一种高效的 CABAC 解码器硬件结构设计

徐美华[1),2)] 吴 明[1)] 周 杰[1)]

[1)](上海大学机电工程与自动化学院，上海 200072) [2)](上海市电站自动化技术重点实验室，上海 200072)

摘 要 相对于其他熵编码而言，基于上下文的自适应二进制算术熵编码(CABAC)具有更大的数据压缩率，但由于其运算复杂，访问存储设备频繁，导致编/解码率较低。针对影响 CABAC 解码速度的“瓶颈”问题，提出了一种高效的 CABAC 解码器硬件结构，包括新的存储访问方式、优化的解码核心单元结构以及子解码器级联的方式。实验结果表明，该硬件结构可显著提高 CABAC 的解码效率。

关键词 H.264/AVC 基于上下文的自适应二进制算术熵编码 概率模型更新 级联方式 自适应

中图法分类号：TN911.72 **文献标识码**：A **文章编号**：1006-8961(2006)11-1588-04

A High-performance Hardware Architecture of CABAC Decoder

XU Mei-hua[1),2)], WU Ming[1)], ZHOU Jie[1)]

[1)](*School of Mechatronical Engineering and Automation, Shanghai University, Shanghai* 200072)

[2)](*Shanghai Key Laboratory of Power Station Automation Technology, Shanghai* 200072)

Abstract Comparing to other entropy coding standard, CABAC(context adaptive binary arithmetic coding) has more significant enhancement in compression. However, its encoding/decoding efficiency is low due to calculation complexity and costs in memory access. To deal with the “bottleneck” problem in decoding process, the article proposes efficient hardware architecture for CABAC decoder, including a new memory access mode, an optimizing decoding core and four concatenating decoding engines. The experimental results show that the decoding efficiency is greatly increased by the hardware architecture.

Keyword H.264/AVC, context adaptive binary arithmetic coding(CABAC), model renewal, concatenate mode, self-adaptive

1 引 言

基于上下文的自适应二进制算术编码(context adaptive binary arithmetic coding, CABAC)是 H.264/AVC 所建议的两种熵编码方案之一，相对于另一种熵编码方案——基于上下文自适应可变长编码(CAVLC)，它具有更高的数据压缩率：在同等编码质量下 CABAC 要比 CAVLC 节约 10% ~15% 的比特率[1]。

CABAC 可以实现很高的数据压缩，但由于不管是二进制解码过程还是上下文模型保持与更新都需要大量的计算及存储器件的访问，因此大大增加了解码所需的时间。如不进行优化，解出一个单独的 bin 就需要花费十几个时钟周期，这对于实时通讯来说是不能接受的。

针对影响 CABAC 解码中的“瓶颈”，即多次访问存储器件影响解码速率，本文提出了新的存储组织方式，并根据 CABAC 句法元素的特性，设计出子 4 个子解码器级联的方式用来进一步提高解码速率。

基金项目：上海市重点学科建设项目(T0103)；上海市高等学校科技基金重点项目(05ZZ03)

收稿日期：2006-06-15；**改回日期**：2006-07-31

第一作者简介：徐美华(1957 ~)，女，副教授。2005 年获上海大学机电工程与自动化学院博士学位。研究方向为图像信号处理及 VLSI 结构设计。E-mail：mhxu@staff.shu.edu.cn

2　影响 CABAC 解码效率的环节

影响解码器解码速度的主要原因是由于在解码过程中需频繁访问存储设备所致,下面将进行详细分析。

(1)在解一个 bin 的计算过程中,需用到 rLPS(低频率符号概率区间)。而为了简化计算,rLPS 已被固化成 4×64 的表格,存于 ROM 中。读取 rLPS 需花费一个时钟周期的时间。

(2)低频率符号(least probable symbol, LPS)与高频率符号(most probable symbol, MPS)的出现概率被分别量化成 64 个整数值。概率之间的状态转移索引号(pStateIdx)构成一个 2×64 的表格[2],存于 ROM 中。查表时会花费一个时钟周期。

(3)句法元素的每个 bin 都对应一个或多个上下文概率模型(context model)。每个模型由 7bits 构成,在 CABAC 中共定义了 459 种概率模型,故在 RAM 中需维护 459×7 的动态表格[3]。每解一个 bin,都需先从 RAM 读出它对应的概率模型值,解完码后,再根据解码结果对模型进行更新,然后写入 RAM 中。这样,解一个 bin 要访问 RAM 两次(读和写),需花费两个时钟周期[4]。

(4)在重新正则化过程(renormalization)中,从输入码流中读取码时也会花费一个时钟周期。

综上所述,解一个单独的 bin 的过程中访问存储设备就会花费 4~5 个时钟周期,从而严重地影响了解码器的效率。

3　CABAC 解码器的优化

图 1 所示为笔者设计的解码器的总体结构,其中输入码流管理模块提供码流给核心解码单元;

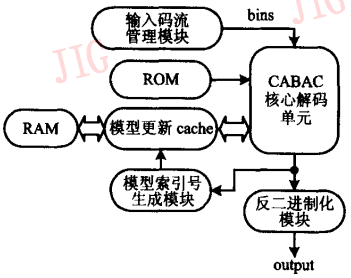

图 1　CABAC 解码器总体结构

Fig. 1　Framework or CABAC decoder

ROM 中存储解码过程中所需的表格信息;解码单元与模型更新 cache(高速缓冲存储器)、RAM 组成读写模型通路;模型索引号生成模块用于为下一个待解的 bin 提供模型索引号 ctxIdx;反二进制模块则用于将解码后的数据流组合成句法元素。

3.1　存储结构的优化

3.1.1　概率模型更新的优化

每解一个 bin 时,概率模型更新都需花掉两个时钟周期。笔者从电脑中 CPU、内存及 cache 三者之间的关系中得到启发:如果在核心解码单元与 RAM 之间插入一个模型更新 cache,并在 cache 中先存入一部分模型,那么解码过程中模型的读取与更新都可在 cache 里完成。到一定时候,再将 cache 里的模型值写入 RAM 中,并从 RAM 中读取另一部分模型。

显然,如将 cache 定义为寄存器组,则核心解码单元访问 cache 并不浪费时钟。所花费时间只是 cache 访问 RAM 的时间。

CABAC 中共有 459 种模型,根据句法元素及它们在码流中出现的时间相关性可将它们分成 25 组:(1)mb_type(I, SI 帧);(2)mb_skip_flag;(3)mb_type(P, SP 帧);(4)sub_mb_type(P 帧);(5)mb_type(B 帧);(6)sub_mb_type(B 帧);(7)abs_mvd;(8)ref_idx;(9)mb_qp_deltal;(10)mb_pred;(11)code_block_pattern;(12)mb_field_decode_flag;(13)transform_size_8×8_flag;以及 12 组宏块类型,即与残差数据相关的 4 个句法元素:①significant_coeff_flag,②last_significant_coeff_flag,③coded_block_flag,④coeff_abs_level_minus1 可按当前的宏块类型(CABAC 的宏块类型共有 6 种)分成 12 组:帧宏块的类型 0~类型 5(6 组);场宏块的类型 0~类型 5(6 组)。

分组后的句法元素中最多含有 44 种概率模型,故可将 cache 设成 44×7bits 的寄存器组[6]。另外若将 RAM 组织成 64×105bits 的结构,则一个时钟周期可对 15 个模型(15×7)进行读取或写入,这样即使含模型最多的分组(含 44 个模型)也可在 3 个周期内完成访问。

3.1.2　查表存储结构的优化

在解码过程中有以下两处需要查表:(1)需查表取出当前 rLPS 的值,rLPS 的值构成一个 4×64 的 2 维表格,表格入口参数为 pStateIdx(0~63)及 rangeIdx(0~3);(2)解码完成后,如解出码为低频率码 valLPS,则 pStateIdx 的更新是没有规律的,需

用查表来完成[5]。

为了提高访问效率,可将两个表格合成一个,采用如表 1 所示的存储结构。

表 1 rLPS 与 pStateIdx 的存储结构

Tab. 1 Storage architecture for rLPS and pStateIdx

null	last_state	last_rLPS	current_rLPS	next_rLPS
(111:104)	(103:96)	(95:64)	(63:32)	(31:0)

其中,last_rLPS, current_rLPS, next_rLPS 均为 32bits 寄存器变量,各包含 4 个 rLPS 值。表 1 的入口参数为 pStateIdx,last_rLPS 代表解出 LPS 码后,pStateIdx 进行更新后所对应的 4 个 rLPS 值;current_rLPS 代表当前 pStateIdx 所对应的 4 个 rLPS 值;next_rLPS 代表解出码为 MPS,对 pStateIdx 进行更新后所对应的 4 个 rLPS 值。last_state 对应的解出码为低频率符号 LPS 对 pStateIdx 进行更新后的值。采用这样的存储结构,既提高了效率,又保证了在 ctxIdx 相等的情况下,并可连续完成连续两个 bin 的解码。

3.2 核心解码部件的优化

核心解码部件包括 2 个常规解码器与 2 个旁路解码器。本文设计中,采用 4 个子解码器级联的方式来最大程度地提高解码效率。

3.2.1 核心解码部件的结构探讨

通过研究 CABAC 句法结构,发现大部分句法元素在码流中出现的频率很低,即在一个宏块中只出现一次或少数几次;而与残差相关的句法元素则出现频率很高,在一个宏块中可出现几十次或数百次。

与残差相关的句法元素共有以下 4 个[5]: significant_coeff_flag, last_significant_coeff_flag, coded_block_flag, coeff_abs_level_minus1,特别是 coeff_abs_level_minus1 的解码占了 CABAC 解码过程的最大的工作量,故本文将主要针对此句法元素进行优化。

coeff_abs_level_minus1 在二进制化时采用了 UEGK0 (unary/k - th order Exp - Golomb)的编码方式,这种编码包括前缀 prefix 及后缀 suffix 两部分。前缀部分采用了截断一元编码(trancated unary),只有当 coeff_abs_level_minus1 的值大于 14 时,才会有后缀部分,后缀采用了 Exp - Golomb (EG)编码。综合考虑 coeff_abs_level_minus1 二制化编码方式,可采用图 2 所示的树型结构来进行解码[6]。

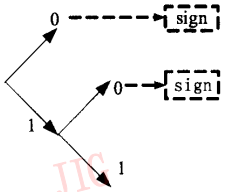

图 2 coeff_abs_level_minus1 解码的树型结构

Fig. 2 Decoding tree for coeff_abs_level_minus1

当 coeff_abs_level_minus1 的值小于 15 时,如解出码为"0"时,则该句法单元解码结束。后接 coeff_sign_flag (1bit,即残差数据的符号位),并将该位送到旁路解码器进行解码。

通常可采取如下方法提高解码效率:常规解码器 1 解出的 bin 为'1'时,ctxIdx 并不变化,级联的常规解码器 2 可以对下一个 bin 解码,此时一个周期可解出 2 个 bin;如解码出的 bin 为'0'时,则级联的旁路解码器立即对下一个 bin(符号位)进行解码,在这种情况下,一个周期可以解出 3 个 bin。

当 coeff_abs_level_minus1 的值大于或等于 15 时,这时就要涉及到后缀的解码。coeff_abs_level_minus1 的后缀部分为 EG 码,而 EG 编码同样是由前缀 prefix 及后缀 suffix 两部分组成。对于 EG 码的前缀部分,同样可以应用图 2 所示的树形结构来加快解码。并通过将两旁路解码器级联来解 EG 码的 prefix 部分,这时一个周期同样可以解出 2 个 bin。

从上面的分析可以看出,采用图 2 所示的树形结构可以大幅提高解码效率。

3.2.2 核心解码部件总体结构

图 3 所示为核心解码部件的整体结构。

核心解码部件用了 4 个子解码器进行解码,其中包括 2 个常规解码器和 2 个旁路解码器。这样的结构主要是针对句法元素 coeff_abs_level_minus1 而设计的。在解 coeff_abs_level_minus1 时,一个周期可解出 1 ~ 3 个 bin,从而显著提高了解码速率,其中大部分的句法元素的解码是在常规解码器 1 中完成的。这时,一个周期可解出 1 个 bin。

为了使这 4 个解码器协调工作,本文设计了下列控制信号:

(1) dec_mode (3bits):dec_mode 用来标明解码器当前的状态。0:正在进行常规解码;1:正在解 EG 码的前缀(prefix)部分;2:正在解 EG 码的后缀(suffix)部分;3:由旁路解码器 1 在解符号位;4:由

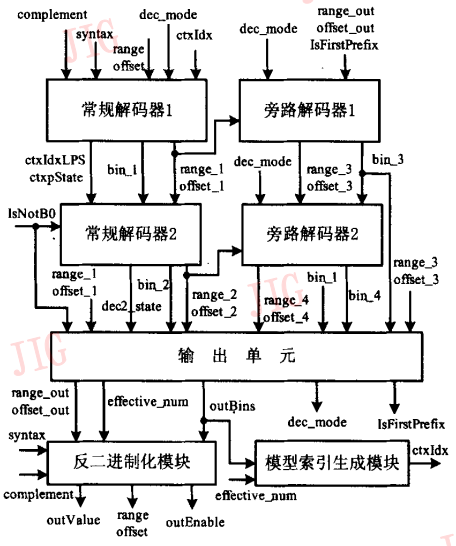

图 3　核心解码部件整体结构

Fig. 3　Framework of core decoding components

旁路解码器 2 在解符号位。

(2) ctxIdxLPS (32bits):用来传递给常规解码器 2 的 4 个低频率码概率区间值。常规解码器 1 从 ROM 中读出数据(见表 1),即完成解码。如解出码为高频率码,就将 32bits 的 next_rangeLPS 传给常规解码器 2;如为低频率码,则将 last_rangeLPS 传给常规解码器 2。

(3) ctxpState (7bits):用于将当前所用的模型值传给常规解码器 2。

(4) IsNotB0 (1bit):用来判定当前所解的是否为句法元素 coeff_abs_level_minus1 的第 0 位(b_0 位)。因为该句法元素 b_0 位所对应的模型编号(ctxIdx)与其他各位是不同的。解 b_0 位后,下一位并不能立即由常规解码器 2 接着解码。

(5) dec2_state (2bits):此信号标明常规解码器 2 的解码状态。dec2_state 的高位用来表示解出码为 LPS 或 MPS,0:LPS;1:MPS。dec2_state 的低位用来表示解码成功与否,0:失败;1:成功。

(6) IsFirstPrefix (1bit):是否开始解 EG 码的前缀(prefix)部分。

(7) effctive_num(2bits):前一个时钟周期解出多少个 bin。

4　结　论

本文提出的高效 CABAC 解码器硬件结构是采用 FPGA stratix(TM) EP1S40F780C5 做系统仿真,其在输入典型的 4M/s 的 NTSC D1 精度码流的情况下,平均每个宏块可在 800 个时钟周期内完成解码。而相同的码流在 100MHz 的时钟频率下,每个宏块在若能 2 000 个时钟周期内解码,即可基本能实现实时通讯。实验结果表明,由于采用了硬件加速,从而显著提高了 CABAC 的解码效率。硬件解码的优化结构不仅能满足实时应用要求,且能实现更高的图像精度和提高视频质量,相应的储存结构重组等硬件加速思想也可应用在 CABAC 编码器结构设计上。这将推动 CABAC 的广泛应用。

参考文献(References)

1　Witten I H, Neal R M, Cleary J G. Arithmetic doding for data compression[J]. Communications of the ACM, 1987, **30**(6): 520 ~ 540.

2　Nunez L, Chouliaras V A. High_performance arithmetic coding VLSI macro[J]. IEEE Transactions on Consumer Electronics, 2005, **51**(8): 112 ~ 118.

3　Detlev Marpe, Heiko Schwarz, Thomas Wiegand. Context-based adaptive binary arithmetic coding in the H. 264/AVC video compression standard[J]. IEEE Transactions on Circuits and Systems for Video Technology, 2003, **13**(7): 620 ~ 636.

4　Osorio R R, Bruguera J D. Arithmetic coding architecture for H.264/AVC CABAC compression system [A]. In: Euromicro Symposium on Digital System Design[DB/OL], http://ieeexplore ieee. org, 2004: 62 ~ 69.

5　Wiegand T, Sullivan G J, Luthra A. Overview of the H. 264/AVC video coding standard [J]. IEEE Transactions on Circuits and Systems for Video Technology, 2003, **13**(7): 560 ~ 576.

6　Yu Wei, He Yun. A High performance CABAC decoding architecture [J]. IEEE Transactions on Consumer Electronics, 2005, **51**(4): 1352 ~ 1359.