Continuum IRAM Detector

P. de Vicente, G. Paubert, J.A. López-Pérez,
R. Bolaño, J. González, M. Patino, C. Almendros

Informe Técnico IT-CDT 2014-12

Revision history

Version	Date	Author	Updates
1.0	14-08-2013	P. de Vicente	First draft

Contents

1 Introduction **3**

2 Description of the equipment **3**

3 Software installation **8**
3.1 Software in the Suzaku 8
3.2 Applications to retrieve and view the data 13

4 Mounting the 6-channel continuum backend **16**

5 Tests **16**

1 Introduction

A new 6 channels continuum detector has been built at Yebes Observatory. It is foreseen that more units of this detector will be built for all RAEGE antennas and for the 40 m radiotelescope. This detector is a copy of the IRAM detector, able to process up to 16 channels, designed and built by G. Paubert with the help of S. Sánchez (IRAM Granada, Spain). This report **only** describes the mounting process and tests. Requests for further information on the design should be addressed to G. Paubert at Institute de Radioastronomie Millimetrique.

2 Description of the equipment

The IRAM detector is a continuum backend which can process up to 16 channels simultaneously. The system has an input band between 50 MHz and 18 GHz and uses an analog to digital conversion of 16 bits at 100 kHz. It is composed of several parts:

1. a detecting circuit using a diode
2. a second circuit which regulates and stabilizes the temperature to 50 C and does the analog to digital conversion
3. a central board with a FPGA which collects the signals from several detectors, provides the power supply to the previous circuits and interfaces with a remote user which requests the measurements.

The detected values are periodically delivered via sockets to a remote server. The detecting circuit with the diode and the regulating circuits are in two boards which are enclosed on an aluminium box. Below we include a brief description on both boards.

The detecting circuit uses a diode part number model MP1602 from M Pulse Microwave and a choke P/N:CC45T47K240G100 from Piconics. They are mounted on a board, designed by Gabriel Paubert, with dimensions 2×2.5 cm which is printed on both faces. The blue regulator (see Fig. 1) is used to adjust the zero level of the detector and can be setup with a small screw driver. The golden pad that can be seen on the left panels is prepared to solder the pin from the connector on top of it.

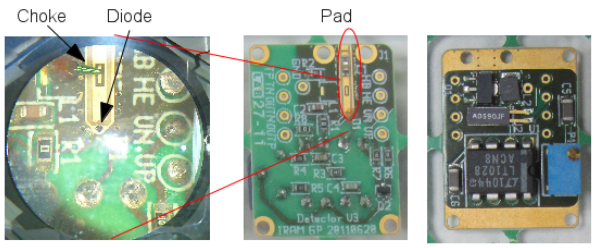

Figure 1: *Board for the detector. The panels on the right show the two faces. The blue regulator is used to setup the zero offset. The most left panel is a zoom of the golden pad to show the location of the choke and the diode.*

The temperature regulator and ADC board, designed by Salvador Sánchez, is larger, 10×4 cm, and although all its components are mounted on one face it is printed on both sides. It has a 15 pin high density D-sub male connector. Fig. 2 shows the populated face. The ADC uses 16 bits, and a 100 KHz signal. The clock runs at 2.5 MHz which means that a sample is taken every 25 clock cycles. The signal for every bit is transmitted using LVDS (Low-Voltage Differential Signaling). In Fig. 2 we have annotated places where signals can be monitored to debug the overall function of the detector:

- TP. Test points for 5 V, +15 V and -15 V
- Clock signal in the ADC
- ADC output
- Analog signal before ADC and copy.
- Signal in the LVDS emitters.

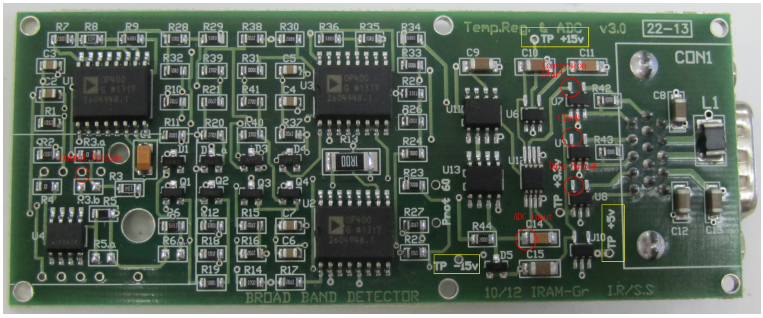

Figure 2: *Temperature regulator and ADC board. Some test points have been marked for debugging purposes: Input voltage (from the diode), power supply at 15V and -15V, input signal at the ADC, output signal from the ADC, clock and LVDS emitter (conversion start). The hole on the left bottom side allows to pass a screw driver through it for the blue regulator in the detector board*

Both boards are installed inside a small aluminium box specially milled (on both faces) to hold them. The small board is lodged in one of the sides covered by a small cover. Fig. 3 shows the hole with several lowerings to accomodate the regulator and chips on the board. The cables soldered in the 8 points (4 at each side) of the board pass through two slots in the internal wall towards the larger board on the other side of the box. There is also another connection through a circular hole that allows to access the blue regulator with a screw driver. The signal from the receiver is injected trough a SMA connector on one lateral side of the box.

The larger board is lodged in the other side of the aluminium box and requires a special milling to adapt to the D-SUB connector and a lowering to allow for space for a plastic cover for the cables. As mentioned before both boards are connected through several cables but we do not have details on these connections since they were done by G. Paubert at IRAM.

Aluminium boxes schematics were provided by IRAM in STP format (3D format) and were converted by Javier González to AutoCAD format. Boxes were manufactured in the Yebes workshop. The boxes underwent a chemical bath of nickel in Yebes chemical lab to allow a

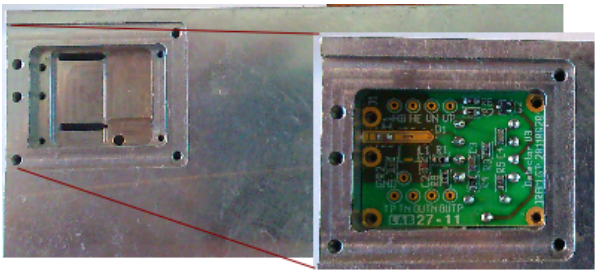

Figure 3: *Detector board in its enclosing inside the aluminium box. The left picture shows the lowerings for the components on the board, plus two slots where the cables pass through towards the other side of the box. There is also a hole for accesing the screw in the blue regulator. Cable connections are done in the 8 circles, 4 on each side of the board, labelled as: HB, HE, VN, VP and TP, IN, OUTN, OUTP. There is also a hole, not visible from above, in front of the golden pad for the SMA pin.*

better soldering of the bit perl. The central SMA connector (0,46 mm diameter) is mounted in a bit perl (Thunderline-Z) whose external face is soldered to the nickeled box. The hole requires very precise dimensions and a step in the box wall (a hole with different diameters to lock the perl inside, see Fig. 4). Unfortunately due to the lack of a proper tool in the workshop the holes did not have the nominal sizes and this posed some trouble in the high frequency end of the detector (above 14 GHz). The pin inside the box was soldered to the golden pad using a bonding machine.

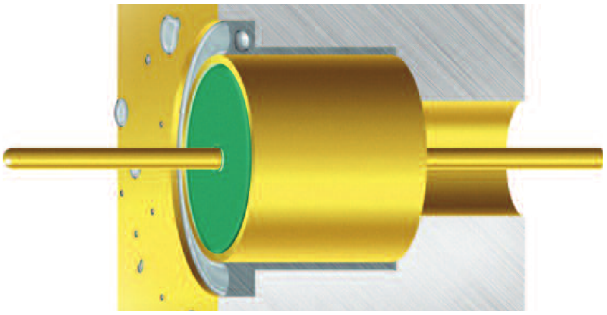

Figure 4: *Drawing of a bit perl connector for SMA inside its hole. The central conector ends at the SMA on one side and the other end is soldered onto the pad in the diode board. The receptacle has to be carefully milled to hold the perl bit correctly*

Once the pin was soldered on the pad the small cover was placed and screwed. Fig. 5 shows the moment in which the Temperature regulator and ADC board was about to be mounted on the other side of the aluminium box and Fig. 6, the closed box from different views. In the appendix we include the schematics of the box.

The main board where the Suzaku is mounted was designed by S. Sánchez and G. Paubert and is displayed in the upper panel of Fig. 7. The Suzaku is mounted in the center of the board. It contains a FPGA from Xilinx plus a PowerPC405 (32 bit RISC core) CPU whose specifications can be found at `http://suzaku-en.atmark-techno.com/series/suzaku-v`. The lower panel shows a schematics of the Suzaku with special emphasis on the serial port, the ethernet port and the supply voltage connector. The FPGA can manage peripherals and has 86

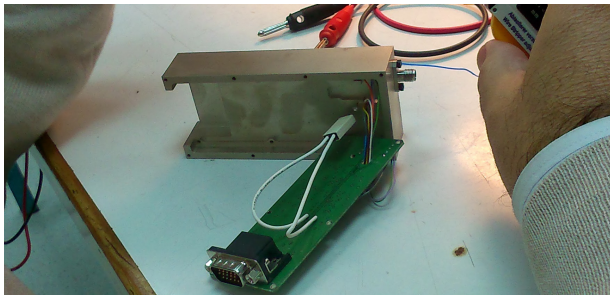

Figure 5: *Detector box while passing the cables from one side to the other. The visible side is where the temperature regulator and ADC board is inserted.*

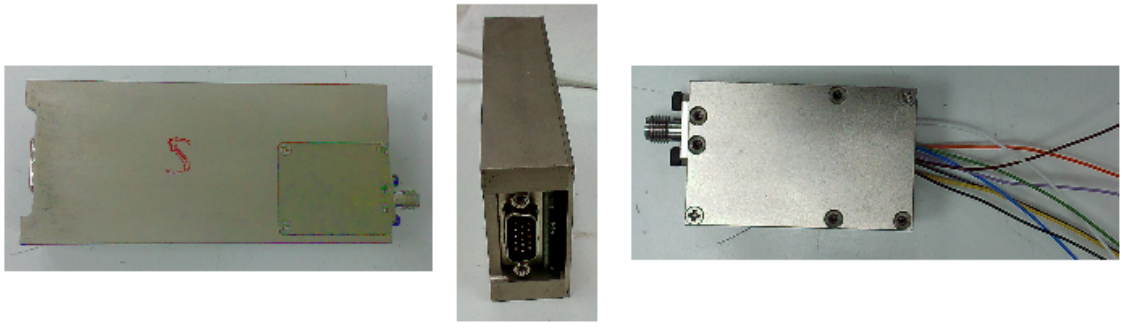

Figure 6: *Views of the box. Left: below the small cover is the detector board. Center: the DB-15 connector is used for the signals and for supplying the voltage to the box. Right: wires to the DB-15 port were not soldered yet.*

pins for I/O connections. In the Fig. 7 we can see that most of the pins are soldered, in lower layers of the multilayer board to the 16 connectors distributed on 2 rows of 8 connectors each at both sides. In the picture there is a color flat cable connected in position 9 while doing tests. A LED at the bottom of the Suzaku board lights indicating if the board is running correctly (more details can be found in next section).

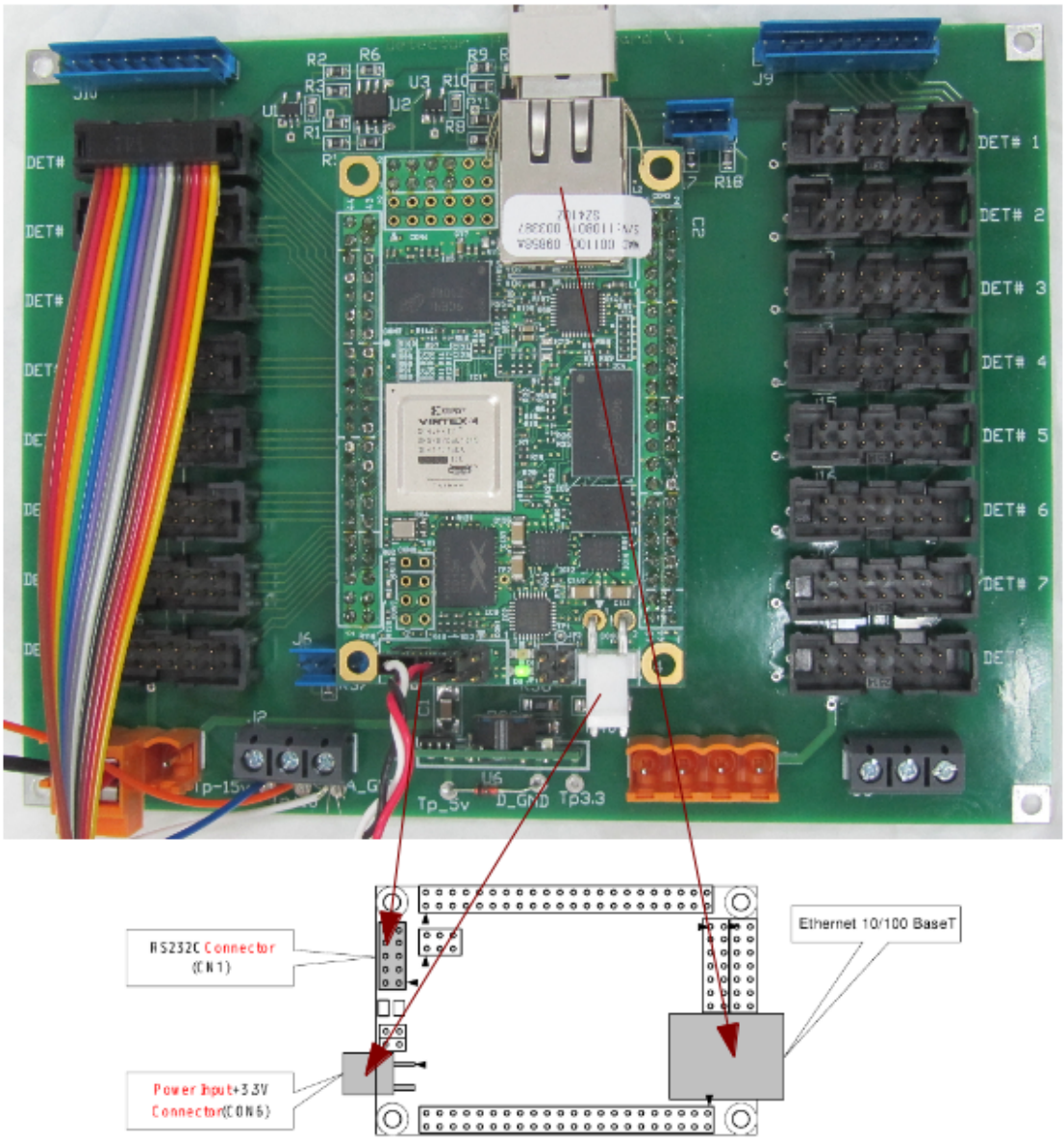

Figure 7: *Scheme of the Suzaku with the ports (serial, ethernet), I/O pins and supply connector.*

The Suzaku CPU runs Linux 2.6 (μClinux distribution). The linux kernel and applications have to be cross developed in Debian. In next section we provide some information on this task. As with the temperature regulator and ADC board, there are some test points at this board which can be used for debugging. They have been tagged in Fig. 8 and they include the voltage

from the diode, the 100 KHz signal for the ADC and the 2.5 MHz clock.

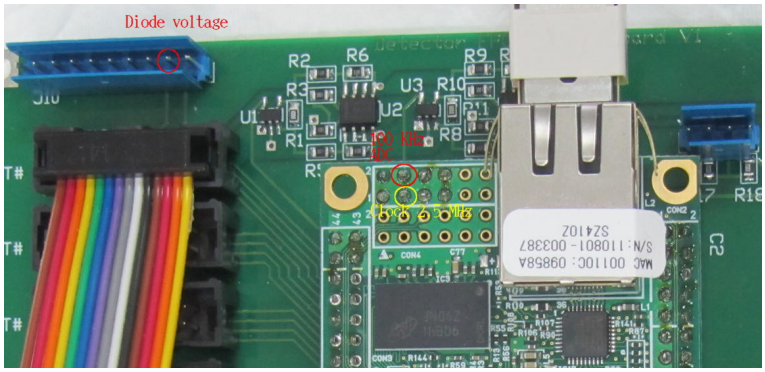

Figure 8: *A zoom of the Suzaku integration board with 3 test points tagged. Top left (red): diode voltage from detector on port 9. Top (red): 100 KHz signal for ADC. Bottom (yellow): 2.5 MHz clock signal.*

Connections between the D-15 pin connector at the temperature regulator and ADC board and the flat cable to be plugged in the main Suzaku board are summarized on table 2.

Signal	Pin Suzaku	Color	Pin D-15
Digital GND	1	brown	13
+ 5V	2	red	15
-15 V	3	orange	14
+ 15V	4	yellow	11
Analog GND	5	green	12
Voltage diode	6	blue	10
Clock +	7	violet	1
Clock -	8	grey	6
Conversion Start +	9	white	3
Conversion Start -	10	black	2
Data -	11	brown	5
Alarm	12	red	9
Data +	13	orange	4
OK (temp?)	14	yellowo	8
Not connected	15	green	7

Table 1: *Connections for the flat cable between the detector and the Suzaku boart.*

3 Software installation

3.1 Software in the Suzaku

The Suzaku-V runs a Linux Kernel (μClinux) on a power-pc architecture and it uses the FPGA as a peripheral device. Fig. 9 shows a schematics of the block diagram. This requires that the

FPGA is correctly programmed to perform its main function: read and integrate the signals from the detectors. Therefore the main task of the software configuration is loading a customized kernel and the FPGA code.

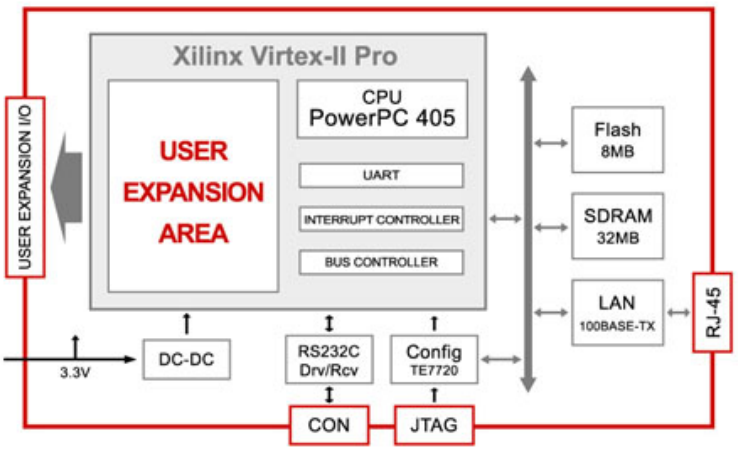

Figure 9: *Block Diagram of the Suzaku-V (SZ310-U00).*

Installing software in the Suzaku requires a host running Linux/Debian i386 where we will cross compile all its code. Multiarchitecture on an AMD 64 distribution was tested and did not work correctly and therefore all software was installed in a laptop running the latest Debian distribution with i386 architecture. The list of packages required for cross compilation is available at `http://download.atmark-techno.com/suzaku/cross-dev/powerpc/deb/`.

Debian packages for cross compiling on a power-pc architecture were downloaded on a separate directory and installed following these steps:

```
cd detector/suzaku/cross-dev/powerpc/deb/
cd libs
dpkg --install *.deb
cd ..
dpkg --install *.deb
```

The Suzaku needs to run a slightly customized kernel with a module, *iram_detector*, which is the driver for */dev/iram-detector*. A daemon *iram-detector* reads the data from the device (which can be considered to be located in the FPGA), integrates and sends the data to an external host using TCP sockets. Hence two main operations have to be fulfilled: installing the linux kernel, the module and the daemon and loading the FPGA code.

G. Paubert, who has written the code to be loaded into the FPGA, the *iram_detector* driver and the *iram-detector* daemon has organized the software in three main directories:

- `fpga`. This directory contains the code for flashing the FPGA with the firmware required to read data from the detectors.
- `suzaku/dist/atmark-dist-20090318`. This directory contains the software written by G. Paubert and is derived from the original distribution directory from Suzaku

- `mrt`. This directory contains the code for the driver, the code for the `iram-detector` daemon, the data server (`datacollector`) that runs on a standard Linux server and gets the data from the former, a graph chart recorder (`vcr`) which displays the detected voltage from all detectors in real time and as a function of time and the Linux driver (`kernel/drivers/char/iram-detector`). The directory structure is as follows:

```
mrt
|-- include
|   |-- backend
|   |   |-- iram-detector.h
|   |   |-- suzaku_iram_detector.h
|   | -- constants.h
|-- linux
|   |-- datacollector
|   |   |-- 1mhz.c
|   |   |-- 1mhz_scratch.c
|   |   |-- continuum.c
|   |   |-- datacollector.c
|   |   |-- datacollector.h
|   |   |-- getconf.c
|   |   |-- Makefile
|   |   |-- threading.c
|   |-- vcr
|   |   |-- gtkhelpers.h
|   |   |-- Makefile
|   |   |-- vcr.c
|   |   |-- vcr.cpp
|   |   |-- vcr_next.c
|-- uclinux
|   |-- detector
|   |   |-- detector.c
|   |   |-- detectors.c
|   |   |-- iram-detector.c
|   |   |-- mcast.c
|   |-- kernel
|   |   |-- drivers
|   |   |   |-- char
|   |   |   |   |-- iram-detector.c
```

Once the packages for cross compilation are installed, two basic compilations are done:

- The FPGA firmware is created making the software in its appropiate directory:

```
cd fpga
make
```

  and the image is generated in `fpga/mrt_cont.bin`

- The Linux kernel to be uploaded in the Suzaku CPU was compiled as follows:

```
cd suzaku/dist/atmark-dist-20090318/
rm config/scripts/tk*.o
make xconfig
make
```

  when running `make xconfig` it is necessary to enable section “network dhcpd” to be able to have a DHCP client. In our first trial this was not chosen and Suzaku could not obtain an IP address from DHCP server.

The previous instructions not only generate a Linux kernel, but also a romfs file system that is loaded into memory in the Suzaku CPU. The romfs contains the *iram-detector* "daemon" among others, like some usual binaries from 'busybox' which supplies the most usual tools available in the standard bash shell. The images from the kernel, the romfs and the whole image are created during compilation at subdirectory `images`:

```
cd suzaku/dist/atmark-dist-20090318/images
ls
image.bin  image.elf  linux.bin  romfs.img
```

`images/image.bin` contains the kernel image (including all its drivers) and the romfs filesystem. The content of the romfs file system can be seen in subdirectory `romfs`.

This philosophy of a romfs filesystem which is loaded in RAM memory on a diskless CPU is typical of μClinux and it is described at
`http://download.atmark-techno.com/suzaku/manual/suzaku_software_manual_en-1.0.2.`
Applications and third party software can be installed easily following the instructions there.

iram-detector can be compiled separatedly or from the linux kernel tree above. In the former case:

```
cd mrt/uclinux/detector
make
```

Both images, the one that configures the FPGA and the whole μCLinux image (kernel plus romfs), are transferred to the Suzaku using ftp:

```
ftp suzaku
cd tmp
put images/image.bin /tmp/image.bin
put fpga/mrt_cont.bin /tmp/mrt_cont.bin
```

and flashed using:

```
telnet suzakuy
cd tmp
netflash -C -b -k -r /dev/flash/fpga mrt_cont.bin
netflash -b -k -n -r /dev/flash/image image.bin
```

After flashing **each** image we should wait at least 2 minutes. Once both images are flashed the Suzaku should be rebooted.

In case of problems with the network, it is still possible to connect to the CPU using a serial connection via RS232. This was the case when we first tested the kernel. As we have mentioned above the DHCP client did not start after booting and it was necessary to edit file `/etc/init.d/network`:

```
#!/bin/sh

PATH=/bin:/sbin:/usr/bin:/usr/sbin

echo "Setting up interface lo: "
```

```
ifconfig lo up 127.0.0.1

echo "Starting DHCP client: "
dhcpcd &
```

and run again `/etc/init.d/network`. This modification allowed the DHCP to start but it was not permanent since the image is decompressed and installed in memory every time the Suzaku boots. This issue was solved as described above when configuring the kernel to be compiled.

A serial port connection requires installing "minicom" at the laptop. The setup should be 115 kbs, 8N1, no flow control and hardware RTS disabled. The pin-out between the Suzaku and the DB9 was done as described in table 3.1.

DB9	Suzaku	Signal
2	5	RX
3	3	TX
5	9	GND

Table 2: *Pin out to get a serial port connection to the Suzaku. The console will be implemented through this connection.*

The Suzaku uses a tiny partition in memory, with a ramfs file system, that once flashed is kept after any reboot. In fact directory `/etc/config` is placed there and contains some files:

```
dhcpcd-eth0.cache  dhcpcd-eth0.info   network           ntp.conf
passwd             rc.local           resolv.conf       resolv.conf.sv
```

Files `ntp.conf` and `rc.local` were setup for Yebes:
ntp.conf:

```
server 193.146.252.15 minpoll 4 maxpoll 6
logfile /tmp/ntp.log
driftfile /etc/config/ntp.drift
```

rc.local:

```
#!/bin/sh
PATH=/tmp:/bin:/sbin:/usr/bin:/usr/sbin
echo "Loading iram-detector module and starting the process"
modprobe iram-detector
iram-detector -m 1 -s 25000 geovlbi >>/tmp/iram-detector.log 2>>/tmp/iram-detect
```

'iram-detector' acts as a daemon which sends the data to 'geovlbi'. Option '-s' states that every dump is composed of 25000 samples, which means data are sent every 0.25 s and option -m is a mask for the valid channels (only in use at the IRAM telescope).
As we mentioned before, after modifying the files it is required to flash them to keep them permanent:

```
flatfsd -s
```

The indication that the flashing procedure completed succesfully comes from a message indicating how many bits have been recorded.

Once the kernel module, `iram-detector` has been loaded in memory and starts working, the red LED from the Suzaku turns into green. To view if the module is working correctly two tests can be run:

```
cat /proc/interrupts
           CPU0
  0:      31481  Xilinx Interrupt Controller Level     eth0
  1:         22  Xilinx Interrupt Controller Edge      uartlite
  3:     421258  Xilinx Interrupt Controller Level     iram-detector
BAD:
```

Around 16000 interruptions per second should appear.

Active partitions can be viewed typing:

```
mount
/dev/mtdblock7 on / type romfs (ro)
/proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
none on /var type ramfs (rw)
none on /etc/config type ramfs (rw)
```

3.2 Applications to retrieve and view the data

Fig. 10 shows a schematics of the connections to get and display data from the detector on a host connected to the local area network. The data sent by *iram-detector* are received by "datacollector" in the host specified in `/etc/rc.local` as a parameter of the daemon. In our case this is "geovlbi.oan.es". "datacollector" acts as a server that delivers the data every 0.25 seconds to any client that connects to it from any host in the LAN. Two applications usually make this connection: "vcr" and the component used in the observation process.

'datacollector' was written by G. Paubert and adapted to our needs. One of the modifications performed consisted on removing the backlog. The original version had a buffer of the last 30 minutes of data (backlog) and it was providing this buffer to any client connected to it. This behaviour caused some problems with our component and was disabled. In function `add_repeater_client` the following code was commented:

```
/**
if (bbc.count > 1) {
    rp->bbc.tail = bbc.head;
}
*/
```

'datacollector' is in the "mrt" tree and compilation is done running make:

```
cd mrt/linux/datacollector
make
```

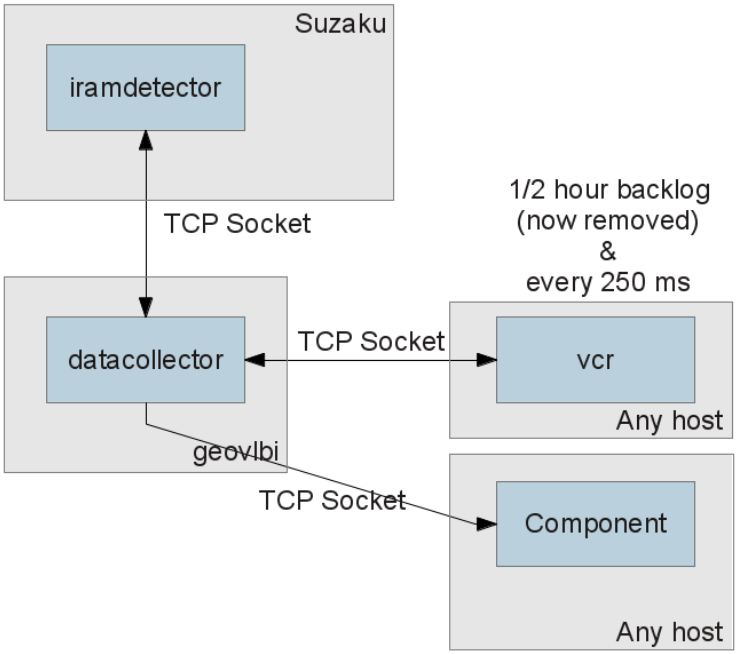

Figure 10: *Data flow among clients and servers*

"vcr" is a graphical chart recorder written by G. Paubert also included in the 'mrt' tree. It communicates with 'datacollector' and can be run at any host. The client requires openGL and as such it is recommended to start it on the console of the host where it runs. Compilation and package installation is achieved as:

```
apt-get install libgtk2.0-dev
cd mrt/linux/vcr
make
```

The client was modified to display the correct tags for the channels in Yebes. It has been compiled in different computers and provides real time data from the detector (see Fig. 11). The application has a panel at the left and a graph recorder at the right. The panel at the left is used to display the voltage from the 6 available detectors. In the upper part the user can select from a dropdown list the channel to be plotted in the chart recorder. Sliding bars, two at the bottom and a second pair at the right allow to control the scale and zoom of the graph. As we will mention later the value of the voltage helps to identify if the IF power is in the correct range. Values above 120 and below 0.1 should be avoided.

Finally a component in C++ was developed at Yebes to obtain the data and serve this data to the FITS writer. The component reads the socket connected to "datacollector", until some data is available. Data acquisition is performed in two steps: it reads a part composed of an identifier and a tag, and decides what to read next depending on the value of the tag:

- If the tag is `TAG_BBC_THERM`, it reads a structure containing thermal data in four fields: a mask, the values and a time tag.

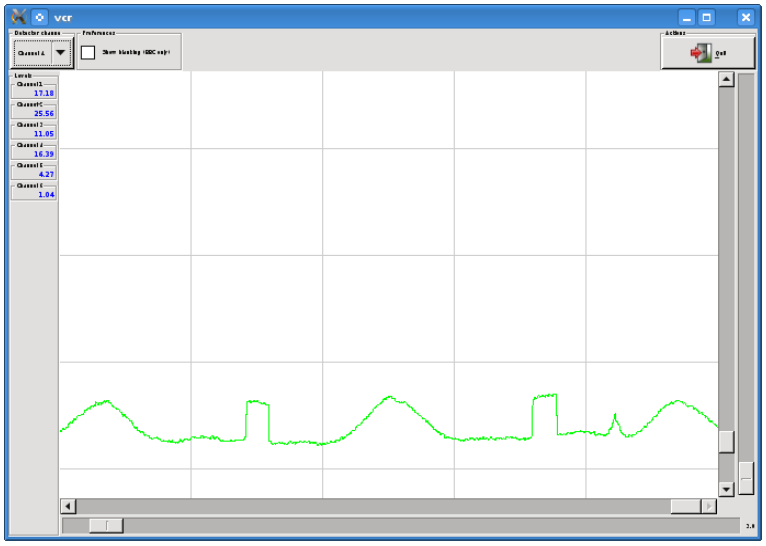

Figure 11: *Graphical chart recorder for the 6 channel continuum detector. Sliding bars at the bottom and right side allow to control the scale and zoom of time and intensity of the signal. The dropdown list on the top left corner allows to choose the channel we want to plot. Values from all channels are displayed in real time on the left panel.*

- If the tag is `TAG_BBC_DATA`, it read a structure containing the data grouped on phases. the data contain information on the phase, flags, integration time in μs, and the current data for the available channels. Currently we do not use phases but the system is prepared for that. That would be the case when using a chopper or a switched noise diode. In this latter case a sync signal should be injected at the module.

The component works better if no backlog is present in the data flow because the data treatment is simpler. The backlog is a feature which allows to recover old data from the last 30 minutes before delivering the live data. It is specially prepared for the 'vcr' application but it increases the complexity of our component. As we mentioned above it is disabled in the "datacollector" code.

The instance is called 'IRAMDETECTOR'. The component has been developed as a characterisc component, where the voltage from every channel is one characteristic. The main advantage of this feature is that we can create graphical monitors to display that data from all receivers. Some basic methods are available:

- *setIntegrationTime(double)*
- *startIntegration()*
- *stopIntegration()*
- *abortIntegration()*
- *getDumpTime()*
- *getThermData()*

- *getData()*

their names identify their goal. The usual procedure to work with this component is to setup the integration time, start integration and subscribe to a notification channel which delivers the data from the 6 channels after each integration period.

4 Mounting the 6-channel continuum backend

All the hardware parts, aluminium boxes with the detector board, temperature regulator and ADC board in each box plus the main Suzaku board were mounted on a 3U 19 inch rack, with space for the power supply and the cabling. The rack has an ethernet and a serial port connection. The former for normal operation (control and data transfer) and the latter for debugging purposes in case the first one fails. It also has a LED which indicates the user if the temperature regulator is working in all detectors. In case of failure in any of them the light will light red. At the front there are six inputs and six outputs which are a copy of the input signal before getting into the detectors.

Fig. 12 shows a composite image with the rear and front view and a view from above without the cover. The rear face shows inputs for 5 MHz, 1 PPS, Status and Blanking signal plus the serial port and ethernet connector. For normal operation the 5 MHz signal is enough. From above one can see that the signal for each channel is splitted and one branch is directed to the detector and the second one goes out again, as a copy. The detected voltage is sent to the Suzaku board, together with temperature information, in a flat cable that also is used to provide power to the detectors.

Installed splitters are model ZFSC-2-4 from Minicircuits and work from 0.2 to 1 GHz. If higher bandwidths were required they should be replaced by new ones with wider bands. Since the future bandwidth of the IFs at the 40m will be 2.5 GHz, splitters covering this bandwidth should be used.

5 Tests

The basic tests consisted on injecting noise into the 6 detectors with a wide band source and measure the linearity along 49 dB. Results are summarized in Fig. 13. Two important conclusions can be drawn:

- The dynamic range of the continuum detector is at least 3 orders of magnitude, an excellent value.

- The linearity of the backend goes from 120 to 0.1 for all detectors.

ADC saturation is reached with an input signal of 50μW or -13 dBm. The detector saturates close to -10 dBm. Diodes are destroyed with signal levels above +18 dBm. The zero offsets were tuned at the lab but we still get non-zero (0.01) offsets in some detectors.

The iram detector, called in the control system of the telescope *idet* backend, has been successfully used for continuum mesaurements at the 13m with the S, X and Ka band receivers.

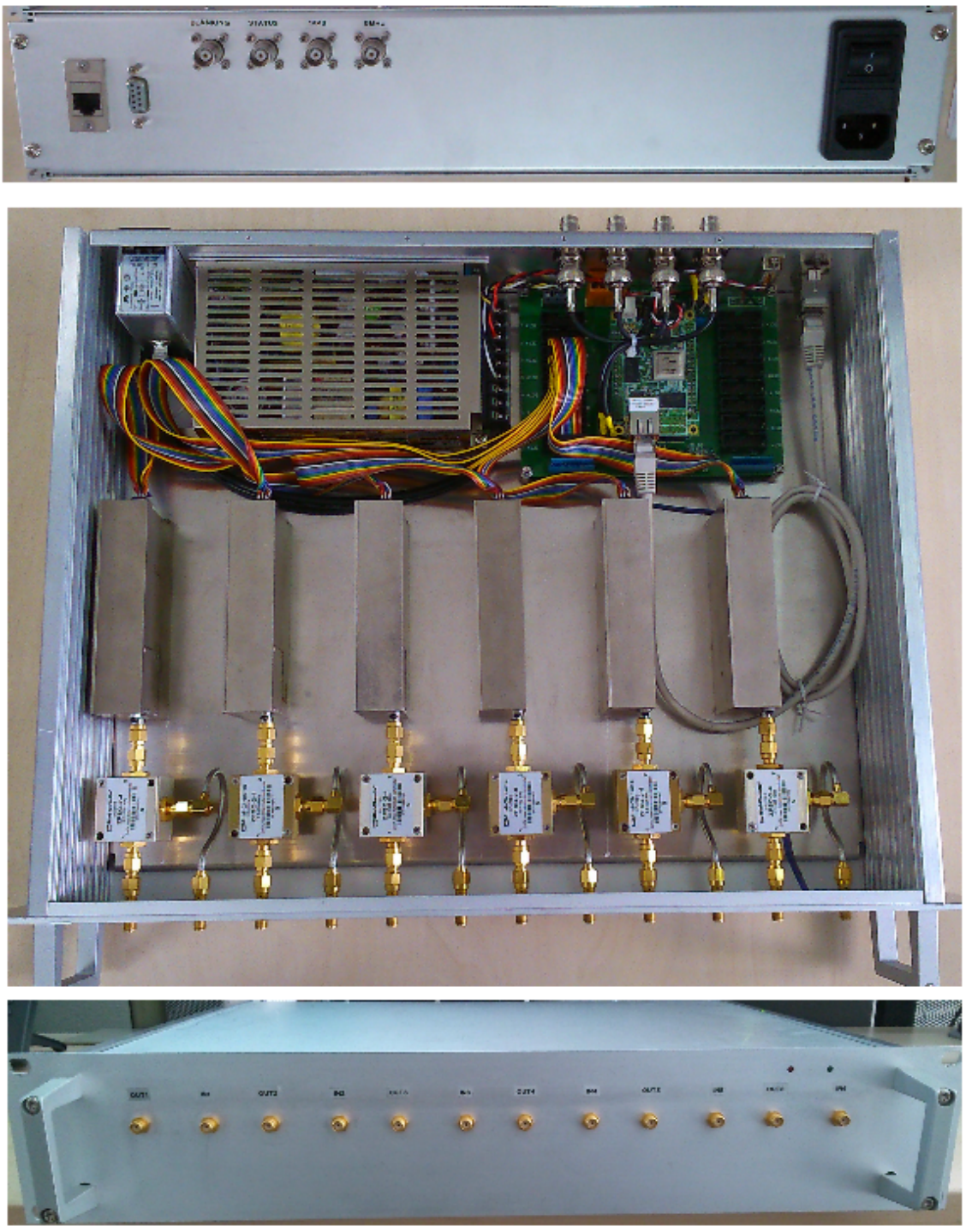

Figure 12: *View of the continuum backend once assembled.*

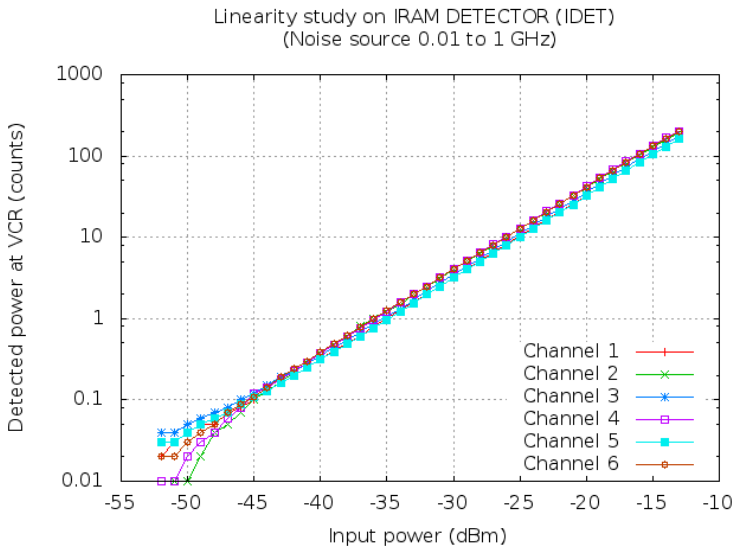

Figure 13: *Behaviour of the detector as function of input power. The linearity range extends for an interval of more than 30 dB.*