

[19] 中华人民共和国国家知识产权局

[51] Int. Cl.

H04S 3/00 (2006.01)
G10L 19/00 (2006.01)



[12] 发明专利申请公布说明书

[21] 申请号 200680024866. X

[43] 公开日 2008年7月9日

[11] 公开号 CN 101218852A

[22] 申请日 2006.7.10

[21] 申请号 200680024866. X

[30] 优先权

[32] 2005.7.11 [33] US [31] 60/697,551

[32] 2005.7.16 [33] KR [31] PCT/KR2005/002290

[32] 2005.7.16 [33] KR [31] PCT/KR2005/002291

[32] 2005.7.16 [33] KR [31] PCT/KR2005/002292

[32] 2005.7.18 [33] KR [31] PCT/KR2005/002306

[32] 2005.7.18 [33] KR [31] PCT/KR2005/002307

[32] 2005.7.18 [33] KR [31] PCT/KR2005/002308

[32] 2005.7.19 [33] US [31] 60/700,570

[86] 国际申请 PCT/KR2006/002688 2006.7.10

[87] 国际公布 WO2007/008010 英 2007.1.18

[85] 进入国家阶段日期 2008.1.8

[71] 申请人 LG 电子株式会社

地址 韩国首尔

[72] 发明人 T·利伯成

[74] 专利代理机构 上海专利商标事务所有限公司

代理人 李 玲

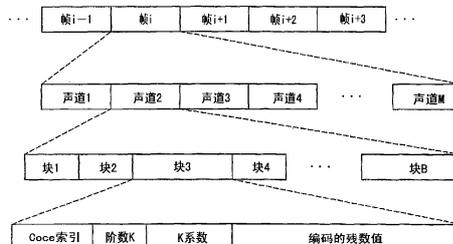
权利要求书 2 页 说明书 29 页 附图 6 页

[54] 发明名称

处理音频信号的装置和方法

[57] 摘要

在一个实施例中，声道映射消息被添加至音频信号的配置信息。所述声道映射消息指示音频信号中的每个声道被映射至再现装置中的哪个扬声器。例如，声道映射信息可包括多个比特，其中每个比特关联于一个扬声器并指示关联的扬声器的音频信号中是否存在声道。



1. 一种处理音频信号的方法，所述方法包括：
将声道映射信息添加至音频信号的配置信息，所述声道映射信息指示音频信号中的每个声道被映射至再现装置中的哪个扬声器。
2. 如权利要求 1 所述的方法，其特征在于，所述声道映射信息包括多个比特，而每个比特与扬声器关联并指示是否有声道存在于已关联的扬声器的音频信号中。
3. 如权利要求 2 所述的方法，其特征在于，所述多个比特为 16 比特。
4. 如权利要求 1 所述的方法，其特征在于，还包括：
将指示符添加于配置信息，所述指示符指示配置信息中是否包含声道映射信息。
5. 如权利要求 1 所述的方法，其特征在于，还包括：
如果声道的音频数据被重置，则将声道重置信息添加于配置信息，所述声道重置信息指示声道的音频数据的重置。
6. 如权利要求 5 所述的方法，其特征在于，还包括：
将指示符添加至配置信息，所述指示符指示配置信息中是否包含声道重置信息。
7. 如权利要求 1 所述的方法，其特征在于，还包括：
将声道信息添加至配置信息，所述声道信息指示音频信号中的声道数。
8. 一种处理音频信号的方法，所述方法包括：
接收具有配置信息和多个声道的音频信号；
从配置信息读取声道映射信息，所述声道映射信息指示音频信号中的每个声道被映射至再现装置中的哪个扬声器；以及
基于声道映射信息处理多个声道。
9. 如权利要求 8 所述的方法，其特征在于，所述声道映射信息包括多个比特，每个比特与扬声器关联并指示已关联的扬声器的音频信号中是否存在声道。
10. 如权利要求 9 所述的方法，其特征在于，所述多个比特为 16 比特。

11. 如权利要求 8 所述的方法，其特征在于，还包括：

从配置信息读取一指示符，所述指示符指示在配置信息中是否包含声道映射信息；以及

基于指示符执行读取声道映射信息的步骤。

12. 如权利要求 8 所述的方法，其特征在于，还包括：

从配置信息读取声道重置信息，所述声道重置信息指示多个声道的音频数据的重置；并且其中

处理声道的步骤基于声道映射信息和声道重置信息处理声道。

13. 如权利要求 12 所述的方法，其特征在于，还包括：

从配置信息读取指示符，所述指示符指示配置信息中是否包含声道重置信息；以及

基于指示符执行读取声道重置信息的步骤。

14. 如权利要求 8 所述的方法，其特征在于，还包括：

从配置信息读取声道信息，所述声道信息指示音频信号中的声道数；并且其中

处理声道的步骤基于声道映射信息和声道信息处理声道。

15. 一种处理音频信号的装置，包括：

编码器，所述编码器被配置成将声道映射信息添加至音频信号的配置信息，所述声道映射信息指示音频信号中的每个声道被映射至再现装置中的哪个扬声器。

16. 一种处理音频信号的装置，包括：

解码器，被配置成接收具有配置信息和多个声道的音频信号，并从配置信息读取声道映射消息，所述声道映射消息指示音频信号中的每个声道被映射到再现装置中的哪个扬声器；以及

解码器，被配置成基于声道映射信息处理诸声道。

处理音频信号的装置和方法

技术领域

本发明涉及一种处理音频信号的方法，更具体地涉及一种编码和解码音频信号的方法和装置。

背景技术

过去曾经以不同方法实现了音频信号的存储和重放。例如，音乐和语音业已通过留声技术（例如唱盘播放机）、磁技术（例如卡式磁带）和数字技术（例如光盘）来记录和保存。随着音频存储技术的发展，需要克服许多难题来优化音频信号的质量和可存储性。

为了音乐信号的存档和宽带传输，无损重建成为比借助诸如 MP3 或 AAC 等在 MPEG 标准中定义的感知编码高效率压缩更为重要的特征。虽然 DVD 音频和超级 CD 音频包括专用的无损压缩方案，但是在内容持有者和广播公司当中需要一种开放和综合性的压缩方案。响应于这种需要，一种新的无损编码方案已经成为 MPEG-4 音频标准的延伸。无损音频编码法由于原始信号的完美重建而实现了没有任何质量损失的数字音频数据压缩。

发明内容

本发明涉及处理音频信号的方法。

在一个实施例中，声道映射信息被加至音频信号的配置信息。该声道映射信息指示音频信号中的每个声道被映射到再现装置中的哪个扬声器。例如，声道映射信息包括多个比特，其中每个比特关联于一个扬声器并指示在关联的扬声器的音频信号中是否存在声道。

在另一实施例中，将指示比特位添加于配置信息以指示配置信息中是否包含声道映射信息。

在又一实施例中，具有配置信息和多个声道的音频信号被接收。声道映射

信息被读取自配置信息。声道映射信息指示音频信号中的每个声道被映射至再现装置中的哪个扬声器。基于声道映射信息处理诸个声道。例如，声道映射信息可包括多个比特，每个比特关联于一个扬声器并指示在关联的扬声器的音频信号中是否存在声道。

本发明还涉及编码音频信号的方法和装置以及解码音频信号的方法和装置。

附图说明

包括于此以提供对本发明的进一步理解，并被结合在本申请内且构成其一部分的附图示出本发明的实施例，其与说明书一起用来解释本发明的原理。在附图中：

图 1 是根据本发明一个实施例的编码器的示例图。

图 2 是根据本发明一个实施例的解码器的示例图。

图 3 是根据本发明一个实施例的压缩的 M 声道文件的比特流结构的示例图。

图 4 是根据本发明一个实施例的分层块切换方法的理念图的示例图。

图 5 是块切换例和相应的块切换信息编码的示例图；

图 6 是根据本发明实施例的多个声道的块切换方法的示例图。

具体实施方式

下面将详细参考本发明的较佳实施例，其具体例子示于附图中。不管在何处，所有附图中相同的参考标号被用来表示相同或相似的部分。

在阐述本发明前，应当注意本发明中公开的多数术语对应于业内公知的通常术语，但某些术语已由申请人根据需要选择并且将在本发明后面的描述中予以公开。因此，基于它们在本发明中的含义理解由申请人定义的术语是较佳的。

在无损音频编码方法中，由于编码进程必须可以完美逆转而没有信息损失，因此编码器和解码器两者的若干部件必须以确定性方式予以实现。

编解码器结构

图 1 是根据本发明的编码器 1 的示例图。

分割部件 100 将输入的音频数据分割成若干帧。在一个帧中，每个声道还可进一步被细分成若干音频采样块以做进一步处理。缓存器 110 存储由分割部件 100 分割后的块和/或帧采样。

系数估算部件 120 估算每个块的最佳的系数值组。系数的数量，即预测器的阶数也可自适应地作出选择。系数估算部件 120 计算数字音频数据块的一组部分自相关系数（parcor）值。部分自相关系数值指示预测器系数的部分自相关系数表征。量化部件 130 将该组部分自相关系数值量化。

第一熵编码部件 140 通过将部分自相关系数值减去偏移值而计算部分自相关系数残数值，并使用由熵参数定义的熵代码对部分自相关系数的残数值进行编码，其中偏移值和熵参数选自最佳表。最佳表选自基于数字音频数据块的采样速率的多个表。分别对于多个采样速率范围预定义多个表以实现发送的数字音频数据的最佳压缩。

系数转换部件 150 将量化的部分自相关系数转换成线性的预测编码(LPC)系数。预测器 160 使用线性预测编码系数从存储在缓存器 110 中的先前原始采样中估算当前预测值。减法器 170 使用存储在缓存器 110 中的数字音频数据的原始值和预测器 160 中估算的预测值计算数字音频数据块的预测残数。

第二熵编码部件 180 使用不同的熵代码编码预测残数并生成代码索引。所选择的代码索引作为辅助信息被发送。第二熵代码部件 180 使用具有不同复杂度的两种选择性编码技术编码预测残数。一种编码技术是公知的 Golomb-Rice 编码（在下文中简称为“Rice 码”）方法而另一种是公知的分组 Gilbert-Moore 码（在下文中简称为“BGMC”）方法。Rice 码具有低复杂度但仍然是高效率的。BGMC 算法编码方案以相比 Rice 码复杂度稍高的代价提供更好的压缩。

最后，多路复用部件 190 将编码的预测残数、代码索引、编码的部分自相关系数残数值和其它附加信息多路复用以形成压缩的比特流。编码器 1 还提供循环冗余校验（CRC）校验和，它主要被提供给解码器以校验解码的数据。在编码器方面，CRC 用来保证压缩的数据能被无损地解码。

其它编码选项包括柔性块切换方案、随机存取和联合声道编码。编码器 1 可使用这些选项提供若干具有不同复杂度的压缩级别。联合声道编码利用立体

声声道或多声道信号之间的相关性。这可通过在某些段中编码两声道之间的差来实现，在这些段中能够比原始声道之一更为有效地编码这种差。这些编码选项将在参数根据本发明的示例性解码器的说明之后更为详细地予以说明。

图 2 是根据本发明的解码器 2 的示例图。更详细地说，图 2 示出由于不必执行任何调整因而复杂度明显低于编码器的无损音频信号解码器。

多路分解部件 200 接收音频信号并将数字音频数据块的编码的预测残数、代码索引、编码的部分自相关系数残数值和其它附加信息多路分解。第一熵解码部件 210 使用由熵参数定义的熵代码以解码部分自相关系数残数值并通过将偏移值加上解码的部分自相关系数残数值计算一组部分自相关系数值；其中偏移值和熵参数被选自一个表，该表是由解码器从基于数字音频数据块的采样速率的多个表中选择的。第二熵解码部件 220 使用代码系数解码多路分解的编码的预测残数。系数转换部件 230 将熵解码的自相关系数值转换成 LPC 系数。预测器 240 使用 LPC 系数估算数字音频数据块的预测残数。加法器 250 将解码的预测残数与估算的预测残数相加以获得数字音频数据的原始块。组装部件 260 将解码的块数据组装成帧数据。

因此，解码器 2 将编码的预测残数和自相关系数残数解码，将自相关系数残数值转换成 LPC 系数，并运用逆预估滤波器以计算无损重构信号。解码器 2 的计算强度取决于由编码器 1 选择的预测阶数。在多数情形下，实时解码即使在低端系统中也是可行的。

图 3 是根据本发明的包括多个声道的（例如 M 声道）压缩音频信号的比特流结构的示例图。

比特流由包括多个声道（例如 M 声道）的至少一个音频帧构成。比特流配置语法（见下面的表 6）中的“channels”字段指示声道数。使用根据本发明的块切换方案将每个声道再分成多个块，这将在后面详细说明。每个再分块具有不同的大小并包括根据图 1 编码的编码数据。例如，一个再分块中的编码数据包含代码索引、预测阶数 K、预测器系数和编码的残数值。如果使用声道对之间的联合编码，则所有声道的块分割是相同的，并且这些块以交织方式被存储。比特流配置语法（表 6）中的“js_stereo”字段指示联合立体声（声道差）是开启的还是关闭的，而 frame_data 语法（见下面的表 7）中的“js_switch”字段

指示是否选择联合立体声（声道差）。否则，每个声道的块分割是独立的。

下面将详细参照附图和后面的语法详细说明前面提到的块切换、随机存取、预测和熵编码选项。

块切换

本发明的一个方面涉及在使用实际编码方案前将每个声道再分成多个块。下面，根据本发明的块分割（或再分）方法被称为“块切换方法”。

分层块切换

图 4 是根据本发明的分层块切换方法的理念图的示例图。例如，图 4 示出将一个声道按分层再分成 32 个块的方法。当在单个帧中提供多个声道时，每个声道被再分（或分割）成高达 32 个块，并且每个声道的再分块构成一个帧。因此，根据本发明的块切换方法由图 1 所示的分割部件 100 执行。此外，如上所述，预测编码和熵编码执行于再分块单元。

总的来说，传统音频无损编码（ALS）包括相对简单的块切换方案。N 个采样的每个声道或者使用一个全长块（ $N_B=N$ ）或者四个长度块 $N_B=N/4$ （例如 1: 4 切换）编码，其中相同的块分割应用于所有声道。在某些情形下，该方案具有某些局限性。例如，尽管只有 1: 1 或 1: 4 切换是可行的，然而不同的切换（例如 1: 2、1: 8 及其组合）在某些情形下是更有效的。另外在传统 ALS 中，对所有声道以相同方式执行切换，尽管不同声道可能得益于不同的切换（如果声道不是相关的则尤为如此）。

因此，根据本发明实施例的块切换方法提供相对灵活的块切换方案，其中一个帧的每个声道按分层被再分成多个块。例如，图 4 示出能分分层地再分成高达 32 个块的声道。在根据本实施例的一个声道中， $N_B=N$ 、 $N/2$ 、 $N/4$ 、 $N/8$ 、 $N/16$ 和 $N/32$ 的任意块组合是可行的，只要每个块是通过双倍长度的高层块的再分产生的。例如，如图 4 中的例子所示，尽管至 $N/4+N/2+N/4$ 的分割是不可行的，然而至 $N/4+N/4+N/2$ 的分割是可行的（例如下面在图 5(e)和图 5 中示出的块切换实例）。换句话说，声道被分成多个块以使每个块的长度等于 $N/(m^i)$ 中的一个值，其中 $i=1, 2, \dots, p$ ，在这里 N 是声道的长度，m 是大于或等于 2

的一个整数，而 p 代表在再分分层中的层数。

因此，在本发明的实施例中，比特流包括指示块切换层的信息以及指示块切换结果的信息。这里，与块切换相关的信息被包含在用于解码处理的语法中，这将在下面进行描述。

例如，作出设定以使块切换处理后产生的最小块尺寸为 $N_B = N/32$ 。然而，这种设定仅为简化本发明说明的一个实例。因此，根据本发明的设定不局限于这种设定。

更具体地说，当最小块大小为 $N_B = N/32$ 时，这表示块切换处理已按分层进行了 5 次，故将其称为 5 层块切换。或者，当最小块大小为 $N_B = N/16$ 时，这表示块切换处理已按分层执行了 4 次，故将其称为 4 层块切换。同样，当最小块大小为 $N_B = N/8$ 时，这表示块切换处理已按分层执行了 3 次，故将其称为 3 层块切换。而当最小块大小为 $N_B = N/4$ 时，这表示块切换处理已按分层执行了 2 次，故将其称为 2 层块切换。当最小块大小为 $N_B = N/2$ 时，这表示块切换处理已按分层执行了 1 次，故将其称为 1 层块切换。最后，当最小块大小为 $N_B = N$ 时，这表示尚未执行块切换处理，故将其称为 0 层块切换。

在本发明的实施例中，指示块切换层的信息被称为第一块切换信息。例如，第一块切换信息可由表 6 中的语法中的 2 比特字段“`block_switching`”表示，这将在后面的处理中予以说明。更具体地说，“`block_switching=00`”表示层 0，“`block_switching=01`”表示层 1 至层 3 中的任何一个，“`block_switching=10`”表示层 4，而“`block_switching=11`”表示层 5。

另外，指示根据上述块切换层对每个分层执行的块切换结果的信息在这些实施例中被称为第二块切换信息，这里，第二块切换信息可由“`bs_info`”字段表示，该字段在表 7 所示的语法中以 8 比特、16 比特和 32 比特中的任何一个表示。更具体地说，如果“`block_switching=01`”（表示层 1 至层 3 的任何一个），则“`bs_info`”由 8 比特表示。如果“`block_switching=10`”（表示层 4），则“`bs_info`”由 16 比特表示。换句话说，高达 4 层的块切换结果可用 16 比特表示。此外，如果“`block_switching=11`”（表示层 5），则“`bs_info`”表示为 32 比特。换句话说，高达 5 层的块切换结果可用 32 比特指示。最后，如果“`block_switching=00`”（表示尚未进行块切换），则不发送“`bs_info`”。这

表示一个声道构成一个块。

分配给第二块切换信息的总比特数是基于第一块切换信息的层值而确定的。这可能会减小最终的比特率。在下面的表 1 中简述第一块切换信息和第二块切换信息之间的关系。

表 1: 块切换层

最大层数	最小 N_B	“bs_info” 的字节数
0 (“block_swithing=00”)	N	0
1 (“block_swithing=01”)	N/2	1 (=8 比特)
2 (“block_swithing=01”)	N/4	1 (=8 比特)
3 (“block_swithing=01”)	N/8	1 (=8 比特)
4 (“block_swithing=10”)	N/16	2 (=16 比特)
5 (“block_swithing=11”)	N/32	4 (=32 比特)

下面，将详细描述配置（或映射）第二块切换信息（bs_info）中每个比特的的方法的一个实施例。

bs_info 字段根据上述实施例可包括高达 4 个字节。关于层 1 至层 5 的比特映射可以是[(0)1223333 44444444 55555555 55555555]。可预留第一比特以指示独立或同步块切换，这将在后面的独立/同步块切换章节更为详细地说明。图 5(a)–5(f)示出一个声道的不同块切换实例，其中发生 3 层块切换。因此，在这些实例中，最小块长度为 $N_B=N/8$ ，并且 bs_info 由一个字节构成。从最大块长度 $N_B=N$ 开始，如果块被进一步再分，则设定 bs_info 比特。例如，在图 5(a)中，根本不存在再分，因此“bs_info”为(0)000 0000。在图 5(b)中，帧被再分((0)1……)而长度为 N/2 的第二块被进一步分成((0)101……)两个长度块 N/4；因此 bs_info 为(0)1010 0000。在图 5(c)中，帧被再分((0)1……)，并且只有长度 N/2 的第一块被进一步分割成((0)110……)长度为 N/4 的两个块；因此“bs_info”为(0)1100 0000。在图 5(d)中，帧被再分((0)1……)，长度为 N/2 的第一块和第二块被进一步分割((0)111……)成长度为 N/4 的两个块，并且只有长度为 N/4 的第二块被进一步分割((0)11101……)成长度为 N/8 的两个块；因此“bs_info”为(0)111 0100。

如上所述，图 5(e)和 5(f)中的实例表示不允许块切换的情形，这是因为图 5(e)中的 $N/2$ 块和图 5(f)中的第一个 $N/4$ 块无法通过再分前一层块而获得。

独立/同步块切换

图 6(a)–6(c)是根据本发明实施例的块切换的示例图。

更具体地说，图 6(a)示出尚未对块 1、2 和 3 执行块切换的实例。图 6(b)示出一个实例，其中两个声道（声道 1 和 2）构成一个声道对，并且在声道 1 和声道 2 中同步地执行块切换。交织也被应用于本例中。图 6(c)示出一个实例，其中两个声道（声道 1 和 2）构成一个声道对，并且声道 1 和声道 2 的块切换独立地进行。这里，声道对指两个任意的音频声道。关于哪些声道被分组声道对的判决可由编码器自动作出或由用户手动作出（例如 L 和 R 声道、Ls 和 Rs 声道）。

在独立块切换中，尽管在所有声道中每个声道的长度可以是相同的，然而可对每个声道独立地执行块切换。即，如图 6(c)所示，声道以不同方式被分成两个块。如果一个声道对的两个声道彼此相关并且使用差编码，则声道对的两个声道被同步地切换。在同步块切换中，声道以相同方式被块切换（即被分成若干个块）。图 6(b)示出这样的实例，并进一步示出这些块可以被交织。如果声道对的两个声道不彼此相关，则差编码是无益的，因此不需要同步地切换声道。相反，更适合独立地切换声道。此外，根据本发明的另一实施例，可将所述独立或同步块切换方法应用于声道数大于或等于 3 的多声道组。例如，如果多声道组的所有声道彼此相关，则可以同步切换多声道组的所有声道。另一方面，如果多声道组的所有声道不彼此相关，则可以独立切换多声道组的每个声道。

此外，“bs_info”字段被用作指示块切换结果的信息。另外，“bs_info”字段也被用作指示对构成声道对的每个声道是否已独立执行或同步执行块切换的信息。在这种情形下，如上所述，可使用“bs_info”字段中的特定比特（例如第一比特）。例如，如果声道对的两个声道彼此独立，则“bs_info”字段的第一比特被置为“1”。另一方面，如果声道对的两个声道彼此同步，则“bs_info”字段的第一比特被置为“0”。

下面，将详细说明图 6(a)、6(b)和 6(c)。

参照图 6(a)，由于没有声道执行块切换，因此不产生关联的“bs_info”。

参照图 6(b)，声道 1 和 2 构成一个声道对，其中两个声道彼此同步并且同步执行块切换。例如，在图 6(b)中，声道 1 和声道 2 被分割成长度为 $N/4$ 的若干块，这些块都具有相同的 bs_info “bs_info=(0)101 0000”。因此，对每个声道对发送一个“bs_info”，这导致比特率降低。此外，如果声道对是同步的，则声道对中的每个块需要彼此交织。这种交织是有益的（或有利的）。例如，一个声道对中的一个声道的块（例如图 6(b)中的块 1.2）依赖于来自两声道的之前的块（例如图 6(b)中的块 1.1 和 2.1），并因此这些之前的块可在当前块之前获得。

参照图 6(c)，声道 1 和 2 构成一个声道对。然而，在本例中，块切换是独立执行的。更具体地说，声道 1 被分割成大小（或长度）高达 $N/4$ 的块并具有“bs_info=(1)101 0000”的 bs_info。声道 2 被分割成大小高达 $N/2$ 的块并具有“bs_info=(1)100 0000”的 bs_info。在图 6(c)所示例子中，在每个声道间独立地进行块切换，并因此不执行块之间的交织处理。换句话说，对于具有独立切换的块的声道，可独立地配置声道数据。

联合声道编码

联合声道编码——也被称为联合立体声——利用立体声信号的两声道之间或多声道信号的任何两个声道之间的相关性。尽管是独立地直接处理两声道 $x_1(n)$ 和 $x_2(n)$ ，然而利用声道之间相关性的简单方法是编码差信号：

$$d(n) = x_2(n) - x_1(n)$$

每个块中的 $x_1(n)$ 、 $x_2(n)$ 和 $d(n)$ 之间的切换可通过比较各信号根据哪两个信号能被最有效率地编码来实现，而不是 $x_1(n)$ 或 $x_2(n)$ 。这种用切换的差编码实现的预测在两个声道彼此非常相似的情形下是有利的。在多声道材料的情形下，可通过编码器重新配置声道以分配适当的声道对。

除了简单的差编码，无损音频编解码也支持更为复杂的方案以利用多声道信号的任意声道之间的声道间冗余。

随机存取

本发明涉及音频无损编码并能够支持随机存取。随机存取支持对已编码的音频信号任何部分的快速存取而不浪费地解码之前的部分。这是采用压缩数据的查找、编辑或流传送的场合下的一个重要特征。为了实现随机存取，在随机存取单元中，编码器需要插入一个帧，该帧能被解码而无需解码之前的帧。插入帧被称为“随机存取帧”。在该随机存取帧中，没有来自之前的帧的采样被用于预测。

下面，将详细说明根据本发明的随机存取的信息。参照配置语法（图 6 所示），关联于随机存取的信息作为配置信息被发送。例如，“random_access”字段被用作指示是否允许随机存取的信息，它由 8 比特表示。此外，如果允许随机存取，则 8 比特“random_access”字段指定构成随机存取单元的帧数。例如，当“random_access=0000 0000”时，不支持随机存取。换句话说，当“random_access>0”时，则支持随机存取。更具体地说，当“random_access=0000 0001”时，这指示构成随机存取单元的帧数为 1。这表示在所有的帧单元中均允许随机存取。此外，当“random_access=1111 1111”时，这指示构成随机存取单元的帧数为 255。因此，“random_access”信息对应于当前随机存取单元中的随机存取帧和下一随机存取单元中的随机存取帧之间的距离。这里，所述距离由帧数表达。

32 比特的“ra_unit_size”字段被包含在比特流中并被发送。这里，“ra_unit_size”字段表示以字节为单位的随机存取单元的大小，并因此表示以字节为单位的从当前随机存取帧至下一随机存取帧的距离。“ra_unit_size”字段或者包含在配置语法（表 6）中或者包含在帧—数据语法（表 7）中。配置语法（表 6）还包括指示比特流中存储“ra_unit_size”信息的位置的信息。该信息被表示为 2 比特的“ra_flag”字段。更具体地说，例如，当“ra_flag=00”时，这表示“ra_unit_size”信息不被存储在比特流中。当“ra_flag=01”时，这表示“ra_unit_size”信息被存储在比特流的帧数据语法（表 7）中。此外，当“ra_flag=10”时，“ra_unit_size”信息被存储在比特流的配置语法（表 6）。如果“ra_unit_size”信息被包含在配置语法中，这表示“ra_unit_size”信息只在比特流上发送一次并且被均等地作用于所有随机存取单元。或者，如果帧数

据语法中包含“ra_unit_size”信息，这表示当前随机存取单元中的随机存取帧和下一随机存取单元中的随机存取帧内之间的距离。因此，由于距离可以改变，对比特流中的每个随机存取单元发送“ra_unit_size”信息。

因此，配置语法（表 6）中的“random_access”字段也被称为第一通用消息。另外，“ra_flag”字段也被称为第二通用消息。在本发明的这个方面中，音频信号包括配置信息和多个随机存取单元，每个随机存取单元包括一个或多个音频数据帧，所述音频数据帧中的一个为随机存取帧，其中配置信息包括：指示诸帧中的两相邻随机存取帧之间的距离的第一通用信息；以及指示哪里存储每个随机存取单元的随机存取单元大小信息的第二通用信息。随机存取单元大小信息以字节表示两相邻随机存取单元之间的距离。

或者，在本发明的这个方面，一种解码音频信号的方法包括：接收具有配置信息和多个随机存取单元的音频信号，每个随机存取单元包含一个或多个音频数据帧，所述音频数据帧中的一个为随机存取帧；从配置信息读取第一通用信息，所述第一通用信息指示诸帧中两相邻随机存取帧之间的距离；并从配置信息读取第二通用信息，所述第二通用信息指示每个随机存取单元的随机存取大小信息被存储在哪里，而随机存取单元大小信息以字节表示两相邻随机存取帧之间的距离。解码器随后访问随机存取单元大小信息并使用该信息以及第一和第二全局信息以执行对音频信号中的音频数据的随机存取。

声道配置

如图 3 所示，音频信号包括根据本发明的多声道信息。例如，每个声道可与音频扬声器的位置形成一一对应的映射。配置语法（下面的表 6）包括声道配置信息，它被表示为 16 比特的“chan_config_info”字段和 16 比特的“channels”字段。“chan_config_info”字段包括将声道映射到扬声器位置的信息而 16 比特的“channels”字段包括指示声道总数的信息。例如，当“channels”字段等于“0”时，这表示声道对应于单声道。当“channels”字段等于“1”时，这表示声道对应于立体声声道中的一个。另外，当“channels”字段等于或大于“2”时，这表示声道对应于多声道中的一个。

下面的表 2 示出构成“chan_config_info”字段的每个比特以及与之对应的

每个声道的实例。更具体地说，当发送的比特流中存在相应声道的情形下，“chan_config_info”字段中的相应比特被置为“1”。或者，当发送的比特流中不存在相应声道时，“chan_config_info”字段中的相应比特被置为“0”。本发明还包括指示配置语法（表 6）中是否存在“chan_config_info”字段的信息。该信息被表示为 1 比特的“chan_config”标志。更具体地说，“chan_config = 0”指示“chan_config_info”字段不存在。而“chan_config = 1”指示“chan_config_info”字段存在。因此，当“chan_config = 0”时，这表示在配置语法（表 6）中最近没有定义“chan_config_info”字段。

表 2

扬声器位置	缩写	chan_config_info 中的比特位置
左	L	1
右	R	2
左后	Lr	3
右后	Rr	4
左侧	Ls	5
右侧	Rs	6
中央	C	7
中央后方/环绕	S	8
低频效果	LFE	9
左向下混频	L0	10
右向下混频	R0	11
单声混频	M	12
(预留)		13-16

此外，声道配置会使相互关联的声道在音频数据的结构中或音频信号中不彼此毗邻。因此，重置声道数据以使相互关联的声道毗邻是有利的。重置声道的方法可在配置信息（表 6）的 chan_pos 字段中给出。此外，配置信息（表 6）可包括指示 chan_pos 字段的声道重置信息是否存在的 chan_sort 字段。

基于读取上述的各种声道信息，解码器处理音频信号以使正确的声道被送至正确的扬声器。

帧长度

如图 3 所示，根据本发明的音频信号包括多个声道或多声道。因此，当执行编码时，关于构成一个帧的多声道的数目的信息以及关于每个声道的采样数的信息被插入到比特流中并被发送。参照配置语法（表 6），32 比特的“samples”字段被用作指示构成每个声道的音频数据采样总数的信息。此外，16 比特的“frame_length”（帧长度）字段被用作指示相应帧中的每个声道的采样数的信息。

此外，“frame_length”字段的 16 比特的值是通过由编码器使用的值确定的，并且被称为用户定义值。换句话说，用户定义值不是固定值，而是在编码处理时任意确定的值。例如，该值可由编码处理的用户设定。

因此，在解码处理中，当通过图 2 所示的多路分解部件 200 接收比特流时，应首先获得每个声道的帧数。该值是根据下面所示的算法得到的。

```

frame=samples/frame_length;
rest=samples%frame_length;
if (rest)
{
    frame++;
    frlen_last=rest;
}
else
    frlen_last=frame_length;

```

更具体地说，每个声道的总帧数是通过将经由比特流发送的“samples”字段确定的每个声道的采样总数除以由“frame_length”字段确定的每个声道的一个帧中的采样数而计算得到的。例如，当由“samples”字段确定的采样总数恰好为由“frame_length”字段确定的每个帧中的采样数的倍数时，则该倍数成为帧总数。然而，如果由“samples”字段确定的采样总数不恰好是由“frame_length”字段确定的采样数的倍数并且存在余数（或剩余数），则总帧数比倍数增加“1”。此外，最末帧（frlen_last）的采样数被确定为余数（或

剩余数)。这表示仅最末帧的采样数与其之前帧不同。通过如上所述地定义编码器和解码器之间标准化的规则，编码器可自由地确定并发送每个声道的采样总数 (“samples” 字段) 以及每个声道的一个帧中的采样数 (“frame_length” 字段)。此外，解码器可通过使用发送信息上的上述算法准确地确定用于解码的每个声道的帧数。

线性预测

在本发明中，线性预测被应用于无损音频编码。图 1 所示的预测器 160 包括至少一个或多个滤波器系数以从之前的采样值预测当前的采样值。随后，第二熵编码部件 180 在对应于预测值和原始值之间的差的残数值上执行熵编码。另外，作用于预测器 160 的每个块的预测器系数值被选为来自系数估算部件 120 的最佳值。此外，预测器系数值被第一熵编码部件 140 熵编码。由第一熵编码部件和第二熵编码部件 180 编码的数据作为比特流的一部分由多路复用部件 190 插入并随后被发送。

下面将详细说明根据本发明的执行线性预测的方法。

用 FIR 滤波器的预测

线性预测适用于语音和音频信号处理的许多场合。在下文中基于有限脉冲响应 (FIR) 滤波器描述预测器 160 的示例性操作。然而，本例明显不构成对本发明范围的限制。

时间离散信号 $x(n)$ 的当前采样可从之前的采样 $x(n-k)$ 大致地预测出。本说明是通过下式给出的。

$$\hat{x}(n) = \sum_{k=1}^K h_k * x(n-k),$$

其中 K 是预测器的阶数。如果预测的采样接近原始采样，则残数如下所示：

$$e(n) = x(n) - \hat{x}(n)$$

它具有比 $x(n)$ 本身更小的方差，因此能更有效地编码 $e(n)$ 。

从输入采样的段开始在对段进行滤波前估算预测器系数的程序被成为前向自适应。在这种情形下，应当发送系数。另一方面，如果从之前处理的段或采样（例如从残数）估算系数，则参照向后自适应。向后自适应程序的优点在于：由于估算系数所需的数据是解码器可得的，因而不需要系数发送。

10 阶左右的前向自适应预测方法被广泛地用于语音编码，并且同样适用于无损音频编码。大多数前向自适应无损预测方案的最大阶数仍然相当小，例如 $K=32$ 。一种例外是超级音频 CD 的专门的 1 比特无损编解码，它使用高达 128 的预测阶数。

另一方面，具有几百个系数的向后自适应 FIR 滤波器通用于许多领域，例如声道均衡和回声对消。多数这些系统是基于 LMS 算法或其变化形式的，这些算法也被推荐用于无损音频编码。由于不一定要将预测器系数作为辅助信息发送，这类具有高阶数的基于 LMS 的编码方案在实践中是可行的，因此它们的数目不对数据速率产生影响。然而，向后自适应的编解码器的缺点在于：必须在编码器和解码器两者中作出适应，这使解码器明显比前向自适应情形下的解码器更为复杂。

前向自适应预测

作为本发明的示例性实施例，前向自适应预测作为一个实例被给出于本文的描述中。在前向自适应线性预测中，一般使用自相关方法或协方差方法通过系数估算部件 120 估算每个块的最佳预测器参数 h_k （根据残数的最小方差）。使用传统的 Levinson-Durbin 算法的自相关方法的额外优点是提供一种迭代地适应预测器阶数的简单方法。此外，该算法本身也计算相应的部分自相关系数。

前向自适应预测的另一方面是确定合适的预测阶数。阶数增加使预测误差的方差减少，这导致残数的较小比特率 R_e 。另一方面，预测器系数的比特率 R_c 随着要被发送的系数的数目而提高。因此，其任务是找到使总比特率最小的最佳阶数。这可通过相对预测阶数 K 使下式最小来表达：

$$R_{total}(K) = R_e(K) + R_c(K),$$

由于预测增益随着更高的阶数单调上升，因此 R_e 随着 K 值下降。另一方面，

由于要发送数量递增的系数，因此 R_c 单调地随着 K 值上升。

最佳阶数的搜索可通过系数估算部件 120 被高效率地执行，所述系数估算部件 120 用递增的阶数递归地确定所有预测器。对于每个阶数计算预测器系数的完整组。另外，可推导出相应残数的方差 σ_e^2 变化，这导致残数的可望比特率的估算。可在每次重复过程中——例如对于每个预测阶数——确定总比特率连同诸个系数的比特率。最佳阶数被发现于总比特率不再减少的那个点上。

尽管从上式可以清楚知道系数比特率对总比特率具有直接的影响，然而 R_c 较慢的增加还使 R_{total} 的最小值偏移至较高的阶（其中 R_e 同样较小），这实现更好的压缩。因此，预测器系数的高效并且仍旧准确的量化在实现最大压缩过程中扮演一个重要的角色。

预测阶数

在本发明中确定预测阶数 K ，所述预测阶数 K 确定线性预测的预测器系数的数目。预测阶数 K 还通过系数估算部件 120 予以确定。这里，关于确定的预测阶数的信息被包含在比特流中并随后被发送。

配置语法（表 6）包括关联于预测阶数 K 的信息。例如，1 比特至 10 比特的“max_order”字段对应于指示最大阶数值的信息。1 比特至 10 比特的“max_order”字段的最高值是 $K=1023$ （例如 10 比特）。作为关联于预测阶数 K 的另一消息，配置语法（表 6）包括 1 比特的“adapt_order”字段，它指示每个块是否存在最佳阶数。例如，当“adapt_order=1”时，可以给每个块提供最佳阶数。在 block_data 语法（表 8）中，最佳阶数被提供为 1 比特至 10 比特的“opt_order”字段。此外，当“adapt_order=0”时，则不对每个块提供单独的最佳阶数。在这种情形下，“max_order”字段成为被作用于所有块的最末阶数。

最佳阶数（opt_order）基于 max_order 字段值和相应块的大小（ N_B ）得以确定。更具体地说，例如当 max_order 被确定为 $K_{max}=10$ 并且“adapt_order=1”时，则考虑相应块的大小即可确定每个块的 opt_order。在某些情形下，opt_order 值可能会大于 max_order（ $K_{max}=10$ ）。

尤其，本发明涉及更高的预测阶数。根据本发明的实施例，在没有分层块

切换的情形下,在长块长度和短块长度之间可以存在 4 的倍数(例如 4096&1024 或 8192&2048)。另一方面,在采用分层块切换的实施例,该倍数可以增加(例如高达 32),从而实现更大的范围(例如对于 16384 下至 512 或甚至高采样速率的 32768-1024)。

在执行分层块切换的实施例,为了更好地使用非常长的块,可采用更高的最大预测阶数。最大阶数可以是 $K_{\max}=1023$ 。在本实施例中, K_{\max} 可由块长度 N_B 限定,例如 $K_{\max}<N_B/8$ (例如当 $N_B=2048$ 时, $K_{\max}=255$)。因此,使用 $K_{\max}=1023$ 需要至少 $N_B=8192$ 的块长度。在本实施例中,配置语法(表 6)中的“max_order”字段可高达 10 比特而 block_data 语法(表 8)中的“opt_order”字段同样可高达 10 比特。具体块中的实际比特数取决于一个块所允许的最大阶数。如果该块是短块,本地预测阶数可小于全局预测阶数。这里,本地预测阶数是通过考虑相应块长度 N_B 确定的,而全局预测阶数是通过配置语法中的“max_order” K_{\max} 确定的。例如,如果 $K_{\max}=1023$,但 $N_B=2048$,由于本地预测阶数为 255,“opt_order”字段被确定为 8 比特(而不是 10 比特)。

更具体地说,可基于下式确定 opt_order:

$\text{opt_order} = \min(\text{全局预测阶数}, \text{本地预测阶数});$

另外,全局和本地预测阶数可通过下式确定:

全局预测阶数 = $\text{ceil}(\log_2(\text{最大预测阶数} + 1))$

本地预测阶数 = $\max(\text{ceil}(\log_2((N_B >> 3) - 1)), 1)$

在本实施例中,预测来自一个声道的再分块的数据采样。使用之前块的最末 K 个采样预测当前块的第一采样。 K 值是基于从上述等式推导出的 opt_order 确定的。

如果当前块是声道的第一个块,则没有来自之前块的采样以供使用。在这种情形下,采用具有渐进阶数(progressive order)的预测。例如,假设相应块的 opt_order 值为 $K=5$,则块中的第一采样不执行预测。块的第二采样使用块的第一采样以执行预测(如同 $k=1$),块的第三采样使用块的第一采样和第二采样以执行预测(如同 $k=2$)等。因此,从第六采样开始的以及在这之后的采样,根据 $k=5$ 的 opt_order 执行预测。如上所述,预测阶数从 $k=1$ 至 $k=5$ 渐进地增加。

当用于随机存取帧时，上述渐进阶数型预测是非常有利的。由于随机存取帧对应于随机存取单元的基准帧，随机存取帧不通过使用之前的帧采样执行预测。即，该渐进预测技术可应用于随机存取帧的开头。

预测器系数的量化

上述预测器系数在图 1 的量化部件 130 中被量化。由于非常小的量化误差也会导致最佳预测滤波器的要求频谱特性的很大的偏差，因此预测系数 h_k 的直接量化对发送而言不是非常有效的。为此，预测器系数的量化基于通过系数估算部件 120 计算得到的部分自相关（反映）系数 r_k 。如上所述，例如系数估算部件 120 是使用传统 Levinson-Durbin 算法处理的。

开头的两个部分自相关系数（对应 γ_1 和 γ_2 ）通过使用下面的函数被量化：

$$a_1 = \lfloor 64(-1 + \sqrt{2\sqrt{\gamma_1 + 1}}) \rfloor;$$

$$a_2 = \lfloor 64(-1 + \sqrt{2\sqrt{-\gamma_2 + 1}}) \rfloor;$$

而其余系数是使用简单的 7 比特的均匀量化器量化的：

$$a_k = \lfloor 64 \gamma_k \rfloor; \quad (k > 2).$$

在所有情形下，结果量化值 a_k 被约束在范围 $[-64, 63]$ 内。

熵编码

如同 1 所示，两种类型的熵编码被用于本发明。更具体地说，第一熵编码部件 140 用于编码上述预测器系数。另外，第二熵编码部件 180 用来编码上述音频原始采样和音频残数采样。在下文中将详细说明这两种熵编码。

预测器系数的第一熵编码

相关技术的 Rice 码被用作根据本发明的第一熵编码方法。例如，量化系数 a_k 的发送是通过生成残数值执行的：

$$\delta_k = a_k - \text{offset}_k$$

它是通过使用第一熵编码部件 140 编码的，例如 Rice 码方法。用于该处理的 Rice 码的相应偏移和参数可从下述表 3、4 和 5 所示的诸组中的一个全局地进

行选择。表索引（即 2 比特的“coef_table”）在配置语法中给出（表 6）。如果“coef_table=11”，这表示不采用熵编码，并且量化的系数每次 7 比特地发送。在这种情形下，偏移一直为 -64 以获得局限于[0,127]的无符号值 $\delta_k = a_k + 64$ 。相反，如果“coeff_table=00”，则选择下面的表 3，并且如果“coeff_table=01”，则选择下面的表 4。最后，如果“coeff_table=10”，则选择表 5。

当接收到图 2 的解码器中的经量化系数时，第一熵解码部件 220 通过使用将残数值 δ_k 与偏移组合以生成部分自相关系数 a_k 的量化索引的处理来重构预测器系数：

$$a_k = \delta_k + \text{offset}_k$$

此后，使用下面的等式执行开头的两个系数（ γ_1 和 γ_2 ）的重构：

$$\begin{aligned} \text{par}_1 &= \lfloor \hat{\gamma}_1 2^Q \rfloor = \Gamma(a_1); \\ \text{par}_2 &= \lfloor \hat{\gamma}_2 2^Q \rfloor = -\Gamma(a_2); \end{aligned}$$

其中 2^Q 表示重构的系数的整数表示所需的常数（ $Q=20$ ）缩放因数，而 $\Gamma(\cdot)$ 是根据经验确定的映射表（未示出为映射表是因为会根据场合而变化）。

因此，根据采样频率提供用于第一熵编码的这三种系数表。例如，采样频率可被分成 48kHz、96kHz 和 192kHz。这里，三个表 3、4、5 被分别提供给每个采样频率。

作为使用一个表的代替，可对整个文件选择三个不同表中的一个。一般根据采样速率选择表。对于 44.1kHz 的材料，本发明的申请人推荐使用 48kHz 表。然而，总地来说，也可通过其它标准选择表。

表 3：用于编码量化系数（48kHz）的 Rice 码参数

系数#	偏移	Rice 参数
1	-52	4
2	-29	5
3	-31	4
4	19	4
5	-16	4
6	12	3
7	-7	3

8	9	3
9	-5	3
10	6	3
11	-4	3
12	3	3
13	-3	2
14	3	2
15	-2	2
16	3	2
17	-1	2
18	2	2
19	-1	2
20	2	2
$2k-1, k>10$	0	2
$2k, k>10$	1	2

表 4: 用于编码量化系数 (96kHz) 的 Rice 码参数

系数#	偏移	Rice 参数
1	-58	3
2	-42	4
3	-46	4
4	37	5
5	-36	4
6	29	4
7	-29	4
8	25	4
9	-23	4
10	20	4
11	-17	4
12	16	4

13	-12	4
14	12	3
15	-10	4
16	7	3
17	-4	4
18	3	3
19	-1	3
20	1	3
$2k-1, k>10$	0	2
$2k, k>10$	1	2

表 5; 用于编码量化系数 (192kHz) 的 Rice 码参数

系数#	偏移	Rice 参数
1	-59	3
2	-45	5
3	-50	4
4	38	4
5	-39	4
6	32	4
7	-30	4
8	25	3
9	-23	3
10	20	3
11	-20	3
12	16	3
13	-13	3
14	10	3
15	-7	3
16	3	3
17	0	3

18	-1	3
19	2	3
20	-1	2
$2k-1, k > 10$	0	2
$2k, k > 10$	1	2

残数的第二熵编码

本发明包含适用于图 1 的第二熵编码部件 180 的编码方法的两种不同的模式，这将在下面予以详细说明。

在简单模式中，使用 Rice 码对残数值 $e(n)$ 进行熵编码。对于每个块，或者使用相同的 Rice 码编码所有的值，或者将块进一步分成四个部分，每个部分用不同的 Rice 码编码。如图 1 所示那样发送这种应用的代码索引。由于存在不同的方法以确定所给出的数据组的最佳 Rice 码，因此编码器根据残数的统计选择合适的代码。

或者，编码器可使用基于 BGMC 模式的更为复杂和高效的编码方案。在 BGMC 模式中，残数的编码是通过将分布曲线划分成两种类型实现的。这两种类型包括属于分布的中心区域的残数 $|e(n)| < e_{\max}$ ，以及属于其尾端的残数。尾端的残数可简单地仅重定中心（即对于 $e(n) > e_{\max}$ ，则提供 $e_t(n) = e(n) - e_{\max}$ ）并使用如上所述的 Rice 码编码。然而，为了编码处于分布曲线中心的残数，BGMC 首先将残数分成 LSB 和 MSB 部分，随后 BGMC 使用分组 Gilbert_Moore（算术）码编码 MSB。最后，BGMC 使用固定长度的直接代码发送 LSB。可选择直接发送的 LSB 的参数 e_{\max} 和数目以使它们仅略微地影响这种方案的编码效率，同时使编码的复杂度明显降低。

根据本发明的配置语法（表 6）和 block_data 语法（表 8）包括关联于 Rice 码和 BGMC 码的编码的信息。现在对这种信息进行详细说明。

配置语法（表 6）首先包括 1 比特的“bgmc_mode”字段。例如，“bgmc_mode”=0 表示 Rice 码。配置语法（表 6）还包括 1 比特的“sb_part”字段。“sb_part”字段对应于与将块分割成子块并对分割的子块编码的方法相关的信息。这里，“sb_part”的意义根据“bgmc_mode”字段值而改变。

例如，当“bgmc_mode=0”时，也就是当采用 Rice 码时，“sb_part=0”表示块不被分割成子块。另外，“sb_part=1”表示以 1:4 子块分割比分割块。另外，当“bgmc_mode=1”时，也就是当采用 BGMC 码时，“sb_part=0”表示以 1: 4 子块分割比分割块。或者，“sb_part=1”表示以 1: 2: 4: 8 子块分割比分割块。

与包含在配置语法（表 6）中的信息对应的每个块的 Block_data 语法（表 8）包括 0 比特至 2 比特的可变“ec_sub”字段。更具体地说，“ec_sub”字段表示存在于实际相应块中的子块的数目。这里，“ec_sub”字段的意义根据配置语法（表 6）中的“bgmc_mode” 字段+“sb_part”字段的值而变化。

例如，“bgmc_mode+sb_part=0”表示 Rice 码不配置子块。这里，“ec_sub”字段是 0 比特字段，这表示不包含任何信息。

另外，“bgmc_mode+sb_part=1”表示 Rice 码或 BGMC 码用来以 1: 4 的比例将块分割成若干子块。这里，只有 1 比特被分配给“ec_sub”字段。例如，“ec_sub=0”表示一个子块（即该块不被分割成若干子块），而“ec_sub=1”指示以配置 4 个子块。

此外，“bgmc_mode+sb_part=2”表示 BGMC 码被用来以 1: 2: 4: 8 的比例将块分割成若干子块。这里，2 比特被分配给“ec_sub”字段。例如，“ec_sub=0”表示一个子块（即该块不被分割成若干子块），而“ec_sub=01”指示 2 个子块。另外，“ec_sub=10”指示 4 个子块，而“ec_sub=11”指示 8 个子块。

如上所述定义在每个块中的子块使用不同的编码方法由第二熵编码部件 180 编码。下面描述使用 Rice 码的一个实例。对于残数值的每个块，或者使用同样的 Rice 码编码所有值，或者如果设定配置语法中的“sb_part”字段则块被分割成四个子块，每个编码的子块具有不同的 Rice 码。在后一种情形下，块数据语法（表 8）中的“ec_sub”字段表示是使用一个块还是四个块。

尽管第一子块的参数 $s[i=0]$ 或者用 4 比特（分辨率 ≤ 16 比特）或者用 5 比特（分辨率 > 16 比特）被直接发送，然而仅发送下列参数 $s[i>0]$ 的差 $(s[i]-s[i-1])$ 。使用适当选择的 Rice 码再次编码这些差。在这种情形下，用作差的 Rice 码参数具有值“0”。

语法

根据本发明的实施例，包含在音频比特流中的各种信息的语法示出于下表中。

表 6 示出音频无损编码的配置语法。这种配置语法可形成分周期地处于比特流中的帧头，可形成每个帧的帧头等。表 7 示出一种帧—数据语法，而表 8 示出一种块—数据语法。

表 6: 配置语法

语法	比特
<pre> ALSspecificConfig() { samp_freq; samples; channels; file_type; resolution; floating; msb_first; frame_length; random_access; ra_flag; adapt_order; coef_table; long_term_prediction; max_order; block_switching; bgmc_mode; sb_part; joint_stereo; mc_coding; chan_config; chan_sort; crc_enabled; RLSLMS (reserved) if (chan_config) { chan_config_info; } if (chan_sort) { for (c = 0; c < channels; c++) chan_pos[c]; } header_size; trailer_size; orig_header[]; </pre>	<pre> 32 32 16 3 3 1 1 16 8 2 1 2 1 10 2 1 1 1 1 1 1 1 6 16 8 16 16 heade r_siz e * 8 </pre>
<pre> orig_trailer[]; if (crc_enabled) { crc; } if ((ra_flag == 2) && (random_access > 0)) { for (f = 0; f < (samples - 1 / frame_length) + 1; f++) { ra_unit_size } } </pre>	<pre> trail er_si ze * 8 32 32 </pre>

表 7: Frame_data 语法

语法	比特
<pre> frame_data() { if ((ra_flag == 1) && (frame_id % random_access == 0)) { ra_unit_size } if (mc_coding && joint_stereo) { js_switch; byte_align; } if (!mc_coding js_switch) { for (c = 0; c < channels; c++) { if (block_switching) { bs_info; } if (independent_bs) { for (b = 0; b < blocks; b++) { block_data(c); } } else{ for (b = 0; b < blocks; b++) { block_data(c); block_data(c+1); } c++; } } } else{ </pre>	<p>32</p> <p>1</p> <p>8, 16, 32</p>

<pre> if (block_switching) { bs_info; } for (b = 0; b < blocks; b++) { for (c = 0; c < channels; c++) { block_data(c); channel_data(c); } } if (floating) { num_bytes_diff_float; diff_float_data(); } </pre>	<p>8, 16, 32</p> <p>32</p>
---	--------------------------------

表 8:Block_data 语法

语法	比特
<pre> block_data() { block_type; if (block_type == 0) { const_block; js_block; (reserved) if (const_block == 1) { { if (resolution == 8) { const_val; } else if (resolution == 16) { const_val; } else if (resolution == 24) { const_val; } else { const_val; } } } } else { js_block; if ((bgmc_mode == 0) && (sb_part == 0) { sub_blocks = 1; } } } </pre>	<p>1</p> <p>1</p> <p>1</p> <p>5</p> <p>8</p> <p>16</p> <p>24</p> <p>32</p> <p>1</p>

<pre> } else if ((bgmc_mode == 1) && (sb_part ==1) { ec_sub; sub_blocks = 1 << ec_sub; } else { ec_sub; sub_blocks = (ec_sub == 1) ? 4 : 1; } if (bgmc_mode == 0) { for (k = 0; k < sub_blocks; k++) { s[k]; } } else { for (k = 0; k < sub_blocks; k++) { s[k],sx[k]; } } sb_length = block_length / sub_blocks; shift_lsbs; if (shift_lsbs == 1) { shift_pos; } if (!RLSLMS) { if (adapt_order == 1) { opt_order; } for (p = 0; p < opt_order; p++) { quant_cof[p]; } } } </pre>	<p>2</p> <p>1</p> <p>varie s</p> <p>varie s</p> <p>1</p> <p>4</p> <p>1...10</p> <p>varie s</p>
---	--

压缩结果

下面，将无损音频编解码与两种最流行的无损音频压缩程序——即开放源代码编解码 FLAC 和 Money 式音频（MAC 3.97）作比较。这里，开放源代码编解码 FLAC 使用前向自适应预测，而 Monkey 式音频（MAC3.97）是作为压缩方面的当前技术发展水平的向后自适应编解码。两种编解码方案均涉及提供最大的压缩的选项（即 flac-8 和 mac-c4000）。编码器的结果确定中等压缩水平（其预测阶数限制于 K_60）以及最大压缩水平（K_1023），两者均具有 500ms 的随机存取。测试执行于 1024MB 内存的 1.7GHz 奔腾-M 系统。测试包括具有 48、96 和 192kHz 的采样速率和 16、24 比特分辨率的将近 1GB 的立体声波形数据。

压缩比

下面，压缩比被定义为：

$$C = [(\text{压缩的文件大小}) / (\text{原始文件大小})] * 100\%$$

其中更小的值代表更好的压缩。检测的音频格式的结果示出于表 9（FLAC 编解码不支持 192kHz 的资料）。

表 9：不同音频格式的平均压缩比的比较（kHz/比特）

格式	FLAC	MAC	ALS 平均	ALS 最大
48/16	48.6	45.3	45.5	44.7
48/24	68.4	63.2	63.3	62.7
96/24	56.7	48.1	46.5	46.2
192/24	—	39.1	37.7	37.6
总共	—	48.9	48.3	47.8

该结果表示最高水平的 ALS 胜过所有格式的 FLAC 和 Monkey 式音频，但尤其适用于高分辨率资料（例如 96kHz/24 比特及以上）。即使在中等水平，ALS 也提供最好的总压缩性。

复杂度

不同编解码的复杂度强烈地取决于实际应用，尤其是编码器的应用。如前面提到的那样，本发明的音频信号编码器正处于发展中。因此，我们将我们的

分析局限于解码器、简单的 C 语言代码实现而不作进一步的优化。压缩的数据是通过当前最佳的编码器实现产生的。图 10 中示出以不同复杂程度编码的用于各种音频格式实时解码的平均 CPU 负载。即使是最大复杂度，解码器的 CPU 负载仅在 20-25%左右，这表示基于文件的解码比实时解码快至少 4—5 倍。

表 10：平均 CPU 负载（在 1.7GHz 奔腾-M 上的百分比），取决于音频格式（kHz/比特）和 ALS 编码器复杂度。

格式	ALS 低	ALS 平均	ALS 最大
48/16	1.6	4.9	18.7
48/24	1.8	5.8	19.6
96/24	3.6	12.0	23.8
192/24	6.7	22.8	26.7

编解码器被设计成提供大范围的复杂程度。尽管最大复杂程度以最低的编码和解码速度的代价实现最高的压缩，然而更快的中等复杂程度仅略微地损害压缩性，但对于最大程度而言（即 48kHz 资料的将近 5%的 CPU 负载），解码的复杂度明显降低。适用低复杂度水平（即 K_15, Rice 编码）相比中等程度仅使压缩性降低 1-1.5%，但是解码复杂性进一步降低 3 倍（即对 48kHz 资料而言小于 2%CPU 负载）。因此，音频数据即使在计算能力很低的硬件上也能被解码。

尽管编码器复杂度可能因为更高的最大阶数和更精良的块切换算法而增加（根据实施例），然而解码器更容易受到较高的平均预测阶数的影响。

前面的实施例（例如分层块切换）和优点仅为示例性的并且不被解释为对所附权利要求书构成限制。上述原理可应用于其它装置和方法，并为业内技术人员所理解。许多选择、修改和变化对本领域内技术人员而言是明显的。

工业应用

本领域内技术人员可以理解可对本发明作出各种修改和变化而不脱离本发明的精神或范围。例如，本发明的诸方面和实施例可轻易地适用于另一音频

信号编解码器，例如无损音频信号编解码器。因此，本发明旨在覆盖本发明所有这些修改和变化。

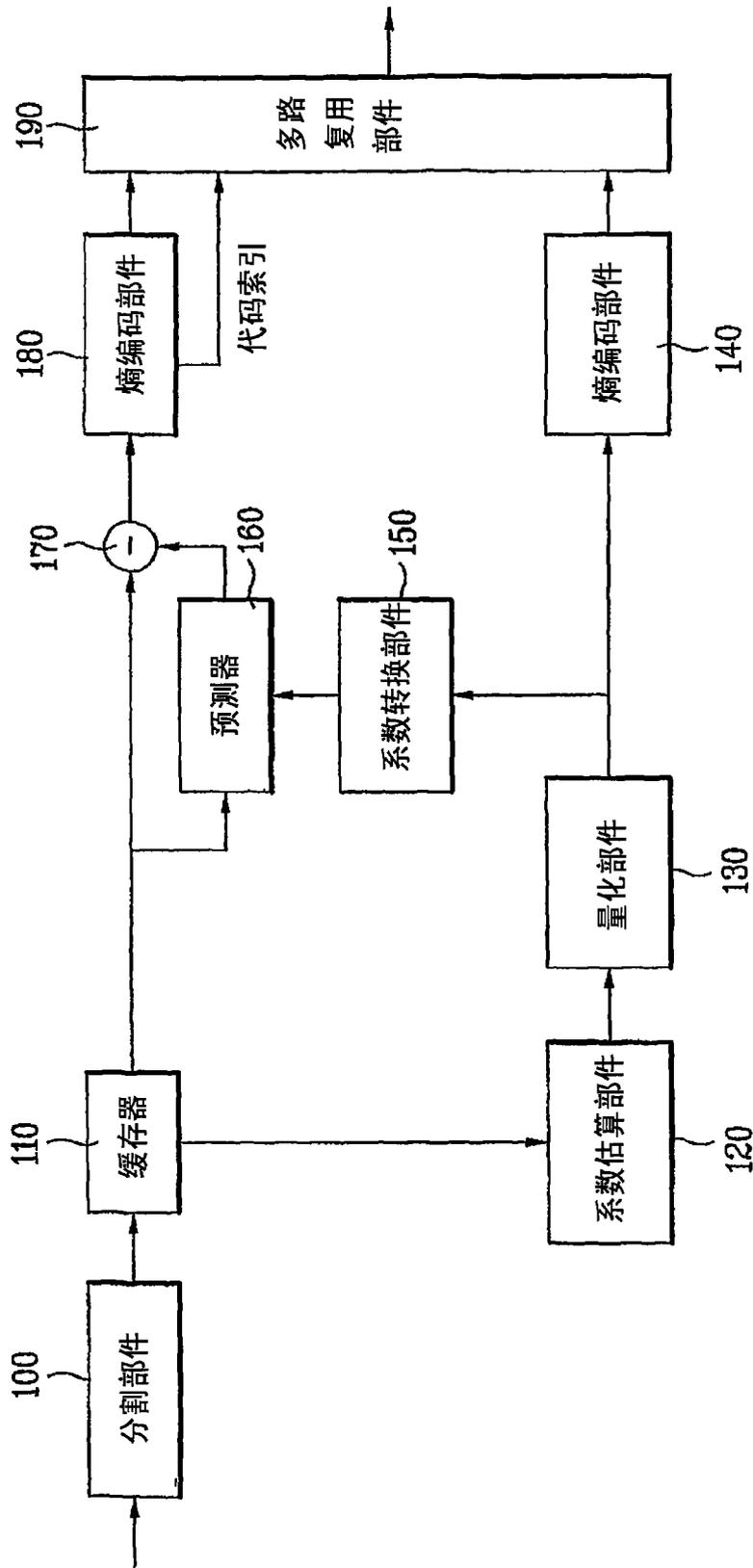


图 1

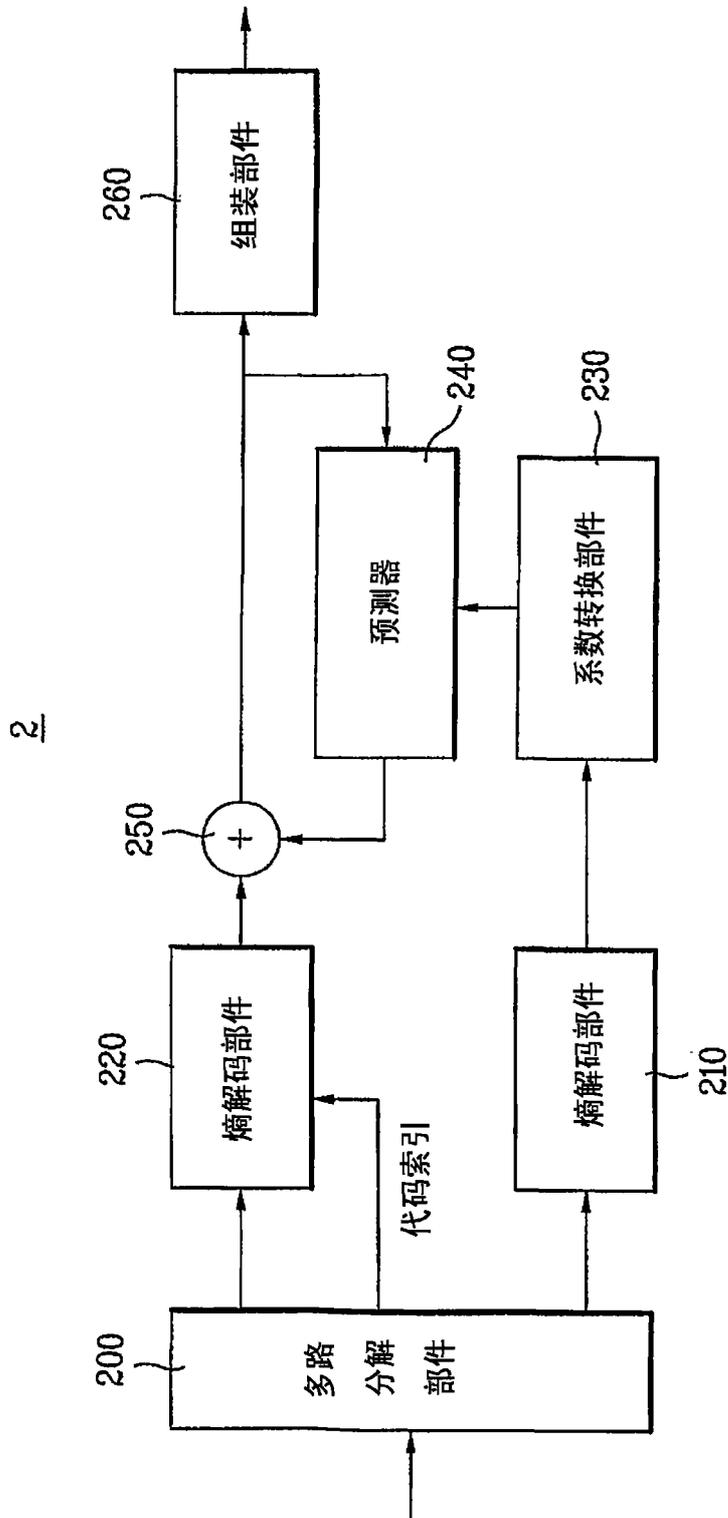


图 2

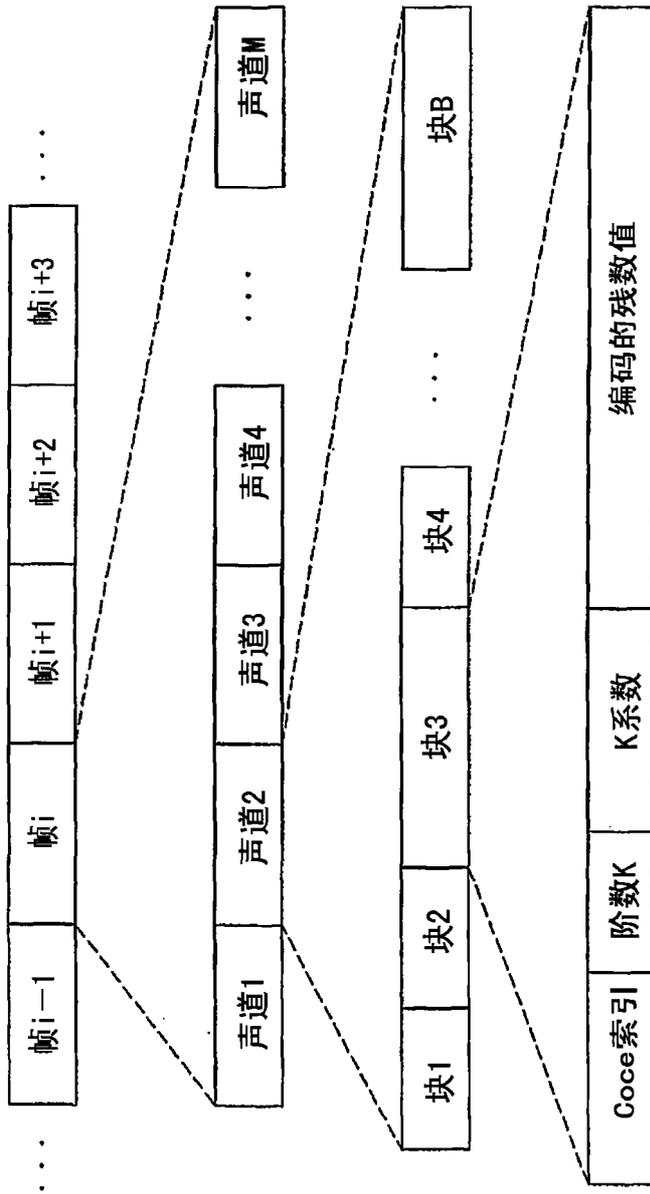


图 3

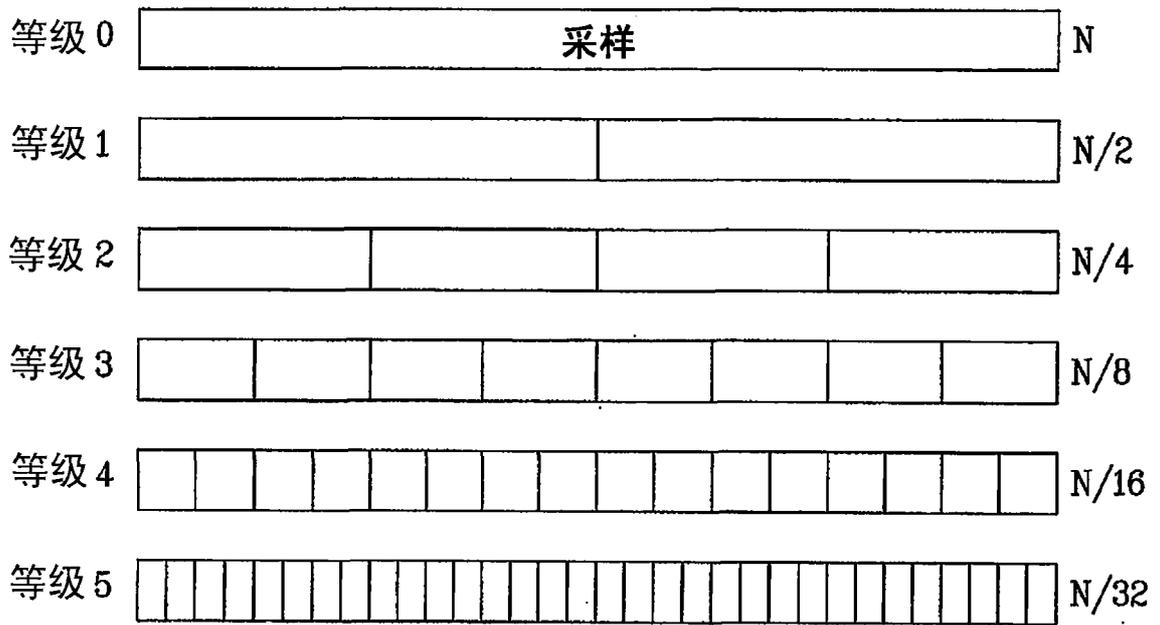


图 4

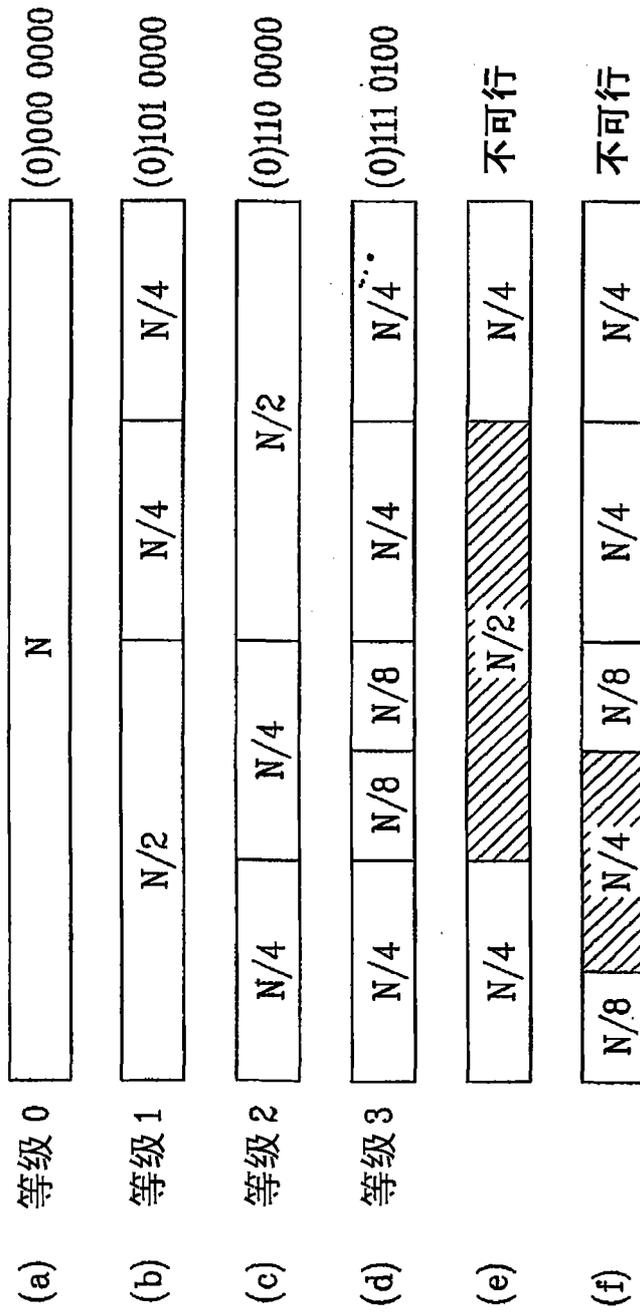


图 5

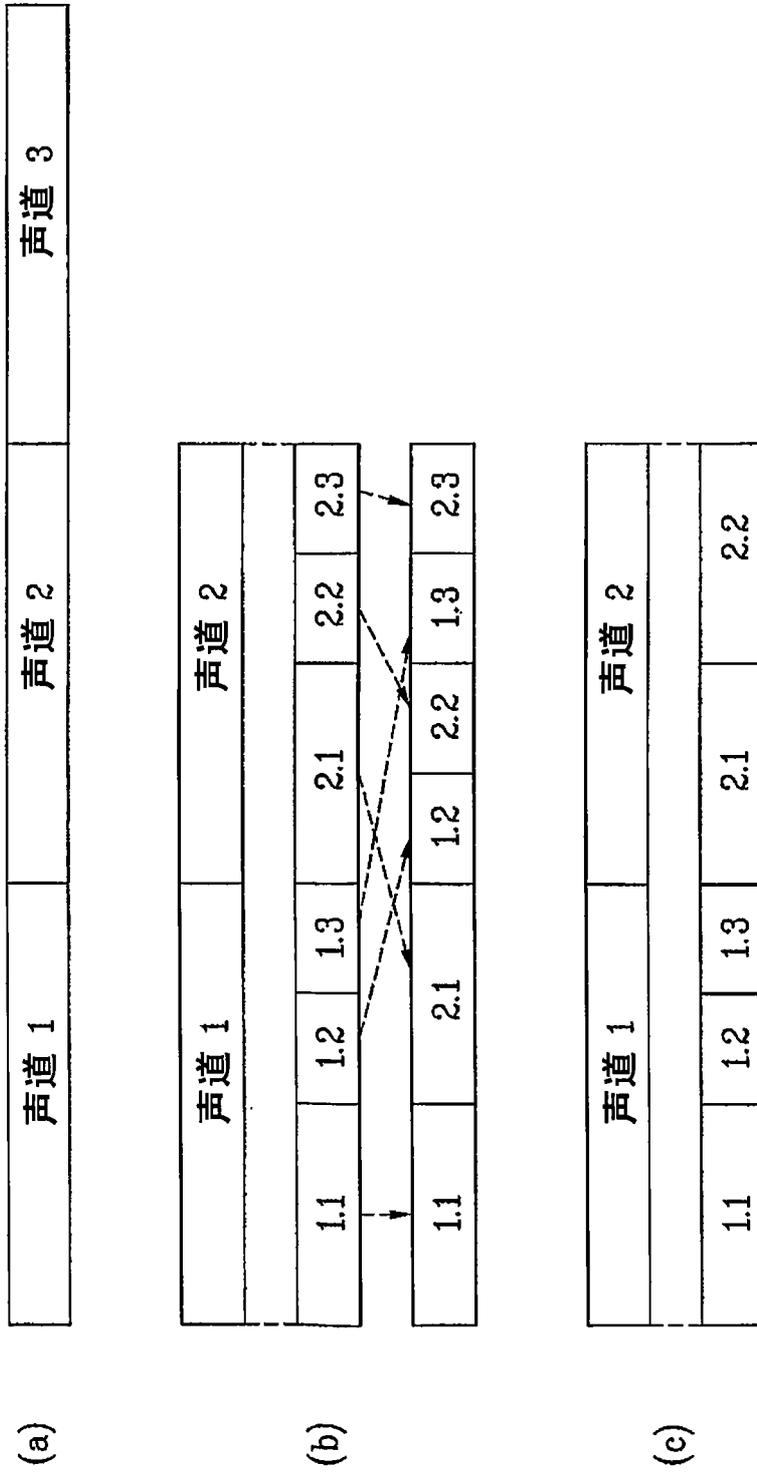


图 6