$  (c)

码元译码输出硬判决.  $\hat{x}_i = \begin{cases} 0, & Q_i > 0; \\ 1, & Q_i < 0. \end{cases}$

(3) CNU 模块包含了计算全部校验节点发送置信度  $u_{j,i}$  和校验及运算的操作. 即

$$u_{j,i} = \frac{3}{4} \cdot \prod_{k=1, k \neq i}^{d_j^c} \text{sgn}(v_{k,j}) \cdot \min_{k=1, k \neq i}^{d_j^c} |v_{k,j}|, \quad s_j = \sum_{k=1}^{d_j^c} u_{k,j}.$$

一般而言, 应该将校验和放在变量节点运算中硬判决之后进行. 但是, 寻址需要占用大量的指令周期. 计算校验和需要对  $H$  中每行非零元素进行寻址, 校验节点运算也需要对非零元素进行寻址进而找到  $v_{k,j}.$  如果将它们一起计算, 则可以避免重复寻址, 也就节省了大量的指令周期. 也就是说, 可以在找到一行的非零元素的位置后, 同时进行校验节点运算和校验和运算, 再对下一行进行寻址, 进行同样的操作, 直到对所有的 504 行完成操作为止.

1.3 校验节点算法优化

由于一共要进行 504 次校验节点运算, 校验节点中求本身元素外最小值, 也就是比较运算在算法中占用了很大一部分指令周期. 为了尽可能的减少其所占用的指令周期数, 在实现时对算法进行了优化. 对于维度为 6 的行, 采用如下的求最小值算法<sup>[4]</sup>.

假定  $a_i (i = 1, 2, \dots, 6)$  表示需要处理的数据信息, 即软信息的绝对值. 首先得到的中间结果为

$$\begin{aligned} x_{12} &= \min(a_1, a_2), & x_{34} &= \min(a_3, a_4), & x_{56} &= \min(a_5, a_6), \\ x_{1234} &= \min(x_{12}, x_{34}), & x_{1256} &= \min(x_{12}, x_{56}), & x_{3456} &= \min(x_{34}, x_{56}). \end{aligned}$$

将以上的  $x_{1234}, x_{1256}, x_{3456}$  3 个中间结果存储下来, 完成运算, 即  $b_1 = \min(a_2, x_{3456}), b_2 = \min(a_1, x_{3456}), b_3 = \min(a_4, x_{1256}), b_4 = \min(a_3, x_{1256}), b_5 = \min(a_5, x_{1234}), b_6 = \min(a_6, x_{1234}).$  然后, 乘以系数 0.75, 就可以得到校验节点运算的最终结果.

$b_i (i = 1, 2, \dots, 6)$  表示校验节点的处理结果, 也就是说对于维度为 6 的行, 只需要进行 12 次比较运算就可以得到相应的信息. 由于采用的 (1 008, 504) 的 LDPC 码有 6 和 7 的行维度, 对于维度为 7 的行, 每个元素在运算结果采用上述算法的基础上, 需要多一次比较运算, 也就是每行需要 19 次比较运算. 在乘以系数 0.75 时, 采用的方法是先用乘法指令乘以 3, 再将数据右移 2 位, 即除 4 操作.

处理符号时, 首先将行中所有信息的符号位存储下来, 并将这些符号位进行异或运算, 运算结果存储在一个变量 sign 中. 在对每个信息的符号位更新时, 将本身的符号位和变量 sign 异或, 就可以得到该节点的符号位. 也就是说, 对于维度为 6 的行, 需要 12 次异或运算来得到所有节点更新后的符号位.

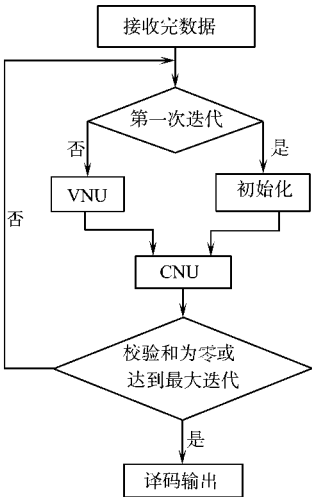


图 1 译码流程图

Fig. 1 Decoding process

2 DSP 实现和结构分析

2.1 测试平台

LDPC 码译码器的测试主要有 4 个模块: 信源、编码、信道和解码模块, 如图 2 所示. 单片机(MCU)通过串口从 PC 机上获得相应的参数, 包括译码帧数、信噪比等, 并将相应的参数传送到在 FPGA 上实现的 Sender 模块. Sender 模块包括信源发生模块、编码器、数字噪声发生模块. 此后, 每当编码器空闲时, 启动信源输出新的信码, 信源同时启动编码器完成新的一帧数据编码.

周而复始, 所有编码码字加入噪声后, 将 7 bit 量化后的码字传送到 DSP 芯片进行译码. 完成一帧的译码后, 译码完的码字传送到在 FPGA 上实现的 Checker 模块来获得相应的帧错误数和比特错误总数, 并将错误总数存储下来, 错误的数目通过 MCU 与 PC 机的串口传送到 PC 机上显示.

DSP 需要从 FPGA 上接收信道软信息进行译码, 译码结束后再向 FPGA 进行译码输出. 这里, 涉及到 FPGA 和 DSP 的通信协议, 即其发送和接收的握手流程. 在两者的通信中起主要作用的是 R/ W (读/ 写信号) 端口, R/ W 指示与外部器件通信期间数据传送的方向. 当 DSP 向 FPGA 请求数据时, R/ W 变为高电平(读方式); 当 DSP 执行一次写操作时, R/ W 变为低电平(写方式). FPGA 根据其相应的电平来与 DSP 通信.

2.2 测试结果

将程序通过硬件仿真器下载到测试平台上进行硬件仿真, 对两种译码流程进行测试比较. 在时钟主频为 20 MHz 时, 如果采用经典的译码流程, 利用数字示波器测试得到迭代一次的时钟周期为 15.2 ms. 迭代一次每比特完成所有的运算需要  $304\ 000/504 \approx 603$  条指令, 得到迭代一次的速率大概为  $504 \times 1\ 000/15.2 \approx 33\ \text{kbit} \cdot \text{s}^{-1}$ . 在 DSP 的时钟主频为 20 MHz 时, 如果采用提出的先进行变量节点运算再进行校验节点运算, 并且把校验和放入校验节点中运算的译码方式. 利用示波器进行测试, 得到迭代一次的时间周期为 12 ms. 也就是相应的指令周期数为  $12\ \text{ms} \times 20\ \text{MHz} = 24$  万条指令周期, 迭代一次每比特完成所有的运算需要  $240\ 000/504 \approx 470$  条指令, 迭代一次的速率大概为  $504 \times 1\ 000/12 \approx 42\ \text{kbit} \cdot \text{s}^{-1}$ . 如果在接收和发送数据时, DSP 的时钟主频为 20 MHz, 则在 DSP 译码时, DSP 的时钟主频 5 倍频为 100 MHz. 利用示波器进行测试, 迭代一次的时间周期为 2.4 ms, 可以得到迭代一次的速率大概为  $504 \times 1\ 000/2.4 \approx 210\ \text{kbit} \cdot \text{s}^{-1}$ .

对于不同信噪比( $R_{SN}$ )下的平均迭代次数( $n$ )进行测试, 结果如图 3 所示. 假定译码所需要的迭代次数为 10 次, 测试得到 DSP 的相应译码速率为  $20.4\ \text{kbit} \cdot \text{s}^{-1}$ . 该译码速率基本满足了低速通信系统的要求. 同时, 设置译码的最大迭代次数为 30 次, 共编译  $2 \times 10^6$  帧, 测试 DSP 译码器在不同比特信噪比( $R_{SN}$ )下的误码率( $R$ ), 并和 C++ 程序的无量化、C++ 程序的 7 bit 量化的仿真结果比较, 可以得到如图 4 所示的误码曲线.

2.3 测试分析

采用的 LDPC 码是不规则码, 需要大量的指令周期来寻址和读数. TMS320C5409 是使用较早的一款 DSP 芯片, 其 RAM 空间有 32 kB, 大概能完成码长为 5 000 的 LDPC 码的译码. 而目前使用较为广泛的 DSP 芯片是 TMS320C64X 系列芯片, 其 RAM 空间有 500 kB, 最高至少能完成 7 万码长的 LDPC 码的译码.

由图 4 给出的误码率曲线可以看到, 实际运行结果与 C++ 程序的 7 bit 量化仿真结果很接近, 略差于 C++ 无量化仿真的结果, 性能达到了设计要求.

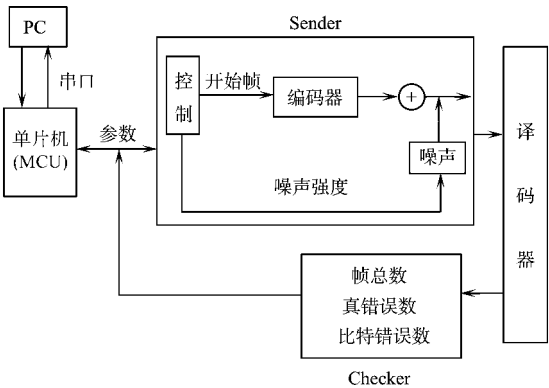


图 2 测试平台  
Fig. 2 Test platform

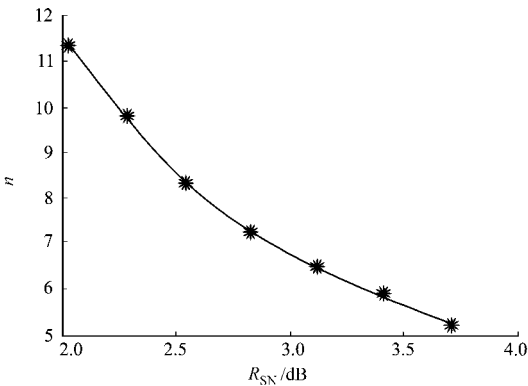


图 3 不同信噪比下的平均迭代次数

Fig. 3 Iteration number at different SNR

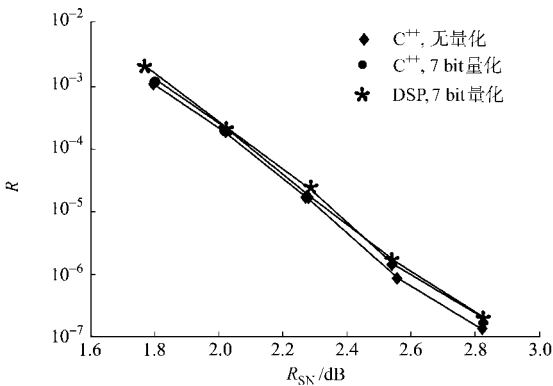


图 4 DSP 译码性能曲线

Fig. 4 Decoding performance on DSP

3 结束语

分析了 LDPC 码的 DSP 实现方法, 将和积算法中的具体运算进行了优化, 给出了与 FPGA 的通信方法、译码的具体流程。测试表明, 使用 DSP 实现 LDPC 的解码是一个可行、高效的方法; 同时, 由于 TMS320C5409 的廉价成本, 使得该方案对于高可靠性的低速率通信系统具有很高的实用价值。

参考文献:

[ 1 ] MACKAY D J C, NEAL R M. Near shannon limit performance of low-density parity-check codes[ J ]. Electron Lett, 1996, 32( 18 ): 1645-1646.

[ 2 ] HU X Y, ELEFATHERIOU E, ARNOLD D M. Regular and irregular progressive edge-growth tanner graphs[ J ]. Information Theory, 2005, 51( 1 ): 386-398.

[ 3 ] CHEN Jing-hu, TANNER R M, JONES C, et al. Improved min-sum decoding algorithms for irregular LDPC codes [ C ] // Proc 2005 IEEE International Symposium on Information Theory. Adelaide: [ s. n. ], 2005: 449-453.

[ 4 ] LECHNER G, SAYIR J, RUPP M. Efficient DSP implementation of an LDPC decoder[ C ] // 2004 IEEE International Conference on Acoustics, Speech, and Signal Processing . Montreal: [ s. n. ], 2004: 665-668.

Implementation of an LDPC on DSP

CHEN Yan

( College of Information Science and Engineering, Huaqiao University, Quanzhou 362021, China )

**Abstract:** This paper has done some research on the implementation of an LDPC decoder on DSP and adopted TMS320C5409 DSP. The sum-product algorithm was optimized and the decoding order was rearranged. The hard decision was calculated in variable nodes and the check sum was calculated in check nodes to avoid repetitive addressing. The method to communicate with FPGA was presented. This decoder is able to decode at  $20.4 \text{ kbit} \cdot \text{s}^{-1}$  on a TMS320C5409 DSP running at 20 MHz.

**Keywords:** low-density parity-check codes; digital signal processor; decoding; checking

( 责任编辑: 鲁斌 英文审校: 吴逢铁 )