READOUT

version: readout-v4.0

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 dbox.cfg at Start DAQ command.

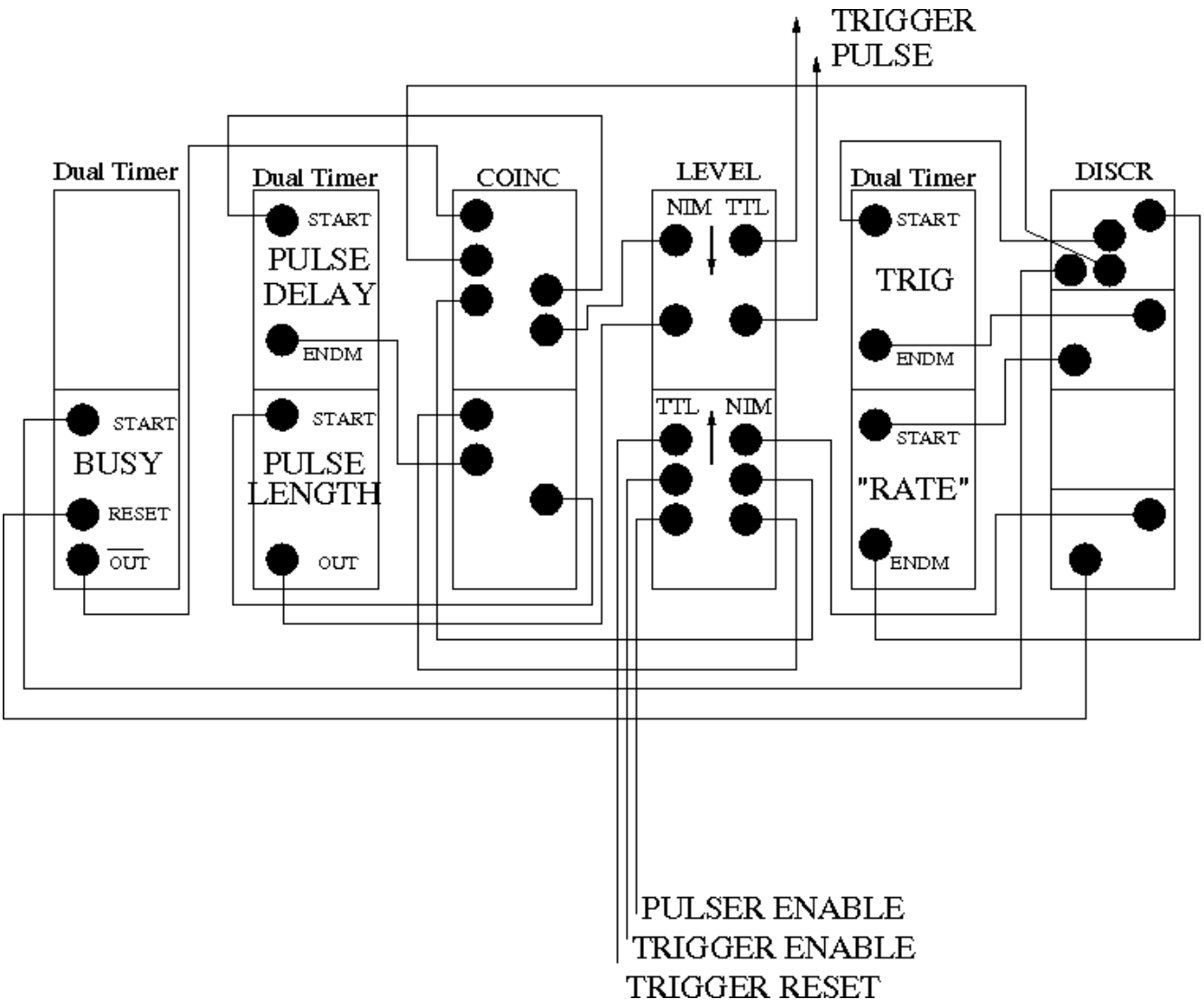

Figure RO-NIM: The NIM trigger logic used in Lund together with the parallel port trigger mode.

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 udpates, 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.

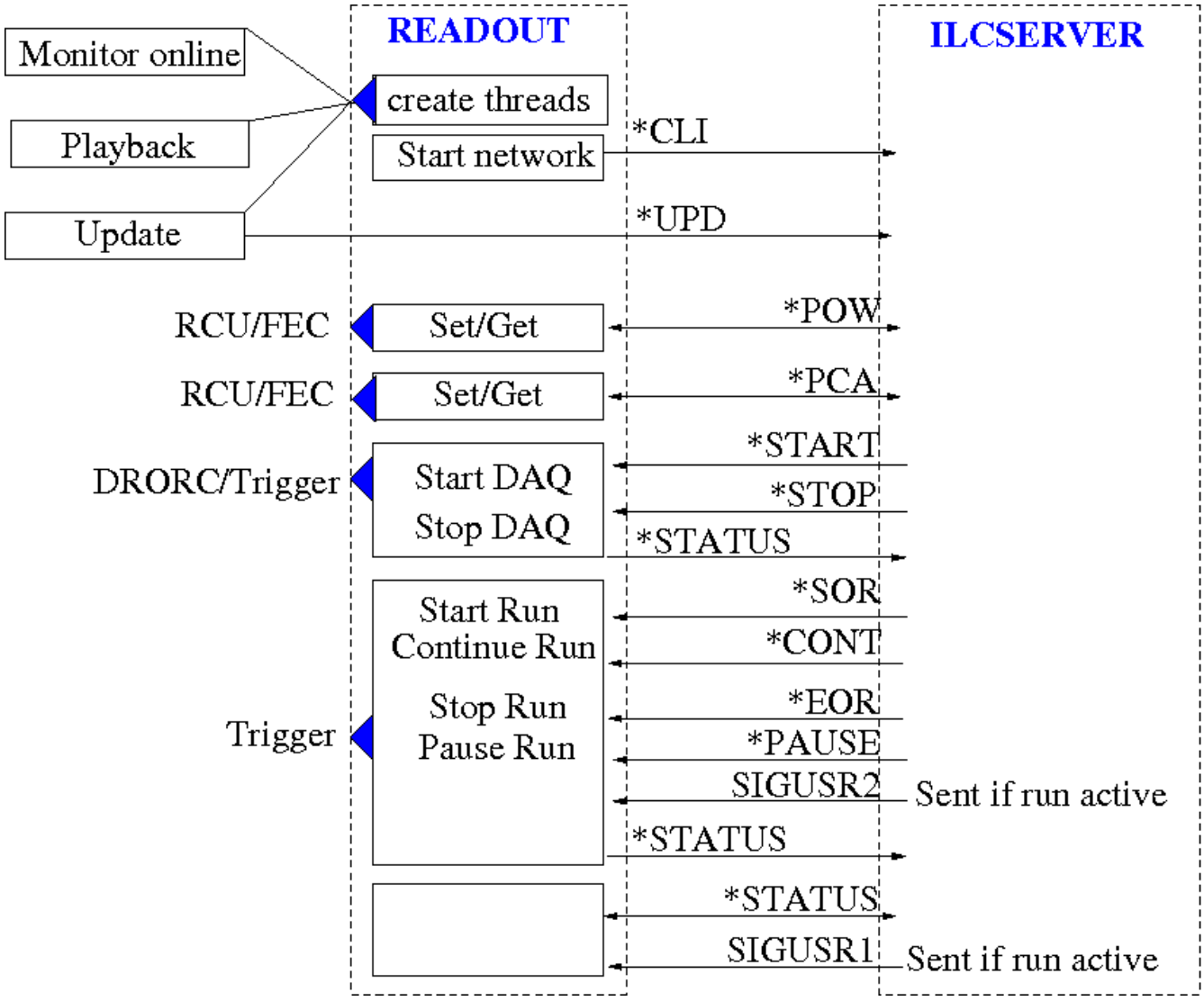

Figure RO-1: The protocol between the ilcserver and readout.

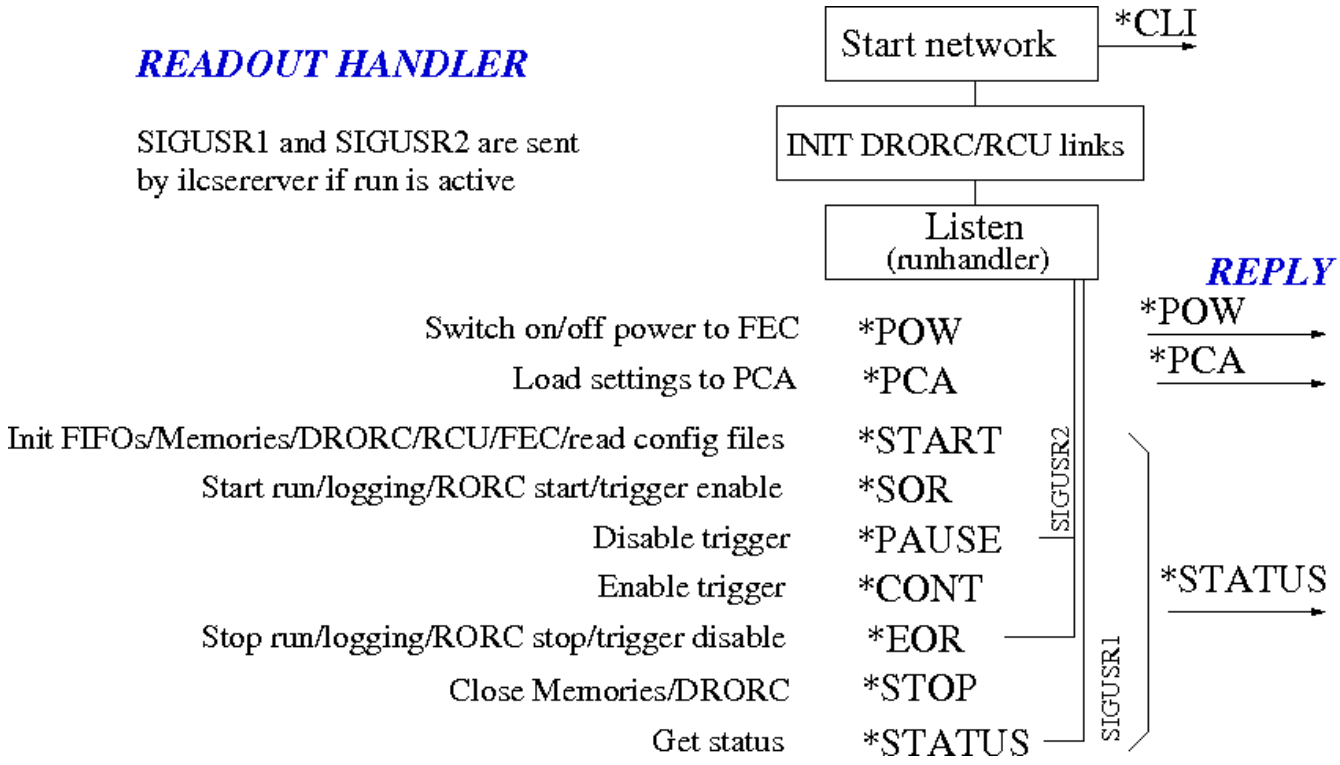

Figure RO-2: Schematic view of run handling in the readout handler.

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.

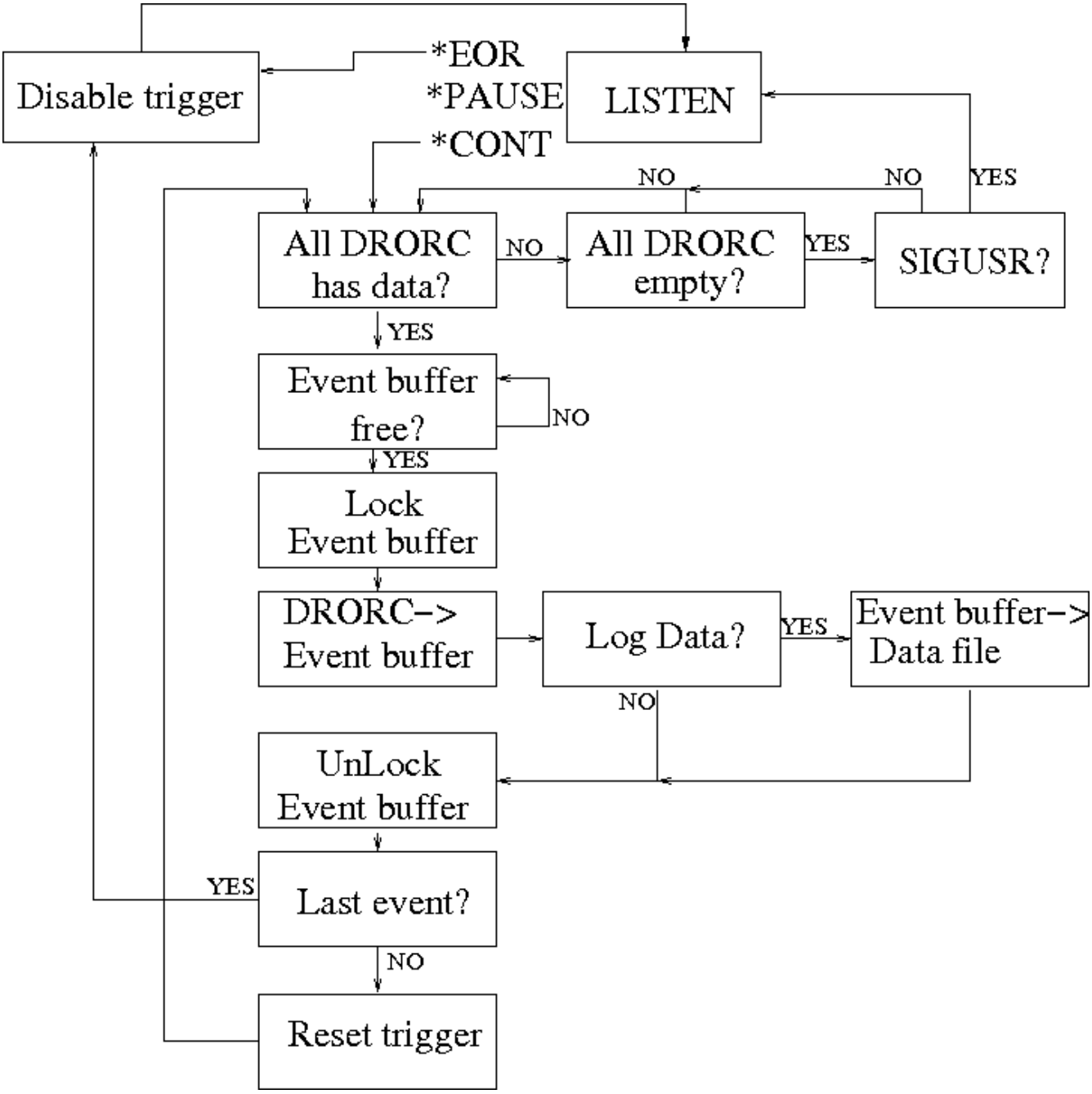

Figure RO-3: Event loop logic

DBOX emulator

in the the catalog dboxserver 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.

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
#
```

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 runlog is always called FILE.RUN-<runnb>
# Note 2. 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)
```

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)
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

daq.cfg

General DAQ configuration
read and written at each start of run
FORMAT:
#RUN.NB <nb> last run number, incremented for each run

drorc.cfg

This file contains the mapping between
RCU and DRORC channel
FORMAT:
RCU.ID <id> id of RCU [0..3]
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.

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

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

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

```
# Settings common to all RCUs
#ALTROIF
#IECHK check error (17:16)
#   00 no check on instruction error
#   01 TPC
#   10 PHOS
#   11 FMD
#   delay (15:14)
#CSTB (programmable delay (Command STroBe delay
#   1-4 cycles (value 0-3)
#CLOCK (13:10) only two bits used
#   00 = 20 MHz
#   01 = 10 MHz
#   10 =  5 MHz
#SAMPLES   samples/channel (9:0)
#
ALTROIF.SAMPLES          1000
ALTROIF.CLOCK            0
ALTROIF.CSTB             2
ALTROIF.IECHK            0
#
#TRGCONF
#SOURCE 1 - software, 2 - hardware, 4 - TTC, 6 -  cable
#MODE  0 - level 0 mapped to level 1, 1 - level 1 mapped to level 1
#LATENCY 13 bits - L2 latency wrt to L1 (max 8191) source = 2,6
#
TRGCONF.SOURCE  6
TRGCONF.MODE    1
TRGCONF.LATENCY 8000
#
#RDOMOD
#RDYRX (3) set: check of RDYRX in TTCrx
#SPARSE (2) set: read only channels with hits
#EXESEQ (1) set: Execute IMEM on SOD/EOD
#MEB (0) number of buffers 0=8 buffers, 1 = 4 buffers
#
RDOMOD.RDYRX    0
RDOMOD.SPARSE   0
RDOMOD.EXESEQ   0
RDOMOD.MEB      1
#
# MAIN
#BPVERSION mapping structure of ACTFECLIST to card switch lines
# 0: TPC and FMD
# 1: PHOS
#
MAIN.BPVERSION  0
#
```

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

```
# Settings common to all ALTROs in all FECs
# FORMAT
# All values are decimal! Might change for some settings
#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_END          <value>         Cycles to sample
#Data path configuration
# f(din) - fpd = use lookup table
#DPCFG.FBCM       <value>         First baseline correction mode (8 = use lookup)
#DPCFG.POLARITY<0,1>  Polarity, when set ADC data inverted
#DPCFG.SBPRE      <value>         Number of presamples excluded from 2nd baseline correcytion
#DPCFG.SBPOST     <value>         Number of postsamples excluded from 2nd baseline correcytion
#DPCFG.SBENB      <0,1>           Enable second baseline correction
#DPCFG.GLITCH     <value>         Glitch filter for 0-suppression
#DPCFG.ZSPRE      <value>         Presamples excluded from suppression
#DPCFG.ZSPOST     <value>         Postsamples excluded from suppression
#DPCFG.ZSENB      <0,1>           zero skip enable bit
#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
```

pca16.cfg

```
# 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> 0-7, 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
```

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. There is one file for each RCU. The pedestal file contains the fixed decimal pedestal value for each channel. Optionally, using the standalone pedestal calculation program, are also the calculated values of pedestals and rms (using entries values) stored. These are ignored

when loading the pedestals.

pedestal-<rcuid>.dat

FORMAT:
#RCUID n
<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 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

readout.c, readout.h	all readout, event handling
runhandler.c, runhandler.h	handling the network run control
monhandler.c, monhandler.h	monitoring threads
ilc_altro.c, ilc_altro.h	ALTRO routines
ilc_rcu.c, ilc_rcu.h	RCU routines
pca16.c, pca16.h	PCA16 configuration
utility.c, utility.h	set power to FEC/load pca16 routines
loghandler.h	data format header definitions
ilc-physmem.h	physical memory definitions
fecrorc_lib.c,fecrorc_lib.h	adopted routines for FEC handling
fecrorc_RCUdefines.h	RCU definitons
fec2rorc_lib_structs.h	needed by fec2rorc_lib.h

fec2rorc_version.h	needed by fec2rorc_lib.c (version number)
typedefs.h	some common definitions

The utility.c need for fec2rorc_lib.c should be moved to use fecrorc_lib.c.

dbox.c, dbox.h	DBOX communication
fec2rorc_lib.c, fec2rorc_lib.h	low level DRORC/DDL routines
fec2rorc_lib_structs.h	structure for DRORC channel open
fec2rorc_version.h	ALICE fec2rorc library version
fec_rcuerror.h	error codes
ilc_altro.c, ilc_altro.h	high level ALTRO routines
ilcfec2rorc_rcu.c, ilcfec2rorc_rcu.h	low level RCU/FEC routines
ilc_rcu.c, ilc_rcu.h	high level RCU routines
ilc-physmem.h	physmem handling
loghandler.h	data format header definitions
monhandler.c, monhandler.h	monitoring threads
pca16.c, pca16.h	PCA16 configuration
readout.c, readout.h	all readout, event handling
runhandler.c, runhandler.h	handling the network run control
typedefs.h	some common definitions
utility.c, utility.h	high level set power to FEC/load pca16 routines

compile the code:
compile-readout
executable:
ilcrorc-readout