(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 102325253 A
(43)申请公布日 2012.01.18

(21)申请号 201110232250.4

(22)申请日 2011.08.15

(71)申请人 复旦大学
地址 200433 上海市杨浦区邯郸路 220 号

(72)发明人 范益波 钟慧波 沈沙 任怀鲁
姜英 曾晓洋

(74)专利代理机构 上海正旦专利代理有限公司 31200
代理人 陆飞 盛志范

(51)Int.Cl.
H04N 7/26(2006.01)

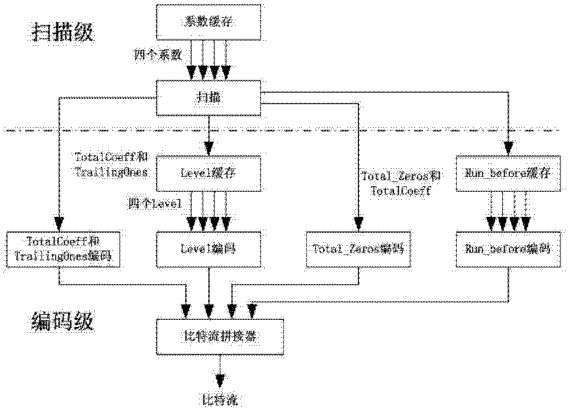
权利要求书 4 页 说明书 13 页 附图 5 页

(54)发明名称

一种四路并行编码的 CAVLC 编码器

(57)摘要

本发明属于视频编码技术领域，具体为一种四路并行编码的 CAVLC 编码器。本发明采用扫描级和编码级并行处理的二级流水线结构，扫描级一次可以扫描四个系数的，大大的减少扫描一个 4x4 块所需要的时间。同样，通过四个 level 和 Run_before 并行编码的方式来缩短编码级所需要的时间。编码级的所需要的时间通过细致的设计，使得其所消耗的时间和扫描级所用的时间相同。这样，整个 CAVLC 的两级流水线可以得到最大的吞吐率，极大地减小完成一个宏块的编码所需要的时钟数。



CN 102325253 A

1. 一种四路并行编码的 CAVLC 编码器， 其特征在于采用扫描级和编码级组成的二级流水结的架构，分为扫描级、编码级和 CAVLC 控制器三部分；其中，扫描级由扫描级控制器，TrailingOnes 计数器、TotalCoeff 计数器、Total_Zeros 计数器、Run_before 计数器和 Level 检测器组成，扫描级用于统计 CAVLC 元素值；编码级由编码级控制器、TotalCoeff 编码器、Total_Zeros 编码器、Level 编码器、Run_before 编码器和比特流拼接器组成，编码级用于编码扫描级统计得到的 CAVLC 元素值；CAVLC 控制器通过控制扫描级控制器和编码级控制器来控制整个 CAVLC 编码器的时序和功能；并且，扫描级与编码级两级中采用一块寄存器来存储扫描级统计得到的结果，使得扫描级和编码级能够流水线并行处理；扫描级完成一个 4x4 块的扫描后，将得到的统计信息直接存储在寄存器中，然后编码级开始编码，而扫描级则继续扫描下一个 4x4 块。

2. 根据权利要求 1 所述的四路并行编码的 CAVLC 编码器， 其特征在于所述的扫描级控制器通过一个状态机来控制扫描级在每个时钟从残差系数缓存中取 4 个残差系数来统计 CAVLC 编码的一些元素：TotalCoeff，Total_Zeros，TrailingOnes，Level，Run_before，具体为：

1) 从系数缓存中取的四个残差系数为 coeff0~coeff3，根据下面代码 (1) 式和 (2) 式所示得到 8 个状态指示位：s0~s3 和 c0~c3，其中 s0~s3 代表各残差系数是否等于 0，c0~c3 代表各残差系数是否等于 ±1：

$$\begin{cases} if\ coeff0=0, & s0=0 \\ else & s0=1 \\ if\ coeff1=0, & s1=0 \\ else & s1=1 \\ if\ coeff2=0, & s2=0 \\ else & s2=1 \\ if\ coeff3=0, & s3=0 \\ else & s3=1 \end{cases} \tag{1}$$

$$\begin{cases} if\ coeff0=\pm1, & c0=1 \\ else & c0=0 \\ if\ coeff1=\pm1, & c1=1 \\ else & c1=0 \\ if\ coeff2=\pm1, & c2=1 \\ else & c2=0 \\ if\ coeff3=\pm1, & c3=1 \\ else & c3=0 \end{cases} \tag{2}$$

2) 根据上面的 8 个状态指示位：s0~s3 和 c0~c3，得到 CAVLC 需要编码的元素值；其中根据 s0~s3，得到 TotalCoeff，Total_Zeros 和 Run_before；并把不为零的系数存储到 Level 缓存中，根据 s0~s3 和 c0~c3，确定 TrailingOnes 的值，具体过程如下：

1> TotalCoeff 的确定：将 s0~s3 中 1 的个数累加起来即为 TotalCoeff 的值；

2> Total_Zeros 的确定：在每个 4x4 块的扫描开始时，将一个内部变量 TZ_EN 设置为

0，TZ_EN 的表示当前 4x4 的 Total_Zeros 计算的使能位，TZ_EN 在 s0~s3 中有某个状态位值为 1 时设为 1；在每个 4x4 块的扫描开始，在 TZ_EN 第一次设置为 1 时，开始计算 s0~s3 中第一个 1 以后的 0 个数，然后对当前 4x4 块后面的扫描时的 s0~s3 状态位的 0 的个数累加；

3> Run_before 确定：在 s0~s3 中，计算每个 1 前面有多少个 0，即为对应的非零残差系数的 Run_before 值；

4> Level 的确定：在 s0~s3 中，相应的等于 1 的数存储在 Level 寄存器中，编码时根据 TrailingOnes 的大小，将前面 TrailingOnes 个从 Level 寄存器中读出的值不作为 Level 编码；

5> TrailingOnes 确定：把 coeff0~coeff3 分成两组，其中 coeff0，coeff1 为低位组，coeff2 和 coeff3 为高位组；对于这两组残差系数，分别有一个对应的有效标志位 Valid_L/Valid_H 表示当前数残差系数组中是否有大于 1 或小于负 1 的情况出现，即代表当前组的计算的 TrailingOnes 是否有效；用 TrailingOnes_L 和 TrailingOnes_H 来分别代表两组残差系数的 TrailingOnes 的计算结果，用 Valid_A 表示当前 4x4 块中是否有残差系数大于 1 或小于负 1 的情况，若 Valid_A 为 1，则表示之前的残差系数中有大于 1 或小于负 1 的情况，于是有下面的 TrailingOnes 的计算方法：

如果 Valid_A=1，那么：TrailingOnes = TrailingOnes_tmp；

如果 Valid_A=0，那么 TrailingOnes 的计算方法如下：

a) 如果 Valid_L = 0，Valid_H= 0，则 TrailingOnes = TrailingOnes_L+TrailingOnes_H+ TrailingOnes_tmp；

b) 如果 Valid_L = 0，Valid_H = 1，则 TrailingOnes = TrailingOnes_L+TrailingOnes_H +TrailingOnes_tmp；

c) 如果 Valid_L=1，Valid_H=0，则 TrailingOnes=TrailingOnes_tmp+TrailingOnes_L；

d) 如果 Valid_L=1，Valid_H=1，则 TrailingOnes=TrailingOnes_tmp+TrailingOnes_L；

其中 TrailingOnes_tmp 表示为暂时存储在内部的 TrailingOnes 计算结果，在每个 4x4 块的扫描开始时 TrailingOnes_tmp 设置为 0，在每一组的 4 个系数统计过程完成后 TrailingOnes_tmp 的值设为 TrailingOnes；最后得到的 TrailingOnes 限定在小于等于 3 的范围。

3. 根据权利要求 2 所述的四路并行编码的 CAVLC 编码器，其特征在于所述的编码级控制器控制 TotalCoeff 编码器和 Total_Zeros 编码器在编码级的第一个时钟时，根据扫描级的 TotalCoeff，Total_Zeros 和 TrailingOnes 的值，查询相应的表格得到码字，编码级控制器通过 TotalCoeff 的值来控制 Level 编码器中在每个时钟编码四个 Level 值，编码级控制器通过 Total_Zeros 的值来控制 Run_before 编码器中在每个时钟编码四个 Run_before 值，编码级控制器控制比特流拼接器在每个时钟把 TotalCoeff 编码器，Total_Zeros 编码器，Level 编码器和 Run_before 编码器产生的码字拼接成为比特流。

4. 根据权利要求 3 所述的四路并行编码的 CAVLC 编码器，其特征在于所述的 TotalCoeff 编码器，Total_Zeros 编码器通过查表即得到相应的码字，Level 编码器和 Run_before 编码器的设计如下：

Level 编码器采用 H.264/AVC 中一个 VLCN 提前计算的技术，同时决定四个 Level 幅值的编码对应的表格序号，实现同时对四个 Level 进行编码，然后把四个 Level 编码得到的码字和对应的长度拼接起来，VLCN 提前计算的代码为：

```
incVLC[7] = { 0, 3, 6, 12, 24, 48, 0xffff };
if( first level  && TotalCoeff>10 &&TrailingOnes<3)
vlc0 = 1;
       else if(first level)
              vlc0 = 0;
else if( abs(reg_level3) > incVLC[reg_vlc3] )
vlc0 = reg_vlc3+1;
       else
vlc0 = reg_vlc3;

if( vlc0 =0 )
vlc1 = vlc0+1;
else if( abs(level0) > incVLC[vlc0] )
vlc1 = vlc0+1;
       else
              vlc1=vlc0;
if( abs(level1) > incVLC[vlc1] )
vlc2 = vlc1+1;
       else
              vlc2 = vlc1;

if( abs(level2) > incVLC[vlc2] )
vlc3 = vlc2+1;
       else
              vlc3 = vlc2;
```

上述代码中 abs 代表的的是取绝对值的操作，incVLC[7] = { 0, 3, 6, 12, 24, 48, 0xffff } 表示的是表格选择的条件；vlc0~vlc3 分别表示编码当前四个 Level 值分别对应的表格序号，level0~ level3 表示当前编码的幅值数，reg_level3 表示前一组四个 Level 值的中的 level3，reg_vlc3 表示编码前一组四个 Level 值时编码 level3 时用的表格序号 vlc3；

然后，根据每个 level 和其相应的表格序号来编码每个 level 值，每个 Level 的码字结构包含前缀，后缀和它们中间的 1；前缀由 0 组成，后缀的长度由 levelSuffixsSize 来表示；

根据四个 Level 值的并行编码出来的结果，按照码字的长度，通过移位以后进行或操作来拼接成 FourLevelCode，即四个 Level 的编码码字；

Run_before 编码器，每个时钟也是编四个 Run_before，编码 Run_before 主要的工作是

查找表 1 ：

表 1

Run_before	ZerosLeft						
	1	2	3	4	5	6	>6
0	1	1	11	11	11	11	111
1	0	01	10	10	10	000	110
2	-	00	01	01	011	101	101
3	-	-	00	001	010	011	100
4	-	-	-	000	001	010	011
5	-	-	-	-	000	101	010
6	-	-	-	-	-	100	001
7	-	-	-	-	-	-	0001
8		-	-	-	-	-	00001
9	-	-	-	-	-	-	000001
10	-	-	-	-	-	-	0000001
11	-	-	-	-	-	-	00000001
12	-	-	-	-	-	-	000000001
13	-	-	-	-	-	-	0000000001
14	-	-	-	-	-	-	00000000001

表 1 中 zerosLeft 表示编码当前 Run_before 时剩余 0 的个数 ;Run_before 从 Run_before 缓存中读取得到 ;每个时钟有四个 Run_before 值从 Run_before 缓存中读取出来，用 Run_before0~Run_before3 来代表这四个 Run_before 值 ;为了查找上述表格，还需要确定这个四 Run_before 值相应的剩余 0 的个数 zerosLeft0 ~ zerosLeft3，其计算过程如下 ：

zerosLeft0 = Reg_zerosLeft ;

zerosLeft1 = zerosLeft0 - Run_before0 ;

zerosLeft2 = zerosLeft1 - Run_before1 ;

zerosLeft3 = zerosLeft2 - Run_before2 ;

其中 Reg_zerosLeft 表示为当前 4x4 块中剩余 0 的个数，其取值为 :如果当前 Run_before 为当前 4x4 块的第一个 Run_before 值，则 Reg_zerosLeft = Total_Zeros ;否则，Reg_zerosLeft = Reg_zerosLeft3- Reg_Run_before3 ;其中 Reg_Run_before3 和 Reg_zerosLeft3 分别表示为上一组四个 Run_before 和其对应的四个 zerosLeft 中的 Run_before3 和 zerosLeft3 ;

根据四个 Run_before 并行编码出来的结果和码字的长度，通过移位以后进行或操作来拼接成 FourRunCode，即四个 Run_before 的编码码字。

5. 根据权利要求 4 所述的四路并行编码的 CAVLC 编码器 , 其特征在于所述的比特流拼接器把 TotalCoeff 编码器，Total_Zeros 编码器，Level 编码器和 Run_before 编码器产生的码字拼接成为比特流 ;编码级在四个时钟内完成，比特流拼接器的时序设计为 :第一个时钟周期，比特流拼接器把 TotalCoeff 码字和第一组 FourLevelCode 打包在一起 ;接下来两个周期，比特流拼接器都只需要打包 Level 编码器输出的 FourLevelCode ;在最后一个时钟周期，比特流拼接器将最后的 FourLevelCode 和 Total_Zeros/Run_beforeCode 打包在一起。

一种四路并行编码的 CAVLC 编码器

技术领域

[0001] 本发明属于视频编码技术领域，具体涉及一种 H.264/AVC 硬件编码器。

背景技术

[0002] H.264/AVC 是 JVT 组织最新提出的一个视频编码标准。因为 H.264/AVC 能够比之前一些编码标准得到更高的压缩效率和图像质量，所以它的应用越来越广泛。在 H.264/AVC 中，规定了两种编码格式：上下文自适应可变长度编码（CAVLC）和上下文自适应算术编码（CABAC）。在 CAVLC 编码方式中，只对量化后的残差系数进行编码，其它信息，比如说宏块头信息等，都以指数哥伦布码的形式来进行编码（Exp-Golomb ）。CAVLC 编码的主要元素有：

1. Coeff_token(TotalCoeff)：这是 CAVLC 中的第一个 VLC 元素。TotalCoeff 表示的是在一个 4x4 中所有非零系数的个数，这个元素编码在一个 4x4 块中所有非零系数的个数和拖尾 1 的个数（TrailingOnes）。TotalCoeff 的取值范围为 0~16。

[0003] 2. TrailingOnes：这个元素表示拖尾 1 的个数。它代表的是在反 Zig-Zag 扫描时，除去 0 系数后起始连续 ±1 的个数，其值的取值范围为 0~3。即如果连续 ±1 的个数超过 3，TrailingOnes 值为 3，其余的 ±1 不作为 TrailingOnes。

[0004] 3. Sign_trail：这个元素表示的是拖尾 1 的符号。每一个符号编码为一比特，0 表示正 1，1 表示负 1。

[0005] 4. Level：这个元素表示在 4x4 块中除去 TrailingOnes 外的其它的非零系数。其编码顺序为反 zig-zag 顺序，Level 的个数取值范围为 0~16。

[0006] 5. Total_Zeros：这个元素表示在以反 Zig-Zag 扫描顺序时第一个非零系数后的总的零系数的个数。

[0007] 6. Run_before：这个元素表示在每一个非零系数前的零的个数。其获取和编码顺序均为反 Zig-Zag 顺序。

[0008] 相比传统的变长编码，CAVLC 可以有更高的编码效率，这是因为 CAVLC 中引入了这个上下文自适应这个特点。另一方面来说，正是因为这个上下文自适应的特点，它导致了很大的数据相关性，这些相关性主要包括：

1. 上述所描述的一些编码元素，只有在对每个 4x4 块的扫描完成后才能完全得到。

[0009] 2. 在编码 Coeff_token 时需要先决定一个变量 nC 的值，而 nC 的值由当前 4x4 块左边和上边已编码 4x4 块中的非零体系数的个数计算得到。

[0010] 3. 编码 Levels 的时候有 7 张表格，而表格的选择跟编码当前 4x4 块的前一个 Level 的绝对值大小和使用的表格编号有关。

[0011] 4. CAVLC 编码时是一个顺序的过程，其过程为 Coeff_token → Sign_trail → Level → Total_Zeros → Run_before。这个顺序的编码过程很难直接用硬件实现并行完成，使得 CAVLC 编码的时间是一个不固定的时间。

[0012] 由于上述 1~4 点所说的数据相关性，使得设计高吞吐率比如说 4Kx2K（4096x2160）或更高分辨率的 CAVLC 硬件编码器变得非常困难。

发明内容

[0013] 本发明的目的在于提供一种高吞吐率的 CAVLC 编码器。

[0014] 要设计高吞吐率的 CAVLC 编码器，需要在电路中解决前面的相关性问题。在一个宏块中，含有亮度(Y)，色度(Cr 和 Cb)三种分量。在图像格式 YUV = 4:2:0 的情况下，一个宏块中有 384 个量化后的残差系数。所以扫描级有可能会成为编码器的瓶颈。另外，由于量化后的残差系数的可能性很多，导致前面提到的一些 CAVLC 元素的结果变化较大。比如说 Level 的个数最多有 16 个，这会导致在编码时的时钟数变化很大，进而限制编码器的吞吐率。本发明主要通过增加硬件并行性来解决限制 CAVLC 编码器吞吐率的一些问题。

[0015] 本发明设计的高吞吐率的 CAVLC 编码器，采用扫描级和编码级组成的二级流水结的架构。本发明的 CAVLC 编码器的顶层架构如图 1 所示。整个 CAVLC 编码器可以分为扫描级，编码级和 CAVLC 控制器三部分。扫描级由扫描级控制器，TrailingOnes 计数器，TotalCoeff 计数器，Total_Zeros 计数器，Run_before 计数器和 Level 检测器组成，扫描级的功能是统计 CAVLC 元素值。编码级由编码级控制器，TotalCoeff 编码器，Total_Zeros 编码器，Level 编码器，Run_before 编码器和比特流拼接器组成，编码级的功能是编码扫描级统计得到的 CAVLC 元素值。CAVLC 控制器通过控制扫描级控制器和编码级控制器来控制整个 CAVLC 编码器的时序和功能。扫描级与编码级两级中采用了一块寄存器来存储扫描级统计得到的结果，这样，就可以使得扫描级和编码级能够流水线并行处理。扫描级完成一个 4x4 块的扫描后，将得到的统计信息直接存储在寄存器中，然后编码级开始编码，而扫描级则继续扫描下一个 4x4 块。通过这样的结构，大大提高了整个编码器的吞吐率。

[0016] 本发明的 CAVLC 编码器具体实现如下：

1. 本发明的扫描级包含有扫描级控制器，TrailingOnes 计数器，TotalCoeff 计数器，Total_Zeros 计数器，Run_before 计数器和 Level 检测器组成。扫描级控制器通过一个状态机来控制扫描级在每个时钟从残差系数缓存中取 4 个残差系数来统计 CAVLC 编码的一些元素，如 TotalCoeff，Total_Zeros，TrailingOnes，Level，Run_before 等。其具体步骤如下所述：

1) 从系数缓存中取的四个残差系数为 coeff0~coeff3。然后根据下面代码 (1) 和 (2) 所示得到 8 个状态指示位：s0~s3 和 c0~c3。其中 s0~s3 代表各残差系数是否等于 0，c0~c3 代表各残差系数是否等于 ±1：

$$\begin{cases} if\ coeff0=0, & s0=0 \\ else & s0=1 \\ if\ coeff1=0, & s1=0 \\ else & s1=1 \\ if\ coeff2=0, & s2=0 \\ else & s2=1 \\ if\ coeff3=0, & s3=0 \\ else & s3=1 \end{cases} \quad (1)$$

$$\begin{cases} if\ coeff0=\pm1, & c0=1 \\ else & c0=0 \\ if\ coeff1=\pm1, & c1=1 \\ else & c1=0 \\ if\ coeff2=\pm1, & c2=1 \\ else & c2=0 \\ if\ coeff3=\pm1, & c3=1 \\ else & c3=0 \end{cases} \quad (2)$$

2）现在我们可以根据上面的 8 个状态指示位 :s0~s3 和 c0~c3，来得到 CAVLC 需要编码的元素值。其中根据 s0~s3，可以得到 TotalCoeff，Total_Zeros 和 Run_before。并把不为零的系数存储到 Level 缓存中。而根据 s0~s3 和 c0~c3 我们就可以确定 TrailingOnes 的值。这些元素值的具体确定过程如下所示 :

1> TotalCoeff 的确定方法 :将 s0~s3 中 1 的个数累加起来即为 TotalCoeff 的值。

[0017] 2> Total_Zeros 的确定方法 :在每个 4x4 块的扫描开始时，将一个内部变量 TZ_EN 设置为 0，TZ_EN 的表示当前 4x4 的 Total_Zeros 计算的使能位，TZ_EN 在 s0~s3 中有某个状态位值为 1 时设为 1。在每个 4x4 块的扫描开始，在 TZ_EN 第一次设置为 1 时，我们开始计算 s0~s3 中第一个 1 以后的 0 个数，然后对当前 4x4 块后面的扫描时的 s0~s3 状态位的 0 的个数累加。

[0018] 3> Run_before 确定方法 :对于每个非零残差系数，都有对应有一个 Run_before 值。在 s0~s3 中，我们计算每个 1 前面有多少个 0，即为对应的非零残差系数的 Run_before 值。

[0019] 4> Level 的确定 :在 s0~s3 中，相应的等于 1 的数存储在 Level 寄存器中。这样可能会出现把拖尾 1 存在 Level 中的情况。编码时需要根据 TrailingOnes 的大小，将前面 TrailingOnes 个从 Level 寄存器中读出的值不作为 Level 编码。

[0020] 5> TrailingOnes 确定方法 :TrailingOnes 的确定需要同时考虑 coeff0~coeff3 是否为 0 和是否等于 ±1。为了确定 TrailingOnes，我们把 coeff0~coeff3 分成两组，其中 coeff0，coeff1 为低位组，coeff2 和 coeff3 为高位组。对于这两组残差系数，分别有一个对应的有效标志位 Valid_L（低位组有效位）/Valid_H（高位组有效位）表示当前数残差系数组中是否有大于 1 或小于负 1 的情况出现，即代表当前组的计算的 TrailingOnes 是否有效。 我们用 TrailingOnes_L（低位组 TrailingOnes 计算结果）和 TrailingOnes_H（高位组 TrailingOnes 计算结果）来分别代表两组残差系数的 TrailingOnes 的计算结果。我们用 Valid_A 表示当前 4x4 块中是否有残差系数大于 1 或小于负 1 的情况，若 Valid_A 为 1，则表示之前的残差系数中有大于 1 或小于负 1 的情况。于是有下面的 TrailingOnes 的计算方法 :

如果 Valid_A=1，那么 :TrailingOnes = TrailingOnes_tmp。

[0021] 如果 Valid_A=0，那么 TrailingOnes 的计算方法如下所描述 :

a）如 果 Valid_L = 0，Valid_H= 0， 则 TrailingOnes = TrailingOnes_L+TrailingOnes_H+ TrailingOnes_tmp。

[0022] b）如 果 Valid_L = 0，Valid_H = 1， 则 TrailingOnes = TrailingOnes_

L+TrailingOnes_H +TrailingOnes_tmp。

[0023] c) 如 果 Valid_L=1, Valid_H=0, 则 TrailingOnes=TrailingOnes_tmp+TrailingOnes_L。

[0024] d) 如 果 Valid_L=1, Valid_H=1, 则 TrailingOnes=TrailingOnes_tmp+TrailingOnes_L。

[0025] 其中 TrailingOnes_tmp 表示为暂时存储在内部的 TrailingOnes 计算结果,在每个 4x4 块的扫描开始时 TrailingOnes_tmp 设置为 0,在每一组的 4 个系数统计过程完成后 TrailingOnes_tmp 的值设为 TrailingOnes。 最后得到的 TrailingOnes 需要限定在小于等于 3 的范围。

[0026] 2. 扫描级扫描得到的统计结果,如 :TotalCoeff, Total_Zeros, TrailingOnes, Level, Run_before 等,需要存储在寄存器中以供后面的编码级来使用。Level 和 Run_before 寄存器采用的是一个寄存器阵列的结构,每次可以向 Level 和 Run_before 寄存器中写 0~4 个数,其个数的控制是根据 s0~s3 来实现的,具体写的个数为(s0+s1+s2+s3)。Level 和 Run_before 寄存器在读的阶段每次读 4 个值。即在每个时钟,编码级会从 Level 和 Run_before 寄存器中每次读取 4 个 Level 和 Run_before。

[0027] 3. 编码级主要由编码级控制器,TotalCoeff 编码器,Total_Zeros 编码器,Level 编码器,Run_before 编码器和比特流拼接器组成。编码级控制器控制 TotalCoeff 编码器和 Total_Zeros 编码器在编码级的第一个时钟时,根据扫描级的 TotalCoeff,Total_Zeros 和 TrailingOnes 的值,并查询相应的表格得到码字。编码级控制器通过 TotalCoeff 的值来控制 Level 编码器中在每个时钟编码四个 Level 值。编码级控制器通过 Total_Zeros 的值来控制 Run_before 编码器中在每个时钟编码四个 Run_before 值。编码级控制器控制比特流拼接器在每个时钟把 TotalCoeff 编码器, Total_Zeros 编码器, Level 编码器和 Run_before 编码器产生的码字拼接成为比特流。TotalCoeff 编码器,Total_Zeros 编码器均是通过查表即可以得到相应的码字。而 Level 编码器和 Run_before 编码器均需要特别的设计,来满足四个 Level 并行编码和四个 Run_before 并行编码。其具体设计过程如下所示 :

3.1 在对 Level 编码时 H.264/AVC 中有 7 张表格来编码 Level,本发明采用了一个 VLCN(编码 Level 的 7 张表格的表格序号)提前计算的技术,同时决定四个 Level 幅值的编码对应的表格序号。其具体实现如图 3 所示。这样就可以同时对四个 Level 进行编码。然后把四个 Level 编码得到的码字和对应的长度拼接起来。VLCN 提前计算的过程具体如下面的代码所示 :

```
incVLC[7] = { 0, 3, 6, 12, 24, 48, 0xffff };
if( first level  && TotalCoeff>10 &&TrailingOnes<3)
vlc0 = 1;
        else if(first level)
                vlc0 = 0;
else if( abs(reg_level3) > incVLC[reg_vlc3] )
vlc0 = reg_vlc3+1;
        else
vlc0 = reg_vlc3;
```

```
if( vlc0 =0 )
vlc1 = vlc0+1;
else if( abs(level0) > incVLC[vlc0] )
vlc1 = vlc0+1;
       else
              vlc1=vlc0;
if( abs(level1) > incVLC[vlc1] )
vlc2 = vlc1+1;
       else
              vlc2 = vlc1;
if( abs(level2) > incVLC[vlc2] )
vlc3 = vlc2+1;
       else
              vlc3 = vlc2;
```

上述代码中 abs 代表的的是取绝对值的操作，incVLC[7] = { 0, 3, 6, 12, 24, 48, 0xffff } 表示的是表格选择的条件。vlc0~vlc3 分别表示编码当前四个 Level 值分别对应的表格序号(即 H.264/AVC 中的 suffixlength)。level0~ level3 表示当前编码的幅值数，reg_level3 表示前一组四个 Level 值的中的 level3。reg_vlc3 表示编码前一组四个 Level 值时编码 level3 时用的表格序号 vlc3。上述代码用硬件实现，可以在极短的时间之内得到 vlc0~vlc3，这样可以做到四个 Level 的并行编码。

[0028] 在得到了四个 Level 值时编码的表格序号 vlc0~vlc3。我们需要根据每个 level 和其相应的表格序号来编码每个 level 值。每个 Level 的码字结构包含前缀(level_prefix)，后缀(level_prefix)和它们中间的 1 组成。其码字结构如图 7 所示。前缀由 0 组成。其中后缀的长度由 levelSuffixsSize 来表示。

[0029] 单个 Level 值的编码具体实现框如图 8 所示。它可以分为两部分，一部分为规则码编码器，另一部分为逃逸码(escape)编码器。逃逸码(escape)编码器占了单个 Level 值的编码器的大部分。本发明中充分利用了硬件实现的并行性，同时判断多个值的大小关系来实现速度上的提升。单个 Level 值的编码器的输出结果为前缀(level_prefix)，后缀(level_prefix)和后缀的长度(levelSuffixsSize)。

[0030] 四个 Level 值的并行编码出来的结果可以根据码字的长度，通过移位以后进行或操作来拼接成 FourLevelCode，即四个 Level 的编码码字，即如图 9 所示。

[0031] 3.2 对于 Run_before 的编码时，每个时钟也是编四个 Run_before。编码 Run_before 主要的工作是查找表 1：

表 1

Run_before	ZerosLeft						
	1	2	3	4	5	6	>6
0	1	1	11	11	11	11	111
1	0	01	10	10	10	000	110
2	-	00	01	01	011	101	101
3	-	-	00	001	010	011	100
4	-	-	-	000	001	010	011
5	-	-	-	-	000	101	010
6	-	-	-	-	-	100	001
7	-	-	-	-	-	-	0001
8		-	-	-	-	-	00001
9	-	-	-	-	-	-	000001
10	-	-	-	-	-	-	0000001
11	-	-	-	-	-	-	00000001
12	-	-	-	-	-	-	000000001
13	-	-	-	-	-	-	0000000001
14	-	-	-	-	-	-	00000000001

表 1 中 zerosLeft 表示编码当前 Run_before 时剩余 0 的个数。Run_before 可以从 Run_before 缓存中读取得到。在本发明中，每个时钟有四个 Run_before 值从 Run_before 缓存中读取出来，用 Run_before0~Run_before3 来代表这四个 Run_before 值。为了查找上述表格，我们还需要确定这个四 Run_before 值相应的剩余 0 的个数 zerosLeft0 ~ zerosLeft3。其计算过程如下所示：

zerosLeft0 = Reg_zerosLeft ;

zerosLeft1 = zerosLeft0 - Run_before0 ;

zerosLeft2 = zerosLeft1 - Run_before1 ;

zerosLeft3 = zerosLeft2 - Run_before2 ;

其中 Reg_zerosLeft 表示为当前 4x4 块中剩余 0 的个数，其取值为：如果当前 Run_before 为当前 4x4 块的第一个 Run_before 值，则 Reg_zerosLeft = Total_Zeros。否则，Reg_zerosLeft = Reg_zerosLeft3- Reg_Run_before3。其中 Reg_Run_before3 和 Reg_zerosLeft3 分别表示为上一组四个 Run_before 和其对应的四个 zerosLeft 中的 Run_before3 和 zerosLeft3。

[0032] 在编码中有两种情况 Run_before 值是不需要编码的，一种是 zerosLeft 为 0 的情况，另一种是当前 Run_before 为当前 4x4 块中的最后一个 Run_before 值。

[0033] 四个 Run_before 并行编码出来的结果可以根据码字的长度，通过移位以后进行或操作来拼接成 FourRunCode，即四个 Run_before 的编码码字。本发明 FourRunCode 和 Total_Zeros 码字组合在一起，存在一个内部寄存器中，在编码级的最后一个时钟送给比特流拼接器。内部存储的码字名字称为 Total_Zeros/Run_beforeCode。Run_before 编码器的结构如图 10 所示。

[0034] 4. 比特流拼接器的作用是把 TotalCoeff 编码器，Total_Zeros 编码器，Level 编码器和 Run_before 编码器产生的码字拼接成为比特流。编码级在四个时钟内完成，本发明

的比特流拼接器的时序设计为：第一个时钟周期，比特流拼接器需要把 TotalCoeff 码字和第一组 FourLevelCode 打包在一起；接下来两个周期，比特流拼接器都只需要打包 Level 编码器输出的 FourLevelCode。在最后一个时钟周期，比特流拼接器将最后的 FourLevelCode 和 Total_Zeros/Run_beforeCode 打包在一起。比特流拼接器的时序如图 11 所示。

[0035] 本发明的有益效果：

通过前面的一些并行技术，本发明可以将一个宏块的残差系数编码最多时间所需要的时钟数缩小为 108 个(未含 latency 时钟数)，而传统的 CAVLC 编码器实现需要 300~500 个时钟。这样编码器的具有极高的吞吐率，可以达到传统 CAVLC 编码器的吞吐率的 3 倍以上。与直接用四个单符号编码的编码器来实现高吞吐率的高性能编码器相比，本发明的硬件面积较小。

附图说明

[0036] 以下结合附图对本发明做进一步的描述。

[0037] 图 1 是本发明中的顶层架构图。

[0038] 图 2 是 CAVLC 的流程图。

[0039] 图 3 是本发明中的 VLCN 提前计算的结构图。

[0040] 图 4 是本发明中的二级流水级的时序转换图。

[0041] 图 5 是本发明中的一个 4x4 块系数扫描的顺序示意图。

[0042] 图 6 是本发明中的扫描级架构图。

[0043] 图 7 是 Level 码字的结构。

[0044] 图 8 是本发明中的单个 Level 编码器的具体实现。

[0045] 图 9 是本发明中的四个 Level 并行编码器的架构。

[0046] 图 10 是本发明中的四个 Run_before 编码器的架构。

[0047] 图 11 是本发明中的比特拼接器的时序图。

具体实施方式

[0048] 顶层架构

本发明设计的高吞吐率的 CAVLC 编码器，采用扫描级和编码级组成的二级流水结的架构。本发明的 CAVLC 编码器的顶层架构如图 1 所示。整个 CAVLC 编码器可以分为扫描级，编码级和 CAVLC 控制器三部分。CAVLC 控制器通过控制扫描级控制器和编码级控制器来控制整个 CAVLC 编码器的时序和功能。扫描级与编码级两级中采用了一块寄存器来存储扫描级统计得到的结果，这样，就可以使得扫描级和编码级能够流水线并行处理。扫描级完成一个 4x4 块的扫描后，将得到的统计信息直接存储在寄存器中，然后编码级开始编码，而扫描级则继续扫描下一个 4x4 块。通过这样的结构，大大提高了整个编码器的吞吐率。扫描级包含有扫描级控制器，TrailingOnes 计数器，TotalCoeff 计数器，Total_Zeros 计数器，Run_before 计数器和 Level 检测器组成，扫描级的功能是统计 CAVLC 元素值。编码级由编码级控制器，TotalCoeff 编码器，Total_Zeros 编码器，Level 编码器，Run_before 编码器和比特流拼接器组成，编码级的功能是编码扫描级统计得到的 CAVLC 元素值。

[0049] 流水线结构设计

本发明采用二级流水线的结构，即扫描级和编码级。每一级都是采用四个时钟来完，这两级流水线时序状态图可以由图 4 表示。扫描级五个状态，编码级五个状态。

[0050] 在扫描级五个状态中，扫描初始代表初始状态， 扫描时钟 0，扫描时钟 1，扫描时钟 2，扫描时钟 3，分别表示扫描级的四个工作状态。

[0051] 扫描采用反 Zig-Zag 顺序，如图 5 所示，0~15 代表在一个 4x4 块中各系数的位置。

[0052] 在扫描时钟 0，扫描的残差系数是 15，14，13，7。在扫描时钟 1，扫描的残差系数是 12，11，10，9。在扫描时钟 2，扫描的残差系数是 6，5，4，3。在扫描时钟 3 扫描的残差系数是 8，2，1，0。

[0053] 在编码级五个状态中，编码初始代表编码级的初始状态，编码时钟 0，编码时钟 1，编码时钟 2，编码时钟 3，分别表示编码级的四个工作状态。

[0054] 扫描级设计

1. 本发明的扫描级包含有扫描级控制器，TrailingOnes 计数器，TotalCoeff 计数器，Total_Zeros 计数器，Run_before 计数器和 Level 检测器组成。扫描级控制器通过一个状态机来控制扫描级在每个时钟从残差系数缓存中取 4 个残差系数来统计 CAVLC 编码的一些元素，如 TotalCoeff，Total_Zeros，TrailingOnes，Level，Run_before 等。扫描级的结构示意图如图 6 所示。其具体步骤如下所示：

1) 从系数缓存中取的四个残差系数为 coeff0~coeff3。然后根据下面代码 (1) 和 (2) 所示得到 8 个状态指示位：s0~s3 和 c0~c3。其中 s0~s3 代表各残差系数是否等于 0，c0~c3 代表各残差系数是否等于 ±1：

$$\begin{cases} if\ coeff0=0, & s0=0 \\ else & s0=1 \\ if\ coeff1=0, & s1=0 \\ else & s1=1 \\ if\ coeff2=0, & s2=0 \\ else & s2=1 \\ if\ coeff3=0, & s3=0 \\ else & s3=1 \end{cases} \tag{1}$$

$$\begin{cases} if\ coeff0=\pm1, & c0=1 \\ else & c0=0 \\ if\ coeff1=\pm1, & c1=1 \\ else & c1=0 \\ if\ coeff2=\pm1, & c2=1 \\ else & c2=0 \\ if\ coeff3=\pm1, & c3=1 \\ else & c3=0 \end{cases} \tag{2}$$

2) 现在我们可以根据 (1) 和 (2) 得到的 8 个状态指示位：s0~s3 和 c0~c3，来得到 CAVLC 需要编码的元素值。其中根据 s0~s3，可以得到 TotalCoeff，Total_Zeros 和 Run_before。并把不为零的系数存储到 Level 缓存中。而根据 s0~s3 和 c0~c3 我们就可以确定

TrailingOnes 的值。这些元素值的具体确定过程如下所述：

1> TotalCoeff 的确定方法：将 s0~s3 中 1 的个数累加起来即为 TotalCoeff 的值。

[0055] 2> Total_Zeros 的确定方法：在每个 4x4 块的扫描开始时，将一个内部变量 TZ_EN 设置为 0，TZ_EN 的表示当前 4x4 的 Total_Zeros 计算的使能位，TZ_EN 在 s0~s3 中有某个状态位值为 1 时设为 1。在每个 4x4 块的扫描开始，在 TZ_EN 第一次设置为 1 时，我们开始计算 s0~s3 中第一个 1 以后的 0 个数，然后对当前 4x4 块后面的扫描时的 s0~s3 状态位的 0 的个数累加。

[0056] 3> Run_before 确定方法：对于每个非零残差系数，都有对应有一个 Run_before 值。在 s0~s3 中，我们计算每个 1 前面有多少个 0，即为对应的非零残差系数的 Run_before 值。

[0057] 4> Level 的确定：在 s0~s3 中，相应的等于 1 的数存储在 Level 寄存器中。这样可能会出现把拖尾 1 存在 Level 中的情况。编码时需要根据 TrailingOnes 的大小，将前面 TrailingOnes 个从 Level 寄存器中读出的值不作为 Level 编码。

[0058] 5> TrailingOnes 确定方法：TrailingOnes 的确定需要同时考虑 coeff0~coeff3 是否为 0 和是否等于 ±1。为了确定 TrailingOnes，我们把 coeff0~coeff3 分成两组，其中 coeff0，coeff1 为低位组，coeff2 和 coeff3 为高位组。对于这两组残差系数，分别有一个对应的有效标志位 Valid_L（低位组有效位）/Valid_H（高位组有效位）表示当前数残差系数组中是否有大于 1 或小于负 1 的情况出现，即代表当前组的计算的 TrailingOnes 是否有效。 我们用 TrailingOnes_L（低位组 TrailingOnes 计算结果）和 TrailingOnes_H（高位组 TrailingOnes 计算结果）来分别代表两组残差系数的 TrailingOnes 的计算结果。我们用 Valid_A 表示当前 4x4 块中是否有残差系数大于 1 或小于负 1 的情况，若 Valid_A 为 1，则表示之前的残差系数中有大于 1 或小于负 1 的情况。于是有下面的 TrailingOnes 的计算方法：

如果 Valid_A=1，那么：TrailingOnes = TrailingOnes_tmp。

[0059] 如果 Valid_A=0，那么 TrailingOnes 的计算方法如下所描述：

a）如果 Valid_L = 0，Valid_H= 0，则 TrailingOnes = TrailingOnes_L+TrailingOnes_H+ TrailingOnes_tmp。

[0060] b）如果 Valid_L = 0，Valid_H = 1，则 TrailingOnes = TrailingOnes_L+TrailingOnes_H +TrailingOnes_tmp。

[0061] c）如果 Valid_L=1，Valid_H=0，则 TrailingOnes=TrailingOnes_tmp+TrailingOnes_L。

[0062] d）如果 Valid_L=1，Valid_H=1，则 TrailingOnes=TrailingOnes_tmp+TrailingOnes_L。

[0063] 其中 TrailingOnes_tmp 表示为暂时存储在内部的 TrailingOnes 计算结果，在每个 4x4 块的扫描开始时 TrailingOnes_tmp 设置为 0，在每一组的 4 个系数统计过程完成后 TrailingOnes_tmp 的值设为 TrailingOnes。 最后得到的 TrailingOnes 需要限定在小于等于 3 的范围。

[0064] 内部寄存器设计

扫描级扫描得到的统计结果，如：TotalCoeff，Total_Zeros，TrailingOnes，Level，

Run_before 等，需要存储在寄存器中以供后面的编码级来使用。Level 和 Run_before 寄存器采用的是一个寄存器阵列的结构，每次可以向 Level 和 Run_before 寄存器中写 0~4 个数，其个数的控制是根据 s0~s3 来实现的，具体写的个数为（s0+s1+s2+s3）。Level 和 Run_before 寄存器在读的阶段每次读 4 个值。即在每个时钟，编码级会从 Level 和 Run_before 寄存器中每次读取 4 个 Level 和 Run_before。

[0065] 编码级设计

编码级主要由编码级控制器，TotalCoeff 编码器，Total_Zeros 编码器，Level 编码器，Run_before 编码器和比特流拼接器组成。编码级控制器控制 TotalCoeff 编码器和 Total_Zeros 编码器在编码级的第一个时钟时，根据扫描级的 TotalCoeff，Total_Zeros 和 TrailingOnes 的值，并查询相应的表格得到码字。编码级控制器通过 TotalCoeff 的值来控制 Level 编码器中在每个时钟编码四个 Level 值。编码级控制器通过 Total_Zeros 的值来控制 Run_before 编码器中在每个时钟编码四个 Run_before 值。编码级控制器控制比特流拼接器在每个时钟把 TotalCoeff 编码器，Total_Zeros 编码器，Level 编码器和 Run_before 编码器产生的码字拼接成为比特流。TotalCoeff 编码器，Total_Zeros 编码器均是通过查表即可以得到相应的码字。而 Level 编码器和 Run_before 编码器均需要特别的设计，来满足四个 Level 并行编码和四个 Run_before 并行编码。其具体设计如下。

[0066] Level 编码器设计

在对 Level 编码时 H.264/AVC 中有 7 张表格来编码 Level，本发明采用了一个 VLCN(编码 Level 的 7 张表格的表格序号)提前计算的技术，同时决定四个 Level 幅值的编码对应的表格序号。其具体实现如图 3 所示。这样就可以同时对四个 Level 进行编码。然后把四个 Level 编码得到的码字和对应的长度拼接起来。VLCN 提前计算的过程具体如下面的代码所示：

```
incVLC[7] = { 0, 3, 6, 12, 24, 48, 0xffff };
if( first level  && TotalCoeff>10 &&TrailingOnes<3)
vlc0 = 1;
       else if(first level)
              vlc0 = 0;
else if( abs(reg_level3) > incVLC[reg_vlc3] )
vlc0 = reg_vlc3+1;
       else
vlc0 = reg_vlc3;
if( vlc0 =0 )
vlc1 = vlc0+1;
else if( abs(level0) > incVLC[vlc0] )
vlc1 = vlc0+1;
       else
              vlc1=vlc0;
if( abs(level1) > incVLC[vlc1] )
vlc2 = vlc1+1;
```

```
        else
                vlc2 = vlc1;
    if( abs(level2) > incVLC[vlc2] )
    vlc3 = vlc2+1;
        else
                vlc3 = vlc2;
```

上述代码中 abs 代表的的是取绝对值的操作，incVLC[7] = { 0, 3, 6, 12, 24, 48, 0xffff } 表示的是表格选择的条件。vlc0~vlc3 分别表示编码当前四个 Level 值分别对应的表格序号(即 H.264/AVC 中的 suffixlength)。level0~ level3 表示当前编码的幅值数，reg_level3 表示前一组四个 Level 值的中的 level3。reg_vlc3 表示编码前一组四个 Level 值时编码 level3 时用的表格序号 vlc3。上述代码用硬件实现，可以在极短的时间之内得到 vlc0~vlc3，这样可以做到四个 Level 的并行编码。

[0067] 在得到了四个 Level 值时编码的表格序号 vlc0~vlc3。我们需要根据每个 level 和其相应的表格序号来编码每个 level 值。每个 Level 的码字结构包含前缀(level_prefix)，后缀(level_prefix)和它们中间的 1 组成。其码字结构如图 7 所示。前缀由 0 组成。其中后缀的长度由 levelSuffixsSize 来表示。

[0068] 单个 Level 值的编码具体实现框图如图 8 所示。它可以分为两部分，一部分为规则码编码器，另一部分为逃逸码(escape)编码器。逃逸码(escape)编码器占了单个 Level 值的编码器的大部分。本发明中充分利用了硬件实现的并行性，同时判断多个值的大小关系来实现速度上的提升。单个 Level 值的编码器的输出结果为前缀(level_prefix)，后缀(level_prefix)和后缀的长度(levelSuffixsSize)。

[0069] 四个 Level 值的并行编码出来的结果可以根据码字的长度，通过移位以后进行或操作来拼接成 FourLevelCode，即四个 Level 的编码码字，即如图 9 所示。

[0070] Run_before 编码器设计

同 Level 编码器一样，对于 Run_before 的编码时，每个时钟也是编码四个 Run_before。编码 Run_before 主要可以通过查找表 1 来实现：

表 1

Run_before	ZerosLeft						
	1	2	3	4	5	6	>6
0	1	1	11	11	11	11	111
1	0	01	10	10	10	000	110
2	-	00	01	01	011	101	101
3	-	-	00	001	010	011	100
4	-	-	-	000	001	010	011
5	-	-	-	-	000	101	010
6	-	-	-	-	-	100	001
7	-	-	-	-	-	-	0001
8		-	-	-	-	-	00001
9	-	-	-	-	-	-	000001
10	-	-	-	-	-	-	0000001
11	-	-	-	-	-	-	00000001
12	-	-	-	-	-	-	000000001
13	-	-	-	-	-	-	0000000001
14	-	-	-	-	-	-	00000000001

在表 1 中 zerosLeft 表示编码当前 Run_before 时剩余 0 的个数。Run_before 可以从 Run_before 缓存中读取得到。在本发明中，每个时钟有四个 Run_before 值从 Run_before 缓存中读取出来，用 Run_before0~Run_before3 来代表这四个 Run_before 值。为了查找上述表格，我们还需要确定这个四 Run_before 值相应的剩余 0 的个数 zerosLeft0~zerosLeft3。其计算过程如下所示：

zerosLeft0 = Reg_zerosLeft ;

zerosLeft1 = zerosLeft0 - Run_before0 ;

zerosLeft2 = zerosLeft1 - Run_before1 ;

zerosLeft3 = zerosLeft2 - Run_before2 ;

其中 Reg_zerosLeft 表示为当前 4x4 块中剩余 0 的个数，其取值为：如果当前 Run_before 为当前 4x4 块的第一个 Run_before 值，则 Reg_zerosLeft = Total_Zeros。否则，Reg_zerosLeft = Reg_zerosLeft3- Reg_Run_before3。其中 Reg_Run_before3 和 Reg_zerosLeft3 分别表示为上一组四个 Run_before 和其对应的四个 zerosLeft 中的 Run_before3 和 zerosLeft3。

[0071] 在编码中有两种情况 Run_before 值是不需要编码的，一种是 zerosLeft 为 0 的情况，另一种是当前 Run_before 为当前 4x4 块中的最后一个 Run_before 值。

[0072] 四个 Run_before 并行编码出来的结果可以根据码字的长度，通过移位以后进行或操作来拼接成 FourRunCode，即四个 Run_before 的编码码字。本发明 FourRunCode 和 Total_Zeros 码字组合在一起，存在一个内部寄存器中，在编码级的最后一个时钟送给比特流拼接器。内部存储的码字名字称为 Total_Zeros/Run_beforeCode。Run_before 编码器的结构如图 10 所示。

[0073] 比特流拼接器设计

比特流拼接器的作用是把 TotalCoeff 编码器，Total_Zeros 编码器，Level 编码器和

Run_before 编码器产生的码字拼接成为比特流。编码级在四个时钟内完成，本发明的比特流拼接器的时序设计为：第一个时钟周期，比特流拼接器需要把 TotalCoeff 码字和第一组 FourLevelCode 打包在一起，接下来两个周期，比特流拼接器只需要打包 Level 编码器输出的 FourLevelCode。在最后一个时钟周期，比特流拼接器将最后的 FourLevelCode 和 Total_Zeros/Run_beforeCode 打包在一起。比特流拼接器的时序图如图 11 所示。

[0074] 结果分析

本发明通过采用增加硬件并行的方式来提高 CAVLC 编码器的吞吐率。本发明可以将编码一个宏块的残差系数最多时间所需要的时钟数缩小为 108 个(未含 latency 时钟数)，而传统的 CAVLC 编码器实现需要 300~500 个时钟。这样编码器的具有极高的吞吐率，可以达到传统 CAVLC 编码器的吞吐率的 3 倍以上。本发明采用 verilog 硬件描述语言实现，综合结果表明其硬件消耗约为传统 CAVLC 编码器的 1.5 倍左右。这样本设计的设计效率为传统 CAVLC 编码器的 2 倍以上。且本设计的最高可以支持到 H.264/AVC 的 4Kx2K(4096x2160) 60 帧每秒的超高分辨率超高性能的编码需求，性能极高。

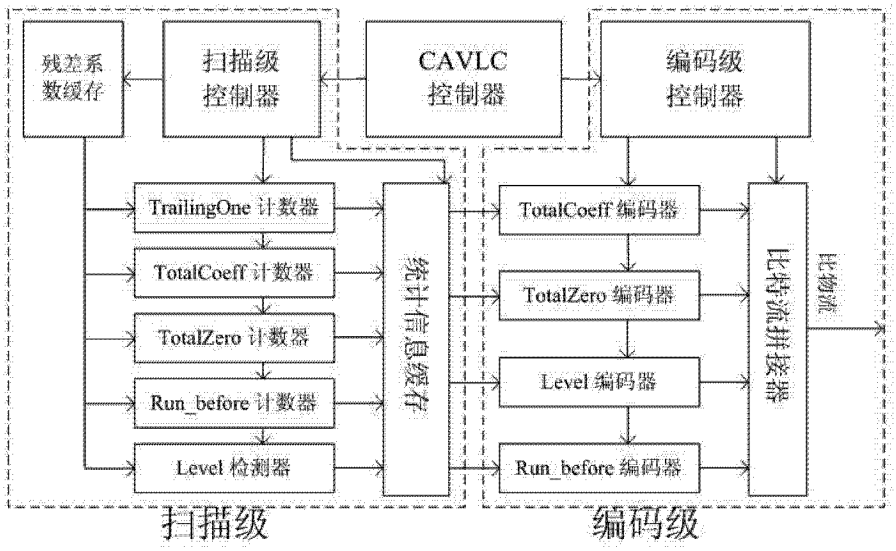

残差系数缓存
扫描级控制器
CAVLC控制器
编码级控制器
TrailingOne 计数器
TotalCoeff 计数器
TotalZero 计数器
Run_before 计数器
Level 检测器
统计信息缓存
TotalCoeff 编码器
TotalZero 编码器
Level 编码器
Run_before 编码器
比特流拼接器
比特流
扫描级
编码级

图 1

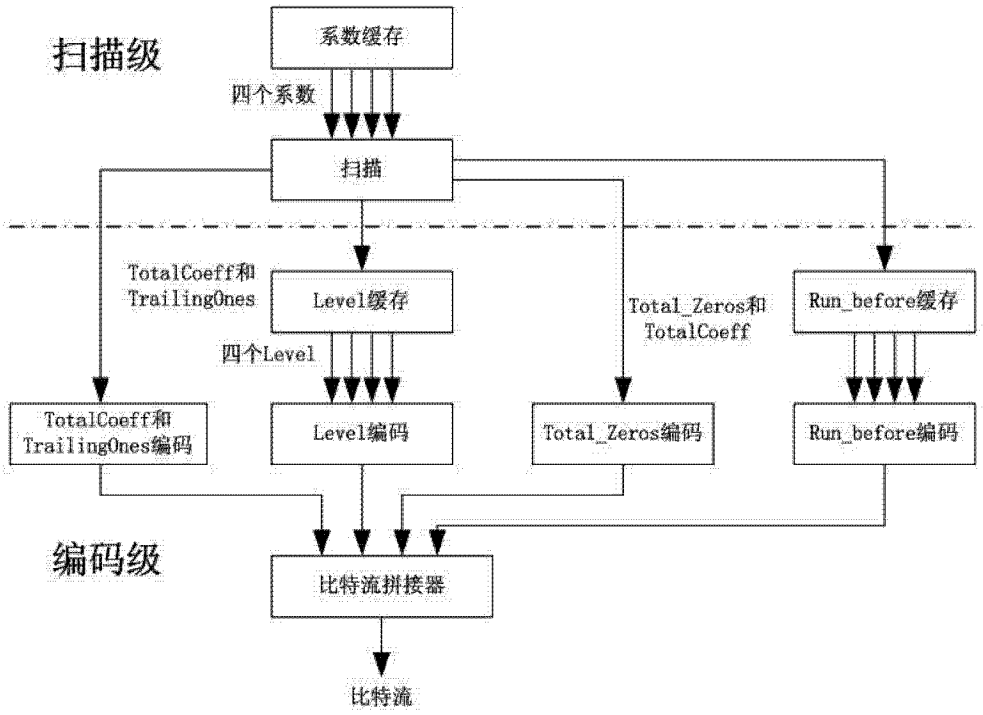

扫描级
系数缓存
四个系数
扫描
TotalCoeff和TrailingOnes
Level缓存
四个Level
Total_Zeros和TotalCoeff
Run_before缓存
TotalCoeff和TrailingOnes编码
Level编码
Total_Zeros编码
Run_before编码
编码级
比特流拼接器
比特流

图 2

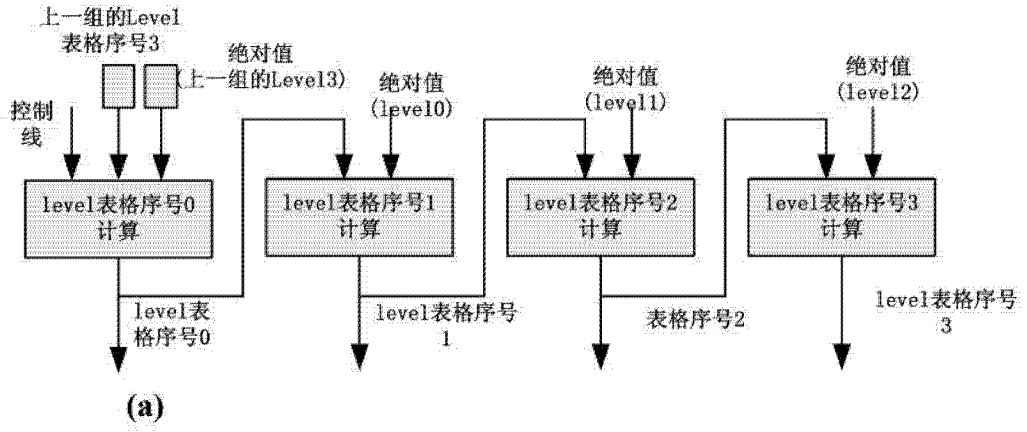

上一组的Level
表格序号3
绝对值
(上一组的Level3)
绝对值
(level0)
绝对值
(level1)
绝对值
(level2)
控制
线
level表格序号0
计算
level表格序号1
计算
level表格序号2
计算
level表格序号3
计算
level表
格序号0
level表格序号
1
表格序号2
level表格序号
3
(a)

图 3

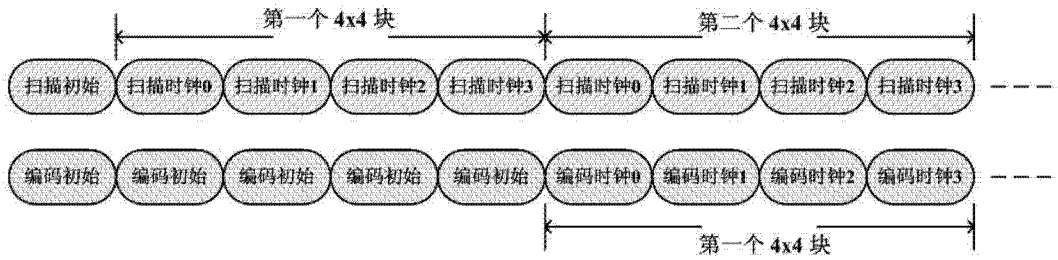

第一个 4x4 块
第二个 4x4 块
扫描初始
扫描时钟0
扫描时钟1
扫描时钟2
扫描时钟3
扫描时钟0
扫描时钟1
扫描时钟2
扫描时钟3
编码初始
编码初始
编码初始
编码初始
编码初始
编码时钟0
编码时钟1
编码时钟2
编码时钟3
第一个 4x4 块

图 4

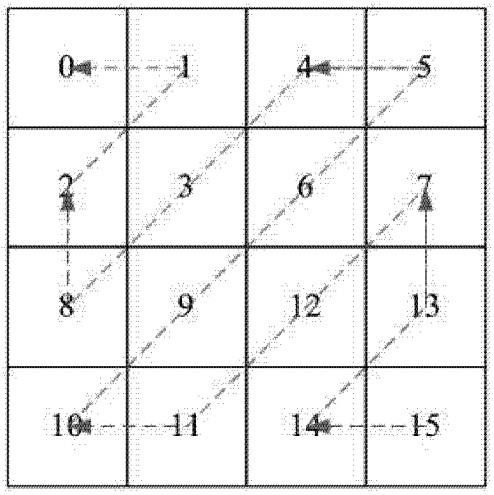

0
1
4
5
2
3
6
7
8
9
12
13
10
11
14
15

图 5

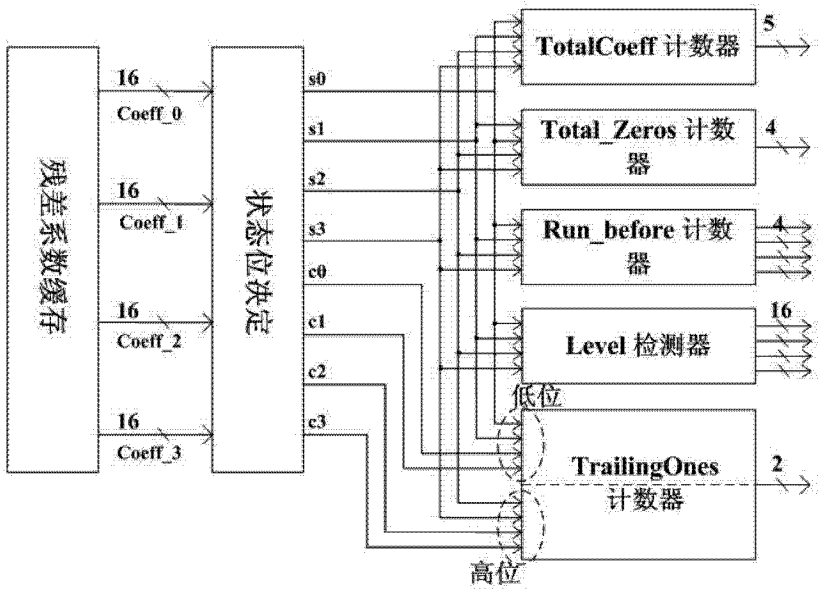

残差系数缓存
16
Coeff_0
16
Coeff_1
16
Coeff_2
16
Coeff_3
状态位决定
s0
s1
s2
s3
c0
c1
c2
c3
TotalCoeff 计数器
5
Total_Zeros 计数器
4
Run_before 计数器
4
Level 检测器
16
低位
TrailingOnes 计数器
2
高位

图 6

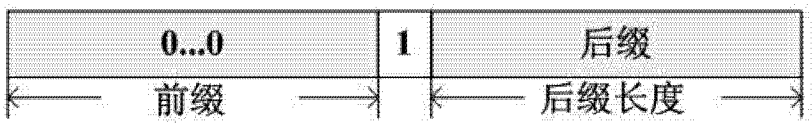

0...0
1
后缀
前缀
后缀长度

图 7

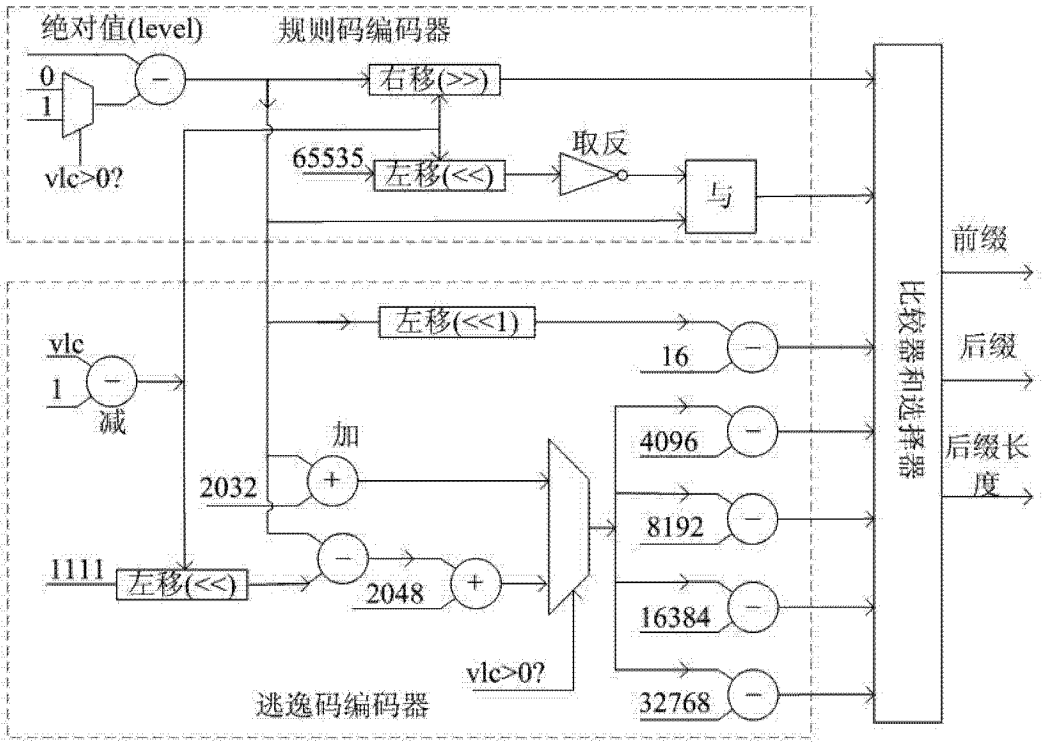

绝对值(level)
规则码编码器
0
1
vlc>0?
右移(>>)
65535
左移(<<)
取反
与
vlc
1
减
左移(<<1)
16
加
2032
4096
8192
1111
左移(<<)
2048
16384
vlc>0?
32768
逃逸码编码器
比较器和选择器
前缀
后缀
后缀长度

图 8

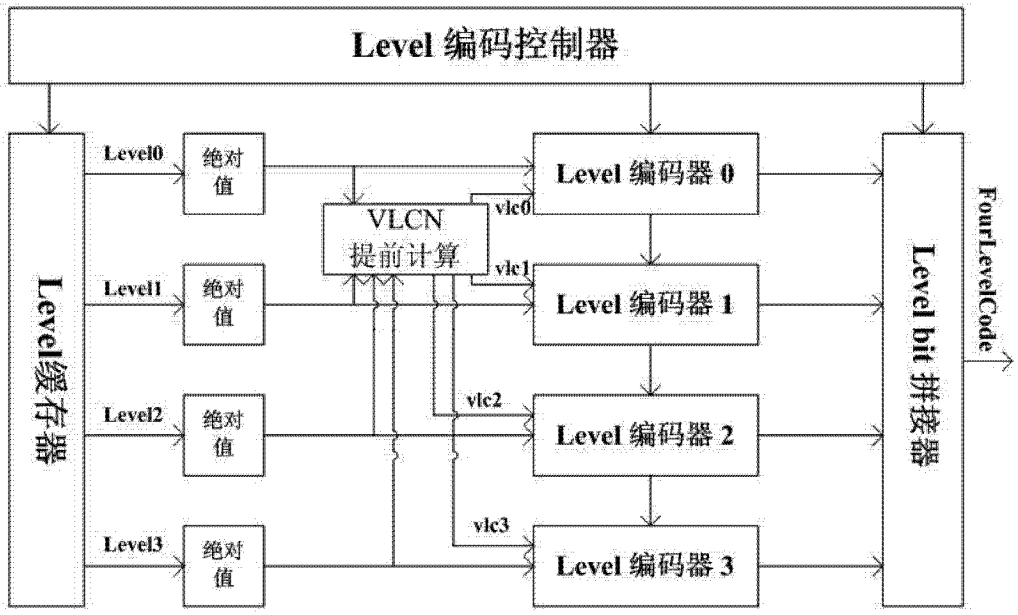

Level 编码控制器
Level缓存器
Level0
Level1
Level2
Level3
绝对值
VLCN
提前计算
vlc0
vlc1
vlc2
vlc3
Level 编码器 0
Level 编码器 1
Level 编码器 2
Level 编码器 3
Level bit 拼接器
FourLevelCode

图 9

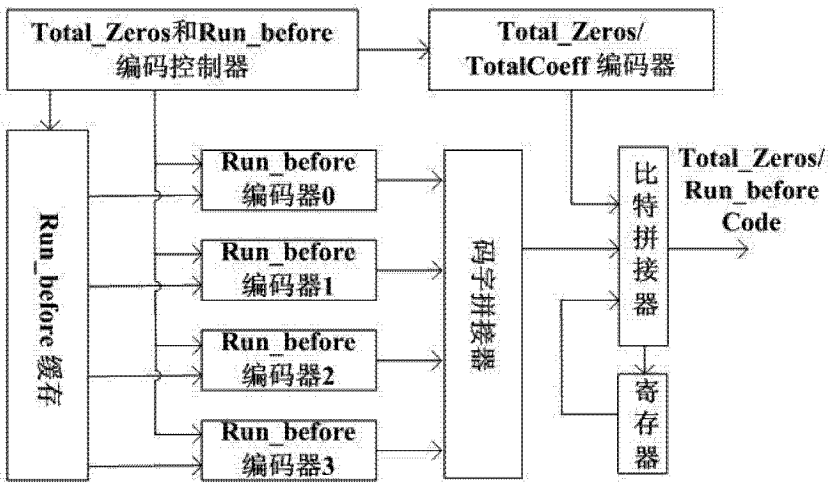

Total_Zeros和Run_before
编码控制器
Total_Zeros/
TotalCoeff 编码器
Run_before 缓存
Run_before
编码器0
Run_before
编码器1
Run_before
编码器2
Run_before
编码器3
码字拼接器
比特拼接器
Total_Zeros/
Run_before
Code
寄存器

图 10

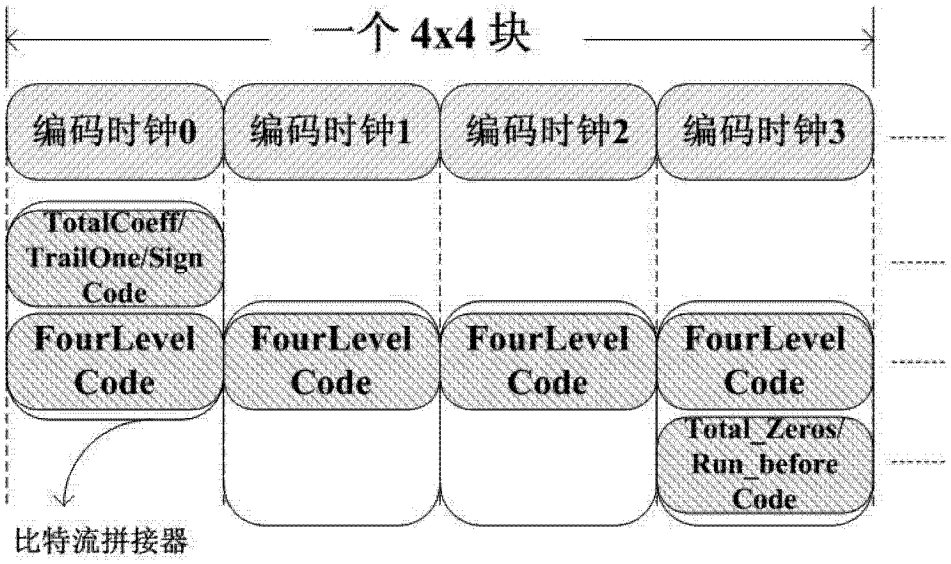

一个 4x4 块
编码时钟0
编码时钟1
编码时钟2
编码时钟3
TotalCoeff/
TrailOne/Sign
Code
FourLevel
Code
FourLevel
Code
FourLevel
Code
FourLevel
Code
Total_Zeros/
Run_before
Code
比特流拼接器

图 11