

文章编号: 2096-1618(2018)06-0650-04

# 一种改进的块交织方法及 FPGA 实现

吕冠达, 杜雨谔, 吴方来, 辜建, 李怡呈

(成都信息工程大学电子工程学院, 四川 成都 610225)

**摘要:**无线图传系统中的块交织方法是对一帧数据进行交织。传统块交织方法将一帧数据完全写入交织块后再按序读出,这在一定程度上影响了数据流的连贯性。乒乓操作能实现数据同时读写,保证输入输出连续,但需要占用更多的系统资源。针对二者之间的矛盾,对原有块交织方法进行了改进。改进方法将交织矩阵扩展为一个正方形,数据执行行列交替读写。仿真结果表明,与传统块交织方法和乒乓操作方法相比,改进的块交织方法不仅能获得连续的数据输出,而且只消耗乒乓操作一半的系统资源。

**关键词:**块交织;输入输出连续;矩阵扩展;行列交替

**中图分类号:** TN929.5

**文献标志码:** A

**doi:** 10.16836/j.cnki.jcuit.2018.06.009

## 0 引言

OFDM 信号在无线信道中传输时会受到许多不确定因素的影响,使信号失真,影响通信质量<sup>[1]</sup>。交织技术能将原先聚集成片的误码分散,将突发性错误转化为随机性错误<sup>[2]</sup>,使错误码字个数在纠错码纠错范围内,交织同时也引入了系统延时。例如 802.11a 协议或其他的无线通信系统中均大部分使用块交织方法<sup>[3-6]</sup>,待交织数据全部写入交织块,再依次按序读出实现交织。

近年来,很多学者提出了不同的减少系统因交织运算而产生延时的方法<sup>[7]</sup>。使用乒乓操作的方法进行交织<sup>[8]</sup>,根据信道状态信息进行符号调整的自适应交织方法<sup>[9]</sup>,以及设计实现了基于 802.16e 的子块交织器<sup>[10]</sup>等。这些方法在不同程度上减少了系统时延,但都是传统块交织方法——读写操作在时间上是分开的,没有在根本上减少延时。乒乓操作虽是读写同时进行,但增加了资源的消耗。

基于此,改进了一种新提出的方法,将交织的二维长方形矩阵扩展为一个正方形<sup>[11]</sup>。对扩展后的矩阵第奇数次操作按行写入数据,第偶数次先按列读出第奇数次写入的数据,然后按列写入新数据。下一个奇数次操作又按行读出上一个偶数次写入的数据,随后按行写入新数据。循环往复,写延迟读一个时钟,所以读写几乎同时进行。在交织深度不变的情况下减少了系统时延和资源消耗。

## 1 原有块交织方法问题

对于一段长度  $L=M \times N$  的数据,传统块交织方法如图 1 所示。实现时用深度为  $L$  的 RAM 作为交织矩阵,是理想的交织存储器,数据按行完全写入 RAM 后再按列读出,完成交织。这种块交织方法在解交织后能将错误码字分散到相隔  $N-1$  个码字,从而实现将突发错误转换为随机错误<sup>[12-13]</sup>。但该方法需要等待矩阵写满之后再读,无法实现数据的连续读写。双口 RAM 因有两组独立的数据线和地址线,因此读写可同时进行<sup>[14]</sup>。利用双口 RAM 的读写时钟不同步虽可实现连续读写,但在交织领域,可能引起数据冲突,所以此种情况下并不适用。

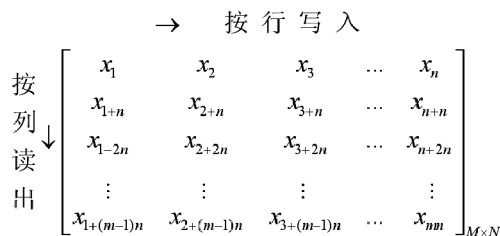


图1 传统块交织方法

若交织系统工作时钟周期为  $T$ ,使用该方法进行交织,每段长度  $L$  的数据产生的延迟时间为  $\Delta_L$ ,则

$$\Delta_L = T \times L \quad (1)$$

若共有  $X$  段待交织数据,即数据流长度  $S = X \times L$ ,则使用该方法实现交织的总延时为

$$\Delta_{S_1} = X \times \Delta_L \quad (2)$$

另一种是基于乒乓操作的块交织方法,乒乓操作流程如图 2 所示。该方法使用两个双口 RAM,通过控制读写使能及读写地址,实现同时读写,减少数据延时<sup>[15]</sup>。但这种方法增加了一倍的资源消耗。

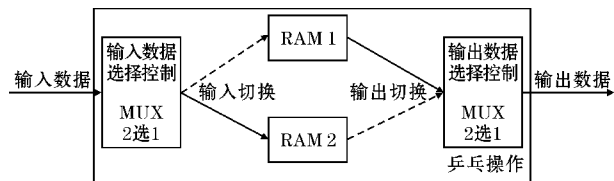


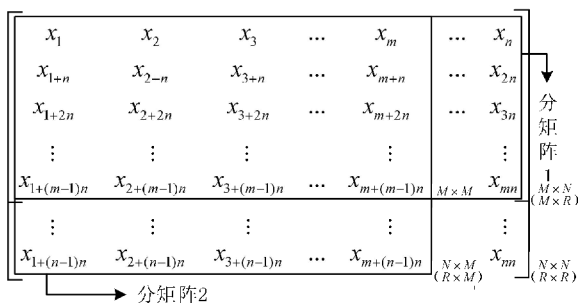
图2 乒乓操作流程

从图2中可以得出,基于乒乓操作的块交织方法的数据在延迟一个RAM深度之后能实现连续输出。同样长度为S的数据流,若使用乒乓操作,在资源消耗增加一倍的情况下数据输出总延时则为

$$\Delta S_2 = \Delta L \quad (3)$$

## 2 改进的块交织方法

将 $M \times N$ 大小的长方形交织器在短边进行扩展,扩展之后形成 $M \times M$  ( $M > N$ 时)或 $N \times N$  ( $M < N$ 时)的正方形矩阵,用 $R \times R$ 表示 ( $R \geq M, R \geq N$ ),即将 $M \times N$ 长方形矩阵扩展为 $R \times R$ 正方形矩阵。如图3所示, $M < N$ 情况下进行纵向扩展,即增加行数,将原来的 $M \times N$ 矩阵扩展为 $N \times N$ 。  $N \times N$ 大矩阵又分成两个分矩阵 $M \times N$ 和 $N \times M$  (中间重叠 $M \times M$ )。实现时用一个双口RAM充当大矩阵,读写时钟同步,写地址时钟始终滞后读地址时钟一个周期,这样保证两块分矩阵重叠部分的数据不会因读写冲突而发生错误。对双口RAM的第奇数次操作按照 $M \times N$ 矩阵进行按行写入新数据,按照 $N \times M$ 矩阵进行按行读出原数据;第偶数次操作则按照 $N \times M$ 进行按列写入新数据,按照 $M \times N$ 进行按列读出原数据。同理, $M > N$ 时即在横向进行列数扩展,本方法的关键为确定不同时刻的读写地址。

图3 交织矩阵扩展与分解 ( $M < N$  时)

按照图3情况将正方形大矩阵分为两个分矩阵,分矩阵内部地址变化如图4和图5所示。如果第奇数次操作是对分矩阵 $N \times M$ 执行按行读出,对分矩阵 $M \times N$ 执行按行写入;那么第偶数次操作则是对分矩阵 $N \times M$ 执行按列写入新数据,以及按列读出分矩阵 $M \times N$ 在上一个操作周期写入的数据。写地址时钟始终滞后

读地址时钟一个周期,所以对地址进行读写操作时不会产生冲突。

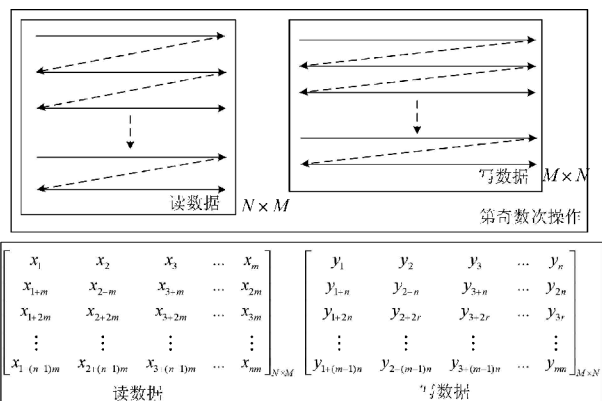


图4 第奇数次操作读写顺序

图4表示第奇数次操作时两个分矩阵内部读写地址在相同时刻的变化情况, $t$ 表示当前读写时刻, $r\_addr$ 和 $w\_addr$ 分别表示读地址和写地址,则第奇数次操作时读写地址变化可表示为:

$$r\_addr = \sum_{t=0}^{nm-1} x(t+1) \quad (4)$$

$$w\_addr = \sum_{t=1}^{mn} y(t) \quad (5)$$

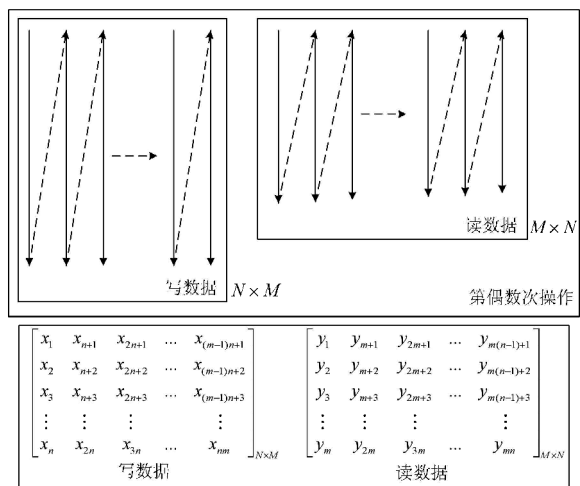


图5 第偶数次操作读写顺序

图5表示第偶数次操作时两个分矩阵内部读写地址在相同时刻的变化,第偶数次操作时不同时刻读写地址情况可表示为:

$$w\_addr = \sum_{t=1}^{nm} x(t) \quad (6)$$

$$r\_addr = \sum_{t=0}^{mn-1} y(t+1) \quad (7)$$

由图4、图5和公式(4)~(7)可以得出,改进后交织方法过程如图6所示,则扩展为 $R \times R$ 正方形的双口RAM读写地址控制可推出公式(8)和公式(9),其中 $P$ 为操作序数, $k$ 为自然序数。

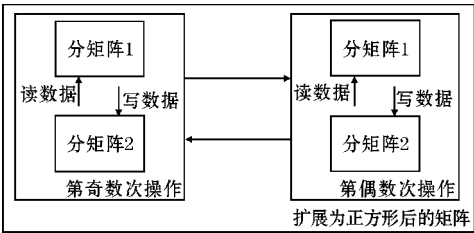


图 6 改进后交织方法流程图

$$r\_addr = \begin{cases} \sum_{t=0}^{nm-1} x(t+1), P=2k+1 \\ \sum_{t=0}^{mn-1} y(t+1), P=2k \end{cases} \quad (8)$$

$$w\_addr = \begin{cases} \sum_{t=1}^{mn} y(t), P=2k+1 \\ \sum_{t=1}^{nm} x(t), P=2k \end{cases} \quad (9)$$

根据设计,整个交织过程中,写当前时刻的数据与读前一个时刻的数据是同时进行的。数据刚被读出,存储器对应位置就有新数据进行写入。如此一直运行,就只需要一块扩展后交织矩阵,即向系统申请一个深度  $L'=R \times R$  的双口 RAM 即可完成数据的交织操作,既节省了资源,又减少了时延。

分析得出,改进块交织方法的延时为第一次写入 RAM 的时间。所以长度为  $S$  的数据流使用这种方法的数据输出总延时为

$$\Delta_{s_3} = \Delta_L \quad (10)$$

所以,在相同时延( $\Delta_{s_3} = \Delta_{s_2}$ )的情况下,改进的块交织方法比乒乓操作方法又减少了近一半的资源损耗。

3 FPGA 仿真实现

此处对改进的交织优化方法进行仿真验证,并与改进前的交织方法进行比较。仿真及对比过程中,系统工作频率100 MHz,假定数据56 bit组成1 帧,实验模拟 5 帧共280 bit数据通过交织。分别使用 3 种方法对数据进行交织运算:传统块交织方法,块交织矩阵使用 7×8,结交织后错误码字被分散到相隔7 bit;基于乒乓操作的块交织方法,两块深度为 56,位宽为1 bit的双口 RAM 作为交织缓存进行 7×8 块交织;改进的块交织方法,将 7×8 矩阵扩展为 8×8 的正方形,再将正方形矩阵分为 7×8 和 8×7 的两个重叠的分矩阵,FPGA 里即申请深度 64,位宽1 bit的双口 RAM。

将运算结果进行对比,并分析 3 种方法的时延与资源消耗情况。根据设计,编写 MATLAB 程序作为数据对比标准,并编写 3 种方法下的 FPGA 程序。

图 6 给出了 FPGA 里采用传统块交织方法、基于乒乓操作块交织方法、改进块交织方法与 MATLAB 交织运算的结果对比,其中横坐标表示数据序号,纵坐标为数据比特信息。从图 7 中离散脉冲线段可以看出,3

种方法的运算结果均与 MATLAB 运算结果一致。以此证明,3 种交织方法运算结果都是正确的。

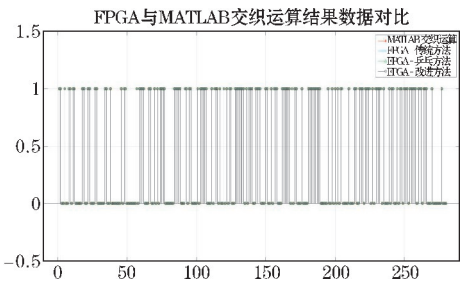


图 7 FPGA 交织运算结果与 MATLAB 运算结果对比

图 8,图 9 和图 10 分别给出了采用传统块交织方法,基于乒乓操作的块交织方法和改进块交织方法进行数据交织的 FPGA 运行结果。从图中可以看出,采用传统块交织方法运算时,数据需等待当前数据全部输出后才能进行新数据的输入,即每帧数据都会延迟 56 个时钟,不便于数据的流水线操作。采用乒乓操作的块交织方法和改进块交织方法则能保证整体数据在延迟 56 个时钟后不间断的连续输出。

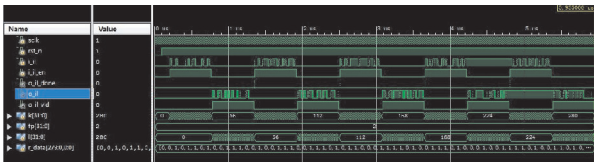


图 8 传统块交织方法运算结果

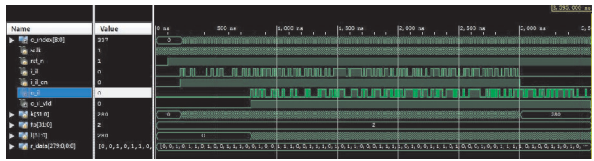


图 9 基于乒乓操作块交织方法运算结果

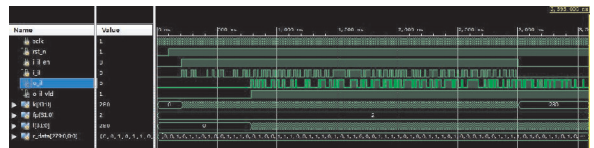


图 10 改进块交织方法运算结果

对 3 种方法的 FPGA 运算产生延时进行计算并分析内部资源消耗情况,结果如表 1 所示。

表 1 3 种块交织方法延时与资源消耗对比

| 方法         | 总延时/ns | 资源消耗/Slice Register |
|------------|--------|---------------------|
| 传统的块交织     | 2800   | 21                  |
| 基于乒乓操作的块交织 | 560    | 48                  |
| 改进的块交织     | 560    | 27                  |

从表 1 可以得出,总延时方面,传统块交织方法产生的延时会随交织数据量增加而成倍增加;基于乒乓操作的块交织方法和改进块交织方法产生的延时是第

一次存入数据的时间,之后就可以保证数据的连续输出。在资源消耗方面,基于乒乓操作的块交织方法是传统块交织方法的两倍多,是改进块交织方法的接近两倍。综合判断,与传统块交织方法和基于乒乓操作块交织方法相比,改进的交织方法可以在实现数据流水线读写的同时又节约近一半的系统资源。

## 4 总结

改进了 OFDM 系统中的交织算法,该方法克服了单个交织矩阵无法实现连续交织的局限性及基于乒乓操作运算的资源耗费问题,采用将二维矩阵扩展为正方形的通过行列交替读写实现块交织,增加了数据的连贯性,大大节约了系统资源。仿真结果表明,与传统块交织方法和基于乒乓操作的块交织方法相比,改进的块交织方法不仅能获取更流畅的数据输出,而且只需乒乓操作方法约一半的系统资源,实现了 FPGA 面积和速度二者矛盾的突破,达到了改进的目的。

## 参考文献:

- [1] Zou W Y, Wu Y. COFDM: an overview[J]. IEEE Transactions on Broadcasting, 1995, 41(1): 1-8.
- [2] Barbulescu A S, Pietrobon S S. Interleaver design for turbo codes[J]. Electronics Letters, 1994, 30(25): 2107-2108.
- [3] 史治国, 洪少华, 陈抗生. 基于 Xilinx FPGA 的 OFDM 通信系统基带设计[M]. 杭州: 浙江大学出版社, 2009.
- [4] 王晓玲. CRC 交织级联编码在 QPSK 通信系统中的应用研究[J]. 微电子学与计算机, 2018(5).
- [5] 王灵垠, 刘琚. 降低 OFDM 系统峰均功率比的时频联合块交织方法[J]. 山东大学学报(理学版), 2012, 47(11): 83-87.
- [6] 刘翠海, 马文骄, 马新. 甚低频通信系统的差错控制方案及其性能仿真[J]. 电讯技术, 2015, 55(12): 1360-1364.
- [7] 张凯斌, 殷柳国, 陆建华. 低误码平底 LDPC 码的块交织构造算法[J]. 清华大学学报(自然科学版), 2010(1): 153-155.
- [8] 陈思宇. 基于 COFDM 无线高速数字图像传输系统发送端研究实现[D]. 成都: 电子科技大学, 2009.
- [9] Lei S W, Lau V K N. Performance analysis of adaptive interleaving for OFDM systems[J]. Vehicular Technology IEEE Transactions on, 2002, 51(3): 435-444.
- [10] 严成. 基于 802.16e 的子块交织器设计与实现[C]. 全国青年通信学术会议, 2006.
- [11] 张善旭. 一种块交织的交织及解交织方法[J]. 信息通信, 2013(7): 35-36.
- [12] 西瑞克斯通信设备有限公司. 无线通信的 MATLAB 和 FPGA 实现[M]. 北京: 人民邮电出版社, 2009.
- [13] Mehedy L, Bakaul M, Nirmalathas A. Frequency interleaving towards spectrally efficient direct detection based optical OFDM systems[C]. proceedings of the Optoelectronics and Communications Conference, 2010.
- [14] 钱黄生, 夏忠珍. 基于 FPGA 双 RAM 乒乓操作的数据存储系统的研究[J]. 科技信息, 2010(21): 508-515.
- [15] 王智, 罗新民. 基于乒乓操作的异步 FIFO 设计及 VHDL 实现[J]. 信息化研究, 2005, 31(6): 13-16.

## An Improved Method of Block Interleaving and FPGA Implementation

LV Guan-da, DU Yu-ming, WU Fang-lai, GU Jian, Li Yi-cheng

(College of Electronic Engineering, Chengdu University of Information Technology, Chengdu, 610225)

**Abstract:** The block interleaving method in the wireless image transmission system is to interleave a frame of data. The traditional method of block interleaving writes one frame of data into the interleaving block completely and then reads it in order. This affects the coherence of the data flow to some extent. It requires more system resources for ping-pong operation to read and write data at the same time, which ensuring continuous input and output. Considering the contradiction between the two methods, this paper improves the original block interleaving method. The improved method expands the interleaving matrix into a square, with data rows and columns alternately reading and writing. The simulation results show that, compared with the traditional block interleaving method and the ping-pong operation method, the improved method not only can obtain continuous output data, but also consumes half of the system resources in ping-pong operation.

**Keywords:** block interleaving; input and output continuously; matrix extension; switch in row and column