[19]中华人民共和国专利局

[11] 公开号 CN 1133512A

[12] 发明专利申请公开说明书

[21]申请号 95117232.8

[51]Int.Cl[6]

H03M 7/30

[43]公开日 1996 年 10 月 16 日

[22]申请日 95.9.29

[30]优先权

[32]94.9.30 [33]US[31]316,116

[71]申请人 株式会社理光

地址 日本东京

[72]发明人 E·L·许瓦兹 M·J·戈米什

J·D·阿伦 M·皮力克

[74]专利代理机构 中国专利代理(香港)有限公司

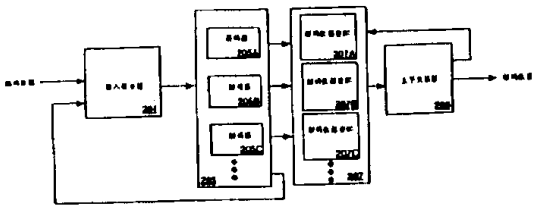
代理人 董 巍 萧掬昌

权利要求书 20 页 说明书 103 页 附图页数 36 页

[54]发明名称 编码与解码数据的方法和装置

[57]摘要

描述了一个解压缩和压缩数据的方法和装置。本发明提供了一个编码器，用在一个具有解码由编码器生成的信息的解码器的压缩系统中。本发明的解码器包括一个为响应数据而产生编码字信息的编码器。编码器不包括一个为响应来自编码器的码字信息而生成一个编码数据流的重新排序单元。重新排序单元包括一个安排编码字到一个解码次序的运行计数重新排序单元和一个将变长编码字结合入定长交织字并以解码器所要求的一个次序输出定长交织字的比特组合单元。



权 利 要 求 书

1. 一个编码一个数据流的方法包括的步骤有:

为响应数据流，生成表示数据流的码字信息，其中码字信息包含多个码字，且进一步地多个码字从被并行处理的数据流的数据中生成;

为了响应码字信息生成编码的数据，其中，生成编码数据的步骤包含所说的多种在被输出码字信息中码字的每一个，这样编码字被输出的次序是根据被每个码字表示的数据流的开始部分。

2. 权利要求 1 中定义的方法，其中多个编码字的每一个在每个运行开始处被输出。

3. 权利要求 1 中定义的方法，其中生成编码数据的步骤进而包含将变长编码字组合成定长数据结构。

4. 权利要求 3 中定义的方法，其中每个定长数据结构包含一个交织。

5. 权利要求 1 中定义的方法，进而包含输出编码字，这样码字按解码顺序排序。

6. 权利要求 1 中定义的方法，进而包含排序码字的步骤。

7. 权利要求 1 中定义的方法，其中生成码字信息的步骤包括:

生成一个码字的概率状态;

根据概率等级选择一个比特生成码;

访问存储器以获得与概率等级相关的一个运行计数。

8. 编码一个数据流的编码系统，所述编码系统包括：

一个编码器被连接接收数据流，以生成包括码字的编码字信息，且进而其中，由编码器从被并行处理的数据中生成多个码字；和

一个重新排序单元被连接到编码器以响应码字信息而生成编码数据，其中重新排序单元重新排序由编码器按一个解码器次序生成的码字，这样被输的码字的次序根据由每个编码字表示的数据流开始部分。

9. 权利要求 7 中定义的编码系统，其中重新排序单元以如同指示码字长度的方式存贮码字。

10. 权利要求 9 中定义的编码系统，其中 1N 编码字被重新排序，以使最高有效“1”比特指示每个码字的长度。

11. 权利要求 8 中定义的编码系统，其中重新排序单元包括：

至少一运行计数重新排序单元以生成在每个运行开始处的码字信息中的所述多个码字；且

至少一比特组合单元以将变长码字结合到定长数据结构中。

12. 权利要求 11 中定义的编码系统，其中每个定长数据结构包含一个交织字。

13. 权利要求 8 中定义的编码系统，其中重新排序单元排序码字。

14. 权利要求 13 中定义的编码系统，进而包括一个连接到重新排序单元以按一个次序存储码字的存储器。

15. 权利要求 8 中定义的编码系统，其中编码器包括：

一个上下文模型；

一个连接到上下文模型的概率估计机；和

一个连接到概率估计机为响应数据流而生成码字的比特发生器结构。

16. 权利要求 15 中定义的编码系统，其中编码器进而包括一个存储多个运行计数的存储器。且使用一个概率等级从概率估计机存取存储器，以提供多个运行计数之一给比特发生器，作为码字信息的部分输出。

17. 权利要求 15 中定义的编码系统，其中比特发生器结构包括：

一个为响应一个索引和一个 MPS/LPS 指示而提供码字信息的比特发生器；和

一个为提供一个运行计数给比特发生器而连接到比特发生器的存储器，此处比特发生器根据索引读取存储器，并根据从读取存储器获得的数据执行比特生成。

18. 权利要求 17 中定义的编码系统，其中编码字信息包括一个指示 MPS/LPS 指示是否包括一个运行开始的第一信号，一个指示 MPS/LPS 指示是否包括一个运行结束的第二信号，和一个码字输出。

19. 权利要求 11 中定义的编码系统，其中比特组合逻辑排序交织字以生成按照在每个流的每个交织字中一个第 N 个前码字排序的作为交织流的编码数据流。

20. 权利要求 11 中定义的编码系统，其中重新排序单元包括一个选择交织字输出到码流的窥探解码器。

21. 权利要求 20 中定义的编码系统，其中重新排序单元包括

多个连接到多个比特组合单元的运行计数重新排序单元，其中多个比特组合单元的每一个生成交织字且窥探解码器从多个交织字选择一个交织字作为到码流的一个输出。

22. 权利要求 11 中定义的编码系统，其中的码字信息包括一个时间标记，且重新排序单元进而包括根据相关时间标记到输出交织字的逻辑。

23. 权利要 22 中定义的编码系统，其中重新排序单元包括连接到多个比特组合单元的多个运行计数重新排序单元，且进而其中逻辑导致交织字根据一个相关的时间标记被输出。

24. 权利要求 23 中定义的编码系统，其中交织字根据最旧的时间标记被输出。

25. 权利要求 11 中定义的编码系统，其中一个单一队列为多个比特组合单元提供码字，且此处多种比特组合单元为作为码流的部分输出生成交织字。

26. 权利要求 25 中定义的编码系统，其中单一队列包括一个单一运行计数重新排序单元，且其中逻辑确定下一个交织字，用于作为码流输出。

27. 权利要求 11 中定义的编码系统，其中一个单一队列提供码字给一个单一比特组合单元。

28. 一个用于具有一个为解码编码器生成信息的解码器的压缩系统的编码系统，所说的编码包括：

一个为响应数据而产生码字信息的编码器；

一个连接到编码器的重新排序单元，其中为响应码字信息，重新排序单元生成一个编码数据流，其中重新排序单元包括一个

为将每个码字放在相应于其的数据开始处的运行计数重新排序单元，和一个被连接的从运行计数重新排序单元接收码字，以将变长码字组合至多个定长交织字，并以解码器要求的次序输出多个定长交织字的比特组合单元。

29. 权利要求 28 中定义的编码系统，其中重新排序单元进而包括一个在重新排序期间存贮码字的存储器。

30. 权利要求 28 中定义的编码系统，其中编码器进而包括一个上下文模型，一个连接到上下文模型的概率估计机制和一个连接到概率估计机制的比特流发生器。

31. 权利要求 30 中定义的编码系统，其中运行计数重新排序单元进而包括:

一个存贮码字的第一存储器;

一个为寻址作为一个队列的第一存储器的第一指示器和第二指示器，其中第一指示器指向被指定作为第一存储器一个输出的第一入口，且第二指示器指向被指定作为在第一存储器中下一个可用且未被分配的存储位置的一个第二入口。

32. 权利要求 31 中定义的编码系统，进而包括一个存储相应于当前被指定存储每个索引编码字的第一存储器中的位置的地址信息的指针存储器。

33. 权利要求 32 中定义的编码系统，其中每个索引指示一个概率等级。

34. 权利要求 32 中定义的编码系统，其中的每个索引指示至少一个上下文。

35. 权利要求 28 中定义的编码系统，其中的重新排序单元进

而包括存储码字的一个码字存储器队列，一个指示至少一个输出码字的头指针，和一个指示至少插入编码字到码字存储器队列的存储器位置的尾指针。

36. 权利要求 35 中定义的编码系统，其中码字存储器队列中的每个码字条目包括一个正确性指示，且进而其中当头指针指定编码字的地址并且码字的正确性指示其正确时，一个码字从码字存储器队列被输出。

37. 权利要求 28 中定义的编码器，其中比特组合单元包括从重新排序单元接收码字和合并码字至多个流的交织字中的比特组合逻辑。

38. 权利要求 37 中定义的编码器，其中比特组合逻辑进而包括多个累加器和多个寄存器，其中多个寄存器中的每一个被连接到多个累加器之一和多个流之一，每个存贮其相关流的一个交织字的寄存器和每个指示流的存贮当前码字的关联寄存器中下一个位置的累加器，这样，根据相关累加器中的值，多个流之一的每个码字被附加到与所说多个流之一相关联的寄存器内容中。

39. 权利要求 38 中定义的编码系统，为了响应至少一个来自多个累加器的信号而包括一个连接到多个移位码字的累加器的移位器，其中根据一个累加器的值来把至少两个码字的部分组合成每个交织字，编码字被移位以附加到多个寄存器之一的内容中。

40. 权利要求 37 中定义的编码系统，进而包括一个以解码器指定次序存贮交织字的重新排序存储器。

41. 权利要求 40 中定义的编码系统进而包括多个相应于多

个交织数据流的指针，其中多个指针中的每个指定多个流中的每一个的下一个交织字的重新排序存储器中的一个位置。

42. 权利要求 28 中定义的编码系统，进而包括多个运行计数重新排序单元，其中每运行计数重新排序单元与编码数据流之一相关联，且进而包括多个为每个编码数据流生成交织字的比特组合单元和从多个比特组合单元选择交织字作为编码输出的一个解码器。

43. 权利要求 28 中定义的编码系统，进而包括多个运行计数重新排序单元，其中每个运行计数重新排序单元与编码数据流之一相关联，并生成多个编码字和一个与多个编码字中每一个相关联的时间标记，并进而包括多个为每个编码数据流生成交织字的比特组合单元，和根据每个交织字中的码字时间标记选择每个交织字的逻辑。

44. 权利要求 43 中定义的编码系统，其中逻辑选择包含一个具有最旧时间标记的码字的交织。

45. 权利要求 28 中定义的编码系统，其中比特组合单元包括多个被连接以从多个流之一接收码字的比特组合单元，且进而包括基于下一个流，从多个比特组合单元中的每一个选择输出交织字的逻辑。

46. 权利要求 28 中定义的编码系统，其中重新排序单元进而包括一有限存储器。

47. 一个处理数据的编码系统包括:

一个根据数据生成索引的索引发生器;和

一个被连接以根据索引提供概率估计的状态表，其中状态表

包括一个第一多元状态和一个第二多元状态，其中每个状态相应于一个码，且其中当在第一多元状态中，状态间转换时相应于第一多元状态的不同码间的转换的发生快于转换处于第二多元状态时，相应于第二多元状态的不同码间的转换。

48. 权利要求 47 中定义的编码系统，其中第一多元状态只被用于一个预确定的数量的索引。

49. 权利要求 47 中的编码系统，其中的第一多元状态只被用于预确定数量的启始指向状态表的索引。

50. 权利要求 47 中定义的编码系统，其中每个第一多元状态与一个 R2 码关联。

51. 权利要求 48 中定义的编码系统，其中第一多元状态包括至少一个到第二多元状态的转换，这样预确定数量的索引之后，状态表从第一多元状态转换到第二多元状态。

52. 权利要求 47 中定义的编码系统，其中每个第一多元状态与一个不同码相关联。

53. 权利要求 47 中定义的编码系统，其中状态表为了响应一个最不可能的符号，从其一个第一多元状态转换到一个第二多元状态。

54. 权利要求 47 中定义的编码系统，其中状态机为了响应一个最可能的符号而增加状态。

55. 一个处理数据的编码系统包括:

一个根据数据生成索引的索引发生器; 和

一个被连接的根据索引提供概率估计的状态表，其中状态表包括多元状态，其中每个状态相应于一个码，且状态表中的每个

码被重复一预定的次数;

其中状态表状态间的转换根据一个可修改的加速项进行，这样在第一时间期间状态间的转换的第一速率不同于第二时间期间的转换的第二速率。

56. 权利要求 55 中定义的编码系统，其中对状态表的更新包括，通过增加或降低加速项来修改 PEM 状态。

57. 权利要求 56 中定义的编码系统，其中在加速项包括一预确定数量时没有自适应加速发生。

58. 权利要求 56 中定义的编码系统，其中加速项根据连续码字的数量来更新。

59. 权利要求 58 中定义的编码系统，其中连续码字包括一个上下文中的连续码字。

60. 权利要求 58 中定义的编码系统，其中连续码字包括在一个概率等级中的连续码字。

61. 权利要求 56 中定义的编码系统，其中的加速项根据交织码字的数量被更新。

62. 解码一个多个码字数据流的熵解码器包括:

接收数据流的多个比特流发生器; 和

一个连接到多个比特流发生器，以为多个比特流发生器提供概率估计的状态表，其中多个比特流发生器用一个 n 的多值 Rn (k) 码响应概率估计，生成数据流中每个码字的一个解码结果，且进而其中状态表包括一个第一多元状态和一个第二多元状态，其中当转换发生在第一多元状态时，第一多元状态中的不同码间的转换快于当转换处于第二多元状态时码间的转换。

63. 权利要求 62 中定义的熵解码器，其中第一多元状态每个包含一个 R2 (k) 码。

64. 权利要求 62 中定义的熵解码器，其中第一多元状态只用于初始化期间。

65. 解码一个多个码字数据流的熵解码器包括：

接收数据流的多个比特流发生器；和

一个被连接以根据索引提供概率估计的状态表，其中状态表包括一个多元状态，其中每个状态相应于一个码，其状态表中的每一个码被重复一预定次数；

此处状态表的状态间的转换根据一可修改的加速项而发生，这样一个第一时间期间的状态间转换的第一速率不同于一第二时间期间转换的一个第二速率。

66. 权利要求 62 中定义的编码系统，其中状态表中的每一个码被重复固定次数。

67. 权利要求 66 中定义的编码系统，其中对状态表的更新包括，通过一个加速项对 PEM 状态进行修改。

68. 权利要求 67 中定义的编码系统，其中在加速项包括一个预定数量时，发生非自适应性加速。

69. 权利要求 67 中定义的编码系统，其中加速项根据连续码字的数量进行更新。

70. 权利要求 67 中定义的编码系统，其中加速项根据交织编码字的数量进行更新。

71. 一个解码多个交织字的解码器，所述解码器包括：

一个接收数据流并输出恰当定位了的编码数据的移位器。

一个连接到移位器以接收恰当定位的编码数据，作为确定编码字类型的编码字的运行长度解码器；

一个连接到运行长度解码器以确定运行长度解码器编码的概率估计机，这样运行长度解码器为了响应每个码字，生成一个运行长度和一个是否出现 LPS 的指示；

其中移位器包括一个变长移位机制以从数据流中移位码字；和

一个为了响应移位机制而被连接以从流中接收码字的多元寄存器，这样定位的编码数据作为一系列编码字被输出。

72. 权利要求 71 中定义的编码器，其中一部分寄存器被连接以从多个寄存器中另一个或从数据流中接收数据。

73. 权利要求 71 中定义的编码器，其中变长移位机制包括一个从多个寄存器中移动数据的筒型移位器。

74. 权利要求 71 中定义的编码器，其中移位器包括一个具有多个寄存器的 FIFO，其中每个寄存器接收来自交织编码数据的数据作为一个输入，并且多个寄存器中至少一个被连接以从所述多个寄存器中另一个接收码字。

75. 权利要求 71 中定义的编码器，其中移位器包括：

被连接以接收码字数据的第一多元寄存器，其中每个第一多元寄存器被连接到一个不同的多元流；

一个被连接以从每个第一多元寄存器接收码字数据作为一个输入并在一时刻从其中一多元流输出码字的复用器；

一个被连接到复用器输出，以从复用器移位码字数据，作为定位的编码数据输出的筒型移位器；

连接到筒型移位器以指示移位码字的比特数量的逻辑; 和

一个具有多元寄存器被连接以从所述复用器 1605 接收编码字的 FIFO, 其中 FIFO 包括一多元寄存器, 其中每个多元寄存器接收数据, 作为来自交织编码字的一个输入且至少一多元寄存器被连接以接收来自另一所述多元寄存器的码字。

76. 一个解码数据的解码系统, 所述的解码系统包括:

一个被连接以接收数据的 FIFO;

一个用于提供上下文的上下文模型;

一个被连接到上下文模型以存储状态信息的存储器, 其中存储器为了响应上下文模型提供的每个上下文而提供状态信息;

一个被连接以从 FIFO 结构接收编码数据的多元解码器和使用来自存储器的状态信息解码 FI FO 结构提供的码字的存储器。其中多元解码器提供可以对一多元编码生成的运行计数。

77. 权利要求 76 中定义的编码系统, 其中的 FIFO 结构提供编码数据给多元解码器, 而与上下文和概率等级无关。

78. 权利要求 76 中定义的编码系统, 其中解码器包括一个存贮运行计数的运行计数存储器, 且此处的运行计数存储器是可根据概率存取的。

79. 权利要求 76 中定义的编码系统, 其中 FIFO 结构为两个解码器提供数据。

80. 权利要求 76 中定义的编码系统, 其中 FIFO 结构包括一个多元输出, 每个解码器一个。

81. 权利要求 80 中定义的编码系统, 其中 FIFO 结构包括一对复用器和选择复用器对, 以确保为每一个解码器提供一个码字

的控制逻辑。

82. 权利要求 81 中定义的编码系统，其中复用器对根据从其中一个多元解码器接收的请求由控制逻辑进行选择。

83. 一个编码输入数据的编码系统包括：

一个被连接以接收输入数据，生成一个多元流形式的编码数据的编码单元，其中编码数据根据一组准则被分配到其中一个多元流。

一个被连接到存贮编码数据多元流的编码单元的固定容量存储器，其中若固定容量存储器溢出，则不太重要的编码数据被丢弃。

84. 权利要求 83 中定义的编码系统，其中存储器包括多个存贮区域，且存贮在每个多个存贮区域每个中的编码数据包括不同重要性等级的编码数据。

85. 权利要求 84 中定义的编码系统，其中一重要等级的编码数据被存贮在至少一个存储着其它重要等级数据的存储器的存储区域中。

86. 权利要求 85 中定义的编码系统，其中所述一重要等级编码数据复盖在所述至少一个存储器存贮区域中的所述另一重要等级的编码数据。

87. 在编码数据时，一个初始化系统中的多个上下文的方法，所述方法包括：

初始化多个上下文，其中每个上下文根据一个计数器值进行存取；

获得当前上下文的一个 PEM 状态，其中获得的步骤包括：

访问指示 PEM 状态的每个上下文的一个存贮指示;

通过比较当前上下文编号和计数器值，来确定被存取的存器位置是否对一个当前操作有效，这样，如果计数器值指示位置已经被初始化，则数据被确定有效;

当被存取的存储器位置是无效时，使用一个上下文的初始 PEM 状态和忽略一个上下文的当前 PEM 状态;和

如果数据有效，则使用当前被分配的上下文 PEM 状态。

88. 权利要求 87 中定义的方法，包括如果 PEM 状态被改变，写入一个新的 PEM 状态的步骤。

89. 一个解码输入数据的解码器包括:

一个提供一个上下文接收器的上下文模型;

一个连接到上下文模型，以根据上下文接收器提供概率状态的存储器;

一个连接到存储器，以根据概率状态生成一个概率等级的逻辑;

一个连接到逻辑以根据概率等级生成一个允许信号的解码器;

连接到解码器并接收编码数据的多个比特发生器，其中每个比特发生器被专门用于至少一个不同的码，其中解码器根据概率等级使能其中多个比特发生器之一，这样所述一个比特流发生器解码编码数据。

90. 权利要求 89 中定义的解码器，至少其中多个比特发生器之一使用一个 R－码解码数据，且至少其中的多个比特发生器之一使用一个非－R－码解码数据。

91. 权利要求 89 中定义的解码器，其中的短运行长度比特发生器作为 R－码解码器运行。

92. 权利要求 89 中定义的解码器，其中长运行长度比特发生器包括一个短运行单元和一个长运行单元，其中短运行单元处理一个第一预确定长度的码，而长运行单元处理任何其余的比特，并确定是否有任何其余的比特被输出。

93. 一个解码输入数据的方法包括的步骤有：

提供一个上下文接收器；

为了获得概率状态，使用上下文接收器存取一个存储器；

根据概率状态生成一个概率等级；

使能其中多元个比特发生器之一，其中每个多个比特发生器被专门用于至少一个不同的码，这样只有所述不同的码被用于解码，这样所述多个比特流发生器之一解码编码的数据。

94. 一个编码输入数据的编码器系统包括：

一个提供一个上下文接收器的上下文模型；

一个连接到上下文模型以根据上下文接收器提供一个概率状态的存储器；

一个连接到存储器以根据概率状态生成一个概率等级的逻辑；

一个连接到逻辑以根据概率等级生成一个使能信号的编码器；

连接到解码器以接收输入数据的多个比特发生器，其中多个比特发生器被专门用于至少一个不同的码，此处编码器根据概率等级使能其中多个比特发生器之一，这样所说的多个比特流发生

器之一编码输入数据。

95. 权利要求 94 中定义的编码器，其中至少一多个比特发生器之一使用一个 R－码编码数据且至少其中一个比特发生器之一使用一个非－R－码编码数据。

96. 权利要求 94 中定义的编码器，其中短运行长度比特发生器作为 R－码编码器运行。

97. 权利要求 94 中定义的编码器，其中长运行长度的比特发生器包括一个短运行单元和一个长运行单元，其中短运行单元处理一个第一预定长度的码而长运行单元处理任何其余的比特，并确定是否有其余的比特被输出。

98. 一个编码输入数据的方法包括的步骤有：

提供一个上下文接收器；

为了获得一个概率状态，使用上下文接收器存取一个存储器；

根据概率状态生成一个概率等级；

使能多个比特发生器之一，其中多个发生器的每一个被专门用于至少一个不同的码，这样只有所述一个不同码被用于编码，这样的所说多个比特流发生器之一编码输入数据。

99. 一个解码编码数据的方法包括多个码字，所述方法包括的步骤有

装载一个计数值至一个与每个运行计数器关联的计数器，其中，相应于一个码字存储器容量的计数值在一个新的运行开始时被用于编码期间，当所述的每个运行计数器的一个新的码字被取出时，所述计数值被装载；

每次取出任何码字，计数值被减少；和

当计数器减到零时，清除与述新的编码字关联的比特发生器状态。

100. 权利要求 99 中定义的方法，其中每个运行计数器相应于一个 PEM 状态。

101. 权利要求 99 中定义的方法，其中每个运行计数器相应于一个上下文接收器。

102. 一个解码编码数据的方法包括多个码字，所述的方法包括的步骤有

每请求一个码字，一个计数器值递增，其中计数器值包括一个当前时间指示；

当第一码字被启动时，存贮计数器值，作为一个被存贮的时间指示；

将存贮的时间指示加上一个编码器存储器的大小与当前时间指示进行比较；

其中当当前时间指示大于存贮的时间指示加编码器存储器大小时，第一码字的比特发生器状态被清除，且一个第二码字被请求。

103. 权利要求 102 中定义的方法，其中存贮的时间指示包括一个时间标记。

104. 权利要求 102 中定义的方法，进而包括重新使用存贮的一个后继码字的时间指示的步骤。

105. 一个解码编码数据的方法包括多个码字，所述方法包括的步骤有

存贮相应于一个码字的一个索引，其中索引被存储在队列中，且码字被请求时索引被存贮；

标记队列中索引入口为无效；

存贮码字到入口并且若码字完成，标记入口为有效；

从队列中输出将被解码的数据，其中若队列入口有效，码字被输出供解码；

输出来自队列入口的数据并且若当从队列输出数据时，队列入口被标记为无效，则指示解码器数据无效，其中为了响应从标记为无效队列接收的数据，解码器清除比特发生器状态信息。

106. 一个解码编码数据的解码器，所述解码器包括：

一个提供上下文的上下文模型机制，其中上下文模型机制包括多个集成电路；

一个连接到上下文模型的存储器，用于存储状态信息，其中为了响应为上下文模型提供的每个上下文，存储器提供状态信息；和

连接到存储器的多个解码器，用于用来自存储器的状态信息解码码字，其中多个解码器使用多个R－码解码编码字，其中多个R－码包括至少一个未被一个最不可能符号跟随的最可能符号的非最长运行。

107. 权利要求106中定义的解码器，其中非最大长度运行计数具有一个唯一的可能码前缀。

108. 一个解码一个有多个码字的码流的系统，所述系统包括：

一个提供上下文的上下文模型机制，其中上下文模型机制包

括多个集成电路;

一个连接到上下文模型的存储器，用于存贮状态信息，其中为了响应上下文模型提供的每个上下文，存储器提供状态信息;和

连接存储的多个解码器，使用来自存储器的状态信息解码码字。

109. 权利要求 108 中定义的系统，其中的上下文模型机制包括至少一个提供来自其中多个集成电路之一的上下文模型和至少一个提供来自多个集成电路的另一个上下文的上下文模型。

110. 权利要求 109 中定义的系统，其中在所述多个集成电路之一上的所述的至少一个上下文模型包括一个零阶上下文模型。

111. 权利要求 108 中定义的系统，其中来自多个集成电路的上下文被直接提供给存储器。

112. 权利要求 108 中定义的系统，其中一个第一上下文的第一部分由一个集成电路提供，而第一上下文的第二部分由一个第二集成电路提供。

113. 权利要求 17 中定义的编码系统，其中比特发生器的比特发生器状态被更新，且进而比特发生器在被更新的状态被写入存储器之前被重用。

114. 权利要求 113 中定义的编码系统，其中比特发生器在一个读出－修改－写入周期的修改阶段内被重用。

115. 权利要求 17 中定义的编码系统，其中比特发生器在写入一个更新的状态至存储器中之前被重用时，生成一个非最小长度运行计数。

116. 权利要求 17 中定义的编码系统，其中的比特发生器使用定义为每个运行长度至少被一个未编码比特跟随的 R－码编码数据，这样同样运行长度的两个编码字绝不会在一行中被解码。

117. 一个解码一个具有多个码字的码流的系统，所述的系统包括：

一个提供上下文的上下文模型机制；

一个连接到上下文模型以存储状态信息的存储器，其中为了响应上下文模型提供的每个上下文，存储器提供状信息；和

连接到存储器的使用来自存储器的状态信息解码码字的多个解码器，其中多个解码器至少之一包括一个容时延解码器。

118. 权利要求 115 中定义的系统，其中至少多个解码器之一基于一个时延后可用的解码数据执行变长移位。

119. 权利要求 117 中定义的系统，其中多个解码器接收变长数据作为输入。

120. 权利要求 119 中定义的系统，其中多个解码器并行解码变长输入数据。

121. 权利要求 117 中定义的系统，其中多个解码器的输出被化分成定长交织字中。

说　明　书

编码与解码数据的方法和装置

此申请为美国专利申请系列号 08/172, 646, 题目为“数据并行编码与解码的方法和装置”, 于 1993 年 12 月 23 日归档的申请的一个延续部分, 而后者又是美国专利申请系列号 08/016, 035, 题目为“数据并行编码与解码的方法和装置”, 于 1993 年 2 月 10 日归档的申请的一个延续部分。

本发明涉及数据压缩和解压缩系统; 具体是涉及在压缩/解压缩系统中数据并行编码和解码的方法和装置。

今天, 数据压缩得到广泛应用, 尤其在存贮和传输大量数据时。目前已经存在很多不同的已成熟的数据压缩技术。压缩技术可分为两个宽的范畴, 即有损编码和无损编码。有损编码导致信息丢失, 以致于不能保证完整重建原始数据。无损压缩使所有信息得以保留, 并且数据压缩方式允许完整的重建。

在无损压缩过程中, 输入符号转化为输出码字。如果压缩成功, 以比输入符号数少的比特表示码字。无损编码方法包括字典编码方法 (例如, Lempel－Ziv), 运行编码, 枚举编码和熵编码。

熵编码包括任何应用已知或估计的符号概率试图把数据压缩近于熵极限的无损编码方法。熵编码包括 Huffman 编码, 算术编码和二进制熵编码。二进制熵编码器为无损编码器, 只按二进制 (是/非) 决定行动, 经常表达为最可能符号 (MPS) 和最不可能符号 (LPS)。二进制熵编码器的几个例子包括 IMB 的 Q－编码器

和一称为 B－编码器的编码器。欲知 B－编码器的更多情况，请看号码为 5, 272, 478, 题目为“熵编码的方法和装置”(J. D. Auen)，于 1993 年 12 月 21 日发布和转让给目前发明的合作代理人的美国专利。还请看于 1993 年 3 月 30 日在 Snowbird, UT 的 Proc 数据压缩会议中由 M. J. Gormish 和 J. D. Allen 所做“有限状态机二进制熵编码”节选第 449 页。B－编码器是一个利用有限状态机进行压缩的二进制熵编码器。

图 1 所示为一先有的用二进制熵编码器的压缩和解压缩系统的方框图。编码时，数据输入到上下文模型 (CM) 101。CM101 将输入数据译成一组或一序列二进制判定并为每一决定提供上下文接受器。二进制判定序列和其相应的上下文接受器均从 CM101 输出到概率估计模块 (PEM) 102。PEM102 接收每一上下文接受器并为每一个二进制判定产生一个概率预测。实际的概率预测典型地由一类表示，称为 PClass。每一 PClass 用于一个概率范围。PEM102 也判断此二进制判定 (结果) 是否处于更具可能状态 (即，是否此判定对应于 MPS)。比特流发生器 (BG) 模块 103 接收概率预测 (即，PClass) 和进行二进制判定是否有可能作为输入的判断。与此相应，BG 模块 103 产生一压缩数据流，输出零或更多比特，来代表原始输入数据。

为进行解码，CM104 为 PEM105 提供一个上下文接受器，而 PEM105 为根据上下文接受器给 BG 模块 106 提供概率等级 (PClass)。连接了 BG 模块 106 接收这个概率等级。作为响应概率等级和被压缩数据，BG 模块 106 返回一个表示此二进制决定 (即事件) 是否处于其最可能状态的比特。PEM105 接收此比特，根据

此接收比特更新概率预测，并把结果返回到CM104。CM104接收此返回比特并用返回比特产生原始数据和为下一个二进制判定更新上下文接受器。

用像IBM的Q－编码器和B－编码器那样的用二进制熵编码的解码器的一个问题是它们太慢，即使用硬件实现也如此。它们的操作需要一个单一的大型慢速反馈环路。为重述解码过程，上下文模型用过去被解码的数据生成一个上下文。概率预测模块用上下文产生一概率等级。比特流发生器用概率等级和经压缩的数据判断下一个比特是预期的还是非预期的结果。概率预测模块用预期的/非预期的结果产生一个结果比特（并为上下文更新概率预测）。此结果比特被上下文模型用来更新其过去的数据历史。解码一单个比特需要所有这些步骤。因为上下文模型在可提供下一个上下文之前必须等待结果比特来更新其历史，所以下一个比特的解码必须等待。希望避免解码下一比特之前必须等待完成反馈环路。换句话说，希望同时解码多个比特或码字，以提高压缩数据解码速度。

用二进制熵编码的解码器的另一个问题是必须处理各种长度的数据。在大多数系统中，将被解码的码字有各种长度。换句话说，其它系统编码各种长度的符号（未编码数据）。处理各种长度数据时，必须在比特级移位数据以为解码或编码操作提供下一个正确数据。在数据流的这些比特级处理可能需要昂贵的和/或慢速的硬件和/或软件。进一步地，现有技术的系统要求这种移位在限制解码器操作的有限时间反馈环路进行。从有限时间反馈环路去掉数据流的比特级处理利于利用并行提高速度。

本发明提供一个无损压缩和解压缩系统。本发明也提供了一个分别并行编码和解码数据的实时编码器和实时解码器。本发明的编码器和解码器形成了一个平衡的并行熵系统，允许以高速/低价硬件进行实时编码和实时解码。

本文描述了解压缩和压缩数据的方法和装置。本发明提供了一个编码器，它用于有一个解码由编码器生成的信息的解码器的压缩系统。本发明的编码器包括，用以产生相应于数据的码字信息的编码器。编码器还包括一个重新排序单元，它可发生与来自编码器的码字信息对应的编码数据流。重新排序单元由一个运行计数重新排序单元和一个比特组合单元构成，前者用于把码字排列成解码顺序，后者用于把各种长度的码字组合为固定长度的交织字并以解码器所要求的顺序输出这些固定长度交织字。

从下面的详细叙述和本发明各种实施例的附图可更全面地理解本发明，然而这些不意味着发明只局限于这些实施例，只为解释和理解方便。

图 1 是一个现有技术的二进制熵编码器和解码器的方框图。

图 2A 是本发明的解码系统的方框图。

图 2B 是本发明的编码系统的一个实施例的方框图。

图 2C 是一个并行处理上下文接收器的本发明的解码系统的实施例的方框图。

图 2D 是一个并行处理概率等级的本发明的解码系统的实施例的方框图。

图 3 表示本发明的非交织码流。

图 4 表示从一组示范性数据提取的交织码流的一个实施例。

图 5 是本发明的 R－编码器的概率预测表和比特流发生器的一个实例。

图 6 是本发明的一个编码器的实施例的方框图。

图 7 是本发明的一个比特发生器的实施例的方框图。

图 8 是本发明的重新排序单元的一个实施例的方框图。

图 9 是本发明的运行计数重新排序单元的一个实施例的方框图。

图 10 是本发明的运行计数重新排序单元的另一个实施例的方框图。

图 11 是本发明的比特组合单元的一个实施例的方框图。

图 12 是本发明的组合逻辑的一个实施例的方框图。

图 13 是本发明的编码器比特发生器的一个方框图。

图 14A 是本发明的解码系统的一个实施例的方框图。

图 14B 是本发明的解码器的一个方框图。

图 14C 是本发明的 FIFO 结构的一个实施例的方框图。

图 15A 表示本发明的解码流水线的一个实施例。

图 15B 表示本发明的解码器的一个实施例。

图 16A 是本发明的移位器的一个实施例的方框图。

图 16B 是本发明的移位器的另一个实施例的方框图。

图 17 是具有外部上下文模型的一个系统的方框图。

图 18 是具有外部上下文模型的另一个系统的方框图。

图 19 是本发明的一个解码器的一个实施例的方框图。

图 20 是具有独立比特发生器的一个解码器的一个实施例的方框图。

图 21 是本发明的一个比特发生器的一个实施例的方框图。

图 22 是本发明的长程单元的一个实施例的方框图。

图 23 是目前发明的短程单元的一个实施例的方框图。

图 24 是本发明的初始化和控制逻辑的一个实施例的方框图。

图 25 是采用一窥探解码器重新排序数据的一个实施例的方框图。

图 26 是重新排序单元的另一个实施例的方框图。

图 27 是用一合并队列重新排序单元的另一个实施例的方框图。

图 28 是应用本发明的一个高带宽系统的方框图。

图 29 是应用本发明的一个带宽匹配系统的方框图。

图 30 是应用本发明的一个实时视频系统的方框图。

图 31 表示本发明的编码数据存储器的一个实施例。

图 32 是本发明的解码时序图。

图 33 表示 MPS 概率和编码效率之间的关系图。

已经描述了并行编码与解码数据的方法和装置。接下来，为全面理解本发明的各个优选实施例，将详述各种具体细节，如具体的比特数，编码器数，具体的概率，数据类型等等。对于熟悉该技术的人来说，不用细述这些具体细节，就能实现本发明。另外，重要电路均以方框图的形式而不是详述来表示以避免对本发明不必要的误解。

下面详述的某些部分以计算机存储器内数据比特操作的算法和符号表示。这些算法描述和符号是那些在数据处理领域有专长的人用于把他们的工作内容最有效地传达给其他同行的方法。

一般情况下，这里的算法被认为是一种自我一致的可达到理想结果的步骤序列。这些步骤需要对物理量的物理处理。通常，(虽不必要)，这些量采取电或磁信号的形式被存贮，传输，组合，比较和处理。已经证明，为一般应用，把这些信号称为比特，值，元素，符号，字符，术语，数字或类似的名称比较方便。

然而，需记住所有这些和类似的术语均需与适当的物理量结合，只是用于这些量的方便标记。除非特别说明，本发明的整个叙述中所用的术语，如“处理”或“计算机计算”或“计算”或“判断”或“显示”或其它类似的均指一个计算机系统或类似的电子计算机设备的行为和过程，它们将计算机系统寄存器和存储器内的代表物理(电子)量的数据，处理和转换成在计算机系统存储器或寄存器或其它此类信息存贮，传输或显示设备内的同样代表物理量的其它数据。

目前发明也涉及执行此种操作的装置。这个装置可按需要而特制，或只包括一个由存贮在此计算机中的计算机程序可选择性启动或预置的通用目的计算机。这里的算法和显示不必然与任何具的计算机或其他装置相关。各种通用目的计算机均可用这里说明的程序或者为实现所需步骤方法而建造更专门的装置证明更为方便。下面叙述各种此类计算机所需的结构。此外，目前发明的叙述不参照任何特定的编程语言。应当理解各种编程语言均用于所述的本发明的实现。

本发明提供了一个并行熵编码系统。此系统包括一个编码器和一个解码器。在一个实施例中，此编码器对数据进行实时编码。类似地，在一个实施例中，本发明的解码器对数据进行实时解码。

实时编码器和实时解码器共同形成一个平衡的编码系统。

本发明提供了一个系统可并行无损地解码经编码的数据。用多个解码源并行解码数据。给多个解码源的每一个分配来自数据流的数据（如码字）以进行解码。数据流分配动态进行，并与与解码源解码数据同时发生，从而并行解码数据流。为了使数据分配以有效地利用解码源的方式进行，数据流是排序的。称为并行数据流。数据的顺序允许每一个解码源可解码任何或所有的编码数据而不用等待来自上下文模型的反馈。

图 2A 表示没有先有技术中慢速反馈环路的本发明的解码系统。输入缓冲器 204 接收编码数据（如码字）和来自解码器 205 的反馈信号并以预定的顺序如本发明的解码器 205 提供（如上下文接受器顺序）编码数据以使其解码。解码器 205 包括多个解码器（如 205A, 205B, 205C 等）。

在一个实施例中，解码器 205A, 205B, 205C 等之中的每一个被提供用于一组上下文的数据。解码器 205 中的每一个由输入缓冲器 204 提供用于其上下文组中每一个上下文接受器的编码数据。用这个数据，205A，205B，205C 等每个解码器均为其上下文接受器组产生解码数据。不需要上下文模型把被编码的数据和一特定的上下文接受器组联系起来。

被解码的数据被解码器 205 发送到解码数据存储器 207（如 207A, 207B, 207C 等）。注意解码数据存储器 207 可存贮即非编码也非未编码的中间数据，如运程计数。在这种情况下，解码数据存储器 207 以紧密的，然而非熵编码形式存贮数据。

连接上下文模型 206，它响应发送到解码数据存储器 207 的

反馈信号，以接收来自解码数据存储器 207（即 207A, 207B, 207C 等）的以前的解码数据，操作独立。因此，存在两上彼此独立的反馈环路，一个位于解码器 205 和输入缓冲器 204 之间，另一个位于上下文模型 206 和解码数据存储器 207 之间。由于避免了大的反馈环路，解码器 205（例如 205A, 205B, 205C, 等）中的解码器能够从输入缓冲存储器 204 中接收到与其相关的码字就将其解码。

上下文模型提供编码系统的存储器部分并根据存储器把一组数据（如一个影像）分为不同的类别（如上下文接收器）。在本发明中，上下文接受器被认为是独立的有序数据组。在一个实施例中，每组上下文接受器有其自己的概率评估模块并且每个上下文接受器都有其自己的状态（概率评估模块共用）。因此，每一个上下文接受器可用一不同的概率评估模块和/或比特流发生器。

这样，数据是有序的或并行的，数据流中的数据被分配到个别的编码器中进行解码。

为使数据流并行，数据可根据或上下文，概率，外涵，或码字序列（以码字为基础）等分割。对编码数据流重新排序与并行化彼此独立，并行化是在任何其它点用于使数据或概率并行的方法。图 2B 表示输入了由上下文模型（CM）求差分的数据的本发明的一个编码系统的并行编码部分。

参与图 2B，独立的上下文并行编码器部分包括上下文模型（CM ）214，概率评估模块（PEMs）215－217，和比特流发生器（BGs）218－220。连接了 CM214 以接收编码输入数据。还连接了 CM21 4到 PEM215 －217 。 PEM215 －217 也分别连接到 GB218－220，它输出相应的编码流 1，2 和 3，每个 PEM和 BG 对

包括一个编码器。因此表示了有三个编码器的并行编码器。虽然只表示了三个并行编码器，但可能用任何个编码器。

CM214 以和传统的 CM 同样方式把数据流分割为不同的上下文并把多个流发送到并行硬件编码源。单一上下文或上下文组被导入独立的概率评估器 (PEM) 215－217 和比特发生器 (BG) 218－219。每个 BG218－220 输出一个编码数据流。

图 2C 是本发明的解码系统的解码器部分的一个实施例。参考图 2C，它表示了具有 BG221－223，PEM224－226 和 CM227 的独立的上下文并行解码器。码流 1－3 相应地连接到 BG221－223 。BGs 221－223 也相应地连接到 PEM224－226 上。PEM224－226 连接到 CM227，它输出重建的输入数据。输入来自几个码流，表示为码流 1－3。给每个 PEM 和 BG 分配一个码流。BG221－223 的每一个返回一个比特，它表示这一二元判定是否处于其较可能状态，PEM224－226 利用这一状态返回解码比特 (如二元判定)。PEM224－226 的每一个与 BG221－223 之一相联系，指出要用哪个编码从其输入码流产生一解码数据流。CM227 通过以合适的序列从比特流发生器选择解码比特来产生一解码数据流，由此重新生成原始数据。因此，CM227 从适当的 PEM 和 BG 获得解压缩数据比特，结果把数据重新排序为原始顺序。注意此设计的控制与数据流方向流向相反。BG 和 PEM 可在CM227 需要之前就解码数据，超前一个或多个比特。选择性地，CM227 可能从一个 BG 和 PEM 要求 (但不接收) 一个比特，然后在使用这个最初被请求的比特前从其它的 BGs 和 PEMs 要求一个或多个比特。

设计图 2C 所示的结构用于紧紧连接 PEM 和 BG。IBM Q－编码器是一个很好的具有紧密连接的PEM 和 BG 编码器的实例。这两者之间的本地反馈环路对系统操作不构成基本限制。

在另一不同的设计中，PEM 可对数据求微分和发送到并行 BG 装置。因此只有一个 CM 和 PEM，BG是重复的。自适应 Huffman 编码和有限状态机编码可以这种方式应用。

图 2D 表示一类似的用 PEM 对数据求微分和送到并行 BG 的解码系统。在这种情况下，并行处理概率等级，每个比特流发生器被分配给特定的概率等级和接收结果。参考图 2D，编码数据流 1－3 被连接到多个比特流发生器(如 BG232, BG233, BG234, 等)之一，连接了发生器以接收数据流。比特流发生器的每一个连接到 PEM235。PEM235 又连接到 CM236。在这个结构中，比特流发生器的每一个均解码编码数据，解码结果由 PEM235 选择 (而不是由 CM236)。每个比特流发生器均从一与概率等级 (即，编码数据可能来自任何上下文接受器) 相关的源接收编码数据。PEM 235 用概率等级选择比特流发生器。概率等级由 CM236 提供的上下文接受器指明。以这种方式，通过并行处理概率等级产生编码数据。

本发明的并行解码系统有很多实现。在一种实施例中，对应于多个上下文接受器的编码数据流可交织成一个由各种编码器需要进行排序的流。在本发明的一个实施例中，编码数据被排序成每个编码器被不断地供以数据，即使编码数据在一个流中被传送到解码器。注意本发明可对所有类型数据操作，包括图象数据。

通过使用可廉价地在集成电路中复制的小的简单的编码器，

编码数据可被迅速地并行解码。在一个实施例中，编码器在使用现场可编程门形阵列(FPGA)芯片或一标准的专用集成电路单元(ASIC)芯片的硬件上实现。并行化和简单的比特流发生器的结合允许编码数据解码以高于原来已有技术的解码器的速度进行，同时保持或超过已有技术的解码系统的压缩效率。

有很多设计问题影响系统操作。下面会提到其中的几个。然而，图2B和2C(和2D)表示的实施例利用多个码流。可以想象能满足这一实施例的具有并行通道的系统：多条电话线路，多头盘驱动等。在某些应用中，只用到或方便使用一个通道或方便。确实，如果需要多重通道，可能因为单个码流的突发性，带宽不能很好利用。

在一个实施例中，码流被串连和连续发送到解码器。一个前缀头有一指针指向每一比特流的起始比特位置。图3表示这种数据结构的一个实施例。参考图3，三个指针301－303指出编码流1，2和3的相应的串连编码的起始位置。在连接到解码器的缓冲器中有完整的压缩数据文件。如果需要，经合适的指针可从适当的位置检索码字。然后指针更新为在那个编码流中的下一个码字。

注意这个方法需要把一个完整的码帧存贮在解码器中，用于编码器的实际目的。如需要一实时系统或突发性低的数据流，则两个帧缓冲存储器可被用来安装在编码器和解码器上。

注意，一个解码器以一给定的判定顺序解码编码字。并行解码时，到码流的请求的顺序是确定的。因此，如果来自并行码流的码字能在编码器以正确的顺序交织，则单一的编码流就足够了。

码字按时地以相同的顺序被送到解码器。在编码器中，一个解码器模块判定码字顺序并把它们装配成单一流。这个模块可能是一个实际的解码器。

当数据长度可变时，把数据送到并行解码器件会出现问题。折开可变长度码字流需要用一移位器定位码字。用于硬件实现时移位器经常选价高昂和/或慢。移位器的控制取决于特定的编码字的长度。这个控制反馈环路使得可变长度移位不能迅速进行。如果拆开比特流的过程在速度不足以跟上多个解码器的单一移位器中进行，则不能实现用单一流馈送多个解码器的优点。

本发明所提供的解决方法是把分配编码数据到并行编码器的问题与用于解码的可变长度的编码字的定位分离开来。每个独立的编码流中的码字被装配或固定长度字，称为交织字。在通道的解码器端，这些交织字可被分配到具有快速硬线数据线路和一简单控制电路的并行解码器单元。

交织字长度大于最大的码字长度非常方便，因为至少有足够的能构成一个码字的比特包含在每个交织字中。交织字可包含可能的码字和部分码字。图 4 表示一组并行码流的交织的实施例。

这些字按解码器的需要进行交织，每个独立的解码器接收整个的交织字。现在比特移位在每个解码器内部进行，保持系统的并行性。注意在图 4 中，每个交织字的第一个码字是此组中留下的最低编码字。例如，来自码流 1 的第一交织字，以最低码字（即 # > 1）开始。然后跟随码流 2 的第一交织字，再跟随码流 3 的第一交织字。然而，不完全包含在一个已经排序的交织字中的下一个最低码字是# 7。因此，流中的下一个字是码流 2 的第二交织

字。

在另一个实施例中，有序交织字组(如在流 1 以# 8 起始的码字，在流 2 以# 7 起始的码字，在流 3以# 11 起始的码字) 被插入交织码流的顺序建立在前组交织字(如在流 1 以# 1 起始的码字，在流 2以# 2 起始的码字，在流 3 以# 4 起始的码字)的基础上，从具有最低号的第一码字的交织字到具有最高号的第一编码字的交织字为止进行排序。因此，在这种情况下，因为以码字# 1 起始的交织字为第一，然后流 1 中的下一个交织字是将要被插入交织流的第二组交织字的第一个，跟随着流 2 中的下一个交织字和流 3 的下一个交织字。注意当第二组交织字被插入交织流后，流 2 中的下一个交织字将是下一个插入流中的交织字，因为码字# 7 是第二组交织字中的最低码字 (流 1的码字# 8 和流 3 的编码字# 11 跟随其后)。

所有设计选择都用实际解码器作为数据流模型，并且延迟以形成交织流。无论如何对于具有编码器和解码器的双工系统并不造成很大的花销。注意这对于任何在一确定性顺序中消耗的任何并行可变长度(或不同尺寸)的数据字可通用。

本发明可应用现有的如 Q－编码器或 B－编码器作为比特流发生元件，它们可并行复制。然而，也可用其它码和编码器。本发明采用的编码器和相关码是简单的。

在本发明中，应用具有简单码而不是复杂码 (如 Q－编码器所用的算数码或 B－编码器所用的多一状态码) 的比特流发生器有很多优点。简单码的好处在于硬件实现比复杂码快和简单得多，并且只需较少的硅。

本发明的另一个优点是可提高编码效率。对于每种概率，用有限量状态信息的码不能恰好达到 Shannon 熵极限。允许单一比特流发生器处理多种概率或上下文的已知技术的编码的硬件实现具有限度，会降低效率。去掉多个上下文或概率等级的限度，允许使用更接近满足 Shannon 熵极限的码。

本发明的一个实施例便所采用的码 (和编码器) 称为 R－码。R－码为自适应码，可把大量不同数目的相同的输入符号转化为一个码字。在一个实施例中，R－编码被参数化，这样可由一个单一解码器设计处理多种概率。此外，本发明的 R－码可由简单高速的硬件解码。

在本发明中，R－码由 R－编码器用于进行编码或解码。在一个实施例中，R－编码器是比特流发生器和概率评估模块的组合。例如，在图 1 中，一个 R－编码器可能包括概率评估模块 102 与比特流发生器 103 的组合和概率评估模块 105 与比特流发生器 106 的组合。

码字代表最可能符号 (MPS) 的运行。一个 MPS 代表一个概率大于 50% 的二元判定的结果。另一方面，最不可能符号 (LPS) 代表概率小于 50% 的二元判定结果。注意当两个结果概率相等时，只要对编码器和解码器做同样的指定，哪个被指定为 MPS 或 LPS 并不重要。表 1 表示在压缩文件中的结果比特序列，对于给定的参数称为 MAXRUN。

为了编码，由一简单计数器计算运行中 MPS 的数目。如果这个数等于 MAXRUN 计数值，则一个 0码字被发送进码流中且计数器被复位。如果遇到 LPS，则专门描述在 LPS 之前的 MPS 符号

数，跟随着比特 N 的 1 被发送进码流。(注意有很多方法指定 N 比特来描述运程长度)。同样计数器又复位。注意 N 所需的比特数取决于 MAXRUN 的值。也要注意码字的 1 的被码也可用。

为了解码，如在码流中的第一个比特是 0，则 MAXRUN 的值输入 MPS 计数器且 PLS 指示被清除。然后 0 比特被放弃。如果第一个比特是 1，则随后的比特受到检查以提取比特 N 且适当的计数 (N) 被输入 MPS 计数器，并且 LPS 指示器被设置。然后包括 IN 编码字的码流比特被放弃。

R－码以表 1 的规则发生。注意给定 R－码 RX(K) 的定义由 MAXRUN 给出。例如:

RX(K) 的 MAXRUN = $X * 2^K - 1$

因此

R2(K) 的 MAXRUN = $2 * 2^2 - 1$

R3(K) 的 MAXRUN = $3 * 2^K - 1$

等等……

注意 R－码是 Golomb 码的子码。也要注意 Rice 码只用 R2(•)码。本发明的 R－码允许使用 R2(K) 和 R3(K) 码，如果需要，也可使用其它的 Rn (K) 码。在一个实施例中，使用了 R2 (K) 和 R3(K) 码。注意对于 n = 2 和任何奇数 (如 R2, R3, R5, R7, R9, R11, R13, R15) 存在 Rn。在一个实施例中，对于R2(K) 码，运行计数 r 在 N 编码；运行计数 r 用 K 比特描述，这样，用 K + 1 比特表示 1N。也在一实施例中，对于 R3(K) 码，比特 N 可包含 1 比特以指出 $n < 2^{(K-1)}$ 还是 $n \geq 2^{(K-1)}$ 和或用 K－1 或 K 比特指示运行计数 r，这样，分别用 K 或 K + 1 比特代表变量 N。在其它实施例中，N

的1的被码可用于码字。在这种情况下，MPS 趋向于产生具有很多 0 的码流，而 LPS 趋向于产生具有很多 1 的码流。

表 2，3，4 和 5 说明用于本发明的一个实施例的某些高效 R－码。应注意的是其它运行长度编码也可用于本发明。替代 R2 (2) 的一个运行长度编码实例在表 6 表示。表 7 和表 8 表示用于一实施例中的码的例子。

表 1 比特－发生编码

码字	意义
0	MAXRUN 连续 MPSs
1N	N 个连续 MPSs，随后是 LPS LPS，N < MAXRUN

表 2 - R2(0)

未解码数据	码字
0	0
1	1

表 3 - R2(1)

未解码数据	码字
00	0
01	10
1	11

表 4 - R3(1)

未解码数据	码字
000	0
001	100
01	101
1	11

表 5 - R2(2)

未解码数据	码字
0000	0
0001	100
001	101
01	110
1	111

表 6 可选 R2(2)

可选	R2(2)
0000	0
0001	111
001	101
01	110
1	100

表 7 - 另一种 R3 (2) 码

优先	R3(2)
000000	0
000001	1000
00001	1010
0001	1001
001	1011
01	110
1	111

表 8　另一选择 R2(2) 码

优选	R2(2)
0000	0
0001	100
001	110
01	101
1	111

在一个实施例中，R2 (0) 码执行无编码：一个输入 0 编码为 0 且一人输入 1 编码为 1 (或反之) 且对概率为 50% 时是最佳的。本优先的实施例的 R2 (1) 码对于概率接近 0. 707 (即 70. 7%) 是最佳的，且 R3 (1) 对于 0. 794 的概率 (79. 4%) 是最佳的。R2 (2) 码对于 0. 841 的概率 (84. 1%) 是最佳的。下面表 9 说明接近最佳的运程长度码，这里概率偏差由下列等式定义：

概率偏差 = －log2 (LPS)

表 9

概率	概率度	最优 Golomb 码
.500	1.00	R2(0)
.707	1.77	R2(1)
.841	2.65	R2(2)
.917	3.59	R2(3)
.958	4.56	R2(4)
.979	5.54	R2(5)
.989	6.54	R2(6)
.995	7.53	R2(7)
.997	8.53	R2(8)
.999	9.53	R2(9)

注意，这些码接近最佳，在于由概率偏差指明的概率范围相对较平均地覆盖这一空间，尽管最佳概率在较高 K 值时不像在较低 K 值时有那么大差别。

请参考 R－码最佳的概率。事实上，只有 R2 (2) 满足熵曲线。实际的考虑是在给定等级内在什么概率范围特定 R－码比所有其它 R－码好。下表提供在 R2 码等级和 R2 及 R3 码等级的概率范围。

对于从 0 到 12 的 R2 码等级，范围位于下表 10 中。例如，只用 R2 时，当 0.50≤概率≤0.6180 时，R2 (0) 是最好的。类似地，当 0.6180≤概率≤0.7862 时，R2 (1) 是最好的。

表 10

表 10 . 从 0 到 12 的 R2 码

码	概率
R2(0)	0.6180
R2(1)	0.7862
R2(2)	0.8867
R2(3)	0.9416
R2(4)	0.9704
R2(5)	0.9851
R2(6)	0.9925
R2(7)	0.9962
R2(8)	0.9981
R2(9)	0.9991
R2(10)	0.9995
R2(11)	0.9998
R2(12)	–

对于 R2 及 R3 编码级，结论在下表 11。例如，应用 R2 及 R3 码时，当 0.6180≤概率≤0.7549 时，R2(1)最好。

表 11

表 11　长度小于或等于 13 比特的 R2 和 R3 码

码	概率
R2(0)	0.6180
R2(1)	0.7549
R3(1)	0.8192
R2(2)	0.8688
R3(2)	0.9051
R2(3)	0.9321
R3(3)	0.9514
R2(4)	0.9655
R3(4)	0.9754
R2(5)	0.9826
R3(5)	0.9876
R2(6)	0.9913
R3(6)	0.9938
R2(7)	0.9956
R3(7)	0.9969
R2(8)	0.9978
R3(8)	0.9984
R2(9)	0.9989
R3(9)	0.9992
R2(10)	0.9995
R3(10)	0.9996
R2(11)	0.9997
R3(11)	0.9998
R2(12)	-

对于一个固定 K, R2 (K) 称为运程长度码。然而，一个固定 K 只对一个概率范围最好。已注意到当接近一最佳概率编码时，根据本发明，R－编码用基本具有相同频率的 0 和 1N 码字。换言之，一半时间内，本发明的 R－编码器输出一个码，而另一半时间输出另一个码。通过检查 0 和 1N 码字的数量，可做出是否使用了最佳编码的判定。即，如果输出太多的 1N 码字，则运程长度太长；另一方面，如果输出太多的 0 码字，则运程长度太短。

由 Langdon 所用的概率评估模块检查每个码字的第一个比特以决定源概率是在当前评估以上还是以下。请看 IMB 技术发布通告，1983 年 12 月第 26 卷第 7B 号由 G. G. Langton 所作的“一个自适应的运程长度编码算法”。以这一判定为基础，K 增大或减小。例如，如果看到指示 MPS 的码字，概率评估太低。因此，根据 Langton 方法，对每个 0 码 K 增加 1。如果看到一个指示小于 MAXRUNMPS 的码字，并跟随一个 LPS (如 1N 码字)，则概率评估太高。因此根据 Langton 方法，对每个 1N 码字，K 减小 1。

本发明允许进行比简单的每个码字给 K 加 1 或减 1 更复杂的概率评估。本发明包括一个概率评估模块状态，它确定要用的码。采用状态表或状态机给状态指定码。

在本发明的一个实施例中，概率评估随每次码字输出变化状态。因此，根据一个码字是否以一个 0 或一个 1 开始，概率评估模块增大或减小概率的评估。例如，如果输出“0”码字，则出现对 MPS 概率评估的增加。另一方面，如果输出“1”码字，则 MPS 概率的评估减少。

先有技术的 Langton 编码器码只用 R2 (K) 码并为每个码字增大或减小 K。本发明则根据状态表或状态机应用 R2 (K) 和 R3 (K) 码，允许自适应速率按应用调节。就是说，如果有少量的静止数据，自适应必须较快以使编码更佳，而如果有大量静止数据，自适应时间可长些，使编码选择得可对剩下的数据取得较好的压缩效果。注意当发生可变次数状态变化时，应用特性特征也可能影响自适应速率。由于 R－码的性质，R－码的评估简单，需硬件小，但却非常强大。图 33 表示编码效率（相对熵正常化的码长）与 MPS 概率对比图。图 33 说明本发明的一些 R－码如何覆盖概率空间。作为一例，图 33 表示了对于约为 0.55 的 MPS 概率，R2 (0) 码的效率是 1.01 (或 1% 劣于) 熵极限。相对地，R2 (1) 码的效率是 1.09 (或 9% 劣于) 熵极限。这个例子表明对于这个特定的低概率情况，用错误码导致编码效率损失 8%。

组合 R3 (K) 码允许以更高的效率覆盖更多的概率空间。图 5 表示对应本发明的一个概率评估状态表的例子。参考图 5，概率评估状态表既表示一状态计数器又表示与表中每一独立状态相关的码。注意表中包括正负两种状态。表中有 37 个正状态和 37 个负状态及零状态。负状态表示与正状态不同的 MPS。在一个实施例中，当 MPS 为 1 时可用负状态，而当 MPS 为 0 时可用正状态，或反之。注意图 5 表中所示仅为一例，其它表可能有更多或更少的状态和一个不同的状态分布。

初始时，对于概率评估为 0.50，编码器处于 0 状态，为 R2 (0) 码 (即元编码)。在处理了每个码字后，根据码字的第一个比特，状态计数男递增或递减。在一个实施例中，一个 0 码字增加了状态

计数器的值；而以 1 开始的码字减小了状态计数器的值。因此，每个码字通过状态计数器引起一状态变化。换句话说，概率评估模块改变状态。然而，连续状态可与同一码联系起来。在这种情况下，不用改变每个码字的码就可完成概率评估。换句话说，为每个码字改变状态，然而在一定时间内，此状态被映射到同一概率上。例如，状态 +5 到 −5 均用 R2(0) 码，而状态 6 到 11 和 −6 到 −11 应用 R2 (1) 码。用本发明的状态表，概率评估允许以非线性状态与同一编码器共存。

应注意的是，对于较低的概率包括了更多的具有同样一 R − 码的状态。原因是在低概率情况下用错误码的效率损失大。运程长度码状态表的性质是在每个编码字后在状态间变换。在一个设计用来随状态变化改变码的状态表中，当在较低概率情况下在状态之间变化时，码在一个非常接近熵效率极限码和一个远离熵效率极限码之间变动。因此其补偿(从编码数据比特数的角度)是可导致在状态之间的变迁。先有技术的概率评估模块，如 Langton 的概率评估元件由于这个补偿而损失性能。

在较高概率运程长度码中，对处于错误码的补偿不这样大。因此，在目前发明中，对较低概率增加了额外状态，使在两个正确状态之间的变动的改变增加，因而降低编码的效率。

注意在一定的实现中，编码器可有初始概率评估状态。换句话说，编码器可能以一预设状态，如状态 18，启动。在一个实施例中，可用一个不同的状态表使某些状态被用于前几个符号，以允许快速自适应，而第二个状态表可被用于剩余符号的慢速自适应，以允许概率评估的微调。以这种方式，编码器能够在编码过程

马上使用更有效的码。在另一个实施例中，码流可为每一个上下文规定一个起始概率评估。在一个实施例中，不按一固定数(如1)进行递增和递减。相反地，可根据已遇到的数据量或数据变化量(稳定性)，使概率评估状态增加一个可变量。下述表21－25是这种表的实例。

如果状态表是对称的，如图5例子所示，只有其一半(包括零状态)需要被存贮或在硬件上实现。在一个实施例中，利用对称性，状态数以信号值(1S)补数方式存贮。以这种方式，可通过采用1的补数的绝对值用表来确定状态和检查信号，以确定MPS到底是1还是0来。这允许所用递增和递减状态的硬件减少，因为状态的绝对值用来索引此表并且1的补数绝对值的计算量很小。在另一个实施例中，为了更高的硬件效率，硬件表可由一硬线连接的或可编程的状态机代替。硬线连接的状态到码的转换器是状态表的一个应用。

本发明提供一个平衡并行熵编码系统。并行熵编码系统包括运行于高速/低成本硬件的实时编码和实时解码。本发明可用于很多无损编码应用，包括但不只限于可写光盘或磁盘数据的实时压缩/解压缩，计算机网络数据的实时压缩/解压缩，在多功能设备(如复印，传真，扫描，打印等)的压缩帧存储器中的图象数据的实时压缩/解压缩和声音数据的实时压缩/解压缩。

规定编码器的性能需要注意一些事情。设计是直接了当的，可以直接设计为给定足够的快速编码数据通道的源数据获得某个速率的编码器。然而在很多应用中，编码器的目标是有效利用编码数据通道。编码数据通道的利用受原始数据接口的最大脉冲

速率，编码器速率和对数据压缩的影响。对于某些取决于在编码器中的缓冲存贮量的内部数据量要考虑这些效果的影响。理想状态是有一个编码器在有效利用编码数据通道的同时能保持编码器速度和高压缩效果且仍然适合最大脉冲速率。

下面描述本发明的编码器。也描述可能与编码器一起使用的解码器。

图 6 为本发明的编码系统的方框图。在一个实施例中，本发明的编码器进行实时编码。参考图 6，编码系统 600 包括一个编码器 602，它连接到一上下文模型 (CM) 以及状态存储器 603，用以响应原始数据 601 以码字信息 604 的方式生成编码信息。码字信息 604 由重新排序单元 606 接收，它又连接到重新排序存储器 607 上。响应码字信息 604，重新排序单元 606 与重新排序存储器 6 07 共同生成编码数据流 608。应注意的是编码系统 600 不限于对码字操作，在其它实施例中，用目前发明的方法，它可对离散模拟波形，可变长度比特码型，通道符号，字母，事件等操作。

编码器 602 包括一个上下文模型 (CM)，一个概率评估机 (PEM) 和一个比特流发生器 (BG)。编码器 602 中的上下文模型和 PEM (概率评估机) 与解码器中的基本相同 (除数据流方向外)。编码器 602 的比特发生器与解码器比特发生器类似，下面有述。编码器 602 的编码结果是代表原始数据的零或更多比特的输出。在一个实施例中，比特流发生器的输出也包括一个或多个控制信号。这些控制信号为比特流中的数据提供一控制通道。在一个实施例中，码字信息可由一个运行指示开始，一个运行指示终止，一个码字和一个识别编码字的运行计数 (无论是由上下文还是概率

等级)的索引构成。下述本发明的比特流发生器的一个实施例。

重新排序单元606接收由编码器602的比特流发生器(如果有)发生的比特和控制信号并发生编码数据。在一个实施例中,由重新排序单元606输出的编码数据包括一交织字流组成。

在一个实施例中,重新排序单元606执行两种功能。重新排序单元606把码字从编码器产生的运行终点移动到解码器需要的运行起始点,并把可变长度码字组合成固定长度交织字和以解码器需要的合适顺序将其输出。

重新排序单元606使用一个暂时性重新排序存储器607。在一个实施例中,其中编码在一工作站上进行,暂时性重新排序存储器607的容量可大于100兆字节。在本发明的平衡系统中,暂时性重新排序存储器607要小得多(如大约1K字节)且是固定的。因此,在一个实施例中,用一个固定量的存储器进行实时编码,即使这将增加解码器或比特率(如当输出在运行完成之前进行时)需要的存储器。本发明的解码器采用例如隐式的,显式的或流信令(如下所述)能够判定重新排序单元的有限存储器的效果。重新排序单元606有有限存储器用于重新排序,但"需要的"存储器不是无限的。必须考虑用于运行终止到运行起点队列和交织字重新排序的有限存储器的效果。

在一个实施例中,本发明的编码系统(和相应的解码系统)用单一的集成电路芯片进行编码(或解码)。在另一个实施例中,一个单一集成电路包括本发明的编码器系统,包括其编码器和解码器及存储器。可加一独立的外部存储器帮助编码。一个多芯片模块或集成电路可既包括编码/解码硬件又包括存储器。

本发明的编码系统可能试图将有效带宽增加到N倍。如果取得的压缩小于N: 1，则编码数据通道被完全利用，但获得的有效带宽增加只等于压缩率。如果取得的压缩大于N: 1，则取得了具有可写的额外带宽的有效带宽。在两处情况下，取得的压缩必须超过由出现在编码系统的缓冲存储量定义的数据的内部区域。

图7表示本发明的编码器的比特发生器的一个实施例。连接比特发生器701以接收一个概率等级和一个未编码比特（如一个MPS或LPS指示）作为输入。响应此输入，比特发生器701输出多个信号。其中的两个输出为控制信号，指示运行的起始和一个运行的终止（每个编码字代表一个运行）分别为起始信号711及终止信号712。一个运行可能同时起始和终止。当一个运行起始或终止时，“索引”输出713包括一个未编码比特的概率等级（或上下文）的指示。在一个实施例中，索引输出713代表比特的概率等级与系统的存贮单元标识的组合，在此系统中，每个概率等级在存储器的几个单元中重复。码字输出714用于当一个运行终止时从比特发生器701输出码字。

存储器702连接到比特发生器701上并包含给定概率等级的运行计数。在比特发生过程中，比特发生器701用索引（如概率等级）从存储器702中读取。从存储器701中读取完毕后，比特发生器701进行比特发生如下。首先，如果运行计数为零，则确定起始信号711，指示一个运行的开始。然后，如果未编码比特等于LPS，则确定终止信号712，指示此运行的终止。同样，如果未编码比特等于一个LPS，则设定编码字输出714为指示码字是1N码字且运行计数已清除，如设定为零（由于为运行终止）。如果未编

码比特不等于 LPS，则运行计数加 1 且一个测试确定运行计数是否等于此码的最大运行计数。如果如此，则确定终止信号 712。码字输出 714 设定为零且运行计数清除（如运行计数设定为零）。如果测试确定此运行计数不等于码的最大值，则运行计数递增。注意索引信号 713 代表作为输入接收的概率等级。

在本发明中，1N 码字发生的方式为不用任何额外信息就可决定其长度。图 12 表示解码器和编码器的 R3 (2) 码字的 1N 码字代表。解码器期望在“1N”码字中的“1”比特为 LSB，且“N”计数部分处于适当的 MSB…LSB 顺序。在解码器顺序中，如果不知道所用的具体码，就不可能从零填充中区分出可变长度码字。在编码器顺序中，码字是倒置的，且最重要的“1”比特部分指示“1N”码字的长度。为了以编码器顺序发生码字，计数值的补数必须倒置。这可通过倒置 13－比特计数然后使其移位与 LSB 对齐来实现。如下详述，比特组合单元把码字倒置成解码器顺序。然而，由于它必须进行移位，此码字的倒置不增加比特组合单元 606 的复杂性。

表 12－R3 (2) 码字的“1N”码字代表

未编码数据	码字	计数值反向	解码器顺序	编码器顺序
			（计数值被划线）	
000000	0		0000000000000	0000000000000
000001	1000	00	000000000<u>00</u>01	000000000<u>10</u>00
00001	1010	01	00000000<u>01</u>01	00000000<u>10</u>10
0001	1001	10	00000000<u>10</u>01	00000000<u>10</u>01
001	1011	11	00000000<u>11</u>01	00000000<u>10</u>11
01	110	0	000000000<u>00</u>11	000000000<u>11</u>0
1	111	1	000000000<u>11</u>11	000000000<u>11</u>1

对于 R3 码，发生“N”码字也需要跟随“1”的比特指示是出现短的还是长的计数。

通过使用存储器的多个单元，目前发明允许流水作业流水线操作。例如，在多端口存储器的情况下，在存储器对一个未编码比特进行读操作的同时，也从存储器对前一未编码比特进行写操作。

本发明的编码器比特发生器的一个实施例包括一个 FPGA，此设计处理一直到 R2(12) 的所有 R 2 和 R3 码。下面列出 AHDL (Altera 硬件描述语言) 源码。

如图 13 所示，此设计由多部分组成。首先，“ENCBG”1301 是此设计的主要部分，其具有处理运行的起始，终止和继续进行的逻辑。其次，“KEKPAND”用于把概率等级扩展成最大运行长度，各种长度掩码，和 R3 码的第一个长码字的长度。“KEXPAND” 1303 与有相同名称的解码器功能相同。第三，“LPSCW” 1304 部分把一个计数值和概率等级的信息用做输入并发生适当的 “1N” 码字。

此设计应用两个流水线阶段。在第一流水线阶段中，计数被递增，概率等级被扩展，并进行长 R3 码字的提取和比较。在第二流水线阶段中，进行所有的其它操作。

encbg.tdf

```
TITLE "Bit Generator for the encoder";

INCLUDE "kexpand.inc";
INCLUDE "lpscw.inc";

SUBDESIGN encbg
(
        k[3..0],
        r3,
        bit,
        count_in[12..0],
        clk
```

```
                                : INPUT;
        start_run,
        end_run,
        index[4..0],
        count_out[12..0],
        codeword[12..0]
                                : OUTPUT;

)
VARIABLE
        k_q[3..0],
        r3_q,
        k_qq[3..0],
        r3qq,
        bit_q,
        bit_q,
        count_in_q[12..0],
        start_run,
        end_run,
        start_run_q,
        index[4..0],
        count_out[12..0],
        count_plus[12..0],
        max_r1[12..0],
        codeword[12..0]
                                : DFF;
        kexpand_                : kexpand;
        lpscw_                  : lpscw;

BEGIN
        lpscw_.clk              = clk;
        k_q[].clk               = clk;
        r3_q.clk                = clk;
        k_qq[].clk              = clk;
        r3_qq.clk               = clk;
        bit_q.clk               = clk;
        bit_qq.clk              = clk;
        count_in_q[].clk        = clk;
        start_run.clk           = clk;
        end_run.clk             = clk;
        start_run_q.clk         = clk;
        index[].clk             = clk;
        count_out[],clk         = clk;
        count_plus[].clk        = clk;
        max_ri[].clk            = clk;
        codeword[].clk          = clk;
```

```
        k_q[]               = k[];
        r3_q                = r3;
        k_qq[]              = k_q[]
        r3_qq               = r3_q;
        bit_q               = bit;
        bit_qq              = bit_q;
        count_in_q[]        = count_in[];
        count_plus[]        = count_in_q[] + 1;
        start_run           = start_run_q;

        stat_run_q          = (count_in_q[] == 0);
        index[0]            = r3_qq;
        index[4..1]         = k_qq[];

        kexpand_.k_reg[]    = k_q[];
        kexpand_.r3_reg     = r3_q;
        lpscw_.r3           = r3_q;
        lpscw_.k_q[]        = k_q[] ;
        lpscw_.r3_q         = r3_qq;
        lpscw_.count[]      = count_in_q[];
        lpscw_.mask[]       = kexpand_.mask[] ;
        lpscw_.r3_split[]   = kexpand_.r3split[] ;
        lpscw_.maxrl_q[]    = max_rl[] ;
        max_rl[]            = kexpand_.maxrl[] ;

IF      (bit_qq) THEN                                   % LPS %
        end_run             = VCC;
        count_out[]         = 0;
        codeword[]          = lpscw_.cw[];

    ELSIF (count_plus[]     == max_rl[]) THEN
        end_run             = VCC;
        count_out[]         = 0;
        codeword[]          = 0;
    ELSE
        end_run             = GND;
        count_out[]         = count_plus[];
        codeword[]          = 0;
    END IF;
END;
```

lpscw.tdf

```
SUBDESIGN lpscw
(
```

```
        r3,
        k_q[3..0],
        r3_q,
        count[12..0],
        mask[11..0],
        r3_split[10..0],
        maxrl_q[12..0],
        clk
                                : input;
        cw[12..0]
                                : output;
)
VARIABLE
        temp[12..0]             : NODE;
        temp_rev[12..0]         : NODE;
        temp_sh[12..0]          : NODE;
        split[11..0]            : NODE;
        r3_long                 : DFF;
        count_minus[11..0]      : DFF;
        mask_q[11..0]           : DFF;
        count_q[12..0]          : DFF;
    BEGIN
        r3_long.clk             = clk;
        count_minus[].clk       = clk;
        mask_q[].clk            = clk;
        count_q[].clk           = clk;
        split[10..0]            = r3_split[];
        split[11]               = GND;
        r3_long = (r3) AND (count[11..0] .= split[]);
        count_minus[]           = count[11..0] - split[];
        mask_q[]                = mask[] ;
        count_q[]               = count[] ;

        % ———————— pipeline stage ———————— %

        IF (r3_long) THEN
            temp[11..0] = (count_minus[]) XOR mask_q[];
        ELSE
            temp[11..0] = count_q[11..0] XOR mask_q[] ;
        END IF;
        temp[12] = GND;

        temp_rev[0] = temp[12] ;
        temp_rev[1] = temp[11] ;
        temp_rev[2] = temp[10] ;
        temp_rev[3] = temp[9] ;
        temp_rev[4] = temp [8] ;
```

```
temp_rev[5] = temp[7] ;
temp_rev[6] = temp[6] ;
temp_rev[7] = temp[5] ;
temp_rev[8] = temp[4] ;
temp_rev[9] = temp[3] ;
temp_rev[10] = temp[2] ;
temp_rev[11] = temp[1] ;
temp_rev[12] = temp [0] ;

CASE k_q[] IS

    WHEN 0 =>     temp_sh[] = 0;
    WHEN 1 =>     temp_sh[0] = temp_rev[12] ;
                  temp_sh[12..1] = 0;
    WHEN 2 =>     temp_sh[1..0] = temp_rev[12..11] ;
                  temp_sh[12..2] = 0;
    WHEN 3 =>     temp_sh[2..0] = temp_rev[12..10] ;
                  temp_sh[12..3] = 0;
    WHEN 4 =>     temp_sh[3..0] = temp_rev[12..9] ;
                  temp_sh[12..4] = 0;
    WHEN 5 =>     temp_sh[4..0] = temp_rev[12..8] ;
                  temp_sh[12..5] = 0;
    WHEN 6 =>     temp_sh[5..0] = temp_rev[12..7] ;
                  temp_sh[12..6] = 0;
    WHEN 7 =>     temp_sh[6..0] = temp_rev[12..6] ;
                  temp_sh[12..77] = 0;
    WHEN 8 =>     temp_sh[7..0] = temp_rev[12..5] ;
                  temp_sh[12..8] = 0;
    WHEN 9 =>     temp_sh[8..0] = temp_rev[12..4] ;
                  temp_sh[12..9] = 0;
    WHEN 10 =>    temp_sh[12..10] = 0;
                  temp_sh[12..10] = 0;
    WHEN 11 =>    temp_sh[10..0] = temp_rev[12..2] ;
                  temp_sh[12..11] = 0;
    WHEN 12 =>    temp_sh[11..0] = temp_rev[12..1] ;
                  temp_sh[12] = GND;
END CASE;

IF  (NOT r3_q)  THEN                                    % R2 %
        cw[] = temp_sh[] OR maxrl_q[] ;
ELSIF  (NOT r3_long)  THEN      % R3 SHORT %
        cw[11..0] = temp_sh[12..1]  OR maxrl_q[11..0] ;
        cw[12]          = GND;
ELSE                                                    % R3 LONG %
        cw[12..1] = temp_sh[12..1]  OR
                        (maxrl_q[11..0] AND NOT mask_q[11..0]) ;
```

```
            cw[0] = temp_sh[0] ;
    END IF;
END;
```

kexpand.tdf

```
TITLE "decoder, k expand logic" ;

SUBDESIGN kexpand
(
        k_reg[3..0] ,
        r3_reg
                                : input;
        maxrl[12..0] ,
        mask[11..0] ,
        r3split[10..0]
                                : output;
)

BEGIN

        TABLE
            k_reg[],r3_reg      = > maxrl[],    mask[], r3split[] ;

            0,    0       = > 1,          0,    X;
            1,    0       = > 2,          1,    X;
            1,    1       = > 3,          1,    1;
            2,    0       = > 4,          3,    X;
            2,    1       = > 6,          3,    2;
            3,    0       = > 8,          7,    X;
            3,    1       = > 12,         7,    4;
            4,    0       = > 16,         15,   X;
            4,    1       = > 24,         15,   8;
            5,    0       = > 32,         31,   X;
            5,    1       = > 48,         31,   16;
            6,    0       = > 64,         63,   X;
            6,    1       = > 96,         63,   32;
            7,    0       = > 128,        127,  X;
            7,    1       = > 192,        127,  64;
            8,    0       = > 256,        255,  X;
            8,    1       = > 384,        255,  128;
            9,    0       = > 512,        511,  X;
            9,    1       = > 768,        511,  256;
            10,   0       = > 1024,       1023, X;
            10,   1       = > 1536,       1023, 512;
            11,   0       = > 2048,       2047, X;
```

```
        11,    1        = > 3072,          2047, 1024;
        12,    0        = > 4096,          4095, X;
    END TABLE;

END;
```

图 8 是重新排序单元的一个实施例的方框图。参考图 8，重新排序单元 606 包括一个运行计数重新排序单元 801 和一个比特组合单元 802。运行计数重新排序单元 801 把码字从编码器产生的运行终点移动到解码器需要的运行开始点，同时比特组合单元 802 把可变长度码字组合成为固定长度交织字并以解码器需要的适当顺序将其输出。

一个“窥探”解码器可用于任何解码器的重新排序，其中，解码器包括在编码器中并以真正解码器需要的编码字顺序提出请求。为了支持窥探解码器，对于每一个流必须独立进行运行计数重新排序。对于易于模拟的解码器，可能用多个以时间标记的序列或单一合成序列进行重新排序。在一个实施例中，用一个序列形式的数据结构来实现每个码字的重新排序，其过程与多个编码数据流的应用无关。下面描述如何进行重新排序。

在编码器中进行的第一个重新排序操作是重新排序每个运行计数，使它在运行开始时就被指定（因为解码器需要用于解码）。需要这种重新排序的原因是编码器直到运行终止才决定一个运行计数（和码字）是什么。因此，从编码数据产生的结果运行计数被重新排序，使解码器能够适当解码运行计数，使其返回到数据流。

返回参考图 8，本发明的重新排序单元 606 由运行计数重新排序单元 801 和比特装组合单元 802 组成。连接了运行计数重新排序单元 801，以接收多个输入，包括起始信号 711，终止信号 712，索引信号 713 和码字 714。将结合图 9 的运行计数重新排序单元详细叙述这些信号。响应输入运行计数重新排序单元 801 生成码字 803 和信号 804。信号 804 指示何时复位运行计数。码字 803 由比特组合单元 802 接收。响应码字 803，比特组合单元 802 发生交织字 805。

下面更详细叙述运行计数重新排序单元 801 和比特组合单元 802。

如上所述，解码器在需要由码字开始编码数据时接收码字。然而，编码器直到由码字编码数据终止时才知道此码字的标识符。

图 9 描述了运行计数重新排序单元 801 的一个实施例的方框图。所述的实施例包括四个交织流，其中每个交织字是 16 比特，且码字长度在 1 到 13 比特之间变化。在这种情况下，重新排序单元 300 可流水线连接以处理所有流。此外，使用一个把运行计数与概率等级相结合的编码器，使在任何时刻能被激活的运行计数的最大数小，在这个实施例中假设为 25。注意本发明不限于四个交织流，16 比特的交织字或 1 到 13 比特的码字长，它还可用于具有或多于或少于 16 比特的交织字和从 1 比特扩展到 13 比特以外的码字长度的或多或少的流。

参考图 9，连接了一个指针存储器 901，以接收索引输入 1713 并产生一个连接到复用器 (MUX) 90 2 的一个输入的地址输出。

MUX902 的其它两个输入被连接以接收一个来自前端计数器903的头指针的形式的地址和来自尾计数器 904 的一个尾指针形式的地址。MUX902 的输出是一个连接到和用于存取一个码字存储器 908 的地址。

也连接了索引输入 713 作为一个输入到 MUX905。MUX905 的另一个输入连接到码字输入 714。MUX905 的输出连接到有效检测模块 906 的一个输入和数据总线 907 上。数据总线 907 连接到码字存储器 908 和 MUX905 的一个输入。控制模块 909 的一个输出也连接到数据总线 907 上。起始输入 711 和终止输入 712 连接到控制组件 909 的独立输入。有效检测模块 906 的输出包括码字输出 803 和信号 804 (图 8)。运行计数重新排序单元 801 也包括控制器逻辑 (未表示, 以避免使目前发明含糊) 以协调运行计数重新排序单元 801 的各种元件的操作。

为了重申, 索引输入 713 识别一个运行。在一个实施例中, 索引指示 25 个概率等级之一。在这种情况下, 需要五个比特代表这个索引。注意如果用概率等级的多个单元, 则可能需要额外的比特指定具体的单元。在一个实施例中, 索引输入识别运行计数的概率等级。当运行终止出现时, 码字输入 714 是码字, 否则是“不关心”。起始输入 711 和终止输入 712 是控制信号指示运行是在开始, 在结束或两者都是。不运行包括一单一的未编码比特时, 运行同时开始和终止。

运行计数重新排序单元 801 响应其输入信号重新排序由比特发生器发生的运行计数。码字存储器 908 在重新排序过程中存贮码字。在一个实施例中, 码字存储器 908 比在某时进行的运行

计数数大，导致更好的压缩。如果码字存储器比某时进行的运行计数的数小，实际上这会限制进行中的运行计数数为存储器中能存的数。在提供良好压缩的系统中，经常发生这种情况，即在具有长运行计数的一个码字的数据累积过程中，很多具有短运行计数的码字将起动（和也可能终止）。这需要有一个大存储器，以避免长运行完成之前被迫停止。

指针存储器 901 存贮位于运行中间的概率等级的码字存储器的位置地址，并以随机存取方式寻址码字存储器 908。指针存储器 901 对于每个可能位于运行中间的概率等级均有一个存贮在码字存储器 908 中的地址的位置。一旦完成对于一个特定的概率等级运行，则对应那个概率等级的存贮在指针存储器 901 中的地址被用来存取码字存储器 908 且完成的码字被写进码字存储器 908 中的那个位置。直到那时，在码字存储器 908 中的那个位置一直包括一个无效输入。因此，指针存储器 901 存贮每个运行计数的无效码字位置。

前部计数器 903 和尾部计数器 904 也提供存取码字存储器 908 的地址。用前部计数器 903 和尾部计数器 904 允许以一个队列或环形缓冲器（如一个先入，先出 [FIFO] 存储器）寻址存储器 908。尾部指针 904 包括在码字存储器 908 中的下一个可用位置的地址，允许将一个码字插入码字存储器 908 中。前部计数器 903 在编码字存储器 908 中包括将被输出的下一个码字的地址。换句话说，前部计数器 903 包括要被从码字存储器 908 中删除的下一个码字的码字存储器地址。在指针存储器 901 中的每一个概率索引（如概率等级）的位置用于记忆当运行开始时尾部指针 904 的

位置，使得当运行终止时合适的码字可被放置在码字存储器 908 的那个位置上。

控制模块 909 发生一个有效信号作为存贮在码字存储器 908 中的数据的一部分，以指示一个输入是否存贮有效编码字数据。例如，如果有效比特位于一个逻辑 1，则码字存储器位置包括有效数据。然而，如果有效比特位于一个逻辑 0，则码字存储器位置包括无效数据。有效检测模块 907 决定每次从码字存储器 809 中读出一个码字时，一个存储器位置是否包括一个有效码字。在一个实施例中，有效检测模块 907 检测存储器位置是否有一个有效码字或一个特殊的无效码。

当启动一个新运行时，一个无效数据输入被加在码字存储器 908 上。无效数据输入作为在码字存储器 908 中的存贮数据流的空间占有者，使运行完时，运行的码字可被存贮在存储器的正确位置（以确保合适的排序以模拟解码器）。在一个实施例中，无效数据输入包括经 MUX905 的索引和一个来自控制模块 909 的无效指示（如一个无效比特）。无效输入被存贮的在码字存储器 908 中的地址由尾部指针 904 给出，随后存贮在指针存储器 901 中作为码字存储器 908 中运行计数位置的提示符。出现在码字存储器 908 中的头指针 903 和尾部指针 904 之间的数据提示作为完成的运行计数（如已重新排序的运行计数）。无效存储器位置的最大数是 0 到 I－1，其中 I 是运行计数数。当运行终止一个码字完成时，用存贮在指针存储器 901 中的地址把运行计数填入码字存储器 908 中。

当一个运行开始时，运行索引存贮在码字存储器 908 中，这

样，如果码字存储器 908 存满，而运行还未完成时，索引与信号 804 结合使用，以复位相关的运行计数器。除了在码字存储器 908 中存贮码字或索引外，一个这里称为“有效”比特的比特用来指示两类数据中哪类被存贮了。

如果不启始或终止一个运行，则运行计数重新排序单元空闲。如果起动一个运行而不终止一个运行且存储器满，则从码字存储器 908 输出一个码字。输出的码字是那个概率等级的在头指针 903 中包括的地址中存贮的码字。然后，如果启动一个运行而又不终止一个运行（无论存储器是否存满），则经 MUX905 将索引输入 713 写入由尾部指针 904 指定的码字存储器地址中。尾部指针 904 然后被写进指针存储器 901 的由索引输入 713（如在用于概率等级的指针存储器 901 的位置）上的数据指定的地址。写完尾部指针 904 后，尾部指针 904 递增。

如果终止一个运行而不启动一个运行，则存贮在相应索引（概率等级）的指针存储器 901 中的地址被读出和用作码字存储器中的位置以存贮编码字输入 714 上的完整的码字。

如果启动一个运行和终止一个运行（即，运行的启始和终止同时），且存储器存满，则从码字存储器 908 中输出一个码字。然后，如果启动一个运行且终止一个运行（无论存储器是否存满），则码字输入 714 被写进码字存储器 908 中由尾部指针 904 指定的地址。然后尾部指针 904 被递增以包括下一个可用位置（如增量值为 1）。

在本发明中，运行计数重新排序单元 801 可能在不同时间输出码字。在一个实施例中，编码字有效或无效时均可被输出。如果

一个存储器存满而运行尚未结束，码字无效时可被输出。为保持一最小速率(即，为进行速率控制)，无效码字也可能被输出。当所有数据都经过运行计数重新排序或当运行计数重新排序单元由于复位操作跳到码字存储器908的中间时，无效码字也可能被输出以注满码字存储器908。注意在这种情况下，解码器必须知道编码器正以这种方式操作。

如上所述，无论何时只要码字存储器908存满，则有一个码字被输出。一旦存储器存满，一旦向码字存储器908进行一个输入(即，开始一个新码字)，则进行一个从码字存储器908的输出。注意当一存储器存满时，输入的更新不引起从码字存储器908的输出。即，运行完成，随后把结果码字写入先前分配的存储器位置不造成存储器的满输出发生。类似地，当一个运行终止，且在指针存储器901中的相应地址与在前部计数器903中地址相同时，码字可被立即输出且前部计数器903增加而不存取码字存储器908。在一个实施例中，当尾部指针递增后，尾部指针904等于头指针903时，会出现存储器满的情况。因此，一旦尾部指针904被增加，运行计数重新排序单元801中的控制器逻辑比较尾部指针904与头指针903，且如果相同，控制器逻辑确定码字存储器908已满和应输出一个码字。在另一个实施例中，编码字可在存储器存满之前被输出。例如，如果由前部寻址的队列部分包括有效码字，它可被输出。这需要反复检查队列的开端以确定码字的状态。注意在文件编码终止时，码字存储器908腾空。

应用运行计数重新排序单元801，一个码字在第一次从码字存储器908的由头指针903规定的地址读取一个值（如数据）时

被输出。码字输出用控制器逻辑控制和协调。有效检测模块 906 进行一测试以确定此值是否为一码字。换句话说，有效检测模块 906 确定此码字是否有效。在一个实施例中，有效检测模块 906 通过检查和每个输入一起存贮的有效比特，决定任何输入的有效性。如果此值为一码字 (即，码字有效)，则此值作为码字输出。另一方面，如果此值不是码字 (即，码字无效)，则只要具有至少和当前运行计数一样长的 MPS 运行的任何码字都能被输出。“0”码字是正确代表目前运行的码字，可被输出。输出进行后，头指针 903 被增加到在码字存储器 908 中的下一位置。或者用具有最短允许运行长度的“1N”使解码器只检查发送一个 LPS 之前是否有一个编码字被迫出。

在一个实施例中，运行计数重新排序单元 801 用两个时钟周期时间操作。在第一个时钟周期中，输入被接收到运行计数重新排序单元 801 中。在第二个时钟周期中，出现一个从码字存储器 908 输出。

一旦头指针 903 寻址一个有效码字，编码字就能被输出时，在某些应用中可能希望当缓冲器满时，只输出一个码字。这导致系统按码字数有一个固定延时，而不是一可变延时。如果存储器 908 在运行启动和被输入与被输出的时间之间能留存一些预定数目的码字，则延时为那个编码字数，因为直到存满才进行输出。因此，有一个码字的恒定延时。注意在其它方法中，重新排序延时仍为变量，例如，编码的或原始数据的量。通过允许存储器 908 在产生输出之前存满，输出端每周期输出一个码字。

注意如果一个码字存储器位置被称为无效的，未用的比特可

能被用来存贮关于为谁进行运行计数的标识符（即，必须存贮必须填在这个位置的上下文接受器或概率等级）。这个信息对于处理存储器存满的情况有用。尤其此信息可用于指示比特发生器，对此特定运行长度的一个码字还未完成且它必须现在完成。在这种情况下，已决定了输出一个可能由于存储器存满发生的无效码字。因此，当系统复位运行计数器时，此信息指示按比特发生器和运行计数，系统将何时重新开始。

对于索引输入，由于流水线原因当概率等级的单元被使用时，索引可包括一个单元识别符。就是说，对于一个特定的概率等级可能有多个运行计数。例如，两个运行计数可能用于百分之八十码，先用一个再用另一个。

由于码字长度为可变的，必须以允许确定其长度的方式，在码字存储器 908 中存贮它们。可以清楚地存贮其长度的同时，不会使存储器用途降至最小。对于 R－码，在存储器中存贮一个零值可指明一个比特“0”码字和“1N”码字可被存贮以使能用一优先编码器，从第一个“1”比特确定长度。

如果码字存储器 908 是多端口的（如双端口）此设计可用流水线进行每一时钟周期处理一个码字。因为从多端口可存取码字存储器 908 中的任何位置，能写入码字存储器 908 的一个位置（就像当一无效或码字被存贮时一样），而另一部分可被读取（像一个码字被输出时一样）。注意在这种情况下，可能必须修改复用器以支持多个数据和地址总线。

一旦由于码字存储器满，编码器输出一个“0”码字且复位运行计数器时，解码器必须做同样的工作。这需要解码器模拟编码

器的码字存储器队列。下面将讨论如何完成这一任务。

注意为在CMOS应用中节省电能，当对于无效运行输出“0”码字时，对于“1N”码字，计数器应设为不能用。这是因为被解码的“1N”码字是有效的，而只有“0”码字是无效的。

图10是根据上下文(与概率等级相反)重新排序接收数据的一个运行计数重新排序单元的另一个实施例的方框图。运行计数重新排序单元1000用R－码进行重新排序。参考图10，重新排序单元1000包括一个指针存储器1001，一个前部计数器1002，一个尾部计数器1003，一个数据复用器1004 (MUX)，一个地址MUX1005，一个计算长度块1006，一个有效检测块1007和一个码字存储器1008。码字存储器1008在重新排序过程中存贮码字。指针存储器1001存贮位于运行中间的上下文接受器的码字存储器位置。前部计数器1002和尾部计数器1003允许除了由指针存储器1001以随机存取方式存取以外，允许码字存储器1008作为一个队列或环形缓冲器而被存取。对于R－码，在存储器中存贮一个零值可表示一个比特“0”码字且“1N”码字能被存贮，这样能使优先编码器从第一个“1”比特确定长度。计算长度模块1006操作如同一个优先编码器。(如果应用其它可变长度码，增加一个“1”比特标志码字的起始，比增加log2比特以弄清存贮长度使存储器更有效)。运行计数重新排序单元1000也包括后级控制器逻辑以协调和控制元件1001－1008的操作。

运行计数重新排序单元1000的操作十分类似根据概率评估的运行计数重新排序单元。如果启动一个新运行，则一个包括上下文接受器的无效输入被写进编码器存储器1008中由尾部指针

1003 指示的地址。尾部指针 1003 地址则被存贮在指针存储器 1001 中的当前运行计数的上下文接受器的地址内。然后，尾指针被增加。当一个运行完成时，则对相应于运行计数的指针存储器 1001 中的指针，被从指针存储器 1001 读出，且码字被并行写入码字存储器 1008 中由那个指针指定的位置。如果既不启动也不终止一个运行，并且在码字存储器 1008 中当头指针 1002 的地址指定的位置不含无效数据，则由前部寻址的码字被读出和输出。然后头指针 1002 递增。对于运行同时开始和终止的情况，码字被写进码字存储器 1008 的由尾部指针 1003 所指定的地址，而尾部指针 1003 被递增。

类似地，当一个运行终止，且在指针存储器 1001 中的相应地址与在前部计数器 1002 中的地址相同，则码字可被立即输出，且前部计数器 1002 中的值被增加而不存取码字存储器 1008。

对于运行计数“由上下文”系统，每个上下文需要一个在指针存储器 1001 中的存储器位置。因此 BG 和 PEM 状态存储器的宽度可被扩展以实现这个存储器。指针存储器 1001 的宽度等于一个码字存储器地址所需要的大小。

在具体应用中，码字存储器 1008 中的位置数可由设计者选择。此存储器的有限大小降低了压缩效率，因此有一个造价/压缩折中。码字存储器的宽度等于最大码字的长度与一个有效/无效指示的比特之和。

下表 13 所示的是应用 R2 (2) 码的一个实施例，它将用于说明重新排序。表 14 表示要被重新排序的数据 (0 = MPS, 较高概率符号; 1 = LPS, 低概率符号)，被上下文加标记。只有两个上下文。

未解码比特数指示在未编码比特时钟周期中的时间。指明了运行的起始和终止，运行终止时显示出码字。

表 13 - R2(2) 码

原	码字
0000	0
0001	100
001	110
01	101
1	111

表 14 - 将被编码的数据例子

未解码比特号	数据	上下文	运行开始/结束	码字
1	0	0	S	
2	0	1	S	
3	0	0		
4	1	1	E	101
5	0	0		
6	0	1	S	
7	0	0	E	0
8	1	1	E	101
9	0	0	S	
10	0	1	S	
11	0	0		
12	0	1		
13	0	0		
14	0	1		
15	1	0	E	100
16	0	1	E	0

表 15- 重新排序操作的例子

未编码比特号	输入	指针		指针存储器		码字存储器				输出
		头	尾	0	1	0	1	2	3	
1	开始0	0	1	0	x	无效	x	x	x	
2	开始1	0	2	0	1	无效	无效	x	x	
3	(重新排序单元空闲)									
4	结束1.101	0	2	0	x	无效	101	x	x	
5	(重新排序单元空闲)									
6	开始1	0	3	0	2	无效	101	无效	x	
7	结束0.0	0	3	x	2	0	101	无效	x	
		1	3	x	2	x	101	无效	x	0
		2	3	x	2	x	x	无效	x	101
8	结束1.101	2	3	x	x	x	x	101	x	
		3	3	x	x	x	x	x	x	101
9	开始0	3	0	3	x	x	x	x	无效	
10	开始1	3	1	3	0	无效	x	x	无效	
11	(重新排序单元空闲)									
12										
13										
14										
15	结束0.100	3	1	x	0	无效	x	x	100	
		0	1	x	0	无效	x	x	x	100
16	结束1.0	0	1	x	x	0	x	x	x	
		1	1	x	x	x	x	x	x	0

表 15 表示对实例数据的重新排序操作。采用一个具有四个位置 0－3 的码字存储器，其足够大以致在此例中不会溢出。每行表示或为某个上下文的运行的起始，或终止或为一个码字输出的一个操作之后，系统的状态。“X”用于指出“不关心”存储器位置。对于某些未编码比特，运行既未开始也未结束，因此运行计数重新排序单元空闲。对于终止运行的编码比特，可输出一个或多个码字，可能引起系统状态的几个变化。

参考表 15，前部和尾部指针初始化为零，表示在码字存储器中不含任何东西（如队列）。所示的指示器存储器有两个存贮位置，每个对应一个上下文。每个位置在比特号 1 之前有“不关心”值。所示的码字存储器具有四个码字的深度，所有都被初始作为“不关心”值。

响应比特号 1 的接收数据，头指针一直指向码字存储器位置。由于解码器期望数据，下一个可用码字存储器位置 0 被分配给一个码字，且有一个无效值被写进存储器位置 0。因为上下文是 0，分配给码字的码字存储器位置地址存贮在用于零上下文（指针存储器位置 0）的指针存储器位置上。因此，一个“0”存贮在指针存储器位置 0。尾部指针增加到下一个码字存储器位置 1。

响应对应于比特号 2 的数据，前部计数器一直指向第一个存储器位置（由于还没有引起它递增的输出）。由于数据对应于第二个上下文，上下文 1，下一个码字存储器位置被分配给码字，作为码字存储器位置 1，如尾部指示器所指示，并且一个无效值被写入此位置。地址，码字位置 1 被写入对应上下文 1 的指针存储器位

置。就是说，第二个码字存储器位置的地址被写入指针存储器位置 1。然后尾部指针递增。

响应对应比特号 3 的数据，重新排序单元空闲，因为运行既未开动又未终止。

响应对应比特号 4 的数据，上下文 1 的运行终止被指明。因此，码字“101”被写入分配给由用于上下文 1 的指针存储器位置指出的上下文 1 (编码字存储器位置 1) 的编码字存储器位置。头部和尾部指针保持相同，且在用于上下文 1 的指针存储器位置中的值不再被用，因此是“不关心”。

响应对应比特号 5 的数据，重新排序单元空闲，因为运行既未开始又未终止。

响应对应比特号 6 的数据，进行如上所述的对于比特 2 的同样类型操作。

响应对应比特号 7 的数据，为上下文 0 的码字运行终止。在这种情况下，码字“0”被写入由用于上下文 0 (指针存储器位置 0) 的指针存储器位置所指示的编码字存储器位置 (码字存储器位置 0)。然后指针存储器位置上的值不再被使用，所以是“不关心”。由头指针指定的码字存储器位置也包括有效数据。因此，输出有效数据而且头指针递增。增加头指针使它指向另一个包括一有效码字的码字存储器位置。因此，这个编码字输出且头指针又被递增。注意在这个例子中，码字能输出时则输出，与码字存储器完全存满时输出相反。

按上述方式，对未编码比特的处理继续进行。注意码字存储器位置不专门用于特定上下文，这样来自任何上下文的码字在整

个数据文件编码过程中均可被存贮在一特定的码字存储器位置上。

图 4 说明比特组合，表示了在比特被组合前后由重新排序单元处理的数据。重新参考图 4，它表示了从号 1 到号 16 的十六个变长码字以指示由解码器使用的顺序。每个编码字被分配到三个编码流之一中。每个编码流的数据被分割成称为交织字的固定长度字。(注意一个单一变长编码字可被分割成两个交织字)。在这个例子中，交织字在一个单一交织流中排序，使得在一特定交织字中的第一个变长编码字（或部分编码字）的顺序决定交织字的顺序。也可能进行其它类型排序标准。交织多个编码流的优点是可用一个单一编码数据信道传输数据，且对于每个流的变长移位可并行或流水线进行。

本发明的比特组合单元 802 从运行计数重新排序单元 801 接收变长码字并将其组合为交织字。比特组合单元 802 由处理变长码字的逻辑和以正确顺序输出固定长度交织字的合并队列型重新排序单元组成。在一个实施例中，从运行计数重新排序单元接收码字的速度为每时钟周期一个码字。图 11 表示比特组合单元 802 的一个实施例的方框图。在随后的实施例中，应用四个交织流，每个交织字是 16 比特，码字长度变化范围为 1 到 13 比特。在一个实施例中，一个单一比特组合单元流水线处理所有流。如果本发明的比特组合单元 802 的双端口存储器（或寄存器文件），它可在每时钟周期输出一个交织字。这可能比需要的保留其它编码器要快。

重新参考图 11，比特组合单元 802 包括组合逻辑 1101，流计

数器 1102, 存储器 1103, 尾部指针 1104 和一个前部计数器 1105。组合单元 1101 被连接以接收码字并连接到流计数器 1102 上。流计数器 1102 又连接到存储器 1103 上。尾部指针 1104 和前部计数器 1105 也连接到存储器 1103。

流计数器 1102 跟踪当前输入编码字与之关联的交织流的轨迹。在一个实施例中，流计数器 1102 反复计数从 0 到 N－1 的流数，这里 N 为流号。一旦流计数器 1102 达到 N－1，它再开始从 0 计数。在一个实施例中，流计数器 1102 是一个两比特计数器，从 0 到 3 计数 (对于四个交织流)。在一个实施例中，流计数器 1102 被初始化为 0(如通过整体复位)。

组合逻辑 1101 合并当前与以前的输入码字以形成交织码字。每个码字的长度均不同。因此，组合逻辑 1101 把这些变长码字打包成固定长度字。由组合逻辑 1101 产生的交织码字被顺序地输出到存储器 1103 并存贮在存储器 1103 中，直到适当时机将其输出。在一个实施例中，存储器 1103 是一个静态随机存取存储器 (SRAM) 或一个具有 64 个 16－比特的寄存器文件。

交织字存贮在存储器 1103 中。在本发明中，存储器 1103 的大小大得足以处理二个情况。一个是正常操作情况，其中一个交织流有最小长度码字而另一个交织流有最大长度码字。这第一种情况需要 3×13＝39 个存储器位置。另一个情况是初始化情况，其中也是一个流有最小长度或短码字，而其它流有最大长度或长码字。对于第二种情况，2×3×13＝78 个存储器位置足够用，PEM 的操作允许一个 56 的紧密范围。

与流计数器 1102 的尾部指针 1104 合作，存储器 1103 进行

重新排序。流计数器 1102 指示一个由存储器 1103 接收的当前码字流。每个交织流与至少一个尾部指针相关。尾部指针 1104 和头指针 1105 进行码字的重新排序。每个流有两个尾指针的原因是当在交织字 N－1 中的数据包括下一个码字的开端时，解码器需要交织字 N。一个尾指针确定在存储器 1103 中存贮来自一给定交织流的下一个交织字的位置。另一个尾指针确定在存储器中存贮下一个以后的交织字的位置。这允许当解码器的交织字 N－1 的请求时间已知时，规定交织字 N 的位置。在一个实施例中，这些指针是 8 个 6－比特寄存器(每流两个尾指针)。

在一个实施例中，在编码开始时，尾指针 1104 被设置成使第一组八个交织字（每流两个）被以来自每流的序列一方式存贮在存储器 1103 中。初始化后，每当组合逻辑 1101 为一特定码流开始一个新交织字时，“下一个”尾指针则被设为“下一个之后”尾指针的值，且码流的“下一个之后”尾指针设定为下一个可用存储器位置。因此，每流有两个尾指针。在另一个实施例中，每流只用一个尾指针，它指示下一个交织字要存贮在存储器 1203 的何处。

前部计数器 1105 用于确定下一个交织字的存储器位置，以从比特组合单元 802 中输出。在所述的实施例中，前部计数器 1105 由一个 6 比特计数器组成，它被增加以每次输出一个完整的交织字。

存储器 1103 除用于重新排序外，也用作编码器和信道之间的 FIFO 缓冲器。理想情况是这个存储器比重新排序需要的大，使当信道不能跟上编码器时，一个 FIFO－几乎存满信号，能被用于阻止此编码器。一个每周期一比特的编码器不能每周期发生一个

交织字。当一个编码器与一个信道匹配良好时，此信道每周期不接受一个交织字，而某个 FIFO 缓冲器是必需的。例如，一个每 32 个时钟周期可接受一个 16 比特交织字的信道，当压缩比为 2：1 或更大些时，对于 2:1 有效带宽扩展将为一个良好的匹配设计。

图 12 为组合逻辑方框图。参考图 12，组合逻辑 1101 由一个长度单元 1201，一组累加器 1202，一个移位器 1203，一个 MUX1204，一组寄存器 1205 和一个或门逻辑 1206 组成。连接长度单元 1201 以接收码字和连接到累加器 1202。累加器及其码字被连接到移位器 1203 上。移位器 1203 连接到 MUX1204 和或门逻辑 1206。MUX1204 也连接到寄存器 1205 和或门逻辑 1206 的一个输出。寄存器也连接到或门逻辑 1206。

在一个实施例中，码字在一个具有未用比特置零的 13 比特总线上输入。这些置零未用比特临近“1N”码字中的“1”，因此一个在长度单元 1201 中的优选编码器可被用于确定“1N”码字的长度和为“0”码字产生一个长度。

累加器 1202 由多个累加器组成，每个用于一个交织流。每个交织流的累加器保存一个已经在当前交织字中的比特流记录。在一个实施例中，每个累加器由一个 4 比特加法器（具有执行功能）和一个用于每个流的 4 比特寄存器组成。在一个实施例中，加法器的输出为累加器的输出。在另一个实施例中，寄存器的输出是累加器的输出。利用从长度单元 1201 接收的码字长度，累加器确定欲被移位以把当前的编码字串联进包括那个流的当前交织字的比特数。

根据累加器的当前值，移位器 1203 排列当前码字以使其适

当地跟随那个交织字中的任何前面的码字。因此，在编码器中的数据被移位成解码器顺序。移位器的输出为 28 比特，它处理一个 13 比特码字必须加在当前交织字中的 15 比特后的情况，使来自当前码字中的比特结束在正在输出的 28 比特的较高的 12 比特。注意移位器 1203 没有反馈操作，因此可流水线操作。在一个实施例中，移位器 1203 由一个柱式移位器组成。

移位器 1205 存贮当前交织字中的比特。在一个实施例中，用于每个交织流的一个 16－比特移位器保存在当前交织字中的先前的比特。

开始，一个流的码字由移位器 1203 接收，同时长度单元 1201 为对应此流的累加器指示编码字的长度。累加器有一个通过整体复位而设定的零值。由于累加器值为零，此码字不被移位但用 OR 逻辑 1206 和相应于此流的寄存器内容进行 OR 运算。然而在某些实施例中，IN 码字必须被移位以适当地定位，即使在一个交织字的开端也如此。这个寄存器已经初始化为零，因此，或运算的结果是把此码字放到成逻辑 1206 输出的最右比特位置上，且通过 MUX1204 反馈到寄存器以存贮，直到从流中出来下一个码字。因此，初始的移位器 1203 以全通操作。注意在第一个码字中的比特数现在被存贮在累加器中。一接收到那个流的下一个码字，累加器中的值就被送到移位器 1203，且此码字被向左移位几个比特以与此交织字中任何以前的输入比特结合。零放置在被移位的字中的其它比特位置。来自相应流的寄存器的比特用或逻辑 1206 与来自移位器 1203 的比特结合。如果累加器不产生一个进位指示(如信号)，则需要更多的比特来完成当前的交织字且或操作的数

据结果通过 MUX1204 返回到寄存器保存。在一个实施例中，MUX1204 由一个 2:1 复用器组成。当累加器产生一个进位时，来自或逻辑 1206 的经或运算的 16 比数据是一个完整的交织字并被输出。MUX1204 使寄存器被第一批 16 比特以后的任何额外比特(如从寄存器 1203 输出 28 比特的前 12 比特)装载及剩余部分用零填充。

对 MUX1204 和交织字输出的控制包括来自累加器的进位信号。在一个实施例中，复用器 1204 用十六个 2: 1 复用器组成，其中 4 个具有一个永远为零的输入。

本发明提供多个进行数据重新排序的选项。例如，在一个具有多个码流的系统中，码流必须重新排序成如图 4 所示的交织字。本发明提供多种完成重新排序成交织字的方法。

把数据重新排序成交织字的一个方法是用一个如图 25 所示的窥探解码器。参考图 25，连接多个运行计数重新排序单元 2501_{A-N}以接收与码字流相伴的码字信息。每个发生一个码字输出和一个长度输出。一个单独的比特组合逻辑 (1101) 单元，如比特组合单元 2502_{A-N}，连接以接收来自运行计数重新排序单元 2501_{A-N}之一的码字和长度。比特组合装置 2502_{a-n}输出交织字，其既与 MUX2503 连接又和窥探解码器 2504 连接。解码器 2504 提供一个由 MUX2503 接收的选择控制信号并向 MUX2503 指示哪个交织字要输出到码流中。

在图 8 中，每个编码数据流有一个运行计数重新排序单元 801，包括运行计数重新排序单元 801。每个比特组合单元把各种长度码字组合成固定长度交织字，每字或 8，16 或 32 比特。每个

比特组合单元如上所述包括寄存器和移位电路。解码器 2504 包括一个能从所有比特组合单元（或在图 25 所示的单独总线或经一公共总线）存取交织字的全面可操作的解码器（包括 BG, PEM 和 CM）组成。一旦解码器 2504 从比特组合单元之一选择一个交织字，则此字被发送进码流。由于接收端的解码器要求数据的顺序与相同的窥探解码器的一样，交织字以适当的顺序发送。

在一个半双工系统中，一个具有窥探解码器的编码器可能是有吸引力的，因为窥探解码器也可用作一般解码器。窥探解码器方案的一个优点是它对于任何具有决定性的解码器的可应用性。下面讨论的替代方案不依靠一个窥探解码器，而只用简单类型的解码器以降低硬件成本。对于在同一时钟周期中解码多个码字的解码器，不可能设计一个比特解码器自身硬件还少的解码器，需要使用窥探解码器。下面将叙述对于每周期最多只解码一个码字的解码器来说，确实存在简单些的模型。

为每周期最多解码一个码字的连续解码器系统的数据重新排序的另一个技术是基于这样的事实，即拟造解码器对编码数据的要求所需的唯一信息是知道码字的顺序（考虑所有的码字而不是单独考虑每个编码数据流的码字）如果每个码字都标有它进入运行计数重新排序单元的时间标记，哪个具用最早时间标记的被组合成交织字的比特即为要被输出的下一个交织码字。

图 26 的方框图表示一个示范性的编码器重新排序单元。参考图 26，此编码系统与图 25 所示的相同，除了时间标记信息也是由每个运行计数重新排序单元 2501_{A-N} 接收的。这个时间标记信息也进一步送到比特组合单元 2502_{A-N}。比特组合单元 2502_{A-N} 为

MUX2503 提供交织字和为逻辑 2601 提供与之相关的时间标记。逻辑 2601 为 MUX2503 提供一个控制信号以选择要被输出到码流中的交织字。

在这个实施例中，窥探解码器由一个简单的比较所替代，它决定比特组合单元 2502_{a-n} 中的哪一个具有最早时间标记的码字(或码字部分)。这样一个系统对于 MUX2503 似乎是具有时间标记的多个队列。逻辑 2601 只需简单地在各种队列之间选择。每个运行计数重新排序单元 2503_{A-n} 的逻辑只稍加变化 (从运行计数重新排序单元 801)以写出运行开始时的时间标记。每个运行计数重新排序单元 2501_{A-n} 均配备用来在码字存储器中存贮时间标记。用足够比特存贮时间标记以计数在编码数据流中的每个码字是足够的。但在某些实施例中，可能用较少的比特。

下面简述具有时间标记的多个队列所用的步骤。对此专业技术人员来说此叙述非常清楚。这些是编码器的操作。对于运行由同一码字启动和终止的情况未做简化。可检查每个编码符号的操作(虽然实际上不需要全做检查)假设交织字的长度是 32 比特。

```
如果(无用于上下文的当前码字){
    将时间放入队列(用于决定下一队列)
    将上下文指针放入队列
    将无效数据放入队列
    指示上下文到队列输入
队尾增加端
}
如果(已有一个码字和 MPS){
```

```
        增加上下文运行计数
    }
    如果(MAX RUN 或 LPS){
        在队列中放置正确数据
        (不需要上下文指针)
        零指针和运行计数在上下文存储器中
        在上下文存储器中更新概率评估
    }
    如果(有效数据位于下一队列前端){
        在输出位置 32 比特数据
        清除队列输入
        增加序列前端
    }
    当(任何队列几乎存满){
        寻找必须在输出放置数据的下一队列)
        当(小于 32 比特有效数据){
            用上下文指针寻找上下文
            零指针和运行计数在上下文存储器中
            放置 MAXRUN 码字于队列数据中
        }
    }
```

解码器操作类似，虽然码字不需要保存在队列中。仍然需要在队列中存储码字的时间标记。

上面讨论的时间标记的功能被用于存贮码字的顺序信息。表

达同样概念的等同方式是通过用一个用于所有码字的单一队列，即一个合并序列。在一合并队列系统中，如图 27 所示，一个单一运行计数重新排序单元 2701 被用于所有交织流。运行计数重新排序单元 2701 为比特组合单元 2502_{A-N} 发生码字，长度和流输出，为 MUX2503 发生输出交织字和为逻辑 2702 发生位置信息，它指示 MUX2503 输出交织字作为码流的一部分。

对于任意流，运行计数重新排序存储器为每个码字存贮一个交织流 ID。每个交织流有其自己的头指针。当一个比特组合单元需要更多数据时，相应的头指针被用来获取所需要的码字以形成一个新交织字。这可能包括查看很多码字存储器位置以决定哪个是适当流的一部分。另一方面，这可能包括在码字存储器中查的额外的区域来实现一个链表。

在本发明中交织流的另一个方法应用一具有固定流排列的合并队列。这个方法在合并队列情况下应用一个尾指针，因此不需要时间标记。象以前的情况一样应用多个头指针，因此在从一特定流中输出数据时无额外消耗。为完成这个过程，到交织流的码字的排列按下列规则进行，对于 N 流：码字 M 被分配给流 M 模 (mod) N。注意，按照这个方法，交织流可具有来自任何上下文接受器或概率等级的码字。如果流数是 2 的乘方，通过删除一些更重要比特可计算 M 模 N。例如，假设码字重新排序存储器用 12 比特寻址且应用四个交织流。尾指针是 12 比特长，两个最不显著比特为下一个码字识别码流。有四个具有 10 比特的前部指示器，每个被模糊地分配给两个最不显著比特的四种可能结合方式之一。尾部和前部指示器均象一般二进制计数器一样被递增。

在解码器中，移位器具有寄存器存贮交织字。此移位器为比特发生器提供适当排列的编码数据。当比特发生器使用某些编码数据时，它通知移位器。移位器从下一个交织流适时提供定位的数据。如果编码数据流的数量为 N，移位器用 N－1 个时钟周期移出用过的数据且或许在特定的交织流将被再次使用前请求另一个交织的编码字。

本发明包括一个支持具有有限重新排序存储器实时编码器的解码器。在一个实施例中，解码器也通过为每个概率等级而非每个上下文接收器维持一个运行计数而包括降低要求的存储器和复杂性。

图 14A 说明了本发明的解码器硬件系统的一个实施例的方框图。参考图 14A，解码器系统 1400 包括先进/先出 (FIFO) 结构 1401，解码器 1402，存储器 1403，和上下文模型 1404。解码器 602 包括多个解码器。编码的数据 1401 被连接以被 FIFO 结构 1401 接收。FIFO 结构 1401 被连接为解码器 1402 提供编码的数据。解码器 1402 被连接到存储器 1403 和上下文模型 1404。上下文模型 1404 也被连接到存储器 1403。上下文模型 604 的一个输出包括解码的数据 1411。

在系统 1400 中，输入到 FIFO 结构 1401 的编码数据 1410 被重新排序和交织。FIFO 结构 1401 包含正常次序的数据。流被移交到解码器 1402。解码器 1402 要求来自这些流的数据按串行和确定性次序。虽然解码器 1402 要求的编码数据的次序是非平凡的，它不是随机的。通过在编码器而非解码器按此次序排序码字，编码的数据可以被交织到一单一流中。在另一实施例中，编码的

数据 1410 可能包括非交织数据的一单一流，其中对每个上下文接收器，上下文类或概率等级的数据被附在数据流上。在这种情况中，FIFO 1410 被一存储区域 1410 替换，以在向解码器 1402 转发数据前接收所有的编码数据，所以数据可能被恰当地分段。

在 FIFO1410 接收编码的数据时，上下文模型 1404 确定当前上下文接收器。在一实施例中，上下文模型 1404 基于上一个像素和/或比特确定当前上下文接收器。虽然没有给出，线路缓存可能包括上下文模型 1404 中。线路缓存提供必要的数据，或模板，上下文模型 1404 通过它们确定当前上下文接收器。例如，上下文根据当前像素附近的像素值，线路缓存可能被用于存储用于提供特定上下文附近的那些像素的像素值。

为了响应上下文接收器，解码器系统 1400 为当前上下文接收器从存储器 1403 中取出解码器状态。在一个实施例中，解码器状态包括概率估计模块 (PEM) 状态和比特发生器状态。PEM 状态确定哪个码用于解码新的码字。比特发生器状态维持当前运行中的一个比特记录。为了响应由上下文模型 1404 提供的一个地址，状态被从存储器 1403 提供给解码器 1402。地址访问存贮着涉及上下文接收器信息的存储器 1403 中的一位置。

一旦当前上下文接收器的解码器状态已经从存储器 1403 中被取出，系统 1400 确定下一个非压缩比特并处理解码器状态。然后解码器 1402 解码新的码字，如果需要，和/或更新运行计数。PEM 状态被更新，如果需要，同样更新比特发生状态。然后解码器1402 将新的编码器状态写入存储器 1403。

图 14B 解释了本发明的一解码器的一个实施例。参考图14B，

解码器包括移位逻辑 1431，比特发生器逻辑 1432，“新 K”逻辑 1433, PEM 更新逻辑 1434, 新码字逻辑 1435, PEM 高速缓存器状态逻辑 1436，编码到掩码逻辑 143 7，编码到 MaxPL，掩码，和 R3Split 扩展逻辑 1438, 解码逻辑 1439, 复用器 1440, 和运行计数更新逻辑 1441。移位逻辑 1431 被连接以接收编码的数据输入 1443, 及状态输入 1442(来自存储器)。移位逻辑 1431 的输出也被连接，作为一个到比特发生逻辑 1432，“新 k”发生逻辑 1433 和 PEM 更新逻辑 1434 的输入。比特发生逻辑 1432 亦被连接以接收状态输入 1442 和生成到上下文模型的解码数据输出。新 k 逻辑 1433 生成一个连接到编码到掩码逻辑 1437 一输入的输出。PEM 更新逻辑 1434 也连接至状态输入 1442 并生成状态输出 (到存储器)。状态输入 1442 亦连接至新码字逻辑 1435 和 PEM 编码状态逻辑 1436 的输入。PEM 编码状态逻辑 1436 的输出被连接，用于被扩展逻辑 1438 接收。扩展逻辑 1438 的输出被连接到解码逻辑 1438 和运行计数更新逻辑 1441。另一个到解码逻辑的输入被连接至编码到掩码 1437 的输出。解码逻辑 1439 的输出被连接至 MUX 1440 的一输入。MUX 1440 的另一输入被连接至状态输入1442。MUX 1440 的选择输入被连接至新编码字逻辑 1435 的输出。MUX 1440 的输出和扩展逻辑 1438 被连接至具有掩码编码逻辑 1437 输出的运行计数更新逻辑 1441 的两个输入。运行计数更新逻辑 1441 的输出被包括在到存储器的状态输出中。

移位逻辑 1431 从编码数据流中移入数据。基于编码的数据输入和状态输入，比特发生逻辑 1432 生成解码的数据到上下文模型。新 k 逻辑 1433 也使用移进的数据和状态输入，来生成一新

的 k 值。在一个实施例中，新 k 逻辑 1433 使用 PEM 状态和编码数据的第一比特来生成新的 k 值。基于新的 k 值，编码到掩码逻辑 1437 为下一个编码字生成一个 RLZ 掩码。下一个码字的 RLZ 掩码被送至解码逻辑 709 和运行计数更新逻辑 1441。

PEM 更新逻辑 1434 更新 PEM 状态。在一个实施例中，PEM 状态使用当前状态被更新。被更新的状态被送至存储器。新的编码字逻辑 1435 确定是否需要一新的码字。PEM 状态到编码逻辑 1436 使用状态输入 1442 确定解码的码。码字被输入到扩展逻辑 1438 以生成最大运行长度，当前掩码和一 R3 分瓣值。解码逻辑 1439 解码码字以产生一运行计数输出。MUX 1440 选择来自解码逻辑 1439 的输出或到运行计数更新逻辑 1441 的状态输入 1442。运行计数更新逻辑 1441 更新运行计数。

包括解码器 1430 的本发明的解码系统 1400，运行于一流水线方式。在一实施例中，本发明的解码系统 600 均以流水线方式确定上下文接收器，估计概率，解码码字，和从运行计数生成比特。解码系统流水线结构的一个实施例在图 15A 中描绘。参考图 15A，本发明的流水线解码过程的一个实施例展示为六个阶段，编号为 1－6。

在第一阶段中，当前上下文接收器被确定 (1501)。在第二阶段中，在上下文接收器已被确定后，一个存储器读出发生，上下文接收器的当前解码器状态被从存储器中取出。如上所述，解码器状态包括 PEM 状态和比特发生器状态。

在当前发明流水线解码过程的第三阶段，一个解压缩比特被生成 (1503)。这允许一个比特对上下文模型可用。在第三阶段期

间还发生两个其它操作。PEM 状态被转换成一个编码类型(1504)并在第三阶段决定是否一个新的码字必须被解码(1505)。

在第四阶段期间，解码系统处理一个编码字和/或更新运行计数 (1506)。在处理一码字并更新运行计数中会调用几个子操作。例如，一个编码字被解码以确定下一个运行计数或运行计数对当前码字 (1506) 更新。如果在解码新的码字时需要，更多的编码数据被从输入 FIFO 中取出。发生在第四阶段的另一个子操作是 PEM 状态 (1507) 的更新。最后，在解码流水线的第四阶段，如果当前码字的运行计数是零 (1508)，新的 PEM 状态被用于确定对于下一个码运行长度零码字(以后描述)是什么。

在本发明解码流水线第五阶段期间，具有一个被更新 PEM 状态的解码器状态被写入存储器 (1509)，并对下一个码字 (1510) 开始移位。在第六阶段中，到下一个码字的移位被完成(1510)。

本发明的流水线解码实际开始时要决定是否开始解码过程。该决定基于是否有充足的数据提供给本发明的解码器。假如没有充足的来自 FIFO 的数据，解码系统被阻塞。在另一种情况下，解码系统在向一不能接收来自解码器生成的所有数据输出的外围设备输出解码的数据时可能被阻塞。例如，当解码器正向--视像显示接口和相关的视频电路提供输出时，视频也许太慢，因此解码器需要被阻塞以允许一视像被捕获。

一旦决定开始解码过程，当前上下文接收器将被上下文模型确定。在本发明中，当前上下文接收器通过检查上一个数据被确定。这样的上一个数据可能被存储在线路缓存器中，并可能包括来自当前线路和/或上一条线路的数据。例如，在一个上下文模板

中，对一给定比特。可能使用反映上一个数据的模板设计来自线路缓存器的比特，这样当前数据的上下文接收器按照被考查的上一个数据是否与模板匹配而被选择。这些线路缓存器可能包括比特移位器。一个模板可能被用于一 n 比特图象的每一个比特平面。

在一个实施例中，上下文接收器通过在下一流水线阶段向存储器输出一个地址来选择。地址可能包括一个预定的比特数，例如三比特，来识别比特平面。通过使用三个比特，像素数据中的比特位置可被识别。用于确定上下文的查板也可以被表示为地址的一个部分。用于识别比特平面的比特和识别模板的比特可被组合以生成包含被那些比特定义的上下文接收器状态信息的存储器中特定位置的地址。例如，通过利用三比特来确定一个特定像素中的比特位置，和模板中每一个上一像素的在相同位置的十个先前比特，一个 13 比特的上下文地址可被生成。

使用上下文模型生成的地址，存储器 (如 RAM) 被访问以获得状态信息。状态包括 PEM 状态。PEM 状态包括当前概率估计。由于一个以上的状态使用同一码，PEM 状态不包括一个概率等级或编码标志，更确切地说是一个表格的索引，诸如图 5 展示的表格。在使用诸如图 5 所示的表格时，PEM 状态也提供最可能符号 (MPS)，作为识别当前 PEM 状态是位于表格的正或负边的方法。比特发生状态可能包括计数值和一个 LPS 是否出现的指示。在一实施例中，为了解码下一个编码字，当前运行的 MPS 值也被包括。在本发明中，比特发生器状态被存贮在存储器中，以降低运行计数器要求的空间。如果系统中对每个上下文的每个计数器的

空间成本低，比特发生状态不必存贮在存储器中。

一旦第四阶段完成，新的比特发生器状态和 PEM 状态被写入存储器中。在第五阶段中，编码的数据流被移位到下一码字。移位操作在第六阶段被完成。

图 14C 是本发明的 FIFO 结构 (1401) 的一实施例的方框图，它解释了两个解码器的交织字缓存。注意使用讲授的本发明可支持任何数量的解码器。如图所示，输入数据和 FIFO 的宽度足以容纳两个交织字。FIFO 1401 由 FIFO 1460，寄存器 1461－62，MUX 1463－1464 和控制块 1465 组成。两个输入编码字被连接作为输入交织字。FIFO 1460 的输出被连接至寄存器 1461－1462 的输入。到 MUX 1463 的输入被连接到寄存器 1461 和 1462 的输出。控制块 1465 被连接以提供到 FIFO 1460，寄存器 1461 和 1462 和 MUX 1463 和 1464 的控制信号。交织字是被输出到两个解码器的输出数据 (输出数据 1 和 2)。每个解码器使用一个请求信号来指示当前字已经被使用而新字将被需要。来自解码器的请求信号被连接到控制块 1465 的输入。控制块 1465 也输出一个 FIFO 请求信号以从存储器请求更多的数据。

起初，FIFO 和寄存器 1461 和 1462 用数据填充并在控制单元 1465 中一有效触发器被置位。无论何时一个请求发生，控制块 1465 按照表 16 所示的逻辑来提供数据。

表 16

双有效	请求 1	请求 2	复用器 1	复用器 2	下一个双有效	FIFO 和寄存器开放
0	0	0	X*	X	0	0
0	0	1	X	寄存器1462	1	1
0	1	0	寄存器1462	X	1	1
0	1	1	寄存器1462	FIFO	0	1
1	0	0	X	X	1	0
1	0	1	X	寄存器1461	0	0
1	1	0	寄存器1461	X	0	0
1	1	1	寄存器1461	寄存器1462	1	1

* X 意味着“不关心”

图 15B 解释了本发明的解码器的一个不同概念观点。参考图 15B, 变长 (编码) 数据被输入到一个解码器。解码器输出固定长度 (解码) 数据。输出也被作为一个延迟的反馈，它被作为一个到解码器的输入来接收。在本发明的解码器中，用于解码的变长移位是基于在一定延迟后可出现的解码数据。延迟容限解码器中反馈延迟不降低吞吐量。

输入的变长数据被化分到如图 4 描述的定长交织字中。解码器使用后面图 16 描述的定长字。解码器和延迟构造一个如图 15 和 32 描述的流水线解码器或多个如图 2A－2D 描述的并行解码器。这样，本发明提供了一个延迟容限解码器。本发明的延迟容限

解码器允许并行处理变长数据。

先前技术的解码器（例如，霍夫曼解码器）不是延迟容限的。为了执行解码下一个码字所需的变长移位要求从解码所有先前码字来确定信息。另一方面，本发明提供了延迟容限解码器。

本发明的解码器提供了移位逻辑以移位交织字到合适的比特发生器来解码。当本发明的移位器不需要任何特殊类型的“上下文”或“概率”并行性。一个编码器分配编码字 M 到流 M 模 N(C 语言中为 M%N)，此处 N 为假定的流数。在本发明中，来自当前流的编码数据被提供，直到一个码字被请求。只有这时数据被切换到下一个流。

图 16 解释了本发明的解码器的移位器的一个实施例，移位器 1600 为四个数据流设计。它允许对每个移位操作四个时钟周期。交织字是 16 比特且最长的码字为 13 比特。参考图 16，移位器 1600 由连接到来自交织编码数据的接收输入的四个寄存器 1601－1604 组成。每个寄存器 1601－1604 的输出作为输入连接到 MUX 1605。MUX 1605 的输出被连接到一筒型移位器 1606。筒型移位器 1606 的输出作为输入被连接到一个寄存器 1607，MUX 和寄存器 1608－1610，和一个长度单元 1611。长度单元 1611 的输出被连接到一累加器 1612。累加器 1 612 的一个输出被反馈并连接到筒型移位器 1606。寄存器 1607 的一个输出作为一个输入被连接到 MUS 和寄存器 1608。MUX 和寄存器 1608 的输出作为一个输入连接到 MUX 和寄存器 1309。MUX 和寄存器 1609 的一个输出作为一个输入被连接到 MUX 和寄存器 1610。MUX 和寄存器 1610 的输出是已定位的编码数据。在一个实施例

中，寄存器 1601－1604 是 16 比特寄存器，筒型移位器 1606 是一 32 比特到 13 比特筒型移位器，而累加器 1612 是一 4 比特累加器。

寄存器 1601－1604 从 FIFO 接受 16 比特字并将它们输入到筒型移位器 1606。至少 32 比特未编码数据总是被提供到筒型移位器 1606。四个寄存器 1601－1604 用两个 16 比特编码数据初始化以启动。这允许每个流至少总有一个新的编码字可用。

对于 R－码，码字长度单元 1611 确定是否出现一“0”或“1N”码字，如果是一个“1N”码字，“1”后的多少比特是当前码字的部分。长度单元，提供同样的功能，在图 12 中被描述。对其它码，确定一码字的长度在技术上是熟知的。

移位器 1600 包括一个四个寄存器组成的 FIFO，其中三个有复用输入。寄存器 1607－1610 中的每个寄存器至少包括一个码字，因此寄存器和复用器的宽度为 13 比特以容纳最长可能的码字。每个寄存器还有一个与之相关的控制触发器（未展示以避免使当前发明不明显），它指示一特定的寄存器是否包含一编码字或是否在等待筒型移位器 1606 提供一个码字。

FIFO 将绝不空。每个时钟周期只有一个码字被使用，且每个时钟周期可移位一个码字。执行移位的延迟被补偿，因此系统提前四个码字开始。由于每个码字被移位到定位的编码数据输出，寄存器 1607－1610 中的其它码字移下。当留在 FIFO 中的码字被存储在寄存器 1620 中，筒型移位器 1606 导致码字被从寄存器 1601－1604 中通过 MUX 1605 读出，以填充寄存器 1607－1609。注意 FIFO 可被设计为一旦其码字被移位到寄存器 1608 就用下

一个码字重新填充寄存器 1607。

筒型移位器 1606，编码字长度计算器 1611 和一个累加器选择 1612 处理变长移位。累加器 1612 有四个寄存器，每个编码数据流一个，包含每个数据流当前码字的定位。累加器 1612 是一用于控制筒型移位器 1606 的四比特累加器。累加器 1612 以从码字长度单元 1611 的值输入来增加其值。当累加器 1612 溢出时 (如，每当移位计数为 16 或以上)，寄存器 1601－1604 被定时移位。每其它 16 比特移位导致从 FIFO 请求一个新的 32 比特字。到累加器 1611 的输入是码字的长度，它被当前码和当前码字的第一或两比特确定。注意某些实施例中，寄存器 1601－1604 在解码能开始前，必须用编码数据来初始化。

当系统请求一个码字时，FIFO 中的寄存器被定时，所以码字可朝输出移动。当筒型移位器 1606 准备好移交一新的码字时，它被复用到 FIFO 中第一个空寄存器中。

在这个实施例中，来自比特发生器的下一个码字信号在决定到交换流前被接收。

如果来自比特发生器的下一个码字信号在决定交换流前不能被保证被接收，一个诸如如图 16B 所示的先行系统被采用。参考图 16B，一个使用先行的移位器 1620 被展示为方块图形式。移位器 1620 包括一个产生当前码数据和下一个编码数据输出的移位器 1600。当前编码数据被连接至码字预处理逻辑单元 1621 的一个输入和一编码字处理单元 1624 的一个输入。下一个编码数据被连接到编码字预处理逻辑单元 1622 的一个输入。来自预处理逻辑单元 1621 和 1622 的输出被连接到一 MUX 1623 的输入。

MUX 1623 的输出被连接到码字处理逻辑 1624 的另一个输入。

使用码字的逻辑化分为两部分，码字预处理逻辑和码字处理逻辑。两个相同的流水线预处理单元 1621－1622 在交织流可被移位前操作。如果流被交换，其中一个预处理单元 1621－1622 生成合适的信息；如果流未被交换，另一个单元生成信息。当流被交换时，合适的码字预处理输出被 MUX 1623 复用到编码字处理逻辑 1624，它完成具有适当码字的操作。

在一个实施例中，可能期望使用多片芯片用于外部存储器或外部上下文模型。在这些实施例中，期望降低生成一个比特与使该比特可为使用多片集成电路的上下文模型使用之间的延迟。

图 17 解释了一个具有一外部上下文模型芯片 1701 和一为每个上下文带存储器的编码器芯片 1702 的系统的一个实施例的一方块图。注意只有与编码器芯片中上下文模型有关的单元被展示；那些技术熟练者明显可看出编码器芯片 1702 包含比特发生器，概率估计，等。参考图 17，编码器芯片 1702 包括一零阶上下文模型 1703，上下文模型 1704 和 1705，一个选择逻辑 1706，一个存储器控制 1707 和一个存储器 1708。零阶上下文模型 1703 和上下文模型 1704－1705 生成连接到选择逻辑 1706 的输出。选择逻辑产生 1706 的另一个输入被连接到外部上下文模型芯片 1701 的一个输出。选择逻辑 1706 的输出被连接到存储器 1707 的一个输入。存储器控制 1707 的一个输出也连接到存储器 1708 的一个输入。

选择逻辑 1706 允许使用一个外部上下文模型或一个内部上下文模型（如，零阶上下文模型 1703，上下文模型 1704，上下文模

型 1705)。选择逻辑 17 06 允许上下文模型 1703 的内部零阶部分被使用，即使在使用外部上下文模型 1701 时也一样。零阶上下文模型 1703 提供一或多比特，而外部上下文模型芯片 1701 提供其余的。例如，最接近的前比特可能被反馈并从零阶上下文模型 1703 获取，而前面比特流进入外部上下文模型 1701。在此方式中，时间紧迫的信息留在芯片上。这省略了最近生成的比特的离开芯片的通信延迟。

图 18 是一具有外部上下文模型 1801，外部存储器 1803 和一编码器芯片 1 802 的系统的方框图。参考图 18，某些存储器地址线由外部上下文模型 1801 驱动，而其它由解码器芯片 1802 中的“零阶”上下文模型驱动。即，来自最近过去解码周期的上下文被零阶上下文模型驱动。这允许解码器芯片以最小延迟提供来自最近过去的上下文信息。上下文模型芯片 1802 领先于上下文信息的其余部分使用在过去的过去解码的比特，所以允许通信延迟。在许多情况下，来自立即过去的上下文信息是零阶马尔科夫状态，且来自更远的过去的上下文信息是高阶马尔科夫状态。图 18 所示的实施例省略了在外部上下文模型芯片 1802 中实现的零阶模型内固有的通信延迟。但是，由于解码器芯片 1802 和存储器 1803，仍可能有一个对产生延迟的比特的上下文接收器测定。

值得注意的是，其它存储器体系结构可以被采用。例如，一个具有在一个芯片中的上下文模型和存储器且编码器在另一个芯片中的系统可被使用。一个系统也可能包括一个带有用作某些上下文的一个内部存储器和用作其它上下文的一个外部存储器的编码器芯片。

图 19 展示了一个具有使用存储器的流水线比特发生器的解码器。参考图 19，解码器 1900 由一个上下文模型 1901，存储器 1902 ，PEM 状态到码块 1903，流水线比特发生器 1905，存储器 1904 和移位器 1906 组成。上下文模型 1901 的输入由来自流水线比特流发生器 1905 的解码后数据组成。移位器 190 6 的输入被连接以接收编码数据。上下文模型 1901 的输出被连接到一个存储器 1902 的输入。存储器 1902 的输出被连接到 PEM 状态到编码器 1903。PEM 状态到码 1903 的输出和来自移位器 1906 的定位编码数据输出被连接到比特发生器 1905 的输入。存储器 1904 使用双向总线也被连接到比特发生器 1905，比特发生器 1905 的输出是解码的数据。

上下文模型 1901 输出一个上下文接收器以响应其输入处的编码数据。根据上下文接收器，上下文接收器被用作一个访问存储器 1902 的地址，以获得一下概率状态。概率状态通过为响应概率状态生成概率等级的 PEM 编码状态模块 1903 接收。然后概率等级被用作一个访问存储器 1904 的地址以获得运行计数。运行计数然后被比特发生器 1905 使用来产生解码数据。

在一个实施例中，存储器 1902 包括一个 1024 * 7 比特存储器(此处 1024 是不同上下文的数量)，而存储器 1904 包括一个 25 * 14 比特存储器(此处 25 是不同运行计数的数量)。

由于比特发生器状态 (运行计数，等) 与概率等级而非上下文模型有关，因此，在一个比特能为上下文模型使用前有附加的流水线延迟。由于更新一个比特发生器状态要花多个时钟周期 (比特发生器状态存储器重访问延迟)，多个比特发生器状态将被用

作每个概率等级。例如，如果流水线是六个时钟周期，则比特发生器状态存储器将对每个概率等级有六个输入。一个计数器被用作选择合适的存储器位置。既使对每个概率等级具有多个输入，存储器的大小典型地将小于上下文数量。存储器可以使用多个SRAM 单元或多端口寄存器文件来实现。

因为一个运行计数可能与多个上下文相关，一个实施例必须升级一或多个上下文概率估计状态。在一个实施例中，导致一个运行结束的上下文 PEM 状态被更新。

代替需要一个读出，一个运行计数被重新读出前的修改和写入，一旦修改完成，一个运行计数就可被重新使用。

图 32 解释了本发明的一具体实施例中一解码操作的时序图。参考图 3 2，一个三个周期解码操作被描绘。信号名被列在时序图的左手栏。任何一个周期期间一个信号的有效性用周期 (或部分) 中一条形指示。在一定情况下，负责生成信号或供给有效信号的单元或逻辑被表示在一点画线框内的有效信号边。在时间上，也提供了这里所公开的具体元素和单元。注意扩展到另一周期的信号的任何部分指示信号只在该段信号被展示扩展到其它周期的时间有效。在多于一周期内分别有效的信号也有表示。这样的一个实例是在第二周期末一点及随后第三周期期间再次有效的模板运行计数信号。注意这表明信号仅在周期末被寄存。一个相关性列表如下面从同一或前一时钟周期到信号被定义有效所当前时间向前置相关性的表 17 所示。

表 17

名称	单元 *	相关性 * *
寄存器文件 1	CM	(前比特, CM 移位寄存器)
编码状态	CM	寄存器文件 1
简型移位	SH	简型移位器输出(定位的编码数据)
长度	SH	(累加器寄存器, 非定位数据寄存器) (K, R3)
累加(累加器)	SH	长度 (上一个累加器寄存器值)
寄存器文件 2	BG	(被寄存的 K, R3)
α (需要的码字)	BG	寄存器文件 2
编码到(掩码, 最大 RL, R3 分辨)	BG	(被寄存的 K, R3)
发生比特(发生器比特)	BG	寄存器文件 2 简型移位器输出(定位的编码数据) 编码到(掩码, 最大 RL, R3 分辨) (寄存器文件 1, 寄存的 MPS)
解码	BG	简型移位器输出(定位的编码数据)
PEM 表	PEM	(寄存的 K, R3)
β (PEM 更新) + +	PEM	寄存的: PEM 表输出, LPS 现值, 继续)
−1(运行计数更新)	BG	(寄存的: 需要的编码字, 运行计数, LPS 现值, 继续)
γ (继续, LPS 现值, 更新)	BG	(寄存的: 需要被寄存的编码字, 运行计数, LPS 现值, 继续)

* CM = 上下文模型, SH = 移位器, BG = 比特发生器, PEM = 概率估计机。

* * [斜体] 含意为来自上一个时钟周期的相关性。

+ + 在一实施例中, 更新 PEM 状态的大多数结合逻辑按“PEM 表”步骤执行;“PEM”更新是一个简单的复用操作。

在某些实施例中, 解码器必须使用编码器有限重新排序缓存器模型。在一具体实施例中, 这个模型使用隐含信令实现构造。

如前解释的编码器, 当一个码字在编码器中启始时, 按照码

字将出现在通道上的顺序，在适当缓冲器中，为码字保留空间。当一个缓存器中最后一个空间为一个新的编码字所保留，则无论已被完全确定与否，某些编码字被放在被压缩的比特流中。

当一部分码字必须被完成时，一个短的且正确定义了目前为止接收符号的码字可以被选择。例如，在一个 R－编码器系统中，假如在一个具有 128 最大运行长度的运行码中必须提供完成一个 100 MPS 系列编码字，则可以使用 128 MPS 的编码字，因为这正确地定义了前 100 个符号。

可选地，可以使用一个定义 100 MPS 的编码字跟随一个 LPS。当码字已经完成时，它可以被移出重新排序缓存器并加到编码流中。这可以允许先前完成的码字也被放入码流中。如果强制完成一部分编码字导致一码字从满的缓存器中移去，则编码可继续。如果一缓存器仍然满，则下一个码字必须再次被完成并加到编码流中。这一过程持续直到满的缓存器不再满。解码器可能利用一个用于每个比特发生器状态信息存储器位置的计数器构造隐含信令编码器。

在一个实施例中，每个运行计数器 (此例中的概率等级) 有一个与编码器 (如，6 比特) 中头或尾计数器相同大小的计数器。每次启动一个新的运行 (一个新的码字被取出)，相应的计数被编码字存储器大小装载。每次启动一个运行 (一个新的编码字被取出)，所有计数器被减 1。任何达到零的计数器导致相应的运行计数被清除 (和 LPS 当前标志被清除)。

本发明中的实时编码要求解码器处理未跟随一个 LPS 并不是最大运行长度的 MPS 的运行。这发生在编码器开始一个 MPS

运行时，但到运行完成之前仍没有充足的有限重新排序存储器来等待。该条件要求下次这个上下文接收器被使用时，一个新的码字被解码，且这个条件必须通知解码器。下面描述三种修改解码器的潜在方法。

当缓存器满时，上下文接收器的运行计数或被迫出的概率等级必须被复位。为了有效地实现它，在码字存储器中存储上下文接收器或概率等级是有用的。由于对没有一个相关编码字的运行才需要，用于存储码字的存储器可以被共享。注意在某些系统中，替代迫出一个不完整码字的方法是，在缓存器满时比特可被迫入缓存器中未处理的（或任何）编码字的上下文/概率等级中。解码器检测它并使用相应的(错误)上下文接收器或概率等级。

在流中信令使用码字通知解码器。在一实施例，R2 (K) 和 R3 (K) 码定义被改变成包括未跟随一个 LPS 的 MPS 的非最大长度运行。这可以通过给最低发生概率的编码字至少增加 1 比特来实现。这允许一个对非最大长度运行计数的唯一可解码前缀。表 18 展示一个允许流内信令的 R2 (2) 码的替换。这一方法的缺点是，R－码解码逻辑必须被改变且每次最低概率码字出现时存在一压缩消费。

· 表 18

原数据	码字
0000	0
0001	1000
001	101
01	110
1	111
000	100100
00	100101
0	10011

在某些实施例中，解码器使用时间标记执行隐含信令。一个计数器通过每次请求码字时加 1 来保持跟踪当前“时间”。当一个码字开始时，当前“时间”被存储到与码字相关的存储器中。一个编码字被首次利用后的任何时间，相应存储的“时间”值加编码器重新排序缓存器长度被与当前“时间”比较。如果当前“时间”大，一个隐含信令被生成，一个新的码字被请求。这样，编码器中有限的重新排序存储器已经被模拟。在一实施例中，使用充足的“时间”值比特以允许所有编码字被枚举。

为了降低需要的存储器，用于时间标记的比特数量被保持一个最小值。如果时间标记使用一个小比特数量，这样的时间值被重用，必须小心的是所有旧的时间标记在计数器开始重用时间前被注释。使 N 是队列或比特发生器状态存储器的地址比特数中较大的。具有 N + 1 比特的时间标记可以被使用。比特发生器状态存储器必须支持多个存取，也许每解码比特两个读出和两个写入。一个计数器被用于穿越比特发生器状态存储器的循环，每个比特

解码计数器加 1。任何太旧的存储器位置被清除，所以在其未来使用时，一个新的编码字被取出。这保证任何时间被重用前所有时间标记被检查。

如果比特发生器状态存储器小于队列，计数（时间标记计数器）速率和要求的存储器带宽可以被降低。这是因为每数个被请求使用整个队列的周期，每个时间标记（每个比特发生器状态存储器一个）必须被检查一次。在一个不同的存储器中存贮时间标记可以降低需要的存储器带宽。在一个对部分运行使用“0”码字的系统中，不必对“1N”编码字检查时间标记。在一个对部分运行使用“1N”码字的系统中，时间标只在生成一个 LPS 前必须被检查。

在某些实施例中，隐含信令在解码期间使用一个队列实现。该方法在一个编码硬件在解码期间可用的半双工系统中可能是有用的。队列的操作和在编码期间几乎一样。当一个新的编码字被请求时，其索引被放到队列中并标志为无效。当来自一个码字的数据完成时，其队列输入被标志为有效，数据被移出队列以给新的码字腾出空间，如果被移出的数据标志为无效，来自该索引的比特发生器状态信息被清除。该清除操作可能要求比特发生器状态存储器能够支持一个额外的写入操作。

写入相对的显式信令通知缓存器溢出作为压缩的数据。一个实例是用一个辅助上下文接收器，该接收器对每一个正常上下文解码使用一次或对每一个被解码的码字使用一次。来自辅助上下文接收器的解码比特指示是否发生需要新的编码字条件，且一个新的码字必须对相应的正常上下文接收器解码。在这种情况下，

该特殊上下文的码字必须被恰当地重新排序。由于该上下文的使用是某些重新排序单元已知的因素的函数(典型的对每个码字使用一次)，重新排序辅助上下文所需的存储器可以被隐含限制或构造。辅助上下文所允许的可能编码可以被限制。

当解码以生成一个新的码字必须被解码的信号时，隐含信令模拟了编码器的有限缓存器。在一实施例中，对每个上下文维持一个时间标记。在一个实施例中，编码器的有限大小重新排序缓存器被直接设计。在一个双半工系统中，由于编码器的重新排序电路在解码期间可用，它可用于生成解码器信号。

隐含信令如何实现依赖于编码器如何识别和处理缓存器满的情况。对一个使用一具有固定分配的合并队列的系统，多个头指针的使用允许“缓存器满”含意的选择。给定一个编码器的设计，一个相应的模型可被设计。

下面提供编码器操作和一个为具有固定流分配，被概率系统并行的合并队列解码器所使用的模型。对于这个实施例，假设重新排序缓存器有256个位置，使用4个交织流，且每个交织字是16比特。当缓存器包含256个项时，在第257码字项可以被放入队列前，一个项必须被送出到一个比特包组合器(如，比特组合单元)。如果必要项可以被更早地逐出。

在某些系统中，移去缓存器中的第一项需要移去充足的比特以完成整个项的交织码字。因此，如果1－比特码字是可能的，移去码字0可能也要求移去16－比特交织字的码字4, 8, 12, ……, 52, 56, 60。为了确保所有的这些缓存器项是有效的，由于存储器是满而强迫填充一个项可以在一个新的码字被输入 (256－16＊

4 = 192) 处的地址 64, 192 位置被执行。

在解码器中，对每个概率有一个计数器。当一个新的码字被用于开始一个运行时，计数器用 192 装载。任何时候一新的码字为任何概率使用时，所有的计数器减少。如果任何计数器达到零，该概率的运行长度被置为零(且 LPS 当前标志被清除)。

使用多个 RAM 单元 (多端口存储器，用快速存储器仿真等) 可能是方便的，每个编码数据流一个单元。这允许所有比特组合单元同时接收数据，因此，读出特定流的多个码字不禁止其它流的读出。

在其它系统中，多个比特组合单元必须根据存储在缓存器中的码字次序，对单一存储器仲裁。在这些系统中，从一个缓存器中移去一个项可能未完成一个交织字。每个比特组合单元典型地依次接收一个交织字的某些部分。每个比特组合单元接收至少等于最短码字长度 (如，1 比特) 的比特数，且至多等于最长码字长度 (如，13 比特) 数量的比特。交织字完成后可被发送，且必须以初始化的顺序发送。在这个实施例中，一个比特组合单元可能必须缓存 13 个交织字，这是用最大长度编码字完成的交织字的最大数量，而另一流使一个正接收最小长度码字的交织字挂起。

一个每一码字需要两个存储器写入和一个读出的系统的硬件可能较一个执行两个写入和两个读出的系统缺乏吸引力。如果具有四个流的实例系统希望这样，比特组合单元 1 和 2 可以共享一个存储器读出周期且比特组合单元 1 和 3 可以共享另一读出周期 (或任何其它任意性组合)。当这未降低所需缓存大小时，应当允许一个更高的进入比特组合单元的传输速率。这可能允许比

特组合单元更好地利用编码数据通道的容量。

每个概率等级有多个比特发生器状态的系统的优点是，系统在一个固定大小存储器溢出时可以支持有损编码。这对一个帧缓存器的图象压缩和只存储有限数量编码数据的其它应用可能是有用的。

对于具有修复容量存储器的系统，每个概率的多个比特发生器状态分配到数据的一部分。例如，八个状态的每一个可以被分配给八比特数据的一个特殊比特面。在这种情况下，一个移位器也被分配给数据的每个部分，相对的是顺序提供下一个编码字的移位器。值得注意的是数据不需要被比特面化分。在编码器中，不执行交织，数据的每个部分被简单地进行比特组合。存储器被分配到数据的每个部分。

编码数据的存储器管理用于在一个固定大小存储器中存储所有数据的系统和用于在一个具有最大允许带宽通道发送数据的系统。在这两个系统中，期望优雅降级成一个有损系统。数据的不同流为具有不同重要性的数据所使用，因此在没有充分的存储或带宽可用时，次要的流可以被存储或不被发送。

使用存储器时，编码数据必须被存储，因此它可被访问，次要数据可以被丢弃而不损失解码重要数据流的能力。由于编码数据是变长的，可以使用动态存储器分配。图 31 表示了一个为三个编码数据流动态分配存储器四单元的例子。一个寄存器文件 3100 (或其它存储) 包括每个流的，一个指针及指示下一个空闲存储器位置的另一指针。存储器 3101 被化分为固定大小的页。

起初分配给流的每个指针指向一存储器页的起点，而空闲指

针指向存储器的下一个可用页。来自一特定流的编码数据被存储在由相应指针寻址的存储器位置。指针随后增加到下一个存储器位置。

当指针到达当前页的最大值时，发生下列情况：下一个空闲页(存储在空闲指针中)的开始地址用当前页存储。(可以使用编码数据存储器的一部分或一个分离的存储器或寄存器文件。)当前指针被置为下一个空闲页。空闲指针被增加。这些活动分配一个新的存储器页给一个特定的流并提供链接以使分配的次序可以在解码期间被确定。

当所有的存储器页均被使用且有来自一个较存储器中最次要数据更严重的流的更多的数据时，可以采取三种措施之一。在所有三种情况下，分配给最不重要数据流的存储器被重新分配给更重要的数据流，且来自最不重要的数据流的后继数据不被存储。

第一种，当前由最次要流使用的页被简单地分配给更重要的数据。由于大多数典型的熵编码器使用内部状态信息，先前存储在该页的所有最次要数据被丢失。

第二种，当前由最次要流使用的页被简单地分配给更重要的数据流。不象上一种情况，指针被置到页的结尾，且随着更重要的数据被写入页，相应的指针递减。如果更重要数据流不要求整个页，该法具有保护在页开始处最次要数据的优点。

第三种，替代重新分配最次要数据的当前页，最次要数据的任何页可以被重新分配。这要求所有页的编码数据被独立编码，这可能会降低已取得的压缩。它也要求相应于所有页起点的未编

码数据被标识。由于最次要数据的任何页可以被丢弃，可以获得到有损编码的优雅降级的更大灵活性。

第三种选择对于在图象区域上取得一固定压缩率的系统可能特别有吸引力。一特定数量的存储器页可以被分配到一个图象区域。次要数据被保持与否依赖在该特定区域中获得的压缩。(如果无损压缩要求较已分配存储器数量少的话，分配给该区域的存储器可能没有被充分利用。) 在一个图象区域上获得一固定压缩率可以支持对图象区域的随机存取。

从两端点写入数据至每一页的能力可以被用于较好地利用系统中整个可用存储器。当所有页被分配时，任何在尾部有充足空闲空间的页可以分配从尾部使用。利用一个页的两端的能力必须与保持对两类数据相遇位置跟踪的成本的平衡 (这与数据类型之一是不重要的且可以被简单地复盖的情况不同)。

现在考虑一个数据在一个通道中被发送而不在存储器中存储的系统。使用存储器的固定大小的页，但每个流只需要一页。(或如果为对通道提供缓存需要乒－乓，也许要两个，这样当写入一个时，另一个可以被读出供输出) 当一个存储器页满时，它被在通道中传送，且只要页一被发送，存储器位置就可以被重新使用。在某些应用中，存储器的页大小可以是用于通道中数据分组的大小或多个分组大小。

在某些通信系统中，例如 ATM (异步转移模式)，可以为分组分配优先级。ATM 有两个优先级等，优先级和次要优先级。只有有充足带宽可用时才传输次要优选级分组。可用一个门限来确定哪个流是优先级，哪个流是次要优先级。另一方法是在编码器处

使用一个门限来不发送重要性低于一个门限的流。

图 20 的一个用于每个码的分离比特发生器的系统的方块图。参考图 20, 解码系统 2000 由上下文模型 2001, 存储器 2002, PEM 状态到码块 2003, 解码器 2004, 比特发生器 2005A－n, 和移位器 2006 组成。上下文模型 2001 的输出被连接到存储器 2002 的一个输入。存储器 2002 的输出被连接到 PEM 状态到码块 2003 的一个输入。PEM 状态到码块 2003 的输出被连接到解码器 2004 的一个输入。解码器 2004 的输出作为比特发生器 2005A－n 的一个允许被连接。比特发生器 2005A－n 也被连接以接收来自移位器 2006 的编码数据输出。

上下文模型 2001, 存储器 2002, 和 PEM 状态到码块 2003 象图 19 中其相对部分那样操作。上下文模型 2001 生成一个上下文接收器。存储器 2002 基于上下文接收器输出一个概率状态。概率状态被 PEM 状态到码块 2003 接收, 2003 为每一概率状态产生概率等级。解码器 2004 根据解码的概率等级开放其中一种比特发生器 2005A－n。(注意解码器 2004 是一个相似于闻名的 74X138 3: 8 解码器的 M 到 2^ M 解码器器电路－－它不是一个熵编码解码器。) 注意由于每个码有一个分离的比特发生器, 某些比特发生器可能使用非 R－码。特别地, 概率接近 60% 的一个编码可能被用以更好地覆盖 R2 (0) 和 R2 (1) 间的概率空闲。例如, 表 19 描绘了这样一个码。

表 19

未编码数据	码字
0 0 0	0 0
0 0 1	0 1
0 1	1 0
1	1 1

如果需要获得期望的速率，一个或多个比特的预解码可以被进行以保证解码数据快速出现。在某些实施例中，为了避免对每个时钟周期能更新大量的运行计数的需求，对长码的码字解码和运行计数两者被分开。

R2(0)码的比特发生器是不复杂的。每次一个比特被请求，一个编码字也被请求。生成的比特是简单的码字(被用 MPS 异或)。

短运行长度的码，例如，R2(1)，R3(1)，R2(2)和 R3(2)，被以下列方式处理。一个码字中的所有比特被解码并存贮在一个由一个小型计数器 (相应 1, 2 或 3) 和一个 LPS 当前比特组成的状态机中。计数器和 LPS 当前比特作为一个 R－码解码器运行。

对较长的码，诸如 k> 2 的 R2(k)和 R3(k)，比特发生器被划分为两个如图 21 的单元。参考图 21，一个 k> 2 的 R2(k)码的比特发生器的结构被表示为一个短运行单元 2101 和一个长运行单元 2102。注意虽然该结构用于 R2 (k> 2) 码，它的操作将相似于 R3(k> 2)码(且对一位技术上熟练的人员是显而易见的)。

短运行单元 2101 被连接以接收一个使能信号和一个作为到比特发生器输入的码字 [0..2] 和一个“全零”和一个“计数零”信

号 (指示一零计数)，两者都来自长运行单元 2102。为了响应这些输入，短运行单元 2101 输出一个解码的比特和一个下一个信号指示，它通知需要一个新的码字。短运行单元 2101 也生成一个计数使能信号，一个计数装载信号和一个计数最大信号到长运行单元 2102。长运行单元 2102 也被连接以接收作为到比特发生器输入的码字 [k. . . 3]。

短运行单元 2101 处理多达长度 4 的运行且相似于 R2 (2) 比特发生器。在一个实施例中，短运行单元 2101 对所有的 R2 (k > 2) 码是相同的。长运行单元 2102 的目的是确定何时输出运行的最后 1－4 比特。长运行单元 2102 有输入，与逻辑和一个大小随 k 变化的计数器。

长运行计数单元 2102 的一个实施例如图 22 所示。参考图 22，长运行单元 2102 由被连接来接收编码字 [k. . . 3] 的与逻辑 2201 组成，如果码字中所有比特为 1 的话输出一个作为一个逻辑 1 的“全 1”信号，从而指示当前码字是一个 1N 码字且运行计数小于 4。非逻辑 2202 也被连接来接收码字并翻转它。非逻辑 2202 的输出被连接到一个比特计数器 2203 的一个输入。比特计数器 2203 也被连接以接收计数允许信号，计数装载信号和计数最大信号。为了响应这些输入，比特计数器 2203 生成一个计数零信号。

在一实施例中，计数器 2003 是一个 k－2 比特计数器并被用作把长运行计数分成四个 MPS 的运行和可能有某些剩余。计数允许信号指示四个 MPS 已经被输出且计数器应当被递减。解码“1N”编码字时用计数装载信号并导致计数器被用码字比特 k 到3 装载。解码“0”编码字时用计数最大信号并用其最大值装载计数

器。一个计数零输出信号指示计数器何时为零。

短运行计数单元2101的一实施例如图23所示。参考图23，短运行计数单元包含一个控制模块2301，一个两比特计数器2302和一个三比特计数器2303。控制模块2301从长运行计数单元接收允许信号，编码字[0...2]，和全1和计数零信号。两比特计数器用于计数为较长运行部分的四比特MP S运行。一个R2 (2)计数器和LPS比特(总计3比特)2303用于生成运行结束处的1－4比特。允许输入指示应当在比特输出端被生成一个比特。未被认定时的计数零输入指示一个四MPS的运行应当被输出。一旦MPS计数器2302到达零时，计数允许输出被确定。当计数零输入被认定时，或LPS使用的R2(2)计数器或一个新的码字被解码且下一个输出被确定。

当新的码字被解码时，执行的动作由码字输入确定。如果输入为“0”码字，MPS计数器2302被使用并且计数最大输出被确定。对于“1N”码字，码字的前三个比特被装载到R2(2)计数器和LPS2303，且计数装载输出被确定。如果全1输入被确定则R2(2)计数器和LPS2303用于生成比特，否则MPS计数器被使用，直到计数零输入被确定。

从一个系统的角度，编码的数量必须少以使系统工作良好，典型地为25或以下。比特和下一个码字输出和开放一个特定比特发生器的解码器所需的复用器的大小必须被限制以保证快速操作。同样，来自移位器的码字扇出对于高速操作不能太高。

每个编码的分离比特发生器允许流水线处理。如果所有的码字导致至少两比特，码字的处理可以在两个周期而非一个周期内

被流水线进行。如果比特发生器是系统的有限部分则可能加倍解码器的速度。完成此的一个方法是运行长度零编码字（码字指示恰为一个 LPS）之后跟随一个为下一个未编码比特的比特。这些可以被称为 RN (k) + 1 码且总是对至少两比特编码。注意 R2 (0) 码字和或许某些其它短码字不需要为了速度而被流水线处理。

分离比特发生器使用隐含信令对自己有好处。用有限存储器编码的隐含信令可以以如下方式完成。每个比特发生器有一个为队列地址尺寸的计数器，例如，当使用一个大小为 512 的队列是 9 比特。每当一个新的码字为一比特发生器所用时，用最大值装载计数器。任何时候任何比特发生器请示一个码字，所有比特发生器的计数器被递减。一旦一个计数器到达零，相应的比特发生器的状态被清除（例如，MPS 计数器，R2 (2) 计数器和 LPS 和长运行计数器被清除）。即使一个具体比特发生器未被使能清除可以发生没有阻塞计数问题。

在每个上下文接收器存储器包括概率估计信息的情况下，为了非常迅速地初始化解码器（如，存储器）可能需要额外的存储器带宽。在解码器有许多上下文且它们都需要被清除时，迅速初始化解码器是一个问题。当解码器支持许多上下文（1k 或更多的）且存储器不能被全部清除时，为了清除存储器将需要不可接受的大量的时钟周期。

为了迅速清除上下文，本发明的某些实施例使用一个额外的比特，这是定义为初始化状态比特，它用每个上下文存贮。这样，一个额外的比特用每个上下文的 PEM 状态（如，8 比特）存贮。

每个上下文接收器的存储器和初始化控制逻辑如图 24 所

示。参考图 2 4, 一个上下文存储器 2401 连接到寄存器 2402。在一实施例中，寄存器 2 402 包括一个指示初始化状态接收器的当前合适状态的一比特寄存器。寄存器 2402 被连接到异或逻辑 2403 的一个输入。另一个到异或逻辑 2403 的输入被连接到存储器 2401 的一个输出。异或逻辑 2403 的输出是有效信号且被连接到控制逻辑 2404 的一个输入。控制逻辑 2404 的其它输入被连接到计数器 2405 的输出和上下文接收器信号。控制逻辑 2404 的一个输出被连接到复用器 2406－2407 的选择输入和计数器 2405 的一个输入。控制逻辑 2404 的另一个输出被连接到复用器 2408 的选择输入。复用器 2406 的输入被连接到计数器 240 5 的输出和上下文接收器指示。复用器 2406 的输出被连接到存储器 2401。复用器 2407 的输入被连接到新的 PEM 状态和零。复用器 2407 的输出被连接到存储器 2401 的一个输入存储器 2401 的输出且初始 PEM 状态被连接到复用器 2408 的输入。复用器 2408 的输出是 PEM 状态出。

解码操作每出现一次, 对寄存器 2402 中的值取补。(即, 每个数据集合, 不是每个解码的比特)。为了确定被访问存储器位置对该解码操作是否有效。异或逻辑 2403 用寄存器值比较被访问存储器位置的有效性。使用异或逻辑 2403 检查初始的状态比特是否匹配寄存器 2402 中的合适状态。如果存储器 2401 中的数据是无效的，则控制逻辑 2404 导致数据被状态到编码逻辑忽略且将用初始 PEM 状态代之。这用复用器 2408 完成。当一个新的 PEM 状态被写入存储器时，初始化的比特被置为寄存器的当前值，因此再次访问时将被认为有效。

在另一个解码操作开始前，每一个上下文接收器存储器入口必须维持其初始化状态比特被置为寄存器的当前值。计数器 2406 遍历通过所有的存储器位置以确保它们被初始化。当一个上下文接收器被使用，但其 PEM 状态入未更新时，未使用的写入周期可以被用于测试或更新计数器 2405 指向的存储器位置。一个解码操作完成后，如果计数器 2405 没有到达最大值，其余上下文在开始下一个操作前被初始化。下列逻辑用于控制操作。

```
写入它 = 伪;
计数器 = 0
全部初始化 = 伪;
当(计数器〈最大上下文接收器 + 1)
    从上下文存储器读出 PEM 状态
    如果((计数器 = = 上下文接收器读)且(写入它))
      写入它 = 伪
      计数器 = 计数器 + 1
    如果((PEM 状态改变)
      写入新的 PEM 状态
    否则  如果(写入它)
      写入初始 PEM 状态到存储器位置“计数器”
      计数器 = 计数器 + 1
    否则
      读出存储器位置“计数器”
      如果(在读出位置内的初始比特处于错误状态)
        写入它 = 真
```

否则

计数器 = 计数器 + 1

全部初始化 = 真;

当(解码)

从上下文存储器读出 PEM 状态

如果(PEM 状态改变)

写入新的 PEM 状态

用在本发明中的 PEM 可能包括一个适配方案以允许较快地自适应而不管出现的数据数量。通过这样作，本发明允许开始时解码自适应更快速，且更多的数据出现时自适应较慢，作为一个提供一个更为准确估计的方法。进一步地，PEM 可能被固化在一个 PEM 状态表/机的一个现场可编程门阵列(FPGA)或 ASIC 实现中。

下面的表 20－25 描述了许多概率估计状态机。为了降低硬件成本，某些表不使用 R3 编码或不使用长编码。除表 20 外的所有表使用“快速适应”特殊状态编码开始直到出现第一个 LPS，快速自适应。这些快速自适应状态在表中表示为斜体。例如，参考表 21，当解码开始时，当前状态为状态 0，如果一个 MPS 发生，则解码器转换到状态 35。只要 MPS 出现，解码器从状态 35 向上转换，最终转换到状态 28。如果一个 LPS 在任何时候出现，解码器转换出粗体快速自适应状态到表示已经被接收数据的正确概率状态的一个状态。

注意对每个表，在一定数量的 MPS 已经被接收后，解码器转

换出快速自适应状态。在期望的实施例中，一旦快速自适应状态已经被退出，没有返回它们的机制，除了重启动解码过程。在其它实施例中，状态表可以被设计成通过允许较快速的自适应来重入这些快速自适应状态，当前发明允许解码器较快到达更斜的码，因此可能从改进的压缩中得益。注意通过改变当前状态 0 的表条目，使一个具体的表被省略，这样表转换根据数据输入只改变一个向上或向下状态。

对于所有表格，每个状态的数据是该状态码，正向更新（向上）的下一个状态和负向更新（向下）的下一个状态。星号指示 MPS 必须以负更新被改变的状态。

表 20

当前状态	码	向上的下一个状态	向下的下一个状态
0	r2(0)	1	0*
1	r2(0)	2	0
2	r2(0)	3	1
3	r2(0)	4	2
4	r2(0)	5	3
5	r2(0)	6	4
6	r2(1)	7	5
7	r2(1)	8	6
8	r2(1)	9	7
9	r2(1)	10	8
10	r2(1)	11	9
11	r2(1)	12	10
12	r3(1)	13	11
13	r3(1)	14	12
14	r3(1)	15	13
15	r2(2)	16	14
16	r3(2)	17	15
17	r2(3)	18	16
18	r3(3)	19	17
19	r2(4)	20	18
20	r3(4)	21	19
21	r2(5)	22	20
22	r3(5)	23	21
23	r2(6)	24	22
24	r3(6)	25	23
25	r2(7)	26	24
26	r3(7)	27	25
27	r2(8)	28	26
28	r3(8)	29	27
29	r2(9)	30	28
30	r3(9)	31	29
31	r2(10)	32	30
32	r3(10)	33	31
33	r2(11)	34	32
34	r3(11)	34	33

* 切换到MPS

表 21

当前状态	码	向上的下一个状态	向下的下一个状态
0	r2(0)	35	35*
1	r2(0)	2	1*
2	r2(0)	3	1
3	r2(0)	4	2
4	r2(0)	5	3
5	r2(0)	6	4
6	r2(1)	7	5
7	r2(1)	8	6
8	r2(1)	9	7
9	r2(1)	10	8
10	r2(1)	11	9
11	r2(1)	12	10
12	r3(1)	13	11
13	r3(1)	14	12
14	r3(1)	15	13
15	r2(2)	16	14
16	r3(2)	17	15
17	r2(3)	18	16
18	r3(3)	19	17
19	r2(4)	20	18
20	r3(4)	21	19
21	r2(5)	22	20
22	r3(5)	23	21
23	r2(6)	24	22
24	r3(6)	25	23
25	r2(7)	26	24
26	r3(7)	27	25
27	r2(8)	28	26
28	r3(8)	29	27
29	r2(9)	30	28
30	r3(9)	31	29
31	r2(10)	32	30
32	r3(10)	33	31
33	r2(11)	34	32
34	r3(11)	34	33
35	r2(0)	36	1*
36	r2(1)	37	2
37	r2(2)	38	4
38	r2(3)	39	6
39	r2(4)	40	10
40	r2(5)	41	16
41	r2(6)	42	19
42	r2(7)	43	22
43	r2(8)	28	25

* 切换到MPS

表. 22

当前状态	码	向上的下一个状态	向下的下一个状态
0	r2(0)	35	35*
1	r2(0)	2	1*
2	r2(0)	3	1
3	r2(0)	4	2
4	r2(0)	5	3
5	r2(0)	6	4
6	r2(1)	7	5
7	r2(1)	8	6
8	r2(1)	9	7
9	r2(1)	10	8
10	r2(1)	11	9
11	r2(1)	12	10
12	r2(1)	13	11
13	r2(2)	14	12
14	r2(2)	15	13
15	r2(2)	16	14
16	r2(2)	17	15
17	r2(3)	18	16
18	r2(3)	19	17
19	r2(4)	20	18
20	r2(4)	21	19
21	r2(5)	22	20
22	r2(5)	23	21
23	r2(6)	24	22
24	r2(6)	25	23
25	r2(7)	26	24
26	r2(7)	27	25
27	r2(8)	28	26
28	r2(8)	29	27
29	r2(9)	30	28
30	r2(9)	31	29
31	r2(10)	32	30
32	r2(10)	33	31
33	r2(11)	33	32
35	r2(0)	36	1*
36	r2(1)	37	2
37	r2(2)	38	4
38	r2(3)	39	6
39	r2(4)	40	10
40	r2(5)	41	16
41	r2(6)	42	19
42	r2(7)	43	22
43	r2(8)	28	25

* 切换到MPS

表.23

当前状态	码	向上的下一个状态	向下的下一个状态
0	r2(0)	35	35*
1	r2(0)	2	1*
2	r2(0)	3	1
3	r2(0)	4	2
4	r2(0)	5	3
5	r2(0)	6	4
6	r2(1)	7	5
7	r2(1)	8	6
8	r2(1)	9	7
9	r2(1)	10	8
10	r2(1)	11	9
11	r2(1)	12	10
12	r3(1)	13	11
13	r3(1)	14	12
14	r3(1)	15	13
15	r2(2)	16	14
16	r3(2)	17	15
17	r2(3)	18	16
18	r3(3)	19	17
19	r2(4)	20	18
20	r3(4)	21	19
21	r2(5)	22	20
22	r2(5)	23	21
23	r2(6)	24	22
24	r2(6)	25	23
25	r2(7)	26	24
26	r2(7)	27	25
27	r2(8)	28	26
28	r2(8)	29	27
29	r2(9)	30	28
30	r2(9)	31	29
31	r2(10)	32	30
32	r2(10)	33	31
33	r2(11)	34	32
34	r2(11)	34	33
35	r2(0)	36	1*
36	r2(1)	37	2
37	r2(2)	38	4
38	r2(3)	39	6
39	r2(4)	40	10
40	r2(5)	41	16
41	r2(6)	42	19
42	r2(7)	43	22
43	r2(8)	28	25

• 切换到MPS

表 ·24

当前状态	码	向上的下一个状态	向下的下一个状态
0	r2(0)	35	35*
1	r2(0)	2	1*
2	r2(0)	3	1
3	r2(0)	4	2
4	r2(0)	5	3
5	r2(0)	6	4
6	r2(1)	7	5
7	r2(1)	8	6
8	r2(1)	9	7
9	r2(1)	10	8
10	r2(1)	11	9
11	r2(1)	12	10
12	r3(1)	13	11
13	r3(1)	14	12
14	r3(1)	15	13
15	r2(2)	16	14
16	r3(2)	17	15
17	r2(3)	18	16
18	r3(3)	19	17
19	r2(4)	20	18
20	r3(4)	21	19
21	r2(5)	22	20
22	r3(5)	23	21
23	r2(6)	24	22
24	r3(6)	25	23
25	r2(7)	26	24
26	r2(7)	27	25
27	r2(7)	27	26
35	r2(0)	36	1*
36	r2(1)	37	2
37	r2(2)	38	4
38	r2(3)	39	6
39	r2(4)	40	10
40	r2(5)	41	16
41	r2(6)	42	19
42	r2(7)	25	22

* 切换到MPS

表 25

当前状态	码	向上的下一个状态	向下的下一个状态
0	r2(0)	35	35*
1	r2(0)	2	1*
2	r2(0)	3	1
3	r2(0)	4	2
4	r2(0)	5	3
5	r2(0)	6	4
6	r2(1)	7	5
7	r2(1)	8	6
8	r2(1)	9	7
9	r2(1)	10	8
10	r2(1)	11	9
11	r2(1)	12	10
12	r2(1)	13	11
13	r2(2)	14	12
14	r2(2)	15	13
15	r2(2)	16	14
16	r2(2)	17	15
17	r2(3)	18	16
18	r2(3)	19	17
19	r2(4)	20	18
20	r2(4)	21	19
21	r2(5)	22	20
22	r2(5)	23	21
23	r2(6)	24	22
24	r2(6)	25	23
25	r2(7)	26	24
26	r2(7)	27	25
27	r2(7)	28	26
28	r2(7)	28	27
35	r2(0)	36	1*
36	r2(1)	37	2
37	r2(2)	38	4
38	r2(3)	39	6
39	r2(4)	40	10
40	r2(5)	41	16
41	r2(6)	42	19
42	r2(7)	25	22
43	r2(8)	28	25

• 切换到MPS

对概率估计增加一个快速自适应只有助于编码开始处。当一个上下文接收器的统计改变较前面描述的 PEM 状态表可跟踪的更迅速时，其它方法可以被用于改进编码期间的自适应。

维持贯穿编码的快速自适应的一个方法是给 PEM 状态更新增加一个加速项。该加速可以通过每一个码重复一常数次数 (如，8) 被合并到一个 P EM 状态表中。然后在更新时，一个加速项 M (如，一个正整数) 可以被增加或从当前状态中删去。当 M 为 1 时，系统按没有加速和最慢自适应发生的系统操作。当 M 大于 1 时，发生较快自适应。起初 M 可以置为某个大于 1 的值以提供一个初始的快速自适应。

本发明的一个更新 M 值的方法是根据连续编码字的数量。例如，如果一个预定数量的码字连续出现，则 M 值被增加。例如，如果四个连续的码字是“0”“0”“0”“0”“1N”“1N”“1N”“1N”，则 M 值增加。另一方面，一个在“0”和“1N”码字间交换的码型可以被用于 M 值的降低。例如，如果四个连续的码字是“0”“1N”“0”“1N”或“1N”“0”“1N”“0”，则 M 值被降低。

另一个加速方法采用每个码被重复 S 次的状态表，此处 S 是正整数。S 是一个反向加速参数。当 S 为 1 时，自适应为快速，而当 S 较大时，自适应较慢。S 值可以被初始化为 1 以提供启始快速自适应。使用一个与上述相似的方法，当四个连续码字是“0”“0”“0”“0”或“1N”“1N”“1N”“1N”时，S 值可以被更新。在这种情况下，S 值被降低。相对地，如果四个连续码字为“0”“1N”“0”“1N”或“1N”“0”“1N”“0”，则 S 值被增加。

连续码字的定义有几种意思。在一个“用上下文”系统中，连续码字可以指在一上下文接收器中连续的码字。在一个“用概率”系统中，连续码字可以指在一个概率等级中的连续码字。可选地，在两者任一系统中，连续码字可以指整个连续码字（不考虑上下文接收器或概率等级）。对这三个实例，需要维护一个码字历史的存贮比特相应为 3 * 上下文接收器数量，3 * 概率等级数量和 3。维持每个上下文接收器的加速可以提供最佳自适应。因为由于未编码数据中一个整体改变经常导致差的跟踪，确定整体加速也可以提供良好的自适应。

任何压缩系统的一个优点是降低对数据集的存贮要求。本发明的并行系统可以替代当前被有无损编码系统完成的任何应用并可以被应用到运行声频，文本，数据库，计算机可执行或其它数字数据，信号或符号的系统中，实例的无损编码系统包括传真压缩，数据库压缩，位图图形图象压缩，和诸如 JPEG 和 MPEG 的图象压缩标准中的变换因数压缩。本发明允许小型有效硬件实现和相对快速的软件实现，使其甚至对于不要求高速的应用成为一个良好选择。

本发明较先前技术的真正优点是以非常高速概率，特别对解码的操作。在这个方式中，本发明可以充分利用昂贵的高速信道，例如高速计算机网络，卫星和地面广播信道。图 28 解释了这样的一个系统，其中广播数据或高速计算机网络提供数据给解码系统 2801，它并行解码数据以产生输出数据。当前硬件熵（例如 Q－编码器）将延缓这些系统的吞吐量。所有这些系统被以高成本设计高带宽。是计数器的生产性放慢了一个解码器的吞吐量。本发明

的并行系统不仅提供了这些高带宽，而且由于数据可以以压缩后的形式被传输而增加了有效带宽。

本发明的并行系统也可应用以获得现代快速通道如 ISDN, CD－ROM，和 SCSI 以外的更有效带宽。这样一个带宽区域系统如图 29 所示，其中来自源，例如一个 CD－ROM，以太网，小型计算机标准接口 (SCSI)，或其它相似源的数据，被连接到解码系统 2901，它接收并对数据解码以产生一个输出。这些通道仍然较某些当前编码器快。通常这些通道被用来服务于一个要求比通道本身更多的带宽的数据源，例如实时视像或基于计算机的多媒体。本发明的系统可以充当带宽匹配的角色。

本发明的系统对一个诸如高清晰度电视 (HDTV) 和 MPEG 视像标准的视像系统的一个熵编码器部分是一个极好的选择。这样的系统展示在图 30，参考图 30，实时视像系统包括被连接到经压缩图像数据的解码系统 3001，系统 3001 解码数据并输出其到有损解码器 3002，有损解码器 3002 可以是变换器，彩色转换器和一个 HDTV 或 MPEG 解码器的子抽样部分。监视器 3003 可以是一台电视或视频监视器。

鉴于当前发明的许多转义和修改对一位已经阅读了前面描述的普通技术人员将没有疑问，理解通过解释方法展示和描述的具体实施例不是被用于限制的。因此，对优选的实施例细节的参考不用于限制其本身就已陈述的权利要求，这些特性对发明本身重要。

因而，数据并行解码和编码的方法和装置已经被描述。

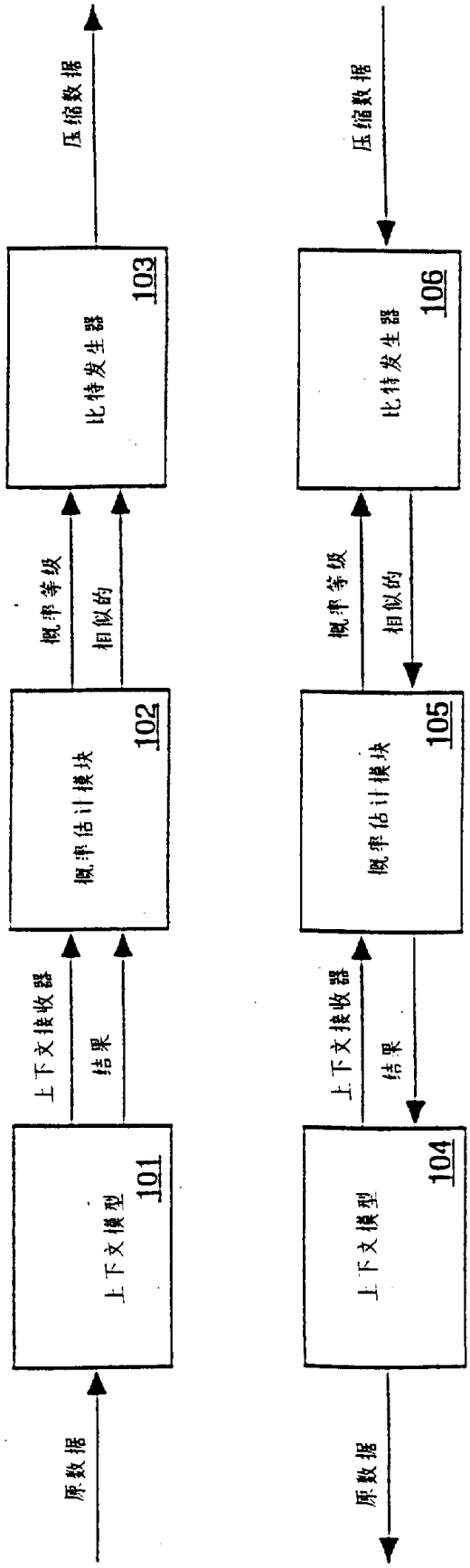

原数据
上下文模型
101
上下文接收器
结果
概率估计模块
102
概率等级
相似的
比特发生器
103
压缩数据
压缩数据
比特发生器
106
概率等级
相似的
概率估计模块
105
上下文接收器
结果
上下文模型
104
原数据

图1

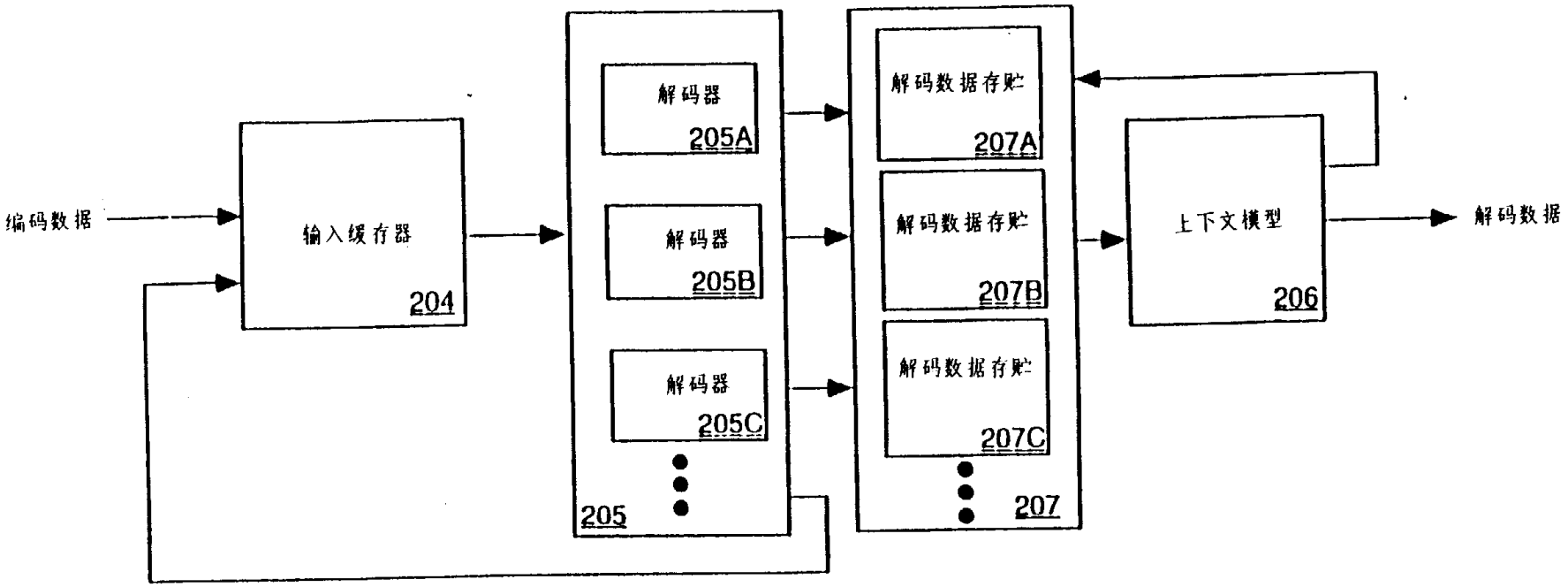

编码数据
输入缓存器
204
解码器
205A
解码器
205B
解码器
205C
205
解码数据存贮
207A
解码数据存贮
207B
解码数据存贮
207C
207
上下文模型
206
解码数据

图 2A

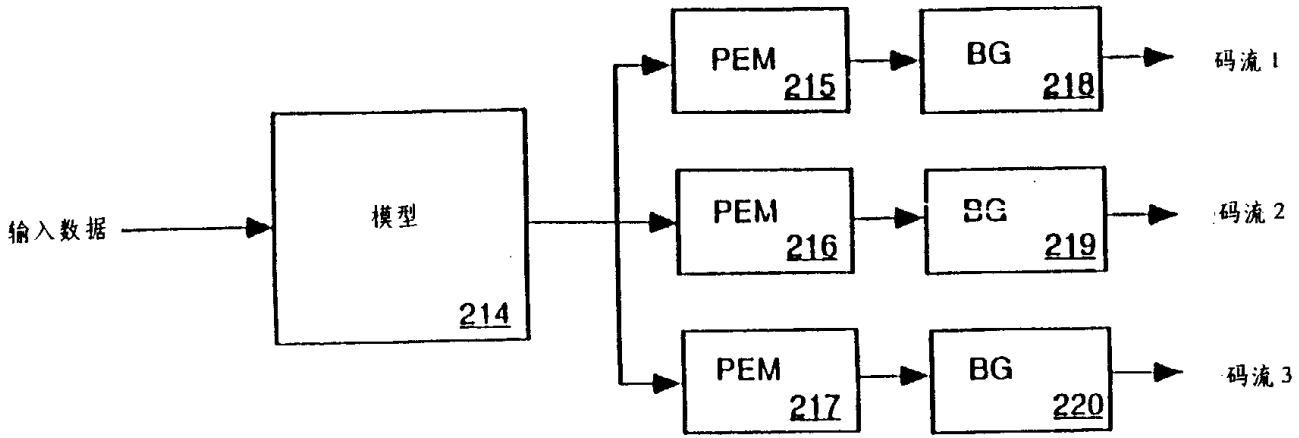

输入数据
模型
214
PEM
215
BG
218
码流 1
PEM
216
BG
219
码流 2
PEM
217
BG
220
码流 3

图 2B

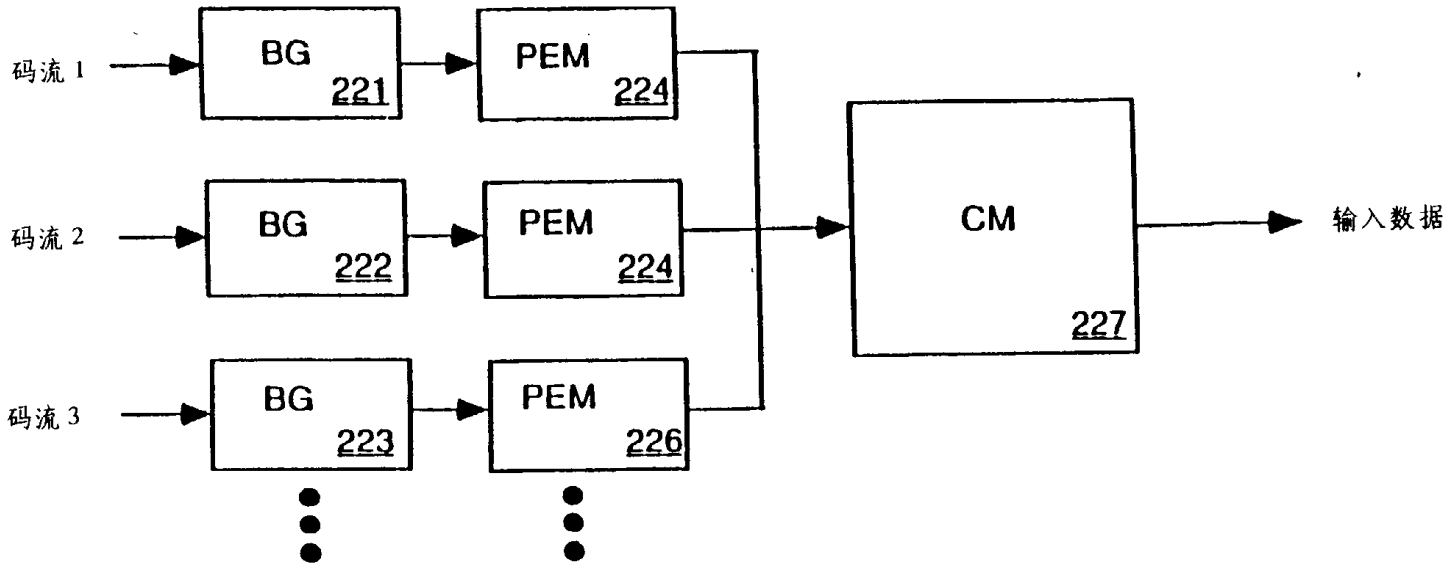

码流 1
BG
221
PEM
224
码流 2
BG
222
PEM
224
CM
227
输入数据
码流 3
BG
223
PEM
226

图 2C

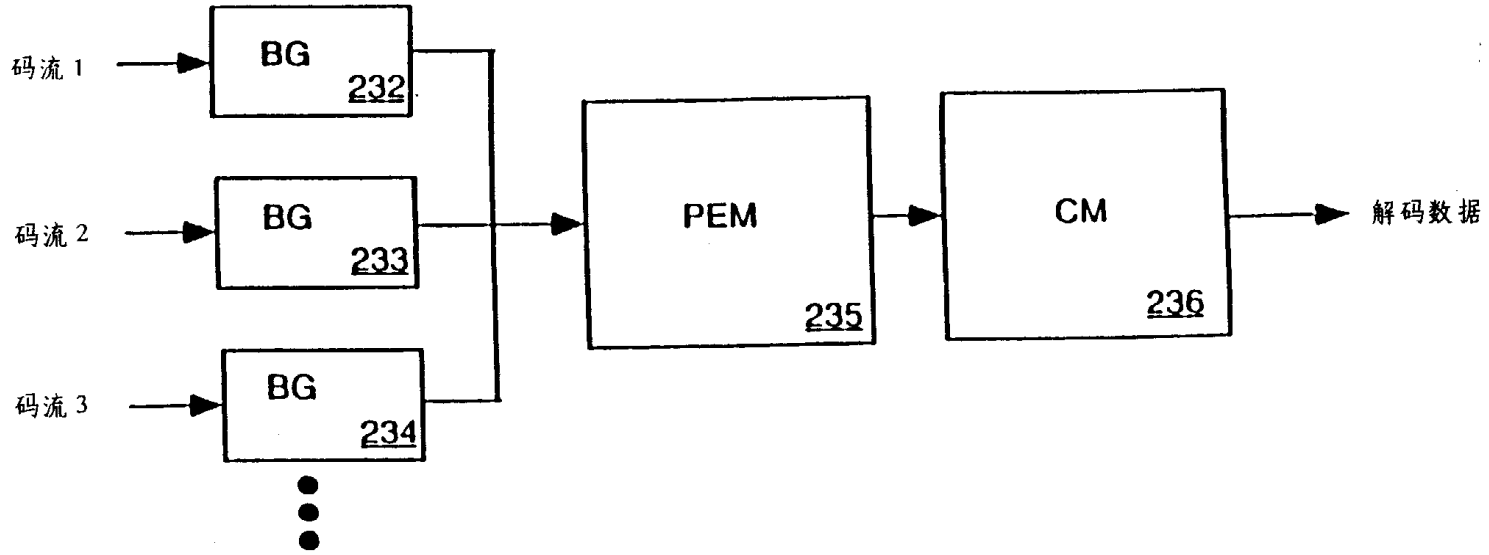

码流 1
BG
232
码流 2
BG
233
PEM
235
CM
236
解码数据
码流 3
BG
234

图 2D

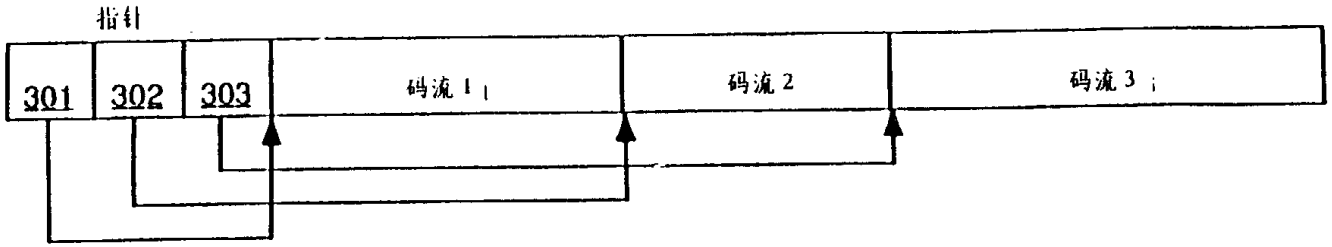

指针
301
302
303
码流 1
码流 2
码流 3

图3

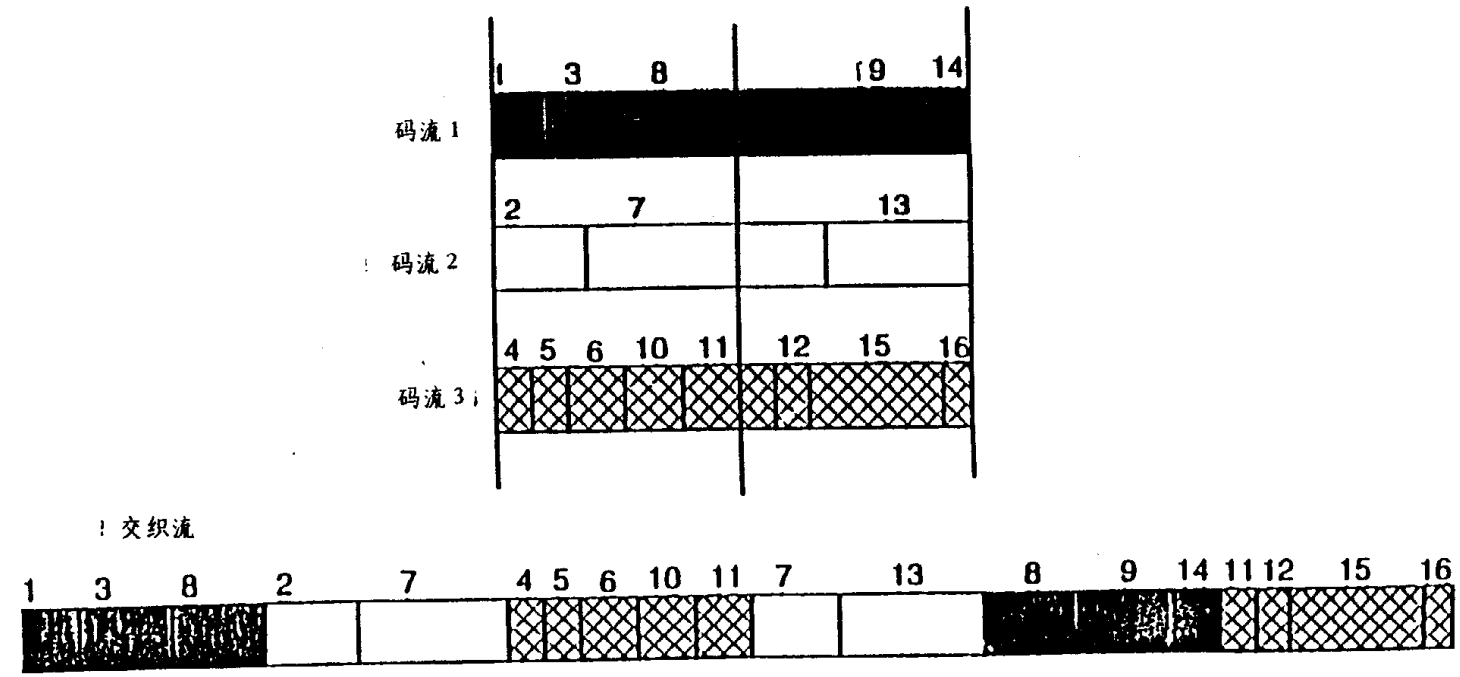

1 3 8
9 14
码流 1
2 7
13
码流 2
4 5 6 10 11
12 15 16
码流 3
交织流
1 3 8 2 7 4 5 6 10 11 7 13 8 9 14 11 12 15 16

图4

状态	码
-35	R2(12)
-34	R3(11)
-33	R2(11)
-32	R3(10)
-31	R2(10)
-30	R3(9)
-29	R2(9)
-28	R3(8)
-27	R2(8)
-26	R3(7)
-25	R2(7)
-24	R3(6)
-23	R2(6)
-22	R3(5)
-21	R2(5)
-20	R3(4)
-19	R2(4)
-18	R3(3)
-17	R2(3)
-16	R3(2)
-15	R2(2)
-14	R3(1)
-13	R3(1)
-12	R3(1)

状态	码
-11	R2(1)
-10	R2(1)
-9	R2(1)
-8	R2(1)
-7	R2(1)
-6	R2(1)
-5	R2(0)
-4	R2(0)
-3	R2(0)
-2	R2(0)
-1	R2(0)
-0	R2(0)
0	R2(0)
1	R2(0)
2	R2(0)
3	R2(0)
4	R2(0)
5	R2(0)
6	R2(1)
7	R2(1)
8	R2(1)
9	R2(1)
10	R2(1)
11	R2(1)

状态	码
12	R3(1)
13	R2(1)
14	R3(1)
15	R2(2)
16	R3(2)
17	R2(3)
18	R3(3)
19	R2(4)
20	R3(4)
21	R2(5)
22	R3(5)
23	R2(6)
24	R3(6)
25	R2(7)
26	R3(7)
27	R2(8)
28	R3(8)
29	R2(9)
30	R3(9)
31	R2(10)
32	R3(10)
33	R2(11)
34	R3(11)
35	R2(12)

图5

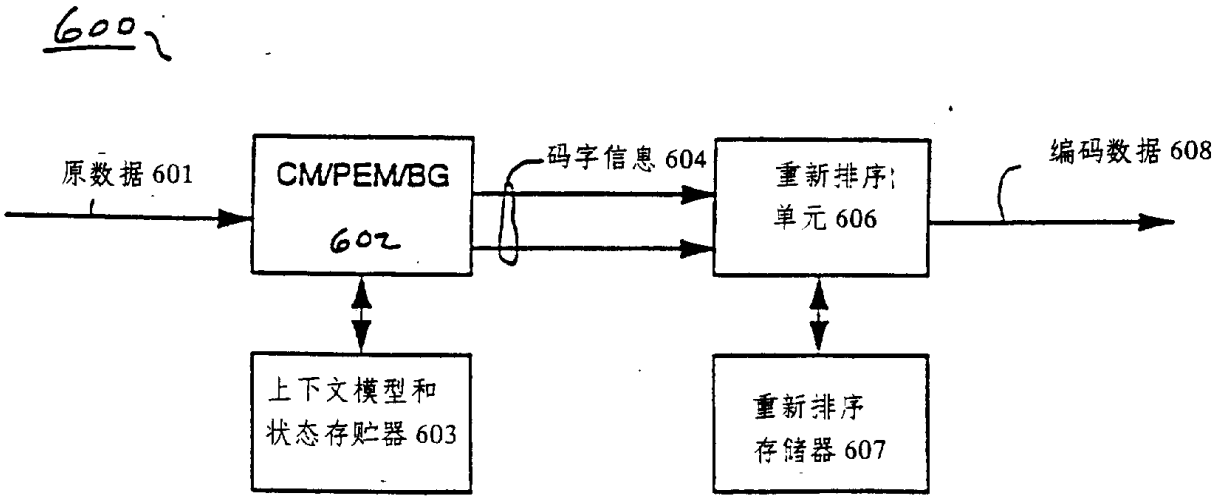

600
原数据 601
CM/PEM/BG
602
码字信息 604
重新排序
单元 606
编码数据 608
上下文模型和
状态存贮器 603
重新排序
存储器 607

图6

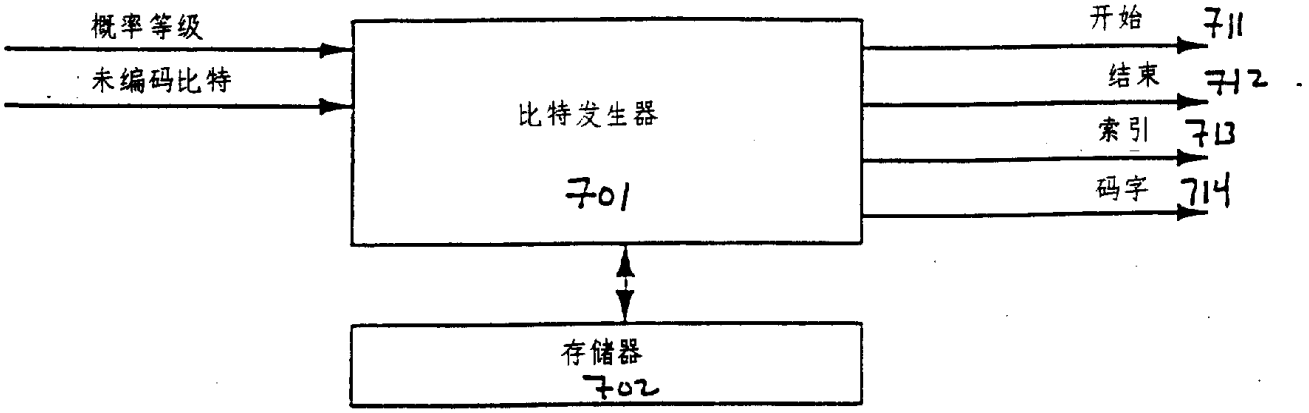


图7

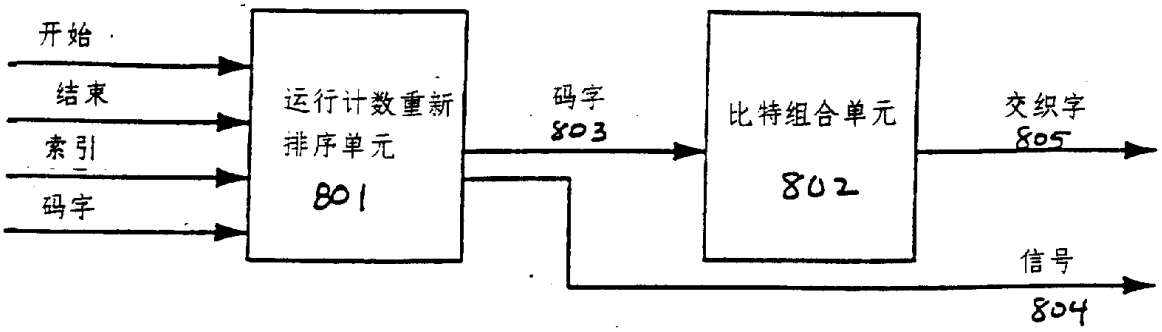


图8

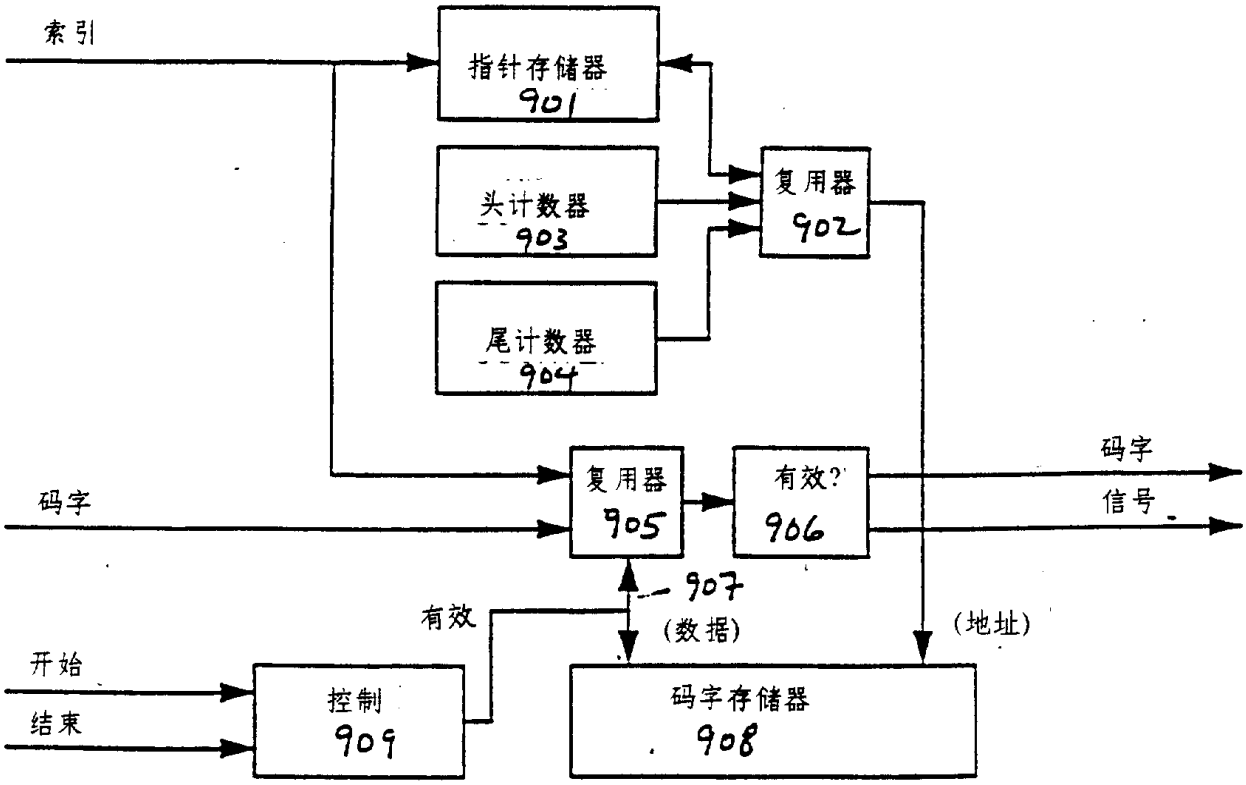

索引
指针存储器
901
头计数器
903
尾计数器
904
复用器
902
复用器
905
有效?
906
码字
码字
信号
907
有效
(数据)
(地址)
开始
结束
控制
909
码字存储器
908

图9

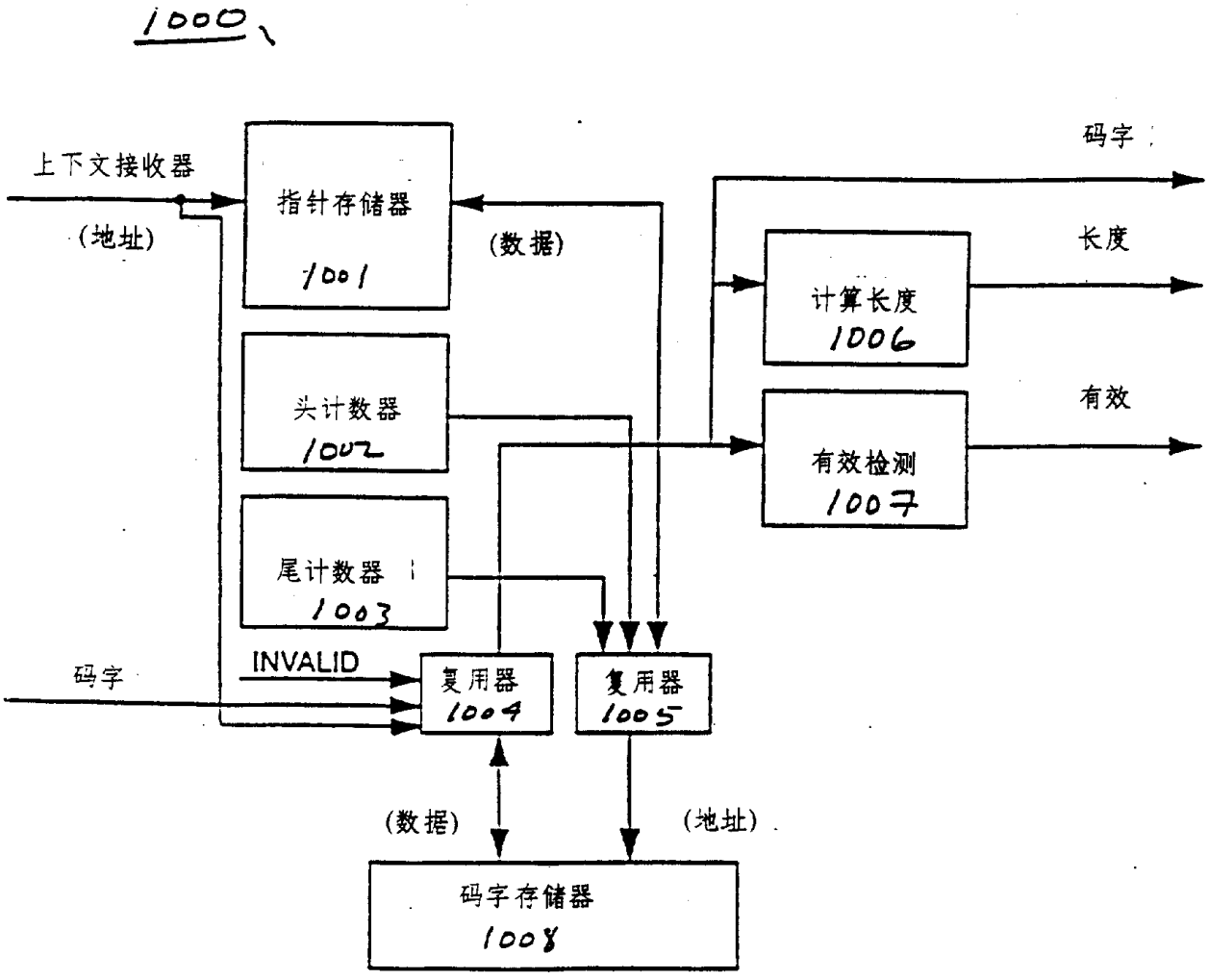

1000
上下文接收器
(地址)
指针存储器
1001
(数据)
码字
长度
计算长度
1006
头计数器
1002
有效
有效检测
1007
尾计数器
1003
INVALID
码字
复用器
1004
复用器
1005
(数据)
(地址)
码字存储器
1008

图10

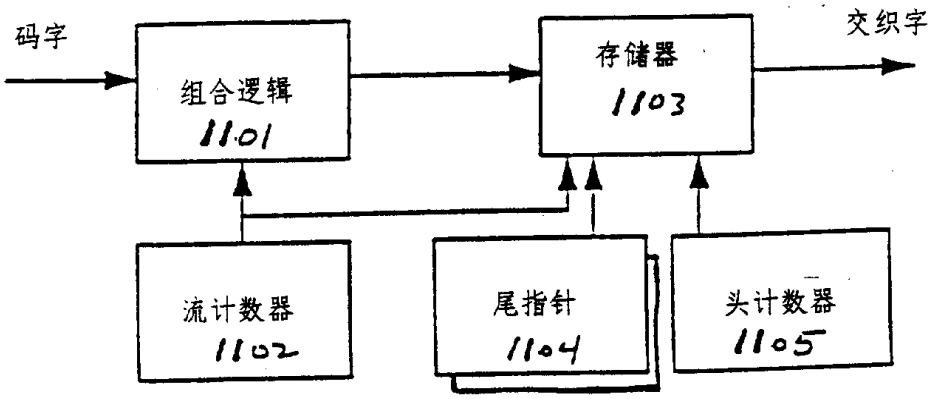


图11

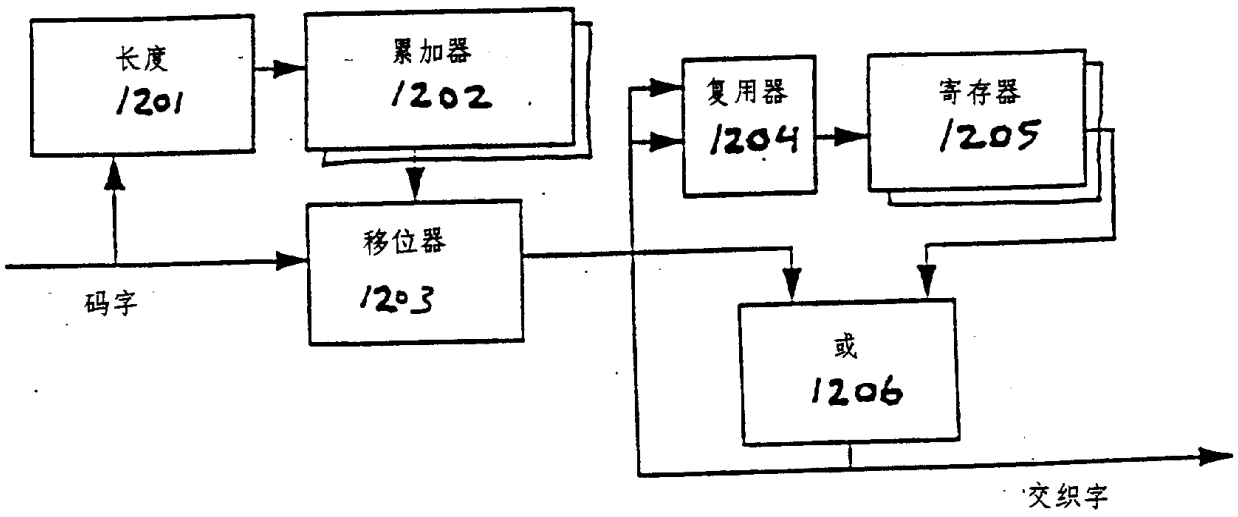


图12

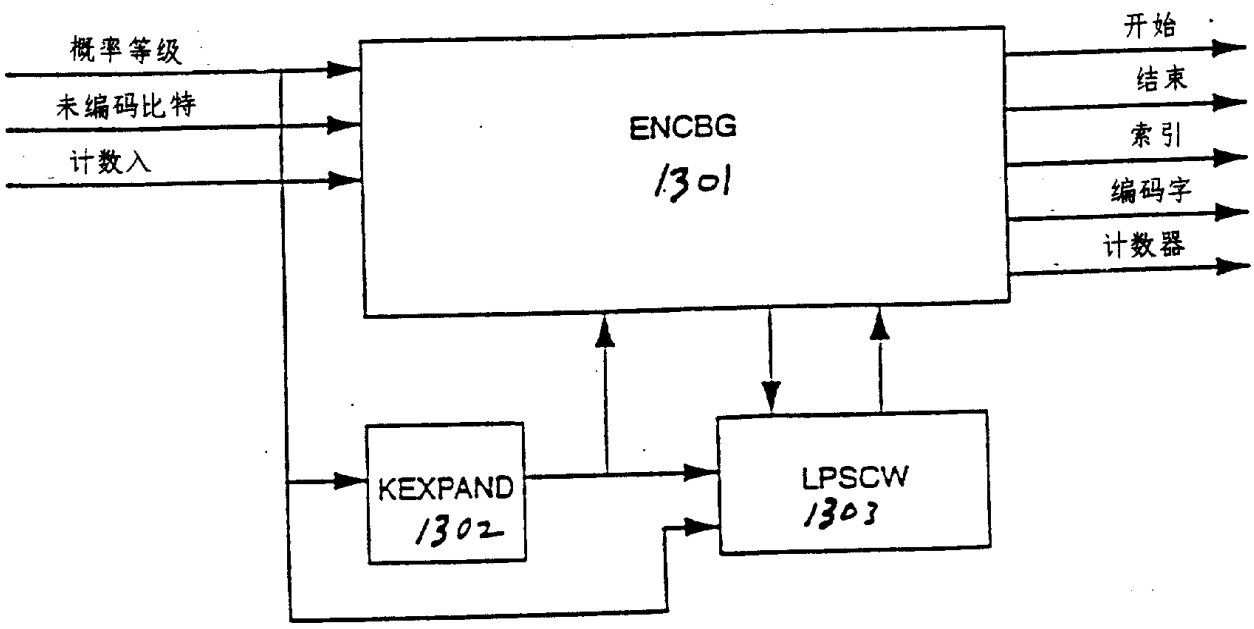

概率等级
未编码比特
计数入
ENCBG
1301
开始
结束
索引
编码字
计数器
KEXPAND
1302
LPSCW
1303

图13

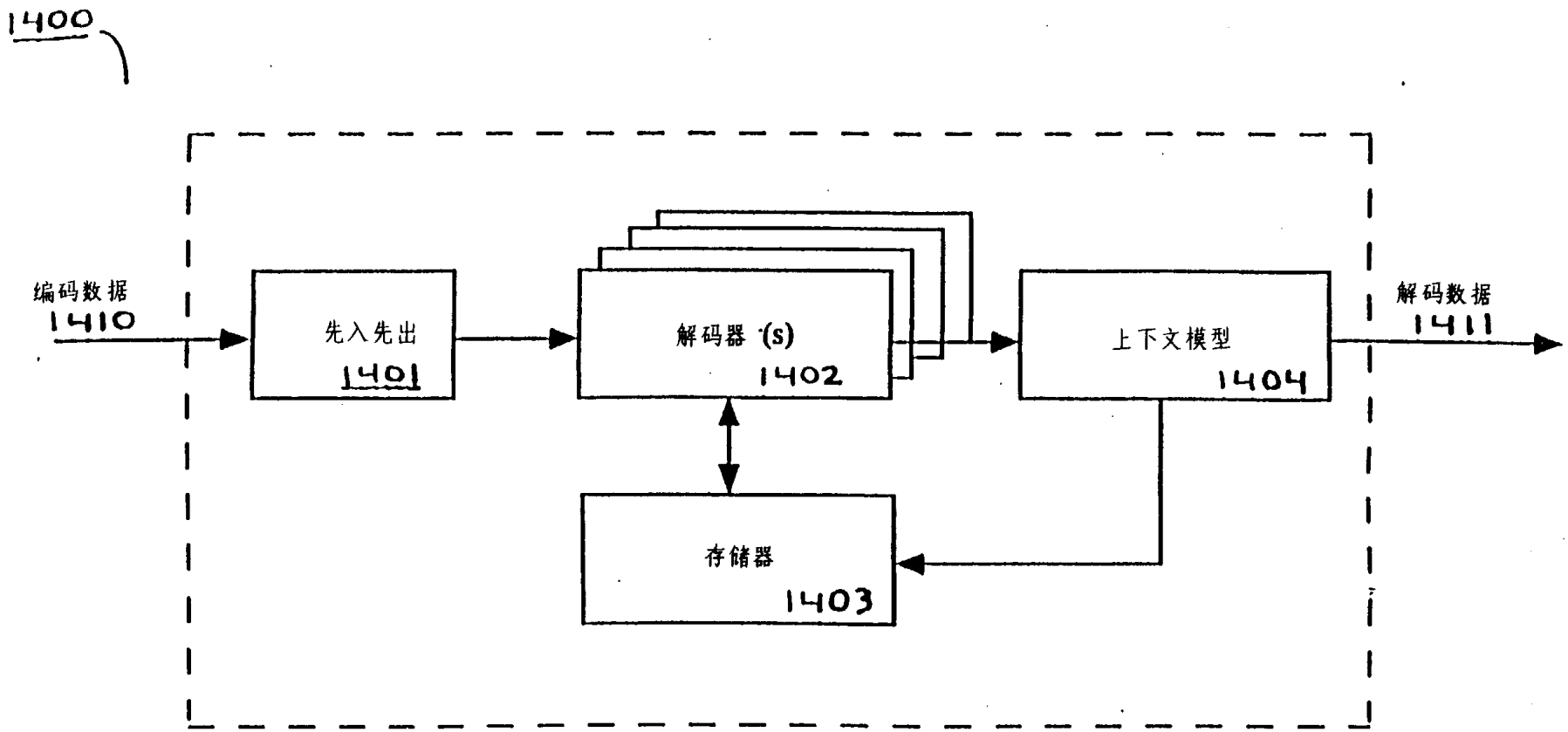

1400
编码数据
1410
先入先出
1401
解码器 (s)
1402
上下文模型
1404
解码数据
1411
存储器
1403

图 14A

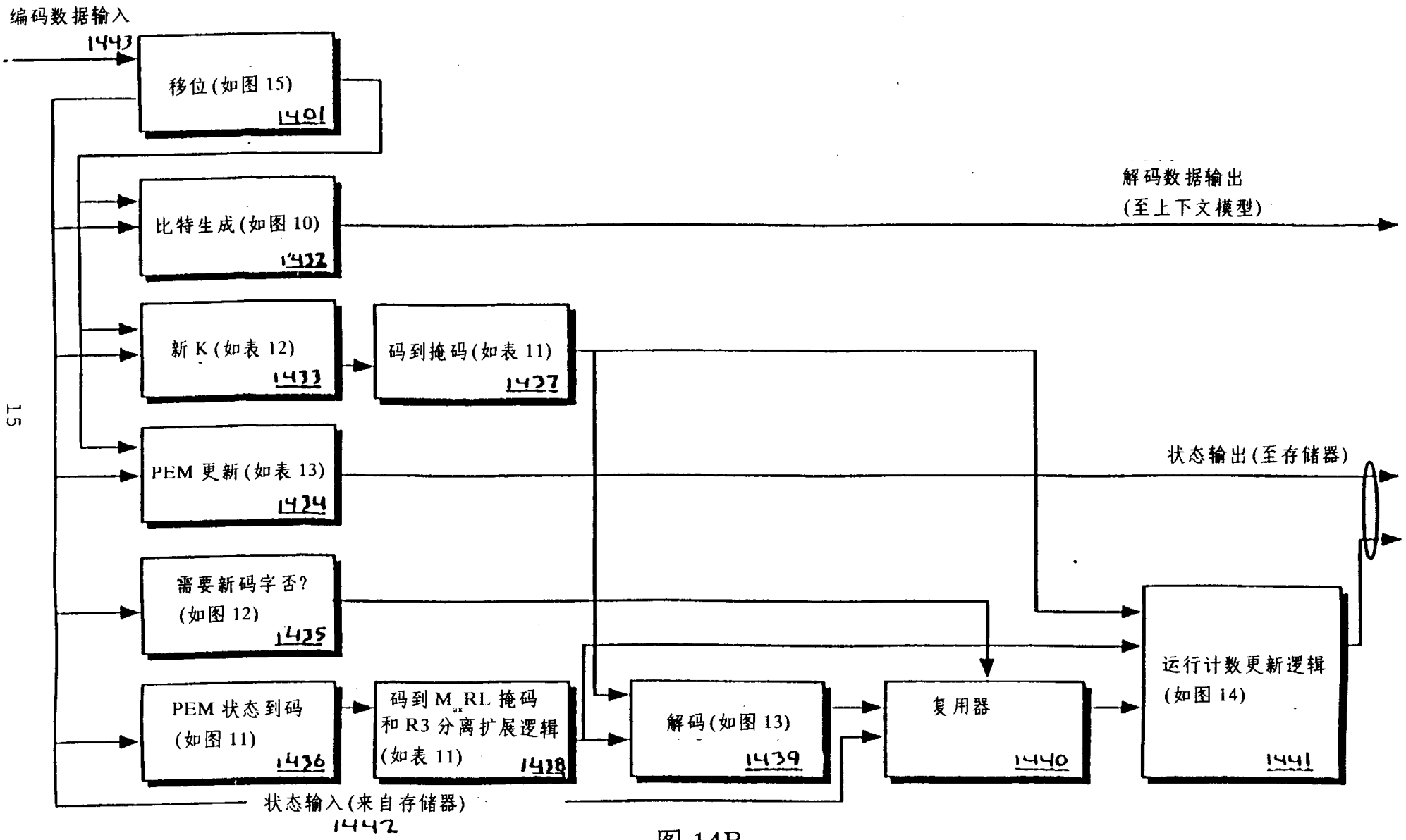

编码数据输入
1443
移位(如图 15)
1401
比特生成(如图 10)
1422
解码数据输出
(至上下文模型)
新 K(如表 12)
1433
码到掩码(如表 11)
1427
PEM 更新(如表 13)
1424
状态输出(至存储器)
需要新码字否?
(如图 12)
1425
运行计数更新逻辑
(如图 14)
1441
PEM 状态到码
(如图 11)
1426
码到 $M_{ax}RL$ 掩码
和 R3 分离扩展逻辑
(如表 11)
1418
解码(如图 13)
1439
复用器
1440
状态输入(来自存储器)
1442

图 14B

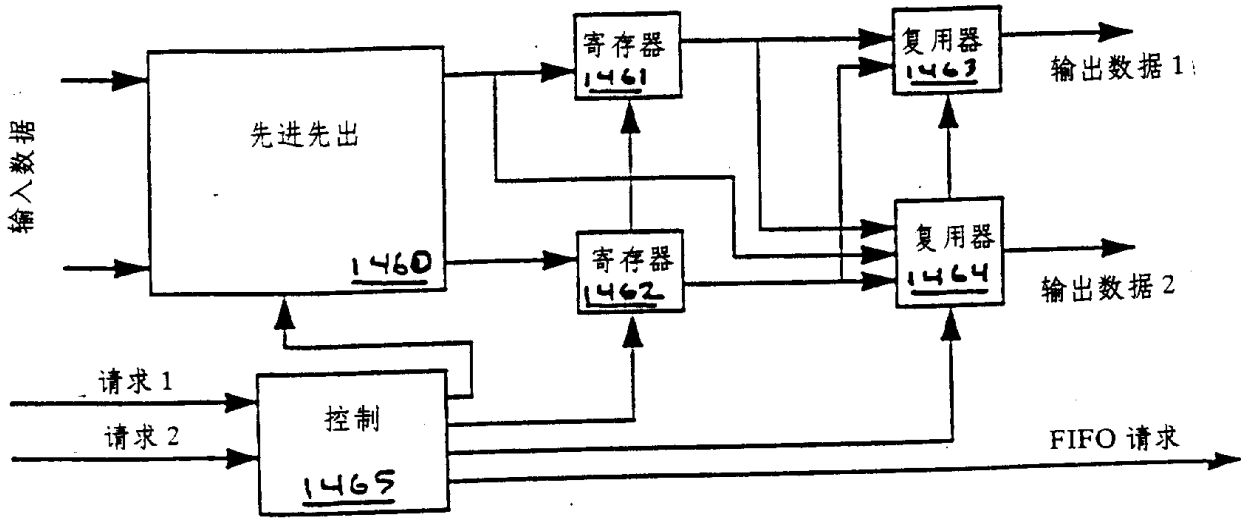

输入数据
先进先出
1460
寄存器
1461
寄存器
1462
复用器
1463
复用器
1464
输出数据 1
输出数据 2
请求 1
请求 2
控制
1465
FIFO 请求

图 14C

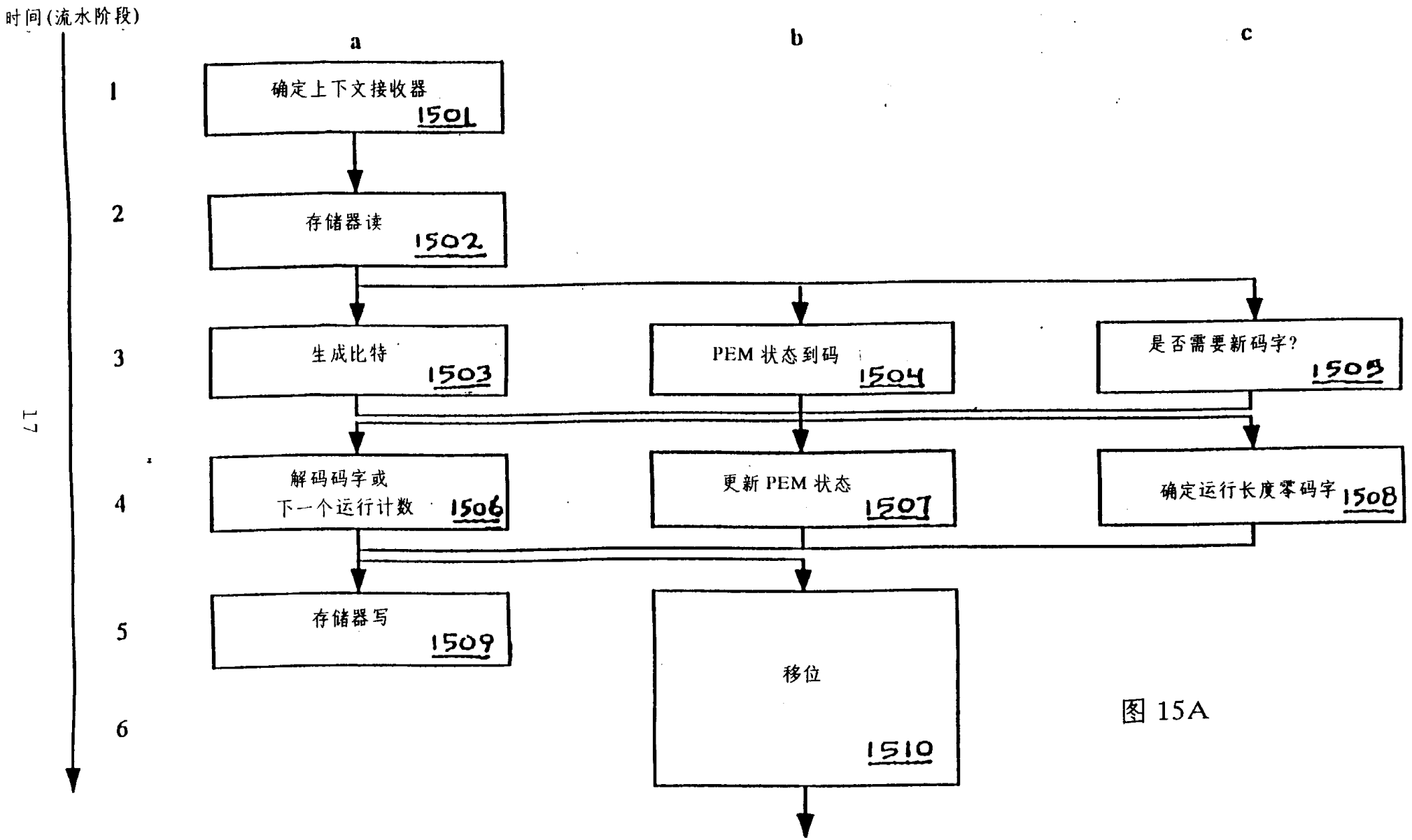

时间(流水阶段)
a
b
c
1
2
3
4
5
6
确定上下文接收器
1501
存储器读
1502
生成比特
1503
PEM 状态到码
1504
是否需要新码字?
1505
解码码字或
下一个运行计数
1506
更新 PEM 状态
1507
确定运行长度零码字
1508
存储器写
1509
移位
1510

图 15A

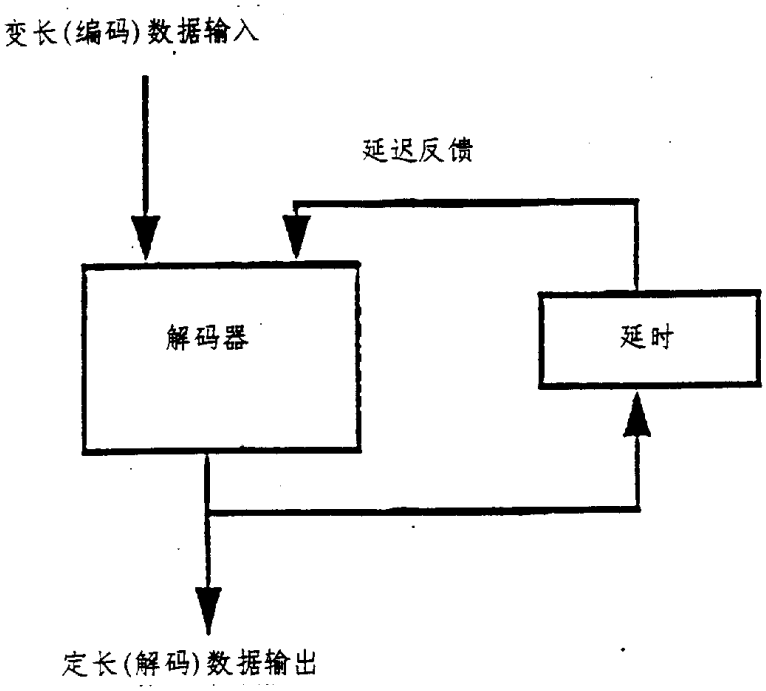


图 15B

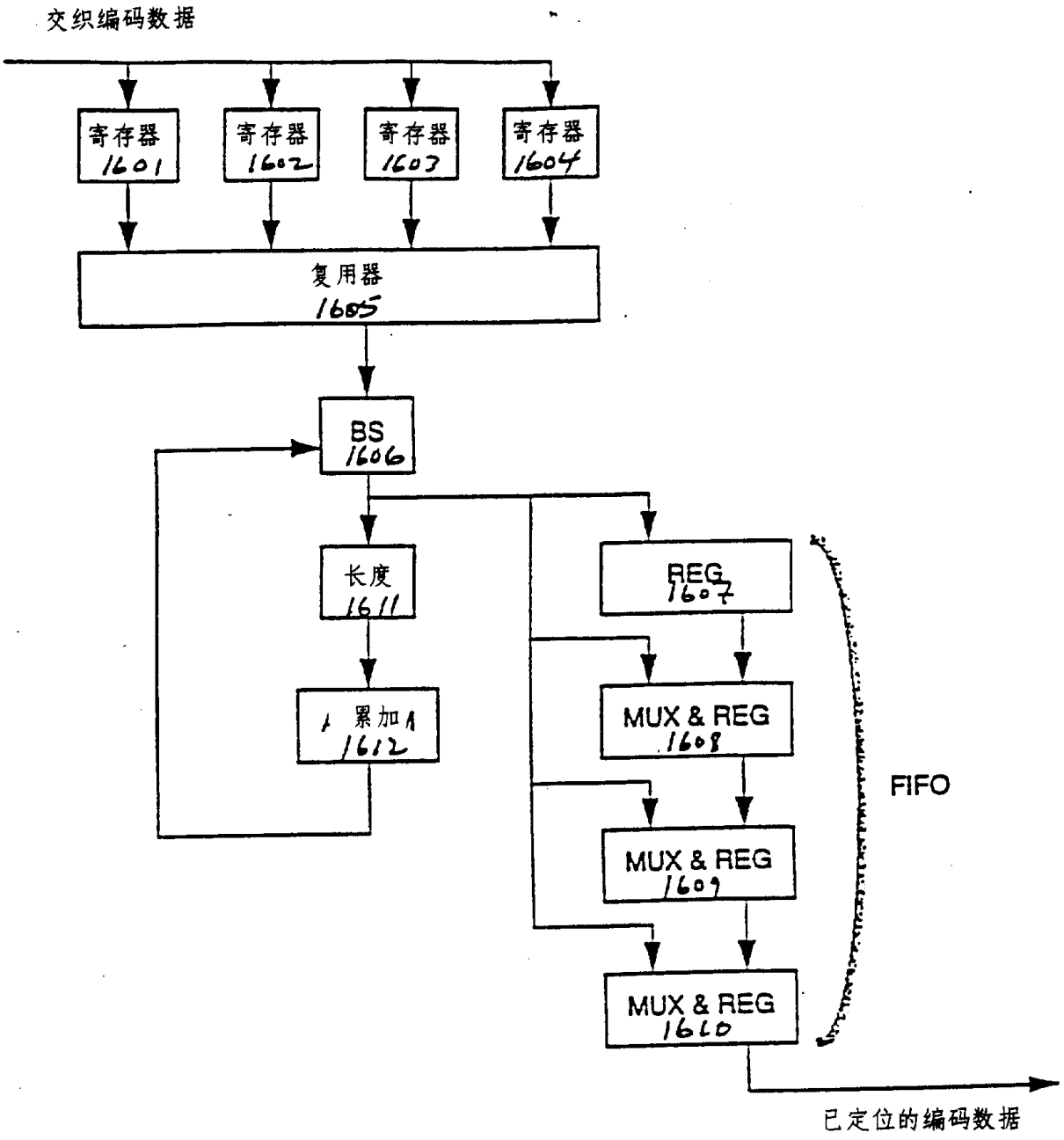

交织编码数据
寄存器 1601
寄存器 1602
寄存器 1603
寄存器 1604
复用器 1605
BS 1606
长度 1611
累加 1612
REG 1607
MUX & REG 1608
MUX & REG 1609
MUX & REG 1610
FIFO
已定位的编码数据

图 16A

1620

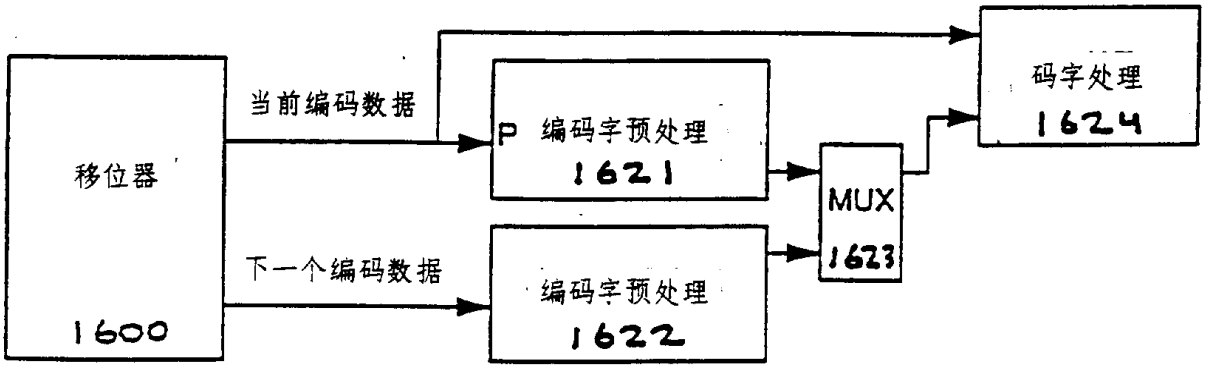

移位器
1600
当前编码数据
下一个编码数据
P 编码字预处理
1621
编码字预处理
1622
MUX
1623
码字处理
1624

图 16B

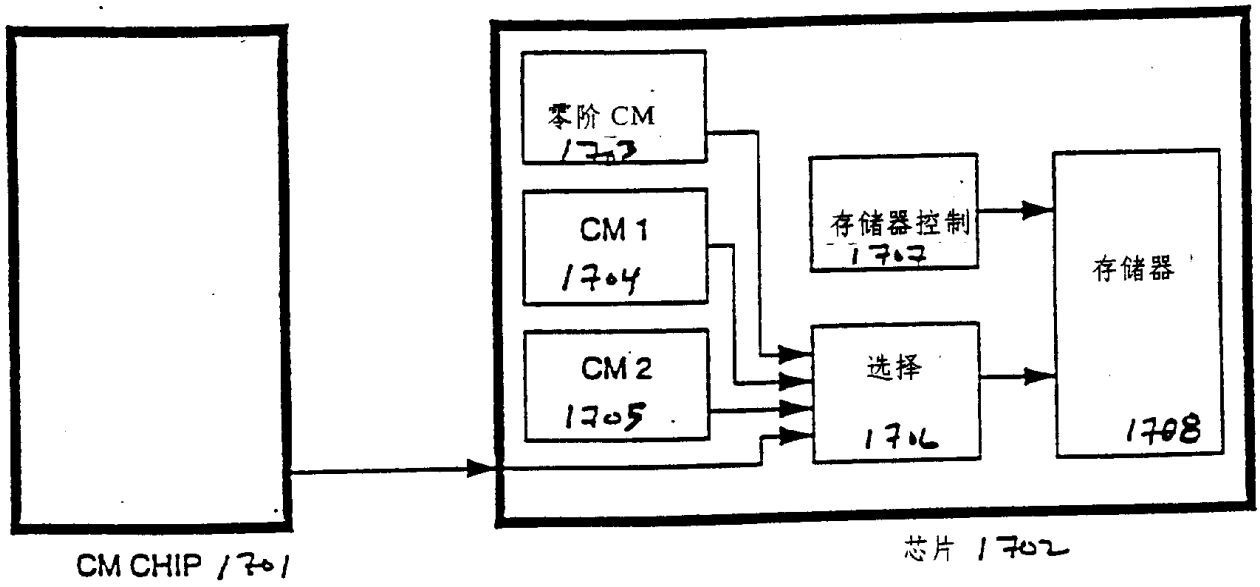

零阶 CM
1703
CM 1
1704
CM 2
1705
存储器控制
1707
选择
1706
存储器
1708
CM CHIP 1701
芯片 1702

图17

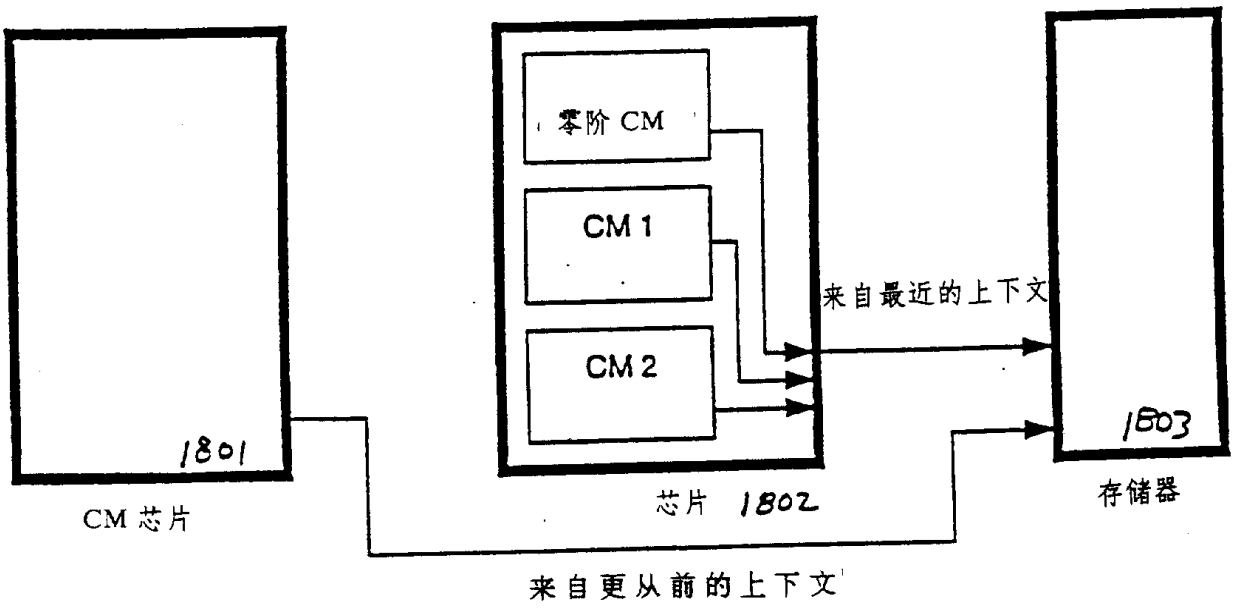


图18

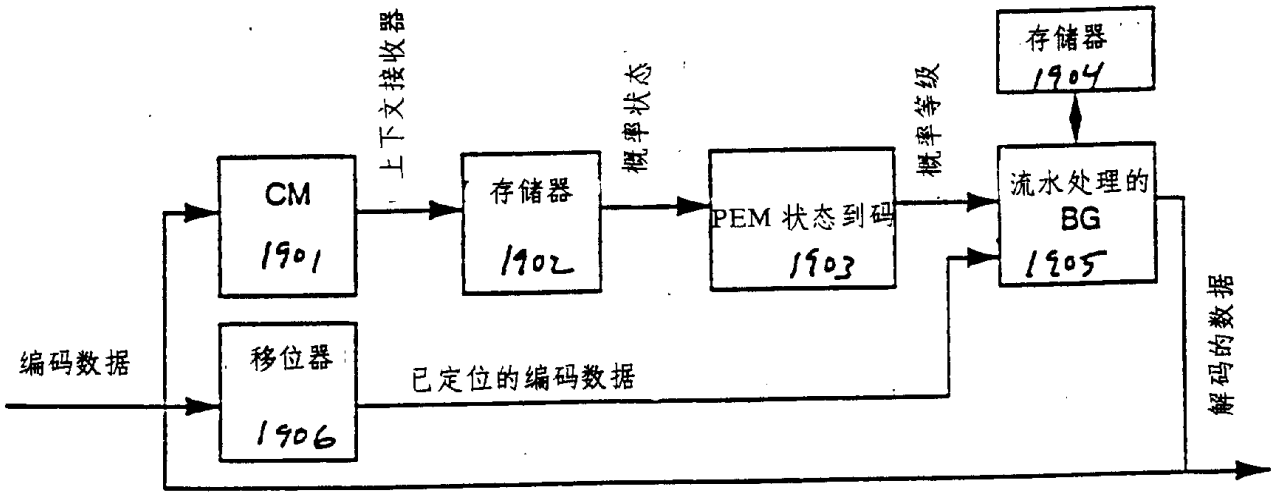

编码数据
CM
1901
移位器
1906
上下文接收器
存储器
1902
概率状态
PEM 状态到码
1903
概率等级
存储器
1904
流水处理的
BG
1905
已定位的编码数据
解码的数据

图19

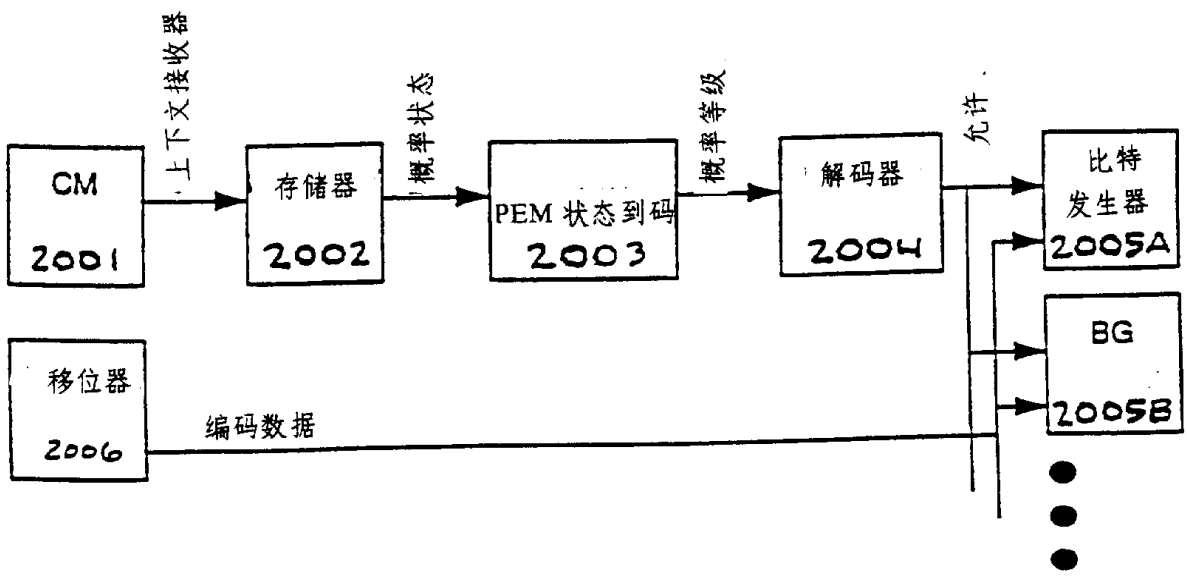

CM
2001
上下文接收器
存储器
2002
概率状态
PEM 状态到码
2003
概率等级
解码器
2001
允许
比特
发生器
2005A
BG
2005B
移位器
2006
编码数据

图20

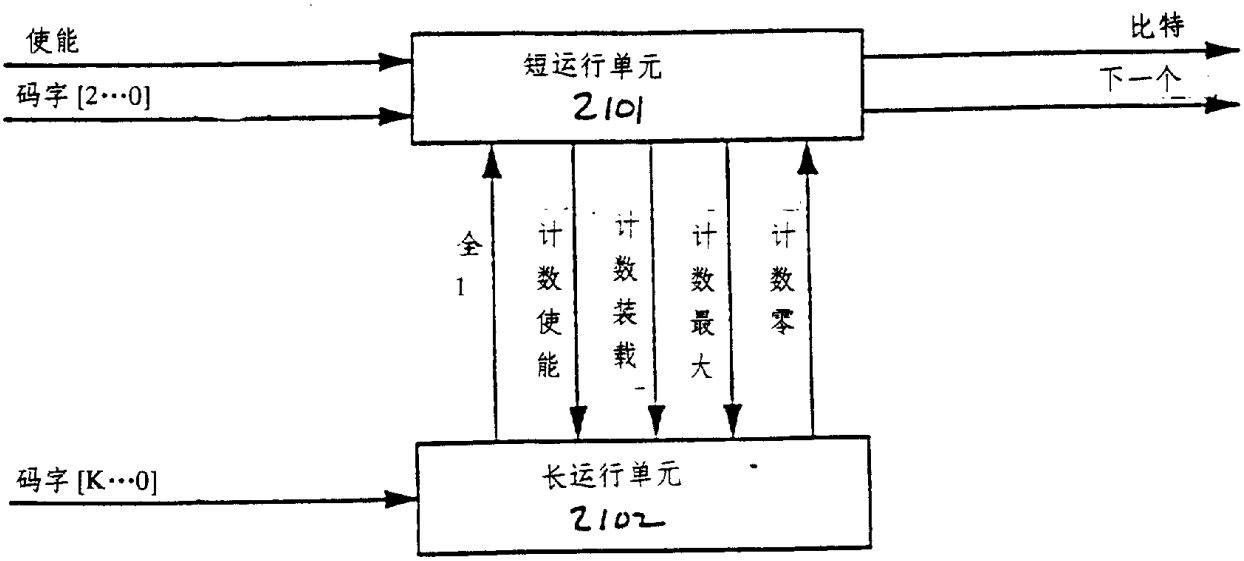

使能
码字 [2…0]
短运行单元
2101
比特
下一个
全1
计数使能
计数装载
计数最大
计数零
码字 [K…0]
长运行单元
2102

图21

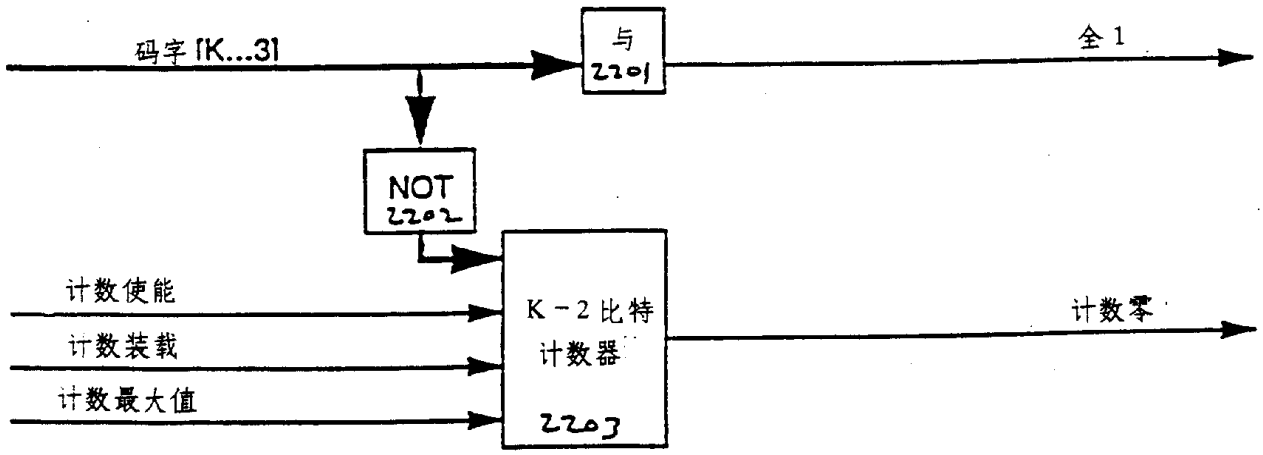


图22

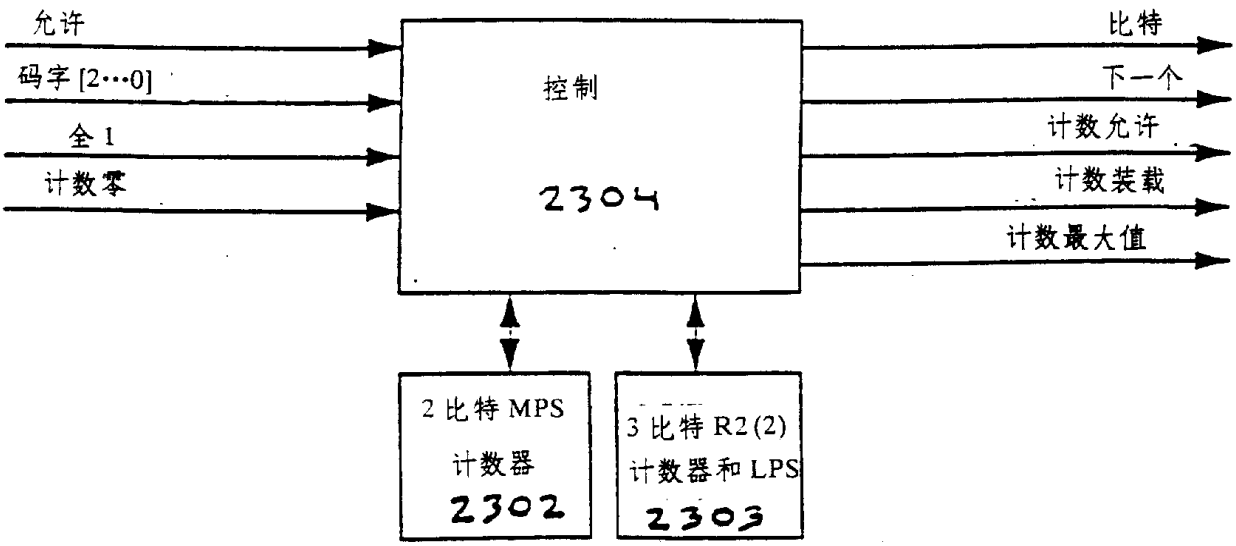

允许
码字 [2…0]
全 1
计数零
控制
2304
比特
下一个
计数允许
计数装载
计数最大值
2 比特 MPS
计数器
2302
3 比特 R2(2)
计数器和 LPS
2303

图23

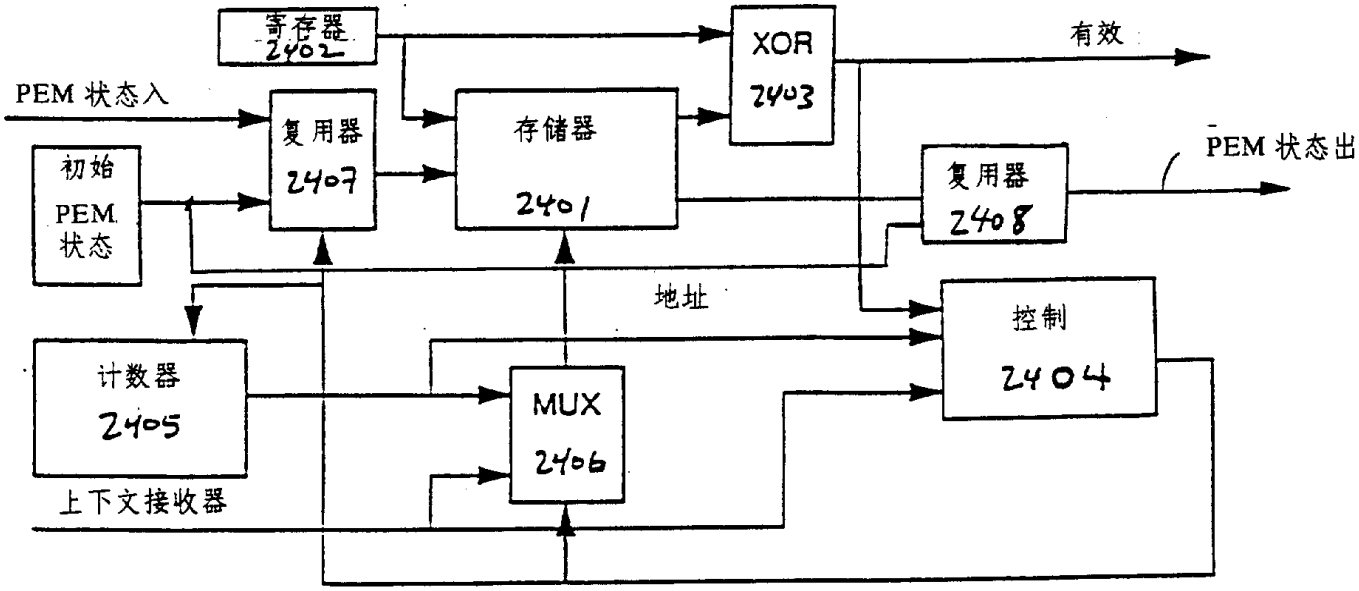

寄存器
2402
XOR
2403
有效
PEM 状态入
复用器
2407
存储器
2401
PEM 状态出
初始
PEM
状态
复用器
2408
地址
控制
2404
计数器
2405
MUX
2406
上下文接收器

图24

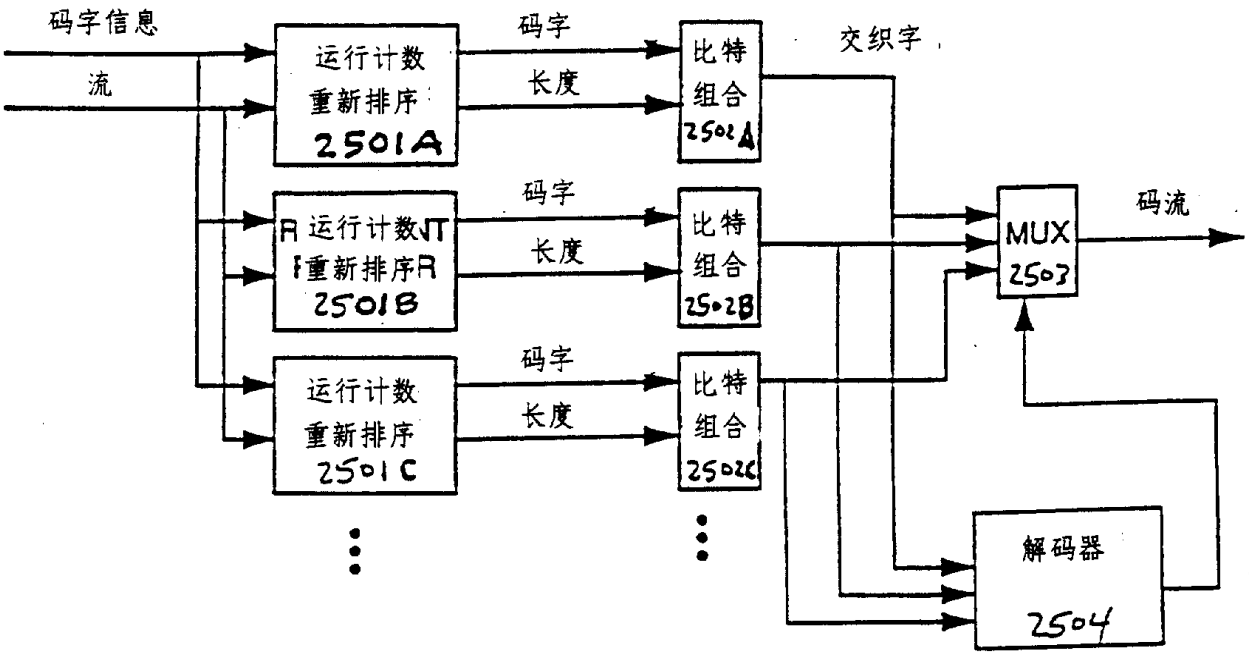

码字信息
流
运行计数
重新排序
2501A
码字
长度
比特
组合
2502A
交织字
运行计数
重新排序
2501B
码字
长度
比特
组合
2502B
MUX
2503
码流
运行计数
重新排序
2501C
码字
长度
比特
组合
2502C
解码器
2504

图25

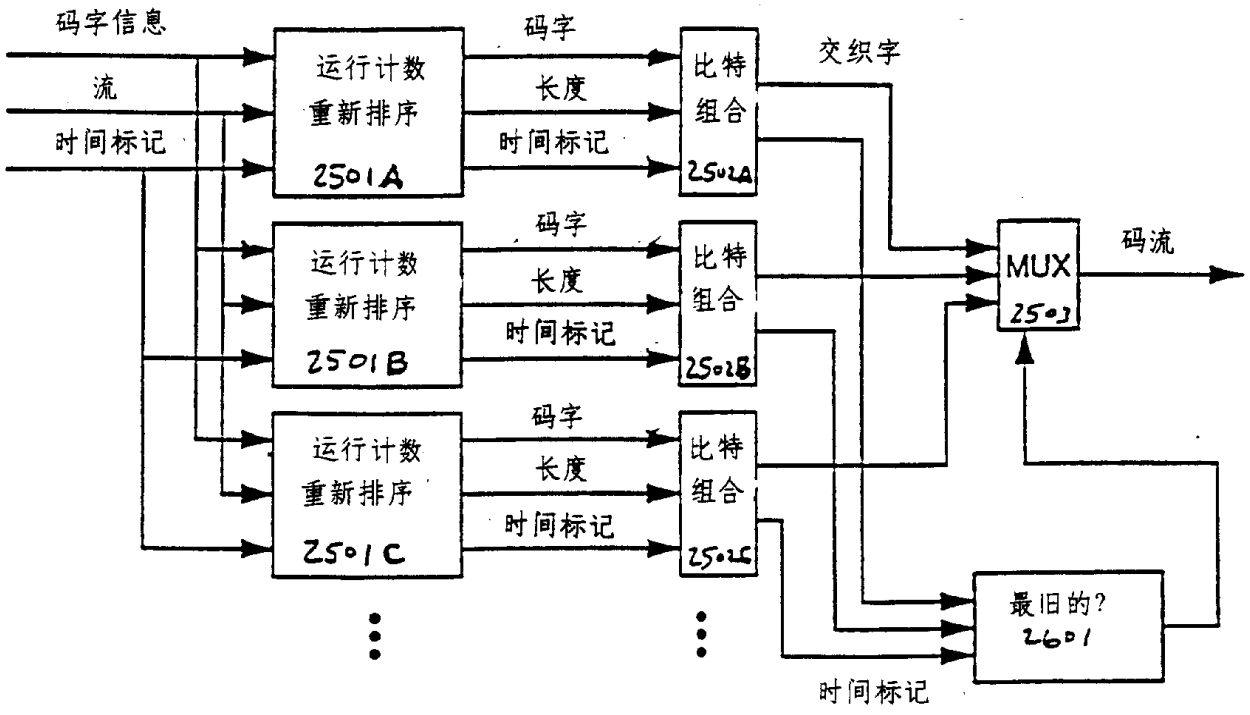

码字信息
流
时间标记
运行计数
重新排序
2501A
码字
长度
时间标记
比特
组合
2502A
交织字
运行计数
重新排序
2501B
码字
长度
时间标记
比特
组合
2502B
MUX
2503
码流
运行计数
重新排序
2501C
码字
长度
时间标记
比特
组合
2502C
最旧的?
2601
时间标记

图26

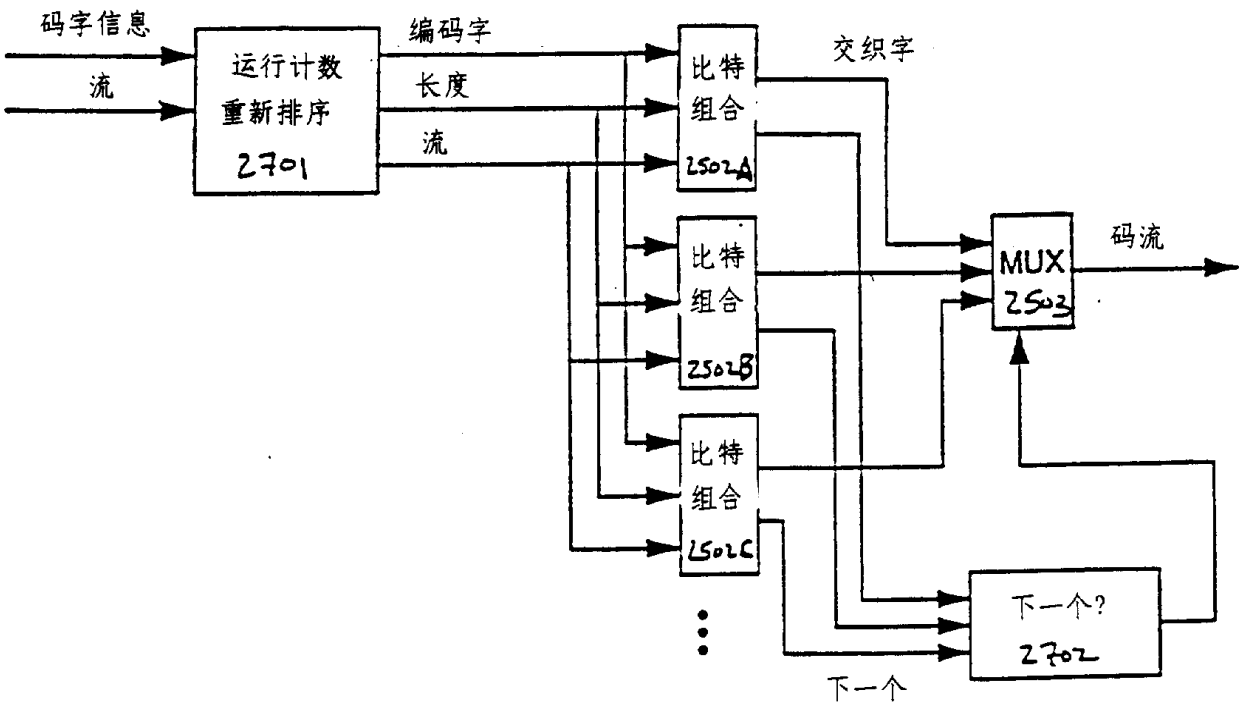

码字信息
流
运行计数
重新排序
2701
编码字
长度
流
比特
组合
2502A
比特
组合
2502B
比特
组合
2502C
交织字
MUX
2503
码流
下一个?
2702
下一个

图27

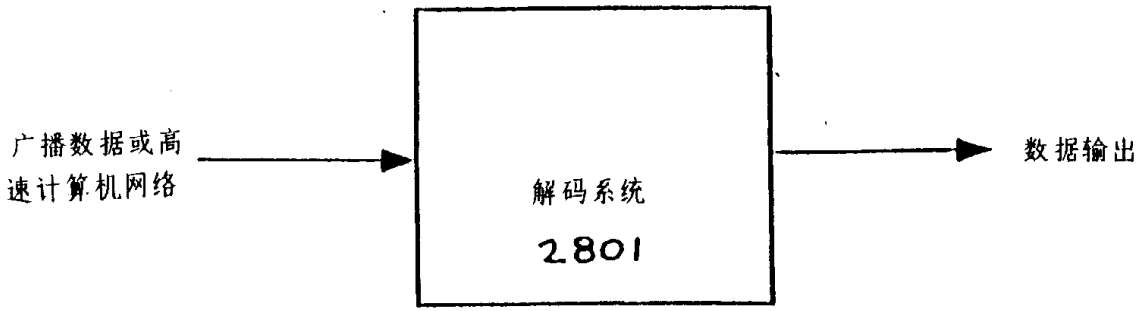

广播数据或高
速计算机网络
解码系统
2801
数据输出

图28

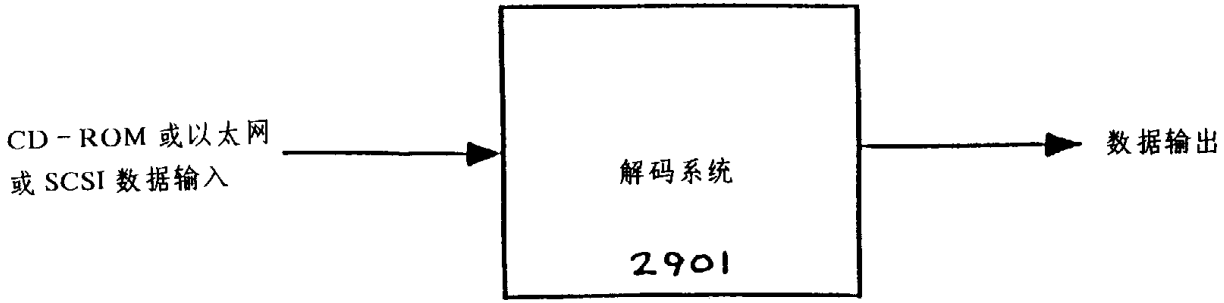

CD－ROM 或以太网
或 SCSI 数据输入
解码系统
2901
数据输出

图29

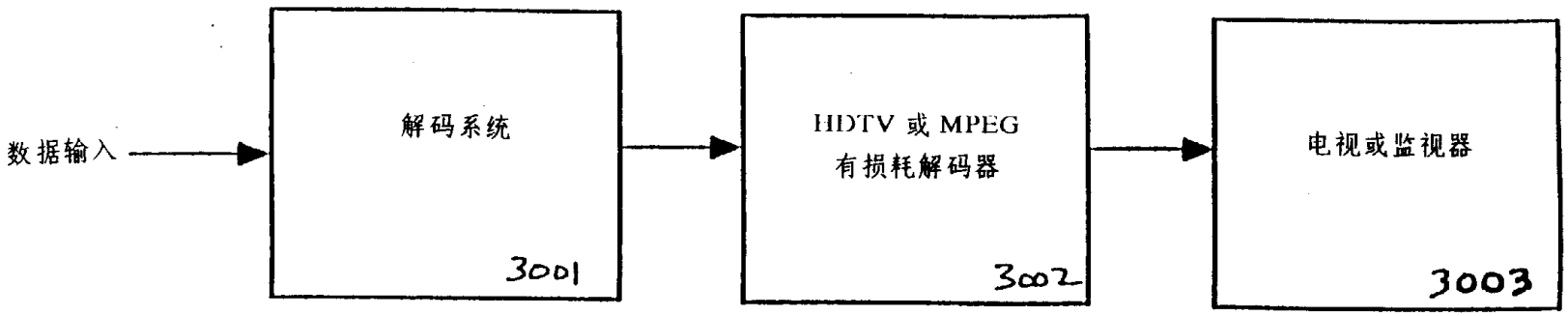


图30

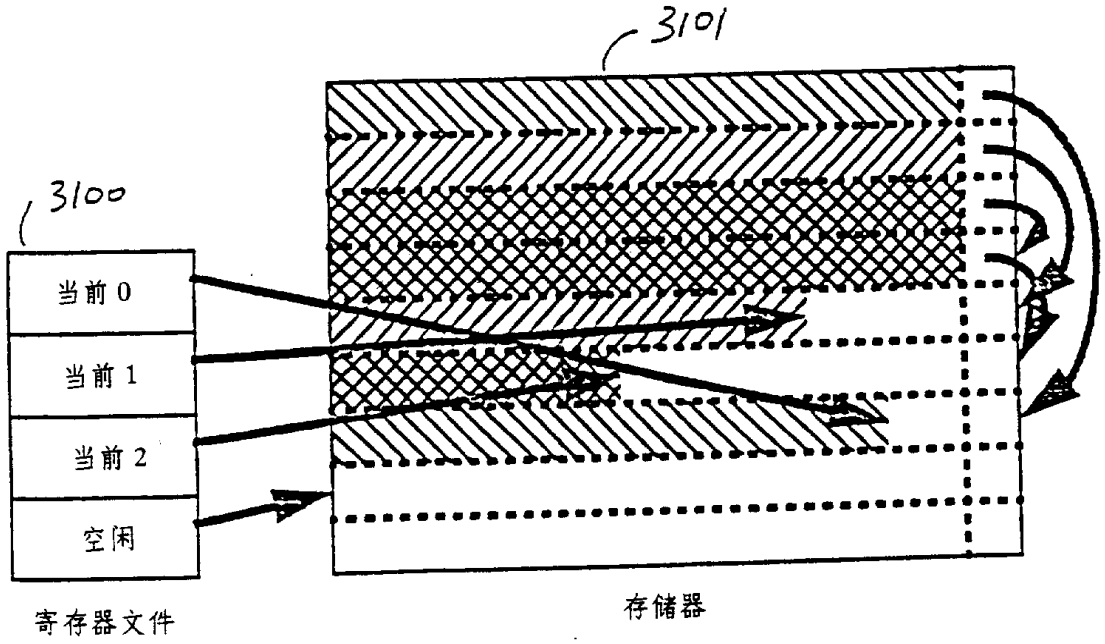


图31

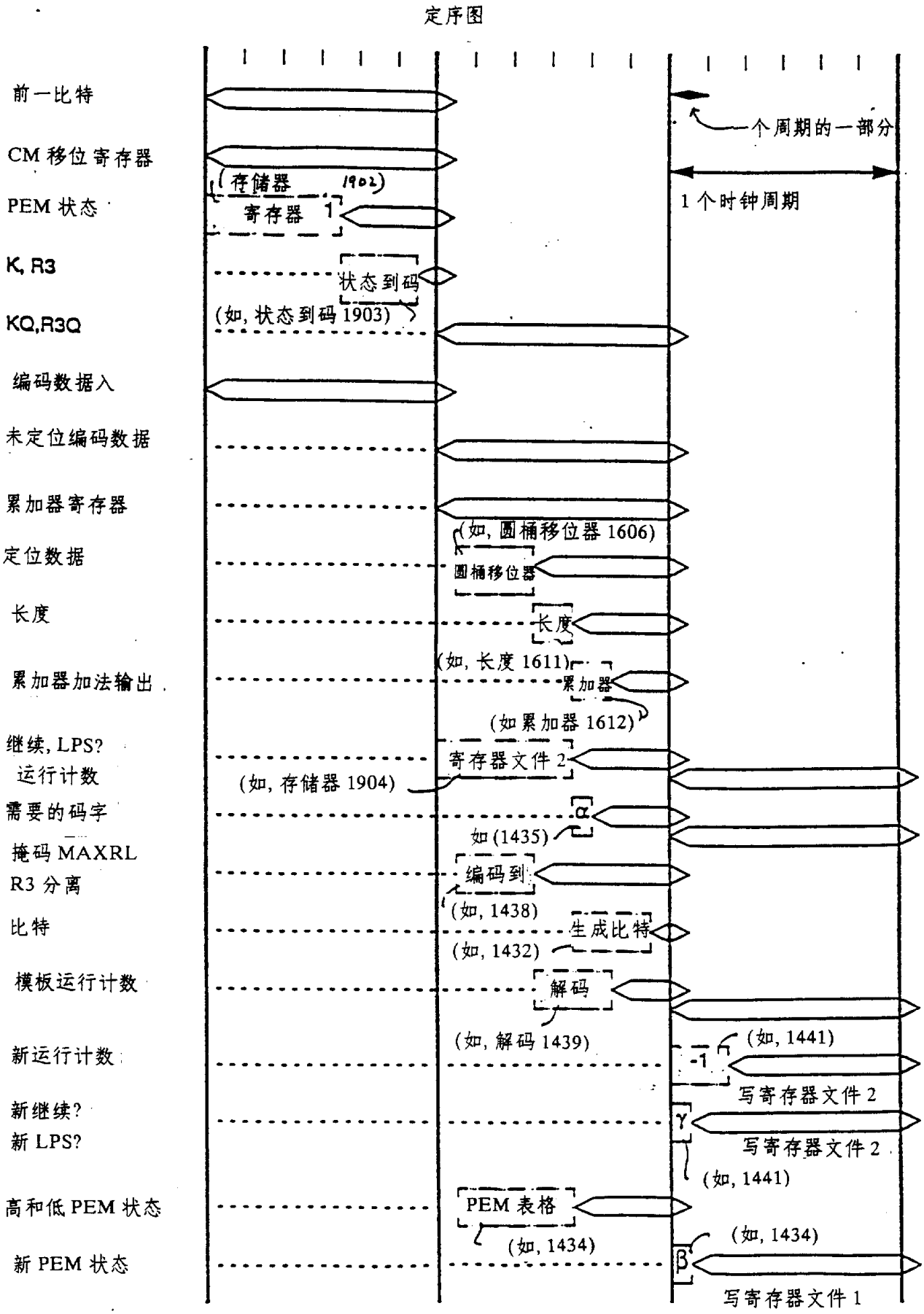

定序图
前一比特
CM 移位寄存器
PEM 状态
K, R3
KQ,R3Q
编码数据入
未定位编码数据
累加器寄存器
定位数据
长度
累加器加法输出
继续, LPS?
运行计数
需要的码字
掩码 MAXRL
R3 分离
比特
模板运行计数
新运行计数
新继续?
新 LPS?
高和低 PEM 状态
新 PEM 状态
一个周期的一部分
1 个时钟周期
(存储器 1902)
寄存器 1
状态到码
(如, 状态到码 1903)
(如, 圆桶移位器 1606)
圆桶移位器
长度
(如, 长度 1611)
累加器
(如累加器 1612)
寄存器文件 2
(如, 存储器 1904)
α
如 (1435)
编码到
(如, 1438)
生成比特
(如, 1432)
解码
(如, 解码 1439)
-1
(如, 1441)
写寄存器文件 2
γ
写寄存器文件 2
(如, 1441)
PEM 表格
(如, 1434)
(如, 1434)
β
写寄存器文件 1

图32

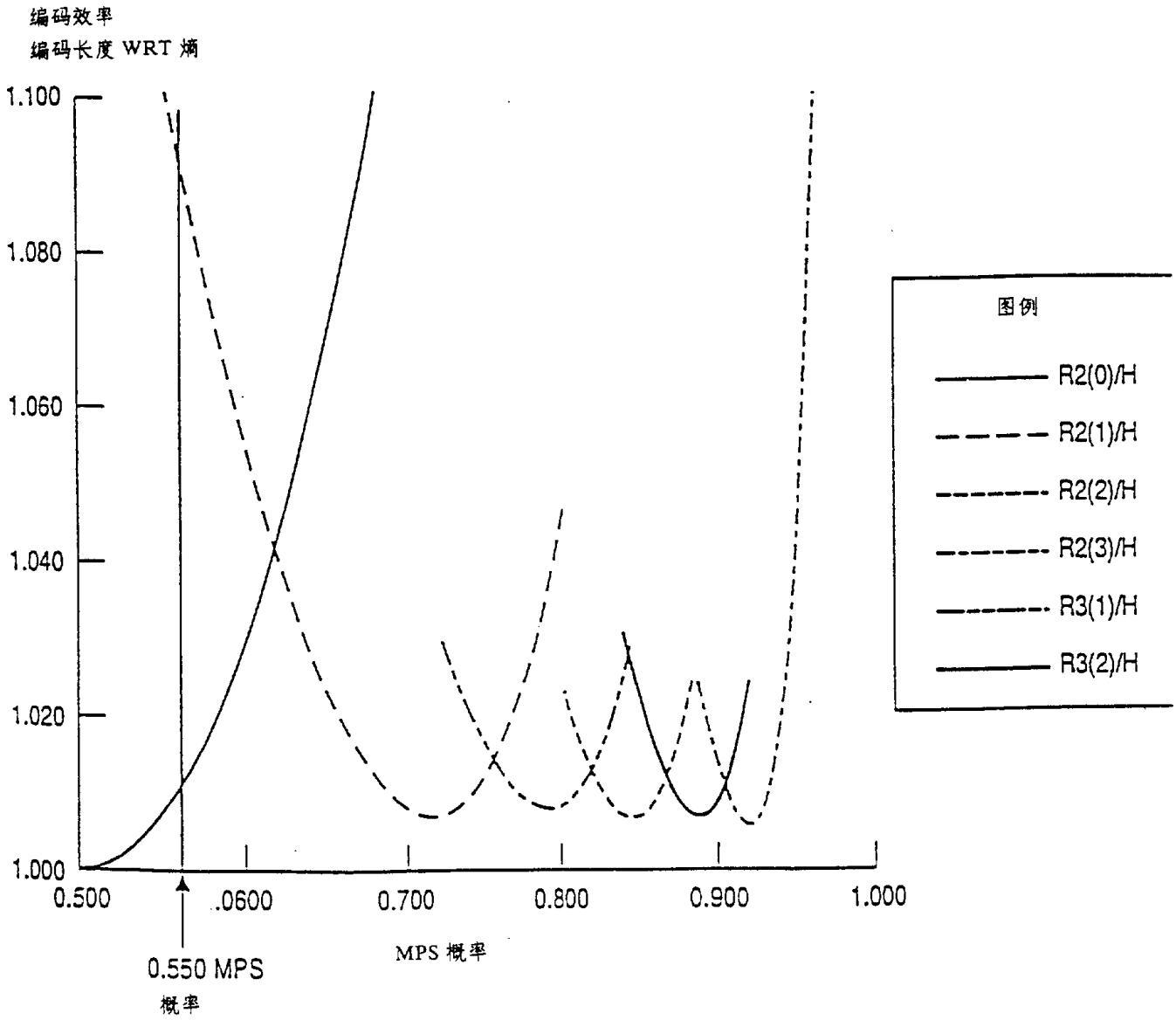

编码效率
编码长度 WRT 熵
1.100
1.080
1.060
1.040
1.020
1.000
0.500
.0600
0.700
0.800
0.900
1.000
0.550 MPS
概率
MPS 概率
图例
R2(0)/H
R2(1)/H
R2(2)/H
R2(3)/H
R3(1)/H
R3(2)/H

图33