IRAM-COMP-071

Revision: 0
2010-06-03

Contact Author

Institut de RadioAstronomie Millimétrique

IF Processor Software

Owner Sebastien Blanchet

Keywords: IF processor, software

Approved by:	Date:	Signature:
A.Perrigouard	June 2010	

Change Record

REVISION	DATE	AUTHOR	SECTION/PAGE AFFECTED	REMARKS
0	2006-09-15	S. BLANCHET	See IRAM-COMP-034	Legacy document

Content

1 Introduction

In June 2006, IRAM has installed a new system called IF processor. This system is used to adapt the signal from the new generation receiver to the narrow-band correlator. This document is the documentation reference for IF processor software: installation, technical documentation, daily usage and troubleshooting.

You need also the hardware documentation of the IF Processor from Philippe Chavatte: *Dual LO2.pdf*

2 Requirements

2.1 Hardware requirement

To run the IF processor control software, you need at least:

- A standard PC, CPU 1GHz, RAM 512 Mo

In this situation, you can run the software in simulation mode.

Although, for a real usage you need also:

- A TPMC816 CAN controller from Tews Technologies. This card uses the PMC format so a PCI carrier is required.

2.2 Installed Hardware

2.2.1.1 Hardware

Ifproc computer:

- CPU: Athlon XP 2500,
- Mainboard: Asus Nforce2
- RAM: 1 Go
- Network device: 3Com 905c

2.2.1.2 BIOS configuration

This is the procedure to configure the BIOS on ifproc

Enter the BIOS
Menu Exit:
Load Setup Default
Now we will only quote the difference from the default values.

Main:

- Set date and Time
- Legacy Diskette A [None]
- Halt on [No Error]

Advanced

- Menu Advanced Bios Features
  - First Boot Device [LAN]
  - Second Boot Device [CDROM]
  - Third Boot Device [Disabled]

    - **APIC Mode [Disabled]** *(The TPMC816 card cannot work without this settings because we use an Adeos kernel)*
- Advanced Chipset Features
    - CPU External Freq [166 Mhz]
- Integreated Peripheral *(we disable several unused peripheral to free IRQs, to have better performances)*
    - USB Controllers [Disabled]
    - Onboard AC97 audio controllers [Disabled]
    - Onboard AC97 modem controllers [Disabled]
    - Onboard Lan (nvidia) [Disabled]
    - Floppy disk access controller [Disabled]
    - Onboard serial port 1 [Disabled]
    - Onboard serial port 2 [Disabled]
    - Onboard parallel port 1 [Disabled]
    - Onboard game port [Disabled]
    - Onboard Midi I/O [Disabled]

Exit and save changes

When the message appears: *Initialize MBA Ctrl+alt+B to configure,* press *Ctrl+alt+B*

- Boot Method: PXE
- Config Message: Enabled
- Message Timeout: 12 seconds
- Boot Failure Prompt: Wait for Timeout
- Boot Failure: Reboot

Type F10 to save

2.2.2 Ifproc-spare

The *ifproc-spare* computer is designed to replace *ifproc* in case of hardware failure.
To use *ifproc-spare* instead of *ifproc*, you have to unplug all the connectors from *ifproc* and then to plug them to *ifproc-spare*.

Warning: Never try to run simultaneously ifproc and ifproc-spare

2.2.2.1 Hardware

- CPU: Athlon XP 1600,
- Mainboard: Asus VIA KT133
- RAM: 512 Mo
- Network device: 3Com 905c

2.2.2.2 BIOS Configuration

Ifproc-spare has not the same hardware than ifproc, so the BIOS configuration is different.
Enter the BIOS
Menu Exit:
Load Setup Default
Now we will only quote the difference from the default values.

Main:

- set date and time
- Legacy Diskette A [None]

Advanced:

- I/O Device Configuration
    - Onboard serial port 1 [Disabled]
    - Onboard serial port 2 [Disabled]
    - Onboard parallel port 1 [Disabled]

    - Onboard Peripheral Resource Control
        - Onboard AC97 modem controllers [Disabled]
        - Onboard AC97 audio controllers [Disabled]
- PCI configuration
    - USB Function [Disabled]

Boot:

- Load onboard ATA bios [Disabled]

Save and Exit

When the message appears: *Initialize MBA Ctrl+alt+B to configure*, press *Ctrl+alt+B*

- Default Boot: Network
- Local Boot: Disabled
- Boot Method: PXE
- Config Message: Enabled
- Message Timeout: 12 seconds
- Boot Failure Prompt: Wait for Timeout
- Boot Failure: Reboot

Type F10 to save

2.3 Software requirements

Operating system

The software should run on any recent Linux (i386 or x86_64) with a 2.6.x kernel.

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.

Recommended software

The following programs are strongly recommended to modify easily the code.

Name	Description	Version	Download
Eclipse/CDT	C and C++ Integrated Development Environment (IDE) for the Eclipse platform.	>= 3.5	http://eclipse.org/cdt
doxygen	Automatic documentation system	>= 1.5.6	http://doxygen.org

All these software are very common, and you should find easily ready-to-install packages for your favorite Linux distribution.

2.4 Network environment

For the real exploitation, the IF Processor software will run on a disk-less computer. So an NFS server is required to export the filesystem. For better security, the main part of the filesystem will be exported as read-only.

3 Building

3.1.1 Extract code from the repository

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

Then use the Makefile to extract automatically the dependencies

```
$ cd IfProcessor
$ qmake
$ make get_deps
```

3.1.2 Build the TPMC816 driver

The TPMC816 driver is optional for the simulation. In this case, you can skip this section.

```
$ cd ~/develSVN/tpmc816
$ su -c "make install"
```

To create devices (and to load the driver)

```
$ su -c "./create_devices.sh"
```

The `create_devices.sh` script creates nodes in `/etc/udev/devices` .

3.1.3 Build the IfProcessor code

```
$ cd ~/develSVN/IfProcessor
$ ./build.sh
```

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

```
$ make doc
```

To install the software and its default settings in `/home/introot/ifproc`
Note: the 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 settings**, use

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

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

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

3.2 Configure environment

You should add `/home/introot/ifproc/bin` to your path.

If you wish to use CanLogger, you should also define `CAN_DB_FILE` as follow:

```
# Define environment variable for IfProc software
export DEVICE_NAME=ifproc
export CAN_DB_FILE=/home/introot/${DEVICE_NAME}/data/${DEVICE_NAME}.db
```

3.3 Initial tests

For this initial test, we run

1. The CanManagers to enable the Can Over IP protocol
2. The ifproc simulator
3. The python xml-rpc client: `SetFrequency`

First we remove the `/dev/tpmc816_*` devices, so the CanManager will run in simulation mode

```
$ su –c "rm –f /dev/tpmc816_*"
$ ifproc-init-can.sh
$ xterm -e ifproc-simul &
$ xterm -e ifproc-server &
```

Now run the `SetFrequency.py –f 1,100000,100001`

```
$ cd ~/develSVN/IfProcessor/scripts/xmlrpc/
$ ./SetFrequency.py -f 1,100000,100001
connect to http://localhost:1086
frequency = ['1,100000,100001']
#SetFrequency() rpc call takes  0.00645017623901  seconds
```

3.4 Start application automatically

Use `/home/introot/ifproc/bin/ifproc-init-device.sh` to start automatically all the required applications.

4 Internal programs

This section describes internal programs that drive the IF Processor. These programs are executed automatically, therefore normal users are not expected to run them.

4.1 CanManager

For a complete description see the document IRAM-COMP-057 *CanIp*

```
$ CanManager -h
CanManager is a bridge between the CAN bus and the CAN/IP protocol
Usage: CanManager [options]
Options:
  -d=name        CAN controller device name to use. If missing,
                      the application runs in simulation mode
  -p=N           Listen to UDP port N
```

```
  -l=N           Limit write speed base CAN messages.
                      'N' is in message/sec. Default value = 0 (no limit)

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


Example:
        CanManager -d=/dev/tpmc816_0 -p=2500 -l=20
```

For the IF Processor software, CanManager must listen on port udp/2501

```
CanManager –d=/dev/tpmc816_1 –p=2501 –l=50
```

4.2 ifproc-server

ifproc-server is a XML-RPC server to remotely control the IF processor.

> **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 a detailed introduction to XML RPC see http://en.wikipedia.org/wiki/XML-RPC

ifproc-server listens for XML-RPC calls on http://localhost:1086/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

4.2.1 Syntax

```
$ ifproc-server -h
IF Processor Server - XML-RPC Server
Listen on port 1086
Usage: ifproc-server [options]
Options:
  -v        Display version information
  -h, -?    Display help
```

4.2.2 API Description

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

```
$ cd ~/develSVN/IfProcessor/scripts/xmlrpc/
$ ./ListMethods.py
# Connect to http://localhost:1086
ifproc.selectIf
ifproc.setFrequency
ifproc.setOffsetPhase
system.listMethods
system.methodHelp
system.methodSignature
system.multicall
system.shutdown
```

```
$ ./Introspection.py
# Connect to http://localhost:1086
--------------------------------------------------------------------
Name       : ifproc.selectIf( int, int, int, int )
Return Type: int
Description: Set Frequency . Return code is always zero.
Syntax: ifproc.selectIf(int unitNum, int if1Freq, int if2Freq, int if1Polar,
int if2Polar)
    - unitNum must be in [1, 6]
    - if1Freq: 0 -> 2GHz, 1 -> 4GHz
    - if2Freq: 0 -> 2GHz, 1 -> 4GHz
    - if1Polar: 0 -> Vertical, 1 -> Horizontal
    - if2Polar: 0 -> Vertical, 1 -> Horizontal

Tip: use multicall to set all the frequencies at once on all the antennas

--------------------------------------------------------------------
Name       : ifproc.setFrequency( int, int, int )
Return Type: int
Description: Set Frequency . Return code is always zero.
Syntax: ifproc.setFrequency(int unitNum, int highFreq, int lowFreq)
    - unitNum must be in [1, 6]
    - highFreq is the (high) frequency in Hertz
    - lowFreq is the (low) frequency in Hertz

Tip: use multicall to set all the frequencies at once on all the antennas

--------------------------------------------------------------------
Name       : ifproc.setOffsetPhase( int, int, int, int, int )
Return Type: int
Description: Set offset and phase . Return code is always zero.
Syntax:  ifproc.setOffsetPhase(int unitNum, int highOffset, int highPhase, int
lowOffset, int lowPhase)
    - unitNum must be in [1, 6]
    - highOffset is the offset (lower) frequency in milliHertz [+/- 2.000.000]
    - highPhase is the initial (lower) phase in milliTurn [0, 999]
    - lowOffset is the offset (lower) frequency in milliHertz [+/- 2.000.000]
    - lowPhase is the initial (lower) phase in milliTurn [0, 999]

Tip: use multicall to set all the offsets and phases at once on all the
antennas

--------------------------------------------------------------------
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.

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

4.3 XML RPC client examples

4.3.1 Python examples

4.3.1.1 Minimal example

XML RPC is very easy to use in Python.

For example to call method `ifproc.setFrequency()` on the server, you need only the following lines.

```
import xmlrpclib
server = xmlrpclib.ServerProxy( "http://localhost:1086" )
server.ifproc.setFrequency(1, 100000, 100001)
```

4.3.1.2 Other Python examples

See directory *scripts/xmlrpc* for other XML-RPC python examples.

4.4 ifproc-server_legacy

This server supports the legacy protocol that was supported by the legacy server (ifpServer.py). It receives commands from bure1.iram.fr (as binary buffers) and sends back status (as a binary buffer). This server exists only to migrate easily existing client. Nevertheless all clients should migrate to the new XML-RPC server.

4.4.1 Syntax

```
$ ifproc-server_legacy -h
IF Processor ServerLegacy - Legacy Binary Server
Listen on port 1080
Usage: ifproc-server_legacy [options]
Options:
  -v        Display version information
  -h, -?    Display help
```

4.4.2 Data structures

To exchange data with bure1, we use two special data structures: IfpRequest_t and IfpStatus_t.

4.4.2.1 IfpRequest_t

This data structure is received from bure1

```
typedef enum{
   IFP_GET,
   IFP_SET_FREQ,,
   IFP_SET_OFFSET_PHASE,
   IFP_GET_TEMPERATURE,
   IFP_SELECT_IF,
   IFP_RESET_DDS,
   IFP_RESET_CPU,
   IFP_ACTION_LAST_INDEX,     /* use for iteration over enum */
} IfpAction_t;

typedef struct  {
   int frequency[2] ;/* main frequency in Hz (0-160e6) for U=0 and L=1 */
   int offset[2];      /*offset frequency in milliHz (-2^15, 2^15-1 for U=0 and
L=1 */
   int phase[2];    /*initial phase in milliturn (0-999) for U=0 and L=1 */
   int ifSelection; /*Select the IF bands to be processed, use
'createIfSelection to computer the value */

/*  IFSelection = 0x0q0r0s0t
   q=0 => IF1_Freq = 2GHz, q=1 => IF1_Freq= 4GHz
   r=0 => IF2_Freq = 2GHz, r=1 => IF2_Freq= 4GHz
   s=0 => IF1_Polarity = Vertical, s=1 => IF1_Polarity = Horizontal
   t=0 => IF2_Polarity = Vertical, t=1 => IF2_Polarity = Horizontal
*/
} IfpUnitSetting_t;

typedef struct {
   IfpAction_t action    /* Enum: IFP_GET, IFP_FREQ, … */;
   IfpSetting_t setting[6]; /* setting for the 6 IF Processor units */
} IfpCommand_t;
```

4.4.2.2 IfpStatus_t

This data structure is sent back to bure1

```
typedef struct  {
   float temperature;  /*degree Celcius, internal temperature*/
   uint32_t temperatureTimeStamp;      /*date when 'temperature'  was
                          measured, unit: in seconds since Epoch */
   float PS1_8A;    /* Volt, 1.8V analog power supply voltage */
   float PS1_8D;    /* Volt, 1.8V digital power supply voltage */
   float PS3_3D;    /* Volt, 3.3V digital power supply voltage */
   float PS5_0D;    /* Volt, 5.0V digital power supply voltage */
   float lock400M;     /* Volt, 400 MHz PLL lock indicator voltage */
   float lock4G;       /* Volt, 4 GHz PLL lock indicator voltage */
   float lock8_1G;     /* Volt, 8.1GHz PLL lock indicator voltage */
   float lock9_9G;     /* Volt, 9.9 GHz PLL lock indicator voltage */
   int canError;    /* number of CAN error on the bus */
   int timer7;          /* timer7 value */
   int unitOn;          /* 1 => unit is online,  0=> unit is off-line */
```

```
} IfpMonitor_t;

typedef struct  {
   IfpMonitor_t monitor[6]; /* monitor points for the 6 IF processor units */
} IfpStatus;
```

4.4.3 Protocol

This section explains the network protocol between bure1 and `ifproc-server_legacy`

`ifproc-server_legacy` receives on port TCP:1080 a `IfpRequest_t` buffer. For this section, we assume that *ifpServer.py* has stored this request in the *req* variable.

According to the *action* value, the server processes the request.

If *action* is equal to:

- **GET**: it gets the status buffer
- **SET_FREQ**: it sets the initial frequency of the 6 IF processor units, upper and lower bands. The *req.settings[*].frequency[*]* fields are used.
- **SET_OFFSET_PHASE**: it sets the frequency offset and initial phases for the 6 IF processor units, upper and lower bands. The fields *req.settings[*].offset[*]* and *req.settings[*].phase[*]* are used.
- **GET_TEMPERATURE**: it gets the internal temperature for the 6 IF processor units. This command is very CPU consuming for the IF processor microcontroller, so it **must never** be requested during the observations. When *status[*].temperature* and *status[*].temperatureTimeStamp* are updated.
- **SELECT_IF:** it selects the IF bands to be processed for the 6 IF processor units.
- **RESET_DDS:** it initializes the DDS (Direct Digital Synthesizer) for the 6 IF processor units.
- **RESET_CPU:** it resets the CPU of the 6 IF processor units.

Sending the appropriate CAN control messages to the IF Processor hardware does all the operations.

When the server has completed the request processing, it has to return an IfpStatus buffer to bure1.

The server issues the CAN monitoring messages immediately after the CAN control messages, but it does not wait that the monitoring value are available. Instead it sends the current Status values. **Therefore the status buffer reflects the IF processor status at the previous request time.**

Note: Temperature reading is very time consuming, so a *GET_TEMPERATURE* request is required to monitor the temperature, otherwise we do not update it. Therefore temperatureTimeStamp contains the updated date (in seconds since Epoch).

4.4.4 Performances

From a client on bure1, the response time of ifproc-server_legacy is less than 5 milliseconds. So we have very comfortable performances.

4.5 ifproc simulator

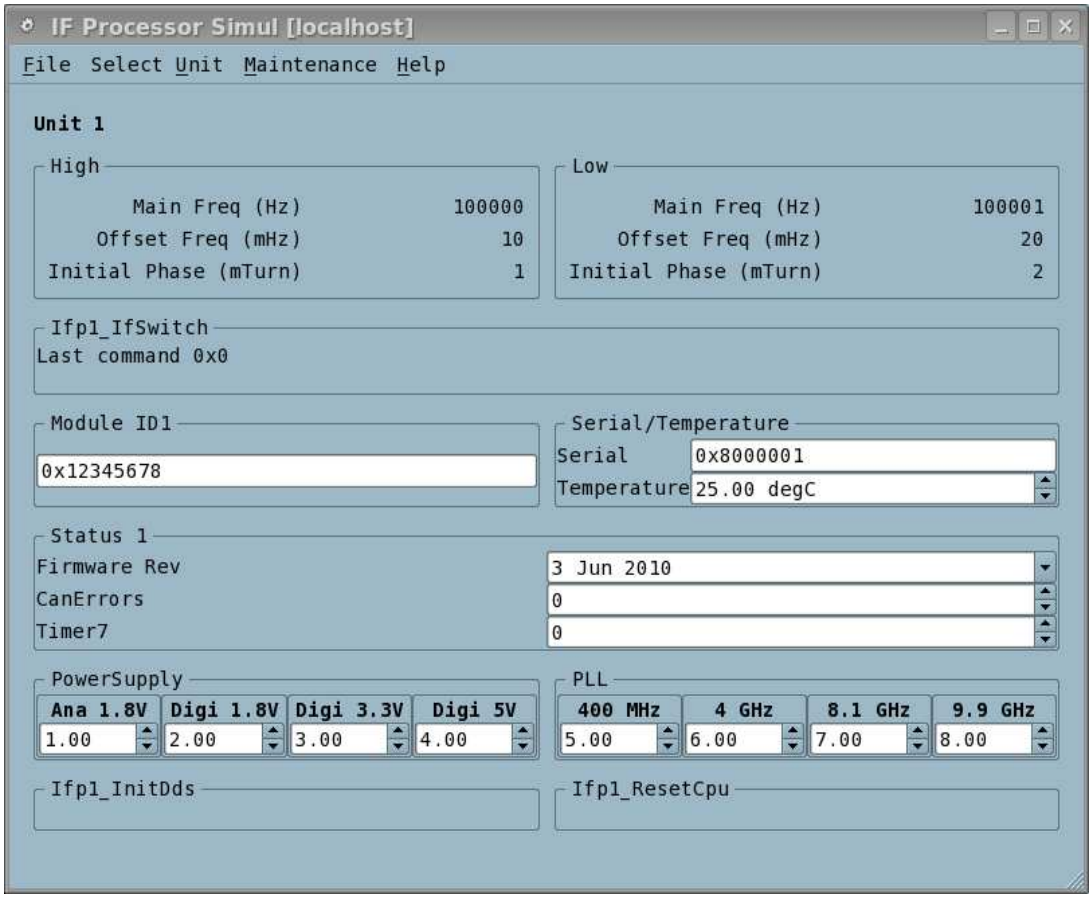
The goal of this program is to simulate the IF processor.

4.5.1 Syntax

```
IF Processor Simul - ifproc simulator
Usage: ifproc-simul [options]
```

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

4.5.2 Screenshot



5 User programs

All the software are installed in `/home/introot/ifproc/bin`

5.1 ifproc-gui

ifproc-gui is a graphical software to send command to the IF Processor.

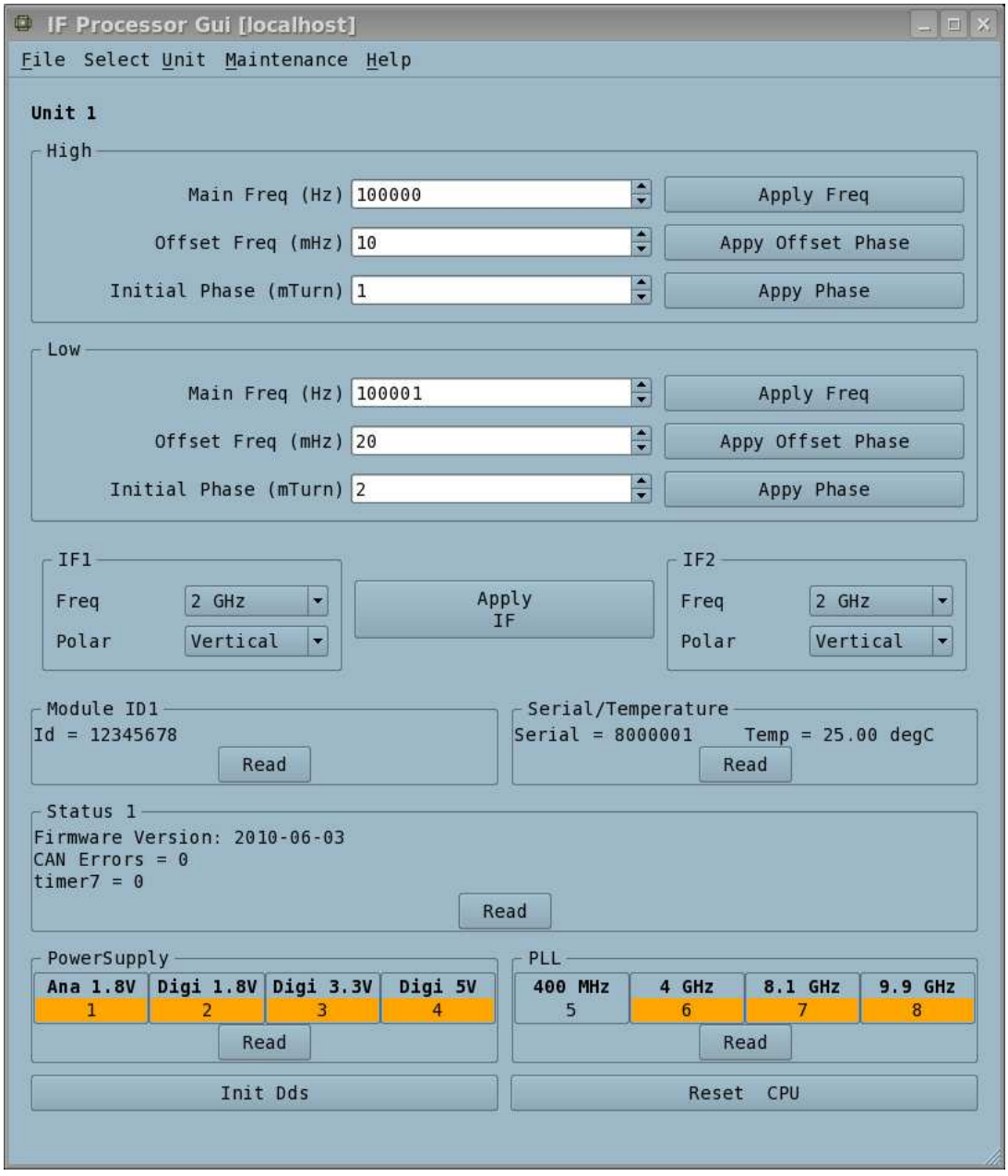
5.1.1 Syntax

```
IF Processor Gui - ifproc GUI
Usage: ifproc-gui [options]
Options:
  -u=1,2,3,4          List of unit numbers to control in [1, 6]
                      By default all the units are selected
  -v                  Display version information
```

```
  -h, -?              Display help

Example:
        ifproc-gui -u=1,2
```

5.1.2 Screenshot



5.1.3 Usage

Procedure to log in ifproc.iram.fr, and execute the program

```
$ ssh -Y backend@ifprocb.iram.fr
```

```
backend@ifprocb.iram.fr password:
$ ifproc-gui
```

5.2 IfpClientTest

ifpClientTest is a small C program to test **`ifproc-server_legacy`**

5.2.1 Compilation on LINUX

This is the installation step on LINUX, to build the software from the CVS repository.

Get the sources:

```
$export BUILDDIR=/home/blanchet/build
$ export CVSROOT=:pserver:blanchet@netsrv1.iram.fr:/CVS/PdB
$ mkdir  $BUILDDIR
$ cd $BUILDDIR
$ cvs login
Logging in to :pserver:blanchet@netsrv1.iram.fr:2401/CVS/PdB
CVS password:
$ cvs co –r FC4-branch LINUX
```

Build and install the software:

```
$ cd $BUILDIR/LINUX/utils
$ make –f  Makefile
$ cd ../ifpProcessor/src
$ make –f Makefile
```

The executable file is `../bin.Linux/ifpClientTest`

5.2.2 Syntax

```
ifpClientTest  [serverHostname]
```

5.2.3 Usage

If *serverHostname* is not specified, the program tries to connect to *ifp-gnb.iram.fr*, (the lab computer for the IF processor). s*erverHostname* is typically used to connect to another computer running an IF Processor Simulator instance).

Use Control-C to quit the program. It is interactive software, so you have to redirect the standard input if you want to use a file instead of using the keyboard to enter command.

Command (not case sensitive)	**Description**
?	Display online help
#	Comment (the other characters are ignored.
D	Display the command buffer (but do not send it)
U *n fU fL oU oL pU pL sc*	Define command buffer value, for unit *#n*, where - *fU* (resp, *fL*) are the upper (resp. lower) main frequency in Hz - *oU* (resp, *oL*) are the upper (resp. lower) offset frequency in mHz - *oP* (resp, *oP*) are the upper (resp. lower) initial phase in milliTurn - *sc* is the *selectIf* command
S *cmd*	Send the command buffer with action=*cmd* The possible values for command are: GET, SET_FREQ, SET_OFFSET_PHASE, GET_TEMPERATURE, SELECT_IF, RESET_DDS, RESET_CPU

	Note: if you issue an SET_OFFSET_PHASE command, the interpreter will wait for one second before accepting a new command. It is very useful to send a new SET_OFFSET_PHASE command every second.
Q	Exit the software.

For example:

```
$ ifpClientTest ifprocb
IfpClientTest
$Id: ifpClientTest.c,v 1.1.2.12 2006/09/13 14:32:53 blanchet Exp $
Connecting to ifprocb:1080 ... OK
Use '?' + Enter to display the Help
ifpClient > U 1 1 2 3 4 5 6 7
ifpClient > D
Command Buffer
action=GET
         freqUpper   freqLower   offU   offL  phaseU  phaseL   IfSelect
unit1 =          1           2      3      4       5       6    0x0007
unit2 =          0           0      0      0       0       0    0x0000
unit3 =          0           0      0      0       0       0    0x0000
unit4 =          0           0      0      0       0       0    0x0000
unit5 =          0           0      0      0       0       0    0x0000
unit6 =          0           0      0      0       0       0    0x0000

ifpClient > S SET_FREQ
Command Buffer
action=SET_FREQ
         freqUpper   freqLower   offU   offL  phaseU  phaseL   IfSelect
unit1 =          1           2      3      4       5       6    0x0007
unit2 =          0           0      0      0       0       0    0x0000
unit3 =          0           0      0      0       0       0    0x0000
unit4 =          0           0      0      0       0       0    0x0000
unit5 =          0           0      0      0       0       0    0x0000
unit6 =          0           0      0      0       0       0    0x0000

Status Buffer
_____________________________________________________________________
unit1 =  21.00 deg Celcius on Wed Sep 13 16:44:42 2006
        PS1_8A PS1_8D PS3_3D PS5_0D lk400M   lk4G lk8_1G lk9_9G
          1.81   1.81   3.32   5.01   1.02   2.01   3.01   4.01

unit2 =  22.00 deg Celcius on Wed Sep 13 16:44:42 2006
        PS1_8A PS1_8D PS3_3D PS5_0D lk400M   lk4G lk8_1G lk9_9G
          1.82   1.82   3.32   5.02   1.03   2.03   3.03   4.02

unit3 =  23.00 deg Celcius on Wed Sep 13 16:44:42 2006
        PS1_8A PS1_8D PS3_3D PS5_0D lk400M   lk4G lk8_1G lk9_9G
          1.83   1.83   3.34   5.04   1.04   2.03   3.03   4.04

unit4 =  24.00 deg Celcius on Wed Sep 13 16:44:42 2006
        PS1_8A PS1_8D PS3_3D PS5_0D lk400M   lk4G lk8_1G lk9_9G
          1.85   1.85   3.34   5.05   1.05   2.05   3.05   4.05

unit5 =  25.00 deg Celcius on Wed Sep 13 16:44:42 2006
        PS1_8A PS1_8D PS3_3D PS5_0D lk400M   lk4G lk8_1G lk9_9G
          1.86   1.86   3.36   5.05   1.06   2.05   3.05   4.05

unit6 =  26.00 deg Celcius on Wed Sep 13 16:44:42 2006
        PS1_8A PS1_8D PS3_3D PS5_0D lk400M   lk4G lk8_1G lk9_9G
          1.90   1.87   3.36   5.06   1.07   2.07   3.07   4.06
```

```
        canError   timer7   unitOn
_____________________________________________________________________
unit1 =        0       71        1
unit2 =        0       72        1
unit3 =        0       73        1
unit4 =        0       74        1
unit5 =        0       75        1
unit6 =        0       76        1

time delay: 1.9 ms
ifpClient > Q
```

5.2.4 Libraries

If you want to write your own software to dialog with ifproc-server_legacy, you need only two files:

- ifpClient.o: to handle easily the command and status buffer
- TcpClient.o to create and use the TCP socket

6 Troubleshooting

This section explains how to solve some troubles on ifprocb.

6.1 Modules

You can check if the device drivers are loaded:

```
$ /sbin/lsmod |grep tpmc
tpmc816drv             14216  2
```

If this module is missing, it means that there was a problem during the computer initialization. You should restart the computer with the 'reboot' command.

6.2 Processes

You can list the current processes and check that all the required processes are present:

Check for ifproc-server and ifproc-server_legacy processes

```
$ ps –edf
```

Look for the following processes:

```
oper      3123     1  0 14:43 ?        00:00:00 CanManager –d=/dev/tpmc816_1 -
p=2501
oper      3168  3165  0 14:44 ?        00:00:00 ifproc-server
oper      3170  3165  0 14:44 ?        00:00:00 ifproc-server_legacy
```

If one of these processes is missing, it means that it has crashed. These programs must never crash, otherwise there is a bug. If it happens, please contact the IRAM computer group. And then you can reboot the computer.