

READOUT

version: readout-v4.1 **DRAFT**

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.

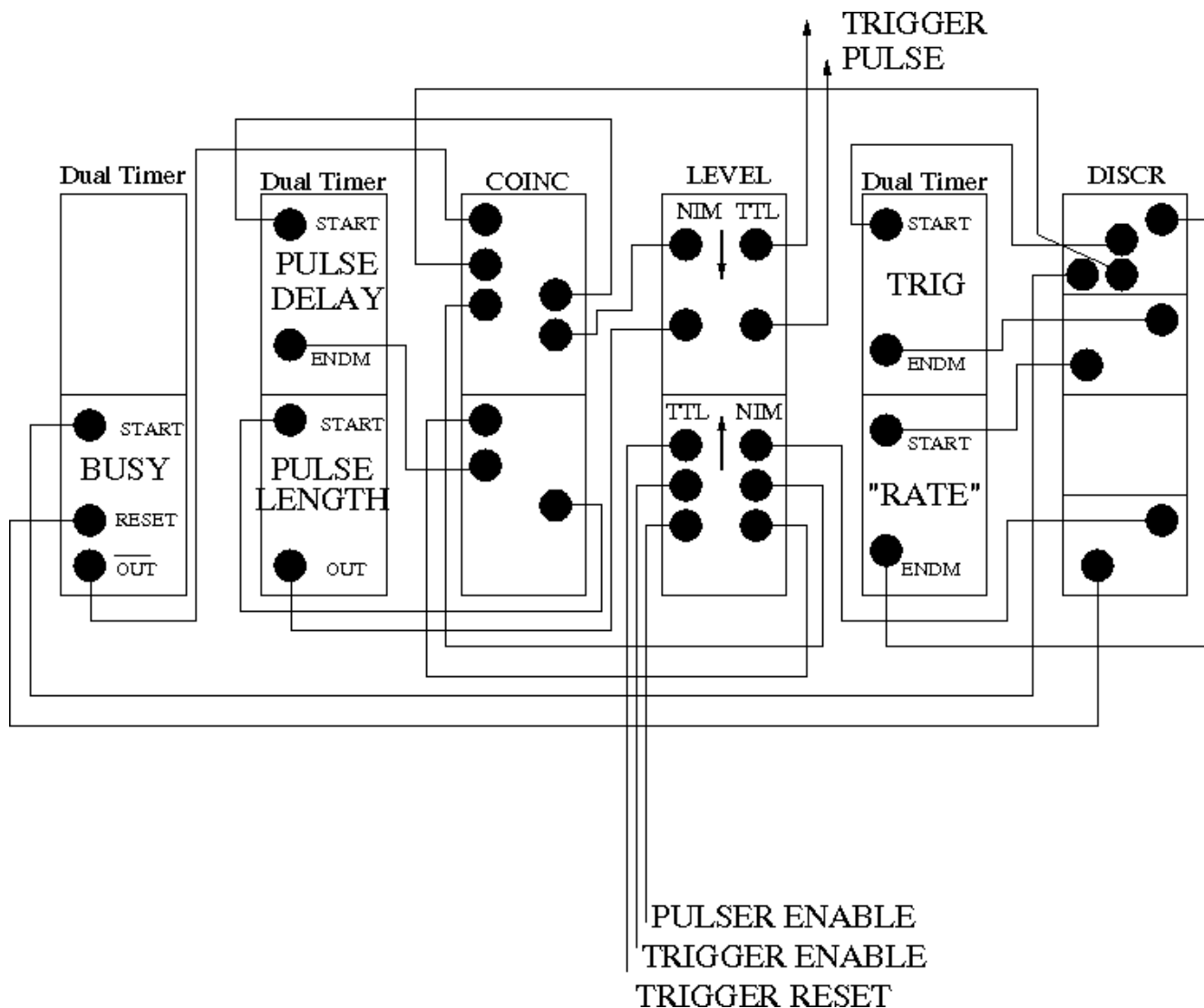


Figure RO-NIM: The NIM trigger logic used in Lund together with the parallel port trigger mode.

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 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.

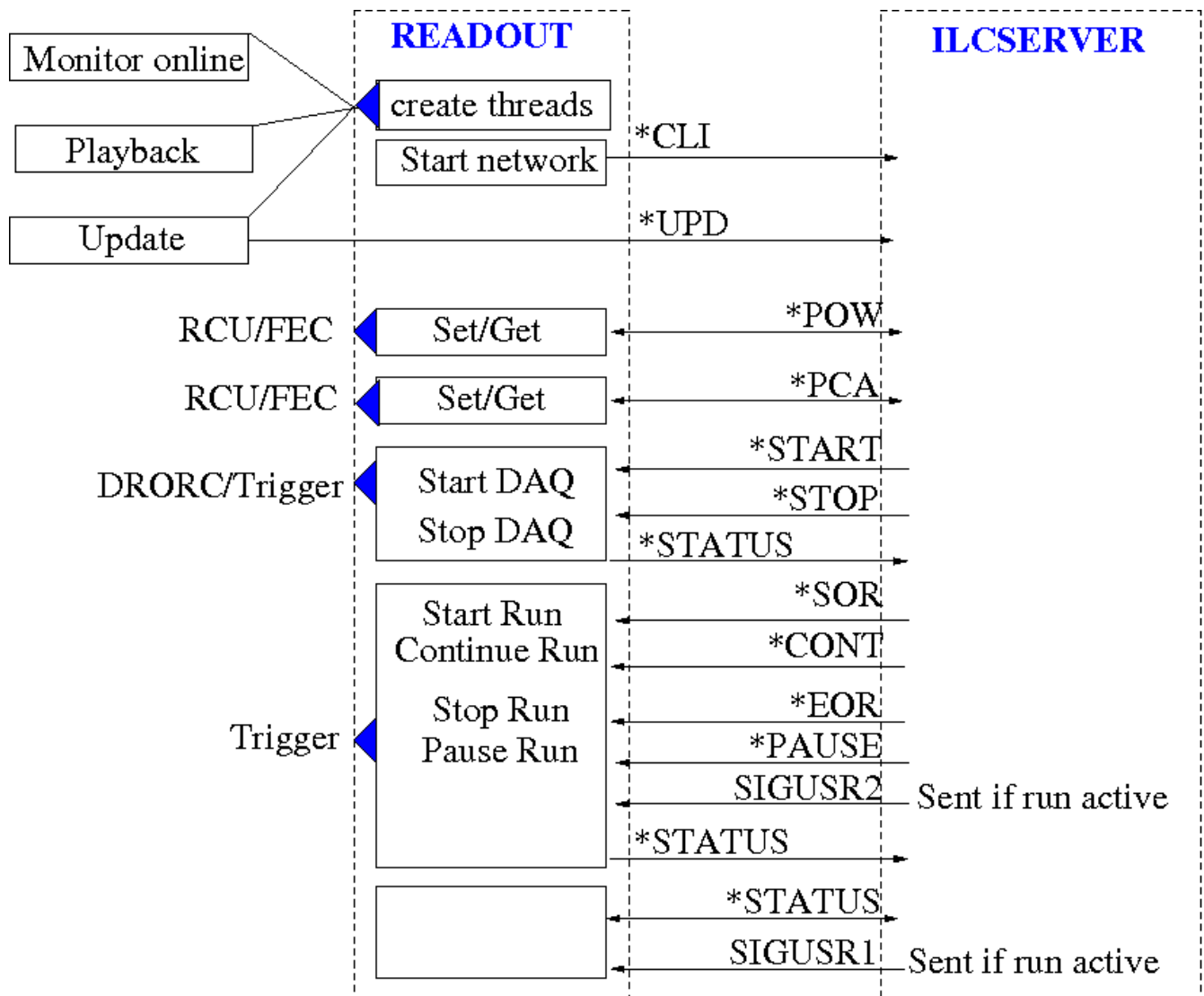


Figure RO-1: The protocol between the ilcserver and readout.

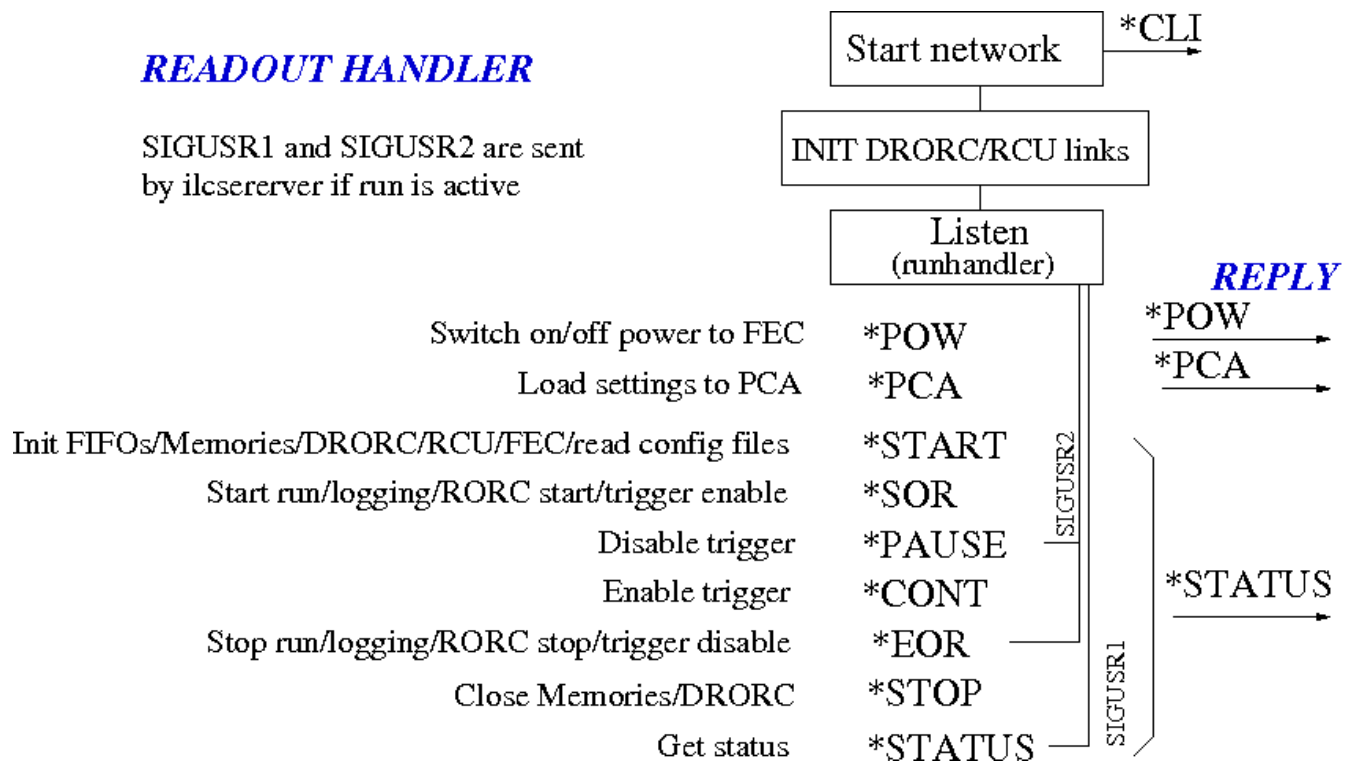


Figure RO-2: Schematic view of run handling in the readout handler.

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.

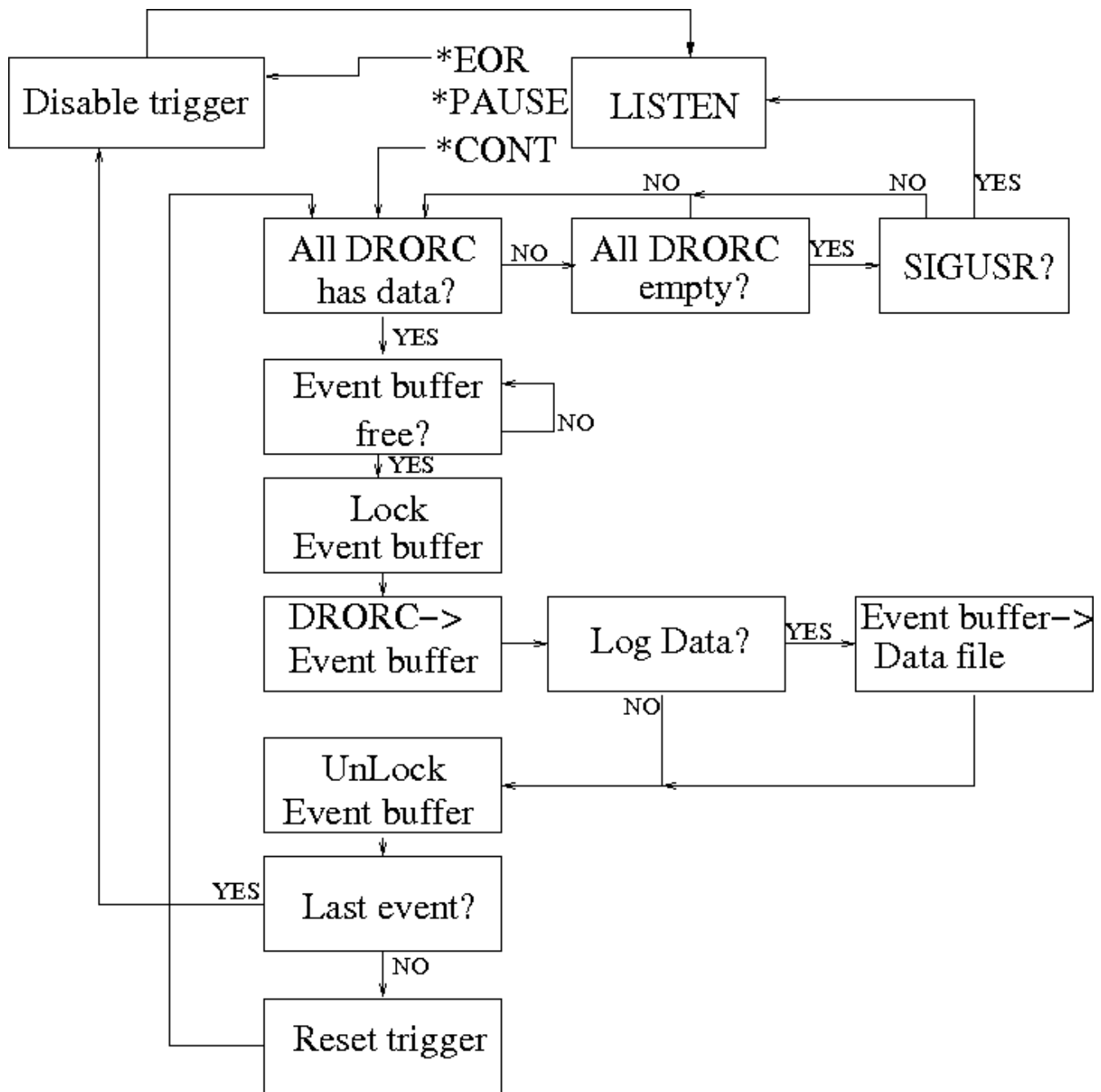


Figure RO-3: Event loop logic

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.

dbox.cfg

```
# configuration file for trigger and the distribution box
# TRIG.SOURCE <source> 1=Parallel 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
```

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)
# 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 main version 41*10 and reversed data = 1
```

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

```

Since in the ALICE backplane are there no dedicated power lines to slot 13,14,15 on a branch, are bits 0,1,2 used also for slots 13,14,15, i.e. one must set bit 0,1,2 respectively to power on these in RCU.POWER. RCU.READOUT is one to one.

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/cable , 4 - TTC, 6 -cable (OBSOLETE)
#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       2
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_SHIFT <value> shift the start of aquisition 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
#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)
# din - fpd do not use lookup table
#DPCFG.FBCM      0
#-> f(din) - fpd = use lookup table subtract fixed pedestal
#DPCFG.FBCM      8
# f(din-vpd) - fpd = use lookup table subtract self pedestal
#DPCFG.FBCM      9
# f(t) - fpd = read pedestal memory as data
#DPCFG.POLARITY<0,1>    Polarity, when set ADC data inverted
#DPCFG.SBPRES    <value>    Number of presamples excluded from 2nd baseline
correction
#DPCFG.SBPOST    <value>    Number of postsamples excluded from 2nd baseline
correction
#DPCFG.SBENB    <0,1>    Enable second baseline correction
#DPCFG.GLITCH    <value>    Glitch filter for 0-suppression
#DPCFG.ZSPRES    <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 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.

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 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

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

*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) - fpd

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

To be done

Make all network ports configurable via cfg file.

Make the sampling clock individual for each RCU.

in rcu-<id>.cfg ALTROI.CLOCK

Option to reread drorc.cfg and found DRORCs/RCUs

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

Test with full system

Test with true DBOX/TLU

Implement common DAQ