(19)国家知识产权局

(12)发明专利

(10)授权公告号 CN 113508600 B
(45)授权公告日 2024.12.24

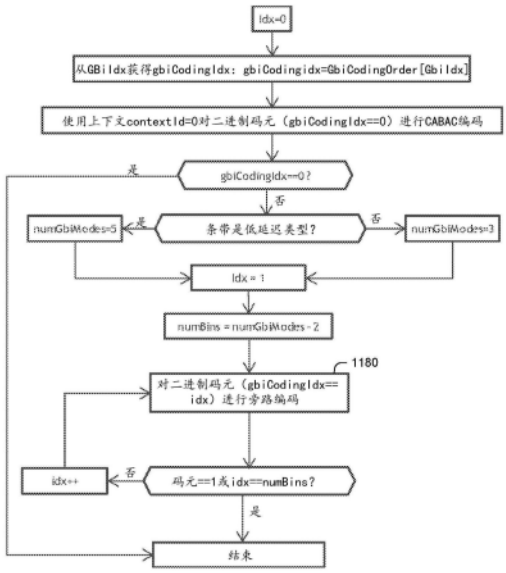
(21)申请号 202080017855.9
(22)申请日 2020.03.04
(65)同一申请的已公布的文献号
申请公布号 CN 113508600 A
(43)申请公布日 2021.10.15
(30)优先权数据
19305278.4 2019.03.11 EP
19305693.4 2019.05.29 EP
(85)PCT国际申请进入国家阶段日
2021.08.31
(86)PCT国际申请的申请数据
PCT/US2020/021011 2020.03.04
(87)PCT国际申请的公布数据
WO2020/185468 EN 2020.09.17
(73)专利权人 交互数字VC控股公司
地址 美国特拉华州
(72)发明人 F.莱林内克 T.波伊里尔 陈娅
(74)专利代理机构 北京市柳沈律师事务所 11105
专利代理师 赵碧洋
(51)Int.Cl.
H04N 19/91(2006.01)
H04N 19/13(2006.01)
H04N 19/88(2006.01)
H04N 19/50(2006.01)
H04N 19/577(2006.01)
(56)对比文件
US 2018332298 A1,2018.11.15
Benjamin Bross, et al., ."Versatile Video Coding (Draft 4)".JVET - v5, Joint Video Experts Team (JVET) of ITU - T SG16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, J.2019,258-259.
审查员 刘喆
权利要求书2页 说明书33页 附图23页

(54)发明名称

减少常规编解码二进制数的数量

(57)摘要

为了以CABAC进行编码,通过二值化过程将非二进制语法元素值映射到二进制序列(二进制数串)。二进制算术编解码引擎以常规(基于上下文的)模式和旁路模式对二进制数进行编码或解码。



CN 113508600 B

1.一种用于视频解码的方法,包括:

对多个二进制码元进行解码,其中使用旁路模式对所述多个二进制码元的开始处的一个或多个二进制码元进行熵解码,并且其中以基于上下文的模式对所述多个二进制码元中的其余部分进行熵解码;以及

基于所述多个二进制码元获得对应于二值化方案的语法元素。

2.根据权利要求1所述的方法,其中,所述二值化方案是截断莱斯二值化方案。

3.根据权利要求1所述的方法,其中,所述语法元素指示一组加权因子中用于在形成两个预测器的加权和时对所述两个预测器中的一个进行加权的加权因子。

4.根据权利要求3所述的方法,其中,响应于所述加权因子获得另一加权因子,所述另一加权因子用于在形成所述加权和时对所述两个预测器中的另一个进行加权。

5.根据权利要求1所述的方法,其中,所述多个二进制码元中的所述其余部分中的每个二进制码元使用不同的上下文模型。

6.一种用于视频编码的方法,包括:

访问要被编码的块;以及

对语法元素进行编码,

其中,多个二进制码元基于所述语法元素和二值化方案被获得,

其中,使用旁路模式对所述多个二进制码元的开始处的一个或多个二进制码元进行熵编码,以及

其中,以基于上下文的模式对所述多个二进制码元中的其余部分进行熵编码。

7.根据权利要求6所述的方法,其中,所述二值化方案是截断莱斯二值化方案。

8.根据权利要求6所述的方法,其中,所述语法元素指示一组加权因子中用于在形成两个预测器的加权和时对所述两个预测器中的一个进行加权的加权因子。

9.根据权利要求8所述的方法,其中,响应于所述加权因子获得另一加权因子,所述另一加权因子用于在形成所述加权和时对所述两个预测器中的另一个进行加权。

10.根据权利要求6所述的方法,其中,所述多个二进制码元中的所述其余部分中的每个二进制码元使用不同的上下文模型。

11.一种用于视频解码的装置,包括一个或多个处理器,其中,所述一个或多个处理器被配置为:

对多个二进制码元进行解码,其中使用旁路模式对所述多个二进制码元的开始处的一个或多个二进制码元进行熵解码,并且其中以基于上下文的模式对所述多个二进制码元中的其余部分进行熵解码;以及

基于所述多个二进制码元获得对应于二值化方案的语法元素。

12.根据权利要求11所述的装置,其中,所述二值化方案是截断莱斯二值化方案。

13.根据权利要求11所述的装置,其中,所述语法元素指示一组加权因子中用于在形成两个预测器的加权和时对所述两个预测器中的一个进行加权的加权因子。

14.根据权利要求13所述的装置,其中,响应于所述加权因子获得另一加权因子,所述另一加权因子用于在形成所述加权和时对所述两个预测器中的另一个进行加权。

15.根据权利要求11所述的装置,其中,所述多个二进制码元中的所述其余部分中的每个二进制码元使用不同的上下文模型。

16.一种用于视频编码的装置,包括一个或多个处理器,其中,所述一个或多个处理器被配置为:

访问要被编码的块;以及

对语法元素进行编码,

其中,多个二进制码元基于所述语法元素和二值化方案被获得,

其中,使用旁路模式对所述多个二进制码元的开始处的一个或多个二进制码元进行熵编码,以及

其中,以基于上下文的模式对所述多个二进制码元中的其余部分进行熵编码。

17.根据权利要求16所述的装置,其中,所述二值化方案是截断莱斯二值化方案。

18.根据权利要求16所述的装置,其中,所述语法元素指示一组加权因子中用于在形成两个预测器的加权和时对所述两个预测器中的一个进行加权的加权因子。

19.根据权利要求18所述的装置,其中,响应于所述加权因子获得另一加权因子,所述另一加权因子用于在形成所述加权和时对所述两个预测器中的另一个进行加权。

20.根据权利要求16所述的装置,其中,所述多个二进制码元中的所述其余部分中的每个二进制码元使用不同的上下文模型。

减少常规编解码二进制数的数量

技术领域

[0001] 本申请实施例通常涉及用于视频编码或解码的方法和装置,并且更具体地涉及用于减少熵编码和解码中的常规编解码二进制数的数量的方法和装置。

背景技术

[0002] 为了实现高压缩效率,图像和视频编解码方案通常采用预测以及变换来利用视频内容中的空间和时间冗余。通常,使用帧内或帧间预测以利用帧内或帧间画面相关性,然后对原始块和预测块之间的、常常被表示为预测误差或预测残差的差进行变换、量化以及熵编解码。为了重构视频,通过进行与熵编解码、量化、变换和预测相对应的逆过程来对压缩数据进行解码。

发明内容

[0003] 根据一实施例,提供了一种视频解码的方法,包括:从比特流中对多个二进制码元进行解码,其中使用基于上下文的模式对所述多个二进制码元中的第一二进制码元进行熵解码,并且其中要以旁路模式对在第一二进制码元之后的每个二进制码元进行熵解码;获得由所述多个二进制码元表示的、对应于二值化方案的索引;以及将对块的预测形成为两个预测器的加权和,其中所述索引指示在形成所述加权和时用于对所述两个预测器进行加权的相应加权因子。

[0004] 根据一实施例,提供了一种视频编码的方法,包括:访问要被编码的块;将对所述块的预测形成为两个预测器的加权和;以及对指示在形成所述加权和时用于对所述两个预测器进行加权的相应加权因子的索引进行编码,其中,使用二值化方案将所述索引二值化为多个二进制码元,其中使用基于上下文的模式对所述多个二进制码元中的第一二进制码元进行熵编码,以及其中要以旁路模式对在第一二进制码元之后的每个二进制码元进行熵编码。

[0005] 根据另一实施例,提供了一种用于视频解码的装置,包括一个或多个处理器,其中所述一个或多个处理器被配置为:从比特流中对多个二进制码元进行解码,其中使用基于上下文的模式对所述多个二进制码元中的第一二进制码元进行熵解码,以及其中要以旁路模式对在第一二进制码元之后的每个二进制码元进行熵解码;获得由所述多个二进制码元表示的、对应于二值化方案的索引;以及将对块的预测形成为两个预测器的加权和,其中所述索引指示在形成所述加权和时用于对所述两个预测器进行加权的相应加权因子。

[0006] 根据另一实施例,提供了一种用于视频编码的装置,包括一个或多个处理器,其中所述一个或多个处理器被配置为:访问要被编码的块;将对所述块的预测形成为两个预测器的加权和;以及对指示在形成所述加权和时用于对所述两个预测器进行加权的相应加权因子的索引进行编码,其中,使用二值化方案将所述索引二值化为多个二进制码元,其中使用基于上下文的模式对所述多个二进制码元中的第一二进制码元进行熵编码,以及其中要以旁路模式对在第一二进制码元之后的每个二进制码元进行熵编码。

[0007] 根据另一实施例,提供了一种视频解码的装置,包括:用于从比特流中对多个二进制码元进行解码的部件,其中使用基于上下文的模式对所述多个二进制码元中的第一二进制码元进行熵解码,以及其中要以旁路模式对在第一二进制码元之后的每个二进制码元进行熵解码;用于获得由所述多个二进制码元表示的、对应于二值化方案的索引的部件;以及用于将对块的预测形成为两个预测器的加权和的部件,其中所述索引指示在形成所述加权和时用于对所述两个预测器进行加权的相应加权因子。

[0008] 根据另一实施例,提供了一种视频编码的装置,包括:用于访问要被编码的块的部件;用于将对所述块的预测形成为两个预测器的加权和的部件;以及用于对指示在形成所述加权和时用于对所述两个预测器进行加权的相应加权因子的索引进行编码的部件,其中,使用二值化方案将所述索引二值化为多个二进制码元,其中使用基于上下文的模式对所述多个二进制码元中的第一二进制码元进行熵编码,以及其中要以旁路模式对在第一二进制码元之后的每个二进制码元进行熵编码。

附图说明

[0009] 图1示出了视频编码器的实施例的框图。

[0010] 图2是示出用于表示压缩的HEVC画面的编解码树单元和编解码树概念的图示示例。

[0011] 图3是示出编解码树单元到编解码单元、预测单元和变换单元的划分的图示示例。

[0012] 图4示出了视频解码器的实施例的框图。

[0013] 图5示出了CABAC解码过程。

[0014] 图6示出了CABAC编码过程。

[0015] 图7是示出了四叉树加二叉树(Quad-Tree Plus Binary-Tree,QTBT)CTU表示的图示示例。

[0016] 图8是示出编解码单元划分模式的图示示例。

[0017] 图9示出了对VVC草案4中的GBi索引进行编解码的过程。

[0018] 图10示出了对VVC草案4中的GBi索引进行解析的过程。

[0019] 图11示出了根据实施例的对GBi索引进行编解码的过程,并且图11A示出了根据另一实施例的对GBi索引进行编解码的过程。

[0020] 图12示出了根据实施例的对GBi索引进行解析的过程,图12A示出了根据另一实施例的对GBi索引进行解析的过程,图12B和图12C分别示出了在非低延迟模式和低延迟模式下的gbiCodingIndex二值化和编解码/解析过程,并且图12D示出了当用信令通知默认GBI模式的第一二进制数等于0时的树。

[0021] 图13是示出AMVP模式中的运动矢量预测器候选列表构建的图示示例。

[0022] 图14是示出被视为构建AMVP候选列表的空间和时间运动矢量预测候选的图示示例。

[0023] 图15示出了VVC草案4中的AMVP中的运动矢量预测索引的CABAC编解码。

[0024] 图16示出了根据实施例的对mvp_l0_flag和mvp_l1_flag语法元素进行旁路编解码的过程。

[0025] 图17示出了VVC草案5中SBT工具的原理。

[0026] 图18示出了对VVC草案5中的INTER编解码单元的SBT模式进行解码的过程。

[0027] 图19A示出了根据实施例的SBT模式解码过程,并且图19B示出了根据另一实施例的SBT模式解码过程。

[0028] 图20A示出了针对4×8和8×4INTRA编解码单元的允许的ISP划分,并且图20B示出了针对具有不同于4×8或8×4的尺寸的INTRA编解码单元的允许的ISP划分。

[0029] 图21示出了多参考行帧内预测的原理。

[0030] 图22示出了对VVC草案5中的“intra_luma_ref_idx”语法元素进行解析的过程。

[0031] 图23示出了根据实施例的对“intra_luma_ref_idx”语法元素进行解析的简化过程。

[0032] 图24示出了可以在其中实现本实施例的各方面的系统的框图。

具体实施方式

[0033] 图1示出了示例视频编码器100,诸如高效视频编解码(HEVC)编码器。图1还可以示出其中对HEVC标准进行改进的编码器或采用类似于HEVC的技术的编码器,诸如由JVET(联合视频探索小组)开发的VVC(Versatile Video coding,通用视频编解码)编码器。

[0034] 在本申请中,术语“重构”和“解码”可以被互换使用,术语“编码”或“编解码”可以被互换使用,并且术语“图像”、“画面”和“帧”可以被互换使用。通常但并非必须地,术语“重构”在编码器侧使用,而“解码”在解码器侧使用。

[0035] 在被编码之前,视频序列可以经历预编码处理(101),例如,将颜色变换应用于输入颜色画面(例如,从RGB 4:4:4到YCbCr 4:2:0的转换),或者对输入画面分量执行重映射,以便获取对压缩更具弹性的信号分布(例如,使用颜色分量之一的直方图均衡化)。元数据可以与预处理相关联,并被附加至比特流。

[0036] 为了对具有一个或多个画面的视频序列进行编码,将画面分割(102)成,例如,一个或多个条带(slice),其中每个条带可以包括一个或多个条带片段。在HEVC中,条带片段被组织成编解码单元、预测单元和变换单元。HEVC规范对“块”和“单元”进行区分,其中“块”寻址样本阵列中的特定区域(例如,亮度,Y),而“单元”包括与块相关联的全部编码的颜色分量(Y、Cb、Cr或单色)的共位块、语法元素以及预测数据(例如,运动矢量)。

[0037] 对于根据HEVC的编解码,如图2中所示,画面被分割成具有可配置尺寸(通常为64×64,128×128或256×256像素)的正方形的编解码树块(Coding Tree Block,CTB),并且编解码树块的连续集合被分组为条带。编解码树单元(Coding Tree Unit,CTU)包含编码的颜色分量的CTB。CTB是分割成编解码块(CB)的四叉树的根,并且编解码块(Coding Block)可以被分割成一个或多个预测块(PB)并且形成分割成变换块(Transform Block,TB)的四叉树的根,如图3中所示。对应于编解码块、预测块和变换块,编解码单元(Coding Unit,CU)包括预测单元(PU)、和变换单元(TU)的树结构集,PU包括全部颜色分量的预测信息,并且TU包括每个颜色分量的残差编解码语法结构。亮度分量的CB、PB、TB的尺寸应用于对应的CU、PU和TU。在本申请中,术语“块”可以被用于指,例如,CTU、CU、PU、TU、CB、PB和TB中的任何一个。此外,“块”还可以被用于指H.264/AVC或其他视频编解码标准中定义的宏块和分割,以及更一般地指各种尺寸的数据阵列。

[0038] 在编码器100中,如下所述,通过编码器元件对画面进行编码。以,例如,CU为单位

对要编码的画面进行处理。使用帧内或帧间模式对每个编解码单元进行编码。当编解码单元被以帧内模式编码时,其执行帧内预测(160)。在帧间模式下,执行运动估计(175)和补偿(170)。编码器决定(105)要使用帧内模式或帧间模式中的哪一个来对编解码单元进行编码,并且通过预测模式标志来指示帧内/帧间决定。通过用从原始图像块减去(110)预测块来计算预测残差。

[0039] 然后,对预测残差进行变换(125)和量化(130)。量化的变换系数以及运动矢量和其他语法元素被熵编解码(145)以输出比特流。作为非限制示例,基于上下文的自适应二进制算术编解码(Context-based Adaptive Binary Arithmetic Coding,CABAC)可以被用于将语法元素编码为比特流。

[0040] 为了使用CABAC进行编码,通过二值化过程将非二进制语法元素值映射为二进制序列,称为二进制数(bin)串。对于二进制数,上下文模型被选择。“上下文模型”是一个或多个二进制数的概率模型,并且取决于最近编解码的码元的统计数据从可用模型的选择中进行选取。每个二进制数的上下文模型由上下文模型索引(也称为“上下文索引”)进行标识,并且不同的上下文索引对应于不同的上下文模型。上下文模型将每个二进制数的概率存储为“1”或“0”,并且可以是自适应的或静态的。静态模型以针对二进制数“0”和“1”具有相等的概率来触发编解码引擎。在自适应编解码引擎中,基于二进制数的实际编解码的值来更新上下文模型。与自适应和静态模型相对应的操作模式分别被称为常规模式和旁路模式。基于上下文,二进制算术编解码引擎根据对应的概率模型对二进制数进行编码或解码。

[0041] 编码器还可以跳过变换并将量化直接应用于未变换的残差信号,例如,在4×4TU的基础上。编码器还可以绕过变换和量化两者,即,在不应用变换或量化过程的情况下直接对残差进行编解码。在直接PCM编解码中,不应用预测并且将编解码单元样本直接编解码为比特流。

[0042] 编码器对编码的块进行解码,以为进一步的预测提供参考。对量化的变换系数进行去量化(140)并进行逆变换(150)以对预测残差进行解码。通过将预测块和解码的预测残差进行组合(155)来重构图像块。将环内滤波器(165)应用于重构的画面,例如,以执行去块/SAO(样本自适应偏移)滤波,从而减少对伪像进行编码。滤波的图像被存储在参考画面缓冲器(180)。

[0043] 图4示出了诸如HEVC解码器的示例视频解码器200的框图。在解码器200中,如下所述,比特流由解码器元件进行解码。视频解码器200通常执行与图1中描述的编码过程互逆的解码过程,其执行视频解码,作为对视频数据进行编码的一部分。图4还可以示出其中对HEVC标准进行改进的解码器或采用类似于HEVC的技术的解码器,诸如VVC解码器。

[0044] 特别地,解码器的输入包括视频比特流,其可以由视频编码器100生成。比特流首先被熵解码(230)以获得变换系数、运动矢量、画面分割信息以及其他编解码的信息。如果CABAC用于熵编解码,则以与编码器上下文模型相同的方式对上下文模型进行初始化,并且基于上下文模型从比特流中对语法元素进行解码。

[0045] 图5描绘了给定输入编解码的比特流的语法元素的CABAC解码过程。这是图6的语法元素编解码过程的互逆过程。

[0046] 到图5的过程的输入包括编解码的比特流,通常符合视频压缩标准,诸如HEVC或VVC。在解码过程的任一点,解码器都知道接下来要解码哪个语法元素。这在标准化的比特

流语法和解码过程中有完全的规定。此外,其还知道当前要解码的语法元素是如何被二值化的(即,被表示为称为二进制数的二进制码元序列,每个等于“1”或“0”),以及二进制数串的每个二进制数是如何被编码的。

[0047] 因此,CABAC解码过程的第一阶段(图5的左侧)对一系列二进制数进行解码。对于每个二进制数,解码器知道二进制数是否已根据旁路模式或常规模式被编码。旁路模式包括简单地读取比特流中的比特并将所获得的二进制数值分配给当前二进制数。这种模式的优点是简单易懂,因此速度快。因此,其对于具有均匀统计分布,即等于“1”或“0”的概率相等,的二进制数而言通常是有效的。

[0048] 相反,如果没有以旁路模式对当前二进制数进行编解码,则意味着已经以所谓的常规模式对其进行编解码,即,通过基于上下文的算术编解码。在那种情况下,所考虑的二进制数的解码进行如下。首先,使用上下文建模器模块获得用于当前二进制数的解码的上下文。上下文的目标是在给定一些上下文或先验信息X的情况下获得当前二进制数值“0”的条件概率。这里的先验X可以是在当前二进制数正在被解码的时候在编码器侧和解码器侧都以同步方式可用的一些已经解码的语法元素的值。

[0049] 通常,用于对二进制数进行解码的先验X在标准中被规定,并且之所以被选择是因为其与要解码的当前二进制数统计相关。使用此上下文信息的好处在于其降低了对二进制数进行编解码的速率成本。这是基于这样一个事实,即当二进制数和X更相关时,二进制数(bin)给定X的条件熵更低。以下关系在信息理论中是众所周知的:$H(bin|X)<H(bin)$。

[0050] 这意味着如果二进制数和X是统计相关的,则二进制数获知X的条件熵低于二进制数的熵。因此,上下文信息X被用于获得二进制数为“0”或“1”的概率。给定这些条件概率,常规解码引擎执行二进制值二进制数的算术解码。然后在获知当前上下文信息X的情况下,二进制数的值被用于更新与当前二进制数相关联的条件概率的值。这被称为上下文模型更新。只要二进制数正在被解码(或被编解码)就更新每个二进制数的上下文模型,这使得对每个二进制元素的上下文建模逐步细化。因此,CABAC解码器逐步学习每个常规编码的二进制数的统计行为。应该注意的是,上下文建模器和上下文模型更新步骤在编码器侧和解码器侧是严格相同的操作。

[0051] 取决于当前二进制数是如何被编解码的,当前二进制数的常规算术解码或其旁路解码导致一系列解码的二进制数。

[0052] 然后,在图5的右侧上示出的CABAC解码的第二阶段将这一系列二进制码元转换为语法元素。语法元素可以采用标志的形式,在这种情况下,其直接采用当前解码的二进制数的值。另一方面,如果根据所考虑的标准规范,当前语法元素的二值化对应于几个二进制数的组合,则会发生二进制码字到语法元素的转换。

[0053] 这进行由编码器完成的二值化步骤的互逆操作。因此,这里执行的逆转换基于这些语法元素各自的解码的二值化的版本来获得这些语法元素的值。

[0054] 例如,使用截断莱斯二值化(truncated Rice binarization)来对与最后有效系数位置的前缀码和合并索引(指示候选在合并候选的列表中的位置)相对应的语法元素进行二值化。对于最后有效系数位置标志,以常规模式对全部二进制数进行编码;对于合并索引,对第一二进制数以常规模式进行编码,并且对其他二进制数以旁路模式进行编码。

[0055] 画面分割信息指示画面是如何被分割的,例如,CTU的尺寸,和CTU被划分为CU以及

在适用时可能被划分为PU的方式。因此,解码器可以根据解码的画面分割信息来将画面划分(235)为,例如,CTU,并且将每个CTU划分为CU。变换系数被去量化(240)和逆变换(250)以对预测残差进行解码。

[0056] 通过将预测块和解码的预测残差进行组合(255)以重构图像块。可以根据帧内预测(260)或运动补偿预测(即,帧间预测)(275)来获得(270)预测块。将环内滤波器(265)应用于重构图像。滤波的图像被存储在参考画面缓冲器(280)。

[0057] 解码的画面可以进一步经历后解码处理(285),例如,逆颜色变换(例如,从YCbCr 4:2:0到RGB 4:4:4的转换)或执行在预编码处理(101)中执行的重映射过程的逆过程的逆重映射。后解码处理可以使用在预编码处理中推导并在比特流中用信令通知的元数据。

[0058] 最近新兴的视频压缩工具被提出,包括压缩域中的新编解码树单元表示,以便在压缩域中以更灵活的方式来表示画面数据。通过更灵活的编解码树的表示,与HEVC标准的CU/PU/TU布置相比压缩效率也有所提高。

[0059] 在一个示例中,四叉树加二叉树(Quad-Tree plus Binary-Tree,QTBT)编解码工具是提供增强的灵活性的一种新工具。在QTBT编解码树中,可以以四叉树和二叉树两种方式对编解码单元进行划分。在QTBT技术中,CU具有正方形或矩形的形状。编解码单元的尺寸总是2的幂,并且通常从4到128。图7中示出了编解码树单元的QTBT编解码树表示的示例。

[0060] 在编码器侧通过速率失真优化过程来决定编解码单元的划分,其可以通过确定具有最小速率失真成本的CTU的QTBT表示来执行。CTU的QTBT分解由两个阶段组成:首先以四叉树方式对CTU进行划分,然后可以进一步地以二叉树方式对每个四叉树叶进行划分,如图7中所示,在图7中,实线表示四叉树分解阶段,并且虚线表示空间地嵌入四叉树叶中的二叉树分解。在条带中,亮度和色度块分割结构是分开的,并且是独立决定的。

[0061] 不再采用分割成预测单元或变换单元的CU。换句话说,每个编解码单元系统地由单个预测单元(2N×2N预测单元分割类型)和单个变换单元(没有到变换树的划分)组成。

[0062] VVC(Versatile Video Coding,通用视频编解码)视频压缩标准中采用的附加CU划分模式(称为水平或垂直三叉树划分模式(HOR_TRIPLE,VER_TRIPLE))包括将编解码单元(CU)划分为三个子编解码单元(子CU),其中相应的尺寸在所考虑的空间划分的方向上等于父CU尺寸的1/4、1/2和1/4,如图8中所示。

[0063] 本申请实施例针对语法元素的编码和解码。在一些实施例中,对一些帧内或帧间预测参数的熵编解码进行修改以降低复杂度。

[0064] 如上所述,通过上下文自适应二进制算术编解码过程对多个二进制码元(或二进制数)进行编解码和解码。这个过程涉及两种对二进制数进行编解码的方式:常规模式和旁路模式。旁路编解码模式远不如常规编解码模式复杂。因此,当与旁路编解码相比使用常规编解码模式没有或几乎没有获得编解码效率改进时,用旁路编解码过程代替常规编解码过程是有益的。

[0065] 在一个实施例中,使用旁路编解码过程来用信令通知表示广义双向预测索引的二进制数串的第一或前几个二进制数。在另一个实施例中,使用旁路编解码过程而不是常规编解码过程来用信令通知使用哪个运动矢量预测器对以AMVP模式编解码的编解码单元(CU)的运动矢量进行编码。

[0066] 实际上,尽管在当前的VVC编解码系统中对这些二进制数使用了基于上下文的算

术编解码,但已经检测到要在这些语法元素中编码的二进制数的条件熵接近信息的1比特。此外,实验结果表明,通过使用所提出的旁路编解码,对VVC压缩性能的影响可以忽略不计。

[0067] 在下文中,我们更详细地描述广义双向预测索引和运动矢量预测器的信令。

[0068] 广义双向预测索引的信令

[0069] VVC草案4中的广义双向预测

[0070] 在VVC草案4(参见“通用视频编解码(草案4)”,B.Bross等人,第13次JVET会议,2019年1月9日-18日,马拉喀什)中,可以通过使用所谓的广义双向预测(GBi)来对帧间CU进行时间预测。在广义双向预测中,根据以下表达式,双向预测块的时间预测被计算为两个参考块的加权平均值:

[0071] $P_{bipred}=((8-w)\times P_0+w\times P_1+4)>>3$

[0072] 其中P_0和P_1是时间预测器,并且w在以下集合中选择:

[0073] -对于低延迟画面(其全部参考画面都在过去):$w\in\{-2,3,4,5,10\}$。对于非低延迟画面(其具有至少有一张过去参考画面和一张未来参考画面):$w\in\{3,4,5\}$

[0074] -基于编码器侧的速率失真优化过程来选择GBi权重w,并在比特流中用信令通知GBi权重w。GBi还与VVC中的各种运动补偿工具相结合,如仿射运动补偿或自适应运动矢量分辨率。

[0075] 用信令通知用于预测CU的GBi权重的语法元素被称为GBi索引。

[0076] VVC草案4中GBi索引的编解码(编码器)

[0077] 在VVC草案4中,GBi索引首先被转换为另一个索引,gbiCodingIndex。确定标志以指示是否要使用默认权重(相等权重)。对于默认权重w=4(预测器P_0和P_1都是相等的权重),该标志被设置为1;而对于其他权重,该标志被设置为0。使用截断莱斯(truncated unary,截断一元)二进制数串对剩余的gbiCodingIndex进行二值化。在专用的CABAC上下文的情况下,对该标志或截断莱斯二进制数串中的每个二进制数进行CABAC编码。具体地,以上下文模型ID 0对该标志进行基于上下文的编解码(即,使用常规模式)。分别以上下文模型ID 4、5和6对二进制数(二进制数1(bin1)、二进制数2(bin2)和二进制数3(bin3)),进行基于上下文的编解码。

[0078] 表1示出了用于低延迟模式的GBi索引(GBiIdx)编解码,其中GBi模式的数量被设置为numGBiModes=5,除第一二进制数之外的二进制数的最大数量被设置为numBins=3。

[0079] 表1.低延迟模式

[0080]

w	GBi索引	gbiCodingIndex	标志	上下文	二进制数串	上下文	gbiParsingOrder
-2	0	4	0	[0]	000	[4, 5, 6]	0
3	1	2	0	[0]	01	[4]	1
4	2	0	1	[0]	---	---	2

[0081]

5	3	1	0	[0]	1	[4, 5]	3
10	4	3	0	[0]	001	[4, 5, 6]	4

[0082] 表2示出了用于非低延迟模式的GBi索引编解码,其中GBi模式的数量被设置为numGBiModes=3,除第一二进制数之外的二进制数的最大数量被设置为numBins=1。

[0083] 表2.非低延迟模式

[0084]

w	GBi索引	gbiCodingIndex	标志	上下文	二进制数串	上下文	gbiParsingOrder
3	1	2	0	[0]	0	[4]	1
4	2	0	1	[0]	---	---	2
5	3	1	0	[0]	1	[4]	3

[0085] 应该注意的是,如表1和表2所示的标志和二进制数串的关联可以直接被视为是gbiCodingIndex的截断莱斯二值化。即,使用截断莱斯(截断一元)二进制数串对gbiCodingIndex进行二值化。在专用的CABAC上下文的情况下,每个二进制数被CABAC编码。具体地,以上下文模型ID 0对第一二进制数(二进制数0(bin0))进行基于上下文的编解码(即使用常规模式)。需要注意的是,第一二进制数针对默认权重w=4(两个预测器P_0和P_1都是相等的权重)被设置为1,并且针对其他权重被设置为0。因此,也可以将第一二进制数视为指示是否要使用默认权重的标志。然后,分别以上下文模型ID 4、5和6对后续的二进制数(二进制数1(bin1)、二进制数2(bin2)和二进制数3(bin3))进行基于上下文的编解码。

[0086] 图9描绘了如VVC草案4中的GBi索引的编解码。在步骤910,将变量Idx设置为0。在步骤920,通过由表1和表2的列“gbiCodingIndex”定义的表GbiCodingOrder将与当前CU相关联的GBiIdx值转换为gbiCodingIndex。在步骤930,对标志(gbiCodingIndex==0)进行编码,其指示值gbiCodingIndex是否等于0。零值对应于当前CU的GBiIdx值等于GBI_DEFAULT的情况,其对应于默认双向预测模式,即w=4的情况。如果gbiCodingIndex的值为零(940),则过程结束。

[0087] 否则,编码器检查(945)条带是否为低延迟类型。如果条带是低延迟类型,则将numGbiModes设置(950)为5;如果条带不是低延迟类型,则将numGbiModes设置(955)为3。在步骤960,将Idx设置为1。在步骤965,将上下文模型ID contextId设置为4。在步骤970,将numBins设置为numGbiModes-2。将gbiCodingIndex二值化为二进制数串。然后以对应的contextId(985)对表示gbiCodingIndex的二进制数逐一进行编码(980),直到全部的二进制数都被编码(990)。该过程在步骤999结束。

[0088] 图10描绘了如在VVC草案4中的GBi索引的解析。解码器从比特流中对值“Idx”进行解码,使用链接解码的值Idx和CU级GBiIdx参数的映射表GbiParsingOrder将“Idx”转换为与输入CU相关联的实际GBiIdx。

[0089] 更具体地,在步骤1010,将变量Idx设置为0。在步骤1020,使用contextId=0对指示值gbiCodingIndex是否等于0(gbiCodingIdx==0)的标志进行解码。零值对应于当前CU的GBiIdx值等于GBI_DEFAULT的情况,其对应于默认双向预测模式,即w=4的情况。如果解码的码元为1,则解码器进行到步骤1090。

[0090] 否则,如果标志为1(1025),则在步骤1030,将Idx设置为1。在步骤1035,将上下文模型ID contextId设置为4。解码器检查(1040)条带是否是低延迟类型。如果条带是低延迟类型,则将numGbiModes设置(1045)为5;如果条带不是低延迟类型,则将numGbiModes设置

(1050)为3。在步骤1060,将numBins设置为numGbiModes-2。然后以对应的contextId(1080)对表示gbiCodingIndex的二进制数迭代地进行逐个解码(1070),直到找到等于1的二进制数码元或者已经解析过numBins(1085)。在每次迭代中,变量Idx都会递增。在步骤1090,通过表GbiParsingOrder将“Idx”转换为GBi索引。该过程在步骤1099结束。

[0091] 提出的GBi索引编解码过程(编码器)

[0092] 表3和表4分别示出了所提出的对低延迟模式和非低延迟模式的GBi索引(GBiIdx)编解码的修改。在该提出的方法中,以旁路模式对第一二进制数进行编码(在表中表示为“b”),并且以常规模式对其他二进制数进行编码。

[0093] 表3.低延迟模式

[0094]

w	GBi索引	gbiCodingIndex	标志	上下文	二进制数串	上下文	gbiParsingOrder
-2	0	4	0	[0]	000	[b, 5, 6]	0
3	1	2	0	[0]	01	[b, 5]	1
4	2	0	1	[0]	---	---	2

[0095]

5	3	1	0	[0]	1	[b, 5]	3
10	4	3	0	[0]	001	[b, 5, 6]	4

[0096] 表4.非低延迟模式

[0097]

w	GBi索引	gbiCodingIndex	标志	上下文	二进制数串	上下文	gbiParsingOrder
3	1	2	0	[0]	0	[b]	1
4	2	0	1	[0]	---	---	2
5	3	1	0	[0]	1	[b]	3

[0098] 图11示出了根据实施例的提出的GBiIdx参数编解码过程。在步骤1165,将contextId设置为5。在步骤1170,以旁路模式而不是基于上下文的算术编解码模式对表3或4的截断莱斯二进制数串的第一二进制数进行编码。以常规模式对其他二进制数,如果有的话,进行编码,与图9中描述的类似。该提出的实施例的优点是降低了编码CU级GBiIdx参数的复杂度。

[0099] 提出的GBi解析过程(解码器)

[0100] 图12示出了根据实施例的提出的对GBiIdx参数解析过程的修改。可以看出,所提出的GBiIdx解析过程的第一步与图10中描述的相同。接下来,在第一解码的码元等于0(非GBI_DEFAULT基础)的情况下,然后对表示截断莱斯二值化串的一系列二进制数进行解析。根据所提出的实施例,使用旁路模式而不是基于上下文的算术解码模式对该串的第一二进制数进行解析。对其他二进制数采用与现有技术相同的方式进行解码。该提出的实施例的

优点是降低了解析GBiIdx CU级参数的复杂度。

[0101] 在上文中,当索引由多个二进制数表示时,以旁路模式对第一二进制数进行编解码,以常规模式对其余的二进制数进行编码。更一般地,可以以旁路模式对多于一个二进制数(在二进制数串的开始处)进行编码,而以常规模式对其余二进制数进行编码。而且,上述例子中描述了截断莱斯二值化,应该注意的是,所提出的编解码方法可以应用于其他二值化方案。另外,GBi索引在上文中被用作语法元素的示例。然而,如上所述的方法可以应用于其他语法元素。

[0102] 需要注意的是,在对一些其他语法元素进行编码时,旁路模式和常规模式都被使用,例如,在对如前所述的合并索引进行编码时。特别地,为了对合并索引进行编码,以常规模式对第一二进制数进行编码,这可能是因为使用上下文信息对其进行编码更有效。对于其余的二进制数,概率可能会更加随机分布,并且对应于等概率的旁路模式也可能同样有效。应当注意的是,这样的编码与在本实施例中所提出的相反,在本实施例中,以旁路模式对第一二进制数进行编码而以常规模式对其余的二进制数进行编码。

[0103] 根据另一实施例,仅以常规模式对第一标志进行编码,该标志指示默认GBi索引(GBI_DEFAULT)是否对应于加权因子w=4。在该实施例中,对在该标志之后的截断莱斯二进制数串的全部二进制数都以旁路模式进行编码。该实施例的优点是进一步降低了GBi索引的编解码和解析的复杂度,可能以稍微降低编解码效率为代价。图11A示出了根据该实施例的提出的gbiCodingIndex编解码过程。在步骤1180,以旁路模式而不是基于上下文的算术编解码模式对表3或4的截断莱斯二进制数串的二进制数进行连续编码。该提出的实施例的优点是进一步降低了编码CU级GBiIdx参数的复杂度,而在视频压缩效率方面没有任何损失。

[0104] 图12A示出了根据该实施例的提出的对gbiCodingIndex解析过程的修改。可以看出,所提出的gbiCodingIndex解析过程的第一步与图10中描述的相同。接下来,在第一解码的码元等于0的情况下(非GBI_DEFAULT情况),然后对表示截断莱斯二值化串的一系列二进制数进行解析。根据所提出的实施例,使用旁路模式(1280)而不是基于上下文的算术解码模式来对该串的二进制数进行解析。

[0105] 表4A描绘了与使用图9的gbiCodingIndex编解码方法的VTM-4.0的性能相比,结合图10的gbiCodingIndex解析方法,当使用该提出的用于gbicodingIndex编解码的实施例(如图11A、图12A中所述)时的VTM-4.0的压缩性能。在相同的目标视频质量下,表中提供的数字对应于所提出的方法的平均比特率降低。因而,负数表示比特率降低,因此编解码效率提高;而正数表示比特率增加,因此编解码效率降低。Y、U和V列分别对应于亮度、色度Cb和色度Cr分量。可以看出,与图9的方法相比,该实施例在亮度方面没有导致平均比特率修改,因此尽管由于使用了更多的旁路编解码的二进制数而降低了复杂度,也不会引入视频压缩效率的任何损失。

[0106] 表4A所提出的用于GBi索引编解码和解析的实施例(图11A、图12A)的压缩性能

[0107]

随机访问主要 10			
	Y	U	V
类别 A1	-0.01%	-0.03%	0.01%
类别 A2	-0.01%	-0.02%	0.06%
类别 B	0.01%	-0.02%	0.02%
类别 C	0.00%	-0.05%	0.00%
类别 E			
总计	0.00%	-0.03%	0.02%
类别 D	0.01%	-0.04%	-0.06%
类别 F (可选)	0.00%	-0.02%	-0.02%

低延迟 B 主要 10			
	Y	U	V
类别 A1			
类别 A2			
类别 B	0.01%	-0.14%	0.05%
类别 C	-0.03%	0.23%	0.05%
类别 E	0.01%	0.30%	-0.33%
总计	0.00%	0.09%	-0.05%
类别 D	0.00%	0.11%	-0.31%
类别 F (可选)	-0.03%	0.07%	0.14%

[0108] 在下午中,对为什么所提出的gbiCodingIndex编解码和解析方法在编解码效率方面没有损失进行说明,即,对来自截断莱斯(Truncated Rice,TR)二进制数串的二进制数的旁路编解码恰好与这些二进制数的算术编解码同样有效。

[0109] 众所周知,算术编解码实现了非常接近香农(Shannon)限制的比特率,即,其编码的码元的熵,因而是一种最优或接近最优的熵编解码方法。

[0110] 该过程的旁路编解码部分的最优性说明用于对gbiCodingIndex进行编解码的二值化与霍夫曼(Huffman)树密切对应。这意味着用于以旁路模式对gbiCodingIndex的二进制数进行编解码的TR二值化对应于霍夫曼树。霍夫曼编解码是一种最优的可变长度编解码方法。此外,众所周知,当满足某些条件时,霍夫曼编解码会生成等于用信令通知的二进制数的熵的平均编解码长度。特别地,如果与霍夫曼树的每个分支相关联的概率是二元的,即等于2的负幂:$1/2^n$,其中n是正整数值,则霍夫曼编解码是最优的。

[0111] 图12B和图12C中的两个树示出了gbiCodingIndex二值化和编解码/解析过程,其分别以非低延迟模式和以低延迟画面间编解码配置由所提出的方案产生。填充节点对应于常规(基于上下文的)编解码的二进制数,而非填充节点对应于旁路编解码的二进制数。假设x是基于上下文编解码的二进制数等于0的概率。与图12B和图12C的树的每个边相关联的值对应于二值化的gbiCodingIndex的二进制数等于1或0的概率。例如,二值化的gbiCodingIndex的第二二进制数等于0的概率为$x \times \frac{1}{2}$,且该二进制数等于1的概率也是$x \times \frac{1}{2}$。图12D图示了与二值化的gbiCodingIndex的旁路编解码的二进制数的编解码/解析

相对应的树,已知用信令通知默认GBI模式的第一二进制数等于0。如上所述,这些二进制数的旁路编解码的最优性指示图12D的树是最优的霍夫曼编解码树,因此具有二元关联的概率值。

[0112] 由语法元素的截断莱斯(TR)二值化得到的二进制数串的概率并不总是二元的。例如,在VVC规范中存在一些其他的语法元素,其在截断莱斯二值化之后被二值化,并且其中全部的二进制数都是基于上下文编解码的,诸如语法元素last_sig_coeff_x_prefix和last_sig_coeff_y_prefix。这两个参数被截断莱斯二值化并且全部的二进制数都是基于上下文编解码的。对其中的一部分进行旁路编解码会导致压缩效率的损失,这表明在旁路编解码的二进制数的二元概率分布的情况下,对应的旁路编解码不会对应于最优的霍夫曼编解码过程。

[0113] 在该实施例中,可以认识到的是,TR二值化之后的gbiCodingIndex的二进制数串的概率接近二元,并且因此应用旁路编解码过程会对应于最优霍夫曼编解码过程,同时产生比常规模式中的算术编解码更低的计算复杂度。

[0114] 运动矢量预测器的信令

[0115] VVC草案4中AMVP模式下的运动矢量预测编解码

[0116] AMVP运动矢量编解码模式涉及使用以下元素对CU的运动矢量进行编解码:

[0117] -帧间预测方向,其指示是使用双向预测还是单向预测来预测当前CU,以及在单向预测的情况下使用哪个参考画面列表。

[0118] -在每个涉及的参考画面列表中的、指示使用哪个(些)参考画面来预测当前CU的一个或多个参考画面索引。

[0119] -对于用于预测当前CU的每个参考画面,使用运动矢量预测器来预测当前CU的实际运动矢量。该MV预测器(或MVP,或AMVP候选)由编码器在包含两个候选的MV预测器列表中选取。对于每个关注的参考画面列表L0和L1,通过标志来实现用信令通知哪个MV候选被选择,该标志分别被标记为mvp_l0_flag和mvp_l1_flag。

[0120] -对于每个参考画面列表L0和L1,当前CU的实际运动矢量与其各自的运动矢量预测器之间的运动矢量差。

[0121] 图13和图14中示出了AMVP运动矢量预测候选列表的构建。该过程基本上包括在当前CU周围的五个空间位置中选择最多两个空间候选,然后精简它们并保留其中最多两个。接下来,在当前条带的所谓同位条带中,在对应于右下位置H的空间位置,或者,如果不可用,在中心位置“中心”,寻找时间MV预测候选。接着,在空间和时间候选之间应用精简过程,并且用至多两个元素的零运动矢量来填充列表。最后,AMVP候选列表正好包含两个运动矢量预测候选。

[0122] 因此,分别被标记为mvp_l0_flag和mvp_l1_flag的单个标志在比特流中用信令通知,并在解码器侧被解析,以便指示AMVP列表中包含的两个元素中的哪个AMVP候选被用于预测每个参考画面列表L0和L1中的当前CU的运动矢量。

[0123] 图15示出了取决于所考虑的参考画面列表L0或L1对标志mvp_l0_flag或mvp_l1_flag进行解析的过程。它包括二进制码元“码元”的基于上下文的算术解码(1510)。这采用了单个CABAC上下文。所考虑的参考画面列表中的当前PU或CU的MV预测器索引被赋值(1520)为解码的码元的值(PU可以是当前CU内的运动分割,如在HEVC中使用的;在VVC草案4

中没有使用PU分割,所以PU对应于CU)。

[0124] 提出的方法:以旁路模式对mvp_l0_flag和mvp_l1_flag进行编解码

[0125] 已经测量到的是,在多个编解码的视频序列上,用于使CABAC用信令通知mvp_l0_flag和mvp_l1_flag语法元素的平均熵非常接近信息的一个比特,这意味着这些标志的CABAC编解码相比于简单的旁路编解码过程没有带来任何好处。

[0126] 在本实施例中,因此提出通过旁路编解码模式对这些标志进行编码和解析。图16示出了提出的针对一个给定参考画面列表对一个CU或PU的MV预测器索引进行解码(1610、1620)的解析过程。与图15的过程的区别在于所关注的二进制数的CABAC解码被该二进制数的旁路解码(1610)所替代。在编码器侧,以旁路模式对表示mvp_l0_flag或mvp_l1_flag的二进制码元进行编码。本实施例的优点是降低了VVC的编解码和解析过程的复杂度,以及对编解码效率的可以忽略不计的影响。

[0127] 根据另一的实施例,可以根据所考虑的CU是否以SMVD(Symmetrical Motion Vector Difference,对称运动矢量差)模式被编解码,来以不同方式对mvp_l0_flag进行编解码。VVC的SMVD运动矢量编解码过程包括对给定CU相对于第一参考画面列表L0的运动矢量差进行编解码。然后从L0运动矢量差推导出另一个参考画面列表(L1)中的考虑的CU的运动矢量差。实际上,在这种模式下,两个运动矢量是对称的。L1运动矢量的两个分量均等于L0运动矢量的相反值。在该实施例中,SMVD情况下的MVP候选可以被以常规模式编码,而传统AMVP运动矢量编解码模式中的MVP候选可以被以旁路模式编解码。这意味着可以指定两个不同的语法元素,经典的mvp_l0_flag和smvd_mvp_l0 flag,后者被用于指定用于以SMVD模式预测CU的运动矢量的MV。

[0128] 根据另一实施例,经典的mvp_l0_flag可以被以常规模式编解码,而smvd_mvp_l0 flag可以被以旁路模式编解码。根据一种变体,经典的mvp_l0_flag可以被以常规模式编解码,并且smvd_mvp_l0 flag也可以被以常规模式编解码,但不同于CABAC上下文的上下文被用于对通用mvp_l0_flag进行编解码。

[0129] 使用VVC草案4语法作为示例,表5列出了以上提出的方法对语法进行修改的示例。具体地,语法元素mvp_l0_flag和mvp_l1_flag的描述符从ae(v)变为u(1),其中ae(v)指示上下文自适应算术熵编解码的语法元素,并且u(n)指示使用n个比特的无符号整数。

[0130] 表5

[0131]

coding_unit(x0, y0, cbWidth, cbHeight, treeType) {	描述符
…	
} else if(treeType != DUAL_TREE_CHROMA) { /* MODE_INTER or MODE_IBC */	
if(cu_skip_flag[x0][y0] = = 0)	
merge_flag[x0][y0]	ae(v)
if(merge_flag[x0][y0]) {	

[0132]

merge_data(x0, y0, cbWidth, cbHeight)	
} else if (CuPredMode[x0][y0] == MODE_IBC) {	
mvd_coding(x0, y0, 0, 0)	
~~**mvp_l0_flag**[x0][y0]~~	~~ae(v)~~
mvp_l0_flag[x0][y0]	u(1)
if(sps_amvr_enabled_flag && (MvdL0[x0][y0][0] != 0 || MvdL0[x0][y0][1] != 0)) {	
amvr_precision_flag[x0][y0]	ae(v)
}	
} else {	
if(tile_group_type == B)	
inter_pred_idc[x0][y0]	ae(v)
if(sps_affine_enabled_flag && cbWidth >= 16 && cbHeight >= 16) {	
inter_affine_flag[x0][y0]	ae(v)
if(sps_affine_type_flag && inter_affine_flag[x0][y0])	
cu_affine_type_flag[x0][y0]	ae(v)
}	
if(inter_pred_idc[x0][y0] == PRED_BI && !inter_affine_flag[x0][y0] && RefIdxSymL0 > −1 && RefIdxSymL1 > −1)	
sym_mvd_flag[x0][y0]	ae(v)
if(inter_pred_idc[x0][y0] != PRED_L1) {	
if(NumRefIdxActive[0] > 1 && !sym_mvd_flag[x0][y0])	
ref_idx_l0[x0][y0]	ae(v)
mvd_coding(x0, y0, 0, 0)	
if(MotionModelIdc[x0][y0] > 0)	

[0133]

mvd_coding(x0, y0, 0, 1)	
if(MotionModelIdc[x0][y0] > 1)	
mvd_coding(x0, y0, 0, 2)	
~~**mvp_l0_flag**[x0][y0]~~	~~ae(v)~~
mvp_l0_flag[x0][y0]	u(1)
} else {	
MvdL0[x0][y0][0] = 0	
MvdL0[x0][y0][1] = 0	
}	
if(inter_pred_idc[x0][y0] != PRED_L0) {	
if(NumRefIdxActive[1] > 1 && !sym_mvd_flag[x0][y0])	
ref_idx_l1[x0][y0]	ae(v)
if(mvd_l1_zero_flag && inter_pred_idc[x0][y0] == PRED_BI) {	
MvdL1[x0][y0][0] = 0	
MvdL1[x0][y0][1] = 0	
MvdCpL1[x0][y0][0][0] = 0	
MvdCpL1[x0][y0][0][1] = 0	
MvdCpL1[x0][y0][1][0] = 0	
MvdCpL1[x0][y0][1][1] = 0	
MvdCpL1[x0][y0][2][0] = 0	
MvdCpL1[x0][y0][2][1] = 0	
} else {	
if(sym_mvd_flag[x0][y0]) {	
MvdL1[x0][y0][0] = −MvdL0[x0][y0][0]	
MvdL1[x0][y0][1] = −MvdL0[x0][y0][1]	
} else	

[0134]

mvd_coding(x0, y0, 1, 0)	
}	
if(MotionModelIdc[x0][y0] > 0)	
mvd_coding(x0, y0, 1, 1)	
if(MotionModelIdc[x0][y0] > 1)	
mvd_coding(x0, y0, 1, 2)	
~~**mvp_l1_flag**[x0][y0]~~	~~ae(v)~~
mvp_l1_flag[x0][y0]	u(1)
} else {	
MvdL1[x0][y0][0] = 0	
MvdL1[x0][y0][1] = 0	
}	
…	

[0135] 表6示出了使用上面提出的复杂度降低方面所获得的性能结果。可以看出,根据所提出的简化,压缩效率几乎没有变化。

[0136] 表6

[0137]

	随机访问主要 10 VTM-2.0.1 以上		
	Y	U	V
类别 A1	0.01%	0.01%	-0.07%
类别 A2	0.00%	0.02%	0.02%
类别 B	0.02%	0.06%	0.06%
类别 C	0.00%	0.01%	0.05%
类别 E			
总计	0.01%	0.03%	0.02%
类别 D	0.02%	0.09%	-0.02%
类别 F(可选)	0.04%	0.10%	0.19%

[0138] 上文中,参考VVC草案4描述了示例。在下文中,参考VVC草案5(见“通用视频编解码

(草案6)”,B.Bross等人,第14次JVET会议,2019年3月19日至27日,日内瓦,瑞士)来描述几个示例,包括子块间变换的信令、子分割内(ISP)编解码模式、多参考行帧内预测、和SMVD(对称运动矢量差)帧间工具。

[0139] 子块变换的信令

[0140] VVC草案5中的子块变换(Sub-block transform,SBT)

[0141] 对于用信令通知为非零残差块的帧间预测的CU,SBT工具以二叉方式将CU划分为2个变换单元(TU)。两个结果TU中的一个具有非零残差,并且另一个只有零残差数据。应用的二叉划分可以是对称的或不对称的。在对称划分的情况下,两个结果TU具有相同的尺寸,其在划分方向上为CU的尺寸的一半。在不对称二叉划分的情况下,一个TU具有等于沿分割方向的父CU的1/4的尺寸,并且另一个TU尺寸是沿分割方向的该CU的尺寸的3/4。

[0142] 除了空间划分之外,具有非零残差的TU使用推断的自适应变换进行编解码。所使用的一维变换取决于非零残差TU的位置,如图17中所示,其中部分“A”是具有非零残差数据的TU,并且另一个TU只有零残差数据。

[0143] 通过3个标志用信令通知所考虑的编解码单元的TU划分。首先,cu_sbt_flag指示对所考虑的CU使用SBT。接下来,在使用SBT的情况下,用信令通知SBT类型和SBT位置信息。这采用以下三个编解码的标志的形式:

[0144] -cu_sbt_quad_flag指示使用不对称二叉划分。如果允许对当前CU采用对称和不对称划分,则对其进行编解码。

[0145] -cu_sbt_horizontal_flag指示二叉划分的方向。如果允许对当前CU采用水平和竖直划分,以及对于先前用信令通知的SBT划分类型(是或者不是不对称),则对其进行编解码。

[0146] -cu_sbt_pos_flag指示用于对所考虑的CU的纹理数据进行编解码的非零残差TU的位置。

[0147] 在VVC草案5中,上述四个标志是基于上下文被编解码的。表7示出了VVC规范对应于这方面的部分。

[0148] 表7.VVC草案5中CABAC上下文到具有上下文编解码的二进制数的语法元素的分配(文档JVET-N1001的截断表9-14)

[0149]

语法元素	binIdx					
	0	1	2	3	4	>=5
…	…	…	…	…	…	…
cu_sbt_flag	(cbWidth * cbHeight < 256) ? 1 : 0	na	na	na	na	na
cu_sbt_quad_flag	0	na	na	na	na	na
cu_sbt_horizontal_flag	(cbWidth = = cbHeight) ? 0 : (cbWidth < cbHeight) ? 1 : 2	na	na	na	na	na
cu_sbt_pos_flag	0	na	na	na	na	na
…	…	…	…	…	…	…

[0150] 图18示出了如VVC草案5中指定的SBT模式的解码过程。在步骤1810,将用于对cu_sbt_flag进行解码的上下文id获得为ctxId＝(width*height)<＝256?1:0。然后在步骤1820,使用上下文ctxId对二进制码元cu_sbt_flag进行CABAC解码。如果cu_sbt_flag等于0((1830),则不使用子块变换工具。否则,如果cu_sbt_flag不为0,则在步骤1840使用上下文ctxId＝0对二进制码元cu_sbt_quad_flag进行CABAC解码。在步骤1850,解码器检查是否允许对当前CU进行竖直和水平划分。如果是,则在步骤1860用于对cu_sbt_horizontal_flag进行解码的上下文id被获得为ctxId＝(cuWidth＝＝cuHeight)?0:(cuWidth<cuHeight?1:2)。然后在步骤1870,使用上下文ctxId对二进制码元cu_sbt_horizontal_flag进行CABAC解码。在步骤1880,使用上下文ctxId＝0对二进制码元cu_sbt_pos_flag进行CABAC解码。该过程在步骤1899结束。

[0151] SBT模式的简化的编解码

[0152] 根据所提出的实施例,“cu_sbt_pos_flag”被以旁路模式而不是常规的(基于上下文的)CABAC模式编解码。实际上,“cu_sbt_pos_flag”的这种简化的编解码在简化了熵编解码的同时对整体编解码器编解码效率的影响可以忽略不计。

[0153] 根据另一实施例,“cu_sbt_quad_flag”被以旁路模式而不是常规模式编解码。这种简化对编解码器性能的影响也可以忽略不计。

[0154] 根据另一实施例,“cu_sbt_pos_flag”和“cu_sbt_quad_flag”都被以旁路模式而不是常规模式编解码。

[0155] 根据另一实施例,“cu_sbt_horizontal_flag”被以旁路模式而不是常规模式编解码。这种简化对编解码器性能的影响也可以忽略不计。

[0156] 根据另一实施例,三个标志:“cu_sbt_pos_flag”、“cu_sbt_quad_flag”和“cu_sbt_horizontal_flag”均被以旁路模式而不是常规模式编解码。如表8中的模拟结果所示,这种整体修改几乎不会导致编解码效率的损失。

[0157] 表8.VTM-5.0上的编解码效率(VVC草案5参考软件)

[0158]

	随机访问主要 10				
	VTM-2.0.1 以上				
	Y	U	V	EncT	DecT
类别 A1	0.02%	-0.17%	0.09%	97%	102%
类别 A2	0.01%	-0.02%	0.07%	99%	104%
类别 B	0.02%	0.01%	-0.01%	94%	101%
类别 C	0.01%	-0.17%	-0.13%	94%	94%
类别 E					
总计	0.01%	-0.08%	-0.01%	96%	100%
类别 D	0.02%	-0.12%	0.24%	97%	91%
类别 F(可选)	0.03%	0.02%	0.01%	98%	97%

[0159] 图19A示出了根据其中“cu_sbt_pos_flag”和“cu_sbt_quad_flag”都被以旁路模式编解码(1940,1980)的实施例的解码过程。图19B示出了根据其中三个标志:“cu_sbt_pos_flag”和“cu_sbt_quad_flag”和“cu_sbt_horizontal_flag”都被以旁路模式编解码的实施例的解码过程。

[0160] 表9.对VVC草案5的修改

[0161]

语法元素	binIdx					
	0	1	2	3	4	>= 5
…	…	…	…	…	…	…
cu_sbt_flag	(cbWidth * cbHeight < 256) ? 1 : 0	na	na	na	na	na
cu_sbt_quad_flag	旁路	na	na	na	na	na
cu_sbt_horizontal_flag	(cbWidth = = cbHeight) ? 0 : (cbWidth < cbHeight) ? 1 : 2	na	na	na	na	na

[0162]

语法元素	binIdx					
	0	1	2	3	4	>= 5
cu_sbt_pos_flag	旁路	na	na	na	na	na
…	…	…	…	…	…	…

[0163] VVC草案5的标准语法规范可以如表10中所示的被修改。特别地,语法元素cu_sbt_quad_flag和cu_sbt_pos_flag的描述符从ae(v)变为u(1)。

[0164] 表10.对VVC草案5中指定的编解码单元语法的修改

[0165]

coding_unit(x0, y0, cbWidth, cbHeight, treeType) {	描述符
…	
if(cu_cbf) {	
if(CuPredMode[x0][y0] == MODE_INTER && sps_sbt_enabled_flag && !ciip_flag[x0][y0] && !MergeTriangleFlag[x0][y0]) {	
if(cbWidth <= MaxSbtSize && cbHeight <= MaxSbtSize) {	
allowSbtVerH = cbWidth >= 8	
allowSbtVerQ = cbWidth >= 16	
allowSbtHorH = cbHeight >= 8	
allowSbtHorQ = cbHeight >= 16	
if(allowSbtVerH || allowSbtHorH || allowSbtVerQ || allowSbtHorQ)	
cu_sbt_flag	ae(v)
}	
if(cu_sbt_flag) {	
if((allowSbtVerH || allowSbtHorH) && (allowSbtVerQ || allowSbtHorQ))	
~~cu_sbt_quad_flag~~	~~ae(v)~~
cu_sbt_quad_flag	u(1)

[0166]

if((cu_sbt_quad_flag && allowSbtVerQ && allowSbtHorQ) || (!cu_sbt_quad_flag && allowSbtVerH && allowSbtHorH))	
cu_sbt_horizontal_flag	ae(v)
~~**cu_sbt_pos_flag**~~	~~ae(v)~~
cu_sbt_pos_flag	u(1)
}	
}	
transform_tree(x0, y0, cbWidth, cbHeight, treeType)	
}	
}	
}	

[0167] 帧内子分割编解码模式的信令

[0168] VVC草案5中的帧内子分割

[0169] VVC草案5中指定的帧内子分割(ISP)编解码模式可以将INTRA CU水平或垂直地划分为2个或4个子分割。该划分取决于块尺寸,如表11中所示。基本上,4×4的CU不能进一步被划分。尺寸为4×8或8×4的CU被划分为2个Tus(即子分割)。其他CU被划分为4个Tus。

[0170] 表11.根据CU尺寸的ISP子分割的数量

[0171]

块尺寸	子分割数量
4×4	不划分
4×8和8×4	2
所有其他情况	4

[0172] 图20A示出了针对4×8和8×4的INTRA编解码单元的允许的ISP划分,并且图20B示出了针对具有不同于4×8或8×4的尺寸的INTRA编解码单元的允许的ISP划分。在以ISP模式进行编解码的CU内,TU被顺序地解码,并使用相同的帧内预测模式从TU到TU进行帧内预测,所述帧内预测在CU级用信令通知。最后,也根据帧内子分割尺寸对残差编解码进行调整。实际上,子分割的尺寸可以为:1×N、N×1、2×N或N×2,在这些相应情况下使用尺寸为:1×16、1×1、2×16或16×2的编解码组。

[0173] 在VVC草案5中,通过两个连续的标志来用信令通知ISP编解码模式。

[0174] -intra_subpartitions_mode_flag指示对给定的帧内CU使用ISP模式。

[0175] -intra_subpartitions_split_flag指示帧内子分区的分割方向。

[0176] 上述两个标志用于对与所考虑的CU相关联的“IntraSubPartitionsSplitType”的值进行解码,如下所示:

[0177] IntraSubPartitionsSplitType＝intra_subpartitions_mode_flag＝＝0?0:(1+intra_subpartitions_split_flag)

[0178] “IntraSubPartitionsSplitType”值的含义如表12所示。

[0179] 表12.如VVC草案5中所指定的IntraSubPartitionsSplitType的含义

[0180]

IntraSubPartitionsSplitType	IntraSubPartitionsSplitType的名称
0	ISP_NO_SPLIT
1	ISP_HOR_SPLIT
2	ISP_VER_SPLIT

[0181] 如表13中所指示的,根据VVC草案5,对这两个标志进行基于上下文的编解码。

[0182] 表13.来自VVC草案5的截断表,指示用于ISP相关语法元素的熵编解码模式

[0183]

语法元素	binIdx					
	0	1	2	3	4	>= 5
…	…	…	…	…	…	…
intra_subpartitions_mode_flag	0	na	na	na	na	na
intra_subpartitions_split_flag	0	na	na	na	na	na
…	…	…	…	…	…	…

[0184] ISP模式的简化的编解码

[0185] 在一个实施例中,要以旁路模式对二进制数“intra_subpartitions_split_flag”,即,用于对CU的ISP模式进行编解码的第二标志,进行编解码。表14和表15示出了对与该实施例相关联的VVC草案5规范的修改。特别地,语法元素“intra_subpartitions_split_flag”的描述符从ae(v)变为u(1)。

[0186] 表14.对指定对语法元素的CABAC上下文分配的VVC草案5表进行修改

[0187]

语法元素	binIdx					
	0	1	2	3	4	>= 5
…	…	…	…	…	…	…
intra_subpartitions_mode_flag	0	na	na	na	na	na
intra_subpartitions_split_flag	旁路	na	na	na	na	na
…	…	…	…	…	…	…

[0188] 表15.对VVC草案5中的编解码单元语法的修改

[0189]

coding_unit(x0, y0, cbWidth, cbHeight, treeType) {	描述符
…	
if(CuPredMode[x0][y0] == MODE_INTRA) {	
if(sps_pcm_enabled_flag && cbWidth >= MinIpcmCbSizeY && cbWidth <= MaxIpcmCbSizeY && cbHeight >= MinIpcmCbSizeY && cbHeight <= MaxIpcmCbSizeY)	
pcm_flag[x0][y0]	ae(v)
if(pcm_flag[x0][y0]) {	
while(!byte_aligned( ))	
pcm_alignment_zero_bit	f(1)
pcm_sample(cbWidth, cbHeight, treeType)	
} else {	
if(treeType == SINGLE_TREE || treeType == DUAL_TREE_LUMA) {	
if((y0 % CtbSizeY) > 0)	
intra_luma_ref_idx[x0][y0]	ae(v)

[0190]

if (intra_luma_ref_idx[x0][y0] = = 0 && (cbWidth <= MaxTbSizeY || cbHeight <= MaxTbSizeY) && (cbWidth * cbHeight > MinTbSizeY * MinTbSizeY))	
intra_subpartitions_mode_flag[x0][y0]	ae(v)
if(intra_subpartitions_mode_flag[x0][y0] = = 1 && cbWidth <= MaxTbSizeY && cbHeight <= MaxTbSizeY)	
~~**intra_subpartitions_split_flag**[x0][y0]~~	~~ae(v)~~
intra_subpartitions_split_flag[x0][y0]	u(1)
if(intra_luma_ref_idx[x0][y0] = = 0 && intra_subpartitions_mode_flag[x0][y0] = = 0)	
intra_luma_mpm_flag[x0][y0]	ae(v)
if(intra_luma_mpm_flag[x0][y0])	
intra_luma_mpm_idx[x0][y0]	ae(v)
else	
intra_luma_mpm_remainder[x0][y0]	ae(v)
}	
if(treeType = = SINGLE_TREE || treeType = = DUAL_TREE_CHROMA)	
intra_chroma_pred_mode[x0][y0]	ae(v)
}	
…	

[0191] 多参考行帧内预测的信令

[0192] VVC草案5中的多参考行帧内预测

[0193] 在VVC草案5中的INTRA CU中使用的多参考行帧内预测基于重建的参考样本执行亮度块的角度帧内预测,这些重建的参考样本属于从3个参考行中选出的当前CU的上方和左侧的一个参考行和列。用于帧内预测的参考行在比特流中通过语法元素“intra_luma_ref_idx”用信令通知。每个参考行都由其索引标识,如图21所示。VVC草案5中实际使用的参考是图21中所示的行0、行1和行3。该语法元素在帧内预测模式之前被用信令通知。在参考行不同于行0的情况下,即,该行最接近该CU。

[0194] “intra_luma_ref_idx”被编解码如下。其被二值化为截断莱斯二进制数串。这意

味着由一系列等于1的常规CABAC二进制数进行编解码,由等于0的常规CABAC二进制数终止。总的来说,最多用信令通知3个二进制数。

[0195] 图22描绘了根据VVC草案5的“intra_luma_ref_idx”的解码过程。在图22中,使用了涉及多达4个参考行的通用过程。数组lineIDx[.]由4个行参考索引组成,并且值“MaxNumRefLInes”表示用于帧内预测的允许的参考行的最大数量。对于VVC,“MaxNumRefLInes”等于3,并且数组lineIdx由以下元素组成:

[0196] lineIdx＝{0,1,3}。

[0197] 语法元素“intra_luma_ref_idx”的VVC解码过程如下进行。该过程的输出是解码的multiRefIdx。特别地,在步骤2210,它检查对当前CU是否允许多个参考行。如果是,则输出值multiRefIdx被初始化(2220)为0。如果MaxNumRefLines不高于(2225)1,则该过程结束。否则,在步骤2230,用单个CABAC上下文、以索引0来解析CABAC常规二进制数。如果其等于0,则multiRefIdx值不变并且该过程结束。否则,multiRefIdx被设置为等于lineIdx[1]。

[0198] 在步骤2255,如果MaxNumRefLines严格高于(2250)2,则使用标识符为1的单个上下文来对第二常规CABAC二进制数进行解码。如果解码的二进制数为0,则multiRefIdx不变并且该过程结束。否则,mutiRefIdx被设置(2260)为等于lineIdx[2]。如果MaxNumRefLines严格高于(2270)3,则使用标识符为2的单个上下文对第三常规CABAC二进制数进行解码(2280)。如果解码的二进制数为0,则multiRefIdx不变并且该过程结束(2299)。否则,mutiRefIdx被设置(2290)为等于lineIdx[3]。应该注意的是,在实践中,根据VVC草案5规范,此步骤不会发生,因为在为VVC草案5选择的设计中,如上所述最多可以使用3个参考行。因此,条件“MaxNumRefLines>3”在VVC草案5的范围内始终为假。

[0199] 如表16中已经说明和示出的,在VVC草案5中使用两个常规编解码二进制数来用信令通知intra_luma_ref_idx语法元素,每个二进制数使用单个CABAC上下文。

[0200] 表16.如VVC草案5中,CABAC上下文分配给“intra_luma_ref_idx”的基于上下文的编解码的二进制数

[0201]

语法元素	binIdx					
	0	1	2	3	4	>= 5
...	...	...	...	...	...	...
intra_luma_ref_idx[ ][ ]	0	1	na	na	na	na
...	...	...	...	...	...	...

[0202] 多参考行索引的简化的编解码

[0203] 在一个实施例中,“intra_luma_ref_idx”语法元素的编解码被简化,并且仅该语法元素的第一二进制数被以常规模式编解码。图23中描绘了针对该语法元素提出的修改的解析过程。特别地,在步骤2355和2380,要以旁路模式对第二二进制数和第三二进制数进行解码。其他步骤与图22中所示的步骤相似。

[0204] 根据本实施例,VVC草案5规范可以被修改为如表17中所示。

[0205] 表17.CABAC上下文分配给“intra_luma_ref_idx”的基于上下文编解码的二进制数

[0206]

语法元素	binIdx					
	0	1	2	3	4	>= 5
…	…	…	…	…	…	…
intra_luma_ref_idx[ ][ ]	0	旁路	na	na	na	na
…	…	…	…	…	…	…

[0207] SMVD标志的信令

[0208] VVC草案5中的SMVD标志

[0209] sym_mvd_flag语法元素指示对INTER编解码单元使用对称运动矢量差。VVC的SMVD运动矢量编解码过程对给定CU相对于第一参考画面列表L0的运动矢量差进行编解码。然后根据L0运动矢量差推导出另一个参考画面列表(L1)中考虑的CU的运动矢量差。实际上,在这种模式下,两个运动矢量差是对称的。L1运动矢量差在x-和y-分量上都等于L0运动矢量差的相反数。在VVC草案5中,以常规CABAC模式使用单个CABAC上下文对sym_mvd_flag进行编解码和解码。

[0210] SMVD标志的简化的编解码

[0211] 在一个实施例中,提出以旁路模式对该标志进行编码和解码。所提出的简化不影响VVC的压缩效率。根据该实施例,VVC草案5语法规范可以被修改为如表18所示。特别地,svm_mvd_flag的描述符从ae(v)变为u(1)。

[0212] 表18.VVC草案5上的smvd_flag的简化的信令

[0213]

coding_unit(x0, y0, cbWidth, cbHeight, treeType) {	描述符
…	
if(slice_type == B)	
inter_pred_idc[x0][y0]	ae(v)
if(sps_affine_enabled_flag && cbWidth >= 16 && cbHeight >= 16) {	
inter_affine_flag[x0][y0]	ae(v)
if(sps_affine_type_flag && inter_affine_flag[x0][y0])	
cu_affine_type_flag[x0][y0]	ae(v)
}	
if(sps_smvd_enabled_flag && inter_pred_idc[x0][y0] = = PRED_BI && !inter_affine_flag[x0][y0] && RefIdxSymL0 > −1 && RefIdxSymL1 > −1)	
~~**sym_mvd_flag**[x0][y0]~~	~~ae(v)~~
sym_mvd_flag[x0][y0]	u(1)
if(inter_pred_idc[x0][y0] != PRED_L1) {	
if(NumRefIdxActive[0] > 1 && !sym_mvd_flag[x0][y0])	
ref_idx_l0[x0][y0]	ae(v)
mvd_coding(x0, y0, 0, 0)	
if(MotionModelIdc[x0][y0] > 0)	

[0214]

mvd_coding(x0, y0, 0, 1)	
if(MotionModelIdc[x0][y0] > 1)	
mvd_coding(x0, y0, 0, 2)	
mvp_l0_flag[x0][y0]	ae(v)
} else {	
MvdL0[x0][y0][0] = 0	
MvdL0[x0][y0][1] = 0	
}	
…	

[0215] 本文描述了各种方法,并且每种方法包括用于实现所述方法的一个或多个步骤或动作。除非该方法的适当操作需要步骤或动作的特定顺序,否则特定步骤和/或动作的顺序和/或使用可以被修改或组合。

[0216] 本申请中描述的各种方法和其他方面可以被用于修改模块,例如,如图1和图4所示的视频编码器100和视频解码器200的熵编码和解码模块(145、230)。此外,本方面不限于VVC或HEVC,并且可以被应用于,例如,其他标准和推荐,以及任何此类标准和推荐的扩展。除非另有说明或技术上排除,否则本申请中描述的各方面可以被单独使用或组合使用。

[0217] 在本申请中使用了各种数值,例如,上下文模型ID。特定值是出于示例目的,并且所描述的各方面不限于这些特定值。

[0218] 图24示出了在其中实现各个方面和实施例的系统的示例的框图。系统2400可以被实现为包括以下组件的设备,并且被配置为执行本申请中描述的一个或多个方面。这种设备的示例包括但不限于各种电子设备,诸如个人计算机,膝上型计算机,智能电话,平板计算机,数字多媒体机顶盒,数字电视接收器,个人视频记录系统,连接的家用电器以及服务器。系统2400的元件可以被单独或组合地体现在单个集成电路、多个IC和/或分立组件中。例如,在至少一个实施例中,系统2400的处理和编码器/解码器元件跨多个IC和/或分立组件分布。在各种实施例中,系统2400经由,例如通信总线或通过专用输入和/或输出端口,通信地耦接到其他系统或其他电子设备。在各种实施例中,系统2400被配置为实现本申请中描述的一个或多个方面。

[0219] 系统2400包括至少一个处理器2410,该处理器2410被配置为执行加载在其中的指令,以实现例如本申请中描述的多个方面。处理器2410可以包括嵌入式存储器,输入输出接口以及本领域已知的各种其他电路。系统2400包括至少一个存储器2420(例如,易失性存储设备和/或非易失性存储设备)。系统2400包括存储设备2440,其可以包括非易失性存储器和/或易失性存储器,包括但不限于:EEPROM、ROM、PROM、RAM、DRAM、SRAM、闪存、磁盘驱动器和/或光盘驱动器。作为非限制性示例,存储设备2440可以包括内部存储设备、附接存储设备和/或网络可访问存储设备。

[0220] 系统2400包括编码器/解码器模块2430,其被配置为,例如,对数据进行处理以提供编码的视频或解码的视频,并且编码器/解码器模块2430可以包括其自己的处理器和存储器。编码器/解码器模块2430表示可以被包括在设备中以执行编码和/或解码功能的一个或多个模块。众所周知,设备可以包括编码模块和解码模块中的一个或两个。另外,如本领域技术人员已知的,编码器/解码器模块2430可以被实现为系统2400的分离元件,或者可以作为硬件和软件的组合被合并到处理器2410内。

[0221] 要加载到处理器2410或编码器/解码器2430上以执行本申请中描述的各方面的程序代码可以被存储在存储设备2440中,然后被加载到存储器2420上以由处理器2410执行。按照各种实施例,处理器2410、存储器2420、存储设备2440以及编码器/解码器模块2430中的一个或多个可以在本申请所述的过程的执行期间存储多个项目中的一个或多个。这样存储的项目可以包括但不限于:输入视频、解码的视频或解码的视频的部分、比特流、矩阵、变量以及来自等式、公式、运算和运算逻辑的处理的中间或最终结果。

[0222] 在若干实施例中,处理器2410和/或编码器/解码器模块2430的内部的存储器被用于存储指令并为编码或解码期间需要的处理提供工作存储器。然而,在其他实施例中,处理

设备(例如,处理设备可以是处理器2410或编码器/解码器模块2430)外部的存储器被用于这些功能中的一个或多个。外部存储器可以是存储器2420和/或存储设备2440,例如,动态易失性存储器和/或非易失性闪存。在若干实施例中,外部非易失性闪存被用于存储电视的操作系统。在至少一个实施例中,诸如RAM的快速外部动态易失性存储器被用作视频编解码和解码操作的工作存储器,视频编解码和解码操作诸如用于MPEG-2、HEVC或VVC。

[0223] 如块2405所示,可以通过各种输入设备来提供到系统2400的元件的输入。这种输入设备包括但不限于:(i)接收例如由广播者通过空中发送的RF信号的RF部分、(ii)复合输入终端、(iii)USB输入终端,和/或(iv)HDMI输入终端。

[0224] 在各种实施例中,块2405的输入设备具有如本领域中已知的相关联的相应输入处理元件。例如,RF部分可以与适合于下述操作的元件相关联:(i)选择所需频率(也称为选择信号,或将信号频带限制在一频带内);(ii)对所选信号进行下变换;(iii)再次频带限制到较窄的频带以选择(例如)在某些实施例中可以被称为信道的信号频带;(iv)对下变换的和频带限制的信号进行解调;(v)执行纠错;以及(vi)进行解复用以选择所需的数据分组流。各种实施例的RF部分包括用于执行这些功能的一个或多个元件,例如,频率选择器、信号选择器、频带限制器、信道选择器、滤波器,下变换器,解调器,纠错器和解复用器。RF部分可以包括执行这些功能中的各种功能的调谐器,上述功能包括例如将接收到的信号下变换到较低频率(例如,中频或近基带频率)或基带。在一个机顶盒实施例中,RF部分及其相关联的输入处理元件接收通过有线(例如电缆)介质发送的RF信号,并通过滤波,下变换以及再滤波至期望的频带来执行频率选择。各种实施例对上述(及其他)元件的顺序进行了重新布置,移除这些元件中的一些,和/或添加执行类似或不同功能的其他元件。添加元件可以包括在现有元件之间插入元件,例如,插入放大器和模数转换器。在各种实施例中,RF部分包括天线。

[0225] 另外,USB和/或HDMI终端可以包括用于跨USB和/或HDMI连接将系统2400连接至其他电子设备的相应接口处理器。应当理解的是,输入处理的各方面,例如里德-所罗门纠错,可以根据需要例如在单独的输入处理IC内或在处理器2410内实现。类似地,USB或HDMI接口处理的各方面可以根据需要在单独的接口IC内或在处理器2410内实现。解调的,纠错的和解复用的流被提供给各种处理元件,包括例如与存储器和存储元件组合操作的处理器2410和编码器/解码器2430,从而根据需要对数据流进行处理以用于在输出设备上呈现。

[0226] 系统2400的多个元件可以被设置在集成壳体内。在该集成壳体内,使用合适的连接布置2415,例如本领域已知的内部总线,包括I2C总线、布线和印刷电路板,各个元件被互连并且在彼此之间传输数据。

[0227] 系统2400包括经由通信信道2490启用与其他设备进行通信的通信接口2450。通信接口2450可以包括但不限于被配置为在通信信道2490上发送和接收数据的收发器。通信接口2450可以包括但不限于调制解调器或网卡,并且通信信道2490可以例如在有线和/或无线介质内实现。

[0228] 在各种实施例中,使用诸如IEEE 802.11的Wi-Fi网络,将数据流传输给系统2400。这些实施例的Wi-Fi信号经由适用于Wi-Fi通信的通信信道2490和通信接口2450被接收。这些实施例的通信信道2490通常被连接到接入点或路由器,该接入点或路由器提供对包括互联网的外部网络的访问,以允许流式传输应用以及其他过顶通信。其他实施例使用通过输

入框2405的HDMI连接来传递数据的机顶盒向系统2400提供流式传输的数据。还有其他实施例使用输入框2405的RF连接向系统2400提供流式传输的数据。

[0229] 系统2400可以将输出信号提供给各输出设备,包括显示器2465、扬声器2475和其他外围设备2485。在实施例的各种示例中,其他外围设备2485包括独立DVR、盘播放器、立体声系统、照明系统以及基于系统2400的输出提供功能的其它设备中的一个或多个。在各种实施例中,控制信号使用信令在系统2400与显示器2465、扬声器2475或其他外围设备2485之间通信,所述信令诸如AV.Link、CEC、或者在无论有无用户干预的情况下启用设备到设备控制的其他通信协议。输出设备可以经由通过相应接口2460、2470和2480的专用连接,通信地耦接至系统2400。可替换地,输出设备可以通过经由通信接口2450使用通信信道2490被连接至系统2400。显示器2465和扬声器2475可以与系统2400的其他组件被集成在例如电视的电子设备中的单个单元中。在各种实施例中,显示接口2460包括显示驱动器,例如,定时控制器(T Con)芯片。

[0230] 例如,如果输入2405的RF部分是分离的机顶盒的一部分,则显示器2465和扬声器2475可以可替换地与其他组件中的一个或多个分离。在其中显示器2465和扬声器2475是外部组件的各种实施例中,输出信号可以经由专用输出连接被提供,该专用输出连接包括例如HDMI端口,USB端口或COMP输出。

[0231] 根据一实施例,提供了一种视频解码的方法,包括:从比特流中多多个二进制码元进行解码,其中使用熵解码引擎的旁路模式对所述多个二进制码元中的第一二进制码元进行解码;以及基于二值化方案,响应于所述多个二进制码元生成语法元素的值。

[0232] 根据一实施例,提供了一种视频编码的方法,包括:访问指示语法元素的值的多个二进制码元;以及对所述多个二进制码元进行编码,其中使用熵编码引擎的旁路模式对所述多个二进制码元中的第一二进制码元进行编码。

[0233] 根据另一实施例,提供了一种用于视频解码的装置,包括一个或多个处理器,其中所述一个或多个处理器被配置为:从比特流中多多个二进制码元进行解码,其中使用熵解码引擎的旁路模式对所述多个二进制码元中的第一二进制码元进行解码;以及基于二值化方案,响应于所述多个二进制码元生成语法元素的值。该装置还可包括耦接到所述一个或多个处理器的一个或多个存储器。

[0234] 根据另一实施例,提供了一种用于视频编码的装置,包括一个或多个处理器,其中所述一个或多个处理器被配置为:访问指示语法元素的值的多个二进制码元;以及对所述多个二进制码元进行编码,其中使用熵编码引擎的旁路模式对所述多个二进制码元中的第一二进制码元进行编码。

[0235] 根据另一实施例,提供了一种视频解码的装置,包括:用于从比特流中对多个二进制码元进行解码的部件,其中使用熵解码引擎的旁路模式对所述多个二进制码元中的第一二进制码元进行解码;以及用于基于二值化方案,响应于所述多个二进制码元生成语法元素的值的部件。

[0236] 根据另一实施例,提供了一种视频编码的装置,包括:用于访问指示语法元素的值的多个二进制码元的部件;以及用于对所述多个二进制码元进行编码的部件,其中使用熵编码引擎的旁路模式对所述多个二进制码元中的第一二进制码元进行编码。

[0237] 根据另一实施例,通过执行以下操作来形成包括编码的视频的信号:访问指示语

法元素的值的多个二进制码元;以及对所述多个二进制码元进行编码,其中使用熵解码引擎的旁路模式对所述多个二进制码元中的第一二进制码元进行编码。

[0238] 根据一实施例,使用所述旁路模式对所述多个二进制码元中的一个或多个其他二进制码元进行解码或编码。

[0239] 根据一实施例,对所述多个二进制码元的其余部分进行基于上下文的解码或编码。所述多个二进制码元的其余部分的每个二进制码元可以使用不同的上下文模型。在另一实施例中,使用所述旁路模式对所述多个二进制码元中的全部二进制码元进行解码或编码。

[0240] 根据一实施例,二进制标志被基于上下文的编码或解码,其中进一步响应于所述二进制标志,所述语法元素的所述值被生成。

[0241] 在一个实施例中,所述标志指示在生成所述块的两个时间预测器的所述加权平均值时是否应用相等的权重。

[0242] 根据一实施例,所述语法元素指示在生成块的两个时间预测器的加权平均值时使用的权重的索引。

[0243] 在一个实施例中,所述语法元素指示要使用哪个运动矢量预测器对块的运动矢量进行编码或解码。

[0244] 在一个实施例中,确定是否对所述块应用SMVD(对称运动矢量差),并且仅当对所述块应用SMVD时才使用所述旁路模式。

[0245] 在一个实施例中,截断莱斯二值化被用作二值化方案。

[0246] 在一个实施例中,语法元素被以旁路模式编码和解码。该语法元素指示,当子块变换被使用时,用于对当前编解码单元的纹理数据进行编解码的非零残差变换单元的位置。

[0247] 在一个实施例中,语法元素被以旁路模式编码和解码,该语法元素指示在子块变换中是否使用了不对称二叉划分。

[0248] 在一个实施例中,语法元素被以旁路模式编码和解码,该语法元素指示在子块变换中使用的二叉划分的方向。

[0249] 在一个实施例中,语法元素被以旁路模式编码和解码,该语法元素指示当前编解码单元到帧内子分割的划分的方向。

[0250] 在一个实施例中,用于表示语法元素的第一二进制数被以常规模式编码和解码,该语法元素指示哪个参考行被用于帧内预测,并且用于表示所述语法元素的一个或多个剩余二进制数被以旁路模式编码和解码。

[0251] 在一个实施例中,语法元素被以旁路模式编码和解码,该语法元素指示是否对当前编解码单元使用对称运动矢量差编解码模式。

[0252] 一实施例提供了一种计算机程序,该计算机程序包括当被一个或多个处理器执行时使得一个或多个处理器执行根据上述实施例中任一个的编码方法或解码方法的指令。本实施例中的一个或多个还提供了一种计算机可读存储介质,该存储介质上存储有用于根据上述方法对视频数据进行编码或解码的指令。一个或多个实施例还提供了一种其上存储有根据上述方法生成的比特流的计算机可读存储介质。一个或多个实施例还提供了用于发送或接收根据上述方法生成的比特流的方法和装置。

[0253] 各种实现方式都涉及解码。如本申请中所使用的“解码”可以包含例如对接收到的

编码的序列执行的过程的全部或部分,以便产生适合于显示的最终输出。在各种实施例中,这样的过程包括通常由解码器执行的过程中的一个或多个,例如,熵解码,逆量化,逆变换和差分解码。基于特定描述的上下文,短语“解码过程”是专门指操作的子集还是广义上指更广泛的解码过程将是清楚的,并且相信本领域技术人员会很好地理解。

[0254] 各种实现方式都涉及编码。以与上述关于“解码”的讨论类似的方式,如在本申请中使用的“编码”可以包含例如对输入视频序列执行的过程的全部或部分,以便产生编码的比特流。

[0255] 应该注意的是,本文所使用的语法元素,例如,用于描绘GBi索引的语法,是描述性术语。因此,它们不排除使用其他语法元素名称。

[0256] 本文描述的实现方式和方面可以以例如,方法或过程、装置、软件程序、数据流或信号的方式来实现。即使仅在单一形式的实现方式中进行了讨论(例如,仅作为一种方法讨论),所讨论的特征的实现方式也可以以其他形式(例如,设备或程序)来实现。装置可以以例如,适当的硬件、软件和固件来实现。方法可以在,例如装置,例如处理器中实现,该处理器通常指的是处理设备,包括例如,计算机、微处理器、集成电路或可编程逻辑设备。处理器还包括通信设备,例如计算机、手机、便携式/个人数字助理(“PDA”),以及其他有助于终端用户之间的信息的通信的设备。

[0257] 对“一个实施例”或“一实施例”或“一个实现方式”或“一实现方式”以及它们的其他变型的引用意味着结合该实施例描述的特定特征、结构、特性等包括在至少一个实施例中。因而,贯穿本申请的各处出现的短语“在一个实施例中”或“在一实施例中”或“在一种实现方式中”或“在一实现方式中”以及任何其他变型的出现不一定全部是指相同的实施例。

[0258] 另外,本申请可以涉及“确定”各种信息。对信息进行确定可以包括例如对信息进行估计、对信息进行计算、对信息进行预测或从存储器检索信息中的一种或更多种。

[0259] 进一步地,本申请可以涉及“访问”各种信息。对信息进行访问可以包括以下中的一种或多种:例如,对信息进行接收、对信息进行检索(例如从存储器中)、对信息进行存储、对信息进行移动、对信息进行复制、对信息进行计算、对信息进行确定,对信息进行预测或对信息进行估计。

[0260] 此外,本申请可以涉及“接收”各种信息。接收与“访问”一样,是一个广义的术语。对信息进行接收可以包括例如,对信息进行访问或对信息进行检索(例如,从存储器中)中的一个或多个。而且,在以下操作期间通常以一种或另一种方式涉及“接收”:例如对信息进行存储、对信息进行处理、对信息进行发送、对信息进行移动、对信息进行复制、对信息进行擦除、对信息进行计算、对信息进行确定、对信息进行预测或对信息进行估计。

[0261] 应当认识到的是,对以下“/”、“和/或”以及“至少一个”中的任一个的使用,例如在“A/B”、“A和/或B”和“A和B中的至少一个”的情况下,旨在包含仅选择列出的第一选项(A)或仅选择列出的第二选项(B)或两个选项(A和B)都选。作为进一步的示例,在“A、B和/或C”和“A,B和C中至少一个”的情况下,这种短语旨在包含仅选择列出的第一选项(A)、或仅选择列出的第二选项(B)、或仅选择列出的第三选项(C)、或仅选择列出的第一和第二选项(A和B)、或仅选择列出的第一和第三选项(A和C)、或者仅选择列出的第二和第三选项(B和C)、或者选择全部三个选项(A和B和C)。如对于本领域和相关领域的普通技术人员所清楚的,这可以被扩展到所所列出的尽可能多的项目。

[0262] 此外,如本文中所使用的,词语“信号”尤其是指向对应的解码器指示某些东西。例如,在某些实施例中,编码器向解码器用信令通知分段式线性模型中的段的数量。以这种方式,在一实施例中,在编码器侧和解码器侧都使用相同的参数。因此,例如,编码器可以将特定参数发送(显式信令)给解码器,以使得解码器可以使用相同的特定参数。相反地,如果解码器已经具有特定的参数以及其他参数,则可以不用发送就使用信令(隐式信令)以简单地允许解码器知道并选择该特定参数。通过避免传输任何实际函数,在各种实施例中实现了比特节省。应当认识到的是,可以以各种方式来完成信令。例如,在各种实施例中,使用一个或多个语法元素、标志等来将信息用信令通知对应的解码器。尽管前面涉及词语“信号”的动词形式,但词语“信号”在本文中也可以用作名词。

[0263] 如对于本领域的普通技术人员将显而易见的是,实现方式可以产生各种信号,这些信号被格式化为携带可以例如被存储或发送的信息。信息可以包括例如用于执行方法的指令或由所描述的实现方式之一产生的数据。例如,信号可以被格式化为携带所描述的实施例的比特流。这种信号可以被格式化为例如电磁波(例如,使用频谱的射频部分)或基带信号。格式化可以包括例如对数据流进行编码并且利用编码的数据流来调制载波。信号所携带的信息可以是例如模拟或数字信息。众所周知,该信号可以通过各种不同的有线或无线链路传输。这种信号可以被存储在处理器可读介质上。

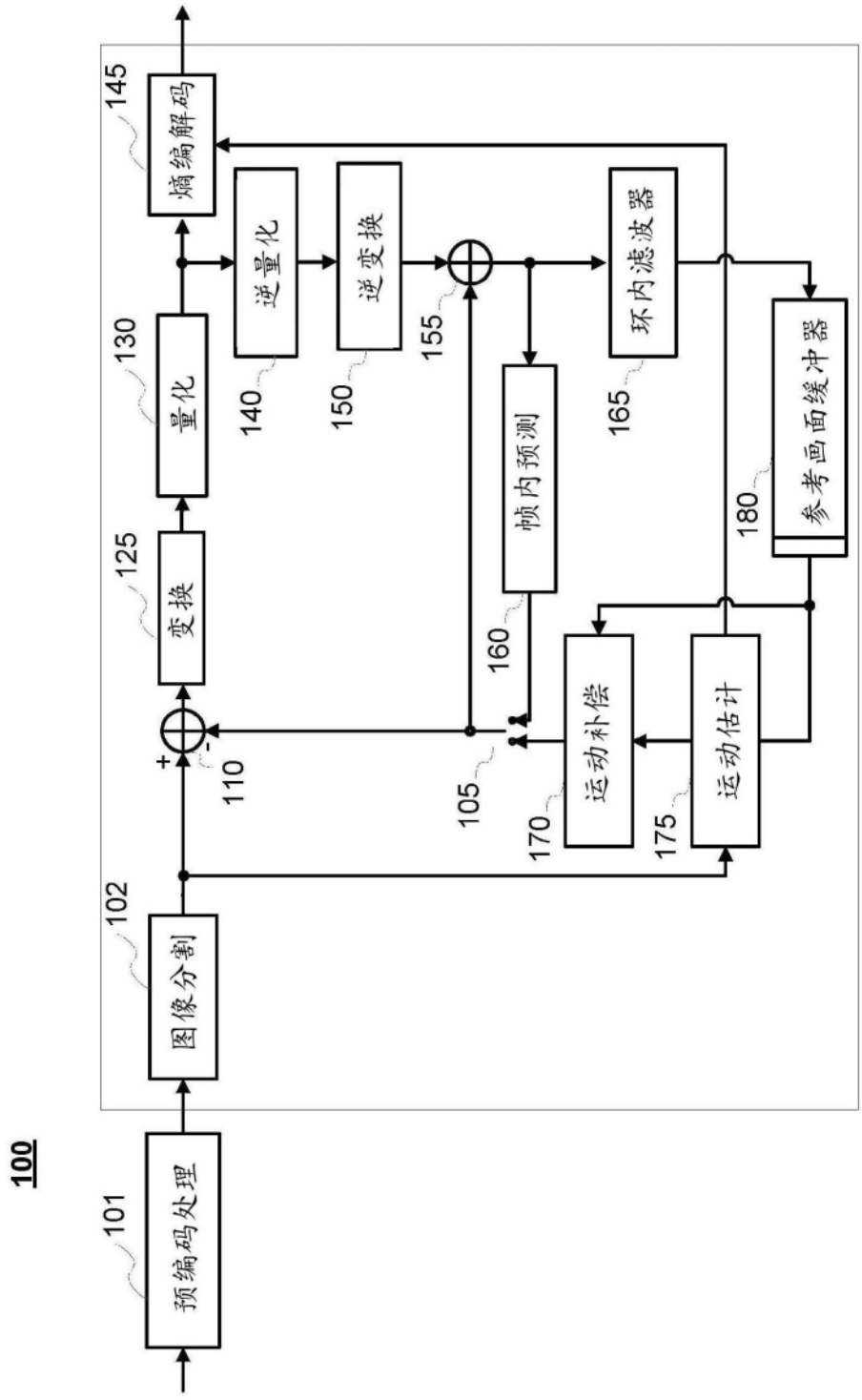

100
101
预编码处理
102
图像分割
110
125
变换
130
量化
145
熵编解码
140
逆量化
150
逆变换
155
160
帧内预测
165
环内滤波器
180
参考画面缓冲器
105
170
运动补偿
175
运动估计

图1

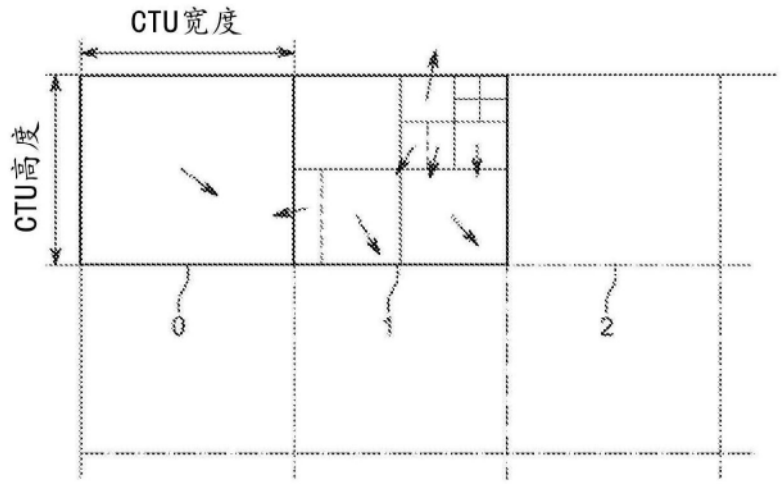

CTU宽度
CTU高度
0
1
2

图2

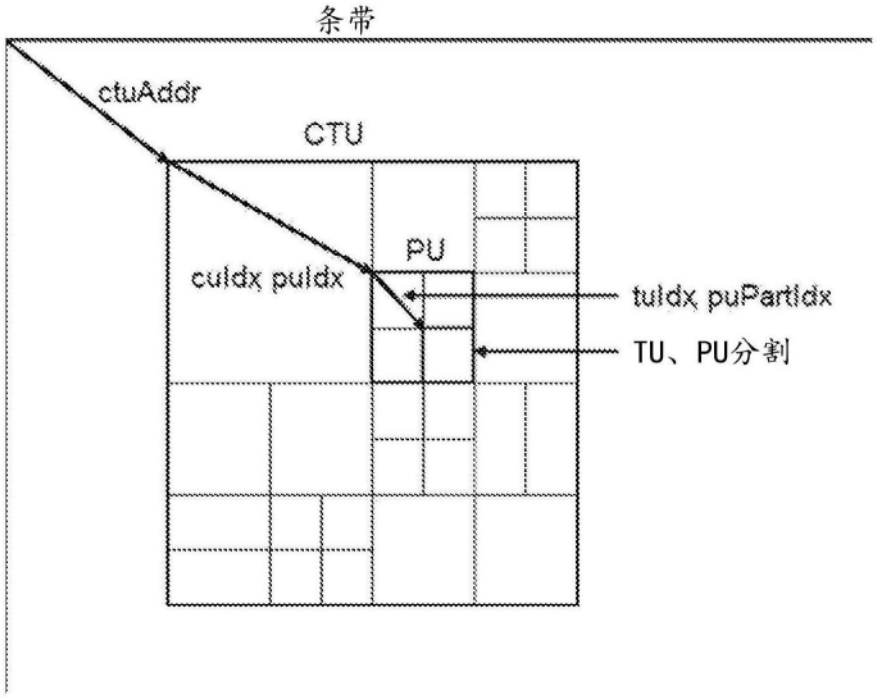

条带
ctuAddr
CTU
PU
cuIdx puIdx
tuIdx puPartIdx
TU、PU分割

图3

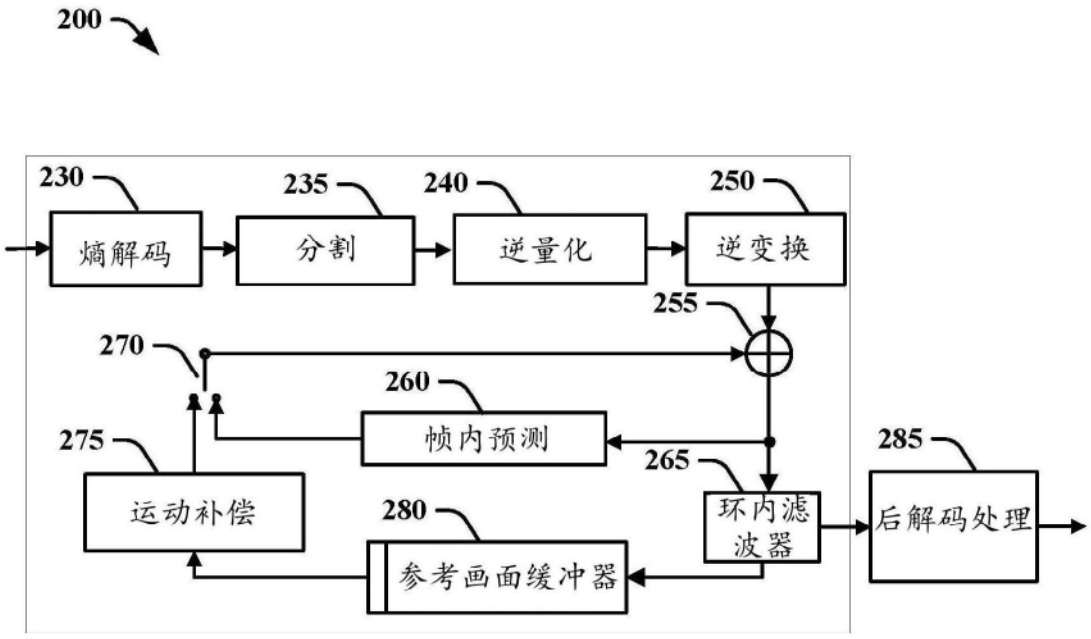

200
230
熵解码
235
分割
240
逆量化
250
逆变换
255
270
260
帧内预测
275
运动补偿
265
环内滤波器
285
后解码处理
280
参考画面缓冲器

图4

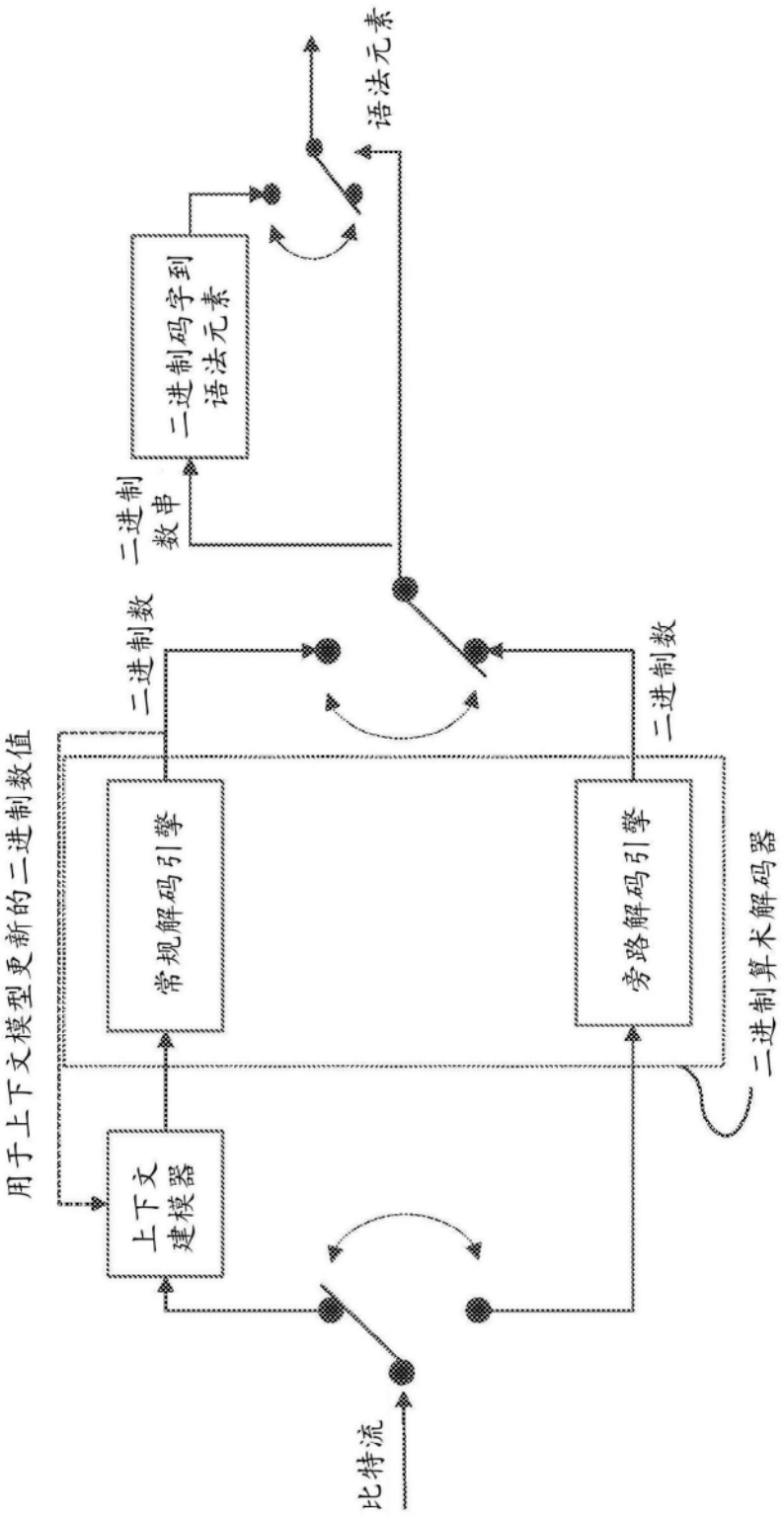

比特流
上下文建模器
用于上下文模型更新的二进制数值
常规解码引擎
旁路解码引擎
二进制算术解码器
二进制数
二进制数
二进制数串
二进制码字到语法元素
语法元素

图5

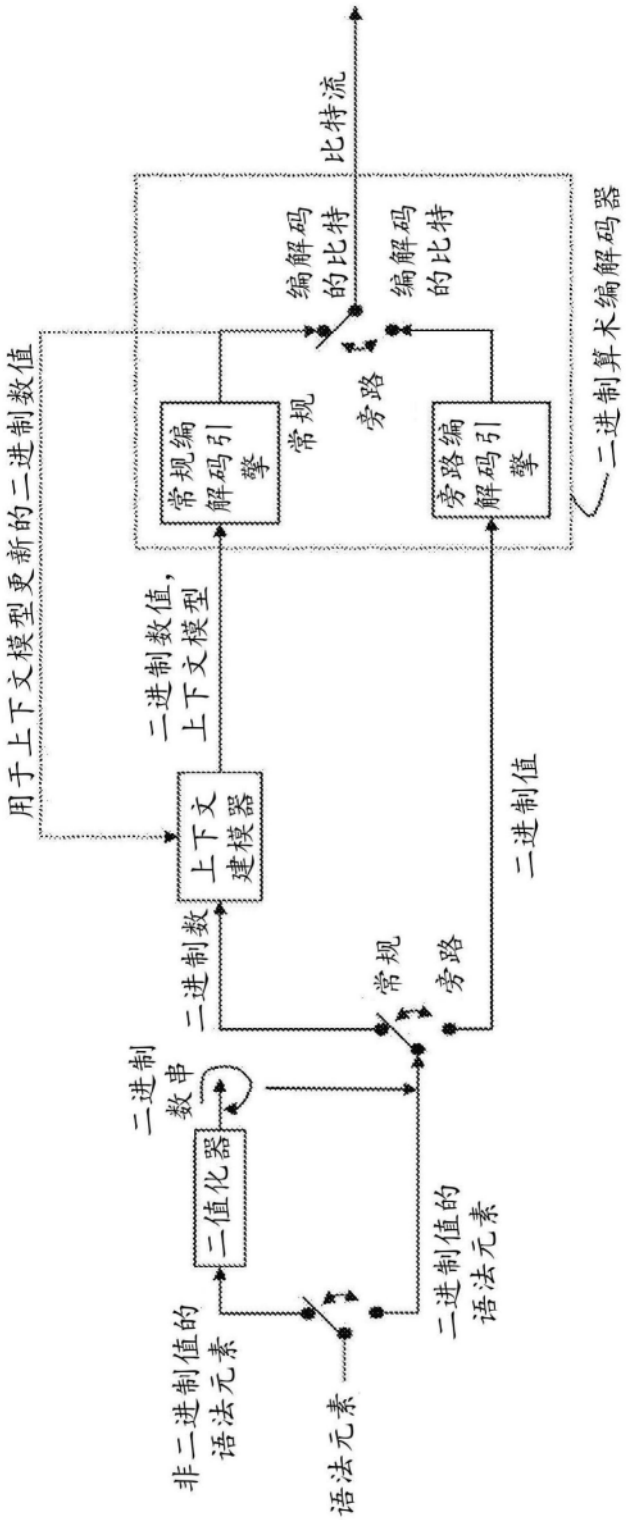

比特流
编解码的比特
编解码的比特
常规编码引擎
常规
旁路
旁路编码引擎
二进制算术编解码器
用于上下文模型更新的二进制数值
二进制数值，上下文模型
上下文建模器
二进制数
常规
旁路
二进制值
二进制数串
二值化器
非二进制值的语法元素
语法元素
二进制值的语法元素

图6

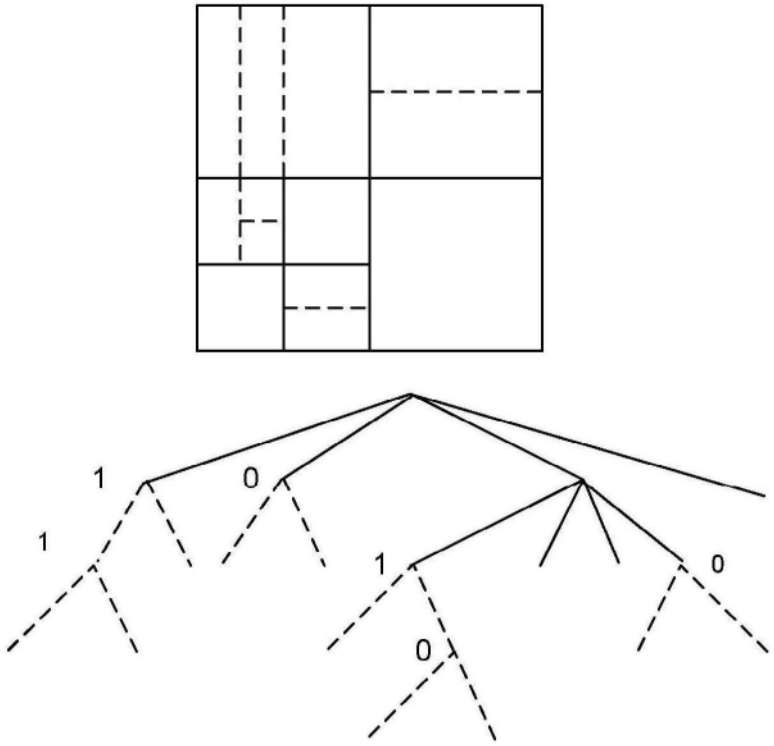


图7

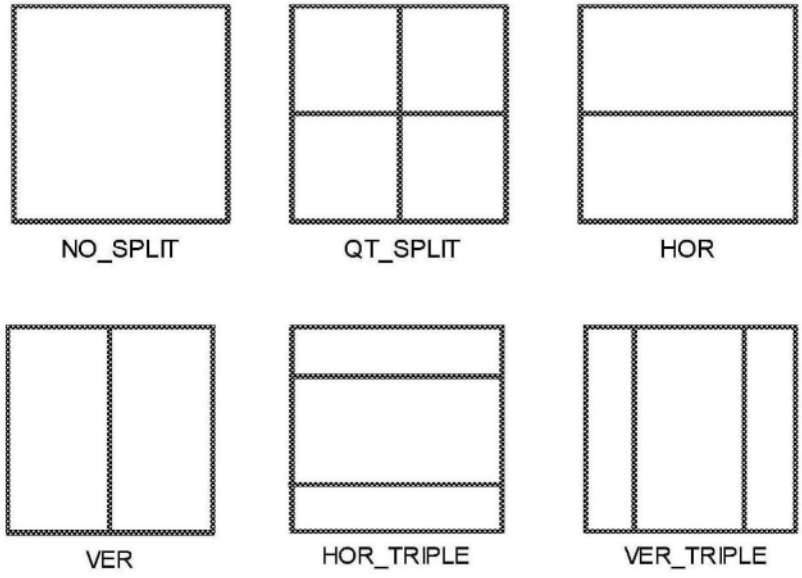


图8

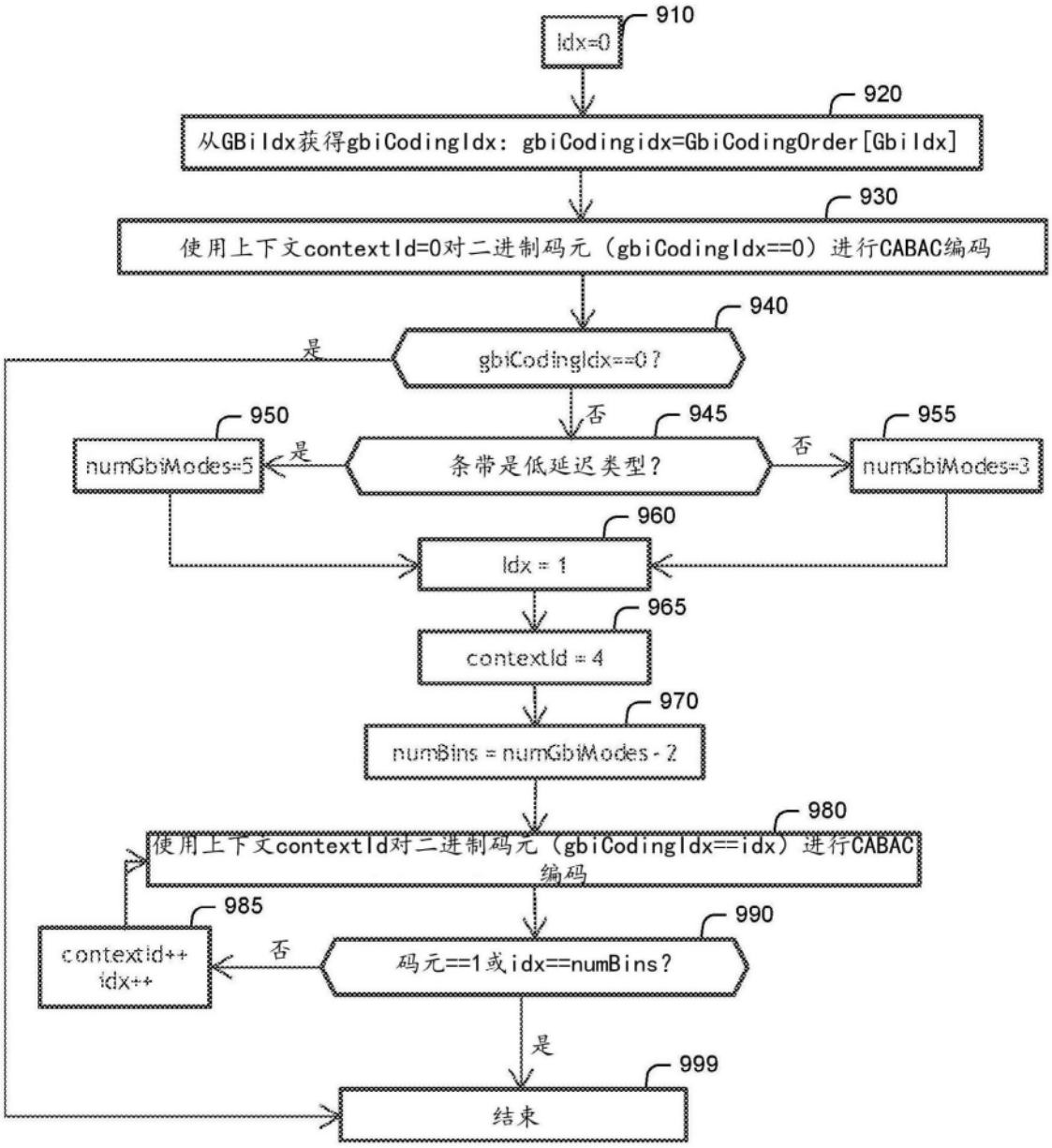

910
Idx=0
920
从GBiIdx获得gbiCodingIdx: gbiCodingidx=GbiCodingOrder[GbiIdx]
930
使用上下文contextId=0对二进制码元（gbiCodingIdx==0）进行CABAC编码
940
gbiCodingIdx==0?
是
否
950
numGbiModes=5
是
945
条带是低延迟类型?
否
955
numGbiModes=3
960
Idx = 1
965
contextId = 4
970
numBins = numGbiModes - 2
980
使用上下文contextId对二进制码元（gbiCodingIdx==idx）进行CABAC编码
985
contextid++
idx++
否
990
码元==1或idx==numBins?
是
999
结束

图9

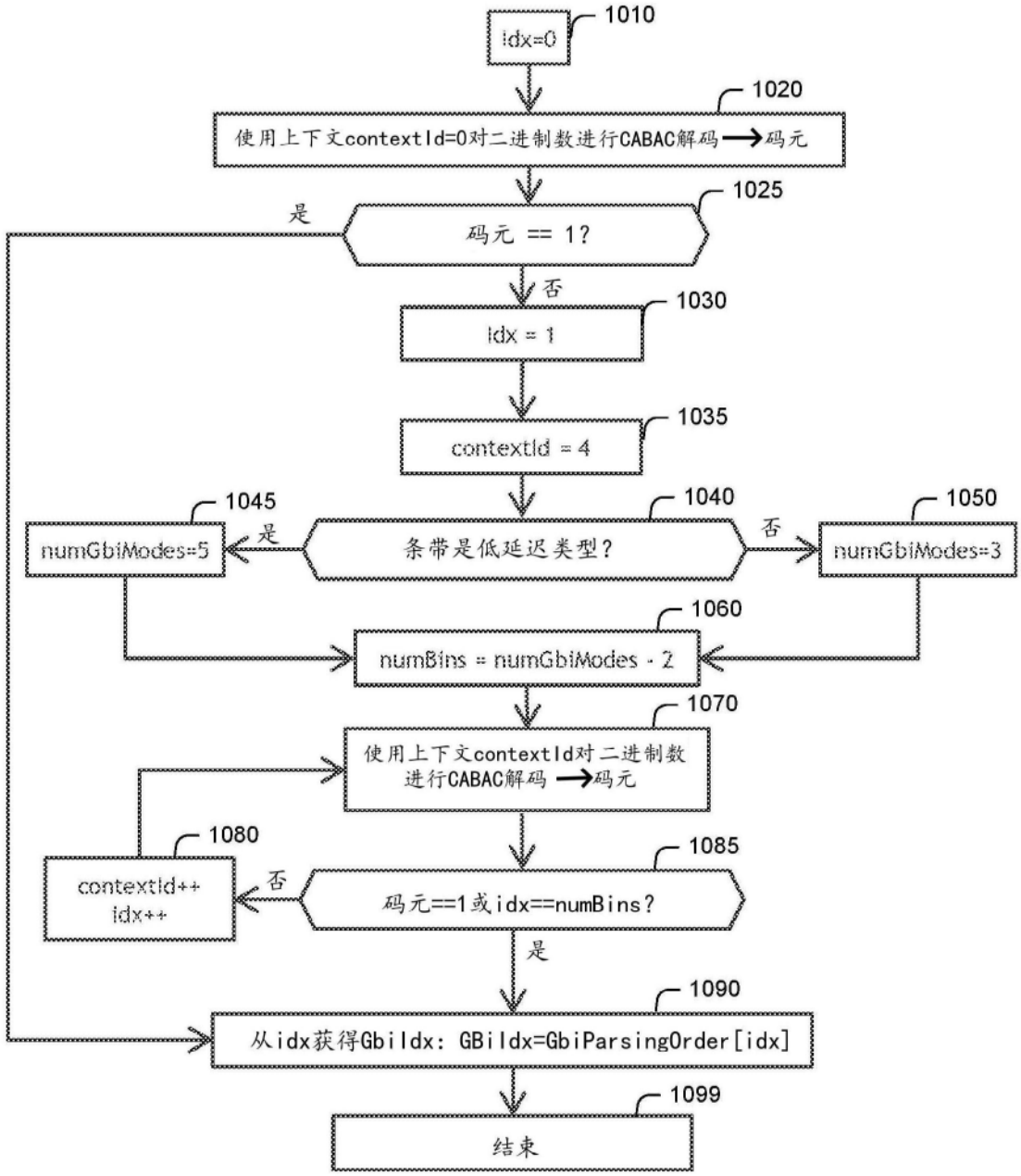

1010
idx=0
1020
使用上下文contextId=0对二进制数进行CABAC解码→码元
1025
是
码元 == 1?
否
1030
idx = 1
1035
contextId = 4
1045
numGbiModes=5
是
1040
条带是低延迟类型?
否
1050
numGbiModes=3
1060
numBins = numGbiModes * 2
1070
使用上下文contextId对二进制数
进行CABAC解码→码元
1080
contextId++
idx++
否
1085
码元==1或idx==numBins?
是
1090
从idx获得GbiIdx: GBiIdx=GbiParsingOrder[idx]
1099
结束

图10

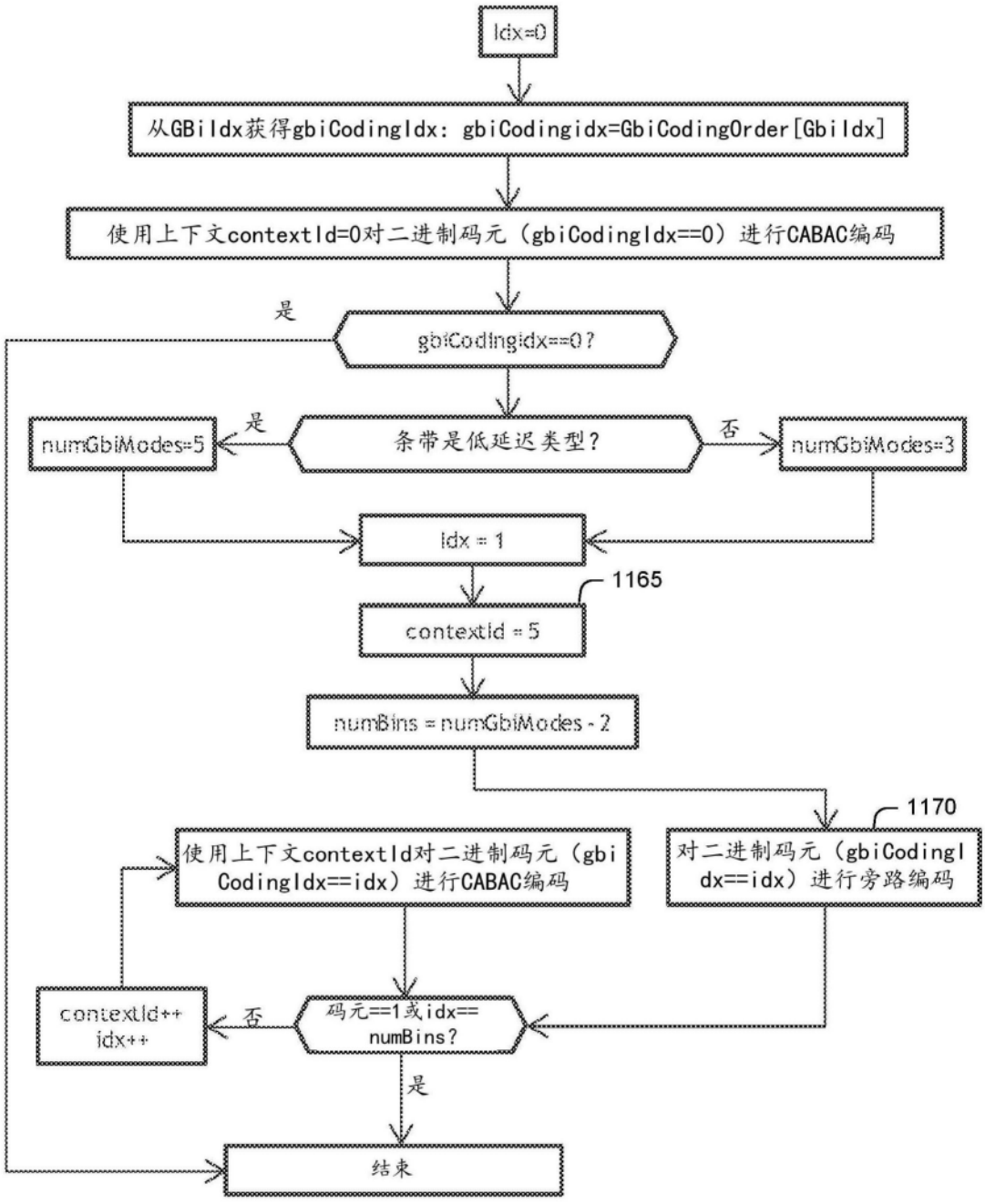

Idx=0
从GBIIdx获得gbiCodingIdx：gbiCodingidx=GbiCodingOrder[GbiIdx]
使用上下文contextId=0对二进制码元（gbiCodingIdx==0）进行CABAC编码
是
gbiCodingidx==0?
numGbiModes=5
是
条带是低延迟类型?
否
numGbiModes=3
Idx = 1
1165
contextId = 5
numBins = numGbiModes - 2
1170
使用上下文contextId对二进制码元（gbi CodingIdx==idx）进行CABAC编码
对二进制码元（gbiCodingI dx==idx）进行旁路编码
contextId++ idx++
否
码元==1或idx== numBins?
是
结束

图11

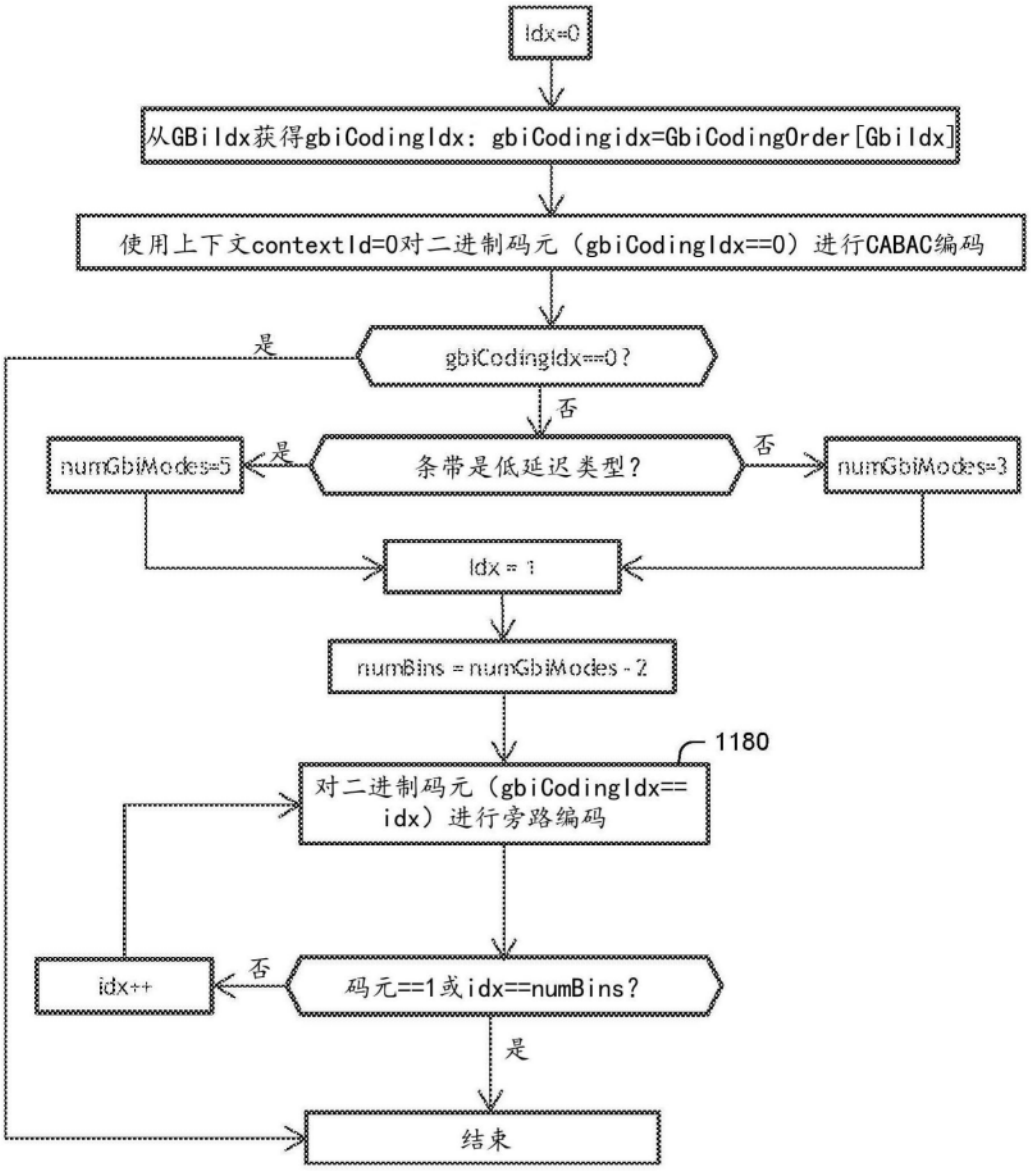

Idx=0
从GBIIdx获得gbiCodingIdx: gbiCodingidx=GbiCodingOrder[GbiIdx]
使用上下文contextId=0对二进制码元（gbiCodingIdx==0）进行CABAC编码
是
gbiCodingIdx==0?
否
numGbiModes=5
是
条带是低延迟类型?
否
numGbiModes=3
Idx = 1
numBins = numGbiModes - 2
1180
对二进制码元（gbiCodingIdx== idx）进行旁路编码
idx++
否
码元==1或idx==numBins?
是
结束

图11A

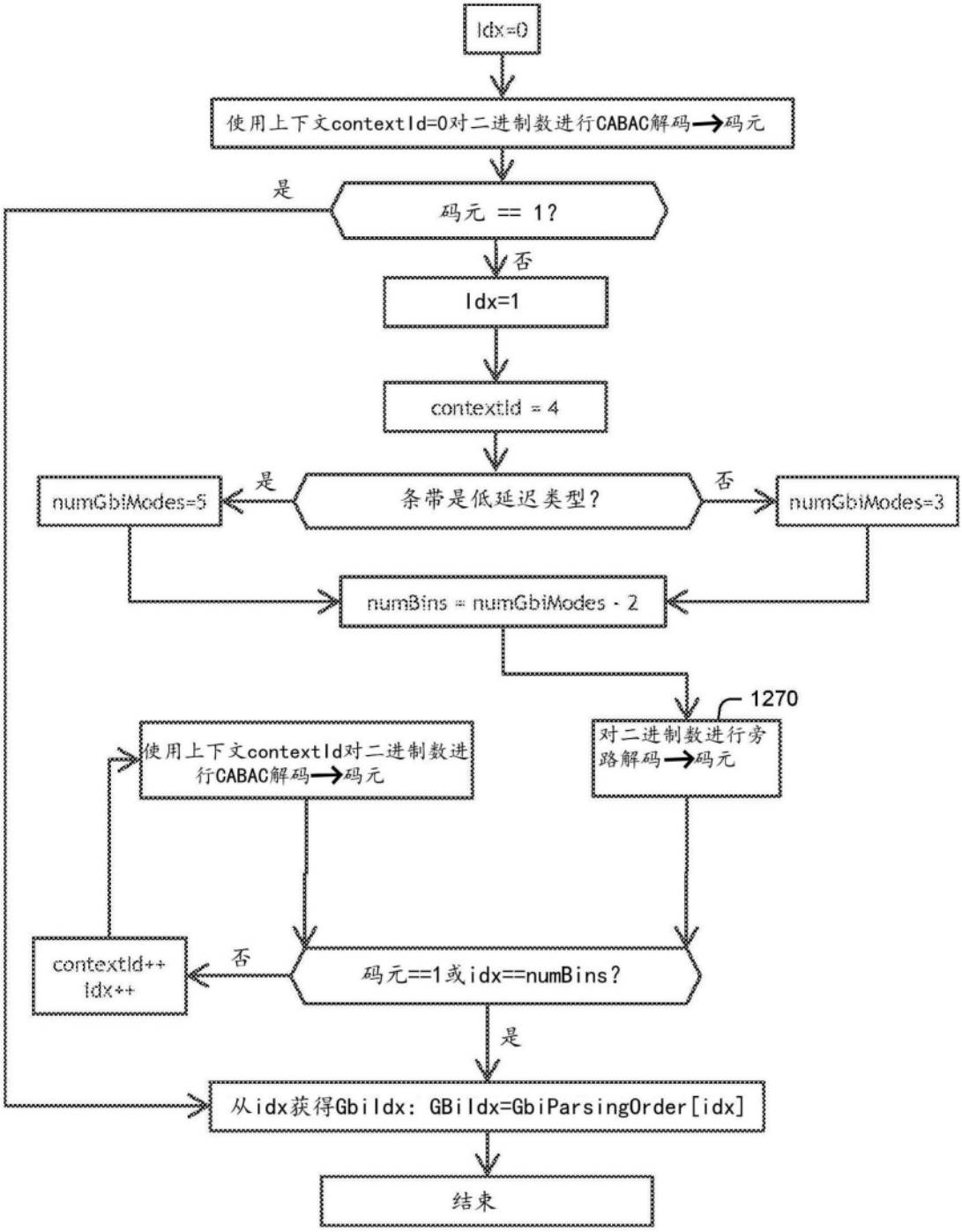

Idx=0
使用上下文contextId=0对二进制数进行CABAC解码→码元
是
码元 == 1?
否
Idx=1
contextId = 4
numGbiModes=5
是
条带是低延迟类型?
否
numGbiModes=3
numBins = numGbiModes - 2
1270
对二进制数进行旁路解码→码元
使用上下文contextId对二进制数进行CABAC解码→码元
contextId++
Idx++
否
码元==1或idx==numBins?
是
从idx获得GbiIdx: GBiIdx=GbiParsingOrder[idx]
结束

图12

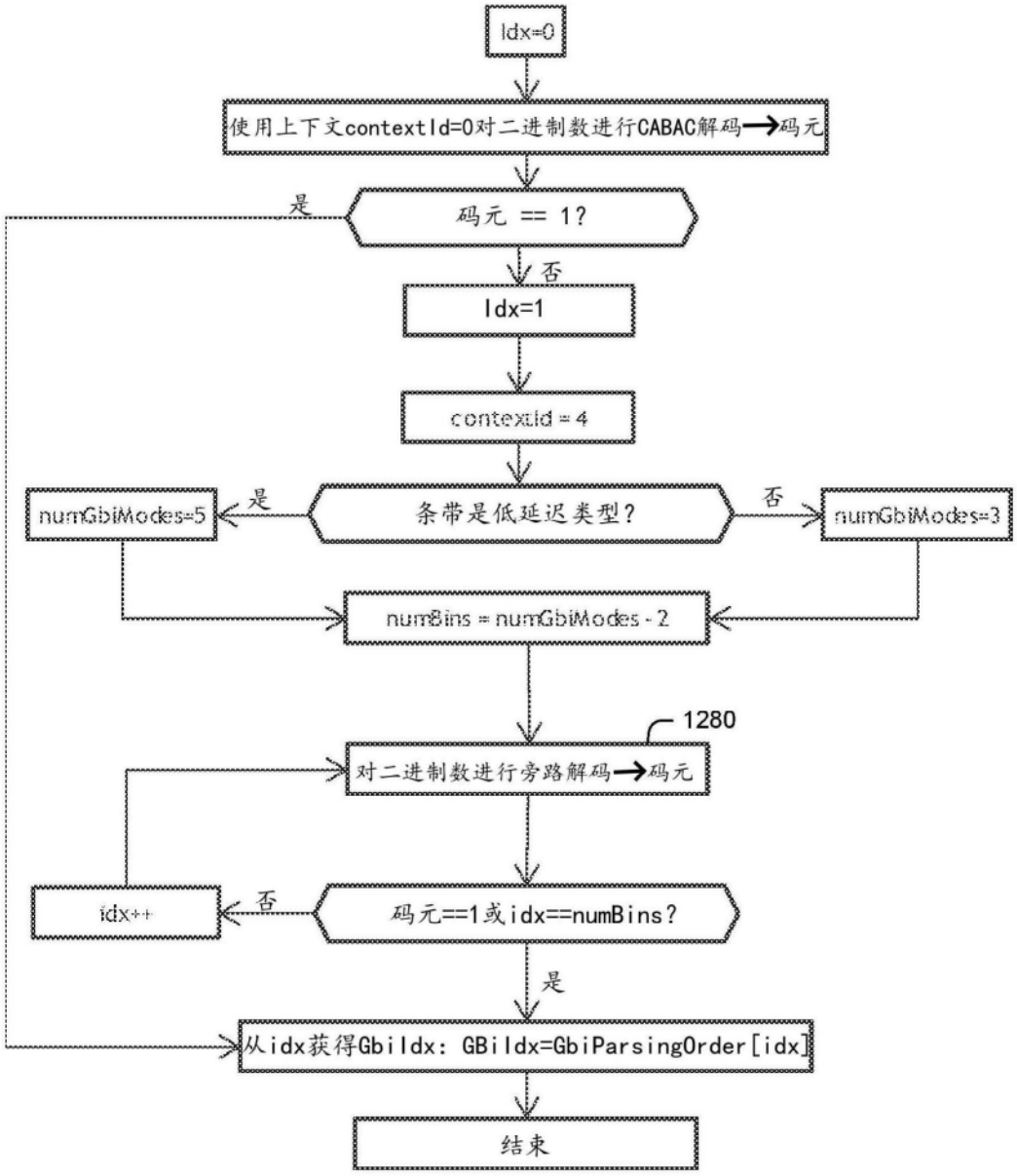

Idx=0
使用上下文contextId=0对二进制数进行CABAC解码→码元
是
码元 == 1?
否
Idx=1
contextId = 4
numGbiModes=5
是
条带是低延迟类型?
否
numGbiModes=3
numBins = numGbiModes - 2
1280
对二进制数进行旁路解码→码元
idx++
否
码元==1或idx==numBins?
是
从idx获得GbiIdx: GBiIdx=GbiParsingOrder[idx]
结束

图12A

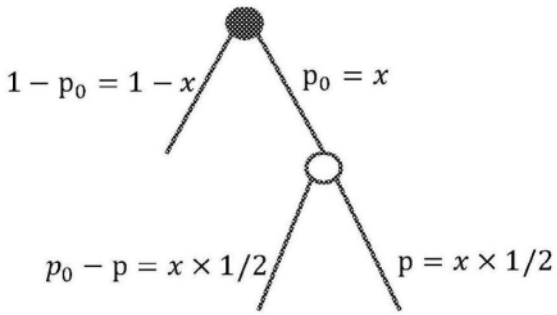

1 − p0 = 1 − x
p0 = x
p0 − p = x × 1/2
p = x × 1/2

图12B

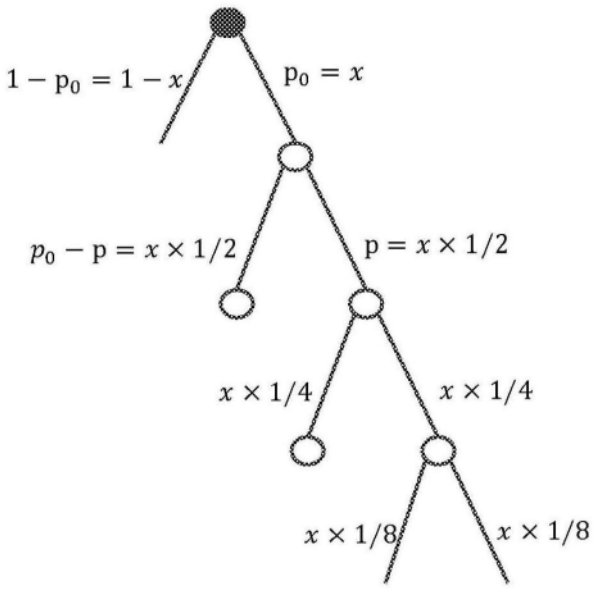

$1-p_0=1-x$
$p_0=x$
$p_0-p=x\times 1/2$
$p=x\times 1/2$
$x\times 1/4$
$x\times 1/4$
$x\times 1/8$
$x\times 1/8$

图12C

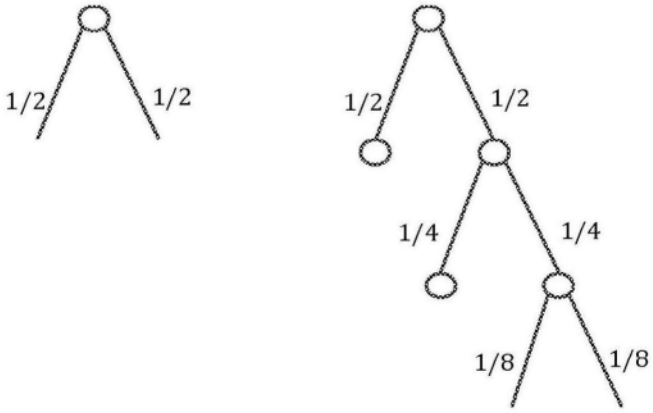

1/2
1/2
1/2
1/2
1/4
1/4
1/8
1/8

图12D

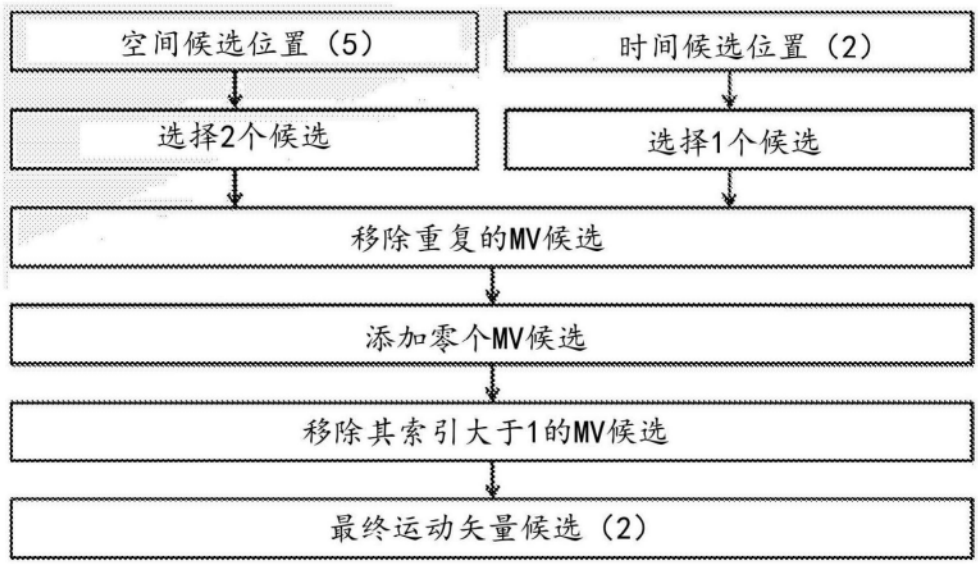

空间候选位置（5）
时间候选位置（2）
选择2个候选
选择1个候选
移除重复的MV候选
添加零个MV候选
移除其索引大于1的MV候选
最终运动矢量候选（2）

图13

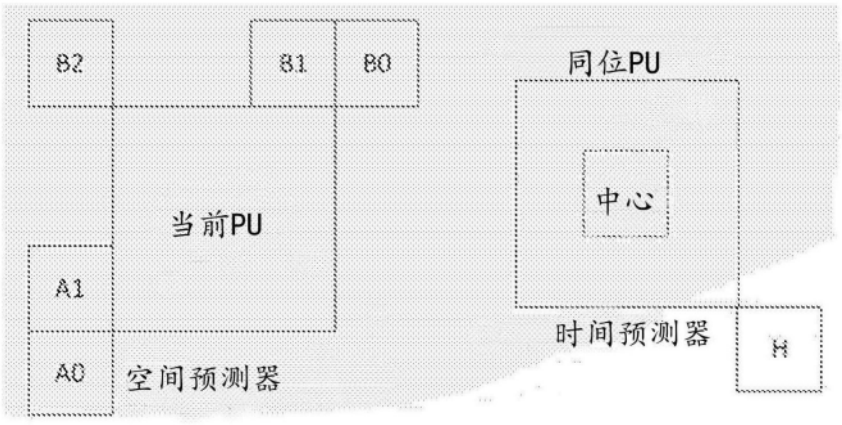

B2
B1
B0
同位PU
当前PU
中心
A1
A0
空间预测器
时间预测器
H

图14

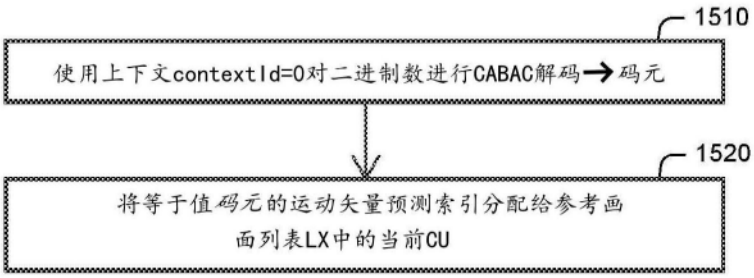

1510
使用上下文contextId=0对二进制数进行CABAC解码→码元
1520
将等于值码元的运动矢量预测索引分配给参考画面列表LX中的当前CU

图15

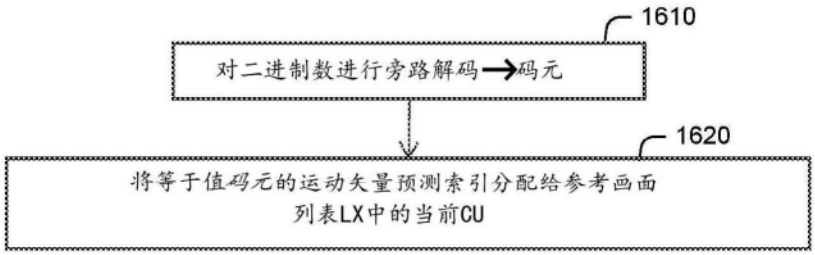

1610
对二进制数进行旁路解码→码元
1620
将等于值码元的运动矢量预测索引分配给参考画面
列表LX中的当前CU

图16

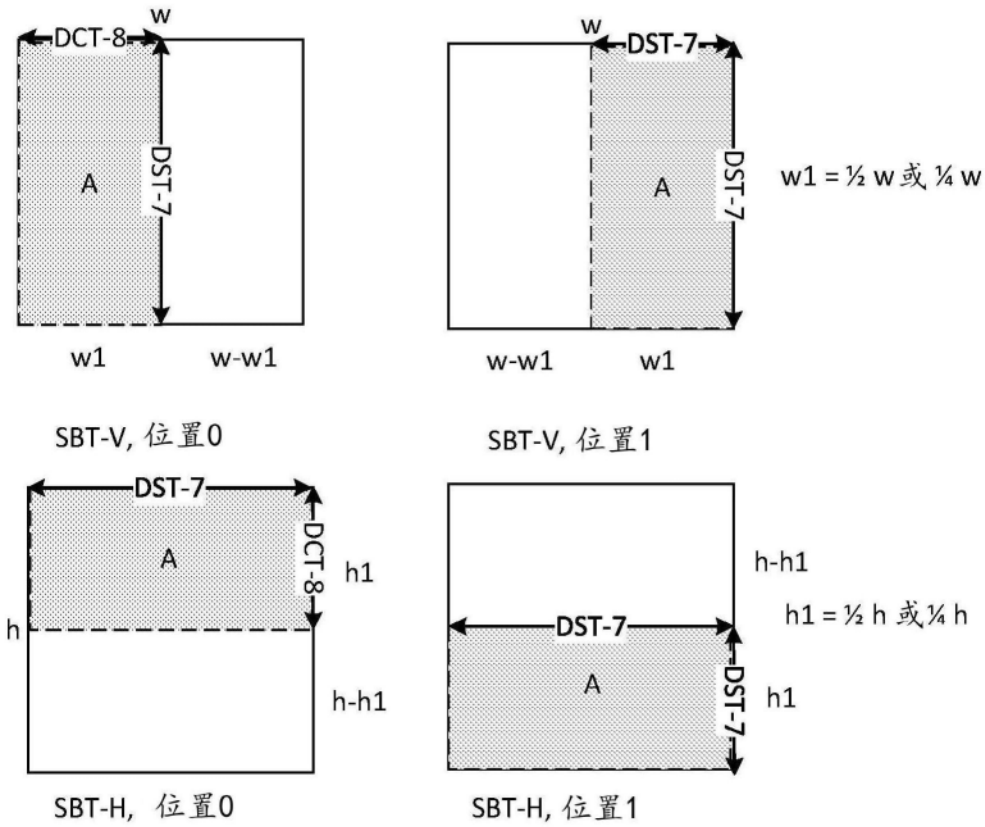

w
DCT-8
A
DST-7
w1
w-w1
SBT-V, 位置0
w
DST-7
A
DST-7
w1 = ½ w 或 ¼ w
w-w1
w1
SBT-V, 位置1
DST-7
A
DCT-8
h1
h
h-h1
SBT-H, 位置0
h-h1
h1 = ½ h 或 ¼ h
DST-7
A
DST-7
h1
SBT-H, 位置1

图17

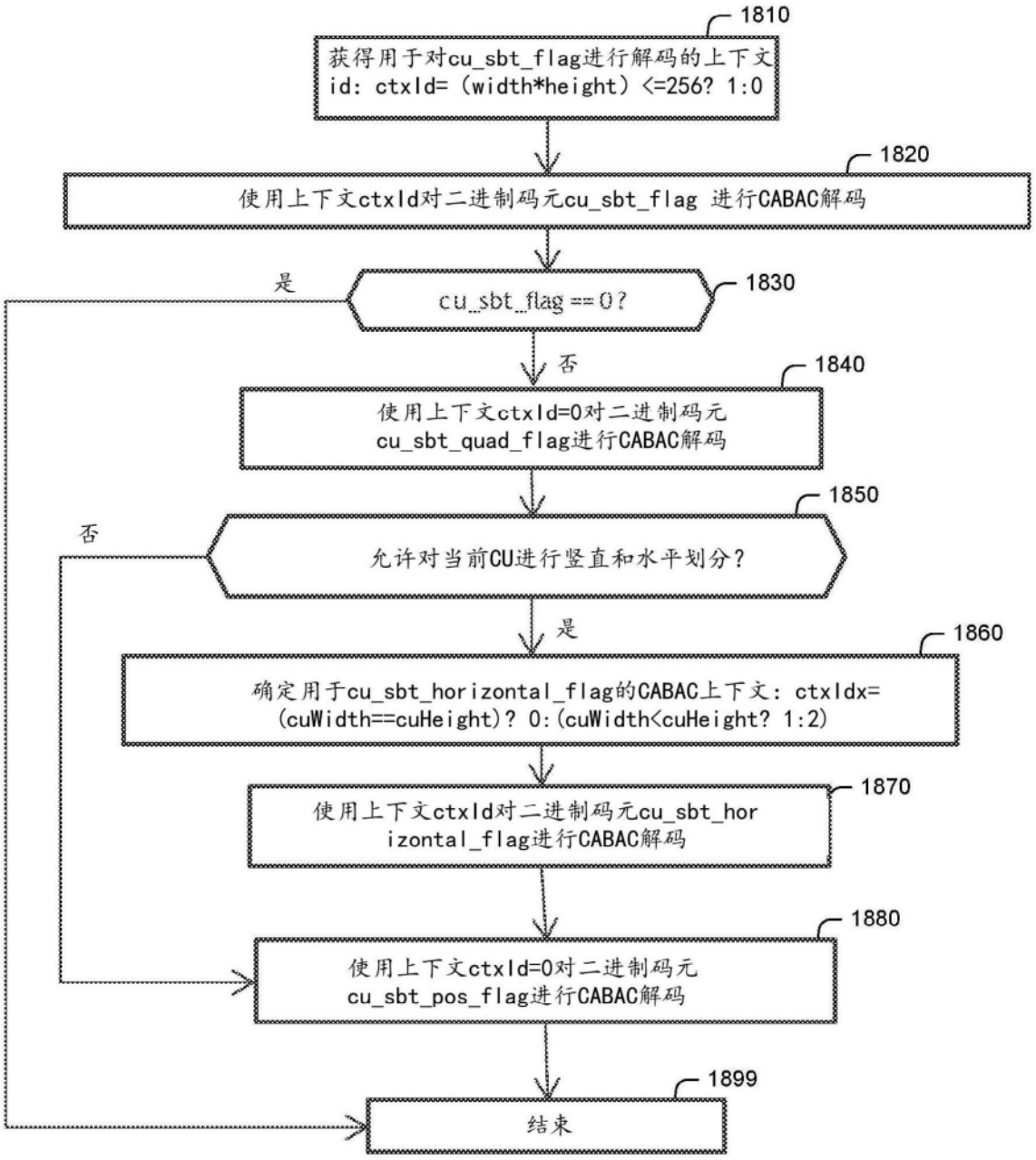

1810
获得用于对cu_sbt_flag进行解码的上下文id: ctxId= (width*height) <=256? 1:0
1820
使用上下文ctxId对二进制码元cu_sbt_flag 进行CABAC解码
是
cu_sbt_flag == 0?
1830
否
1840
使用上下文ctxId=0对二进制码元cu_sbt_quad_flag进行CABAC解码
1850
否
允许对当前CU进行竖直和水平划分?
是
1860
确定用于cu_sbt_horizontal_flag的CABAC上下文: ctxIdx= (cuWidth==cuHeight)? 0:(cuWidth<cuHeight? 1:2)
1870
使用上下文ctxId对二进制码元cu_sbt_horizontal_flag进行CABAC解码
1880
使用上下文ctxId=0对二进制码元cu_sbt_pos_flag进行CABAC解码
1899
结束

图18(现有技术)

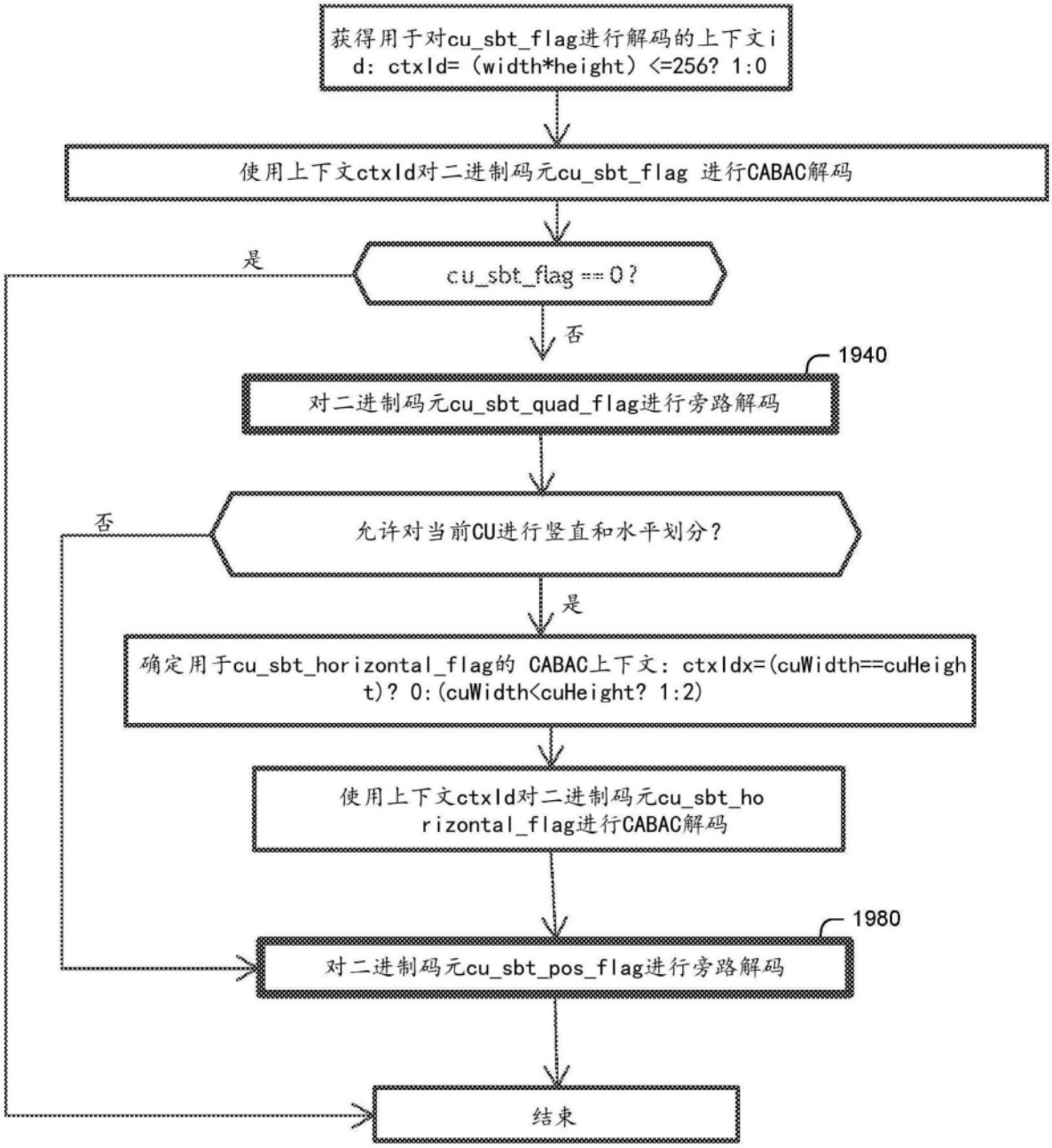

获得用于对cu_sbt_flag进行解码的上下文id: ctxId= (width*height) <=256? 1:0
使用上下文ctxId对二进制码元cu_sbt_flag 进行CABAC解码
是
cu_sbt_flag == 0?
否
1940
对二进制码元cu_sbt_quad_flag进行旁路解码
否
允许对当前CU进行竖直和水平划分?
是
确定用于cu_sbt_horizontal_flag的 CABAC上下文: ctxIdx=(cuWidth==cuHeight)? 0:(cuWidth<cuHeight? 1:2)
使用上下文ctxId对二进制码元cu_sbt_horizontal_flag进行CABAC解码
1980
对二进制码元cu_sbt_pos_flag进行旁路解码
结束

图19A

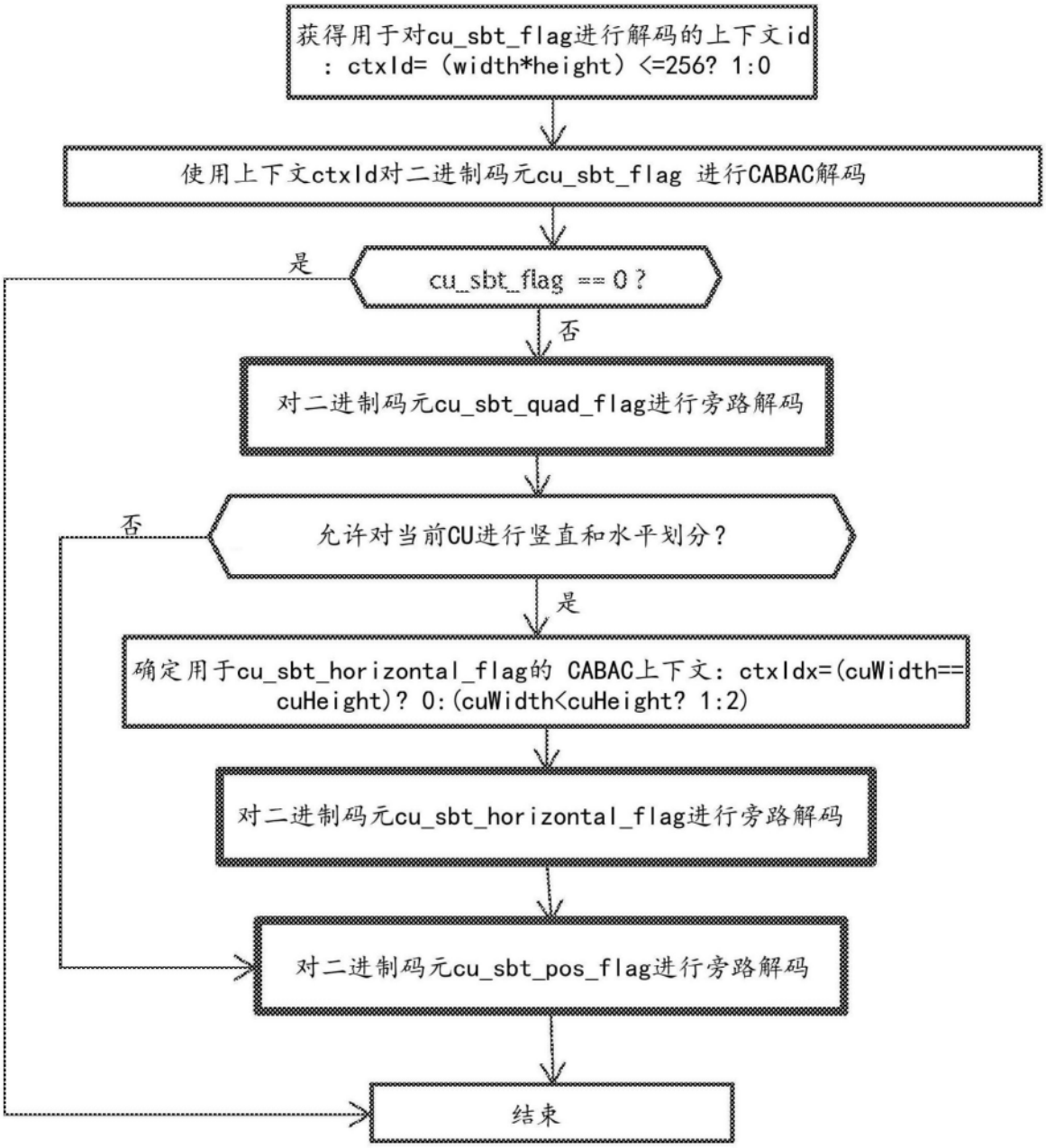

获得用于对cu_sbt_flag进行解码的上下文id
: ctxId= (width*height) <=256? 1:0
使用上下文ctxId对二进制码元cu_sbt_flag 进行CABAC解码
是
cu_sbt_flag == 0?
否
对二进制码元cu_sbt_quad_flag进行旁路解码
否
允许对当前CU进行竖直和水平划分?
是
确定用于cu_sbt_horizontal_flag的 CABAC上下文: ctxIdx=(cuWidth==
cuHeight)? 0:(cuWidth<cuHeight? 1:2)
对二进制码元cu_sbt_horizontal_flag进行旁路解码
对二进制码元cu_sbt_pos_flag进行旁路解码
结束

图19B

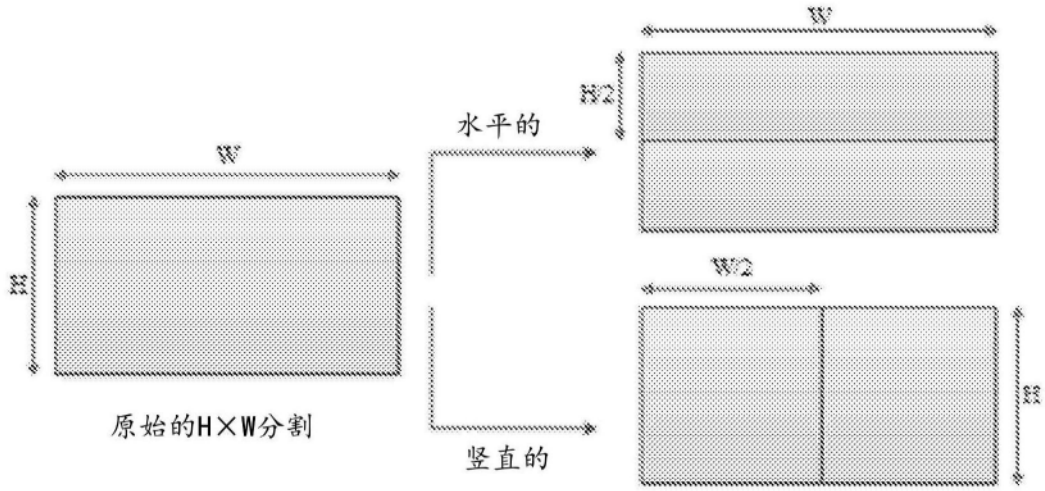


图20A

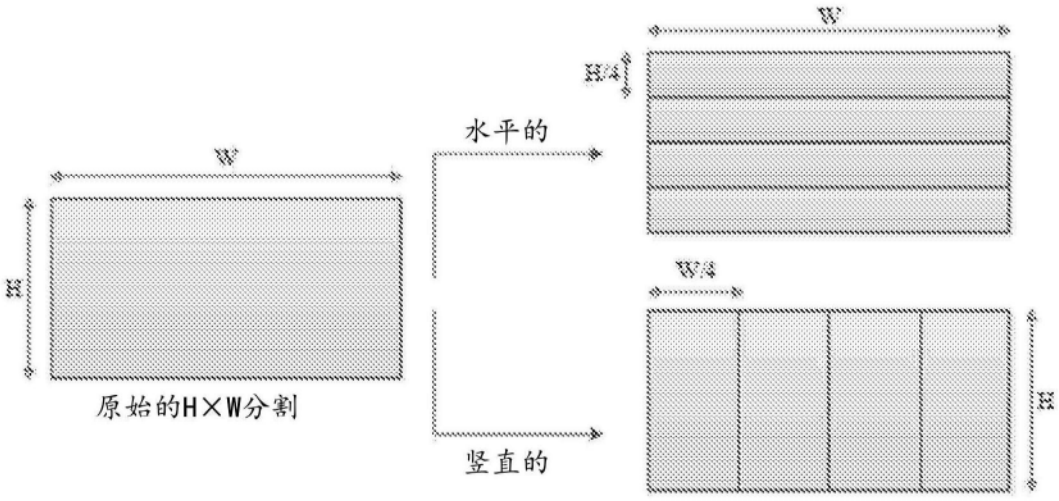


图20B

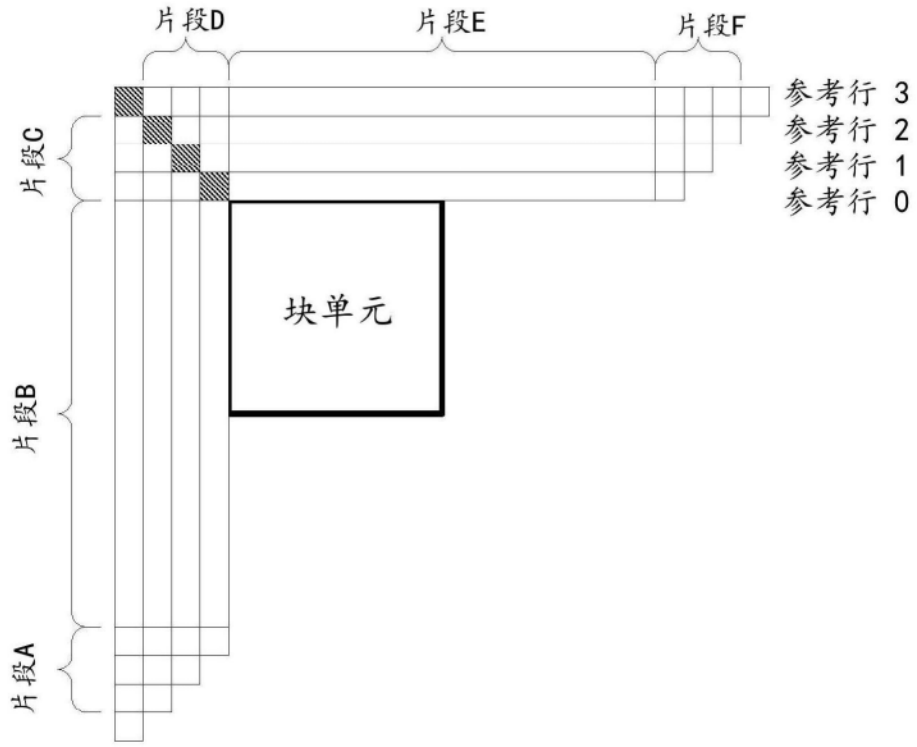


图21

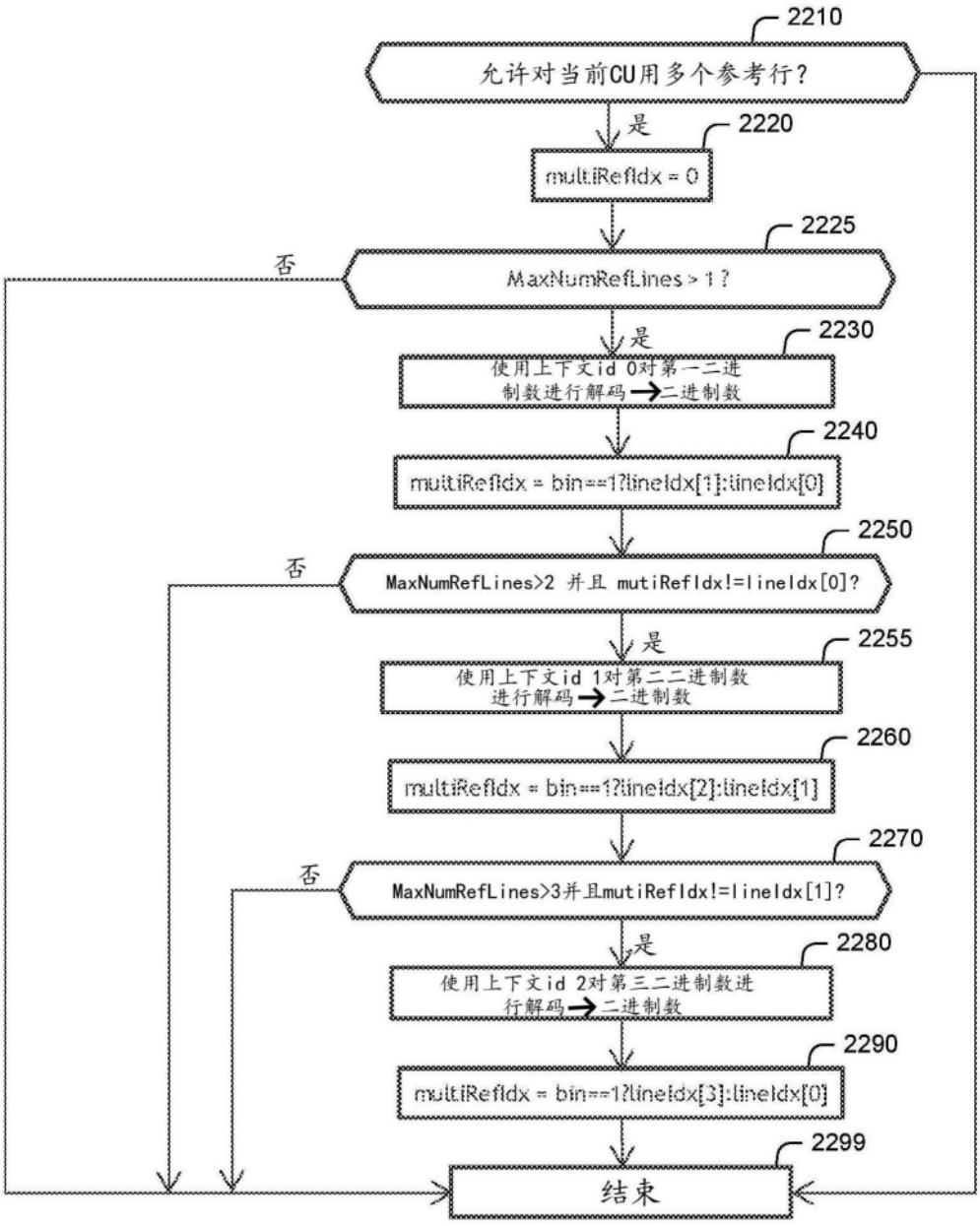

2210
允许对当前CU用多个参考行?
是
2220
multiRefIdx = 0
2225
否
MaxNumRefLines > 1?
是
2230
使用上下文id 0对第一二进制数进行解码→二进制数
2240
multiRefIdx = bin==1?lineIdx[1]:lineIdx[0]
2250
否
MaxNumRefLines>2 并且 mutiRefIdx!=lineIdx[0]?
是
2255
使用上下文id 1对第二二进制数进行解码→二进制数
2260
multiRefIdx = bin==1?lineIdx[2]:lineIdx[1]
2270
否
MaxNumRefLines>3并且mutiRefIdx!=lineIdx[1]?
是
2280
使用上下文id 2对第三二进制数进行解码→二进制数
2290
multiRefIdx = bin==1?lineIdx[3]:lineIdx[0]
2299
结束

图22(现有技术)

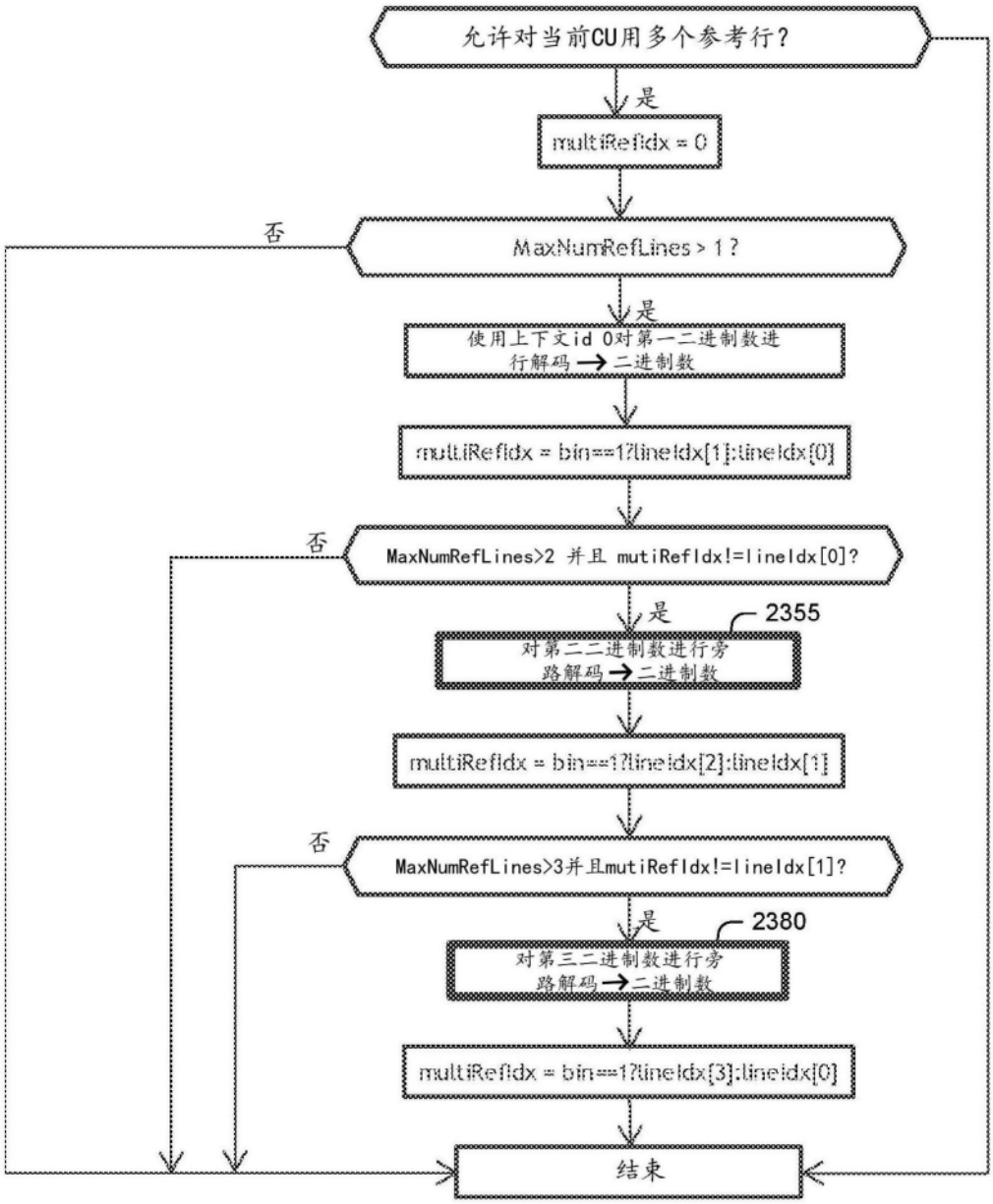

允许对当前CU用多个参考行?
是
multiRefIdx = 0
否
MaxNumRefLines > 1?
是
使用上下文id 0对第一二进制数进行解码 → 二进制数
multiRefIdx = bin==1?lineIdx[1]:lineIdx[0]
否
MaxNumRefLines>2 并且 mutiRefIdx!=lineIdx[0]?
是
2355
对第二二进制数进行旁路解码 → 二进制数
multiRefIdx = bin==1?lineIdx[2]:lineIdx[1]
否
MaxNumRefLines>3并且mutiRefIdx!=lineIdx[1]?
是
2380
对第三二进制数进行旁路解码 → 二进制数
multiRefIdx = bin==1?lineIdx[3]:lineIdx[0]
结束

图23

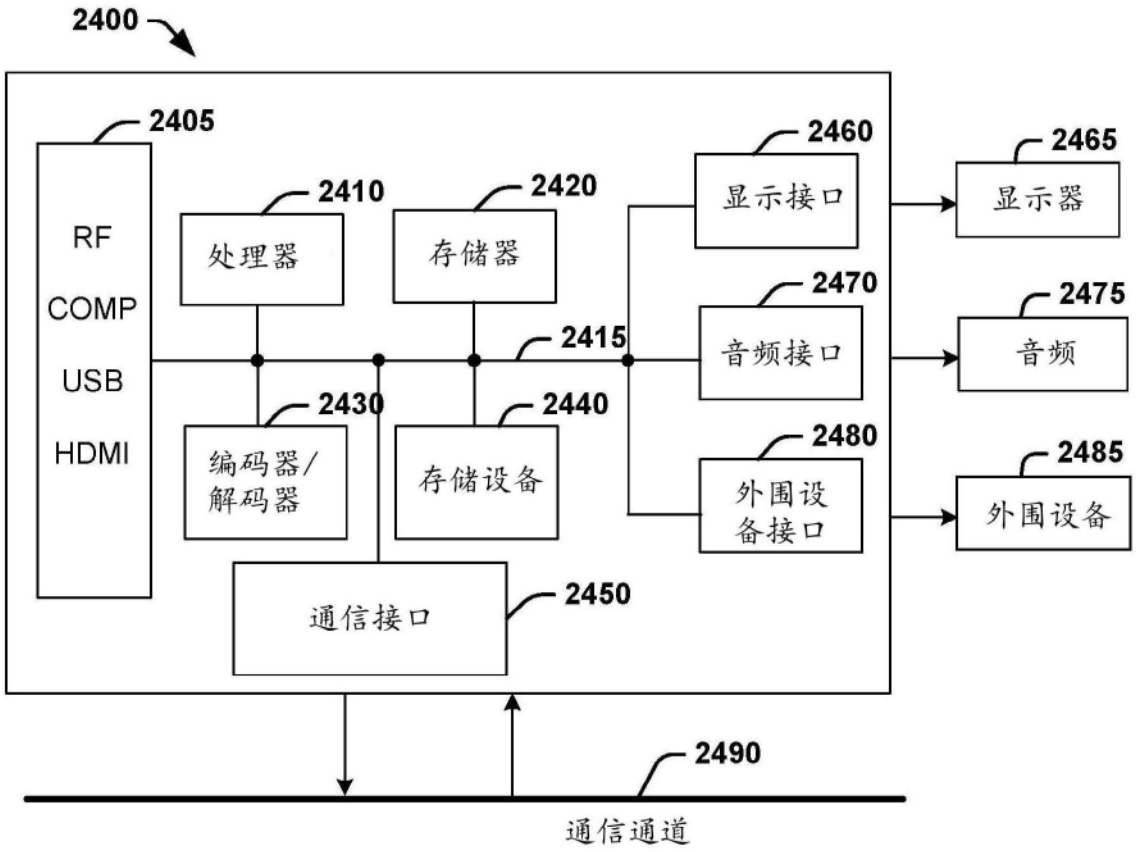

2400
2405
RF
COMP
USB
HDMI
2410
处理器
2420
存储器
2415
2430
编码器/
解码器
2440
存储设备
2450
通信接口
2460
显示接口
2470
音频接口
2480
外围设
备接口
2465
显示器
2475
音频
2485
外围设备
2490
通信通道

图24