



AFEII-t Note: Data Readout Format

DATE: 6/1/2007
FROM: PAUL RUBINOV
RE: **READOUT** FORMAT OF A/T DATA

BACKGROUND

This note describes the current and proposed future scheme for packing AFEII-t data for readout. In fact, the AFE boards (both AFE1 and AFEII-t) feature two readout buses: a 40 MByte/s 8bit parallel bus which is used to send ADC information after a L1 accept has been issued to the detector by the trigger system and a 530MByte/s LVDS bus used to send discriminator data to the Central Track Trigger (CTT) components of the trigger every crossing. Unfortunately, due to the architecture of the trigger system, the high speed LVDS bus is not available for readout of the data for off line analysis. Data that is available off line must be read via the 40 Mbyte/s “gray cable” bus. From now on, this note will discuss only this data.

The current AFEII-t readout scheme is designed to match the AFE1 readout format for compatibility. This means that when installing AFEII-t boards, any software that uses the data from the board does not need to know if the data came from an AFE1 or an AFEII-t board. The data contains the digitized value of the hit amplitude for those channels that are above threshold. Channels with a digital value below some readout threshold are “zero suppressed” and not read out. However, the AFEII-t has a feature which the AFE1 does not: the ability to measure the time a discriminator has fired, relative to the crossing clock. This additional data does not fit neatly into the AFE1 readout format.

CURRENT FORMAT

The readout scheme for the gray cable is designed to reduce the data size of the readout from a given board of 512 channels. The current format consists of a pair of bytes, one dedicated to the channel address, followed by a data byte. The address byte is necessary because the data is zero suppressed, and some indication of which channel the data represents is required. The standard format for the analog data is also grouped by module, with 64 channels per module and 8 modules per board, with a special separator that identifies the module within a board. For symmetry this separator also consist of two bytes: the “Chip ID” byte and the “zero” byte. To be able to distinguish the separator from the usual channel address byte, the “Chip ID” always has the most significant bit set, while the channel address, since it needs to identify only 64 channels, never has the most significant bit set. This format is shown in Table 1.

In addition, the AFE boards are also part of the L1 trigger system, providing one discriminator bit for every channel (for CFT axial boards) to the trigger every crossing, and since the trigger system itself does not record this raw information, the AFE packs this information into the data stream for L3 readout in a special block, distinct from the analog hit data. This discriminator hit block is packed into a format that is similar to the format that is used for the analog block. Like the analog block, it contains a “Chip ID” and the “zero” byte followed by alternating bytes of address and data. The block always consists of 64 bytes of address and 64 bytes of data (representing the 512 trigger bits). However, since this digital data is never zero suppressed, the address bytes are entirely superfluous. The format for the trigger data block is shown in Table 2.

NEW FORMAT

In order to take advantage of the timing capability of the AFEII-t, the timing information must be sent off-line via the gray cable, same as the amplitude information. Both are required, therefore the amount of data is doubled. To reduce the impact of this new data on the readout time (and potentially the dead time) of the detector, we propose a new, more compact readout format for the data. We propose to change the format of both the analog data block and the trigger data block, but in fact these changes are independent and can be implemented individually, which should aid in testing and debugging of the new format. For those readers who are familiar with the detailed architecture of the board, the change will be implemented by new firmware; new COLLECTOR firmware to implement the trigger block change, and new AFPGA firmware to implement the analog block change.

The format for the analog block will still consist of data grouped by module, with the Chip ID byte and the “zero” byte separating the modules, however, the channel address bytes will be replaced by the timing data, so the data will still consist of byte pairs, but now they will be timing/amplitude byte pairs instead of address/amplitude pairs. Since some indication of which channels are readout is still required, 8 additional bytes will be appended between the Chip ID/zero byte and the timing data/amplitude data portions of each module. These 8 bytes will indicate which of the 64 channels of the module that are “hit” for analog readout in the same way as the trigger data block indicates which of the 512 channels have been hit for trigger readout. This format is shown in Table 3. It is important to note that the convention for bits will be as follows: the least significant bit of the first byte will represent the lowest numbered channel- the channel that reads out first in the current format, and the most significant bit of the last byte will represent the highest numbered channel- the channel that reads out last in the current format. This format, which we will refer to as the “hit map” scheme is more efficient than the current “channel address” scheme when there are more than 8 channels hit in a given module. In other words, the new format is more efficient than the current format when the module occupancy exceeds 12.5% (when the same amount of data is transmitted). Although the new format is more efficient for transmitting the additional timing data than the current format, a disadvantage of the new format is that it is harder to detect a transmission error, since there is less redundant information built into the format. In order to remedy this deficiency, the new format will include a 16 bit CRC word at the end of the data for each module. In order for the CRC to provide a useful data integrity check, the algorithm needs to be clearly defined. We hereby define that the CRC will be computed for each modules data independently, including the Chip ID/zero byte using the 16-bit CRC-CCITT specification as given in <http://www.itu.int/rec/T-REC-X.25-199610-I/en>. A general discussion of the CRC can be found Ref 1 and the CCITT.X25 is also discussed in Ref 2. The VHDL implementation for this CRC is given in Appendix 1 of this note.

The trigger data block is modified by simply dropping the address bytes completely, as shown in Table 4.

The use of the new format will be indicated to the software by changing the Chip IDs currently used by adding 16 (0x10) to the value of each Chip ID.

Table 1: Format for analog data from each of 8 modules on AFE.

Byte number	Byte name	Comment
1	Chip ID	Always has MSB set (0x80 to 0x87)
2	Zero byte	Always "0x00"
3	Channel address	MSB never set
4	Channel data	Gray coded amplitude value 0 to 255
5	Channel address	Next non suppressed channel address
6	Channel data	
...		
2n-1	Channel address	
2n	Channel data	

Table 2: Format for the trigger block of the AFE.

Byte number	Byte name	Comment
1	0x8F	Chip ID byte- always the same
2	Zero byte	Always "0x00"
3	00	Always 00
4	Channel hit data	Data for first byte, first module
5	10	Channel address byte- always the same
6	Channel hit data	Data for first byte, second module
7	20	Channel address byte- always the same
8	Channel hit data	Data for first byte, third module
9	30	Channel address byte- always the same
10	Channel hit data	
...		
19	01	
20	Channel hit data	
21	11	
22	Channel hit data	
2n-1	ij	Channel address for (i+1)th byte, (j+1)th module
2n	Channel hit data	Data for (i+1)th byte, (j+1)th module
...		
127	67	
128	Channel hit data	Data for last byte, next to last module
129	77	
130	Channel hit data	Data for last byte, last module

Table 3: Proposed new format for AFEII-t analog data block.

Byte number	Byte name	Comment
1	Chip ID	Always has MSB set (0x90 to 0x97)
2	Zero byte	Always "0x00"
3	Hit map byte 1	Bit map of first 8 channels (1=hit). Ch0= LSB
4	Hit map byte 2	
5	Hit map byte 3	
6	Hit map byte 4	
7	Hit map byte 5	
8	Hit map byte 6	
9	Hit map byte 7	
10	Hit map byte 8	
11	Channel timing	Timing value for first non zero suppressed channel
12	Channel data	Amplitude for first non zero suppressed channel
13	Channel timing	
14	Channel data	
...		
2n-1	Channel timing	Timing value for last non zero suppressed channel
2n	Channel data	Amplitude for last non zero suppressed channel
2n+1	CRC MSB	Most significant byte of the CRC
2n+2	CRC LSB	Least significant byte of the CRC

Table 4: Proposed new format for AFEII-t trigger data block.

1	0x9F	Chip ID byte- always the same
2	Zero byte	Always "0x00"
3	Channel hit data	Data for first byte, first module
4	Channel hit data	Data for first byte, second module
5	Channel hit data	Data for first byte, third module
66	Channel hit data	Data for last byte, last module

APPENDIX 1

The VHDL implementation for X25 specification for an 8 bit data width.

```
-----  
-- File: PCK_CRC16_D8.vhd  
-- Date: Mon Jul 16 13:42:10 2007  
--  
-- Copyright (C) 1999-2003 Easics NV.  
-- This source file may be used and distributed without restriction  
-- provided that this copyright statement is not removed from the file  
-- and that any derivative work contains the original copyright notice  
-- and the associated disclaimer.  
--  
-- THIS SOURCE FILE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS  
-- OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED  
-- WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.  
--  
-- Purpose: VHDL package containing a synthesizable CRC function  
-- * polynomial: (0 5 12 16)  
-- * data width: 8  
--  
-- Info: tools@easics.be  
--      http://www.easics.com  
-----
```

```
library IEEE;  
use IEEE.std_logic_1164.all;  
  
package PCK_CRC16_D8 is  
  
    -- polynomial: (0 5 12 16)  
    -- data width: 8  
  
    function nextCRC16_D8  
    ( Data: std_logic_vector(7 downto 0);  
      CRC:  std_logic_vector(15 downto 0) )  
    return std_logic_vector;  
  
end PCK_CRC16_D8;  
  
library IEEE;  
use IEEE.std_logic_1164.all;  
  
package body PCK_CRC16_D8 is  
  
    -- polynomial: (0 5 12 16)  
    -- data width: 8  
  
    function nextCRC16_D8  
    ( Data: std_logic_vector(7 downto 0);  
      CRC:  std_logic_vector(15 downto 0) )  
    return std_logic_vector is  
  
        variable D: std_logic_vector(7 downto 0);  
        variable C: std_logic_vector(15 downto 0);  
        variable NewCRC: std_logic_vector(15 downto 0);
```

```

begin

D := Data;
C := CRC;

NewCRC(0) := D(4) xor D(0) xor C(8) xor C(12);
NewCRC(1) := D(5) xor D(1) xor C(9) xor C(13);
NewCRC(2) := D(6) xor D(2) xor C(10) xor C(14);
NewCRC(3) := D(7) xor D(3) xor C(11) xor C(15);
NewCRC(4) := D(4) xor C(12);
NewCRC(5) := D(5) xor D(4) xor D(0) xor C(8) xor C(12) xor C(13);
NewCRC(6) := D(6) xor D(5) xor D(1) xor C(9) xor C(13) xor C(14);
NewCRC(7) := D(7) xor D(6) xor D(2) xor C(10) xor C(14) xor C(15);
NewCRC(8) := D(7) xor D(3) xor C(0) xor C(11) xor C(15);
NewCRC(9) := D(4) xor C(1) xor C(12);
NewCRC(10) := D(5) xor C(2) xor C(13);
NewCRC(11) := D(6) xor C(3) xor C(14);
NewCRC(12) := D(7) xor D(4) xor D(0) xor C(4) xor C(8) xor C(12) xor C(15);
NewCRC(13) := D(5) xor D(1) xor C(5) xor C(9) xor C(13);
NewCRC(14) := D(6) xor D(2) xor C(6) xor C(10) xor C(14);
NewCRC(15) := D(7) xor D(3) xor C(7) xor C(11) xor C(15);

return NewCRC;

end nextCRC16_D8;

end PCK_CRC16_D8;

```

REFERENCES

1. http://www.repairfaq.org/filipg/LINK/F_LINK_IN.html
2. <http://www.joegeluso.com/software/articles/ccitt.htm>