The Calicoes project

Calice OnlinE System

High Performance Middleware For the Calice DAQ System

Frédéric Magniette, LLR

ANR-2010-BLANC-0429-01

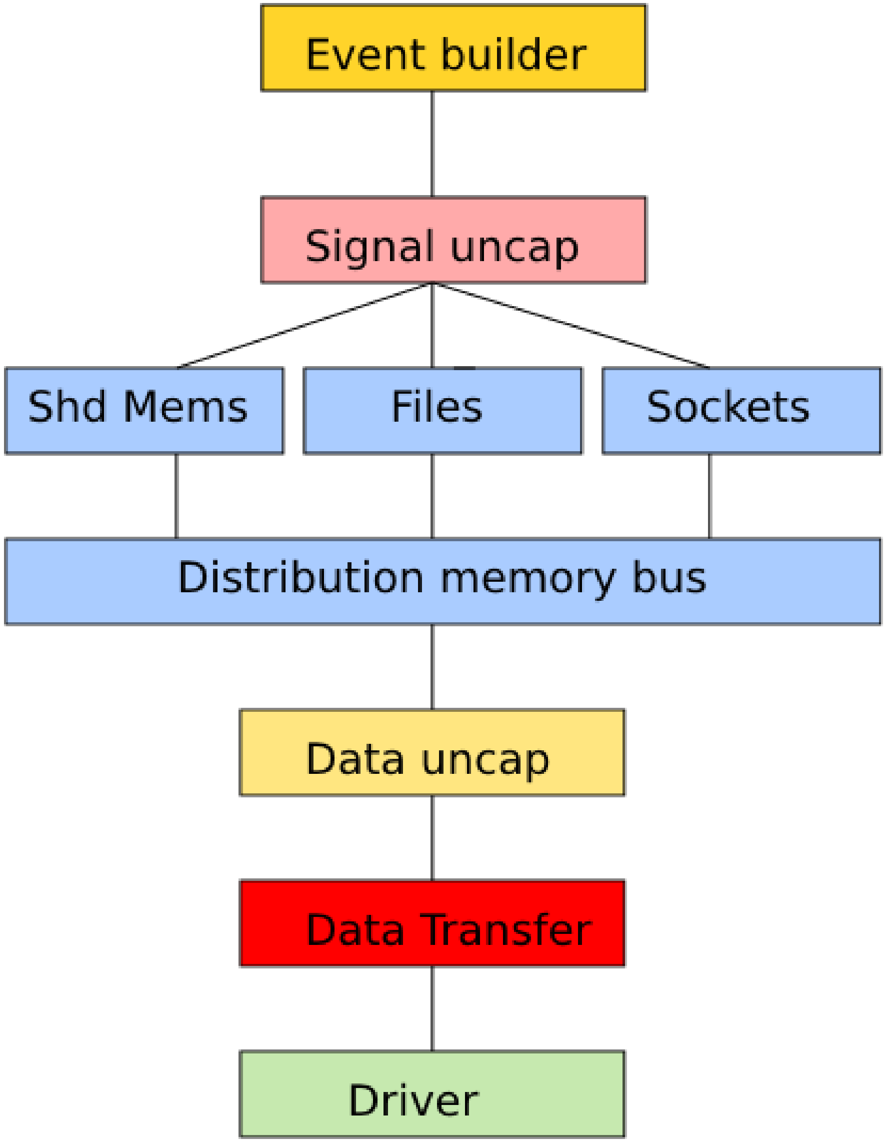
The Needs

- Acquisition Chain : dev from scratch
- Control-Command Chain (Liblda)
- Human-Machine interfaces
  - For DAQ debugging
  - For data acquisition
- Properties
  - Calice compatible (DIF-LDA-ethernet-Multiroc)
  - Highly reliable
  - Strong evolutivity (developpement phase)

Design Concepts

- Internal Modularity
  - Functional blocks with easy communication channels (on the Unix model)
  - Every block is making few but making it fast and good
  - Easy evolutivity
- Parallelism
  - Massively multi-threads approach
  - Pipelined treatments
  - Low-latency code (no barrier)
- External Genericity
  - Connected to the outside with open formats (xml, tcp sockets)

Acquisition Chain



Global
BD

Skiroc

Generic

LDA/
GCC/
DIF

Raw
socket

Ethernet
Usb
PCI(e)

building
events

(de)coding
ASIC signal
and control

Offering
various
distribution
mode

Checking
Uncapping
Reordering
Tagging

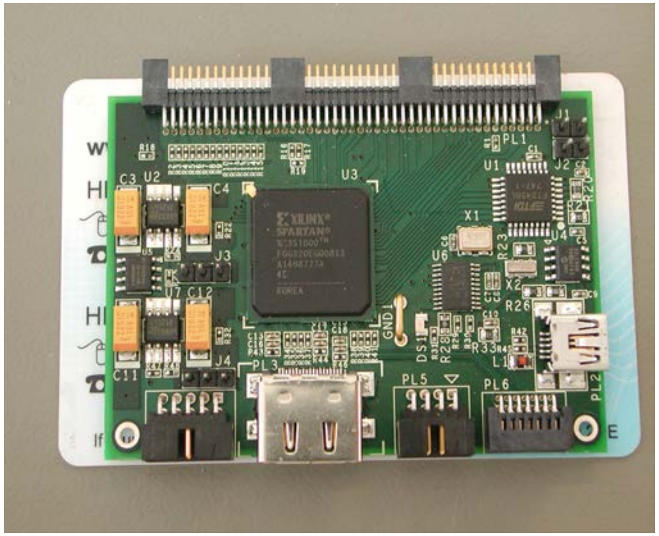
Transfering
from/to media

Interfacing
kernel and
hardware

Low level Transfer

- Standard Ethernet driver (e1000)
- Low-level socket (SOCK_RAW)
- Limited in rate by the SOCK_RAW implementation
- Projects to speed up: 3 alternatives
  - Migrating DAQ to kernel compatible code (UDP)
  - Specific kernel driver for DMA transfer of actual ethernet format
  - Specific ethernet adapter (ODR?)

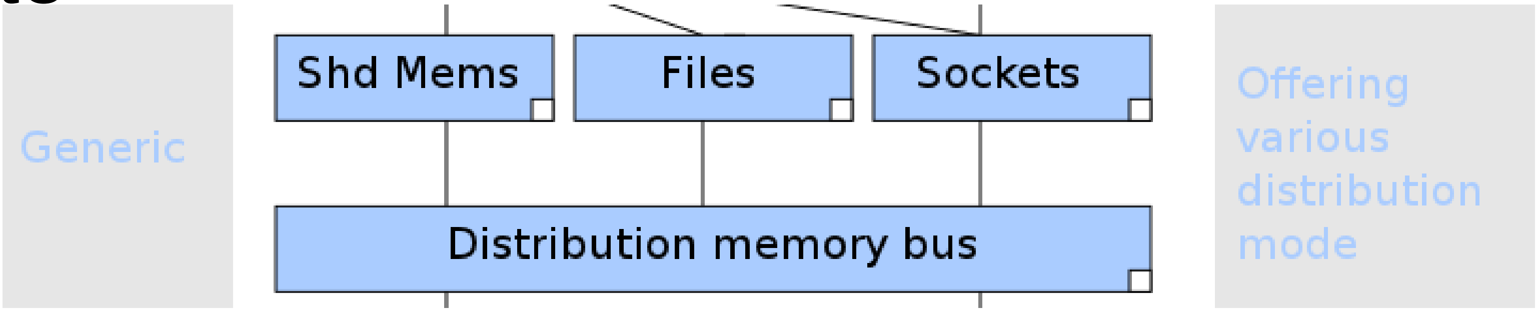
LDA/GCC/DIF Uncap



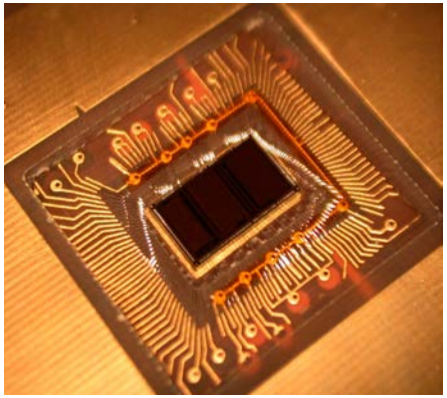
- One of the two specific parts of the system (dependant of the DIF Firmware version)
- Very basic : uncap, DIF tagging
- Projects
  - Control improvement : integrity control (CRC), packet loss control and measurement, reordering
  - Implementation of local-id for multi-LDA support

The distribution bus

- Split data into beams (by dif)
- Offer the maximum flexibility in data treatment
  - Offline through files
  - Remote online through sockets
  - High performance online through Shared Memory
- Pipelined treatments for maximizing the flow rate



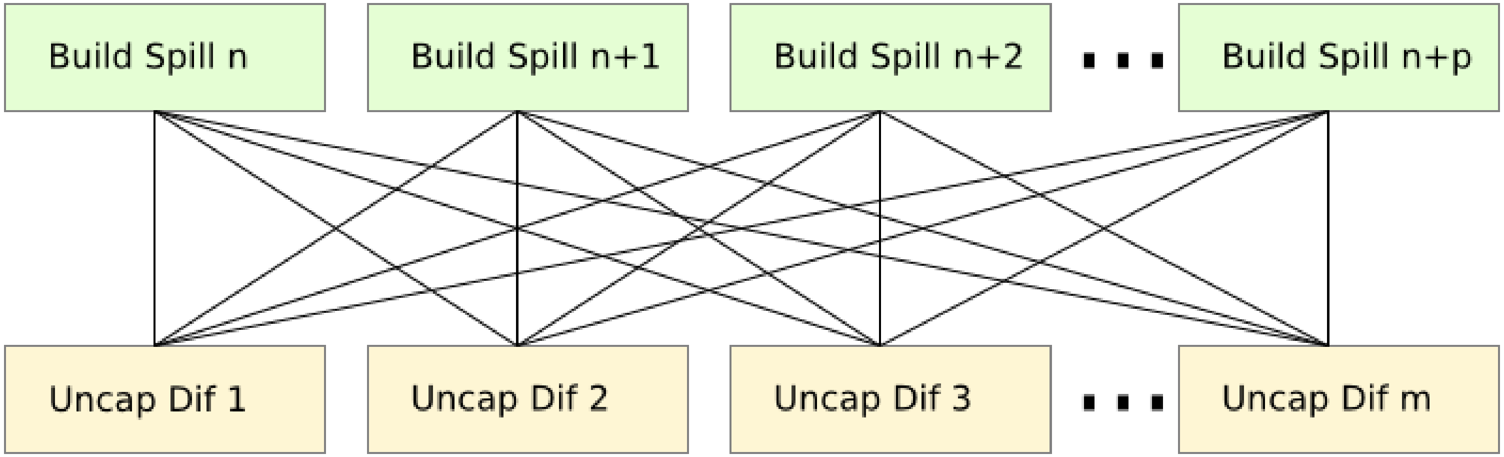
Signal Uncap



- The second specific block of the system (dependant on the roc chip version)
- Actually adapted for skiroc, can be ported easily to spiroc
- Uncaps the skiroc data to produce isolated physical events
- Computes real-time statistics

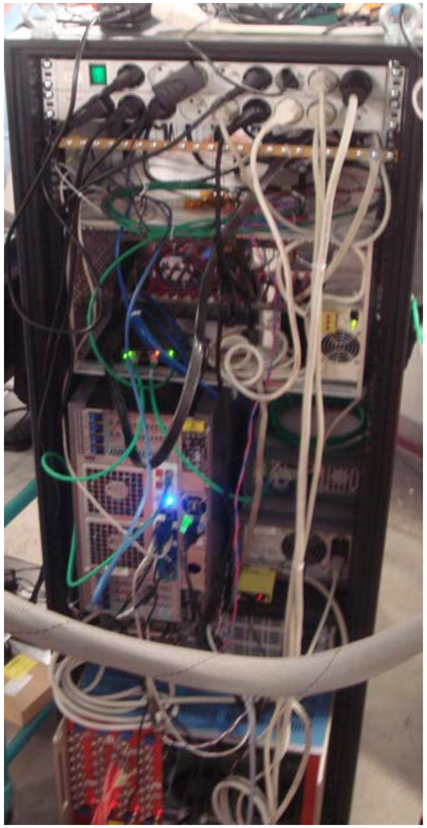
Online Event building

- Not implemented now
- Basic idea : change the multiplexing from location to time
- Massively multi-thread for handling the data rate
- Project
  - defining a good pivot format (as it exists in DH-CAL)
  - Interfacing Marlin processor with uncap threads through TCP sockets



The Control-Command Chain

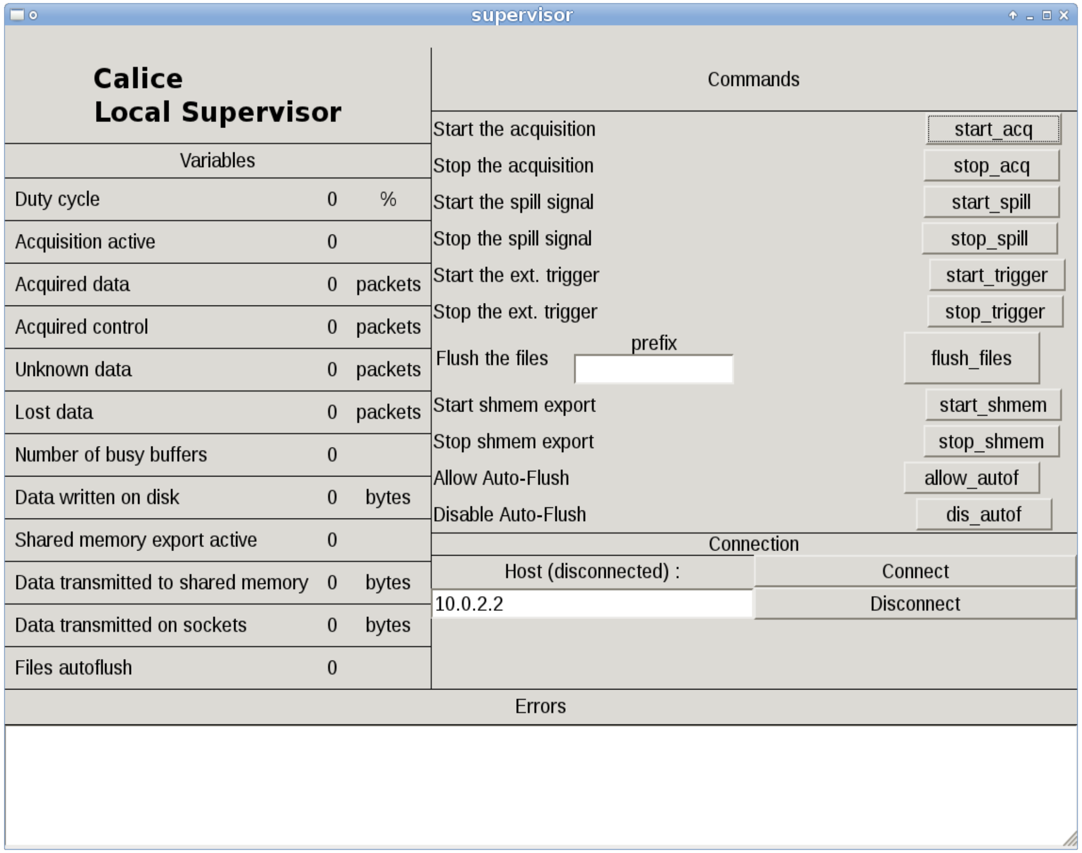
- SI-W ECAL actual hardware



- All this stuff is controled by specific little drivers (python and C), shell scripts and the libLDA
- All accessible through a command module via a TCP socket with xml commands

Local Supervisor

- Local supervisor / monitor
- Allows users to send commands and read variables through the variable and command modules
- Generic design based on Gtk3/Xml description files (for high flexibility)



**Calice
Local Supervisor**

Variables		
Duty cycle	0	%
Acquisition active	0	
Acquired data	0	packets
Acquired control	0	packets
Unknown data	0	packets
Lost data	0	packets
Number of busy buffers	0	
Data written on disk	0	bytes
Shared memory export active	0	
Data transmitted to shared memory	0	bytes
Data transmitted on sockets	0	bytes
Files autoflush	0	

Commands	
Start the acquisition	start_acq
Stop the acquisition	stop_acq
Start the spill signal	start_spill
Stop the spill signal	stop_spill
Start the ext. trigger	start_trigger
Stop the ext. trigger	stop_trigger
Flush the files prefix	flush_files
Start shmem export	start_shmem
Stop shmem export	stop_shmem
Allow Auto-Flush	allow_autof
Disable Auto-Flush	dis_autof

Connection

Host (disconnected) :	Connect
10.0.2.2	Disconnect

Errors

Pycaldaq

- Embedded Python scripts to pilot the whole detector
- Program or interractive sessions

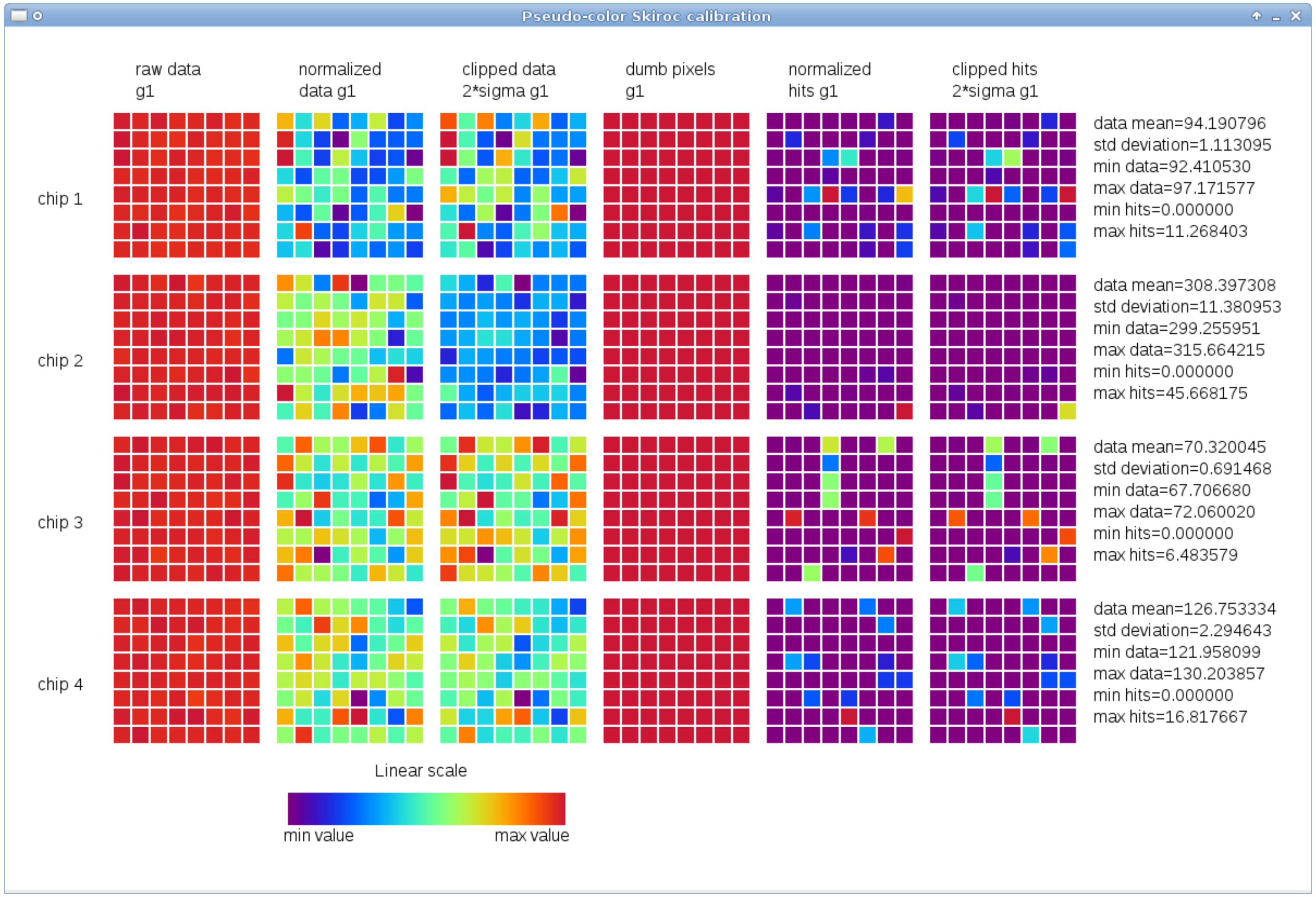
```
import caldaq

def acq_run():
    caldaq.start_acq()
    caldaq.start_spill()
    caldaq.start_trigger()
    time.sleep(15)
    caldaq.stop_trigger()
    caldaq.stop_spill()
    caldaq.stop_acq()

caldaq.stop_acq()
caldaq.stop_spill()
caldaq.stop_acq()
caldaq.flush_files("trash")
for chan in range(0,64):
    caldaq.system("hack_config/set_trigchan config/calib_base.txt 4 \
                                              %d config/calib/calib_chan%d.txt" % (chan,chan))
    for trig in range(150,401):
         caldaq.system("hack_config/set_gtrigger config/calib/calib_chan%d.txt  4 \
                                              %d config/calib/calib_chan%d_trig%d.txt" % (chan,tri
        caldaq.send_config("config/calib/calib_chan%d_trig%d.txt" % (chan,trig))
        acq_run()
        caldaq.flush_files("calib_chan%d_trig%d" % (chan,trig))
```

The Framework connection

- Connection to XDAQ (or any other framework Doocs, Tango) through the TCP/XML command module
- The framework can be interfaced at any level through a generic device server configured by an xml command file
- Symmetrically, the framework can also provides devices to integrate in the control-command scheme via a specific command module

Statistics GUI

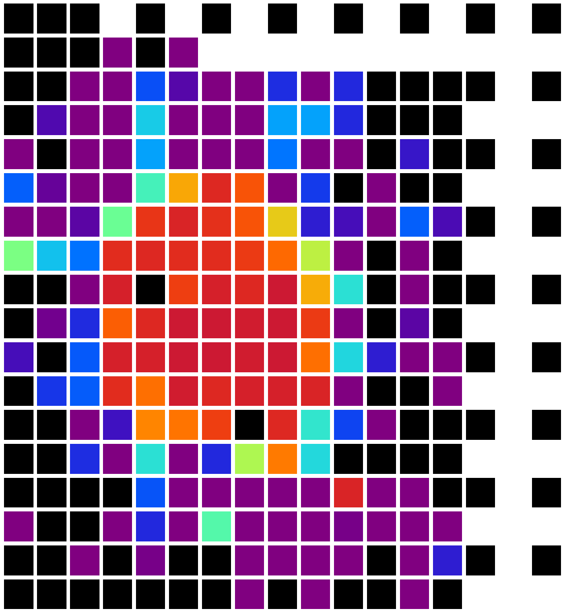
- Display_stats : a tool for displaying the calibration statistics, online or offline



```
data mean=94.190796
std deviation=1.113095
min data=92.410530
max data=97.171577
min hits=0.000000
max hits=11.268403

data mean=308.397308
std deviation=11.380953
min data=299.255951
max data=315.664215
min hits=0.000000
max hits=45.668175

data mean=70.320045
std deviation=0.691468
min data=67.706680
max data=72.060020
min hits=0.000000
max hits=6.483579

data mean=126.753334
std deviation=2.294643
min data=121.958099
max data=130.203857
min hits=0.000000
max hits=16.817667
```

Hitcam

- Online and offline visualisation of hits (internal trigge
- Plugged via tcp socket to the a
- Adjustable gain
- Video generation
- Allows to proceed to :
- beam spotting
- online visual data control
- beam supervision



Results

- Good data rate : perfect for 10 Hz Spill
- Very good reliability
  - 15 days of beamtest at DESY without bugs
  - 3 weeks of calibration without a problem (20480 configurations loaded, 122880 data files)
- Easy to use
  - Multiples graphical interfaces
  - Programming interface
- Good evolutivity
  - Any block can be improved and modified easily by anybody (especially python command modules)
  - Easy to adapt to another setup : AH-Cal collaboration

Perspectives

- Increasing the flow rate by specific soft or hard modification
- Migrating the control-command to unified Tcp/Xml system
- Additionnal online monitoring (data quality, calibration...)
- Aggregating the signal events in an event-builder
- Integrating the system in the XDAQ framework
- Handling global configuration files through the unified control-command system

Thank you for your attention, any questions ?