Decoding CODA files for the Moller polarimeter

Don Jones
Dec. 2019

Decoding the binary CODA files

This was accomplished by a combination of

1. Using cefdmp on a Hall A computer to view CODA moller_data_xxxxx.dat files event by event
2. Reading the binary in hex mode in a viewing program (Emacs M-x hexl-mode)
3. Running the standalone CODA event viewer code provided by Bob Michaels http://hallaweb.jlab.org/software/tools/simpleAna_18Oct2016.tar
4. Finding snippets of explanations about CODA online such as https://hallaweb.jlab.org/equipment/daq/dstruct_year2001.html
5. Tracing the logic/action of the FORTRAN analyzer rawread.f
6. User manuals for the TDC and scaler (included in Git repos)

CODA header-- 32 bytes (8 words)

NOTE!! A CODA header event starts every block=0x8000 bytes!

```
0020 0000 0000 0000 0800 0000 0800 0000
0020 0000 0200 0000 aa00 0000 0001 dac0
```

c0da0100 in little endian

Header word definitions

1. Size of block in 4-byte words (=8192)
2. Block number starting at 0
3. Size of header in 4-byte words (=8)
4. Start of first event in this block
5. Number of words used in this block
6. Version of file format (=1)
7. Reserved
8. Magic number for error detection (=c0da0100)

Note: these headers are excluded from cefdmp output

CODA data types and event types

word_1 gives length of event in 4-byte words (not including itself)

```
0400 0000 cc01 1100 1a31 4f5d 7443 0000
0100 0000
```

word_2 encodes event type and data type. If the word =aabb ccdd (which is little endian for 0xddccbbaa)

- event type = word>>16 (or ddcc)**
- data type = (word & 0x0000ff00)>>8 (or bb)

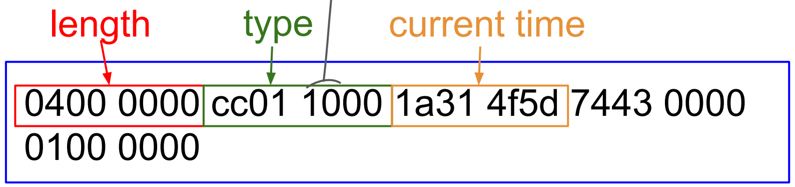
In the example above we have an event type of 0x11=17 and a data type of 0x1=1

**dd always seems to be 0

Name	Data type	Event type	Word 2	Word 2 in Hex
Sync	1	16	cc01 1000	0x1001cc
Prestart	1	17	cc01 1100	0x1101cc
Go	1	18	cc01 1200	0x1201cc
Pause	1	19	cc01 1300	0x1301cc
End	1	20	cc01 1400	0x1401cc
EPICS	16	131	cc10 8300	0x8310cc
Physics++	16	<15	cc10 0100	0x110cc

++Event types <15 are various types of triggers

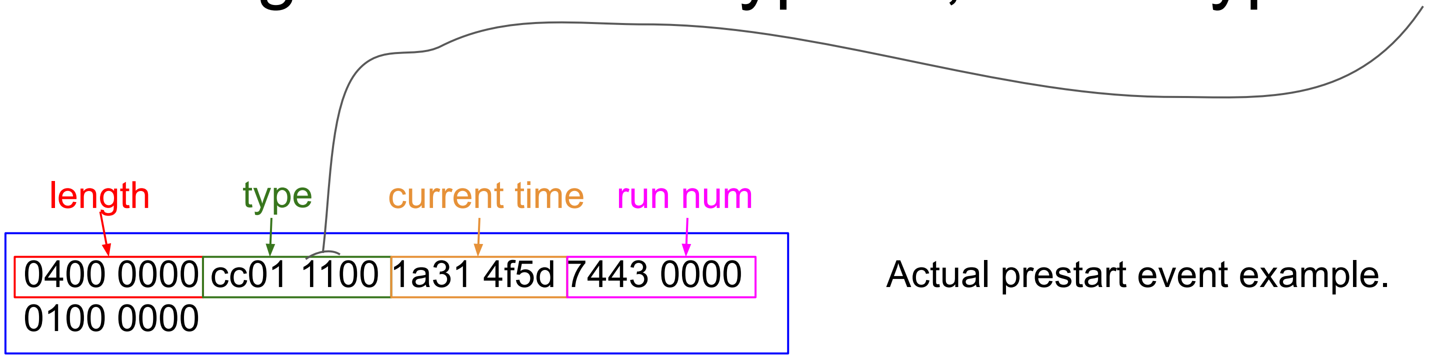
Decoding CODA data type=1, event type=16 (Sync)



```
0400 0000 cc01 1000 1a31 4f5d 7443 0000
0100 0000
```

Note! This is **not an actual sync event** example. I didn’t come across a “sync” event in the files I looked at but inferred from the kumac that the only interesting information from a sync event is the current time directly following the type word.

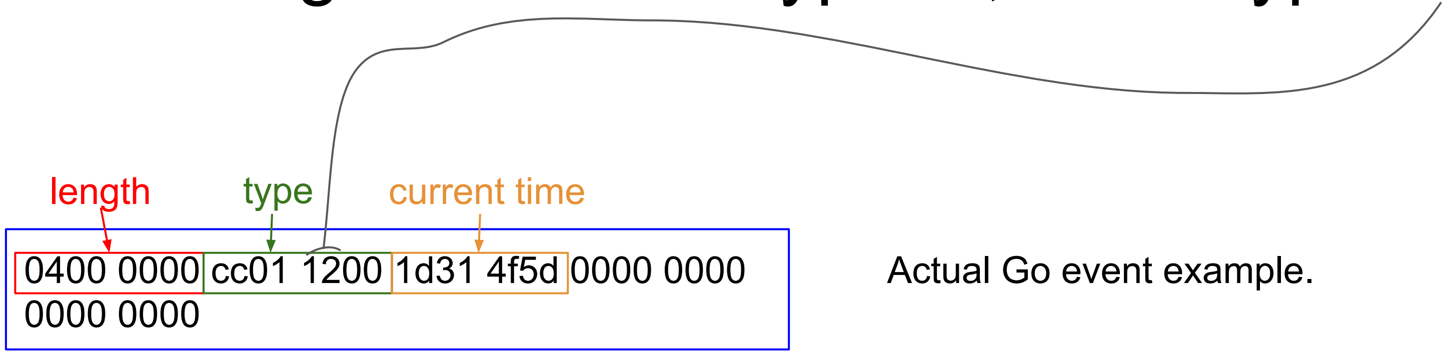
Decoding CODA data type=1, event type=17 (Prestart)



```
0400 0000 cc01 1100 1a31 4f5d 7443 0000
0100 0000
```

Actual prestart event example.

Decoding CODA data type=1, event type=18 (Go)

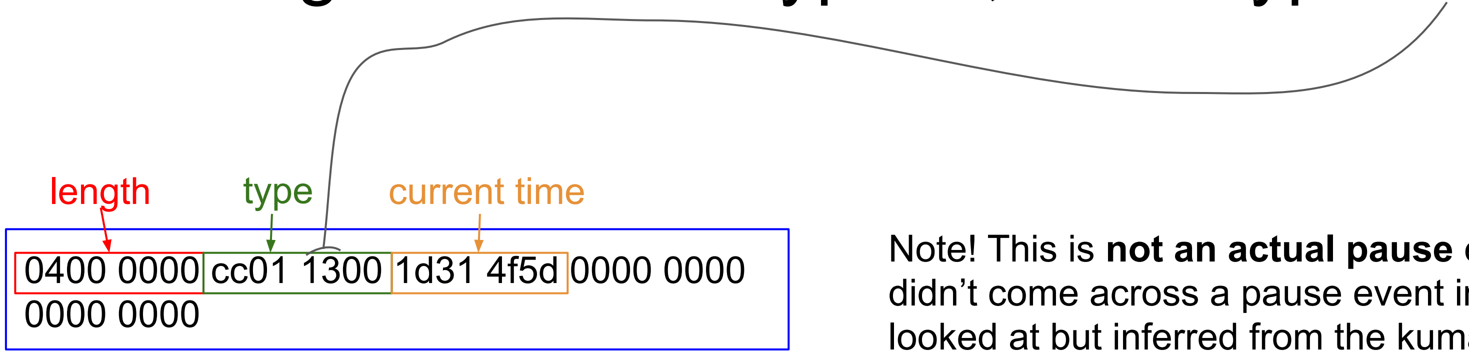


```
0400 0000 cc01 1200 1d31 4f5d 0000 0000
0000 0000
```

Actual Go event example.

The current time from the Go event is the closest we have to start of data taking time.

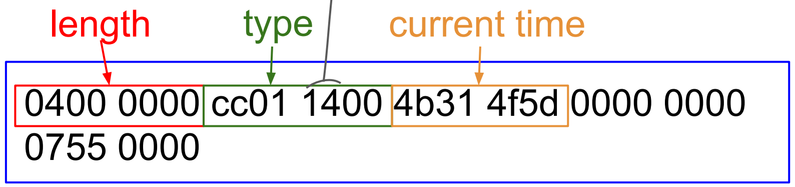
Decoding CODA data type=1, event type=19 (Pause)



```
0400 0000 cc01 1300 1d31 4f5d 0000 0000
0000 0000
```

Note! This is **not an actual pause** event example. I didn't come across a pause event in the files I looked at but inferred from the kumac that the only interesting information from a pause event is the current time directly following the type word.

Decoding CODA data type=1, event type=20 (End)



```
0400 0000 cc01 1400 4b31 4f5d 0000 0000
0755 0000
```

Actual End event example.

Note that there are still events that occur after the End event hits the datastream. **However, events after the End event are not utilized.**

Decoding CODA data type=16, event type=131 (EPICS)

Beginning of EPICS event example.

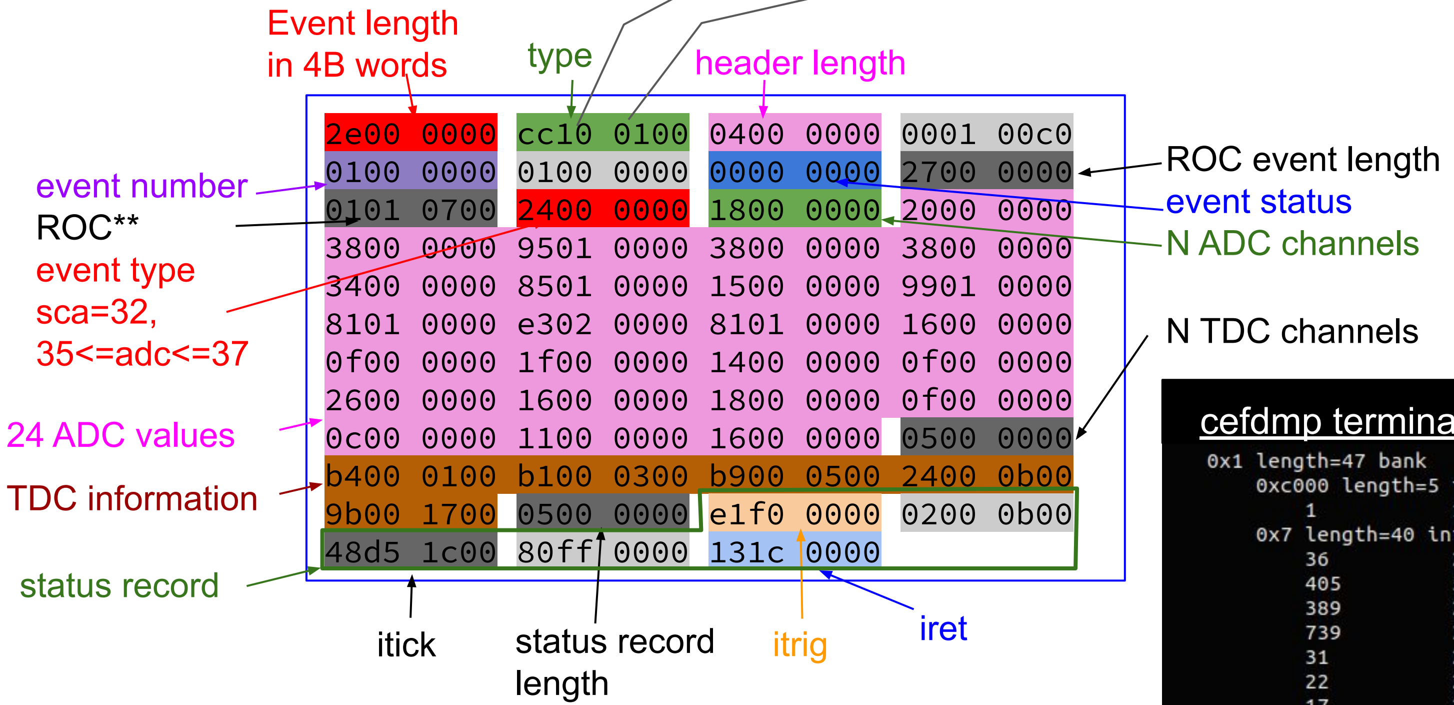
length type

```
e802 0000 cc10 8300 e602 0000 0003 0000
5361 7420 4175 6720 3130 2031 373a 3033  or  Sat Aug 10 17:03
3a35 3920 4544 5420 3230 3139 0a49 504d  or  :59 EDT 2019.IPM
3143 3230 2e58 504f 5320 2020 2020 2020  or  1C20.XPOS
2020 2020 2020 2020 2020 202d 302e 3033  or                -0.03
```

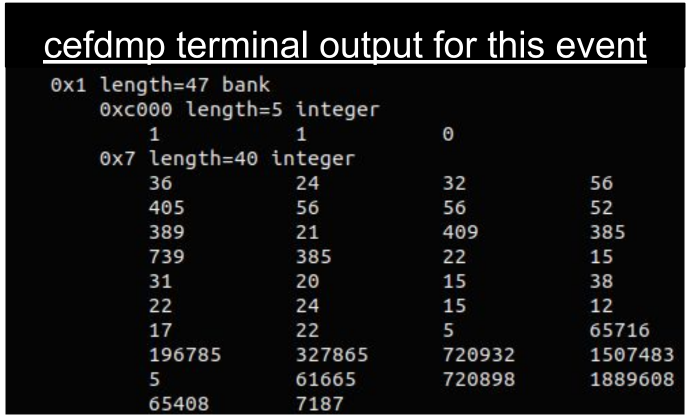
EPICS events are recorded in ASCII not binary.
No one has built a decoder for Moller EPICS events yet to my knowledge.

Decoding CODA data type=16, event type=1 (Physics)

ADC readout



```
2e00 0000 cc10 0100 0400 0000 0001 00c0
0100 0000 0100 0000 0000 0000 2700 0000
0101 0700 2400 0000 1800 0000 2000 0000
3800 0000 9501 0000 3800 0000 3800 0000
3400 0000 8501 0000 1500 0000 9901 0000
8101 0000 e302 0000 8101 0000 1600 0000
0f00 0000 1f00 0000 1400 0000 0f00 0000
2600 0000 1600 0000 1800 0000 0f00 0000
0c00 0000 1100 0000 1600 0000 0500 0000
b400 0100 b100 0300 b900 0500 2400 0b00
9b00 1700 0500 0000 e1f0 0000 0200 0b00
48d5 1c00 80ff 0000 131c 0000
```



cefdmp terminal output for this event

```
0x1 length=47 bank
    0xc000 length=5 integer
        1           1           0
    0x7 length=40 integer
        36          24          32          56
        405         56          56          52
        389         21          409         385
        739         385         22          15
        31          20          15          38
        22          24          15          12
        17          22          5           65716
        196785      327865      720932      1507483
        5           61665       720898      1889608
        65408       7187
```

**ROC number = (0x70101 & 0xff0000)>>16=7

Specific ADC event example

Note: the meaning of words marked with ‘?’ is not known but these are not utilized in the analysis chain.

event len	CODA type	headr len	?
event num	?	evnt stat	ROC len
ROC Num	Moll type	N ADC cha	ADC [0]
ADC [1]	ADC [2]	ADC [3]	ADC [4]
ADC [..]	ADC[..]	ADC [N-1]	N TDC cha
TDC [0]	TDC [1]	TDC [2]	
TDC[N-1]	stat len	trig info	?
itick	?	rtrn code	

2e00 0000	cc10 0100	0400 0000	0001 00c0
0100 0000	0100 0000	0000 0000	2700 0000
0101 0700	2400 0000	1800 0000	2000 0000
3800 0000	9501 0000	3800 0000	3800 0000
3400 0000	8501 0000	1500 0000	9901 0000
8101 0000	e302 0000	8101 0000	1600 0000
0f00 0000	1f00 0000	1400 0000	0f00 0000
2600 0000	1600 0000	1800 0000	0f00 0000
0c00 0000	1100 0000	1600 0000	0500 0000
b400 0100	b100 0300	b900 0500	2400 0b00
9b00 1700	0500 0000	e1f0 0000	0200 0b00
48d5 1c00	80ff 0000	131c 0000	

General ADC event structure

Evt Len (including header)	Evt type	Header Len=-4	?
Evt. num.	?	Evt status	ROC Len
Coda info (including ROC)	Moller evt type	N_ADC ch	ADC[0]
...	...	ADC[N_ADC-1]	N_TDC ch
TDC[0]	...	TDC[N_TDC-1]	N_TDC2 ch**
TDC2[32]	...	TDC2[N_TDC2]	Status Record Len=5
Trigger info (itrig)	?	itick	?
iret			

**This second bank of TDC information is only read out if Physics event type=37. I don’t have an example of this type of event.

TDC information: (see Scaler.pdf in Git repos)

- The first word is N_TDC
- The TDC is only 16 bit which only requires half the 32 bit word so the first 16 bits are the TDC values TDC[i] & 0xffff which for the example below gives 0xb4, 0xb1, 0xb9, 0x24, 0x9b
- The last 17th bit encodes the TDC trigger phase (ited) with leading edge = 1 and trailing edge = 0
    - ited[i] = (TDC[i]>>16) & 1, which for the first example below is (0x100b4>>16)&1 = 1
- Bits 18-22 are channel number (itch)
    - itch[i] = ((TDC[i]>>17) & 0x1f) which for the example below is 0, 1, 2, 5, 11 (Note that the FORTRAN code increments these by 1 so the channel numbering starts at 1)

```
0500 0000 b400 0100 b100 0300 b900 0500
2400 0b00 9b00 1700
```

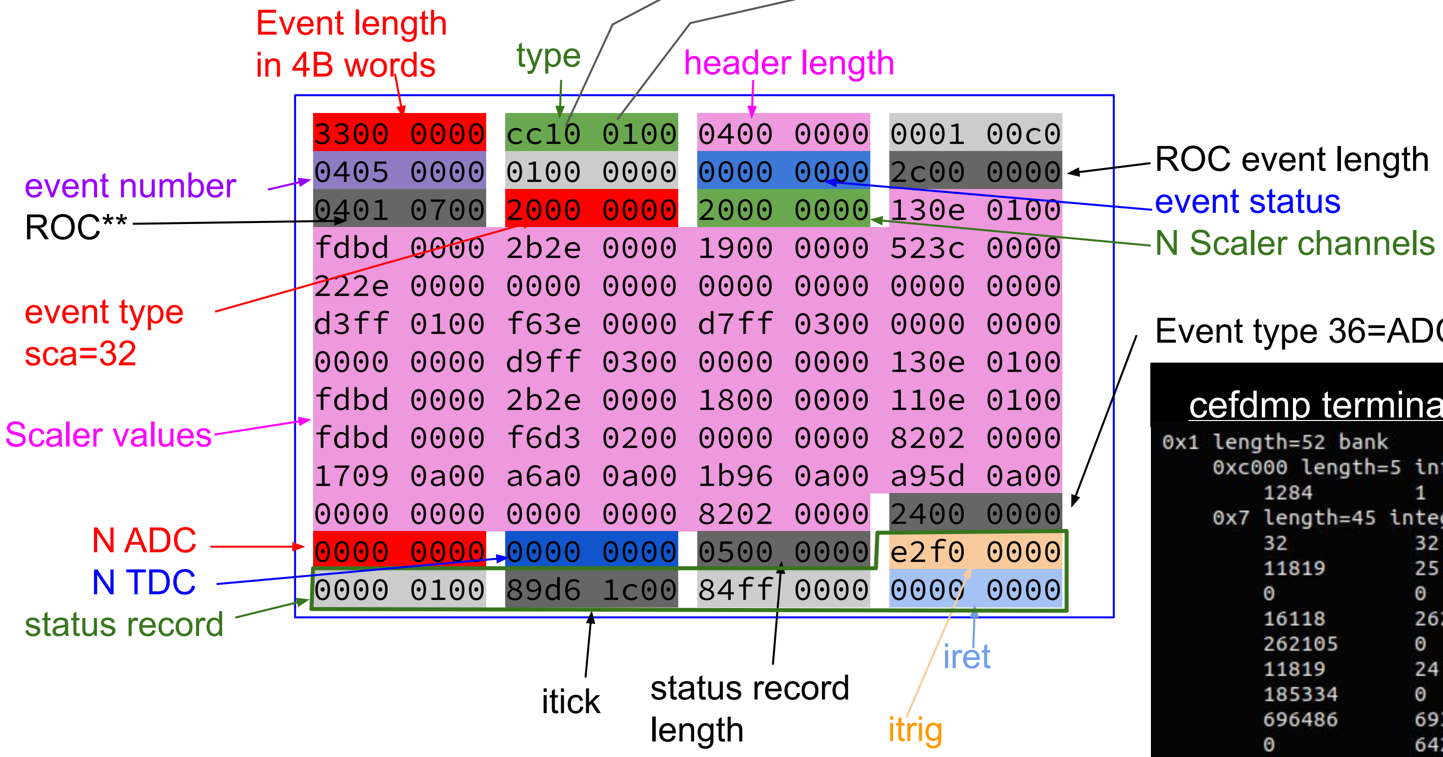
Trigger information:

- The first word of the status record is the length of the status record
- The first 8 bits of the second word is the 8 channels of itrig (either 0 or 1) encoded bitwise so 0xe1 =1110 0001 giving itrig[]={1,0,0,0,0,1,1,1}
- itick is the optional 4th word of the status record and is the number of ticks per event of the 120 Hz CPU clock
- iret is the optional 6th word of the status record and is a return/error code

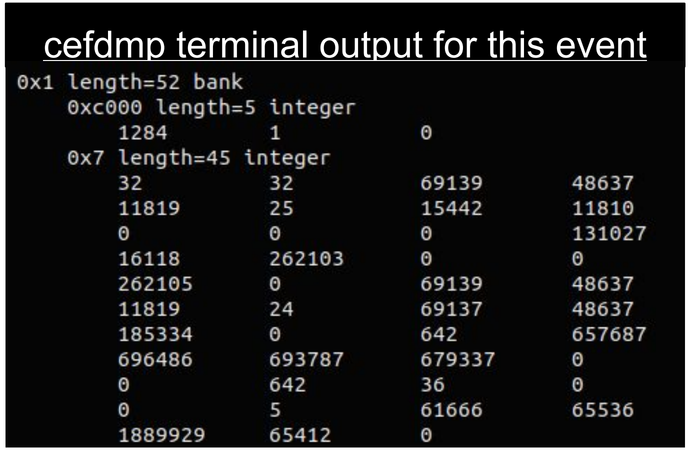
```
0500 0000 e1f0 0000 0200 0b00 48d5 1c00
80ff 0000 131c 0000
```

Decoding CODA data type=16, event type=1 (Physics)

Scaler readout



```
3300 0000 cc10 0100 0400 0000 0001 00c0
0405 0000 0100 0000 0000 0000 2c00 0000
0401 0700 2000 0000 2000 0000 130e 0100
fdbd 0000 2b2e 0000 1900 0000 523c 0000
222e 0000 0000 0000 0000 0000 0000 0000
d3ff 0100 f63e 0000 d7ff 0300 0000 0000
0000 0000 d9ff 0300 0000 0000 130e 0100
fdbd 0000 2b2e 0000 1800 0000 110e 0100
fdbd 0000 f6d3 0200 0000 0000 8202 0000
1709 0a00 a6a0 0a00 1b96 0a00 a95d 0a00
0000 0000 0000 0000 8202 0000 2400 0000
0000 0000 0000 0000 0500 0000 e2f0 0000
0000 0100 89d6 1c00 84ff 0000 0000 0000
```



cefdmp terminal output for this event

```
0x1 length=52 bank
    0xc000 length=5 integer
        1284        1           0
    0x7 length=45 integer
        32          32          69139       48637
        11819       25          15442       11810
        0           0           0           131027
        16118       262103      0           0
        262105      0           69139       48637
        11819       24          69137       48637
        185334      0           642         657687
        696486      693787      679337      0
        0           642         36          0
        0           5           61666       65536
        1889929     65412       0
```

**ROC = (0x70104 & 0xff0000)>>16=7

stat len	trig info	?
itick	?	rtrn cod

Specific scaler readout example

3300 0000	cc10 0100	0400 0000	0001 00c0
0405 0000	0100 0000	0000 0000	2c00 0000
0401 0700	2000 0000	2000 0000	130e 0100
fdbd 0000	2b2e 0000	1900 0000	523c 0000
222e 0000	0000 0000	0000 0000	0000 0000
d3ff 0100	f63e 0000	d7ff 0300	0000 0000
0000 0000	d9ff 0300	0000 0000	130e 0100
fdbd 0000	2b2e 0000	1800 0000	110e 0100
fdbd 0000	f6d3 0200	0000 0000	8202 0000
1709 0a00	a6a0 0a00	1b96 0a00	a95d 0a00
0000 0000	0000 0000	8202 0000	2400 0000
0000 0000	0000 0000	0500 0000	e2f0 0000
0000 0100	89d6 1c00	84ff 0000	0000 0000

event len	CODA type	headr len	?
event num	?	evnt stat	ROC len
ROC Num	Moll type	N Scal ch	Sca [0]
Sca [0]	Sca [0]	Sca [0]	Sca [0]
Sca [..]	Sca [..]	Sca [N-1]	Moll type
N_ADC cha	N_TDC_cha	stat len	trig info
?	itick	?	rtrn code

**ROC = (0x70104 & 0xff0000)>>16=7

General scaler event structure

Evt Len (including header)	Evt type	Header Len=4	?
Evt. num.	?	Evt status	ROC Len
ROC info	Moller evt type	N_Sca ch=32	Sca[0]
...	...	Sca[N_Sca-1]	Moller evt type=36
N_ADC=0	N_TDC=0	Status Record Len=5	Trigger info (itrig)
?	itick	?	iret

It looks like in order to include the status record for scaler events, they append an ADC+TDC+status record event with 0 channels for ADC and TDC