JRA1 Telescope: NI Flex RIO DAQ

Labview Telescope DAQ demonstration software overview

G. Claus[1], Mathieu Goffe[1], Kimmo Jaaskelainen[1], Cayetano Santos[1], Matthieu Specht[1]

January 17, 2011

Abstract

The EUDET JRA1 Pixel Telescope is using a custom-made data acquisition system since a couple of years. In preparation for AIDA, the group decided to investigate different off the shelf I/O systems. The advantage of such a system is the easier support and the availability over the next years. The IPHC group selected the NI Flex Rio system and prepared LabView sources, which can rather easy be connected to the existing DAQ. In this memo the Labview telescope DAQ demonstration software is documented.

[1] IPHC, Strasbourg, France

Inhaltsverzeichnis

1 Introduction

The telescope DAQ software is a Labview application developed under Labview 2009. Labview is used for GUI and Flex RIO board driver and as a kind " top level software " responsible of the management of DAQ operation. The JTAG configuration, run configuration, data processing (to extract frames with trigger), saving data to disk, are written in C and C++ (eudet_frio library) and compiled in a dll named eudet_frio_dll.dll.

The interface between telescope DAQ software and EUDET DAQ software via Ethernet can be written in this DLL. Because it may be easier to write it in C rather than in Labview graphical language.

But, debugging this interface with the whole DAQ chain and moreover compiled in a DLL may be difficult, that's why a DAQ emulator has been developed. It has roughly the same functionalities as the DAQ. It's a C++ Builder application (no Labview code) which see the eudet_frio library as a part of it's source code (not as a DLL) therefore you can use Borland's debugger if needed. Of course you don't need the hardware to run this application.

2 How to compile the DLL

2.1 Introduction

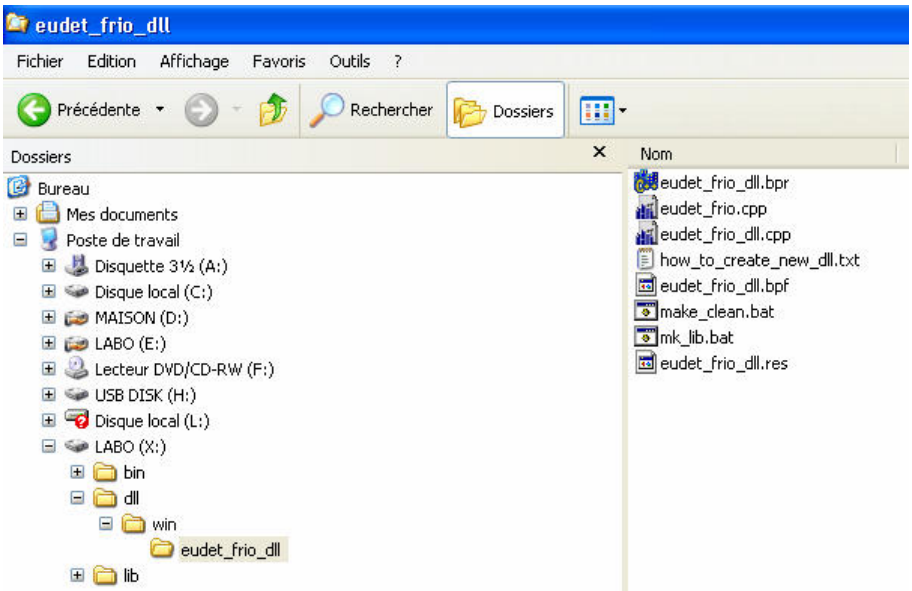
The DLL source code is a part of the C source architecture installed on the host PC. Therefore the DLL will be compiled on this PC and the binary file (*.DLL) will be copied on the PXIe crate afterward.

If the source are not installed on the PC, please follow the procedure described in the document " 2__c_source_arch.pdf".

If the source are installed, please don't forget to execute ch_prod.bat in order to create the virtual drives X:, Y: and L:

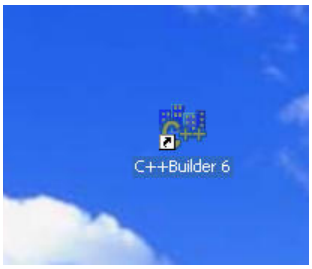
2.2 DLL project directory

The DLL project is in directory x:\dll\win\eudet_frio_dll

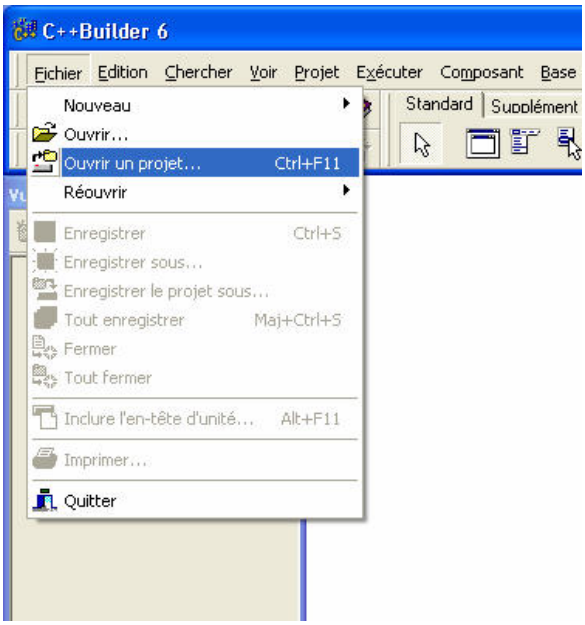


2.3 Compiling the DLL

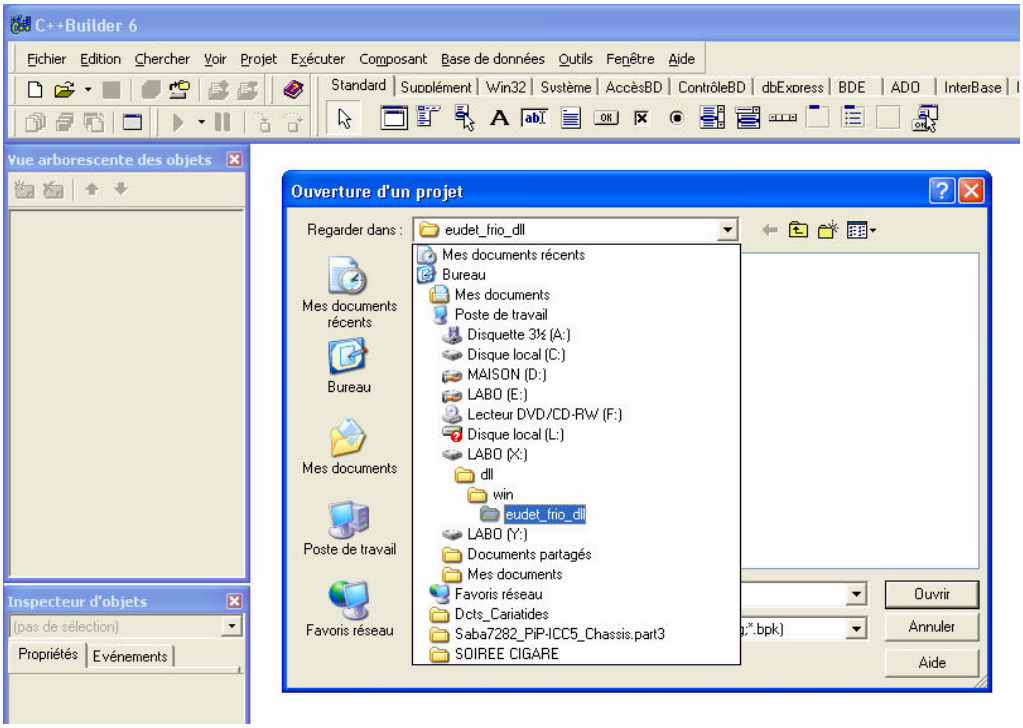
Launch C++ Builder by a click on its desktop icon



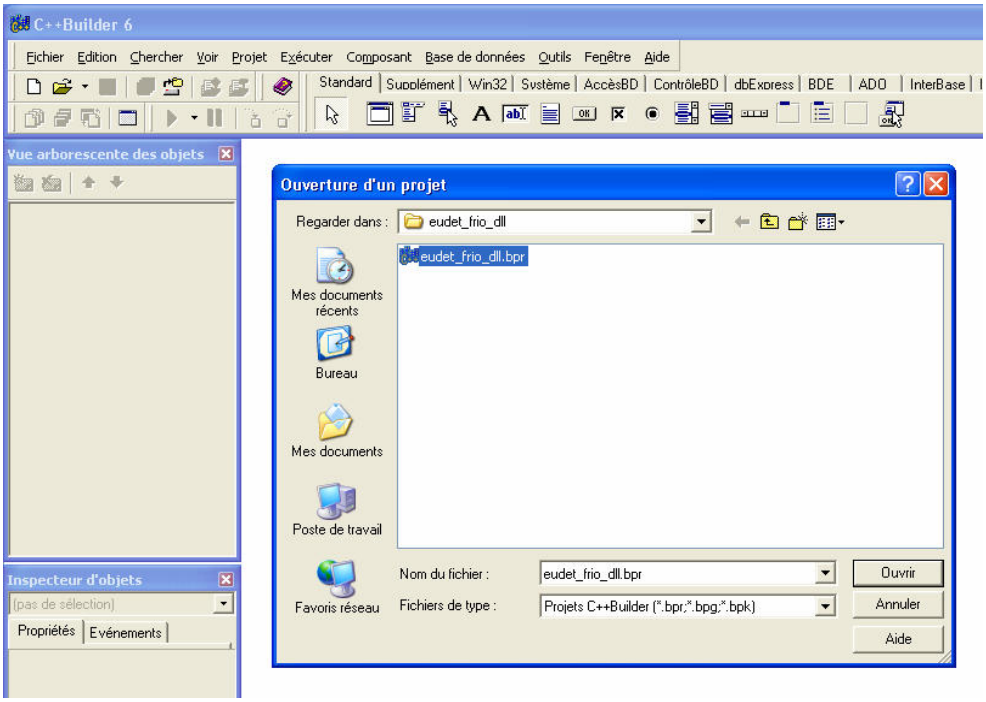
Menu “ File” – “ Open project ”



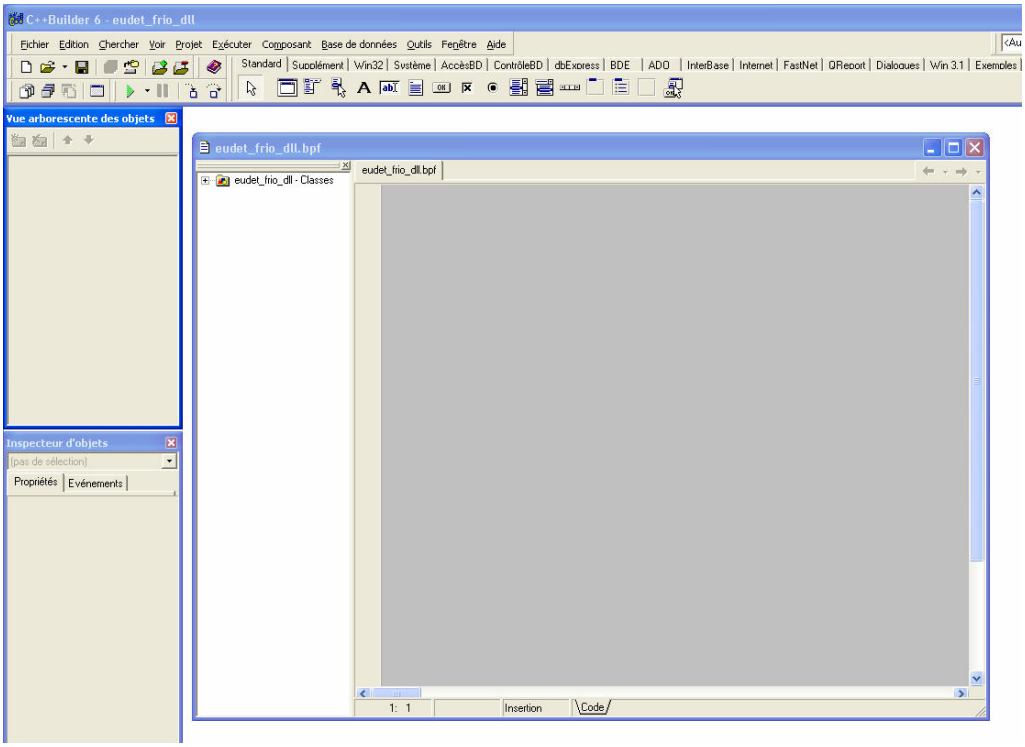
Open the directory x:\dll\win\eudet_frio_dll



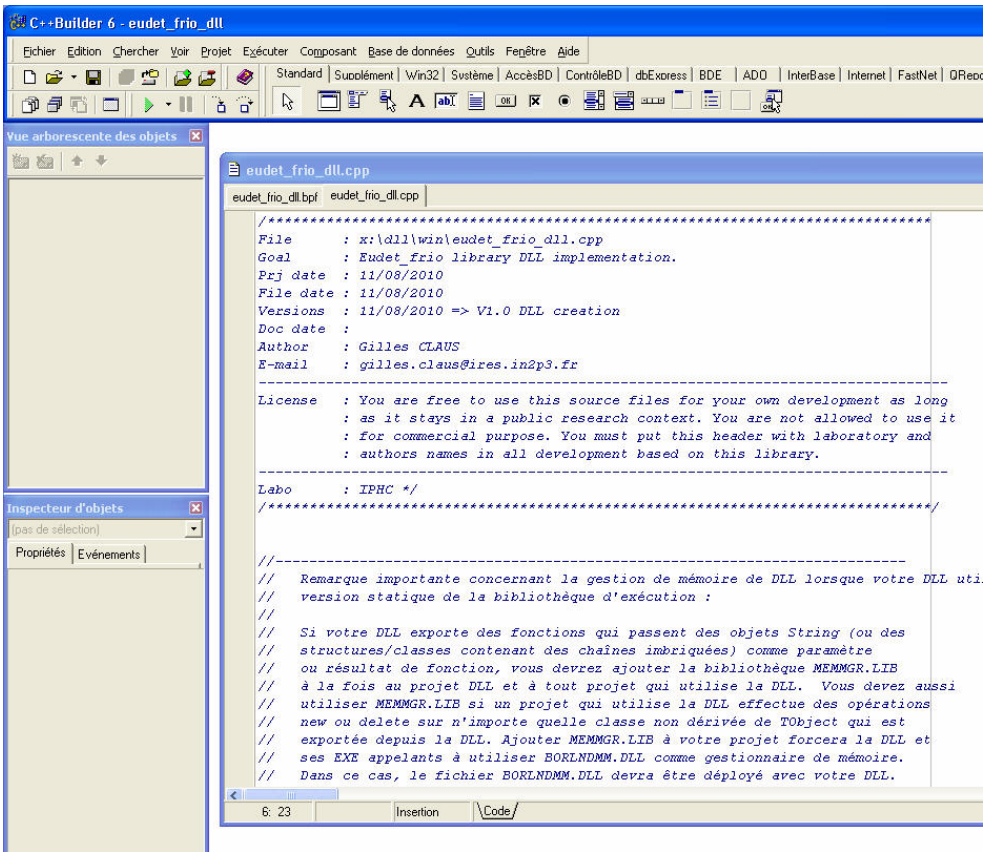
Open the project file eudet_frio_dll.bpr



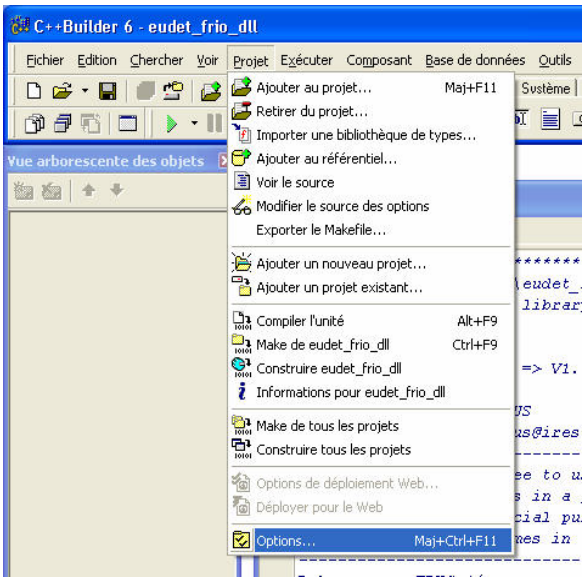
You will get this window



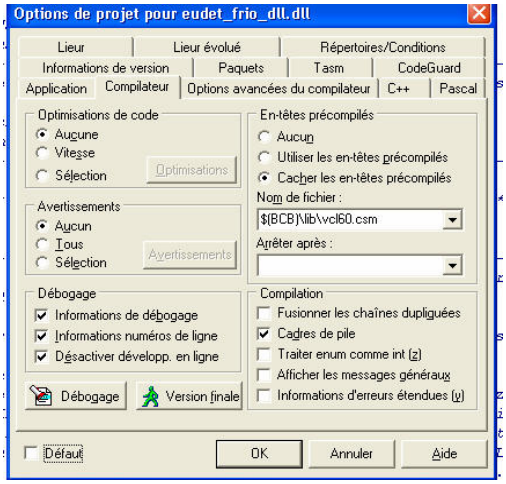
This is the project files list : *.bpf and *.cpp which includes all source files.



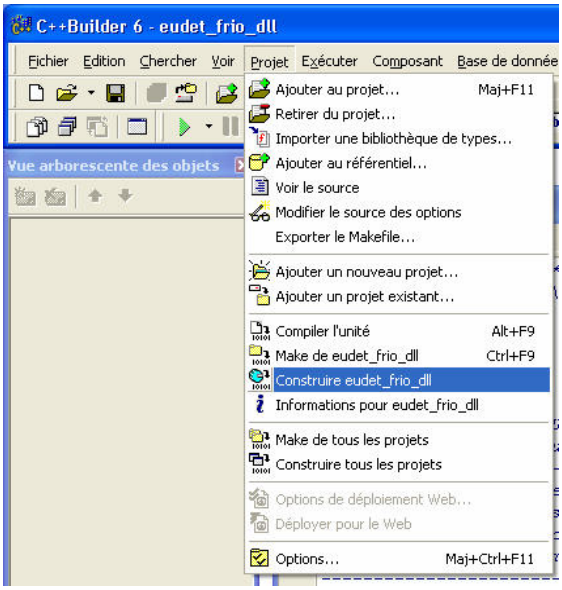
Open project options → Menu " Project " – " Options "



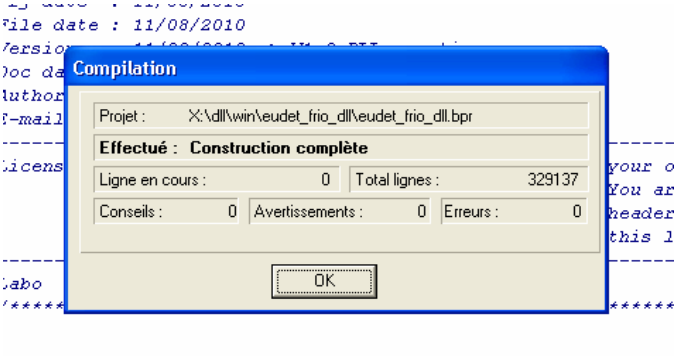
Panel compiler, disable warnings → sub panel " Warnings" – " None "



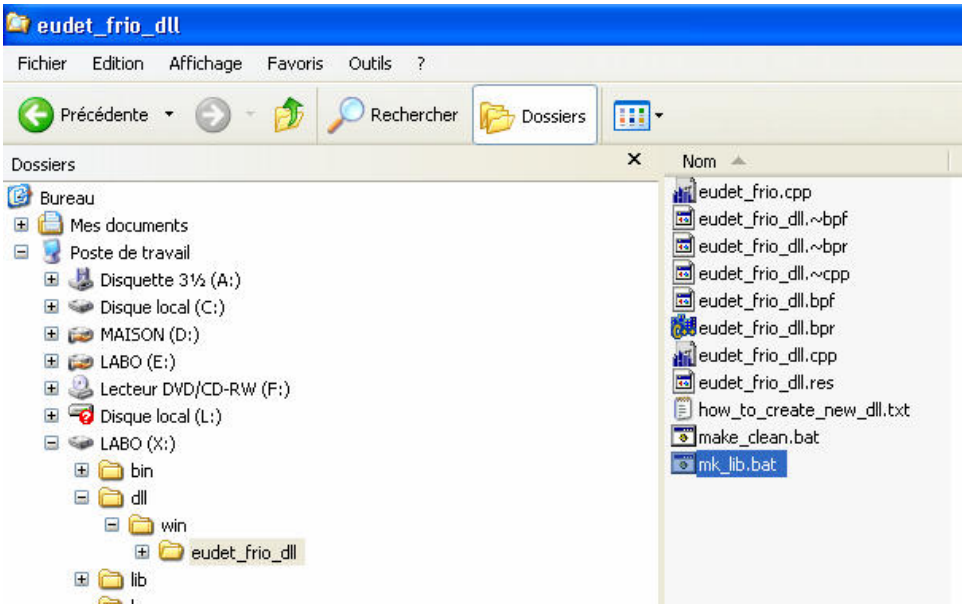
Compile DLL → Menu " Project " – " Build eudet_frio_dll "



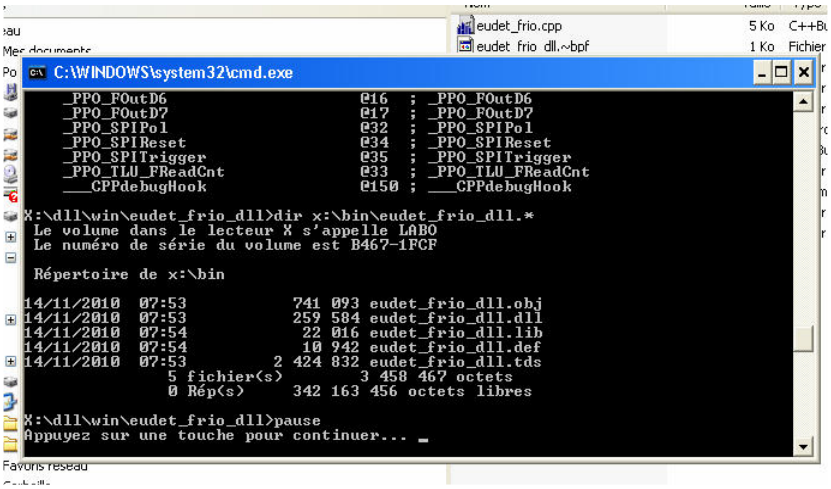
You should get no errors as compilation result



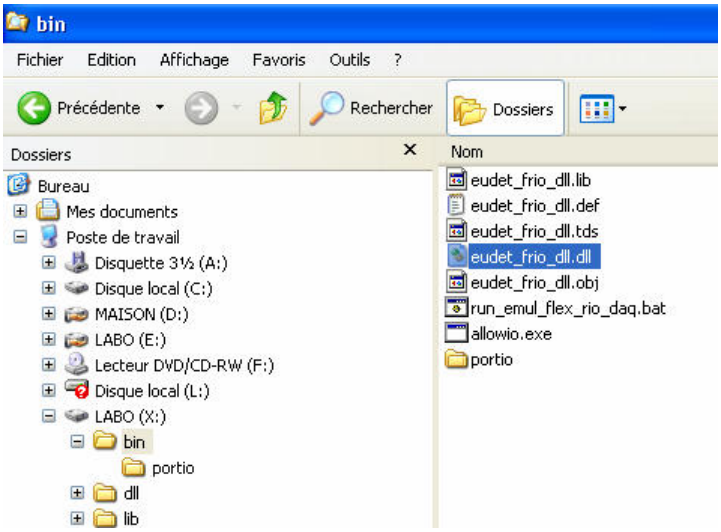
Now execute mk_lib.bat which is located in x:\dll\win\eudet_frio_dll. It will create DLL interface files which may be needed in certain cases.



A dos window shell will pop-up, close it when execution is finished



The DLL files are created in directory x:\bin

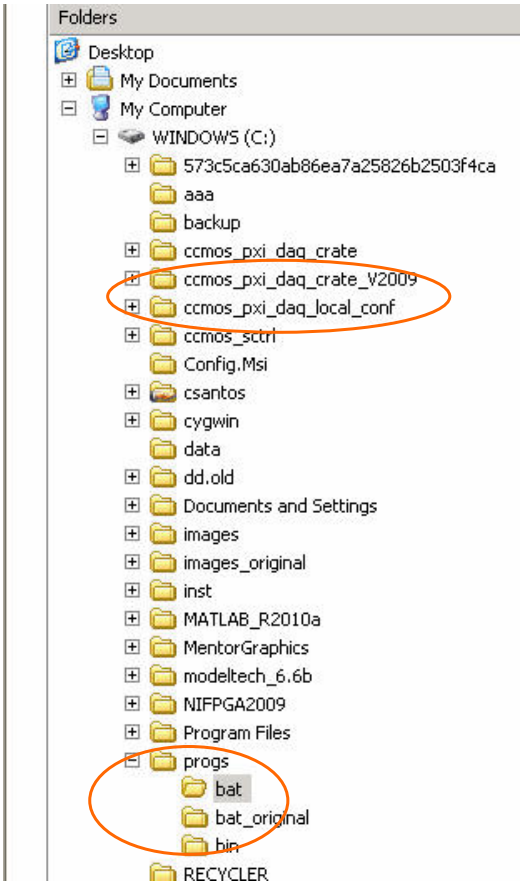


3 DAQ sources (Labview) installation on PXIe carte

This document will not cover source files installation on PXIe crate, this section will be written later. This chapter will just list things in order to show you where they are installed.

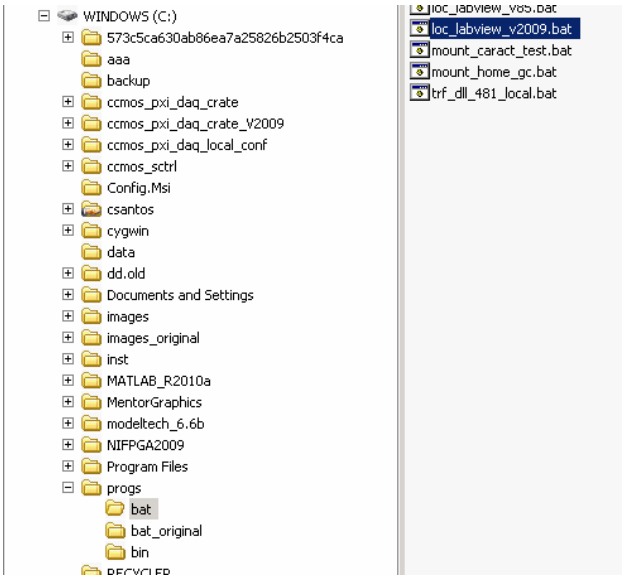
Three directories are needed + the firmware installation directory

- **C:\progs**
- **C:\ccmos_pxi_daq_crate_v2009**
- **C:\ccmos_pxi_daq_local_conf**

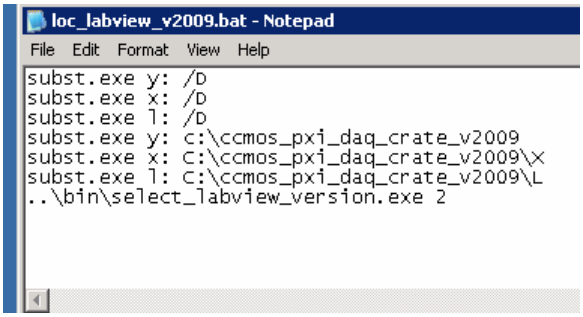


First of all you must configure system, this is done by the batch file " load_labview_v2009.bat ". You can start it by a click on its desktop icon.

This file is located in C:\progs\bat

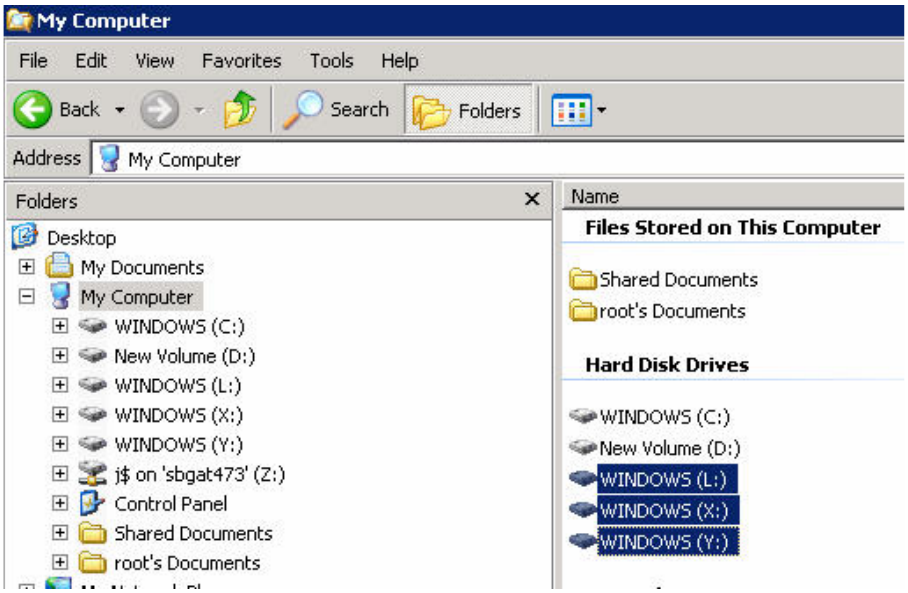


You should not need to modify it, but in case you can edit it.



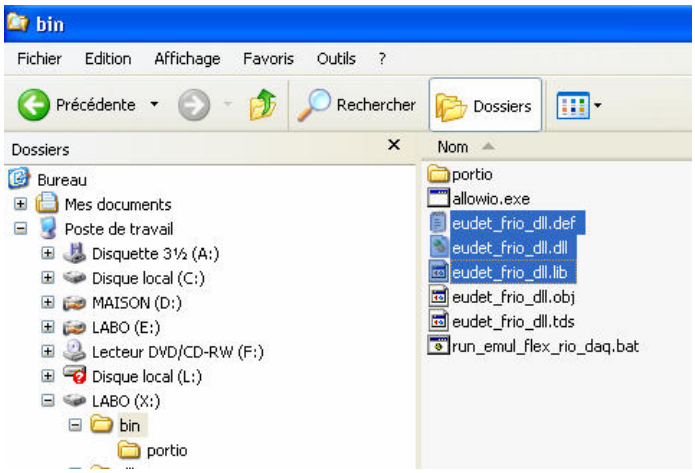
The following virtual drives will be created :

- **Y: → root of the whole source tree**
- **X: → root of C, C++ source code tree**
- **L: → root of Labview source code tree**



4 DLL copy from host PC to PXIe carte

We always compile the DLL on the host PC, not on the PXIe crate, because C++ Builder is installed on the PC not on the crate. Therefore we must copy DLL binary files from the directory x:\bin of the PC to the directory x:\bin of the crate. This is not a huge task as the crate can " mount " the PC disk, and it may be automated via a batch file.



There are three files to copy :

- **eudet_frio_dll.def**
- **eudet_frio_dll.lib**
- **eudet_frio_dll.dll**

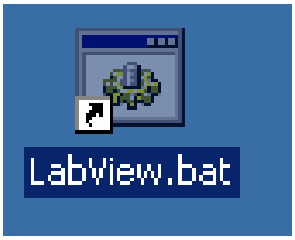
I am not sure that all three are needed for Labview, but I didn't found the time to check, therefore please copy all of them to avoid problems and loose time.

5 DAQ demonstration

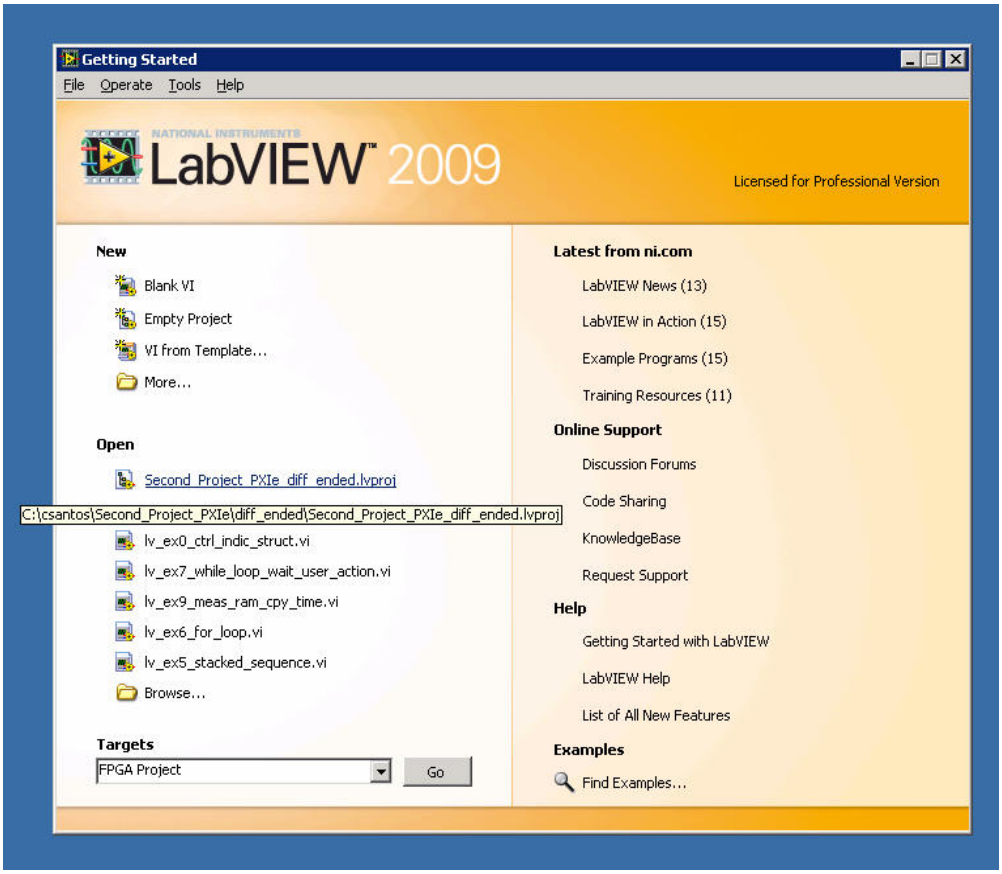
5.1 How to start Labview & load project

Execute " loc_labview_v2009.bat " if it's not already done. You need to do it only one time after logging on the carte.

Start Labview via the batch file " Labview.bat ", because it encapsulates the parallel port driver we need (this batch must be installed in Labview bin directory)



The Labview window shows up, select "Second_Project_PXIe_diff_ended.lvproj"

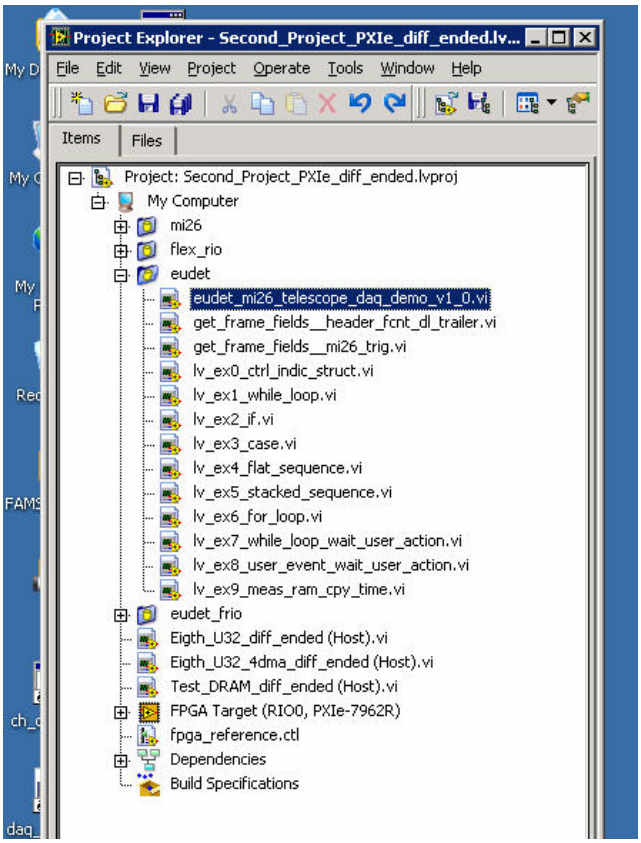


WARNING !

The project file path has changed, it's not the one displayed on the above screen shot, now it is

C:\flexrio_mi26_fw\14_december_2010\lv_2009
project_pxie_diff_ended\flexrio_mi26_lv2009_pxie_diff_ended.lvproj

The " Project Explorer " window will appear, but you will not get access to GUI immediately, it will take some time … please wait, that's the only thing you can do …



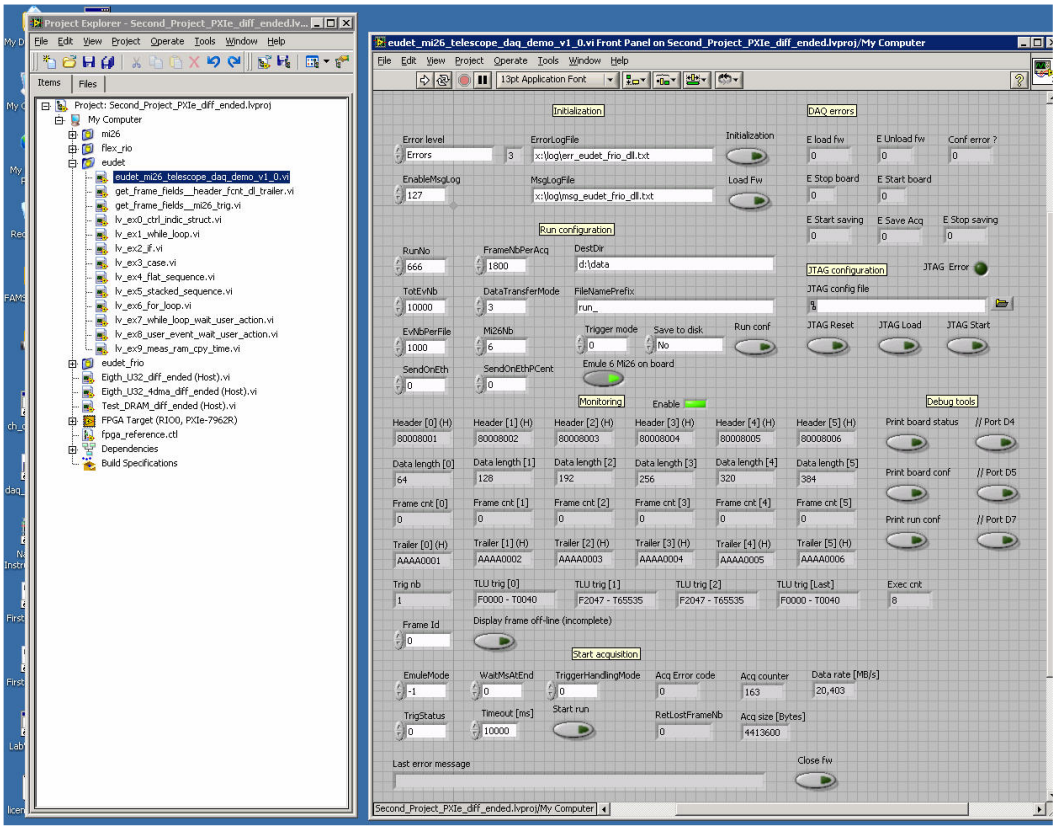
Now, select the file " eudet_mi26_telescope_daq_demo_v1v0.vi ".

WARNING !

The application file name has changed, it's not the one displayed on the above screen shot, now it is :

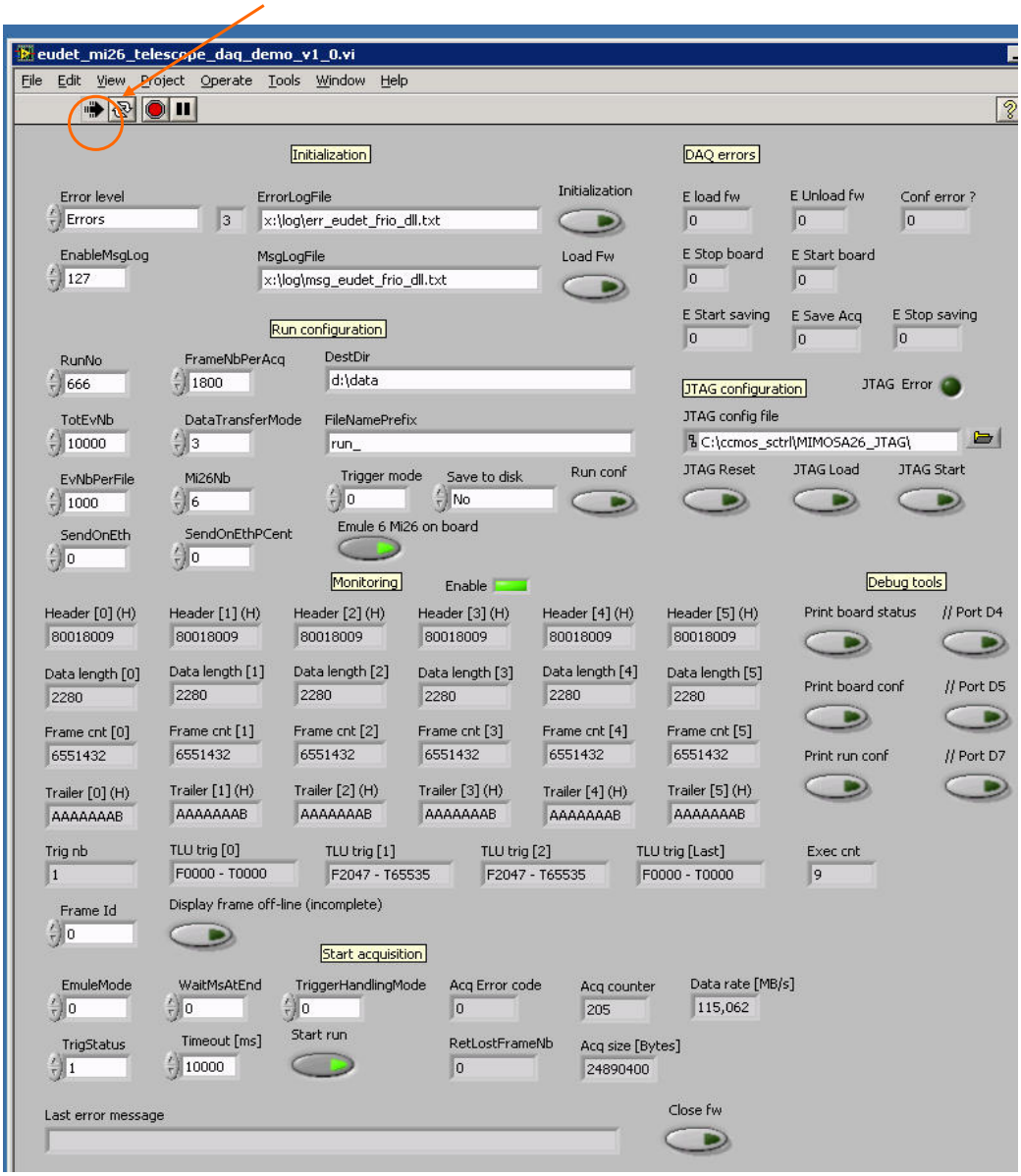
→ eudet_mi26_telescope_daq_demo_v1_1.vi

The DAQ demo GUI windows should appear.

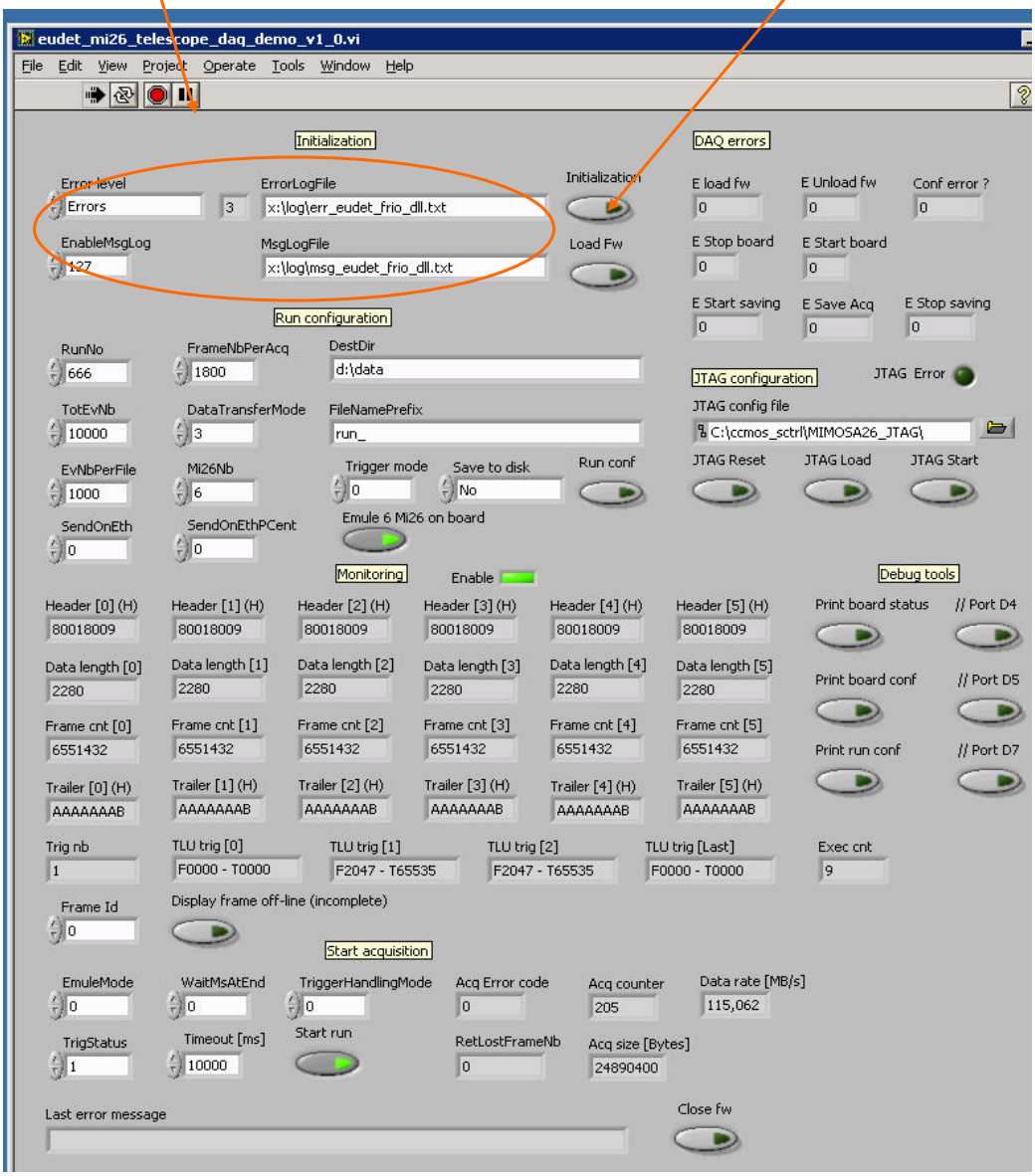


5.2 GUI overview

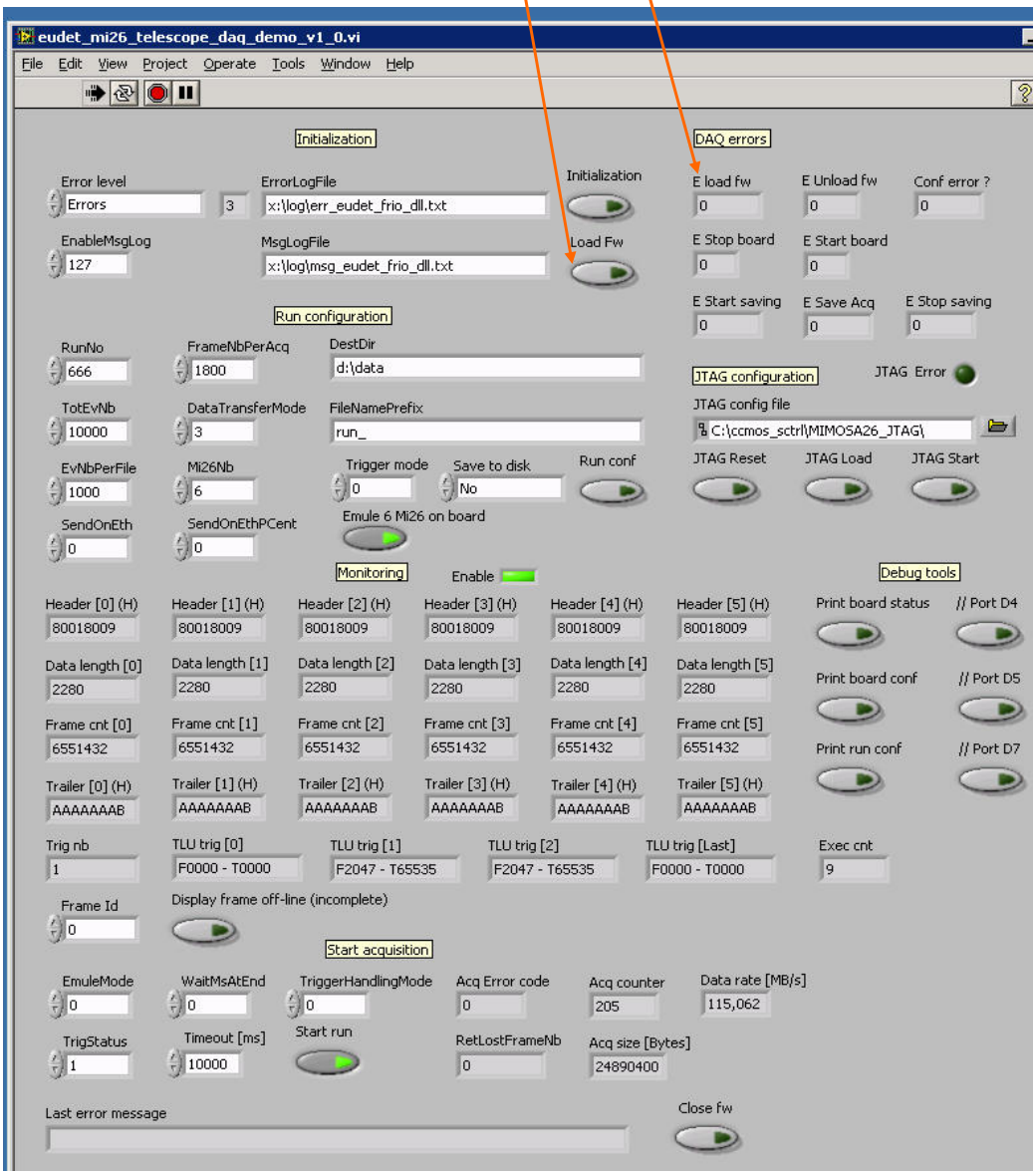
Click on the " black arrow " to start the software.



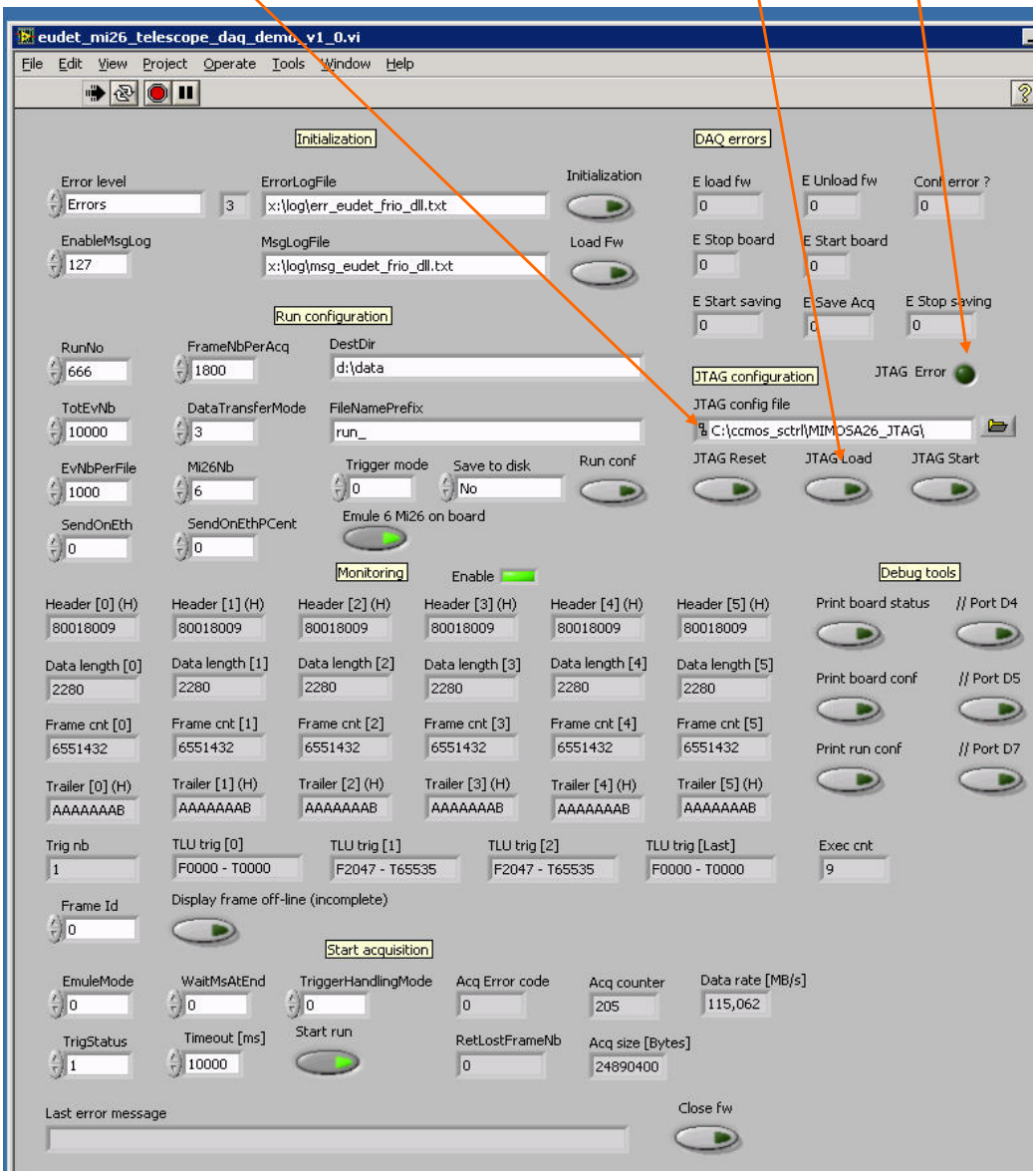
Errors and messages log files, and log level can be configured here. After setting errors and messages, click on initialization button.



Load the firmware by a click on the "Load Fw " button, if the operation failed an error code (value < 0) will be displayed in indicator " E load fw "

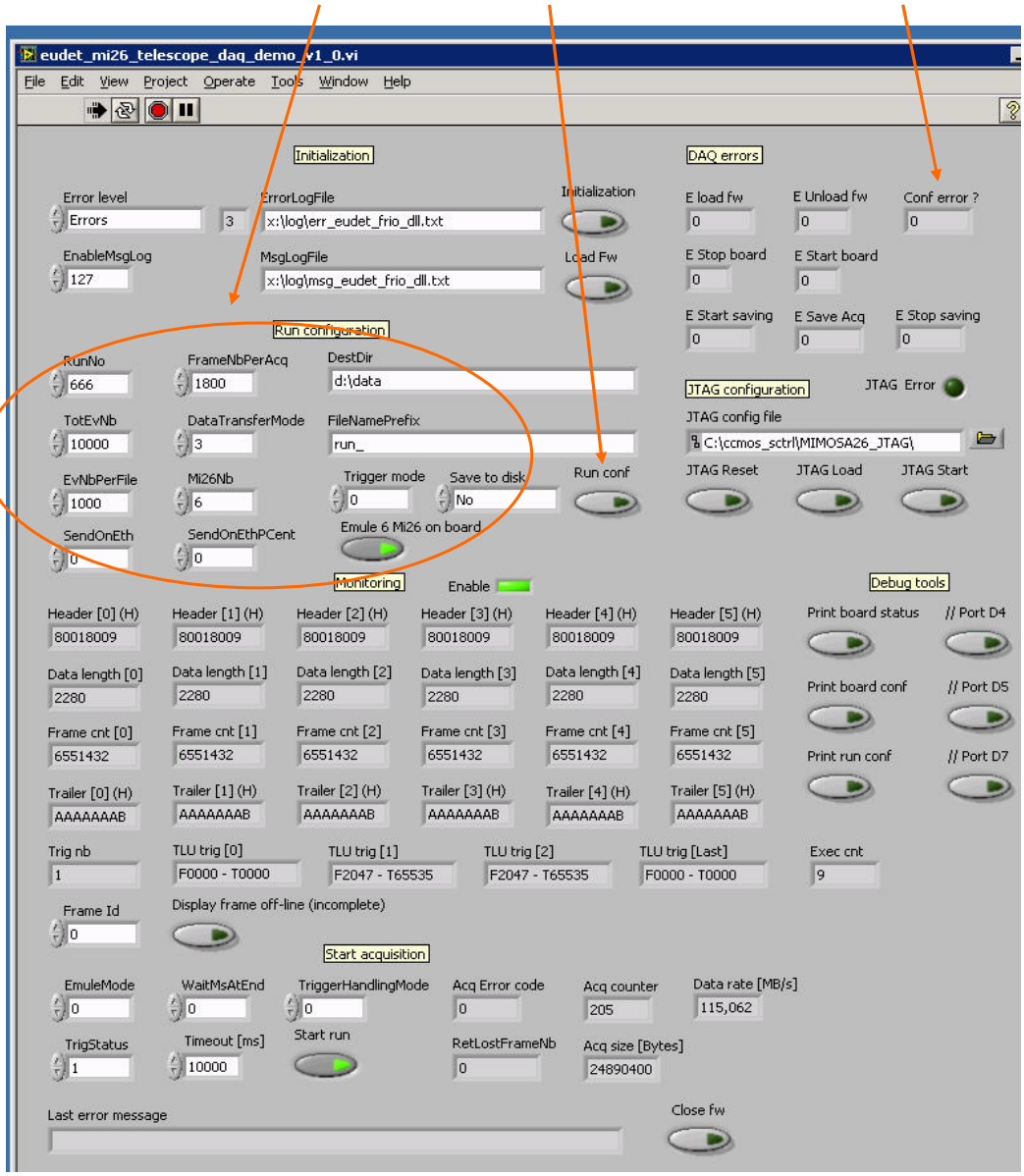


Select JTAG file, configure Mimosa 26 by a click on "JTAG load", the "JTAG error " led will become red in case of configuration error.

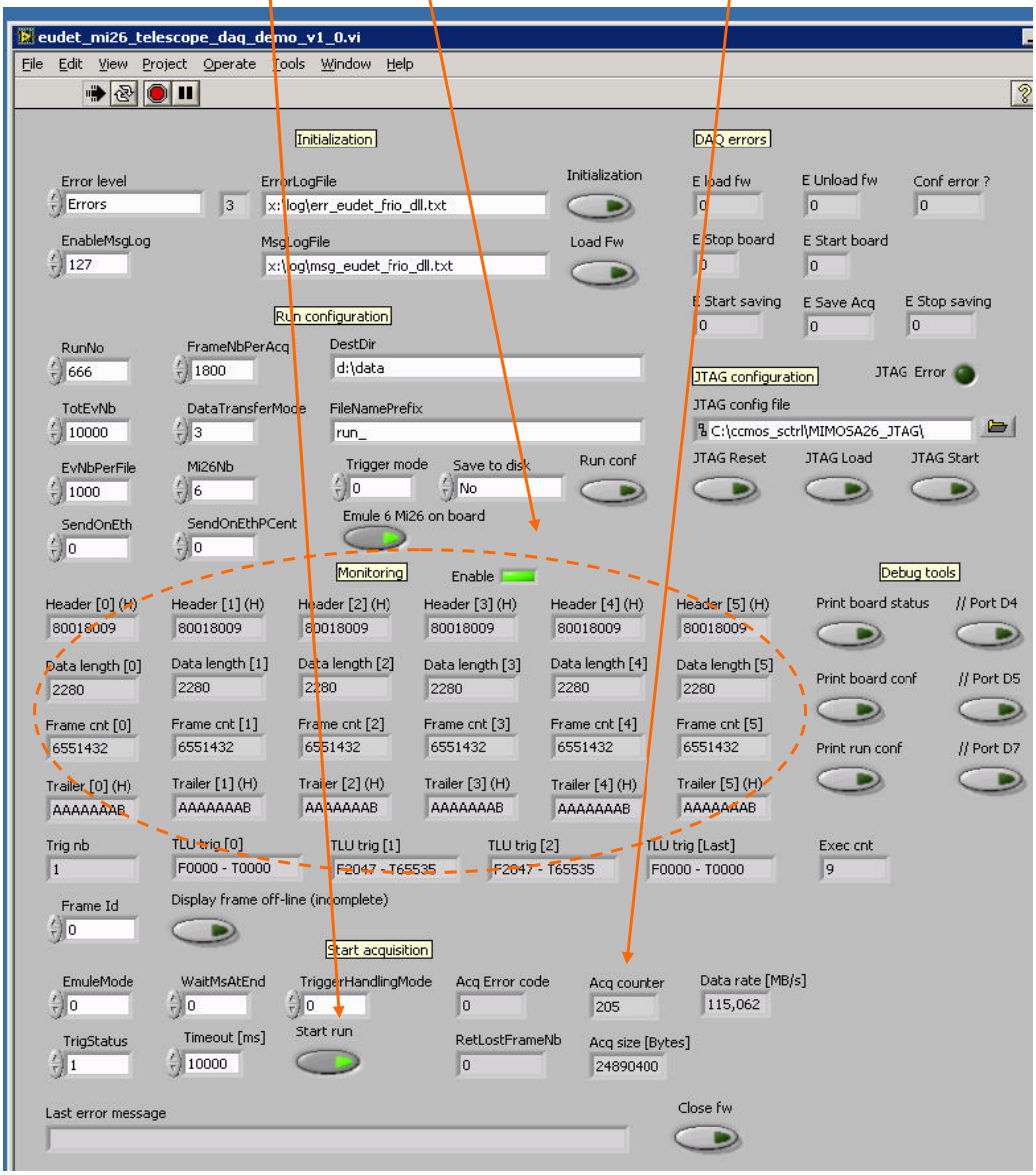


The default JTAG file to load to test DAQ is : daq_test_2x80MHz_6_chip.mcf.

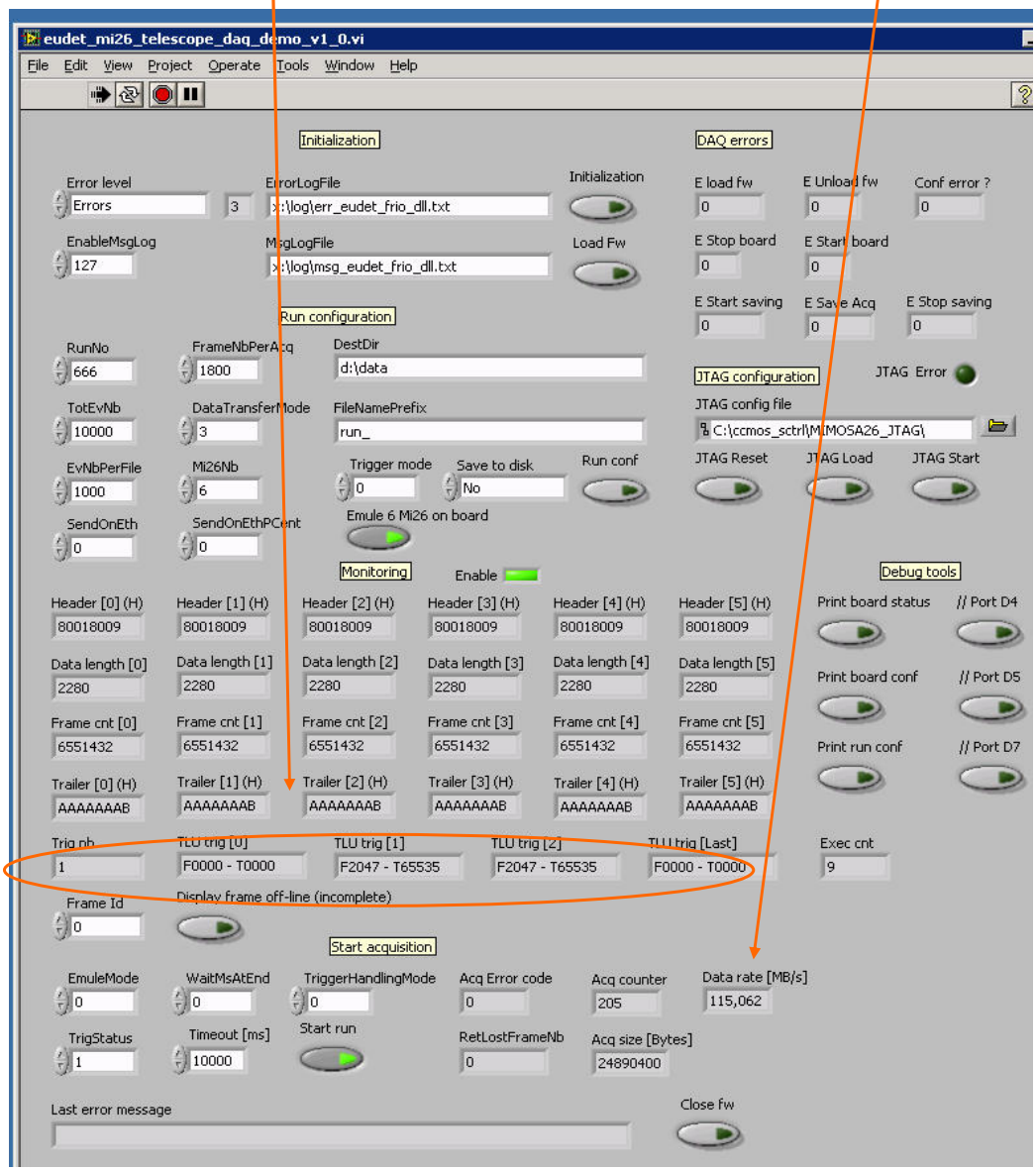
Configure run parameters, click on " Run conf ", error displayed in "Conf error ?".



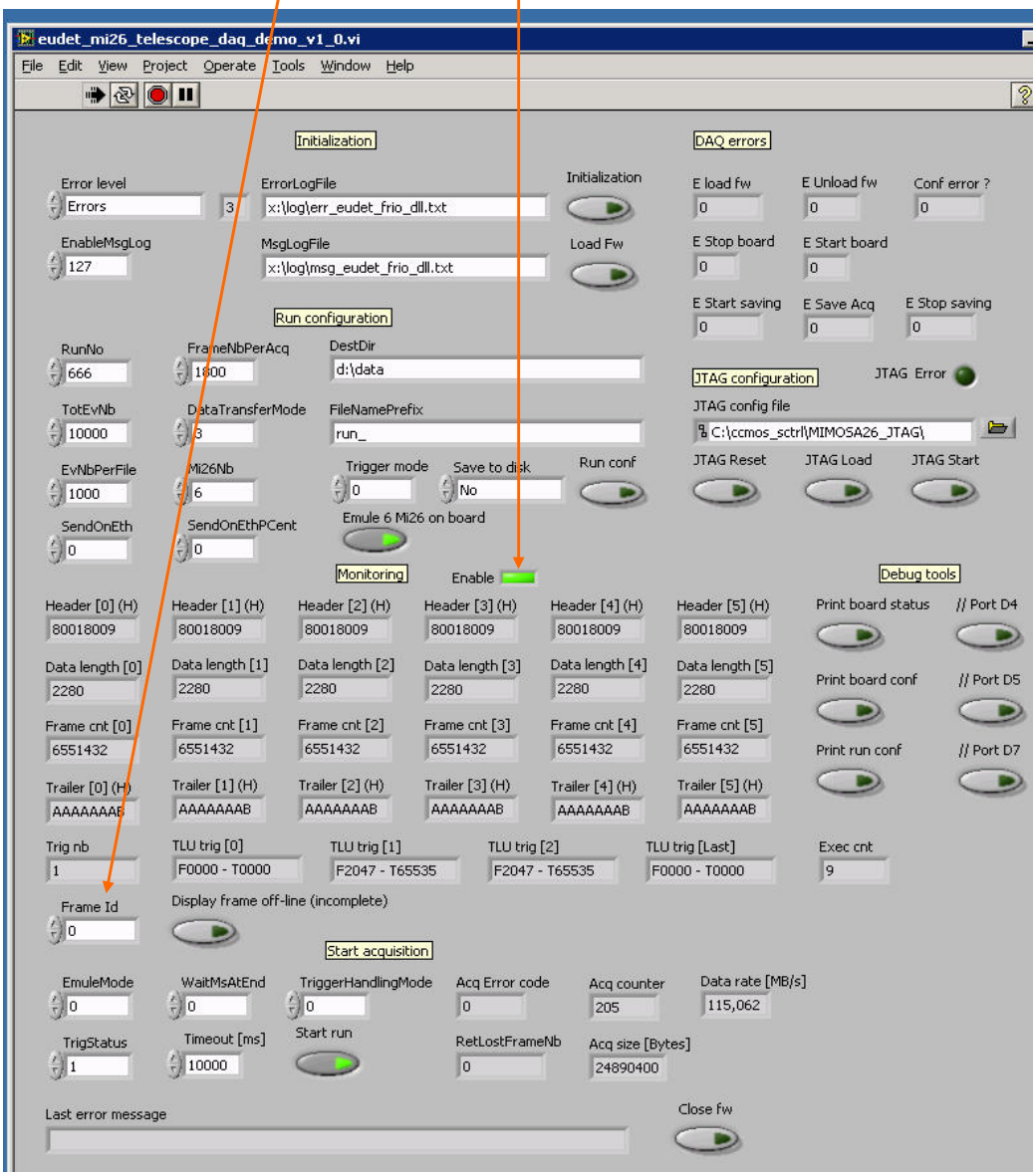
Click on the " Start run " to start the acquisition, " Acq counter " should increase and the values of header, data length, … trailer will be displayed here.



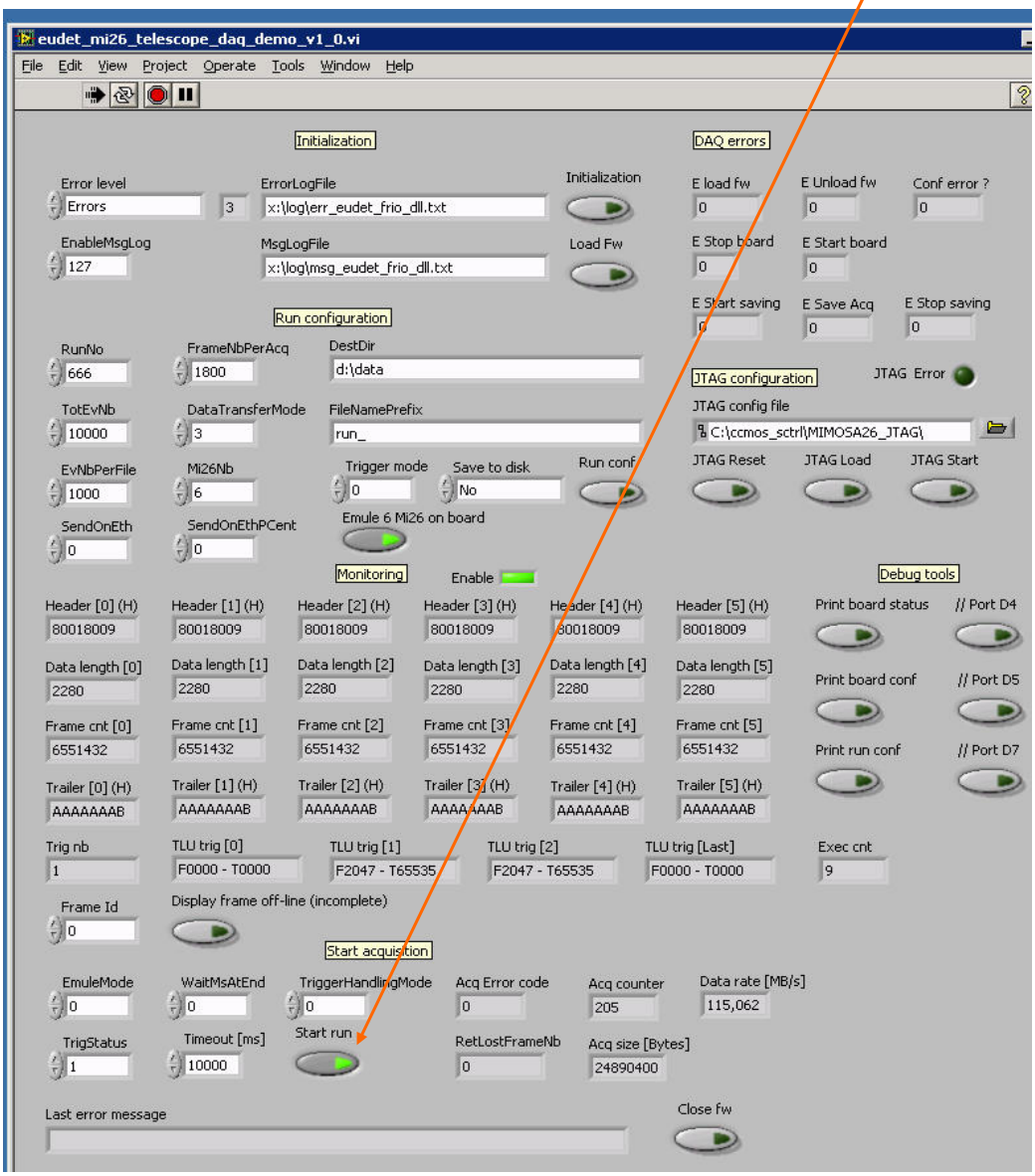
Triggers number and trigger values are displayed here, an evaluation of the data stream rate in MB/s is also calculated on-line by averaging of the last 10 acquisitions.



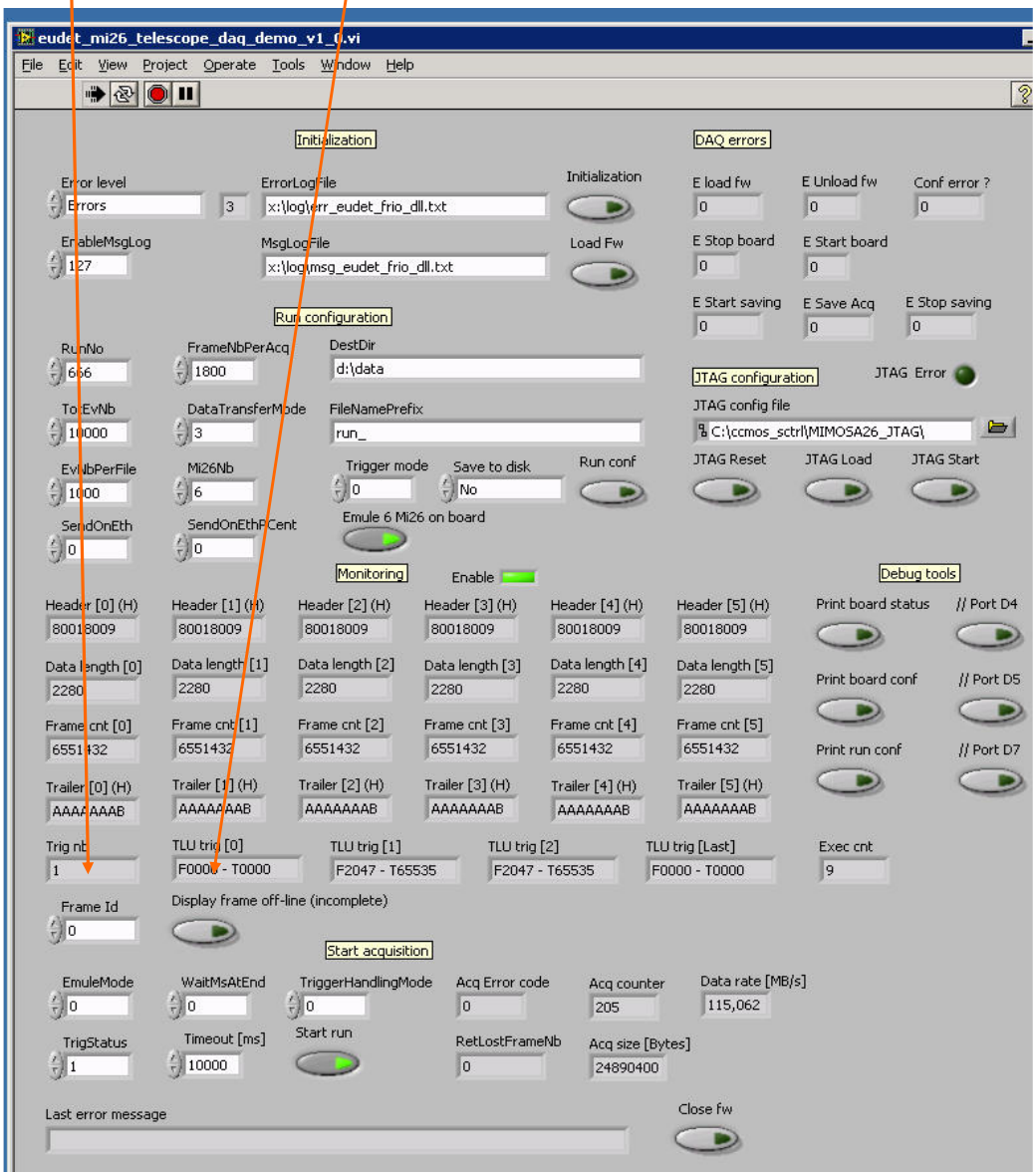
The frame displayed on-line is the one selected by " Frame Id". This on-line monitoring can be disabled by a click on " Enable " control.



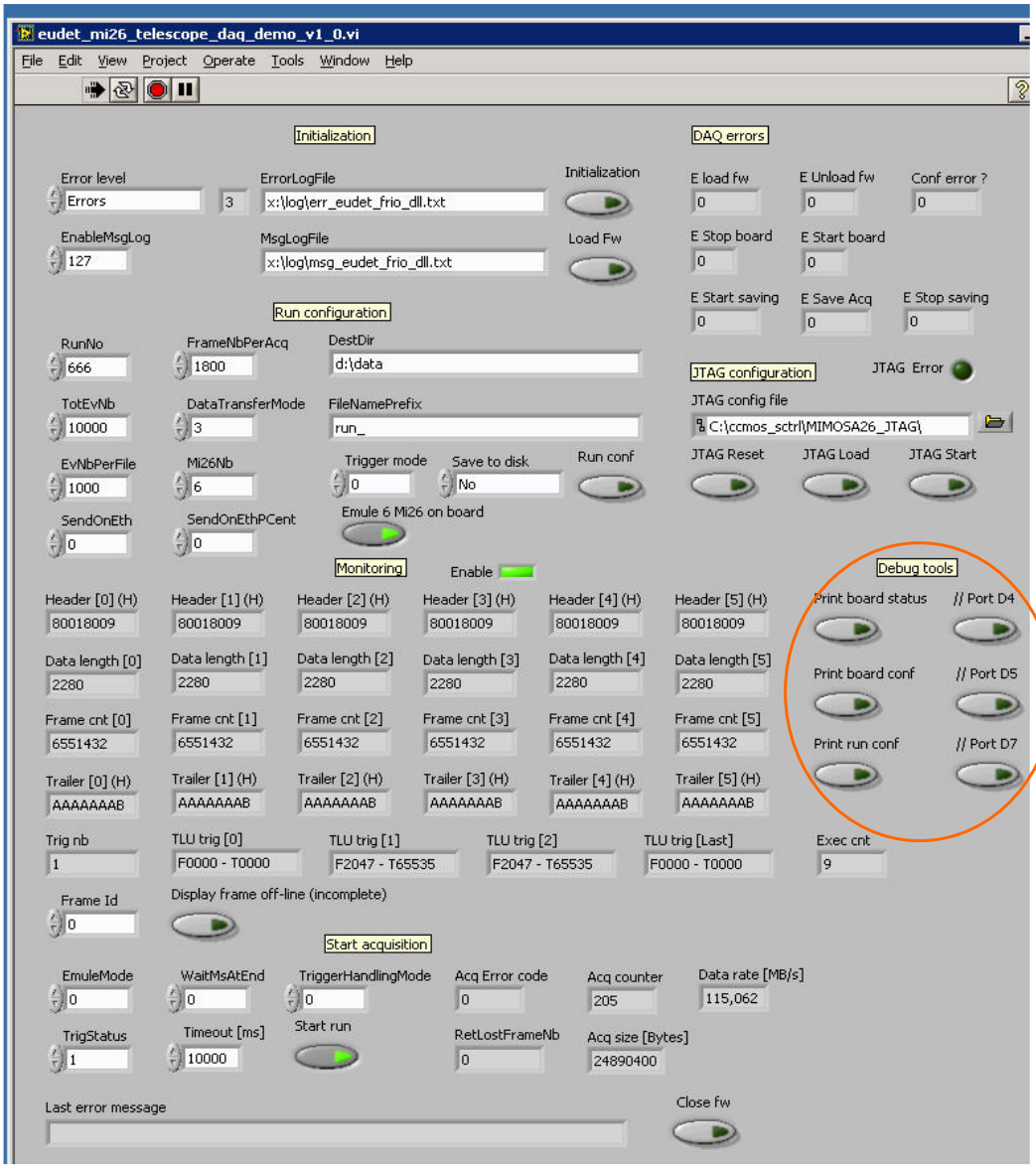
While acquisition is running the " Start run ".button is green, click again on it in order to stop the acquisition.



The frames can also been displayed off-line (DAQ stopped), select the "Frame Id".and click on " Display frame …". WARNING : Only the frame counter will be displayed, because this code is not finished → the user can do it as an exercise ;-).



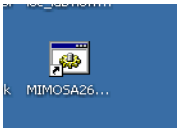
Some debug tools are also provided : print the context record in log file and parallel port lines control.



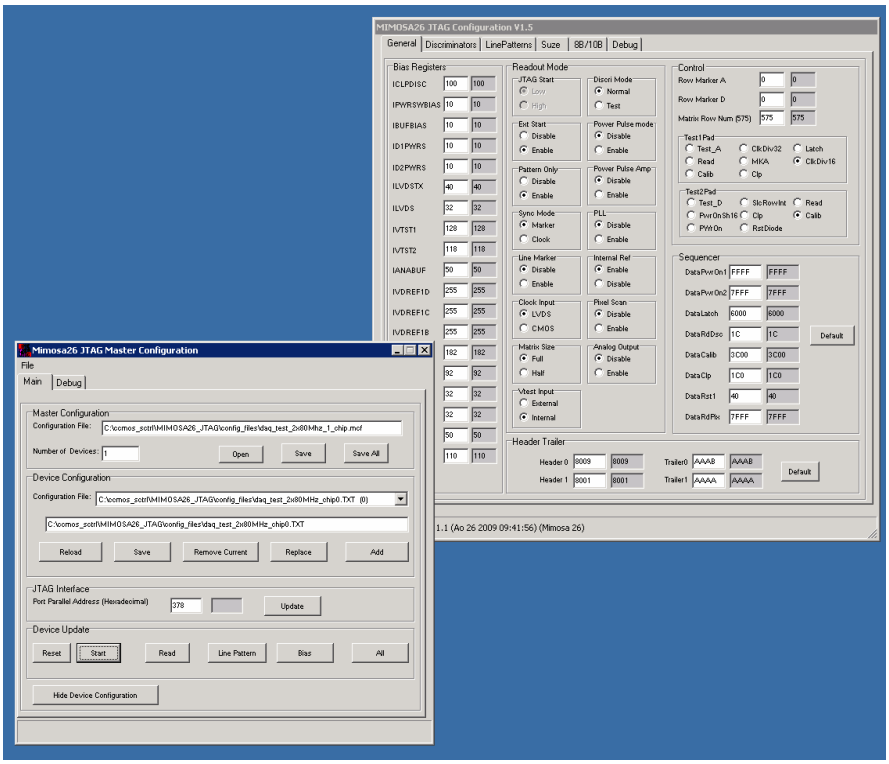
5.3 How to configure JTAG

First of all, launch the JTAG software. The DAQ application can do it automatically, but not for all versions of JTAG, therefore please do it manually.

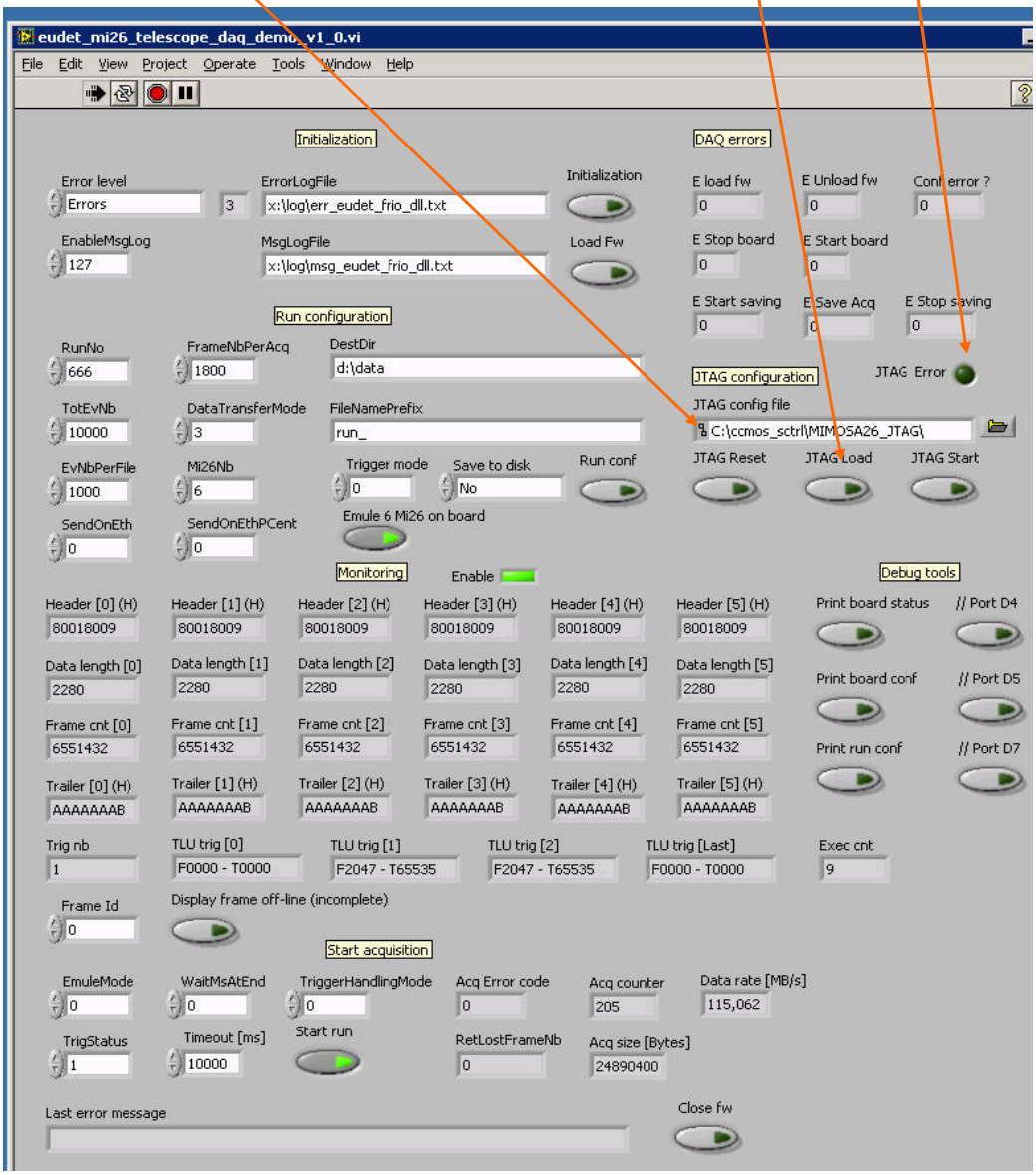
Click on the desktop " Mimosa 26 JTAG " icon.



The following windows will appear, you don't need to load any file, just start the software, that's all.



Select JTAG file, configure Mimosa 26 by a click on "JTAG load", the "JTAG error " led will become red in case of configuration error.

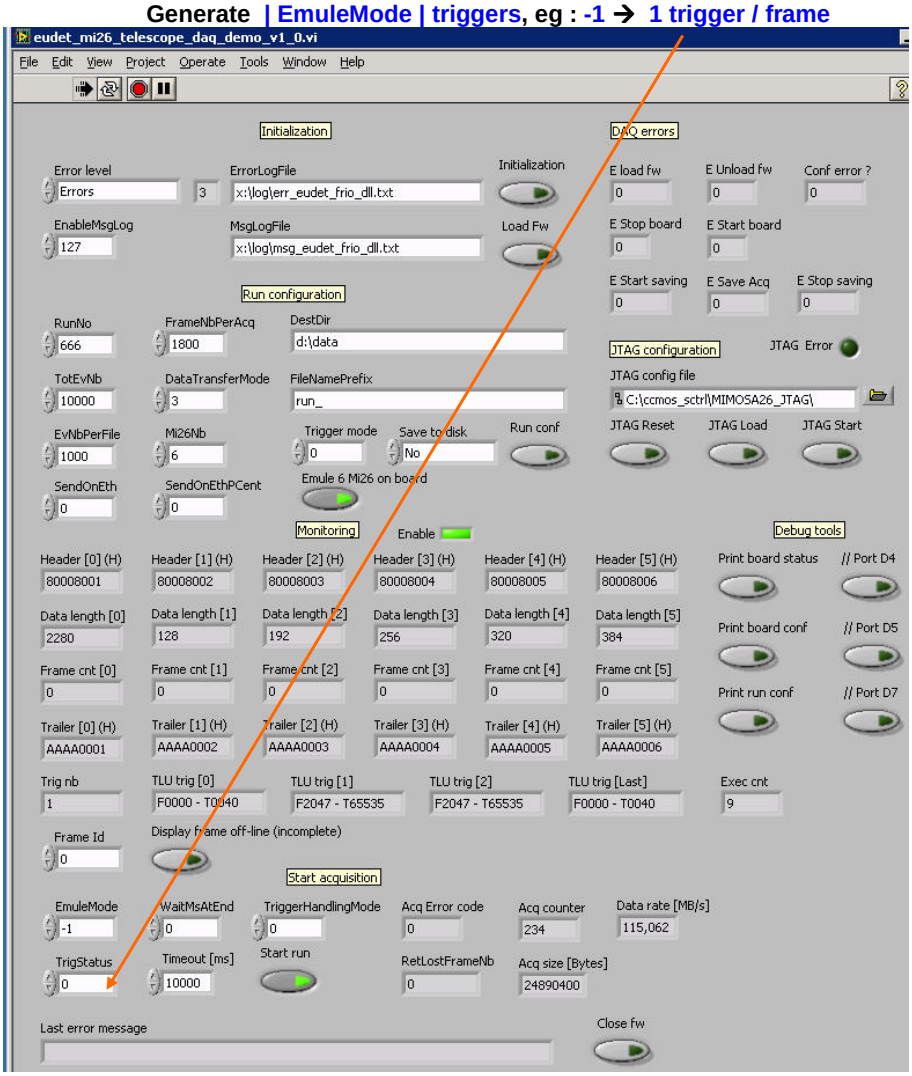


5.4 How to configure emulation modes

The data emulation is controlled by the fields " EmuleMod ", "TrigStatus" and "Emule 6 Mi26 on board".

The "EmuleMode" control :

- **= 0 → No data emulation → DAQ provides Telescope data**
- **= 1 → Telescope data overwritten by emulated data – No trigger**
- **< 0 → Telescope data overwritten by emulated data + Trigger(s) Generate | EmuleMode | triggers, eg : -1 → 1 trigger / frame**

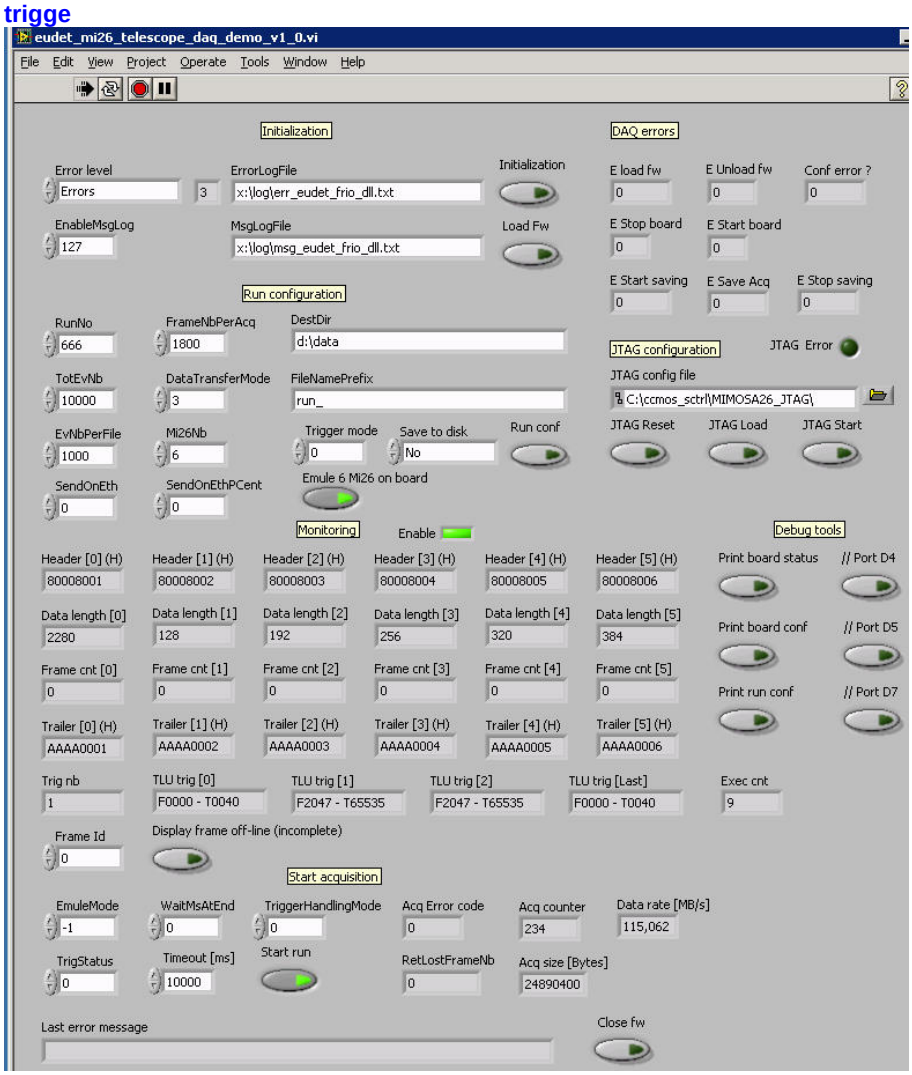


The “TrigStatus” control :

- **= 0 → DAQ provides Telescope data – No trigger**
- **> 0 → Overwrite the trigger info from Flex RIO but NOT the data**
  Generate “ TrigStatus ” triggers per frame
  It’s a way to force trigger number given by board

The control “ TrigStatus ” has priority on “ EmuleMode ”, because it’s the last one processed by software. For example, if “ TrigStatus ” = 3 and “ EmuleMode ” = -1, telescope data will overwritten by emulated data (rs will be emulated not 1.EmuleMode <> 0) BUT 3

trigge



Control “Emule 6 Mi26 on board” → see 5.5.1 Introduction.

5.5 Running the DAQ software

5.5.1 Introduction

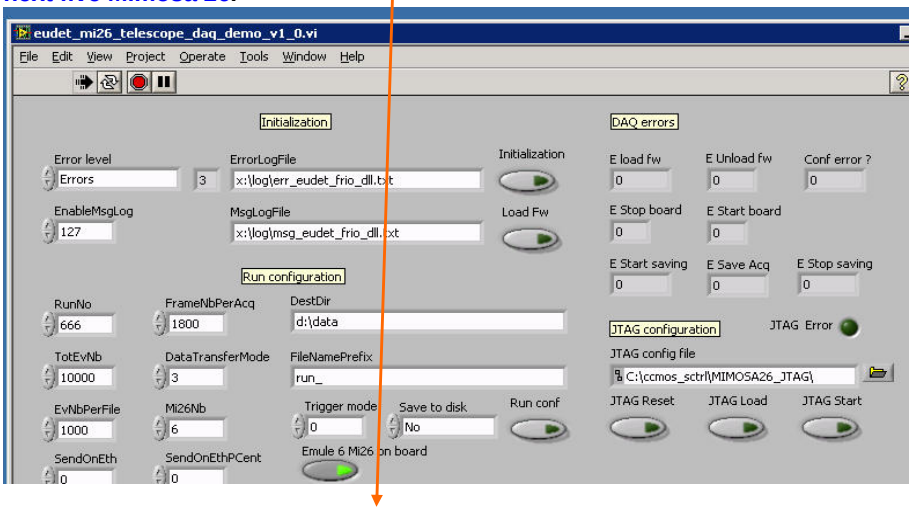
The DAQ software has, like the emulator, four modes to read data, selection is done via the control " DataTransferMode" :

- **0 → IPHC**
- **1 → EUDET 1**
- **2 → EUDET 2**
- **3 → EUDET 3**

Please read the DAQ emulator documentation to learn more about theses modes.

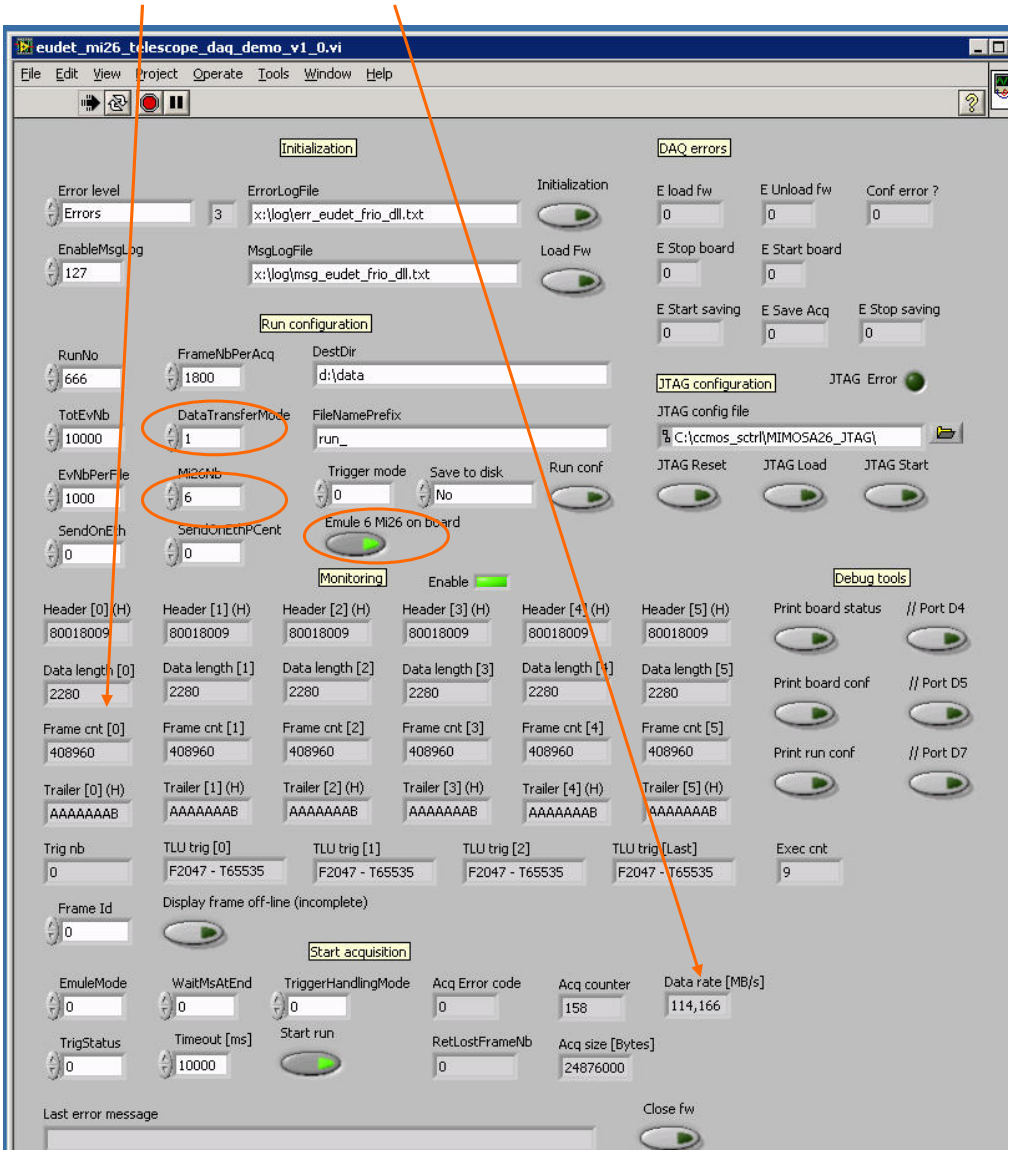
The DAQ software can also emulate data but its functionalities are limited compared to the DAQ emulator. The header, trailer, trigger values are hard coded in emulation functions, they are not configurable from GUI. Nevertheless it can emulate Mimosa 26 data and especially triggers, their number can be configured from GUI.

The DAQ also have an option to duplicate Mimosa 26 data, because sometimes it difficult to keep a system for week with 6 Mimosa 26 installed on it … This option is enabled by the control " Emule 6 Mi26 on board", in this case only one Mimosa 26 is needed, connected to the first pair of links (D00, D01), a copy of his data stream will be done in memory part reserved for the next five Mimosa 26.



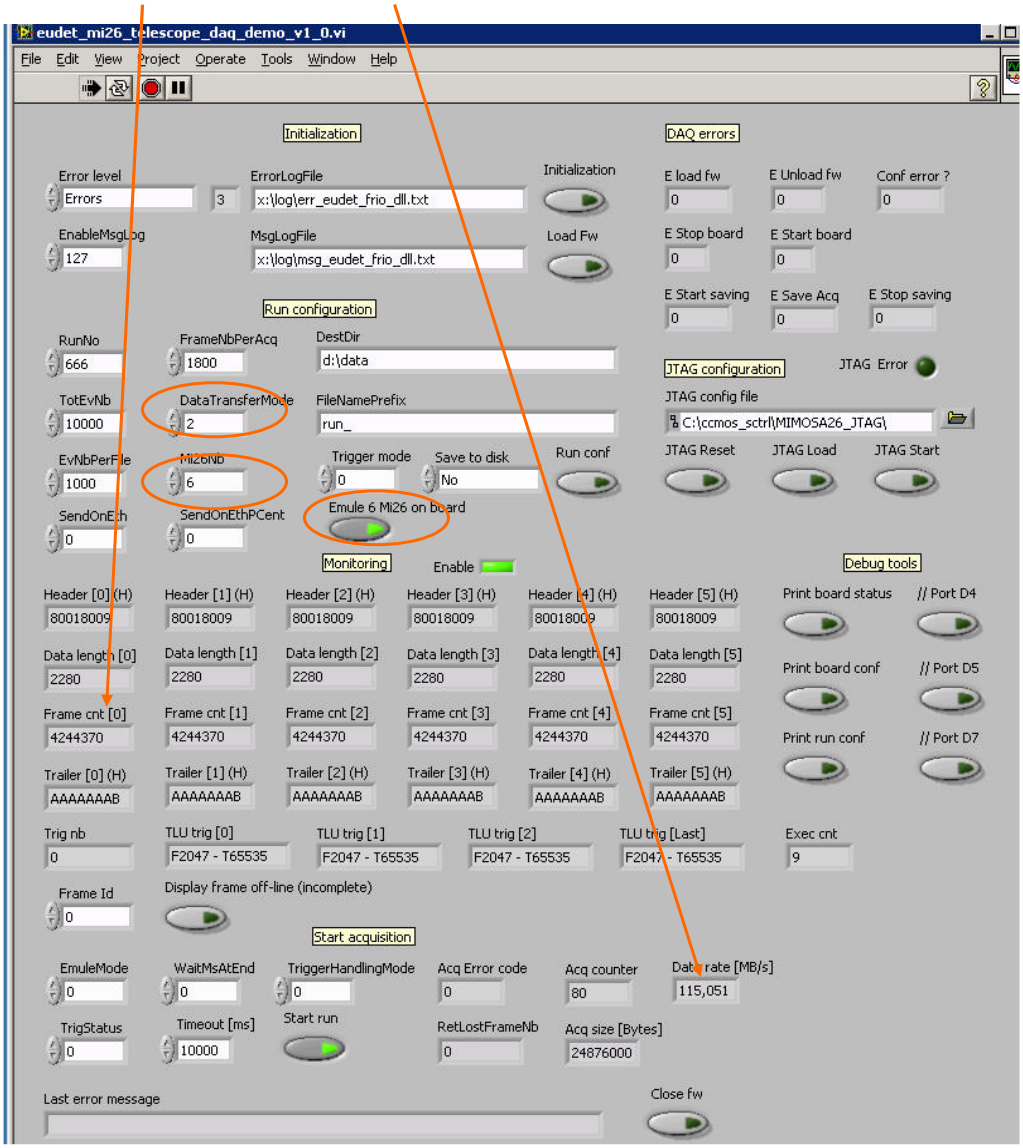
5.6 Mode EUDET 1 – 1 Mi26 x 6 – full frame length….

Mode EUDET 1 selected, one Mi 26 connected, 6 Mi 26 emulated on board, full frame length by setting Mi26 in pattern mode via JTAG. We see that frame size is the maximum and data rate close to 6 x 20 MB/s = 120 MB/s.



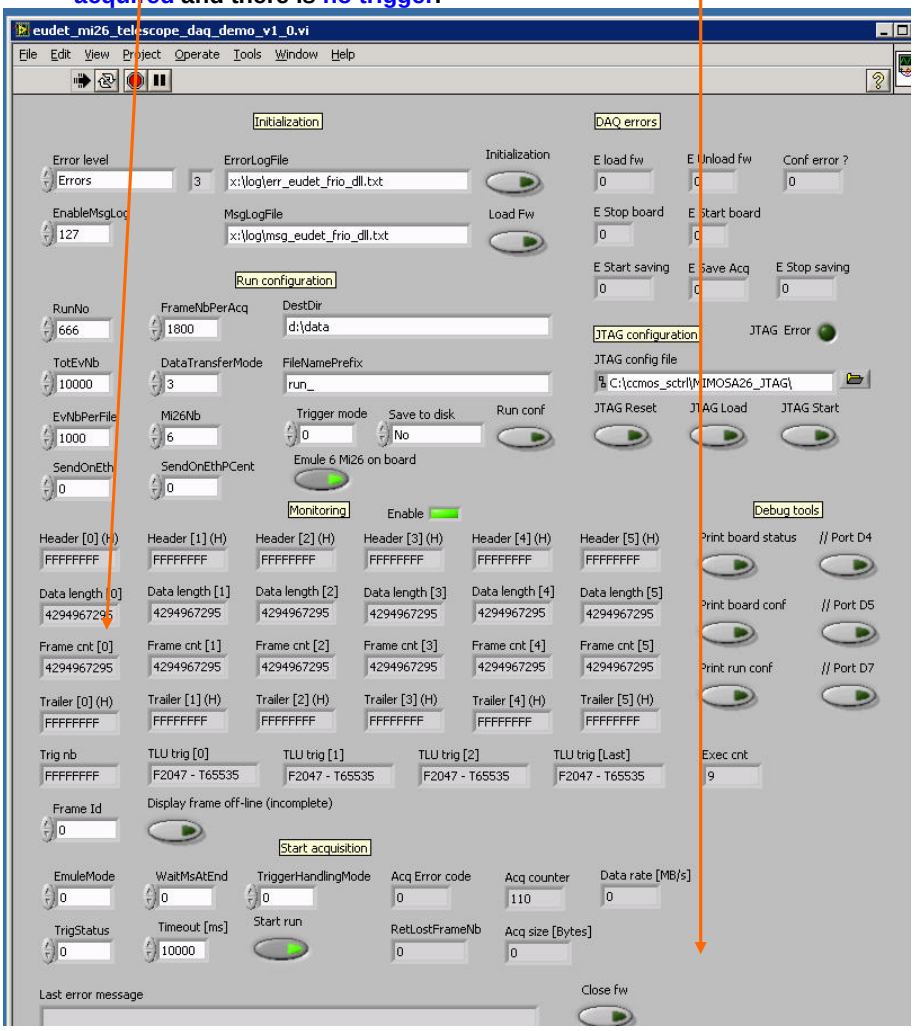
5.6.1 Mode EUDET 2 – 1 Mi26 x 6 – full frame length….

Mode EUDET 2 selected, one Mi 26 connected, 6 Mi 26 emulated on board, full frame length by setting Mi26 in pattern mode via JTAG. We see that frame size is the maximum and data rate close to 6 x 20 MB/s = 120 MB/s.



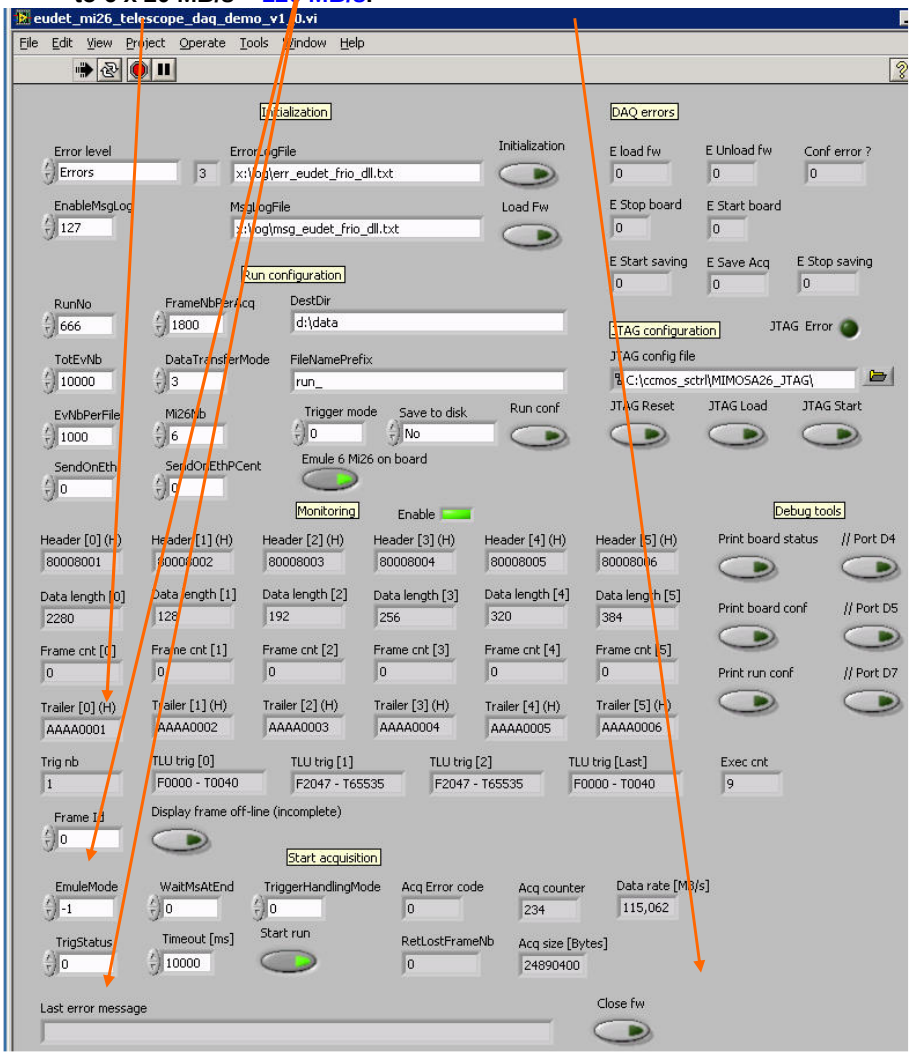
Mode EUDET 3 – 1 Mi26 x 6 – full frame length – No trigger….

Mode EUDET 3 selected, one Mi 26 connected, 6 Mi 26 emulated on board, full frame length by setting Mi26 in pattern mode via JTAG, but no trigger. We see that data are default values ($FFFFFFFF) and data rate = 0 ! It's normal because in mode EUDET 3 only frames with trigger are acquired and there is no trigger.



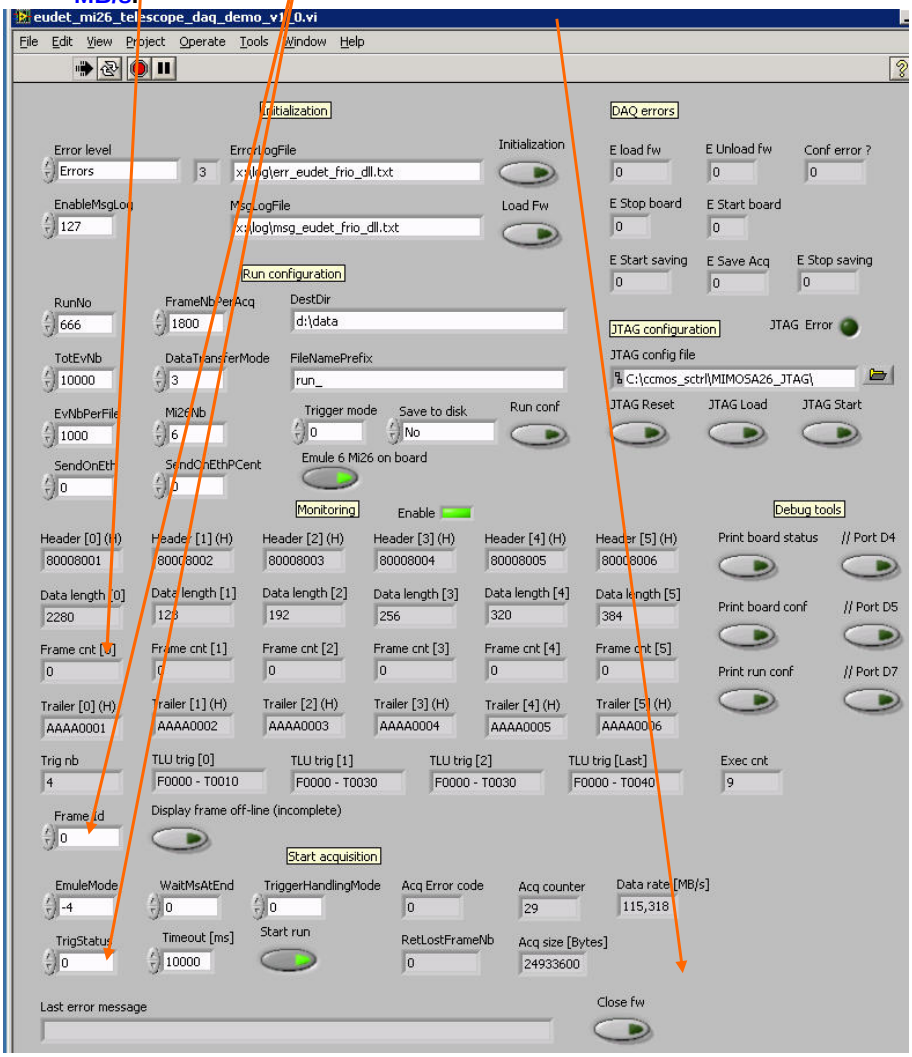
5.6.2 Mode EUDET 3 – 1 Mi26 x 6 – full frame length – 1 trigger / frame….

Mode EUDET 3 selected, one Mi 26 connected, 6 Mi 26 emulated on board, full frame length, one trigger per frame. The triggers are emulated via the parameter " Emule mode " set to -1, trigger number = abs (Emule mode). Now we get data, frame size is the maximum and data rate close to 6 x 20 MB/s = 120 MB/s.



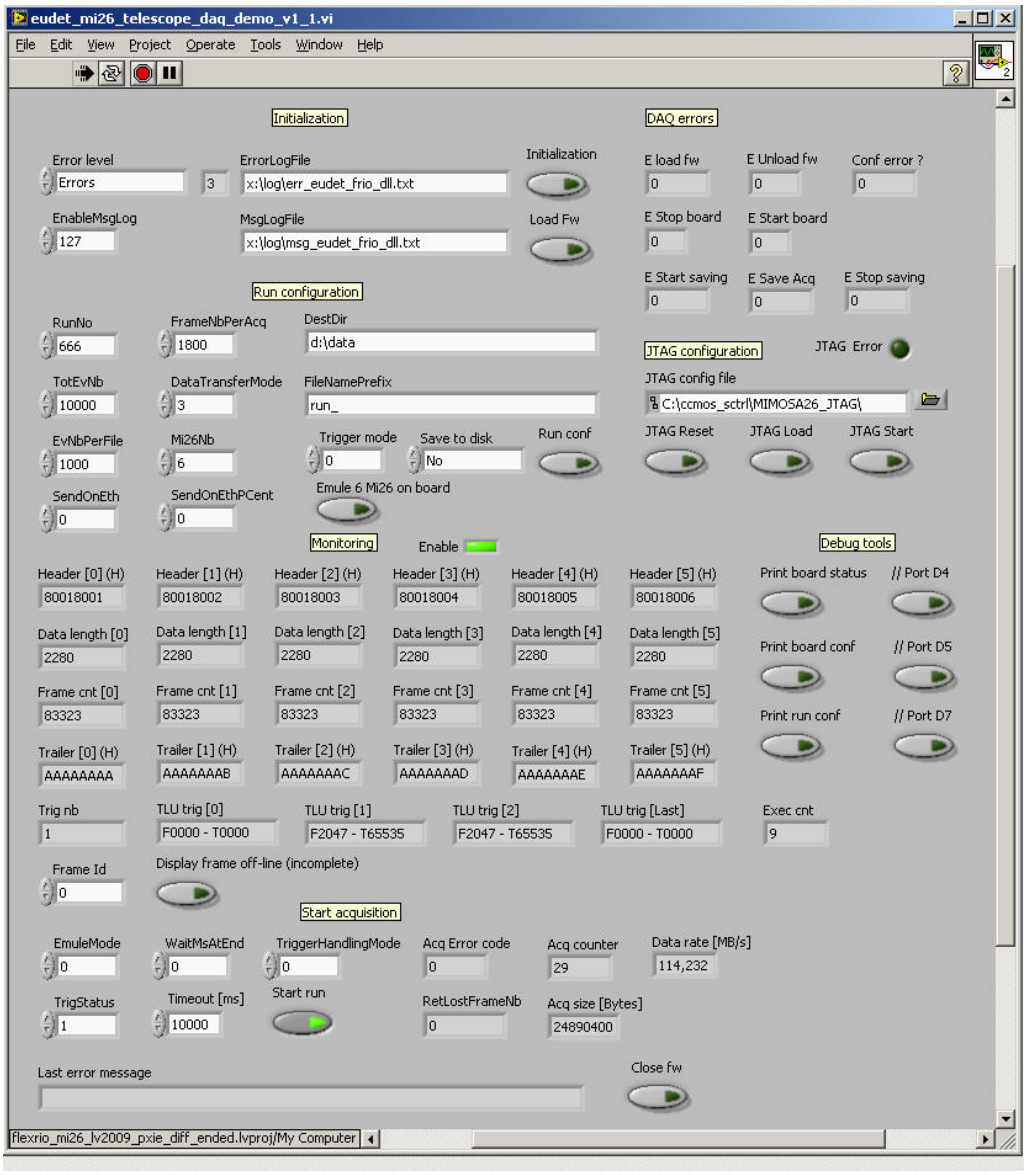
5.6.3 Mode EUDET 3 – 1 Mi26 x 6 – full frame length – 4 triggers / frame….

Mode EUDET 3 selected, one Mi 26 connected, 6 Mi 26 emulated on board, full frame size, 4 triggers / frame. The triggers are emulated via the parameter " Emule mode " set to -1, trigger number = abs (Emule mode). We see that frame size is the maximum and data rate close to 6 x 20 MB/s = 120 MB/s.



5.6.4 Mode EUDET 3 – 6 Mi26 x 6 – full frame length – 1 trigger / frame….

Mode EUDET 3 selected, six Mi 26 connected, full frame length by setting Mi26 in pattern mode via JTAG, one trigger emulated. We see that frame size is the maximum and data rate close to 6 x 20 MB/s = 120 MB/s.



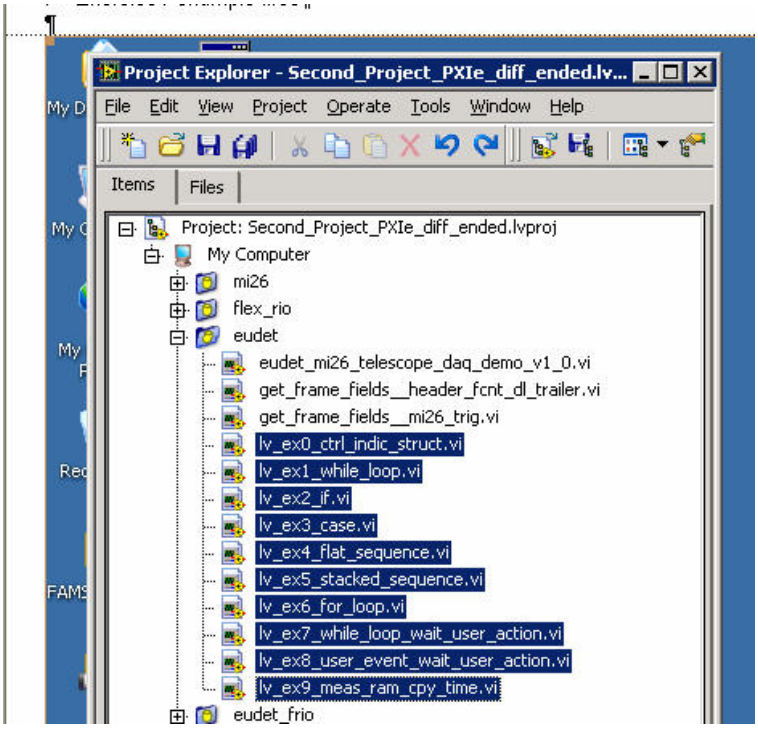
6 Labview tutorial

6.1 Introduction

The goal is to make a short tutorial about Labview graphical programming. I will present you the main Labview language structures and show you how to use them via simple programs examples.

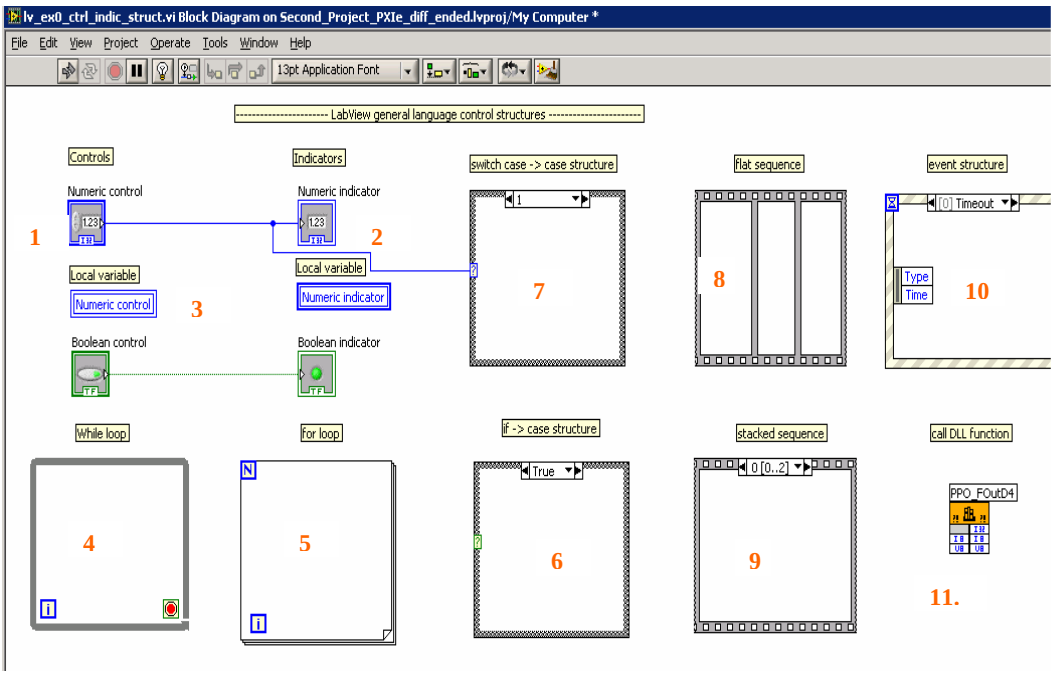
6.2 List of examples / exercises

This is the list of examples.

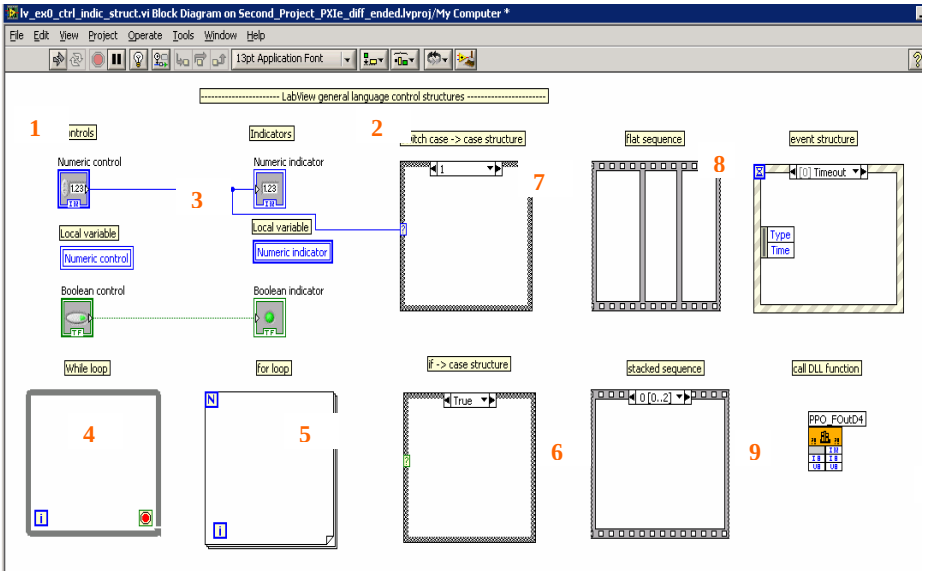


6.3 Controls, indicators & structures

The Labview GUI is called " Panel " and the source code " Diagram ". On the above diagram you can see the main Labview " components " : controls, indicators, and program control structures.



1. " Numeric control" is an input field in which user set values
2. " Numeric indicator " is an output field which displays results
3. Local variable is a way to create a variable associate to a control or an indicator, eg : " Numeric control ", " Numeric indicator ". The default way to interconnect " things " in Labview is wires … but it get quickly messy … local variables can help you to make the source code more readable.
4. " While loop" is the equivalent of the C while (..) loop. The code inserted in the box is executed until a condition tells to stop.
5. The " For loop " is the equivalent of C for (; ;) loop, it executes the code in the box N times, i is the loop index.



6. The “ case structure” with only two cases (True / False) is the equivalent of C If / else test. The box has two sides, one executed if input Boolean “?” is true, the other if it’s false.

7. The “ case structure” with more than two cases (input = integer) is the equivalent of the C switch case instruction. The box has one side per case value, the case corresponding to the input “?” is executed.

8. The “ flat sequence ” is the equivalent of sequential C code = simple code written on consecutive lines without any branch instruction. The sequence has frames from left to write. Their content is executed one after the other from left to right. This structure seems strange and useless, but in fact it is useful because Labview programming is “ data driven ” not executed sequentially

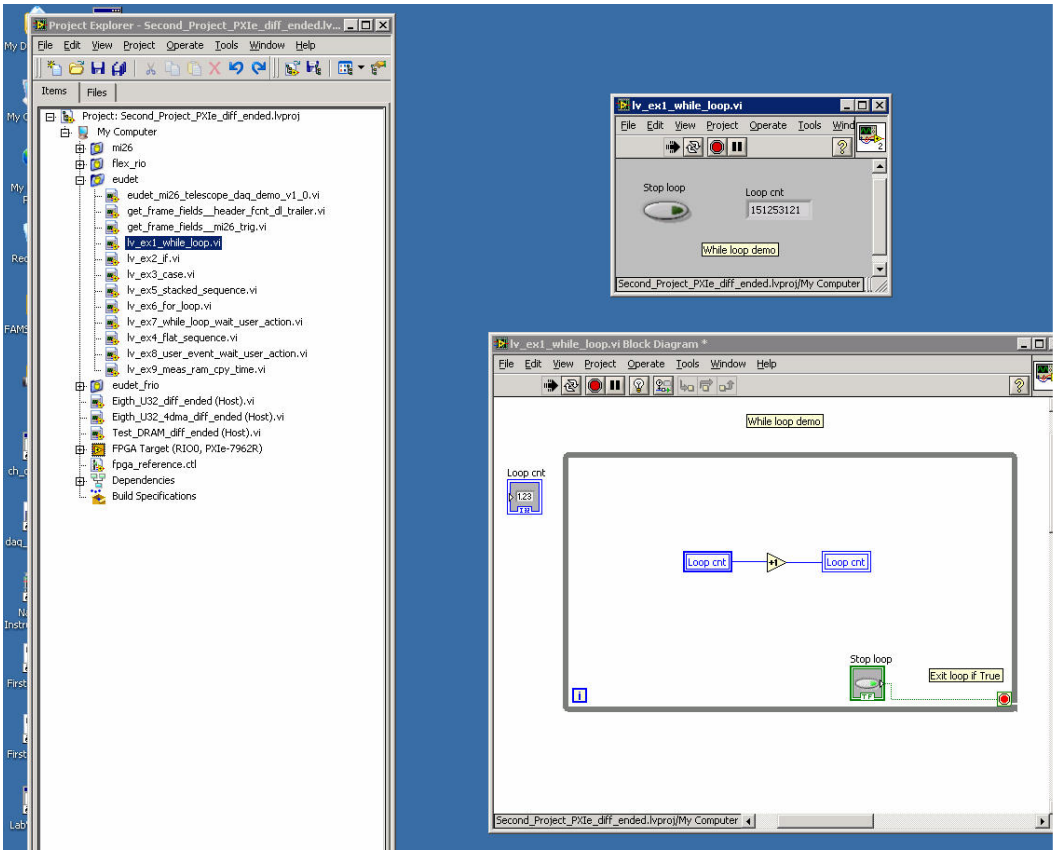
9. The “ staked sequence ” is the same structure as “ flat sequence” but it’s displayed in a compact way : staked, that’s all.

10. The “ Event ” structure is an event handler which links code execution to GUI events or user events. It’s an equivalent of a “ call back function ” in an IDE like C++ Builder.

11. The “ call DLL function ” is a way to call a function from a DLL. In fact it encapsulates the function in a Labview Vi.

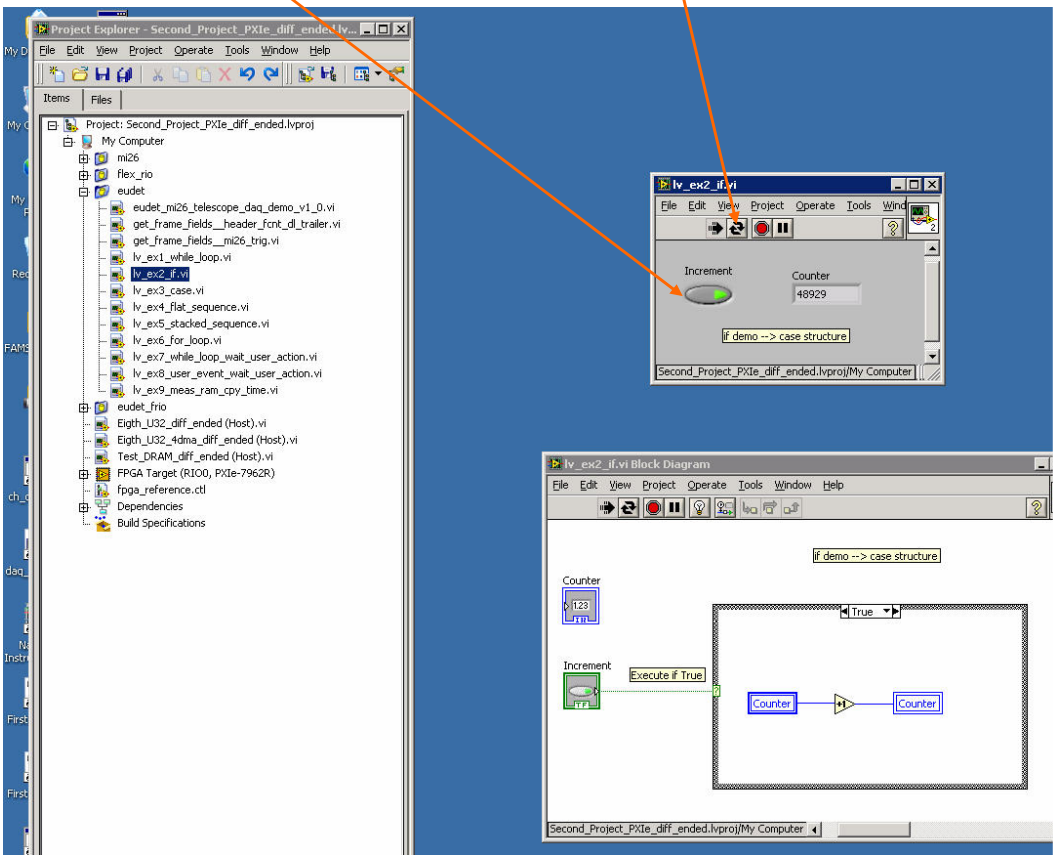
6.4 While loop

Run the program by a click on black arrow, the loop counter will increment, until you click on the " Stop loop " button. Notice the local variable used for loop counter.



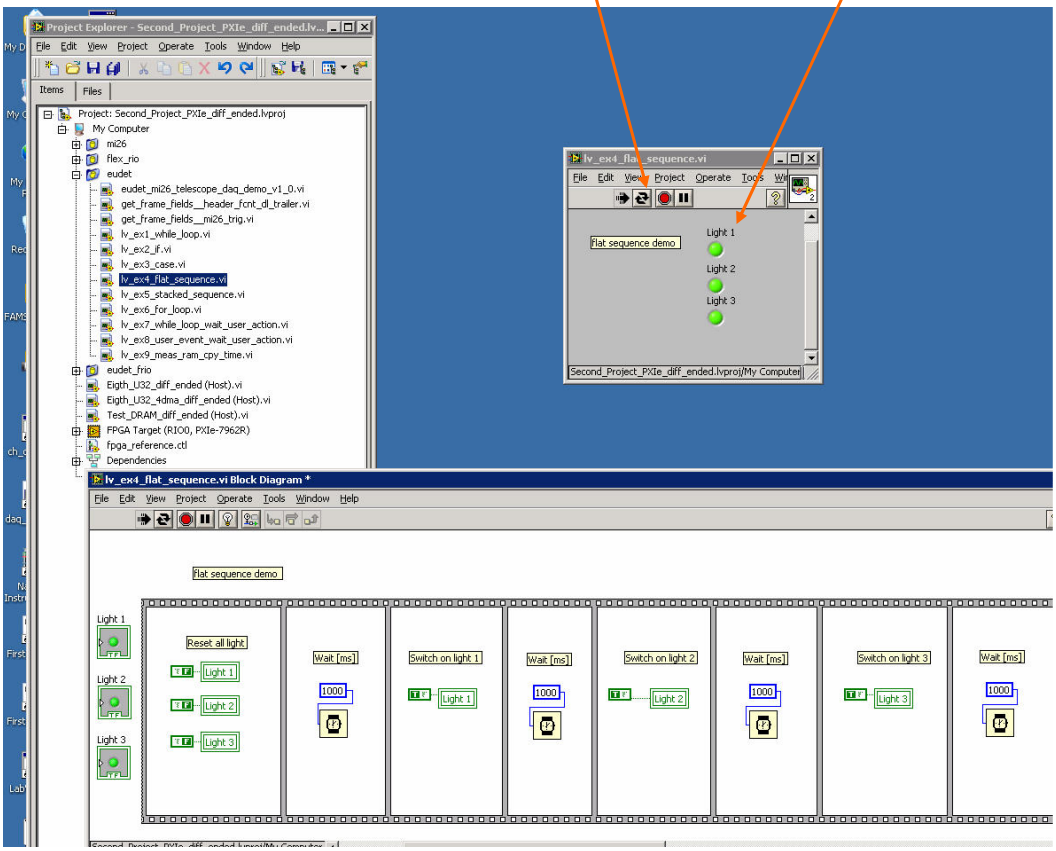
6.5 If

Run the program by a click permanent execution button.
Click on the " Increment " button, the loop counter will increment while button is on (green) and stop when it gets off.



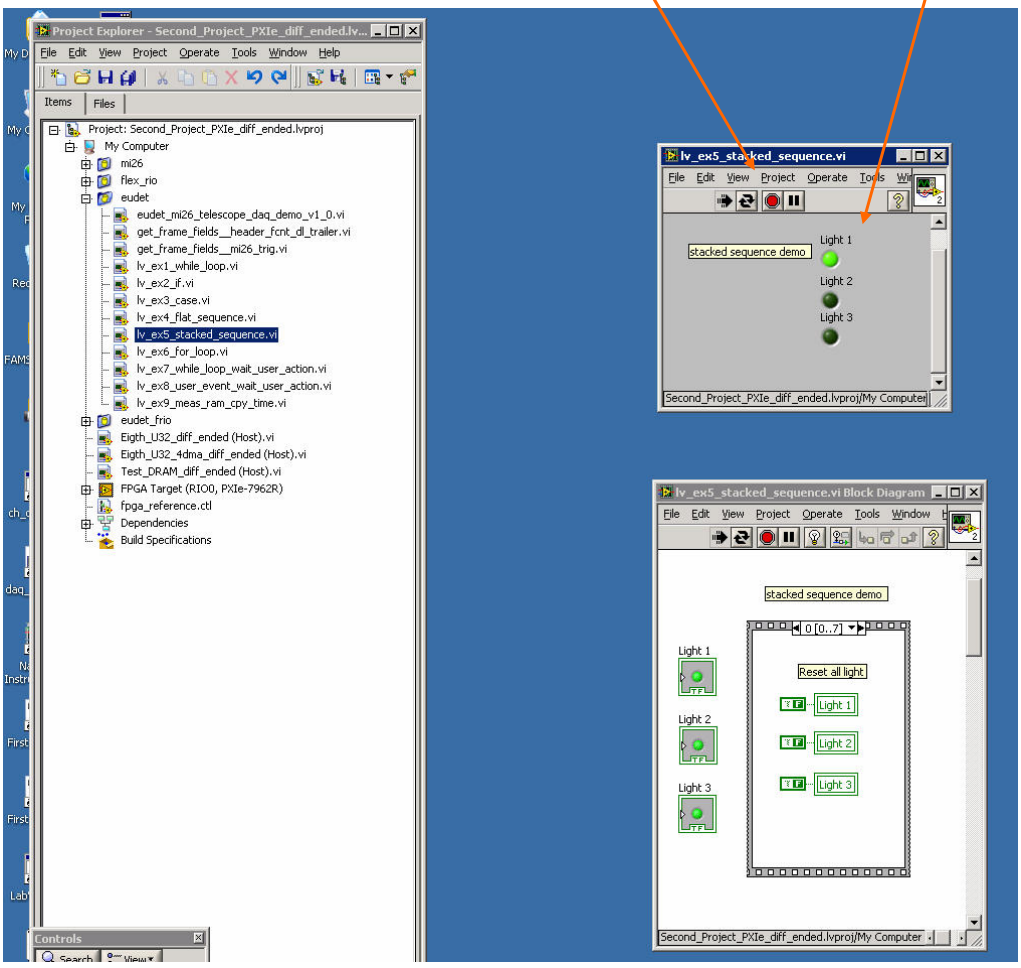
6.6 Flat sequence

Run the program by a click permanent execution button.
The sequence will execute step by step from left to right, the light will switch on one after the other, with a delay of 1000 ms.



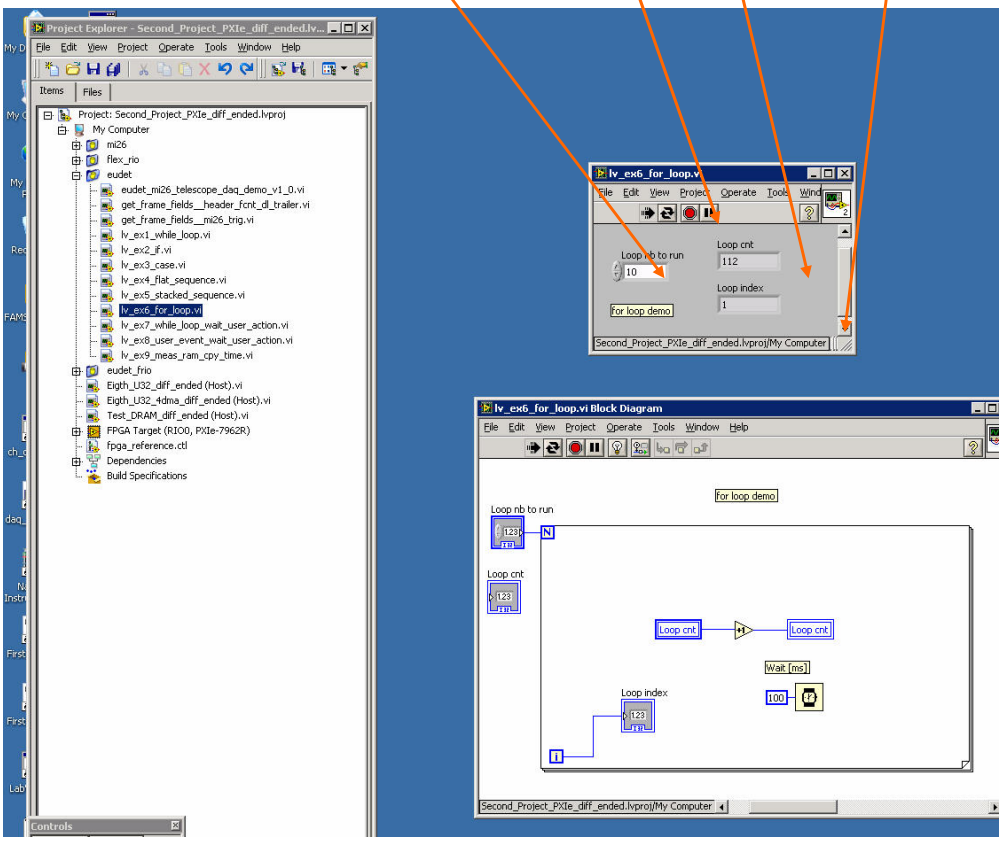
6.7 Staked sequence

Run the program by a click permanent execution button.
The sequence will executes step by step from left to right, the light will switch on one after the other, with a delay of 1000 ms. The result is the same as with the " flat sequence " (6.6) the only difference is the way the structure is displayed in diagram.



6.8 For loop

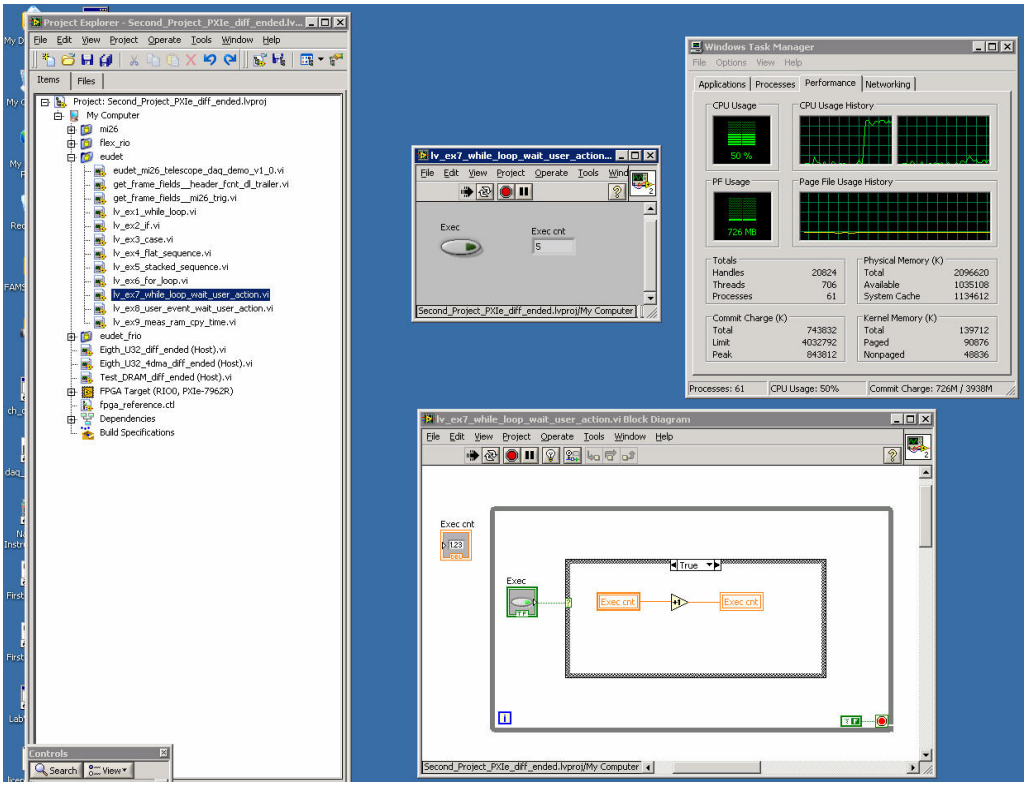
Run the program by a click permanent execution button.
The loop will execute " Loop nb to run " times, you can check it via " Loop index " indicator, but as permanent execution is enabled, the loop will restart automatically. Therefore the " Loop cnt " indicator will indicates a number higher than " Index" because it counts since beginning of program execution.



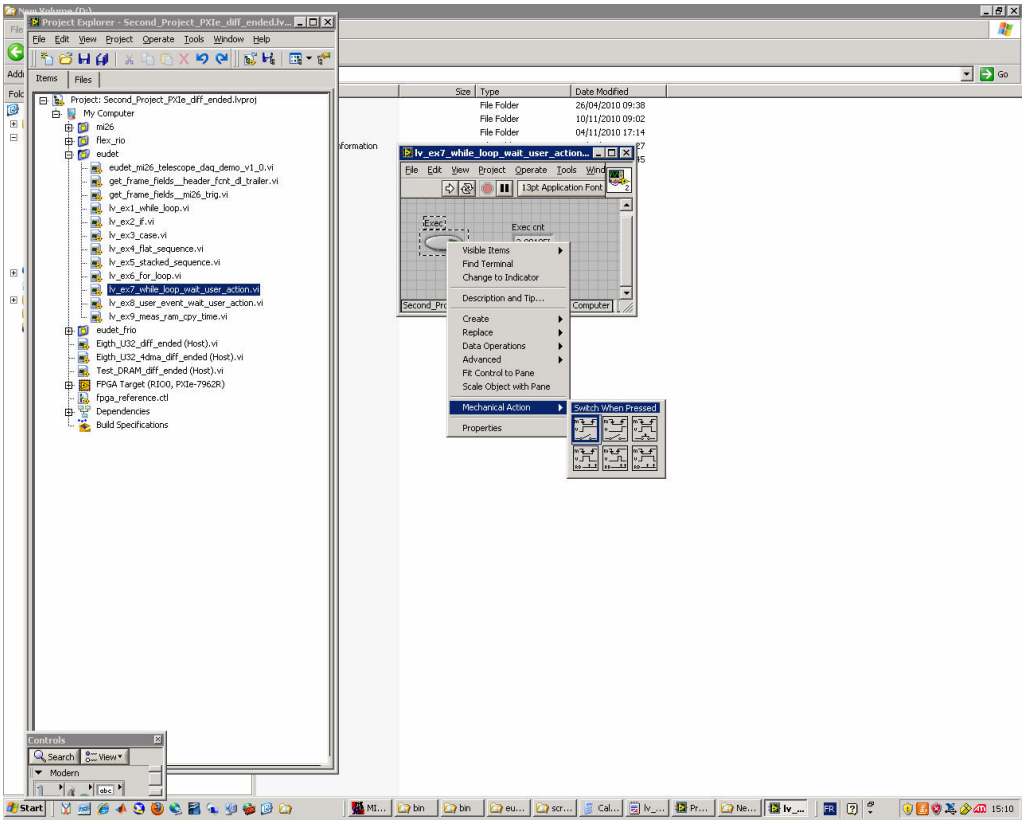
6.9 While loop wait user action

Run the program by a click on black arrow. The loop runs and the " Exec cnt " indicator increments while the " Exec " switch is on (green).

The windows tasks manager show that Laview uses 50 % of the CPU to perform this task … it's because it's a polling of the " Exec " switch state ! it's not event driven …



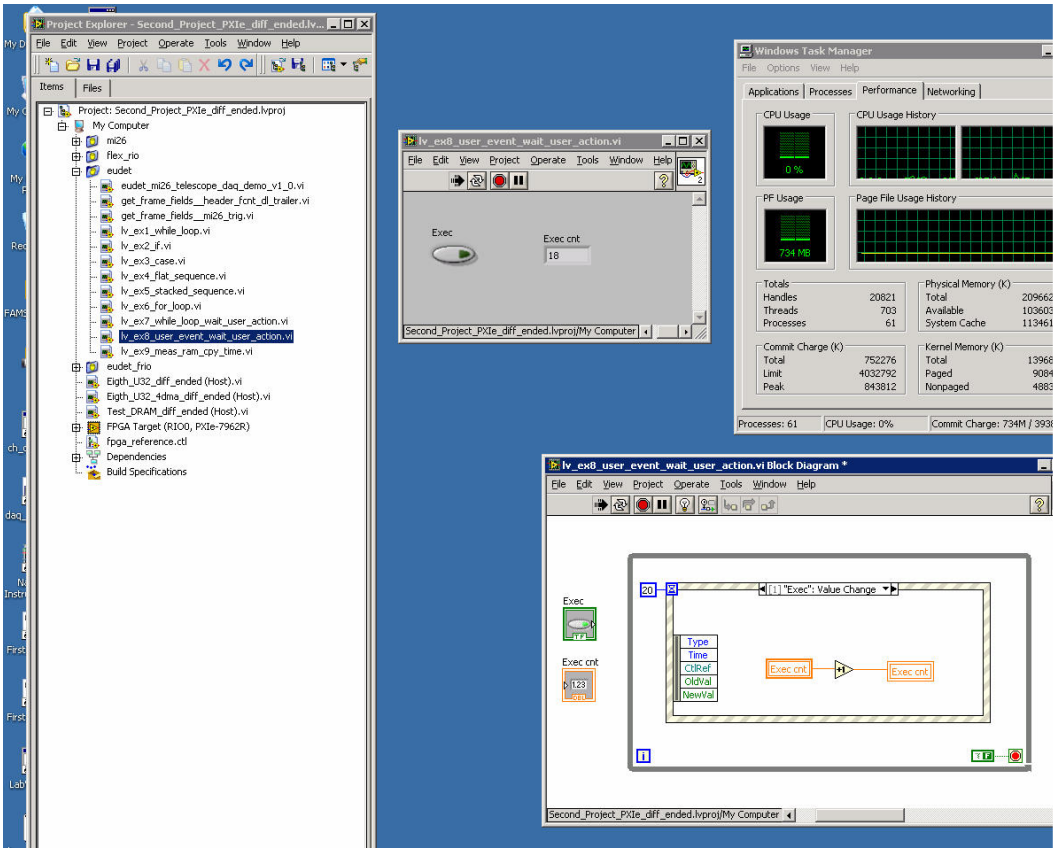
Please notice the switch property " mechanical action " set to " Switch when pressed " → this is a simple ON / OFF switch like the one used to control the light of this room.



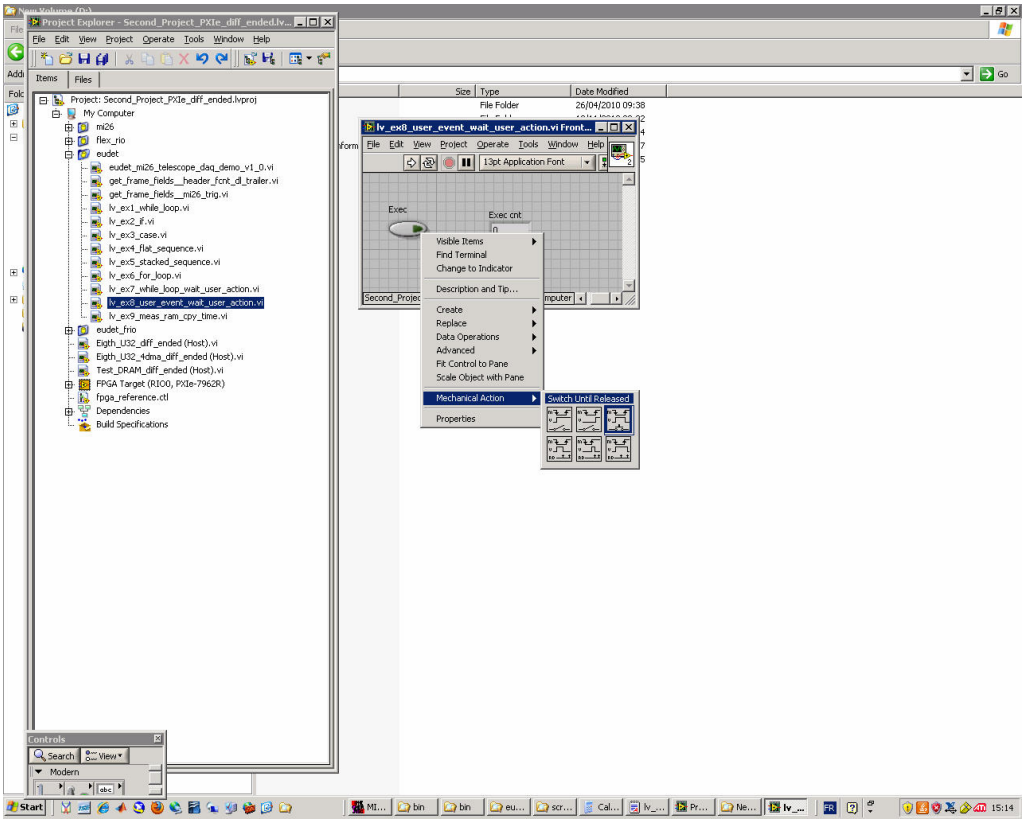
6.10 Event wait user action

Run the program by a click on black arrow. The loop runs and the " Exec cnt " indicator increments while the " Exec " switch is on (green).

The windows tasks manager show that Laview uses 0 % of the CPU to perform this task … it's because now it's event driven … compare to result of (6.9) !!!



Please notice the switch property " mechanical action " set to " Switch until released " → this is " push button " like the one we use for a ring.

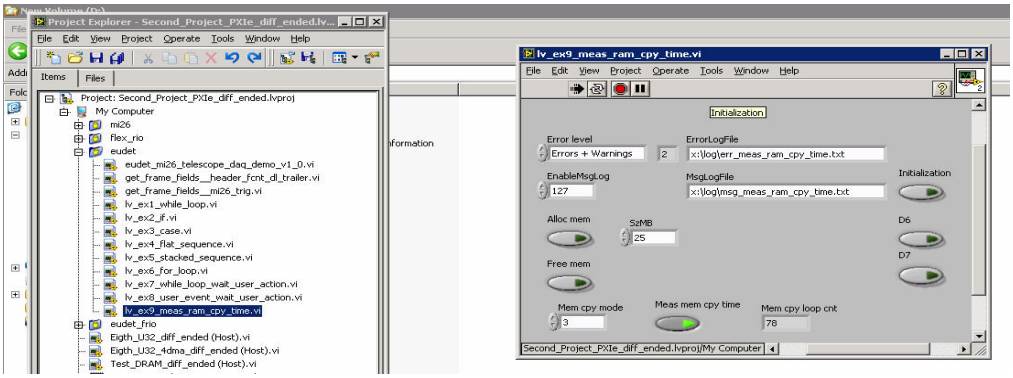


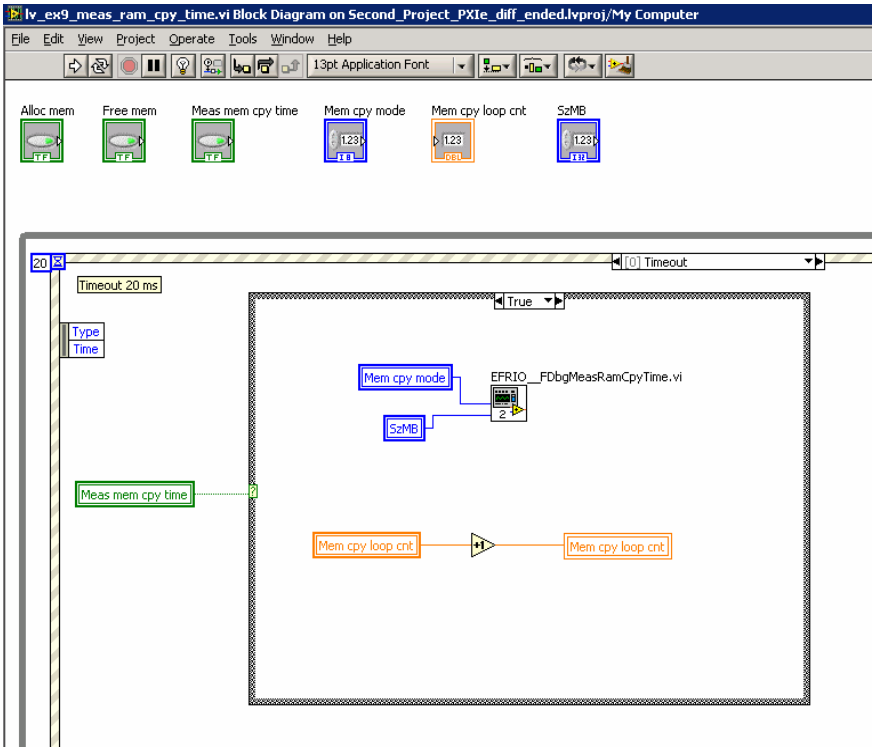
6.11 DLL function call

It will be explained in next version of documentation.

6.12 RAM copy execution time measurement

This program uses most of the control structures shown in the examples. It also uses the " Event " structure in a different way than in the examples, by using the " time out " event. It's a way to make a polling but to free CPU between two pool cycles. The execution time is not displayed in GUI, it must be measured with an oscilloscope on parallel port line D6 (or D7 ?).




lv_ex9_meas_ram_cpy_time.vi Block Diagram on Second_Project_PXIe_diff_ended.lvproj/My Computer
File Edit View Project Operate Tools Window Help
13pt Application Font
Alloc mem
Free mem
Meas mem cpy time
Mem cpy mode
Mem cpy loop cnt
SzMB
20
Timeout 20 ms
Type
Time
[0] Timeout
True
Mem cpy mode
SzMB
EFRIO__FDbgMeasRamCpyTime.vi
Meas mem cpy time
Mem cpy loop cnt
Mem cpy loop cnt

7 DAQ source code

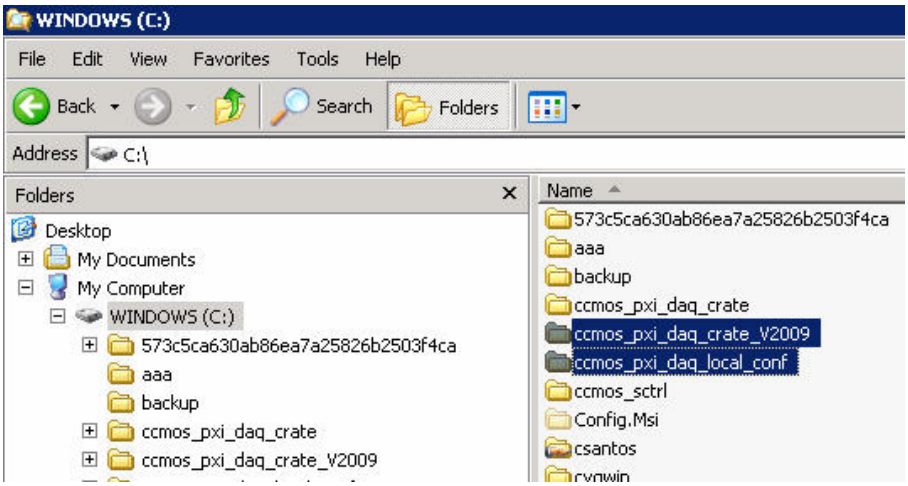
7.1 Introduction

The goal of this chapter is to give you an overview of DAQ Labview source code. We can't go into details, but we can explain the job of each part and how it runs.

7.2 DAQ source tree

Two directories contain the DAQ Labview source code :

- **ccmos_pxi_daq_local_conf → configuration files for old PXI DAQ (not for EUDET)**
- **ccmos_pxi_daq_crate_V2009 → source files**

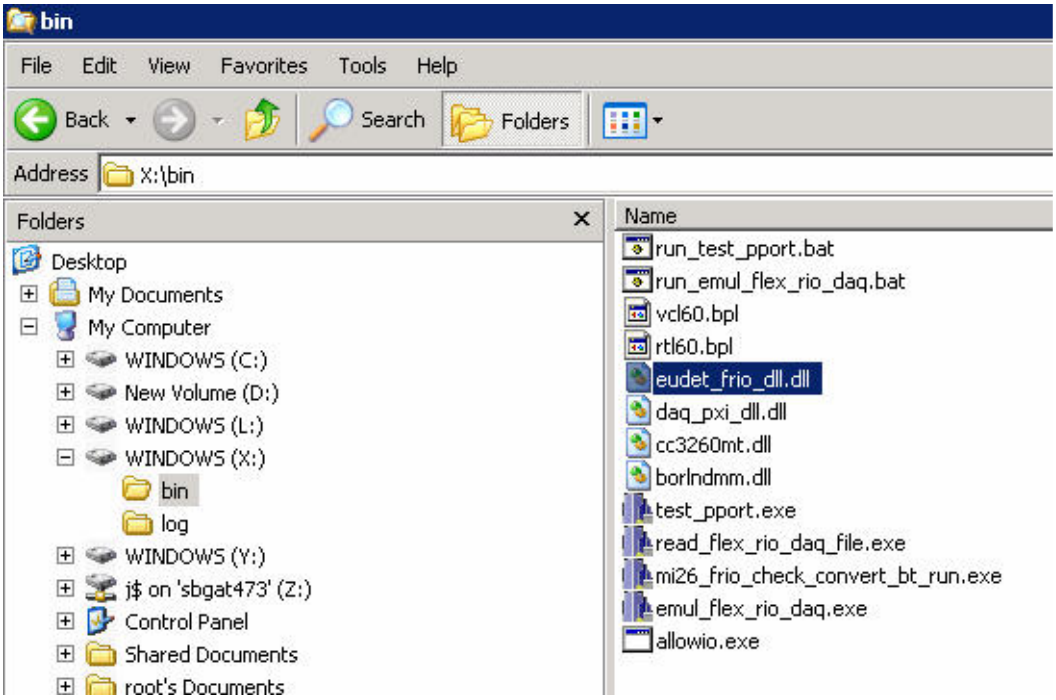


The batch " loc_labview_v2009.bat " creates three virtual drives → Y:, X:, L:

The batch " loc_labview_v2009.bat " creates three virtual drives → Y:, X:, L:

- **Y → Points to ccmos_pxi_daq_crate_V2009**
- **X → Points to ccmos_pxi_daq_crate_V2009\X**
  It contains the binary part of the C tree architecture → DLL
- **L → Points to ccmos_pxi_daq_crate_V2009\L**
  It contains the Labview source files for EUDET Telescope DAQ

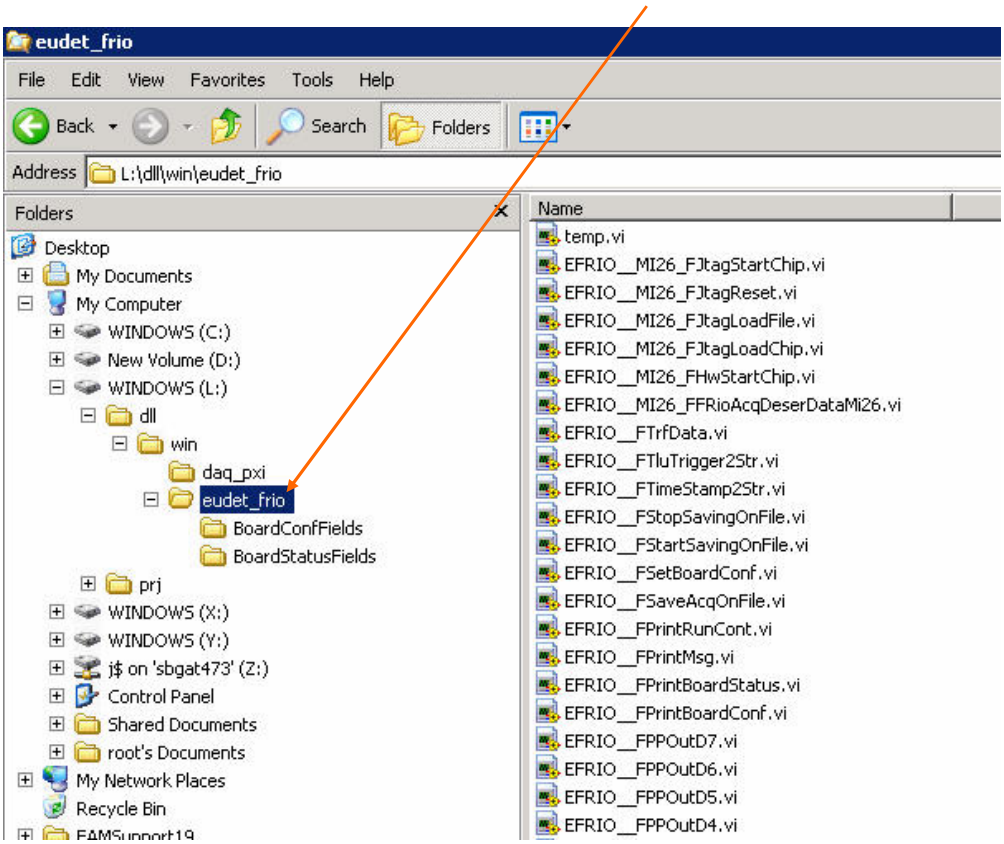
  The X: virtual drive contains the eudet_frio_dll.dll and the log files directory.



The L: virtual drive contains the eudet_frio_dll.dll interface to Labview.

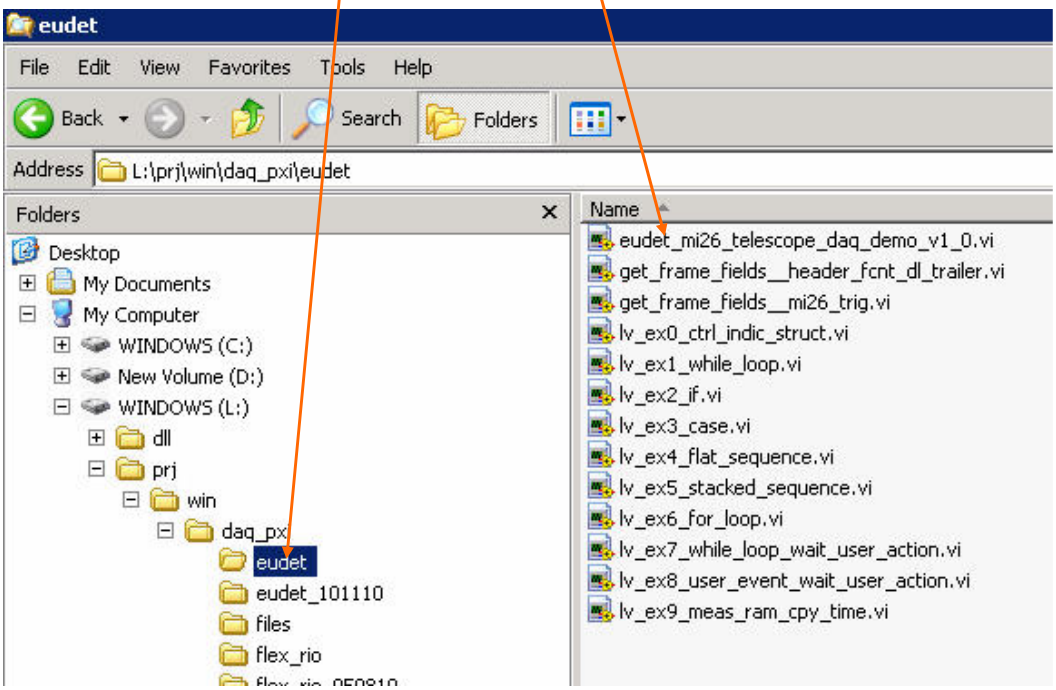
It means one Vi (Virtual Instrument ⇔ function in C code) which " encapsulates " each function of the DLL. Each Vi has the same name as the DLL function, example : EFRIO__MI26_FJtagStartChip.vi encapsulates the DLL function EFRIO__MI26_FJtagStartChip (…).

These Vi are installed in directory L:\dll\win\eudet_frio



The L: virtual drive contains EUDET Telescope DAQ project.

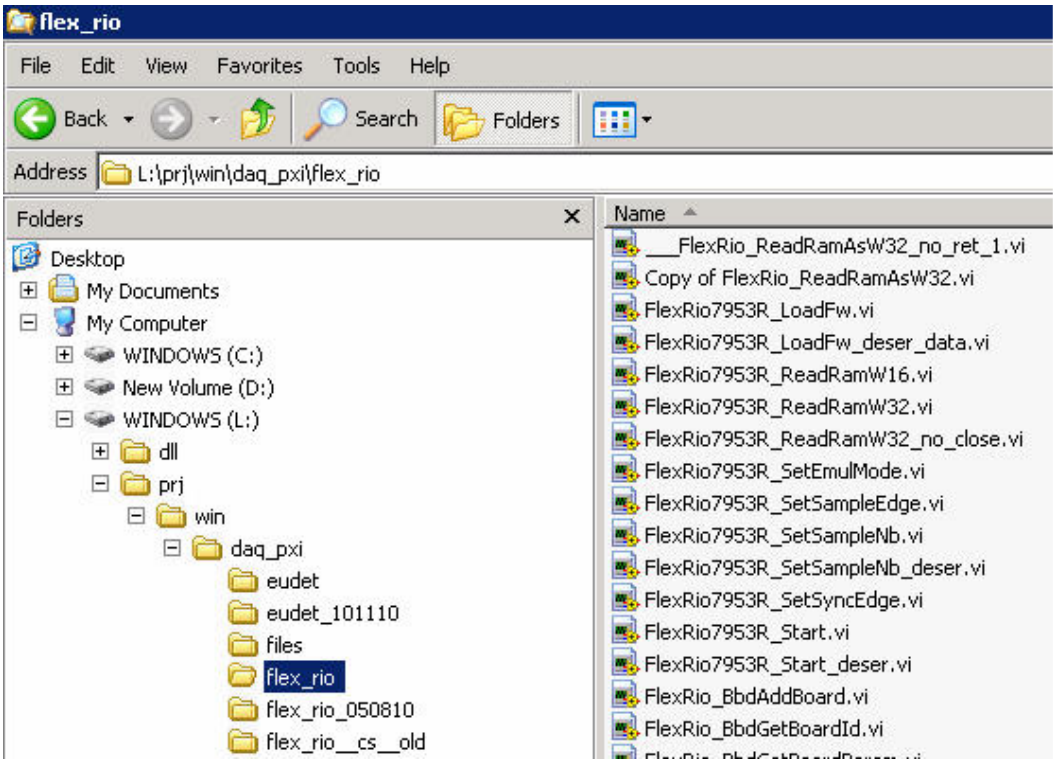
The DAQ source file eudet_mi26_telescope_daq_demo_v1_0.vi is installed in the directory L:\prj\win\daq_pxi\eudet.



The L: virtual drive contains Flex RIO board control Vi. This is an API written in Labview to configure the Flex RIO board.

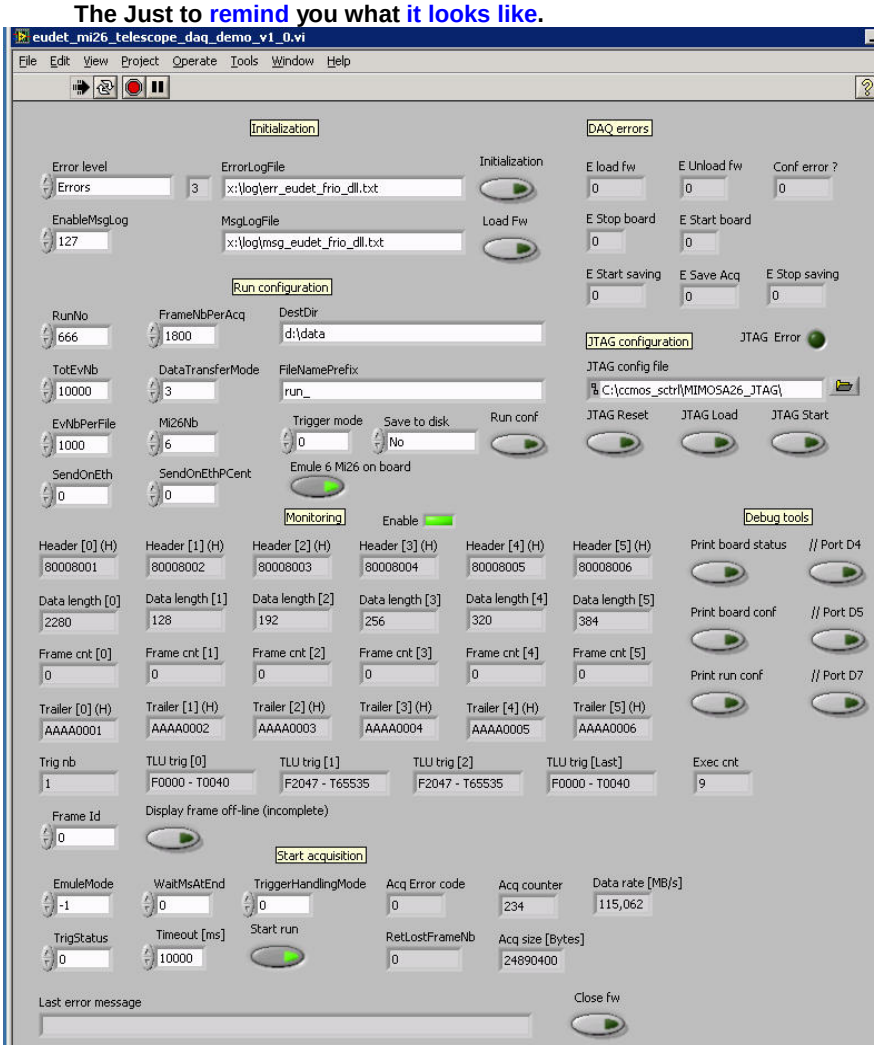
We can't explain each Vi during this training, we will see the most useful when we will look into in the DAQ application source code.

Theses files are located in directory L:\prj\win\daq_pxi\flex_rio.



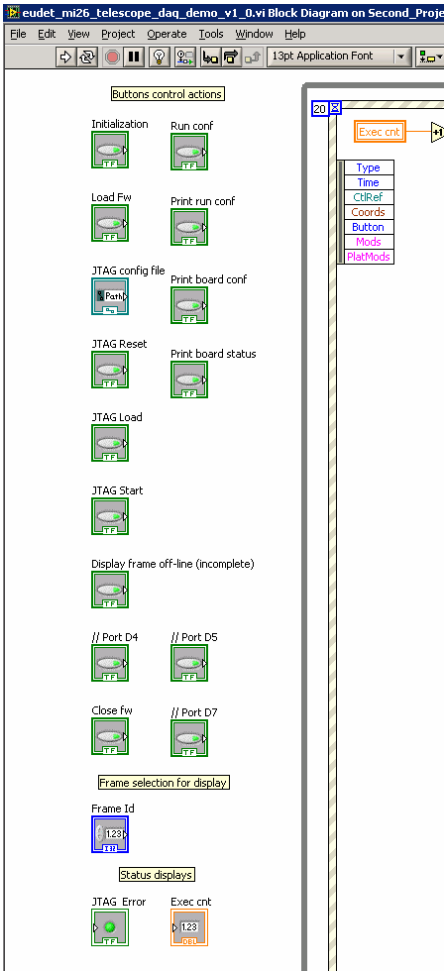
7.3 DAQ software GUI

The Just to remind you what it looks like.



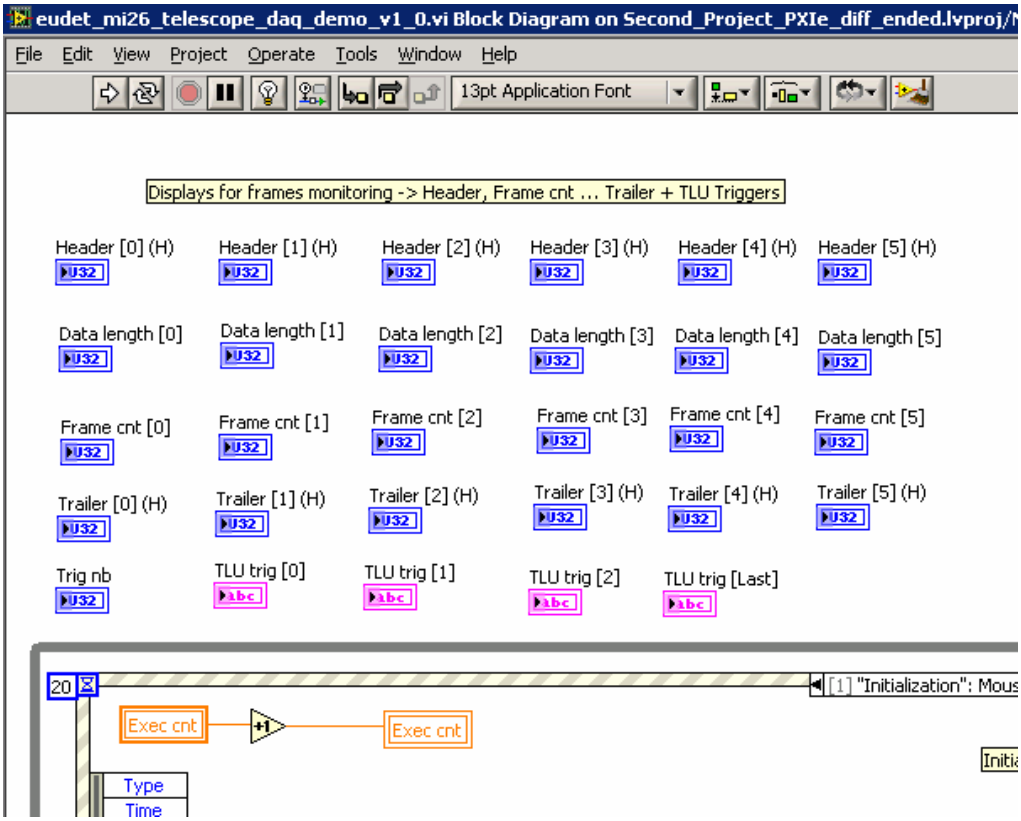
7.4 Controls of the DAQ software

It's the controls used on DAQ software GUI. As you can see they are on the left, out of any control structure (loop and so on), it's because I prefer to use local variables to access them rather than having a lot of wires which cover the diagram …



7.5 Indicators of the DAQ software

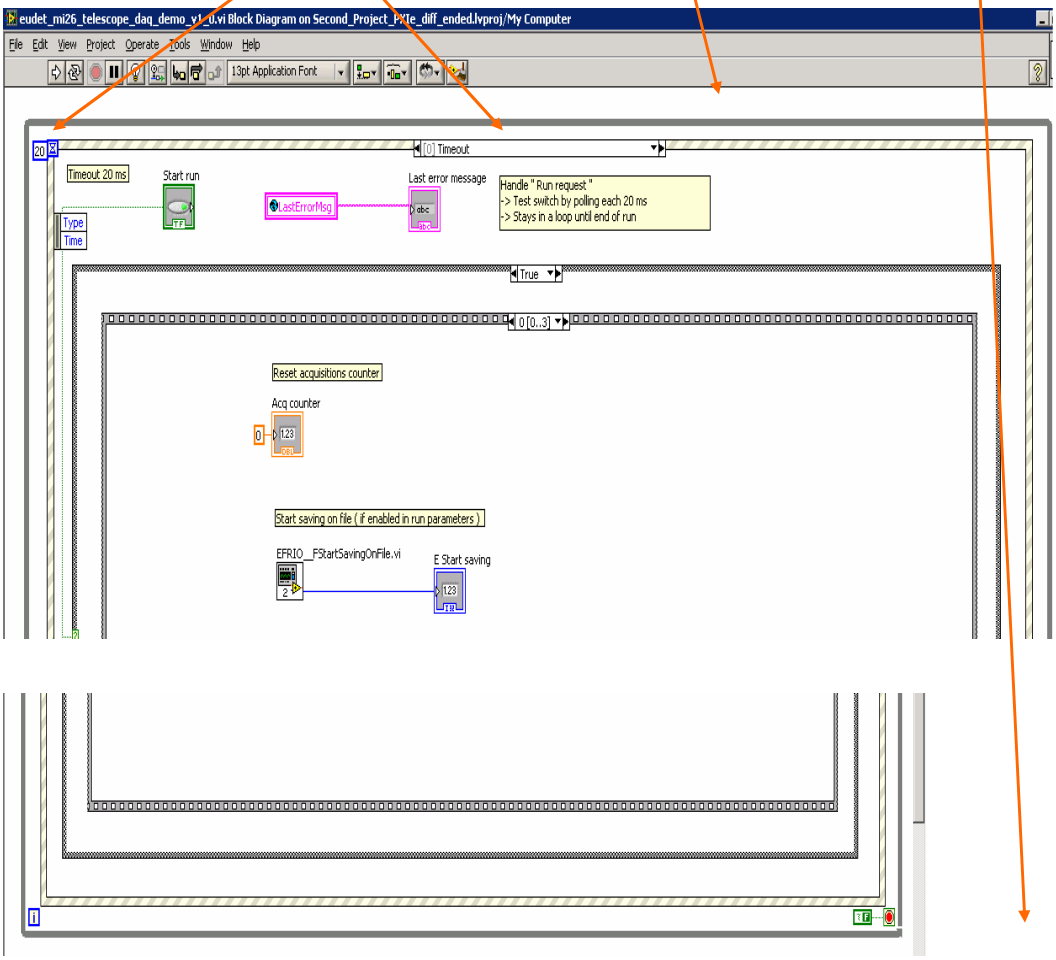
It's the indicators (or most of them) used on DAQ software GUI. As you can see they are on the top, out of any control structure (loop and so on), it's because I prefer to use local variables to access them rather than having a lot of wires which cover the diagram …



7.6 The main “ endless ”loop …

The whole DAQ software is in an “ endless ” while loop (exit condition set to false by a constant) because the “ event ” structure has some limitations. Therefore the only way I found is to encapsulate the “ event ” structure in a “ while (1) loop”.

It will not waste CPU time because of the “ event ” structure “ time out ” event which will release CPU each 20 ms. It’s a way to have both : polling and event driven code in the same application.

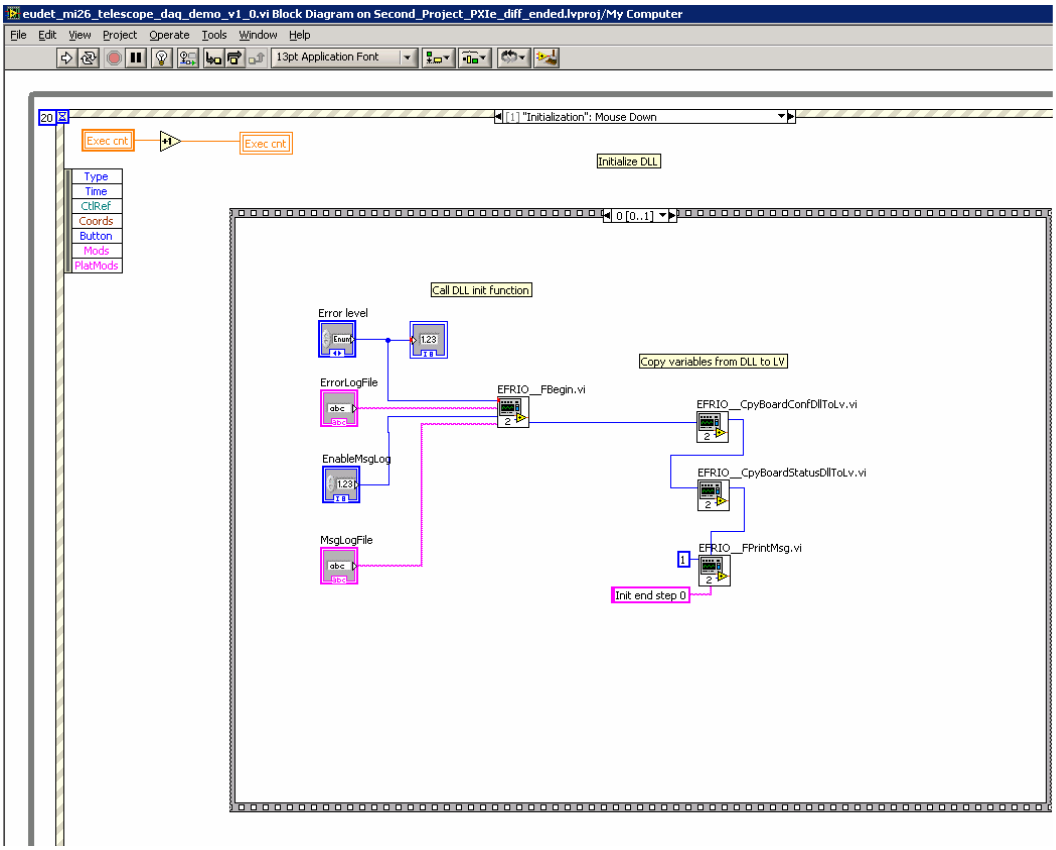


7.7 The initialization code → " Initialization " button

This code initialize the library, it's called by a click on " Initialization " button via the " event " structure on mouse down event.

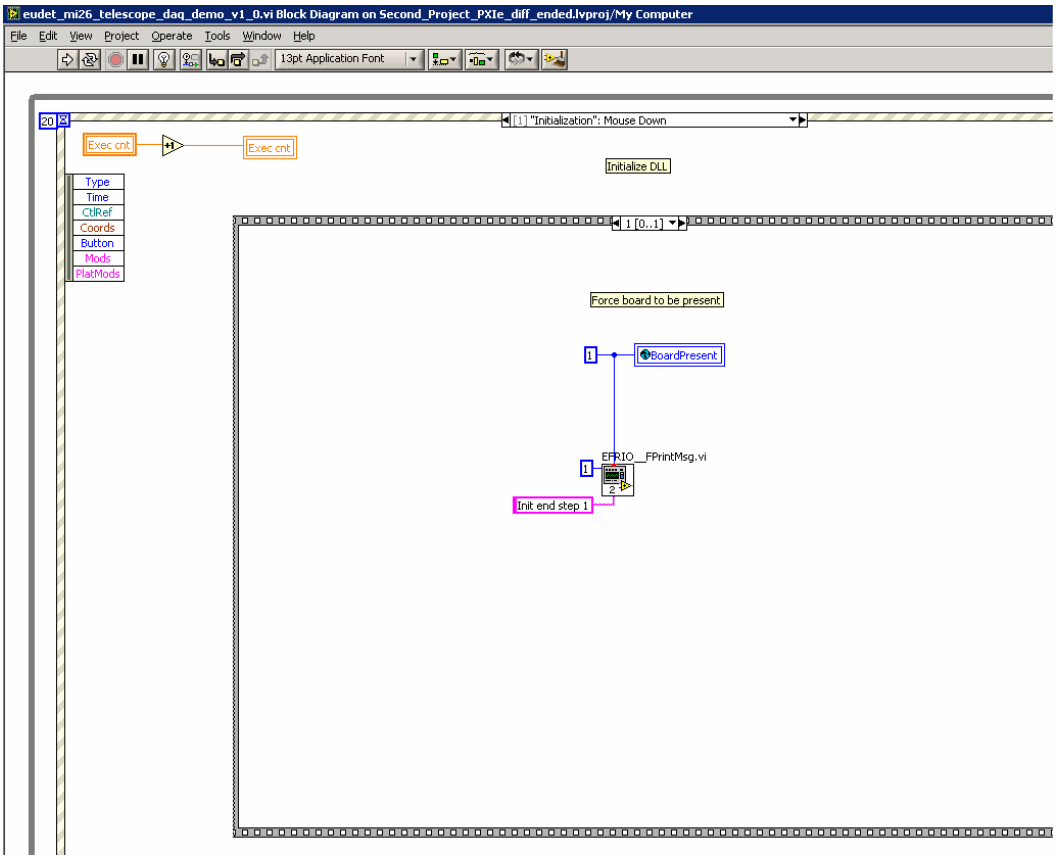
Step 0

It calls the eudet_frio DLL initialization function → EFRIO__FBegin (…).



7.8 Step 1

It forces the board state to be present and prints a log message.

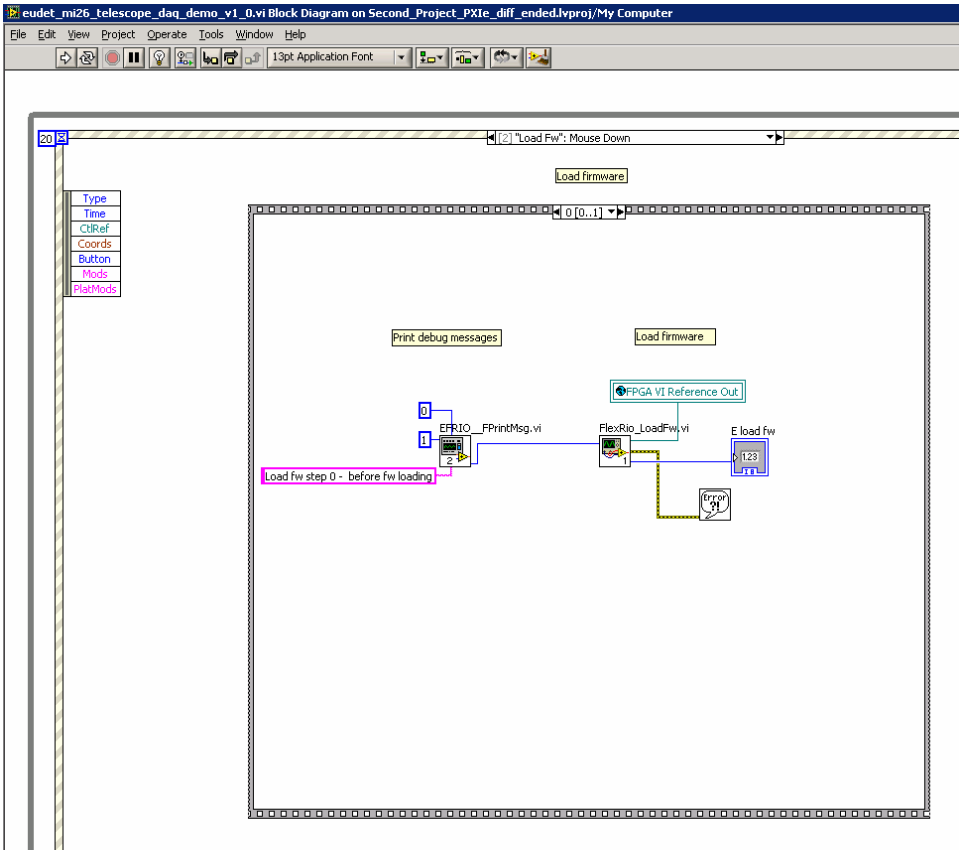


7.9 The firmware loading code → " Load Fw " button

This code loads the firmware in Flex RIO board, it's called by a click on " Load Fw " button via the " event " structure on mouse down event.

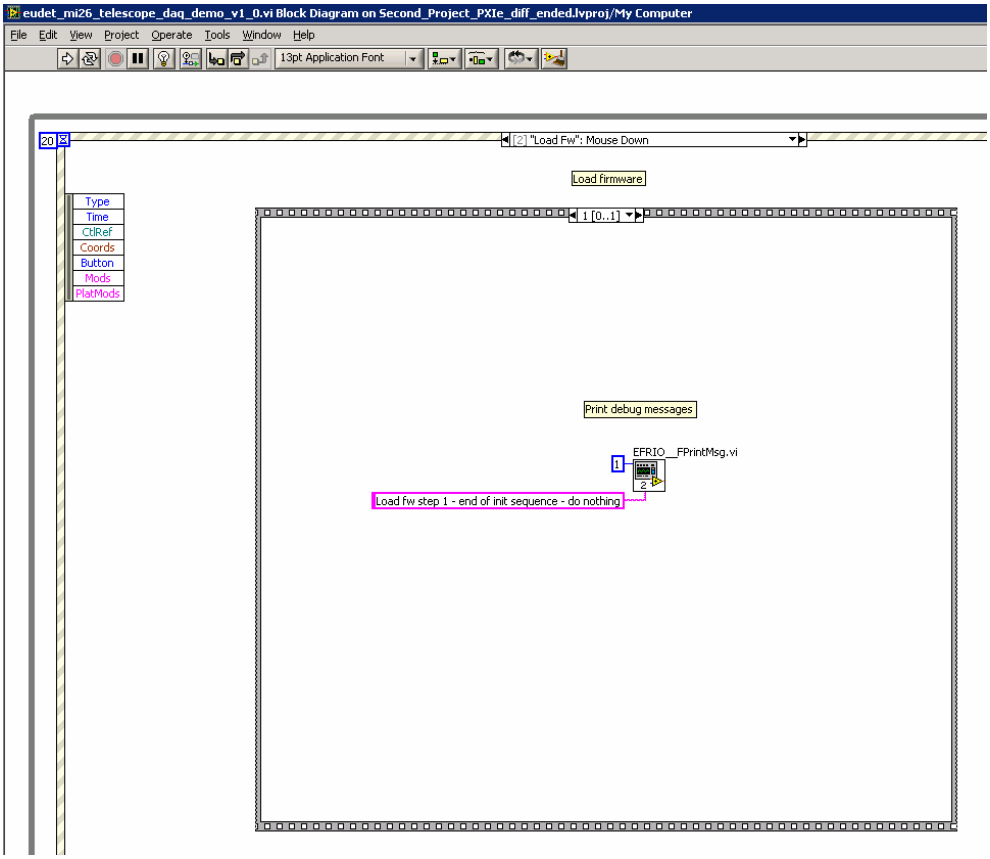
7.9.1 Step 0

It calls the fw loading Vi → FlexRio_LoadFw.vi



7.9.2 Step 1

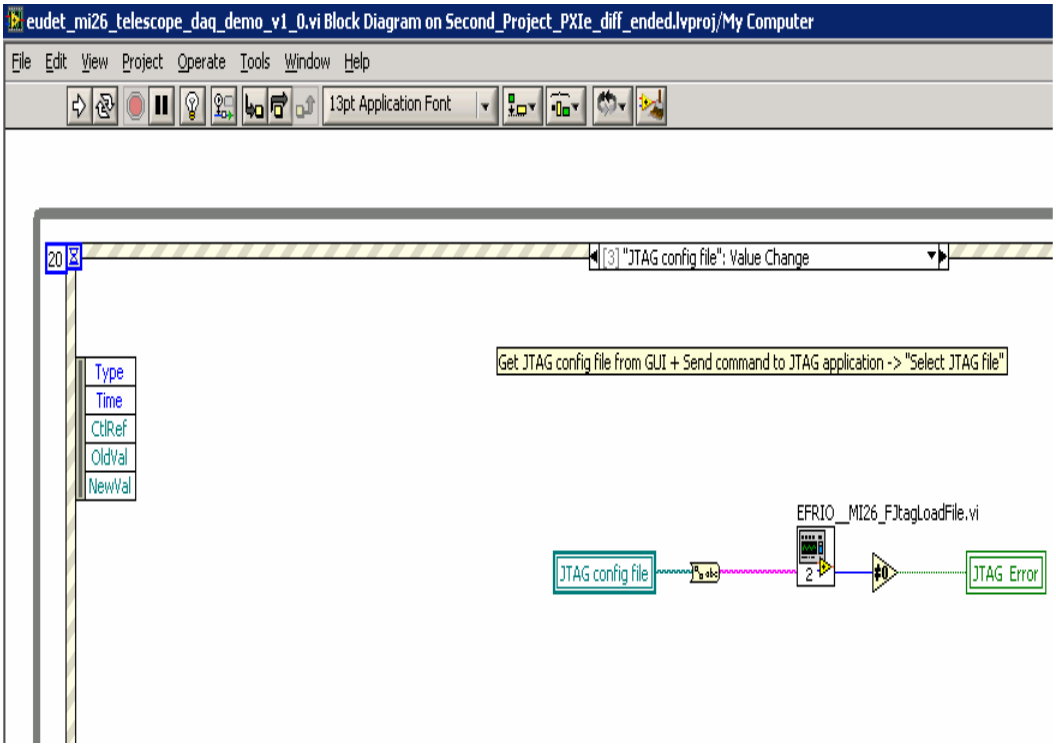
It prints a log message.



7.10 The JTAG code → " Initialization " button

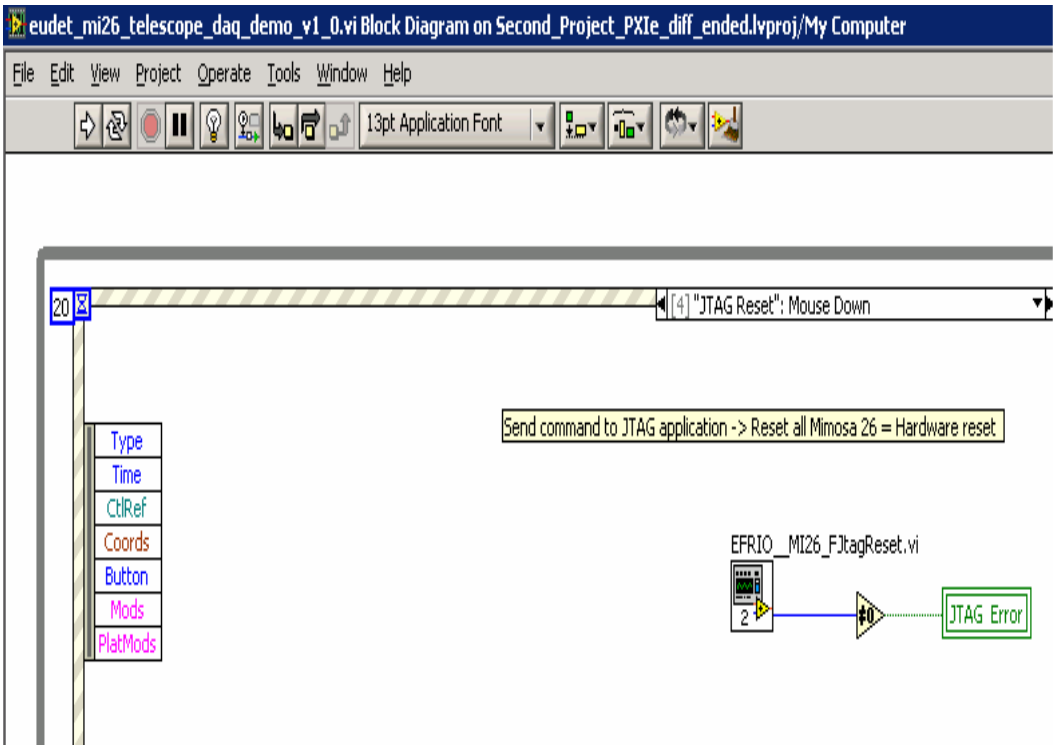
7.10.1 JTAG configuration file loading → " On file selection "

This code tells the JTAG application to load a JTAG configuration file via COM interface. It's called on file name change. It calls the eudet_frio DLL JTAG loading file function → EFRIO__MI26_FJtagLoadFile (…)



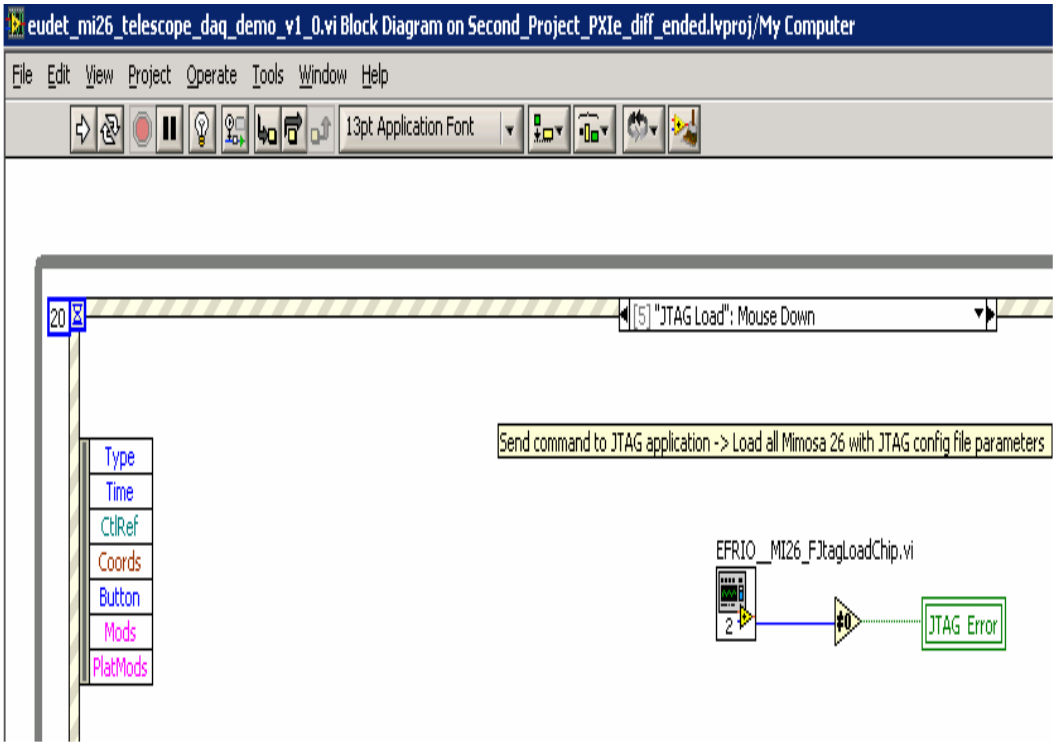
7.10.2 JTAG Reset chip → " JTAG Reset " button

This code tells the JTAG application to reset all the Mimosa 26 via COM interface. It's called by a click on " JTAG Reset ".button. It calls the eudet_frio DLL JTAG reset function → EFRIO__MI26_FJtagReset (…).



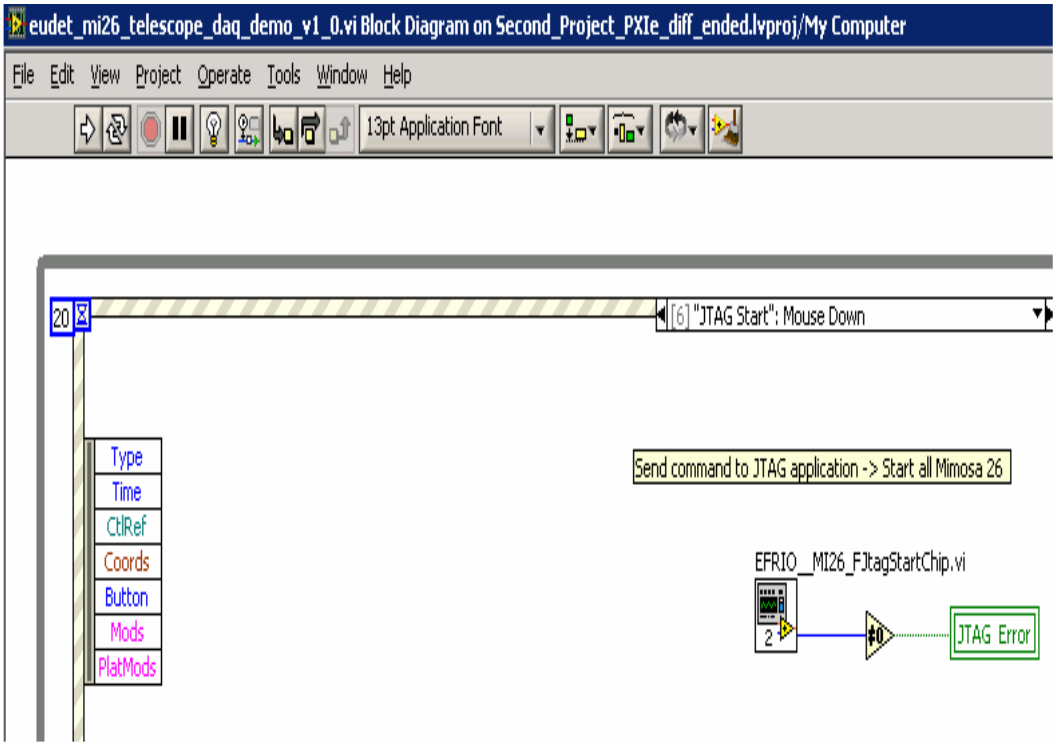
JTAG Load chip → " JTAG Load " button

This code tells the JTAG application to load all the Mimosa 26 via COM interface. It's called by a click on " JTAG Load ".button. It calls the eudet_frio DLL JTAG load chip function → EFRIO__MI26_FJtagLoadChip (…).



JTAG Start chip → " JTAG Start " button

This code tells the JTAG application to start all the Mimosa 26 via COM interface. It's called by a click on " JTAG Start ".button. It calls the eudet_frio DLL JTAG start chip function → EFRIO__MI26_FJtagStartChip (…).

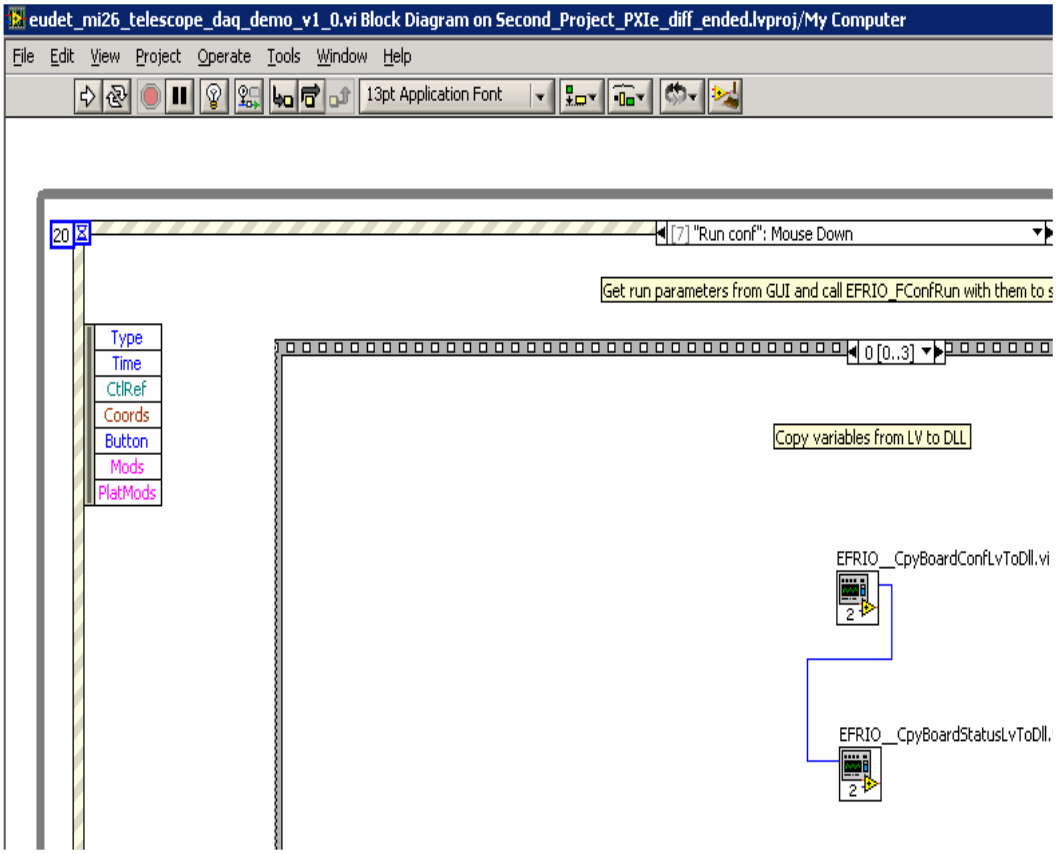


Run configuration → " Run Conf " button

This code sets run configuration.

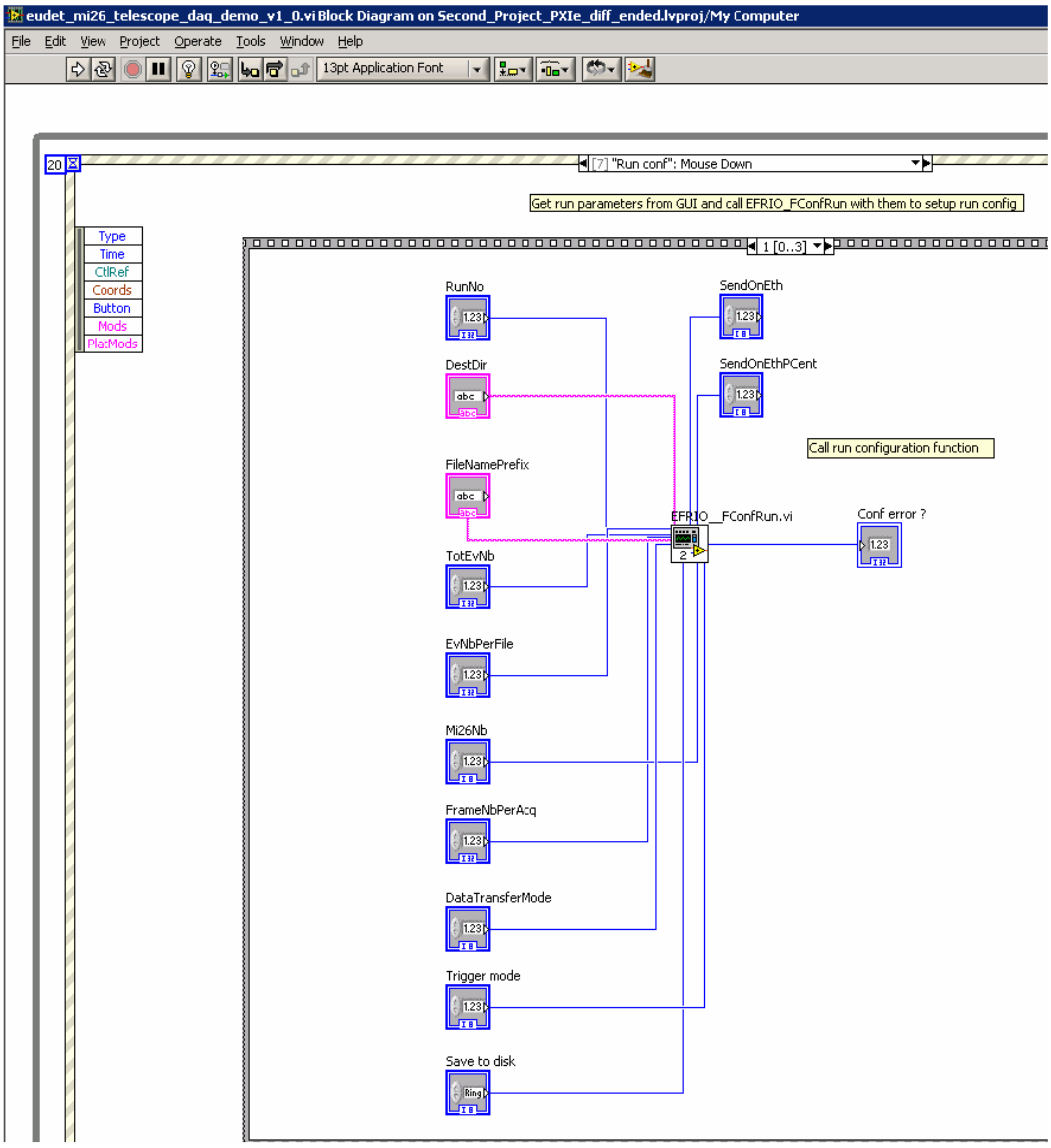
Step 0

It makes a copy of Labview global variables to DLL, because it's the easiest way to get current state of board configuration and status Vi in the DLL.



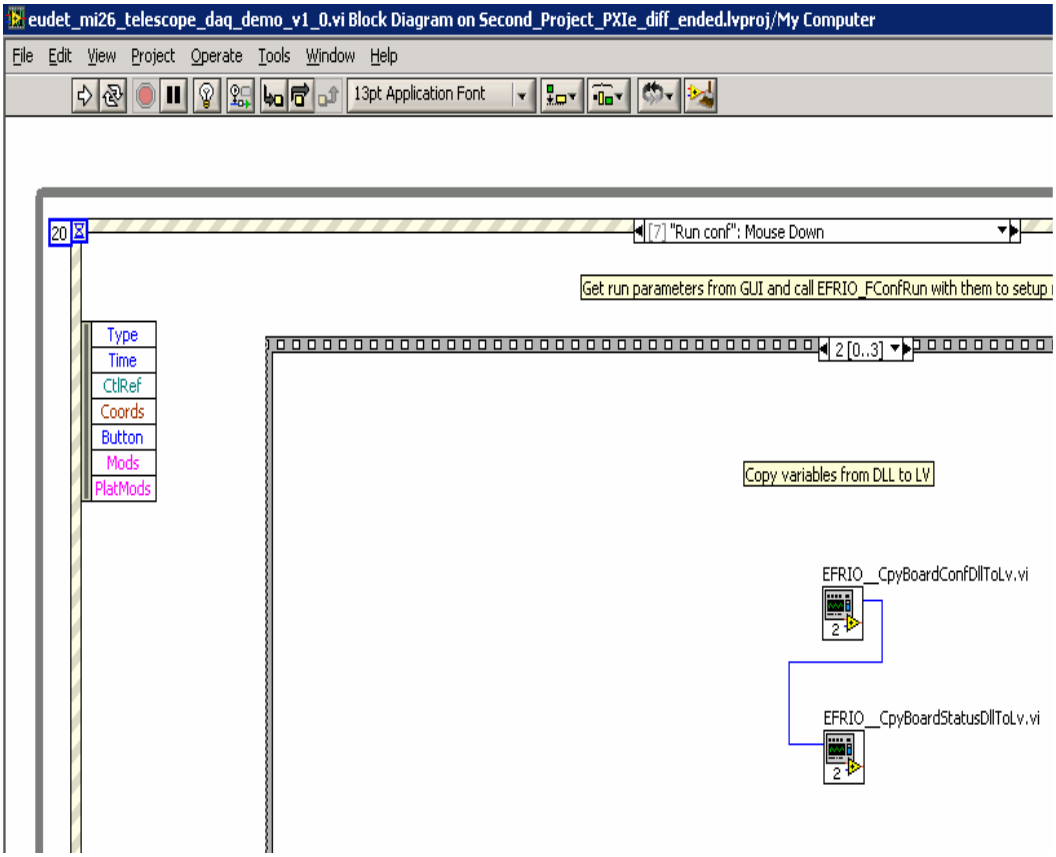
Step 1

It gets run parameters from GUI and call the eudet_frio DLL run configuration function → EFRIO__FConfRun (…).



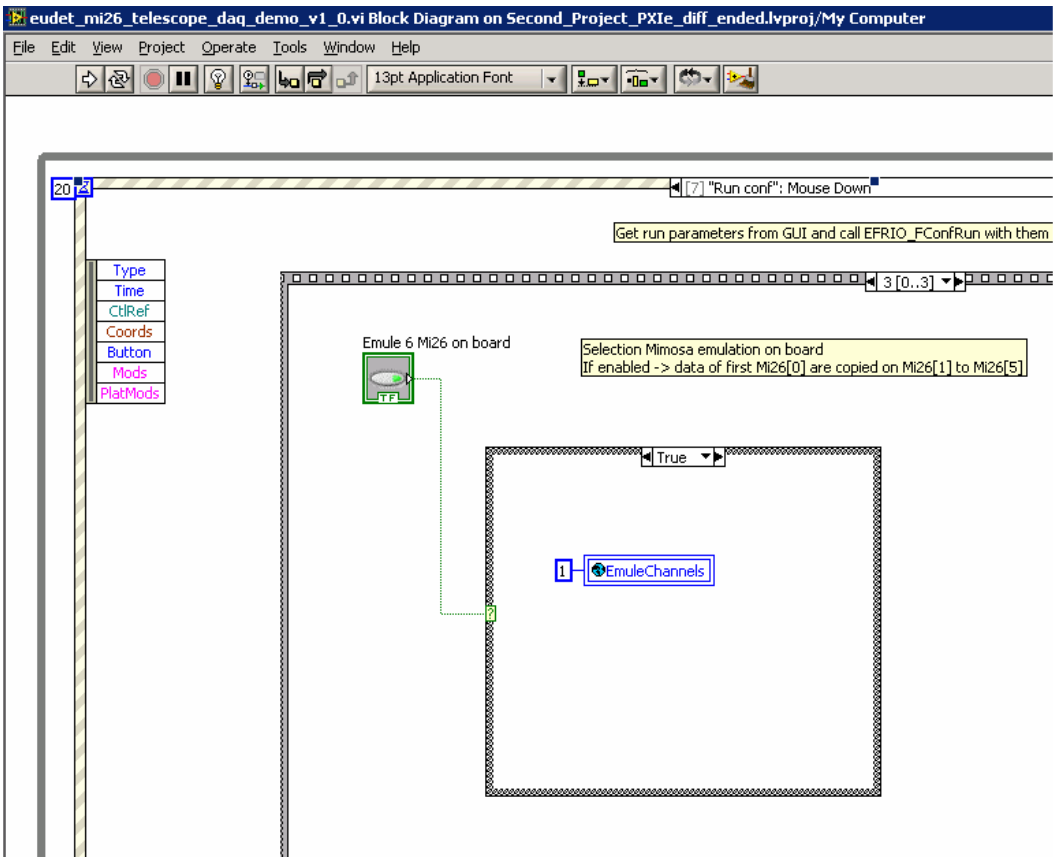
Step 2

It makes a copy of the DLL board configuration and status variables to the Labview global, because it's the easiest way to get current state of board configuration and status from the DLL to Labview. It's the complementary operation of the one done in Step 0.



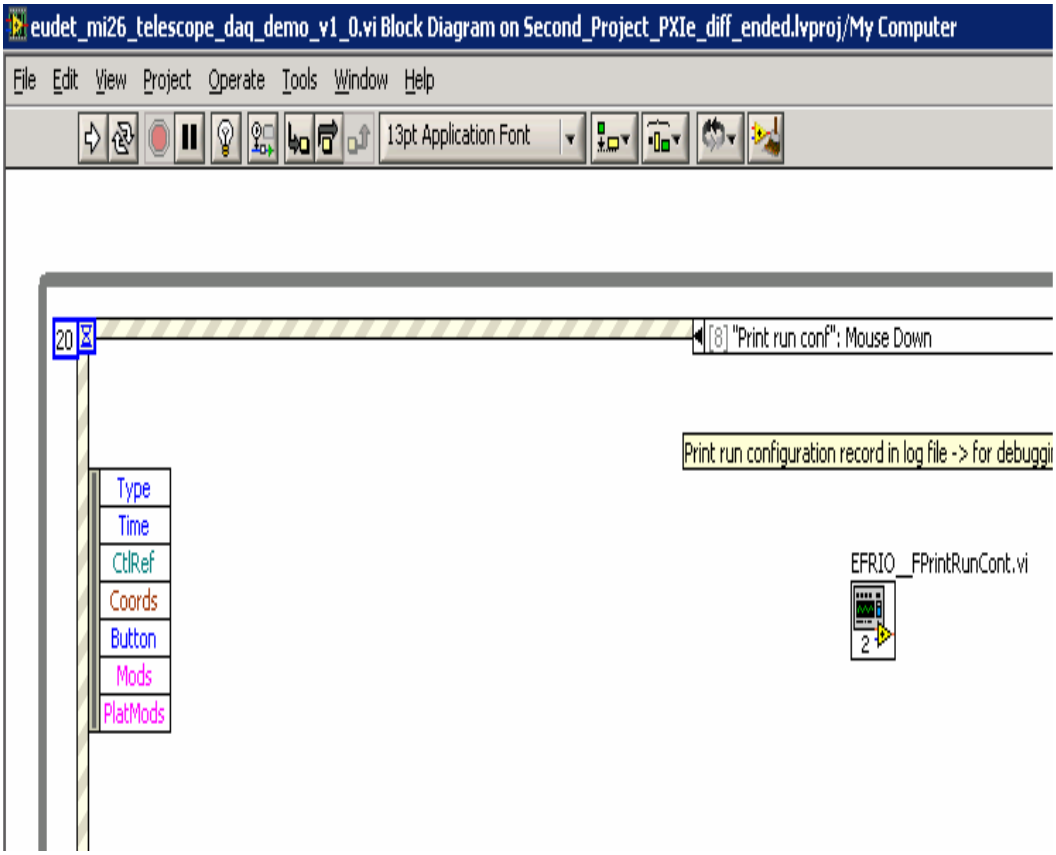
Step 3

It enables or not the emulation of 6 Mimosa 26 on board. It is done by setting the state of the global variable " Emule channels" in function of the switch " Emule 6 Mi26 on board " state.



Print run configuration record in log file → “Print run conf” button

This code prints the run configuration record in log file. It calls the eudet_frio library function EFRIO_FPrintRunCont (…) on a click on button “ Print run conf ”.

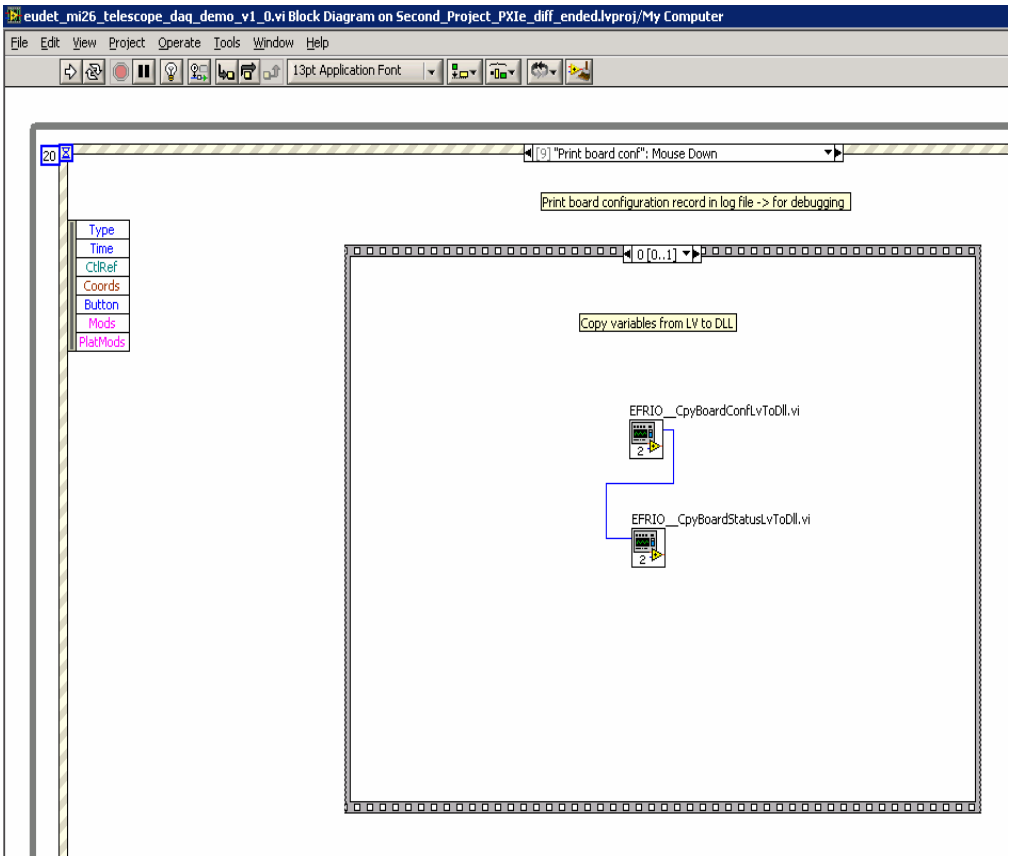


Print board configuration record in log file → "Print board conf" button

This code prints the board configuration record in log file.

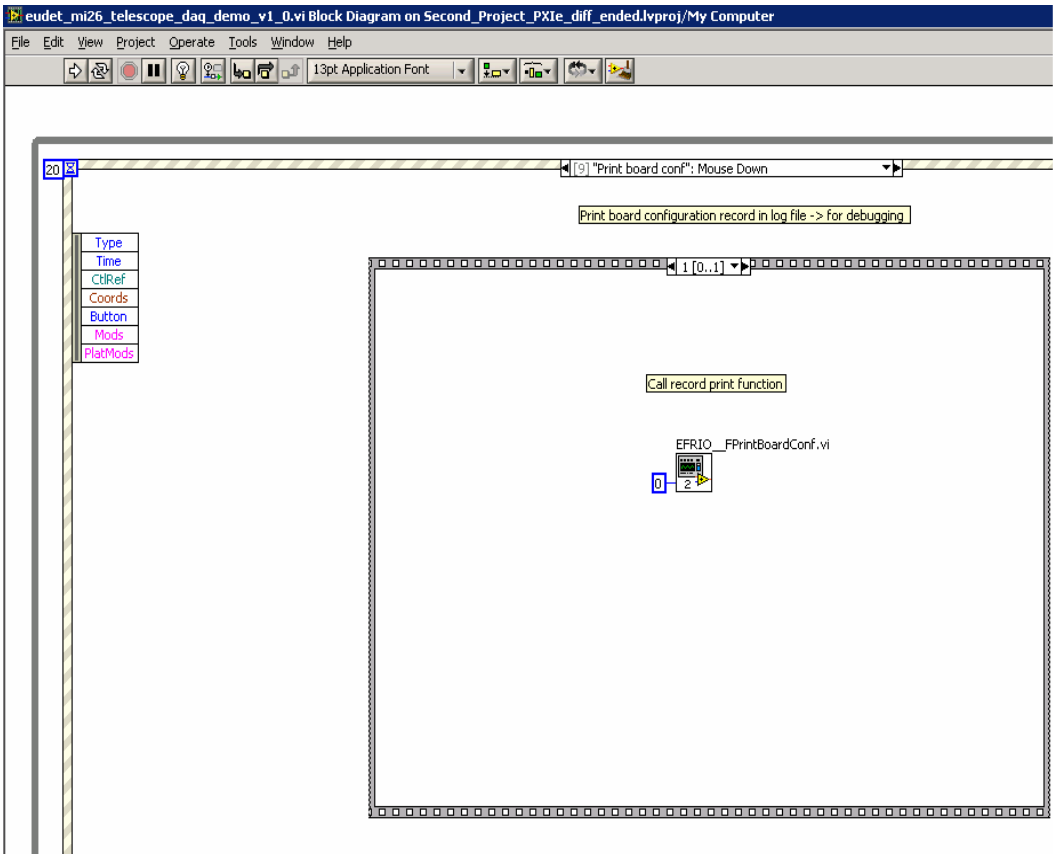
Step 0

Copy Labview global variables to DLL context records.



Step 1

It calls the eudet_frio library function EFRIO_FPrintBoardConf (…) .

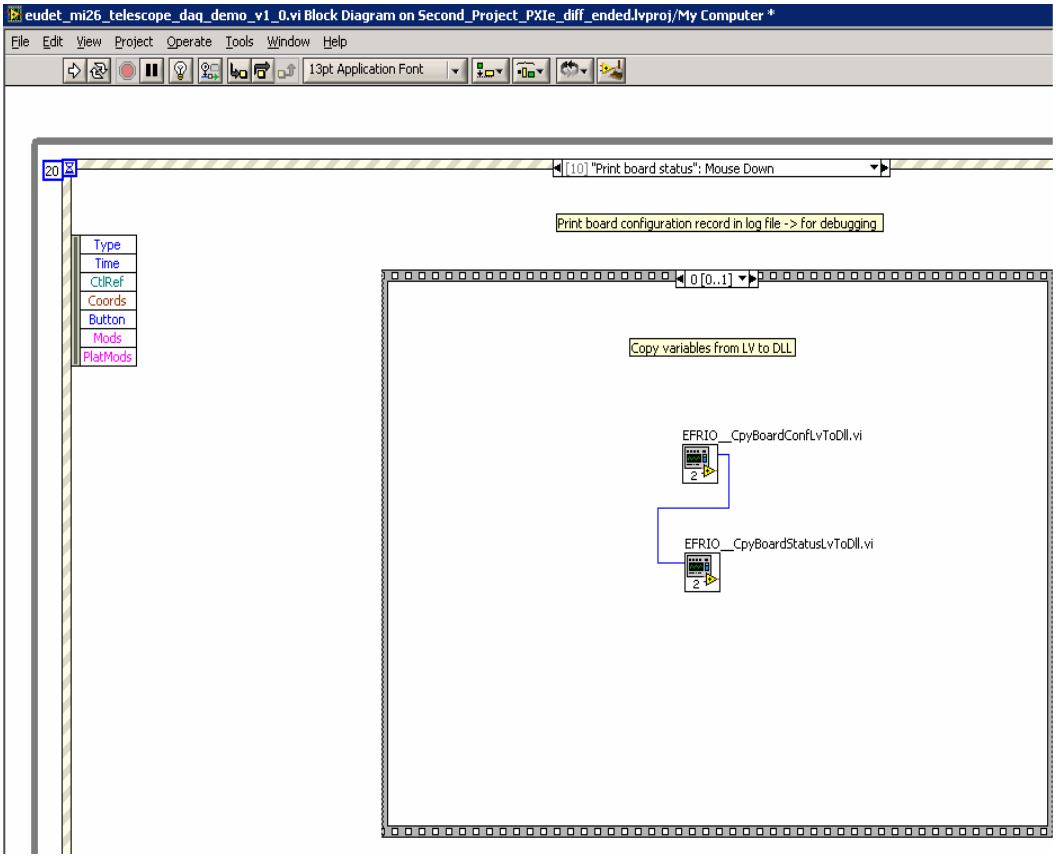


Print board status record in log file → “Print board statusf” button

This code prints the board status record in log file.

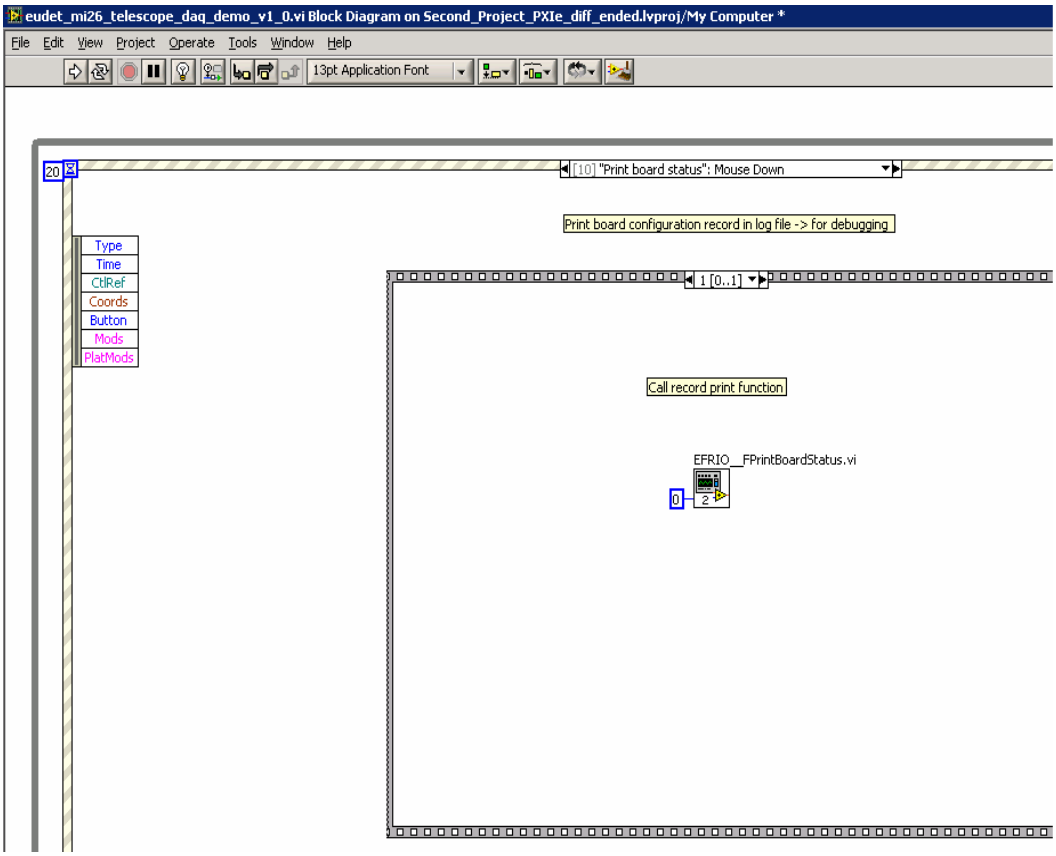
Step 0

Copy Labview global variables to DLL context records.



Step 1

It calls the eudet_frio library function EFRIO_FPrintBoardStatus (…).



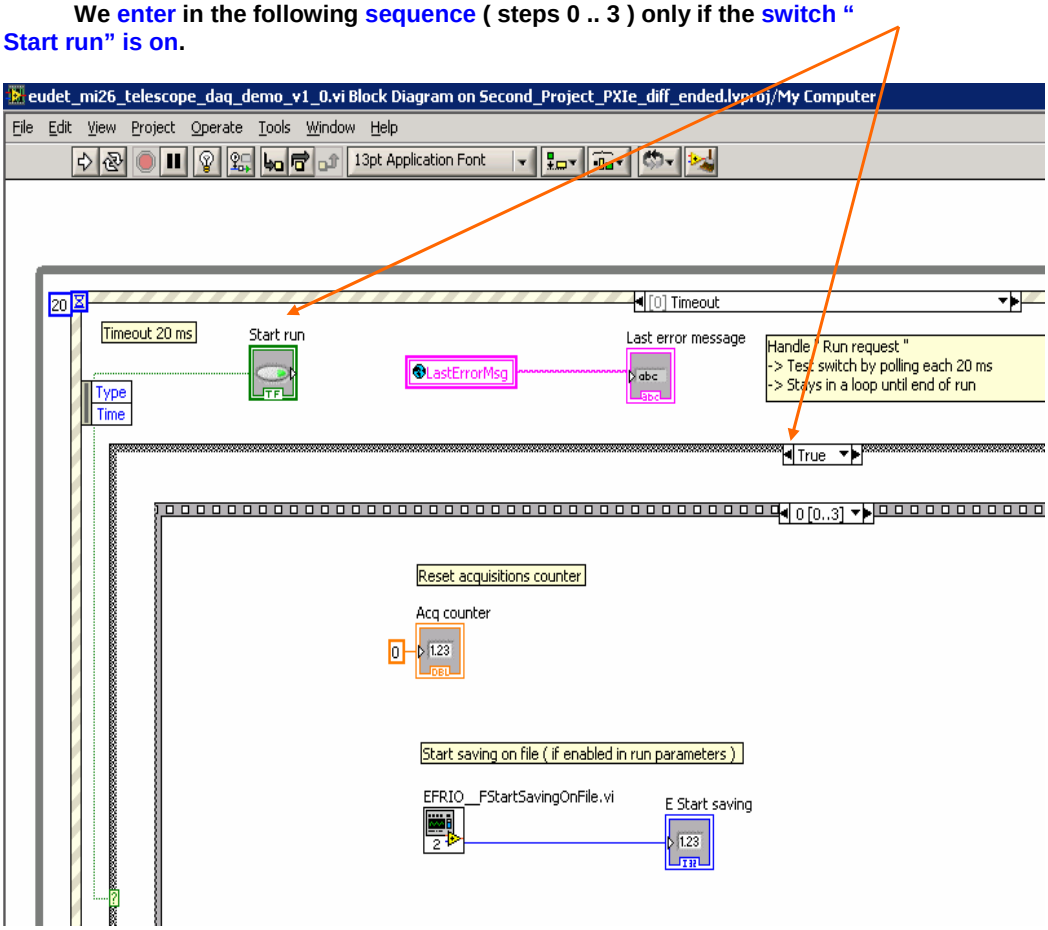
The acquisition code → " Time out " of " Event " structure

This code detects the Start and Stop run commands, controls the Flex RIO board and the on-line monitoring indicators of GUI.

It is " written " in the time out event and not on an event connected to the button " Start run " because it contains the acquisition loop which would lock the event structure.

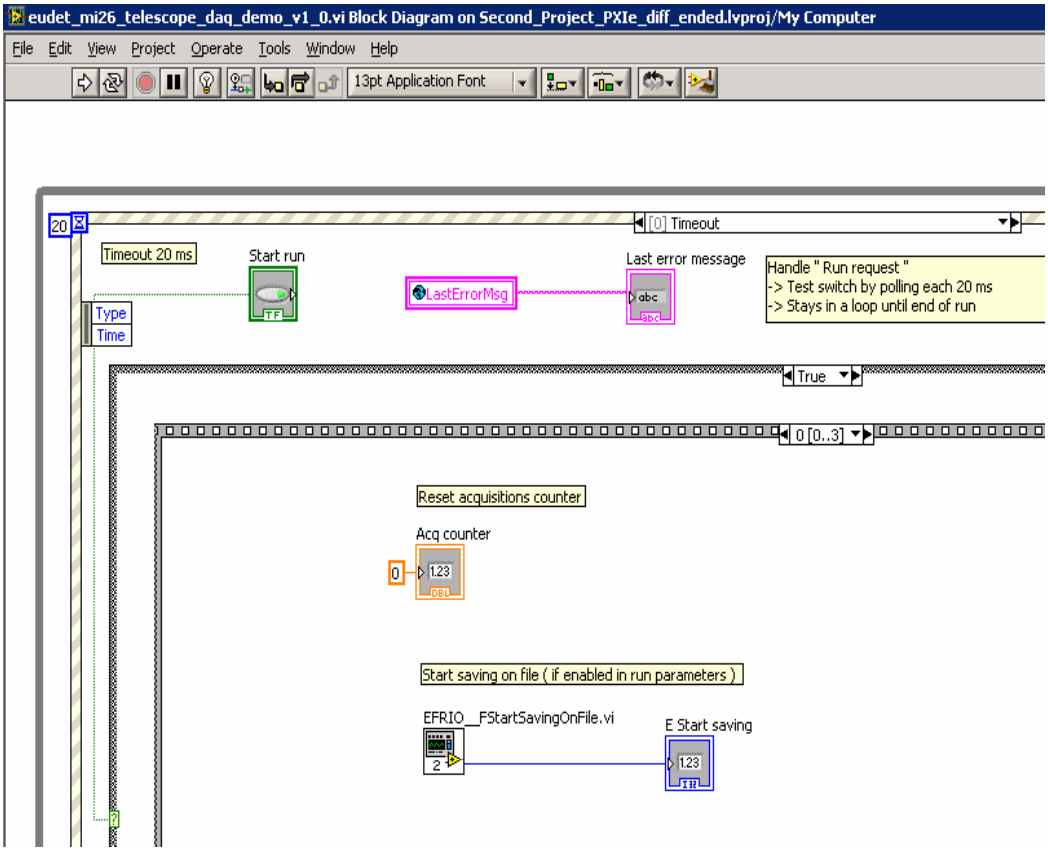
The " If Start run"

We enter in the following sequence (steps 0 .. 3) only if the switch " Start run" is on.



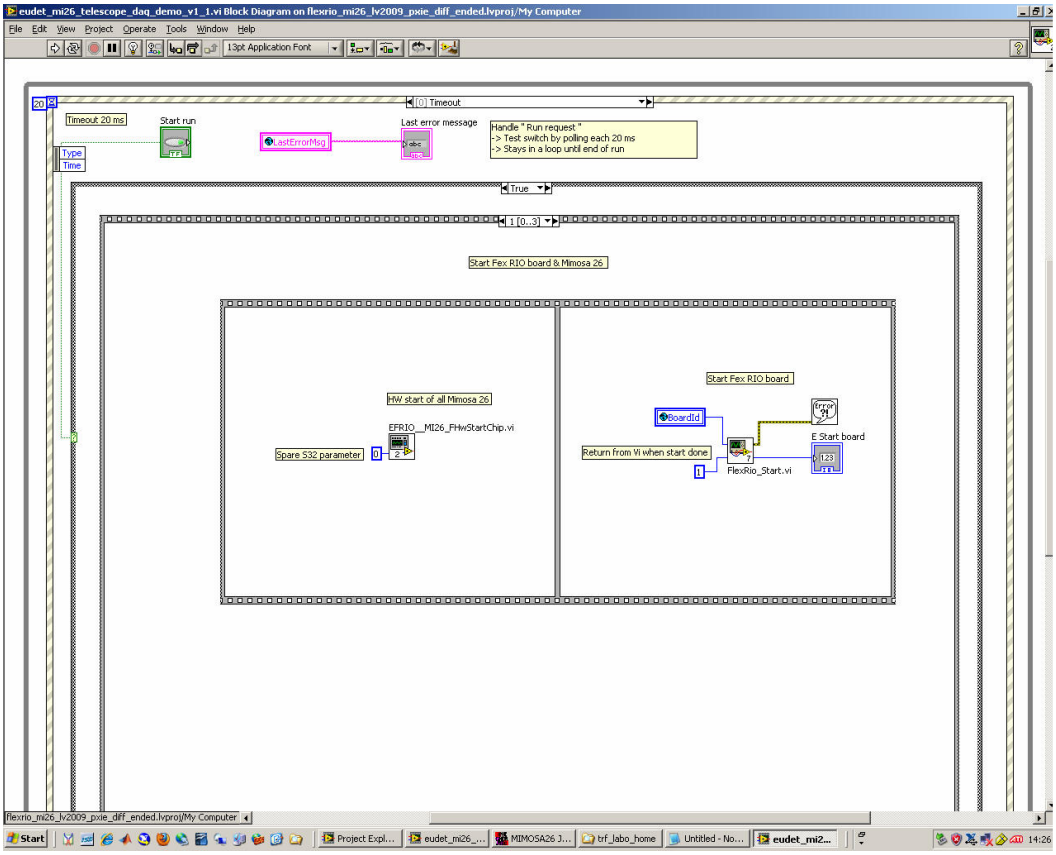
Step 0 → Init

It resets the acquisition counter and call the eudet_frio data saving function (it's behaviour is under run control parameters) → EFRIO__FStartSavingOnFile.



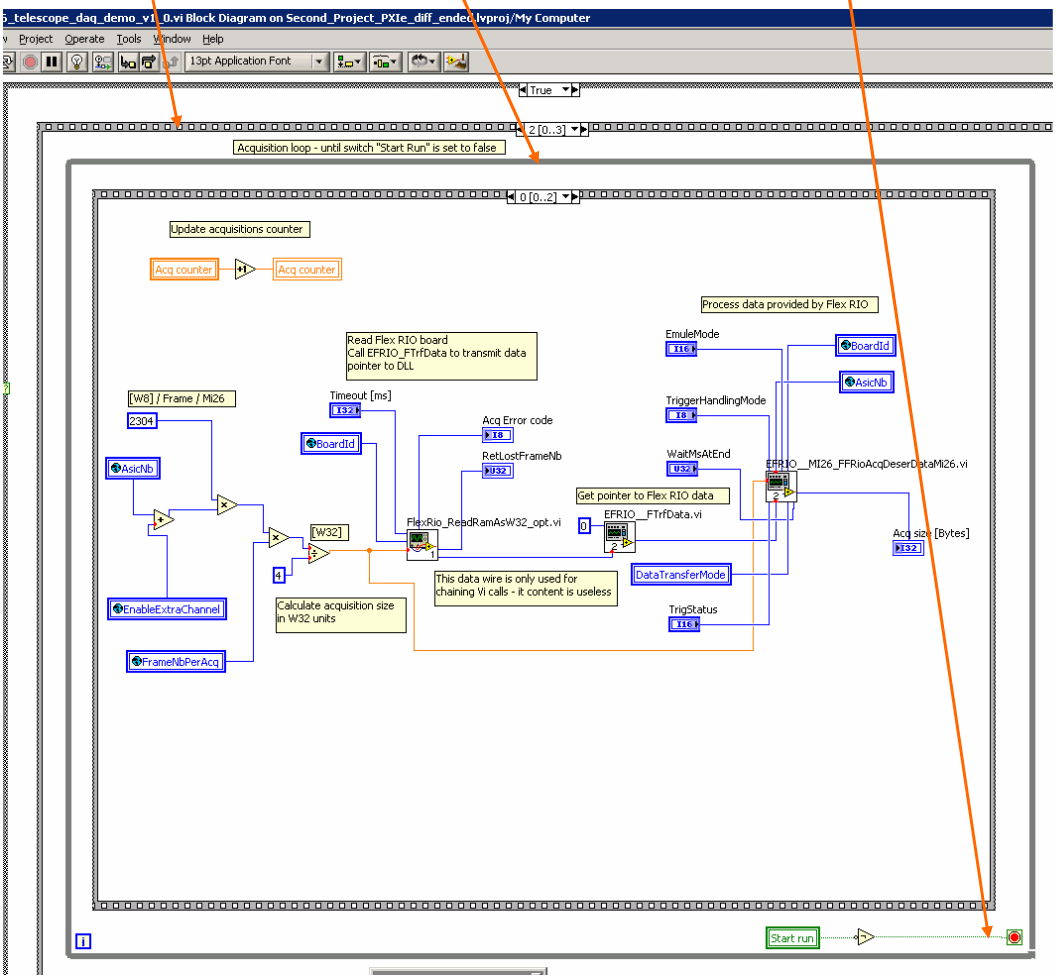
Step 1 → Start FlexRio and Mimosa 26

It starts FlexRio board by a call to Vi " FlexRio_Start ", then it starts Mimosa 26 by a call to the eudet_frio DLL function EFRIO__MI26_FHwStartChip (). This function generates a pulse on parallel port pin D6 which is connected to Mimosa 26 Start input.



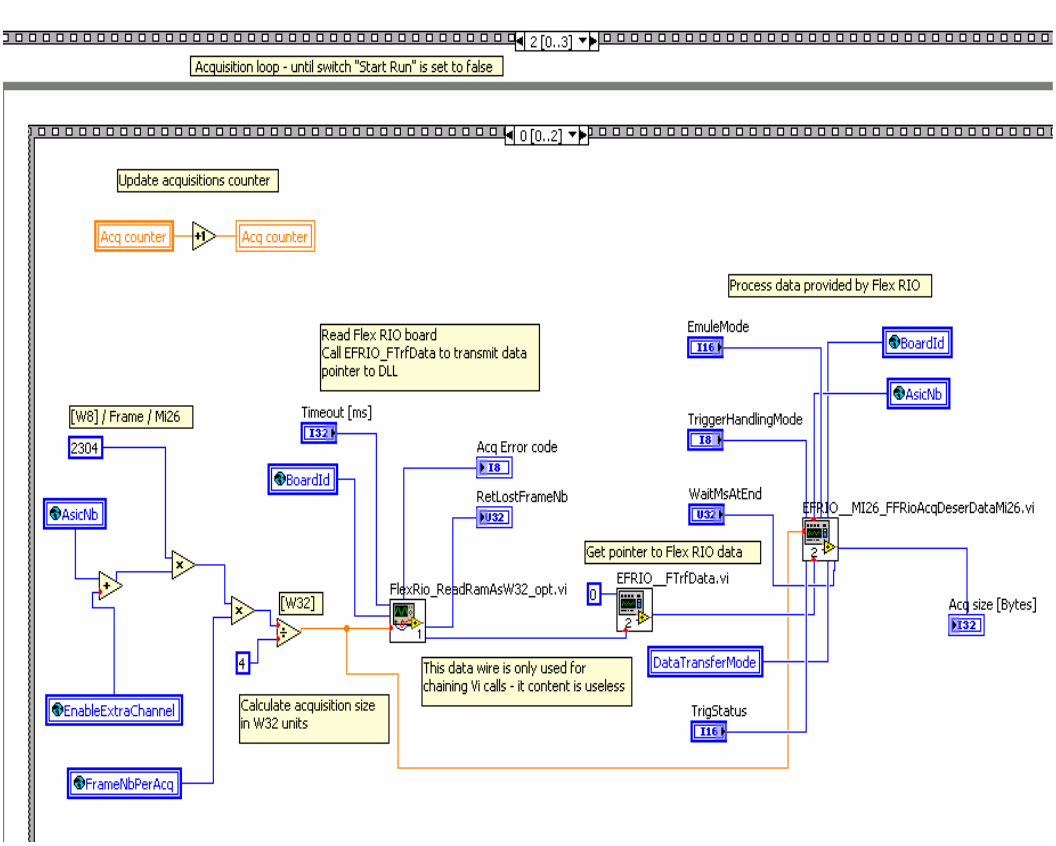
Step 2 → Acquisition loop

This step contains the acquisition loop which will run until " Start run " button become false.



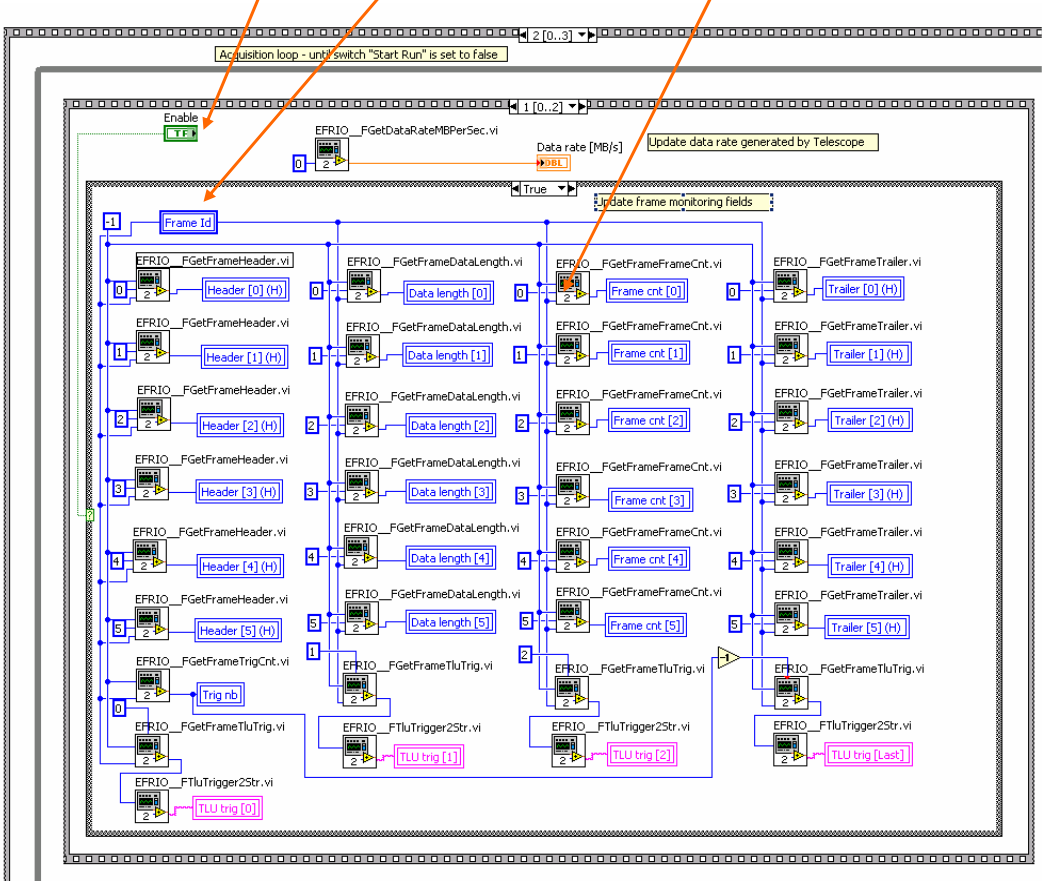
Step 2 (Acquisition loop) – Step 0

This code calls the Vi to read the FlexRio board " FlexRio_ReadRamAsW32_opt.vi " and " pass data " to the eudet_frio DLL data processing function EFRIO__MI26_FFRioAcqDeserDataMi26.



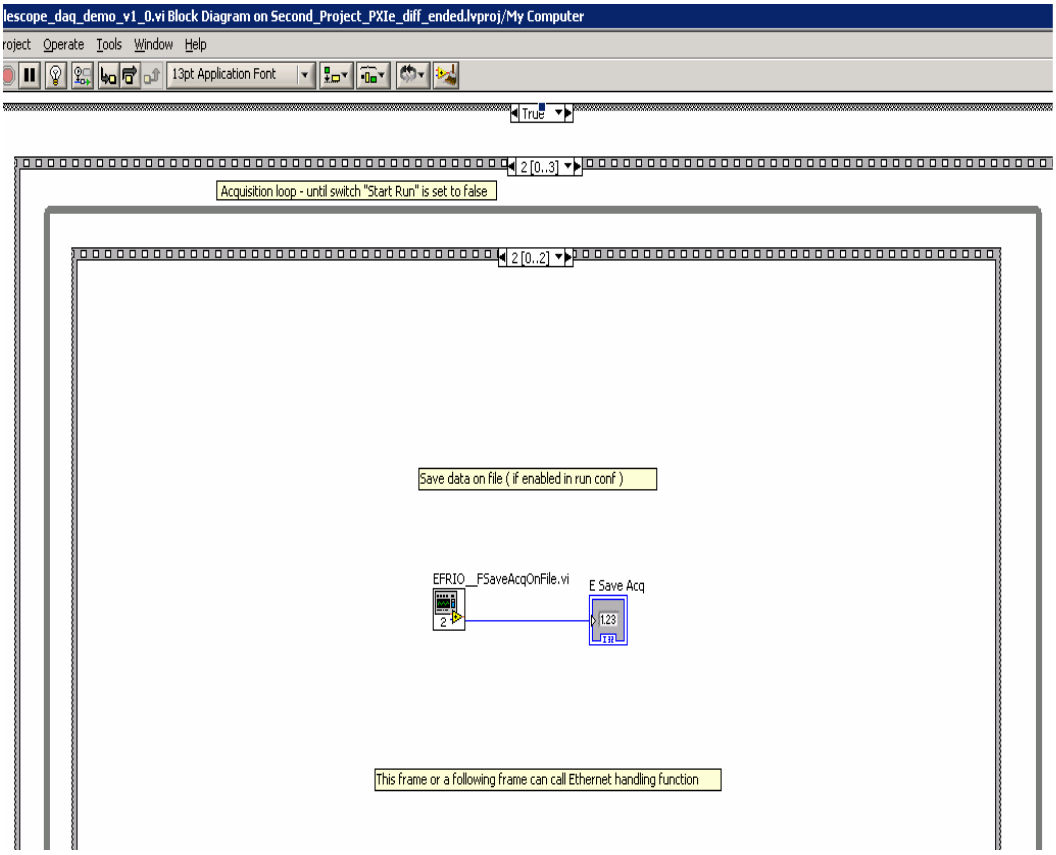
Step 2 (Acquisition loop) – Step 1

If the on-line monitoring is enabled, this codes displays the " relevant "fields (header … trailer) of the Mimosa 26 frame selected. It calls eudet_frio DLL function to get fields values.



Step 2 (Acquisition loop) – Step 2

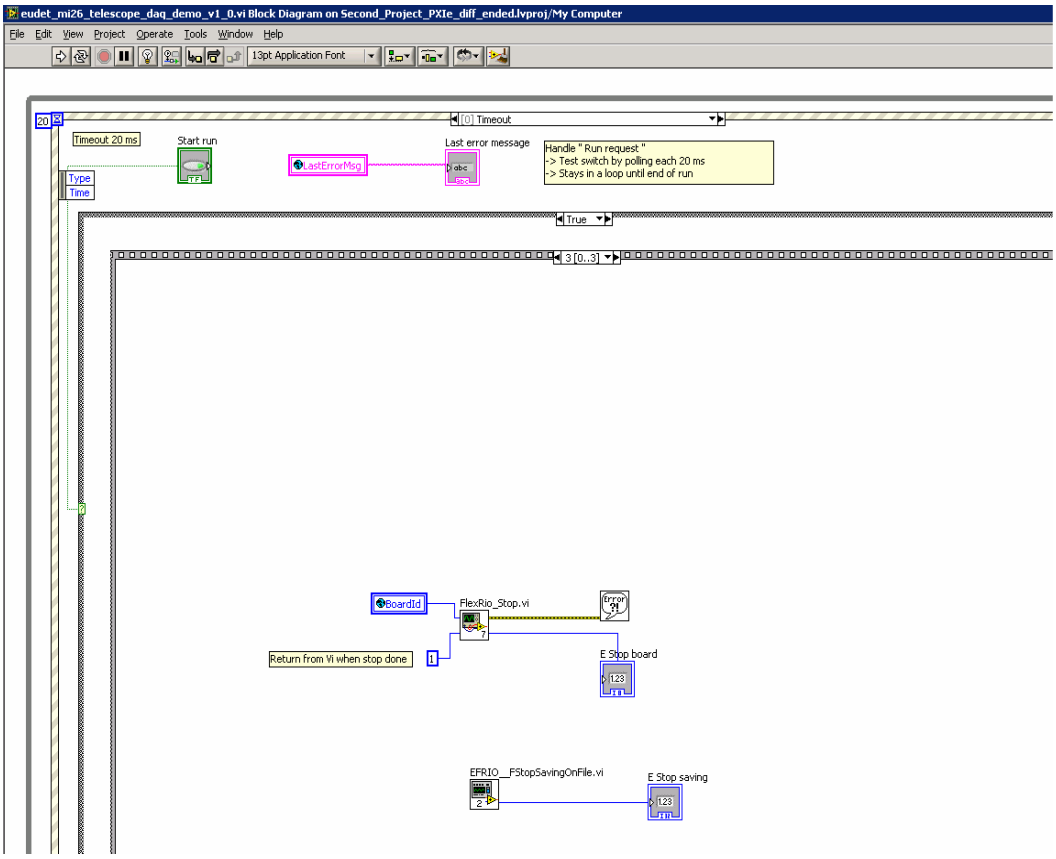
This code calls the eudet_frio DLL data saving function, EFRIO__FSaceAcqOnFile, and it can contain the code to send data on Ethernet to EUDET DAQ.



Step 3

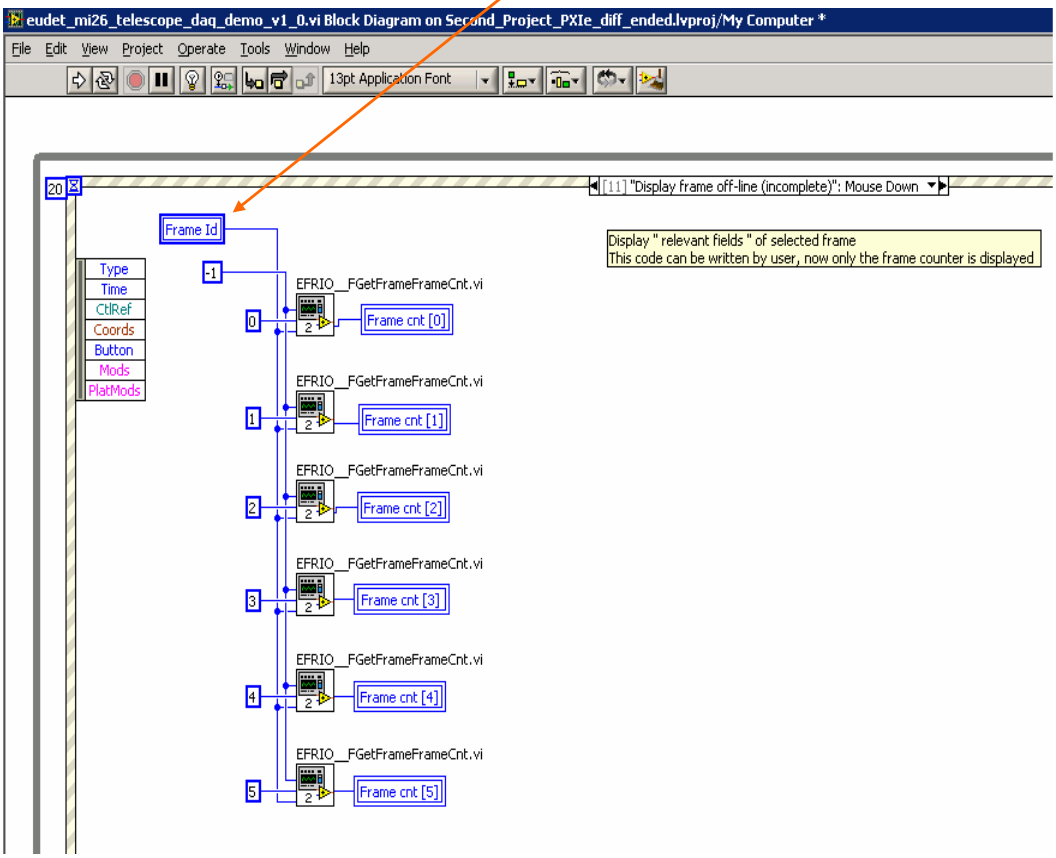
This code stops the acquisition, calls the FlexRio board stop Vi (FlexRio_Stop) and the eudet_frio DLL data saving stop function → EFRIO__FStopSavingOnFile (…).

The good question is " how do we go there ? " as there is no test on " Start run " button … We should not forget that in previous step (No 2) we were in a loop and the exit condition was " Start run" = FALSE.



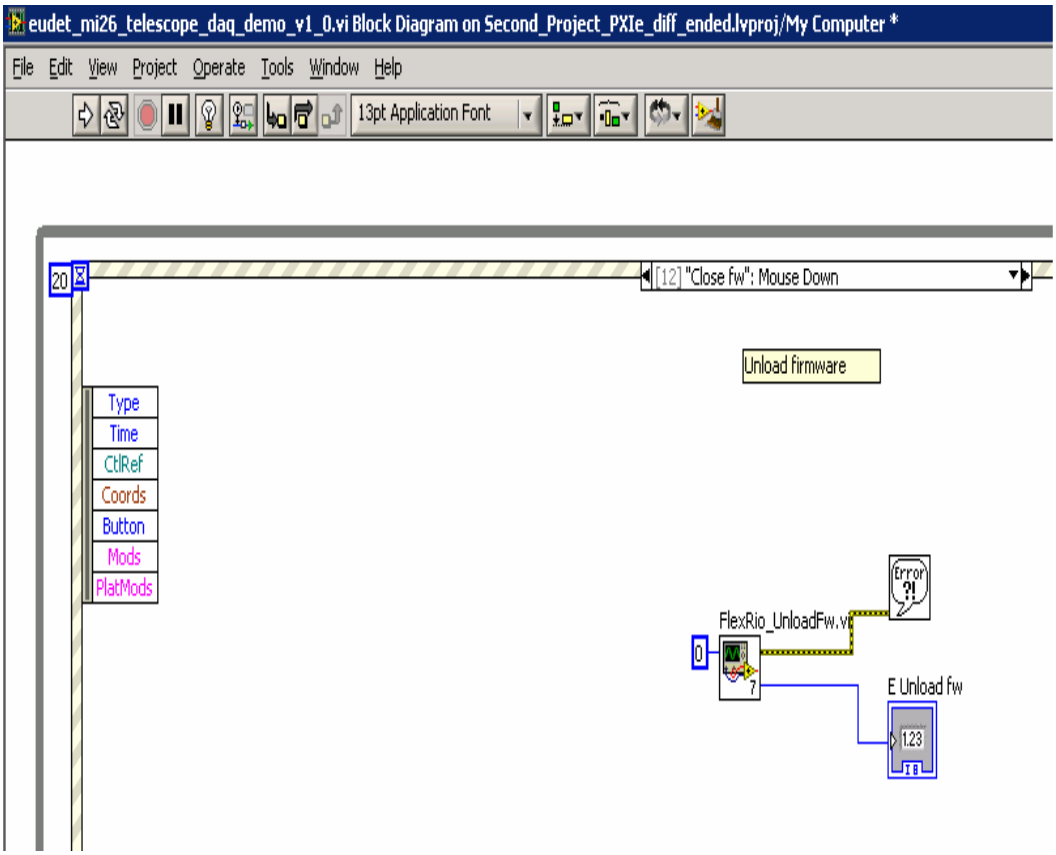
Display frames off-line → “Display frame off-line …” button

This code displays the frame selected by “ Frame Id ” when user clicks on the button “ Display frame off-line … ”. But he code is incomplete, only frame counter is displayed, the user can write the missing code as an exercise.



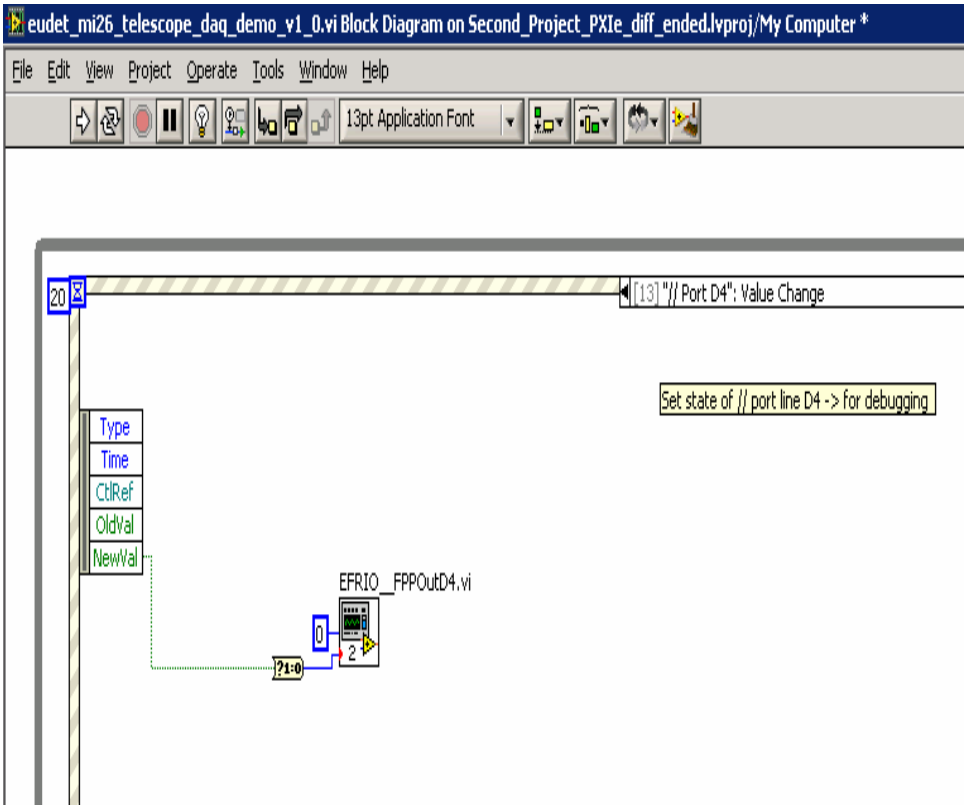
Close firmware → “Close fw” button

This code closes the firmware, it calls the Vi FlexRio_UnloadFw.



Control parallel port pin D4 → "// Port D4" button

This code controls the state of the pin D4 of parallel port. It call the eudet_frio DLL function EFRIO__FPPOutD4. The same code exists for D5 and D6.



Acknowledgement

This work is supported by the Commission of the European Communities under the 6th Framework Programme "Structuring the European Research Area", contract number RII3-026126.