

MBS (Multi Branch System)



**The General Purpose Data Acquisition System MBS
at GSI (and elsewhere)**

**A short Introduction into
VME, PCI Express and combined Systems**

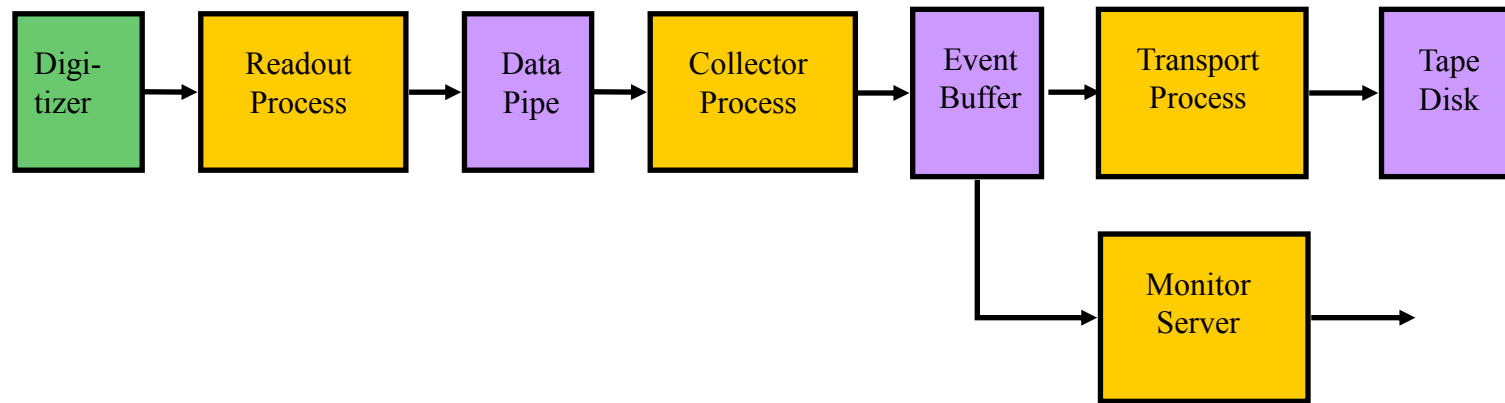
**(White Rabbit Time Stamping DAQ Synchronization)
and
(High Resolution TDCs ($\ll 7$ ps RMS))**

User Comment

The Multi Branch System (MBS) [10] is a DAQ framework developed and maintained at GSI. It is used at over 100 installations world-wide. The job of the MBS is to set up memory mappings to access the various data acquisition modules (DAMs) and to handle trigger synchronisation between multiple sub-systems. Furthermore, it takes care of transport and combination of the sub-system data into complete events. At a first glance, the most prominent weakness seems to be that the MBS does not handle the actual DAM setup or read-out. This however turns out to be its greatest strength! Not limiting the user to a predefined set of modules and read-out modes, but instead leveraging the full power of a user-written read-out function per subsystem in the C programming language makes it extensible and flexible. This is especially important when combining a multitude of separate detectors — developed more or less independently — into larger setups.

Håkan T. Johansson
PHD 2010

MBS Single Processor System (Process View)



Readout Process: Readout digitizers (ADCs, TDCs, etc) and write data in sub-event pipe

Collector Process: Collect sub-event from sub-event pipe and build formatted events
This process can read other sub-event pipes if accessible on memory mapped device written by slave processors (Single Branch System)

Transport Process: Provide data logging with various methods (see previous slide)

Monitor Server: Provide data samples on request to online monitoring analysis systems via TCP sockets

MBS User Readout Interface: f_user.c

```
int f_user_get_virt_ptr (long *pl_loc_hwacc, long pl_rem_cam[])
{
    // user code: create virtual pointer to be used in f_user_init and f_user_readout
    // f_user_get_virtual_ptr called once at every start acquisition
}

int f_user_init (        unsigned char        bh_crate_nr,
                        long                 *pl_loc_hwacc,
                        long                 *pl_rem_cam,
                        long                 *pl_stat)
{
    // user code: initialize/setup all hardware controlled by processor
    // f_user_init called for each crate specified at start acquisition
}

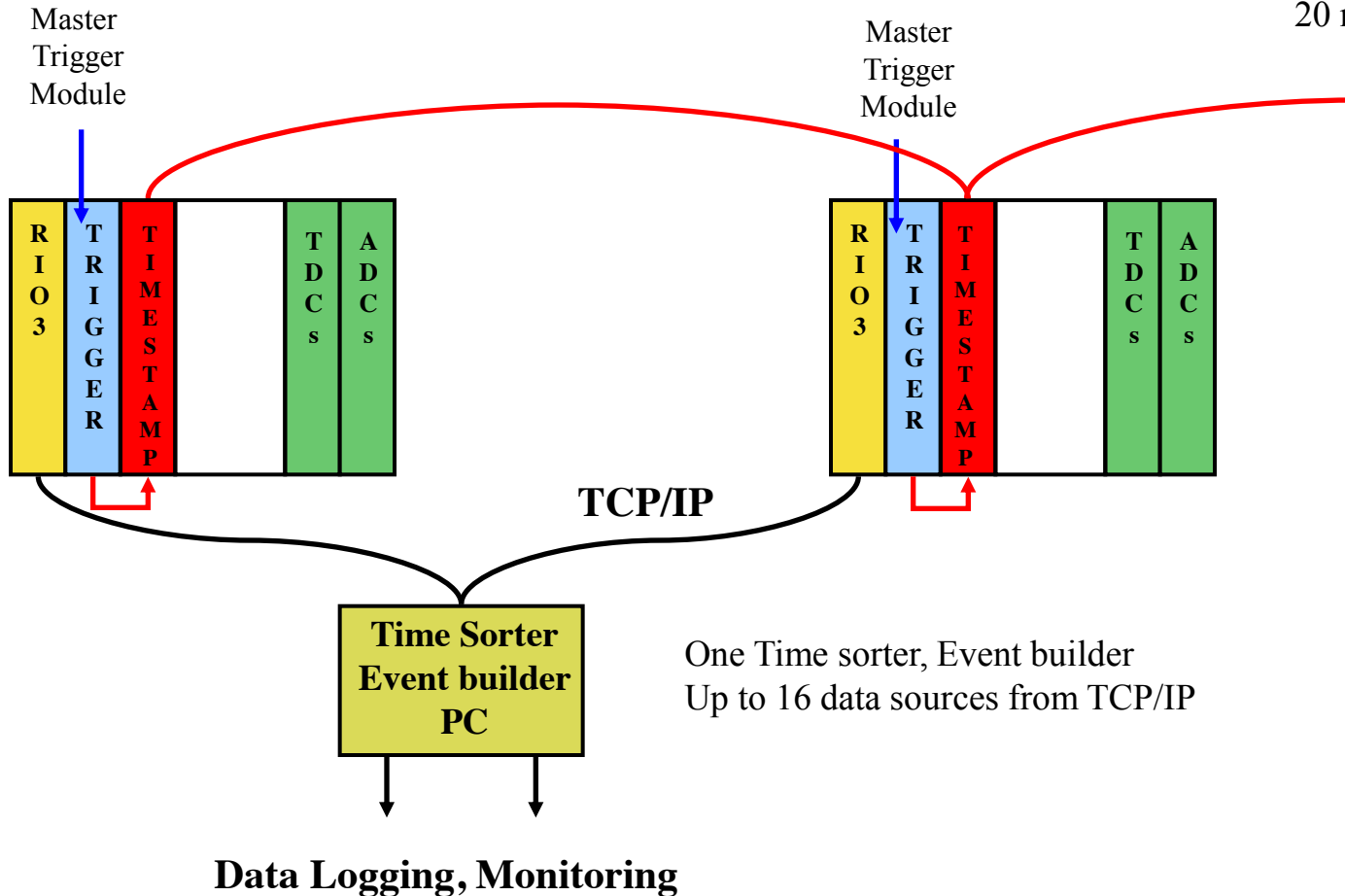
int f_user_readout (    unsigned char        bh_trig_typ,
                        unsigned char        bh_crate_nr,
                        register long        *pl_loc_hwacc,
                        register long        *pl_rem_cam,
                        long                 *pl_dat,
                        s_veshe             *ps_veshe,
                        long                 *l_se_read_len,
                        long                 *l_read_stat)
{
    // user code: readout of all modules controlled by processor
    // called for each crate specified once per accepted trigger
}
```

setup.usf (only part)

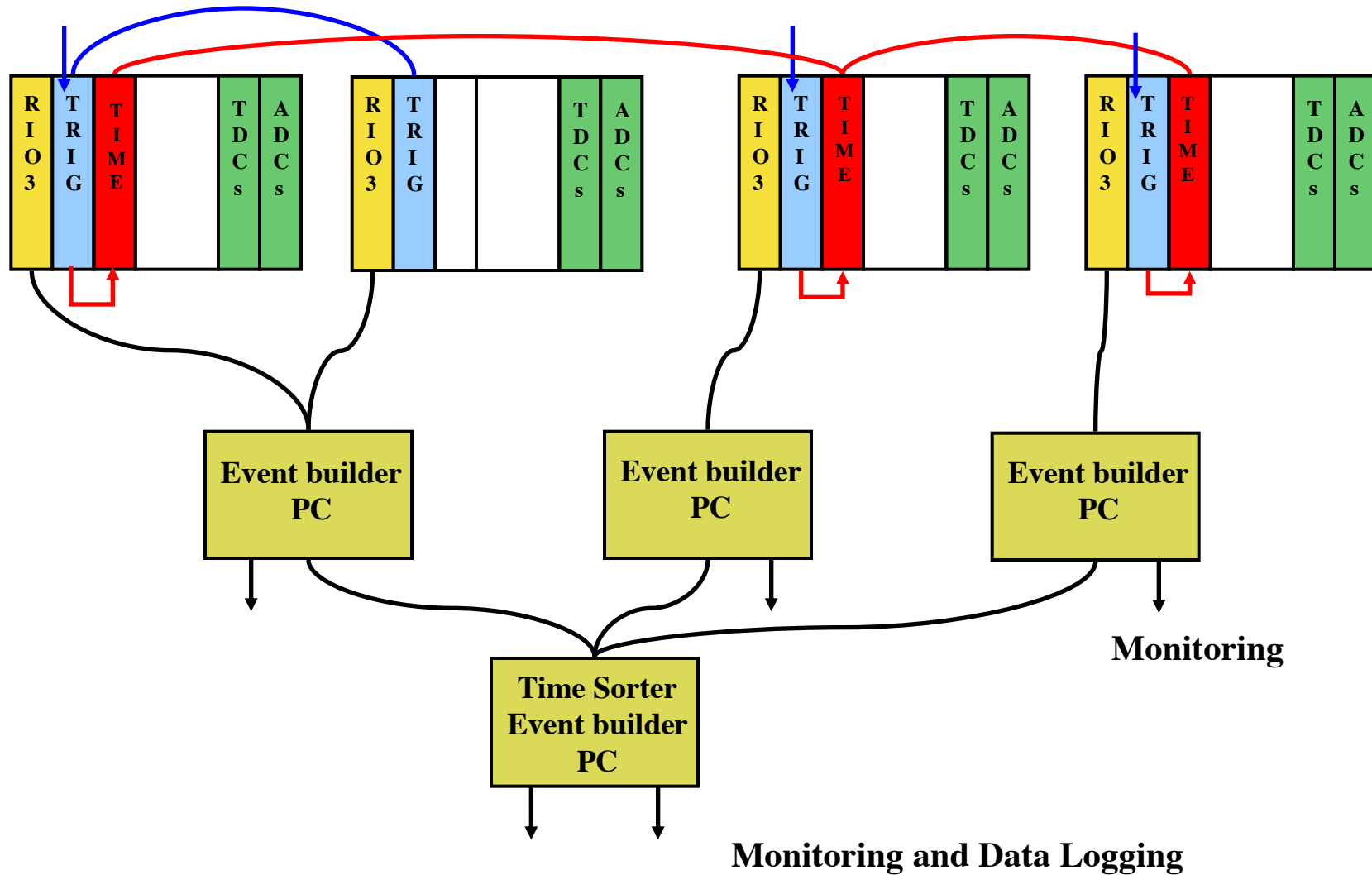
```
SBS_SETUP_TABLE: -
#*****
MASTER          = 12, -                               # master is RIO4
#*****
LOC_MEM_BASE     = (0x5300000, 0x0, 0x0, 0x0, -       # VME address window base at 0x03000000 ..
                  0x0, 0x0, 0x0, 0x0, -
                  0x0, 0x0, 0x0, 0x0, -
                  0x0, 0x0, 0x0, 0x0), -
#-----
LOC_MEM_LEN      = (0x500000, 0x0, 0x0, 0x0, -       # .. with 0x500000 byte window size
                  0x0, 0x0, 0x0, 0x0, -
                  0x0, 0x0, 0x0, 0x0, -
                  0x0, 0x0, 0x0, 0x0), -
#*****
LOC_PIPE_TYPE    = (1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0), -
#*****
LOC_PIPE_BASE    = (0xe800000, 0x0, 0x0, 0x0, -       # sub-event data pipe base address
                  0x0, 0x0, 0x0, 0x0, -           # at 0x8000000 in processor DRAM ..
                  0x0, 0x0, 0x0, 0x0, -
                  0x0, 0x0, 0x0, 0x0), -
#-----
PIPE_SEG_LEN     = (0x800000, 0x0, 0x0, 0x0, -       # .. with total size of 0x800000 bytes
                  0x0, 0x0, 0x0, 0x0, -
                  0x0, 0x0, 0x0, 0x0, -
                  0x0, 0x0, 0x0, 0x0), -
#-----
#
#   crate. nr: 0    1    2    3    4    5    6    7
PIPE_LEN         = (100000, 0, 0, 0, 0, 0, 0, 0, -   # at maximum 100000 sub-events can
#   crate. nr: 8    9   10   11   12   13   14   15   # be stored in sub-event pipe
#               0, 0, 0, 0, 0, 0, 0, 0), -
#*****
#   crate. nr: 0    1    2    3    4    5    6    7
RD_FLG          = (1, 0, 0, 0, 0, 0, 0, 0, -       # 1: switch readout specified in f_user.c on
#   crate. nr: 8    9   10   11   12   13   14   15   # 0: switch it off
#               0, 0, 0, 0, 0, 0, 0, 0), -
#*****
COL_MODE        = 0, -                               # 0: local event-building
#*****
#   crate. nr: 0    1    2    3    4    5    6    7
TRIG_CVT        = (30, 0, 0, 0, 0, 0, 0, 0, -       # trigger conversion time: 30 := 3 us
#   crate. nr: 8    9   10   11   12   13   14   15
#               0, 0, 0, 0, 0, 0, 0, 0), -
#*****
```


MBS Multi Processor System (Time Stamp Synchronization(TITRIS))

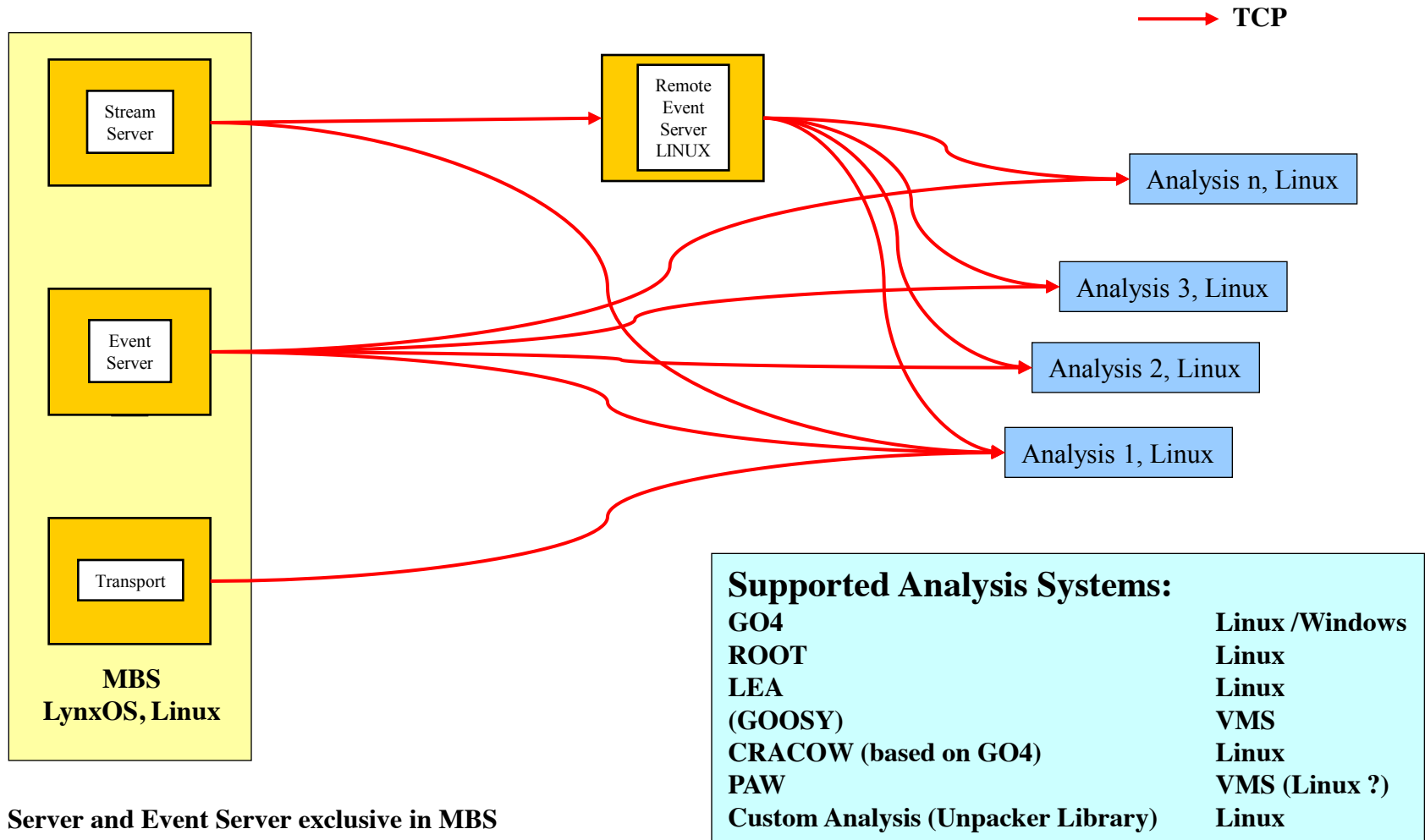
Time Synchronization: Up to 31 Modules,
> 250 m,
20 ns resolution



Combined Trigger – and Time Stamp Synchronization



MBS Online Data Analysis (Online Monitoring)



Stream Server and Event Server exclusive in MBS
Event selection filter in Event Server and Remote Event Server



PCI Express Systems

- Identical trigger handling as described for VME
- Readout functions and setup files have identical structures
- Identical possible DAQ/MBS topologies
- Identical data logging and monitoring
- Per design integrated into any MBS system

What is the different?



Two mandatory Modules for PCI EXPRESS MBS Systems:

- PEXOR3 or KINPEX1
- TRI XOR

Data Collector PEXOR3 and KINPEX1 (Identical functionality, KINPEX1 new)



PEXOR3, Lattice FPGA

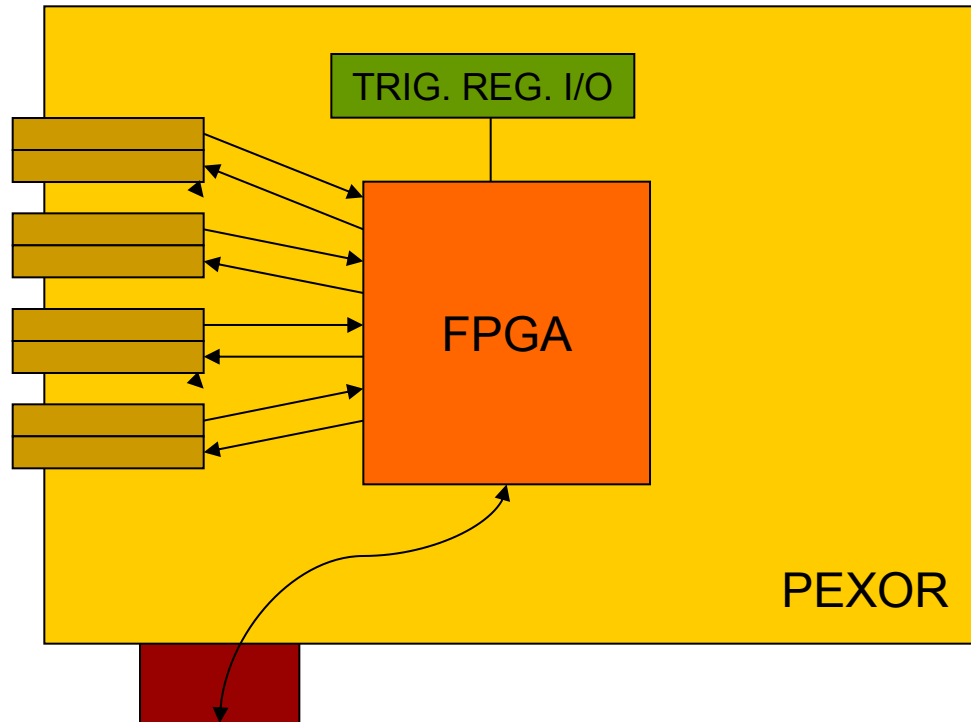
KINPEX1, Xilinx Kinpex FPGA



PEXOR3 / KINPEX1 Front End Data Collector

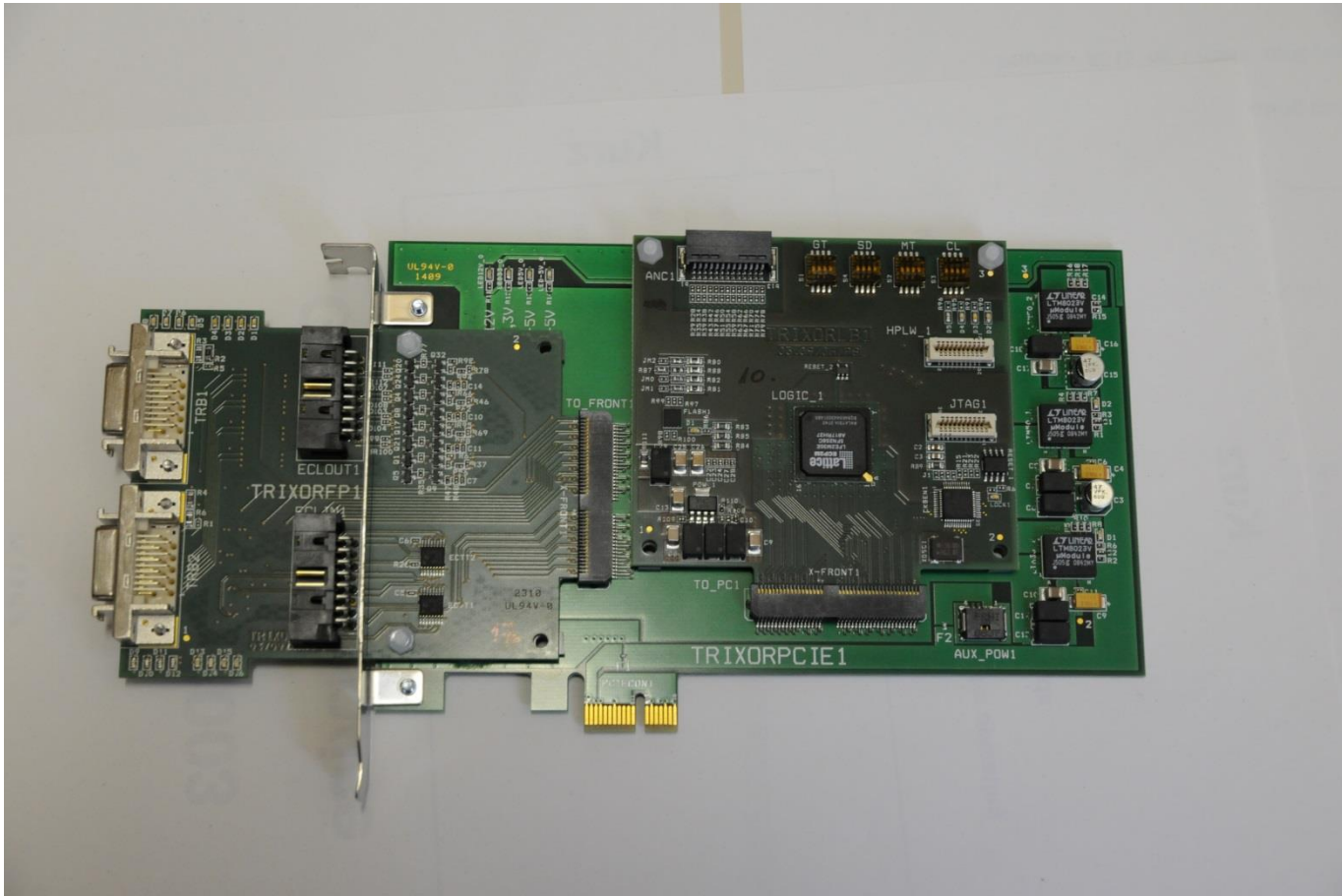
4 SFP fiber pairs
each 2 Gbit/s to
connect 4 x 256
front end boards
at maximum

after 10/8 coding:
800 (4 x 200) MB/s
payload speed per SFP



4 Lane PCI Express
> 600 MB/s FPGA -> PC DRAM payload speed

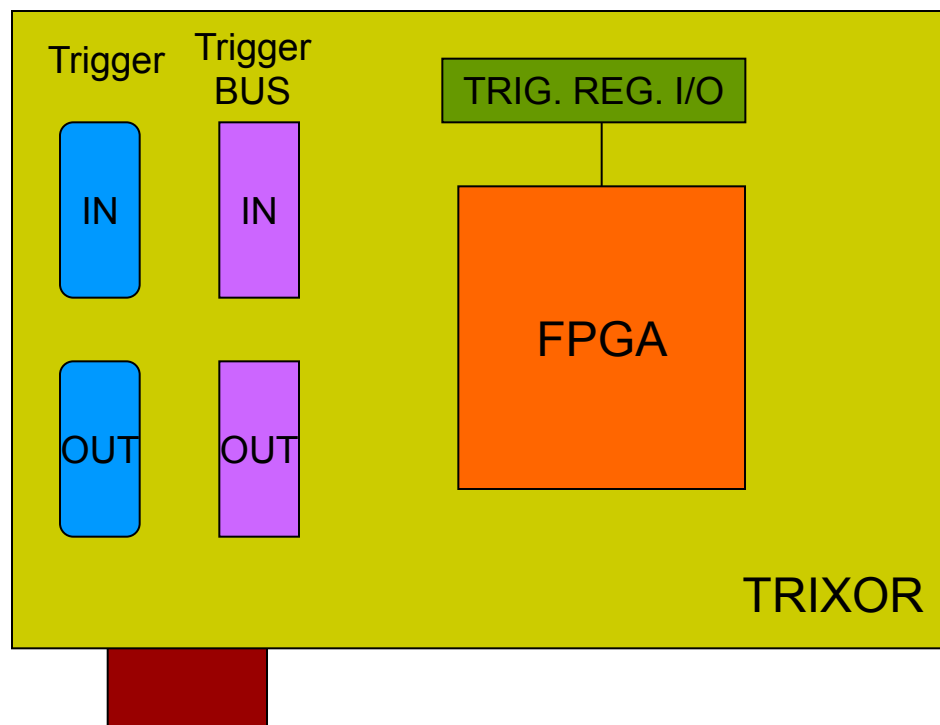
TRIXOR (Identical Functionality as TRIVA3,-5,-7)



TRIXOR

TRIXOR: - Identical functionality as TRIVA3/5/7

- Can be plugged in PCI Express or PCI slots. Takes only power.
- Works as master (trigger from input) and slave (trigger from trigger bus).
- Several TRIXOR and TRIVA can be interconnected via the trigger bus to compose synchronous bigger MBS systems.
- Communication with the TRIXOR via PEXOR and the Trigger Register I/O connector.
- Accepted trigger send a signal via the Trigger I/O connector to the PEXOR, which is then transformed into a PCI Express interrupt to notify the PC readout processor.



PCI Express, PCI (only power)

PEXOR3-TRIXOR at Work



Available Frontend Boards

Existing and running

- FEBEX3: Pipelining ADC, 50 MHz, 14 bit, Input -1V to +1V
- FEBEX4: Pipelining ADC, 100 MHz, 14 bit, Input -1V to +1V
- GEMEX, NYXOR: nXYTer triggered readout
- TAMEX3: 16+1 channel TDC, TOT, time and TOT 10 ps RMS, for R3B Neuland readout
- TAMEX2: 16, 32 channel TDC, TOT, time and TOT 10 ps RMS, for general purpose
- POLAND: SEM Grid and MWPC readout for FAIR accelerator beam control

Planned

- CLKTDC: 400ps TDC, Low cost, 128 input channels, PCB in layout phase
- FEBEX5: Pipelining ADC, 500 MHz, 16 bit, Input -1V to +1V, Design started



Structure for `f_user.c`, `setup.usf` and `set_mo.usf` identical for VME and PCI Express based systems.

MBS commands, messages, data logging, data monitoring identical for VME PCIe systems.

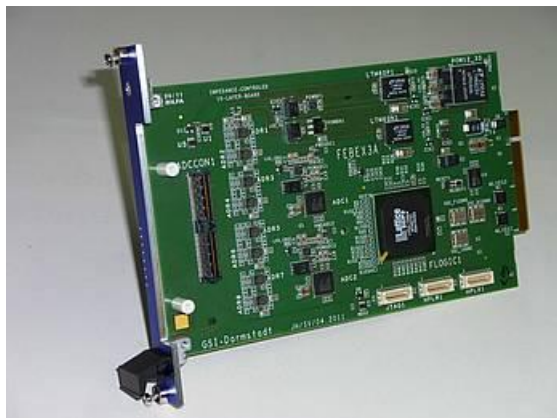
`setup.usf` needs only a few (static) changes (controller type, etc).

`set_mo.usf` needs usual customization for topology and node names.

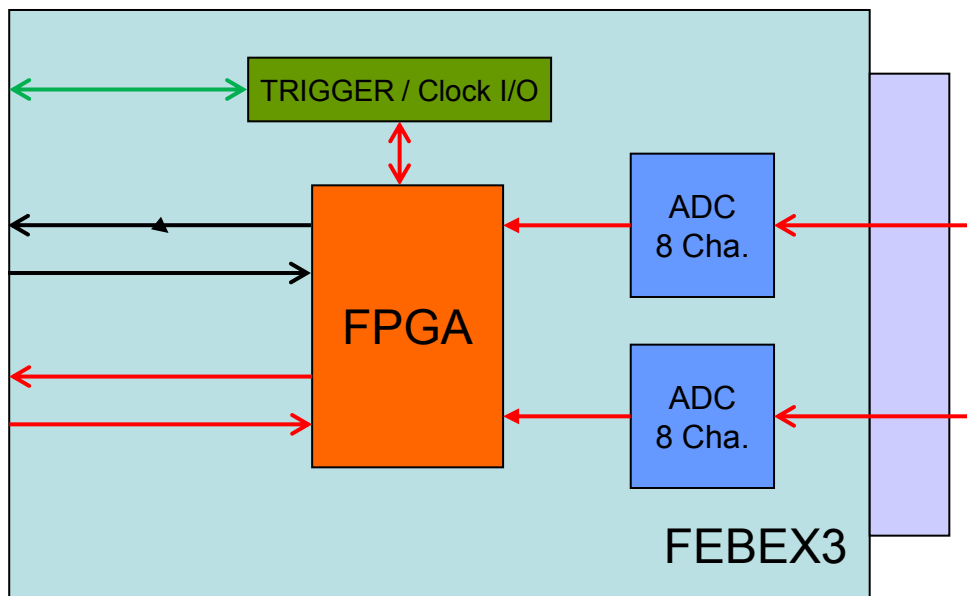
`f_user.c` must be customized (as usual).

FEBEX3: Pipeline ADC Front End Board

50 MHz Sampling Rate, 14 Bits



2 pairs of 2Gb/s
serial data I/O



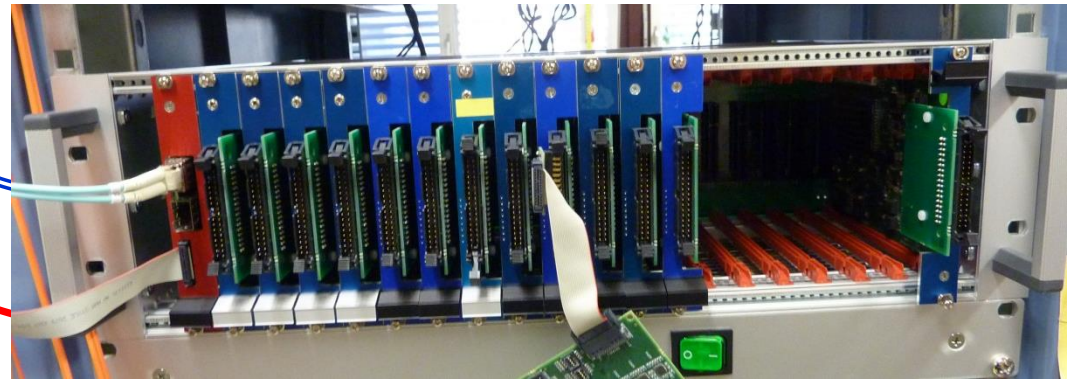
Various Adapter boards
available

Differential Input
-1V ---- +1V

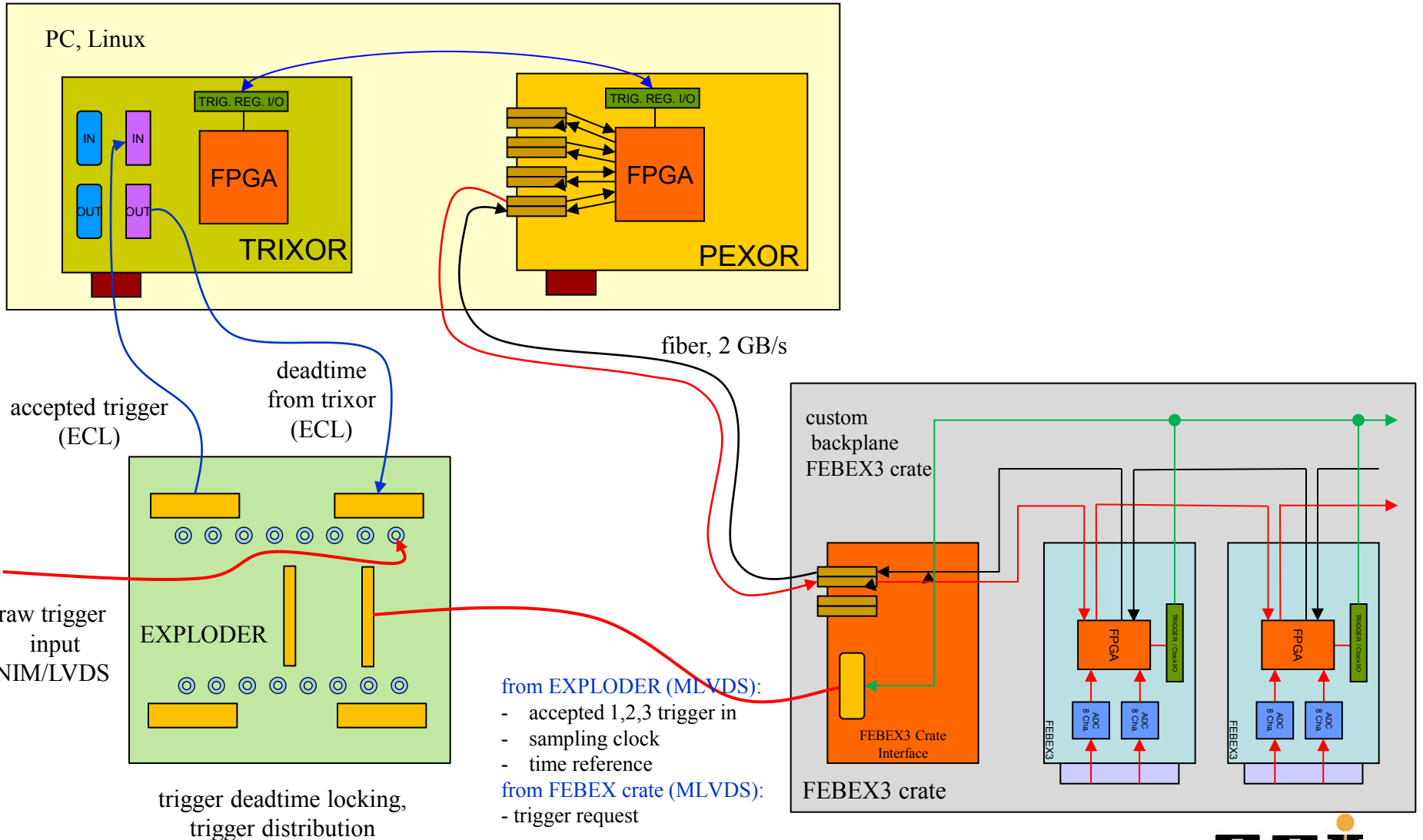
Basic FEBEX3 MBS readout System



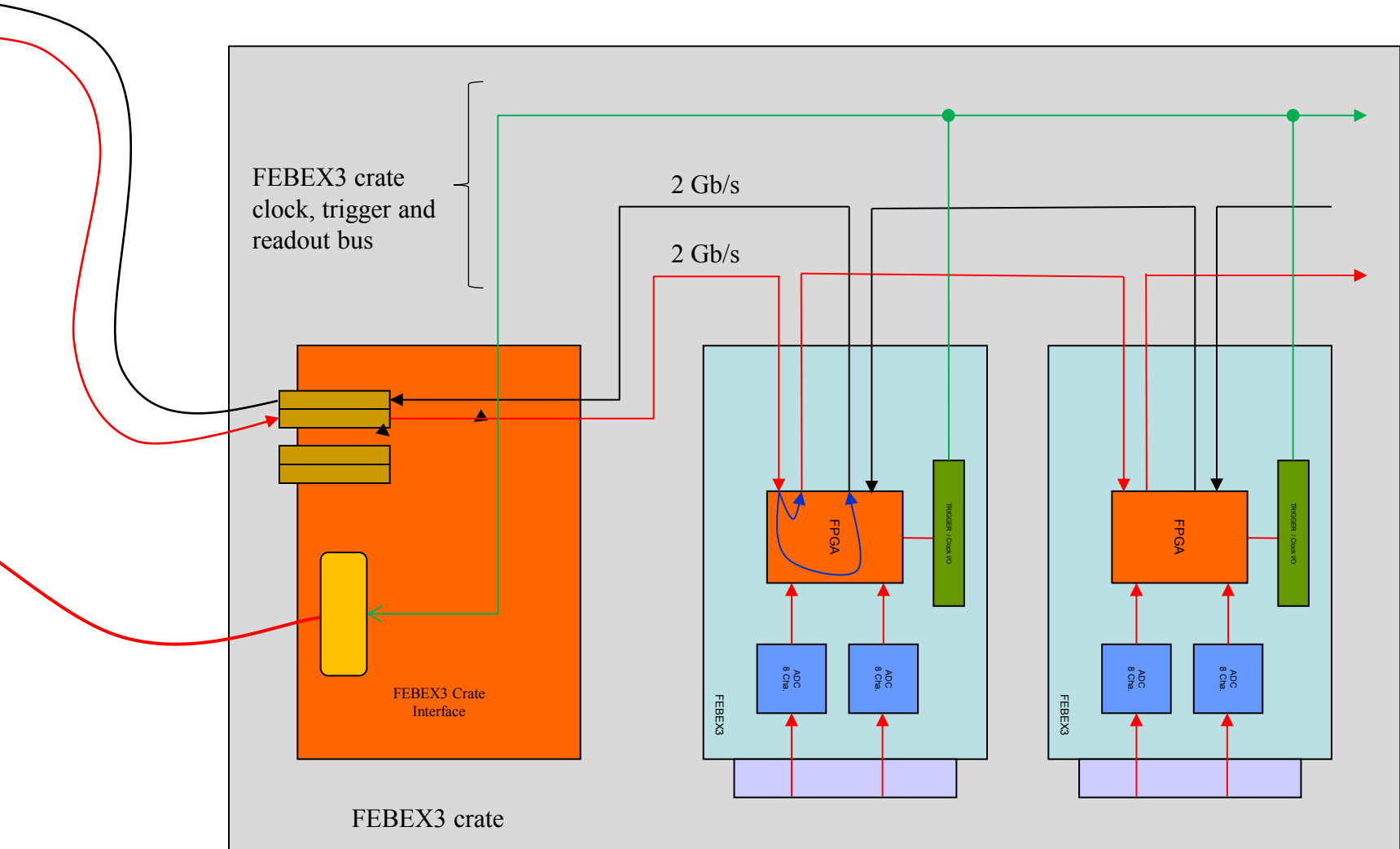
fiber, 2 GB/s



Basic FEBEX3 MBS readout System



FEBEX3 Crate





Data Transfer Protocols

- GOSIP for optical fibers between PEXOR/KINPEX and front-end boards (FEBEX3, TAMEX, ...)
- PCI Express between PEXOR/KINPEX and PC DRAM

Data Transfer Protocol for PEXOR-FEBEX3 (GOSIP) (Gigabit Optical Serial Interface Protocol)

Star – versus Chain topologies between frond-ends and concentrator/readout boards:

Star: Easy transfer protocol, limited number of frontends. Hubs needed

Chain: (More) complicated transfer protocol, highly scalable in size and speed.

GOSIP:

supports data transfer between PEXOR and FEBEX, TAMEX, ... (front-ends) for systems of sizes between 1 front-end and 1024 front-end cards. For data speed optimization, each front-end is equipped with two pairs of 2 Gb/s serial transfer lines. See previous transparencies, see GOSIP protocol below.

GOSIP supports **Chain Initialization** of all four PEXOR chains. During initialization each front-end gets a unique module id, starting from 0 with the front-end closest to the PEXOR,

GOSIP supports **Transparent Mode** (read and write access) for slow control and setup issues from PEXOR to each frontend with a speed of ~ 100K accesses/s:

A r/w request is send from the PEXOR upstream to front-end 0. Each frontend examines if the request is for itself. If yes, it passes the result (read: ack and data, write ack) downstream to the PEXOR. If the request is for a different front-end the request is passed upstream to the next front-end in the chain. Due to the chain topology chosen, data coming downstream a chain, can pass all front-ends, via a FIFO, without intervention, until it reaches finally the PEXOR.

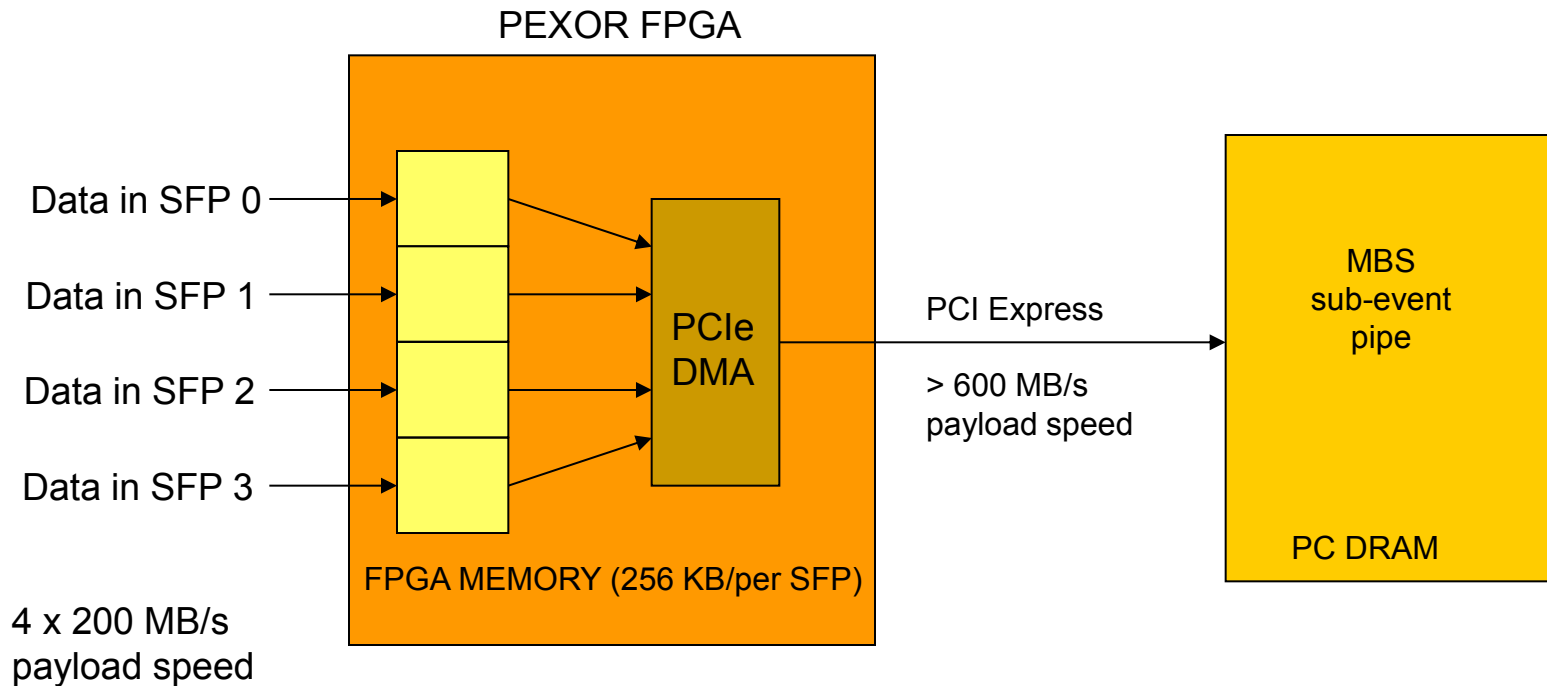
GOSIP supports **Token Mode** for fast data transfer from front-ends to PEXOR:


In token mode the readout token is initiated by the PEXOR, which sends the token to the first front-end. This sends its data packet down to the PEXOR and sends afterwards the token to the next front-end in the chain. The last front-end in the chain sends the token after sending its data packet back to the token, which completes the transfer. payload speeds of 4 times 200 MB/s have been measured.

Fast Token Data Transfer Options

PEXOR -> PC DRAM / MBS

- 1) **Parallel token data sending** on for all connected front-end chains. Sequential DMA, initiated from MBS user readout function, from PEXOR to PC DRAM. Used for “small” data sizes.
- 2) **Sequential token data sending** for all connected front-end chains. Sequential DMA, automatically initiated by PEXOR FPGA from PEXOR to PC DRAM. Used for “big” data sizes. Limit is PC DRAM, not PEXOR memory.

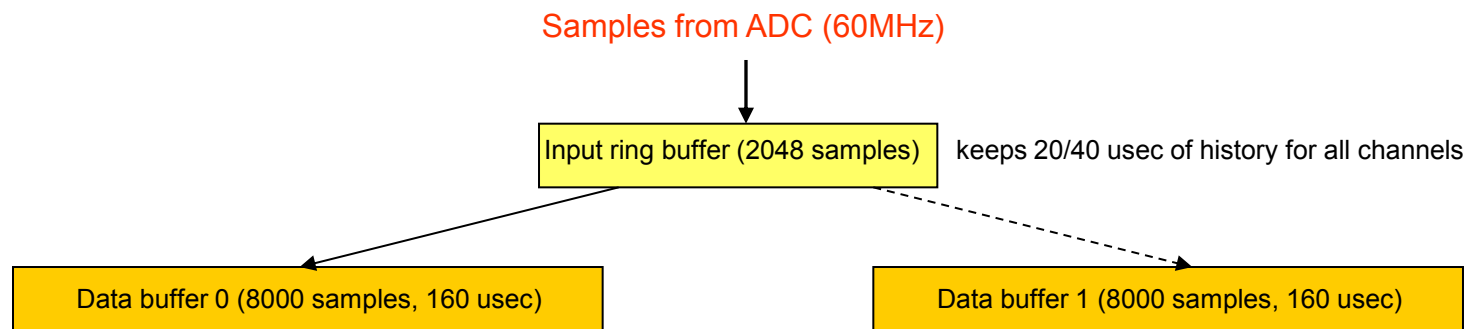




Selected Features of
Pipeline ADC
FEBEX3 and **FEBEX4**

FEBEX3 Digital Frontend

FEBEX3 input stage implemented in FPGA for each ADC Input channel:



Input ring data buffer 0/1 for each ADC channel

Input ring buffer accepts ADC samples with the speed of the ADC *without interruption*

On occasion of a trigger, content of Input buffer is copied with the speed of the ADC in a toggling mode into one of the data buffers

Length of input ring buffer defines maximum pre trigger window (see also next slide): 50 MHz: 40 usec, 100 MHz 20us

Length of data buffer defines the maximum trigger window length (for hit finders/ energy filter): 50 MHz: 160 usec, 100 MHz 80us

Similar ring – and data buffer structure for GEMEX, TAMEX1/2 and other front-end boards

(Selected) FEBEX3 Features

- FEBEX3: 16 channels, 50 MHz, 12 bit or 14bit, +/- 1 V input signals
- ADC input circular buffer (per channel, 40 usec)
- Double signal trace buffer (per channel, 160 usec)
- Trigger windows with pre-, post trigger time
- Ancillary I/O:
 - IN (3): Physics trigger, sync. trigger
 - IN: Common clock (for hit timing)
 - OUT: Trigger request from hit finder. Common OR from all channels
- Two hit finder (or self trigger) algorithms (3 step, bibox filter)
- Dead time clear before readout possible (double trace buffer!)
- Data Output: Complete trace in trigger window and hit time (from common clock)
- Energy filter algorithm for charge sensitive pre-amplifier signals (1.9 keV for ^{60}Co)
- Various data format options: traces, energy filter traces, compressed energy and time, etc

(Selected) FEBEX3 Setup Parameters

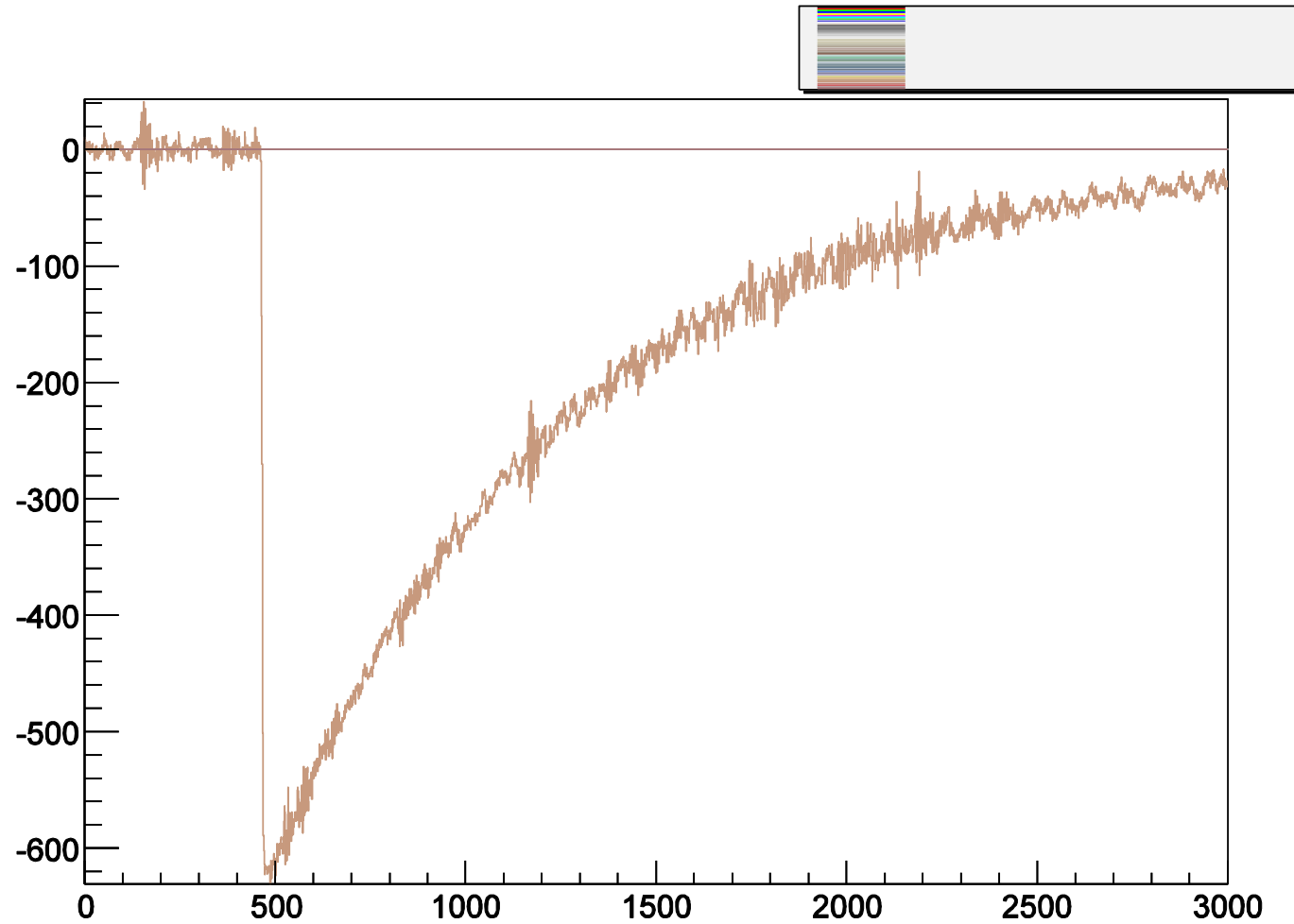
All parameters per channel:

- Enable/Disable Channel
- Enable/Disable Self trigger (hit finder)
- Enable/Disable Data reduction
- Set positive or negative input signals
- Data format selection (compressed, with traces, with filter traces, ...)

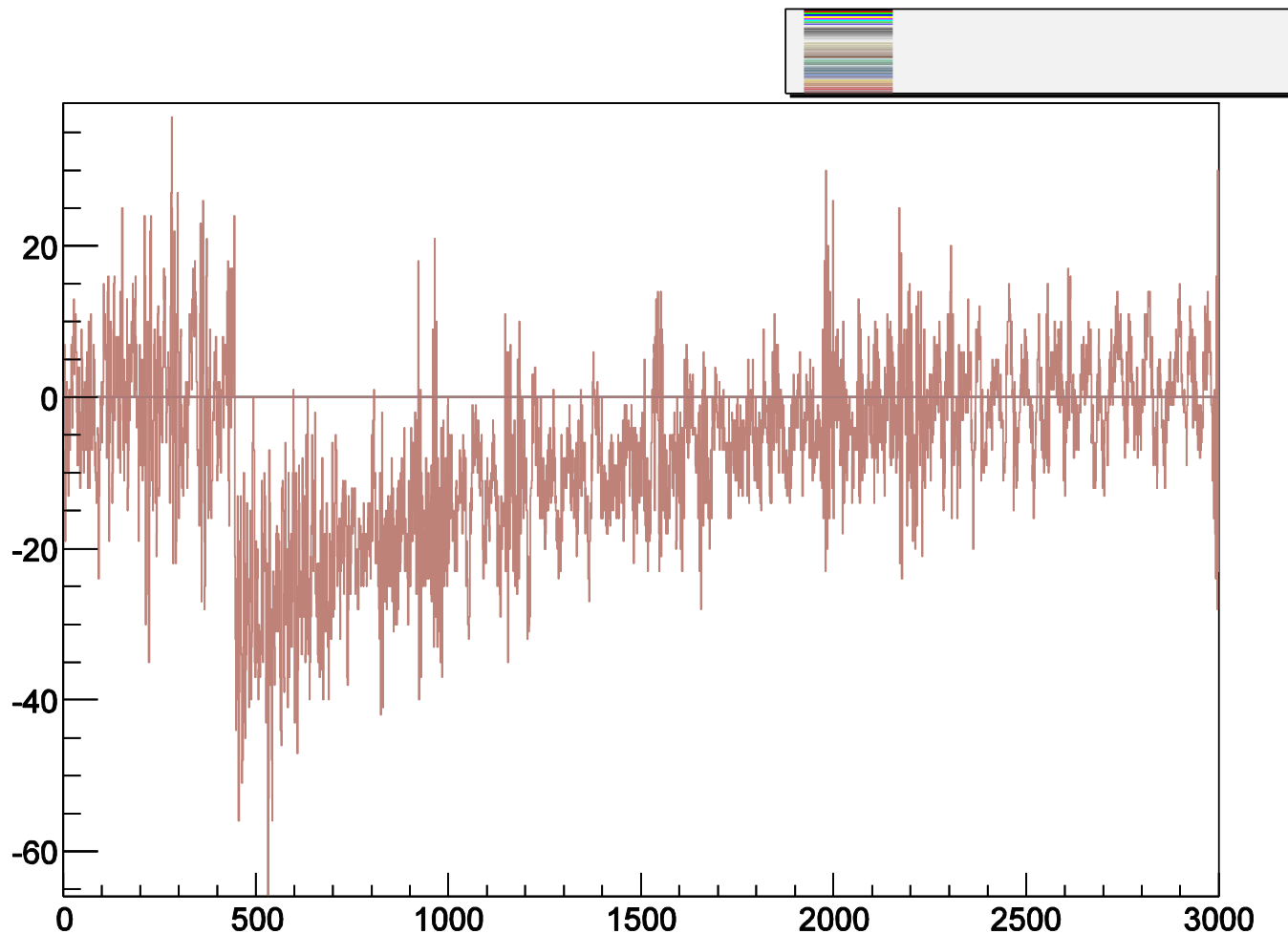
- Pre-trigger time (0 - 40 us, in nr. of ADC samples (2048), granularity 20 ns)
- Trigger window length (0 - 160 us, in nr. of ADC samples (8000), “

- Select Self trigger methods:
 - 3 step
 - Trapezoidal filter, select one out of 4 different scanning frequencies
60 -, 30 -, 15 -, 7.5 MHz, Signal thresholds

Example Trace (pulse height: ~ 300 mV)



Example Trace (pulse height: ~ 15 mV)





“Dead-time free” Data Taking
with Triggered MBS DAQ
???

Fast Token Data Transfer Options

Front-ends -> PEXOR

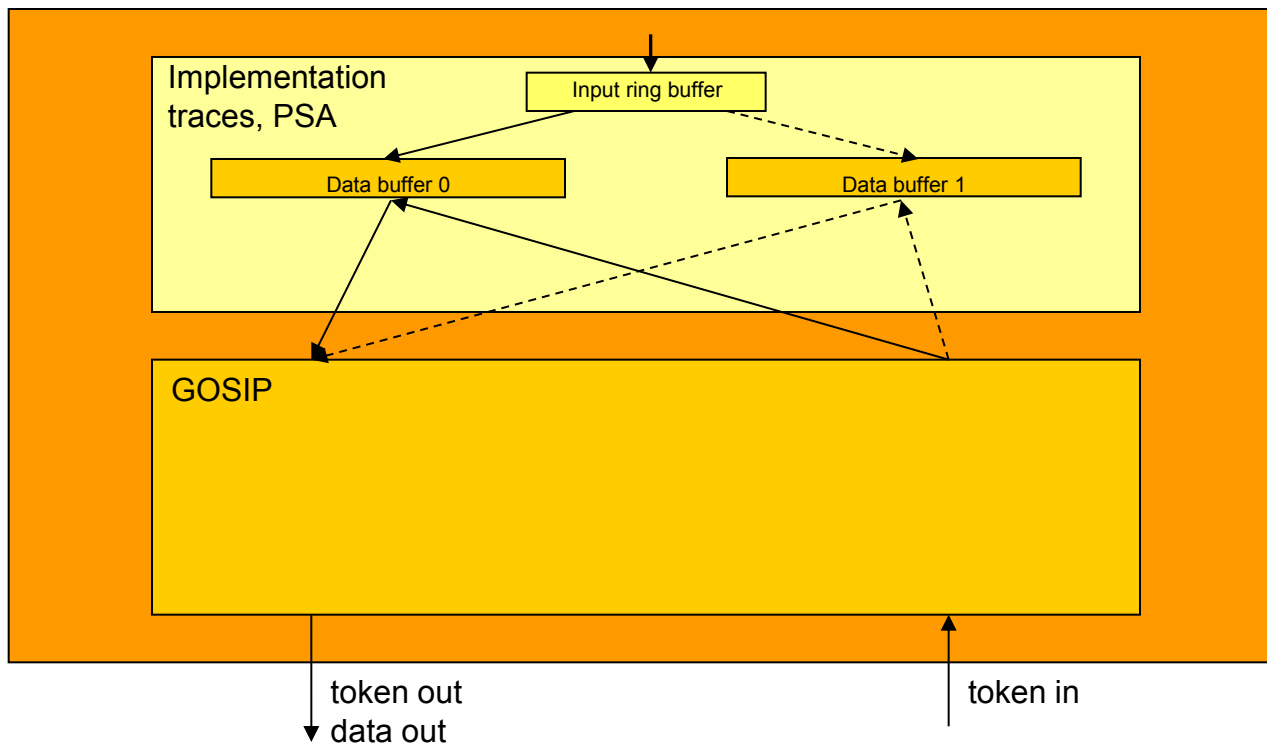
1) Wait for data ready mode:

GOSIP waits until front-end declared data ready before sending content of data buffer 0/1.
For triggered systems!

2) No wait Token:

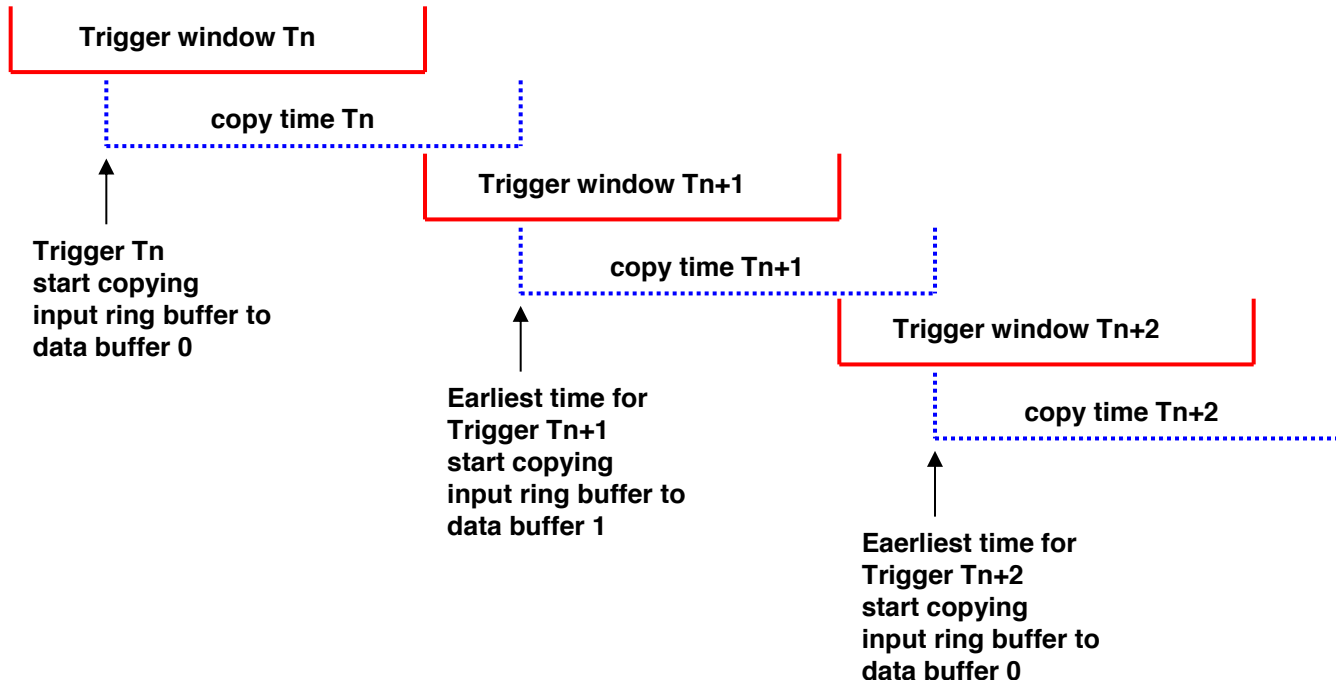
GOSIP sends data from data buffer 0/1 immediately on token arrival time down stream.
Useful for “triggerless” systems!

FEBEX / EXPLODER FPGA (per input/ADC channel)



“Dead-time” free Data Acquisition with Trigger Windows

Trigger window adjustable from 1 to 8000 ADC samples (FEBEX3)



Note: - Trigger windows can be adjacent (no dead time)

- Avoid overlapping trigger windows by setting conversion time (minimum time between two triggers) to trigger window length
- Double data buffers 0/1 allow to release the system dead time **before** actual readout of data from FEBEX to PEXOR.
- Very large data sizes (data rates > 200 MB/s) might delay dead time release in a sense, that adjacent trigger windows (dead time free) are not possible in all cases. In this case the system is not anymore dead time free. This situation is also present in so called “trigger less” systems, when data rates produced in the front-ends, exceeds the bandwidth of a transfer channel.



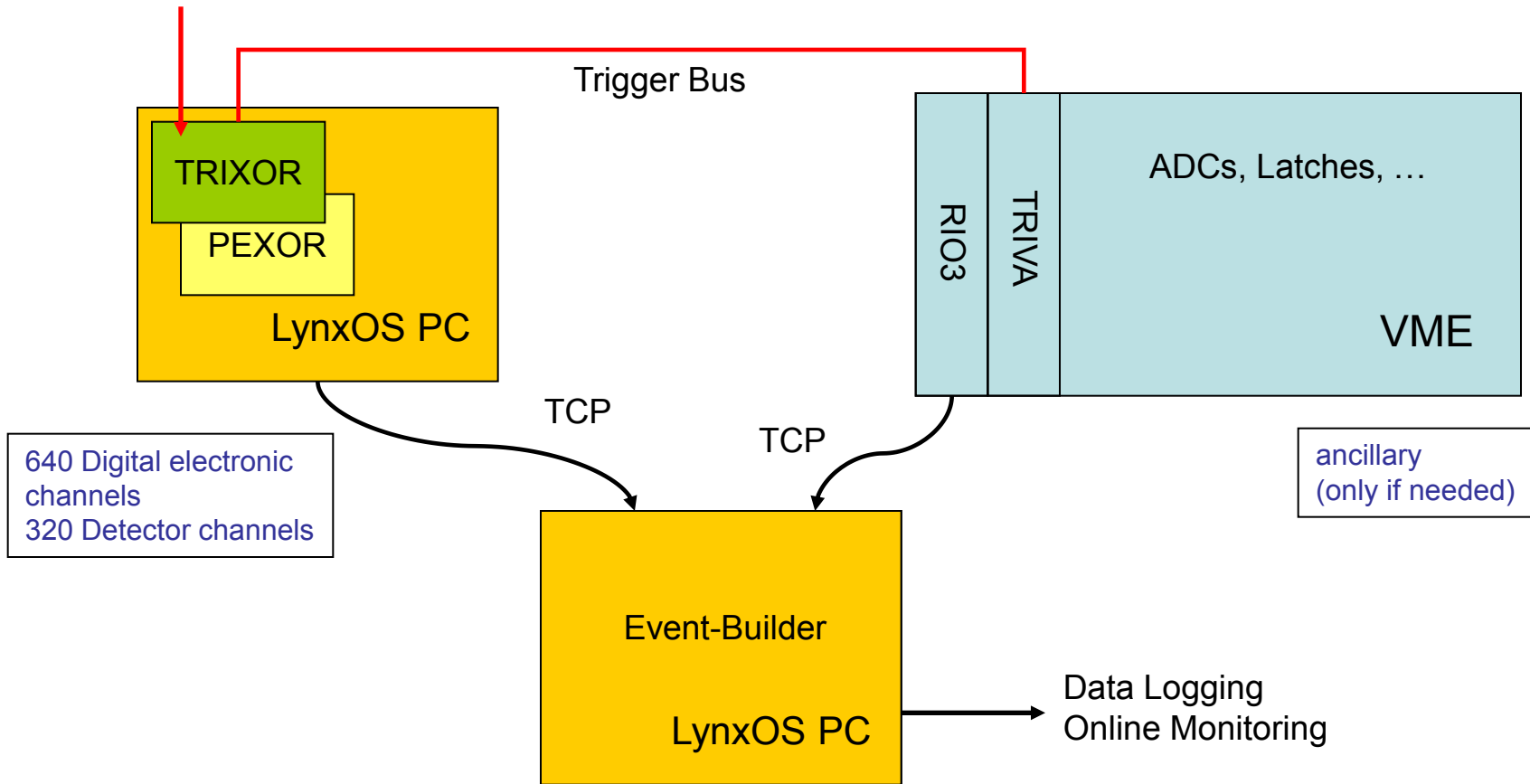
A real Experiment with
PEXOR, TRI XOR, FEBEX

The Search for Element 119, 120 at GSI

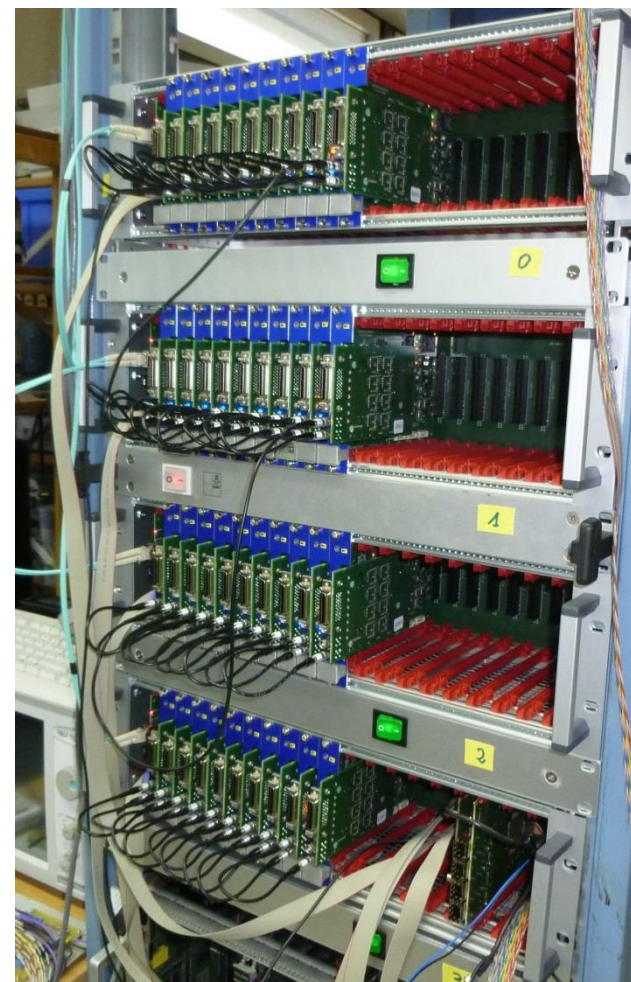
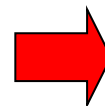
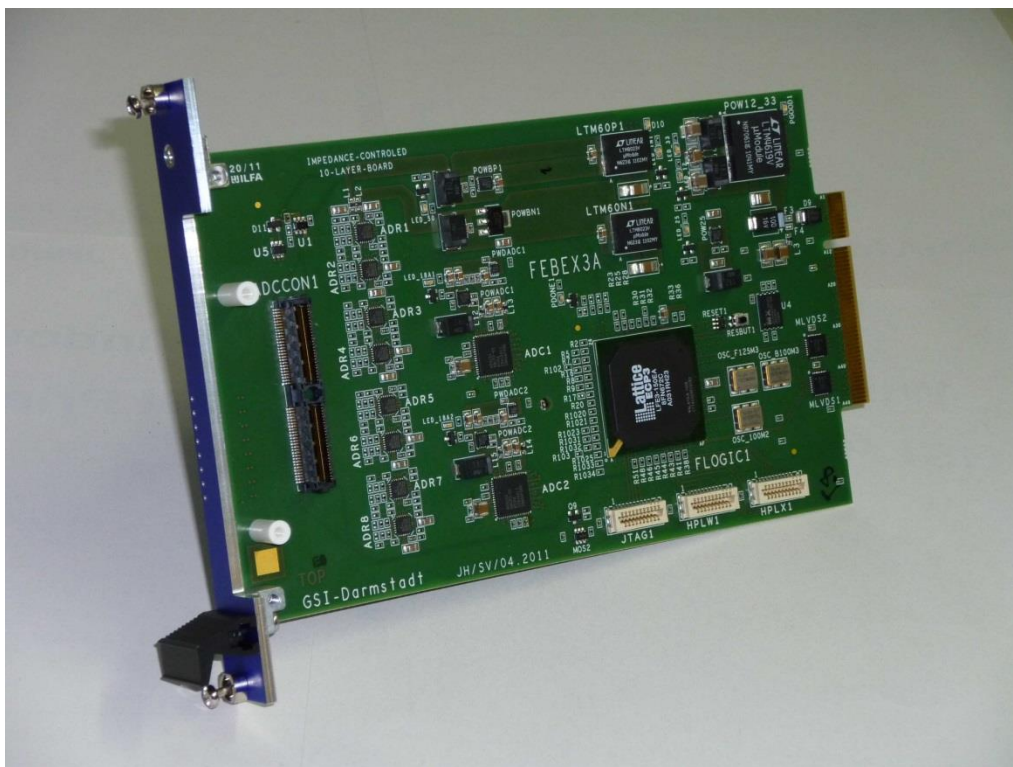
Connecting VME and PCI Express Systems

119/129 MBS DAQ 2011/12

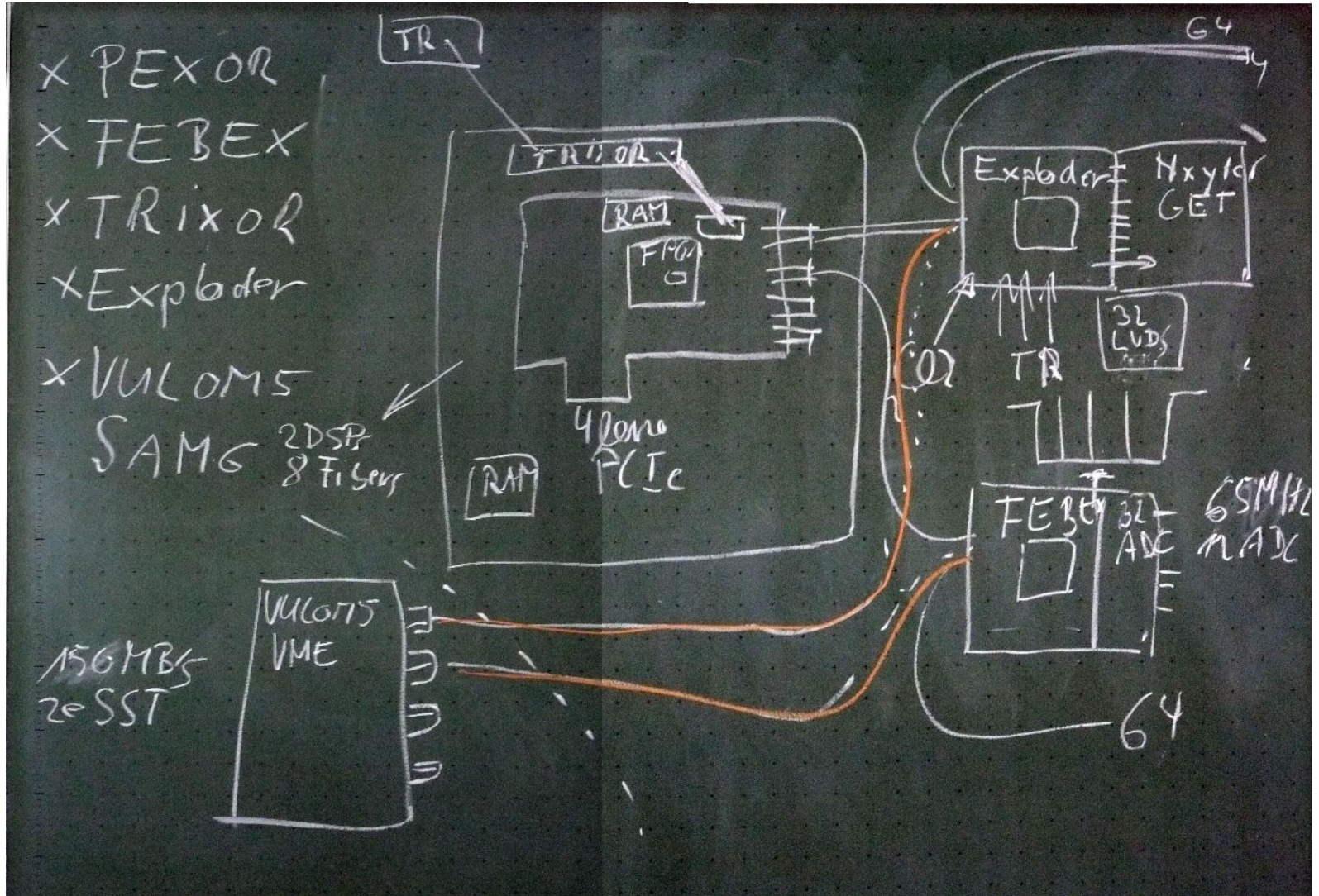
Accepted Trigger In



FEBEX3, 640 Digital Channels



Summary



```

int PEXOR_GetPointer( long PEXOR_BASE_OFF, volatile long *pl_virt_sram, s_pexor *ps_pexor );
int PEXOR_Read( s_pexor *ps_pexor, long *addr, long *data );
int PEXOR_Slave_Read( s_pexor *ps_pexor, long l_sfp, long l_slave, long l_addr, long *data );
int PEXOR_Slave_Write( s_pexor *ps_pexor, long l_sfp, long l_slave, long l_addr, long l_data );
int PEXOR_TK_TX_Mem_Read( s_pexor *ps_pexor, long *addr, long *data );
int PEXOR_TK_Mem_Write( s_pexor *ps_pexor, long *addr, long *data );
int PEXOR_RX_Clear( s_pexor *ps_pexor );
int PEXOR_RX_Clear_Ch( s_pexor *ps_pexor, long ch );
int PEXOR_RX_Clear_Pattern( s_pexor *ps_pexor, long l_ptn );
int PEXOR_TX_Reset_Ch( s_pexor *ps_pexor, long ch );
int PEXOR_SERDES_Reset( s_pexor *ps_pexor );
int PEXOR_TX( s_pexor *ps_pexor, long comm, long addr, long data );
int PEXOR_RX( s_pexor *ps_pexor, int sfp_id, long *comm, long *addr, long *data );
long PEXOR_TK_Data_Size( s_pexor *ps_pexor, long l_sfp, long slave_id );
long PEXOR_TK_Mem_Size( s_pexor *ps_pexor, long l_sfp );
long PEXOR_TK_Mem_Read( s_pexor *ps_pexor, long l_sfp, long **pl_dat );
int PEXOR_RX_Status( s_pexor *ps_pexor );
int PEXOR_Port_Monitor( s_pexor *ps_pexor );
int PEXOR_Show_Version( s_pexor *ps_pexor );
int PEXOR_SFP_Active( s_pexor *ps_pexor, long l_sfp );
int PEXOR_SFP_Disable( s_pexor *ps_pexor, long disa );
int PEXOR_SFP_Show_FIFO( s_pexor *ps_pexor, long ch );
int PEXOR_SFP_Clear_FIFO( s_pexor *ps_pexor );
int PEXOR_Set_LED( s_pexor *ps_pexor, long l_led );
int PEXOR_LED_On( s_pexor *ps_pexor );
int PEXOR_LED_Off( s_pexor *ps_pexor );
int PEXOR_Set_Data_Reduction( s_pexor *ps_pexor, long l_flag );

```



```
|/////
#ifndef PEXOR_NAME
#define PEXOR_NAME "PEXOR"

#include <stdio.h>
#include <stdlib.h>

// byte address offset for PEXOR
#define PEXOR_BASE 0x0
#define PEXOR_REQ_COMM 0x21000
#define PEXOR_REQ_ADDR 0x21004
#define PEXOR_REQ_DATA 0x21008
#define PEXOR_REP_STAT 0x2100c
#define PEXOR_REP_CLR 0x2100c
#define PEXOR_REP_STAT_0 0x21010
#define PEXOR_REP_STAT_1 0x21014
#define PEXOR_REP_STAT_2 0x21018
#define PEXOR_REP_STAT_3 0x2101c

#define PEXOR_REP_ADDR_0 0x21020
#define PEXOR_REP_ADDR_1 0x21024
#define PEXOR_REP_ADDR_2 0x21028
#define PEXOR_REP_ADDR_3 0x2102c

#define PEXOR_REP_DATA_0 0x21030
#define PEXOR_REP_DATA_1 0x21034
#define PEXOR_REP_DATA_2 0x21038
#define PEXOR_REP_DATA_3 0x2103c

#define PEXOR_RX_MONI 0x21040
#define PEXOR_RX_RST 0x21044
#define PEXOR_SFP_DISA 0x21048
#define PEXOR_SFP_FAULT 0x2104c

#define PEXOR_SFP_FIFO 0x21050

#define PEXOR_REP_TK_STAT 0x21060
#define PEXOR_REP_TK_HEAD 0x21070
#define PEXOR_REP_TK_FOOT 0x21080
#define PEXOR_REP_TK_DSIZE 0x21090
#define PEXOR_TK_DSIZE_SEL 0x210a0

#define PEXOR_TK_MEM_SIZE 0x210b0

#define PEXOR_PROG_VERSION 0x211fc

#define PEXOR_TK_MEM 0x28000

#define PEXOR_TK_MEM_0 0x100000
#define PEXOR_TK_MEM_1 0x140000
#define PEXOR_TK_MEM_2 0x180000
#define PEXOR_TK_MEM_3 0x1c0000
```


Acknowledgements

White Rabbit core team:

D. Beck, M. Kreider, C. Prados, S. Rauch, W. Terpstra, M. Zweig

EE: **Jan Hoffmann:** Design of White Rabbit Timing Receivers
 Joern Adamczewski-Musch: Linux device drivers
 Jochen Fruehauf: VFTX, MBS TOF setups

Thank You