GLIB GUI – USER MANUAL

Chapter 1 - Introduction

The GlibGui is a Graphical user interface for the control of GLIB cards. It has been implemented almost entirely in Java. At the lowest level it exchanges UDP messages with the GLIB H/W via a Java class. The format of these UDP packets is defined by the IPbus standard. Alternatively the communication with the GLIB H/W can be done via PCI. In this case a special driver (io_rcc) and some libraries of the ATLAS TDAQ project are required. The GlibGui also includes Java classes for the handling of XML files. These basic classes are independent of the GlibGui and may also be used by people implementing their own high level S/W.

This manual is divided into two parts. The first part will describe the steps necessary to run the application, and the minimum that is required to allow the application to function properly. The second part is a guide, explaining step by step, all the parts of the application.

Chapter 2 - Instalation

2.1. Requeriments

Hardware	
Minimum	Recommended
Pentium2 450 64 Mb RAM	Pentium3 800 256Mb RAM
Software	
Operative System	
Windows XP/Vista/7, Linux, Mac OSX	
JRE (Java Run Time Environment)	
Java JRE 1.6	
Network interface	
1 GBE Ethernet port with RJ45 connector	

2.2. Instalation

To run the application, first check if the Java version installed on your computer is correct (version 1.6). Secondly, several additional libraries must be installed before running the application; these are discussed in the Castor XML documentation (Appendix A).

They are:

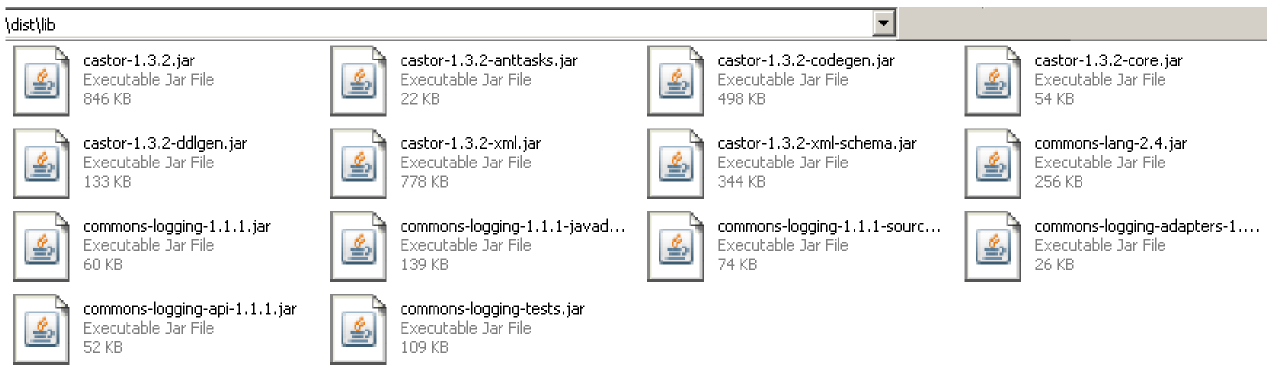
- Castor Package
- Commons-logging Package
- Java Native Access (JNA)

Once they are installed, all the jar files relating to them should be located in the folder 'lib' which will in turn be located in the same folder as GlibGui.jar file.

To install Castor XML, first download the most recent version (it was version 1.3.2 at the time of writing) from the Castor Web Site. Found at http://castor.codehaus.org/download.html

Once unpacked, the *.jar files should be added to the 'lib' folder. The second library required by the GLibGui is **Commons logging**, which enables the user to discover, at runtime, the logging framework available in the class path. Often these libraries are used by other libraries and framework developers to determine what product to use.

To install Commons logging, first download the most recent version (it was version 1.1.1 at the time of writing) from the Apache Commons Web Site. Found at http://commons.apache.org/logging/download_logging.cgi.



```
\dist\lib
castor-1.3.2.jar                    Executable Jar File  846 KB
castor-1.3.2-anttasks.jar           Executable Jar File  22 KB
castor-1.3.2-codegen.jar            Executable Jar File  498 KB
castor-1.3.2-core.jar               Executable Jar File  54 KB
castor-1.3.2-ddlgen.jar             Executable Jar File  133 KB
castor-1.3.2-xml.jar                Executable Jar File  778 KB
castor-1.3.2-xml-schema.jar         Executable Jar File  344 KB
commons-lang-2.4.jar                Executable Jar File  256 KB
commons-logging-1.1.1.jar           Executable Jar File  60 KB
commons-logging-1.1.1-javad...      Executable Jar File  139 KB
commons-logging-1.1.1-sourc...      Executable Jar File  74 KB
commons-logging-adapters-1....      Executable Jar File  26 KB
commons-logging-api-1.1.1.jar       Executable Jar File  52 KB
commons-logging-tests.jar           Executable Jar File  109 KB
```

Figure 1: Necessary libraries

Once these files have been installed, Netbeans may ask for the file "commons-lang-2.4.jar". In this case, Netbeans provides the file "org-appache-commons-lang.jar" which will correctly replace this package. So, the user can find the package there, or he can look for in the path above mentioned (http://commons.apache.org).

In the case of Java Native Access, first download the most recent version (it was version 3.4.1 at the time of writing) from the the Source for Java Technology Collaboration Page. Found at https://github.com/twall/jna/blob/3.4.1/dist/jna.jar

Due to PCI dynamic libraries, before running the GLIB GUI, the user has to execute the next command in the terminal:

```
export LD_LIBRARY_PATH=/afs/cern.ch/atlas/project/tdaq/cmt/tdaq/tdaq-04-00-01/installed/i686-slc5-gcc43-opt/lib
```

Or also, he can change the classpath in the environment variables from the computer, writing the path above mentioned.

This way, the link will be established in the classpath from the machine and the GUI could take the library when it is necessary.

Once all the libraries are added, two important files should be installed in addition. As mentioned in the Castor Xml Documentation, the GUI needs two XML files to work correctly.

These files need to be stored in the same folder as lib:

- Registers.xml: This file contains a model of the registers of the GLIB card. A version of the file with the standard registers of the GLIB can be found in the GlibGui distribution. Users who implement additional registers in the FPGA have to add them to "registers.xml" themselves.
- Instructions.xml: This XML file allows IPbus commands to be executed in batch mode. The GlibGui distribution will contain an example file. It is up to the user to generate his own lists.

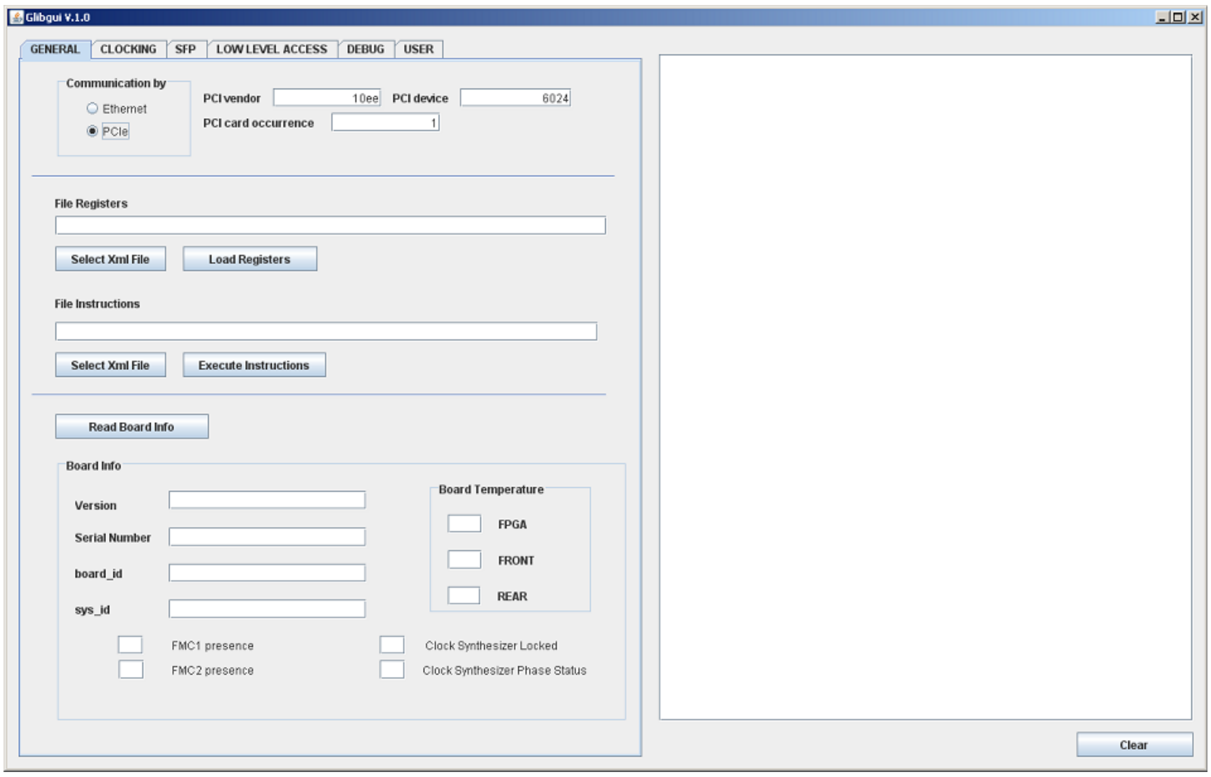
lib
GlibGui.jar
Executable Jar File
181 KB
instructions.xml
XML Document
2 KB
registers.xml
XML Document
7 KB

Figure 2: Content of main folder

Chapter 3 - User Manual

3.1. Workspace

The GUI is intended to communicate with the Glib, so as to handle data exchange. There are six tabs and a screen where the user can see the data exchange procedure with the Glib.



Communication by: Ethernet [✓ PCIe]
PCI vendor: 10ee
PCI device: 6024
PCI card occurrence: 1

Figure 3: Workspace

The tabs behave as described below:

- GENERAL: In this tab the user can specify the *ip* and *port* of the Glib needed to establish the communication. As well, the board information can be read. This information covers:
    - Serial Number
    - Version
    - Board temperature
    - Indicates FMCs presence
    - Indicates clock synthesized locked

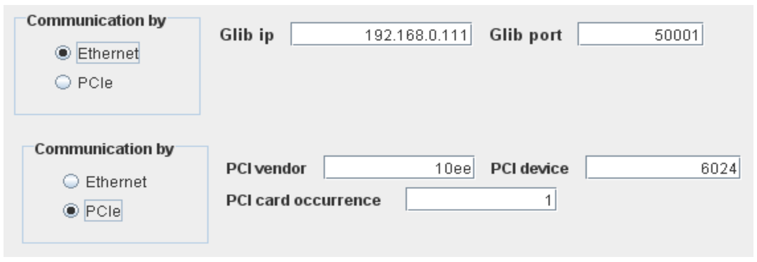
- Board_id and Sys-id

- CLOCKING : In this tab the user can change the parameters from the ctrl register, which take control over the external clocking circuitry.
- SFP: The functions of this tab are still under discussion and have not yet been implemented In this tab the user can check the status from the SFPs. If they have some error, the corresponding box will color in red. If not, will not color.
- LOW LEVEL ACCESS: In this tab, the user can exchange data with the glib in an interactive way.
- DEBUG: This tab is mostly for the S/W developers. It provides additional information about the internal operation of the GLIB GUI.
- USER: The functions of this tab are still under discussion and have not yet been implemented

3.2. General

At first, the user has to select the communication way to access to the GLIB. If Ethernet is selected, two fields will be shown (ip and port). In the case that PCI is selected, three fields will be shown (pci vendor id, pci device id and pci card occurrence). In all these fields, the user has to introduce the corresponding parameters from the GLIB.

Anyway all the parameters that have been used frequently, are inserted default.


Communication by
[● Ethernet] ○ PCIe
Glib ip: 192.168.0.111
Glib port: 50001

Communication by
○ Ethernet [● PCIe]
PCI vendor: 10ee
PCI device: 6024
PCI card occurrence: 1

Figure 4: Select Channel

Once is done, the file (registers.xml) that describes the registers of the GLIB has to be selected and loaded: It is not necessary to load a batch file (Instructions.xml) if only the low level functions will be used.

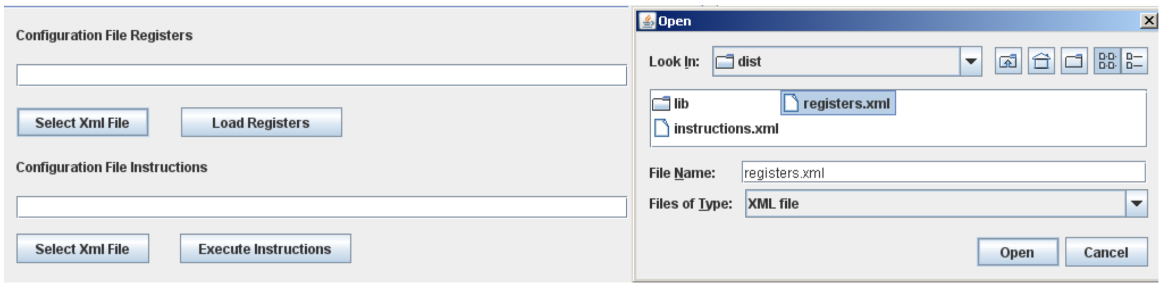


Figure 5: XML file selection

First use the button “Select XML file” in order to select a file. Then click on “Load register”. This will trigger the GLibGui to process the XML file.

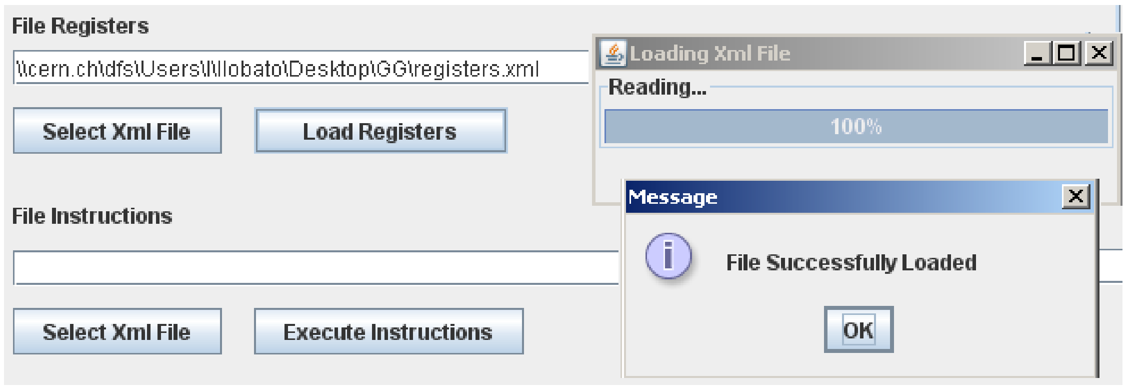
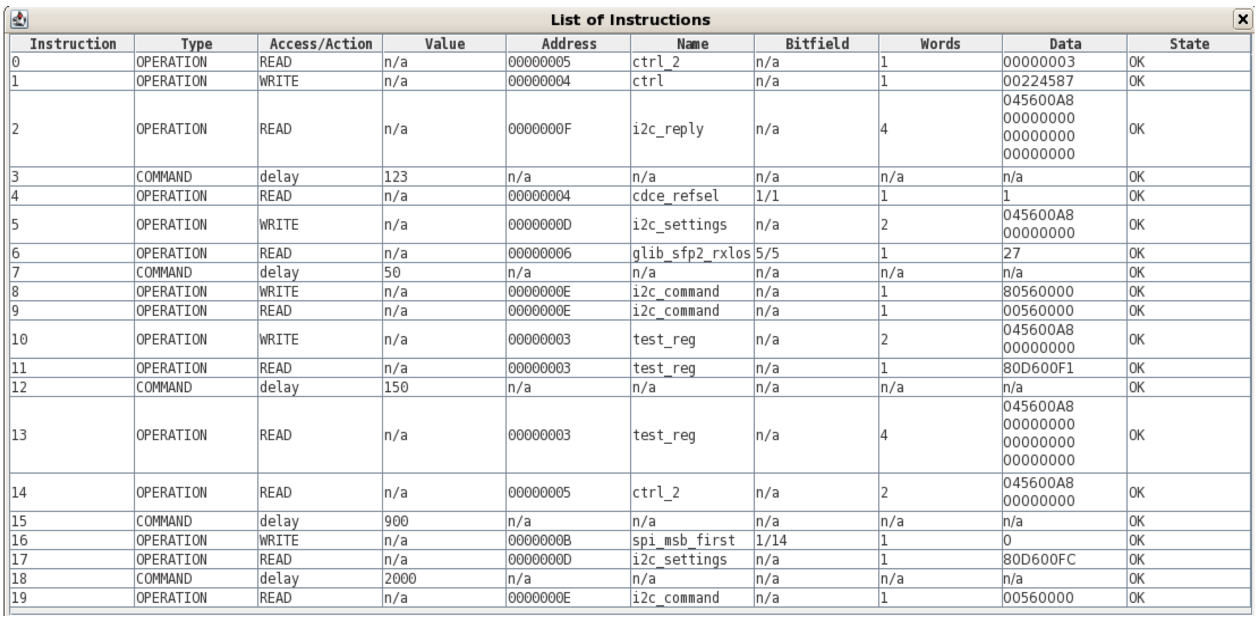


Figure 6: Load successfully

The file “registers.xml” provides the application with the complete configuration of a GLIB in terms of register definitions. Once the file has been loaded the user can check in the Low Level Access tab if all the registers where properly created. (See the corresponding chapter).

Loading a file for the execution of a list of commands works in the same way. The GUI will be able to load lists of IP-bus instructions from XML and to display the result of each command in a table. The status of each instruction (OK or ERROR) will be flagged.. The execution of a list will not be terminated in case of an error.

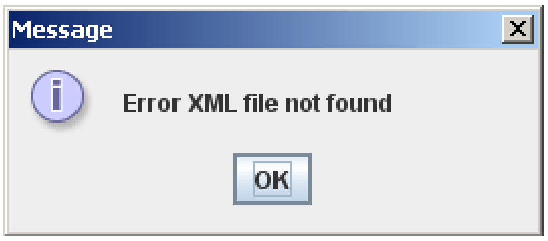


List of Instructions

Instruction	Type	Access/Action	Value	Address	Name	Bitfield	Words	Data	State
0	OPERATION	READ	n/a	00000005	ctrl_2	n/a	1	00000003	OK
1	OPERATION	WRITE	n/a	00000004	ctrl	n/a	1	00224587	OK
2	OPERATION	READ	n/a	0000000F	i2c_reply	n/a	4	045600A8 00000000 00000000 00000000	OK
3	COMMAND	delay	123	n/a	n/a	n/a	n/a	n/a	OK
4	OPERATION	READ	n/a	00000004	cdce_refsel	1/1	1	1	OK
5	OPERATION	WRITE	n/a	0000000D	i2c_settings	n/a	2	045600A8 00000000	OK
6	OPERATION	READ	n/a	00000006	glib_sfp2_rxlos	5/5	1	27	OK
7	COMMAND	delay	50	n/a	n/a	n/a	n/a	n/a	OK
8	OPERATION	WRITE	n/a	0000000E	i2c_command	n/a	1	80560000	OK
9	OPERATION	READ	n/a	0000000E	i2c_command	n/a	1	00560000	OK
10	OPERATION	WRITE	n/a	00000003	test_reg	n/a	2	045600A8 00000000	OK
11	OPERATION	READ	n/a	00000003	test_reg	n/a	1	80D600F1	OK
12	COMMAND	delay	150	n/a	n/a	n/a	n/a	n/a	OK
13	OPERATION	READ	n/a	00000003	test_reg	n/a	4	045600A8 00000000 00000000 00000000	OK
14	OPERATION	READ	n/a	00000005	ctrl_2	n/a	2	045600A8 00000000	OK
15	COMMAND	delay	900	n/a	n/a	n/a	n/a	n/a	OK
16	OPERATION	WRITE	n/a	0000000B	spi_msb_first	1/14	1	0	OK
17	OPERATION	READ	n/a	0000000D	i2c_settings	n/a	1	80D600FC	OK
18	COMMAND	delay	2000	n/a	n/a	n/a	n/a	n/a	OK
19	OPERATION	READ	n/a	0000000E	i2c_command	n/a	1	00560000	OK

Figure 7: Example output for the execution of a list

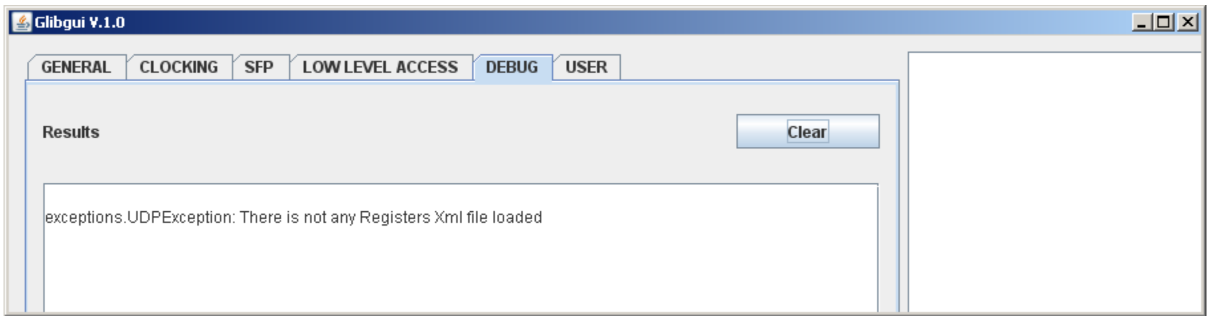
To execute the list of instructions successfully, the registers.xml file must be loaded, if not, an error message will appear.



Error XML file not found

Figure 8: Error

If the user wants the cause of error with more detail, in the Debug tab, may find the origin of the problem.



exceptions.UDPException: There is not any Registers Xml file loaded

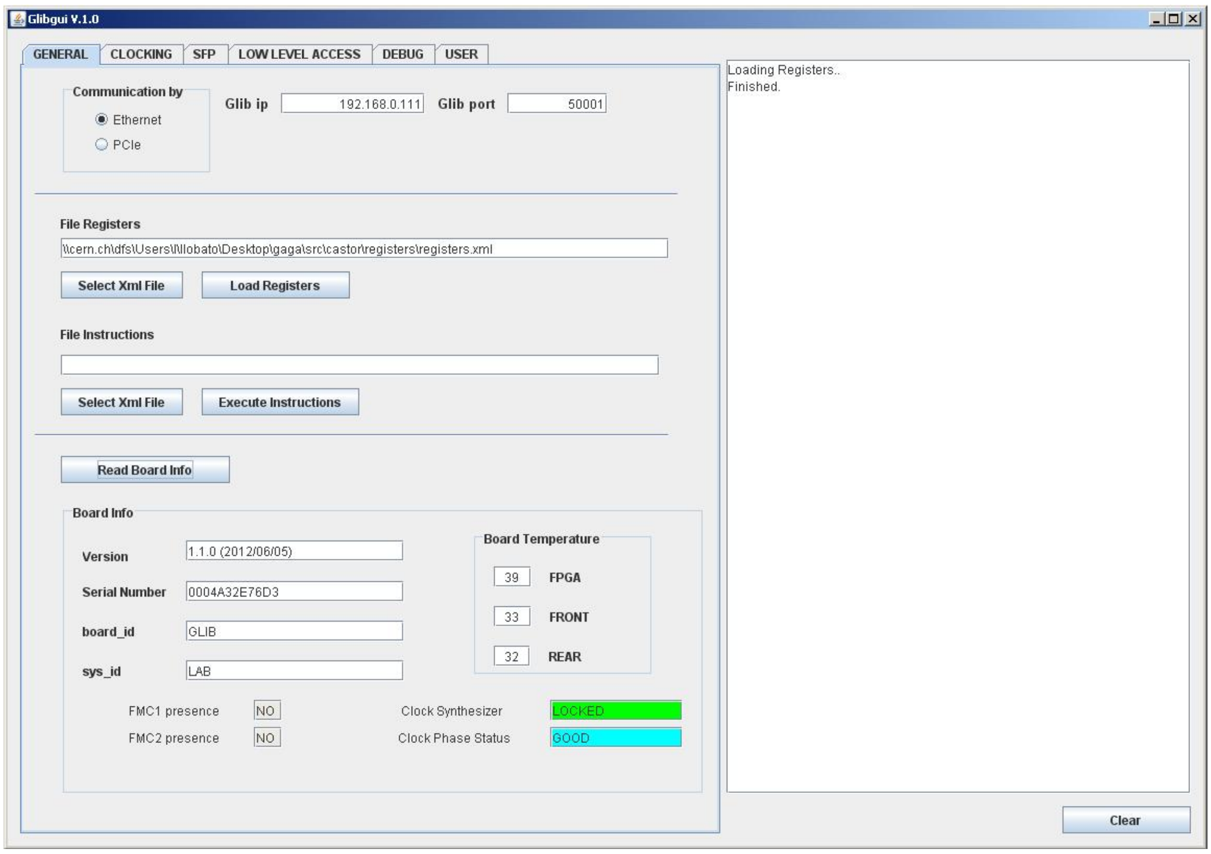
Figure 9: Debug Tab

Board Info" is another field in the GUI. The user has to press "Read Board Info" button and all the information will be shown in the corresponding box. In additional if the application detects the presence of FMCs or that clock synthesizer is locked, the box will color on green.

Moreover, inside of “Board Info” another field is shown, called “Board Temperature” with three fields, which will show the corresponding values from the board.

- FGPA: Measures the temperature of the FPGA’s die
- FRONT: Measures the temperature on the front side of the board
- REAR: Measures the temperature on the rear side of the board.

In the below picture, can be seen an example:



Communication by: (● Ethernet) ○ PCIe
Glib ip: 192.168.0.111 Glib port: 50001

File Registers: \\cern.ch\dfs\Users\l\llobato\Desktop\gaga\src\castor\registers\registers.xml

File Instructions:

Board Info

Version	1.1.0 (2012/06/05)
Serial Number	0004A32E76D3
board_id	GLIB
sys_id	LAB

Board Temperature

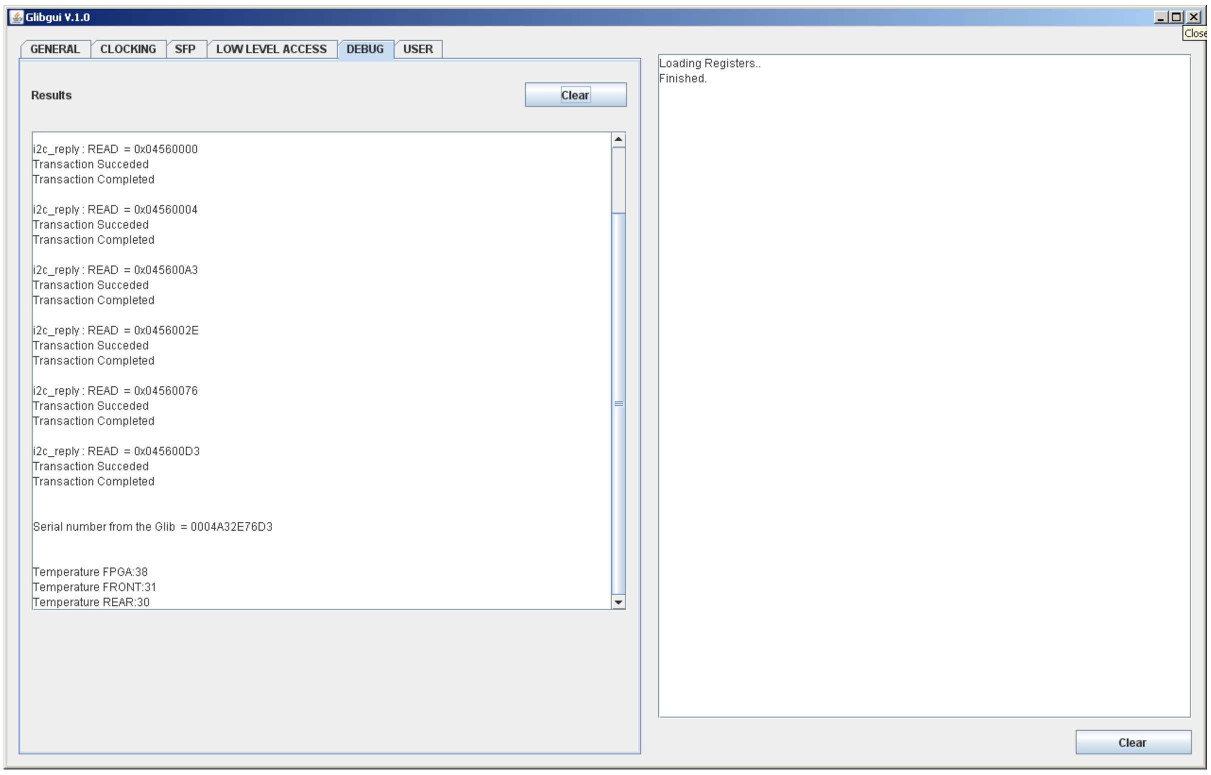
39	FPGA
33	FRONT
32	REAR

FMC1 presence	NO	Clock Synthesizer	LOCKED
FMC2 presence	NO	Clock Phase Status	GOOD

```
Loading Registers..
Finished.
```

Figure 10: Read Board Info

If the user wants to know all the operations carried out in order to read all the information, he can see them in the Debug tab.



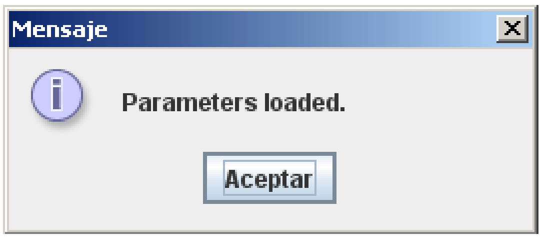
```
i2c_reply : READ  = 0x04560000
Transaction Succeded
Transaction Completed

i2c_reply : READ  = 0x04560004
Transaction Succeded
Transaction Completed

i2c_reply : READ  = 0x045600A3
Transaction Succeded
Transaction Completed

i2c_reply : READ  = 0x0456002E
Transaction Succeded
Transaction Completed

i2c_reply : READ  = 0x04560076
Transaction Succeded
Transaction Completed

i2c_reply : READ  = 0x045600D3
Transaction Succeded
Transaction Completed


Serial number from the Glib  = 0004A32E76D3


Temperature FPGA:38
Temperature FRONT:31
Temperature REAR:30
```

```
Loading Registers..
Finished.
```

Figure 11: Operations from reading board info.

3.3. Clocking

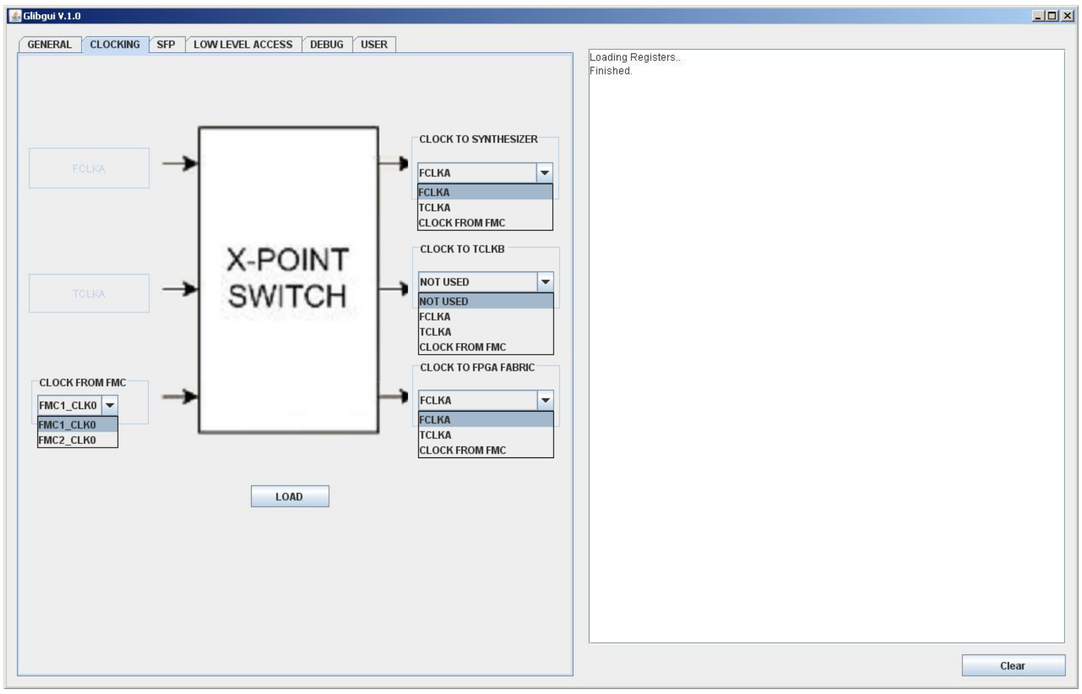
In the second tab, the user can modify the parameters from the control register. Just have to select the options and to press the button load. A message will show up with the next message:



Parameters loaded.

Figure 12: Parameters loaded

In this tab, there are several options. Every option can modify the status of the register control, which is used to configure the clock circuitry.

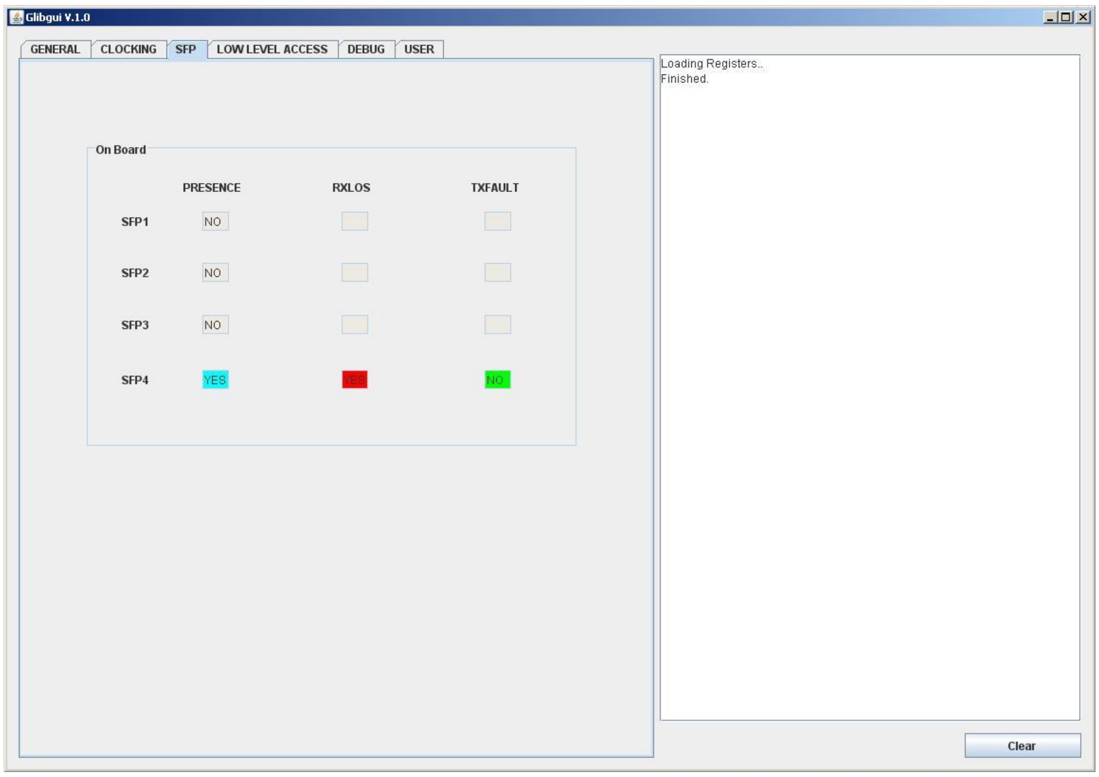


```
Loading Registers..
Finished.
```

Figure 13: Clocking tab

3.4. Sfp

Furthermore, in the SFP tab, the user can check the state from various external components. Currently, SFPs.



On Board

	PRESENCE	RXLOS	TXFAULT
SFP1	NO		
SFP2	NO		
SFP3	NO		
SFP4	YES	YES	NO

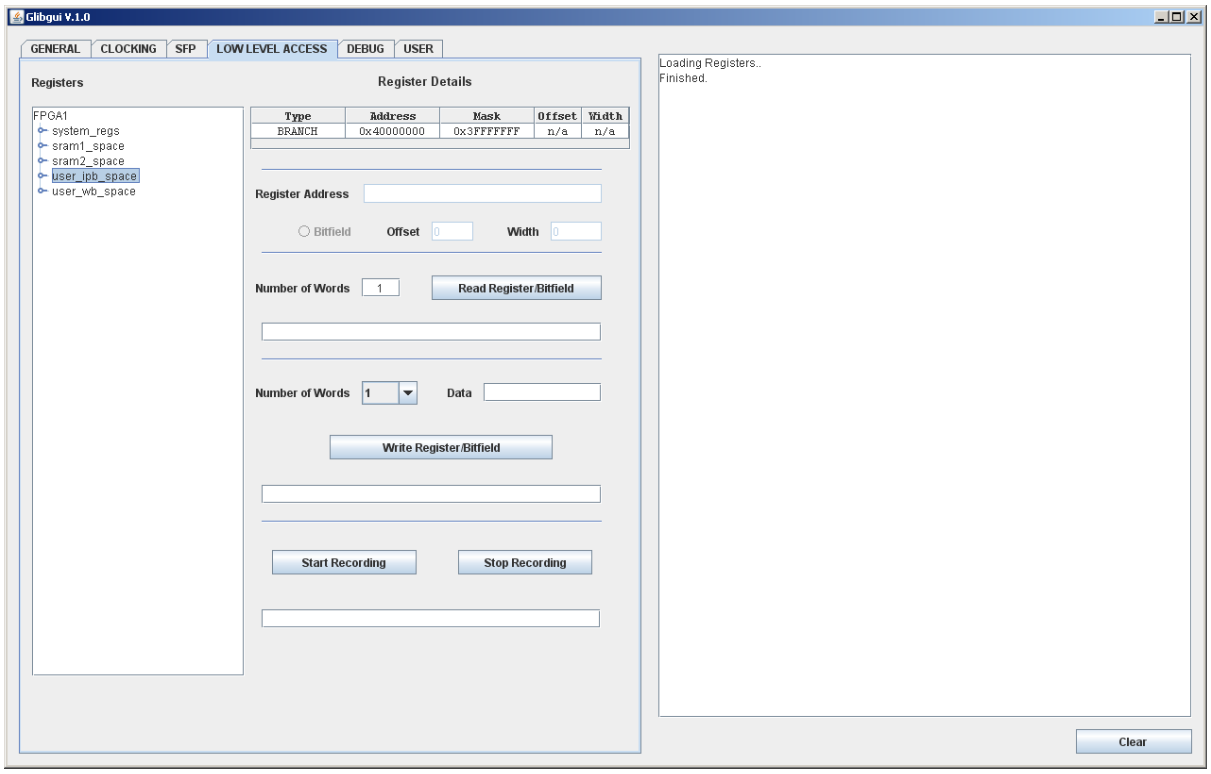
```
Loading Registers..
Finished.
```

Figure 14: Status from the external components SFPs

3.5. Low Level Access

In this tab, the user can find two ways to interact with the Glib. On the left, the tree with the names of the registers loaded previously. The user has to select the name of register which wants to read or write and then, just press the corresponding button.

In order to facilitate the navigation throughout all the registers, there is a table called *Registers Details*, which shows all the attributes with references to the register selected by the user.



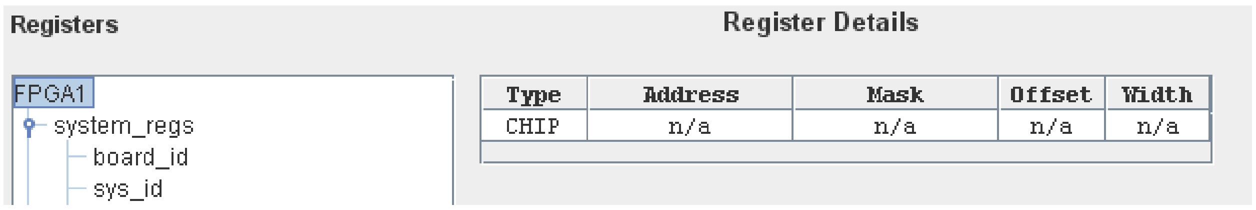
Type	Address	Mask	Offset	Width
BRANCH	0x40000000	0x3FFFFFFF	n/a	n/a

```
Loading Registers..
Finished.
```

Figure 15: Low Level Access tab

The picture above shows the details for a bitfield, but other examples are:

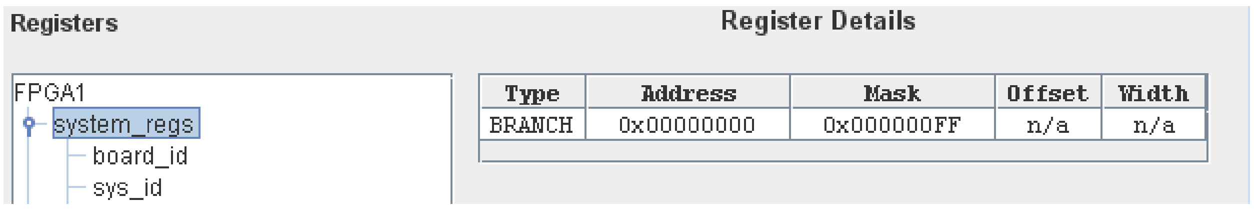
- Name of Chip:



Type	Address	Mask	Offset	Width
CHIP	n/a	n/a	n/a	n/a

Figure 16: Chip details

- Name of Branch:


Registers

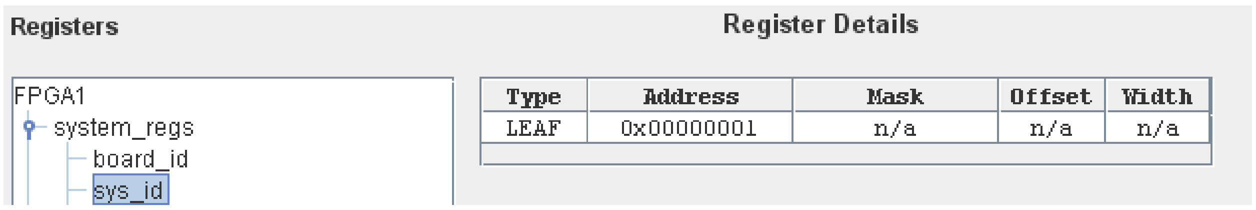
```
FPGA1
  system_regs
    board_id
    sys_id
```

Register Details

Type	Address	Mask	Offset	Width
BRANCH	0x00000000	0x000000FF	n/a	n/a

Figure 17: Branch details

- Name of a Leaf:


Registers

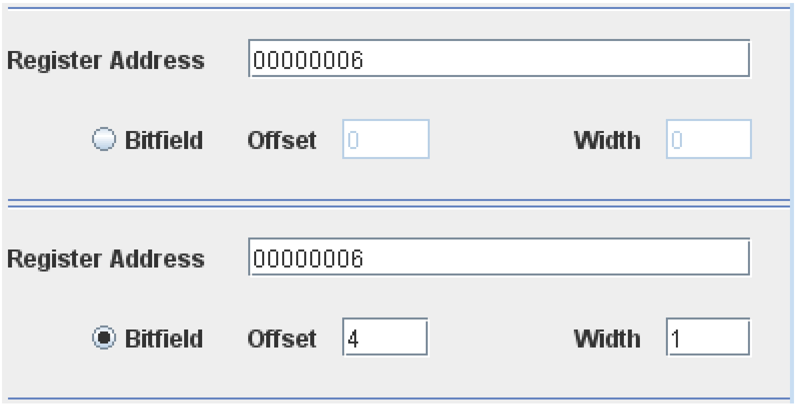
```
FPGA1
  system_regs
    board_id
    sys_id
```

Register Details

Type	Address	Mask	Offset	Width
LEAF	0x00000001	n/a	n/a	n/a

Figure 18: Leaf details

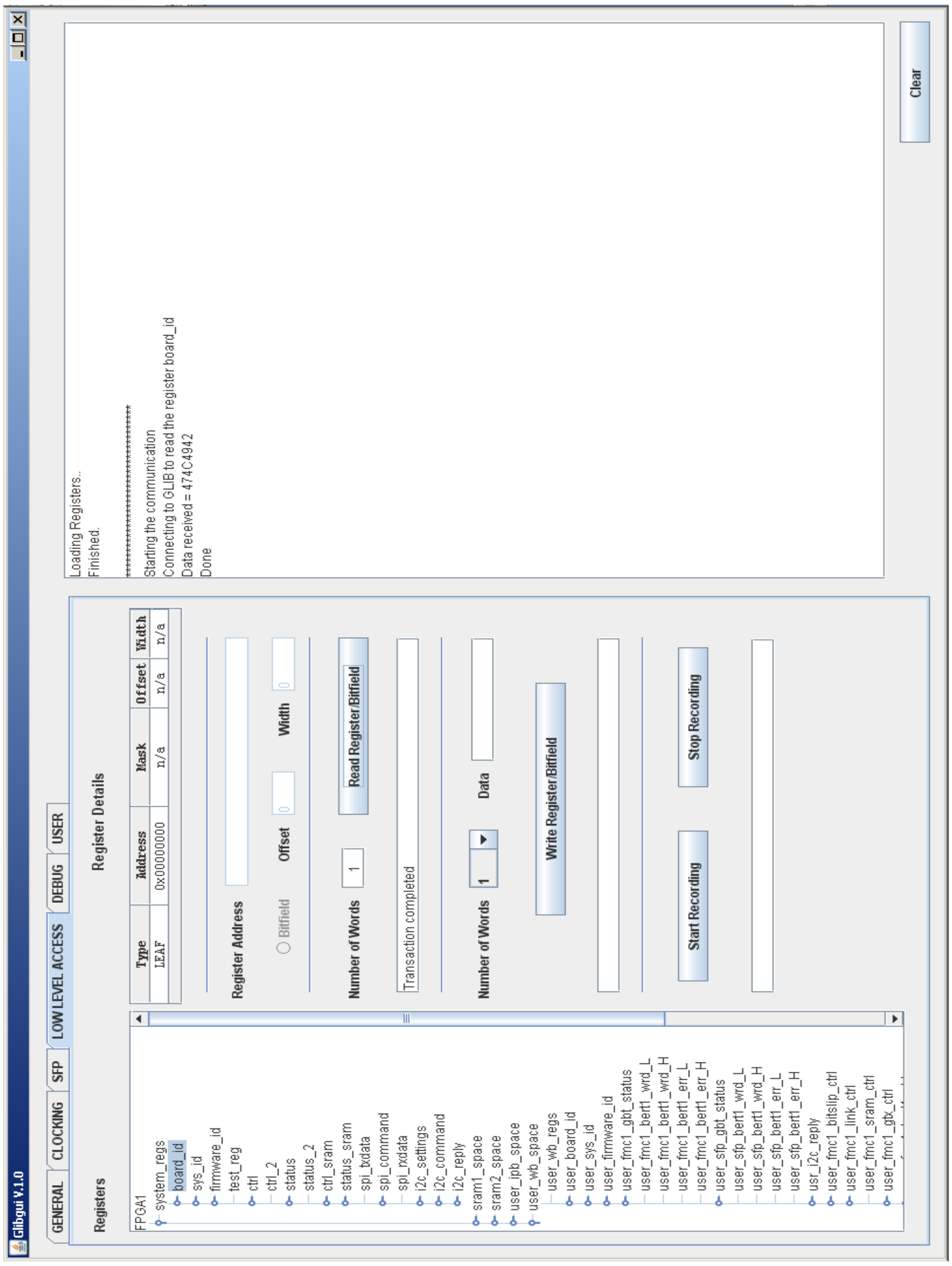
As it was mentioned above, alternatively, the user can read registers specifying the address manually. Once the user uses this input mask, the tree with the names of the registers will be disabled and vice versa. If the user was working with the address entered by hand, and wishes to switch to working with the tree, he just has to press in the label "Registers" and the tree will be enabled again. If the user wants to read a bitfield, he has to check the "Bitfield" button and enter the offset and width.


Register Address: 00000006
( ) Bitfield   Offset: 0   Width: 0

Register Address: 00000006
(●) Bitfield   Offset: 4   Width: 1

Figure 19: Introduce address manually

- Read one register: The user can choose one register from the list, or insert the address by hand. Then, he has to specify the number of words that he wants to read (in this case, one) and press the button "Read Register/Bitfield".



Type	Address	Mask	Offset	Width
LEAF	0x00000000	n/a	n/a	n/a

Offset: 0 Width: 0

Number of Words: 1

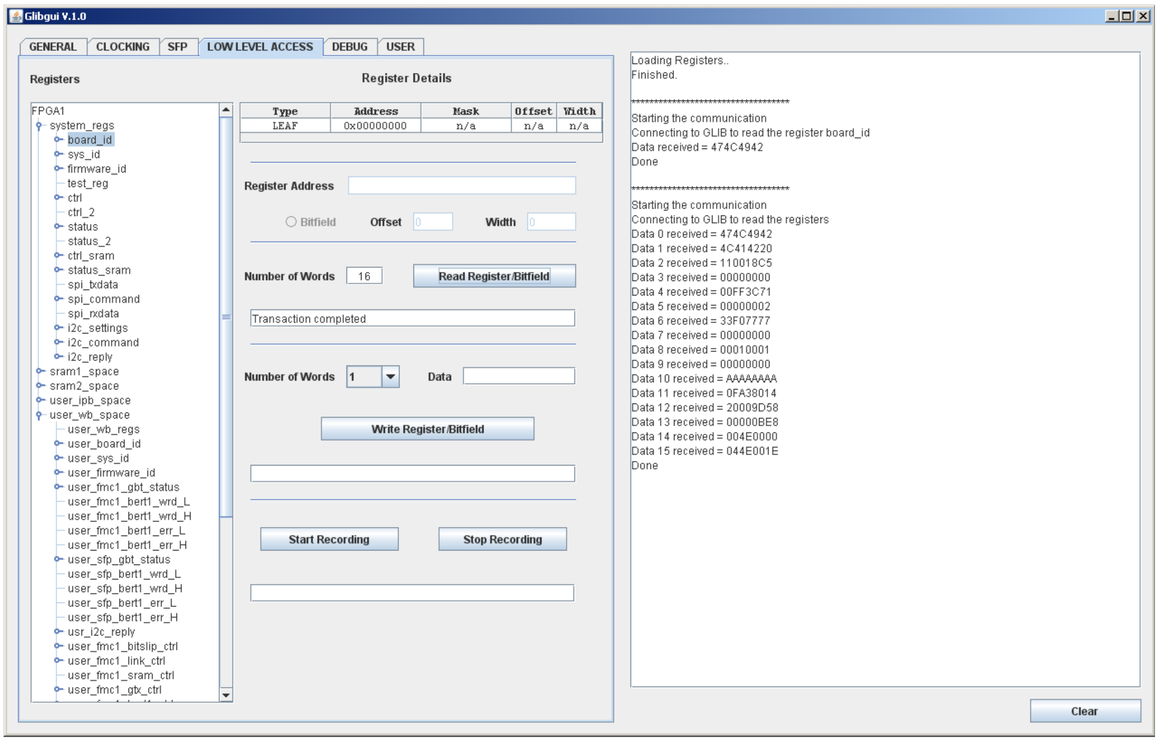
Transaction completed

Number of Words: 1

```
Loading Registers..
Finished.
******************************
Starting the communication
Connecting to GLIB to read the register board_id
Data received = 474C4942
Done
```

Figure 20: Read one register

- Read several registers: The user can choose one register from the list, or enter the address by hand. In this case, the user must enter the number of words that he wants to read, and press the button ‘Read Register/Bitfield’. This way, the GUI will read the contents of register chosen, and the following registers (as many registers as number of words specified).



Type	Address	Mask	Offset	Width
LEAF	0x00000000	n/a	n/a	n/a

Number of Words: 16

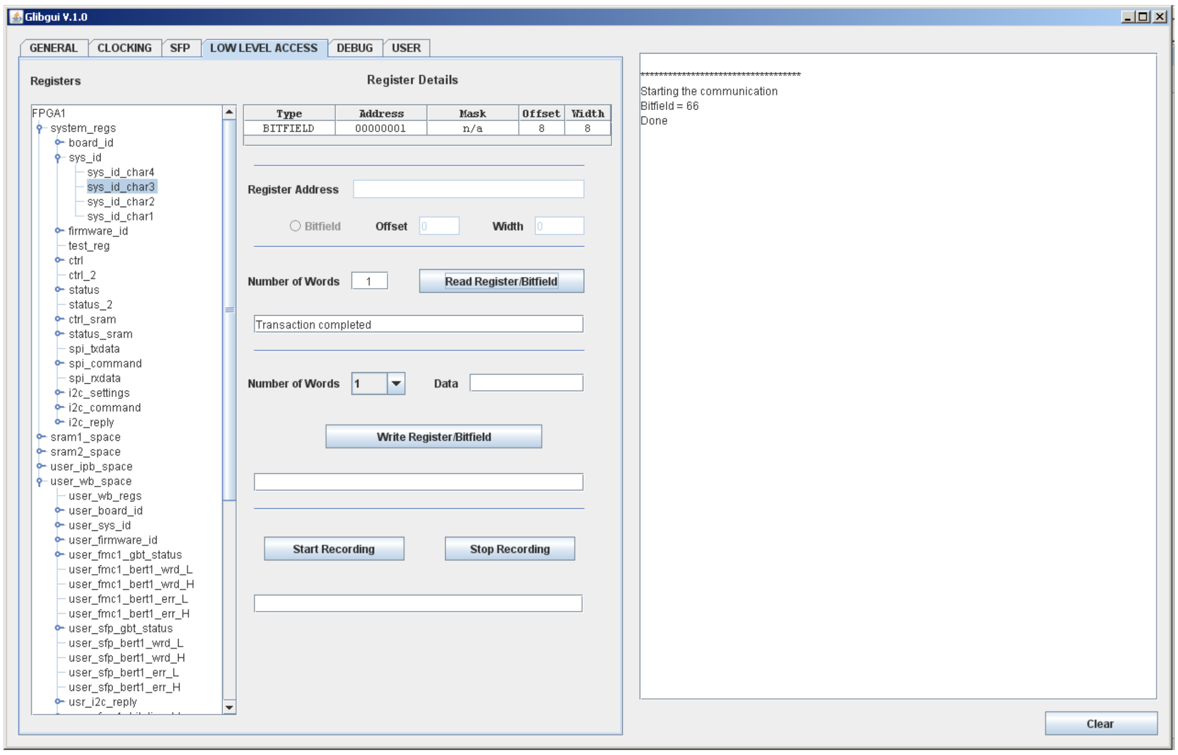
Transaction completed

Number of Words: 1

```
Loading Registers..
Finished.

************************************
Starting the communication
Connecting to GLIB to read the register board_id
Data received = 474C4942
Done

************************************
Starting the communication
Connecting to GLIB to read the registers
Data 0 received = 474C4942
Data 1 received = 4C414220
Data 2 received = 110018C5
Data 3 received = 00000000
Data 4 received = 00FF3C71
Data 5 received = 00000002
Data 6 received = 33F07777
Data 7 received = 00000000
Data 8 received = 00010001
Data 9 received = 00000000
Data 10 received = AAAAAAAA
Data 11 received = 0FA38014
Data 12 received = 20009D58
Data 13 received = 00000BE8
Data 14 received = 004E0000
Data 15 received = 044E001E
Done
```

Figure 21: Read several registers

- Read Bitfield: The user can select one register from the list, or insert the address manually. In this latter case, mark the option bitfield as well and enter the offset and width. If the values are wrong, as above, an error message will appear.



Type	Address	Mask	Offset	Width
BITFIELD	00000001	n/a	8	8

Number of Words: 1

Transaction completed

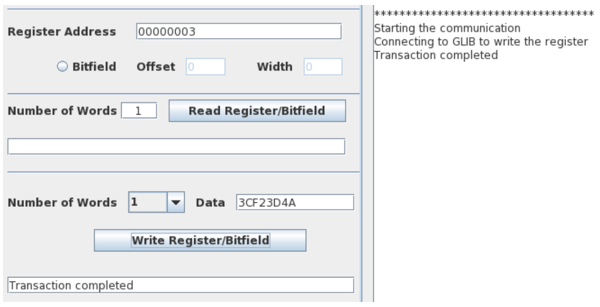
Number of Words: 1

```
************************************
Starting the communication
Bitfield = 66
Done
```

Figure 22: Read bitfield

Writing values to registers works in the same way as reading; the only difference is that the user has to enter the data and to press the button “Write Register/Bitfield”.

- Write one register: The user can select one register from the list, or insert the address manually. Then, the user has to enter the number of words that he wants to write (in this case, one) and the data that he wants to insert, and finally, press the “Write Register/Bitfield” button.



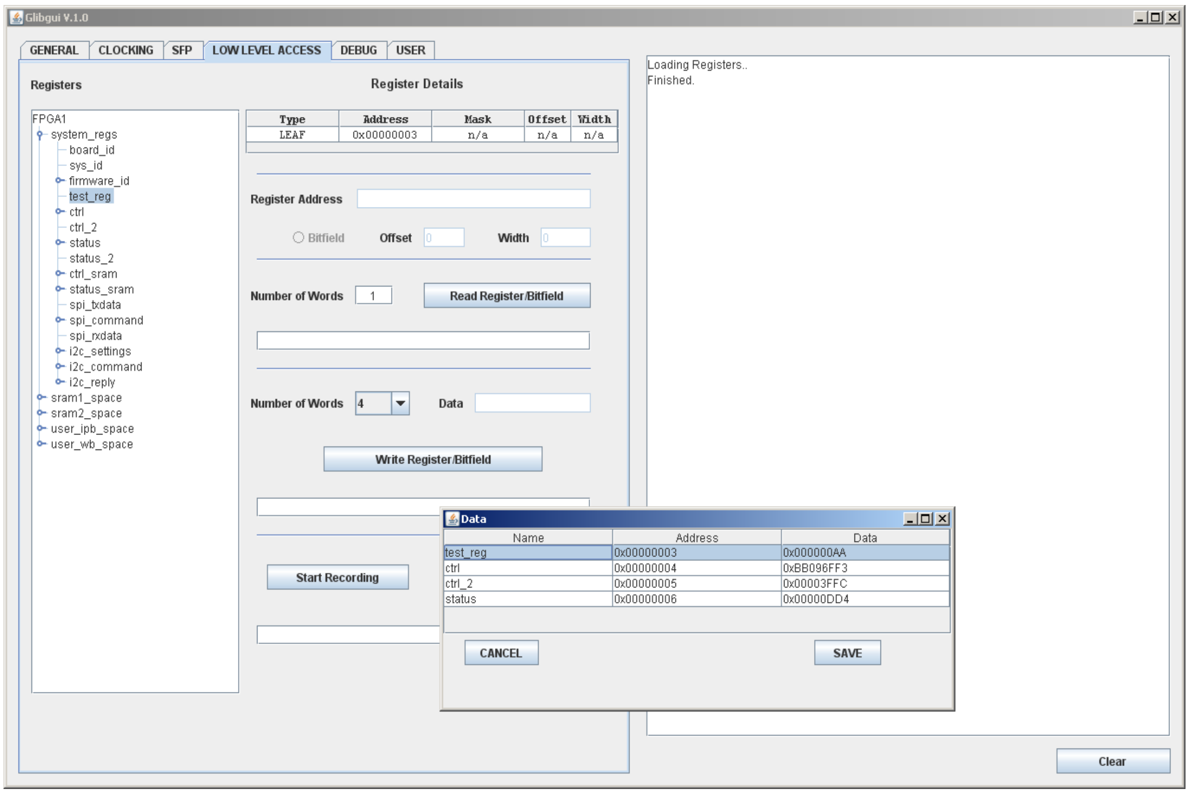
Register Address: 00000003
Bitfield — Offset: 0 — Width: 0
Number of Words: 1
Number of Words: 1 — Data: 3CF23D4A
Transaction completed

```
************************************
Starting the communication
Connecting to GLIB to write the register
Transaction completed
```

Figure 23: Write one register

- Write several registers: The user can select one register from the list, or insert the address manually. Then, the user has to enter the number of words that he wants to write, and press the “Write Register/Bitfield” button. A table will appear with the name and address of the registers that the user wants to write.

  In this way, the GUI will write the contents of selected register as well as the subsequent registers (as many registers as specified in “number of words”).



Glibgui V.1.0

```
FPGA1
  system_regs
    board_id
    sys_id
    firmware_id
    test_reg
    ctrl
    ctrl_2
    status
    status_2
    ctrl_sram
    status_sram
    spi_txdata
    spi_command
    spi_rxdata
    i2c_settings
    i2c_command
    i2c_reply
  sram1_space
  sram2_space
  user_ipb_space
  user_wb_space
```

Type	Address	Mask	Offset	Width
LEAF	0x00000003	n/a	n/a	n/a

Register Address:
Bitfield — Offset: 0 — Width: 0
Number of Words: 1
Number of Words: 4 — Data:

```
Loading Registers..
Finished.
```

Data

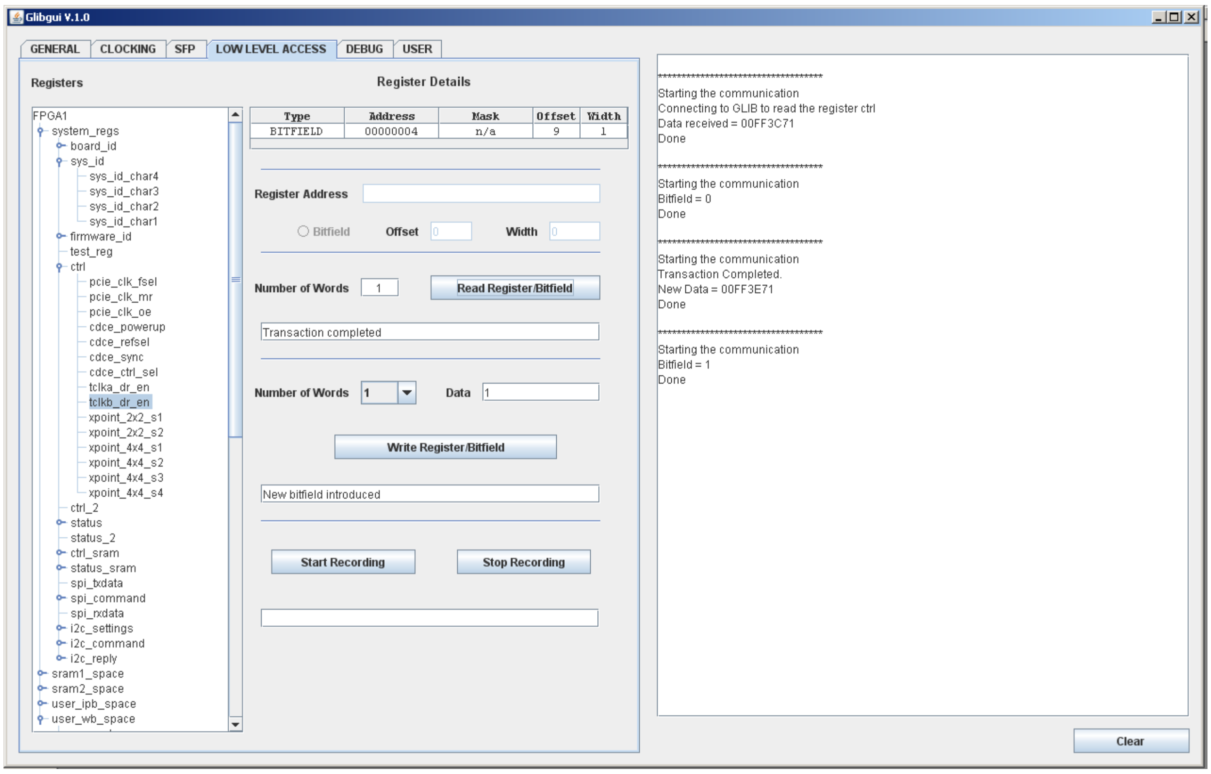
Name	Address	Data
test_reg	0x00000003	0x000000AA
ctrl	0x00000004	0xBB096FF3
ctrl_2	0x00000005	0x00003FFC
status	0x00000006	0x00000DD4

Figure 24: Write several registers

The user can check the data from the registers that were changed, reading from the registers again.

- Write Bitfield: The user can select one register from the list, or insert the address manually. In this last case, the user has tomark the option ‘Bitfield’ as well, as set the offset and width. As it was mentioned above, if the values are wrong, an error message will appear. If the operation was completed successfully, the user can read again to check that the bitfield was changed, as well as the data of the register.

  The next image shows a sequence of this operation (read the register, read the bitfield, change the bitfield and read the bitfield again).



```
FPGA1
  system_regs
    board_id
    sys_id
      sys_id_char4
      sys_id_char3
      sys_id_char2
      sys_id_char1
    firmware_id
    test_reg
    ctrl
      pcie_clk_fsel
      pcie_clk_mr
      pcie_clk_oe
      cdce_powerup
      cdce_refsel
      cdce_sync
      cdce_ctrl_sel
      tclka_dr_en
      tclkb_dr_en
      xpoint_2x2_s1
      xpoint_2x2_s2
      xpoint_4x4_s1
      xpoint_4x4_s2
      xpoint_4x4_s3
      xpoint_4x4_s4
    ctrl_2
    status
    status_2
    ctrl_sram
    status_sram
    spi_txdata
    spi_command
    spi_rxdata
    i2c_settings
    i2c_command
    i2c_reply
  sram1_space
  sram2_space
  user_ipb_space
  user_wb_space
```

Type	Address	Mask	Offset	Width
BITFIELD	00000004	n/a	9	1

Register Address:

Bitfield Offset: 0 Width: 0

Number of Words: 1

Transaction completed

Number of Words: 1 Data: 1

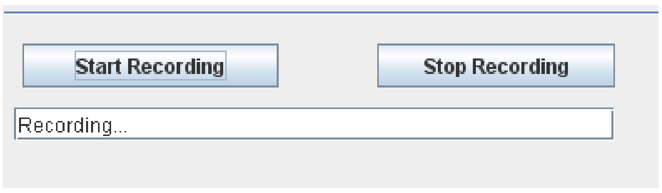
New bitfield introduced

```
***********************************
Starting the communication
Connecting to GLIB to read the register ctrl
Data received = 00FF3C71
Done

***********************************
Starting the communication
Bitfield = 0
Done

***********************************
Starting the communication
Transaction Completed.
New Data = 00FF3E71
Done

***********************************
Starting the communication
Bitfield = 1
Done
```

Figure 25: Procedure Read-Write-Read bitfield

- Record the operations: It also has to be possible for a user to record the commands that he executes interactively into a log file and to re-play them later (see also below: list processing).

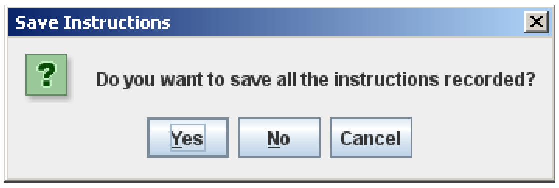
At the precise moment that the user wants to start recording, has to press the "Start Recording" button. At that moment, as will be seen in the image below, a message will be shown in the box, which says"Recording ..". This will mean that all executions made for the user, will be saved.



Recording...

Figure 26: Record Button

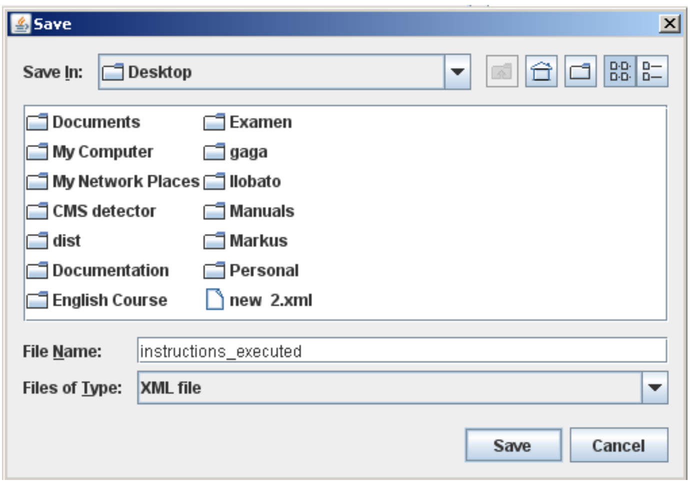
When the user wants to stop recording, has to press the "Stop Recording" and automatically, the program stops recording.


Save Instructions

Do you want to save all the instructions recorded?

Figure 27: Save Instructions

Then, a window will appear asking the user to save the file, and if the user agrees, he has to select the file name (always xml file) and give it to save.

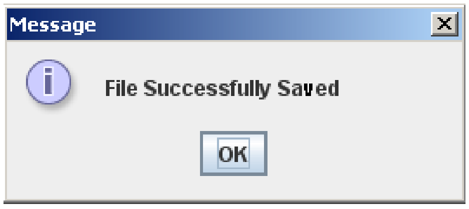


File Name: instructions_executed

Files of Type: XML file

Figure 28: Dialog save instructions

File will be saved automatically, showing a confirmation message.

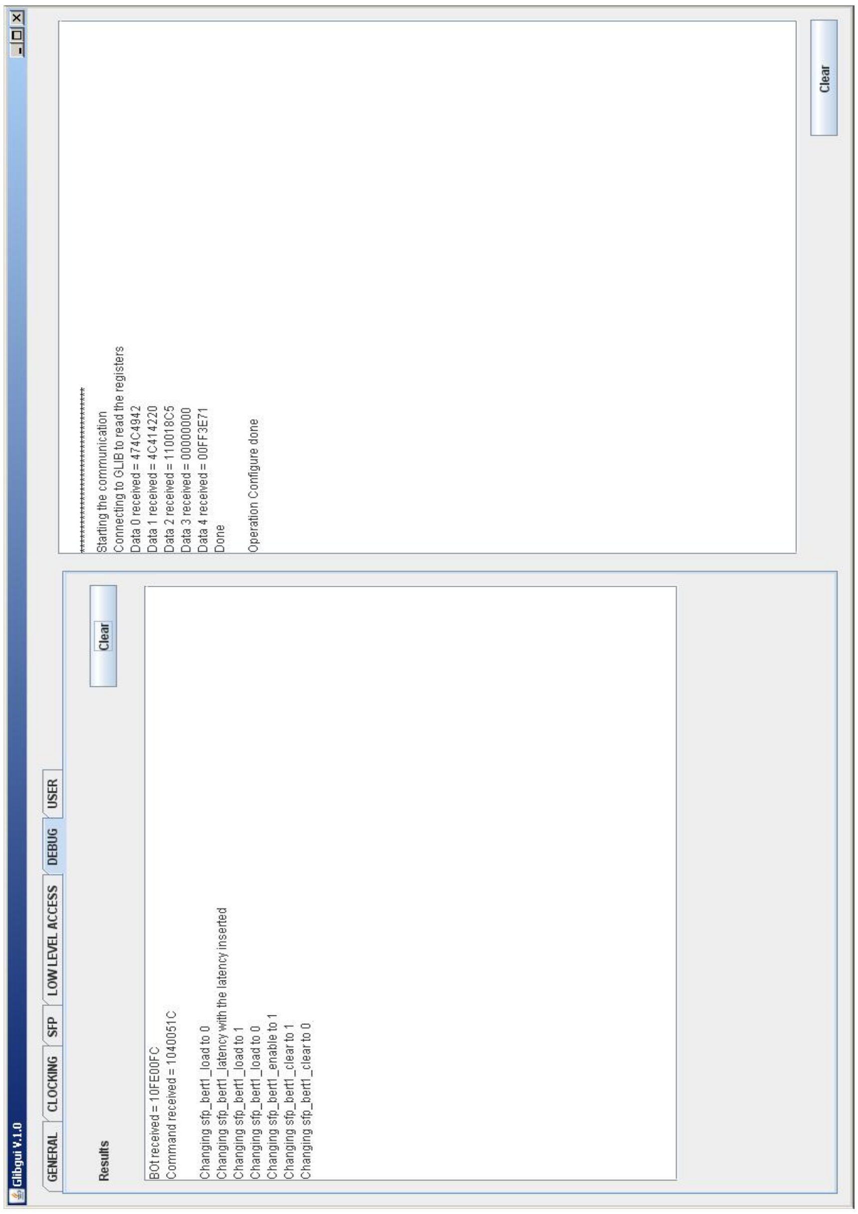

Message

File Successfully Saved

Figure 29: Confirmation saving

3.6. Debug

This tab is intended for the user, to check if all the operations relating to the format of a UDP package communicating with the GLIB. This way, the user can know if an error produced at the time of sending was caused by the package format. Simultaneously you can see the data being received.



```
BOt received = 10FE00FC
Command received = 1040051C

Changing sfp_bert1_load to 0
Changing sfp_bert1_latency with the latency inserted
Changing sfp_bert1_load to 1
Changing sfp_bert1_load to 0
Changing sfp_bert1_enable to 1
Changing sfp_bert1_clear to 1
Changing sfp_bert1_clear to 0
```

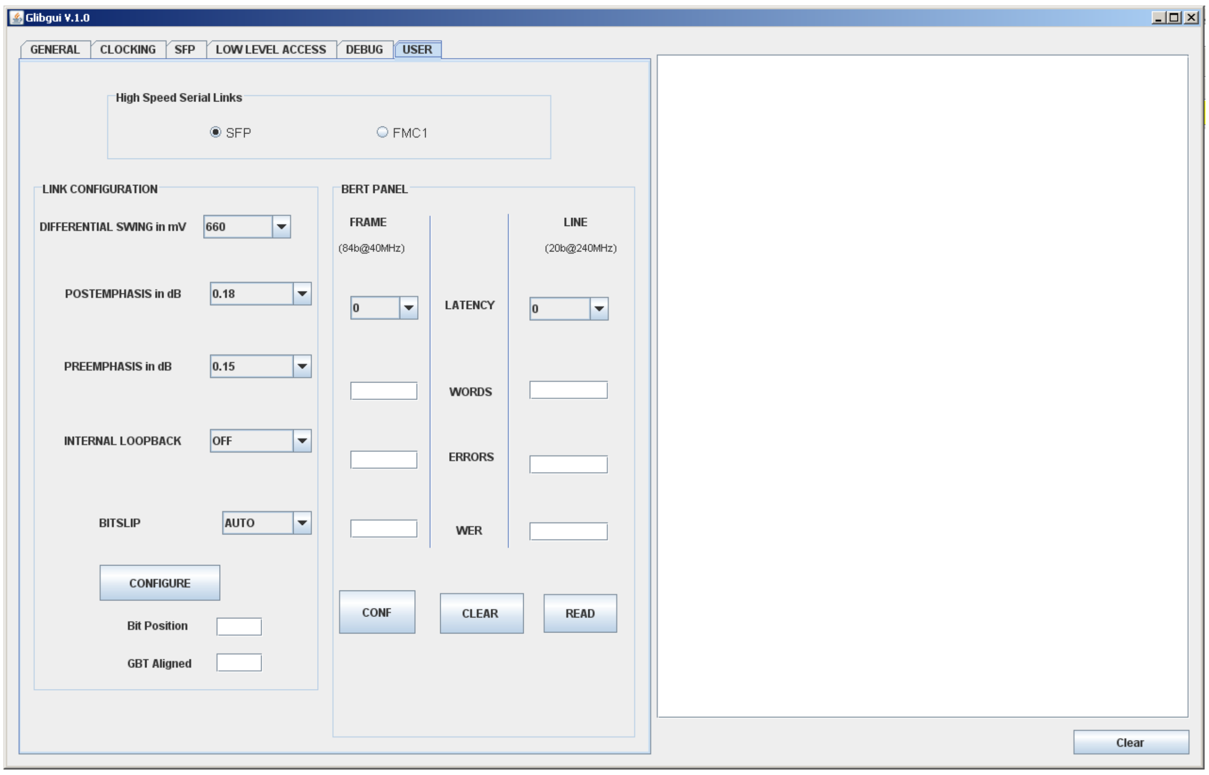
```
****************************************
Starting the communication
Connecting to GLIB to read the registers
Data 0 received = 474C4942
Data 1 received = 4C414220
Data 2 received = 110018C5
Data 3 received = 00000000
Data 4 received = 00FF3E71
Done

Operation Configure done
```

Figure 30: Debug Screen

Also, this tab is used to check the errors that are produced, with more precision. Every time that an error is produced, the exception description will be shown in this tab.

3.7. User

In the last tab, the user can modify the parameters from the user registers.


Glibgui V.1.0

GENERAL | CLOCKING | SFP | LOW LEVEL ACCESS | DEBUG | USER

High Speed Serial Links: [✓ SFP] FMC1

LINK CONFIGURATION

DIFFERENTIAL SWING in mV: 660

POSTEMPHASIS in dB: 0.18

PREEMPHASIS in dB: 0.15

INTERNAL LOOPBACK: OFF

BITSLIP: AUTO

CONFIGURE

Bit Position

GBT Aligned

BERT PANEL

FRAME (84b@40MHz) | LINE (20b@240MHz)

LATENCY: 0 | 0

WORDS

ERRORS

WER

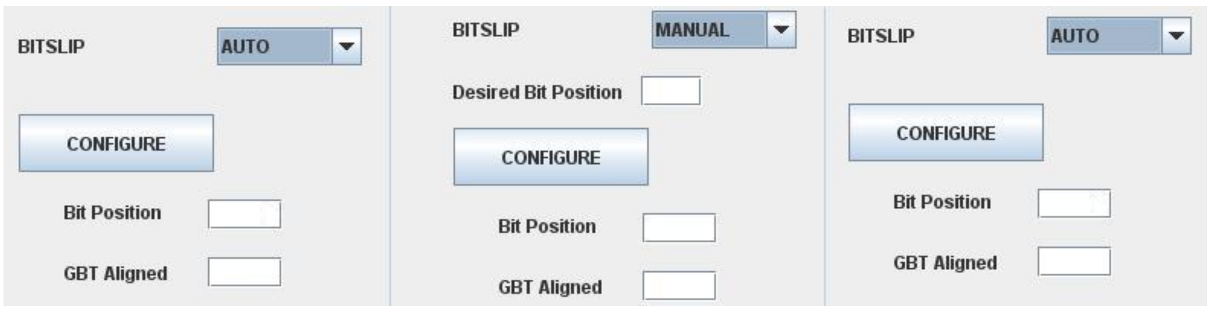
CONF | CLEAR | READ

Clear

Figure 31: User Tab

First at all, has to select the high speed serial link between the two options and then to select the parameters from the Configuration panel. It is very important that the user configure this panel before Bert panel, because if not, it will not work out.

In the option BITSLIP, there are three options.


BITSLIP: AUTO
CONFIGURE
Bit Position
GBT Aligned

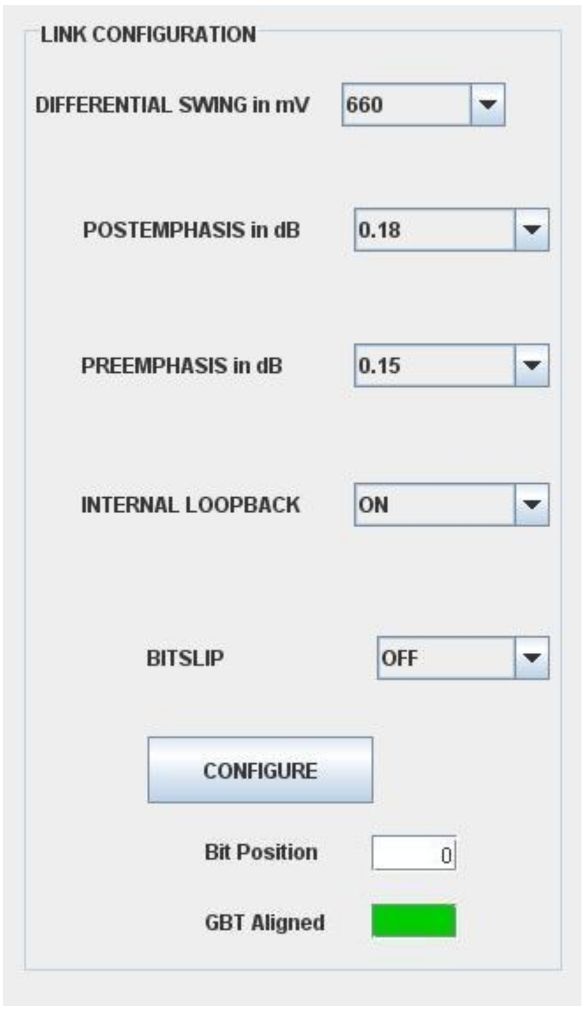
BITSLIP: MANUAL
Desired Bit Position
CONFIGURE
Bit Position
GBT Aligned

BITSLIP: AUTO
CONFIGURE
Bit Position
GBT Aligned

Figure 32: BITSLIP options

In the second case, “MANUAL”, once the user selected this option, a new field will appear in the panel, “Desired Bit Position”, because of the user can introduce the bit position value.

In the other case, once all the parameters were selected, the user has to press the CONFIGURE button and he can get the results. The Bit Position should be a number between 0 and 19.


LINK CONFIGURATION

DIFFERENTIAL SWING in mV: 660

POSTEMPHASIS in dB: 0.18

PREEMPHASIS in dB: 0.15

INTERNAL LOOPBACK: ON

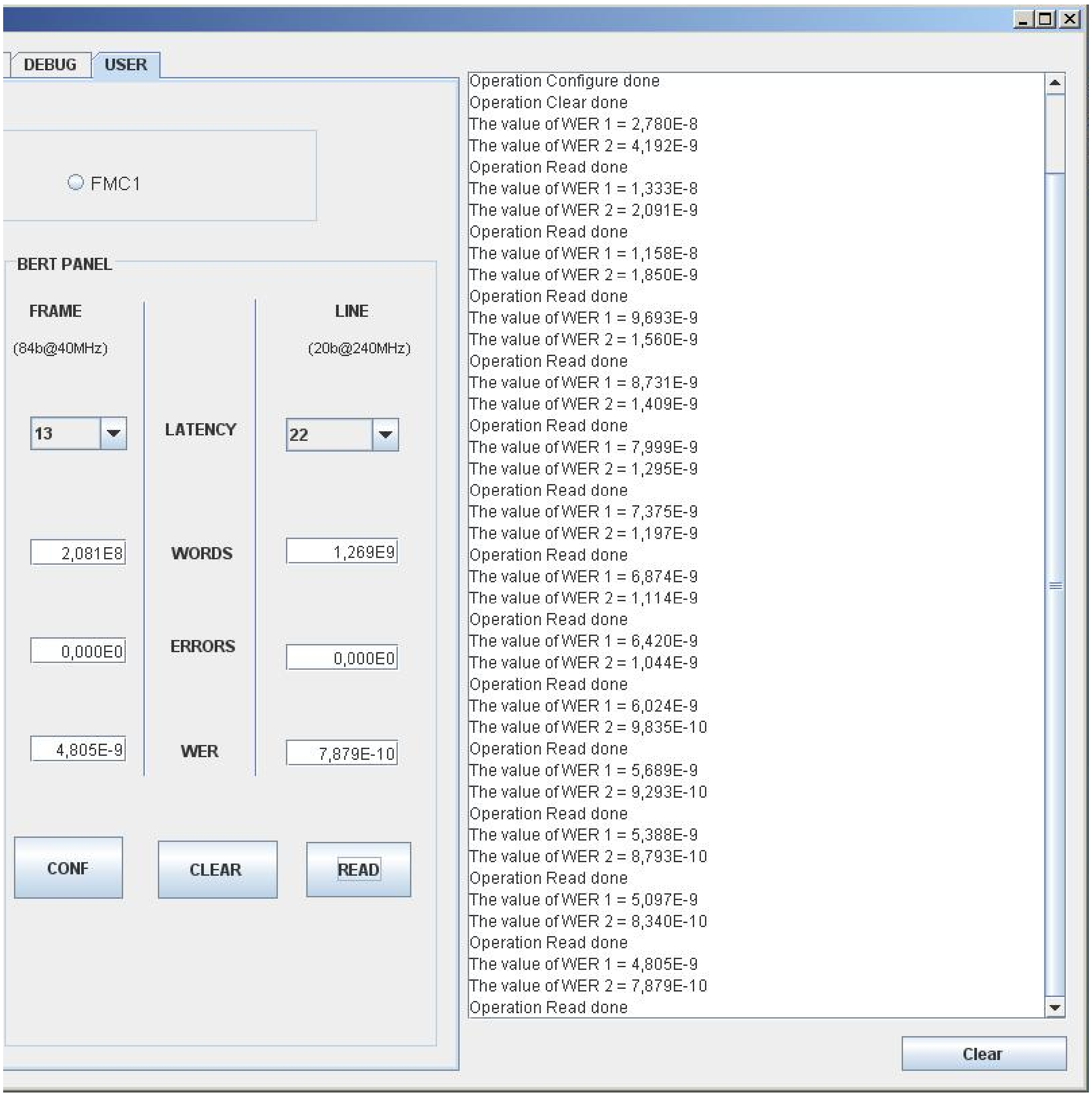
BITSLIP: OFF

Bit Position: 0

GBT Aligned: 🟩

Figure 33: Link Configuration

Once is done, the user can configure the BERT and to get the Word Error Rate in scientific notation. First, the user has to select in both options the latency in the list available. Then, he has to press the button “Configure”. And finally, just has to press any of the rest buttons. In the case of READ button, the number of words, errors and the corresponding word error rate result will appear in fields from each Bert. Also, the user can check the “WER” as many times as he wants, pressing the button READ.

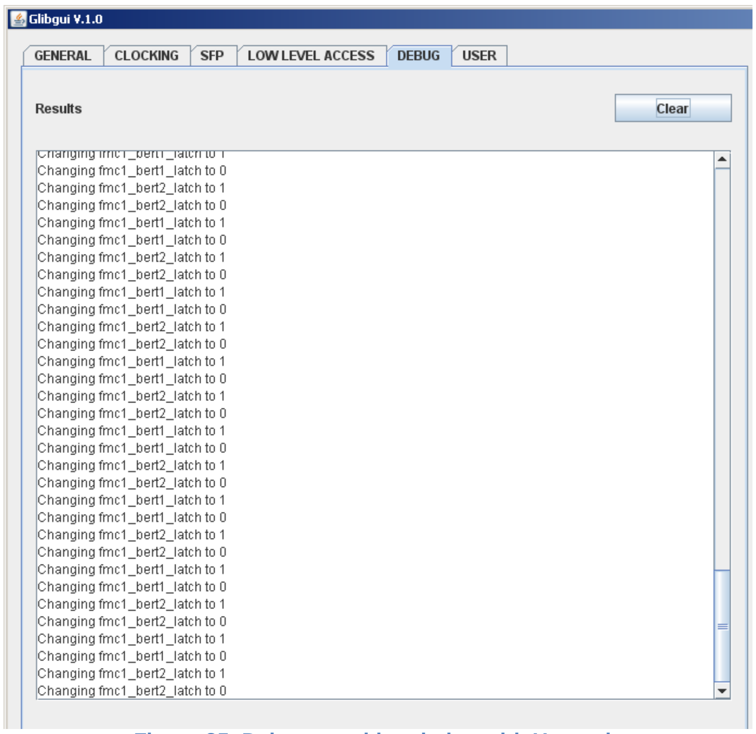


FRAME (84b@40MHz)		LINE (20b@240MHz)
13	LATENCY	22
2,081E8	WORDS	1,269E9
0,000E0	ERRORS	0,000E0
4,805E-9	WER	7,879E-10

```
Operation Configure done
Operation Clear done
The value of WER 1 = 2,780E-8
The value of WER 2 = 4,192E-9
Operation Read done
The value of WER 1 = 1,333E-8
The value of WER 2 = 2,091E-9
Operation Read done
The value of WER 1 = 1,158E-8
The value of WER 2 = 1,850E-9
Operation Read done
The value of WER 1 = 9,693E-9
The value of WER 2 = 1,560E-9
Operation Read done
The value of WER 1 = 8,731E-9
The value of WER 2 = 1,409E-9
Operation Read done
The value of WER 1 = 7,999E-9
The value of WER 2 = 1,295E-9
Operation Read done
The value of WER 1 = 7,375E-9
The value of WER 2 = 1,197E-9
Operation Read done
The value of WER 1 = 6,874E-9
The value of WER 2 = 1,114E-9
Operation Read done
The value of WER 1 = 6,420E-9
The value of WER 2 = 1,044E-9
Operation Read done
The value of WER 1 = 6,024E-9
The value of WER 2 = 9,835E-10
Operation Read done
The value of WER 1 = 5,689E-9
The value of WER 2 = 9,293E-10
Operation Read done
The value of WER 1 = 5,388E-9
The value of WER 2 = 8,793E-10
Operation Read done
The value of WER 1 = 5,097E-9
The value of WER 2 = 8,340E-10
Operation Read done
The value of WER 1 = 4,805E-9
The value of WER 2 = 7,879E-10
Operation Read done
```

Figure 34: BERT panel

In the Debug tab, he can check the operations which are being realized.



```
Changing fmc1_bert1_latch to 1
Changing fmc1_bert1_latch to 0
Changing fmc1_bert2_latch to 1
Changing fmc1_bert2_latch to 0
Changing fmc1_bert1_latch to 1
Changing fmc1_bert1_latch to 0
Changing fmc1_bert2_latch to 1
Changing fmc1_bert2_latch to 0
Changing fmc1_bert1_latch to 1
Changing fmc1_bert1_latch to 0
Changing fmc1_bert2_latch to 1
Changing fmc1_bert2_latch to 0
Changing fmc1_bert1_latch to 1
Changing fmc1_bert1_latch to 0
Changing fmc1_bert2_latch to 1
Changing fmc1_bert2_latch to 0
Changing fmc1_bert1_latch to 1
Changing fmc1_bert1_latch to 0
Changing fmc1_bert2_latch to 1
Changing fmc1_bert2_latch to 0
Changing fmc1_bert1_latch to 1
Changing fmc1_bert1_latch to 0
Changing fmc1_bert2_latch to 1
Changing fmc1_bert2_latch to 0
Changing fmc1_bert1_latch to 1
Changing fmc1_bert1_latch to 0
Changing fmc1_bert2_latch to 1
Changing fmc1_bert2_latch to 0
Changing fmc1_bert1_latch to 1
Changing fmc1_bert1_latch to 0
Changing fmc1_bert2_latch to 1
Changing fmc1_bert2_latch to 0
```

Figure 35: Debug panel in relation with User tab