大规模波形数据存储与检索系统

压缩引擎编码格式文档

合同编号：44030120210001901

编写人员：向隆、尹沛骐

编写日期：2021.07.15

目录

1 引言

1.1 编写的目的与适用范围

本文档为第一级压缩引擎编码格式的说明文档，详细描述了第一级压缩引擎目前设计的改进的编码格式和相应压缩率的简单推导。

本文档的阅读对象是本项目相关的管理人员、FPGA 开发人员、测试人员等。

2 第一级压缩引擎编码方案

第一级压缩引擎的输入数据为用户从 FPGA 板上抓波形得到的由数据帧组成的数据流，每一个数据帧包含一个 256bits 的帧头和帧尾，在帧头和帧尾中间包含的是 1024 字节（32*256bits）的采样数据，根据采样数据宽度（模式）的不同一个数据帧中包含了 32 至 1 个时刻的采样值。对于绝大多数信号在相邻时刻的采样值中只会有少量的信号数值发生变化，第一级压缩引擎的基本设计思想是使用**行程编码（run-length encoding）**和**增量编码（delta coding）**只保留信号相邻时刻采样值中变化的值，将未发生变化的数值从结果中剔除。这样既保留了信号变化的信息，又排除了数据帧格式中冗余的信号值，为第二级压缩引擎提供了较好的数据基础。但是只保留变化的信号值也会带来一个问题，在编码变化信号时需要对发生变化信号的位置同时进行编码，所以对于变化的数据而言其编码长度增加，当相邻时刻的信号值发生大量变化时编码所需的码字长度将会发生膨胀甚至超过原始数据的长度。为尽可能减少因数据变化过多造成的编码长度的膨胀我们设计了如下的编码方案。

2.1 编码方案

第一级压缩引擎对波形数据的编码由以下两部分构成：header、**压缩编码的采样数据**。

2.1.1 header

header 中编码了本次采样的元数据信息：channel number(采样通道编号)、model（采样模式）、start timestamp（起始采样时间戳）。

channel number（采样通道编号）

- 数据范围：0~11
- 使用空间：1Byte
- 编码格式：0000_0000 ~ 0000_0101

model（采样模式）

- 数据范围：0~5
- 使用空间：1Byte

- 编码格式：0000_0000 ~ 0000_0011

start timestamp（起始采样时间戳）

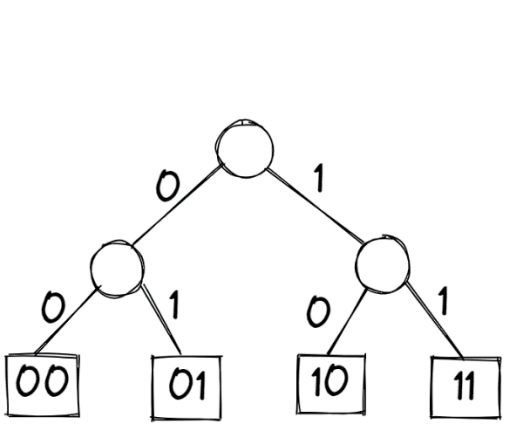
- 数据范围：0~0xff,ff,ff,ff,ff,ff
- 使用空间：8Byte

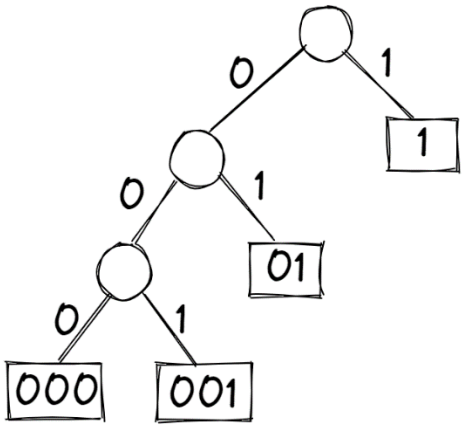
header 总体的编码长度是固定的共 10 个字节，其中各部分的编码长度是固定的无需使用前缀编码将其进行区分。由于 header 的编码长度固定当采样的数据较大时其编码长度几乎可以忽略。

2.1.2 压缩编码的采样数据

为了使 FPGA 算法实现更加简介高效，**假定在第一个采样时刻前的采样数据全部为零**，对所有数据均使用行程编码和增量编码的方式进行编码。进行编码的数据单位为一个时刻的采样数据，编码时首先将当前的采样时刻的数据与上一个数据发生变化时刻的数据进行比较判断得到当前数据中需要编码的数据。对于每一时刻的采样数据，编码时每 8 位（8bits）为一个值变化判断的基本单位，例如对于 256 采样模式一个时刻有 256bits 的采样数据将被分成 32 个（32 列）值变化判断的单位。

编码一个时刻的采样有以下 4 个部分构成：**delta timestamp（编码的采样时刻间隔）、row（行号，一个时刻的采样数据以 256bis 为一行被分为多行）、column group（列组号）、column index（列索引）**，为了区分这 4 个部分的数据需要使用额外的前缀编码将其分开，区分 4 个符号的前缀编码有如下两种形式：





左边的编码方式前缀码长度相同适合于 4 个部分出现频率较为一致的情况，对

于右边的编码方式适用于 4 个部分出现频率差别较大的情况，可以将频率出现最高的部分使用较短编码从而减少整体的编码长度。我们使用右边的编码方式对上述 4 个部分进行编码。具体的编码方式如下：

delta timestamp（采样时刻间隔）

- 编码长度：1Byte（8bits）
- 前缀码：000
- 标记位：0 或 1
- 当标记位为 0 时：0000~1111，表示间隔的采样时刻-1
- 当标记位为 1 时：0001~1000，表示后续有多少个字节用于表示间隔的采样时刻（即采样时刻的间隔大于 16 时，使用额外的字节编码采样间隔的长度 delta timestamp 的后 4 位用于编码所需的字节数）
- 样例：采样时刻间隔为 1（0000_0000），采样时刻间隔为 16（0000_1111），采样时刻间隔为 31（0001_0001, 0001_1111）

在对变化的信号进行编码时每一时刻变化信号的位置有行、列两个坐标组合确定，我们规定 256bits 为一行，一行中每 8bits 组成变化值的基本单位即一行当中有 32 列，每一行中的 32 列共享同一个行号。而这 32 列在进行列编码时有进一步细分为 4 组，每组 8 列，在同一组中的 8 列共享同一个列组号。

row（行号）

- 编码长度： 1Byte
- 前缀码：001
- 行号：00000（0）~11111（31）
- 样例：00000（第 0 行），00001（第 1 行），11111（第 31 行）

column group+flag（列索引编号号）

- 编码长度：1Byte
- 前缀码：1
- 组号：000,001,010,011,100,101,110,111
- 列标志位：0000~1111，用于标识后续值（value）的数量以及在组中

出现的位置

- 列标志位样例：

列标志位	说明	byte0	byte1	byte2	byte3
0001	后面有 1 个字节的 value，按顺序依次是组内的	1			
0010	后面有 1 个字节的 value，按顺序依次是组内的		1		
0011	后面有 2 个字节的 value，按顺序依次是组内的	1	1		
0100	后面有 1 个字节的 value，按顺序依次是组内的			1	
0101	后面有 2 个字节的 value，按顺序依次是组内的	1		1	
0110	后面有 2 个字节的 value，按顺序依次是组内的		1	1	
0111	后面有 3 个字节的 value，按顺序依次是组内的	1	1	1	
1000	后面有 1 个字节的 value，按顺序依次是组内的				1
1001	后面有 2 个字节的 value，按顺序依次是组内的	1			1
1010	后面有 2 个字节的 value，按顺序依次是组内的		1		1
1011	后面有 3 个字节的 value，按顺序依次是组内的	1	1		1
1100	后面有 2 个字节的 value，按顺序依次是组内的			1	1
1101	后面有 3 个字节的 value，按顺序依次是组内的	1		1	1
1110	后面有 3 个字节的 value，按顺序依次是组内的		1	1	1
1111	后面有 4 个字节的 value，按顺序依次是组内的	1	1	1	1

2.1.3 tail

2 压缩引擎一的尾编码为最后一个采样与最近一次发生变化采样的时间戳间隔，即为最后一次采样的 delta timestamp，但其值为固定大小 8byte。

2.2 编码样例

1） header 编码过程

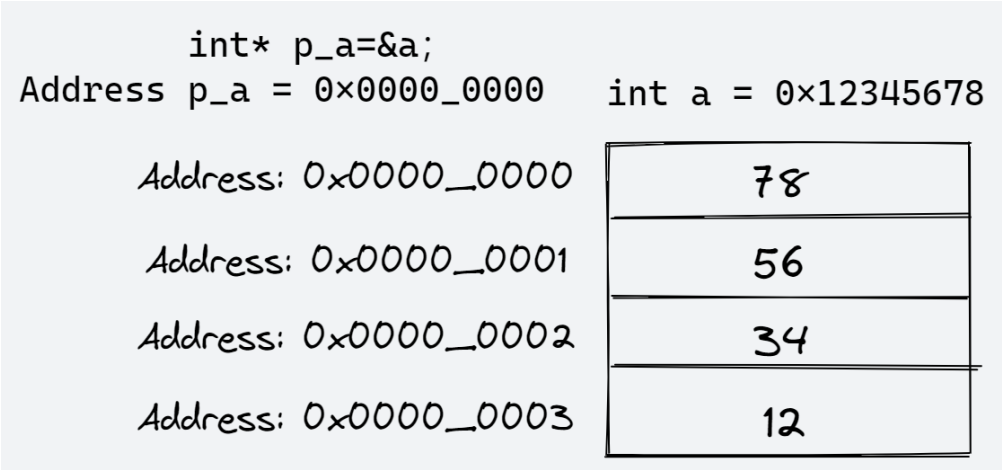
```
header  0000 0000 0123 4567 0000 0000 0000 0000 010a 0100 0000 0020 0020 0000 0001 86a0

parse   byte14  channel 0×0a
        byte7-6 model   0×0020
        byte5-0 timestamp 0×0000_000186a0

coding  channel   0×0a
        model     0×01
        tmestamp  0×a0 0×86 0×01 0×00 0×00 0×00 0×00 0×00

output  0×0a 0×01 0×a0 0×86 0×01 0×00 0×00 0×00 0×00 0×00
```

由于在 c/c++语言中以字符（byte）为单位读取或写入一个变量时变量的首地址为实际编码的低地址，所以编码时数据成逆序的形式。



2） 非第一个时刻数据编码过程

```
ts 100001  row1  95f7 00c1  0000 0000  0000 0000  95b0 61c1  0000 0000  0000 0000  95f7 6100  95f7 61c1
           row2  0000 0000  0000 0000  9500 61c1  0000 0000  0000 0000  00f7 61c1  0000 0000  0000 0000

对row2进行编码
coding
           delta ts: 0000_0000

                  前缀码        行号
                row: 0010_0000

                  前缀码  列组第1组  列flag
 column group+flag: 1000_1011

             vaule: 0×00,0×00,0×00

                  前缀码  列组第3组  列flag
 column group+flag: 1010_1101

             vaule: 0×95,0×61,0×c1

                  前缀码  列组第4组  列flag
 column group+flag: 1011_1101

             vaule: 0×00,0×00,0×00


 column group+flag: 1101_1110

             vaule: 0×f7,0×61,0×c1


 column group+flag: 1110_0111

             vaule: 0×95,0×f7,0×61


 column group+flag: 1110_1111

             vaule: 0×95,0×f7,0×61,0×c1


 output:  0000_0000,
          0010_0000,0×00,0×00,0×00,
          1010_1101,0×95,0×61,0×c1,
          1011_1101,0×00,0×00,0×00,
          1101_1110,0×f7,0×61,0×c1,
          1110_0111,0×95,0×f7,0×61,
          1110_1111,0×95,0×f7,0×61,0×c1
```

2.3 性能分析

我们以 256 模式为例分析当前编码方式的压缩率，因为在进行编码时 256bits 为一行，所以对于其他模式只是 256 模式的整数倍 256 模式的分析可以代表其他模式的情况。

我们考虑数据变化最频繁的情况，即每两个时刻都有数据发生变化每一行数据都需要进行编码。每一次编码首先需要编码 delta timestamp 和 row 供需 16bits，如果我们期望经过编码后数据不发生膨胀（小于等于 256bits）那么将剩余 240bits 用于编码变化的元素。根据变化元素的分布由如下 xx 种情况：

1) 所有的变化都在 1 个列组中，1 个列组中最多有 4 列的数据所需的编码位数为：8bits+8bits*4=40bits，40bits 小于 244bits 当前情况在不发生膨胀的前提下所有变化数据都能进行编码。（编码 4 列）
2) 所有的变化都在 2 个列组中，2 个列组中最多有 8 列的数据所需的编码位数为：2*(8bits+8bits*4) =80bits，80bits 小于 240bits 当前情况在不发生膨胀的前提下所有变化数据都能进行编码。（编码 8 列）
3) 所有的变化都在 3 个列组中，3 个列组中最多有 8 列的数据所需的编码位数为：3*(8bits+8bits*4) =120bits，120bits 小于 240bits 当前情况在不发生膨胀的前提下所有变化数据都能进行编码。（编码 12 列）
4) 以此类推，在不发生膨胀的前提下最大可以编码 6 个列组的信号，共 24 列，此时的翻转率为 24/32=75%。
5) 对于最极端情况，32 列全部都发生变化则需要：8bits+8bits+8*(8bits+4*8bits)=336bits，膨胀 1.31 倍。

如果将标准设为需将原始数据压缩至 1/7 即最多使用 36bits 进行编码，去除 delta timestamp 和 row 后只剩余 20bits 仅能编码 1 列数据，所以编码的翻转率为 8bits/256bits=3.125％。

3 第二级压缩引擎编码方案

第二级压缩引擎的输入为第一级压缩引擎的输出，在第一级压缩引擎与第二级压缩引擎之间需要有一个存储第一级压缩产生数据的缓冲区，第二级压缩引擎每次压缩过程中从数据缓存区 64k 字节（64KB）的数据进行压缩。第二级

压缩引擎考虑的对象为一组相邻的字节，使用了字典转换的压缩思想的 LZ 算法。对于周期性的波形信号其中周期性重复的数据经过第一级压缩引擎后得到保留，使用 LZ 算法可以将数据中后面出现的大量重复信号使用之前出现过信号的索引（distance，length）来表示实现较高的数据压缩率。

3.1 编码方案

第二级压缩引擎对波形数据的编码由以下两部分构成：**header、位图（bitmap）、LZ 编码数据**。

其中位图中的每一位用于标识 LZ 编码数据的字节数组中当某一字节是原始数据还是（distance，length）对。

3.1.1 header

header 用于编码第二部分位图所用的字节数。由于 LZ 编码数据的最大长度是 64K，所以位图的最大长度为 64K/8=8192，使用 2 个字节即可标识。

- 编码长度： 2Byte

3.1.2 位图

位图中的每一位用于标识 LZ 编码数据的字节数组中当某一字节是原始数据还是（distance，length）对。当 LZ 编码数据中为原始数据时为 0，为（distance，length）对为 1。

3.1.3 LZ 编码数据

LZ 编码的数据为字节数组

LZ 编码的数据根据 LZ 算法匹配的长度分为两类：

1) 匹配长度小于 3 个字节的子串以元数据的形式直接输出
2) 匹配长度大于等于 3 个字节的子串以(distance,lenght)对形式输出
   - distance：第一个字节为标识单位，（用来确定接下来要读多少个字节）。定义 BYTE_NUM_0，BYTE_NUM_1，BYTE_NUM_2 分别为额外要读 0 个字节、1 个字节以及两个字节的标识数量。示例：三个数字分别为 64、186、6，则表示（0-63）区间无需额外读取，（64-249）

区间需额外读取一个字节，（250-255）区间需额外读取两个字节。这个可以自行定义（对应压缩率、耗时的选择）。当不额外读取字节时，距离的值就等于该字节的值+1；当额外读取一个字节时，标识单位 X 所表示的范围为 256，范围为(1+BYTE_NUM_0+(X-BYTE_NUM_0)*256, 1+BYTE_NUM_0+（X-BYTE_NUM_0）*256+256)，接下来读取的字节 Y 标识具体的 bias 值。当额外读取两个字节时，标识单位 X 所表示的范围为(1+BYTE_NUM_0+BYTE_NUM_1*256+(X-BYTE_NUM_1)*65536,1+BYTE_NUM_0+BYTE_NUM_1*256+(X-BYTE_NUM_1)*65536+65536)，接下来读取两个字节 Y 和 Z，Y+Z*256 为具体的距离 bias 值。

- ◆ 当获得 distance 后，可根据距离的长度先找到所对应的标识单位。然后再计算得出所需的额外字节。

- length:length 的范围为 3~256，使用 1 个字节进行编码即可。

3.1.4 性能分析

第二级压缩引擎的编码大小极为依赖数据，当数据中存在较多的重复字节时会获得较好的压缩效果，header 的大小为 2 字节可以忽略不计，位图的最大可能长度为 64K/8=8192 字节，LZ 数据编码部分由于当 distance 最大时为 3 个字节加上 length 的 1 个字节当编码匹配长度为 3 且 distance 极大的情况会产生 1 个额外字节的膨胀当该种情况发生的次数非常少。

4 FSE 压缩引擎编码方案

FSE 压缩引擎的输入为第二级压缩引擎的输出，使用的是有限状态熵编码方式。 FSE（或 tANS-tabled 非对称数字系统）是 Jarek Duda 提出的非对称数字系统的一种变体，它将算术编码（AC）的精度和哈夫曼编码的速度结合起来。FSE 是一个状态机，它存储对消息进行编码和解码所需的所有信息。作者认为，霍夫曼编码是 FSE 的一种特殊退化状态，其中编码每个符号所需的位数为整数。ANS 的基本思想类似于 AC，但它在每一步后都会产生更大的范围，而 AC 会使范围越来越窄。ANS 和 FSE 中也有范围重整化，重整化是在创建编码表的过程中

计算的（因此，将编码表存储在内存中并将表传输到解码器是一种快速的折衷方法）。值得注意的是，FSE 解码是反向工作的，即它将最后一个符号解码为第一个符号。

4.1 编码方案

FSE 压缩引擎对波形数据的编码由以下两部分构成：header、FSE 编码数据。

4.1.1 header

header 中包含 256 个 16bits 整数来表示 FSE 的 freqs，4 个 32bits 整数分别表示 byte_offset，state，encoded_size 和 data_size。

- 编码长度： 528 Byte

4.1.2 FSE 编码数据

FSE 编码的数据为比特数组。通过有限状态熵编码思想构建。编码过程分为计算 freqs，获得 encode_table，以及编码三个步骤。第一步为根据输入比特流中各个字符的出现次数计算他们的频率并做标准化。第二个步骤为根据第一步生成的 freqs 数组，创建一个备查表。接下来是一个例子展示。

state	A	B	C
1	2	3	5
2	4	6	10
3	7	8	15
4	9	11	20
5	12	14	25
6	13	17	30
7	16	21	
8	18	22	
9	19	26	
10	23	28	
11	24		
12	27		
13	29		
14	31		

如上图所示。编码可以从任何行和字母表的任何符号（A、B、C）开始。如果消息中的第一个符号是 B，第一行是 1，则我们从横穿第 1 行和第 B 列的位置获取新的行号，即 3，并移动到下一行。如果下一个符号是 A，则从 A 列和第 3 行的交叉处读取下一行号。这个数字是 7，我们可以继续，直到我们的表的最后扩展。

解码和编码一样基本。有任何数字，比如说 16，我们可以提取符号 A 和前面的第 7 行。回到表中的第 7 个元素，我们可以提取另一个符号 a 和另一个新行 3。最后，我们从元素 3 中提取 B 和行 1，这是表和消息末尾的最高点。成功的编码和解码是可能的，需满足以下条件：表中的左列包含所有序列号；A、B、C 列中的所有数字都是唯一的（无重复）；每行中的数字都大于该行的数字；所有列中的数字都已排序。

每个步骤后获得的行数称为状态。有了状态，我们可以恢复消息。如果我们将行数除以列中的值，则当选择列中的数字以匹配对应符号的出现概率时，此编码器成为熵编码器。在这个表中，如果我们将行数除以列 A 中的任何元素，

我们得到大约 0.45。B 的比率为 0.35，C 的比率为 0.2。根据公式 statei=statei-1/P，状态在每一步上都在增长，这意味着在编码结束时，我们将有一个长度等于 1/[P{B}*P{A}*P{A}…]的数字，这显然与熵一致。

在解释编码过程之前，让我们先说明解码过程。假设我们要解码以下二进制表达式：11000011110。根据 ANS 编码器的主要特性进行反向解码。根据表的大小，我们指定状态缓冲区的大小为 5 位。在第一步中，我们读取上面显示的编码表达式的最后 5 位，即 11110，并用这个数字（即 30）对表进行寻址。我们提取符号 C 并获得新的状态 6 或二进制格式 110。为了使我们的新状态缓冲区匹配大小 5，我们从表达式中再读取 2 位，得到 11000。这是 24。对于 state=24，我们提取符号 A 并创建新的 state 11，它是二进制格式的 1011。我们再读一位，得到（10110）22。然后我们做同样的事情：我们提取 B，获取状态=8，再读取一位，获取 16，提取 A，获取 7，读取最后两位，然后获取消息标记 11111 的结尾。我们知道这是消息的结尾，因为没有更多的位可读取，状态与默认标记匹配。

Encoding process

state before output of bits	31	16	22	24
output bits	11	0	0	00
state after bit output	7	8	11	6
processed symbol	A	B	A	C
state after processing	16	22	24	30

解码过程比编码简单得多。在编码中，我们需要输出位来提供我们在解码中使用的这个简单规则。编码过程如左表所示。我们可以从任何 5 位数字开始，但我们更喜欢区分结束/开始标记 31（11111）。编码的概念稍微复杂一点。我们在处理符号之前输出位。正如您所看到的，我们无法处理 31，因为没有行号 31，所以我们根据需要输出尽可能多的位来处理符号。我们可以发现这一点，因为我们知道我们在处理什么符号。它是 A，我们需要输出两个最低有效位，即 11。我们在表格中显示了整个过程。输出位之前的数字是 31。在这个数字下

面是我们输出到流的两个位。输出状态后的结果=7。我们用符号 A 处理 7，得到新状态=16。下一个符号是 B。根据表，我们无法处理状态为 16 的 B，因此我们需要输出一个最低有效位 0，并使新状态为 8。在处理状态=8 的 B 之后，我

们得到 22，以此类推。我们必须输出最低有效位，因为我们需要减少数字。在我们处理最后一个符号 C 之后，我们得到 state=30，并且我们完全输出它，所以以我们的输出缓冲区看起来就像上面所示。

4.1.3 性能分析

FSE 压缩引擎的编码大小较不依赖于数据。FSE 压缩引擎编码大小与数据的信息熵有关，更接近与理论算术编码的值。FSE 运行时长远低于第二级压缩引擎。以下为压缩数据结果表：

模式	256		512	
翻转率	R20	R40	R20	R40
原大小	51000000	51000000	51000000	51000000
GZip	19027613	**35540351**	20564402	**38286255**
LZ77	24494012	43329419	26274077	44894734
LZ77+FSE	**18566119**	36433144	**19250342**	38941524
模式	1024		8192	
翻转率	R20	R40	R20	R40
原大小	51000000	51000000	51000000	51000000
GZip	23161759	41296250	32574706	**46793145**
LZ77	28377073	47602324	36340283	51939390
LZ77+FSE	**21520063**	41686945	**28968427**	48661534