EUDET-Memo-2010-25


# IPHC & NI Flex RIO DAQ for EUDET Mimosa 26 Beam Telescope

JRA1: Strasburg Group

January 17, 2011

## 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.

# IPHC & NI Flex RIO DAQ for EUDET Mimosa 26 Beam Telescope
## Planning of DAQ training – IPHC Strasbourg December 2010

# OUTLINE

- Why a DAQ based on NI COTS → Flex RIO board
  - System architecture & cost / EUDRB solution
- Flex RIO evaluation done at IPHC → June 2010 beam test @ CERN
  - Check that board + fw able to read Mimosa 26 without loosing frames
  - Evaluation of the bandwidth required to transmit data ( in real conditions )
- Our proposal for Flex RIO integration in EUDET DAQ
  - Firmware, software, interface to TLU → Labview application + DLL
  - To be done by collaboration → Interface Application / EUDET DAQ via Ethernet
- The DAQ emulator & Eudet flex RIO library ( C code )
- The DAQ Labview application
- Planning of the week

# Why a DAQ based on COTS : Save Human resources & Money

## Easier to make and maintain copies of Telescope DAQ

- No need of acquisition boards production
- No need of acquisition boards testing
- Availability of boards → Replacement of broken boards

## Performances & cost point of view

- **In worst case ( Mimosa 26 frames full ) we MUST sustain 6 Mimosa 26 x 20 MB/s = 120 MB/s**
  - **EUDRB VME → Max ~ 80 MB/s – Flex RIO PXIe board ~ 800 MB/s**
- **Cost**
  - **EUDRB cost ~ 2000 € x 6 boards = 12 000 € - Need only one Flex RIO + Adaptator module ~ 6000 €**
  - **Cost of PXIe crate + CPU / VME carte + CPU ~ the same → ~ 3000 € + 5500 € = 8500 €**
  - **Labview software ( FPGA + standard ) → ~ 2 x 2300 € = 4600 € ( But LynxOS – VxWorks are not for free )**

## Conclusion

- **Flex RIO is a viable solution on technical point of view**
- **The cost should be the same or less ( we need Labview SW only on the development system ☺ )**

# Why a DAQ based on COTS : The Flex RIO Board

## NI FlexRIO FPGA Modules

### NI PXI-795xR, NI PXIe-796xR *NEW!*

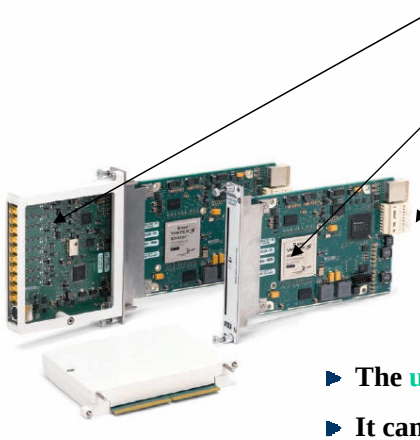
- NI FlexRIO FPGA modules
- PXI and PXI Express
- Xilinx Virtex-5 SXT and LX FPGAs
- Programmable with the LabVIEW FPGA Module
- Up to 512 MB onboard DRAM
- Peer-to-peer data streaming between PXI Express modules at more than 800 MB/s
- Up to 16 DMA channels
- 132 single-ended or 66 differential data lines to adapter module interface
- Up to 66 Gbits/s adapter module bandwidth
- Multi-adapter module synchronization for high-channel-count applications
- Adapter modules available from NI and third parties as well as through custom development with the NI FlexRIO Adapter Module Development Kit (MDK)

**Operating Systems**
- Windows 7/Vista/XP/2000
- LabVIEW Real-Time

**Required Software**
- LabVIEW
- LabVIEW FPGA Module

**Recommended Software**
- NI FlexRIO Adapter Module Development Kit

**Driver Software**
- NI-RIO
- NI FlexRIO adapter module support



| Model | Bus/Form Factor | FPGA | FPGA Slices | FPGA DSP Slices | FPGA Memory (Block RAM) | Onboard Memory (DRAM) |
|---|---|---|---|---|---|---|
| NI PXIe-7965R | PXI Express | Virtex-5 SX95T | 14,720 | 640 | 8,784 kbits | 512 MB |
| NI PXIe-7962R | PXI Express | Virtex-5 SX50T | 8,160 | 288 | 4,752 kbits | 512 MB |
| NI PXIe-7961R | PXI Express | Virtex-5 SX50T | 8,160 | 288 | 4,752 kbits | 0 MB |
| NI PXI-7954R | PXI | Virtex-5 LX110 | 17,280 | 64 | 4,608 kbits | 128 MB |
| NI PXI-7953R | PXI | Virtex-5 LX85 | 12,960 | 48 | 3,456 kbits | 128 MB |
| NI PXI-7952R | PXI | Virtex-5 LX50 | 7,200 | 48 | 1,728 kbits | 128 MB |
| NI PXI-7951R | PXI | Virtex-5 LX30 | 4,800 | 32 | 1,152 kbits | 0 MB |

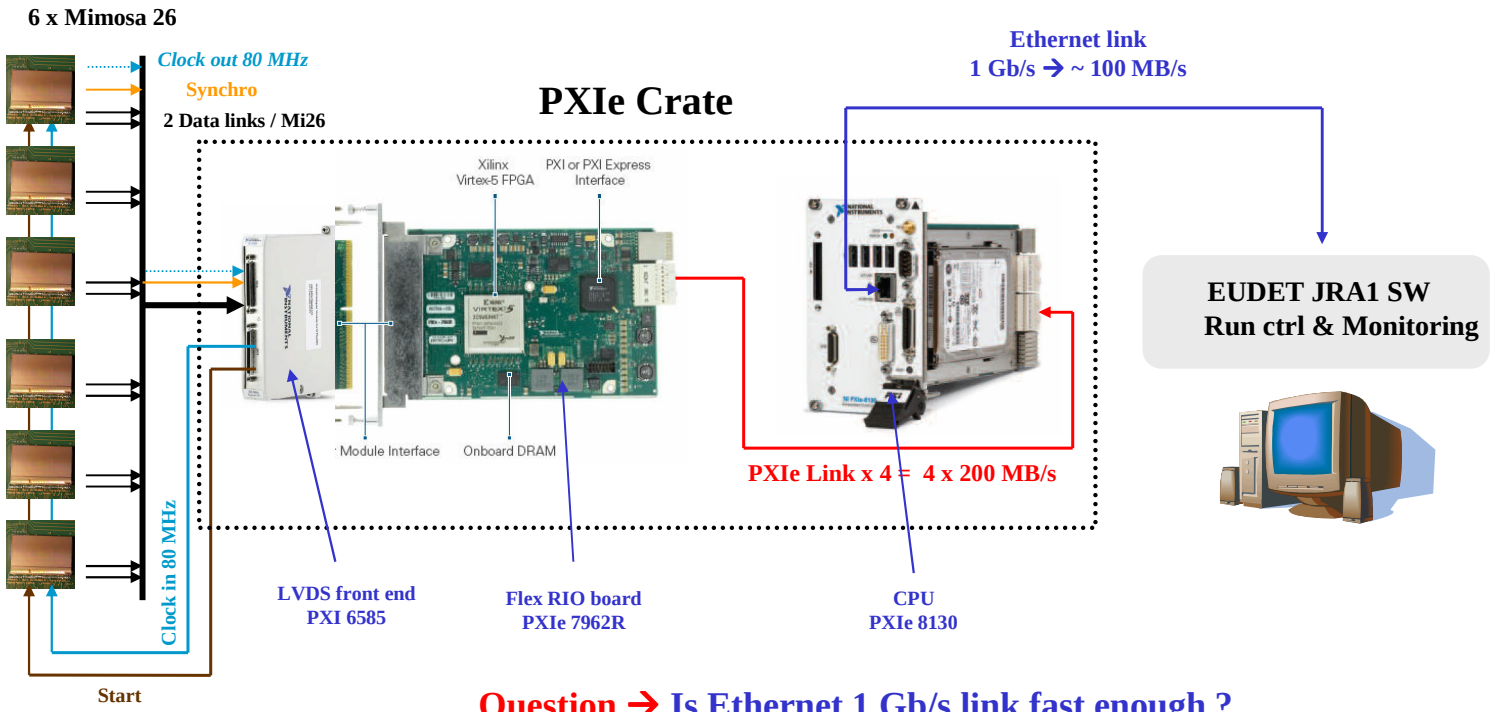
## Main characteristics

- **66 Differential inputs**
- **User defined Adaptator Module**
  - **LVDS I/O module → NI 6585**
- **User defined on-board fw**
- **PXIe bus x 4**
  - **~ 200 MB/s / Lane**
  - **4 lanes / board = ~ 800 MB/s ?**

## Main advantage

- **The user can develop his own firmware**
- **It can be written in Labview FPGA & VHDL**

# Why a DAQ based on COTS : System architecture



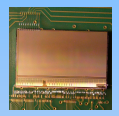

## Question → Is Ethernet 1 Gb/s link fast enough ?

- To sustain 6 x Mimosa 26 ( full frames ) @ 8680 frames /s ~ 120 MB/s → No !
  - Ex : Acq 6 Mi26 x 1800 frames / 207 ms @ 200 MB/s = 119 ms → 88 ms for Ethernet → 270 MB/s >> 1 Gb/s
- But Mimosa 26 frames may not be full & trigger rate not 8680 Hz
  - We must evaluate data throughput needed in real beam condition

# Flex RIO evaluation : Summer 2010 beam test

## From the idea …

**MAPS = Capteur à pixels**
**→ position des particules**

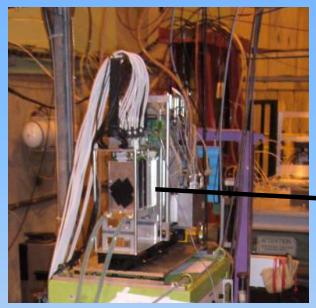


**Mimosa 26**
**~ 21 mm x 10 mm**
**600 K Pixels**
**Pixels 18,4 x 18,4 µm**

**NI Module**
**32 I/O LVDS**

**Flex RIO Board**
**PXIe 7962R**

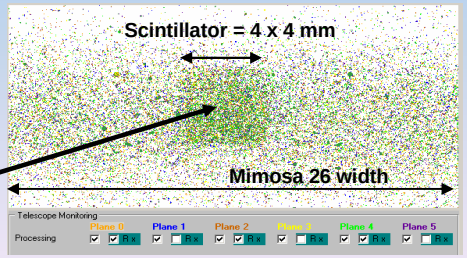
## to the beam test area …



***Beam Telescope***
***6 Mimosa 26 → 120 MB/s***

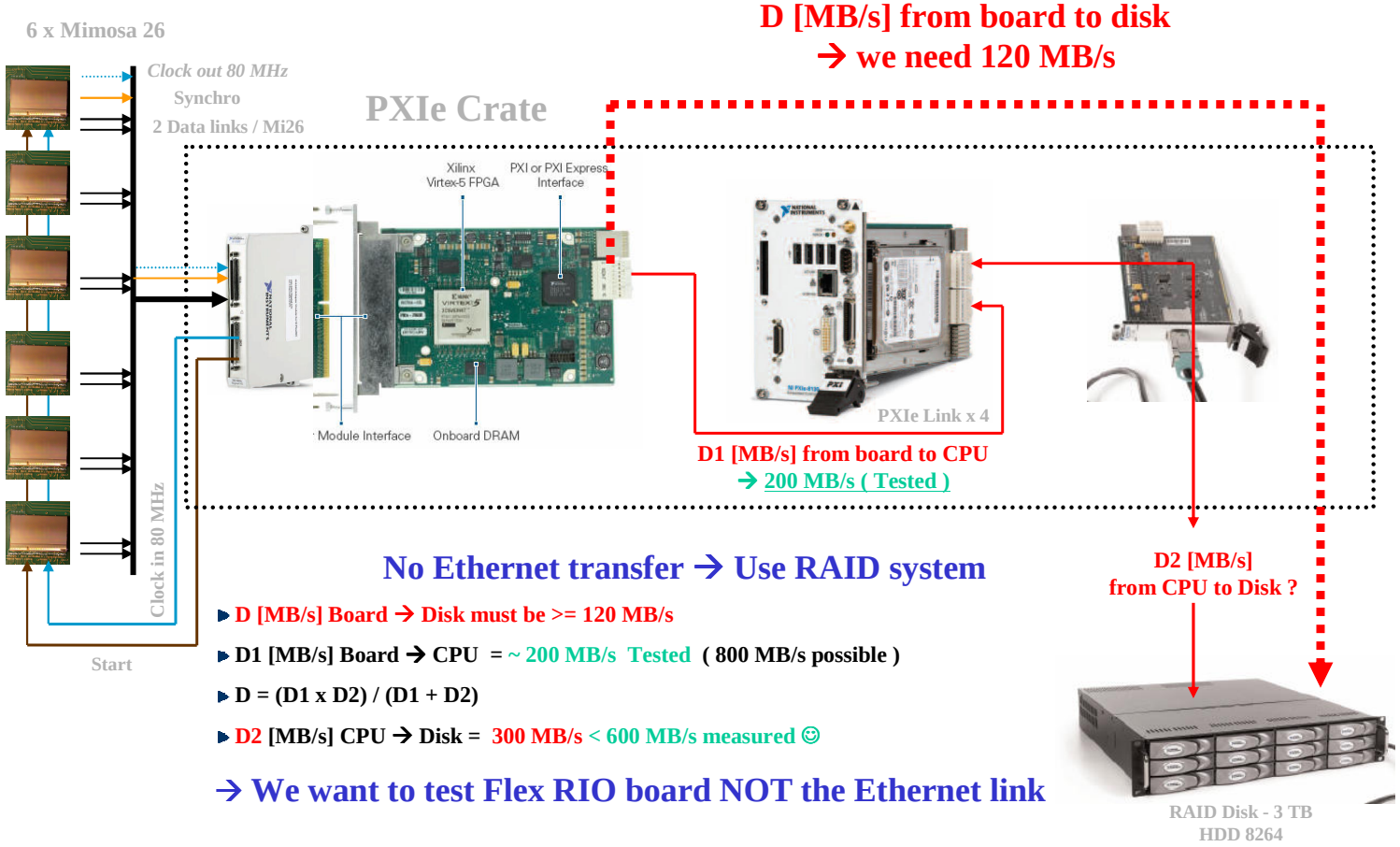
***DAQ : Flex RIO - PXIe***

***Scintillator shadow***




***DAQ Monitoring → Hits cumul 6 planes***

# Flex RIO evaluation : System architecture used



# Flex RIO evaluation : Beam test results

## Tests condition

- **Readout of 6 x Mimosa 26 → Two act as a DUT**
- **Scintillator 7 mm x 7 mm – Trigger rate 3,3 Khz & 8,3 KHz**
- **Continuous readout → Take all frames during spill ( spill detected by polling a HW signal )**

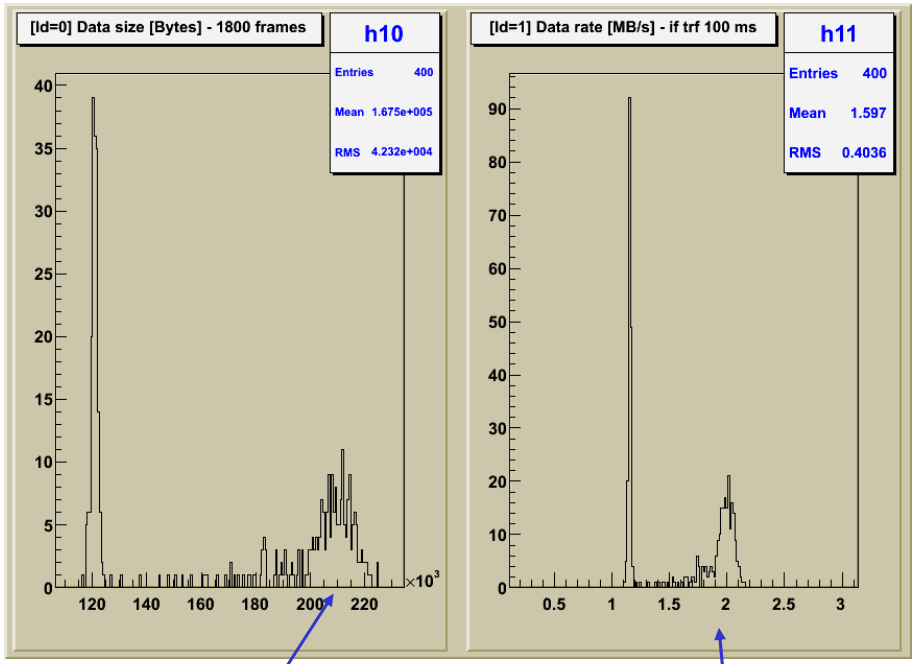
## Results on Physics point of view

- **No change on results / test with old PXI system ( 94 % dead time )**
- **No strange behaviour of sensors**

## Results on DAQ point of view

- **Run of 400 acquisitions of each 1800 frames taken → 720 000 frames**
- **No frame lost, no data corruption ( testing frame header, frame cnt, trailer )**
- **Evaluation of real data throughput → Next slide**

# Flex RIO evaluation : Data throughput in beam test conditions




Average of maximum data size
for 1800 frames
~ 210 KB

Required BW
if we have 100 ms to transfer data
~ 2 MB/s

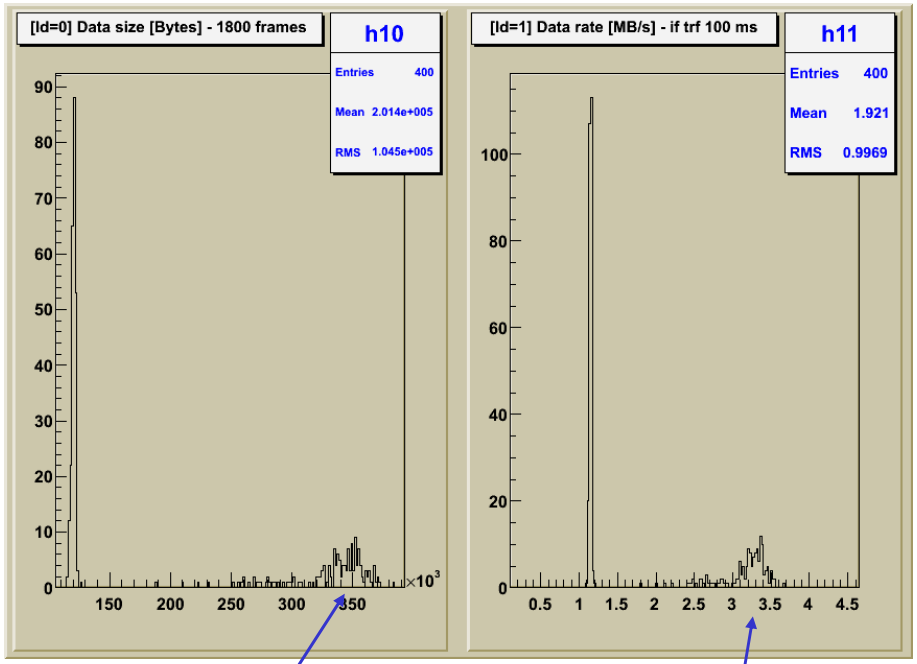
## Test conditions

- 6 x Mimosa 26
- Scintillator 7 mm x 7 mm
- Trigger rate 3,2 Khz
- Mimosa 26 threshold S/N = 8
- Measure data size of 1800 consecutive frames
  - Real data size → frame DataLength field

## Test results

- Maximum size is ~ 210 KB
- Ethernet BW to transfer data in 100 ms
  - ~ 2 MB/s < 1 Gb/s ~ 100 MB/s
- Why transfer data in 100 ms ?
  - Free time evaluated on our system
  - 1800 frames = 207 ms cycle
  - ~ 110 ms free time available

# Flex RIO evaluation : Data throughput in beam test conditions




Average of maximum data size
for 1800 frames
~ 340 KB

Required BW
if we have 100 ms to transfer data
~ 3,4 MB/s

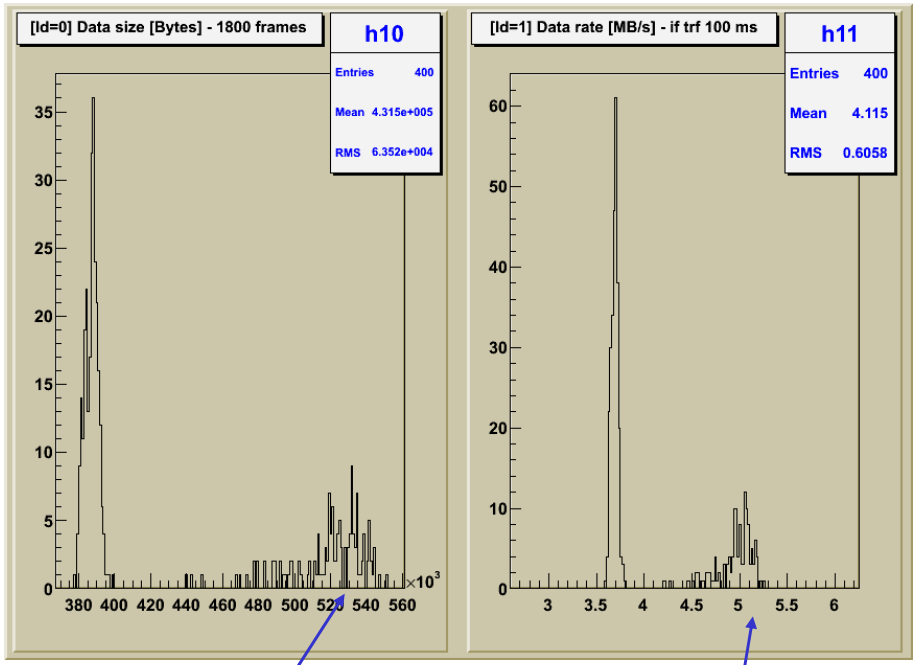
## Test conditions

- 6 x Mimosa 26
- Scintillator 7 mm x 7 mm
- Trigger rate 8,3 Khz
- Mimosa 26 threshold S/N = 8
- Measure data size of 1800 consecutive frames
  - Real data size → frame DataLength field

## Test results

- Maximum size is ~ 340 KB
- Ethernet BW to transfer data in 100 ms
  - ~ 3,4 MB/s < 1 Gb/s ~ 100 MB/s
- Why transfer data in 100 ms ?
  - Free time evaluated on our system
  - 1800 frames = 207 ms cycle
  - ~ 110 ms free time available

# Flex RIO evaluation : Data throughput in beam test conditions




Average of maximum data size
for 1800 frames
~ 530 KB

Required BW
if we have 100 ms to transfer data
~ 5,1 MB/s

## Test conditions

- 6 x Mimosa 26
- Scintillator 7 mm x 7 mm
- Trigger rate 5 Khz
- Mimosa 26 threshold S/N = 8 – 1 DUT S/N = 5
- Measure data size of 1800 consecutive frames
  - Real data size → frame DataLength field

## Test results

- Maximum size is ~ 530 KB
- Ethernet BW to transfer data in 100 ms
  - ~ 5,1 MB/s < 1 Gb/s ~ 100 MB/s
- Why transfer data in 100 ms ?
  - Free time evaluated on our system
  - 1800 frames = 207 ms cycle
  - ~ 110 ms free time available

# Conclusion

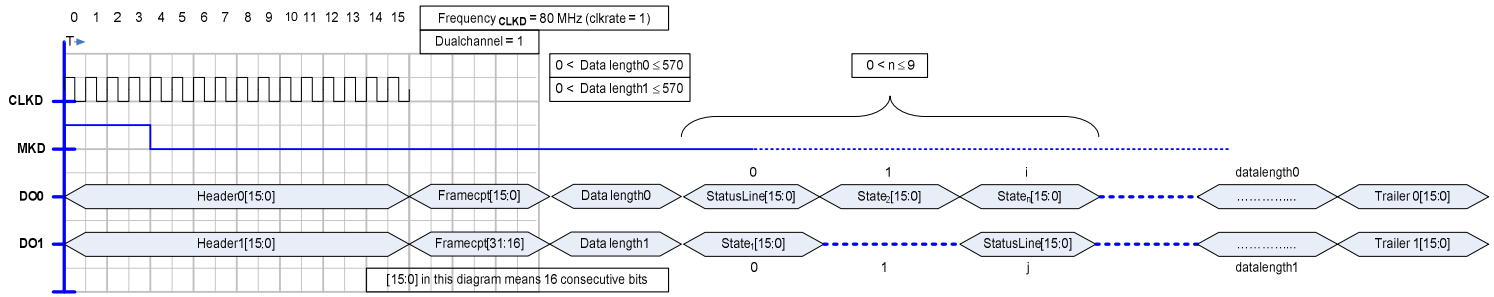
- For a reference plane we will cut at S/N = 8
- This means a maximum data size of up to 340 KB / 1800 frames
- Which requires an Ethernet BW of 3,4 MB/s to transfer data during 100 ms maximum
- It's far below a 1 Gb/s Ethernet specifications → BW limit ~ 100 MB/s

→ There should be no problem to transmit data over 1 Gb/s Ethernet

# Flex RIO DAQ proposal : Mimosa 26 data stream

## Readout configuration N° 3 : 2 serial outputs @ 80 MHz

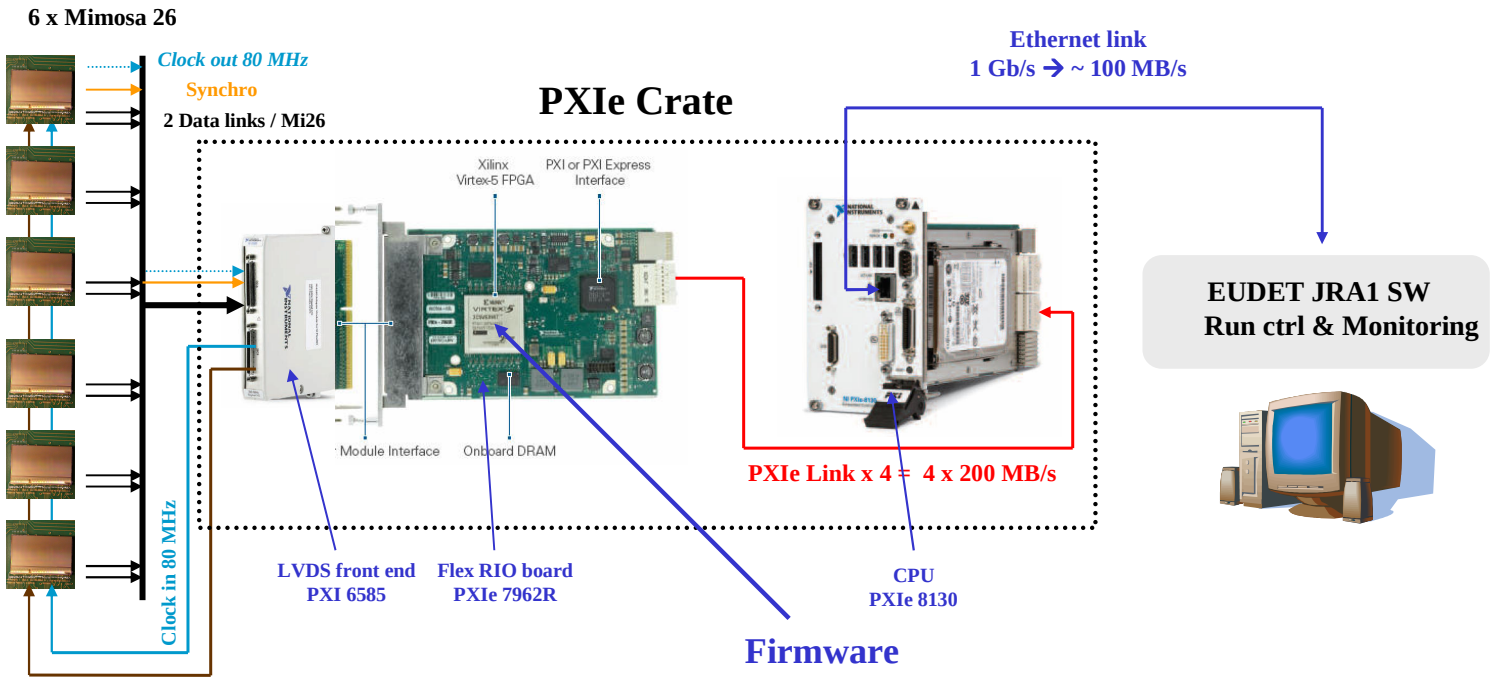
- **Provides the whole states memory size : 1140 W16 ( word of 16 bits ) – 570 W16 / link**




### Summary

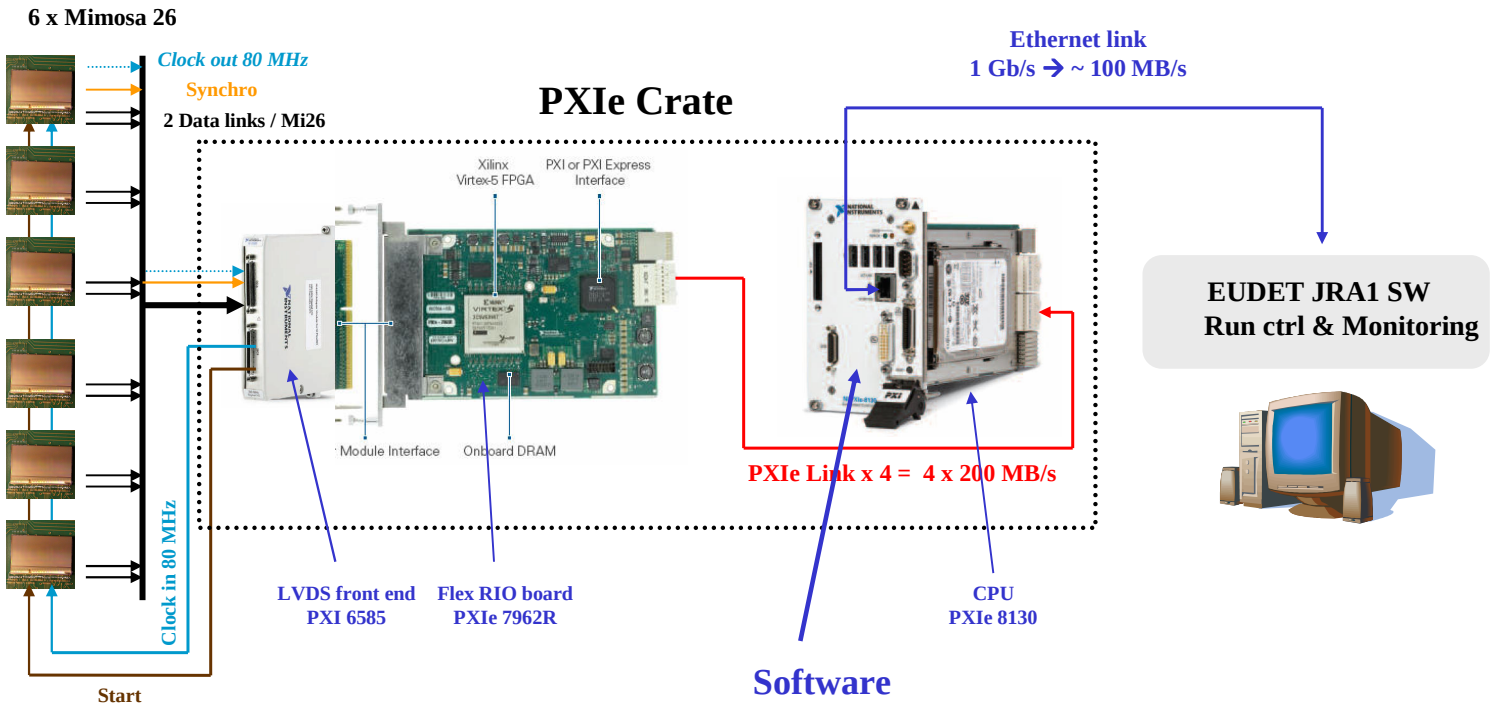
- **Data generated on rising edge of Mimosa 26 clock**
- **Header** → **16 bits / output**
- **Frame counter** → **16 bits / output**
- **Data length ( useful part of data )** → **16 bits / output ( Sum the 2 W16 to get matrix W16 size )**
- **Data** → **Max = 570 x 16 bits / output**
- **Trailer** → **16 bits / output**
- **Padding zero** → **32 bits / output**
- **Maximum stream size per output : 9216 bits = 576 W16 = 1152 W8 … Can be less → Defined by Data length field**

# Flex RIO DAQ proposal : FW tasks




- **Deserialize data ( 12 x W16 deserializer ) → Don't care about header, trailer … → W16 stream**
- **Acquire all frames → Don't care about trigger ( we have enough BW Flex RIO / CPU to do it )**
- **Acquire full frame size → Don't care about real data length ( enough BW to do it )**
- **Store trigger counter from TLU ( up to 288 triggers / frame )**
- **Store Time stamp / trigger info from Flex RIO ( up to 288 triggers / frame )**

# Flex RIO DAQ proposal : SW tasks




- Extract ONLY frames with trigger from data stream provided by Flex RIO
- Cut the frame size to useful part → Defined by the Mimosa 26 of the six having the highest frame size
- Build variable length records ( including trigger info ) → Ready to be sent on Ethernet
- Send data over Ethernet → To be done → SW optimized to let maximum free time for this task

# Why splitting tasks between FW & SW ?

- We had less than one year to
  - Learn how to use this new system → New environment HW, FW, Labview, Labview FPGA
  - Evaluate HW & FW
  - Develop a DAQ demonstrator
- We need to distribute this system ( EUDET, and … ? )
- Therefore we have decided to
  - Minimize FW development time → Keep it as simple as possible
  - Move tasks which may need " adjustment " on SW
    - Because it will be easier for user & also for us to modify SW rather than FW
    - With only 6 Mimosa 26 to read → it seems possible to it by SW

# Flex RIO trigger & TLU trigger …

- Flex RIO handles a trigger input
  - Increments a trigger counter
  - Stores index of Mimosa 26 line read while trigger occurs → Flex RIO trigger / time stamp
    - Can register up to 288 triggers per frame ( 1 x W32 available / trigger )
  - The trigger counter is tested by SW to extract frames with trigger

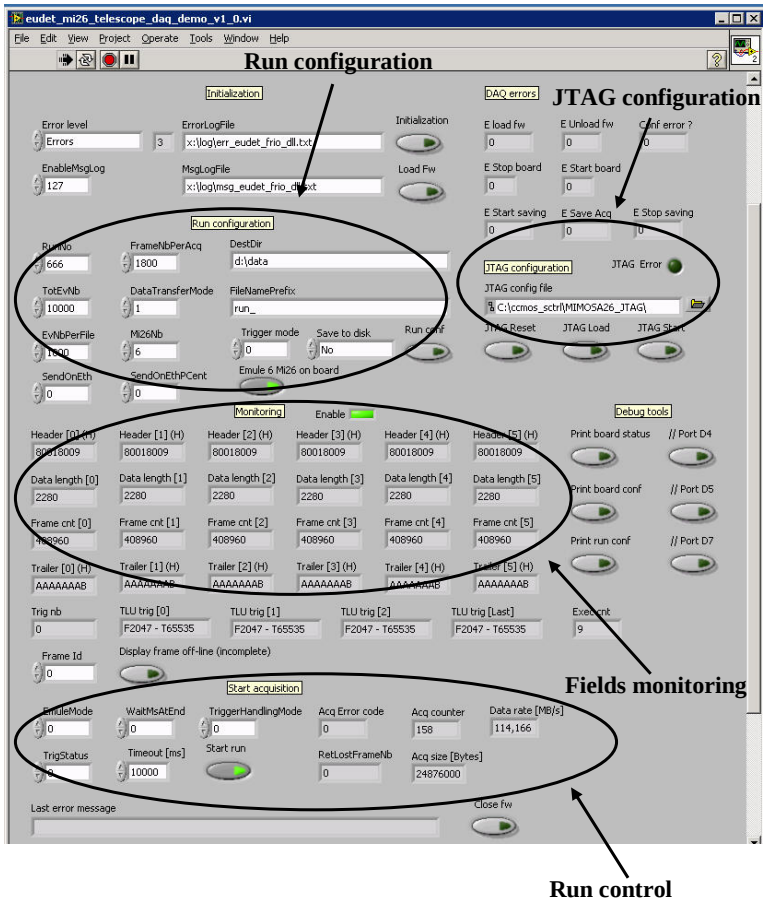
- Flex RIO reads TLU trigger counter
  - TLU trigger data handshake mode → Can store up to 288 triggers / frame ( 1 x W32 / trigger )
  - TLU simple handshake mode → Nothing to store from TLU, but can store a time stamp

- How to handle both Flex RIO trigger input & TLU triggers
  - Provide a " direct " trigger → independent from TLU logic
  - Duplicate the trigger line provided by TLU
    - One to generate Flex RIO trigger
    - One to handle readout of TLU trigger counter by TLU

# How many frames to read after trigger ?

- On Mimosa 26 point of view
  - The matrix of pixels read while trigger occurs will appear one frame later on data stream
    - Latency of one frame → One frame processed while previous one sent
  - There is also a pipeline of 4 lines in Mimosa 26
  - Therefore if a trigger occurs on frame n
    - The pixels associated to it will appears on frame n + 1
    - We also need the following frame ( to complete maxtrix readout ) → n + 2
    - Due to pipeline, if trigger occurs on line >= 572 it requires one more frame → n + 3
  - Conclusion → We need 3 frames to build a physics event
- On Flex RIO point of view
  - We will store trigger on the frame it appears → Don't handle Mimosa 26 latency of 1 frame
- Conclusion
  - We need 4 frames to build a physics event ( first with trigger info, but "empty" + 3 frames )
  - It's defined by a constant in flex RIO library ( can be modified if needed )

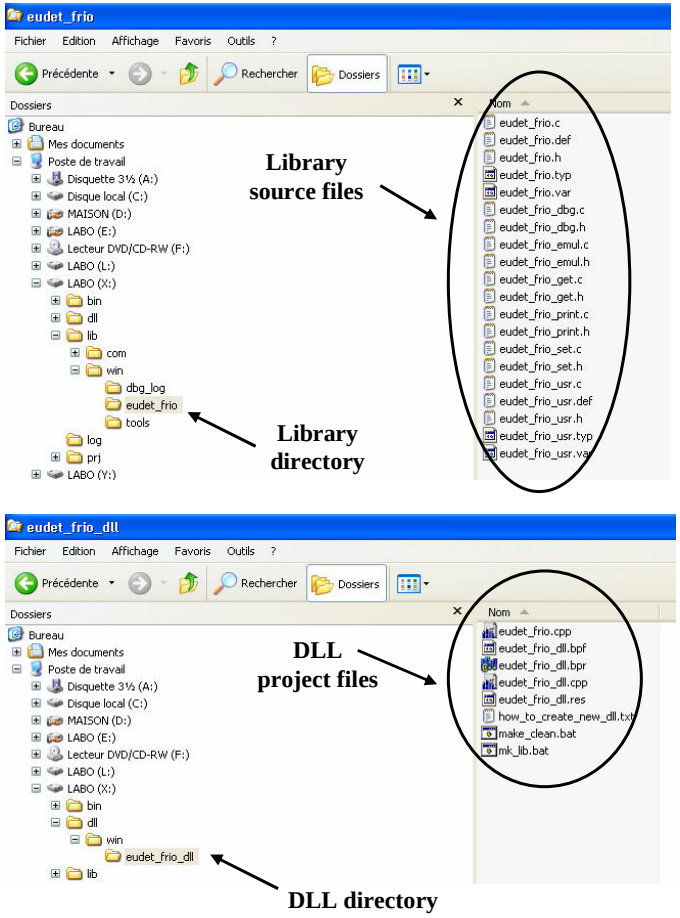
# Flex RIO DAQ proposal : Software




## Labview application

- JTAG configuration
- Run control
- Monitoring of frames fields + triggers
  - Header, frame counter … trailer
- Store one acquisition = 1800 frames
  - Can monitor frame by frame on/off line
- Can store data on disk
- The idea → modify it to
  - Act as a slave
  - Under EUDET SW control
  - Receive run control param via Ethernet
  - Send acquired data over Ethernet

# Flex RIO DAQ proposal : Software







## Eudet lib & DLL → eudet_frio

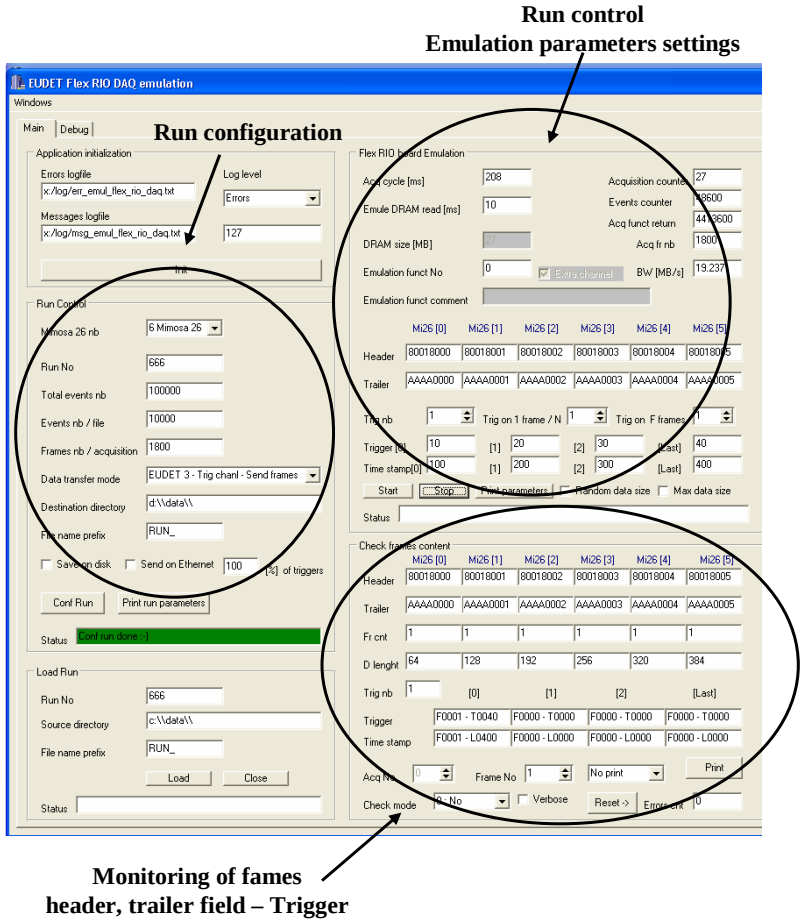
- One library named eudet_frio
- Compiled in a DLL called by Labview
- Standard C & C++ code
- Compiled with C++ Builder
  - Can use another compiler ( VC++ ? )
- Tasks handled by this library
  - Interface to JTAG application (COM)
  - Extract frames with triggers
  - Check data integrity ( header, trailer … )
  - Add trigger information to frame
  - Build variable length records ( 6 x Mi26 )
  - Save data on disk
- Can implement interface to EUDET DAQ
  - Add Ethernet handling
  - Empty src files eudet_frio_usr.* foreseen for this

# DAQ emulator : Why ?

## And now an emulator … why ?

- **Because HW, FW & SW are not ready ?**
  - **No ! The system is ready ☺**
- **Because … it can be hard to develop interface to EUDET DAQ with the whole system …**
  **you need to understand the following topics & "play" with all of them together …**
  - **HW → Mimosa 26, JTAG configuration, Flex RIO board**
  - **SW → The DAQ Labview SW**
  - **SW → The eudet_frio compiled in a DLL ( C code )**
  - **SW → The eudet DAQ software**
  - **It's a long test cycle → Need to run the system (HW) to test …**
- **Testing & debugging Ethernet code in a DLL may not be easy**
  - **Need to compile DLL, then execute the application to test**
  - **As it's a Labview application which call the DLL → No integrated debugging tools ( may exists ? )**
- **The idea**
  - **DAQ emulation application, GUI compiled with C++ Builder**
  - **See eudet_frio lib as a a part of it's source files ( not as a DLL ) → Can use Borland's debugger if needed**
  - **Don't need any HW to develop & test Ethernet interface / EUDET DAQ**

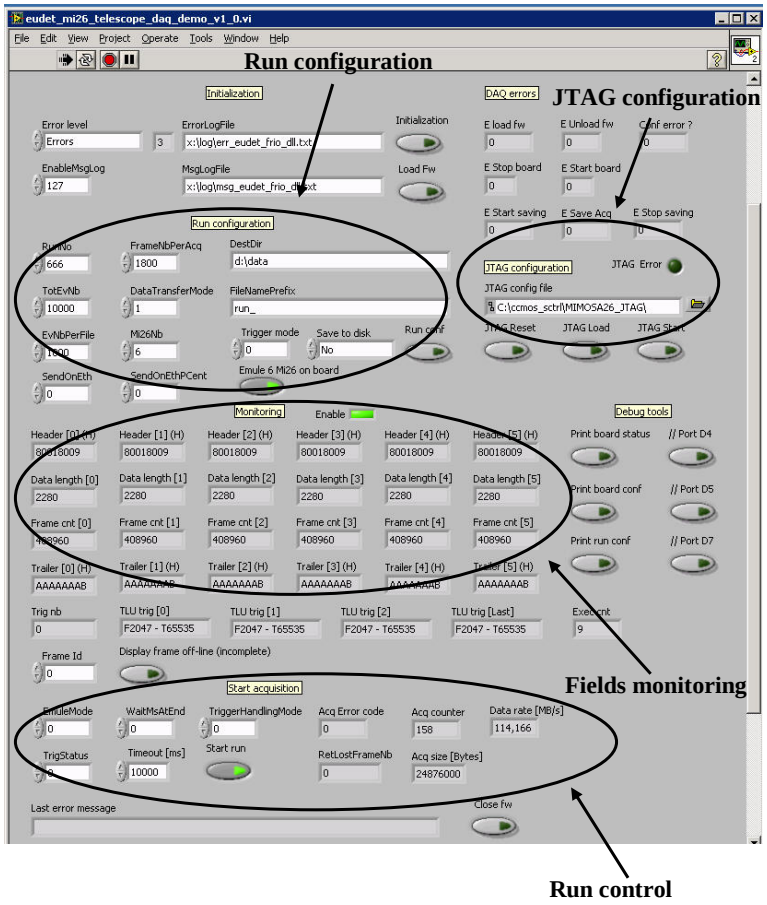
# DAQ emulator : What can we do with it ?




## DAQ Emulator

- Emulates 1 or 6 Mimosa 26
  - Header, frame counter, trailer
  - Trigger nb / frame & trigger values
  - Fixed or random data part size
  - Data part filled with 0 ( but can be modified )
- Emulation parameters
  - controlled from GUI
  - or coded in emulation functions ( eudet_frio )
- Few code in the application itself
  - Uses functions written in eudet_frio
  - No extra effort to move back to Labview
- Can print frames fields content
  - GUI or in text mode
- Can store data on disk in a run file
- Can load a run file from disk and scan it

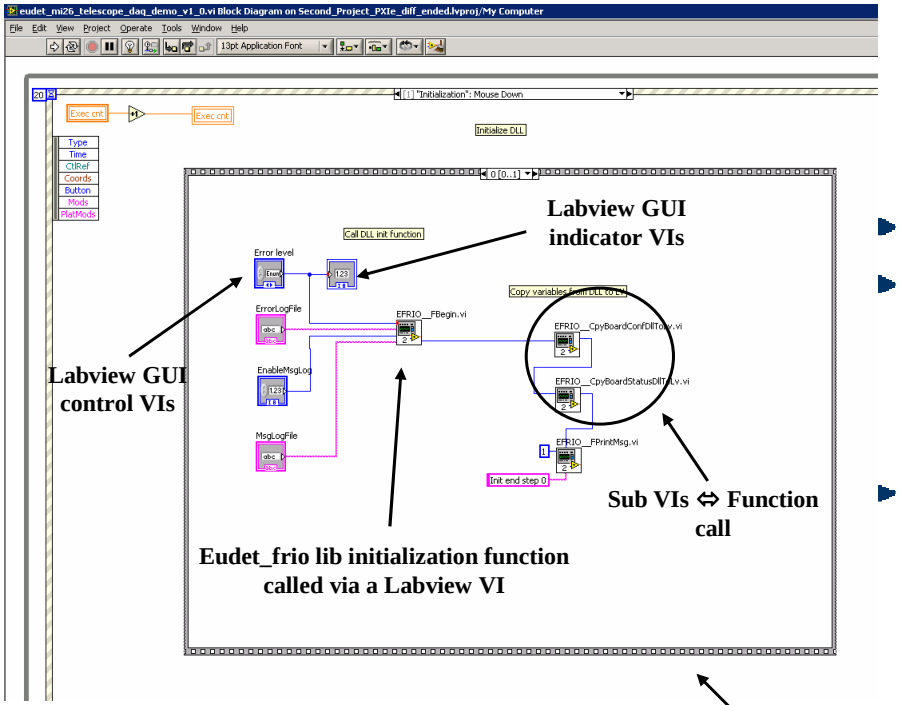
# DAQ Labview application : The GUI




## Labview application

- JTAG configuration
- Run control
- Monitoring of frames fields + triggers
  - Header, frame counter … trailer
- Store one acquisition = 1800 frames
  - Can monitor frame by frame on/off line
- Can store data on disk
- The idea → modify it to
  - Acts as a slave
  - Under EUDET SW control
  - Receive run control param via Ethernet
  - Send acquired data over Ethernet

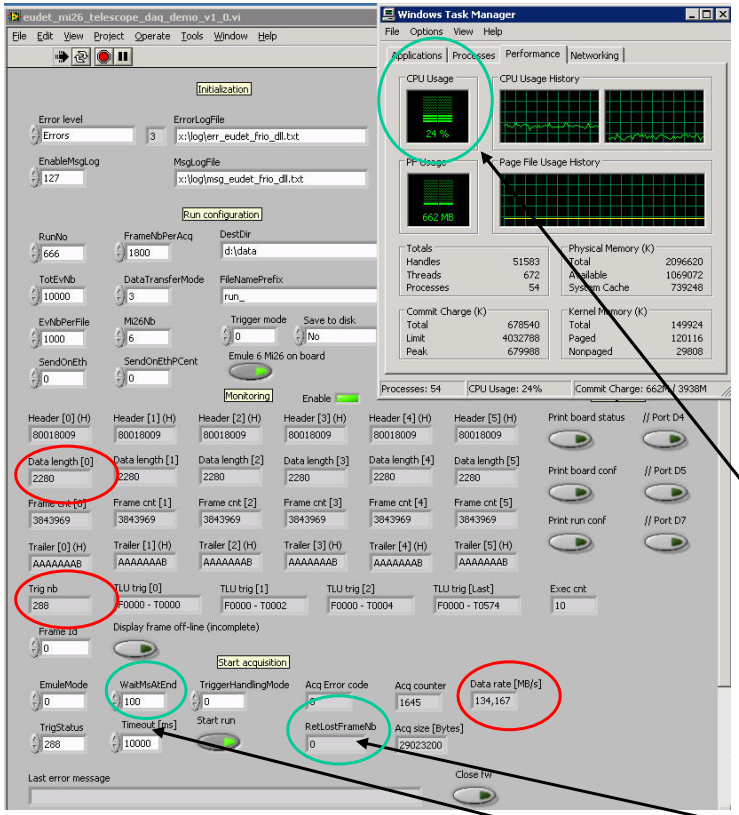
# DAQ Labview application : The source code




## The source code

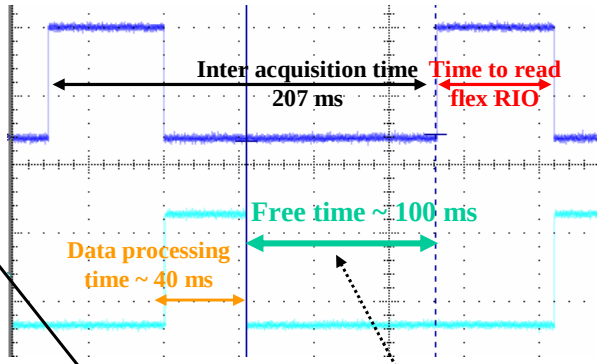
- It's Labview graphical programming
- Labview is used for
  - GUI
  - To high level control → Sequence operations
  - Board driver
- Labview call eudet_frio function
  - Via eudet_frio.dll
  - Functions encapsulated in Vi

# DAQ Labview application : Performances




## Test conditions

- Readout of 6 Mimosa 26
- 1800 frames / Acq → 1800 x 115,2 µs = 207 ms
- Maximum frame size
- 288 emulated triggers / frame




## Results

- Only 25 % CPU used ! – Data rate 134 MB/s
- No frame lost
- 100 ms free time between acquisitions

# System Status : What is done / still to be done ?

## Status → HW, FW, SW

- **Hardware**
  - **How to setup the system**
  - **IPHC boards documentations**
    - **→ Will be provided in January 2011**
- **Firmware**
  - **Done – Upgrade may be needed depending on trigger handling decisions**
  - **Doc → Not needed ( We will upgrade & maintain FW )**
- **Software**
  - **Done**
  - **Optimized**
    - **25 % CPU usage – 100 ms free to send data over Ethernet**
  - **DAQ demonstration SW provided**
  - **Documentation → OK**
  - **Light DAQ SW to show interface / EUDET → End December 2010**
  - **Data stream errors handling → Will be done in January 2011**

# Planning of the week : From SW to HW …

- Tuesday morning
  - Introduction / Overview of Flex RIO DAQ system
  - DAQ system demonstration
- Tuesday afternoon
  - C source files installation & DAQ emulator compilation
  - Play with DAQ emulator & start to look at source files ( Application & eudet_frio library )
- Wednesday morning
  - Deeper look into eudet_frio source files
- Wednesday afternoon
  - Labview DAQ source files installation – Small tutorial on Labview programming
  - Flex RIO firmware
- Thursday morning
  - Labview DAQ application & Source code
- Thursday afternoon
  - Setup the system → Mimosa 26, TLU, Flex RIO and SW
- Friday morning
  - Discussions – Next steps