The local and remote monitoring software for the AMiBA telescope receivers

A report on my internship at
Academia Sinica Institute of Astronomy and Astrophysics,
Taipei, Taiwan.

EL HARIRI Madani
madani.el-hariri@insa-lyon.fr
Table of Contents

I - Abstract...........................................................................................................................4
II - Project description........................................................................................................5
   The Academia Sinica........................................................................................................5
   ASIAA ..............................................................................................................................5
   The AMiBA project.........................................................................................................7
      Overview.......................................................................................................................7
      Inside the receiver........................................................................................................8
      The receiver electronic box..........................................................................................9
      The monitoring computers.........................................................................................9
   The software.....................................................................................................................11
      Overview.......................................................................................................................11
      Understanding the basics............................................................................................11
      Glish.............................................................................................................................12
      The remote monitoring program...............................................................................12
      The local monitoring software ..................................................................................13
      Some room for improvement....................................................................................13
III- Solutions......................................................................................................................15
   Having a proper display.................................................................................................15
   Redesign of the client program's UI ...............................................................................15
   Upgrades to the existing client/server communication protocol..................................15
      Multiple receiver monitoring....................................................................................15
      Receiver list synchronization....................................................................................16
      Multiple clients..........................................................................................................16
      Monitoring the digital datas......................................................................................16
   Having a workable logfile............................................................................................17
   Temperature alert..........................................................................................................17
   Writing an user manual...............................................................................................17
IV - Conclusion..................................................................................................................18
Appendix I - COBE and WMAP sky surveys................................................................19
Appendix II - The AMiBA receiver..................................................................................20
Appendix III - The receiver electronic box.....................................................................22
Appendix IV - Annotated listing of the client program...................................................23
Appendix V - Annotated listing of the server program...................................................35
Appendix VI - Annotated listing of the glish script.........................................................45
Appendix VII – The AMiBA monitoring software user manual....................................55
I - Abstract

This document is a technical report on the training course I took between August 2nd and December 17th at the Academia Sinica Institute of Astronomy and Astrophysics in Taipei, Taiwan, as part of my 5th year in the GMD (mechanical engineering and development) department of INSA. The main subject is the monitoring software of the AMiBA telescope, which, under the supervision of Homin Jiang, I had to improve and be sure that it is ready for testing in Hawaii as soon as possible.
II - Project description

The Academia Sinica

The Academia Sinica was established in 1927 as a multidisciplinary research institution under control of the Nationalist government of the Republic of China. While still affiliated directly to the Presidential Office of R.O.C., Academia Sinica, now headed by Nobel laureate Yuan T. Lee, enjoys independence and autonomy in formulating its own research objectives. Its major tasks are to undertake in-depth academic research on various subjects in the sciences and humanities, ranging from Taiwan history to mathematics, and from molecular biology to social sciences. Altogether, there are 28 institutes, most of which have their headquarters in Academia Sinica's campus in Nankang, eastern Taipei.

Working at Academia Sinica is a large number of foreign scientists, from many different countries including India, the United States, France and Russia.

Academia Sinica is not the only research center in Taiwan, but it is by far the most important.

ASIAA (Academia Sinica Institute of Astronomy and Astrophysics)

The IAA is Taiwan's national institute for astronomy and astrophysics. It is part of Academia Sinica, but its headquarters have been moved to the National Taiwan University campus in central Taipei. The institute has an important office in Hilo, Hawaii as well. Hawaii is where all but one telescopes used by the IAA are located. The institute has an international crew of more than 90 people, of which about one third belong to the technical staff.

Since ASIAA is an institute in the Republic of China, its official language is
naturally Chinese (Mandarin). However, as ASIAA intends to become a leading research institute in the international community, the working language of ASIAA is English. In all seminars, colloquia, and meetings, English is the standard method of communication.

The astronomers working at IAA are specialised in radio-interferometry, that is, combining the power of several small receivers in the radio wavelengths domain to conduct extremely detailed sky surveys. The IAA is involved in many international projects in radioastronomy, like the SMArT (the SubMilimeter Array of Taiwan, working together with the United-States-operated SubMilimeter Array) or the CFHT (Canada-France Hawaii Telescope), but its greatest project to date is AMiBA.

In order to work more efficiently despite its small size, the IAA is part of a Ministry of Education Research Excellence Initiative on Cosmology and Particle Astrophysics (CosPA) that also includes theoretical and experimental research efforts in cosmology.
The AMiBA project

Overview

AMiBA, the Array for Microwave Background Anisotropy, is an ambitious project aimed at two goals. First, AMiBA will improve the results of such experiments as COBE (1992) and WMAP (2001) in surveying the cosmic microwave background radiation of the early universe (CMB, see Appendix I). The CMB is a rather new field of research in astrophysics and as such is the subject of an ever growing number of international experiments - around a dozen are slated to start operating in the next 3 years. The two most important competing experiments for AMiBA are SZA, led by a team of scientists from the University of Chicago, and AMI, operated by the University of Cambridge. As far as the CMB is concerned, AMiBA will undertake a survey of the sky in the 90 GHz +/-10 Hz bandwidth to measure slight fluctuations of intensity and of polarization of the CMB (thus giving a sharper, clearer picture of the 300,000-year old universe than COBE and WMAP did; see appendix I).

The second scientific goal of AMiBA is the survey of the early galactic clusters through the measurement of the Sunyaev-Zeldovich effect. This effect causes the black-body spectrum of the cosmic background radiations to be slightly shifted toward higher frequencies when going through galactic clusters. This shift is expected to be roughly one thousandth of a degreee. By measuring this slight shift, one would be able to determine the size and speed of the early galactic clusters, and having information about those very old, gigantic structures can help us understand how the early universe was and how it evolved until now.

When fully completed, the experiment will involve an array of 19 telescopes mounted on a 6-meter, fully steerable platform, but the first stage of the experiment is scheduled to start operating with 7, 0.6m-wide dishes before March 2005. For now, the whole project is in test phase, with 2 prototype antennas mounted at Mauna Loa in Hawaii.
Inside the receiver

Each dish of the AMiBA telescope is coupled to a receiver and to a set of correlators. The receiver’s function is to amplify the detected signals with minimum distortion and loss of fidelity, and provide these signals, into room temperature environment, for further signal processing and correlation. The correlation process, that is, comparing the signals coming form all the sources against each other, is what makes interferometry unique. Refer to Appendix II for a diagram showing what takes place inside the receiver.

Basically, the radio frequency (RF) signal collected by the feed-horn goes through two processes before being fed to the internal amplifier (since it works with a local oscillator, this part is referred to as the IF/LO module) and then to the correlator module:

- it is split into two signals with orthogonal polarizations, through the Orthogonal Mode Transducer (OMT)
- each signal is then amplified so that it can be correctly processed

Before RF pre-amplification, the signal is weak and must be prevented from being disturbed or even damaged by any kind of external noise, especially thermal noise and radiation coming form the outside. Thus, the corrugated feed-horn and the OMT must be placed in as cold and radiation-free an environment as possible.

This is achieved by placing them in a vacuum chamber, which is protected from radiation by a golden plate shield and maintained at a very low temperature by an helium-powered cryogenic pump.

As the temperature in the various parts of the vacuum chamber is a critical parameter when operating the telescope, one need to be able to measure it in every receiver at any time. For that purpose, we use 4 temperature sensors:

- one on the shield, measuring the temperature inside the vacuum chamber
- one on the cold head of the cryogenic pump
- one on each of the two pre-amplifiers

The data coming from these sensors and several other critical voltages in the correlator module are then output to the receiver electronics box.
The receiver electronic box
The receiver electronics box acts as an interface between the receiver and the computer. See Appendix III for a detailed view of its components. The key part is the dataset interface board, which is linked to the serial port of a computer through an RS485 fibre modem. The dataset interface board can convert the voltage values at its connectors pins and send them in ASCII format to the computer upon reception of the appropriate command. The dataset can understand two kind of command: requests for a certain value identified by its dataset address (monitoring commands), or control commands which trigger the sending of a voltage value to a specified dataset address. These control commands are used in the local monitoring program to turn on or off 5 modules of the IF/LO.

The monitoring computers
In order to be able to know the temperatures inside the receiver and the state of the IF/LO modules from virtually anywhere, a simple client/server architecture has been designed. The local monitoring computer in Hawaii
would act as the server, retrieving the values from the dataset board, displaying them on the screen, and sending them on request to any client computer. The client is most likely to be a computer used by an astronomer during a remote observation session at the IAA’s office in Taipei.

A receiver in testing phase, before being shipped to Hawaii
The software

Overview

The AMiBA software had been written by Michael Kesteven and Homin Jiang, based on a previous one developed for the Australia Telescope. The version I was given in August could communicate with the dataset interface board through the serial port, and had a very limited client-and-server architecture for the remote and local monitoring programs. However, it could only perform a limited number of tasks. A great number of improvements had to be made.

Understanding the basics

The first part of my job was to read a lot of documents. First, I had to get acquainted with the Linux operating system, and the C language. Computer programming was part of the courses at INSA so I had a good knowledge of the basic theory. However, C is a somewhat more complex language than Fortran or Delphi and getting familiar with it involved a lot of research on the Internet and at the IAA library. During that time, I had to get familiar with Linux as well. On this matter, I wish to express my thanks to the IAA's computer staff who made sure that my adaptation to Linux was as smooth as possible.

I also had to learn the GLISH scripting language and browse through the existing code to try and get a grasp of its purpose and its structure. Getting into someone else's code is never an easy task, but the code I was given, though it made use of many C libraries and specific functions, was well written and had a fair share of comments.

The next step towards my understanding of the program was to run the software in the lab, where a receiver was being assembled and tested. With this test I could see two obvious flaws that would have to be addressed prior to any other modification: the software couldn't get any real data from the receiver (I would almost only get null values when the receiver was at room temperature), and the client had a very raw interface.

Then, I looked more carefully at the code and did some extensive research on all the keywords, procedures and functions I didn't know (which is, most of them). I then drew a rough flowchart to clear everything up. I tried some small changes here and there in the code and tested the effect of each. Eventually, I came to know the software almost as well as if I had written it myself.
**Glish**

*Glish* is a freely available programming language, written by Vern Paxson and Chris Saltmarsh at Lawrence Berkeley Laboratory. It is now maintained and developed by Darrell Schiebel at the National Radio Astronomy Observatory in Charlottesville, Virginia.

*Glish* is used for building loosely-coupled distributed systems. "Loosely-coupled" means the programs in a *Glish* system communicate with one another at fairly low rates (perhaps a hundred times a second). "Distributed systems" means programs in the *Glish* system can run on different computers, communicating transparently over a network.

The main thrust of *Glish* is that individual programs in the system should be wholly modular, with no knowledge of other programs or data types that might exist in the system. *Glish* provides a uniform way for programs to communicate without knowing about one another. Communication occurs because the programs are written in terms of *events*, i.e. name/value pairs. In the usual case, programs receive an event, perform an action in response to that event, and possibly generate one or more new events associated with the response.

For instance, in the AMiBA software, the glish script (Grxmon.g) is used to generate a graphical interface and thus make a link between the user activity and the main local monitoring program (MonClientRmt.cc). Basically, the glish script is one loop, asking the main program (through the sending of a Glish event named “mon”) to poll the dataset board for the desired set of values, then waiting for a reply (a Glish event which name is “result” and value is an array holding the datas) from the main program and displaying the datas on the interface.

*The remote monitoring program (the client)*

(filename: amibadisp.c)

This program is written in C and is the simplest of the two, since it only retrieves the datas from the server and displays them on the screen. From the beginning, it was conceived as a multi-threaded application. The 3 threads are the *main thread*, the *keyboard input thread* and the *timer thread*, the latter two being linked to the main thread by two pipes. The *timer thread*, which period can be set by the user, is used to have the program request in a cyclic and steady way the new datas from the local monitoring computer. The
keyboard input thread manages, as one would guess, the user's key strokes and hands them on to the main thread for appropriate response. Eventually, the main thread handles the communication with the server and the display of the datas.

**The local monitoring software (including the server program)**
(filenames: Grxmon.g and MonClientRmt.cc)

The Grxmon.g glish script is a graphical interface to the MonClientRmt C++ program. It is to be run on a local monitoring computer in Hawaii. MonClientRmt sends commands to the Dataset interface and retrieves the datas from it through the serial port of the computer. The commands are sent to switch on or off the LO/IF modules. The datas are displayed then written to a logfile and eventually dispatched to any remote monitoring PC running the client software.

MonClientRmt is a two-threaded program. The main thread handles the communication with the glish script (reception and sending of events), while a detached thread handles the connection with the monitoring clients.

During its execution, Grxmon.g sends several Glish events to MonClientRmt. Among these, the most important ones are sent for each receiver being monitored: a cmd event which turns the modules of the LO/IF system on or off, then the two data-requesting events mon (for the voltages and temperatures) and Dimon (for the status of the LO/IF modules). The value associated to these holds the dataset adresses of all the monitored values. Upon reception of the mon or Dimon event, the program connects to the dataset interface and requests the datas, then stores them. The datas are then sent back to the user interface encapsulated as a glish event. They are then stored or displayed as needed. If the MonClientRmt program can't establish a communication with the dataset, it will reply to the mon or Dimon event by a fail event.

**Some room for improvement...**

The most important improvements that were to be made are listed here.

- investigate why the temperature data received from the dataset board were incoherent with the real temperature
- allow several remote monitoring clients to connect to one local monitoring server
- allow the monitoring of the digital datas as well as the analog.
- design a better user interface for the client program
- synchronize the list of available receivers between the client and the server, so that the client does not try to monitor a receiver that is off
- allow the local and remote monitoring of more than one receiver, and allow the user to easily switch between the receivers
- output a logfile that could be imported into a spreadsheet
- make sure that the user gets a warning when the temperature is too high
- write an user manual for both programs

The next section will cover those issues and how I addressed them.
III- Solutions

In this section I will go through the various improvements I added to the software. I will not go into technical details that would require a good knowledge of the C language. For these details, please refer to the commented listings in Appendix IV, V and VI.

Having a proper display

As I wrote earlier, during the first tests of the software, the client would only display 1 correct temperature out of 4. This prevented me from doing any serious testing on the data transmission between the client and the server. After a thorough investigation of the software and the hardware, the glitch turned out to come from a short circuit on the mainframe of the receiver electronic box.

Redesign of the client program' s UI (user interface)

In the previous version, the user interacted with the program by typing some commands (like 'help', 'list' etc.) followed by the Return key. In addition to that, the graphical interface was very basic and could easily be messed up if the user did something wrong. Communication with the software was made simpler, so that the user now only has to hit a single key to act on the program and a list of all the possible keystrokes is permanently displayed at the bottom of the screen. This is preferable for a casual user as it means that one doesn' t have to learn the commands in order to use the software. Also, switching between the various receivers has been made a lot easier: hitting any of the 9 digit keys will activate the monitoring of the corresponding receiver. As for the interface, I polished it up through the use of the the ncurses.h library.

Upgrades to the existing client/server communication protocol

Multiple receiver monitoring
Multiple receiver monitoring was achieved by allowing the client to send over the network a request command named “SHOW RX#” (with the pound sign being replaced by a digit) to the server. In reply to this command, the MonClientRmt program sends back the values of all the monitored datas for the requested receiver.

Receiver list synchronization
The first version of the program had the user select himself which receivers he thought was available for monitoring. This of course would need a phone call from Taipei to Hawaii and someone near the telescope to tell which receivers are on and which are off. Instead of this, I added a command to the client/server protocol to synchronize the list between the two computer: “listRx”, when sent to the server, causes it to send out an array holding the status of all the receivers. This command is issued by the client each time a new set of data is requested, so that the user is always aware of which receivers are being monitored by the server.

Multiple clients
The software has to allow several people to monitor the receivers from a remote location; this means letting the server software dispatch the datas to several clients.

The server program establishes communication with a client by 'listening' to a specified port of an internet connection through a given socket. The previous version of the program had the socket closed as soon as a connection with a client was created, thus preventing the server from accepting more than one client.

If we want to have several remote PC monitoring the datas at the same time, we have to leave a socket open, listening for incoming connection requests and creating one connection for each. Thus, I inserted in the server program an infinite loop that cycles through the listening socket, waiting for connection requests, and the already-established connections, waiting for new commands and treating them sequentially.

Monitoring the digital datas
Apart from the voltage values, the dataset interface board can provide the computer with some binary (ON/OFF) informations on 5 modules of the internal amplifier. Those binary datas can be accessed by just sending a request with the proper dataset address to the interface board. They are then displayed on the server’s and the client's interface.
Having a workable logfile
Having a logfile that can easily be fed to a spreadsheet such as OpenOffice Calc or Excel makes post processing of the information much easier. Fortunately, it is rather easy to create such file. A simple text file holding the numerical values separated by a space or a comma and with each row ending with a carriage return, can be imported to a spreadsheet. Outputting such a file was made possible by GLISH’s file writing procedures, which are fairly close to those of C.

Alert the user when the temperature is too high
The devices inside the vacuum chamber are designed to work at a temperature below 20K. The nominal temperature is 11K, but some variations can occur and, in any case, if the temperature inside the vacuum chamber exceeds 20K, the user has to be warned immediately. This is achieved by a visual alarm on the server side (the interface background blinks red) and an audio-visual alarm on the client side (the computer beeps and highlights the temperature value).

Writing an user manual

After having written a somewhat final version of the monitoring software, I had to make sure that it could be used by anyone, astronomer, or member of the technical staff. To do this, I had to ‘forget” everything I had learnt about Linux, C and the code I had written, and think like someone who would use the monitoring software for the first time. The manual had to describe all the procedures to install and operate both the server and the client software and also think of all the possible case of the software going wrong.

Writing the manual made me realize that I had to give some corrections to my software in order to make it more user-friendly. These changes included adding more explicit error messages, and overall an easier to use interface.
IV – Conclusion

After a few weeks of bugtesting, the software now performs all its functions smoothly.
The learning curve, albeit steep at the beginning, gradually became gentle although, with each improvement I added, I had to read over the documentation to learn about a specific subject (GUI management, networking, etc.). The end product, along with its user manual, can be run on any Unix system. The only requirement is to have Glish installed on the server system.
The client software may lack a more elaborated GUI, but one may wonder if that is really needed for that kind of professional software.
Appendix I

COBE and WMAP sky surveys, showing the temperature fluctuations of the cosmic background radiation. WMAP’s precision exceeds COBE’s and will in turn be excelled by AMIBA.
Appendix II

Schematics of an AMiBA receiver
Inside the receiver
Appendix III

The receiver electronic box

- F38 HEMT Bias card
- F36 relay & mux
- F30 temperature card
- F82 multiplexer
- F83 dataset interface board

Receiver Rx1-7

Temperature sensors

Local monitor PC (Hilo, HW) glish + MonClientRmt.cc

Remote monitor PC (Taipei) amibadisp.c

RS485 fibre modem

Ethernet
Appendix IV
Annotated listing of the client program (amibadisp.c)

/***
08-Mar-2004 Modify the code for remote Amiba Rx monitoring: Homin
17-Dec-2004 Modified the code: Madani
***/

#include "amibadisp.h"
#include <curses.h>

#define NO_OF_MON_PTS 16
#define NO_OF_DI_MON_PTS 2 // 2 bytes, = 16 bits = max. 16 values
#define NO_OF_RX 7

/*@ prototypes */
int clearErr (void);
int helpFunction (void);

#define ON 1
#define OFF 0

/*@ globals */
int timerDS[2];
int keybDS[2];
WINDOW *report;
WINDOW *keybd;
WINDOW *errwin;

int exit_flag = 0;
int timerCycle = 60; // initial refresh time, in seconds
int timerState = OFF;
int quiet_flag = 0;
int ncell;
int recCount;
int sockFD;
int availableRx[NO_OF_RX];

/******************************************************************************
int main (int argc, char **argv) {

    fd_set gmask;
    struct timeval timeout;
    int stat, kbd, nb;
    char msg[80];
    if (init_rxdisp() < 0) //brings on the user interface
        goto finished;
    }
timerState = ON;
selectedRx=rx[0];
refreshMon();
while (!exit_flag) {
    FD_ZERO (&gmask);
    FD_SET (timerDS[0], &gmask); //timerDS[0] is the read-end of the timerDS pipe
    FD_SET (keybDS[0], &gmask); //keybDS[0] is the read-end of the keybDS pipe
    timeout.tv_sec = 610; // higher than the greatest cycle (10 mins * 60 secs)
    timeout.tv_usec = 0;
    select (0, &gmask, NULL, NULL, &timeout);
    if (FD_ISSET (report, &gmask)) {
        if (fread (msg, 1, 80, report) <= 0) //check for new message
            continue;
        msg[80] = 0;
        win ERP (report, msg);
        msg[80] = 0;
    } else if (FD_ISSET (keybds, &gmask)) {
        if (fread (msg, 1, 80, keybds) <= 0) //check for new message
            continue;
        msg[80] = 0;
        keybd ERP (msg);
        msg[80] = 0;
    }
    } else if (FD_ISSET (errwin, &gmask)) {
        errwin ERP (msg);
        msg[80] = 0;
    }
    } while (1);
    finished: return 0;
}*/

/******************************************************************************/
stat = select (FD_SETSIZE, &mask, 0, 0, &timeout); // we wait until we have command or a timer tick (ie either timerDS[0] or keybdDS[0] is non-empty)
clearErr(); // clear the error window
if (stat == 0) {
    sprintf (msg, "timeout in main");
    reportMsg (msg);
}
else if (FD_ISSET (keybdDS[0], &mask) != 0) { // we have a keystroke
    nb = read (keybdDS[0], &kbd, 1); // reading the keyb input
    stat = processKbd (kbd); // process the command typed by the user
}
else if (FD_ISSET (timerDS[0], &mask) != 0) { // the timerDS pipe is ready to be read - we have a timer tick
    nb = read (timerDS[0], msg, 80);
    if (timerState == ON) {
        refreshMon (); // refresh the monitor ! ie display the data
    }
}
finished;
sleep(2); // for the operator to view the error message
endwin(); // exits the ncurse mode

int clearErr (void)
    /* clears the error window */
{
    char msg[80];
    sprintf (msg, " ");
    reportMsg (msg);
    return (0);
}

int helpFunction ()
    /* displays a list of useful commands */
{
    timerState = OFF;

    wmove (report, 1, 1);
    wclrtobot (report);

    waddstr (report, "\n RXMON HELP PAGE \n\n");
    waddstr (report, " c ........... set the monitor refresh rate\n");
    waddstr (report, " z ........... turns ON/OFF silent mode\n");
    waddstr (report, " q ........... quit\n");
    waddstr (report, " 1 - 8 ....... select a receiver \n");

    wstandout(report);
    waddstr (report, "\n\n enter Rx# to continue\n");
    wstandend(report);

    box(report, 0, 0);
    wnoutrefresh (report);
    dopupdate ();
}
int init_rxdisp(void)

    /* starts the terminal window (3 windows: report, errwin and keybd), connect the
       socket, then read the data about all the available receivers */

    {
        char msg[80];
        int stat;

        pthread_t rxThread;
        pthread_t kbdThread;

        initscr (); // initiates a terminal window (enables the ncurses library)
        noecho (); // don't echo the keyboard strokes
        cbreak (); // don't wait for a carriage return to process the keyboard strokes
        curs_set (0); // don't show the cursor

        report = newwin (18, 0, 0, 0);
        box (report, 0, 0);
        wnosrefresh (report);

        errwin = newwin (3, 0, 18, 0);
        box (errwin, 0, 0);
        wnosrefresh (errwin);

        keybd = newwin (0, 0, 21, 0);
        box (keybd, 0, 0);
        wnosrefresh (keybd);

        doupdate (); // updates the terminal

        stat = pthread_create (&rxThread, NULL, &timerCreate, NULL); // creates the timer
        thread then checks if this worked
        if (stat != 0) {
            sprintf (msg, "failed to create the timer thread\n");
            reportMsg (msg);
            return (-1);
        } 

        stat = pipe (timerDS); // creates the timer pipe which will allow data transfer between
        the timer thread and the main program
        if (stat != 0) {
            sprintf (msg, "failed to create the timer pipe\n");
            reportMsg (msg);
            return (-2);
        } 

        stat = pthread_create (&kbdThread, NULL, &getInput, NULL); // creates the keybd thread
        then checks if this worked
        if (stat != 0) {
            sprintf (msg, "failed to create the keyboard thread\n");
            reportMsg (msg);
            return (-3);
        } 

        stat = pipe (keybdDS); // creates the keybd pipe
        if (stat != 0) {
            sprintf (msg, "failed to create the keybd pipe\n");
            reportMsg (msg);
            return (-4);
        }
printf (msg, "Connecting to the monitor socket \n");
reportMsg (msg);

stat = sockConnect (); // creates a socket and connects to the host
if (stat != 0) {
    printf (msg, "failed to create the desk socket \n");
    reportMsg (msg);
    return (-5);
}

printf (msg, "found the server...retrieving rx list");
reportMsg (msg);

stat = listRx (); // retrieve a list of the available Rx from the server
if (stat<0) {
    printf (msg, "server not responding");
    reportMsg (msg);
    return(-6);
}

refreshKbd (); // display a list of the possible key strokes in the lower window

// sleep(3); /* display for 3 sec */
installCell();

printf (msg, "initialization complete...monitoring");
reportMsg (msg);

return (0);

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

void refreshKbd(void)

    /** depending on which receivers are present, display a list of the possible key
     * inputs */
{
    char msg[80], temp[80];
    int i;

    sprintf(msg, "press 
");
    for (i=0; i<NO_OF_RX; i++) {
        if (rx[i].state == ON) {
            sprintf(temp, "%i ", i+1);
            strcat (msg, temp);
        }
    }

    sprintf(temp, "%h c z q ");
    strcat (msg, temp);
    mwaddstr(keybd, 1, (80-strlen(msg))/2, msg); // print in the keyb window
    box (keybd, 0, 0);
    wnoutrefresh(keybd);
    douptate ();
}

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

int installCell (void)

    /** opens the formatting cell file, puts all the cells information (data name, unit,
     * etc.) into the cell variable */

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

void *getInput (void *arg)

    /** writes the user's keyboard entry in the keyb pipe */
int ch;

while(1) {
    ch = getch(); // wait for the user to press a key
    write (keybdDS[1], &ch, 1); //write it in the write-end of the keybdDS pipe
}

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

int listRx (void)

    /* asks to the server a list of all the available Rx */

{
    char str[80];
    char msg[80];
    int ln, i;
    int ok=0;
    int stat=0;
    struct timeval timeout;
    fd_set mask;

    timeout.tv_sec = 3;
    timeout.tv_usec = 0;

    timerState = OFF;

    wmove (report, 1, 1);
    waddstr (report, "Available receivers : \n\n");
    wclrbot (report);
    wnooutrefresh (report);
    dupdate (); // preparing to list all the available receivers in the report window

    nrx = 0;

    sprintf (msg, "listRx");
    ln = strlen (msg);
    if (send (sockFD, &msg, ln, 0) != -1) { //sends the "listRx" command to the server
        FD_ZERO (&mask);
        FD_SET (sockFD, &mask);

        if (select (FD_SETSIZE, &mask, 0, 0, &timeout) == -1) {
            sprintf (msg, "timeout waiting for the list");
            reportMsg (msg);
            signalErr();
            stat = -1;
            goto done;
        }

        if (recv (sockFD, availableRx, NO_OF_RX*sizeof(int), 0)<1) { //receive 7*4 bytes
            sprintf (msg, "server not responding");
            reportMsg (msg);
            signalErr();
            stat = -1;
            goto done;
        }
    }

    for (i=0;i<NO_OF_RX;i++) { //builds the availableRx array and displays a list of receivers available for monitoring

printf(msg, "Rx%d", i+1);
strcpy(rx[i].name, msg);
if (availableRx[i]==0) {
    rx[i].state=OFF;
} else { //if the Rx is available, print it on the screen
    rx[i].state=ON;
    printf(msg, "%s\n", rx[i].name);
    waddstr(report, msg);
    ok=1;
}
if (ok == 0) {
    printf(msg, "No available receiver");
    reportMsg (msg);
}
refreshKbd(); // refresh the rx list at the bottom of the screen
timerState = ON;
}
else {
    stat = -1;
    printf(msg, "unable to send listRx command");
    reportMsg (msg);
}
done:

box(report, 0, 0);
wnostrefresh (report);
doupdate ();
return (stat);

yyyyMMdd

int processKbd (int ch)
/* act on the user's keyboard entry */
{
    char msg[80];
    if ((ch>48)&&(ch<58)) { // we have a digit
        selectFunction(ch);
        return(0);
    }
    switch(ch) {
    case 104: // 104 is ASCII code of "h"
        helpFunction();
        return(0);
        break;
    case 99: // "c"
        setTimer();
        return(0);
        break;
case 113: // "q"
    sprintf (msg, "quitting...");
    reportMsg (msg);
    exit_flag = 1;
    return(0);
    break;

case 122: // "z"
    if (quiet_flag == 0) {
        sprintf (msg, "Silent mode ON");
        reportMsg (msg);
        quiet_flag = 1;
    }
    else if (quiet_flag == 1) {
        sprintf (msg, "Silent mode OFF");
        reportMsg (msg);
        quiet_flag = 0;
    }
    return(0);
    break;
    
default:
    sprintf (msg, "invalid key");
    reportMsg (msg);
    return(-1);
    break;
}

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

int setTimer (void)

    /* reads a value from the keyb then sets the timer cycle to the corresponding value */
{
    fd_set gmask;
    struct timeval timeout;
    int stat;
    int nb, c;
    char msg[80];

    FD_ZERO (&gmask);
    FD_SET (keyboard[0], &gmask);
    timeout.tv_sec = 10;
    timeout.tv_usec = 0;
    c=0;
    sprintf (msg, "press: 1 for 10 secs, 2 for 1 min, 3 for 2 mins, 4 for 10 mins");
    reportMsg (msg);
    stat = select (FD_SETSIZE, &gmask, 0, 0, &timeout); // wait for a keystroke

    if (FD_ISSET (keyboard[0], &gmask) != 0) {
        nb = read (keyboard[0], &c, 1);
        if ((c>48)&(c<53)) {
            switch(c) {
            case 49:
                timerCycle = 10;
                break;
            case 50:
                timerCycle = 60;
                break;
            }
break;
    case 51:
    timerCycle = 120;
break;
    case 52:
    timerCycle = 600;
break;
    default:
    sprintf(msg, "invalid key");
    reportMsg (msg);
    return(-1);
break;
    }
    sprintf (msg, "polling period set to %i secs", timerCycle);
    reportMsg (msg);
    return(0);
    } else return(-1);
}
/**
  * refreshMon (void)
  *
  * read the data points and write them to the screen */
{
    char lmsg[80];
    int row, col, ln, stat;
    struct timeval timeout;
    fd_set mask;
    #define BUFF_SIZE 1024
    int nb;
    char temp[32];
    char omgs[80];
    float x[NO_OF_MON_PTS];
    unsigned char DIx[NO_OF_DI_MON_PTS*8];
    unsigned int i;
    char t;
    timeout.tv_sec = 3;
    timeout.tv_usec = 0; // timeout is 3 secs
    wnmove(report, 1, 1);
    wclrtobot(report);
    wnoutrefresh(report);
doupdate(); // clears the report window before anything

listRx();

sprintf(lmsg, "SHOW %s", selectedRx.name);
ln = strlen(lmsg);
stat = send(sockFD, lmsg, ln, 0); // sends the msg containing "SHOW RX?" through the socket

for (;;) {
    FD_ZERO(&mask);
    FD_SET(sockFD, &mask);
    st

    at = select(FD_SETSIZE, &mask, 0, 0, &timeout);

    if (stat == -1) { // nothing to read
        sprintf(lmsg, "timeout ");
        reportMsg(lmsg);
        signalErr();
        goto done;
    }

    memset(x, 0, NO_OF_MON_PTS*sizeof(*x)); // initializes x and DIx, just in case...
    memset(DIx, 0, MAX_CELL_PER_RX*sizeof(*DIx));

    nb = recv(sockFD, x, NO_OF_MON_PTS*sizeof(*x), 0); // gets a message which length is NO_OF_MON_PTS*sizeof(float) in x, rather than MAX_CELL_PER_RX*sizeof(x)

    if (nb < 0) { // error
        reportMsg("ABoss socket has died");
        return (-1);
    }

    sprintf(temp, "Monitoring %s every %i seconds", selectedRx.name, timerCycle);
    mvwaddstr(report, 1, 1, temp);

    for (ncell=0; ncell<MAX_CELL_PER_RX; ncell++) {
        if (ncell<4) {
            sprintf(&((cell[ncell].value[0]), "%s%s %6.1f%s", selectedRx.name, cell[ncell].name, x [ncell], cell[ncell].unit );
        }
        else {
            sprintf(&(cell[ncell].value[0]), "%s%s %7.2f%s", selectedRx.name, cell[ncell].name, x [ncell], cell[ncell].unit );
        }

        wmove(report,cell[ncell].row+3,cell[ncell].col+1); // row+1 col+1
        if ((ncell==1)&&(x[ncell]>20)) {
            wstandout(report);
        }

        sprintf(lmsg, "Rx temperature over 20K");
        reportMsg(lmsg);
        signalErr();
    }
    waddstr(report,&(cell[ncell].value[0]));
    wstandend(report);

    box(report, 0, 0);
    wnoutrefresh(report);
    doupdate();
nb = recv (sockFD, DIX, 16, 0);
if (nb < 0) {
    reportMsg ("A boss socket has died");
    return (-1);
}

waddstr (report, "\n\n");
getyx (report, row, col);

strcpy (temp, " IF1: ");
dispDI (temp, 0, DIX, row, col);
strcpy (temp, " IF2: ");
dispDI (temp, 2, DIX, row, col+24);
strcpy (temp, " SWITCH:");
dispDI (temp, 4, DIX, row, col+48);
strcpy (temp, " DOUBLER: ");
dispDI (temp, 6, DIX, row+2, col);
strcpy (temp, " 42G Amp: ");
dispDI (temp, 8, DIX, row+2, col+24);
goto done;
}

done:;
recCount++;

printf (lmsg, "\n\n record: %d", recCount);

waddstr (report, lmsg);

box (report, 0, 0);
wnoutrefresh (report);
douupdate ();
return (0);
}

/*************************************************************/
void dispDI (char *msg, int bitpos, char *DIX, int row, int col)

    /* displays the digital datas */
{
    char temp[16];

    mvwaddstr (report, row, col, msg);

    if (DIX[bitpos]) {
        strcpy (temp, " power ON ");
        mvwaddstr (report, row, col+9, temp);
    
        wstandend (report);
    }
    else {
        strcpy (temp, " power OFF \n");
        mvwaddstr (report, row, col+9, temp);
    }

    if (DIX[bitpos+1]) {
        strcpy (temp, " ");
        mvwaddstr (report, row+1, col, temp);
        strcpy (temp, " alarm ON \n");
    }
mvwaddstr(report, row+1, col+9, temp);
wstandend(report);
}
else {
    strcpy(temp, "          alarm OFF \n");
    mvwaddstr(report, row+1, col, temp);
}
box (report, 0, 0);
wnoutrefresh(report);
doupdate();
}

int reportMsg (char *msg)
/* clears the error window (errwin); writes the error message in it*/
{...}

int selectFunction (int i)
/* switch monitoring to another receiver */
{
    int stat;
    char msg[80];
    i = i - 48;

    if (rx[i-1].state != ON) {
        sprintf(msg, "receiver %d not available", i);
        reportMsg(msg);
        return(-1);
    }

    selectedRx = rx[i-1];
    sprintf(msg, "receiver #%d selected", i);
    reportMsg(msg);

    return(0);
}

int signalErr (void)
/* alerts the user: causes the interface to flash and the computer speaker to beep*/
{...}

void *timerCreate (void *arg)
/* the timer thread: each timerCycle seconds, writes something on the write-end of the timerDS pipe */
{
    char msg[80];
for (;;) {
    sleep (timerCycle);

    sprintf (msg, "cycle");
    write (timerDS[1], msg, strlen(msg));
}

/*============================================================================*/
int checkLoBossServer (char *host, int *port)
/* reads the configuration file rxmon.reg
   for the server's IP address and port
   and writes them to the 'host' and 'port' variables */
[...]
/*============================================================================*/
int sockConnect (){
/*
 * establish the connection to the specified host/port.
 * return 0 if OK, -1 on error. The socket ID is lodged in
 * common memory.
 * This is a connect-with-timeout.
 */
[...]
Appendix V
Annotated listing of the server program (MonClientRmt.cc)

/* Data Set test application - ASIAA model
-----------------------------------------------*/

#include <Glish/Client.h>
[...]
#include <sys/socket.h>

#define BLOCKSIZE 128
#define PKTSZ 8

#define NO_OF_MONPTS 16
#define NO_OF_DI_MONPTS 2
#define NO_OF_CMDPTS 4
#define COM_BAUD 38400

#define MAX_RX_NO 7

/***************************************************************************/
#define ON 1
#define OFF 0
#define PORT 5100
/***************************************************************************/

int open_port (int *fd, int port, int baud);
[...]
void *ds_lan (void *arg);

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

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

float val;
unsigned char dataLSB, dataMSB;

char errmsg[80], message[4], response[2];
int ds_addr, cmd, cmdh, cmdl;

static unsigned char CMD = 192;
static char MON = 64;
static char SYN = 22;
static char ESC = 27;
static char ACK = 6;
static char NAK = 21;
static char BEL = 7;

static char p_syn = 49;
static char p_esc = 48;
static char p_ack = 50;
static char p_bel = 51;
static char p_nak = 52;

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

int COM_port;
int ds_fd;

int cycle;

int exit_flag = 0;
int timerCycle = 5;
int timerState = OFF;

/**************************For remote server socket operation
-----------------------------------*/

float* x = new float[NO_OF_MON_PTS * MAX_RX_NO];
float* xx = new float[NO_OF_MON_PTS];

unsigned char* DIX = new unsigned char[NO_OF_DI_MON_PTS * 8 * MAX_RX_NO];
unsigned char* DIXx = new unsigned char[NO_OF_DI_MON_PTS * 8];

int rxno;
int availableRx[MAX_RX_NO];

int main (int argc, char ** argv)
{
    unsigned char null_byte = 0x00;
    unsigned char dsa;                     /* DataSet address, 0...31 */
    unsigned char pntid;                   /* MUX point number 0...63 */

    int fst_analog_level;

    char ch, i_string[80];
    int i, j, k, cadl;
    unsigned int madl, madh;
    int muxCount;//temporary use*/
    int m, FilterBegin;
    char device[12];

    fd_set gmask;

    char msg[80];
    char kbd[80];

    int nb, ncell;
    /***************************************************************************/
    Client c(argc, argv);
    GlishEvent* e;
    Value* v, RetVal;
    char head[80];
    char body[80];

    int stat;

    char *ptr;
    int* intptr = new int[NO_OF_MON_PTS + 2];
    int* cmdptr = new int[NO_OF_CMD_PTS * 2 + 2];
    int* comprtr = new int; /* com port */

    int port;
    int addr;
    int monpt;

    if (MonClientInit() == -1) { //Initialization
        printf("initialize failed\n");
        exit(-1);
    }
    /***************************************************************************/

    while (e = c.NextEvent()) { //If we receive an glish event
        strcpy(head, "echo");
        sprintf(body, "event : %s\n", e->Name());
        c.PostEvent (head, body);
if (!strcmp (e->Name(), "listRx")) { //we have a 'listRx' event

    v = e->value;
    intptr = v->IntPtr();
    printf("available Rxs:\n"); //printout on the terminal window

    for (i=0;i<MAX_RX_NO;i++) { //read the list of available receivers
        availableRx[i]=intptr[i];
        printf("Rx%d = \%d\n",i, intptr[i]);
        if (intptr[i]==0) {
            for (j=0; j<NO_OF_MON_PTS; j++) {
                x[(i-1)*NO_OF_MON_PTS+j]=0;
            }
        }
    }
}

else if (!strcmp (e->Name(), "sleep")) { //sleep for a few seconds
    v = e->value;
    intptr = v->IntPtr();
    printf("sleeping %isecs...\n", *intptr);
    sleep(*intptr);
    strcpy (head, "result");
    c.PostEvent (head,"result");
}

else if (!strcmp (e->Name(), "mon")) { //get datas
    v = e->value;
    intptr = v->IntPtr();
    rxno = *intptr++; // retrieve the number of the RX
    stat = ClientReadDs(intptr,x);
    strcpy (head, "result");
    if (stat == 0) {
        for (i=0; i<NO_OF_MON_PTS;i++) {
            xx[i]=x[(rxno-1)*NO_OF_MON_PTS+i];
        }
        Value * RetVal = new Value(xx,NO_OF_MON_PTS);
        c.PostEvent (head, RetVal); //report back to the glish GUI
    }
}

else if (!strcmp (e->Name(), "Dimon")) { // get digital datas
    v = e->value;
    intptr = v->IntPtr();
    rxno=*intptr++;
    stat = ClientReadDsDI(intptr,DIX);
    strcpy (head, "Diresult");
    if (stat == 0) {
        for (i=0; i<NO_OF_DI_MON_PTS*8;i++) {
            DIXxx[i]=DIX[(rxno-1)*NO_OF_DI_MON_PTS*8+i];
        }
        Value * RetVal = new Value(DIXxx,NO_OF_DI_MON_PTS*8);
        c.PostEvent (head, RetVal);
    }
    else {
        c.PostEvent (head,"DI FAIL");
    }
}

else if (!strcmp (e->Name(), "cmd")) { //send a command to a LO/IF module
v = e->value;
cmdptr = v->IntPtr();
stat = ClientCmdDs(cmdptr);

strcpy (head, "CmdOK");
c.PostEvent (head, "OK");
}
else if (!strcmp (e->Name(), "OpenCOM")) { //open COM port
...
}
else if (!strcmp (e->Name(), "CloseCOM")) { //close COM port
close(ds_fd);
}
else if (!strcmp (e->Name(), "quit")) break; //the end
}
finished;
exit(0);
}

/*****************************************
int ClientReadDs (int* intptr, float *x)

/* request a set of values from the dataset interface board */
{
int COM_port;
int ds_addr;
int MonPt;

unsigned char null_byte = 0x00;
unsigned char ds_a;                  /* DataSet address, 0...31 */
unsigned char pntid;                 /* MUX point number 0...63 */

int fst_analog_level;
char ch, message[4], response[2], i_string[80];
int i, j, k, cadl, cmd, cmdh, cmdl;
int madl, madh;
int m;

char device[12];
/* socket variables */
int thrd;
int port;
int stat;

/* set up the COM port */

COM_port = *intptr;
ds_addr = *(intptr+1);

/* Send a command in order to clear the power-up warning flag */

message[0] = 96;
message[1] = 0;
message[2] = 0;
i = ds_cmd (ds_fd, ds_addr, message);
if (i == 0) {
    printf ("\n dataset on-line = OK\n");
}
else if (i == -10) {
    printf("\n dataset on-line - power-up warning flag cleared\n");
} else {
    printf("**** ERR on check - DS_CMD status = %d\n", i);
}

/* Loop, requesting a monitor point, getting the result, and logging it */

for (MonPt=0;MonPt<NO_OF_MON_PTS;MonPt++) { // loop forever
    madl = *(intptr+2+MonPt);
    madh = 0;
    if (madl > 255) {
        madh = 1;
        madl -= 256;
    }

    message[0] = madh;
    message[1] = madl;
    message[2] = 0;
    message[3] = 0;
    i = ds_mon(ds_fd, ds_addr, message, response);
    if (i >= 0) {
        j = response[1] & 0xFF;
        k = response[0] & 0xFF;
        dataMSB = j;
        dataLSB = k;
        fst_analog_level = dataMSB * 256 + dataLSB - 2048;
        x[(rxno-1)*NO_OF_MON_PTS+MonPt] = (5.0 * fst_analog_level) / 4096.0; // store the
        value into an array
        x[(rxno-1)*NO_OF_MON_PTS+MonPt] = x[(rxno-1)*NO_OF_MON_PTS+MonPt] * 2.0; // scale
        the value
        if (MonPt<4) {
            x[(rxno-1)*NO_OF_MON_PTS+MonPt] = x[(rxno-1)*NO_OF_MON_PTS+MonPt] * 100.0; //
            rxno-1 because in C arrays start at 0 not 1.
        }
    }
}
else {  
    printf("\n ** ERR - code %d\n", i);
    usleep(500000);
    printf("Reset the serial line \n");
    stat = close(ds_fd); // close the serial line
    if (stat != 0) {
        perror("Unable to close port - ");
        printf("requested port : %d = %s\n", port, device);
        exit (1);
    }
    stat = open_port(&ds_fd, COM_port, COM_BAUD);
    if (stat < 0) {
        printf("fatal error - unable to open serial line \n");
        exit (-1);
    }
    else {
        printf("Serial line reset \n");
    }
    return(-1); // stop polling the dataset, we can assume it is down
}

usleep (10000); /* have to delay a while, otherwise error occur */
int ClientReadDsDI (intptr, unsigned char *x)
{
    int COM_port;
    int ds_addr;
    int MonPt;

    unsigned char null_byte = 0x00;
    unsigned char dsad;                     /* DataSet address, 0...31 */
    unsigned char pntid;                     /* MUX point number 0...63 */

    char ch, message[4], response[2], i_string[80];
    int i, j, k, cadl, cmd, cmdh, cmdl;
    int madl, madh;
    int m;
    char device[12];

    /* socket variables */
    int thrd;
    int port;
    int stat;

    /* set up the COM port */
    COM_port = intptr;
    ds_addr = *(intptr+1);

    /* Send a command in order to clear the power-up warning flag */
    message[0] = 96;
    message[1] = 0;
    message[2] = 0;
    i = ds_cmd (ds_fd, ds_addr, message);
    if (i == 0) {
        printf ("\n DI dataset on-line - OK\n"");
    } else if (i == -10) {
        printf ("\n DI dataset on-line - power-up warning flag cleared\n");
    } else {
        printf ("**** DI ERR on check - DS_CMD status = %d\n", i);
    }

    /* Loop, requesting a monitor point, getting the result, and logging it */
    for (MonPt=0;MonPt<NO_OF_DI_MON_PTS;MonPt++) {
        /* loop forever */
        madl = *(intptr+2+MonPt);
        madh = 0;
        if (madl > 255) {
            madh = 1;
            madl -= 256;
        }

        message[0] = madh;
        message[1] = madl;
        message[2] = 0;
        message[3] = 0;
        i = ds_mon (ds_fd, ds_addr, message, response );
    }
if (i >= 0) {
    j = response[1] & 0xFF;
    k = response[0] & 0xFF;

dataMSB = j;
dataLSB = k;
printf("dataLSB = 0x%x\n", dataLSB);
x[MonPt*8+(rxno-1)*NO_OF_DI_MONPTS*8] = dataLSB & 0x01;  // stores the bits into an array
x[MonPt*8+1+(rxno-1)*NO_OF_DI_MONPTS*8] = (dataLSB & 0x02)>>1;
x[MonPt*8+2+(rxno-1)*NO_OF_DI_MONPTS*8] = (dataLSB & 0x04)>>2;
x[MonPt*8+3+(rxno-1)*NO_OF_DI_MONPTS*8] = (dataLSB & 0x08)>>3;
x[MonPt*8+4+(rxno-1)*NO_OF_DI_MONPTS*8] = (dataLSB & 0x10)>>4;
x[MonPt*8+5+(rxno-1)*NO_OF_DI_MONPTS*8] = (dataLSB & 0x20)>>5;
x[MonPt*8+6+(rxno-1)*NO_OF_DI_MONPTS*8] = (dataLSB & 0x40)>>6;
x[MonPt*8+7+(rxno-1)*NO_OF_DI_MONPTS*8] = (dataLSB & 0x80)>>7;
}
else {
    printf ("\n ** ERR - code %d\n", i);
    /* reset the serial line */
    usleep(50000);
    printf ("Reset the serial line \n");
    stat = close (ds_fd);
    if (stat != 0) {
        perror("Unable to close port - ");
        printf ("requested port : %d = %s\n", port, device);
        exit (1);
    }
    stat = open_port (&ds_fd, COM_port, COM_BAUD);
    if (stat < 0) {
        printf ("fatal error - unable to open serial line \n");
        exit (-1);
    } else {
        printf ("serial line reset \n");
    }
    return(-1);  // stop polling the dataset, we can assume it is down
}
usleep (10000); /* have to delay a while, otherwise error occur */
    /* the monitor loop */
return (0);
}

/*****************************************************/
int ClientCmdDs (int* intptr) {

    /* sends a command request to the dataset interface board */

    int COM_port;
    int ds_addr;
    int CmdPt;
    int stat;

    COM_port = *intptr;
    ds_addr = *(intptr+1);
    for (CmdPt=0; CmdPt<NO_OF_CMDPTS; CmdPt++) {

Dout(ds_addr,*(intptr+2+CmdPt),*(intptr+3+CmdPt));
}
return(0);
}/**
* Dout(int DSaddr,int IPort,char cntl)
{
    message[0]=IPort;
    message[1]=0;
    message[2]=cntl;
    message[3]=0;
    return ds_cmd(ds_fd,DSaddr,message);
}
/***
* Module containing the dataset routines.
* Two main subroutines, along with a supporting cast.
* DS_CMD -- send a control command
* DS_MON -- request monitor data
*/

int ds_cmd(int fd, int address, char *message )
{/* assemble a message holding a control command, then send it to the dataset */

int ds_mon (int fd, int address, char *message, char *response )
{/* assemble a message requesting a dataset value, then send it to the dataset, wait for a response and return that response to the main program */

/***************************/
void *ds_lan (void *arg)
{/* This thread creates the server socket; allowing the remote user to monitor the dataset */
int listener;
struct sockaddr_in Sock;
struct sockaddr_in hostSock;
struct hostent host;
struct hostent *hostptr;

char hostname[256];

int retval, addrlen, ib, i, more, nb, nsock, TESTPORT, verbose, on, optlen, nbyt, stat,
FD, fdmax, newfd;
char msg[BLOCKSIZE+2];
char lmsg[128];
char* temp;
unsigned int c;

fd_set master;
fd_set read_fds;
struct sockaddr_in remoteaddr;
pthread_detach ( pthread_self ( ) );

TESTPORT = ( int ) arg;
verbose = 1;

/* create the socket; bind; then listen */

printf ( "
 Create the socket 
" );

if ( ( listener = socket ( AF_INET, SOCK_STREAM, 0 ) ) < 0 ) { // open a socket
    printf ( " error creating the socket\n" );
    exit ( 1 );
}

printf ( " socket ID = %x\n", listener );

on = 1;
optlen = sizeof ( on );
stat = setsockopt ( listener, // set the options for the socket
    SOL_SOCKET,
    SO_REUSEADDR,
    ( const char * ) &on,
    optlen );

if ( stat < 0 ) {
    printf ( " error: socketopt\" );
    printf ( " socketopt error %X", errno );
    return ( NULL );
}

addrlen = sizeof ( Sock );
Sock.sin_family = AF_INET;
Sock.sin_port = htons ( TESTPORT );
Sock.sin_addr.s_addr = htonl ( INADDR_ANY );

if ( bind ( listener, ( struct sockaddr * ) &Sock, addrlen ) < 0 ) { // start a socket for
    'listening' to the network
    perror ( " binding error\" );
    exit ( 1 );
}

printf ( " socket port # %d\n", ntohs ( Sock.sin_port ) );

if ( listen ( listener, 0 ) < 0 ) {
printf("Listen error \n");
exit(1);
}
nsoc = 0;
FD_SET(listener, &master);
fmax = listener;

for (;;) { // start listening for incoming connections
  read_fds = master;
  if (select(fmax+1, &read_fds, NULL, NULL, NULL) == -1) {
    printf("select error");
    exit(1);
  }
  for(i=0; i<=fmax; i++) {
    if (FD_ISSET(i, &read_fds)) {
      if (i == listener) { // handle new connections
        addrlen = sizeof(remoteaddr);
        if ((newfd = accept(listener, (struct sockaddr *)&remoteaddr, (socklen_t *)&addrlen)) == -1) {
          printf("accept error");
        } else {
          FD_SET(newfd, &master);
          if (newfd > fmax) { // keep track of the maximum fd
            fmax = newfd;
          }
          printf("new connection from \%s on socket \%d\n", inet_ntoa(remoteaddr.sin_addr), newfd);
        }
      } else { // handle data from a client
        if ((nbv = recv(i, msg, BLOCKSIZE, 0) <= 0) || (strstr(msg, "quit") != NULL)) {
          // error or connection closed by client
          printf("socket \%d hung up\n", i);
          close(i);
          FD_CLR(i, &master);
        } else { // we got some data from the client
          if (strstr(msg, "SHOW") != NULL) {
            if ((temp=strstr(msg, "Rx"))!=NULL) { // we have a 'SHOW RX' command
              char* temp;
              if ((c>48) && (c<58)) { // is it a figure ?
                temp = temp + 2;
                c = c - 48;
                c --; // because arrays start at 0 with C
              } else if (strstr(msg, "listRx") != NULL) { // we have a 'listRx' command
                stat = send(i, availableRx, MAX_RX_NO*sizeof(int), 0); // send the list of available receivers
                stat = send(i, Dl+NO_OF_DI_MON_PTS*8*c, 16, 0); // send the digital data
              }
            }
          } else if (strstr(msg, "SHOW") != NULL) {
            // we have a 'SHOW RX' command
            if ((temp=strstr(msg, "Rx"))!=NULL) { // we have a 'SHOW RX' command
              char* temp;
              if ((c>48) && (c<58)) { // is it a figure ?
                temp = temp + 2;
                c = c - 48;
                c --; // because arrays start at 0 with C
              } else if (strstr(msg, "listRx") != NULL) { // we have a 'listRx' command
                stat = send(i, availableRx, MAX_RX_NO*sizeof(int), 0); // send the list of available receivers
                stat = send(i, Dl+NO_OF_DI_MON_PTS*8*c, 16, 0); // send the digital data
              }
            }
          }
        }
      }
    }
  }
}
Appendix VI
Annotated listing of the glish script used to generate and control the graphical interface of the server (Grxmon.g)

```glish
#                    G r x c n s l . g
# Generates the GUI for, and controls execution of,
# the AMIBA receiver monitoring and control
# Adapted from amiba.g
# remote monitor port "MonClientRmt" April 2004
# homin

include 'glishtk.g'
data set := client ('MonClientRmt')

# Check whether DISPLAY is defined.
if (! has_field(environ, 'DISPLAY')) {
    print 'DISPLAY environment variable is not set, bailing out!'
    exit
} else {
    print 'Using DISPLAY', environ.DISPLAY
}

# Clients
#######
ctrl := F
intr := F
busy := F

local logger, sendto
mstat := T
port := 2 #COM port number

# Work variables
#------------
wrk := [=]

# change to homin#
wrk.host := 'homin'
wrk.port0 := 5500 #16040#
wrk.port1 := 5505 #16041#

wrk.palette := [
enabled = "#000000 Enabled widgets", # Tk default, foreground
disabled = "#a3a3a3 Disabled widgets", # Tk default, foreground
background = "#d9d9d9 Normal background", # Tk default, background
selected = "#c3c3c3 Selected text", # Tk default, background
active = "#ececece Active widgets", # Tk default, background
select = "#b03060 Check button dimple", # Tk hardset
normal = "#000000 Unexceptional messages", # Text
reminder = "#ffccc Reminders", # Text, &c.
warning = "#ff9933 Warnings", # Text
]```

include 'glishtk.g'
data set := client ('MonClientRmt')

# Check whether DISPLAY is defined.
if (! has_field(environ, 'DISPLAY')) {
    print 'DISPLAY environment variable is not set, bailing out!'
    exit
} else {
    print 'Using DISPLAY', environ.DISPLAY
}

# Clients
#######
ctrl := F
intr := F
busy := F

local logger, sendto
mstat := T
port := 2 #COM port number

# Work variables
#------------
wrk := [=]

# change to homin#
wrk.host := 'homin'
wrk.port0 := 5500 #16040#
wrk.port1 := 5505 #16041#

wrk.palette := [
enabled = "#000000 Enabled widgets", # Tk default, foreground
disabled = "#a3a3a3 Disabled widgets", # Tk default, foreground
background = "#d9d9d9 Normal background", # Tk default, background
selected = "#c3c3c3 Selected text", # Tk default, background
active = "#ececece Active widgets", # Tk default, background
select = "#b03060 Check button dimple", # Tk hardset
normal = "#000000 Unexceptional messages", # Text
reminder = "#ffccc Reminders", # Text, &c.
warning = "#ff9933 Warnings", # Text
error = "\#ff3333 Serious errors", # Text, &c.
sendto = "\#009999 Send event logging", # Text
recv = "\#336600 Recv event logging", # Text
sched = "\#00ccff Sched record mode", # Text
debugp = "\#9900cc Debug parameter updates", # Text
debugw = "\#9900cc Debug widget updates", # Text
action = "\#eeec Action buttons", # Background
rflas h = "\#ddbbcc Widget read flash", # Background
wflas h = "\#ccddbb Widget write flash", # Background
help = "\#000099 Help / information", # Foreground
stop = "\#dd0000 Stop and Exit buttons", # Foreground
current = "\#336600 Current action status widget", # Foreground
offline = "\#ff9933 Subsystem status: offline/disabled", # Background
online = "\#66ff33 Subsystem status: online/ok", # Background
suberror = "\#ff3333 Subsystem status: error", # Background
unknown = "\#00ccff Subsystem status: unknown" # Background

FCC_status := ' '
TRK_status := ' '
PMAC_status := ' '
last_command := 'None'
last_rx_sel := 'None'
you_have_control := T
selected_rx := 0
platform_selected := '1'
bellON := T
ComOpened := F
OnScreen := 1 # show Rx1 by default
RxNo := 7
global rx:= [ ]

rxname := "amiba"

DELAY := 10
rx[1].dsa := 11
rx[2].dsa := 12
rx[3].dsa := 13
rx[4].dsa := 14
rx[5].dsa := 15
rx[6].dsa := 16
rx[7].dsa := 17

for(i in 1:RxNo)

rx[i].is_available:=F #turn all the receivers OFF by default
rx[i].is_present:=F

rx[i].bias.byte :=0
rx[i].temp1.addr := 304
rx[i].temp2.addr := 305
rx[i].temp3.addr := 306
rx[i].temp4.addr := 307
rx[i].vd1.addr := 328
rx[i].vd2.addr := 329
rx[i].vd3.addr := 330
rx[i].vd4.addr := 331
rx[i].id1.addr := 336
rx[i].id2.addr := 337
rx[1].id3.addr := 338
rx[1].id4.addr := 339
rx[1].id5.addr := 320
rx[1].id6.addr := 321
rx[1].id7.addr := 322
rx[1].id8.addr := 323
rx[1].biasDO.addr := 98
rx[1].biasDO.bit0 := 0  # bit 0 of 98 for bias 0 control
rx[1].biasDO.bit1 := 1  # bit 1 of 98 for bias 0 control
rx[1].biasDO.bit2 := 2  # bit 2 of 98 for bias 0 control
rx[1].biasDO.bit3 := 3  # bit 3 of 98 for bias 0 control
rx[1].loDO.addr := 99
rx[1].loDO.byte := 255  # default all off
rx[1].loDO.IF1 := 1
rx[1].loDO.IF2 := 2
rx[1].loDO.Switcher := 4
rx[1].loDO.Doubler := 8
rx[1].loDO.amp42G := 16
rx[1].loDI.addr := 101
rx[1].lo2DI.addr := 102

rx[1].is_available := T  # turn the first receiver ON

# Setup for logging window
opf_w.version := '1.0'
opf_w.tz_name := 'Taiwan time (GMT+0800)'  # Local timezone name.
opf_w.tz_offset := 8*3600  # Local timezone offset, sec.
opf_w.longitude := '150:00:00.00'  # Observatory longitude.
opf_w.latitude := '-35:00:00.00'  # Observatory latitude.
opf_w.altitude := 0.0  # Observatory altitude.
opf_w.azparm := [-270, 270, 15]  # Azimuth limits (deg), and drive rate
                               # (deg/min).
opf_w.elparm := [15, 90, 15]  # Elevation limits and drive rate.

# Logging window matters
opf_w.log.file := './data/opf.log'  # Log file name.
opf_w.log.dayno := 0  # Log unix day number.
opf_w.log.index := 0  # Log message buffer index.
opf_w.log.count := 0  # Log message buffer count.
opf_w.log.flush := 20  # Log message buffer limit.
opf_w.log.config.black := '#000000'  # Text colour, unexceptional messages.
opf_w.log.config.blue := '#000080'  # Text colour, send event logging.
opf_w.log.config.green := '#004000'  # Text colour, warnings.
opf_w.log.config.cyan := '#006767'  # Text colour, sched record mode.
opf_w.log.config.red := '#ff0000'  # Text colour, recv event logging.
opf_w.log.config.purple := '#800080'  # Text colour, debugging messages.
opf_w.log.config.purple := '#ff0000'  # Text colour, serious errors.
opf_w.log.scroll := F  # enable auto-scrolling

# Set up Tk widgets:
tk_hold()  # don't display until it is all complete
main_fr := frame(title='Local Monitoring Program (Grxmon)', side='left') # start to draw the GUI

main_fr_left := frame(main_fr, side='top') # left frame
main_fr_right := frame(main_fr, side='top') # right frame

action_fr := frame(main_fr_left, side='left', relief='raised')
LO_fr := frame(main_fr_left, side='top', relief='raised', pady=2)
setup_fr := frame(main_fr_left, side='top', relief='raised', pady=2)
display_fr := frame(main_fr_left, side='top', relief='raised', pady=2)
log_fr := frame(main_fr_right, side='top', relief='raised')

period_fr := frame(display_fr, side='left') # display the 'change rate' buttons
period_label := label(period_fr, 'Monitoring rate')
period_button1 := button(period_fr, 'Fast (10 sec)', type='radio')
period_button2 := button(period_fr, 'Slow (60 sec)', type='radio')

monitoring_rate := 60
period_button2->state(T)

whenever period_button1->press do {
    monitoring_rate := 10
}
whenever period_button2->press do {
    monitoring_rate := 60
}

# Action frame

start_b := button(action_fr, text='START', type='radio', width=16, height=2) # 'start' button
stop_b := button(action_fr, text='STOP', type='radio', width=16, height=2) # 'stop' button

stop_b->state(T) # initialize

exit_b := button(action_fr, 'Exit', width=16, height=2, background='red') # 'exit' button

whenever start_b->press do { # the main loop

    # Open serial port
    dataset->OpenCOM([port])
    await dataset->ComOpen
    logger(spaste('COM port', $value), 'dataset')
    if( as_string($value) == 'FAIL' ) exit
    else ComOpened := T

    while ( start_b->state()==T)
        for(i in 1:RxNo) { # build the list of available receivers
            if (rx[i].is_available==T) {
                availableRx[i] := 1
            } else {
                availableRx[i] := 0
            }
        }
    dataset->listRx(availableRx) # then send it to the server program
    for(i in 1:RxNo) {

if (rx[i].is_available == T) {
    datadir := as_string(spaste("./data/Rx", i))
    Rdim := as_string(spaste("/Rx", i))
    Rdim := as_string(spaste(Rdim, Rdim))
    Rdim := as_string(spaste(Rdim, "_")))
    rx[i].filename := as_string(spaste(spaste(">> ./data", Rdim), CurrentDate))
    rx[i].fd := open(rx[i].filename, "DF")  # open the logfile for digital data
    rx[i].filename := as_string(spaste(spaste(spaste(">> ./data", Rdim), CurrentDate), ".csv"))
    rx[i].fd := open(rx[i].filename)  # open the logfile for analog values
    dataset->cmd([port, rx[i].dsa, rx[i].biasDO.addr, rx[i].bias.byte, rx[i].loDO.addr, rx[i].loDO.byte])  # send out control command to dataset through the server program
    await dataset->CmdOK
    cmd := spaste('port ', port, ' addr ', rx[i].dsa)
    print " cmd : ", cmd
    cmd := spaste('Query Rx', i, ' ', cmd)
    logger(cmd, 'dataset')
    dataset->mon([i, port, rx[i].dsa, rx[i].temp1.addr, rx[i].temp2.addr, rx[i].temp3.addr, rx[i].temp4.addr, rx[i].vd1.addr, rx[i].vd2.addr, rx[i].vd3.addr, rx[i].vd4.addr, rx[i].id1.addr, rx[i].id2.addr, rx[i].id3.addr, rx[i].id4.addr, rx[i].vg1.addr, rx[i].vg2.addr, rx[i].vg3.addr, rx[i].vg4.addr])  # send out monitoring command to dataset through the monitoring program
    # added the component 'i' at the beginning of the event value
    await dataset->result  # monitoring fail
    if (as_string($value) == 'FAIL') {
        cmd:=spaste('Rx',i,’ does not respond'
        logger(cmd, 'dataset')
    } else {
    ****************************reporting ****************************
    if (OnScreen==i) {
        Arc_name_e->text(rx[i].filename)  # show archive
        l_data_l->text(spaste(spaste("Rx", i), ' at ', shell("date +%b%d%T")))  # report the values to the GUI
        l_temp1_e->delete ('start', 'end')
        l_temp2_e->delete ('start', 'end')
        l_temp3_e->delete ('start', 'end')
        l_temp4_e->delete ('start', 'end')
        l_vd1_e->delete ('start', 'end')
        l_vd2_e->delete ('start', 'end')
        l_vd3_e->delete ('start', 'end')
        l_vd4_e->delete ('start', 'end')
        l_id1_e->delete ('start', 'end')
        l_id2_e->delete ('start', 'end')
        l_id3_e->delete ('start', 'end')
        l_id4_e->delete ('start', 'end')
        l_vg1_e->delete ('start', 'end')
        l_vg2_e->delete ('start', 'end')
        l_vg3_e->delete ('start', 'end')
        l_vg4_e->delete ('start', 'end')
    $value::print.precision:=3
if ($value[2] > 20) { // we have an abnormal temperature
    l_temp2_e->background('red')
l_temp_f->background('red')
    rx[i].T2alarm->background('red')
    logger(spaste('Temperature too high in receiver #', i))
} else {
    l_temp2_e->background('lightgrey')
l_temp_f->background('lightgrey')
    rx[i].T2alarm->background('lightgrey')
}

l_temp3_e->insert (as_string($value[2]))
l_temp4_e->insert (as_string($value[4]))

from one in Glish

l_vd1_e->insert (as_string($value[5])) // array start
l_vd2_e->insert (as_string($value[6]))
l_vd3_e->insert (as_string($value[7]))
l_vd4_e->insert (as_string($value[8]))
l_id1_e->insert (as_string($value[9])) // array start
from one in Glish

l_id2_e->insert (as_string($value[10]))
l_id3_e->insert (as_string($value[11]))
l_id4_e->insert (as_string($value[12]))
l_vg1_e->insert (as_string($value[13])) // array start
from one in Glish

l_vg2_e->insert (as_string($value[14]))
l_vg3_e->insert (as_string($value[15]))
l_vg4_e->insert (as_string($value[16]))

} // CurrentTime := shell("date +%b%d%T") // archive the datas to the logfile

fprintf(rx[i].fd,"%s",CurrentTime)
for(j in 1:16) {
    if (j<5) {
        fprintf(rx[i].fd,'%5.1f,',$value[j])
    } else {
        fprintf(rx[i].fd,'%7.3f,',$value[j])
    }
    fprintf(rx[i].fd,'\n')
}

dataSet->D-mon([i, port, rx[i].dsa,rx[i].l01DI.addr,rx[i].l02DI.addr] )
#request digital datas

wait

dataSet->D-res // monitoring fail
if( as_string($value) == 'FAIL') {
    logger('D-res', 'dataset')
} else {
    if (OnScreen == i) { // report the values to the GUI
        if($value[1] ) IF1_power->background ('green')
        else IF1_power->background ('red')
        if($value[2] ) IF1_alarm->background ('green')
        else IF1_alarm->background ('red')
        if($value[3] ) IF2_power->background ('green')
        else IF2_power->background ('red')
        if($value[4] ) IF2_alarm->background ('green')
        else IF2_alarm->background ('red')
if($value[5] ) Switch_power->background ('green')
else Switch_power->background ('red')
if($value[6] ) Switch_alarm->background ('green')
else Switch_alarm->background ('red')

if($value[7] ) Doubler_power->background ('green')
else Doubler_power->background ('red')
if($value[8] ) Doubler_alarm->background ('green')
else Doubler_alarm->background ('red')

if($value[9] ) Amp42G_alarm->background ('green')
else Amp42G_alarm->background ('red')
if($value[10] ) Amp42G_power->background ('green')
else Amp42G_power->background ('red')

CurrentTime :=shell("date +%b%d%T")
printf(rx[i].fdDI,'\n%s: ' ,CurrentTime) # archive the datas to the logfile

printf(rx[i].fdDI, "%4i", $value[1:10])
if (rx[i].T2alarm->background()=='red'){
  printf(rx[i].fdDI,' %s','Temp2 Alarm')
}

if (i == RxNo) {
  dataset->sleep([monitoring_rate]) # wait a little bit before another round of dataset queries
  await
  dataset->result
}

} # end of for loop
} # end of whenever loop

# Logging Frame
l_title_f := frame (log_fr, side='top')
l_title_l := label(l_title_f, text='DATA DISPLAY')

l_alarm_f := frame(log_fr,side='left')
l_alarm_l := label(l_alarm_f, text='Alarms (Temp2>20K): ')
for(i in 1 :RxNo){
  rx[i].T2alarm := label (l_alarm_f,spaste( 'Rx',i), width=3)
}

l_data_f := frame (log_fr, side='left')
l_data_n := label (l_data_f, 'Data shown for:')
l_data_l := label (l_data_f, relief='sunken')
l_data_l->text('Rx# and time')

l_name_f := frame (log_fr, side='left')
l_name_l := label (l_name_f, 'Archive filename')
Arc_name_e := label (l_name_f, relief='sunken')
l_temp_f := frame (log_fr, side='left',relief='groove')
l_vd_f := frame (log_fr, side='left',relief='groove')
l_id_f := frame (log_fr, side='left',relief='groove')
l_vg_f := frame (log_fr, side='left',relief='groove')

# creates the space for display of the 4 temperature values, and the 12 voltage and current values

# Temp
l_temp1_f := frame (l_temp_f, side='top')
l_temp1_l := label (l_temp1_f, 'Temp 1')
l_temp1_e := entry (l_temp1_f, relief='sunken', width=7)
[...]

# VD
l_vd1_f := frame (l_vd_f, side='left')
l_vd1_l := label (l_vd1_f, 'VD1')
l_vd1_e := entry (l_vd1_f, relief='sunken', width=5)
[...]

# ID
l_id1_f := frame (l_id_f, side='left')
l_id1_l := label (l_id1_f, 'ID1')
l_id1_e := entry (l_id1_f, relief='sunken', width=5)
[...]

# VG
l_vg1_f := frame (l_vg_f, side='left')
l_vg1_l := label (l_vg1_f, 'VG1')
l_vg1_e := entry (l_vg1_f, relief='sunken', width=5)
[...]

#=============================================================================

# IF/LO state Logging Frame
IFLOstat_f := frame (log_fr, side='top')
IFLOstat_l := label (IFLOstat_f, 'IF/LO state (digital datas)')

IF1_f := frame (IFLOstat_f, side='left',relief='groove')
IF1_l := label(IF1_f,'IF1',width=7)
IF1_power := label(IF1_f,relief='groove', text='On', width=6, background='green')
IF1_alarm := label(IF1_f,relief='groove', text='Alarm', width=6, background='green')

IF2_f := frame (IFLOstat_f, side='left',relief='groove')
IF2_l := label(IF2_f,'IF2',width=7)
IF2_power := label(IF2_f,relief='groove', text='On', width=6, background='green')
IF2_alarm := label(IF2_f,relief='groove', text='Alarm', width=6, background='green')

Switch_f := frame (IFLOstat_f, side='left',relief='groove')
Switch_l := label(Switch_f,'Switch',width=7)
Switch_power := label(Switch_f,relief='groove', text='On', width=6, background='green')
Switch_alarm := label(Switch_f,relief='groove', text='Alarm', width=6, background='green')

Doubler_f := frame (IFLOstat_f, side='left',relief='groove')
Doubler_l := label(Doubler_f,'Doubler',width=7)
Doubler_power := label(Doubler_f,relief='groove', text='On', width=6, background='green')
Doubler_alarm := label(Doubler_f,relief='groove', text='Alarm', width=6, background='green')

Amp42G_f := frame (IFLOstat_f, side='left',relief='groove')
Amp42G_l := label(Amp42G_f,'42G',width=7)
Amp42G_power := label(Amp42G_f,relief='groove', text='On', width=6, background='green')
Amp42G_alarm := label(Amp42G_f,relief='groove', text='Alarm', width=6, background='green')
_label := label(setup_fr, 'Monitoring Setup ')

### Digital I/O status ****************************

for(i in 1:RxNo) {
   rx[i].setup_fr := frame(setup_fr, side='left', relief='groove')
   rx[i].label := label(rx[i].setup_fr,spaste('Rx',i),background='green')
   rx[i].setup_fr.on:=button(rx[i].setup_fr,'On','check',value=i)
   rx[i].setup_fr.bias:=button(rx[i].setup_fr,'bias','check',value=i)
}

rx[1].setup_fr.on->state(T)

onscreenlabel := label(display_fr,'Local display')

onscr_fr := frame(display_fr, side='left', relief='groove')
onscr:=[]
for(i in 1:RxNo){
   onscr[i] := button(onscr_fr,spaste('Rx',i),'radio',value=i)
}
onscr[1]->state(T)

# *** Monitor control ****************************

for(i in 1:RxNo) {
   whenever rx[i].setup_fr.on->press do { # the user requested to monitor receiver number i, so make it available
      if( rx[$value].setup_fr.on->state()==T ) rx[$value].is_available :=T
      else {
         rx[$value].is_available :=F
         ok:=F
         if ($value == OnScreen) {
            for (j in 1:RxNo) {
               if (rx[j].is_available==T) {
                  OnScreen:=j
                  onscr[j]->state(T)
                  ok:=T
                  break
               }
            }
            if (ok==F) {
               OnScreen:=1
               onscr[1]->state(T)
               logger('NO ACTIVE RECEIVER')
            }
         }
      }
   }
}

for(i in 1:RxNo){
   whenever onscr[i]-->press do { # the user requested receiver number i to be shown on the GUI
      if( rx[$value].is_available ==F){
         onscr[$value]-->state(F)
         OnScreen := 0
         print "Wrong Key :\a"
      } else {
         OnScreen := $value
         print $value
      }
   }
}

# *** bias control *****************************

for(i in 1:RxNo) {
   whenever rx[i].setup_fr.bias-->press do { # the user requested to apply a bias voltage
to the receiver number i
    if (rx[$value].setup_fr.bias->state()==T) rx[$value].bias.byte := 255
    else rx[$value].bias.byte := 0
  }
}

##### LO frame #####

LO_label := label(L0_fr, 'LO/IF Control ')

for (i in 1:RxNo) {
  rx[i].LO_fr := frame (LO_fr, side='left', relief='groove')
  rx[i].LO_label := label(rx[i].LO_fr,spaste('LO',i),background='green')
  rx[i].LO_fr.IF1 := button(rx[i].LO_fr,'IF1','check',value=i)
  rx[i].LO_fr.IF2 := button(rx[i].LO_fr,'IF2','check',value=i)
  rx[i].LO_fr.Doubler := button(rx[i].LO_fr,'Doubler','check',value=i)
  rx[i].LO_fr.amp42G := button(rx[i].LO_fr,'42Gamp','check',value=i)
  rx[i].LO_fr.Switcher := button(rx[i].LO_fr,'Switcher','check',value=i)
}

# *** IF1 control *************************

for (i in 1:RxNo) {
    whenever rx[i].LO_fr.IF1->press do {
        if (rx[$value].LO.fr.IF1->state()==T){
            rx[$value].loDo.byte := rx[$value].loDo.byte -( rx[$value].loDo.byte % 2 )
        } else {
            if ( (rx[$value].loDo.byte % 2) == 0 ) rx[$value].loDo.byte += rx[$value].loDo.byte
        }
    }
} ...

and so on with IF2, Switcher, Doubler and 42G Amplifier controls...

} #

## Message logging window

# Log window.
opf_c.f1226 := frame(main_fr_right, side='left', borderwidth=0)
opf_c.f1227 := frame(main_fr_right, side='left', borderwidth=0)
opf_c.log.tx := text(opf_c.f1226, width=50, height=15, relief='ridge',
   wrap='none', fill='both', disabled=T)
opf_c.log.vsc := scrollbar(opf_c.f1226)
opf_c.log.hsc := scrollbar(opf_c.f1227, orient='horizontal')
opf_c.log.pad := frame(opf_c.f1227, width=20, height=10, expand='none',
   relief='sunken')

whenever
  opf_c.log.hsc->scroll,
opf_c.log.vsc->scroll do {
    opf_c.log.tx->view($value)
  }

whenever
  opf_c.log.tx->yscroll do {
    if ($value[2] == 1) {
        if (!opf_w.log.scroll) {
            opf_c.log.pad->background('blue')
opf_w.log.scroll := T
        }
    } else {
        if (!opf_w.log.scroll) {
            opf_c.log.pad->background('red')
opf_w.log.scroll := F
        }
    }
  }
function ($value)  
}  
  opf_c.log.vsc->view($value)  
}  

whenever  
  opf_c.log.tx->xscroll do {  
    opf_c.log.hsc->view($value)  
  }  

# GUI events:  
whenever exit_btn->press do {  
  if (ComOpened == T)  
    dataset->CloseCom()  
  dataset->quit()             # quit MonClient  
  sendto (ctrl, 'quit', 'quit')  
  exit  
}  

busy := client('timer','-oneshot')  

tk_release ()             # enable the display  

# logger() writes a message to the logging window. A timestamp may be  
# prepended in either brief or full form.  

const logger := function(msg='', colour='black', timestamp='brief', flush=F,  
                          alarm=F)  
  [ ... ]  
}  

# Send an event to the specified agent and also log it.  
const sendto := function(agent=ctrl, event, value='')  
  [ ... ]  
}
Appendix VII:
the AMiBA monitoring software user manual
Procedure to operate the AMIBA monitoring software

Madani El Hariri
Last revised 11/23/04

Revision history:
10/01/04: First version
10/12/04: 2nd version, with more details and screenshots
10/13/04: minor changes
10/26/04: idem
11/23/04: idem

Table of Contents
Using the local monitoring software (Grxmon.g and MonClientRmt)..........................2
Screenshot of the local monitoring program user interface........................................4
Using the remote monitoring software (amibadisp)..................................................5
Screenshot of the remote monitoring program user interface.................................7
**Using the local monitoring software (Grxmon.g and MonClientRmt):**

1) Plug the Rs232 cable to the 2nd serial port of the computer: there should be two COM port with 9 pins on the back of the PC; plug the cable to the bottom one.

To install the software, unzip the file Grxmon.zip to the directory /Grxmon/. (open a terminal window and type `unzip Grxmon.zip -d ~/.` at the Linux prompt) AND before using the software for the first time, create the directories for the recorded data. To do so, type

```
    cd ~/Grxmon
    mkdir data
    cd data
    mkdir Rx1
    mkdir Rx2, etc. to mkdir Rx7
```

2) To start the user interface, type:

```
    cd ~/Grxmon
    glish Grxmon.g
```

If the message “**glish: Command not found**” appears then you have to install glish on this system.
If you mistyped the command and Glish starts, you can stop it at anytime by hitting CTRL and C keys.

3) Wait for the graphical interface to appear (see p.7, then press the **START** button on the upper-left corner of the screen.

4) Choose the receivers you want to monitor by pressing the “On” buttons in the frame called “**Monitoring Setup**” on the left of the screen. At start-up, monitoring of Rx1 is on by default.

Choose your monitoring rate (ie the polling period): 6 queries/minute (period: 10s) or 1 query/minute (period 60s.)

All the receivers for which the “On” button is red are being monitored and having their data archived to a logfile. You can turn off the monitoring of any receiver at any time by pressing the “On” button again.

If you turn a receiver “On” and the receiver returns no response, then the program will show you “**Rx# is not responding**” in the logging window.
However, if a receiver should be functioning properly but the software is not able to monitor the data, reboot the computer. The access to the serial port sometimes block for an unknown reason and rebooting Linux somehow enables serial port access again.
To retrieve the archived datas import the file `data/Rx#/Rx#2004Oct25.csv`, (where the pound sign is the number of the receiver you want, and `2004Oct25` should be the date of the recording) in a spreadsheet. On a Linux system, open it as a regular file (file menu > open) with OpenOffice Calc (Red Hat Menu > Office > Open Office Calc). To review the LO/IF state datas, you should type `emacs Rx#/Rx#2004Oct25DI` it the Linux prompt.

4) To choose which set of datas you want to display locally, click on one of the buttons in the lower-right frame called “Local display”.

The datas are shown on the right part of the screen, and are updated every 5 seconds. The digital datas are displayed in the form of buttons that turn red when off and green when on. You can see when the set of datas that is being displayed has been last updated and from which receiver it came, next to the “Data shown for:” label. If Temp2 becomes critical (higher than 20 Kelvins) in a receiver, an alarm turns red in the upper-right part of the interface.

5) To apply a bias voltage to an HEMT, press one of the “bias” buttons in the “Monitoring Setup” frame.

6) To control the local oscillator and turn on/off one of its components, use the buttons in the “LO/IF Control” frame.

To exit the remote monitoring program, use the Exit button.
Screenshot of the local monitoring program user interface:
Using the remote monitoring software (amibadisp):

1) To install the software, type

```
unzip amibadisp.c -d ~/.
```

in a Linux terminal window.

new

Then type

```
cd ~/amibadisp/.
```

If the address of the local monitoring computer (the server) has changed since last time the program was used, open the file rxmon.reg (type `emacs rxmon.reg` at the prompt) and replace the first line by the new IP address of the server.

2) To start the remote monitoring program, type

```
amibadisp
```

You should see an interface like the one on page 6. The interface is divided in 3 parts. Starting from top, they are: the report window, where the datas are displayed; the message window, where you can see some useful messages about the state of the program and the error messages, and 'keyboard reminder' window, where a list of all the possible keystrokes (include the available receivers) is displayed at anytime.

If the message “Connecting to host xxx.xxx.xxx.xxx, port 5100 | press ctrl-c to abort” is displayed in the message window on startup, then the server cannot be found. Press CTRL key and C to abort program execution then check the adress in rxmon.reg.

If the program displays “server is not responding” then make sure the local monitoring software is running on the server.

If the programs displays “failed to create the desk socket” check if the file rxmon.reg is present in the local directory and if the ethernet is not down. Press CTRL key and C to abort program execution then try again.

If the connection can be established, the datas are retrieved from the server regularly after an interval of time that is set to 2 seconds by default but can be changed by the user.

The lowest part of the screen is refreshed each time a new set of datas is requested and displays all the available receivers and the allowed keystrokes.
See the end of this manual for a complete list.

3. By default, the program displays the datas coming from Rx1. To switch to another receiver, press the digit key corresponding to its number (key “2” for Rx2, etc.).

If all the datas are zero, the monitoring you’re receiving has been turned off on the server. Just switch monitoring to another receiver.

If the temperature #2 is too high in the receiver being monitored, there’s an alarm. To switch to a silent alarm, press z.

The allowed keystrokes are:

- h to get a short help
- c to change the monitoring rate. Usage is: c then a digit key. The period can be set to 10, 60, 120 or 600 seconds.
- z to switch silent mode on/off.
- q to quit the program.
Screenshot of the remote monitoring program user interface:

<table>
<thead>
<tr>
<th>Monitoring Rx1 every 2 seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rx1temp1</td>
</tr>
<tr>
<td>Rx1Vds1</td>
</tr>
<tr>
<td>Rx1Ids1</td>
</tr>
<tr>
<td>Rx1Vgs1</td>
</tr>
</tbody>
</table>

IF1: power OFF    alarm OFF
IF2: power OFF    alarm OFF
SWITCH: power OFF alarm OFF
DOUBLER: power OFF    42G Amp: power OFF alarm OFF

record: 7

press 1 h l c q