IRAM-COMP-059

Revision: 4
2010-08-10

Contact Author

Institut de RadioAstronomie Millimétrique

22GHz Receiver Control

New version based on NTP

Owners Alain Perrigouard (perrigou@iram.fr)
Francis Morel (morel@iram.fr)
Sebastien Blanchet

Keywords:

Approved by:	Date:	Signature:
A.Perrigouard	October 2009	

Change Record

REVISION	DATE	AUTHOR	SECTION/PAGE AFFECTED	REMARKS
03	05OCT09	F. Morel	Hardware	Suppressed hardware synchro using 1pps.
04	2010-08-06	S. Blanchet	Software	Update software section after removing hardware synchro 1pps.

Content

1 HARDWARE DESCRIPTION

1 22G VME Board description:

This board was specifically designed to control the 22G Receiver.
The board connects to the Receiver through 12 low-cost plastic fiber optics, in order to avoid interferences and ground loops.
The main parts of this board are 6 [31-bit + overflow] counters, used to integrate the V-to-F outputs issued from the receiver. These counters are first registered and then cleared upon each "LATCH" command received from the VME bus. The blanking (lost each second) time is typically 180 nanoseconds.
The board was built using a FPGA Altera EPF10K30.
The VME BUS is used as a 16-bit wide bus (D16 norm). This implies that each readout of a 32-bit word needs 2 accesses on the VME Bus.

1.1 Fiber outputs TO the receiver :

CMD_ LOAD_ON	Lit fiber moves the reference load in front of the receiver.	Static command bit
CMD_NOISE_ON	Lit fiber turns ON the noise diode of the receiver.	Static command bit
CMD_PWR	Unused, but functional	Static command bit

1.2 Fiber inputs FROM the Receiver:

LOAD_ON	Fiber is lit if the Reference Load is in front of the Receiver	Static Status bit
2.0000 MHz Reference	V/F reference frequency clock	Frequency encoded signal
LOAD_T	Reference Load temperature	Frequency encoded signal
PELTIER_T	Peltier regulator temperature	Frequency encoded signal
ALARM	Receiver alarm	Static Status bit
F3	Channel 3	Frequency encoded signal
F2	Channel 2	Frequency encoded signal
F1	Channel 1	Frequency encoded signal
F0	Channel 0	Frequency encoded signal

1.3 22G registers addresses:

All the registers are mapped in the A16/D16 VME I/O space => Address Modifier=29/2D. The board uses 128 word (even) addresses (XX00 to XXFE). The board Base-address bits [15-8] are selectable using 2 encoding wheels, **RC1 [A15-A12]**, and **RC2 [A11-A08].**
The VME A16 address on Plateau de Bure of the 22G Board is 0x1000.

1.4 Input registers (read-only):

Channel 0 LSW	Base-address + 0x0	Data bits[15-0]
Channel 0 MSW	Base-address +	Bit[15] = Overflow

	0x2	Bits[14-0]=Data bits[30-16]
Channel 1 LSW	Base-address + 0x4	Data bits[15-0]
Channel 1 MSW	Base-address + 0x6	Bit[15] = Overflow Bits[14-0]=Data bits[30-16]
Channel 2 LSW	Base-address + 0x8	Data bits[15-0]
Channel 2 MSW	Base-address + 0xA	Bit[15] = Overflow Bits[14-0]=Data bits[30-16]
PELTIER_T LSW	Base-address + 0xC	Peltier temperature bits[15-0]
PELTIER_T MSW	Base-address + 0xE	Bit[15] = Overflow Bits[14-0]=Peltier temperature bits [30-16]
LOAD_T LSW	Base-address + 0x10	Load temperature bits [15...0]
LOAD_T MSW	Base-address + 0x12	Bit[15] = Overflow Bits [14-0]=Load temperature bits[30-16]
2 MHZ LSW	Base-address + 0x14	2 MHz Ref bits[15-0] **(LSB = 500 nanoseconds)**
2 MHZ MSW	Base-address + 0x16	Bits [15-0]=2MHz Ref bits[31-16]
Channel 3 LSW	Base-address + 0x18	Data bits[15-0]
Channel 3 MSW	Base-address + 0x1A	Bit[15] = Overflow Bits[14-0]=Data bits[30-16]

1.5 Status register (read-only): Base-address + 0x1C

15	14-12	11-8	7-6	5	4-3	2	1	0
ERR	0	BD_CODE	0	ALARM	0	NOISE_ON	LOAD_ON	0

Description of the STS register bits :

Bit 15	ERR	ERR = ALARM
Bits[14-12]	0	Don't care
Bits[11-8]	BD-CODE	4-bit board revision code (0x03 for 2009 version)
Bits[7-6]	0	Don't care
Bit 5	ALARM	Set if Receiver Alarm is ON
Bit 4	0	Don't care
Bit 3	0	Don't care
Bit 2	NOISE_ON	Set when the Noise Diode has been requested to turn ON. This bit is **NOT** read from the receiver
Bit 1	LOAD_ON	Set when the Reference Load is in front of the Receiver
Bit 0	0	Don't care

1.6 Command Register (read/write): Base-address + 0x1E

Bits[15-4]	3	2	1	0
X	CMD_LATCH	CMD_NOISE_ON	CMD_LOAD_ON	CMD_PWR

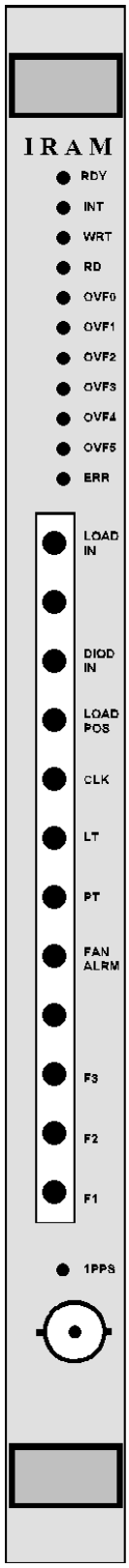
Description of the CMR bits :

Bits[15-4]	X	Irrelevant, always read as zeros.
Bit[3]	CMD_LATCH	Setting this bit with stop all counters, latch their contents into the output buffers, clear the counters and restart them. This operation takes less than 200 nanoseconds.

Bit[2]	CMD_NOISE_ON	When set, turns On the Noise Diode.
Bit[1]	CMD_LOAD_ON	When set, moves the Load in front of the Receiver.
Bit[0]	CMD_PWR	Unused, but functional

Data read from CMR will be the last value written into this register.

1.7 22G Board Front-Panel:



Red LEDs are used for displaying ERROR conditions only.

From top to bottom:

11 LEDs :

READY	Green	Power ON
INT	Yellow	Flash upon “latch” VME command
WRT	Yellow	Flash upon “write” VME access
READ	Yellow	Flash upon “read” VME access
OVF0	Red	Channel 0 Overflow
OVF1	Red	Channel 1 Overflow
OVF2	Red	Channel 2 Overflow
OVF3	Red	Channel 3 Overflow
OVF4	Red	Peltier_Temp Overflow
OVF5	Red	Load_Temp Overflow
ERR	Red	The Altera chip did not initialize upon power-on

N.B: All leds "OVFx" will turn on if ALARM = 1.

3 optical outputs (see Command register)

Output name	**Signal name**	**Comments**
LOAD IN	CMD_LOAD_ON	When ON, move the reference load in front of the receiver
	CMD_PWR	Unused in actual design, but functional
DIOD IN	CMD_NOISE_ON	When ON, turns ON the noise diode of the receiver

9 optical inputs (see Status register)

Input name	**Signal name**	**Comments**
LOAD POS	LOAD_ON	Turns ON when the reference load is in front of the receiver
CLK	2 MHZ	2 MHz reference from the receiver
LT	LOAD_T	Reference Load temperature measurement
PT	PELTIER_T	Peltier cooler temperature measurement
FAN ALRM	ALARM	Receiver alarm
	F3	Channel 3 measurement
F3	F2	Channel 2 measurement
F2	F1	Channel 1 measurement
F1	F0	Channel 0 measurement

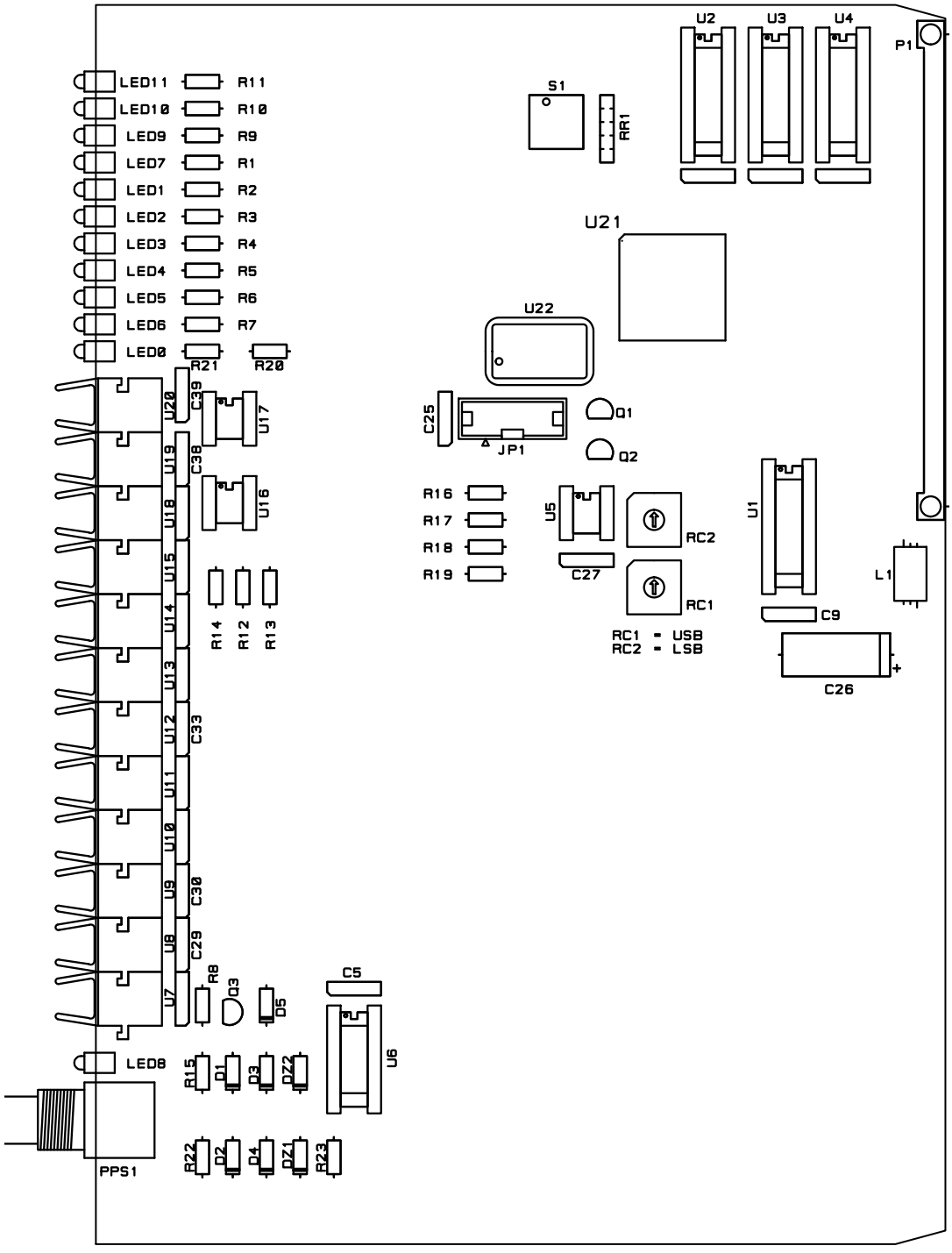
1 yellow LED:
"pps" pulse, unused in last design, always OFF.

1 BNC connector:
"TU01" input, unused in last design.

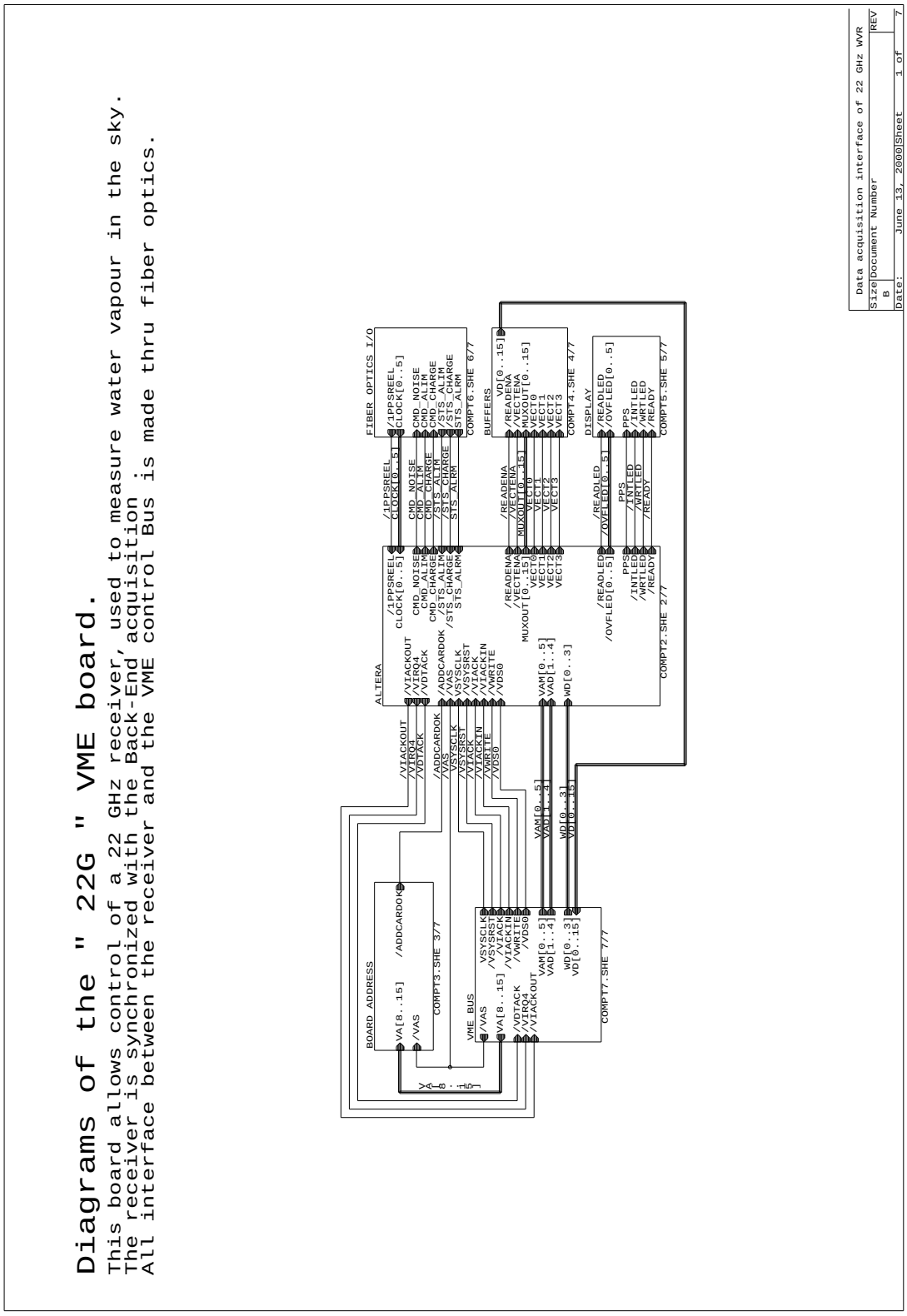
1.8 Getting started:

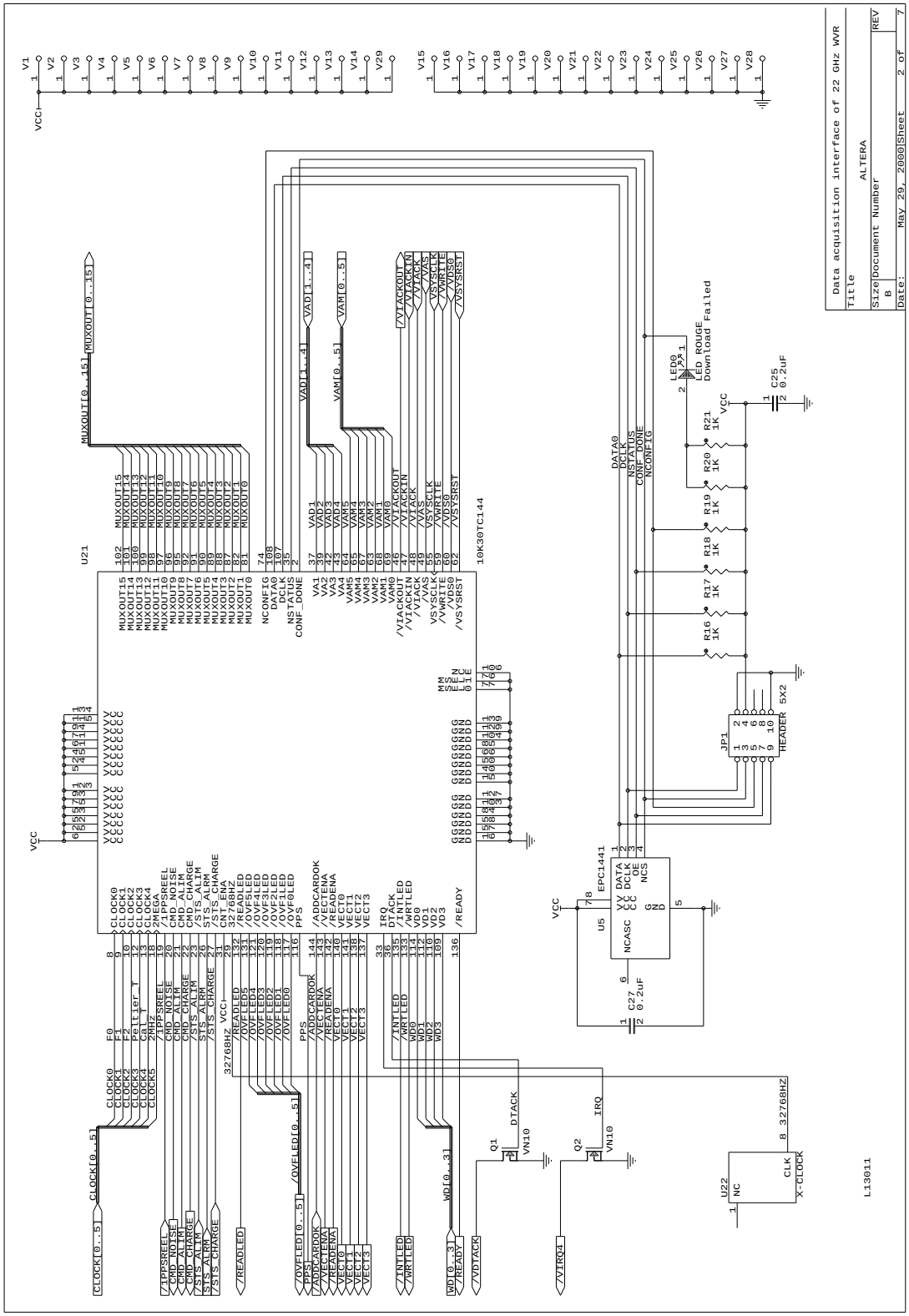
Select Base-Address [A15-A8] using RC1 and RC2.
Insert the board into the VME crate.
Connect the Receiver’s fiber optics.
Turn on the crate. The green LED "READY" should turn ON.

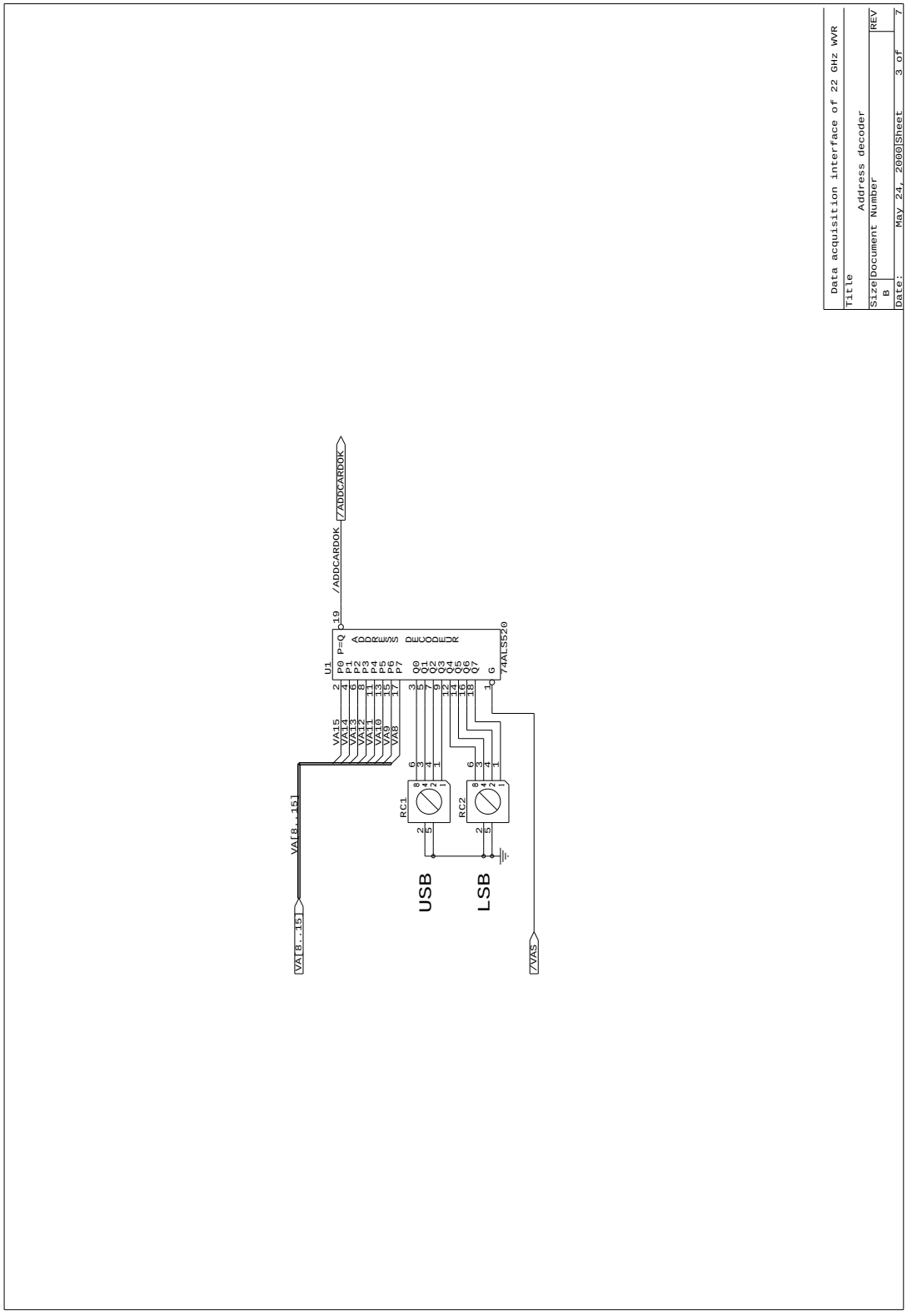
1.9 22G Board layout:

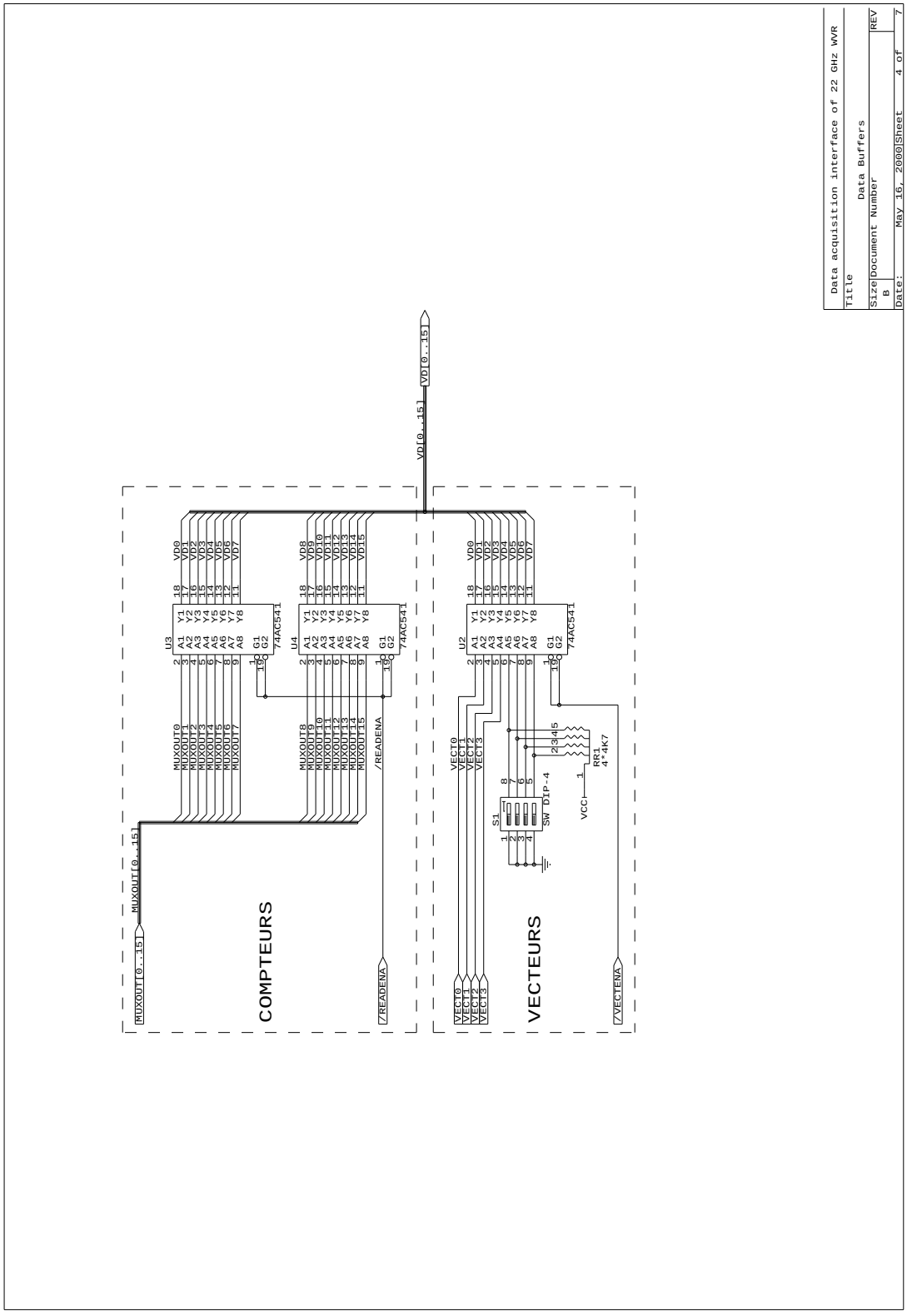


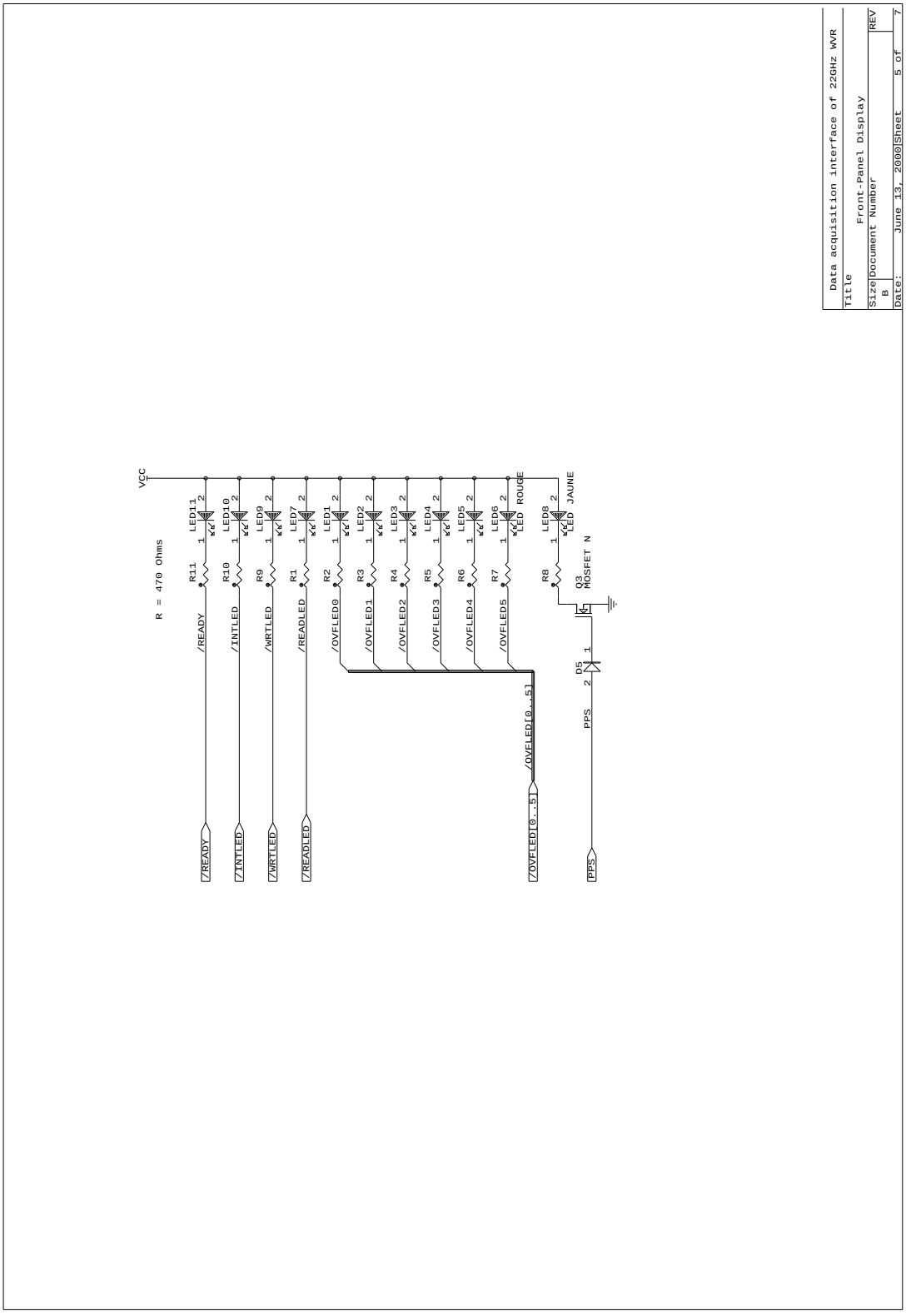
1.10 22G Board schematics:

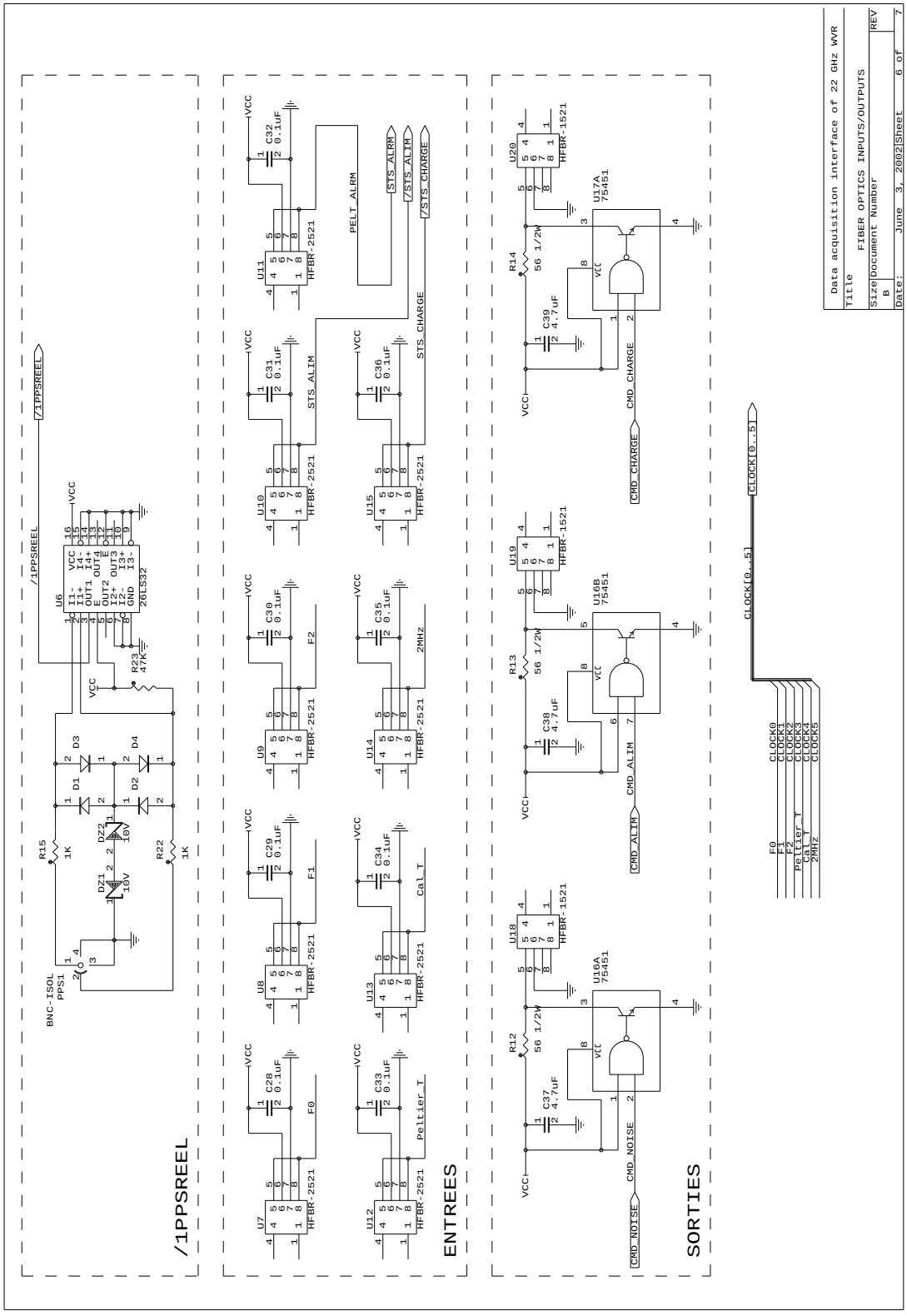


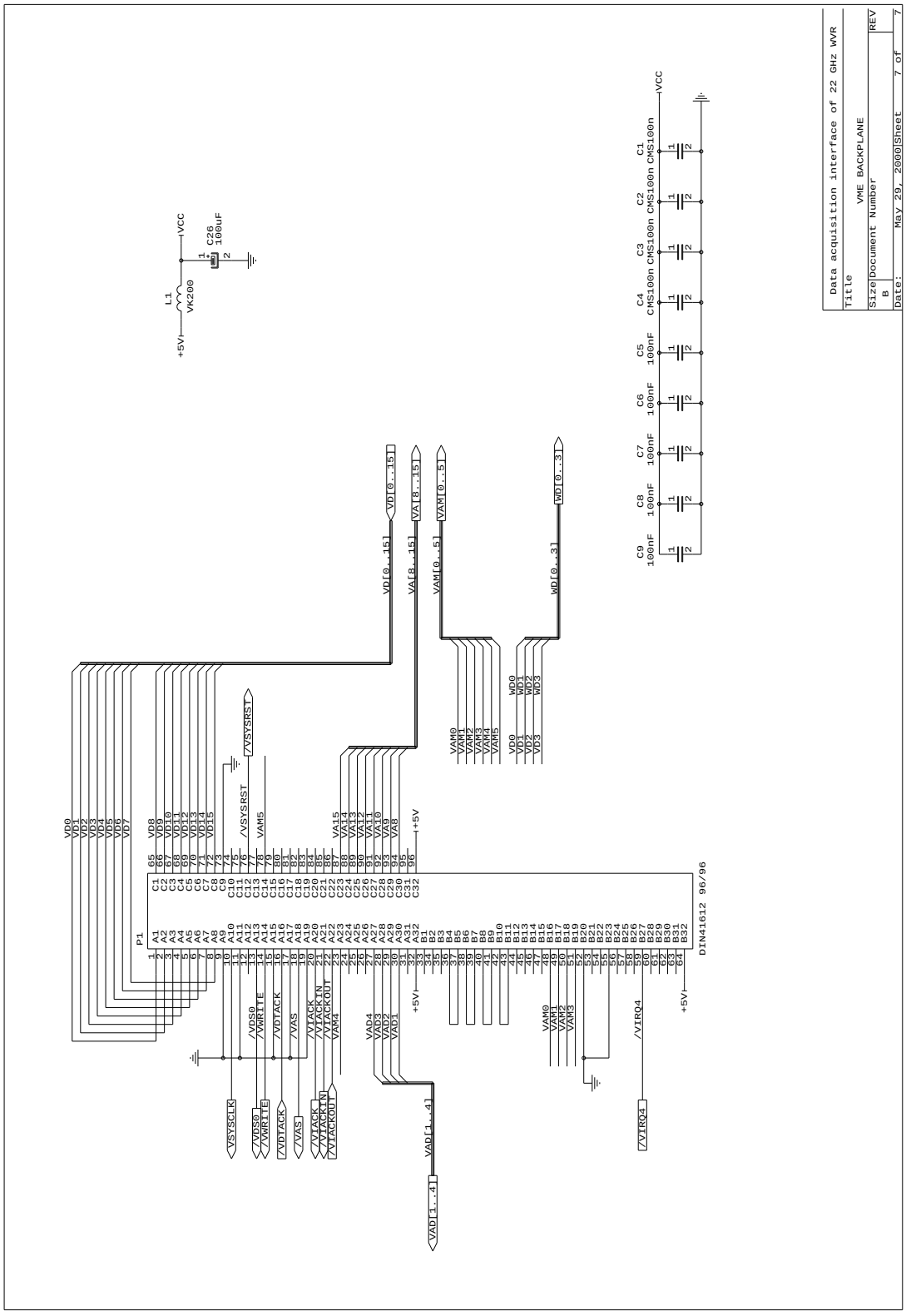

Data acquisition interface of 22 GHz WVR
Title
ALTERA
Size B
Document Number
REV
Date: May 29, 2000
Sheet 2 of 7
U21
10K30TC144
MUXOUT[0..15]
VAD[1..4]
VAM[0..5]
CLOCK[0..5]
/OVFLED[0..5]
WD[0..3]
/READY
/VDTACK
/VIRQ4
/INTLED
/WRTLED
/READLED
/1PPSREEL
CMD_NOISE
CMD_ALIM
CMD_CHARGE
/STS_ALIM
/STS_ALRM
/STS_CHARGE
/ADDCARDOK
/VECTENA
/READENA
VECT0
VECT1
VECT2
VECT3
PPS
32768HZ
DTACK
IRQ
/VIACKOUT
/VIACKIN
/VIACK
/VAS
VSYSCLK
/VWRITE
/VDS0
/VSYSRST
NCONFIG
DATA0
DCLK
NSTATUS
CONF_DONE
NCONFIG
LED0
LED ROUGE
Download Failed
JP1
HEADER 5X2
U5
EPC1441
NCASC
U22
X-CLOCK
Q1
Q2
VN10
R16 1K
R17 1K
R18 1K
R19 1K
R20 1K
R21 1K
C25 0.2uF
C27 0.2uF
L13011


Data acquisition interface of 22 GHz WVR
Title
Address decoder
Size
B
Document Number
REV
Date:
May 24, 2000
Sheet
3 of
7
U1
74ALS520
/ADDCARDOK
/ADDCARDOK
VA[8..15]
VA[8..15]
VA15
VA14
VA13
VA12
VA11
VA10
VA9
VA8
RC1
RC2
USB
LSB
/VAS


Data acquisition interface of 22 GHz WVR
Title
Data Buffers
Size B
Document Number
REV
Date: May 16, 2000 Sheet 4 of 7
VD[0..15]
MUXOUT[0..15]
COMPTEURS
VECTEURS
/READENA
/VECTENA
U3
U4
U2
74AC541
SW DIP-4
RR1
4*4K7
VCC


Data acquisition interface of 22GHz WVR
Title
Front-Panel Display
Size B
Document Number
REV
Date: June 13, 2000
Sheet 5 of 7
VCC
R = 470 Ohms
R11
R10
R9
R1
R2
R3
R4
R5
R6
R7
R8
LED11
LED10
LED9
LED7
LED1
LED2
LED3
LED4
LED5
LED6
LED8
LED ROUGE
LED JAUNE
Q3
MOSFET N
D5
/READY
/INTLED
/WRTLED
/READLED
/OVFLED0
/OVFLED1
/OVFLED2
/OVFLED3
/OVFLED4
/OVFLED5
/OVFLED[0..5]
PPS


/1PPSREEL
ENTREES
SORTIES
Data acquisition interface of 22 GHz WVR
FIBER OPTICS INPUTS/OUTPUTS
Document Number
June 3, 2002
Sheet 6 of 7
CLOCK[0..5]
CMD_CHARGE
CMD_ALIM
CMD_NOISE
STS_ALRM
/STS_ALIM
/STS_CHARGE
PELT_ALRM
F0
F1
F2
Peltier_T
Cal_T
2MHz


Data acquisition interface of 22 GHz WVR
VME BACKPLANE
May 29, 2000
DIN41612 96/96
P1
VCC
+5V
L1
VK200
C26
100uF
C1
C2
C3
C4
C5
C6
C7
C8
C9
CMS100n
100nF
VD0..15
VA8..15
VAM0..5
WD0..31
VSYSCLK
/VSYSRST
/VDS0
/VWRITE
/VDTACK
/VAS
/VIACK
/VIACKIN
/VIACKOUT
VAM4
VAM5
VAD1..4
/VIRQ4

2 SOFTWARE DESCRIPTION

The 22GHz receiver or water vapor radiometer control software is distributed between tasks running on VME processors and, commands and operations executed on the central computer bure1.

All receivers in each antenna receiver cabin are under control of a VME chassis equipped with a VMIC board running Linux as operating system. Those single board computers are named antX2, where X is the antenna number.

On bure1, a command **wvr** is provided to request calibration sequences. The low level function **write_wvr()** passing the request to the periodic task **interp22GHz** connected to all available radiometers is also provided to be included in any operator front task (e.g. OBS).
The periodic task is driven by messages received every second from the VME real time processor named clock. First, the task connects to all VME processors ant*2 having a radiometer under control and then, every second, collects blocks of data corresponding to the last 3 seconds and transfers eventually any new calibration sequence command.

3 Cabin software

This section explains the installation and the usage for the programs that run in the cabin.

3.1 Installation

3.1.1 Hardware

Required hardware

1. VMIC VMIVME 7700
2. VME 22GHz board

3.1.2 Software

Operating system
The software has been tested only with Linux (i386) 2.6.15 with RTAI 3.5. Nevertheless, it should run on any recent Linux with RTAI.

Mandatory software

The following libraries are required to build the software

Name	Description	Version	Download
g++	GNU C++ compiler	>= 4.1	http://gcc.gnu.org
subversion	Subversion is an open source version control system.	>= 1.5	http://subversion.tigris.org
Qt	C++ Cross-platform application framework	>= 4.4.3	http://www.qtsoftware.com
Sqlite	Embedded SQL database	>= 3.3.6	http://sqlite.org
CxxTest	Unary testing framework for C++	>= 3.10.1	http://cxxtest.tigris.org
Xmlrpc-c	XML-RPC for C and C++.	>= 1.06	http://xmlrpc-c.sourceforge.net

All these libraries are very common (except CxxTest), and you should find easily ready-to-install packages for your favorite Linux distribution.

3.1.3 Building

Extract the code from the repository

```
$ mkdir ~/develSVN
$ cd ~/develSVN
$ svn co svn://svn.iram.fr/PdB/RT_22GHz/trunk RT_22GHz
```

Then use the Makefile to extract automatically the dependencies

```
$ cd RT_22GHz
$ qmake
$ make get_deps
```

Build everything

```
$ ./build.sh
```

Optional, you can build the API documentation. The documentation will be created in the *doxydoc* subdirectory.

```
$ make doc
```

To install the programs and their default settings in `/home/introot/r22g`
Note: the shared libraries are installed in `/home/introot/lib`

```
$ su -c "./install.sh all"
```

Warning:
Normally, you need not to change the default settings, but if you wish to update only the software **without reinstalling the default settings**, use

```
$ make -f Makefile.install programs
```

Nevertheless it is safer to backup `/home/introot/r22g` and `/home/introot/lib` first, so you can restore the original installation in case of errors.

```
$ tar cvfz ~/r22g-backup-`date --iso`.tar.gz /home/introot/{r22g,lib}
```

3.1.4 Start application automatically

Add `/home/introot/r22g/bin/r22g-manager` to `/etc/rc.local` to start the control software at startup.

3.2 General description

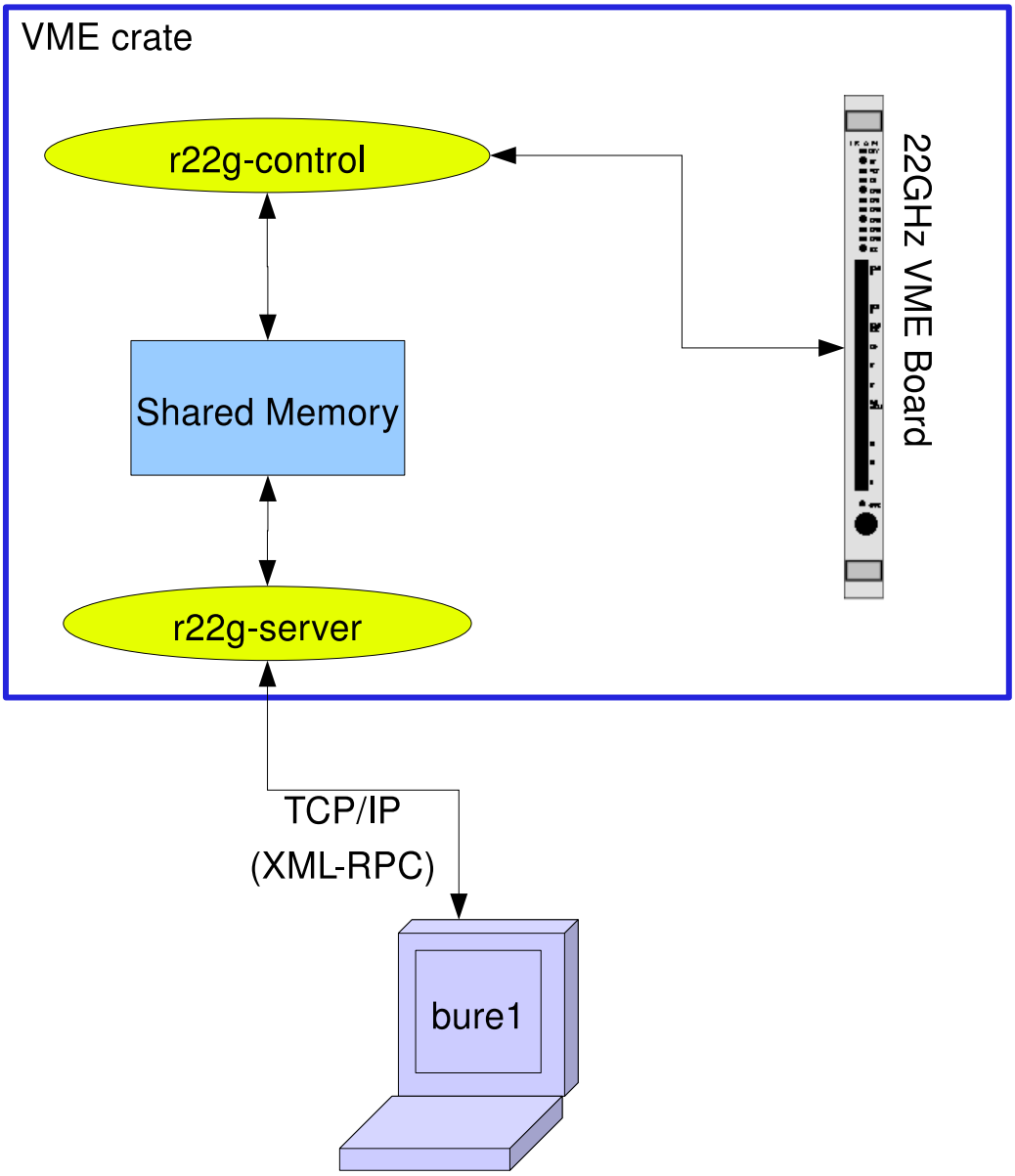
The previous control software uses a physical time bus with 1 pulse per second (1 pps) that generates a VME interrupt. Unfortunately, the electrical signal of the time bus becomes noisy with long distances. Therefore, an alternative solution must be found.

Therefore a software time bus (NTP) has replaced the physical time bus.

> *NTP (Network Time Protocol) is a protocol, designed to synchronize the clocks of computers over a network. It is an Internet standard protocol, widely used around the world.*

Then the controlling software is written as a real-time application that synchronized itself on NTP via the local clock.

> *For the real-time application, we choose RTAI (Real Time Application Interface), it is a real-time engine for Linux*



In each antenna cabin, **r22g-control** drives the 22GHz VME board and **r22g-server** handles the network requests from bure1.
Both applications communicate through a shared memory (`/dev/shm/rt_22GHz.shm`)

Note: in fact **r22g-server_legacy** replaces **r22g-server**. See details later in this document.

3.3 Programs

3.3.1 r22g-dump

r22g-dump print the shared memory.
The shared memory named is: `/dev/shm/rt_22GHz.shm`
The shared memory type is: `R22G_Common_t`, defined in file `libs/RT_Types.h`

3.3.1.1 Syntax

```
$ r22g-dump –h
22GHz Shared memory Dumper
Dump the 22GHz shared memory
Usage: r22g-dump [options]
Options:
  -v        Display version information
  -h, -?    Display help
```

3.3.1.2 Example

```
$ r22g-dump
----------Calibration----------------------
request.nphase   = 0
request.duration = 0  0  0  0  0  0
request.control  = 0x0  0x0  0x0  0x0  0x0  0x0
iphase           = 0
ilast            = 0
----------Readout----------------------
             Ch0       Ch1       Ch2   Peltier     LoadT     Clock       Ch3
channel  =     0         0         0         0         0   2000278   1000139
status   = 0x8320
control  = 0x0
ut_sec   = 65776
----------Measure----------------------
channel          = 0  0  0  289.8  279.7
status           = 0x8320
control          = 0x0
ut_sec           = 65774
----------Measure----------------------
channel          = 0  0  0  289.8  279.7
status           = 0x8320
control          = 0x0
ut_sec           = 65775
----------Measure----------------------
channel          = 0  0  0  289.8  279.7
status           = 0x8320
control          = 0x0
ut_sec           = 65776
-----------------------------------
index            = 2
it1sCnt          = 26
vref             = 2.898  2.797
gain             = 99  19.2
netRequestNum    = 0
error            = 0
```

3.3.2 r22g-manager

r22g-manager is the supervisor program that:
- Load real time modules
- Execute r22g-control and r22g-server_legacy

Features:
- Protection against concurrent executions: the program checks and insures that only one instance of r22g-manager is running.
- Autocleaning: subprocesses and modules are cleaned when exiting.
- Signals are caught to insure that cleaning is always done.

If you try run simultaneously two instances of r22g-manager, the 2nd instance detects the 1st one and kills it before starting.

3.3.2.1 Syntax

```
$ r22g-manager -h
R22G Manager
Start all required programs to operate the 22GHz board.
Safely clean the system when exiting:
  - 22Ghz programs are stopped.
  - RTAI modules are unloaded.
  - Signals are caught to insure that cleaning is always done.

Usage: r22g-manager [options]
Options:
  -v          Display version information
  -h, -?      Display help
  -c=arg      Pass 'arg' to r22g-control. Can be use several times.
  -s=arg      Pass 'arg' to r22g-server_legacy. Can be use several times.

Example:
  - Display r22g-control and r22g-server_legacy help:
        r22g-manager -c=-h -s=-h
```

3.3.3 r22g-logger

r22g-logger monitor the 22GHz shared memory

3.3.3.1 Syntax

```
$ ./r22g-logger -h
R22G Logger - Logger
Log receiver 22GHz value (read in the shared memory)
Usage: r22g-logger [options]
Options:
  -o=outputFileName   Output filename (if missing use stdout)

  -v                  Display version information
  -h, -?              Display help
```

3.3.3.2 Example

```
$ r22g-logger
# outputFilename = /dev/stdout
QFile::at: Cannot set file position 0
# Log receiver 22GHz value from the shared memory
# host: ant22b, start at Mon Sep 13 15:29:49 2010
#   sysclk    ut_sec control  status     ch0     ch1     ch2 peltier   loadT
clock2M     ch3
 48589.005     48589     0x0   0x300  108376  150555  148671 1000008  270805
2000017       0
 48589.503     48589     0x0   0x300  108376  150555  148671 1000008  270805
2000017       0
 48590.504     48590     0x0   0x300  108373  150555  148658 1000003  270796
2000008       0
```

3.3.4 r22g-control

r22g-control drives the 22GHz VME board. It is a RTAI-LXRT program. LXRT is an extension on RTAI to run hard realtime programs in userspace (as opposed to kernel space)

Each second, **r22g-control** reads the 22GHz VME counters and store them in a shared memory. Then it sleeps until the next second.

So the main routine looks like

```
forever {
   latch_vme_counters();
   store_counters_in_shared_memory();
   apply_calibration_next_state();
   sleep_until_the_next_second();
}
```

3.3.4.1 Syntax

```
$ r22g-control -h
22GHz Control Sofware
Control the 22GHz board
Read the 22GHz board every second
Usage: r22g-control [options]
Options:
  -v        Display version information
  -h, -?    Display help
```

3.3.5 r22g-server

r22g-server is a XML-RPC server to remotely control the 22GHz receiver.

> *What is XML RPC ?*
>
> *XML-RPC is a remote procedure call protocol that uses XML to encode its calls and HTTP as a transport mechanism. This protocol is very simple to use, and can be used from any programming language (many opensource libraries are available)*
>
> *For an quick introduction to XML RPC see http://en.wikipedia.org/wiki/XML-RPC*

r22g-server for XML-RPC calls on http://localhost:1089/RPC2

Note: The path /RPC2 is the default path for a XML-RPC server. Therefore, it can be sometimes omitted (it depends on the implementation library)
This server supports:

- introspection http://xmlrpc-c.sourceforge.net/introspection.html
- multicalls

3.3.5.1 Syntax

```
$ r22g-server -h
R22G Server - XML-RPC Server
Listen on port 1089
Usage: r22g-server [options]
Options:
  -v        Display version information
  -h, -?    Display help
```

3.3.5.2 API description

Since the XML-RPC server support introspection, we can retrieve the API with a simple program

```
$ cd develSVN.net/RT_22GHz/scripts/xmlrpc/
[blanchet@gre-vmic4 xmlrpc]$ ./ListMethods.py
# Connect to http://localhost:1089
r22g.getData
r22g.setCalibration
system.listMethods
system.methodHelp
system.methodSignature
system.multicall
system.shutdown
```

```
$ ./Introspection.py
# Connect to http://localhost:1089
------------------------------------------------------------------------
Name       : r22g.getData(  )
Return Type: struct
Description: Returns the 22GHz receiver data

------------------------------------------------------------------------
Name       : r22g.setCalibration( int, array, array )
Return Type: int
Description: Set Calibration.
Return code:
    - 0 : OK
    - 1 : Error
Syntax:  setCalibration( int nphase, array durationList, array controlList )

    - nphase is the number of calibration phase between 1 and 6
    - durationList is an array of integers. Each array item is a phase duration
in second
    - controlList is an array of integers. Each array item is the control word
to apply during the calibration phase
------------------------------------------------------------------------
Name       : system.listMethods(  )
Return Type: array
Description: Return an array of all available XML-RPC methods on this server.

------------------------------------------------------------------------
Name       : system.methodHelp( string )
```

```
Return Type: string
Description: Given the name of a method, return a help string.

----------------------------------------------------------------------
Name       : system.methodSignature( string )
Return Type: array
Description: Given the name of a method, return an array of legal signatures.
Each signature is an array of strings.  The first item of each signature is the
return type, and any others items are parameter types.

----------------------------------------------------------------------
Name       : system.multicall( array )
Return Type: array
Description: Process an array of calls, and return an array of results.  Calls
should be structs of the form {'methodName': string, 'params': array}. Each
result will either be a single-item array containg the result value, or a
struct of the form {'faultCode': int, 'faultString': string}.  This is useful
when you need to make lots of small calls without lots of round trips.

----------------------------------------------------------------------
Name       : system.shutdown( string )
Return Type: int
Description: Shut down the server.  Return code is always zero.

----------------------------------------------------------------------
```

3.3.6 r22g-server_legacy

r22g-server_legacy provides the same functionality than **r22g-server**, but it uses the former binary protocol. This program is needed until all the clients are rewritten to use the XML-RPC protocol.

The program listens on port tcp/1051.
This server is multi-threaded, so it supports several connections simultaneously.

3.3.6.1 Syntax

```
R22G Legacy Server
22GHz TCP server
Usage: r22g-server_legacy [options]
Options:
  -v        Display version information
  -h, -?    Display help
```

3.3.6.2 Protocol description

The binary protocol is complex, so this is a precise description.
Warning: The data must be sent in the network order (i.e. the most significant byte before)

To get data:
The client sends to the server a `RequestGet_t` structure with `calibrationSize = 0`

```
typedef struct {
   int32_t calibrationSize;
} RequestGet_t;
```

The server replies an `AnswerMeasure_t` structure.

```
#define Measure_Channel_MaxIndex 5
/** Data sent back to the observatory's main computer */
typedef struct {
   /** Normalized channels to transfer */
   float channel[Measure_Channel_MaxIndex];

   /** status register */
   uint16_t status;

   /** control register */
   uint16_t control;

   /** number of elapsed time since 00h00 UTC
    *   0 <= ut_sec < NUMBER_OF_SECONDS_IN_24_HOURS */
   uint32_t ut_sec;
} Measure_t;

#define ARCHIVE_MAX_MEASURE_NUMBER 3
typedef struct {
   Measure_t       measure[ARCHIVE_MAX_MEASURE_NUMBER];
} AnswerMeasure_t;
```

So, the client retrieves the measures for the last 3 seconds. The measure are chronologically sorted (the oldest before).

To send a calibration sequence

The client sends to the server a `RequestCalibrate_t` structure with `calibrationSize = 26`

```
#define MAX_PHASE_NUM 6

typedef struct  {
   /** number of phase  0 < nphase <= MAX_PHASE_NUM */
   uint16_t nphase;

   /** phase duration in second  */
   uint16_t duration[MAX_PHASE_NUM];

   /** control register to apply */
   uint16_t control[MAX_PHASE_NUM];
} CalibrationRequest_t;

typedef struct {
   /** @brief always equal to sizeof(calibration) */
   int32_t calibrationSize;
   CalibrationRequest_t calibration;
} RequestCalibrate_t;
```

There are up to 6 phases in a calibration sequence block. The number of phases is **nphase**. **duration[]** and **control[]** indicate the duration and the type of each phase.

The accepted values for control register are:

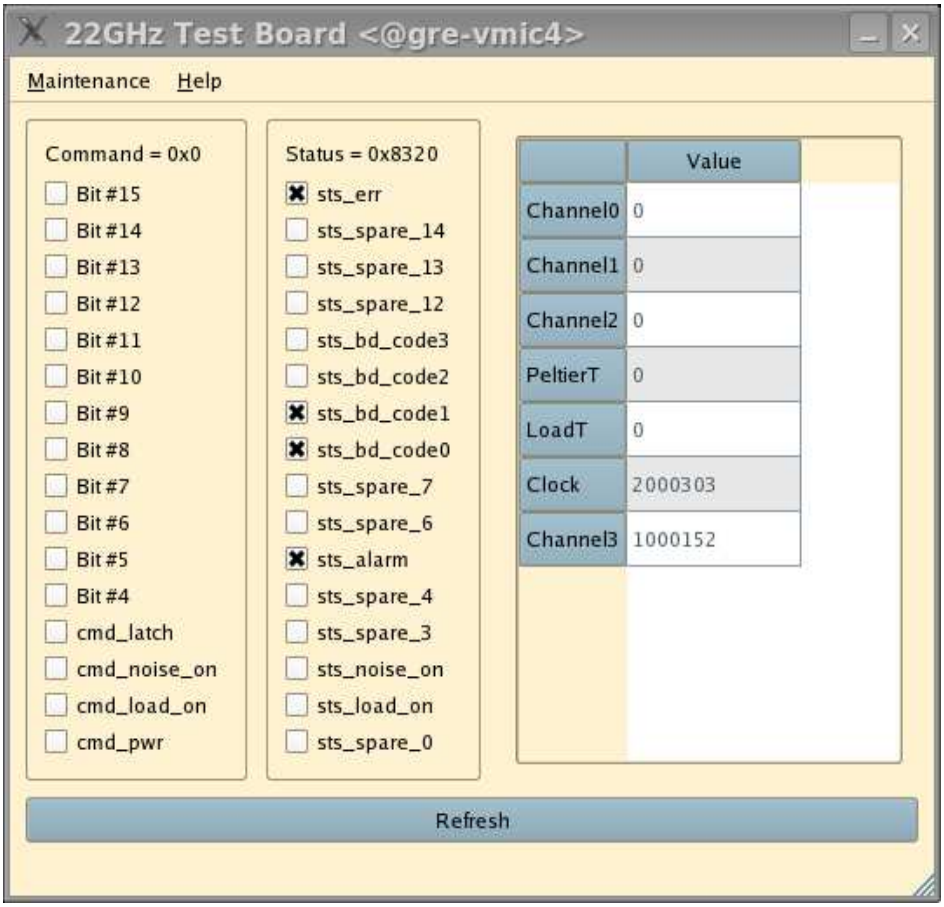
Control register value	Meaning
0x0	Load Off, Noise Off
0x2	Load On, Noise Off
0x4	Load Off, Noise On
0x6	Load On, Noise On

The server replies an `AnswerMeasure_t` structure. (See the *get data* case below)

The calibration sequence begins at the next second pulse.

3.3.7 r22g-test-board

r22g-test-board is a graphical program to set/read the 22GHz board registers.
It can be use to test the board or to watch that happen when other programs run.



3.3.8 r22g-client_legacy

r22g-client_legacy is the test program for **r22g-server_legacy**.

3.3.8.1 Syntax

```
r22g-client_legacy -h
R22G Client Legacy - Client Legacy
Test the legacy server for the 22GHz receiver
Usage: r22g-client_legacy [options]
Options:
  -s=serverName       Server hostname. Default = localhost
  -n=nphase           Number of calibration phase
  -d=d1,d2,d3         Duration of each calibration phase in seconds
  -c=c1,c2,c3         Control word for each calibration phase in seconds
  -m                  Monitor calibration sequence
  -p                  Print the answer details
```

```
  -v                  Display version information
  -h, -?              Display help


Example:
        r22g-client_legacy -n=2 -d=1,2 -c=4,2 -m

The program accepts deliberately invalid arguments to test the server program
robustness
Return code:
        - 0: OK
        - 1: An error has happened
```

3.3.9 Example

```
$ r22g-client_legacy -n=2 -d=1,2 -c=4,2 -m
serverName = localhost
nphase = 2
durationList = 1,2,0,0,0,0
controlList = 0x4,0x2,0x0,0x0,0x0,0x0
ut_sec = 53747.951
calibrationSize = 26 ; nphase = 2 ; duration = 1,2,0,0,0,0  ;  control =
0x4,0x2,0x0,0x0,0x0,0x0
[ ut_sec = 53747 ; control = 0x0 ]
[ ut_sec = 53748 ; control = 0x0 ]
[ ut_sec = 53749 ; control = 0x4 ]
[ ut_sec = 53750 ; control = 0x2 ]
[ ut_sec = 53751 ; control = 0x2 ]
[ ut_sec = 53752 ; control = 0x0 ]
[ ut_sec = 53753 ; control = 0x0 ]
```

On this example, we can see that the calibration sequence runs as expected:

- request is sent at 537473.95
- control word 0x4 is applied from 53748 to 53749 (1 sec)
- control word 0x2 is applied from 53749 to 53751 (2 secs)

3.3.10 r22g-update-db.sh

This program must be run after modifying settings in `/home/introot/r22g/data`
Typically, if you modify the *vref* and *gain* values for Peltier and Load Temperature in `/home/introot/r22g/data/config.sql`, then you have to run this program to apply the modifications.

Note: If you modify any setting, please send an email to program author to save your modification in the source code repository. Otherwise your modification may be overwritten when the software is reinstalled.

3.3.10.1 Syntax

```
r22g-update-db.sh
```

3.3.10.2 Example

```
$ r22g-update-db.sh
    cd /home/introot/r22g/data
    rm -f r22g.db
# Update r22g.db ...
    sqlite3 r22g.db < ./config.sql
    sqlite3 r22g.db < ./receiverId.sql
    sqlite3 r22g.db < ./serverId.sql
    sqlite3 r22g.db < ./TestBoard_Channel.sql
    sqlite3 r22g.db < ./TestBoard_Register.sql
# OK
```

4 Control room software

On bure1 several share memory areas are opened. Data related to the radiometer is found in the area called “RECE”.
When the application **interp** is executed the command **icps –ma** should show:

```
m    1607 0x52454345 --rw-rw-rw-  perrigou  computer  perrigou  computer      0
3360 23535 24524 14:27:08 no-entry 13:57:10
```

4.1 s_receiver.h

`s_receiver.h` describes each block of the set of 5 blocks contained in the share memory area “RECE”. There is one block per antenna and so far only 5 antennas are considered.
The include file `s_receiver.h` has been updated to contain the radiometer request and the block of data uploaded from the micros ant*2:

```
struct swvrRequest {            /* radiometer calibration request: */
        unsigned short nphase;  /* number of phase (0 < nphase <=6 ) */
        unsigned short duration[6]; /* duration in seconds */
        unsigned short control[6];  /* control word */
};

struct swvrTransfer {           /* radiometer data: */
        float channel0;         /* normalized channel 0 */
        float channel1;         /* normalized channel 1 */
        float channel2;         /* normalized channel 2 */
        float pelTemp;          /* normalized Peltier data */
        float ambTemp;          /* normalized load temperature data */
        unsigned short status;  /* status word */
        unsigned short control; /* control word (applied ~1s earlier)*/
        unsigned int ut_sec;    /* time associated to data and status*/
};

struct swvr {
        struct swvrRequest request;
        struct swvrTransfer transfer[3]; /* last 3s collected data */
};

struct s_receiver {
        struct sreceiver rec[2];
        struct sgeneral gen;
        struct swvr wvr;
        int f_rec[2];
        int lpclos[2];
        float temp[5];
```

```
        int f_wvr;
};
```

With the utility **c2f** the include file **s_receiver.h** is translated to **s_receiver.f**:

```
structure/ swvrRequest /
      integer*2 nphase
      integer*2 duration(6)
      integer*2 control(6)
      end structure
      structure/ swvrTransfer /
      real*4 channel0
      real*4 channel1
      real*4 channel2
      real*4 pelTemp
      real*4 ambTemp
      integer*2 status
      integer*2 control
      integer*4 ut_sec
      end structure
      structure/ swvr /
      record/ swvrRequest / request
      record/ swvrTransfer / transfer(3)
      end structure
      structure/ s_receiver /
      record/ sreceiver / rec(2)
      record/ sgeneral / gen
      record/ swvr / wvr
      integer*4 f_rec(2)
      integer*4 lpclos(2)
      real*4 temp(5)
      integer*4 f_wvr
end structure
```

4.2 Command wvr

The command **wvr** is used to request a new calibration sequence. As for all other commands already available in /control/command there is the possibility to indicate that this command is applicable to a specified antenna (a number from 1 to 6) or a telescope (a number from 1 to 6) which is a set of up to 5 antennas.
Remenber that the command **tel *tel_nbr*** gives the numbers of all the antennas belonging to this set.

Usage: wvr [qualifiers] wvr_parameters [qualifiers]
  wvr_parameters: nphase [duration control]...
  nphase: number of phase from 1 to 6
  duration: time duration in seconds (integer positive)
  control: 2 load on
        4 noise diode on
        6 both on
  qualifier: TELESCOPE telescope_number(1-6)
         ANTENNA antenna_number(1-6)
  Either ANTENNA or TELESCOPE is required

There should be a set { duration control} for each phase. So this set should be repeated nphase time.

The command checks the parameters, fills a block of data defined by **struct swvrRequest** and calls the function **write_wvr()** to save the parameters in the share memory area “RECE”, in the antenna specificied blocks.

The signature of this function is:

```
void write_wvr(p_iant, p_itel, p_request, p_general, p_receiver)
int *p_iant, *p_itel;
struct swvrRequest *p_request;
struct s_general *p_general;
struct s_receiver *p_receiver;
```

p_iant is a pointer to an antenna number.
p_itel is a pointer to a telescope number.
If the telescope number is different of zero the request structure is saved in the share memory area “RECE” in the antenna blocks, for all the antennas belonging to this telescope.
If the telescoper number is equal to 0, the antenna number should be different of zero, and the request structure is saved in the specified antenna block.
p_request is a pointer to a structure of type **struct swvrRequest**
p_general is pointer to the share memory area “GENE”.
p_receiver is a pointer to the share memory area “RECE”.

For each antenna concerned by this calibration sequence request the variable f_wvr is set to 1.

In Fortran the function may be called with the following arguments:

```
INCLUDE 's_general.f'
RECORD /S_GENERAL/ XG
COMMON /GENERAL/ XG
INCLUDE 's_receiver.f'
RECORD /S_RECEIVER/ XR(NB_ANT)
COMMON /RECEIVER/ XR
INTEGER IANT, ITEL
RECORD /SWVRREQUEST/ WVR
CALL WRITE_WVR(ANT, TEL, WVR, XG, XR)
```

4.3 dmp

This task is a debug program which displays the values of the receiver share memory area “RECE”.

Usage: dmp [qualifiers]
qualifier: ANTENNA antenna_number(1-5)

As the executable is not in the binary path, call this command with its absolute path. For instance: /control/receiver/dmp A 1

4.4 interp22GHz

This task connects to all micros ant*2 which can respond on port 1051 and which run **r22g_server_legacy**.
The task is started with the command:
/control/receiver/interp22GHz &

A semaphore which reports occurrence of 1s pulses drives the task. The task **flag1s** connected to the micro clock receives every second synchronization messages and signals these time events by clearing semaphores.
The task **interp22GHz** in an endless loop waits for the semaphore **flg_ut22GHz** to execute the following sequence:

The task tries non-blocking connections to all micro ant*2. Any returned positive socket indicates the success of the connection and prevent any further connection. The task will try to connect to all antennas (receiver cabin micro) not yet connected or temporary disconnected.
Once this connection test executed, the task sends and receives data only to the connected micros.
For each connected micro, the task

- Sends a 4 byte word called **request** equal to zero if no calibration sequence is pending for this micro or equal to the size of the request block sent later at the end of the transaction with this antenna.
- Receives a buffer of 3 block of data corresponding to the last 3 seconds of acquisition, status and control.
- If a calibration sequence is pending, i.e. if **f_wvr** is different of zero, the task sends the request block and clear **f_wvr**. The calibration sequence will starts at the next 1s interrupt on the connected micro.

The share memory area is locked with semaphore **flg_s_rec** when the 3 blocks of data are written and when a calibration sequence block is sent. This prevent any collision with the command **wvr** and in particular the function **write_wvr()** when a new a calibration sequence is requested and also with any other task trying to read the radiometer data as long as the task uses the same procedure of exclusive access to the share memory area “RECE”.

A 4 s timeout is reset every second before trying to connect to all unconnected micros and transferring data to connected ones. This timeout releases any send or receive operations stuck for any reason. Usually when a micro ant*2 is disconnected (due for instance to a reboot), the current Ethernet transfer function does not recover and gets stuck. The 4s timeout prevents this problem and let **interp22GHz** to reconnect later to this micro.