一种 LZ4 压缩算法的 FPGA 硬件加速设计方法

王佳伟[1] 盛庆华[1] 秦 宽[1] 陈 凯[1] 单 铮[1] 黄小芳[2]
WANG Jiawei SHENG Qinghua QIN Kuan CHEN Kai SHAN Zheng HUANG Xiaofang

摘 要 随着信息技术和互联网的快速发展，数据量呈现爆炸式增长，对高效无损压缩算法的需求日益迫切。为解决传统软件压缩算法在存储系统中资源占用过多和压缩速度较慢的问题，提出了一种基于 FPGA 的 LZ4 无损压缩算法硬件加速系统。该系统包括 UART 串口收发模块、LZ4 无损压缩模块和 XXH32 校验码生成等模块，并在 Xilinx AXU2CGB 平台上进行了验证。实验结果表明，该系统不仅成功实现了数据的 LZ4 无损压缩，还在保持与软件压缩几乎相同压缩比的前提下，大幅提升了压缩速度。

关键词 LZ4；无损压缩算法；XXH32；UART；FPGA

doi：10.3969/j.issn.1672-9528.2025.02.044

0 引言

随着信息技术的迅猛发展，世界进入了数据驱动的时代，全球数据总量快速增长已成为共识[1]，数据存储、传输和处理的需求以前所未有的速度增长。这对存储设备容量、网络带宽和数据处理能力提出了更高要求，推动了无损压缩算法需求的增加[2]。无损数据压缩在保留数据信息完整性的前提下，通过重组编码降低数据冗余，减少数据量[3]，这对于卫星图像、医学图像[4]、大数据[5]等领域至关重要。

自从以色列研究者 Ziv 等人[6]提出 LZ77 压缩算法以来，逐渐涌现出多种基于字典匹配的 LZ 压缩算法的衍生算法。LZ 系列压缩算法作为当前应用最为广泛的一种压缩算法，其显著的特点在于对一定长度的字符串进行标记，而不是对单个符号进行编码[7]。Collet 于 2011 年提出了 LZ4 算法[8]，该算法是对 LZ77 算法的速度优化版本，比其他 LZ77 系列算法实现了更高的压缩和解压吞吐率[9]，在许多高吞吐量系统中被广泛采用。目前的 LZ4 压缩算法是以 x86 架构的计算机为主要目标，并且在处理效率和资源占用方面存在明显不足[10]。因此通过 FPGA 实现 LZ4 压缩可以释放存储系统中的运算资源同时可以提高压缩效率。

1 系统总体设计

LZ4 无损压缩系统的结构组成框图如图 1 所示。该系统包括串口接收、字节缓冲缓存模块、绝对地址生成模块、字节分割模块、Hash 匹配模块、编码模块、LZ4 控制状态机模块、XXH32 校验字生成和串口发送等模块。LZ4 无损压缩系统的工作原理是通过串口接收到原数据文件和原数据文件长度，然后把数据分成两路，一路先用 FIFO 缓存起来以待所有数据缓存完成再送入 LZ4 压缩模块进行压缩以及编码，另一路数据来进行 XXH32 校验字的生成，然后生成的 XXH32 校验字送入 LZ4 无损压缩模块中用来编码。最后把编码压缩完成的数据文件通过串口发送。

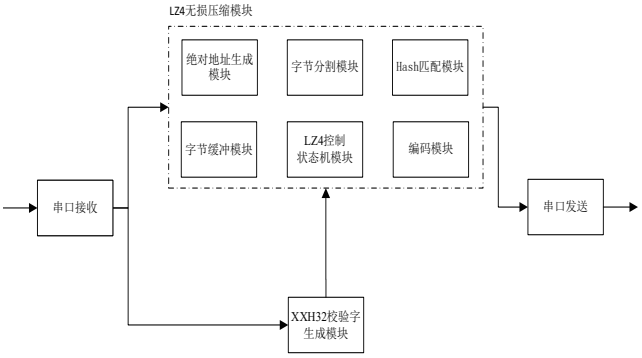

图 1 总体设计框图

1.1 LZ4 编码标准

LZ4 无损算法属于 LZ77 系列的压缩算法，是一种字典类型的无损压缩算法。LZ4 无损压缩的压缩文件格式如表 1 所示，一个完整的 LZ4 压缩文件包含了文件标识符、文件描述符和若干个数据块结构，以及结束符和校验字。数据块包含了数据块长度、数据块和数据块校验和。

表 1 LZ4 编码文件结构

	文件标识符	文件描述符	数据块长度	数据块
字节数	4	3~5	4	0~*n*
	数据块校验和	…	结束符	校验字
字节数	0~4	…	4	4

1. 杭州电子科技大学电子信息学院 浙江杭州 310018
2. 杭州电子科技大学信息工程学院 浙江杭州 311305
[基金项目] 国家级大学生创新创业训练计划支持（202310336045）

本文设计的文件描述符结构如表 2 所示，文件描述符包含了 FLG、BD、内容大小、字典 ID 以及 HC。FLG 这 1 个字节用来控制文件的模式，在这一个字节中第 6~7 位是文件的版本号，第 5 位是块独立标志，第 4 位是块校验和标志，第 3 位是内容大小标志，第 2 位是内容校验和标志，第 1 位保留，同时保留位都必须为 0，第 0 位为字典 ID 标志。BD 字节包含了第 7 为保留，第 4~6 位为最大数据块的大小，第 0~3 位是保留。在文件描述符后面接数据块的长度，数据块内包含了多个数据块结构，如表 3 所示。token 是一个字节值，分为两个 4 位字段，每个字段的范围从 0 到 15，高 4 位代表着未匹配的长度，低 4 为表示匹配的长度减去 4，因为在 LZ4 中最小的匹配长度为 4 个字节。如果 token 的高 4 位为 15，则需要额外的未匹配字符长度来做补充，在额外的未匹配字符长度是以 0xFF 为标志代表后面还有额外的未匹配字符长度，对于额外的匹配字符长度则看 token 的第 4 位。未匹配字符就是待压缩文件中没有得到匹配的部分，直接原文复制到数据块中。匹配偏移则是表示要从之前的数据复制匹配的位置。本设计在 FLG 中设置了数据块校验和标志为 0，所以没有数据块校验和的部分。结束符为固定的 0x00000000，校验字是对待压缩文件原文进行 XXH32 快速摘要算法生成的。

表 2 文件描述符结构

	FLG	BD	内容大小	字典 ID	HC
字节数	1	1	0~8	0~4	1

表 3 数据块结构

	token	额外的未匹配字符长度	未匹配字符	匹配偏移	额外的匹配字符长度
字节数	1	0~*n*	0~*n*	2	0~*n*

1.2 LZ4 压缩原理

LZ4 无损压缩算法是在最初的 LZ77 算法的基础上，通过改变压缩格式和简化压缩步骤得到的 [11]，该算法基本工作原理是通过扫描输入的数据流，寻找并记录下重复出现的数据序列，这些被记录下来的重复数据序列被称之为“匹配项”，LZ4 无损压缩算法会用特殊的格式来表示这些匹配项的位置和长度，从而实现了数据的压缩。具体来说，LZ4 无损压缩算法会输出一个标记（token），这个标记包含了指向之前记录下来数据中匹配项的相对位置偏移以及匹配项的长度。

LZ4 压缩过程如图 2 所示，主要经过生成文件头、读入字节进行 Hash 值运算匹配、读入后续字节进行扩展匹配以及更新 Hash 表。Hash 值计算采用 32 位的斐波那契散列法，也就是利用黄金分割数来做乘数计算 [12]，其公式为：

$$Hash = value \times 2\,654\,435\,761 >> 17 \quad (1)$$

然后取 Hash 的低 15 位为字符对应用来存储 Hash 表中 Hash 计算标志和字符在原始数据中的绝对位置以及字符的原文的地址。在与 Hash 表匹配的过程中，利用计算 4 个字节的 Hash 值而得到的地址查找 Hash 表完成匹配。更新 Hash 表的过程也是对 Hash 计算标志和字符在原始数据中的绝对位置以及字符的原文以 Hash 值的低 15 位作为地址存储更新。

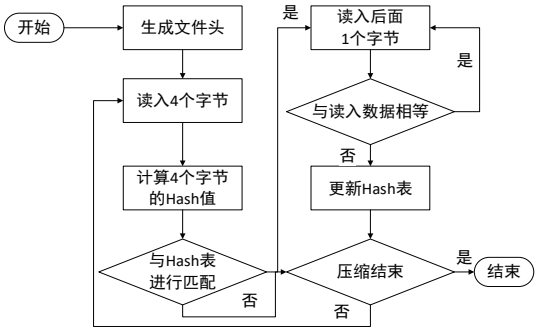

图 2 LZ4 压缩处理流程

1.3 XXH32 快速摘要算法

XXH32 校验字生成模块采用的算法是快速摘要算法里哈希算法中的一种，由 Chen Kaige 在 2012 年开发，主要用于有效地处理大量数据。XXH32 校验字生成算法以极端的速度为设计目标，生成 32 位固定长度的校验字，该算法适用于快速数据比较和数据完整性检查。XXH32 校验字生成算法通过简化的多项式计算来混合输入数据，并且确保了较低的冲突率。但是由于该算法的设计不侧重于安全性，所以 XXH32 不应用于加密或高安全性要求的场合，但特别适用于那些需要高速数据处理但不需要高安全性的应用场合。正是 XXH32 运算快并且能验证数据是否有变化，所以 LZ4 无损压缩文件格式里选择了 XXH32 快速摘要算法作为校验字。

2 FPGA 逻辑设计

2.1 LZ4 压缩模块

LZ4 无损压缩模块结构图如图 3 所示，LZ4 无损压缩模块包含了字节缓冲模块、绝对地址生成模块、字节分割模块、Hash 匹配模块、编码模块和 LZ4 控制状态机模块。

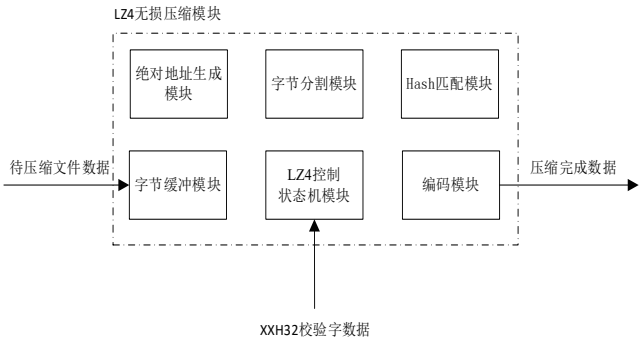

图 3 LZ4 无损压缩模块结构图

数据通过串口传输到字节缓冲模块中，并且字节缓冲模块会根据字节分割模块和 Hash 匹配模块的需要来传递数据。各个模块都受 LZ4 控制状态机模块的控制，数据也会经过 LZ4 控制模块来传递。压缩完成的数据存储在编码模块的输出缓存中，当压缩完成后串口发送模块会读取输出缓存中的数据并发送出去。

2.1.1 字节缓冲模块

字节缓冲模块的主要功能是用来缓存待压缩数据，等待后续字节分割模块和 Hash 匹配模块的使用。字节缓冲模块与串口接收模块相连接，串口模块接收待压缩数据后经过一个 FIFO 传入字节缓冲模块内的双口 RAM 中。后续字节分割模块和 Hash 匹配模块分别经过一个 FIFO 和 RAM 从字节缓冲模块里读取数据。

2.1.2 绝对地址生成模块

绝对地址生成模块的功能如图 4 所示，功能就是对于输入数据不停地生成对应的绝对地址以及后 3 个字节的地址，用来给后续的 Hash 匹配模块提供绝对地址以计算对应的字符偏移和在 Hash 表中作为存储的数据。

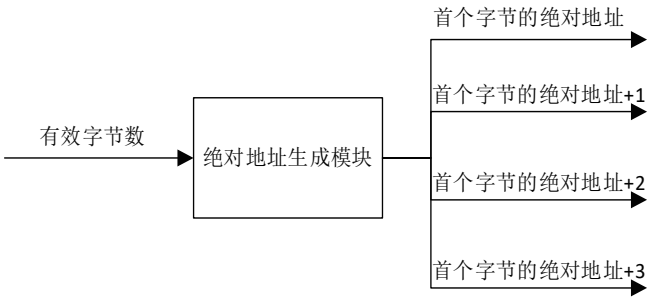

图 4 绝对地址计数生成模块

2.1.3 字节分割模块

字节分割模块的功能如图 5 所示，能提前读取 4 个字节的待压缩数据，并向后移位取出四组 4 个字节数据送入 LZ4 无损压缩模块的控制状态机中，然后送入扩展匹配模块中用来扩展匹配，也就是完成了字符的分割。在字节分割模块内有一个 12 个字节大小的缓存 FIFO，这个 FIFO 用于对之前非整四个字节的数据进行重新拼接。

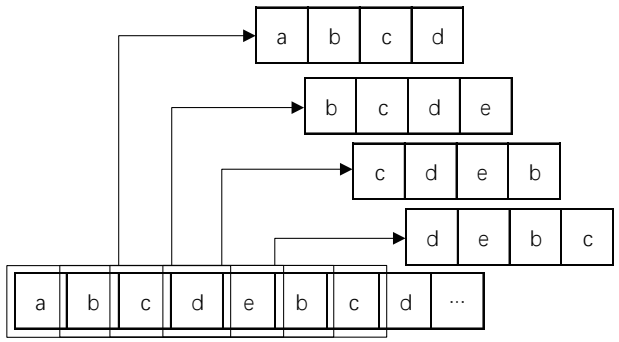

图 5 字符分割模块功能图

2.1.4 Hash 匹配模块

Hash 匹配模块状态机如图 6 所示，Hash 匹配模块包含 7 种状态，分别是空闲状态、Hash 表清除状态、Hash 表写状态、Hash 匹配地址计算状态、Hash 匹配更新状态、等待状态和停止状态。Hash 匹配模块主要是把输入的要匹配的待压缩数据进行 Hash 匹配以及后续的扩展匹配，然后把匹配的结果即匹配成功信号和相对位置输出到整个 LZ4 无损压缩模块的控制状态机中，以便于后续编码模块生成数据块 token 和对应的信息。

2.1.5 编码模块

编码模块状态机如图 7 所示，编码模块主体包含 13 种状态，分别是空闲状态、幻方数写入状态、文件信息写入状态、数据块长度写入状态、标记写入状态、额外匹配字符长度写入状态、未匹配字符数据写入状态、匹配偏移写入状态、额外匹配长度写入状态、数据块结束状态、结束符写入状态、XXH32 校验字写入状态和结束状态。其中还有两个状态用来等待前级数据输入和输出缓存到正常，写入到输出缓存的方式为有效字节个数和缓存数据寄存器。编码模块的输入来自

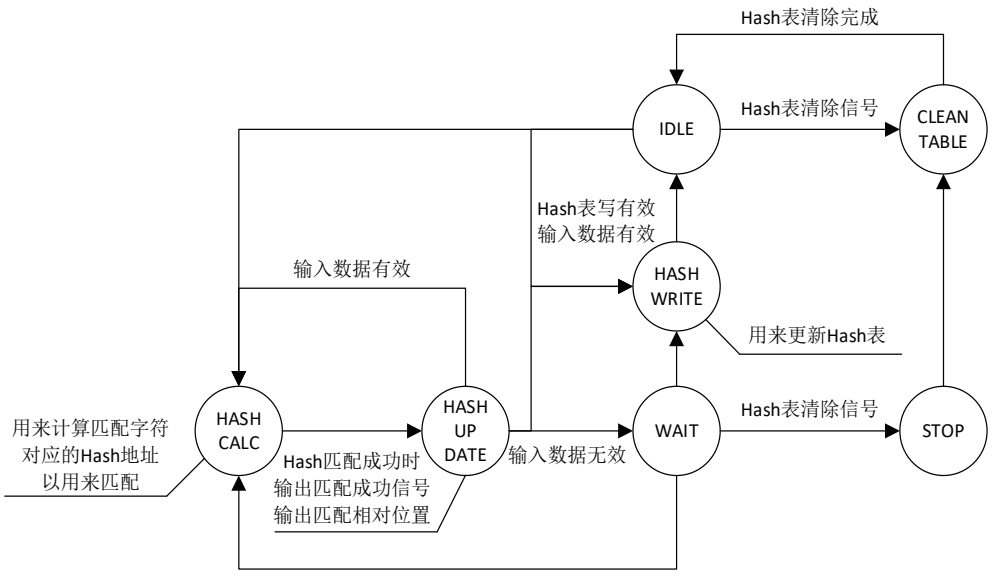

图 6 Hash 匹配模块状态机

匹配模块和 XXH32 校验字模块，以及收到 LZ4 控制状态机模块的控制。

2.1.6 LZ4 控制状态机模块

LZ4 控制状态机模块状态转移图如图 8 所示，LZ4 控制状态机模块是协调各个模块数据的中介，LZ4 控制状态机模块主要负责控制各个模块和对应的数据传输。

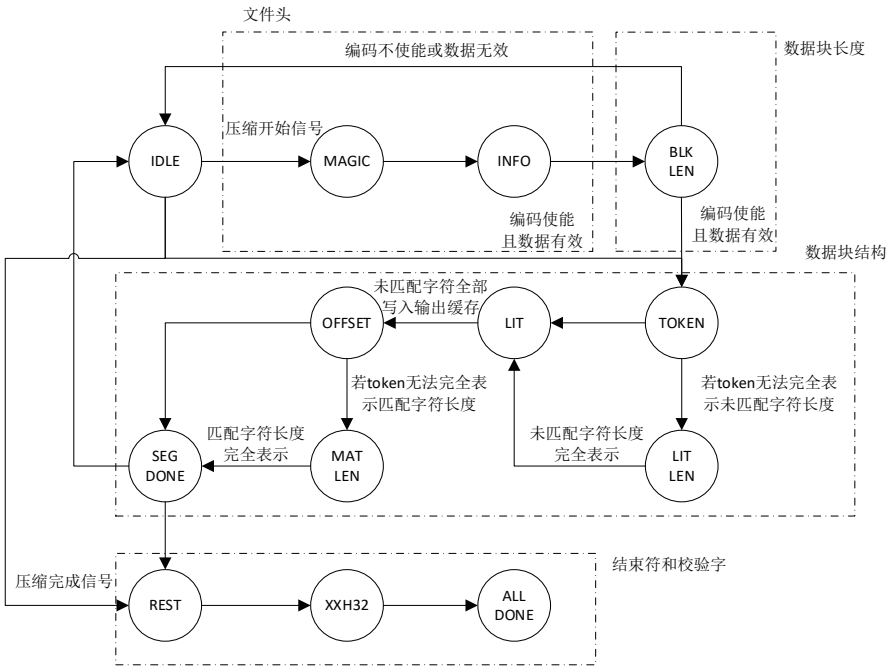

图 7 编码模块状态机

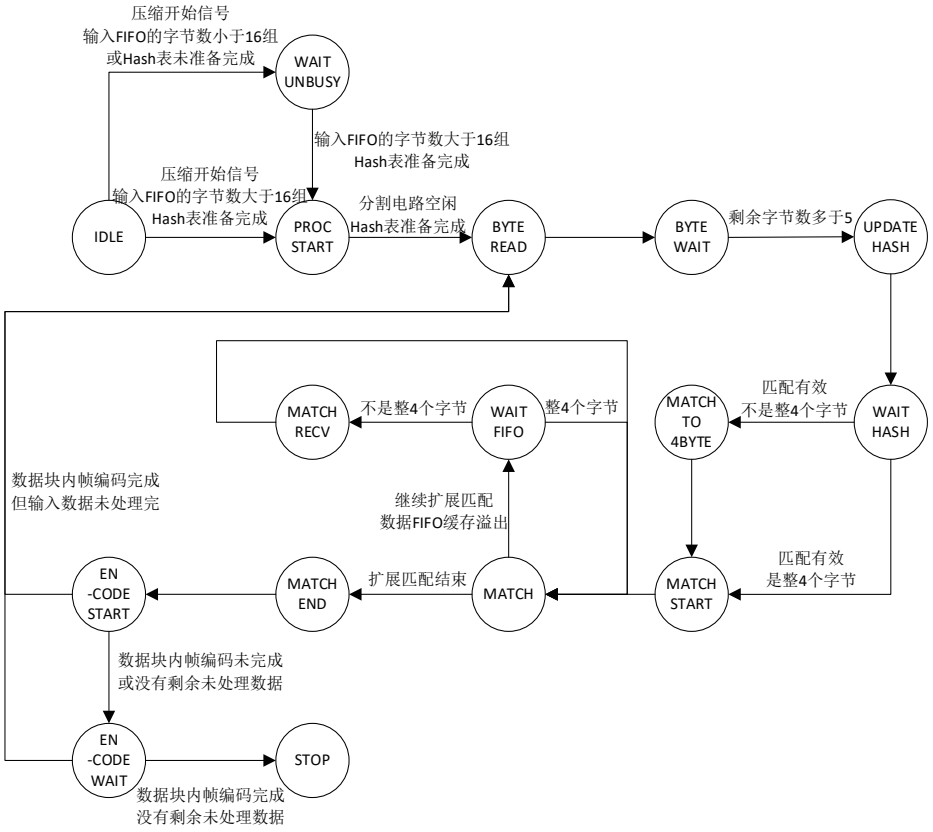

图 8 LZ4 控制状态机模块状态转移图

2.2 XXH32 校验字生成模块

XXH32 校验字生成流程如图 9 所示。校验字生成模块分成两个部分，分别用来处理非最后 16 个字节以内的数据和剩余字节。这种对输入到校验字生成模块的所有数据都进行了计算操作，能很好地验证文件的正确性。同时在校验字生成的过程中利用了大量的 ROTL 循环移位操作和各种位操作，通过并行化和流水线化来提高整体的计算速度和性能。其中 ROTL 循环移位的公式为：

$$\begin{aligned} &\mathrm{ROTL}(value, shift) \\ &= (value << shift) | \\ &[value << (32-shift)] \end{aligned} \quad (2)$$

3 测试方案与测量结果

测试方案：验证平台由 Xilinx AXU2CGB 板卡与通用计算机构成，使用 Calgary 与 Canterbury 语料库中的文件进行测试，FPGA 工作频率为 200 MHz，通过 UART 通信从计算机接收数据并进行压缩，测试压缩比与压缩速度，并与刘勇等人在 Core i3 3220，工作频率 3.3 GHz，Windows 7 64 位操作系统的通用计算机上测试的软件压缩结果进行对比，测试结果如表 4、表 5 所示。由于硬件压缩工作频率为 200 MHz，而软件压缩工作频率为 3.3 GHz，因此通过测量硬件压缩所消耗的时钟数并折算其对应 3.3 GHz 工作频率下的压缩速度进行对比。

测试结果分析：从测试结果可以看出，硬件 LZ4 的压缩比与软件相差较小，同时在相同工作频率下压缩速度约为软件压缩的 10 倍。

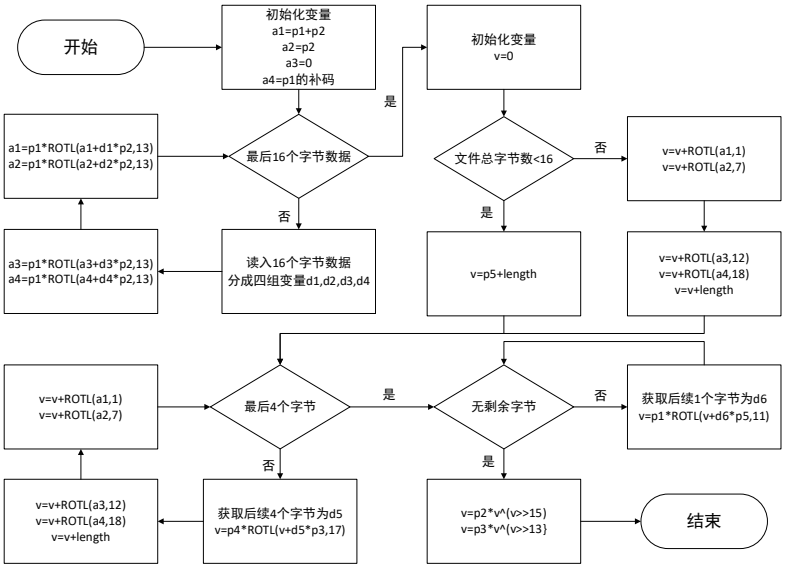

图 9 XXH32 校验字生成流程图

表 4 压缩比比较

测试文件	软件压缩比 /%	硬件压缩比 /%
bib	52.2	51.1
obj1	60.1	60.5
paper2	59.1	59.1
progc	52.8	45.5
trans	32.4	35.7
alice29.txt	59.7	61.4
asyoulik.txt	63.1	64.0
cp.html	48.4	48.3
field.c	46.8	47.7
grammar.lsp	51.4	52.4
sum	49.2	49.7
xargs.1	62.9	62.8

表 5 压缩速度比较

测试文件	软件压缩速度 /(MB·s^{-1})	消耗时钟数 / Cycle	硬件压缩速度 /(MB·s^{-1})
bib	237.2	121 193	2 889.2
obj1	221.7	21 198	3 192.6
paper2	210.2	117 603	2 199.7
progc	277.0	42 987	2 899.9
trans	367.4	83 152	3 546.2
alice29.txt	218.2	198 393	2 412.6
asyoulik.txt	200.3	165 574	2 379.3
cp.html	276.4	24 665	3 139.2
field.c	227.6	12 449	2 818.7
grammar.lsp	186.1	3960	2 957.2
sum	316.0	39 890	3 016.9
xargs.1	156.6	5042	2 638.4

4 结语

本文通过 UART 串口收发模块、LZ4 无损压缩模块和 XXH32 校验字生成等模块实现了对数据的 LZ4 硬件无损压缩，并在 Xilinx AXU2CGB 硬件平台上成功运行，通过 UART 通信从计算机接受数据进行压缩，从测试结果来看该方案在保持与软件 LZ4 压缩近乎相同的压缩比的同时，在相同工作频率下压缩速度约为软件压缩的 10 倍。

参考文献：

[1] 辛冬播 , 李廷军 . 大数据时代数据总量增长的新摩尔定律辨析 : 吉姆·格雷是否真的提出过新摩尔定律？ [J]. 软件导刊 , 2022, 21(9): 236-240.

[2] 顾巍 . 基于 FPGA 的 LZ4 无损压缩算法优化设计 [D]. 南京 : 东南大学 ,2017.

[3] SAYOOD K. Introduction to data compression[J/OL]. Multimedia information and systems, 2018[2024-06-12].https://www.sciencedirect.com/book/9780128094747/introduction-to-data-compression#book-description.

[4] LEIVA L, VAZQUEZ M, TOSINI M, et al. FPGA based implementation of imageZero compression algorithm[J]. IEEE latin america transactions, 2020, 18(2): 344-350.

[5] QIAO W K, DU J Q, FANG Z M,et al.High-throughput lossless compression on tightly coupled CPU-FPGA platforms[C/OL]//2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines(FCCM).Piscataway: IEEE, 2018[2024-06-03].https://ieeexplore.ieee.org/abstract/document/8457630.

[6] ZIV J，LEMPEL A. A universal algorithm for sequential data compression [J]. IEEE transaction on information theory, 2003, 23(3)：337-343.

[7] 李小遐 , 高杨 . 一种基于字典的无损压缩改进算法研究 [J]. 自动化与仪器仪表 , 2016(2):123-124.

[8] COLLET Y. RealTime data compression: development blog on compression algorithms[EB/OL].(2011-05-26)[2024-08-26].https://fastcompression.blogspot.com/2011/05/lz4-explained.html.

[9] 刘谱光 , 魏子令 , 黄成龙 , 等 . 一种基于 FPGA 加速的高性能数据解压方法 [J]. 计算机学报 ,2023,46(12): 2687-2704.

[10] 刘勇 , 郭建刚 , 方震 . 一种 LZ4 无损压缩电路设计 [J]. 电子技术应用 , 2022, 48(12):59-64.

[11] 吴涛 . 基于 LZ4 算法的无损压缩硬件设计与 WIFI 传输 [D]. 南京 : 东南大学 , 2021.

[12] 蒋浩 . 基于无进位乘法的滚动哈希及其在 LZ4 数据压缩中的实现 [D]. 合肥 : 中国科学技术大学 ,2021.

【作者简介】

王佳伟（2002—），男，浙江绍兴人，本科，研究方向：FPGA 硬件加速。

盛庆华（1978—），男，浙江兰溪人，硕士，副教授，研究方向：FPGA 硬件加速、电子系统集成等。

黄小芳（1984—），女，浙江浦江人，硕士，工程师，研究方向：视频编码、嵌入式应用等。

（收稿日期：2024-10-18）