第 34 卷 第 6 期 计 算 机 工 程 2008 年 3 月
Vol.34 No.6 Computer Engineering March 2008
·多媒体技术及应用· 文章编号：1000—3428(2008)06—0229—04 文献标识码：A 中图分类号：TN919.81

# 基于 H.264 的熵编码结构

何 俊，田应洪，洪志良
(复旦大学专用集成电路与系统国家重点实验室，上海 201203)

**摘 要：**Exp-Golomb 和 CAVLC 是 H.264 引入的新的熵编码形式，通过引入上下文的方式，减少编码码流，提高鲁棒性。该文提出一种熵编码的硬件结构，采用全 0 子块探测，双 RAM 结构，流水线技术，以及通过计算代替查找表的方法，加快编码过程，同时减少硬件的复杂度。FPGA 综合结果显示，关键路径为 11.449 ns，系统时钟最高支持到 87.346 MHz。

**关键词：**基于内容的变长编码；指数哥伦布编码；H.264 协议；超大规模集成电路

# VLSI Architecture for H.264 Entropy Encoding

HE Jun, TIAN Ying-hong, HONG Zhi-liang
(ASIC & System State Key Laboratory, Fudan University, Shanghai 201203)

**【Abstract】**This paper presents a high performance VLSI architecture design for H.264/AVC entropy encoding. In the design, a fast encode all-zeros block method is proposed to accelerate the encoding process. A double RAM method is designed to avoid the mistake of storage and be easy to control data as a block. A pipeline technology is proposed to erase the RAM and make the process faster. The synthesis result shows the proposed architecture can improve the performance of the CAVLC.

**【Key words】**Context based Adaptive Variable Length Coding(CAVLC); Exp-Golomb; H.264; Very Large Scale Integration(VLSI)

在视频处理中，视频压缩扮演重要角色。作为一种新的视频压缩编码，H.264 在相同的压缩质量情况下，码率相对于MGPEG-4, H.263和MPEG-2，分别减少 39%, 49%和 64%[1]。H.264 首次引入指数哥伦布(Exp-Golomb)编码和CAVLC(Context based Adaptive Variable Length Coding)2 种熵编码分别对参数和残差进行编码。

## 1 编码基本原理

图 1 为H.264[2]编码的结构图，熵编码的原理是通过对出现频率较高的数据采用较短码字，对于出现频率较低的数据采用较长码字，使平均码字接近熵的大小。这里采用熵编码对参数和数据进行进一步的压缩，将十进制的数据压缩后，采用二进制的码流表示，并且按照一定顺序封装输出。

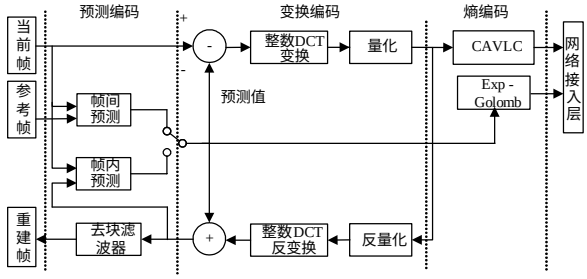


**图 1 H.264 编码器**

### 1.1 Exp-Golomb 基本原理

指数哥伦布编码是对参数编码，指数哥伦布编码过程分为 2 个步骤：(1)将参数从有符号数变为无符号数，(2)对参数进行指数变长编码。指数编码由前缀和后缀组成，前缀由一串连续的 0 以及末尾的一位 1 组成，0 的个数表明前缀的大小，同时表明后缀的长度。在解码时，通过查找码流中出现的第一个 1 前 0 的个数，可以得到前缀的大小，同时得到后缀的长度，按照后缀长度从码流中读出相应长度的码流作为后缀，最后通过前缀和后缀的大小就可以得到编码值的大小。H.264 中采用分层打包的概念，将序列参数、图像参数和片分开打包传输。图 2 为输出码流的结构图，其中序列参数、图像参数和片参数提前设置，不需实时编码，而宏块参数则是在编码的过程中产生，需要进行实时的编码。

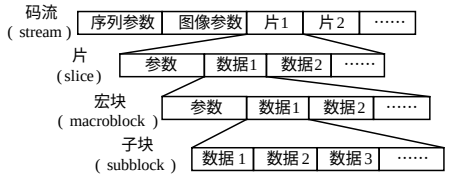


**图 2 码流组织形式**

### 1.2 CAVLC 基本原理

CAVLC[3]是对数据进行的编码，在一块 6×16 的宏块中，按照亮度、色度的顺序对 4×4 子块的数据依次进行编码，每个子块的编码由 5 部分组成，分别是非 0 系数标记(CoeffToken)、正负 1 标记(TrailingonesFlag)、非 0 系数大小(Level)、0 系数总和(Totalzeros)和非 0 系数间连续 0 的个数(Runbefore)。

(1)非 0 系数标记：通过非 0 系数个数(TotalCoeff)和正负 1 个数(Trailingones, Trailingones 小于等于 3)可以编码出非 0 系数标记。

(2)正负 1 标记：对正负 1 编码，1 编码为 0，-1 编码为 1。

**基金项目：**华为科技基金资助项目(YJCB2005019BA)
**作者简介：**何 俊(1982－)，男，硕士研究生，主研方向：视频压缩算法及VLSI实现；田应洪，博士研究生；洪志良，教授、博士生导师
**收稿日期：**2007-03-23 **E-mail：**042052044@fudan.edu.cn

(3)非 0 系数大小：对其余非 0 数进行编码。

(4)0 系数总数：对最末一个非 0 数之前出现 0 的个数的编码。

(5)非 0 系数间连续 0 的个数：对 2 个非 0 数之间 0 的个数进行编码。

### 1.3 数据装配的基本原理

图 2 是输出码流的组织形式，序列参数、图像参数和片相互独立地传输，每个片需要携带所属图像的编号，这样虽然增加了编码后数据的冗余度，但是对于通信中的鲁棒性大大增强。在编码时，由于存在一些特殊码流用作特定目的，如果在编码码流中出现相同的码流就会引起解码时的误操作，因此在编码出现特殊码流时，需要在关键码流中插入一个字节以避免解码误操作。

## 2 硬件结构

图 3 方框中是熵编码和封装部分的硬件框架图。经过预测编码和 DCT(Disperse Cosine Transform)量化后，参数和图像数据分别传到指数哥伦布和 CAVLC 进一步进行编码。参数和数据编码后，存储在相应的 RAM 中，向封装部分发送传输请求信号。封装部分由一个状态机控制，如果符合输出状态，则响应信号，允许编码部分传输码流，否则一直等待，直到状态准备好后，才发送响应信号。整个 H.264 系统通过 DMA 总线与外部 SDRAM 交互数据。因为编码前数据输入、编码后数据输出以及重建帧数据的暂存都需要利用 DMA 来交互数据，编码后比特流的数据量最小，所以这里设置比特流输出占用总线的优先级最低。

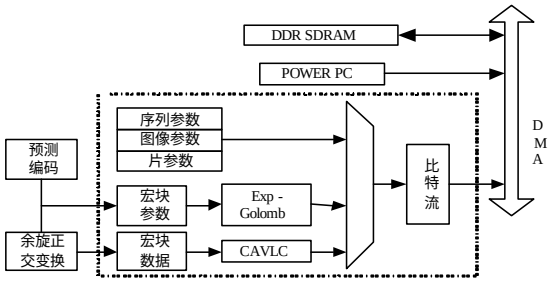


**图 3 熵编码框架**

### 2.1 Exp-Golomb 的硬件结构

图 4 为 Exp-Golomb 的硬件结构图，由于编码的参数需要按照一定顺序编码输出，但是参数到来的先后顺序是不确定的，因此采用状态机控制编码顺序，状态机按照参数排列的顺序进行控制。参数传进来后，判断是否为状态机所指向的参数，如果是则编码，否则存储到 RAM 中，当编码完某个参数后，状态机指向下一个应该编码的参数，当检查到这个参数已经传递进来后，进行编码，否则等待，直到状态机指向的参数输入后，再进行编码。这样通过状态机控制，按照传输顺序依次对参数进行编码。其中 ue, se, me 是针对不同参数的编码方法。

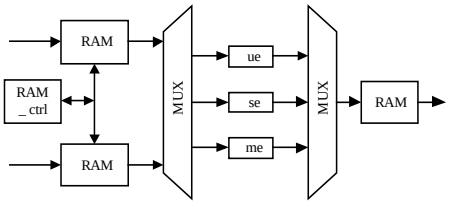


**图 4 Exp-Golomb 框架**

由于前一宏块的参数在没有完全编码传输出去前，下一宏块的参数可能已经传递过来，因此采用前向双 RAM 结构，交替对每一宏块的参数进行存储。前一宏块的参数存储到一块 RAM，下一宏块的参数则会存储到另一块 RAM，而再下一块的宏块参数则又会存储到第一块的 RAM。通过双 RAM 结构，避免后一宏块的数据覆盖前一宏块参数的可能性，同时通过分开存储两块不同宏块的参数，使对每一宏块参数的操作变得更加简单。

### 2.2 CAVLC 的硬件结构

#### 2.2.1 整体结构

图 5 为 CAVLC 的硬件结构图，整个硬件工作流程如下：一个子块输入后，对其进行逆向 Zigzag 扫描，扫描时按照串行方式逐个判断。(1)如果数据是非 0 数，则 TC（TotalCoeff）加 1；(2)如果数据是正负 1，对正负 1 进行编码，存入 TrailingonesFlag 中，同时 T1(Trailingones)加 1；(3)如果数据是其他非 0 数，对其进行 Level 编码；(4)当检测到非 0 数后，TZ(Totalzeros)和 RB(RunBefore)部分开始工作，检测到 0 时，TotalZeros 和 RunBefore 同时加 1，如果检测到非 0 数，则传递 RunBefore 的值，进行编码，同时 RunBefore 值设为 0。当一个子块扫描完成后，就开始对 CoeffToken, TotalZeros 和 RunBefore 的编码。参数的编码采用查找表得到，编码表放在相应的 ROM 中，通过参数值计算得出编码表中的地址值，查找地址值，就可以得到编码值。

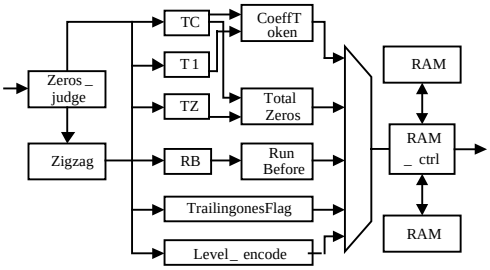


**图 5 CAVLC 硬件结构**

经过统计，如表 1 所示，I 帧中全 0 子块占到整个编码子块的 8.6%到 43.1%，P 帧中占到 46.1%以上。对待全 0 子块，按照通常做法采用反向串行扫描，每一子块都需要 16 个扫描周期后，才能再进行数据编码，浪费大量时间。这里采用快速判断全 0 子块，在 Zigzag 扫描之前增加一级快速判断全 0 子块的结构 Zeros_judge。

**表 1 全 0 子块统计**

| 图像序列(352×288) | 全 0 子块比例 | |
|---|---|---|
| | I 帧/(%) | P 帧/(%) |
| Forman | 20.0 | 88.4 |
| Mobile | 9.3 | 46.1 |
| Mother | 43.1 | 99.0 |
| News | 34.7 | 95.9 |
| salesman | 11.0 | 97.3 |
| tempete | 8.6 | 60.6 |
| bridge | 17.5 | 86.7 |

图 6 是全 0 子块判断部分的硬件结构图，采用 4 个或非门和 1 个与非门来构建，16 位并行输入。如果输出为 1 时，判断为非全 0 子块，继续进行串行扫描；如果输出为 0，则可以判断为全 0 子块，直接跳过反向 Zigzag 扫描部分，进行 Coefftoken 的编码。当输入子块是色度的直流分量时，每个子块只有 4 位数据，这时将其余 12 位输入设置为 0。在通常情况下，子块的非 0 数主要集中在前几位，这时如果全部扫描，则费时较多。将第 1 级的电路或非门的输出引出，当 4 路都为 0 时，即是全 0 子块，则直接跳过扫描过程进行

CoeffToken 编码；当右边 3 个输出都为 1 时，表示非 0 数集中在前 4 个数中，则只用从第 4 位开始向前扫描；相似的，当右边 2 个和 1 个输出为 1 的情况下，则分别从第 8 位和第 12 位开始逆向扫描，当 4 位输出都为 1 时，则从最后 1 位开始扫描。

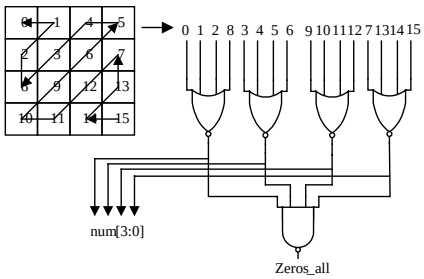


**图 6　全 0 子块探测**

当宏块所划分的 8×8 的块是由 4 个全 0 子块构成时，不需要传递编码参数，通过 cbp(coded_block_pattern)参数的值来标记，如图 7 所示。当 8×8 的块不是全 0 块时，这时设置 cbp 的值为 1，同时传输编码数据。当 8×8 的块是全 0 块时，设置 cbp 值为 0，这时不需要传输编码数据。因此，当 8×8 块为全 0 子块时，需要将已经编码好的码流丢掉，这里采用地址跳变法来实现。编码后的码流先存储到 RAM 中，然后当整个宏块编码完成后再将整个宏块的编码值传输出去。首先记录 8×8 块编码前的 RAM 地址作为基本地址，当编码数据传输进来时，地址根据编码码流进来而跳变，当 8×8 块的编码完全编码完成后，判断 8×8 块是否由 4 个全 0 子块组成，如果都为全 0 子块，则 RAM 的地址跳变回基本地址，即丢掉 4 个子块的数据编码，同时基本地址保持不变。如果不是由 4 个全 0 子块组成，则地址值保持不变，基本地址的值修正为现在的地址值，即下一个 8×8 块编码前的初始值。接着不断重复上面的地址跳变，根据 8×8 的块是否为全 0 块对编码数据进行相应取舍。

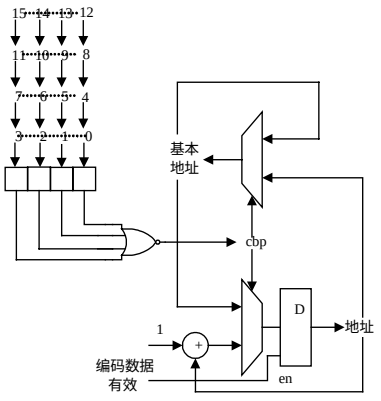


**图 7　地址跳变硬件结构**

在前一个宏块的编码还没有完全传递出去前，下一宏块的数据编码可能已经传递进来，所以这里采用后向双 RAM 结构，对编码后的码流进行存储。

2.2.2　CoeffToken 编码硬件结构

在一个子块的反扫描完成后，可以得到子块的 2 个参数 TotalCoeff 和 Trailingones，然后通过这 2 个参数检索二维查找表，就可以得到 CoeffToken 的编码值。

2.2.3　TrailingonesFlag 编码硬件结构

TrailingonesFlag 是采用 1 个二选一的选择器进行编码，当数据为 1 时，选择器选择编码输出为 0，当数据为-1 时，则选择器编码输出为 1。

2.2.4　Level 编码硬件结构

Level 的编码值可以通过查找表的方法得到，总共有 7 个表，如果采用查找表的方法，则需要占用很多的 ROM，所以这里采用计算的方法代替查找表的方法。

图 8 为 Level 的硬件编码框架图，整个结构采用 3 级流水的硬件：(1)对编码参数进行正数化，得到 LevelCode 的值。(2)通过 suffixlength, LevelCode 2 个参数计算得到前缀 Level_prefix、后缀 Level_suffix 和后缀长度 levelsuffixsize 3 个参数值。(3)通过前缀、后缀和后缀长度 3 个参数计算得到编码值 code 和编码长度 length。

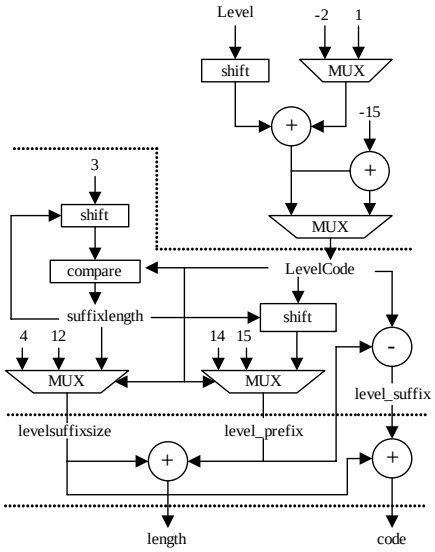


**图 8　Level 硬件结构**

Level 编码是与扫描过程同时进行的，当扫描到需要进行 Level 编码的数据时，直接进行编码。相邻两个数据都需要进行 Level 编码时，由于采用流水线技术，可以直接进行编码，而不需要 RAM 来存储需要编码的数据。

在对一个子块进行熵编码时，并不是每个数据都需要进行 Level 编码，如果 Level 编码从一个子块的开始直到一个子块的结束一直工作，则很多情况下，编码都是在做无用功。这里采用一个 3 位位宽的参数 state 来标记每一步是否执行，每一位对应每一步骤是否执行，1 表示运算，0 表示不运算。当需要编码的数据输入后，将参数 state 左移一位，然后设置 state 第 0 位为 1；当检测到 state 的某一位为 1 时，则相应步骤进行运算，否则不计算。当检测到 state 的 3 位不是全 0 时，表示正在进行 Level 的计算，则每隔一个时钟 state 的值左移一位，否则当 state 全 0 时，表示此时不需要进行 Level 编码，state 保持不变。这样通过 state 控制 Level 编码是否执行，达到减少功耗的目的。

2.2.5　TotalZeros 编码硬件结构

TotalZeros 的编码与 CoeffToken 的编码方法相同，在一子块的扫描完成后，通过 TotalCoeff 和 TotalZeros 2 个参数检索二维查找表得到 TotalZeros 的编码值。

2.2.6　RunBefore 编码硬件结构

RunBefore 的参数编码通过 RunBefore 和 ZerosLeft 2 个参数检索二维查找表得到，由于 RunBefore 的编码值只有 42 种，因此采用直接计算来代替查找表法减少 RAM 资源，如

图 9 所示。

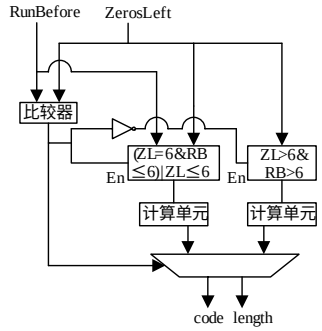


**图 9　RunBefore 硬件结构**

### 2.3　装配部分硬件结构

装配部分采用了一个 88 位的移位寄存器来进行码流的组织，如图 10 所示。采用 32 位的输入位宽，32 位的输出位宽，这样只需通过 4 个并行的检测器来检查是否有与保留序列相匹配的序列，当编码码流中检查到与保留序列相同的时候，则在相应位置插入一个字节 0x03。当检测到装配器中的码流长度少于 48 位时，则允许参数和数据向封装传递数据。当数据大于 48 位时，则向总线提请传输数据，如果总线空闲，则会响应传输请求，这时进行数据传输。

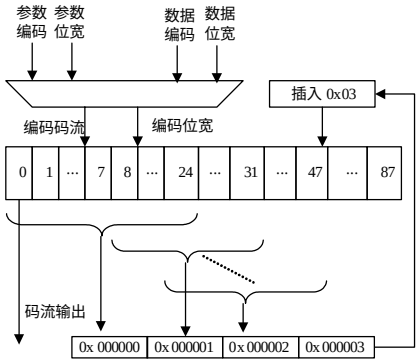


**图 10　装配硬件结构**

## 3　综合结果

本文通过仿真测试验证。首先编写 Verilog HDL 语言，采用 ModelSim 仿真通过后，使用 xilinx 的 ML310FPGA 开发板来进行综合验证。

使用 ISE8.1 对代码进行综合，综合的各部分结果如表 2 所示，硬件逻辑总开销为 3 911 slice，最长路径的时间为 11.449 ns，对应系统工作最高时钟 87.346 MHz。表 2 最后一行是本文所建议的设计结构与文献[4]所用结构的一个资源占用对比，可以看到本文所建议的结构在资源占用大小和关键路径上比文献[4]所采用的结构都有很大的提高。

**表 2　FPGA 占用资源**

| | Number of Slices | Flip Flops | LUTs | BRAMs | Criticalpath /ns | CLK /MHz |
|---|---|---|---|---|---|---|
| Exp-Golomb | 657 | 643 | 1 207 | 3 | 4.940 | 202.429 |
| CAVLC | 2 588 | 1 571 | 4 842 | 4 | 11.449 | 87.346 |
| BGU | 666 | 124 | 1 273 | 0 | 9.892 | 101.092 |
| 本文所提结构 | 3 911 | 2 338 | 7 322 | 7 | 11.449 | 87.346 |
| 文献 4 所提结构 | | 15 622 | 84 902 | | 31.326 | 31.9 |

## 4　结束语

本文针对 H.264 的熵编码结构，提出一种新的结构：采用双 RAM 结构，对中间数据进行存储，既有利于数据的完整性，同时便于针对块的操作；采用全 0 子块的快速判断编码，加快了编码速度；采用计算方法代替查找表法，节省 RAM 资源；采用流水线结构，减少结构的大小，并且节约 RAM 资源；通过状态机控制 Level 工作，达到减少功耗的目的。整个模块综合结果为 7 322 slice，关键路径为 11.449 ns，对应 87.346 MHz 的时钟频率。

## 参考文献

[1] Joch A, Kossentini F, Schwarz H, et al. Performance Comparison of Video Coding Standards Using Lagragian Coder Control[C]//Proc. of ICIP. New York: [s. n.], 2002: 501-504.

[2] ITU-T. ITU-T Recommendation H.264 [S]. 2003.

[3] Richardson I E G. H. 264 and MPEG-4 Video Compression[M]. [S. l.]: John Wiley & Sons Ltd., 2003.

[4] Amer I, Badawy W, Jullien G. Towards Mpeg-4 Part 10 System on Chip: A VLSI Prototype for Context-based Adaptive Variable Length Coding [C]//Proc. of Signal Processing System. Berlin: [s. n.], 2004: 275-279.

(上接第 228 页)

表 1 列出了标准遗传算法和基于信息熵的免疫遗传算法对数据 Data1 进行聚类分析的处理结果，为了比较，每个算法分别运行 6 次，每次运行的初始解(种群)都不相同。标准遗传算法的最大迭代次数设定为 200，基于信息熵的免疫遗传算法的迭代次数设定为 50。

**表 1　Data1 的处理结果($k$=6)**

| 运行次数 | 标准遗传算法 | 基于信息熵的免疫遗传算法 |
|---|---|---|
| 1 | 530.819 4 | 530.010 4 |
| 2 | 531.164 7 | 530.004 7 |
| 3 | 531.563 9 | 530.000 9 |
| 4 | 532.154 0 | 530.003 3 |
| 5 | 531.433 2 | 530.000 0 |
| 6 | 532.009 1 | 530.000 8 |

通过以上结果分析可以看出，对于 Data1，用基于信息熵的免疫遗传算法进行聚类分析能得到相对较好的最优准则函数值(530.0 小数点后保留 1 位)，该算法得到的结果明显优于标准遗传算法得到的结果，即使是最差的结果也优于标准遗传算法最好的结果。

## 参考文献

[1] Han Jiawei, Kamber M. 数据挖掘概念与技术 [M]. 北京: 机械工业出版社, 2001: 168.

[2] Selim S Z, Ismail M A. K-means-type Algorithms: A Generalized Convergence Theorem and Characterization of Local Optimality[J]. IEEE Trans. on Pattern Analysis and Machine Intelligence, 1984, 6(1): 81-87.

[3] 傅景广, 许　刚, 王裕国. 基于遗传算法的聚类分析[J]. 计算机工程, 2004, 30(4): 122-124.

[4] Ansari N, Hou E. 用于最优化的计算智能[M]. 李　军, 边肇祺, 译. 北京: 清华大学出版社, 1999: 238-240.

[5] Ishida Y, Adachi N. Active Noise Control by an Immune Algorithm: Adaptation in Immune System as an Evolution[C]//Proc. of ICEC'96. New Jersy: IEEE Press, 1996: 150-153.

[6] 郑建刚, 王行愚. 基于信息熵的 DNA 免疫遗传算法[J]. 计算机仿真, 2006, 23(6): 164.