Auger North RDA Local Station Controller User Manual
Preliminary

Laurent Guglielmi (laurent.guglielmi@apc.univ-paris7.fr)
Bernard Courty (bernard.courty@apc.univ-paris7.fr)
Stéphane Colonges (stephane.colonges@apc.univ-paris7.fr)
Cyril Dufour (cyril.dufour@apc.univ-paris7.fr)

April 23, 2013

Abstract

This paper describes the hardware and software of the Auger North Local Station Controller prototype number 2.

Contents

List of Tables

List of Figures

1 Hardware Description

The PCB size is 160x200 mm.

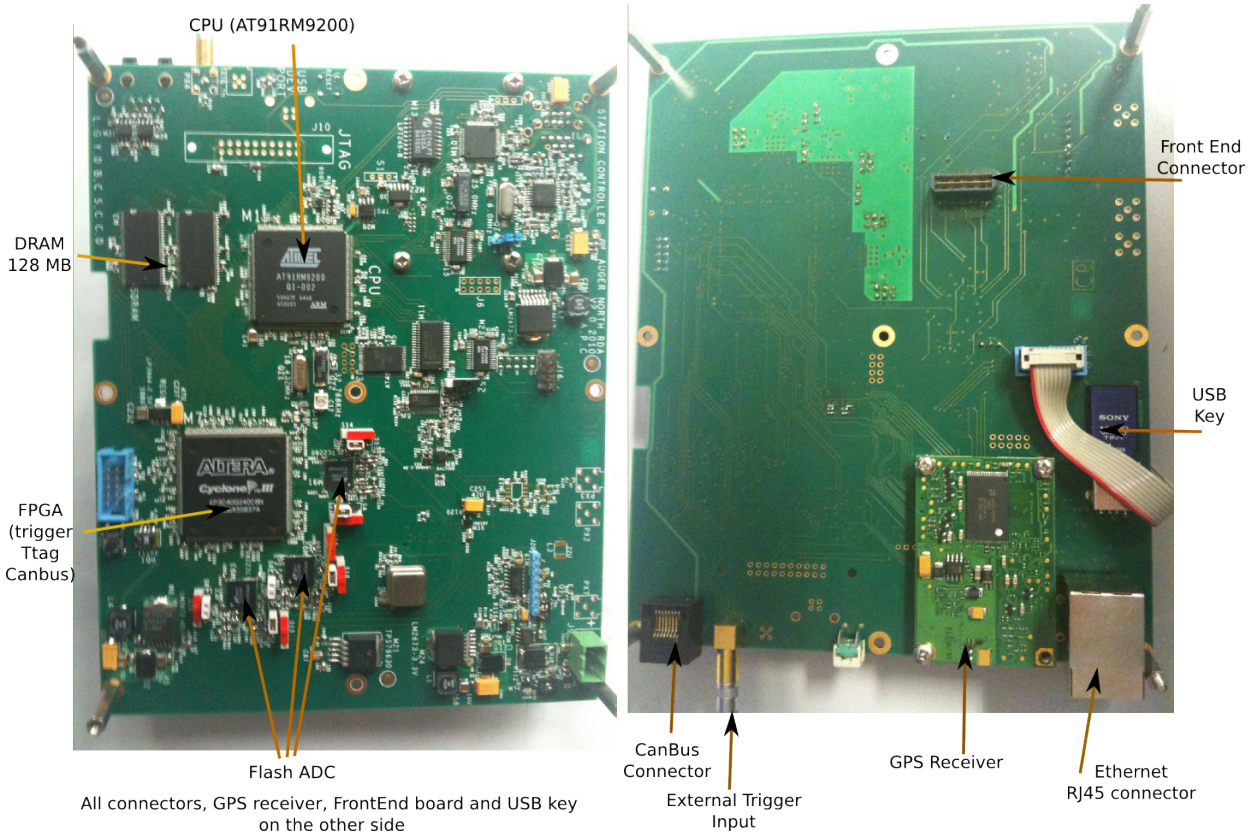

Figure 1: Up and down sides. Connectors are all on the down side.

1.1 LSC Content

The schematics are available in AppendixG

1.1.1 Ethernet interface

1.1.2 CanBus Interface

Phillips sja1000 implemented as an IP in the Cyclone FPGA.

1.1.3 Gps Receiver

1.1.4 FPGA

Cyclone III housing the Time Tagging, Canbus interface and the Trigger firmware.

1.1.5 Serial IOs

1.1.6 Ram Disk

/ram0 is an 8 Megabytes ram disk, created at each reboot/power on of the LSC. Note that this is not a “static” ram disk, as it is re-initialized at each reset. It is mostly used to store temporary files (log files mostly), to store downloaded files and to house a copy of the USB disk services and acquisition executables and configuration files (TBDone).

1.1.7 USB key

A 4 Gbytes USB key, with 2 partitions:

- / : the root file system, 1 Gbytes
- /scr: Acquisition use, 3 Gbytes

1.1.8 Slow Control ADC

Analog Device AD7490, 12 bits, 16 channels.

Channel	Usage	Comment
0	Front-end	
1	Front-end	
2	Front-end	
3	Front-end	
4	Battery Voltage	12 Volts
5	Front-end Voltage	12 Volts
6	FPGA Voltage	1.2 Volts
7	VCC	3.3 Volts
8	Core Voltage	1.8 Volts
9	Front-end current	
10	VCC Current	
11	Battery Current	
12	V5 Voltage	5 Volts
13	Unused	
14	Unused	
15	Unused	

Table 1: ADC AD7554 channels assignement. The currents are measured "on the fly" and thus are very changing. Only the average over several measures is meaningful.

1.1.9 Slow Control DAC

Texas Instrument DAC7554, 12 bits, 0-2.5 Volts. Channel 0 is the FrontEnd High Voltage setting.

1.1.10 Slow Control Temperature/Hygrometry

Sensirion SHT11 Humidity and Temperature sensor. The sensor is installed directly on the LSC PCB. It measures the temperature and the relative humidity.

1.1.11 Flash ADCs

3 dual channels 10 bits, 100 MHz Flash ADCs LTC2280. The dynamic is selectable by a switch, either [-1,+1] V or [-0.5,+0.5] V. The Current setting: [-1, +1] Volts, ==> 2 V = 1024 adu. As a consequence, the base line (0) is around 512 ADU; this cannot be modified.

Each FADC chip can be powered by a switch. Unused FADC should be powered off to minimize power consumption.

Switch (PCB label)	Sample	Value	FE Channel	FADC (PCB label)
S10	1	Anodex1	anach2	M92 - Pin 16/15
S11	3	Dynode #5	anach4	M92 - Pin 1/2
S12	0	Anodex30	anach1	M91 - Pin 1/2
S13	5	0x3FF	unused	M93
S14	2	Anodex0.1	anach3	M91 - Pin 16/15
S15	4	0x3FF	unused	M93

To change the synamic, move the jumper at position '0' on the following switches

Switch	Channel
S7	Anodex1
S6	Dynode
S4	Anodex30
S5	Anodex0.1

1.1.12 Front-End interface

One SAMTEC connecteur. Assignement of the pins is shown in the table 2. The format of the samples on 2 consecutive 32 bits word is shown in figure 2 for the Test (Courty) FPGA Trigger Firmware and in figure 3 for the final FPGA Trigger Firmware.

Signal	SAMTEC Pin	Name	Comments
Anode x 30	30	anach1	FADC
Anode x 1.	31	anach2	FADC
Anode x 0.1	34	anach3	FADC
Dynode	35	anach4	FADC
HV Dac	5	hvdac1	HV setting, 0-2.5 Volts. 1 V = 727 HV
HV Dac	6	hvdac2	

Table 2: Pin assignement between the SAMTEC connector and the FADC's.

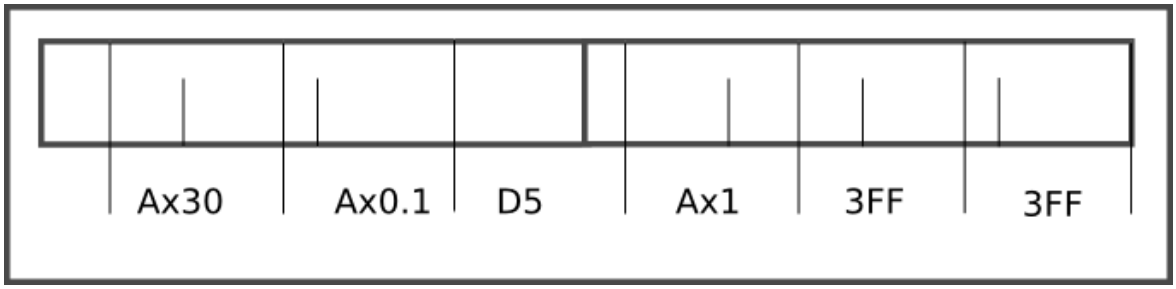

Figure 2: Test FPGA firmware: format of the samples .

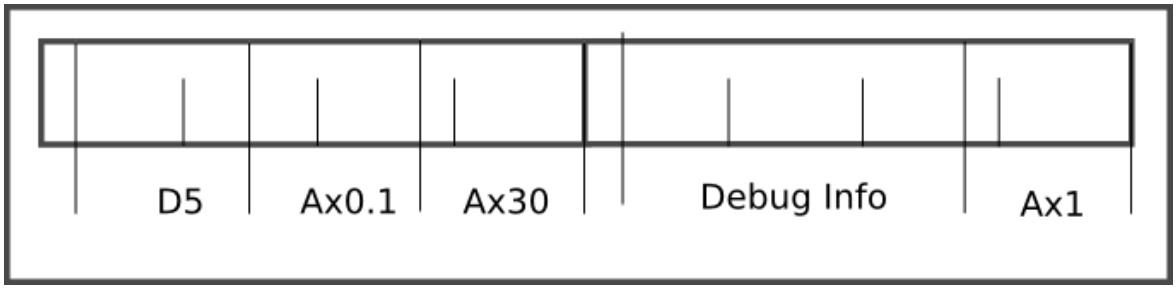

Figure 3: Final FPGA Firmware: format of the samples.

1.2 Setup

1.3 IRQs

The table 3 describes the interrupts specific to the Auger North Acquisition, not all the interrupts defined in the system.

PIO	Pin	Periph A	Periph B	Assignement	Comment
PB28	123	FIQ	-	FIQ	Unused, too difficult to use!
PB29	124	IRQ0	-	1PPS - AIC 25	Positive Edge, Priority 7
PA26	71	TWCK	IRQ1	EVTCLKS - AIC 26	Positive Edge, Priority 5
PA25	72	TWD	IRQ2	EVTCLKF - AIC-27	Positive Edge, Priority 6
PA23	69	TXD2	IRQ3	TXD2	Not an IRQ
PA2	44	SPCK	IRQ4	SPCK	...
PA3	45	NPSC0	IRQ5	NPSC0	...
PA16	60	EMDIOL	IRQ6	EMDIOL	....

Table 3: IRQ assignement

1.4 Bank Registers

- CS5 = 0x60000000: Front-end - 0x3082
- CS6 = 0x70000000: Time Tagging - 0x3082
- CS7 = 0x80000000: Slow Control - 0x3083

2 Operating System

The OS used in the LSC is Linux Debian 2.6.27 with I-Pipe and Xenomai 2.4.10. Boot is possible from the network OR the USB key (the default).

2.1 Drivers

2.1.1 SPI

The master SPI has 4 channels:

- Channel 0: the Serial Eprom housing u-Boot
- Channel 1: the DAC (DAC7554) via device /dev/spidev0.1
- Channel 2: the ADC (AD7490) via device /dev/spidev0.2
- Channel 3: unused (was connected to Canbus interface mcp2515, no longer used)

2.1.2 Canbus

The driver is the "lincan-0.3.4" driver. Not a socket driver, interfaces to the sja1000 chip implemented in the Cyclone III FPGA.

2.2 Interrupts handling

3 specific interrupts are not handled by standard Xenomai or Linux drivers (see table 3).

2.2.1 1PPS IRQ

The 1PPS Interrupts are handled by a Xenomai RealTime task, not a RTDM driver.

2.2.2 /EVTCLKF Interrupts

The /EVTCLKS (Muon Event) interrupts are handled by a Xenomai RealTime task, not a RTDM driver.

2.2.3 /EVTCLKS Interrupts

The /EVTCLKF (Fast Event) interrupts are handled by a Xenomai RealTime task, not a RTDM driver.

3 Sofware, Compilation and installation

The specific LSC software is available from the SVN repositiory at MTU, in the directory **an_lsc_sw**. The main development line is in **trunk**, tagged versions are in **tags**. The hierarchy is divided into several sub directories:

- Services: all the software related to general services (message server, gps receiver control, etc).
  - Services/src: all the sources
  - Services/include: include files
  - Services/config: configuration files
  - Services/bin_lsc: executable and tar balls
  - Services/lib_lsc: libraries
- Acq: All the software related to the acquisition/triggering
- Manager: general utilities to help editing, downloading, etc (used on the host, Linux or Mac)
- Documents
  - Documents/UserManual
  - Documents/Devices: description of the devices (manufacturer data)
  - Documents/Schematics: LSC schematics
  - Documents/ElectronicBox: detailed description of the electronic box
- Crosstools
  - Crosstools/crosstools-0.43.tar.gz: the tar ball used to download, compile and implement cross tools for ARM (gcc, ld, etc)
- TarBalls: several tar balls containing include files and libraries needed to compile the software (linux and xenomai stuff)

3.1 Installing cross tools

The file **trunk/CrossTools/crosstools-0.43.tar.gz** contains all the files needed to compile and install the ARM cross tools, compiler, loader, etc. The cross tools version to use is *gcc-3.4.5-glibc-2.3.6* ;to install the crosstools:

1. Copy the tar ball to a local directory
2. Untar the tar ball in the local directory
   (a) creates a directory **crosstools-0.43**
3. `cd crosstools-0.43`
4. Edit the file: **demo-arm.sh**
   - Uncomment (suppress the sign #) the line containing '`#eval ‘cat arm.dat gcc-3.4.5-glibc-2.3.6.dat‘ sh all.sh --notest --nounpack`'
   - Comment (add the sign #) the line containing '`eval ‘cat arm.dat gcc-4.1.0-glibc-2.3.2-tls.dat‘ sh all.sh --notest`'
5. Edit the file: **arm.dat**
   - replace the line '**TARGET=arm-unknown-linux-gnu**' with '**TARGET=arm-linux**'
6. Execute the script **demo-arm.sh**. The script will download all the necessary files, compile the cross tools and install them in **/opt/crosstool/gcc-3.4.5-glibc-2.3.6/arm-linux/bin/**. This operation takes a rather long time, be patient.
   - The tools are named like: *arm-linux-gcc*, *arm-linux-ld*, etc

NOTE For installation on Mac OS X, have a look at the file **CrossTools/note-for-mac.txt**

3.2 Installing linux and xenomai stuff

Many include files and libraries are necessary to compile correctly the LSC software. The **trunk/TarBalls** directory contains all the tar balls that have to be 'untarred'. The script **trunk/lsc_install** is provided to do that and must be executed in the **trunk** directory (the tar balls are untarred directly in **trunk**; as a consequence the following directories are created in **trunk** (but re not managed by svn):

- lincan-0.3.4: CanBus stuff
- linux-2.6.27: Linux OS stuff
- xenomai-2.4 and xenomai: Xenomai stuff

3.3 Preparing for compilation

For legacy reasons, the LSC software is not implemented under an IDE (like eclipse). All Makefiles are based upon several environment variables that MUST be defined properly. A template script **trunk/set_lsc_sw.tpl** is provided. It MUST be adapted to local specifities and mst be run prior to any attempt to compile the LSC software.

3.4 Compiling

The executables compiled are automatically created in the directory **trunk/Services/bin_lsc** or **trunk/Acq/bin_lsc**. In addition, in the same directories, 2 scripts **dot**ar and **dotartest** are used to create the tar balls l**sc.tar.bz2** and **lsctest.tar.bz2** (for Services) or **acq.tar.bz2** and **acqtest.tar.bz2** (for Acq). These scripts are launched automatically when compiling the whole Services or Acq packages (see below).

3.4.1 Single process

Just move to the relevant directory (in **trunk/Services/src** or **trunk/Acq/src**), edit the source(s) files and just type `make`.

3.4.2 The whole Services or Acq package

In **trunk/Services/src** or **trunk/Acq/src**, a general **Makefile** can (re)make all the Services or Acq software. Just type `make all` to clean, compile, link, create the lsc tar balls and run doxygen; `make full` creates a new version number and then acts like `make all`.

3.4.3 The whole LSC software

A script `mklsc`, in **trunk/Manager/bin**, chains `make all` or `make full` for Services and Acq, `mklsc all` to make all, `mklsc full` to make full.

3.5 Downloading to the LSC

In the lab, without any radio, the best solution is to use the ethernet connection to download the software. All the executables, packages or specific test modules, MUST be downloaded to **/root/LSC/bin**, using **scp** or **rsync**, and of course the tar balls (if downloaded) should be untarred at the same place (**tar xjf <tarball>**).

4 Software description

The LSC software is available at the SVN repositiory at **svn://<username>@server1.auger.mtu.edu/an_lsc_sw** .

The LSC software is organised in 2 packages (see figure 4), the **Services** package (see section 5) and the **Acquisition** package (see section 6.1).

The services handle the communications with CDAS (via the radio), the control of the GPS receiver as well as the control of the 1PPS interrupts and time keeping, the upload and download of files and the execution of shell remote commands from CDAS. The acquisition can be started/stopped independently of the services. The core of the services is the Message Server (msgsvr); all communication to and from CDAS go through the Message Server. The communication between any task and the Message Server are handled with Linux Message Queues. Any task willing to receive or send messages to or from CDAS must be registered by the Message Server.

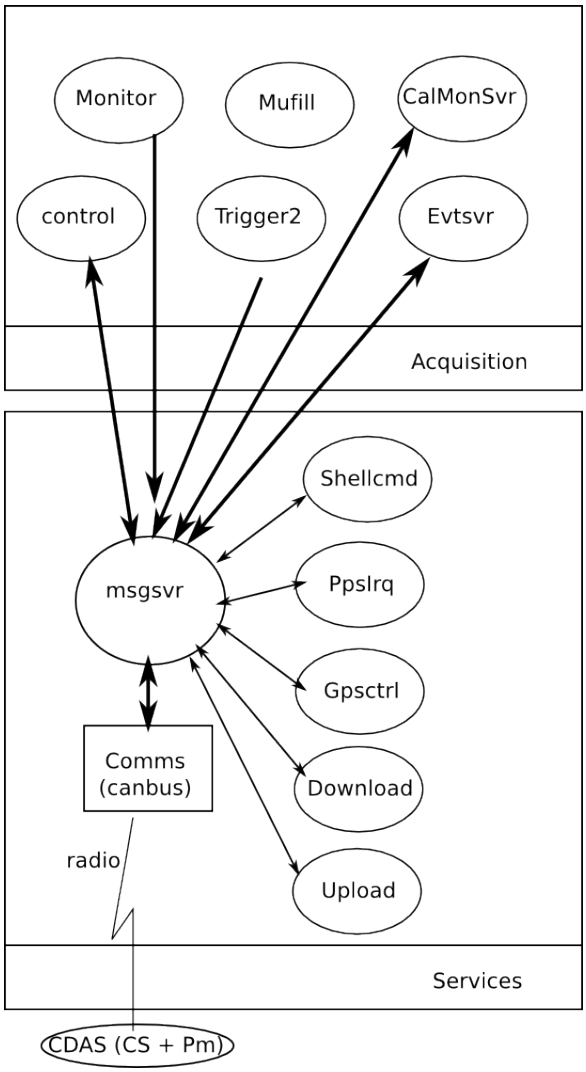

Figure 4: Global Organisation of the LS Controller Software.

4.1 Communications LSC to CDAS

Messages between LS and CDAS are transfered in units called '*frames*' (do not confuse with CanBus Frames). A *frame* from LS to CDAS is made of at most 288 bytes (2304 bits). A *frame* from CDAS to LS may contain up to 2048 bytes (16 Kbits) TBConfirmed. A frame may hold several *messages*. *Messages* that are too large to fit in one *frame* are sent by "*slices*". The principle is to fill the frame as much as possible.

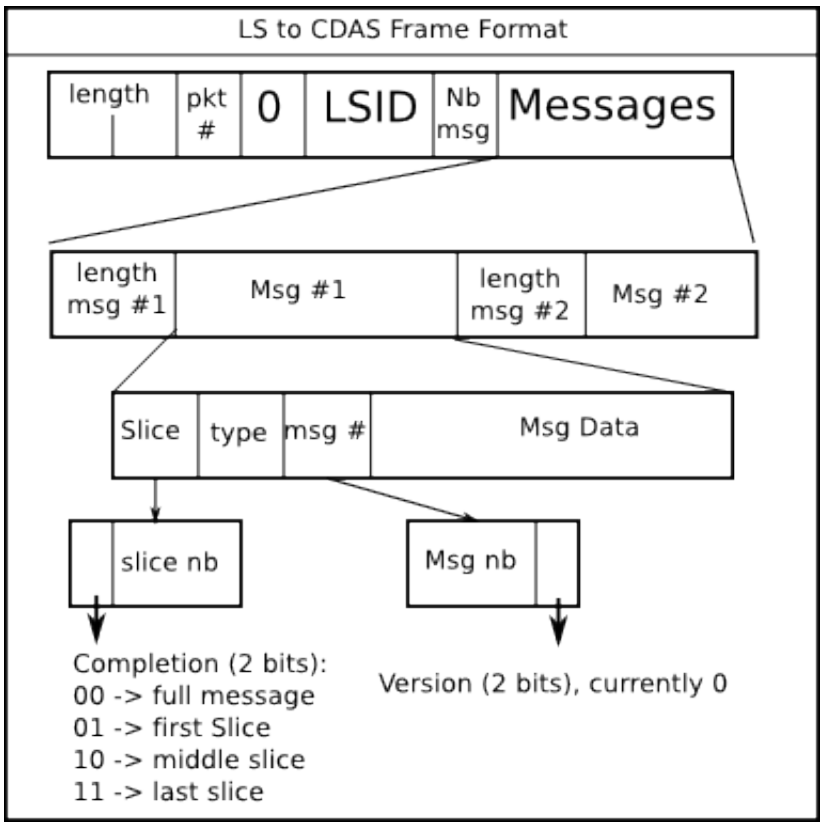

Figure 5: LS to CDAS Frame format. The length of the frame does NOT include itself. The length of a message DOES include itself.

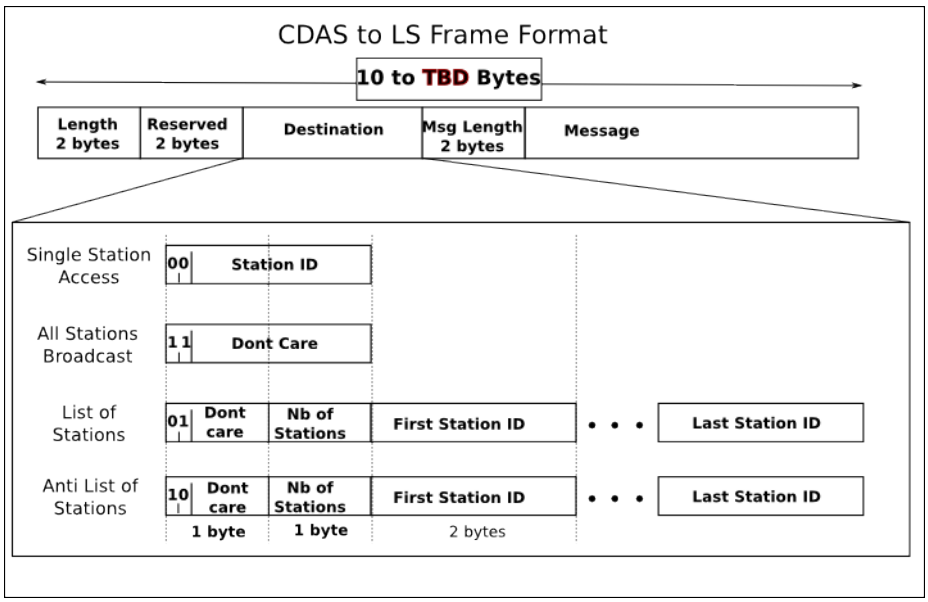

Figure 6: CDAS to LS Frame format. From CDAS, only one message at a time is sent, thus no number of messages.

4.1.1 LS to CDAS Frame Format

An *LS to CDAS frame* may hold several *messages*. It is made of a header followed by as many *messages* as needed. There is no trailer. The maximum size of a frame is 288 (TBC) bytes. See Figure 5.

1. Frame length: one short integer (16 bits), the total length of the frame, **NOT including itself**.
2. Frame NB: one byte, incremented at each frame.
3. Reserved: one byte unused, set to 0.

4. LSID: one short integer, the LS ID.
5. Nb Msg: one byte, the number of messages contained in the frame.

4.1.2 Message Format

A *message* is made of a header followed by as many bytes of data as needed (and taking into account the maximum size of the frame). The message IDs are described in Appendix C.1.1 and C.1.2.

- Message Length: 1 short integer (16 bits), the total nb of bytes of the message (**including itself**)
- Completion/Slice: 1 byte for completion of message and slice number
  - Completion: 2 most significant bits (00: full message, 01: first slice, 10: middle slice, 11: last slice of the message).
  - Slice number: 6 less significant bits, incremented at each slice
- Message Type: 1 byte. See Appendix C.1.1 for the list of defined message IDs.
- Message Number/Version: 6 most significant bits are the message number, 2 less significant are a message version. The message number is the same for all the slices of a given message.

4.1.3 CDAS to LS Frame Format

A *frame* hold ony one *message*. It is made of a header followed by the *message*. There is no trailer. See Figure 6. The format of the message itself is the same as the LS to CDAS message format.

- Frame Length: one short integer (16 bits), the total length of the frame **NOT including itself**,
- Reserved: 2 bytes unused. should be set to 0.
- Destination: the destination LsId or a list of LsId, as a set of short integers. The 2 most significant bits of the Destination describes the type of destination.
  - 00: Single LsId. The 14 other bits hold the LsId.
  - 11: Broadcast (all LSs). The 14 other bits are irrelevant.
  - 01: List of LsId. The 14 other bits hold the number of following LsId, each coded on a short int.
  - 11: AntiList od LsId. Same as above: all the LS except those in the antilist.
- The format of the message is the same as the LS to CDAS messages.

NOTE See Appendix C.1.2 for the list of defined message IDs.

4.2 Communications LSC to Local Radio

4.2.1 CanBus

4.3 Startup operations

4.3.1 PowerOn/Reboot

On PowerOn or Reboot, after having setup the Linux and Xenomai stuff, the following operations are performed

1. Load the FPGA firmware (including Canbus transceiver sja1000, the time tagging firmware and the trigger firmware).
2. Load the canbus driver lincan.
3. Set Interrupts levels and priorities for 1PPS, /EVTCLKF and /EVTCLKS.
4. Initialize the ramdisk /ram0 (8 MBytes). Note that by default, services and acquisition use the ramdisk to read configuration files and write log files.
5. Copy the configuration files from /root/LSC/config to /ram0.
6. Launch the services.
   (a) The msgsvr send the frame “Power Status” to the radio.
7. Once the GPS receiver is OK (TRAIM solution good), the message server

(a) Send to radio
- 1PPS status frame (OK)
- Date/time
- Position

(b) Launches the acquisition itself.
Note that the acquisition is started only when the GPS is OK.
Note that at this point the triggering is **NOT started**, but monitoring data are sent (by default every 5 minutes).

This is the end of the startup operations, the LSC is ready to handle triggers, send T2, monitoring data, etc.

The reboot takes about 1.5 minutes. In principle one should not have to reboot frequently the LSC (as it is the case in AS). The acquisition can be stopped/started independently of the services (see section 6.2.1).

If for some reason, the services must be restarted (because a new version has been downloaded) no need to reboot the LSC; the utility program **srv** can be used for that (see section 5.2.1).

5 Services Software

5.1 Services Tasks

5.1.1 Message Server (msgsvr)

The Message Server is the heart of the communications between processes and Comms. Messages sent from a process to CDAS are passed to the msgsvr. The message server stores the messages, when awakened by the 1PPS signal (every second) concatenates the stored messages into a frame and sends the frame to the radio. Messages have a priority, HIGH, MEDIUM, LOW. Concatenation into frames is made in the order of the priorities.

In the same way, frames from CDAS are received by the msgsvr. The frame is split into messages, and the messages are passed to the recipient process. In order to receive messages from CDAS, a process must be registered to the msgsvr, indicating which ID(s) of messages it should receive (see section 5.4.4). Note that several processes can receive the same message IDs. Some message IDs are handled directly by the msgsvr:

- M_REBOOT: Reboot the LSC. The msgsvr performs the following actions
  - Send a POWER_STATUS frame to Local Radio, with data[1] = 0 (Failure Imminent. Some other code could/should be used for "Reboot Imminent" ?)
  - Stop the acquisition
  - Save the log files to /scr/log/yyyymmdd-hhmmss directory.
  - Reboot the LSC
- M_CONFIG_TO_FLASH: Save the configuration files (**SvrConfig.cfg, GpsConfig.cfg, AcqConfig.cfg**) to **/root/LSC/config**.

5.1.2 GPS Control (gpsctrl)

Gpsctrl handles all the operations of the GPS Receiver. At startup, the position (as set in the GpsConfig.cfg file) is loaded into the GPS Receiver, the hold position is set according to the configuration file (note that Hold mode should be the default for the M12M Timing Receiver. This means also that the position of each station should be known with a reasonnable accuracy). Gpsctrl reads the messages sent by the GPS Receiver; when the TRAIM solution becomes good, ppsirq is launched and the msgsvr is signaled. If/when the TRAIM Solution becomes BAD, the msgsvr is also signaled (and in turn informs the radio).

5.1.3 1PPS Interrupts (ppsirq)

Ppsirq is in charge of the reception of the 1PPS Interrupts, keeps the time and signals the processes that need that information. Processes willing to receive 1PPS signals should register to **ppsirq** (see section 5.4.5); **ppsirq** is launched by **gpsctrl** when the GPS Receiver TRAIM Status becomes OK.

5.1.4 Remote Shell Command Execution (shellcmd)

Shellcmd executes the command passed in the message and returns the exit code of the command.

5.1.5 Upload

Handles file transfer from LSC to CDAS

5.1.6 Download

Handle file transfer from CDAS to LSC.

5.1.7 Tank Power Control (tpcb)

Tpcb receives Canbus frames issued by the TPCB and stores the information into a shared memory. The TPCB sends its data every minute.

5.2 Services Utilities

5.2.1 Srv

srv is used to (re)start, stop services and acquisition. It can also give the status of the services. The services are launched at boot time in the **/etc/rc.local** script.

Usage `srv [<options>] <action>`

Options

- `-v` : Verbose mode.
- `--boot`: Used to indicate that it is started just after a poweron/reset.
- `--pld`: Reload the FPGA firmware and the canbus driver before (re)Start.
- `--dir=<path>`: Use <path> as the working directory (default is the ramdisk **/ram0**).
- `--bin=<path>`: Use <path> as the executable directory (default '**/root/LSC/bin**').
- `--test`: Use '**/root/LSC_TEST/bin**' as executable directory.
- `--version`: Prints the current version.

Actions

- `start`: Launches all the services tasks.
- `restart`: Stop then start all the services and acquisition tasks. Usefull from CDAS (via a remote shell command).
- `stop`: kill gracefully the services and acquisition tasks.
  - NOTE: if the services are stopped, they cannot be re-started from CDAS ! From CDAS avoid **stop**, use **restart**.
- `kill`: kill "brute force" all the services and acquisition tasks.
  - NOTE: as with the `stop` action, a `kill` cannot be recovered from CDAS.
- `halt`: kill services, acquisition tasks and halt the system before a power-off.
  - NOTE: this action is **mandatory** before powering off the system (that's Linux).
  - NOTE: as with the `stop` action, a `halt` cannot be recovered from CDAS.
- `rmq`: removes all the message queues (in case '`srv stop`' failed to do the job).
- `status`: check that all service tasks are running. The return code contains the number of tasks that should be running but are not (ie are dead for some reason). Normal status is 0.

Gps issues If, for some reason, the GPS receiver does not get ready, the ppsirq process does not start. It is possible to force the GPS to look ready by sending the command:

```
stop -4001 gpsctrl
```

5.2.2 BuildConfig

Buildconfig creates or modify the 2 files **SvrConfig.cfg** and **GpsConfig.**cfg. The structures are described in Appendix C.3 and C.2.

Usage buildconfig [<opt> [<opt>] ...]

Options

- `--lsid=<id>` : Set LSID
- `--loc=<where>` : Set the location. Possible locations are read from the configuration file **locations.cfg** (see Appendix E.1).
- `--lat=<mas>` : Latitude in MAS
- `--lon=<mas>` : Longitude in MAS
- `--dlat=<deg>` : Latitude in Decimal Degrees
- `--dlon=<deg>` : Longitude in Decimal Degrees
- `--north=<cm>` : Northing (in cm)
- `--east=<cm>` : Easting (in cm)
- `--zone=<str>` : UTM Zone (3 characters)
- `--alt=<mas>` : Altitude in centimetres
- `--hold=<0|1>` : Hold Position (0 = False, 1 = True)
- `--off=<nanos>` : Gps Offset in nanos (default 0)
- `--version` : Print the version.
- `-v, --verbose` : Verbose
- `-?, --help` : What you see now

5.2.3 Saveconfig

Saveconfig is a shell script that copies the configuration files AcqConfig.dat, GpsConfig.dat and SvrConfig.dat from the ram disk /ram0 into /home/user/LSC/config.

Saveconfig is invoked either directly or by sending the signal SIG_SAVE_CONFIG to msgsvt or by sending the message type M_CONFIG_TO_FLASH.

5.2.4 Svrstatus

Status and statistics of the message server (msgsvr).

5.2.5 Gpsstatus

Usage gpsstatus [<options>]

Options

- `-g : Prints the GPS time and GPS to UTC offset.`
- `-s` : Small Status.
- `-f` : Full Status.
- `-F` : Super Full Status (including the sawtooth).
- -p <sec> : Periodic repeat (in seconds). Default is single shot.

5.2.6 Tpcbstatus

Usage tpcbstatus [<opt> [<opt>] ...]

Options

- `--wait` : Wait till valid data are available
- `--loop` : Infinite loop
- `--save=<path>` : Save data to <path>
- `--ascii` : Save in ASCII (default is BINARY)
- `--version` : Print the version.
- `-v, --verbose` : Verbose
- `-?, --help` : What you see now

5.2.7 Stop

Stop sends a "signal" to some processes. Note that these are not actually linux signals (in the usual linux meaning) but are specific messages, sent to the message queue of the process, that simulate the OS9 signals. A signal is just a 32 bits integer, the action taken by the process is process dependent; however some signal values are of general interest (see Appendix C.4for the list and meaning of signals).

```
stop -<signal> <process> ...  <process>
```

Example `stop -4001 gpsctrl` : forces gpsctrl to look ready even if no GPS antenna or no satellites are visible/tracked.

Example `stop -12345 control` : starts triggering.

5.2.8 GenMsg

Send a message or a - small - file as a generic message (ie an ASCII string) to CDAS. The file should be small, although there is no hard coded limit to the file size. Genmsg can be issued by CDAS as a remote shell command.

```
genmsg --file=<path>
genmsg --msg=<string>
```

Example1 (shell command from CDAS): `ipcs >ipcs.txt ; genmsg --file=ipcs.txt`

Example2: `genmsg --msg="`srv status -v`"`

Example3: `genmsg --msg="`tpcbstatus -v`"`

Example4: `genmsg --msg="(grep W t1read.log | wc)"` <— Count the number of warnings in a log file. Note the parenthesis.

5.2.9 sigauger

Lists the signals that are known and handled by different tasks (see Appendix C.4for the list and meaning of signals).

5.3 Test Utilities

A global test named lstest can be used to check that there is no problem with the LSC. lstest is a shell script, using the various test programs described below.

5.3.1 ADC (ttadc)

Usage `ttadc [options]`

Options

- `-n, --loop=<n>` : Reads <n> times and makes averages
- `-o, --out=<path>` : Output File.
- `--tmout=<usec>` : Set conversion timeout at <usec> micros
- `--reset=<y/n>` : Set/Unset reset ADC at each read

- `-v, --verbose` : Verbose.
- `-t` : Print the time.
- `-?, --help` : What you see.

5.3.2 DAC (ttdac)

Usage ttdac [options]

Options

- `-c, --chan=<n>` : Dac channel (0 to 3). By default set all.
- `-V, --volt=<vv>` : Voltage (0. to 2.5).
- `-d, --ddu=<dd>` : Dac Units (0 to 4095).
- `-z, --zero` : Set 0 volts when finished.
- `-v --verbose` : Verbose.
- `-?, --help` : What you see.

5.3.3 Time Tagging (tt_ttag)

Usage tt_ttag [<options>]

Options

- `-n --count` : Number of seconds (default infinite)
- `-v, --verbose` : Verbose. Print data.
- `-?, --help` : What you see now.

5.3.4 1PPS (ttpps)

Usage ttpps [<options>]

Options

- `-z, --reset` : Reset the TTAG before any other operation.
- `-o, --out=<path>` : Output written to <path>. Default stdout
- `-n, --nirq=<nn>` : Number or 1PPS IRQ. Default infinite, stop with CTRL-C
- `-v, --verbose` : Verbose

5.3.5 Tpcb (tt_tpcb)

5.3.6 GPS Receiver (gpstest)

5.3.7 Front End

5.4 Libraries

5.4.1 ShmLib

- int ShmBind(char * name, int size, int * shmid)
- void * ShmAttach(char * name, int size, int * shmid)
- int ShmDetach(void * mem)
- void ShmRemove(void * mem, int shmid)

5.4.2 SemLib

```
int SemLink( char * name, unsigned int * key ) ;
int SemAttach( char * name, unsigned int * key ) ;
int SemDetach( int the_id ) ;
int SemP( int the_id ) ;
int SemV( int the_id ) ;
int SemSet( int the_id, unsigned int value ) ;
int SemStatus( int the_id, struct semid_ds * status ) ;
int SemValue( int the_id, int * value ) ;
```

5.4.3 MsgQueueLib

- unsigned int IpcMsgCreate(const char * name, unsigned int * key)
- unsigned int IpcMsgBind(const char * name, unsigned int * key)
- int IpcMsgRemove(int msgid)
- unsigned int IpcMsgCheck(int msqid)

5.4.4 MsgSvrClientLib

- unsigned int MsgSvrClientOpen(char * qname, int nb_id, unsigned char * msg_id)
  - Arguments
    - qname: the name of the queue of the client, actually the process name. Messages sent by the msgsvr are sent to this queue.
    - nb_id: number of CDAS message IDs expected by this client.
    - msg_id: array of CDAS message IDs. Upon reception, by the msgsvr, of messages from CDAS, only the messages with the relevant IDs are passed to the client.
  - Return
    - the ID of the msgsvr queue, or -1 in case of error.
- int MsgSvrClientClose(char * qname, unsigned int mqueueid)
  - Arguments
    - qname: the name of the queue of the client.
    - mqueueid: the queue id of the msgsvr (given by MsgSvrClientOpen)
  - Return
    - 0 if OK, an error code otherwise.
- int MsgSvrClientSend(int queuid, int priority, void * buf, int size):
  - Usage: Send a message to the msgsvr for the CDAS.
  - Arguments:
    - queuid: the queue id of the msgsvr (given by MsgSvrClientOpen)
    - priority: one of HIGH_PRIORITY, MEDIUM_PRIORITY, LOW_PRIORITY.
    - buf: pointer to the message. The message must have the following format:
      - Type: one 32 bits integer with the CDAS message type.
      - The content of the message. The format of this part is message dependent.
    - size: overall size of the message.
  - Return
    - 0 if OK, -1 otherwise and errno contains the error number.
- int MsgSvrClienSignal(char * task_name, int signal) ;
  - Usage: Send a 'signal' to a given Auger task.
  - Arguments
    - task_name: the name of the task.
    - signal: the signal number

The list of recognized signals is available at Appendix **C.4**.

5.4.5 PpsClientLib

Any process wishing to reveive the 1PPS information should register to the ppsirq process. Upon detection of a 1PPS by ppsirq, ppsirq sends a "signal" to the client's queue. The signal number is PPS_SIGNAL_READY (see sigauger.h).

- int PpsClientRegister(char * my_queue)
  - Arguments
    - my_queue: the name of the queue of the client, actually the process name. At each 1PPS signal, a message is sent by ppsirq to this queue.
  - Return
    - 0 if OK, -1 otherwise and errno contains the error number.

5.4.6 TtagLib

- int TtagOpen()
- int TtagClose()
- int TtagGetVersion()
- int TtagGetId()
- int TtagGetFast(TIME_STAMP * date) ;
  - Reads the /EVTCLKF timestamps (two timestamps, leading and trailing edges used to comote the deadtime).
- int TtagGetSlow(ONE_TIME * date) ;
  - Reads the /EVTCLKS timestamp. In this case only one timestamp.
- unsigned int TtagGetCalib()
- unsigned int TtagGet100MHz()
- unsigned int TtagGetSecond()
- void TtagReset() ;

5.4.7 CanLib

- int CanlibVersionNb()
- char * CanlibVersion()
- int CanlibSendDataMsg(int first_id, unsigned char * msg, int length)
- int CanlibSendRadioMsg(unsigned char * msg, int length)
- int CanlibGetDataMsg(CANBUS_FRAME * frame)
- int CanlibPassToCanDriver(CANBUS_FRAME * frame, int dbg_flag)
- int CanlibOpen()
- int CanlibClose()

5.4.8 LogfileLib

- void LogSetProgName(const char *name)
- void LogInitTime(unsigned int *temps)
- void LogSetFileName(const char *fname)
- void LogSetNewFile(const char *fname)
- int LogPrt(const char *str, int we, const char * fun)
- FILE *LogOpen(void)
- void LogClose(void)
- int LogDebug(const char * fun, int we, const char * fromat, ...)
- int LogPrint(int we, const char *format, ...)
- int LogPrintTimed(int we, const char *format, ...)
- int LogPrintDate(int we, const char *format, ...)
- int LogPrintSysDate(int we, const char *format, ...);

5.4.9 LinkLib

Generic loibrary to handle linked list of any kind of item.

- void * LinkCreate(int size)
- int LinkDelete(void *old)
- int LinkSwap(void *one, void *two)
- void * LinkNext(void *one)
- void * LinkPrev(void *one)
- void * LinkAddHead(void *new, void **head, void **tail)
- void * LinkAddTail(void *new, void **head, void **tail)
- void * LinkAfter(void *new, void *old)
- void * LinkBefore(void *new, void *old)
- void * LinkUnlink(void *old)
- void * LinkFind(void *first, void *last, int (*func)()) ;

5.4.10 SlowLib

5.4.11 GpsUtilLib

- unsigned char * short_to_bytes(unsigned char *pb, unsigned short val) ;
- short bytes_to_short(unsigned char * pb) ;
- unsigned char * int_to_bytes(unsigned char *pb, unsigned int val) ;
- unsigned char * int_to_3bytes(unsigned char *pb, unsigned int val) ;
- unsigned char * int_to_2bytes(unsigned char *pb, unsigned int val) ;
- int bytes_to_int(unsigned char *pb) ;
- int bytes3_to_int(unsigned char * pb) ;
- long dswab(char *from) ;
- double mas_to_deg(int mas) ;

  - Convert MAS (MillArcSecond) to decimal degres.

- int deg_to_mas(double deg) ;
  - Convert decimal degres to MAS (MilliArcSecond)
- unsigned int gps_seconds(int yy, int mm, int dd, int hh, int mn, int ss) ;
  - Computes GPS seconds from GPS receiver date.
- char * Gps2Utc(unsigned int gps, unsigned int offset) ;
  - Returns the UTC time (computed from the GPS time and GPS/UTC offset) as a string like 'YYYY/MM/DD hh:mm:ss'
- unsigned int Gps2UtcTime(unsigned int gps, unsigned int offset) ;
  - Returns the UTC time in second (linux standard)
- char * Gps2Fname(unsigned int gps, unsigned int offset) ;
  - Returns a string 'YYYYMMDD_hhmmss' representing UTC time (usable as a file name)
- char * UtcDateStr(time_t * utc) ;
  - Return a string like 'YYYY/MM/DD hh:mm:ss'

6 Acquisition Software

6.1 Acquisition tasks

6.1.1 Control

Contrary to Auger South software, control has a minor role. It is only used to

- Initialize all shared memory modules,
- Check that all the acquisition processes are still alive.
- Starts/Stop the triggering. To be compatible with Auger South, start/stop triggering is made with:
  - Start Triggering: `stop -12345 control`
  - Stop Triggering: `stop -12346 control`

Control propagates the triggering state to the other tasks involved.
Starting and stopping the whole acquisition is now devoted to the utility "**das**" (see section6.2.1)

6.1.2 T1irq

T1irq is a Xenomai RT task that handles the /EVTCLKF Interrupts. Upon reception of an interrupt **t1irq** increments the RT Semaphore 'FastIrqSem' (rt_sem_v).

6.1.3 T1read

T1read is a Xenomai RT tasks. It waits on the RT semaphore 'FastIrqSem' (rt_sem_p), reads the event and the timestamp into the FastBuffer and increments the Linux semaphore 'FastReadySem' (SemV).

6.1.4 T1fake

T1fake generates fake T1's (/EVTCLKF). Used when there is no Front End board. A software Trigger is generated at ~ 100 Hz and catched by **t1irq** just like any normal T1.

6.1.5 Muirq

Muirq plays the same role as t1irq for the muon (/EVTCLKS) interrupts. Muirq is a Xenomai RT task that handles the /EVTCLKS Interrupts. Upon reception of an interrupt **Muirq** signals the RT Semaphore 'SlowIrqSem' (rt_sem_v).

6.1.6 Muread

Muread is a Xenomai RT tasks. It waits on the RT semaphore 'SlowIrqSem' (rt_sem_p), reads the event and the timestamp into the SlowBuffer and increments the Linux semaphore 'SlowReadySem' (SemV).

6.1.7 Mufill

Mufill waits on the Linux semaphore 'SlowReadySem' (SemP), moves the event from the SlowBuffer into the MuonBuffer. Mufill performs the calibration processing (NOT IMPLEMENTED YET)

6.1.8 Mufake

Generates fake Muon triggers (/EVTCLKS)by mean of a 'Slow Soft Trigger' in the FPGA Firmware. This feature is implemented in the Courty firmware version; it would be nice to have in the final trigger implementation. The /EVTCLKS Interrupt is catched by Muirq just like a normal /EVTCLKS.

6.1.9 Grbread

Every second, reads the GRB Counters and store into the AcqStatus shared memeory.

6.1.10 Trigger2

Trigger2 waits on the Linux semaphore 'FastReadySem' (SemP), moves the event from the FastBuffer into the EventBuffer, checks if it is a T2 (**T2 algo are NOT IMPLEMENTED YET**), and builds the M_T2_YES message. Trigger2 is a client of **ppsirq (**see section **5.1.3)** and every second sends the M_T2_YES message to CDAS.

6.1.11 EvtSvr

6.1.12 Monitor

Monitor reads the Slow control ADC values every N seconds (defined in AcqConfig.dat), computes averages every M minutes (defined in AcqConfig.dat), and signals **calmonsvr** that it is time to send the monitoring block to CDAS. The TPCB data are also handled in the same way, except that the read period is 1 minute (hard coded in the TPCB firmware). In addition it stores minimum and maximum values of each channel; this is specially useful for the currents as it permits to check that there is no unexpected current peaks. See table 1 for the assignement of the ADC channels and Appendix D.3 for the description of the Monitoring Block.

6.1.13 Calmonsvr

Calmonsvr sends the latest monitoring block when signaled by **monitor** or by a request from CDAS (message ID; M_MONIT_REQ)

NOTE ***The format of the monitoring block is not the same as in Auger South***

6.2 Acquisition Utilities

6.2.1 Das

The Das utility is used to start, stop or get the status of the acquisition processes.

Usage `das [<options>] <action>`

Options

- `-v` : Verbose mode.
- `--dir=<path>`: Use <path> as working directory (where the logfiles are written)/
- `--bin=<path>`: Use <path> as the executable directory (default '/**root**/**LSC**/**bin**').
- `--test`: Use '/**root**/**LSC_TEST**/**bin**' as executable directory.
- `--version`: Prints the current version/

Actions

- `start`: Launches all the acquisition tasks.
- `restart`: Stop then start all the acquisition tasks. Usefull from CDAS (via a remote shell command)
- `stop`: kill gracefully all the acquisition tasks and make sure that message queues are all removed.
- `kill`: kill "brute force" all the acquisition tasks.
- `rmq`: removes all the message queues (in case '`das stop`' failed to do the job).
- `status`: check that all das tasks are running. The return code contains the number of tasks that should be running but are not (ie are dead for some reason).

6.2.2 Gocalib

Calibrate the PMT: search the value for the High Voltage to have a T1 rate at 100 Hz. The T1 rate is caculated by **getrate**. (see section 6.2.3)

Note that gocalib does not write the threshold to the acquisition configuration file. And make sure that, if the muon trigger is enabled, the muon threshold is not too low, otherwise the system may be overflooded with muons IRQs.

Options: see table 4

Option	Long option	Comment
	`--vmin=<volts>`	Set minimum voltage at <volts> (default 500V)
	`--vfirst=<volts>`	Set first search volts at <volts> (default 1000V)
	`--vlast=<volts>`	Set last search voltage at <volts> (default 1500V)
`-f`	`--fast`	Fast calib. getrate counts T1 on 4 seconds (default 8)
`-V`	`--vem=<counts>`	Use <counts> as VEM (default 50)
`-w`	`--write`	Write the voltage, if foind, to the AcqConfig.cfg
`-W`	`--Write`	Write the voltage, even if NOT found.
	`--test`	Test mode. Use t1fake to generate Soft Triggers (default TotA)
	`--version`	Print the version number.
`-?`	`--help`	What you see.
`-v`	`--verbose`	Increase verbosity

Table 4: Gocalib Options.

6.2.3 Getrate

Getrate counts the number of T1 events during a certain amount of time, and returns it as the exit code. As **getrate** uses the /EVCLKF interrupts, the normal acquisition MUST be stopped. Getrate is used by **gocalib**.

Implemented, ok with gocalib in TEST mode.

Options: see table 5.

Option	Long Option	Comment
`-t`	`--time=<sec>`	Time in seconds (default 5)
	`--test`	Test mode, use Soft triggers (default TotA)
	`--version`	Print the version
`-v`	`--verbose`	Increase verbosity
`-?`	`--help`	What you see

Table 5: Getrate Options

6.2.4 Showersim

Generate at a given time a LED Flasher Event.

Options: see table 6.

Option	Long option	Comment
	`--gps-<gps_time>`	Set the date in GPS time.
	`--utc=<utc_time>`	Set the date in UTC time (seconds since 1970).
	`--date=<utc_date>`	Set the date as "YYYY/MM/DD hh:mm:ss" UTC.
	`--vext-<volts>`	Set the Vext voltage (default 1V).
	`--a-base=<volts>`	Set the LedA Base voltage (default 1V).
	`--a-dac=<volts>`	Set the LedA Dac voltage (default 2 V).
	`--b-base=<volts>`	Set the LedB Base voltage (default 1 V).
	`--b-dac=<volts>`	Set the LedB Dac voltage (default 2 V).
	`--vmax`	Set the maximum possible voltage (about 11 Volts)
	`--version`	Print the version number.
`-v`	`--verbose`	Increase verbosity.
`-?`	`--help`	What you see.

Table 6: Showersim Options

6.2.5 Golin

Compute linearity with the LED Flasher. Golin first reads the configuration file 'golin.cfg'. the configuration file is an ascii file with one line (lines beginning with '#' are ignnored as comments) containing in this order:

- AreaBegin: the beginning of the sample index where the peak is searched. This value can be actually searched if the option '–auto' is set.
- BaseOffset: the relative index from where the base line is calculated. The actual sample index used is (AreaBegin-BaseOffset).

- SignalArea: the number of samples used to search the peak. The peak is searched from AreaBegin to (AreaBegin + SignalArea). Similarly, the base line is computed from (AreaBegin - BaseOffset) to (AreaBegin - BaseOffset + SignalArea)
- NbSteps: Number of steps with increasing Led voltages to calculate the linearity.
- NbShots: Number of events used at each stp.

Golin scans the Leds to find the minimum Base values (for each LED) to find a peak well above the base line and to find the maximum Vext value to get the saturation. Then the Led Dac voltages are increased at each step and the requested number of events are generated and read out. Once done, if everything went well, the linearity data ascii files **migoled.pk** and **migoled.ch** (for peak and charge respectively) are generated. The format of these file is the same, one line per step:

- LED A mean (peak or charge)
- LED A sigma(peak or charge)
- LED A+B mean (peak or charge)
- LED A+B sigma(peak or charge)
- LED B mean (peak or charge)
- LED B sigma (peak or charge)

Options:

- `-c` : Dont Create Linearity files (default YES to files '**migoled.pk**' and '**migoled.ch**')
- `-b` : Create Ascii files for all shots with peak, charge, ... to file '**golin.bin**'
- `-s` : Set Saturation voltages, does not check saturation, just set VEXT at maximum (2.5V). Default is to search the saturation on Vext for each Led and use the maximum of the 2 values found.
- `--auto` : Search the position of the peak automatically by setting VEXT and DACs to maximum (saturation). Overrids the peak position found in the configuration file.
- `--vext-ab` : Search Vext maximum using Leds A+B. Default is A and B separately, the max of the 2 is taken.
- `--chan=<ch>` : Search linearity of PMT channel *<ch>*. *<ch>* must be one of **a1** (Anode), **a0.1** (Anode x0.1), **a30** (Anodex30), **d5** (Dynode). Default is **a1** (Anode).
- `--save` : Save signals to a binary file '**signals**'
  - The number of events saved per step is calculated as: `nb = MAX_SIGNALS_SAVE / (NbSteps * NB_LED)` where MAX_SIGNALS_SAVE = 200 and NB_LED = 3 (LED_A, LED_B, LED_A+B)
  - The traces format is:
    - For each trace 1 short with a tag: 0x8000 | (led & 0x3) | (step<<2)
    - For each sample 4 shorts: Anode, Anode*0.1, Anode*30, Dynode
- `--freq=<hz>` : LEDs are fired at *<hz>* frequency. Default is 100 Hz (the maximum).
- `--mkcfg` : Create default configuration file '**golin.cfg**'
- `--version` : Print the version.
- `-v, --verbose` : Increase Verbosity
- `-?` : Short Help.
- `--help` : Long Help.

6.2.6 Acqconfig

Create/Modify the acquisition configuration file (on the ramdisk) **AcqConfig.cfg**. At boot time, the default AcqConfig.dat file is copied from /home/user/LSC/config to the ram disk /ram0.

- Options:
- `-z, --default : Create a default config file`
- `-c, --config=<path> : Use <path> as default config file`
- `-o, --output=<path> : Use <path> as the output config file`
- `-m, --mod-file=<path> : <path> contains modifications to be applied`
- `-l, --short-list : Print modified items`
- `-L, --long-list : Print the actual content of the config file`
- `-h, --show-items : Print all items that can be modified`
- `--version : Print the version.`
- `-v, --verbose : Verbose`
- `-?, --help : What you see.`

Below is the list of items that can be modified with acqconfig. To modify a parameter, just type acqconfig followed by <param_name>=<value>. For example:
`acqconfig HVPmt=1200` sets the default High Voltage value to 1200 Volts.
`acqconfig v1=1200` is exactly the same thing (using the short name)

Param Name	short name	Current value (% means decimal 0x means hexa)	Comment
Version	(v)	= 0x4d908120 [%1301315872]	Config version Nb (UTC seconds)
Vem	(vm)	= %50 [0x32]	VEM in ADU
T2Algo	(a2)	= %3 [0x3] (Random T2)	T2 Algorithm (1 to 4)
MaxT1Rate	(mx1)	= %200 [0xc8]	Maximum authorized T1 Rate in Hz (default 200
MaxT2Rate	(mx2)	= %40 [0x28]	Maximum T2 Rate in Hz (default 40)
MaxT1Miss	(ms1)	= %200 [0xc8]	Maximum T1 lost
MonRate	(mr)	= %5 [0x5]	Monitoring read out rate en seconds.
MonAverage	(ma)	= %300 [0x12c]	Monitoring averaging and sending to cdas in s
MonSend	(m)	= %300 [0x12c]	?????
HVPmt	(v1)	= %500 [0x1f4]	PMT High Voltage
TotA_Thresh	(tat)	= 0x300 [%768]	Time over Threshold A threshold
TotA_Width	(taw)	= 0x80 [%128]	... width
TotA_Occup	(tao)	= 0x1 [%1]	... occupancy
TotA_Enable	(tae)	= %0 [0x0]	... enable (1 enabled, 0 disabled)
TotB_Thresh	(tbt)	= 0x28a [%650]	Same for ToT–B
TotB_Width	(tbw)	= 0x80 [%128]	
TotB_Occup	(tbo)	= 0x20 [%32]	
TotB_Enable	(tbe)	= %0 [0x0]	
TotD_Lower	(tdl)	= 0x200 [%512]	Time over Threshold Deconvoluted
TotD_Upper	(tdu)	= 0x264 [%612]	
TotD_Width	(tdw)	= 0x80 [%128]	
TotD_Occup	(tdo)	= 0x20 [%32]	
TotD_FD	(tdfd)	= 0x0 [%0]	
TotD_FN	(tdfn)	= 0x0 [%0]	
TotD_Enable	(tde)	= %0 [0x0]	
Muon_Enable	(mue)	= %1 [0x1]	Muon trigger enable
Soft_Delay	(sfd)	= %100 [0x64]	Software Trigger Delay
Soft_enable	(sfe)	= %0 [0x0]	... Enable (1 enabled, 0 disabled)
Ext_enable	(exe)	= %0 [0x0]	External trigger enable
Grb_enable	(grb)	= %1 [0x1]	Gamma Ray Burst enable

6.2.7 Acqstatus

Dump the status of the aqcusition.
Options

- `-t`: the exit vode contains the number of T1. Useful when executed from CDAS.
- `-T`: the exit code contains the number of T2. Useful when executed from CDAS.
- `-m`: The exit code contains the number of Muons. Useful when executed from CDAS.
- `-v`: Verbose mode

If no option is given, the exit code is made of 3 bits:

- D0: Acquisition started if 0, 1 otherwise
- D1: Triggering Started if 0, 1 otherwise
- D2: Front End present if 0, 1 otherwise

In normal operation - when triggering is started - in a radio only mast or tower, the exit code should be 4. In full tanks it should be 0.
The following information are displayed with '`acqstatus -v`'

```
Aqcuisition Status at 985000718 - 2011/03/24 11:18:38
  NO Front End Board, Fake Triggers
  T2 Algo: Random [3]
  GPS Start Time: 984994924 - 2011/03/24 09:42:04
Triggers Running
  GPS Start Time: 984994937 - 2011/03/24 09:42:17
   Duration 5781 seconds
  TotA: 0, TotB: 0, TotD: 0, Sft: 0, Ext: 0, Miss: 0, Unk: 564082
  Total T1: 564082 - T1 Rate: 97.6
  T2: 112600 - T2 Rate: 19.5
  DeadTime: 2.128149 (0.037 %), 3.773 us/evt
  Last T1 at 985000718.016422416
  Muon: 0, Miss: 0
  GRB Counter: 717
```

6.2.8 Monitstatus

6.2.9 Minispy

Usage `minispy [<options>]`

Options

- `--period=<sec>` : Period in seconds (default 5)
- `-s, --single` : Single shot (default is infinite loop)
- `--version` : Print the version.
- `-v, --verbose` : Verbose
- `-?, --help` : What you see now

6.3 Acquisition Libraries

6.3.1 Buflib: Generic Circular Buffer Management

6.3.2 Fblib: Fast (T1) Buffer Management

6.3.3 Mblib: Slow (muons) Buffer Management

6.3.4 Felib: Front End

6.3.5 Ledlib: Led Flasher

7 Installing a new LSC

1. Preparation in the lab

   (a) Connect to the LS with the console cable.
   (b) Login as root.
   (c) Stop the services and acquisition: `srv -v stop`
   (d) Set ethernet interface "local": `set_local_ip ;`
      - The local IP address is set as **192.168.1.<nn>** where <nn> is the CPU Number (from 97 to 124).
   (e) `cd /root/LSC/config`
   (f) `buildconfig --loc=<location-name>` (see section 5.2.2and E.1)
   (g) `halt`

   One can now switch off the LSC

2. In the field

   (a) Connect to the LSC with the console cable.
   (b) Power on the LSC
   (c) Login as root
   (d) `cd /ram0`
   (e) `gpstime yyyy/mm/dd.hh:mm:ss` where *yyyy/...* is the current (reasonnably accurate within 3 minutes) UTC time, to help the GPS receiver to get a good TRAIM solution.
   (f) After a few seconds, the GPS receiver should be OK. Check with
      - `gpsstatus -f`
      - `srv -v`
      - `das -v`

3. Back to the lab.

   - Connect to the LSC with the console cable and connect the ethernet cable.
   - open a Kermit connection
   - Power On the LSC
   - Login as root
   - If available, set ethernet interface to DHCP: `set_dhcp_ip`

   One can now ssh to the LSC.

8 Tips and Howtos

8.1 Changing an LSC cpu number

If necessary, it is possible to change the cpu number of an LSC.

- connect to the LSC via the serial line (console)
- enter the command: `setcpunb <nnn>` where <nnn> is the desired cpu number.
- reboot the LSC: `reboot`

8.2 Testing a new LSC Board

A test script **lstest** is provided to make a first global test of the devices of the board. **lstest** generates log files in the directory */scr/LsTests/<nn>/<yymmdd_hhmmss>* where *<nn>* is the LSC number, and *<yymmdd_hhmmss>* is the date and time; the summary file is named lstest_<nn>.log (see section F for an example of the lstest log file). In order to run **lstest** one should

1. Stop the standard LSC software: `srv stop`
2. Set the UTC date (approximately): `date --set='YYYY/MM/DD hh:mm:ss'`
3. run lstest: `lstest`

lstest checks the following devices

1. The CANBUS, using the TPCB
2. The Time Taging firmware
3. The GPS receiver (only read and write access to the receiver)
4. The generation of 1PPS Interrupts
5. The FPGA Firmware and /EVTCLKF interrupts generation.
6. The slowcontrol ADCs (read and check that values read are reasonnable)
7. The slowcontrol DAC (just set the 4 DAC channels at 1.25 Volts. Note that the DACs are set to 0 at the end of this test)
   - In order to verify that the DACs output are correct, one should, once **lstest** is finished, use the command **ttdac**.
8. Test the Humidity and Temperature sensor SHT11
9. If the option '-l' is used (**lstest -l**), the commuication with the Led Flasher Controller is also tested.

At the end of each test, the result is printed to the console and written to the log file.

8.3 Replacing a USB key in a LSC

1. Connect to the LSC with the serial line and login
2. Halt the LSC (if still running)
   - `halt`
     - wait till the message "system halted" appears
3. Power off
4. Replace the key
5. Power on and login
6. Update the configuration of the LSC:
   (a) Set the proper cpu number

- `setcpunb <nnn>` where <nnn> is the number **on the sticker of the LSC** (not the one on the key !)
  Example: `setcpunb 124`

(b) Select the static local IP address. The static address is made from the cpu number: 192.168.1.<nnn> (otherwise DHCP is used).

- `set_local_ip`

(c) Set the proper position of the tank/mast/tower:

- `buildconfig --loc=<location>`
  <location> is the tank/mast/tower name as shown in the file 'locations.cfg' (see below the file).
  Example: `buildconfig --loc=a-tank` (case dont care)

(d) Reboot the LSC:

- `reboot`

The LSC is ready to run, Ethernet access is available at IP address 192.168.1.<nnn>.,the GPS position is now OK.

8.4 Testing a new Trigger Firmware version

8.4.1 Global test

The initialisation of the FPGA is made by the shell script `/root/LSC/bin/lsinit`. The script loads the FPGA with the file `/root/LSC/config/pld.rbf` which is a link to the current rbf file `cur_pld.rbf`.

To test a new version of the pld code:

1. Download the rbf file to test (let's say `new_pld.rbf'`) into `/root/LSC/config`
2. Create a link to this file
   (a) `cd /root/LSC/config`
   (b) `ln -fs new_pld.rbf pld.rbf`
3. Restart the LSC software with option '`--pld`'
   (a) `srv --pld restart`
   will stop all LSC specific software, reinitializes the FPGA and restart the services and acquisition processes.

When finished, if the new version does not work, relink the `cur_pld.rbf` file to `pld.rbf` and restart the software with `srv --pld restart`.

If the new version is OK, rename or copy it as `cur_pld.rbf` , restart the software with `srv --pld restart` and that's it.

8.4.2 Specific test

1. Download the new rbf file to **/root/LSC/config**
   - `scp blabla.rbf root@192.168.2.112:LSC/config`
2. Kill the standard LSC software
   - `srv -v stop`
3. Load the new code into the FPGA
   - `fpga /root/LSC/config/blabla.rbf`

- Re-initialize CanBus driver
  – /root/modprobe125

At this point, one can use any test program. 2 test programs are available:

- `ttfe`: does not use the FE library. Probably better for first tests, to prepare the FE Library.
- `ttfenitz`: uses the FE Library.

Sources are in `Acq/src/Tests/FeNitz` and `Acq/src/Tests/TtFe` respectively. Once compiled the executable code are in **Acq/bin_lsc.**

Download the code with:

- `cd $Acq_Bin`
- `scp ttfe root@<host>:LSC/bin` where <host> is the host name or IP address of the LSC (192.168.2.112 at MTU)

Execute the code with

- `ssh root@<host>`
- `cd /ram0`
  ttfe (as well as most test programs) creates a logfile **'ttfe.log**'. In order to avoid to pollute the /root directory, it is better to set the ram disk as the working directory.
- `ttfe <options>`

8.5 Running without GPS antenna

It is sometimes necessary to test the LSC software without a GPS antenna (or the position is such that the receiver cannot see or track satellites). There is a possibility to "fool" the GPS receiver with a signal telling "dont mind the satellites, everything is OK". Just log to the LSC and issue the command:

`stop -4001 gpsctrl`

In this case, the 1PPS is generated internally by the GPS receiver itself.

In order to have a reasonnable GPS time, it is highly recommended to run first the command:

`gpstime 'YYYY/MM/DD hh:mm:ss'` to set the GPS receiver time. The date here is the **UTC** date/time.

8.6 Setting the GPS Receiver at PowerOn

Upon PowerOn, in order to help the GPS receiver to acquire satellites and become ready, it is good practice to set the GPS receiver time with an accuracy better than one minute. For this purpose use the command:

`gpstime 'YYYY/MM/DD hh:mm:ss'` where 'YYYY/MM/DD ...' is the **UTC** time.

8.7 How to generate long data streams

It is possible - to test of the radio interface - to fill almost completely the data streams from LS to CDAS with T2 messages. It's a bit tricky but works. To do that one should modify 2 parameters in the acquisition configuration file

- The T2 algorithm: up to now, 4 T2 algorithms are defined:
  - T2_STANDARD_ALGO: compute qualiy of T1, not yet implemented.
  - T2_LED_ALGO: keep only LED triggers, not yet implemented.
  - T2_RANDOM_ALGO: generate 20% randomly (used for fakes T1).
  - T2_ALL_ALGO: keep all T1's as T2 up to MAX_T2_RATE
- MAX_T2_RATE: the maximum number of timestamps in one T2 message. By default it is set at 40.

By setting the T2 algo at T2_ALL_ALGO (4), at a T1 rate of 100 Hz, 100 T2 are generated. On should then adjust the MAX_T2_RATE in order to fill the 288 bytes of a pkt sent to CDAS; in one pkt, with only one message, there is an offset of 12 bytes (frame header + message header), plus the GPS second (4 bytes); that leaves 272 bytes for the time stamps.

Thus the maximum of timestamps is 272/3 = 90. The good value however is 89, because in addition to the time stamps, on additional 24 bytes value is added as the Gamma Ray Burst counter (fake for the moment, but ...).

Thus, to set the proper values:

1. Log into the LSC (root, root)
2. `cd /ram0`
3. Change the acquisition configuration file (**a2** means T2 algo, **mx2** means MAX_T2_RATE, see section 6.2.6):
   - `acqconfig a2=4 mx2=89`
4. Restart acquisition: `das restart`

5. Restart triggering: `stop -12345 control`

Note In this case, there is no more room available to send any other message. Monitoring messages (sent every 5 minutes) shall never arrive to CDAS, as T2 have the highest priority.

Note After a reboot or power on, the acquisition configuration is taken from /root/LSC/config. Thus it is necessary to repeat the procedure. Going back to the normal behaviour, use

`acqconfig a2=3 mx2=40` (random trigger, maximum t2 rate 40)

A Appendix A - Svn Repository Content

B Appendix B - Description of the messages

B.1 M_Ready Message

The M_READY message is sent by the LSC when it is ready to send data (T2, Monitoring blocks, etc) to CDAS (when the GPS becomes OK). It is also sent upon request from CDAS (message M_WAKEUP sent by CDAS).

NOTE The format of the M_READY message is NOT the same as in Auger South.

1. Run status: 1 byte.
   - Reflects the status of the acquisition. 3 bits (see section 6.2.7)
2. Poweron/reset: 1 byte
   - 4 bits
     - D0; 1 if poweron, 0 if reset (currently always 0, maybe later ...)
     - D1: 1 if first startup, 0 if restart
     - D2: 1 if sent because Wireless was bad and is now ok, 0 for another reason.
     - D3: 0 if 1PPS OK, 1 otherwise.
3. GPS/UTC Offset: 1 short int
4. Software version: 1 unsigned int
   - The general format is (hexa): YYMMDDxx, where YY=year, MM=month, DD=day, xx=00 if test, 1 if "almost final", 2 if "final"
     For example: 0x11083000
5. Config version: 1 unsigned int
6. UTM Northing: 1 signed int
7. UTM Easting: 1 signed int
8. Altitude: 1 positive int
9. Current GPS time: 1 int

B.2 T2 message

The timestamps of the Level 2 Triggers are sent every second. The format is:

1. GPS second: 1 int
2. As many as needed time stamps on 3 bytes:
   (a) 20 LSB: the time in micros relative to the GPS second.
   (b) 4 MSB: the trigger type (ToT-A, ToT-B, ToT-D, Soft trigger, External Trigger)[1].

B.3 Monitoring Message

The monitoring message is sent every 5 minutes to CDAS. The format is fully described in Section D.3. It contains only 32 bits integers: averages, minimum and maximum values are multiplied by 1000 to avoid use of floats.

NOTE The format of the Monitoring Block message is NOT the same as in Auger South.

- GPS time
- GPS/UTC offset
- Count: nb of reads to make average

[1] A long time ago, these bits were supposed to be an indication of the energy of the event. This is no longer the case, even in Auger Siuth.

- ADC Average Array [13 items] (see table 1for description of the ADC channels)
- ADC Min Values Array [13 items]
- ADC Max Values Array [13 items]
- High Voltage Dac value
- LSC PCB Temperature Average (SHT11 device)
- LSC PCB Humidity Average (SHT11 device)
- LSC PCB Min Temperature (SHT11 device)
- LSC PCB Max Temperature (SHT11 device)
- LSC PCB Min Humidity (SHT11 device)
- LSC PCB Max Humidity (SHT11 device)
- TPCB Panel Current
- TPCB Panel Voltage
- TPCB Load Current
- TPCB Load Voltage
- TPCB Athmospheric Pressure
- TPCB Temperature

Note For the TPCB, only average values are available.

C Appendix B - Services data structures and include files

C.1 Message IDs

C.1.1 Messages from LSC to CDAS (in central_local.h)

```
typedef enum {          /**< Messages from Local to Central station */
  M_READY,              /**< from Control to CS      */
  M_RUN_START_ACK,      /**<      Control    CS - Obsolete     */
  M_RUN_PAUSE_ACK,      /**<      Control    CS - Obsolete     */
  M_RUN_CONTINUE_ACK,   /**<      Control    CS - Obsolete     */
  M_RUN_STOP_ACK,       /**<      Control    CS - Obsolete     */
  M_T2_YES,             /**< 5     Trigger2   CS      */
  M_T3_EVT,             /**<      EvtSvr     CS      */
  M_T3_MUONS,           /**<      EvtSvr     CS      */
  M_CONFIG_SET_ACK,     /**<      Control    CS      */
  M_MONIT_REQ_ACK,      /**<      CalMon     CS      */
  M_MONIT_SEND,         /**< 10     CalMon     CS      */
  M_CALIB_REQ_ACK,      /**<      CalMon     CS      */
  M_CALIB_SEND,         /**<      CalMon     CS      */
  M_BAD_SEQUENCE,       /**<      Control    CS      */
  M_BAD_VERSION,        /**<      Control    CS      */
  M_MSG_ERROR,          /**<  15    MsgSvrIn   CS      */
  M_DOWNLOAD_ACK,       /**<      Control to CS      */
  M_SHELL_CMD_ACK,      /**<      Control to CS      */
  M_MODULE2FLASH_ACK,   /**<      Control to CS - Obsolete      */
  M_LOG_SEND,           /**<      Control to CS      */
  M_UPLOAD_SEND = M_LOG_SEND,
  M_SET_PARAM_ACK,      /**<  20    Control to CS - Obsolete      */
  M_UNKNOWN,            /**<      Control    CS      */
  M_GENERIC_STR = M_UNKNOWN,      /**<        all   to CS
                                  Generic Warning/Error/Whatever ASCII string */
  M_LAST_CENTRAL        /**<  Must be the LAST */
} MsgTypeOut;
```

C.1.2 Messages from CDAS to LSC (in central_local.h)

```
typedef enum {          /* Messages from Central to Local station */
  M_REBOOT,             /* from CS      to Msgsvr */
  M_WAKEUP,             /*      CS         Msgsvr */
  M_RUN_ENABLE,         /*      CS         Control - Obsolete */
  M_RUN_START_REQ,      /*      CS         Control- Obsolete */
  M_RUN_PAUSE_REQ,      /*      CS         Control- Obsolete */
  M_RUN_CONTINUE_REQ,   /*      CS         Control- Obsolete - 5 */
  M_RUN_STOP_REQ,       /*      CS         Control- Obsolete */
  M_T3_YES,             /*      CS         EvtSvr  */
  M_CONFIG_SET,         /*      CS         Control */
  M_FLASH_TO_CONFIG,    /*      CS         Control */
  M_CONFIG_TO_FLASH,    /*      CS         Control - 10 */
  M_MONIT_REQ,          /*      CS         CalMonSvr  */
  M_CALIB_REQ,          /*      CS         CalMonSvr  */
  M_DOWNLOAD,           /*      CS         Download - 0xd (13)*/
  M_DOWNLOAD_CHECK,     /*      CS         Download - 0xe*/
  M_SHELL_CMD,          /*      CS         Shellcmd (was "OS9_CMD"- 15 */
  M_MODULE2FLASH,       /*      CS         ControlObsolete */
  M_LOG_REQ,            /*      CS         Upload */
  M_UPLOAD_REQ = M_LOG_REQ,
  M_SET_PARAM,          /*      CS         Control - Obsolete */
  M_GPS,                /*      CS         GPS - Obsolete      */
  M_MSG_ERROR_IN,       /*   MsgSvrIn      Control - 20 */
  M_NO_INPUT,           /* No msg from CDAS expected */
  M_LAST_LOCAL          /***  Must be the LAST *****/
```

```
} MsgTypeIn;
```

C.2 GpsConfig

C.2.1 gpscommon (gpscommon.h)

```
#define SERIAL_NUMBER_LENGTH 18

enum {
  TRAIM_SOLUTION_OK,
  TRAIM_SOLUTION_ALARM,
  TRAIM_SOLUTION_UNKNOWN
} ;

typedef struct {
  int lat_mas ;
  int lon_mas ;
  int height ;
} POSITION_MAS ;

typedef struct {
  int northing ;
  int easting ;
  int height ;
  char zone[4] ;
} POSITION_UTM ;

/* Gps geography */
typedef struct {
  int Latitude ;                /**< Coordinates (in MAS) */
  int Longitude ;
  int Altitude ;

  int Northing ;
  int Easting ;
  char Zone[4] ;
} GPS_GEOGRAPHY ;


typedef struct {
  int month, day, year ;
  int hour, minute, second ;
  int gmt_offset ;
  int hour_offset ;             /**< Offset (signed) to GMT */
  int minute_offset ;           /**<    ..................    */
} GPS_TIME ;

/* Tout ! */

typedef struct {
  int sat_id ;                  /**< Satellite ID */
  int doppler ;                 /**< Doppler in Hz */
  int elevation ;               /**< Elevation in degrees */
  int azimuth ;                 /**< Azimuth in degrees */
  int health ;                  /**< Health - 0 = healthy, 1 = unhealthy */
} SATELLITE_STATUS ;

typedef struct {
  int satellite ;
  int frac_time ;
} SATELLITE_FRACTIONAL ;
```

```
typedef struct {
  int svid ;
  int mode ;
  int strength ;
  int iode ;                    /**< ????? */
  int status ;
} SATELLITE_DATA ;

typedef struct {
  int pulse ;
  int pps_sync ;                /**< 0 = UTC, 1 = GPS time */
  int traim_solution ;
  int traim_status ;
  unsigned int rem_svid ;
  unsigned short accuracy ;
  char sawtooth ;
  SATELLITE_FRACTIONAL sat_frac[12] ;
} GPS_TRAIM_STATUS ;
```

C.2.2 gpsconfig (gpsconfig.h)

```
typedef struct {
  GPS_GEOGRAPHY Geography ; /**< Position */
  int HoldPositionMode ;    /**< 0 if floating position mode
                                 1 if HoldPosition mode
                                 2 if survey mode
                             */
  int Time ;                /**< Default initialisation time */

  unsigned int Htype ;    /**< GPS ellips. reference */

  unsigned int Mask ;       /**< Mask angle degrees  0.. 89  */
  unsigned int AlarmLim ;   /**< Alarm limit in 100s of nsec */
  unsigned int CabDel   ;   /**< Antenna cable delay nsec    */
  int GPSoffset ;             /**< Offset (given by CDAS) */
  int reserved[16] ;        /**< Reserved for future use */
} GPS_CONFIG ;
```

C.3 svrconfig (svrconfig.h)

```
typedef struct {
  int initialized ;
  int ServicesVersion ;
  int CpuNumber ;               /**< Hardware CPU Number */
  int LsId ;                     /**< Software LS Number (defined by CDAS */
} SVR_CONFIG ;
```

C.4 List of signals

```
  666 - mufill        - SAVE_CALIB              - Save Calibration to files
  777 - mufill        - START_MUONRUN           - Start a Muon Run (save data to file)
 1000 - Msgsvr, Gpsctrl - READ_CONFIG             - Re-read Configuration file
 1000 - Download      - DOWNLOAD_RESET          - Reset Download status
 1001 - ppsirq        - PPS_SEM_ENABLE          - Enable RT semaphore for grbread
 1001 - Msgsvr        - SAVE_CONFIG             - Save Configuration files
 1100 - Msgsvr        - PPS_IS_GOOD             - PPS Good, tell to Radio
 1101 - Msgsvr        - PPS_IS_BAD              - PPS Bad,tell Radio
 1102 - Msgsvr        - GPS_HAS_DATE            - GPS got the date, tell Radio
 1103 - Msgsvr        - GPS_READY               - Ssend M_READY to CDAS
```

```
 2000 - Calmonsvr      - MONIT_SEND                 - Send Monitoring block
 2001 - Calmonsvr,trigger2 - STOP_SAVE                  - Stop saving data to file
 2002 - Calmonsvr,trigger2 - START_SAVE                 - Start saving data to file
 3000 - All            - RESET_VERBOSE              - Reset Verbosity Level
 3001 - All            - INCREMENT_VERBOSE          - Increment Verbosity Level
 3002 - All            - DECREMENT_VERBOSE          - Decrement Verbosity Level
 3010 - All            - MEMLIB_STATUS_LOG          - Log Memory Usage
 4000 - Gpsctrl        - GPSCTRL_GET_SATELLITES     - Get Nb of Satellites
 4001 - Gpsctrl        - GPSCTRL_FORCE_OK           - Force OK even if no Antenna
 4002 - Gpsctrl        - GPSCTRL_SEND_MREADY        - Send M_READY Msg to CDAS
 4003 - Gpsctrl        - GPSCTRL_STATUS_STR         - Send Gps Status Generic Msg
 4004 - Gpsctrl        - GPSCTRL_SEND_DATE          - Send Date/time to Radio
 4005 - Gpsctrl        - GPSCTRL_SET_DATE           - Set Sytem date/time
 5000 - t1read, grbread - SHOW_NBEVT                  - List current nb of T1/Grb
 5001 - t1fake         - FAKE_T1_MIN                - Minimum (1) Fake T1 / second
 5200 - t1fake         - FAKE_T1_MAX                - Maximum (200) Fake T1 / second
 6000 - t1fake         - FAKE_NO_RANDOM             - Fixed Period Fake Triggers
 6001 - t1fake         - FAKE_RANDOM                - Random Period Fake Triggers
10000 - All            - PPS_SIGNAL_READY           - 1PPS occurred (from ppsirq)
12345 - Control        - START_TRIGGER              - Start Triggers
12346 - Control        - STOP_TRIGGER               - Stop Triggers
```

D Appendix B - Acquisition data structures and include files

D.1 Time stamps (timestamp.h)

```
#if !defined(_TIMESTAMP_H_)
#define _TIMESTAMP_H_

typedef struct {
  unsigned int second ;
  unsigned int nano ;
} ONE_TIME ;

typedef struct {
  ONE_TIME first ;
  ONE_TIME secnd ;
} TIME_STAMP ;

#endif
```

D.2 Fast and Muon Events (events.h)

```
#if !defined(_EVENTS_H_)
#define _EVENTS_H_

/*********************
  Definitions for Fast (aka T1) events
*********************/
/*
  At 100 MHz, 1024 samples = 10.24 micros, should be enough
  With 6 FADC, 10 bits ==> 8 bytes (64 bits) per sample
   ==> 8 Kbytes per event
  100 events per second
   ==> 800 Kbytes per second
  10 seconds of data stored
   ==> 8 Mega Bytes

  As we have 128 Meg of Ram, we could easily use 16 Megs for the event buffer
   ==> store ~ 20 seconds of T1 data

*/
#include "timestamp.h"

#define FAST_BUFFER_NAME "FastBuffer"
#define EVENT_BUFFER_NAME "EvtBuffer"

#define FAST_SAMPLE_NUMBER 1024
#define FAST_EVTS_PER_SECOND 100
#define FAST_EVTS_MAX_TIME 20
#define EVENT_BUFFER_NB_EVENTS  (FAST_EVTS_PER_SECOND*FAST_EVTS_MAX_TIME)

#define FAST_UNKNOWN_TYPE 0xFFFF

typedef struct {
  unsigned int fadc123, fadc456;
} FAST_SAMPLE ;

typedef struct {
  TIME_STAMP date ;             /**< Starting + falling time */

  int micro_off ;               /**< Offset in micros */
  unsigned short T2T1_type ;    /**< TOT or Threshold or ... */
```

```
  unsigned short nsamples ;      /**< Nb of samples. Should be equal to
                                  FAST_SAMPLE_NUMBER */
  unsigned int resvrd[8] ;       /**< Reserved for future use */
  FAST_SAMPLE data[FAST_SAMPLE_NUMBER] ;
} FAST_EVENT ;

/*********************
  Definitions for Slow (aka Muon) events
*******************/
/*
  Maximum 0x2000 muons per "event":
   A "muon event" IRQ is generated when the muon buffer is full.
*/
#define MUON_SAMPLE_NUMBER 0x2000
typedef struct {
  unsigned char sum, pmt ;
} MUON_DATA ;

typedef struct {
  unsigned int time_tag ;
  MUON_DATA sample0, sample1, sample2, sampl3 ;
} MUON_SAMPLE ;

typedef struct {
  TIME_STAMP date ;
  int bufsize  ;
  MUON_SAMPLE data[MUON_SAMPLE_NUMBER] ;
} MUON_EVENT ;

#endif
```

D.3 Monitor block structure (monitor.h)

```
#if !defined(_MONITOR_H_)
#define _MONITOR_H_

/*****************************************

  $Author:: guglielmi          $
  $Date:: 2010-11-19 15:54:35 #$
  $Revision:: 247              $

*****************************************/

/* Monitor Block */
typedef struct {
  unsigned int time ;          /**< @brief GPS Time in second */
  int gps_utc_offset ;           /**< @brief Offset between GPS and UTC time */
  int count ;                  /**< @brief Nb of items used in averaging */
  int adc[MAX_NB_ADC] ;        /**< @brief Averages of ADC values */
  int adc_min[MAX_NB_ADC] ;
  int adc_max[MAX_NB_ADC] ;
  unsigned int dac ;              /**< @brief Dac value - No way to read back ! */

  int pcb_th,                    /**< @brief Temperature of the LSC PCB (SHT11 sensor) */
    pcb_rh ;                      /**< @brief Humidity ... */
  int pcb_th_min,
    pcb_th_max,
    pcb_rh_min,
    pcb_rh_max ;
```

```
  // TPCB Data
  int tpcb_panel_current,         /**< @brief panel current in mAmps */
    tpcb_panel_voltage,     /**< @brief panel voltage in mVolts */
    tpcb_load_current,          /**< @brief load current in mAmps */
    tpcb_load_voltage,          /**< @brief load voltage in mVolts */
    tpcb_pressure,                  /**< @brief athmospheric pressure in 10th hpa */
    tpcb_temperature ;          /**< @brief temperature in 10th degrees */

    // Radio monitoring data - Preliminary
    int lr_data[MAX_LR_MONIT_DATA] ;
} MONITOR_BLOCK ;

typedef struct {
  time_t last_update ;          /**< @brief Date of last update */
  int count ;                       /**< @brief Nb of monitoring measures averaged */
} mon_tpcb_t ;

/* Definitions for the monitor circular buffer (library BMLib) */
#define MONITOR_NB_OF_BLOCKS 10
#define MONITOR_BUFFER_NAME "monitbuf"
#define SCALE_FACTOR 1000.

#endif
```

D.4 T3 Request Messages (acq_msg.h)

```
#if !defined(_ACQ_MSG_H_)

#define _ACQ_MSG_H_

/**
 * @defgroup acq_msg  Specific Acquisition Message definitions
 * @ingroup acq_include
 *
 */

/**@{*/
/**
 * @file   acq_msg.h
 * @author Laurent Guglielmi <laurent.guglielmi@apc.univ-paris7.fr>
 * @date   Wed Feb  9 15:30:31 2011
 *
 * @brief  Specific Acquisition Message definitions
 *
 *
 */

/**
 * @struct T2_MESSAGE
 * @brief Message containing the T2 time stamps, sent every second.
 *
 * T2_MESSAGE is made of:
 *    - 32 bits: The GPS Second of the following T2 time stamps.
 *    - For each timestamp
 *        - 4 bits: the event type.
 *            - 0x7 is specific to GRB data (should be the latest item).
 *        - 20 bits: the time in <B>microseconds</B> relative to
 *          the GPS second.
 *
 */
typedef struct {
```

```
  int seconds ;
  unsigned char the_t2s[2] ;
} T2_MESSAGE ;

#define EVTID_MASK 0x3FFF
#define EVTID_AGAIN_SHIFT 14
#define EVTID_AGAIN_MASK 0x3


/**
 * @struct T3_YES_MESSAGE
 * @brief Message received from CDAS
 *
 * T3 Request Message from CDAS. The message is made of
 *    - evtid: Event ID
 *    - again: If the same event is requested again
 *    - time: time stamp in seconds and microseconds
 *    - microref: offset to the timestamp
 *    - delta: search between timestamp-delta and timestamp+delta
 *
 */
typedef struct {
  int evtid ;
  int again ;
  ONE_TIME time ;
  unsigned char microref ;
  unsigned char delta ;
} T3_YES_MESSAGE ;

/**
 * @struct T3_EVT_MESSAGE_HEADER
 * @brief T3 Message Header
 *
 * - evtid: Event ID (from CDAS T3 Request)
 * - comprssed: 1 if compressed, 0 othewise
 * - error: Error code
 */
typedef struct {
  unsigned short evtid ;
  char compressed ;
  char error ;
} T3_EVT_MESSAGE_HEADER ;

/**
 * @struct T3_EVT_MESSAGE
 * @brief T3 Message
 *
 * The data part of the message is made of
 *   - The Fast Event
 *   - CalibX data (Not Implemented Yet)
 *   - CalibH data (Not Implemented Yet)
 *   - Ttag data (see below t3_ttag.h file)
 *
 */
typedef struct {
  T3_EVT_MESSAGE_HEADER header ;
  unsigned char data[4] ;
} T3_EVT_MESSAGE ;

/**@}*/
```

```
#endif
```

D.5 Time Tagging data in T3 message (t3_ttag.h)

```
#if !defined(_T3_TTAG_H_)
#define _T3_TTAG_H_

/*****************************************

  $Author:: guglielmi            $
  $Date:: 2011-01-20 14:11:54 #$
  $Revision:: 536                $

*****************************************/
/**
 * @defgroup t3_ttag.h  Specific T3 Ttag data definitions
 * @ingroup acq_include
 *
 */

/**@{*/
/**
 * @file   t3_ttag.h
 * @author Laurent Guglielmi <laurent.guglielmi@apc.univ-paris7.fr>
 * @date   Wed Feb  9 15:30:31 2011
 *
 * @brief  Specific T3 Ttag data definitions
 *
 *
 */


typedef struct {
  unsigned int cur_100, /**< @brief calib 100 of scurrent econd */
    next_100,                    /**< @brief calib 100 of second+1 */
    pprev_saw,                   /**< @brief Sawtooth of second-2 */
    prev_saw,                    /**< @brief Sawtooth of second-1 */
    cur_saw,                     /**< @brief Sawtooth of current second */
    rcv_offset ;                 /**< Receiver offset (from gpsconfig) */
} TTAG_BLOCK ;

/**@}*/

#endif
```

E Configuration Files

E.1 Locations.cfg

Locations.cfg is an ASCII file containing for each local station its name and position, one line per station. It is used by **buildconfig** to set the **GpsConfig.cfg** file. The UTM coordinates are here for information and can be omitted; they are NOT used by buildconfig, but computed from Latitude/Longitude.

Format

1. Name.
2. Latitude in MAS (**M**illi**A**rc**S**econd).
3. Longitude in MAS.
4. Altitude in cm.
5. Northing [optional].
6. Easting [optional].
7. UTM Zone [optional].

Example

```
# Format
# Name latitude(MAS) longitude(MAS) altitude(cm) Northing Easting Zone
# UTM coordinates are for information only, not read by buildconfig,
#   but computed from lat/long by buildconfig.
#
# Note: case does NOT care
#

# Locations of the RDA Tanks and Radios
A-Tower 136521713 -369399101 119400 4199930 709995 13S
B-Tank 136533215 -369268456 117900 4200367 713176 13S
C-Tank 136587316 -369345447 116900 4201986 711253 13S
D-Tank 136641115 -369268717 117100 4203693 713083 13S
E-Mast 136690069 -369346234 115200 4205153 711152 13S
F-Mast 136737420 -369273047 115800 4206659 712900 13S
G-Mast 136789594 -369346264 114800 4208221 711072 13S
H-Tower 136845002 -369278400 113300 4209972 712683 13S
I-Mast 136902316 -369347624 114700 4211695 710949 13S
J-Mast 136947122 -369281639 112700 4213118 712522 13S
K-Tank 136632637 -369404480 116700 4203346 709776 13S
L-Tower 136681831 -369479021 119400 4204816 707918 13S
M-Mast 136631935 -369537078 120600 4203242 706540 13S
N-Tower 136693489 -369601377 118600 4205100 704923 13S
O-Mast 136636439 -369669267 119600 4203300 703310 13S
P-Tank 136585735 -369268450 116200 4201986 713134 13S
Q-Tank 136624151 -369308194 115700 4203145 712133 13S
R-Tank 136605989 -369313172 116000 4202582 712026 13S
S-Tank 136637721 -369345155 116000 4203540 711220 13S
T-Tank 136624258 -369308682 115700 4203148 712121 13S
U-Tank 136629933 -369329145 116500 4203310 711617 13S
#
# Various locations outside the RDA
#
lamar 137087290 -369426520 110600 4217347 708879 13S
# APC Laboratory ( Paris, Electronics Lab - 5th floor )
apc 175783723 8580286  5000 5408609 454747 31U
# Alternate APC location ( Paris, Francois Arado Center )
face 175779720 8575952 5000 5408493 454638 31U
```

```
# Mines lab location
CSM_lab 143104978 -378805496 175600 4400126 480814 13S
# Mines Auger Mounting Hall
CSM_hall 143086230 -378796215 182000
# MTU
mtu 69635225 -318763080 30000 5219757 382783 16T
```

F Lstest log file example

```
++++++ LsTest - Cpu Number: 98 - Fri Sep 16 08:49:24 UTC 2011

>>>> Test TPCB and CAN (may last 1 minute)
Start Fri Sep 16 08:49:24 2011
  ==== TPCB and CANBUS OK ====
>>>> Test TTAG (about 10 seconds)
TTAG ID: 54545646 - Version: F
  ==== TTAG OK ====
>>>> Test GPS Receiver (about 1 minute)
  ==== GPS Receiver OK ====
>>>> Test 1PPS Irq (about 10 seconds)
  ==== 1PPS IRQ OK ====
>>>> Test FE Trigger
Front End Board Absent
  **** NO Front End Board ****
Courty FPGA, use Soft Trigger
Ttag ID: 0x54545646
Nb Events: 100, Nb IRQ: 100
Cleanup
  ==== FE FPGA OK ====
>>>> Test ADC

Averages over 10 read
        FE-1 - 0.000 < 0.000 < 0.000
        FE-2 - 0.000 < 0.000 < 0.000
        FE-3 - 0.000 < 0.000 < 0.000
        FE-4 - 0.000 < 0.000 < 0.000
      V-BATT - 12.041 < 12.042 < 12.048
    V-FE-12V - 12.041 < 12.045 < 12.048
 V-Fpga-1.2V - 1.200 < 1.201 < 1.201
  V-VCC-3.3V - 3.269 < 3.270 < 3.271
 V-CORE-1.8V - 1.786 < 1.797 < 1.802
    I-FE-12V - 0.004 < 0.013 < 0.033
       I-VCC - 0.098 < 0.329 < 0.584
      I-BATT - 0.207 < 0.318 < 0.478
        V-5V - 4.987 < 4.987 < 4.988
  ==== ADC OK ====
>>> Test DACs
  ==== DAC OK ====
>>> Test SHT11  (about 10 seconds)

Averages over 10 measurements
Temperature - 34.8 < 34.9 < 34.9
Hygrometry  - 23.9 < 23.9 < 23.9
  ==== SHT11 OK ====


---------- EVERYTHING OK ----------
```

G LSC Schematics

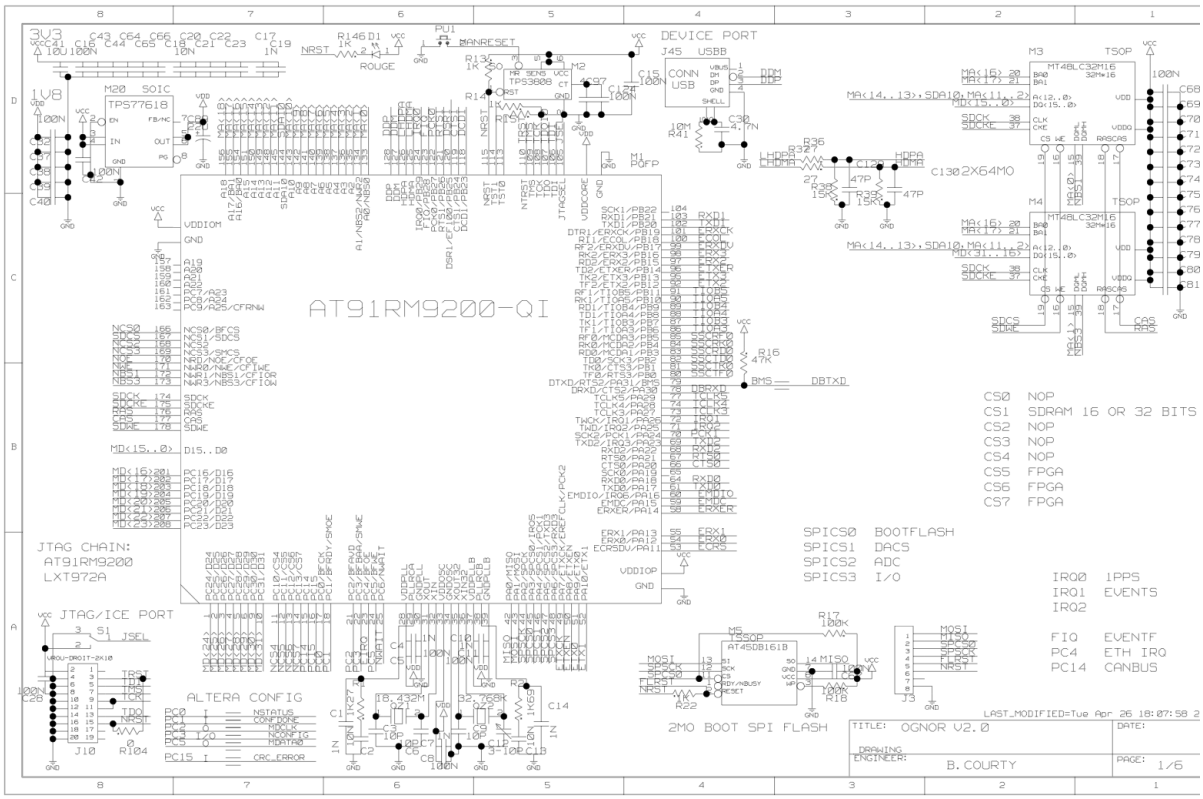

Figure 7: LSC Atmel ARM9 CPU

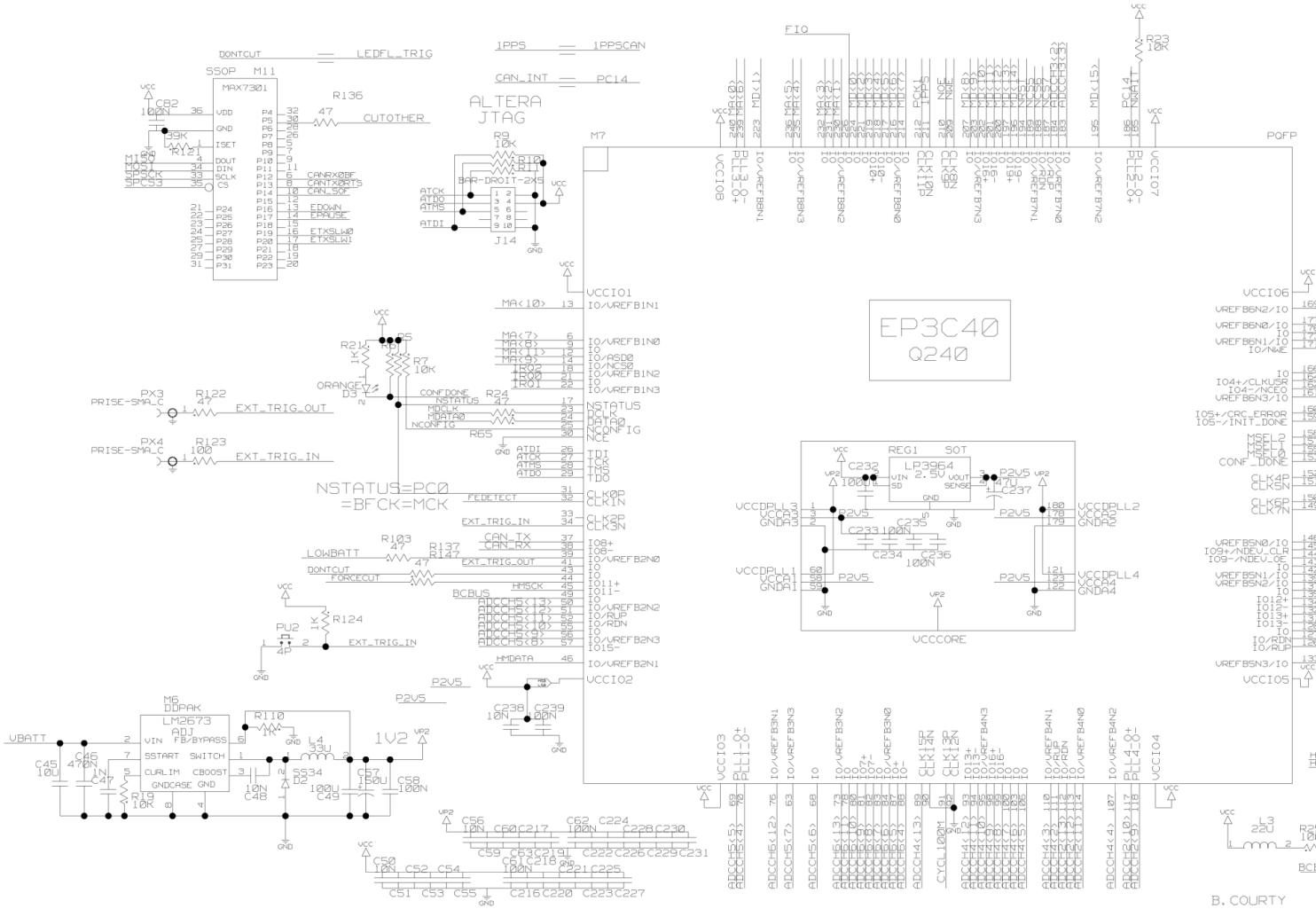

Figure 8: LSC: Cyclone

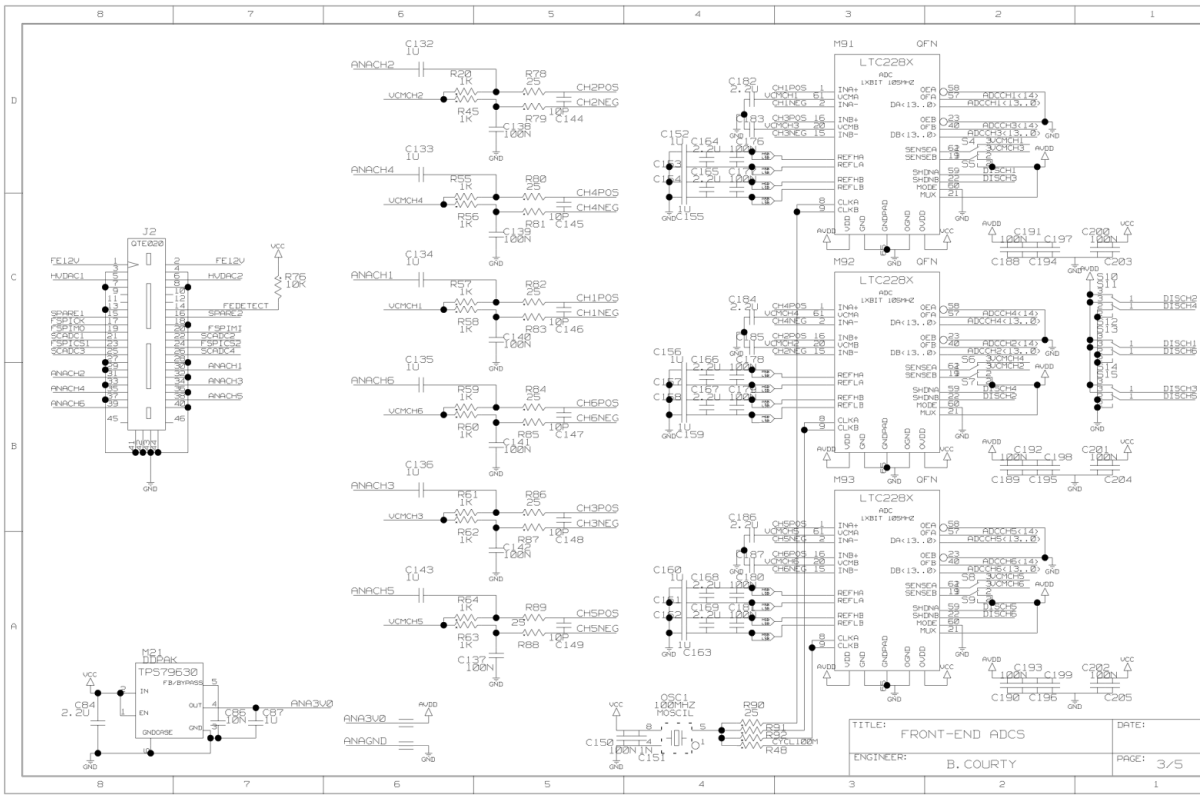

Figure 9: LSC: FrontEnd and Flash ADCs

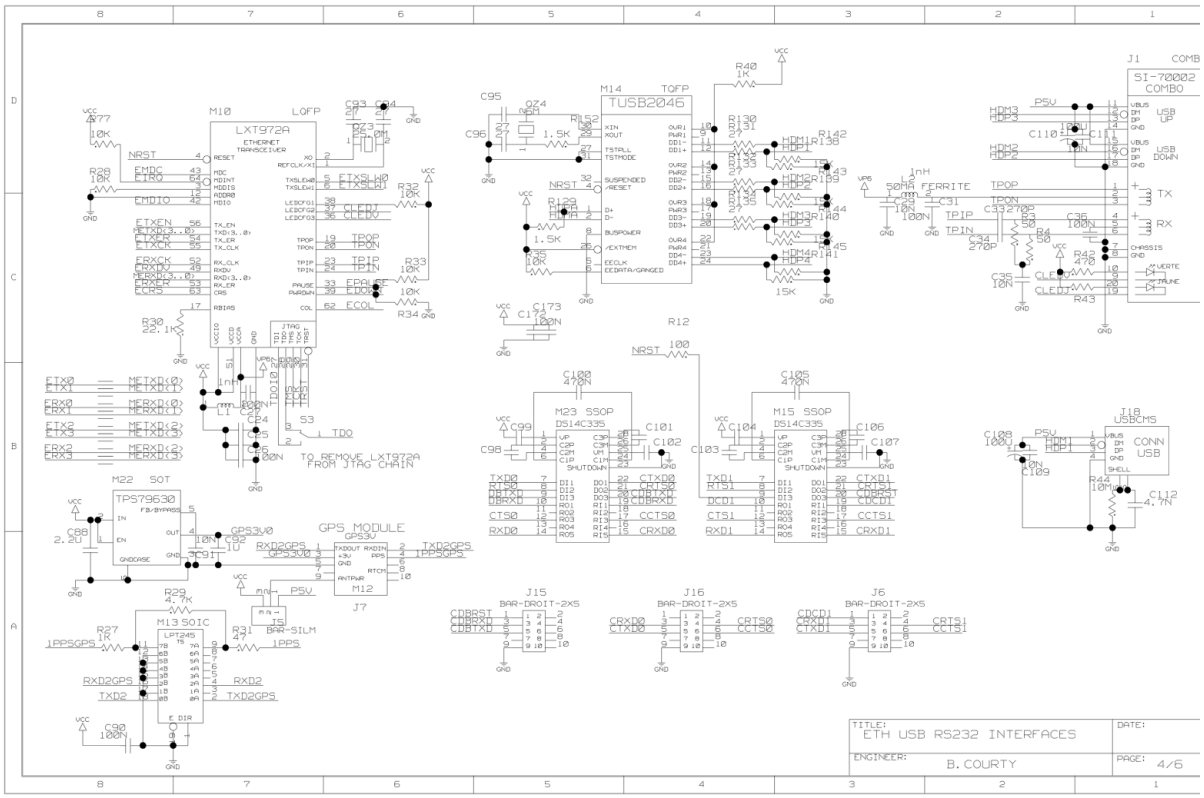

Figure 10: LSC: Ethernt, USB, RS232

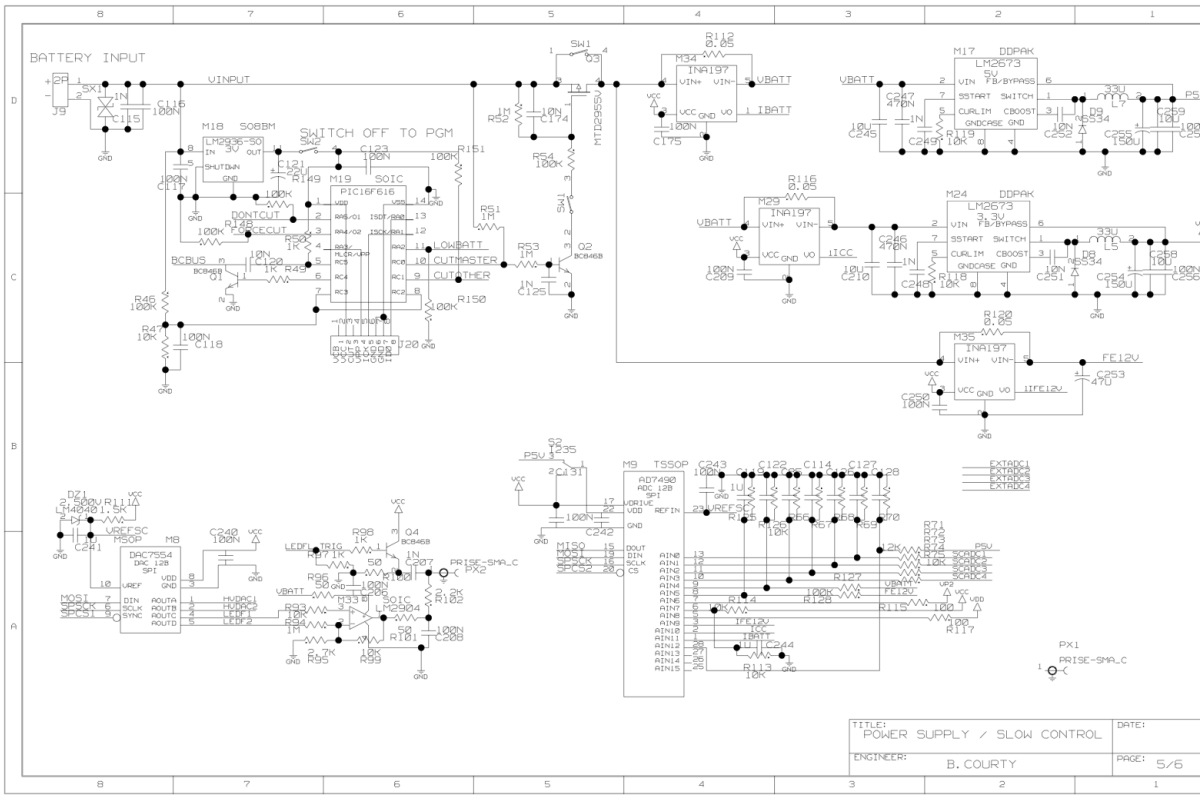

Figure 11: LSC: Power Supply, Slow control

H FrontEnd Schematics

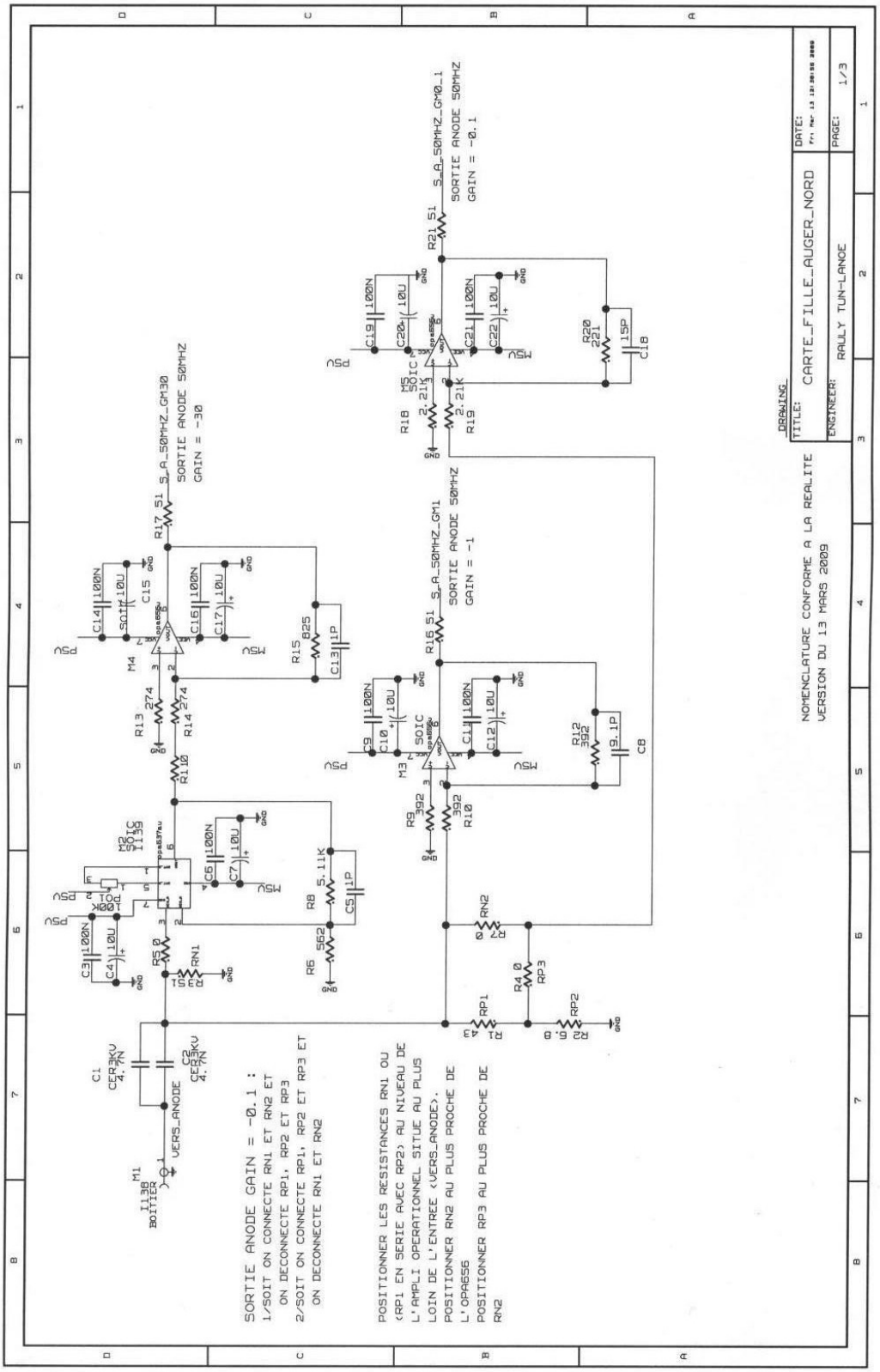

Figure 12: FrontEnd page 1

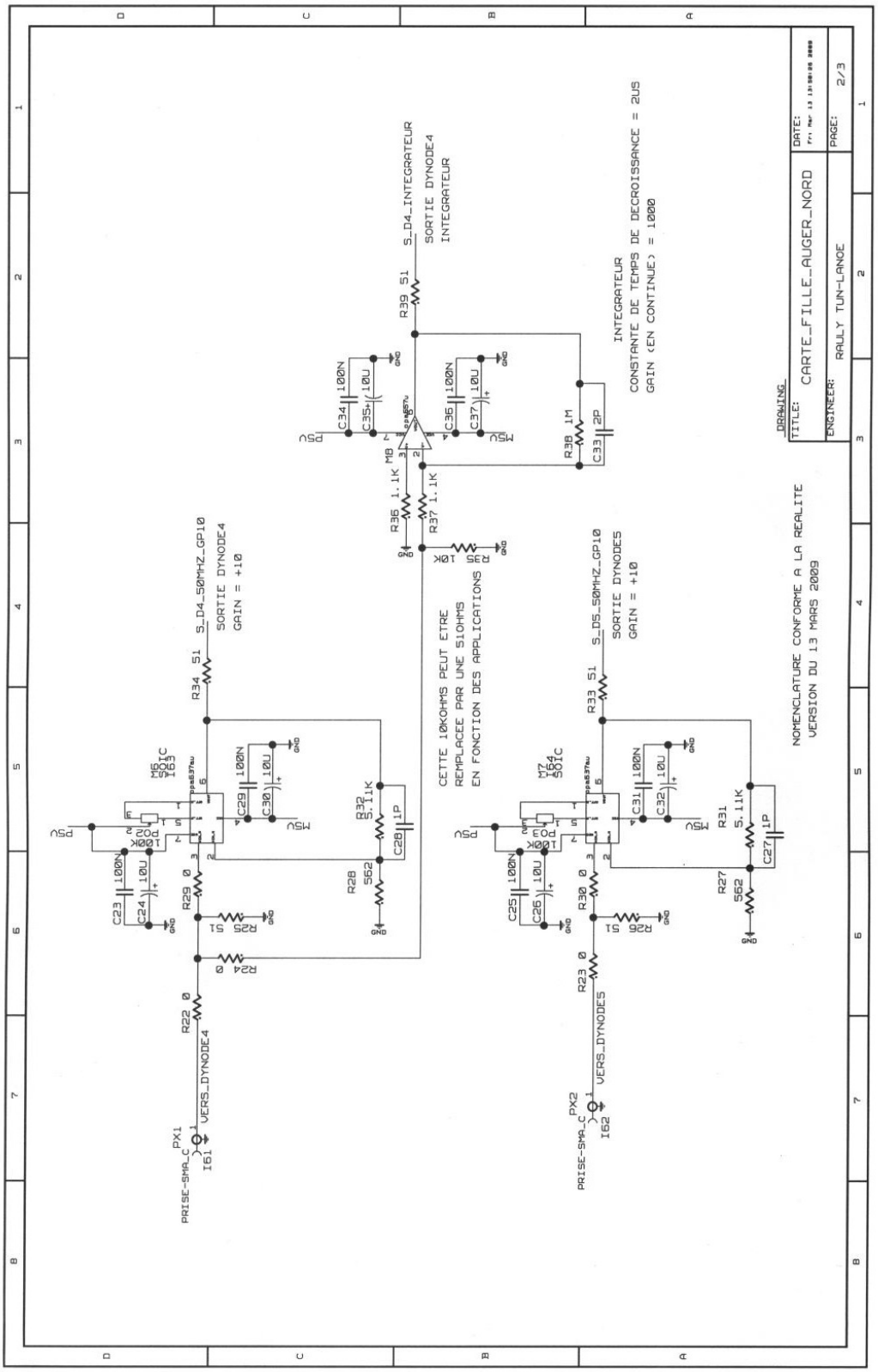

Figure 13: FrontEnd page 2

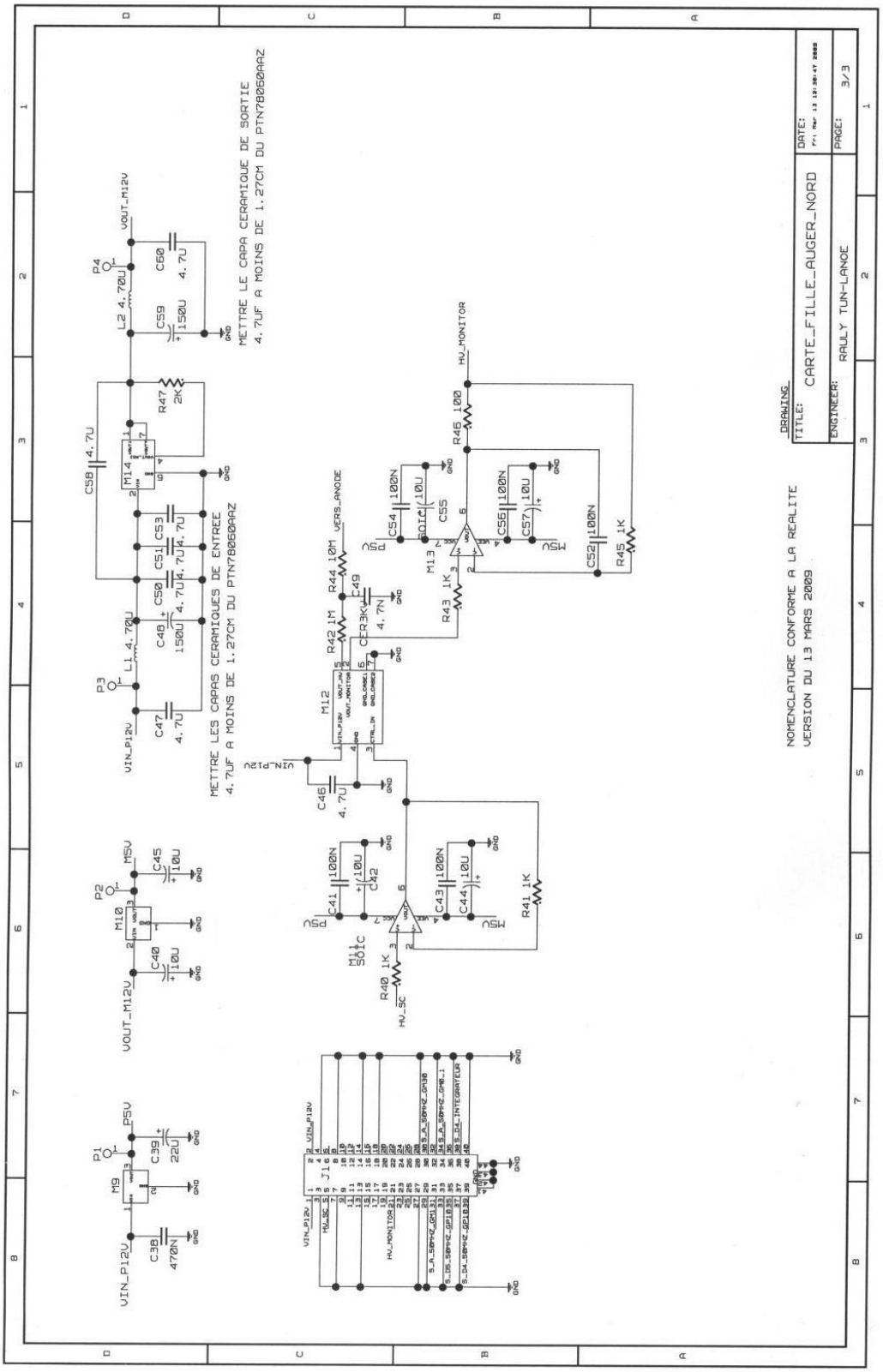

Figure 14: FrontEnd page 3