ZEUS CALDAQ

Transputer System

Technical Notes

Version 2.8

Henk Boterenbrood & Andres Kruse

v2.8: Apr 22, 1998
v2.7: Nov 28, 1997
v2.6: Jul 04, 1997
v2.5: Jan 15, 1997
v2.4: Jun 19, 1996
v2.3: Mar 13, 1996
v2.2: Oct 11, 1995
v2.1: Aug 15, 1995

Contents

1 Introduction

This document provides all kinds of practical information concerning the CALDAQ transputer system: configuration files, error messages, code files, network layout, some utilities[1]. All stuff described here can be found on the hosts *aritra* and *caldev*.

It does not claim to be complete ! More information to be found in the documents in the reference list.

2 Transputer Identifiers

Most transputers in the CALDAQ transputer network are identified by a unique hexadecimal number; often messages in the logfile are accompanied by this number. Table 1 lists them. 'Crate-number' is a number between 1 and 16.

Transputer name	**Identifier**	**Remarks**
FCAL TRIGGER (LAYER1)	#1010 + i	i = crate-number-1
BCAL TRIGGER (LAYER1)	#1020 + i	i = crate-number-1
RCAL TRIGGER (LAYER1)	#1040 + i	i = crate-number-1
FCAL LAYER2	#1210 + i	i = 0,1,2,3,4
BCAL LAYER2	#1220 + i	i = 0,1,2,3,4
RCAL LAYER2	#1240 + i	i = 0,1
LAYER3	#1400	
LAYER3/LAYER23-MONITOR	#1401	
FCAL READOUT	#2010 + i	i = crate-number-1
BCAL READOUT	#2020 + i	i = crate-number-1
RCAL READOUT	#2040 + i	i = crate-number-1
FCAL ROCOLLECT	#4010	
BCAL ROCOLLECT	#4020	
RCAL ROCOLLECT	#4040	
FCAL MONITOR	#8013	
BCAL MONITOR	#8024	
BCAL DATACOLLLECT	#8025	
DATACOLLLECT	#8026	
TESTGSLT	#8027	
REBOOTER	#8040	
RCAL MONITOR	#8041	
MONITOR HOST	#8042	

Table 1: CALDAQ transputers and their identifiers (hexadecimal).

[1] A separate document [1] describes the available CALDAQ transputer network debugging and test utilities.

3 Digital Cards

3.1 Distribution

Tables 2, 3 and 4 give an overview of the current number of Digital Cards in the CALDAQ-system, as well as their distribution over the different components/subdetectors that are read out by the CALDAQ-system.

FCAL Digital Cards			
Crate	**Total**	**CAL**	**FPC**
1	17	17	-
2	16	16	-
3	16	16	-
4	15	15	-
5	4	-	4
6	16	16	-
7	-	-	-
8	16	16	-
9	16	16	-
10	-	-	-
11	16	16	-
12	-	-	-
13	15	15	-
14	16	16	-
15	16	16	-
16	17	17	-

Table 2: FCAL Digital Card distribution.

The Digital Cards in FCAL crates 6, 8, 9 and 11 (with CAL regions around the beampipe) were supposed to be read out in two parts (per crate), the second part being called FCAL 'crates' 5, 7, 10 and 12. In reality this meant two 2TP-modules per 'real' VME-crate. So far this has not been implemented, because of the implementation difficulties of having two readout transputers in one VME-crate, thus as well breaking the (software-)symmetry in the system.

Options:

- implement as initially planned (requires four 2TP-modules), for improved system performance,
- add 4 VME-crates and redistribute Digital Cards (requires four VME-crates and four 2TP-modules), for improved system performance,
- up to 4 extra VME-crates can be added containing hardware for new components/subdetectors, if the readout system requires extensions (BTW: there is room for extensions in the RCAL part as well).

NB: In November 1997 an FCAL crate 5 has been added to the CALDAQ-system, containing Digital Cards of the Forward Plug Calorimeter (FPC).

BCAL Digital Cards			
Crate	**Total**	**CAL**	**Others**
1	16	16	-
2	12	12	-
3	16	16	-
4	12	12	-
5	16	16	-
6	12	12	-
7	16	16	-
8	12	12	-
9	16	16	-
10	12	12	-
11	16	16	-
12	12	12	-
13	16	16	-
14	12	12	-
15	16	16	-
16	12	12	-

Table 3: BCAL Digital Card distribution.

RCAL Digital Cards										
Crt	**Total**	**CAL**	**LED**	**LAS**	**SRTD**	**FNC**	**PRES**	**PRT**	**BPC**	**BPRES**
1	15	15 (1-15)	-	-	-	-	-	-	-	-
2	16	16 (1-16)	-	-	-	-	-	-	-	-
3	14	11 (1-11)	-	-	3 (12-14)	-	-	-	-	-
4	14	11 (1-11)	-	-	3 (12-14)	-	-	-	-	-
5	14	11 (1-11)	-	-	3 (12-14)	-	-	-	-	-
6	14	11 (1-11)	-	-	3 (12-14)	-	-	-	-	-
7	16	16 (1-16)	-	-	-	-	-	-	-	-
8	16	16 (1-16)	-	-	-	-	-	-	-	-
9	8	-	4 (1-4)	1 (5)	-	-	-	-	3 (6-8)	2 (9-10)
10[2]	-	-	-	-	-	-	-	-	-	-
11	14	-	-	-	-	-	14 (1-14)	-	-	-
12	3	-	-	-	-	2 (1-2)	-	1 (3)	-	-
13[3]	0	-	-	-	-	-	-	-	-	-
14[3]	0	-	-	-	-	-	-	-	-	-
15[3]	0	-	-	-	-	-	-	-	-	-
16[3]	0	-	-	-	-	-	-	-	-	-

Table 4: RCAL Digital Card distribution over components (between brackets the locations in the crate, with crate slots numbered from left to right).

[2]SRTD-FLT crate

[3]crate slot available for system extensions

3.2 Calibration Constants

Next follows the (OCCAM-style) structure of the 2 calibration constants blocks which are being produced for every Digital Card (from include file **calibcnst_def.inc**):

```
[4096]INT      block1, block2 :

VAL INT        Blk1.Control.Word    RETYPES  block1[0] :
VAL []INT      Pedestals.Block      RETYPES  [block1 FROM 1 FOR (136*24)] :
VAL [][136]INT Pedestals            RETYPES  Pedestals.Block :
VAL []INT      ADC.Ground           RETYPES  [block1 FROM 3265 FOR 4] :
VAL INT        pC.to.GeV            RETYPES  block1[3269] :
VAL []INT      DU.Offset            RETYPES  [block1 FROM 3293 FOR 24] :
VAL []INT      Channel.Ctrl.Word    RETYPES  [block1 FROM 3317 FOR 24] :
VAL [][2]INT   H.to.Q.High          RETYPES  [block1 FROM 3341 FOR (2*24)] :
VAL [][2]INT   H.to.Q.Low           RETYPES  [block1 FROM 3389 FOR (2*24)] :
VAL INT        HCUT                 RETYPES  block1[3437] :
VAL INT        SLECUTL              RETYPES  block1[3438] :
VAL INT        SLECUTH              RETYPES  block1[3439] :
VAL []INT      Channel.Ctrl.Word.48 RETYPES  [block1 FROM 3440 FOR 48] :
VAL INT        Blk1.Format          RETYPES  block1[4089] :
VAL INT        XOR.1                RETYPES  block1[4090] :

VAL INT        Blk2.Control.Word    RETYPES  block2[0] :
VAL []INT      Gains.Block          RETYPES  [block2 FROM 1 FOR (136*24)] :
VAL [][136]INT Gains                RETYPES  Gains.Block :
VAL []INT      ADC.Count            RETYPES  [block2 FROM 3265 FOR 4] :
VAL []INT      TOffs.High           RETYPES  [block2 FROM 3269 FOR 24] :
VAL []INT      TOffs.Low            RETYPES  [block2 FROM 3293 FOR 24] :
VAL INT        RDU.Factor           RETYPES  block2[3317] :
VAL INT        DAC.Value            RETYPES  block2[3318] :
VAL []INT      Testarray.Block      RETYPES  [block2 FROM 3319 FOR (4*24)] :
VAL [][4]INT   Testarray            RETYPES  Testarray.Block :
VAL []INT      T.Poly.Data          RETYPES  [block2 FROM 3415 FOR 3] :
VAL []INT      T.Poly.QINJ          RETYPES  [block2 FROM 3418 FOR 3] :
VAL []INT      H.Poly.Data          RETYPES  [block2 FROM 3421 FOR 4] :
VAL []INT      H.Poly.QINJ          RETYPES  [block2 FROM 3425 FOR 4] :
VAL INT        Blk2.Format          RETYPES  block2[4089] :
VAL INT        XOR.2                RETYPES  block2[4090] :
```

A more detailed listing of the constants for Calorimeter Digital Cards can be found in tables in [2]. Tables 5 and 6 show a more detailed subdivision of the constant blocks for the PRESAMPLER Digital Cards that handle 48 channels (Calorimeter Digital Cards handle the high and low gain of 24 channels).

Constant Block 1		
Word #	**Data**	**Remarks**
0	Block control word	= 1
1	Pped ref val for chan 1	scaled by 2^{11}
2-59	58 relative Ppeds for chan 1	
60	Bped ref val for chan 1	
61-68	8 relative Bpeds for chan 1	
69	Pped ref val for chan 2	
70-127	58 relative Ppeds for chan 2	
128	Bped ref val for chan 2	
129-136	8 Bpeds ref val for chan 2	
....	repeat above block 24 times (3264 words total)	scaled by 2^{11}
3265-3268	4 Ground values (1 per ADC)	scaled by 2^{11}
3269	pc $\Rightarrow$ Mips conversion factor	unused temporarily
3270-3316	-	blank temporarily
3317-3340	-	not used
3341-3436	96 H to Q constants 48 (a,b)	a scaled by 200, b by $100 * 2^{17}$
3437	HCUT	scaled by $128 = 2^7$
3438-3439	-	
3440-3487	48 Channel control words	
3488-4088	-	
4089	Block format identifier	
4090	XOR-word	XOR of the constants block

Table 5: Calibration constants block 1 of 48-chan Digital Cards.

Constant Block 2		
Word #	**Data**	**Remarks**
0	Block control word	= 2
1	Pgain ref for chan 1	(scaled by 2^{21}, but during the const loading, we have the ref*chan gain shifted 2 bits left in DSP, which means the final gain of Pgain for each channel on RAM is still 2^{21} scaled, like Bgain)
2-59	58 relative Pgains for chan 1	scaled by 2^{21}
60	Bgain ref for chan 1	not used
61-68	8 relative Bgains for chan 1	scaled by 2^{21}
....	repeat for 48 channels (3264 words total)	
3265-3268	4 ADC count (1 per ADC)	store as $(1/adcperV) * 2^{23}$
3269-3316	48 Toffs for 48 chans	$*2 + 128$
3317	-	
3318	DAC value	
3319-3414	-	
3415-3417	Time poly const (c1,c2,c3)	for data trigger $c_n/512 * 2^{23}$
3418-3420	Time poly const (c1,c2,c3)	for Qinj trigger $c_n/512 * 2^{23}$
3421-3424	H poly constants (d1-d4)	for data trigger $d_n/32 * 2^{23}$
3425-3428	H poly constants (d1-d4)	for Qinj trigger $d_n/32 * 2^{23}$
3429-3620	192 test arrays for 48 channels (baseline max/min, time max/min)	E in factor of 128 T in $*2 + 128$
3620-4088	-	
4089	Block format identifier	
4090	XOR-word	XOR of the constants block

Table 6: Calibration constants block 2 of 48-chan Digital Cards.

4 Hardware Configuration File

The hardware configuration file describes per crate the hardware present (Digital Cards and the components they belong to, SLT-transputers, other hardware) and parameters if applicable.

The hardware configuration file is an ASCII-file and is read via a symbolic link from:

/zeus/transputer/online/v*xx.y*/bin/host/hwconfig/hwconfig.dat

The actual **hwconfig.dat** file can be found in directory **~calec_rc/defaults/hwconf**.

The syntax is as described below:

```
!             crate.id 1st crate
!             total number of parameters
!             parameters
!             crate.id 2nd crate
!             total number of parameters
!             parameters
!             ...
!
! Each block of parameters:
!             key number of 1st hardware option
!             number of parameters for this option
!             parameters
!             key number of 2nd hardware option
!             number of parameters for this option
!             parameters
!             ...
!
! Key numbers are:
!
! 1 : Digital Card setup
!     format:
!          #words for this option
!          no.of.digital cards
!          component.id component.id ... component.id
!
! 2 : CAL-SLT layer1 there ?
!     format:
!           #words for this option
!           0 ! no
!     OR
!           1 ! yes
!
! 3 : CAL-SLT layer2 there ?
!     format:
!           #words for this option
!           0 ! no
!     OR
!           1 ! yes
!
! 4 : CAMAC readout setup
!     format:
!           #words for this option = number of parameters (0 = not present)
!
! 5 : SRTD-FLT readout setup
!     format:
!           #words for this option = number of parameters (0 = not present)
!           SRTD-FLT initialized by transputer: 0=no, 1=yes
!           SRTD-FLT TriggerCard mask
!           Read out scalers on ENVIRONMENTAL trigger: 0=no, 1=yes
```

Crate.id is a hexadecimal number (see chapter '*Transputer Identifiers*') defined as:

crate.id = #2000 + CAL + (crate-number - 1)

with **CAL**=#10 (FCAL) or **CAL**=#20 (BCAL) or **CAL**=#40 (RCAL).

Component.id is equal to the bit-number of the appropriate component in the bank-bitpattern.

5 Data-acquisition Configuration Files

The DAQ configuration files (***.dat** setup files) are ASCII-files and can be found in directory **~calec_rc/defaults**. A configuration file is read by the local CALDAQ RunControl at every run-startup and the data therein is sent to the CALDAQ transputer network as part of the SETUP command.

Here's an example setup file (NB: in this listing line numbers are added at the start of each line, for clarity):

```
 1:  6              /* experiment number */
 2:  6              /* run number        */
 3:  0000F5AF       /* FCAL boot    map  */
 4:  00000DFF       /* RCAL boot    map  */
 5:  0000FFFF       /* BCAL boot    map  */
 6:  0000F5AF       /* FCAL readout map  */
 7:  00000DFF       /* RCAL readout map  */
 8:  0000FFFF       /* BCAL readout map  */
 9:  600007FE       /* FCAL banks: CAL=2,SR=4,PR=8,LED/LAS=0x10/0x20,FNC=0x40,   */
10:  600007FE       /* RCAL banks  PRT=0x80,BPC=0x100,CAMAC=0x200,SRFLT=0x400,   */
11:  600007FE       /* BCAL banks  SLT1=0x800,SLT2=0x1000 (see file 'bank_bits') */
12:  17 16 16 15  0 16  0 16 16  0 16  0 15 16 16 17  /* FCAL cards/crate */
13:  15 16 14 14 14 14 16 16  8  0 14  3  0  0  0  0  /* RCAL cards/crate */
14:  16 12 16 12 16 12 16 12 16 12 16 12 16 12 16 12  /* BCAL cards/crate */
15:  50             /* triggers/configuration */
16:  300            /* requested trigger rate */
17:  19             /* run type (see file 'run_types') */
18:  0              /* not used */
19:  charge1.acf    /* CNF file */
20:  leopard.dac    /* LED CNF file */
21:  8              /* environment bits (see file 'environment_bits') */
22:  100            /* percentage of events to CALEC */
23:  4              /* data-compression option (see file 'compression_modes') */
24:  1              /* DSP code config, 0=m&s,1=time&energy,2=player,3=test */
25:  Qinj run, all components
26:  1129           /* calib constants (-1=none, 0=dummy, other=version) */
27:  0              /* CAL-SLT/polynomial constants version */
28:  2              /* to CALEC:bit0=testtriggers,bit1=fraction,bit2=all you can */
29:  00000000       /* FCAL crates force samples bitpattern */
30:  00000000       /* RCAL crates force samples bitpattern */
31:  00000000       /* BCAL crates force samples bitpattern */
```

Line 1 and 2 containing the experiment and run numbers are not significant; the CALDAQ readout system derives the run number from the BOR filename it gets from RunControl.

Line 3, 4 and 5 contain the boot mask for FCAL, RCAL and BCAL respectively; bits 0 to 15 of each mask stand for crate 1 to 16; if a crate is switched on (but not necessarily takes part in the readout) the corresponding bit has to be set to 1.

Line 6, 7 and 8 contain the readout mask for FCAL, RCAL and BCAL respectively; bits 0 to 15 of each mask stand for crate 1 to 16; if a crate is taking part in the readout the corresponding bit has to be set to 1.

Lines 9, 10 and 11 contain the socalled 'bankbit-pattern' for FCAL, RCAL and BCAL respectively, described in chapter 6 ('*Bank Bit Pattern*').

The lines 12, 13 and 14 containing the number of Digital Cards per crate are only in this file to make these numbers appear in the BOR-banks; the CALDAQ readout system extracts the number of Digital Cards from the **hwconfig.dat**

hardware configuration file, described in chapter 4 ('*Hardware Configuration File*').

Line 15 and 16 contain the number of triggers per 'configuration' (of the frontend electronics) and the requested trigger rate respectively; these numbers are only taken into account by the CALDAQ system for standalone runs; the number of configurations is determined from the electronics configuration file (see line 19).

Line 17 contains the run type, coded in a decimal number; see table 7 for a list of possible run types.

Line 19 contains the front-end electronics configuration file name; these files are described in chapter 13 ('*Front-End Electronics Configuration Files*').

Line 20 contains the LED DAC-settings file name; these files are described in chapter 14 ('*LED DAC-Settings Files*').

Line 21 contains a decimal number, the binary representation of which describes an environment: component 'parts' and/or global components which are participating in the readout of the CALDAQ system in this run:

Dec.value	**Bitmask**	**Component participating**
1	0x01	GFLT
2	0x02	GSLT
4	0x04	(Local) EventBuilder
8	0x08	CAL equipment computer (*aritra*)
16	0x10	CAL Second Level Trigger transputer network
32	0x20	BPC Second Level Trigger transputer
128	0x80	Test-GSLT transputers (replacing GSLT)

Line 22 contains the percentage of events sent to and stored on the CAL equipment computer, that is: if the 'spy'-option on line 28 is set to getting a 'fraction'.

Name	Id
DAQ Runs	
PHYSICS.MODE	1
DUMMY.MODE	2
DUTY.CYCLE.MONITORING	13
LASER	14
DUNO	15
LED.AC	16
LED.DC	17
EMPTY	18
QINJ	19
LED.QINJ	20
-[4]	21
-[4]	22
Calibration Runs	
DUNO.MONITORING	30
PEDESTALS	31
GAINS	32
DC.LINEARITY	33
Q.LINEARITY	34
LASER.PULSE.SHAPE	35
Q.PULSE.SHAPE	36
DUTY.CYCLE	37
ADC.TO.VOLTS	38
WRITE.CONSTANTS	39
Checkout Runs	
Q.CHECKOUT	40
PEDESTALS.CHECKOUT	41
DUNO.CHECKOUT	42
Calibration Runs[5]	
PEDESTALS.CONSTANTS	50
GAINS.CONSTANTS	51
DUNO.MONITORING.CONSTANTS	52
Q.LINEARITY.CONSTANTS	53
ADC.TO.VOLTS.CONSTANTS	54

Table 7: CALDAQ run types.

[4] do not use; for RunControl internal use
[5] transputers calculate the calibration constants

Line 23 contains the coded zerosuppress/compression option to be used:

Option	Description
0	No compression.
1	Zero-suppression to the CxTENE-bank; all channels that have a spark behaviour are saved; with spark.energy.cut = 500 MeV spark.imbalance.cut = 0.9(REAL32)
2	Zero-suppression to the CxTENE-bank; with energy.cut = 50 MeV
3	Compression of CxTENE-bank entries that comply to: ((0 <= left PM-energy < 32) OR (-128 <= left PM-energy < 0)) and ((0 <= right PM-energy < 32) OR (-128 <= right PM-energy < 0)) into CxPECO-bank entries, keeping the CxTENE-bank entries.
4	Same as 3, but now deleting CxTENE-bank entries that moved to the CxPECO-bank.
5	Same as 4, but now the CxPECO-bank is deleted.
6	- Compression of xxTENE-bank entries that comply to: ((0 <= left PM-energy < E_THR) OR (-128 <= left PM-energy < 0)) and ((0 <= right PM-energy < E_THR) OR (-128 <= right PM-energy < 0)) into xxCOEN-bank entries, while keeping the xxTENE-bank entries, with E_THR = 16 for PRESAMPLER and E_THR = 32 for CAL, BPC, FNC, PRT, SRTD, Table Diodes (LASER). - The BOR-event contains the xxPMNO-bank. - The xxDCCN is compressed: redundant entries (on a crate-by-crate basis) are marked by bit 15 of the first word (the ID) set.
7	Same as 6, but xxTENE-bank entries that moved to the xxCOEN-bank are deleted, as well as redundant xxDCCN entries.
8	Same as 7, but now the xxCOEN-bank is deleted and the BOR-event does not contain the xxPMNO-bank.

Line 24 contains the number of the DSP-code configuration file to use; see chapter 11 ('*DSP-code Files*') and 12 ('*DSP-code Configuration Files*').

Line 25 contains a string describing the type of run etc. (appears e.g. on the user screen of the Local CALDAQ RunControl).

Line 26 contains the calibration constants version to be downloaded; if the number is -1 no constants will be downloaded, if the number is 0 socalled 'dummy' constants will be downloaded; see chapter 8 ('*Calibration Constants Files*').

Line 27 contains the CAL Second Level Trigger constants version to be downloaded to the CAL-SLT transputers in case of physics runs or the polynomial constants version to be downloaded to the readout transputers in case of calibration runs; see chapter 9 ('*CAL-SLT Constants Files*') and 10 ('*Polynomial Constants Files*').

Line 28 contains a decimal number, the binary representation of which describes which 'spy'-option is to be used, meaning which events and the number of events taken ('spied') from the Local EVB buffer and sent to the CAL equipment computer to be stored locally on disk:

- **bit 0** set means: take all testtriggers;
- **bit 1** set means: take a fraction, the percentage of which is set in line 22;
- **bit 2** set means: take as many events as possible without disturbing the central data-acquisition.

'Take-fraction' (bit 1) overrules 'take-as-many-as-possible' (bit 2). To store all events locally, 'take-fraction' should be set to 1 and the percentage (in line 22) to 100.

Lines 29, 30 and 31 contain a mask for FCAL, RCAL and BCAL respectively, defining in which crates the samples of the Digital Cards will be forced to be generated; bits 0 to 15 of each mask stand for crate 1 to 16; if in a crate the samples should be produced the corresponding bit has to be set to 1.

The files referred to in the comment of some of the lines of the DAQ configuration file can be found in directory **~calec_rc/defaults/info_caldaq** and contain information which also can be found in this document:

- **bank_bits**: contains the definition of the bits of the 'bank-bitpattern'; see chapter 6 ('*Bank Bit Pattern*').
- **run_types**: contains a list of the decimal codes for the different run types; see table 7.
- **compression_modes**: contains a description of the data-compression options listed above.

6 Bank Bit Pattern

The 'bank-bitpattern' is a 32-bit bit mask used to enable/disable the readout of the different components individually and to enable/disable certain features, e.g. BOR/EOR bank generation, means & sigmas calculations. Table 8 shows the definition of the different bits in the 'bank-bitpattern' word (see file **bankbitpattern.inc**).

The bank-bitpattern can be set per Calorimeter in the CALDAQ-system configuration file (see chapter 5 ('*Data-acquisition Configuration Files*')).

Bit	Bit-mask	Bit-name	Description
0	0x00000001	COMPONENT.NONE	don't use
1	0x00000002	COMPONENT.CAL	CAL in run
2	0x00000004	COMPONENT.SRTD	SRTD in run
3	0x00000008	COMPONENT.PRES	PRESAMPLER in run
4	0x00000010	COMPONENT.LEDMONITOR	LED in run
5	0x00000020	COMPONENT.LASERMONITOR	LASER in run
6	0x00000040	COMPONENT.FNC	FNC in run
7	0x00000080	COMPONENT.PRT	PRT in run
8	0x00000100	COMPONENT.BPC	BPC in run
9	0x00000200	COMPONENT.CAMAC	CAMAC crate in run
10	0x00000400	COMPONENT.SRTDFLT	SRTD-FLT in run
11	0x00000800	COMPONENT.CALSLTL1	CAL-SLT layer1 in run
12	0x00001000	COMPONENT.CALSLTL2	CAL-SLT layer2 in run
13	0x00002000	COMPONENT.TEST	TEST-component in run[6]
14	0x00004000	COMPONENT.BPRES	Barrel-PRES in run
15	0x00008000	COMPONENT.FPC	FPC in run
16	0x00010000	-	-
17	0x00020000	-	-
18	0x00040000	-	-
19	0x00080000	-	-
20	0x00100000	DISABLE.SAMPLE.BANKBIT	no sample banks
21	0x00200000	-	-
22	0x00400000	-	-
23	0x00800000	-	-
24	0x01000000	-	-
25	0x02000000	-	-
26	0x04000000	-	-
27	0x08000000	NOT.CALIB.BANKBIT	no intermediate banks in calibration sequence
28	0x10000000	ZEUS.BOREOR.BANKBIT	BOR/EOR in ZEUS runs
29	0x20000000	BOREOR.BANKBIT	BOR/EOR
30	0x40000000	EVDATA.BANKBIT	event data
31	0x80000000	MSIG.BANKBIT	means & sigmas

Table 8: Meaning of the bits in the bank-bitpattern ('-' means 'not used (yet)').

[6]For any Digital Card assigned to the **TEST** component a bank containing the full DC output page will be produced; meant for Digital Card DSP code test/development.

7 Banks

7.1 Generation

In this chapter will be described how the selection of banks to be generated on the READOUT transputers takes place.

Per (sub)component and per triggertype there is a list of banks (see file **banksets.inc**) to be produced; at every startup a hardware configuration file (**hwconfig.dat**, see chapter 4 ('*Hardware Configuration File*')) is read that contains amongst other things per crate a list of Digital Cards and the component each Digital Card belongs to; from this information and the banklists the READOUT transputer compiles a banklist per triggertype containing all banks to be produced from all components present in that crate for that particular triggertype.

There are some limitations to the Digital Card distribution related to the generation of banks:

- Digital Cards in one crate belonging to one subdetector should be located next to one another.
- Digital Cards of one subdetector can only be put in more than one of the FCAL, BCAL or RCAL parts of the CALDAQ system when different bank names are defined for each of the CALDAQ system parts in which the subdetector is present.

At every startup a socalled 'bank-bitpattern' is sent along with the hardware configuration (can be set separately for FCAL, BCAL and RCAL; see chapter 5 ('*Data-acquisition Configuration File*')).

Using the 'bank-bitpattern' the generation of banks can be steered; all banks of a certain component may be 'switched off' (i.e. its banks are not produced) or certain bank 'types' or 'classes' (e.g. *BeginOfRun banks*) may be enabled/disabled; see chapter 6 ('*Bank Bit Pattern*').

For every bank in the compiled banklists the required bank-bitpattern is compared to the received value and generation of the bank is disabled if the required bank-bitpattern is not a subset of the received value (this applies only to the bits in the bank-bitpattern that steer the generation of bank classes).

The most important constants governing the generation of banks:

- **COMPONENT.BANKSET.LIST**: a two-dimensional list of banklist identifiers (indices into array **BANKSET.BANKLIST**) ordered by component and by triggertype (in file **banksets.inc**).
- **BANKSET.BANKLIST**: a list of banklists ordered by banklist identifier (in file **banksets.inc**).
- **BANK.CONTROL**: the required bank-bitpattern for every possible bank can be found (in file **bankcontrol.inc**).

Adding a new bank(set) or component involves additions/changes to several files in the **include**, **readout**, **libs** and **test** subdirectories in **/zeus/transputer/online/v***xx.y***/src/** (CALDAQ transputer code version *xx.y*).

Appendix B shows a complete list with short descriptions of all additions and modifications needed to incorporate a new Digital Card based subdetector in the CALDAQ readout system.

The most important include files are:

- **bankbitpattern.inc**: contains the definitions of the bits in the bank-bitpattern
- **bankcontrol.inc**: contains a list of required bank-bitpatterns, one per bank
- **banks.inc**: contains bank indices and per bank the number of columns
- **banks_info.inc**: contains per bank info about the component it belongs to, the amount of data associated with it and possible alternative banks (per event only one of the alternative banks can be generated); this info is used by the eventsize calculation tool described below
- **banks_maxsize.inc**: contains per bank and per calorimeter a maximum number of rows to avoid event sizes too large to be handled upstream by the EVB/TLT (e.g. if samples are forced in all crates)
- **banksets.inc**: contains lists of banks to be generated per component and per triggertype
- **zebra_decl.inc, zebra_init.inc**: contains definitions and lists of hollerith constants used by the ROCOLLECT transputer to build the banks in ZEBRA-format.

7.2 Tools for Checking

For every CALDAQ transputer code version **xx.y**, in directory
/zeus/transputer/online/v*xx.y***/src/test**,
a tool can be found to check the integrity of several of the lookup tables in the above mentioned include files. The program is run by typing '**run_testtables**'. (NB: check if the tool is up-to-date with the readout code by typing '**make -f testtables**').
To be used every time when new components and/or banks are added to the system.

In the same directory a tool can be found to display for every crate for every triggertype the banks generated and the sizes of the events produced; at the same time it checks the configuration file syntax and checks whether buffer sizes in the readout system are big enough to contain all bank data; the bank listings produced provide a way to check if newly added banks are generated for the right triggertype in the right crate and if the right banks are generated for newly added components or Digital Cards.

The program reads the hardware configuration from file
~calec_rc/defaults/hwconfig.dat
(through a symbolic link) and uses the component banklists of the current transputer code version.
(NB: check if the tool is up-to-date with the readout code by typing '**make -f check_readout**').
To be used every time when new components and/or banks are added to the system.

The program is run by typing '**run_check_readout** <*option*>'
from directory **/zeus/transputer/online/v***xx.y***/src/test**; when no option is given the following information is displayed;

- the maximum size of an event generated which possibly is generated in any of the crates of the readout (one number)

- the total size of the data received on the ROCOLLECT transputer per calorimeter and per triggertype
- the sizes of the event passed on to the LocalEVB transputer per calorimeter and per triggertype
- any problems with the configuration file syntax or buffer size limitations are reported when they occur

The available options are **f**, **b**, **r** or **a**; when one of these options is given the following extra information about the FCAL, BCAL, RCAL or all calorimeter crates respectively is displayed, per crate:

- component bitpattern (see chapter 6 ('*Bank Bit Pattern*'))
- the number of Digital Cards in the crate, listing which Digital Card belongs to which component
- parameters for some of the components (if present)
- lists of banks generated, per triggertype
- size of event data, per triggertype

8 Calibration Constants Files

Digital Card calibration constants file names are read from a file in directory **/data/calib/dummy** (in case socalled 'dummy' constants are used) or **/data/calib/v***nnnn*, named *x***cal_calib.files**, in which *nnnn* is a 4-character version number and *x*= **f**, **b** or **r** (for FCAL, BCAL and RCAL respectively).

There is one calibration constants file per crate. The files are in ZEBRA-format.

The version number used is the number on the 26th line of the DAQ configuration file (the line with the comment: *calib constants*).

If the version number is -1 no calibration constants are downloaded to the Digital Cards.

9 CAL-SLT Constants Files

For runs using the CAL-SLT (physics runs) SLT constants are read from a file in directory **/calcon/cslt**, named **sltconstv***nnnn***.fz**, in which *nnnn* is a 4-character version number. The file is in ZEBRA-format.

The version number used is the number on the 27th line of the DAQ configuration file (the line with the comment: *CAL-SLT/polynomial constants version*).

10 Polynomial Constants Files

For calibration runs polynomial constants are read from a file in directory **/calcon/calib/poly**, named **bfrv***nnnn***.poly**, in which *nnnn* is a 4-character version number. The file is in ZEBRA-format.

The version number used is the number on the 27th line of the DAQ configuration file (the line with the comment: *CAL-SLT/polynomial constants version*).

11 DSP-code Files

During the boot procedure of the CALDAQ system several code-files are stored in the memory of the REBOOTER transputer (code for DATACOLLECT transputers, ROCOLLECT transputers, READOUT transputers, LAYER1 TRIGGER transputers, TESTGSLT transputers and the Digital Card DSPs), and stay there as long as the CALDAQ system is running; if individual transputers in the CALDAQ system have to be rebooted the code which is stored in the REBOOTER's memory is used.

This means that a code change for any of these transputers or the DSPs always requires a full reboot of the CALDAQ system in order for the new code to be used.

DSP-code files are read via symbolic links in directory:

/zeus/transputer/online/v*xx.y***/bin/readout**

The link names are **daq_***x***.dsp**, with 0<=*x*<=9 or a<=*x*<=f, so that a total of 16 different DSP-codes can be used at the same time.

When adding a DSP-code or changing a DSP-code it is necessary to check/update the 'DSP.CODE.DESCR' array in include file **dsp_codes.inc**, describing some properties of the DSP-code.

Which DSP-code is booted on which Digital Card DSP is controlled by a DSP-code configuration file which is read via a symbolic link to directory:

/zeus/transputer/online/v*xx.y***/bin/host/dcconf**

The actual files can be found in directory **~calec_rc/defaults/dcconf** and are named **dcconf_*n*.conf**, with 0<=*n*<=16, so that a total of 16 different DSP-code configurations are possible.

Which **dcconf_*n*.conf** is used is determined by the number on the 24th line of the DAQ configuration file (the line with the comment: *DSP code config*).

12 DSP-code Configuration Files

The DSP-code configuration files are the '**dcconf_*n*.conf**' files mentioned in the previous chapter.

A DSP-code configuration file is a simple ASCII-file containing 17*16*3 numbers, 17 entries per crate, 16 per crates per calorimeter, 3 calorimeters (FCAL, BCAL and RCAL); each entry denotes the DSP-code version to be used to boot one Digital Card, ordered per VME-crate from left to right, and per calorimeter from crate 1 to crate 16; e.g. if the number is 3 then the code pointed to by **daq_3.dsp** will be used (see chapter 11 ('*DSP-code Files*')).

13 Front-End Electronics Configuration Files

The control signals for the calorimeter frontend electronics ('NEVIS Electronics') are generated by a number of NIM modules; these modules are configured through serial links coming from 4 *Serial Cards* in the F/RCAL Subsystem VME-crate; settings of the F/R/BCAL Analog Cards are also downloaded via these cards.

In ZEUS the *Serial Card* channels are connected as follows:

Serial Card 1
chan 0 ⇒ -
chan 1 ⇒ Control Fanout FCAL
chan 2 ⇒ Analog Cards FCAL
chan 3 ⇒ Analog Cards FCAL
chan 4 ⇒ Table Card
chan 5 ⇒ Pipeline Controller
chan 6 ⇒ Format Card
chan 7 ⇒ Pulser Card

Serial Card 2
chan 0 ⇒ -
chan 1 ⇒ Control Fanout RCAL
chan 2 ⇒ Analog Cards RCAL
chan 3 ⇒ Analog Cards RCAL
chan 4 ⇒ -
chan 5 ⇒ -
chan 6 ⇒ -
chan 7 ⇒ -

Serial Card 3
chan 0 ⇒ -
chan 1 ⇒ Control Fanout BCAL
chan 2 ⇒ Analog Cards BCAL
chan 3 ⇒ Analog Cards BCAL
chan 4 ⇒ -
chan 5 ⇒ -

chan 6 ⇒ -
chan 7 ⇒ -

Serial Card 4 (duty-cycle settings)
chan 0 ⇒ Clock fanout modules
chan 1 ⇒ -
chan 2 ⇒ -
chan 3 ⇒ -
chan 4 ⇒ Start/Stop Calorimeter clock (??)
chan 5 ⇒ -
chan 6 ⇒ -
chan 7 ⇒ -

The *Serial Cards* are controlled by the HOST-transputer which downloads the configuration data to the cards. Downloading of configuration data can be disabled (at CALDAQ startup) by setting an environmental variable **USESERIALCARDS** on the host computer to **FALSE**.

The configuration files are read via a symbolic link to directory:
/zeus/transputer/online/v*xx.y*/bin/host/cnf

The actual configuration files can be found in directory **~calec_rc/cnf**, in the form of ASCII-files (***.acf** files) or binary files (***.cnf** files).

Once at every startup of the CALDAQ system the following configuration files get downloaded:

- **table.cnf** (to Serial Card 1)
- **pulser.cnf** (to Serial Card 1)
- **format.cnf** (to Serial Card 1)
- **duty.cnf** (to Serial Card 4)

At every start of a run, configuration data is downloaded which is read from a file whose name is taken from the 19th line of the DAQ configuration file (the line with the comment: *CNF file*). The same configuration data is downloaded to each of the *Serial Cards* 1, 2 and 3, corresponding to FCAL, RCAL and BCAL respectively.

(NOTE: because this data can also contain settings for the NIM modules connected to *Serial Card* 1 channels 4 to 7, nothing can be connected to these same channels on *Serial Cards* 2 and 3, although they are available...).

(NOTE: in case of run types DUTY.CYCLE.MONITORING and DUTY.CYCLE configuration data gets only downloaded to *Serial Card* 4; moreover preceeding the configuration data from the above-mentioned file, data from file **duty_cycle_mon_ch.acf** or **duty_cycle_ch.acf** respectively is downloaded).

At every abort-run or end-of-run configuration data from file **endrun.cnf** is downloaded (to *Serial card* 1 only).

A configuration data ASCII-file consists of one or more configurations, each configuration consisting of one comment line and 8 lines of data, one line per *Serial Card* channel; which *Serial Card* the data is downloaded to is decided by the HOST-transputer and depends on the specific configuration file being read.

The syntax of a data-line is:

repeat-count #databytes <databyte> <databyte> .....

The repeat-count signifies how many times the list of databytes have to be downloaded to the particular *Serial Card* channel.

Example of a configuration:

```
-- charge 1
  0    0
  1    1  255
 20    3   63   63  166
 20    3   63   63  166
  1    2   50   64
  1    4   90    0    0  192
  0    0
  0    0
```

In this example no data is downloaded to *Serial Card* channels 0, 6 and 7, one word (255) is downloaded to channel 1, three bytes are downloaded 20 times to channels 2 and 3.

More detailed information on the meaning of the bytes and bits downloaded to the different cards of the control electronics can be found in [3].

14 LED DAC-Settings Files

The LED DAC-settings file is read via a symbolic link to directory:

/zeus/transputer/online/v*xx.y*/bin/host/leopard

The actual files can be found in directory **~calec_rc/leopard**. The tool to create the LED DAC-settings file (called **leomaker**) and its documentation can be found in the same directory.

A DAC-settings file is an ASCII-file with lines with the following syntax:

<mux-number> <LED-box> <LED-number> <DAC-setting>

with 1<=mux-number<=4 (FCAL right (south), FCAL left (north), RCAL right (south), RCAL left (north) resp.), 0<=LED-box<=12, 0<=LED-number<=3 and 0<=DAC-setting<=255.

Which DAC-settings file is used is determined by the filename on the 20th line of the DAQ configuration file (the line with the comment: *LED CNF file*).

15 SRTD-FLT Configuration Files

The SRTD-FLT configuration file is read via a symbolic link from:

/zeus/transputer/online/v*xx.y*/bin/host/hwconfig/srtd_flt.zeus

for physics runs and from

/zeus/transputer/online/v*xx.y*/bin/host/hwconfig/srtd_flt.qinj

for other runs.

The actual files can be found in directory **~calec_rc/defaults/hwconf**. A tool is available to create these configuration files (refer to the SRTD-FLT experts...).

An SRTD-FLT configuration file contains 5 structures: 4 for each sector and 1 for the EVB/FLT-Card. Each sector structure consists of 3 structures for the sector's 3 Trigger Cards. Each Trigger Card structure has 80 bytes of data. The EVB/FLT-Card structure has 272 bytes of data.
The Trigger Card structure is as follows:

Byte	**Contents**	**Remarks**
0-3	address	card's VME-address
4-51	threshold chan 0-23	bits 0-7=low, 8-15=high
52-53	enable chan 0-15	bitpattern for 16 channels
54-55	enable chan 16-23	bitpattern for 16 channels
56-67	delays	6-bit delays, stored on byte boundaries
68-69	status	for status register
70-75	fifo	for FIFO control registers
76-79	pad bytes	

The EVB/FLT-Card structure is as follows:

Byte	**Contents**	**Remarks**
0-3	address	card's VME-address
4-5	offsets	for offset registers
6-15	control	EVB/FLT control signals
16-143	linear chan 0-63	timing lookup tables sectors 0,1
144-271	linear chan 0-63	timing lookup tables sectors 2,3

And the full structure of card configurations:

Byte	**Contents**
0-79	Sector 0 Card s_0
80-159	Sector 0 Card s_1
160-239	Sector 0 Card l
240-319	Sector 1 Card s_0
320-399	Sector 1 Card s_1
400-479	Sector 1 Card l
480-559	Sector 2 Card s_0
560-639	Sector 2 Card s_1
640-719	Sector 2 Card l
720-799	Sector 3 Card s_0
800-879	Sector 3 Card s_1
880-959	Sector 3 Card l
960-1271	EVB/FLT Card

16 PM-number Files (Cable Files)

The socalled cable files or PM-number files contain tables with channel numbers ordered according to their position on the Digital Cards in the VME-crates. There is a file for each of the components utilizing Digital Cards (except LED which sends its numbers via *RPN-logic files*):
CAL, SRTD, PRES, BPC, FNC, LASERMONITOR, PRT and BPRES.

Each component can thus use its own numbering scheme. The numbers appear e.g. in the time/energy banks and sample banks.

The files are accessed through symbolic links in directory

/zeus/transputer/online/v*xx.y*/bin/host
and are named **calcable.dat, srtdcable.dat, prescable.dat, bpccable.dat, fnccable.dat, lasermonitorcable.dat, prtcable.dat, bprescable.dat** respectively.

The actual files are in directory
/zeus/transputer/online/v*xx.y*/bin/cablefiles
The files are in ZEBRA-format and contain one bank per crate containing Digital Cards for that particular component.

The ZEBRA-files are generated from ASCII-files containing the PM-number tables; the tool named **cable2zebra** (just type **cable2zebra** to run the tool) and the original ASCII-files can be found in directory
/zeus/transputer/tools/cable2zebra

The format of the ASCII-files is as follows:

```
<crate-id> <number.of.PM.numbers>
<PM-no0> <PM-no1> <PM-no2> <PM-no3> <PM-no4> <PM-no5> <PM-no6> <PM-no7>
<PM-no8> <PM-no9> ...........

<crate-id> <number.of.PM.numbers>
<PM-no0> <PM-no1> ...........

etc. etc.
```

in which *crate-id* is a hexadecimal number, written in decimal and defined as:
crate.id = #2000 + CAL + (crate-number - 1)
with **CAL**=#10 (FCAL) or **CAL**=#20 (BCAL) or **CAL**=#40 (RCAL).

As an example follows the LASERMONITOR PM-number ASCII-file (1 Digital Card (24 channels) in RCAL crate 9 (crate-id = 8192 + 64 + 9 - 1 = 8264); the channels which are zero are unused channels:

```
 8264  24
   0   128   129   130   131   132     0   133
 134   135   136   137     0     0     0     0
   0     0     0     0     0     0     0     0
```

17 RPN-logic Files

Some components required to be able to design and use their own formulas for time and charge reconstruction from the samples received from the Digital Cards (although this can now also be done by writing a new DSP-code and booting this code on the appropriate Digital Cards). A provision was made in the CALDAQ-system to make this possible.

To reconstruct time and charge in this way is a slow process (typical maximum readout rate 5 Hz) and is meant only for testtriggers and standalone runs (e.g. 'means-and-sigma' runs). A full description can be found in [6].

For the LED-component RPN-logic file names are read from a file in directory **~calec_rc/leopard**, named **rpnlogic.files**. The first file in **rpnlogic.files** contains a list of formula identifiers, 10 per channel (channels in the order they appear on the Digital Cards in the VME-crate), the first 5 of which are for time reconstruction and the other 5 for charge reconstruction, ordered by testtrigger type as follows:

Formula#	*TriggerType*
0	Empty
1	Qinj

2	LED (run-type != LED.QINJ)
3	LED (run-type = LED.QINJ = 20)
4	LASER

The rest of the files in the list in **rpnlogic.files** contain the reconstruction formulas, one formula per file, the formula identified by its position in **rpnlogic.files** (first file contains formula 0, second file contains formula 1, etc.). All files listed in **rpnlogic.files** are in ZEBRA-format.

18 BOR Files

At the start of a run the following files are downloaded to the CALDAQ network:

- **/data/calib/uno/unov***nnnn***.fz**, in which *nnnn* is a 4-character version number (this UNO BOR data ends up in the CxBU-banks; only downloaded for physics runs),
- **/data/calib/srtd/srpctomipv***nnnn***.fz**, in which *nnnn* is a 4-character version number (these SRTD pC-to-MIP conversion factors end up in the SRBM-bank; only downloaded for physics runs, *NB: was never used, now taken out of the code*,
- **/calcon/calib/poly/bfrv***nnnn***.poly**, in which *nnnn* is a 4-character version number (these polynomial constants for calibration constants calculations; only downloaded for calibration runs),
- **/zeus/data/CBOR***n***.X** (actually it is the name provided by the CALDAQ RunControl program in the **ACTIVATE** command message), in which *n* is the run number (this BOR data ends up in the CxBO-banks).

19 Online Log File

The CALDAQ transputer network logfile (named **iserver.log**, which can be found in directory ~**calec_rc/log**) contains mostly messages of the format:

<**sender-name**>**:** <**message**>

in which **sender-name** is the name of the process, routine or transputer sending the message.

A complete description of the messages present in the logfile is not available...; why and where the messages are generated can be found in the source code and is mostly a matter of concern to the code expert. However a few message types will be explained here in more detail.

Messages starting with '**###**' signify that something serious or possibly fatal occurred.

Error messages originating from the CSBs (name of the message sender in this case: **CSBREPORT**) are worth noting; some of these messages are accompanied by a more explanatory message on the RunControl screen. Here is a list of these messages:

- ◇ **ARE1.ERROR mask=**<n>
  ARE2.ERROR mask=<n>
  ARE3.ERROR mask=<n>
  CSB received an interrupt from ARE1-, ARE2- or ARE3-board respectively, with the source(s) of the interrupt in bitmask *n*, meaning that the

connected transputer(s) set the error flag (but did not necessarily halt, if the code was compiled in undefined mode, e.g. the CAL-SLT code); which transputer is connected to which ARE-connection can be found in chapter '*CSB Connections*',

- ◇ **could not send NEXT TRIGGER**
  serious problem with process that controls NEVIS frontend control electronics (on HOST-transputer, via Serial Cards); it does not accept the command to generate the next trigger (in standalone runs),

- ◇ **CSB.MESS.FAIL dest=**<n>
  the CSB failed to pass on a message to the next CSB (destination id = *n*),

- ◇ **CSB.UNKNOWN.COMMAND cmd=**<n>
  the CSB received an unknown command *n* from HOST,

- ◇ **CSB.UNKNOWN.DESTINATION dest=**<n>
  the CSB received an unknown destination transputer number *n* from HOST,

- ◇ **EVT1.EVENT mask=**<n>
  EVT2.EVENT mask=<n>
  EVT3.EVENT mask=<n>
  CSB received an (unexpected) interrupt from EVT1-, EVT2- or EVT3-board respectively, with the source(s) of the interrupt in bitmask *n*; a transputer in panic can draw attention this way...; which transputer is connected to which EVT-connection can be found in chapter '*CSB Connections*',

- ◇ **EVT1.TIMEOUT tp.id=**<n>
  EVT2.TIMEOUT tp.id=<n>
  EVT3.TIMEOUT tp.id=<n>
  CSB timed out on an expected event (interrupt) from the EVT1-, EVT2- or EVT3-board respectively, from transputer with identifier *n*,

- ◇ **LKC1.OINT.TIMEOUT mask=**<n> **bytes.sent=**<m>
  LKC2.OINT.TIMEOUT mask=<n> **bytes.sent=**<m>
  one or more transputers connected to the LKC1- or LKC2-board respectively did not accept a byte sent via the LKC-board; bitmask *n* shows in the bits which are NOT 1 which connected transputer did not accept; the number of bytes of the message sent via LKC before the failure occurred is *m*.

- ◇ **LKC.UNKNOWN.COMMAND cmd=**<n>
  CSB received an unknown message byte *n* via one of its LKCs,

- ◇ **LKS1.CONFIG.FAIL**
  LKS2.CONFIG.FAIL
  LKS3.CONFIG.FAIL
  CSB failed to configure its LKS1-, LKS2- or LKS3-board because of a configuration link communication problem,

- ◇ **LKS.REQUEST.DISABLED (Warning!)**
  CSB received a request for an LKS-link connection, but the permission to use this link is not yet given (by the HOST-transputer); this warning might occur in the network startup phase; messages are queued and should appear in the logfile as soon as the LKS-links are enabled later on in the startup procedure,

- ◇ **READOUT.OK ackn failed**
  READOUT.NOT.OK ackn failed
  communication problem with CSB-message receiver process and the main process (both processes run on the HOST-transputer),
- ◇ **TRP.ERROR1**
  device connected to the TRP-ARI connector set its error flag,
- ◇ **TRP.ERROR2**
  device connected to the TRP-ARO connector set its error flag,
- ◇ **TRP.NONEMPTY.EVTREG after init: trp.requestin.reg=**<n>
  the event register of the TRP-board is not zero after initialization (as it should be),
- ◇ **TRP.UNEXP.EVENT expected=**<n> **received=**<m>
  the TRP-board received an unexpected interrupt or interrupts; n and m are interrupt bitmasks (see [5] for their definition),
- ◇ **unknown CSB source**
  the CSBREPORT process received a message with an unknown sender identifier; unlikely that this will ever happen, but if it happens it is serious because it means there probably is a CSB hardware problem,
- ◇ **unknown message tag** <n>
  the CSBREPORT process did not understand message identifier n it received from the CSB (it wasn't any of the identifiers described below).

Error messages originating from different processes on the READOUT transputers are also sent via the CSB (name of the message sender in this case also: **CSBREPORT**), to enable notification on the RunControl screen, are worth noting; these messages are accompanied by a more explanatory message in the logfile, written there directly by the READOUT transputer (through its monitor link).

- ◇ **CAMAC.INIT.ERROR**
  initialization of the CAMAC hardware (for LASER in RCAL crate 9) failed (procedure is cccz, ccci, cccc, cclm (ADC, N=2), cclm (TDC, N=5)),
- ◇ **CAMAC.STATUS.ERROR**
  error occurred while reading out data from CAMAC for LASER (one or more of: LAM timeout ADC (N=2), data read error ADC (N=2), LAM timeout TDC (N=5), data read error (N=5)),
- ◇ **CALIB.XMIT.ERROR**
  an XOR-checksum error occurred in the downloading of calibration constants blocks from host to READOUT transputer (compare to **DC.XOR.ERROR**),
- ◇ **CALIB.CNST.MISSING**
  calibration constants for one or more of the Digital Cards in the crate are missing from the download from host to READOUT transputer,
- ◇ **DC.BRC.BOOT.FAIL**
  booting the Digital Cards in a crate by broadcast method failed,
- ◇ **DC.DATA.MISMATCH**
  a mismatch was found between the DSP calculated time and energy sums and the transputer calculated sums (the check is performed on a regular basis during runs for CAL Digital Cards only),

◇ **DC.DATA.TIMEOUT**
a timeout occurred while waiting for an event to appear in the Digital Card DPM (although the GSLT-decision has been received already),

◇ **DC.DOWNLOAD.FAIL**
downloading of one or more blocks of calibration constants to one or more Digital Cards failed,

◇ **DC.GLOBALEXEC.FAIL**
giving the Digital Card *exec* command failed (during means&sigmas readout in calibration runs),

◇ **DC.GLOBALREAD.FAIL**
setting the Digital Card *read* flag failed (during means&sigmas readout in calibration runs),

◇ **DC.HEADER.ERROR**
a mismatch occurred between the headerword of the first Digital Card in the crate and another in this crate,

◇ **DC.PAGENO.ERROR**
the page number from a Digital Card page header does not match the page number read from the Digital Card OFDR,

◇ **DC.PAGENO.ORDER**
the page number read from the Digital Card OFDR does not match the expected number,

◇ **DC.PARITY.ERROR**
a parity error occurred on the Digital Card for this event (the least significant bit of the control byte (byte 3) of the header word is set),

◇ **DC.XOR.ERROR**
the Digital Card reports a mismatch between the XOR-checksum(s) of calibration constants blocks received and the one(s) calculated by the DSP,

◇ **EVENT.TOO.BIG**
an event was generated with a size larger than the available buffer size (this can only happen if the code is compiled in undefined mode),

◇ **GSLT.BUFFER** <description>
the GSLT buffering process detected a corrupted GSLT-decision; the nature of the corruption is explained in *description*,

◇ **GSLT.DC.MISMATCH**
a mismatch was found between the FLT-number or the triggertype provided by the Digital Card and the FLT-number or triggertype provided by the GSLT-decision,

◇ **HWPARAMS.ERROR**
an error was detected in the hardware parameters received from the host,

◇ **POLY.CONST.ERROR**
an error was detected in the format of the downloaded polynomial constants (needed for calibration runs),

◇ **RO.DATASEND.FAIL**
the sender process is trying now for about 30 seconds to send event data to the ROCOLLECT transputer,

- ◇ **TWOTP.BERRL.EVT**
  the transputer detected VME-bus errors,
- ◇ **TWOTP.TIMEOUT.EVT**
  the transputer detected refresh timeout errors (probably caused by not getting VME access).

At regular intervals during a run a STATUS.RUN command is given; when the CAL-SLT is taking part in the run and the 'status' hasn't changed in between two STATUS.RUN commands the status of the CAL-SLT is printed in the logfile.

For a READOUT transputer the status printed looks like this (example):

```
TP.ID #201D #000031F7 #000031EC #000031EC #00000000
           #00000546 #00000000 #00000091
```

in which #201D is the transputer identifier; the 7 numbers following are respectively:

1. the FLT-trigger number processed by the **read.trigger.data()** process
2. the FLT-trigger number of the last GSLT-decision received by the **gslt.decision.buffer()** process
3. the FLT-trigger number of the last GSLT-decision processed by the **read.cal.data()** process
4. the total number of polls necessary in this run while waiting for Digital Card data while the GSLT-decision was received already
5. the GSLT-trigger number of events sent by the **send.data()** process to the ROCOLLECT transputer
6. the total number of times in this run a timeout or transmission error occurred while sending from READOUT to ROCOLLECT transputer, when using the 'SECURE' send option in **sender.opp**
7. the total number of times the **send.data()** process has had to wait for permission to send an event to the ROCOLLECT transputer in this run

For a LAYER1 trigger transputer the status printed looks like this (example):

```
TP.ID #101D #00000000 #000031F8 #00003206 #00000000
           #00000000 #00000000 #00000000
```

in which #101D is the transputer identifier; the following 7 numbers are respectively:

1. not used
2. the number of events processed by the LAYER1 algorithm
3. the number of data blocks sent to LAYER2 (includes all events plus 13 'FORWARD.CONSTANTS' data blocks plus 1 'BECOME.ACTIVE' data block)
4. not used
5. not used

6. the number of arithmetic errors/overflows that occurred in the LAYER1 algorithm process (in which part(s) of the algorithm an error occurred can be found per event in a word in the CAL-SLT offline databanks)

7. not used

For a LAYER2 trigger transputer the status printed looks like this (example):

```
TP.ID #1214 #31F8 #0000 #0000 #3206 #3206 #0000 #3206 #0000
            #0000 #0000 #0000 #0000 #0000 #0000 #0000
```

in which #1214 is the transputer identifier; the following 15 numbers are respectively:

1. the number of events processed by the LAYER2 algorithm

2. the number of arithmetic errors/overflows that occurred in the LAYER2 algorithm process (in which part(s) of the algorithm an error occurred can be found per event in a word in the CAL-SLT offline databanks)

3. the number of events received by the input process (on link 0); includes all events plus 13 'FORWARD.CONSTANTS' plus 1 'BECOME.ACTIVE'

4. idem (on link 1)

5. idem (on link 2)

6. idem (on link 3)

7. the number of events sent to LAYER3; includes all events plus 13 'FORWARD.CONSTANTS' plus 1 'BECOME.ACTIVE'

8. not used

9. not used

10. the number of times a transmission error occurred in the reception of an event from LAYER1 (on link 0)

11. idem (on link 1)

12. idem (on link 2)

13. idem (on link 3)

14. not used

15. not used

For a LAYER3 trigger transputer the status printed looks like this (example):

```
TP.ID #1400 #31F8 #0000 #3206 #3206 #3206 #0000 #31F8 #0000
            #0000 #0000 #0000 #0000 #0000 #0000 #0000
```

in which #1400 is the transputer identifier; the following 15 numbers are respectively:

1. the number of events processed by the LAYER3 algorithm

2. the number of arithmetic errors/overflows that occurred in the LAYER3 algorithm process (in which part(s) of the algorithm an error occurred can be found per event in a word in the CAL-SLT offline databanks)

3. the number of events received by the input process (on link 1); includes all events plus 13 'FORWARD.CONSTANTS' plus 1 'BECOME.ACTIVE'
4. idem (on link 2)
5. idem (on link 3)
6. not used
7. the number of events sent to the GSLT
8. not used
9. not used
10. the number of times a transmission error occurred in the reception of an event from LAYER2 (on link 1)
11. idem (on link 2)
12. idem (on link 3)
13. not used
14. not used
15. not used

20 Standalone Test Runs

It has been made possible to run the complete CALDAQ system including the Digital Cards (the DSPs) and the CAL-SLT transputer network with a set of real events in order to test the functioning and the maximum performance of the CALDAQ system. The socalled **TESTGSLT** transputers simulate the workings of the GSLT decision-making and subsequent decision distribution.

Command **run_it** to perform this kind of standalone run can be found in directory **/zeus/transputer/online/v***xx.y***/bin/host**.

The setup parameters in the DAQ configuration file normally received from RunControl are hardcoded in the source code found in file
/zeus/transputer/online/v*xx.y***src/host/standalone_runcontrol.occ**;
it includes a simple user-interface from which runs can be started, aborted etc.

A special DSP-code (the socalled 'player code') is booted on the Digital Cards which produces events in the Digital Card DPM with a frequency that can be set by the user (NB: the Digital Cards run unsynchronized in this mode!).

The events produced are either random energy-noise events (when no 'real' events have been downloaded to the Digital Cards) or the events have been read from (a) ZEBRA-formatted file(s) whose name has/have been read from file **evpfiles.dat** in directory **/zeus/transputer/online/v***xx.y***/bin/host**. These files also contain the SLT-data banks so that on the TESTGSLT transputers the results of the SLT-network can be checked against what is expected.

21 CSB connections

For a description of the CSB crates and their function in the CALDAQ system see chapter 2.5 and 3.5 of [4]; for a detailed description of the CSB hardware see [5].

On the FCAL, BCAL and RCAL CSB the 64-pin connectors labelled x**CAL 2ND LEVEL TRIGGER-READOUT** are numbered from 00 to 15 and connect to the 2TP-modules in FCAL/BCAL/RCAL crates numbered 1 to 16 respectively.

Every socalled READOUT transputer of the connected 2TP-modules (the 2TP's *Y*-transputer) is connected (internal to the CSB) to the ARE1- and the EVT1-board; the 2TP-modules in crates 1 to 16 are connected to ARE1-0 to ARE1-15 and EVT1-0 to EVT1-15 respectively; the socalled TRIGGER transputers (the 2TP's *X*-transputers are similarly connected to the ARE2- and EVT2-boards.

All connections labeled **TRIG-***n* connect the TRIGGER transputer in crate **n+1** to LAYER2 of the Second Level Trigger transputer network; see the transputer network scheme in appendix A.

Tables 9 to 15 give an overview of most of the rest of the external LKS, ARE and EVT connections of the CSBs.

FCAL	
Connector	**Connected to link**
LKS1-SMLK	TRP 1
LKS1-24	LKS2-24
LKS1-25	LKB-BMLK
LKS1-26	FCAL ROCOLLECT 1
LKS1-27	FCAL ROCOLLECT 0
LKS1-28	FCAL MONITOR 1
LKS1-29	TEST-GSLT Y-1
LKS1-30	GSLT decision
LKS2-SMLK	TRP 2
LKS2-24	LKS1-24
LKS2-25	-
LKS2-26	FCAL ROCOLLECT 2
LKS2-27	FCAL ROCOLLECT 3
LKS2-28	FCAL MONITOR 2
LKS2-29	REBOOTER 1
LKS2-30	-

Table 9: FCAL CSB LKS connections.

BCAL	
Connector	**Connected to link**
LKS1-SMLK	TRP 1
LKS1-24	LKS2-24
LKS1-25	LKB-BMLK
LKS1-26	BCAL ROCOLLECT 1
LKS1-27	BCAL ROCOLLECT 0
LKS1-28	BCAL MONITOR 1
LKS1-29	TEST-GSLT Y-2
LKS1-30	GSLT decision
LKS2-SMLK	TRP 2
LKS2-24	LKS1-24
LKS2-25	BCAL DATACOLLECT 1
LKS2-26	BCAL ROCOLLECT 2
LKS2-27	BCAL ROCOLLECT 3
LKS2-28	BCAL MONITOR 2
LKS2-29	REBOOTER 3
LKS2-30	TEST-GSLT X-0

Table 10: BCAL CSB LKS connections.

RCAL	
Connector	**Connected to link**
LKS1-SMLK	TRP 1
LKS1-24	LKS2-24
LKS1-25	LKB-BMLK
LKS1-26	RCAL ROCOLLECT 1
LKS1-27	RCAL ROCOLLECT 0
LKS1-28	RCAL MONITOR 1
LKS1-29	TEST-GSLT Y-3
LKS1-30	GSLT decision
LKS2-SMLK	TRP 2
LKS2-24	LKS1-24
LKS2-25	-
LKS2-26	RCAL ROCOLLECT 2
LKS2-27	RCAL ROCOLLECT 3
LKS2-28	RCAL MONITOR 2
LKS2-29	REBOOTER 2
LKS2-30	-

Table 11: RCAL CSB LKS connections.

BCAL LKS3	
Connector	**Connected to link**
LKS3-SMLK	TRP 3
LKS3-00	LAYER3 0
LKS3-01	GSLT input
LKS3-02	TEST-GSLT X-3
LKS3-04	-
LKS3-05	-
LKS3-06	-
LKS3-07	-
LKS3-08	-
LKS3-09	-
LKS3-10	-
LKS3-11	-
LKS3-12	-
LKS3-13	-
LKS3-14	-

Table 12: BCAL CSB LKS3 (external LKS-board, located in BCAL SSC (Sub-System Crate)) connections.

FCAL	
Connector	**Connected to transputer**
ARE3/EVT3-00	FCAL LAYER2 #0
ARE3/EVT3-01	FCAL LAYER2 #1
ARE3/EVT3-02	FCAL LAYER2 #2
ARE3/EVT3-03	FCAL LAYER2 #3
ARE3/EVT3-04	FCAL LAYER2 #4
ARE3/EVT3-05	FCAL MONITOR
ARE3/EVT3-06	-
ARE3/EVT3-07	-
ARE3/EVT3-08	-
ARE3/EVT3-09	-
ARE3/EVT3-10	-
ARE3/EVT3-11	-
ARE3/EVT3-12	-
ARE3/EVT3-13	-
ARE3/EVT3-14	-
ARE3/EVT3-15	FCAL ROCOLLECT

Table 13: FCAL CSB ARE/EVT connections.

BCAL	
Connector	**Connected to transputer**
ARE3/EVT3-00	BCAL LAYER2 #0
ARE3/EVT3-01	BCAL LAYER2 #1
ARE3/EVT3-02	BCAL LAYER2 #2
ARE3/EVT3-03	BCAL LAYER2 #3
ARE3/EVT3-04	BCAL LAYER2 #4
ARE3/EVT3-05	BCAL MONITOR
ARE3/EVT3-06	BCAL DATACOLLECT
ARE3/EVT3-07	DATACOLLECT
ARE3/EVT3-08	-
ARE3/EVT3-09	-
ARE3/EVT3-10	-
ARE3/EVT3-11	-
ARE3/EVT3-12	-
ARE3/EVT3-13	-
ARE3/EVT3-14	TEST-GSLT
ARE3/EVT3-15	BCAL ROCOLLECT

Table 14: BCAL CSB ARE/EVT connections.

RCAL	
Connector	**Connected to transputer**
ARE3/EVT3-00	RCAL LAYER2 #0
ARE3/EVT3-01	RCAL LAYER2 #1
ARE3/EVT3-02	-
ARE3/EVT3-03	-
ARE3/EVT3-04	-
ARE3/EVT3-05	RCAL MONITOR
ARE3/EVT3-06	MONITOR.HOST
ARE3/EVT3-07	LAYER3
ARE3/EVT3-08	LAYER3 MONITOR
ARE3/EVT3-09	-
ARE3/EVT3-10	-
ARE3/EVT3-11	-
ARE3/EVT3-12	-
ARE3/EVT3-13	-
ARE3/EVT3-14	REBOOTER
ARE3/EVT3-15	RCAL ROCOLLECT

Table 15: RCAL CSB ARE/EVT connections.

22 LAYER2/3 transputers

Table 16 shows in which crates the LAYER2 and LAYER3 transputers are located and which LAYER1 and LAYER2 transputers connect to which LAYER2 and LAYER3 transputer respectively.

Transputer (id)	**In Crate**	**Link 0-3 connections**			
FCAL LAYER2 #0 (#1210)	FCAL #16	#1015	#101A	#1017	#1018
FCAL LAYER2 #1 (#1211)	FCAL #16	#1401	#101D	#101E	#101F
FCAL LAYER2 #2 (#1212)	FCAL #15	(#1014	#101B	#1016	#1019) [7]
FCAL LAYER2 #3 (#1213)	FCAL #15	#1401	#1013	#1010	#1011
FCAL LAYER2 #4 (#1214)	RCAL #8	#1401	#1012	#101C	-
BCAL LAYER2 #0 (#1220)	BCAL #8	#1023	#1024	#1025	#1026
BCAL LAYER2 #1 (#1221)	BCAL #8	#1400	#1020	#1021	#1022
BCAL LAYER2 #2 (#1222)	BCAL #12	#102A	#102B	#102C	#102D
BCAL LAYER2 #3 (#1223)	BCAL #12	#1400	#1027	#1028	#1029
BCAL LAYER2 #4 (#1224)	RCAL #7	#1400	#102E	#102F	-
RCAL LAYER2 #0 (#1240)	RCAL #7	#1044	#1045	#1042	#1043
RCAL LAYER2 #1 (#1241)	RCAL #8	#1040	#1041	#1046	#1047
LAYER3 #0 (#1400)	FCAL #14	GSLT	#1221	#1223	#1224
LAYER3 #1 (#1401)	FCAL #14	HOST	#1214	#1211	#1213

Table 16: LAYER2/LAYER3 location and connections.

23 OCCAM Preprocessor

Program **occpp** is an OCCAM preprocessor, developed by Andres Kruse. A more comprehensive description of the program than given here can be found in [7]. The purpose of **occpp** is to extend the number of available compiler metacommands, as follows:

- **#DEFINE** *variable*
  a statement to define a variable.
- **#UNDEF** *variable*
  undefine a variable.
- **#IFDEF** *variable*, **#ENDIF**
  if the variable is defined the code that is enclosed between these two statements will be processed further.
- **#IFNDEF** *variable*, **#ENDIF**
  if the variable is defined the code will NOT be processed further.
- **#ELSE**
  extension to **#IFDEF** and **#IFNDEF**

With the **-D** and **-U** option this preprocessor can be used as a tool to have conditionally compiled code.
The syntax is as follows:

occpp [-v] [[-Dvariable1] [-Dvariable2]...] [[-Uvariable1] [-Uvariable2]...] file[s] with:

- **-h**
  give usage information (no processing)
- **-v**
  print information while processing
- **-V**
  print debug information
- **-F**
  force occpp to overwrite existing .occ and .inc files (normally it will only overwrite .occ and .inc files if they differ from the new version (good for make!))
- **-Dvariable** define the variable to be TRUE this cannot be overwritten by an **#UNDEF** in the file !
- **-Uvariable** define the variable to be FALSE this cannot be overwritten by an **#DEFINE** in the file !
- **-o file** name of the output file then only one input file is allowed
- **file[s]** names of the occam source files; the extension has to be **.opp** or **.ipp** or **.ppp**.

occp will automatically generate files with the extension ".occ", ".inc" and ".pgm"

[7] These transputers are not present in the current system.

References

[1] Henk Boterenbrood,
Debugging and Test Utilities for the CALDAQ Transputer Network
Version 2.1, Internal documentation, NIKHEF-H, Amsterdam, August 1995.

[2] Steve Ritz,
DAQ5.DSP,
ZEUS-Note 92-76.

[3] A. Caldwell and S. Ritz,
User Interfaces to the CAL Electronics Readout,
ZEUS-Note 92-46, April 1992.

[4] Hermen van der Lugt,
The Data-Acquisition and Second Level Trigger System for the ZEUS Calorimeter,
PhD-thesis, Universiteit van Amsterdam, April 1993.

[5] Arthur de Waard,
Hardware Description of the Control and Switch Box Crates
Version 2.1, Internal documentation, NIKHEF-H, Amsterdam, August 1991.

[6] Andres Kruse,
User Defined ONLINE Charge and Time Reconstruction for the Calorimeter LASER and LED Monitor Readout,
NIKHEF-H, Amsterdam.

[7] Andres Kruse,
OCCPP: An OCCAM Preprocessor,
NIKHEF-H, Amsterdam.

A CALDAQ Transputer Network Map

The following figure shows a detailed map of the layout of the CALDAQ transputer network, including TPM VME-addresses and most transputer link numbers.

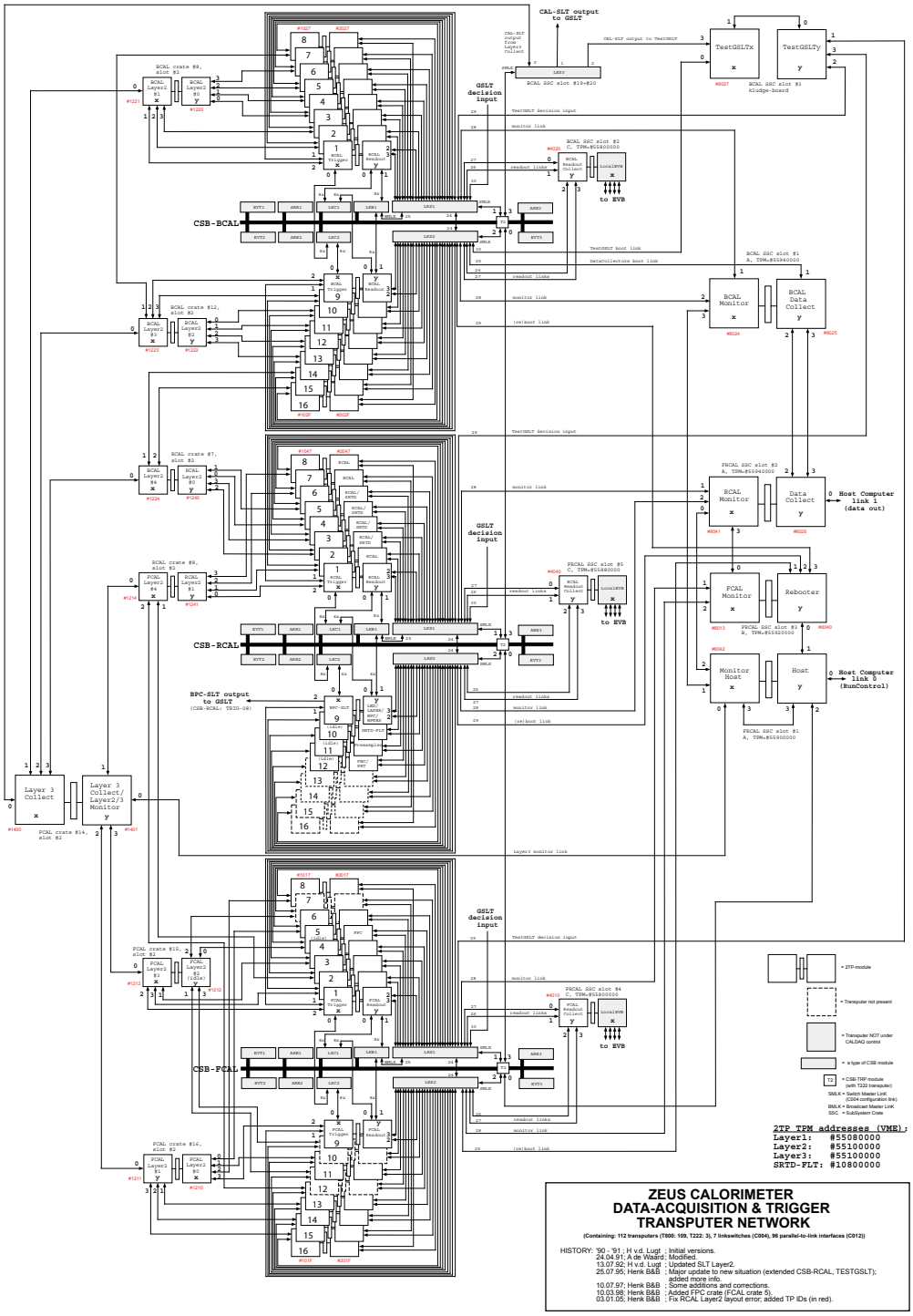

Figure 1: The CALDAQ transputer network.

B Adding a Digital Card Based Component to the CAL Readout

The following list sums up the source code files and configuration files involved –together with a short description of the changes/additions necessary– when adding a completely new –Digital Card based– component to the CALDAQ system.

The example below is based on the work necessary to include the BARREL PRESAMPLER (BPRES) in the readout.

```
=======================================================

component/banks declaration stuff
=================================

include/bankbitpattern.inc:
- increase NO.COMPONENTS
- define component id (bit number in bankbitpattern)
- COMPONENT.BITPATT, BANKNAMES

include/bankcontrol.inc:
- add to BANK.CONTROL (here: BPRES banks):
  [ PBBA.BANK, BOREOR.BITPATT ],
  [ PBPM.BANK, BOREOR.BITPATT ],
  [ PBDC.BANK, EVDATA.BITPATT ],
  [ PBTE.BANK, EVDATA.BITPATT ],
  [ PBCO.BANK, EVDATA.BITPATT ],
  [ PB8S.BANK, EVDATA.BITPATT ],
  [ PB6S.BANK, EVDATA.BITPATT ],
  [ PBXO.BANK, EVDATA.BITPATT ],
  [ PBUM.BANK, EVDATA.BITPATT ],
  [ PBDM.BANK, EVDATA.BITPATT ],
  [ PBPQ.BANK, MSIG.BITPATT ],
  [ PBPP.BANK, MSIG.BITPATT ],
  [ PBPL.BANK, MSIG.BITPATT ],
  [ PBPD.BANK, MSIG.BITPATT ],
  [ PBDU.BANK, MSIG.BITPATT ],

include/banks.inc:
- adjust MAX.NO.BANKS
- add "VAL INT XXxx.BANK IS ...." (with 'XX' the 2-letter id for the component;
                                   'PB' for BPRESAMPLER)
- add "VAL INT xxxx.NCOLS IS ..." if needed
- add to BANK.NCOLS (here: BPRES banks):
  [ PBBA.BANK, xxBA.NCOLS ],
  [ PBPM.BANK, xxPM.NCOLS ],
  [ PBDC.BANK, PRSD.NCOLS ],
  [ PBTE.BANK, PRTE.NCOLS ],
  [ PBCO.BANK, xxCO.NCOLS ],
  [ PB8S.BANK, PRS8.NCOLS ],
  [ PB6S.BANK, PRS6.NCOLS ],
  [ PBXO.BANK, xxXO.NCOLS ],
  [ PBUM.BANK, xxUM.NCOLS ],
  [ PBDM.BANK, PRDM.NCOLS ],
  [ PBPQ.BANK, PRPQ.NCOLS ],
  [ PBPP.BANK, PRPP.NCOLS ],
  [ PBPL.BANK, PRPL.NCOLS ],
  [ PBPD.BANK, PRPD.NCOLS ],
  [ PBDU.BANK, xxDU.NCOLS ],

include/banks_info.inc:
- add entries to BANK.INFO
- add entries to BANK.DESCR

include/banks_maxsize.inc:
- add entries to BANK.MAX.NROWS
```

```
include/banksets.inc:
- add bankset entries:
  ('xxxx' = component name; the actual numbers should be different).
VAL INT  xxxx.BOR.BANKSET     IS 57 :
VAL INT  xxxx.EOR.BANKSET     IS 58 :
VAL INT  xxxx.NORMAL.BANKSET  IS 59 :
VAL INT  xxxx.ENV.BANKSET     IS 60 :
VAL INT  xxxx.UNO.BANKSET     IS 61 :
VAL INT  xxxx.MS.BANKSET      IS 62 :
- add banksets (or lists):
  ('xxxx' = component name).
  xxxx..BOR.BANKLIST
  xxxx..EOR.BANKLIST
  xxxx..NORMAL.BANKLIST
  xxxx..ENV.BANKLIST
  xxxx..UNO.BANKLIST
  xxxx..MS.BANKLIST
- add banksets to BANKSET.BANKLIST (=list of banklists)
- add banksets list to COMPONENT.BANKSET.LIST
  (=lists of bankset indices (index into BANKSET.BANKLIST)
   ordered per trigger type)

include/zebra_decl.inc:
- add entries for bank headers (2 'holleriths' per bank)

include/zebra_init.inc:
- add 1 entry per bank in ZEBRA.BANK.NR, ZEBRA.FCAL.BANK.ID,
  ZEBRA.BCAL.BANK.ID and ZEBRA.RCAL.BANK.ID

include/rcparams.inc:
- add new control commands for downloading the PM-numbers:
  ('xxxx' = component name; actual numbers should be different).
VAL BYTE  GET.xxxx.PM.NUMBERS.LKS  IS  #5D(BYTE) :
VAL BYTE  GET.xxxx.PM.NUMBERS.LKC  IS  #DD(BYTE) :

include/dc.inc:
- if necessary add DSP code dependent DC page data offsets and
  number of channels.

Declaration/init stuff
======================

libs/decode_hwparams.occ:
- add 'xxxx.from.dc, xxxx.for.dc' ('xxxx' = component name).

libs/send_numbers.occ:
- add 'send.pmnumbers()' call to 'send.numbers()' routine
  (if PM-numbers are to be downloaded).

readout/hwparams_decode.occ:
- add 'xxxx.from.dc, xxxx.for.dc' ('xxxx' = component name).

readout/hwparams_init.occ:
- inititialize 'xxxx.from.dc, xxxx.for.dc' ('xxxx' = component name).

include/hwparams_mem.inc:
- add 'xxxx.from.dc, xxxx.for.dc' ('xxxx' = component name).

readout/caldata.opp:
- add ('xxxx' = component name):
  INT xxxx.pm.numbers.from, xxxx.pm.numbers.for :
- add ('xx' = component 2-letter id; 'pb' for BPRESAMPLER):
  []INT xxco.bank :
  INT   xxco.bank.size :
  and
  INT no.xxdc.rows, no.xxte.rows, no.xxco.rows :
  INT no.xx6s.rows, no.xx8s.rows :
```

```
  INT xxdc.start, xxte.start, xxco.start, xx6s.start, xx8s.start :

- add 'make.dummy.pm.numbers()' call
- add entries for reception of PM-numbers:
  IF
    tag = GET.xxxx.PM.NUMBERS.LKS
      get.pm.numbers()
    tag = GET.xxxx.PM.NUMBERS.LKC
      get.pm.numbers()

readout/ro_csb_intf.occ:
readout/calib_csb_intf.occ:
- add entries for reception of PM-numbers:
    tag = GET.xxxx.PM.NUMBERS.LKS
      ...
    tag = GET.xxxx.PM.NUMBERS.LKC
      ...

readout/calcconst.opp:
- add entry for reception of PM-numbers:
    tag = GET.xxxx.PM.NUMBERS.LKS
      ...

readout/print_hwparams.occ:
- add new component to print out

Means&Sigma stuff
=================

readout/caldata_msinit.occ:
- add initialization to 'PROC init.meansigma.stuff()':
      IF
        (component.mask /\ BPRES.BITPATT) <> 0
          init.meansigma.for.component( COMPONENT.BPRES,
                                        BPRES.from.dc,
                                        BPRES.for.dc,
                                        BPRES.pm.numbers.from,
                                        BPRES.pm.numbers.for )
        TRUE
          SKIP

readout/caldata_msmem.inc:
- add MS-banks for the new component to MSIG.bank.list[][]:
    [ COMPONENT.BPRES,        QINJ.TEST.TRIGGER,   PBPQ.BANK ] ,
    [ COMPONENT.BPRES,        EMPTY.TEST.TRIGGER,  PBPP.BANK ] ,
    [ COMPONENT.BPRES,        LED.TEST.TRIGGER,    PBPD.BANK ] ,
    [ COMPONENT.BPRES,        LASER.TEST.TRIGGER,  PBPL.BANK ] ,
    [ COMPONENT.BPRES,        UNO.TEST.TRIGGER,    PBDU.BANK ] ,

readout/makemsdatabanks.occ:
- add making of the MS-banks:
      xxPQ.BANK, xxPP.BANK, xxPD.BANK, xxPL.BANK,
  and
      xxDU

DAQ stuff
=========

readout/makeboreorbanks.occ:
- add making of:
xxPM.BANK
xxBA.BANK

readout/makedcdatabanks.occ:
- add making of:
xxDC.BANK
xxTE.BANK
xx6S.BANK (xx8S.BANK)
```

```
xxUM.BANK
xxDM.BANK

readout/makenondcdatabanks.occ:
- add making of:
xxCO.BANK
xxXO.BANK

Test tools
==========
test/check_readout.occ
- add 'xxxx.from.dc, xxxx.for.dc' ('xxxx' = component name).
- add 'component = COMPONENT.xxxx' to FUNCTION calc.crate.event.size().
- exclude xx6S and xx8S banks if event sizes without samples are requested.

Notes:
1. Run the test tool ('run_check_readout') to get bank lists on a per crate
basis and (maximum) event sizes; also run 'run_testtables' to check the
consistency of the different bank tables in the include files.

2. When these tests do not detect any problems one can be reasonably confident
there will be no runtime errors due to inconsistent bank info or buffers
that are too small to contain all event data.

3. Compile the test tools, then run the tools, only then compile the actual
CALDAQ stuff.

Configuration
=============

~calec_rc/defaults/hwconf/hwconfig.dat:
- add Digital Cards with proper component number in appropriate crate,
  and increase the total number of DCs for that crate.

~calec_rc/defaults/dcconf/dcconf_N.conf
with 0<=N<=...
- put appropriate numbers to boot a particular DSP code on the new
  Digital Card(s).

~calec_rc/defaults/*.dat:
- set the bankbit for the new component in the bankbit pattern(s)
  for F, R and/or BCAL; e.g. for BPRESAMPLER with DCs in RCAL:
   you have to add 0x00004000 to the RCAL 'bankbitpattern'
   e.g. in ALLCAL_PHYSICS.DAT change line
 70001FEE       /* RCAL banks  PRT=0x80,BPC=0x100,CAMAC=0x200,SRFLT=0x400,   */
   to
 70005FEE       /* RCAL banks  PRT=0x80,BPC=0x100,CAMAC=0x200,SRFLT=0x400,   */
- change the number of DCs in the lines stating the number of cards per
  crate according to the number of DCs added.

PM-number file:
- if required make a new PM-number zebra-file using the 'cable2zebra'
  tool in directory /zeus/transputer/tools/cable2zebra
- copy the resulting zebra-file to /zeus/transputer/online/vXX.Y/bin/cablefiles
  and create a link to this file in /zeus/transputer/online/vXX.Y/bin/host
  (with XX.Y, the tp-code version number).

~calec_rc/bin/start...
- the runcontrol program should point to the new transputer code version
  if a new version has been made
```