

READOUT

version: readout-v4.2

last update: 11 august 2009

The DAQ server (ilcserver) fork and exec the readout program PROG_PATH which is defined in ilcserver.cxx. The exec'ed program start up three threads, in addition to the main readout. Two of these threads handle requests for on line events and playback events read from a file. The third send regularly an status update. The run commands exchanged with the run control the system are sent as text strings over the network. The command starts with *<COMMAND> followed by a number of arguments, i.e. *<COMMAND> ARG1 <value> ARG2 <value> ... <ARGn> value. It is the task of the ilcserver to interpret the commands from the remote control system and decode it into these commands as understood by the DAQ readout program. In the case of the local run control do the run control send the commands as they are given here, i.e. there is no decoding by the ilcserver, except when a run is active. When a run is active and the readout is polling for data is the readout thread not listening to the network since it is busy polling for data, instead is the server communicating with it by writing the received command to the network port and issues then a unix signal:

SIGUSR1	status requested
SIGUSR2	an action is requested

When the readout recognize the unix signal, it exits the readout loop and reads the command from the network port. After having serviced the action does it resume polling for data, unless the command was pause or end run. The protocol between the ilcserver and readout is shown in figure RO-1. The arguments of the commands are described in the document about the local run control.

Trigger(s)

The implemented hardware trigger is a NIM system which can be controlled from the parallel port of the computer, see fig RO-NIM. One must run the readout program as root to access the parallel port. If one is not root then the readout program will exit. There are three lines used in the parallel port:

0 - trigger reset (active low)

1 - trigger enable (active low)

2 – disable (off) or enable (on) pulser that generates a pulse to the front end card (active low)

There is one triggering mode in which the program handles the hardware trigger through the parallel port. There are two modes for triggering with the Distributor Box (DBOX). In one mode is a trigger expected from the DBOX through the network, and in the data packet the data from the DBOX. The second mode is data driven, i.e. the readout is waiting for data in the DRORC. After the DRORC has been read is the program waiting for data from the DBOX. After all data has been read is always an acknowledge sent to the DBOX. When using the DBOX is the parallel port not used.

The trigger configuration is read from the file dbbox.cfg at Start DAQ command.

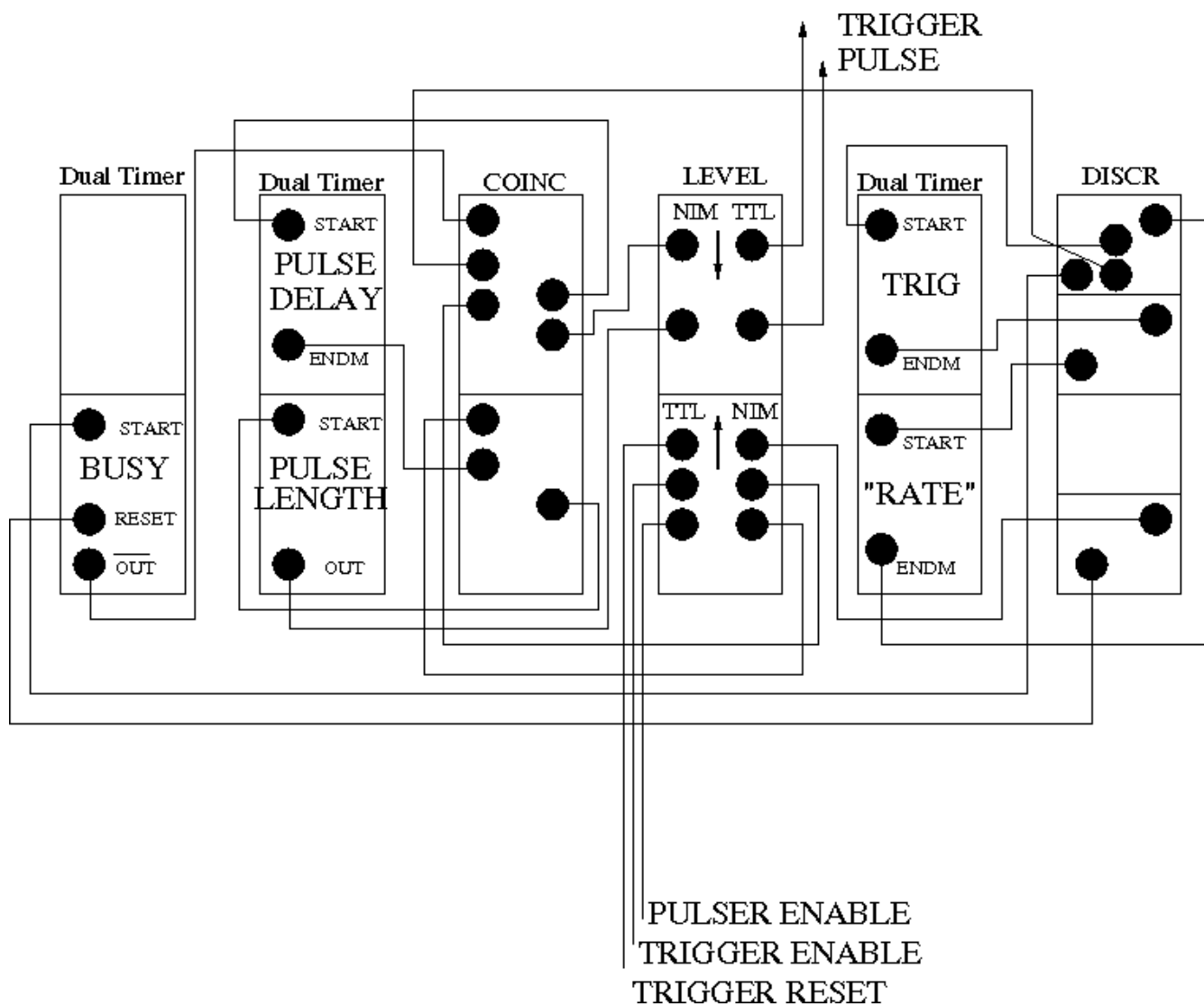


Figure RO-NIM: The NIM trigger logic used in Lund together with the parallel port trigger mode.

Program logic

When the readout program starts does it search for installed DRORC and active RCUs. This scan is done only at start of the program, therefore must the RCUs to be used already be on power and running before the readout program is started. The RCU runs a linux kernel and must boot before it is ready. The readout program starts threads for monitoring and status updates, and opens the network sockets used. The command *CLI is sent to the ilcserver when the readout task is ready and starts to listen on the network port for receiving commands.

The commands to power on/off FEC and load the PCA settings must be received when the DAQ is stopped, since it is using the RORC-RCU link in an incompatible way from the actual data readout. *START is starting the DAQ by setting up the physical memory in the PC and DRORC-RCU links for data taking. *STOP is stopping the DAQ by closing the physical memory in the PC and the DRCOR-RCU link for data taking. At *START is also all configurations loaded into the RCU/FEC/ALTRO, e.g. active channels, pedestals, run configuration parameters. The configurations are read from configuration files as described later. For a change to become active must the *STOP - *START sequence be done. A schematic description is given in figure RO-2.

When receiving the *SOR is the event readout loop entered. If data logging is requested is the data file opened. When in this loop is the program not listening on the network port. Instead are the unix signals SIGUSR1 and SIGUSR2 used to interrupt the event loop when something has been written into the socket for run handling commands. It then exits the loop to read and execute the command. These commands can be *PAUSE, *EOR, and *STATUS. If the run is paused or ended, then the program continues to listen on the network, until a new *SOR or *CONT is received.

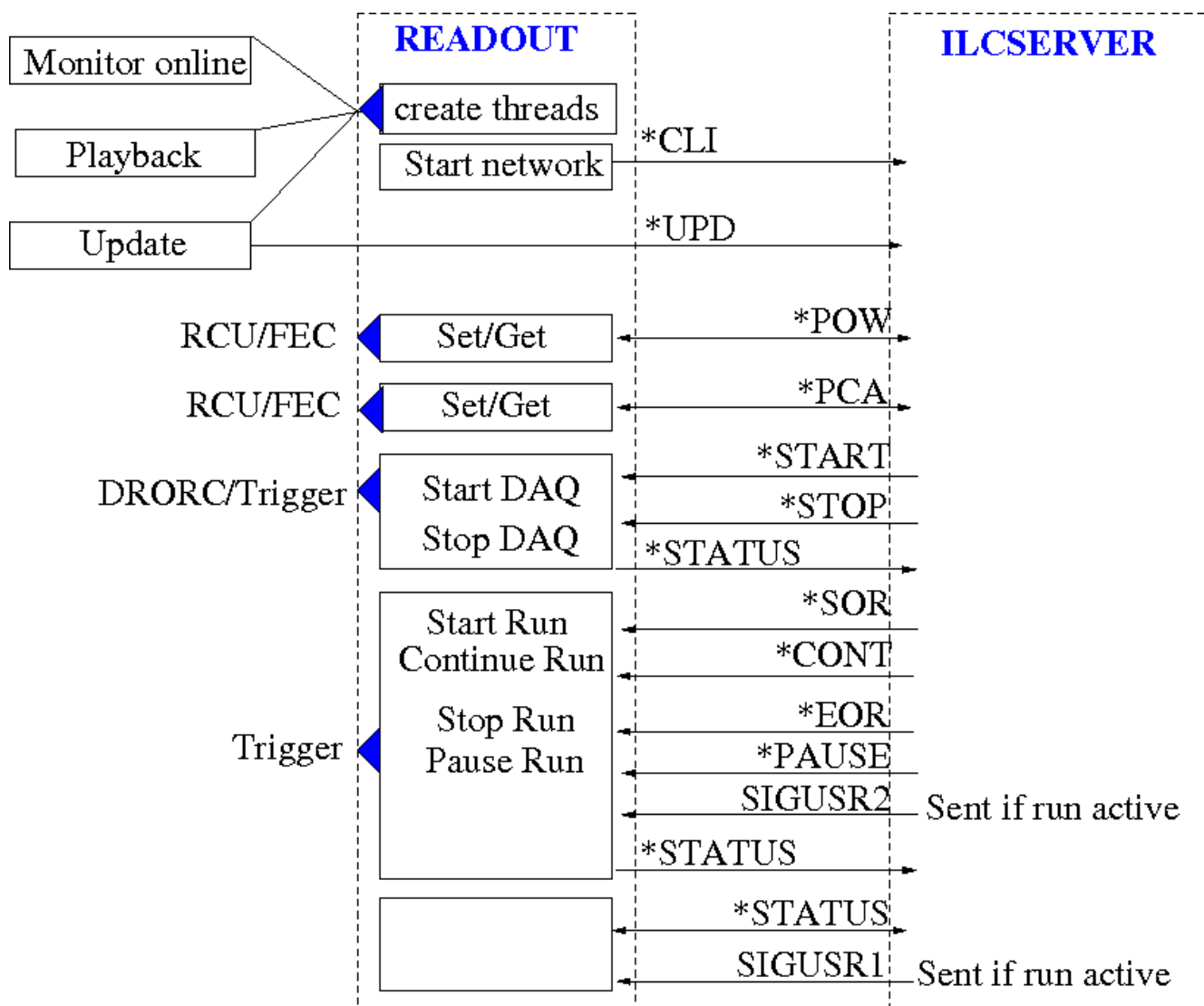


Figure RO-1: The protocol between the ilcserver and readout.

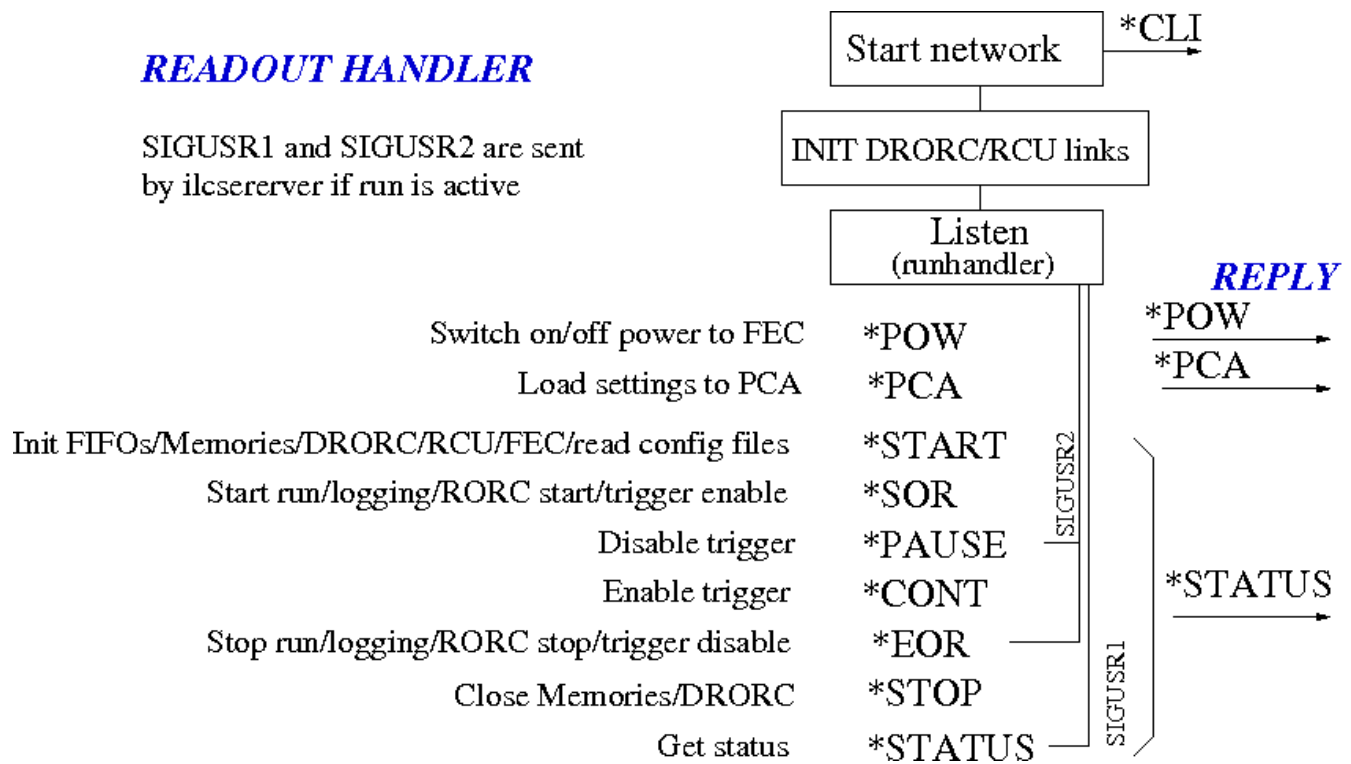


Figure RO-2: Schematic view of run handling in the readout handler.

Event loop

In the event loop is the active DRORCs polled for data. When all active DRORC are ready with new data is the readout waiting for the event buffer to be available (it may be locked by other threads using it). Since there is only one event buffer at the moment does it have be available for writing. When the event buffer is available it is locked and the data from the DRORCs is moved to the event buffer and formatted with headers. If data logging is requested then the event is a written to disk. The event buffer is unlocked. If a SIGUSRn has been received (flagged by a variable set in the signal handler) then the event loop is stopped and the request is handled. Figure RO-3 gives a view of the structure.

In the case of triggering through network by the DBOX is the readout waiting for a network connection before start polling the DRORC for data. If using the second DBOX mode is the readout waiting for data from the DBOX after the DRORC data has finished. The DBOX data is added as the same data structure as for the RCU data but with the RCUID set to 100.

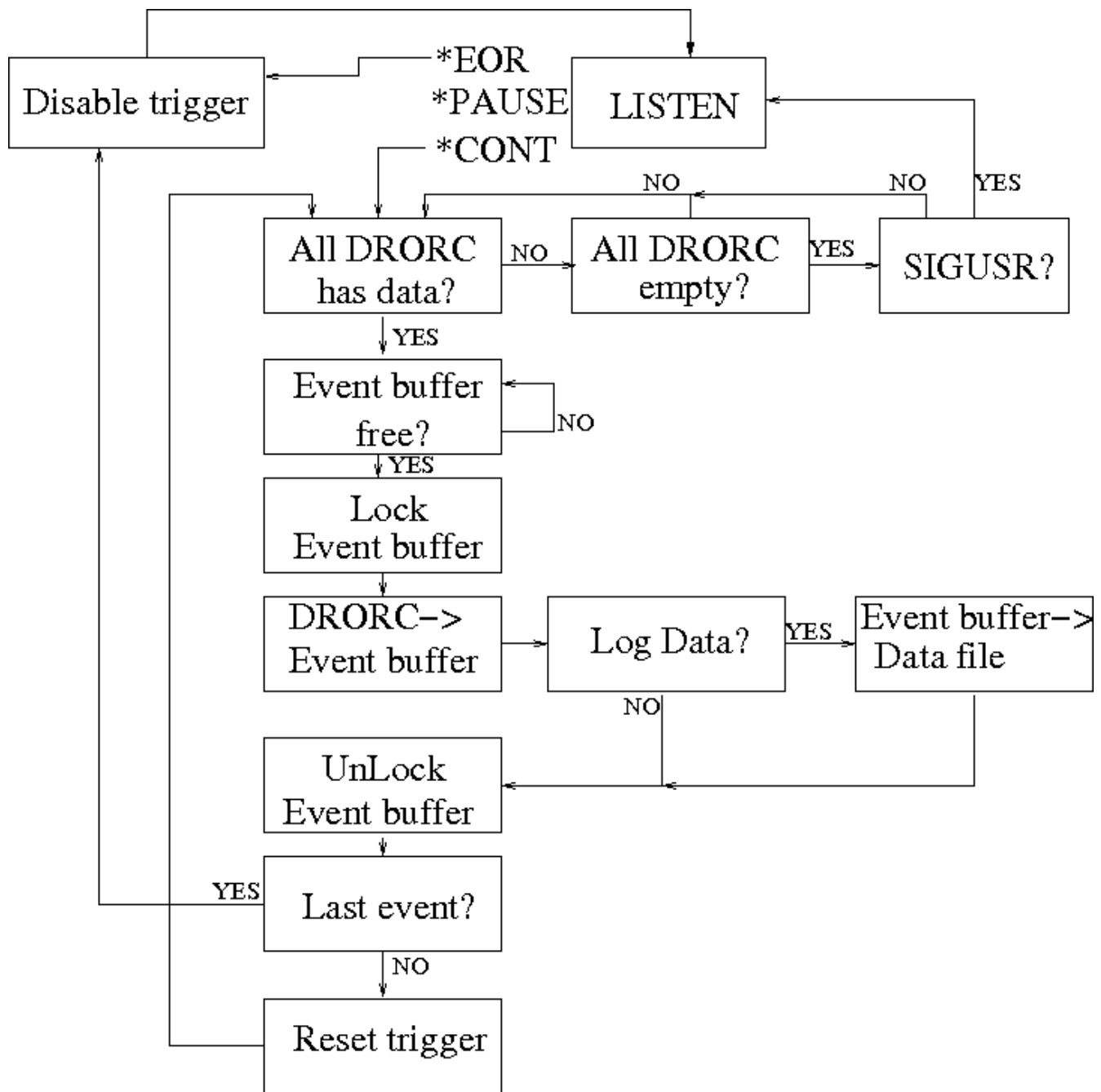


Figure RO-3: Event loop logic

DBOX emulator

in the the catalog dbxserver is a simple emulator of a DBOX. It use the parallel port to handle the hardware trigger. If TRIG_POLL is defined then an extra wait is done after the trigger to simulate the time it takes for the DBOX to send the data to the readout task.

Configuration files

The configuration of the front end is read from configuration files. The location of these files are in the location given by `CONFIG_DIR` which is defined in `readout.h`. This location can be changed with the command option `-configdir <configuration file directory>`. The first file read is `files.cfg`, which define the file locations of other files. The examples are those used in the testbeam at DESY 200907.

dbox.cfg

```
# configuration file for trigger and the distribution box
# TRIG.SOURCE    <source> 1=Parallell port, 2 = DBOX, 3 – data driven, with DBOX
# DBOX.SERVER    <ip-address> address of DBOX
# DBOX.PORT      <port> server port on the box
# DBOX.RND       <microseconds> random trigger rate
# DBOX.BSY       <microseconds> timeout of busy signal
# DBOX.ACKEN     <0,1> enable ethernet event acknowledge
# DBOX.BSYTO     <0,1> enable busy time out
# DBOX.SOURCE    <0,1,2,3,4> trigger source of DBOX
# 0 = TLU, 1 = NIM, 2 = random, 3 = ethernet, 4 = button
#
# DBOX.POLLTO    <milliseconds> timeout in waiting for data from DBOX
#               -1 = infinite
#
# added 20081208
# Is encoded into TRG;WORDCONF# command
#DBOX.CLOCK      1 FPGA oscillator, 2 external (not implemented)
#DBOX.INIT       0 reset DBOX only, 1 ALTRO reset, 2 AFTER reset
#DBOX.NIMTRIG    <hex> OR mask of NIM trigger inputs
# Should we getch the EUDAQ run number in case of DBOX running?
#EUDAQ.FETCH     0 – do not fetch, 1 – fetch
# EUDAQ run number file location
#EUDAQ.RUNNBFILE remote filename where EUDAQ run number is stored
# END
```

Example test beam 200907:

```
TRIG.SOURCE      1
DBOX.SERVER       131.169.207.111
DBOX.PORT         4000
DBOX.RND          19999
DBOX.BSY          1000000
DBOX.SOURCE       0
DBOX.ACKEN        1
DBOX.BSYTO        1
DBOX.POLLTO       1000
DBOX.CLOCK        1
DBOX.INIT         0
DBOX.NIMTRIG      1
EUDAQ.FETCH       0
EUDAQ.RUNNBFILE   <user>@flctpc-tlupc:LCTPC/DAQNoTS/eudaq/data/runnumber.dat
```

files.cfg

```
#
# Definitions of where files are read from/written to
# these are also defined in readout.h
# the values here override the program default
#
# Directories
#DIR.CONFIG      <configuration file directory> (default /misc/ilcdaq/config/)
#DIR.DATA        <data file directory> (default misc/ilcdaq/data/)
#DIR.LOG         <log file directory> (default /misc/ilcdaq/log/)
#
#Files
# Note 1. the configuration file names are defined in run.cfg
#FILE.MSGLOG     <name of message log file> (default msglog)
#FILE.MONLOG     <name of monitor log file> (default monlog)
#FILE.RUNLOG     <name of run log file> (default runlog)
# ADDED version 4.1
# config files: pca16, global-altro, global-rcu rcu-X
# Should we copy the configuration files at SOR
#CFG.COPY       0 – do not copy files, 1 – copy files
#DIR.CFGCOPY    directory to copy files to
# pedestal files pedestals-<rcu>-<shiftreg>|-<dac>.dat
# Should we copy the pedestal files at SOR to pedestastals-<rcu>-<runnb>.(l)dat
#PED.COPY       0 – do not copy files, 1 – copy files
#DIR.PEDCOPY    directory to copy files to
```

Example test beam 200907:

```
DIR.CONFIG      /misc/ilcdaq/config-v4.2/
DIR.DATA        /misc/ilcdaq/data/
DIR.LOG         /misc/ilcdaq/log-v4.2/
FILE.MSGLOG     msglog
FILE.MONLOG     monlog
FILE.RUNLOG     runlog
CFG.COPY        1
DIR.CFGCOPY     /misc/ilcdaq/configstore/
PED.COPY        1
DIR.PEDCOPY     /misc/ilcdaq/pedstore/
```

run.cfg

```
# Run information
#RUN.TYPE      <run type id>      user defined identifier
#RUN.COMMENT    Just a comment line
# Configuration files – note that the extension is automatically set to
# .pedestal, .physics by the program
#PED.ALTRO     <file with altro settings for pedestal run> (default is global-altro)
#PHYS.ALTRO    <file with altro settings for physics run> (default is global-altro)
#PED.RCU       <file with RCU settings for pedestal run> (default is global-rcu)
#PHYS.RCU      <file with RCU settings for physics run> (default is global-rcu)
# ALTRO.LOOKUP <type>
# type 0 - not used
# type 1 - straight lookup table
```

```

# type 2 - polarity = 1 invert (1023-ADC) some channels (8-15)
# type 3 - polarity = 0 invert (1023-ADC) some channels (0-7)
# type 4 fixed value 0x2AA
# type 5 fixed value 0x155
# type 6 some peaks
# type 7 read from file "lookuptable.dat"
# The lookup table is used to invert 8 channels of the PCA16 which has opposite polarity as to what
# the ALTRO chip expects.
# LOOKUP.DEST n - 0 = ALTRO registers, 1 = RCU PMEM
# LOOKUP.VERIFY n - 0 = do not verify written, 1 = do verify
# RUN.ONROERROR 0 - do nothing on error, else stop run
# RUN.DATAFORMAT 1 (test beam 200907)
# TIMEOUT in waiting for all RCUs to have data - milliseconds
# should be >100 and <1000
# RUN.ROTIMEOUT <milliseconds>
# REREAD set to != 0 to read data twice
# RUN.REREAD <value>
#
# ----- VARIABLES READ ONLY ONCE AT PROGRAM START
# ----- THESE VALUES MAY BE CHANGED FROM RUN CONTROL
#
# PROG.INIT 1
# Load lookup tables at power on - read only program start
# LOOKUP.POW 0- do not load, 1- load
# Load lookup tables at start daq - read only program start
# LOOKUP.START 0- do not load, 1- load

```

Example test beam 200907:

```

RUN.TYPE 1
RUN.COMMENT Just a comment line
PED.ALTRO global-altro
PHYS.ALTRO global-altro
PED.RCU global-rcu
PHYS.RCU global-rcu
ALTRO.LOOKUP 2
LOOKUP.DEST 1
LOOKUP.VERIFY 1
RUN.ONROERROR 0
RUN.DATAFORMAT 1
RUN.ROTIMEOUT 500
RUN.REREAD 0
PROG.INIT 1
LOOKUP.POW 1
LOOKUP.START 0

```

daq.cfg

```

# General DAQ configuration
# read and overwritten written at each start of run.
# Not normally edited, but one can reset the run number here.
# FORMAT:
# RUN.NB <nb> last run number, incremented for each run

```

#RUN.MONEVENTS <value> last value used for monitoring
#RUN.LOGGING <value> last status of monitoring

Example test beam 200907:

RUN.NB 8055
RUN.MONEVENTS 1
RUN.LOGGING 0

drorc.cfg

This file contains the mapping between
RCU and DRORC channel
FORMAT:
RCU.ID <id> id of RCU [0..3]
DRORC.REVISION <revision> revision number of DRORC
DRORC.SERIAL <serial> serial number of DRORC for this RCU
DRORC.CHANNEL <channel> DIU on DRORC (0,1) for this RCU
DRORC.STATUS [0,1] channel status: 0-not used,1-used

The serial number must correspond to an installed DRORC, else will the RCU not be read out.

NOTE: *this file is read only once, when the readout program starts. It is the only time active channels and RCU links that are up are scanned. This means that the RCUs must be switched on before the readout program is started.*

Example test beam 200907:

RCU.ID 0
DRORC.REVISION 4
DRORC.SERIAL 9034
DRORC.CHANNEL 0
DRORC.STATUS 0

RCU.ID 1
DRORC.REVISION 4
DRORC.SERIAL 9034
DRORC.CHANNEL 1
DRORC.STATUS 0

RCU.ID 2
DRORC.REVISION 4
DRORC.SERIAL 10081
DRORC.CHANNEL 0
DRORC.STATUS 1

RCU.ID 3
DRORC.REVISION 4
DRORC.SERIAL 10081
DRORC.CHANNEL 1
DRORC.STATUS 0

rcu-<rcuid>.cfg where rcuid is the RCU ID given in drorc.cfg

For information about the current version of RCU firmware see the documentation:
[RCUFirmwar070509.pdf](#)

```
# This file contains information of active FEC/ALTRO
# for the RCU id as given in filename ruc_<id>.cfg
# RCU.ID id must be repeated as first statement
# then default values follows
# Each FEC ALTRO can be individually be changed from
# default ACL setting.
#FORMAT
#RCU.ID          <rcuid>      MUST BE FIRST STATEMENT!!
#RCU.POWER       <hexvalue>   Which FEC to power on (hexbit pattern)
#RCU.READOUT     <hexvalue>   Which FEC to readout (hexbit pattern)
#RCU.ACL         <hexvalue>   Default ALTRO active channel list (for each FEC to readout)
#RCU.ZSUP        <hex>        Default ALTRO zero suppression (for each FEC to readout)
#RCU.PEDSUB      <hex>        Default ALTRO pedestal subtraction (for each FEC to readout)
#FEC.ID          <fecid>      IF NOT GIVEN FEC.ID = 0 IS ASSUMED
#FEC.ZSUP        <hex>        Zero suppression pattern for this FEC
#FEC.PEDSUB      <hex>        Pedestal subratcuon pattern for this FEC
#ALTRO.ID        <altrochip>  Altro chip- default 0 if not given
#ALTRO.ACL       <hexvalue>   Active channels int this FEC and ALTRO chip
#ALTRO.ID        <altroid>    Altro channel within ALTRO chip - default 0 if not given
#ALTRO.ACL       <hexvalue>   Active channels of this FEC and ALTRO chip
# RCU IDENTIFICATION – NOT YET USED!!!!
#RCU.PARTITION   1   (0-7) To be used to identify physical location = RCUID
#RCU.SECTOR      1   (0-31) 1 for Lund Test Setup - FEC test
#RCU.SIDE        1   0 = Test beam, 1 = Test Setup somewhereRCU.ID          2
```

Example test beam 200907:

```
RCU.POWER       3FFF0FFE
RCU.READOUT     3FFF0FFE
RCU.ACL         FFFF
RCU.ZSUP        FF
RCU.PEDSUB      FF
RCU.PARTITION   1
RCU.SECTOR      1
RCU.SIDE        1
FEC.ID          0
#FEC.ZSUP       FF   Zero suppression pattern for this FEC
#FEC.PEDSUB     FF   Pedestal subtraction pattern for this FEC
#ALTRO.ID       0    Altro channel - default 0 if not given
#ALTRO.ACL      FFFF Active channels this FEC and ALTRO
#ALTRO.ID       7    Altro channel - default 0 if not given
#ALTRO.ACL      FFFF Active channels this FEC and ALTROFEC.ID          1
FEC.ID          2
FEC.ID          3
FEC.ID          4
FEC.ID          5
FEC.ID          6
```

FEC.ID	7
FEC.ID	8
FEC.ID	9
FEC.ID	10
FEC.ID	11
FEC.ID	12
FEC.ID	13
FEC.ID	14
FEC.ID	15
FEC.ID	16
FEC.ID	17
FEC.ID	18
FEC.ID	19
FEC.ID	20
FEC.ID	21
FEC.ID	22
FEC.ID	23
FEC.ID	24
FEC.ID	25
FEC.ID	26
FEC.ID	27
FEC.ID	28
FEC.ID	29
FEC.ID	30
FEC.ID	31

How to calculate RCU.READOUT and RCU.POWER:

Since the ALICE backplane have no dedicated power lines to the slots 13,14,15 on a branch, power bits 0,1,2 are used also for slots 13,14,15, i.e. one must set bit 0,1,2 respectively to power on these in RCU.POWER. The same is valid for power bits 16,17,18 and 29,30,31. Though there is no power line to 13,14,15,29,30,31 must these power bits be set if the readout bits are set. RCU.READOUT is one to one.

You can use the figure below as an example. First line is empty, can be used to fill in the values wanted. Second line is an example of modules to read out, a 1 means module is to be read, 0 is not to be read. Third line is an example on modules to power on. Module 0 is not readout, neither is 13, i.e. Power 0 and 13 is set to 0. Module 1 is not read out, but 14 is, therefore must power 1 and 14 be set. Module 2 is read out but not 15, i.e. Power 2 is set but not 15.

Module 16 is not read out, but 29 is, i.e. power 0 and 29 is set to 1. Module 17 is read out, but 30 is not, therefore must power 17 be set and 30 not set. Module 18 is not read out but 31 is, i.e. Power 18 and 31 must be set.

Fill in with 1=FEC is to be readout/powerd on, 0=FEC is not to be read out/powerd on

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Example 1=FEC is to be readout, 0 = FEC is not read out:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	1	0	1	0	1	1	1	1	0	1	0	0	1	0	0	1	1	0	1	1	1	1	1	1	1	0	0
A				S				7				A				4				D				F				C			

i.e. RCU.READOUT A57A4DFC

What to power on(remember 13 has same power as 0, 14 as 1, 15 as 2, 29 as 16, 30 as 17, 31 as 16)?

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	1	0	1	0	1	1	1	1	1	1	1	0	1	0	0	1	1	0	1	1	1	1	1	1	1	1	0
A				S				7				F				4				D				F				E			

i.e. RCU.POWER A57F4DFE

global-rcu.<runmode> runmode can be physics,pedestal(test to be implemented)

Settings common to all RCUs

#ALYTOIF.IECHK <value> check error (17:16)

0 no check on instruction error

1 TPC

2 PHOS

3 FMD

delay (15:14)

#ALTROIF.CSTB <value> programmable Command STroBe delay

1-4 cycles (value 0-3)

#ALTROIF.CLOCK (13:10) only two bits used

00 = 20 MHz

01 = 10 MHz

10 = 5 MHz

#ALTROIF.SAMPLES <value> samples/channel (9:0)

Only for book keeping (0-1000)

#

#TRGCONF.SOURCE <value>

1 - software, 2 - hardware, 4 - TTC, 6 - cable

#TRGCONF.MODE <value>

0 - level 0 mapped to level 1, 1 - level 1 mapped to level 1

#TRGCONF.LATENCY <value>

13 bits - L2 latency with respect to L1 (max 8191) for source = 2,6

#

#RDOMOD.RDYRX if 1: check of RDYRX in TTCrx

#RDOMOD.SPARSE if 1: read only channels with hits

#RDOMOD.EXESEQ if 1: Execute IMEM on SOD/EOD

#RDOMOD.MEB <value> number of buffers 0=4 buffers, 1 = 8 buffers
 #
 #MAIN.BPVERSION <value> mapping structure of ACTFECLIST to card switch lines
 # 0: TPC and FMD
 # 1: PHOS

Example test beam 200907:

ALTROIF.SAMPLES	1000
ALTROIF.CLOCK	0
ALTROIF.CSTB	0
ALTROIF.IECHK	0
TRGCONF.SOURCE	6
TRGCONF.MODE	1
TRGCONF.LATENCY	8191
RDOMOD.RDYRX	0
RDOMOD.SPARSE	0
RDOMOD.EXESEQ	0
RDOMOD.MEB	0
MAIN.BPVERSION	0

global-altro.<runmode> runmode can be physics,pedestal(test to be implemented)

Settings common to all ALTROs in all FECs
 # All values are decimal!
 #Zero suppression threshold and offset
 #ZSTHR.OFFSET <value> Offset to be added to the signal
 #ZSTHR.ZS_THR <value> Zero suppression threshold
 #Second baseline correction threshold
 #BCTHR.THR_LO <value> Lower threshold of the acc. window
 #BCTHR.THR_HI <value> Upper threshold of the acc. window
 #Trigger configuration
 #TRCFG.ACQ_START <value> Number of cycles to wait before start
 #TRCFG.ACQ_SHIFT <value> shift the start of acquisition depending on ALTRO
 # 0 = no shift
 # else
 # chip 0 1 2 3 4 5 6 7
 # shift 0 1 2 3 0 1 2 3 i.e. each side of the card
 # Used only for test of reducing noise produced by the clock, set to 0 else.
 #TRCFG.ACQ_END <value> Cycles to sample
 #Data path configuration
 #DPCFG.FBCM <value> First baseline correction mode
 # 0: din - fpd = do not use lookup table
 # 8: f(din) - fpd = use lookup table *STANDARD SETTING!*
 # 9: f(din-vpd) - fpd = use lookup table subtract self pedestal
 # 10: f(t) - fpd = read pedestal memory as data
 #DPCFG.POLARITY <0,1> Polarity, when set ADC data inverted
 #DPCFG.SBPRES <value> Presamples excluded from 2nd baseline correction
 #DPCFG.SBPOST <value> Postsamples excluded from 2nd baseline correction
 #DPCFG.SBENB <0,1> Enable second baseline correction
 #DPCFG.GLITCH <value> Glitch filter for 0-suppression (0-3)
 #DPCFG.ZSPRES <value> Presamples excluded from suppression (0-3)

#DPCFG.ZSPOST	<value>	Postsamples excluded from suppression (0-7)
#DPCFG.ZSENB	<0,1>	zero skip enable bit
# this bit is overridden from the run control		
#BFNPT.PWSV	<0,1>	when set, power save
#BFNPT.FLT_EN	<0,1>	enable digital filter
#BFNPT.NBUF	<0,1>	0: 4 buffers, 1: 8 buffers
#BFNPT.PTRG	<0-15>	Pretrigger samples

Example test beam 200907:

ZSTHR.OFFSET	0
ZSTHR.ZS_THR	5
BCTHR.THR_LO	0
BCTHR.THR_HI	0
TRCFG.ACQ_START	0
TRCFG.ACQ_SHIFT	0
TRCFG.ACQ_END	300
DPCFG.FBCM	8
DPCFG.POLARITY	0
DPCFG.SBPRES	0
DPCFG.SBPOST	0
DPCFG.SBENB	0
DPCFG.GLITCH	2
DPCFG.ZSPRE	3
DPCFG.ZSPOST	7
DPCFG.ZSENB	1
BFNPT.PWSV	0
BFNPT.FLT_EN	0
BFNPT.NBUF	0
BFNPT.PTRG	0

pca16.cfg

This file is used to set default settings, is overwritten with values when doing PcaLoad from run control.

```
# PCA16 settings
# default is same for all FEC and RCU
# CAN NOT YET BE SET INDIVIDUALLY
#PCA16.PREAMP_ENABLE    <value> 1 = enable preamp mode
#PCA16.GAIN              <value> 0-3, gain setting
#PCA16.SHAPER            <value> 0,1,3,7, shaping setting
#PCA16.SHUTDOWN          <value> 1 = shutdown the PCA16
#PCA16.POLARITY          <value> 0,1 should be 1 for GEMs
#PCA16.DECAY_BIAS        <value> 0-2505, value to DAC
# Shiftreg : BE DAC : 500    comment showing the shift register and DAC value
```

Example test beam 200907:

PCA16.PREAMP_ENABLE	0
PCA16.GAIN	3
PCA16.SHAPER	7
PCA16.SHUTDOWN	0

PCA16.POLARITY 1
PCA16.DECAY_BIAS 500

When loading the PCA is a delay needed in the program to allow the FEC to react to commands. This delay may be configurable in future version.

Pedestal files

The pedestals for the ALTRO chip are read from files. The location of these files are in CONFIG_DIR which is defined in readout.h or in files.cfg. There is one file for each RCU as the name pedestals-<rcuid>-<pcasettings = %X-%d,shiftreg,dac>.dat. The pedestal file contains the fixed decimal pedestal value for each channel Since pedestals depend on the pca16 settings must there be a pedestal file for each pca16 setting used.

Pedestal files are taken in special runs with a pulser trigger and without beam. It is the monitoring program that calculates the pedestals, and the monitoring program must therefore be running.

PCA16 SHIFTREG SETTING BITS ARE:

0: preamp enable
1-2: gain (0-3)
3-5: shaper (0,1,3, or 7)
6: shutdown
7: polarity

pedestal-<rcuid>-<pcasettings = %X-%d,shiftreg,dac>.dat

FORMAT:

#RCUID n
##*PCA 008003E8 (shiftreg << 16) | (DAC setting)
##*EVENTS 1600
##*RUN 6203
<channel> <fixed pedestal - decimal> [<fixed pedestal – float> <rms – float> <entries>]

Log files

Logfiles are written to the directory LOG_DIR as defined in readout.h, or as defined in files.cfg. There are three types of logfiles; message log, run log, and monitor log.

Message log file – Information and debug output

Run log file – Status of run, started, stopped, paused, continued etc...

Monitor log file – status of the monitor server threads

These files have a very debugging content which is not yet non-expert readable.

Data files

The binary data files get the name readout-<runnb>_<filenb>.dat where <runnb> is the RUN.NB taken from the file daq.cfg. The location of these files are in DATA_DIR which is defined in readout.h. When the file size exceeds the value MAX_FILE_SIZE (defined in loghandler.h), then the file is closed and a new one is opened with <filenb> incremented by one. The first file in a run has <filenb> = 0. Before the file is closed is an End Of File record written. In the beginning of the new file is a

Beginning Of File record written. MAX_FILE_SIZE must be given in 32-bit words.

Files

Source code and executable are in readout-v4.2-pro.

dbox.c,.h	DBOX handling
fec2rorc_lib.c,.h	RORC library
fec2rorc_version.h	version of RORC library
fec_rcuerror.h	RCU error codes
fec2rorc_lib_structs.h	needed by fec2rorc_lib.h
ilc_altro.c,.h	ALTRO routines
ilcfec2rorc_rcu.c,.h	RCU/ALTRO library
ilc_rcu.c,.h	RCU routines
loghandler.h	data format header definitions
monhandler.c,.h	monitoring threads
pca16.c,.h	PCA16 configuration
readout.c,.h	all readout, event handling
runhandler.c,.h	handling the network run control
typedefs.h	some common definitions
utility.c,.h	power to FEC/load pca16 routines

compile the code:

compile-readout

executable:

ilcrorc-readout

Changes

2.3 * update thread - send *UPD

2.4 * change of ilcserver port 2002->2000

* initialize clientlen properly in runhandler.c

2.5 * try to read from expected FECs to be powered on

and send in *POW Pi <powered>to GUI

* send runnb/runmode to monitor program in monOnline if first request in new run. Last runnb/runmode sent if no run.

3.0 * Start implement DBOX handling: dbx.cfg = trigger config

4.0 * New RCU firmware: v2.0 (no RcuLoadLookup)

4.1 *20081208 Implement loading/checking of pedestal memory using RCU instruction memory

*20081208 Implement DBOX handling according to DBOX documentation

*Implement DBOX data into event header: dbxtime,dbxevtn,tlevtn

*Add PCA settings in run header:

```
pca->preamp_enable = pca->shiftreg & 1;
pca->gain = (pca->shiftreg >> 1) & 0x03;
pca->shaper = (pca->shiftreg >> 3) & 0x07;
pca->shutdown = (pca->shiftreg >> 6) & 1;
pca->polarity = (pca->shiftreg >> 7) & 1;
pca->decay_bias = pca->dac;
SOR dataword after times = pca->shiftreg << 16 | pca->dac;
```

Add ALTRO settings in run header

pedestal subtraction, zero suppression, some ALTRO/RCU register

*Config files:

copy configfiles to RUNCONF_DIR/<name>-<runnb>.cfg

*If pedestal subtraction done

Read pedestal file used as: pedestals-<id>-<pcasettings = %X-%d,shiftreg,dac>.dat

Store pedestal file used as: RUNPED_DIR/pedestals-<id>-<runnb>.dat

*Write ALTRO test data in pedestal memory (i.e. lookup table):

- 0) do not use lookup table
- 1) DPCFG = 10, i.e. $f(t) - f_{pd}$
- 2) no pedestal run
- 3) different cluster samples to the pedestal memory - add 0x2AA, 0x133, some peaks.
2AA = 4, 155 = 5, PEAKS = 6

*Insert DRORC_REVISION into rmap instead of a DEFINE in utility.c

TO BE -> HEPFORGE (DONE)

*20090111 The READOUT_MEMORY was overwritten by loading the lookup table using the RCU instruction memory. Changed in RcuLoadLookup.

*20090118 Removed in dbxPoll (dbx.c) that it was calling disconnectclient. Had wrong arguments, moved that and error output to be after dbxPoll in readout.c. inserted `#include <stdlib.h>` in dbx.c

*20090119 in ilcStartOfRun:

in ilc-physmem.h Number of FIFOs used 128->16

/* for safety - empty the data area with spurious events - if any */

```

        if (daq_rorc.active[ch]) {

<   else {
<   if (ag.dpcfg_zsenb) daq_param.zerosup = 1;
<   else daq_param.zerosup = 0;
<   }

        *added zerouppression and pedestalsubrtaction flags to runheader:
<   sorheader.runsettings = 0;
<   if (daq_param.zerosup) {
<   sorheader.runsettings |= 1;
<   }
<   if (daq_param.pedsub) {
<   sorheader.runsettings |= 2;
<   }

    *in loghandler.h
        int runsettings;
        } SORHEADER;

    *in readout.c ilcSetActiveFec
<   if (ret) {
<   sprintf(msgbuf,"%s: *ERROR* AltroReadPedestalFile %s returned with error
%d\n",__FUNCTION__,filename,ret);
<   daq_status.feerrors++;
<   }

```

TO BE -> HEPFORGE

20090123

added the possibility to read the lookuptable from a file
lookuptable.dat, manipulated to get pulses at different times
when simulating events

20090124

ilc_altro.c
bug in AltroDisableZerosup: used data2 to write value, should be data1.

20090125

ilc_altro.c
bug in AltroShiftAcqStart: data2 -> rdata: else if (data != rdata) {

20090126

ilc_altro.c
in RcuLoadLookup/RcuReadLookup insert myacl to load only active channels in chip,
and for all channels in an active chip

add delays from 100 -> 1000 microseconds to allow 5MHz sampling.

20090127

loghandler.h - changed data format version to 410

typedefs.h, readout.h, readout.c , run.cfg - added the possibility to stop the run at a RCU readout error (ERR_REG2), for test of hardware readout.

20090205

can stop run if roerror if error flagged in RCU trailer

data format read from run.cfg

typedefs.h : onroerror, dataformat

readout.c: added end run if roerror. dumpRunConfig(). use daq_param.dataformat in SOR/BOF header

added one DRORC: added the initialize of 4 memory buffers in ilcOpen

20090206

To handle mapping of channels correctly:

```
// if ( (ch = ilcFindRorcSerial(rmap[j].serial,j,&daq_rorc)) >= 0) {  
    if ( (ch = ilcFindRorcSerial(rmap[j].serial,rmap[j].channel,&daq_rorc)) >= 0) {
```

20090209

copyfiles: only if logging data

version 4.2

20090415

make it possible to reread event, i.e. load active list twice.

RUN.REREAD flag in run.cfg

dataformat = (RUN.DATAFORMAT << 16) | DAQ_VERSION

0 - reversed data

#RUN.DATAFORMAT 0

1 - 3 10 bits words in each 32 bit word.

RUN.DATAFORMAT 1

*20090625

in run.cfg

----- VARIABLES READ ONLY ONCE AT PROGRAM START

----- THESE VALUES MAY BE CHANGED FROM RUN CONTROL

PROG.INIT

#Load lookup tables at power on - read only program start

LOOKUP.POW 1

#Load lookup tables at start daq - read only program start

LOOKUP.START 0

Modified readRunConfig and dumpRunConfig

added _USE_NOHW flag

Implement timeout waiting for RCU - if any RCU is missing the trigger (if all miss DBOX takes care)

To be done

Make all network ports configurable via cfg file.

Make the sampling clock individual for each RCU.

in rcu-<id>.cfg ALTROIF.CLOCK

Option to reread drorc.cfg and found DRORCs/RCUs at any time

Test with full system

Test with true DBOX/TLU

Implement common DAQ