February 19, 1997
Version 1.0

STAR Note 282
TPC Table Formats for the Cosmic-Ray Test

R. Bossingham
Nuclear Science Division
Lawrence Berkeley National Laboratory

MiniDAQ TPC data formats are defined for the TPC cosmic-ray test within the STAF table framework. In particular, a format is defined for zero-suppressed, pad-mapped data in a way that is usable and efficient enough to provide a starting point for designing a final DAQ format: data is indexed by sector and pad row, with 35% storage overhead for Au-Au events.

Introduction

An interim TPC data format for MiniDAQ to use in the System Test and Cosmic-Ray Test is defined here. A DAQ format standard does not yet exist, though it has been discussed by Mike Levine, Doug Olson, Lanny Ray, Jo Schambach, Craig Tull, myself and others. The definition here borrows from those earlier discussions—in particular, the idea of segregating overhead from data, allowing overhead tables to define multiple data tables. Susanna Jacobson has also contributed to the design through discussions and her testing work on MiniDAQ.

The final DAQ format will presumably differ, since the interim standard is partly determined by MiniDAQ limitations. Nonetheless, the interim format is intended to be realistic, so the format is less tuned for the sparse data of the Cosmic-Ray Test than for Au-Au collisions. The format has evolved since my first proposal, for both practical and aesthetic reasons. STAF problems with word-boundary alignment required that long int's be placed before four char variables in the rows of the index table; the format was generalized to include, as a subset, the one now in use; "black" events and gain, translation and FEE-location tables are supported; bit manipulation by the user is no longer required; and parallel data for a channel or pixel is put into parallel tables.

Supported Data Types

MiniDAQ supports only limited hierarchy for its output tables, while STAF does not explicitly support hierarchy within a table. MiniDAQ is also constrained, more or less, to group data by readout board—multiplying the number of tables by six—while the FEE cards on these boards may be mapped randomly. There are also several different classes of data needed off-line, especially for debugging. These factors result in more readout-board orientation and fewer tables than might be optimal for DAQ. Still, the data structures were chosen with storage and access efficiency in mind, and chosen to be "natural," exploiting the capabilities of STAF, rather than just a bag o'bytes.

A central design choice was to segregate all overhead into two tables per readout board, with additional tables for pure data; defined data types are stored with the overhead, for proper interpretation of the remaining overhead. These are listed in Table 1. The data types are defined only to the extent that the table structure may be affected, and do not necessarily reflect the exact content.

For rows listing two data types, the even one suppresses bad channels, the odd one does not. The distinction is useful for the off-line decoding in some cases. For example, a 10-bit ADC table of type 7 data from one readout board can be made identical to the tables now in use, and could be used without referencing the overhead tables. (On the other hand, zero-suppression makes the structure of types 1 and 3 data unpredictable.)

Type	Type Definition			Index 1	Index 2	
0, 1	Pixel Scalars	Zero-suppressed	Mapped	Row (45–1)	Pad (1–184)	
2, 3	”	”	Native	Group (1–72)	Channel (1–16)	
4, 5	”	Black (all pixels)	Mapped	Row (45–1)	Pad (1–184)	
6, 7	”	”	Native	Group (1–72)	Channel (1–16)	
8, 9	”	Zero-suppressed	Special	Marker (200)	Channel (1–20)	
10, 11	”	Black (all pixels)	Special	Marker (200)	Channel (1–20)	n
16, 17	Channel Scalars		Mapped	Row (45–1)	Pad (1–184)	
18, 19	”		Native	Group (1–72)	Channel (1–16)	
32, 33	Channel Tables		Mapped	Row (45–1)	Pad (1–184)	
34, 35	”		Native	Group (1–72)	Channel (1–16)	
48	RDO FEE Locations					
64	RDO Event Header					

Table 1: Defined data types. Odd-numbered data types do not suppress bad channels.

Event Format Definition

Three basic table types are used:

Index Table Each 12-byte row always includes the row number for data in the related structure table, the data type as defined in Table 1, the sector and readout board providing the data. For some data types, index 1 helps to specify the data source and a row number points to a location in additional data tables. The row definition appears in Table 2. Note that there is no requirement that all entries of an event type appear sequentially or in any particular order. This flexibility may be helpful if tables from different readout boards are merged offline.

Structure Table This table begins with a list (redundant, but possibly useful) of the data types that it might contain , followed by information defining the additional data tables; see Tables 3, 4 and 5. Data high in the hierarchy, such as readout board headers and FEE location lists, are included directly; see Tables 6 and 7.

Data Tables These tables contain most of the actual data: ADC values, pedestal averages, gain shifts, etc. To the extent possible, tables are defined with parallel structure. The mean pedestal, for example, is put in row n of one table, the rms in row n of another, and the shift from the previous value in row n of another; any of these tables can be produced with little extra overhead. A convention must be defined to allow the off-line analysis to infer the names of all possible data tables from the data type.

Entry	Bytes
Structure table row number	4
Data table row number	4
Data type	1
Sector number (1–24; 0, if unknown)	1
Readout board location (1–6; 0, if unknown)	1
Index 1	1
Row size	12

Table 2: TPC index table row definition: 12-byte rows, one table per readout board.

Entry	Bytes
N_{types}: number of possible data types in table	1
List of N_{types} data types	$N_{\text{types}} \times 1$
Structure information or data	

Table 3: TPC structure table format: one-byte rows, one table per readout board. The interpretation of most of its contents varies according to data type.

Entry	Bytes
N_2: Number of index 2 entries	1
Index 2	1
Number of sequences starting in first 256 pixels (0–128)	1
Number of sequences starting in second 256 pixels (0–128)	1
Location of first pixel in sequence (0–255)	1
Number of other pixels in sequence (0–255)	1

Table 4: Structure table contents for data types 0–3, 8, 9 (zero-suppressed pixel data). The corresponding data tables have one row per pixel.

Entry	Bytes
N_2: number of index 2 entries	1
List of index 2 entries	$N_2 \times 1$

Table 5: Structure table contents for data types 4–7, 10, 11 (black, pixel data), 16–19 (channel scalars), and 32–35 (channel tables). The corresponding data tables have 512, 1 or 1024 rows per channel.

Datum	Bytes
FEE 1 location	1
FEE 2 location	1
FEE 3 location	1
.	.
.	.
.	.
FEE 36 location	1

Table 6: Structure table contents for data type 48: FEE locations on a readout board.

Entry	Bits	Definition
0	4	trigger CMD
1	4	DAQ CMD
2	4	trigger token
3	8	trigger token LSW
4	8	bunch crossing counter
5	4	detector ID
6	8	readout unit ID
7	8	spare
8	8	spare
9	8	spare
10	8	spare
11	8	spare
12	8	tagword ($de)
13	8	tagword ($ad)
14	8	tagword ($fa)
15	8	tagword ($ce)
16	8	FEE 1 location ID
.	.	.
.	.	.
.	.	.
51	8	FEE 36 location ID
52	-	detector specific (undefined)
	.	.
	.	.
	.	.
63	-	detector specific (undefined)

Table 7: Structure table contents for data type 64: readout board event header (from "Front-End Electronics to DAQ Receiver Board Fiber Optics Interface Specification Version 1.0a" by V. Lindenstruth et al.; and from discussions with Fred Bieser.)

Pixel Data (Types 0–7)

The primary pixel data are ADC values. Raw, ten-bit values are listed as one unsigned short int per table row; translated, 8-bit values are listed as an unsigned char. Track data would normally be zero-suppressed, minimizing data storage and off-line analysis. However, raw pedestal data must contain the full data for all pixels, since pedestals can vary by several counts for a single channel. The zero-suppressed data formats can support black events, but other, more efficient formats would usually be used.

Additional information is needed to interpret pixel data: whether it is gain corrected; the threshold and other criteria used for any zero suppression; and the bit patterns designating overflows. This is to be passed as a one-row table in a beginning-of-run event.

Besides raw ADC's, pedestal runs can produce tables of derived data: pedestal means, rms's, shifts, valid measurements, etc. These tables are also defined with one row per pixel and appropriate variable types.

For data types 0–35, even type numbers indicate the suppression of channels defined as "bad." Note that channels not connected to a pad are always suppressed for pad-mapped data, since the mapping for them is undefined.

Mapped data The readout board event header contains enough information to map the ADC information onto pads, eliminating the need to do so offline. Most or all tracking data is expected to be mapped. Since the map cannot change during a run, the mapping information need only be calculated once. Further, since data is, in any case, not read sequentially from Rosie VRAM memory, there is little performance penalty once the map has been calculated.

The data for a given readout board is ordered by descending pad row number, and increasing pad number within a row. The pad-row ordering might be exploited for tracking and quick multiplicity estimates. Data from all pixels for a given pad are grouped, as are data from all pads in a given pad row, but pad rows may be split between two or more readout boards.

RDO-referenced (native) data There are two obvious reasons to use data referenced to the readout board channels (native format), rather than to the pad locations. First, this is the format used in the system test from mid-1996 onwards, and is now supported by the pad monitor and analysis software; with data type 7, 10-bit ADC data tables with identical format can by produced. Second, some studies may be more oriented to the electronics (e.g., noise), than to the TPC itself. In particular, channels that are not connected to a pad are suppressed for pad-mapped data.

Channels for a given readout board are recorded in the same (somewhat complex) order as received (16 channels from each of the first nine cards, followed by the other 16 channels from each of these cards, and so on). Data from all pixels for a given channel are grouped.

Anode-Wire Data (Types 8–11)

For the cosmic-ray test, one FEE card will be dedicated to anode-wire readout for 20 wires. This is indicated by a distinctive FEE location marker (200); the data are stripped from the normal TPC data stream, inverted, given a special event type, and put into a separate data table.

Channel Scalars and Tables (Types 16–19, 33–35)

Aside from pedestals, most other electronics characteristics (gain, time offset, time constant, etc.) can be described by single parameters applying to all pixels within a channel. As with pixel data, multiple tables with parallel structure are defined in conjunction with a single index/structure table pair. Also similarly, one can suppress bad channels and optionally map the data onto pads.

The 10-to-8-bit lookup table for a channel is calculated from the scalar channel information for that channel, so one would normally neither want or need to export it. However, a data type is defined to do this for debugging. These 1024-entry tables are quite large. If they were exported on a regular basis, a data type for compressed tables could be defined using a non-lossy, 19:1 compression algorithm, already within MiniDAQ.

Readout Board Data (Types 48, 64)

FEE location data Another type of TPC data comes from reading out FEE card locations. These are read as a highly redundant, black event; the obvious format is shown in Table 6. The FEE locations in the table should be the same as those in the event header, but might be used for debugging and to define analysis geometry at the beginning of a run.

Readout board event header A header arrives with the data from each readout board. Its contents are given in Table 7. This information is crucial to format the tables: the sector number, RDO number, and FEE locations are all obtained in this way. Including the header in the data stream is useful for debugging, etc.

Design Discussion

Several possibly controversial choices were made in defining the interim format:

1. Two overhead tables per Rosie, besides the data tables (0–6 groups of tables per event). For the system test, the six readout boards will usually give 18 tables per event, with more for debugging or calibration runs. This choice eases the MiniDAQ implementation, but complicates the analysis, somewhat.

2. The MiniDAQ event builder will request all tables from each Rosie that might be filled for a given run type (e.g., both 8-bit and 10-bit ADC data tables). The Cyclone boards will return zero byte counts for unwanted tables, and the event builder will suppress them.

3. The tables are logically linked, requiring at least three tables to pass ADC data. This was done for aesthetic reasons and efficiency, but is also exploited to ease the MiniDAQ implementation. Specifically, the Cyclone board is implicitly asked to calculate the tables when the Index Table is requested, and implicitly asked to clean up when the Structure Table is requested.

4. Twenty-two data types have been defined, somewhat complicating the off-line analysis. However, the tables have been designed to require only minimal decoding work, and the task should be relatively easy. Also, any subset of the data types can be ignored, while still analyzing others.

5. The readout board header is used to obtain sector and RDO numbers, requiring that the hardware addresses be set correctly; in addition, the header provides the FEE card location (hence, the row and pad) for pad-mapped data, requiring firmware support on the readout board.

6. Pad rows—but not individual pads—are indexed; there is no provision to randomly access pads. This saves storage space, possibly at the expense of future analysis efficiency.

7. Pixel sequences are split into those starting in the first or the second half of the time buckets, and sequences may not exceed 256 pixels. Within these limitations, two bytes define the extent of a sequence. The split into 256-pixel halves is made at the pad level, so that the data for a pad can be easily recombined.

Storage Overhead

To estimate storage overhead, we use the same approximation for Au-Au events as in previous data format discussions: $N_{\rm seq} = 4$ sequences per pad, $L_{\rm seq} = 8$ pixels per sequence. We assume one set of tables per sector, with two bytes listing the data types at the start of each sequence tables; we do not include the overhead for implementing the tables.

There are 1080 pad rows and 136,560 pads in the TPC, so the amount of 8-bit ADC data is

$$L_{\rm seq} N_{\rm seq} N_{\rm pads} = 8 \cdot 4 \cdot 136,560 = 4,369,920 \text{ bytes} ,$$

while the storage overhead is 35%:

$$48 + 13 N_{\rm rows} + (3 + 2N_{\rm seq}) N_{\rm pads} = 48 + 13 \cdot 1080 + 11 \cdot 136,560 = 1,516,248 \text{ bytes} .$$

Now consider a single cosmic-ray track (90 hits). If these (minimum-ionizing) hits are 3 pads wide, 5 pixels long, the size of the 8-bit ADC data is

$$L_{\rm seq} N_{\rm wide} N_{\rm rows} = 5 \cdot 3 \cdot 90 = 1350 \text{ bytes} ,$$

while the storage overhead is 187%:

$$4 + 13 N_{\rm rows} + (3 + 2N_{\rm seq}) N_{\rm wide} N_{\rm rows} = 4 + 13 \cdot 90 + 5 \cdot 3 \cdot 90 = 2524 \text{ bytes} .$$

Hierarchical structures are inefficient for sparse events; a simple list of the sequences in a sector would have an overhead of "only" 106%. However, the expected data volume from the cosmic-ray test is still acceptable. One expects 43,200 events from 24 hours of running at a 0.5 Hz event rate; this amounts to a data volume of 162 MB, not counting table overhead. Running continuously for 75 days (*not* planned), would still yield a manageable 11.9 GB.

Black (e.g., pedestal) events have relatively little overhead because of their known structure. For 10-bit data stored as short ints, the ADC data size is

$$139,837,440 \,\text{bytes}\ ,$$

and the storage overhead is 0.11%:

$$48 + 13N_{\text{rows}} + N_{\text{pads}} = 48 + 13 \cdot 1080 + 136,560 = 150,648 \,\text{bytes}\ .$$

Of course, the format now used—without support for zero-suppression or mapping—has negligible storage overhead, but 0.11% is also nearly negligible. In any case, one expects to store little data without zero suppression in the future.

How Squishy are Our Tables?

The MiniDAQ table format defined above is not the tightest possible; compromises were made, with ease of implementation and ease of use being given much higher priority than storage efficiency. In spite of this, the Au-Au event overhead is competitive with other proposed schemes.

First consider the index table; its fractional contribution to the size of Au-Au events is negligible, but becomes significant for sparse events. We do not try to determine the exact information content of the tables for realistic events, but make only simple estimates:

1. Four bytes are used for each of the table row entries; with one table per readout board, less than 2.5 bytes would suffice.

2. Two separate bytes are used for the sector and RDO numbers to avoid bit manipulation, but these could be squeezed into one byte.

3. The data type would fit into 5 bits.

4. Index 1 could fit into 7 bits.

So, overall, the index table could be compressed by ~38%.

The structure table also consumes some extra space, especially for zero-suppressed data:

1. The list of data types included in the table is redundant, given only for convenience.

2. The number of sequences starting in 256 pixels cannot plausibly exceed 128, and even 5–6 bits might suffice for these counters.

3. The number of pixels in a useful, zero-suppressed sequence should not exceed 32, so 5 bits should suffice for the sequence length.

Overall, the structure table could be compressed by ~19% for zero-suppressed Au-Au events. The worst offender among the data tables was already discussed: the 8-to-10-bit lookup tables can be compressed by a factor of 19 or more. Another example is the storage of 10-bit data in a short int variable; such tables are trivially compressible by ~38%.

What does this mean for 8-bit, zero-suppressed, Au-Au data? Not much, actually. The 8-bit ADC data is not easily compressed, and the overhead adds only 35%; the savings come to about 5%.

How Squishy are STAR's Off-line Pixel Tables?

It is also interesting to compare the MiniDAQ data format defined here with that currently used to store TPC data from the Slow Simulator. Each pad is indexed with four long int's, and each pixel consumes another long int:

```
struct tss_tppad{
     long       jpix;       /* offset into the tppixel table */
     long       nseq;       /* number of sequences for this pad */
     long       secpad;     /* unique pad id within a sector */
     long       tpc_row;    /* encoded sector-row = 100*sector + row */
};

struct tss_tppixel{
     long       datum;      /* encoded tzero, nbins, adc values */
};
```

For our "typical" Au-Au event, the total storage in this format is

$$(16 + 4N_{\rm seq}L_{\rm seq})N_{\rm pads} = (16 + 4 \cdot 4 \cdot 8) \cdot 136,560 = 19,664,640 \text{ bytes} .$$

In contrast, storing 10-bit data as short ints and using the MiniDAQ format would require 8.33 MB for data storage and 1.45 MB for overhead, for a total of 9.78 MB—a factor of two less. (At the cost of some aesthetic ugliness and doing bit manipulation, one could pack three 10-bit ADC's into a long int variable for the MiniDAQ format, shrinking the total storage requirement to 7.01 MB, or 2.8× less than the current Slow Simulator approach. Of course, the comparison is not entirely fair, since the `tppad` table addresses data by pad, rather than row.

Implications for Configuration Management

Primitive configuration management will be used for the system/cosmic-ray test, supporting a small number of pre-defined configurations. For a given configuration and DAQ action word, MiniDAQ will request tables from a fixed list of named tables; those with zero length are suppressed. Therefore, two runs having explicitly different lists of associated tables require two separate configurations.

An explicit event-builder mechanism will be used to suppress all tables from a given Rosie, but, to suppress only a subset of those tables, the Cyclone boards must return a zero length for unwanted tables. Run Control and MiniDAQ must therefore make the requisite information available to the Cyclone boards at configuration time. The above data format, with overhead and data segregated, facilitates such a scheme, since data tables are logically independent.