

A new BPM-FEC
Concept, Design and Test
Results

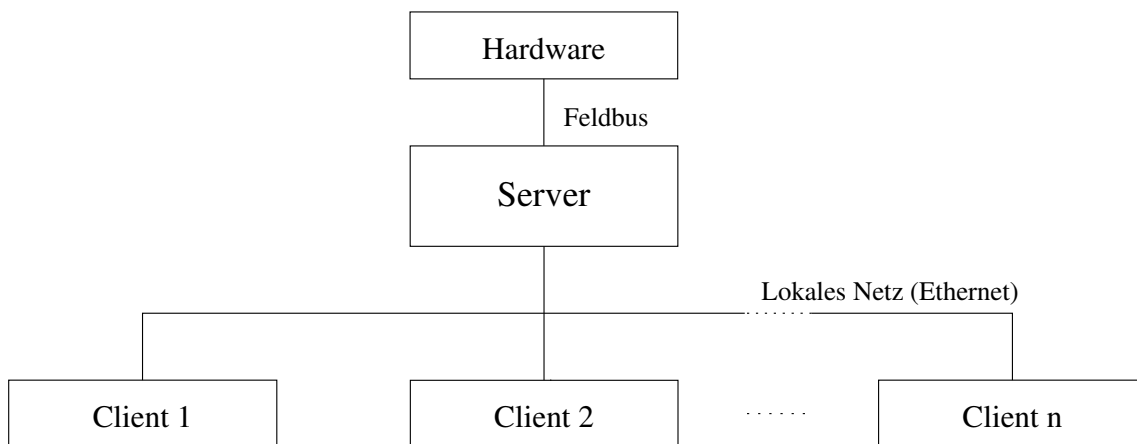
J. Tutas, M. Weber
III. Physikalisches Institut
RWTH Aachen
52056 Aachen

October 28, 1998

Contents

- Analysis
- Requirements
- Solutions
- Realisation
- Test results

Analysis of HERA control system

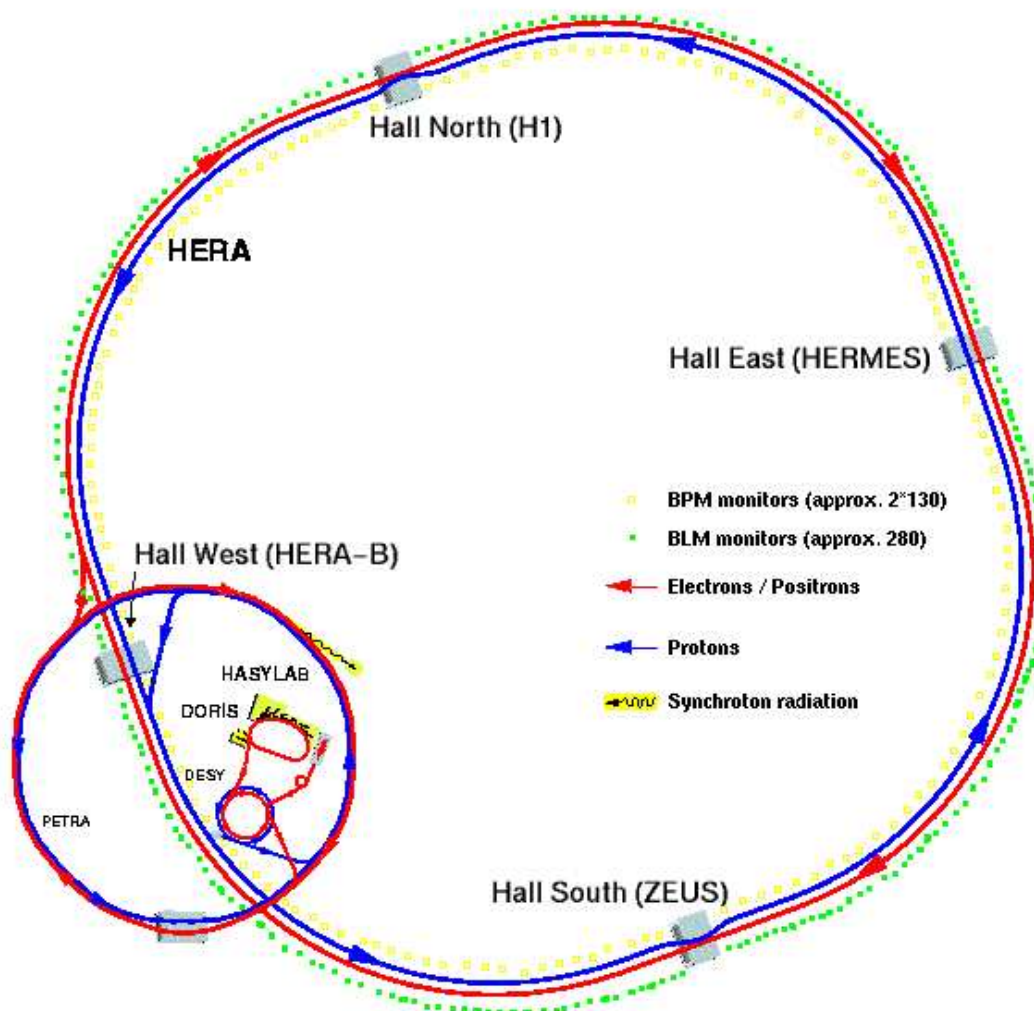


- Three parts: hardware, server, client
- One server for a distinct piece of hardware
- Many clients can access one server (office/control room)
- Servers monitor hardware constantly
- Clients get buffered data (must not wait for hardware readout)
- Servers communicate over two channels

Why use a field bus? Let's discuss it at the example of the BPM Server.

Picture shows: Modules are located along the HERA accelerator. To reduce expensive wiring, connect many modules with one data transfer line, give each module a specific address. Transfer data serially instead parallel.

The BPM Server



- Beam Position Measurement (BPM)
- Beam Loss Measurement (BLM)
- Alarm detection → beam loss archiving
- Data conversion: raw → physical meaning

The BPM server has proved to be very reliable, why design a new one? For two main reasons:

1. The BPM server is slow. Slow means: Limited to the SEDAC data transfer speed. We should improve this.
2. There exists a vast amount of different server types (e. g. the HIT FEC is a Windows FEC, the BPM FEC was a DOS FEC and is now a Linux FEC). We should find a general concept that applies to all (also future) servers, simplifying maintenance and the creation of a new server and maintaining.

We need concurrent execution, because the BPM Server has to accomplish different tasks that are loosely coupled. (BPM/BLM/Alarm). Therefore: Each Task should run independent and not interfere with other tasks. The user wants to have information from all these systems at the same time.

Real time capability is needed, because the read-out time for one BLM is 32 ms, and the time to detect ready data in orbit mode of PPD is only 36 ms. We must guarantee that the BPM FEC detects fresh data *each* time, because elsewhere synchronisation is lost.

Last, the system should be easily maintained.

Requirements for a new BPM-FEC

- must fit into existing client/server model
- improvement of DAQ speed
- concurrent execution of different tasks
- real time capability
- portability
- modularity, scalability
- easy maintenance

and their solutions

(ACOP net kernel)

(Four servers
instead of one server)

(multithreading with POSIX Threads)

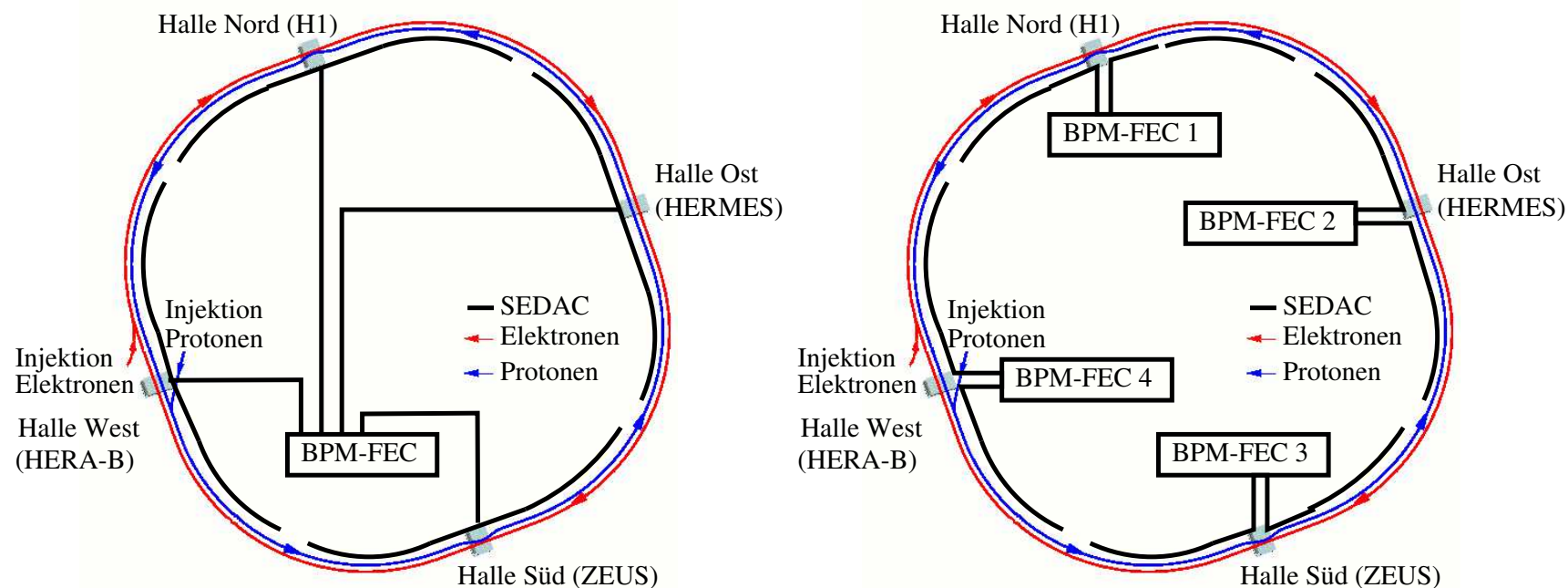
(The OS VxWorks
5.3.1 is real time capable)

(C++, stick to POSIX.1,
POSIX.4 and POSIX Threads)

(OOP concept)

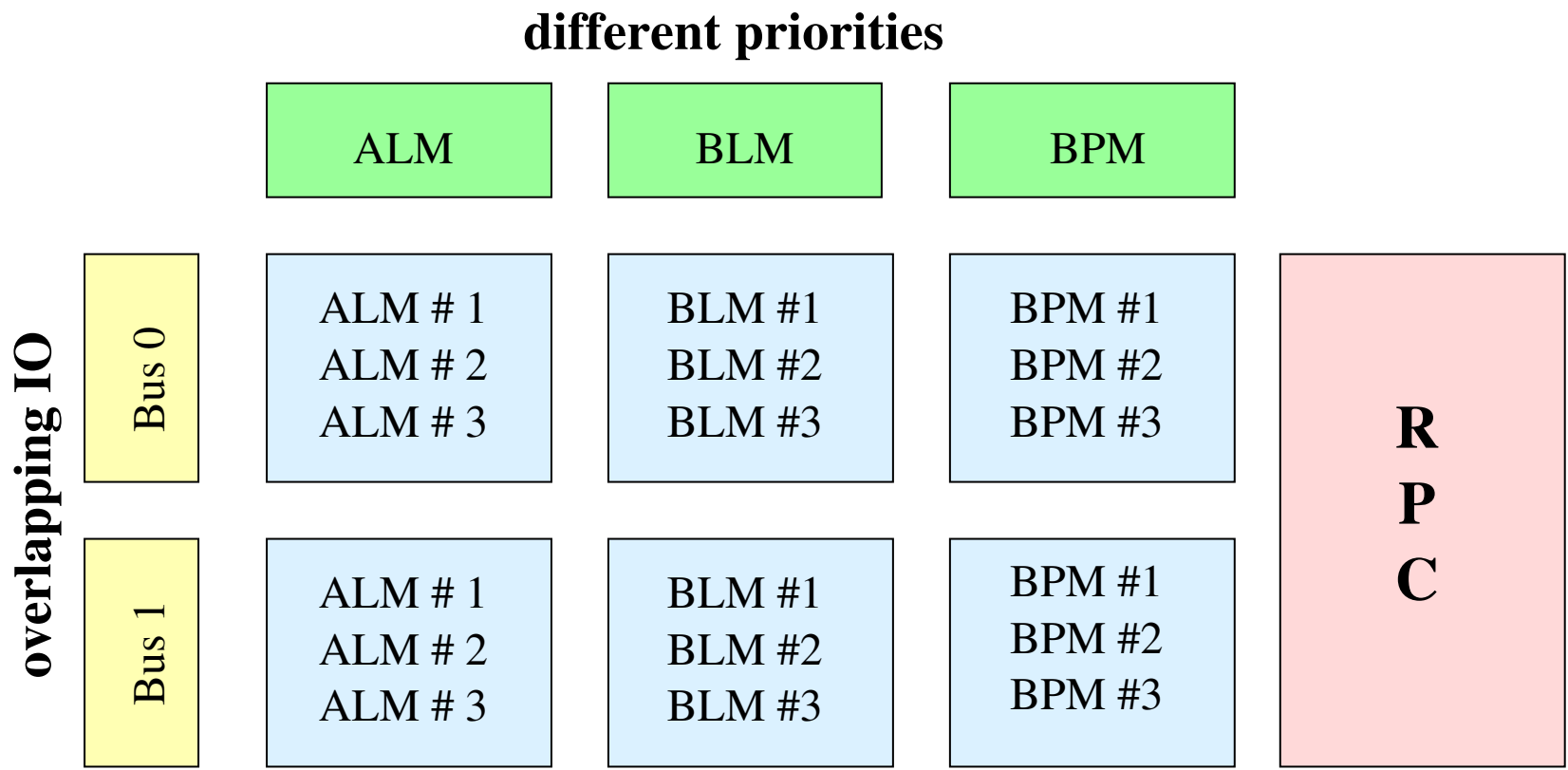
(OOP concept,
sourcecode documentation)

Four servers to improve DAQ speed



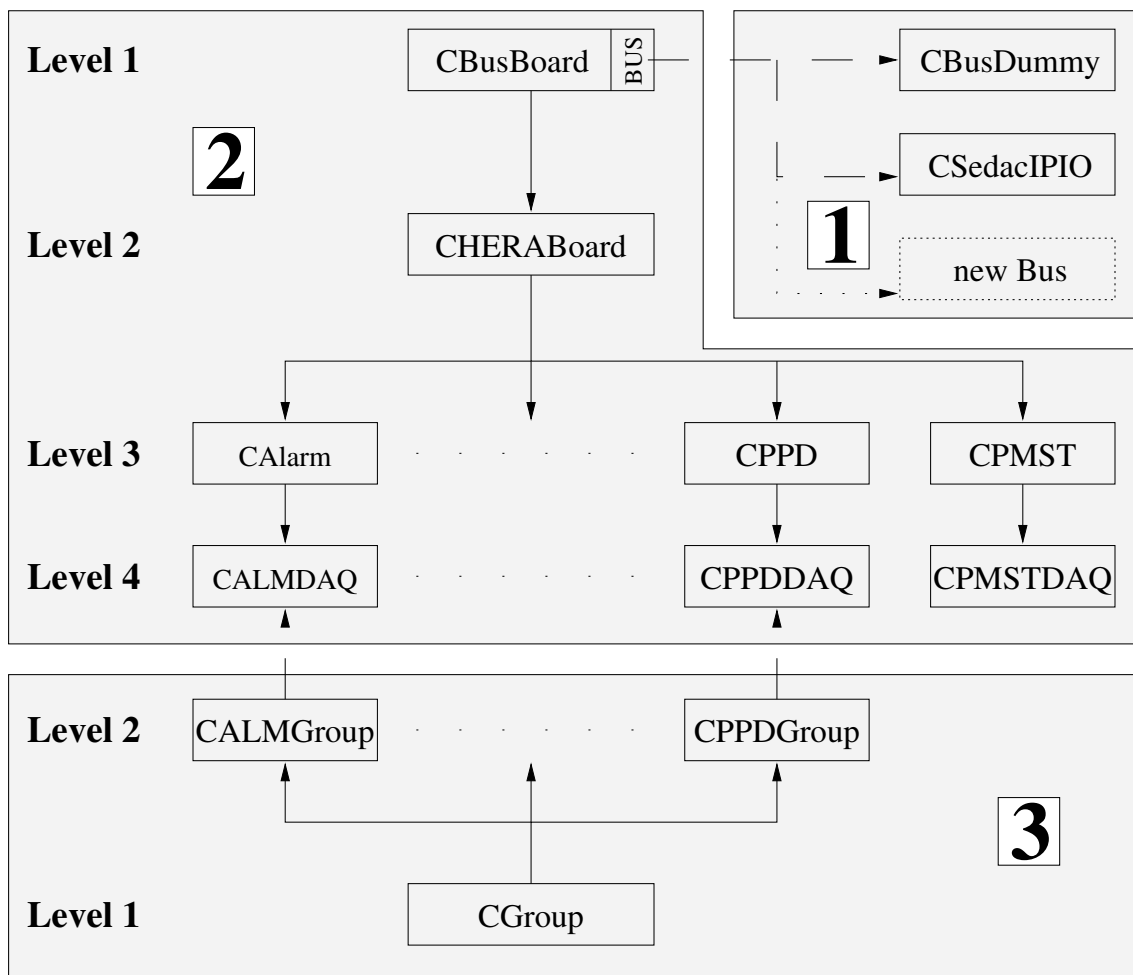
- gain in speed by factor 2
- FEC's communicate with a master (Ethernet)

Concurrency in the BPM Server



- synchronization plays an important role

The OO concept



1. Field bus IO
2. Boards communicating with server over field bus
3. Group of modules of the same type

The bus system class CBus

- The only hardware dependent class
- Classes developed: SEDAC and DUMMY IO
- Supports synchronous read and write
- Offers a default bus error handling function
- User can specify it's own error handling function
- Possible reactions on bus errors are
RETRY, IGNORE, ABORT
- SEDAC default error function: RETRY
until three errors in a second, then ABORT
→ only severe errors must be trapped
- Offers exclusive use of field bus for
time-critical operation

The field bus boards I

```
class CBusBoard
{
public:
    /* Input: reference to bus class, Address on this bus */
    CBusBoard(CBus & TheBus, BusAddress TheAddress);
    virtual ~CBusBoard();

    /* Print all relevant information about module */
    virtual void Print();

    /* do a general hardware reset of this module */
    virtual void Reset();

    /* connect to current operation status */
    virtual bool Connect();

    /* test if this module is ok, returns true on success */
    virtual bool Test() = NULL;

    /* disable, enable and test for enable */
    virtual void Disable();
    virtual void Enable();
    bool IsEnabled();

    /* test if module has been initialized */
    bool IsInitialized();

    /* lock and unlock access to this module, 0=success */
    virtual int Lock();
    virtual int Unlock();
};
```

The field bus boards II

Level 2: HeraBoard adds identification of a module with equipment name and number

Level 3: Abstraction from register accesses to functionality of the module.

```
// Reset Freeze signal
void CAlarm::ResetFreeze()
{
    assert(IsEnabled());

    // reset alarms
    Write(RESET, RESET_DATA);
}
```

Creation of **Read** and **Set** functions. Direct hardware access.

Level 4: New, specialized functionality for DAQ. Data is buffered → **Get** functions, special **DAQRoutine** reads all necessary registers

The Group classes

Level 1: Template class, can take a specified number of BusBoards. Performs special tasks:

- Initialization through special files (common part / board specific part)
- Starts / Stops DAQTask for all bus interfaces
- RPC access to any module through operator []
- Initializes Hardware (Connect/Reset)
- Locking hierarchy for MT safety

Level 2: Board specific class:

- Reads board specific part from initialization file
- Adds synchronisation in DAQTask
- Handles double buffering

Initialization files

%No,	Bus,	Address,	Name,	Offs. X,	Pos. X,	Offs. Y,	Pos. Y
0,	1,	3840,	WL126,	-1,	0,	-1,	0
1,	1,	4176,	WL200,	8,	197X,	0,	164Y
2,	1,	4432,	WL248,	8,	253X,	0,	226Y
				...			
129,	1,	3744,	WL54U,	8,	044X,	0,	044Y
130,	1,	3408,	WL540,	8,	096X,	0,	096Y
131,	1,	3488,	WL540,	8,	128X,	0,	128Y
1000,	12,	5,	1998,	mkw,	,	,	

common to all files	board specific part
---------------------	---------------------

- human readable files
- CSV^a format (supported by almost all spread sheet applications)

^aComma Separated Value

There exist five different DAQ modes. The first two are thought for injection, the other three for a stored beam. Synchronized data means: Turn “n” from each PPD module corresponds to the same physical turn in HERA-p.

BPM Data Acquisition

Five different operation modes supported:

Injection Mode: waits for an injection and reads 1024 consecutive turns from all PPD^a's

Single Turn Injection Mode: a preselected turn is read after injection from all PPD's

Orbit Mode: takes regularly synchronized position and intensity data from all PPD's. A special movie feature is supported.

Q Mode: reads 1024 synchronized, consecutive turns from all PPD's

SQ Mode: similar to Q mode. Instead of consecutive turns, 256 averages from 128 turns are collected from all PPD's

^aProton Position Digital module

ALM & BLM Data Acquisition

BLM: Data can not be synchronized.

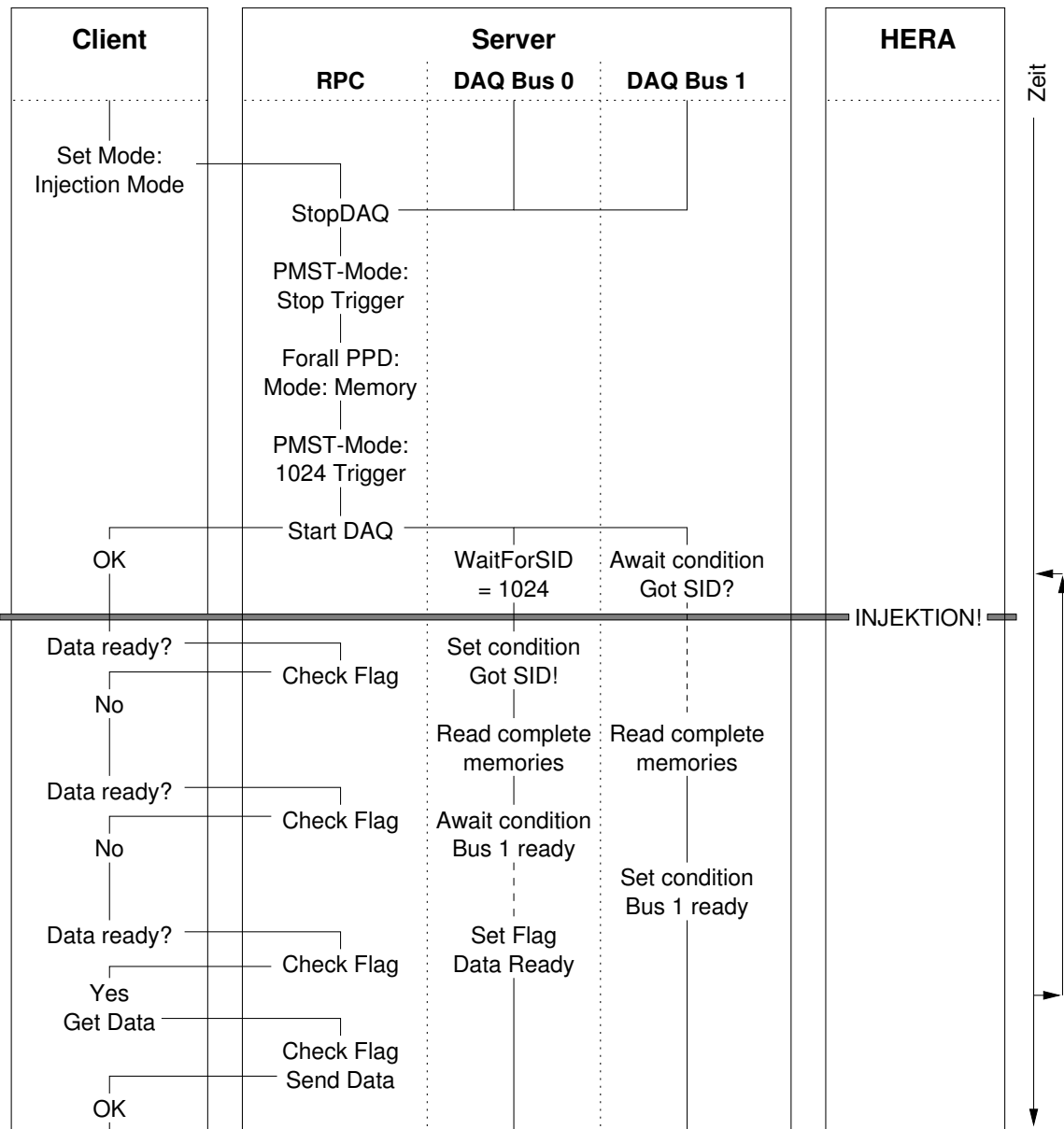
Monitors are read one after another.

Supports the following features:

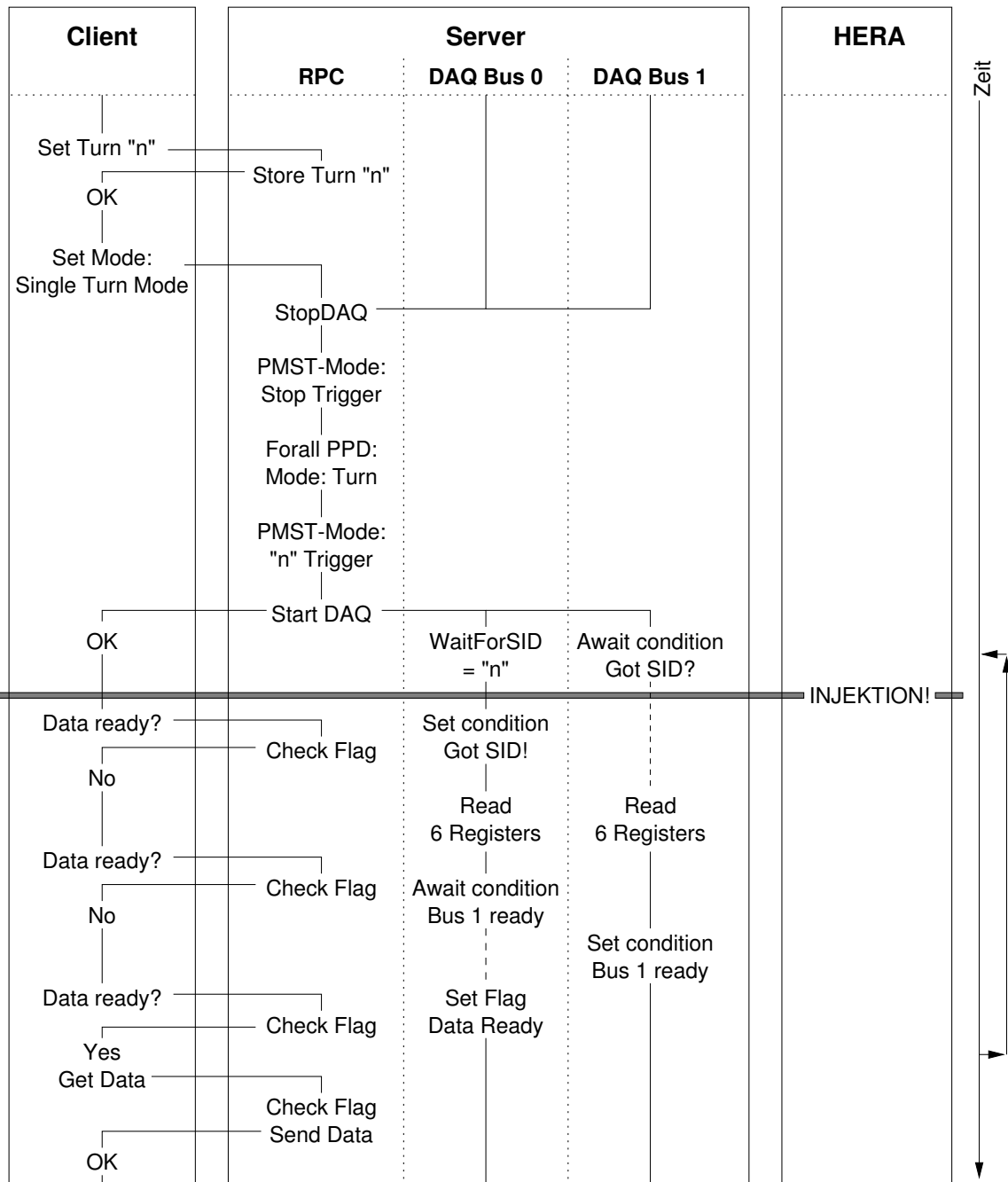
- For each Monitor, either average loss counts over 0.667 s or 86 s selectable
- automatic archiving of beam loss history started on beam dump
- can record beam loss “movies”
- adjusts automatically alarm generation limit to HERA Energy (important for energy ramping)

ALM: Watches alarm status in background. If a FREEZE is detected, BLM archiving is started.

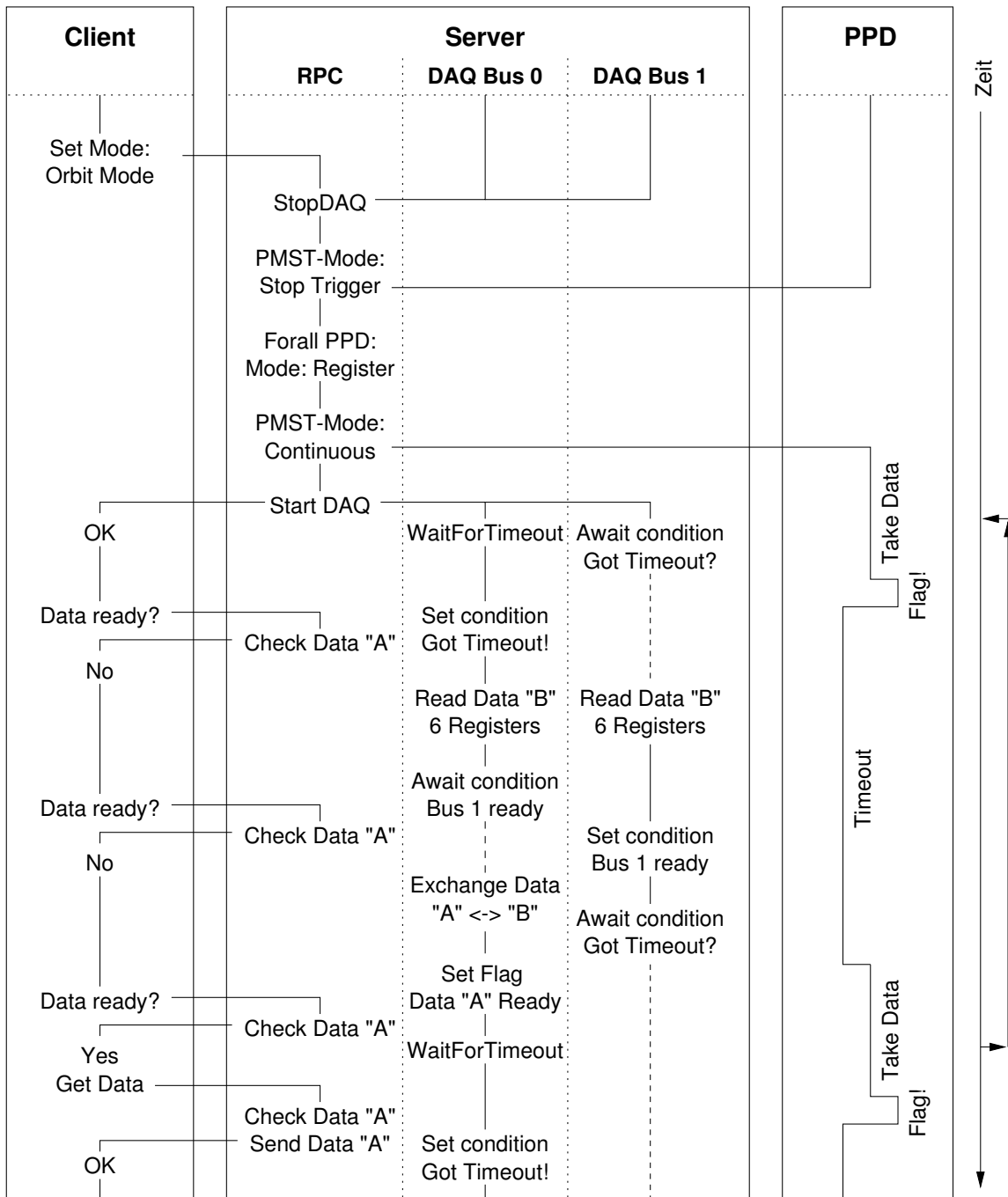
Injection Mode DAQ



Single Turn Injection Mode DAQ



Orbit Mode DAQ



CBus Test Results

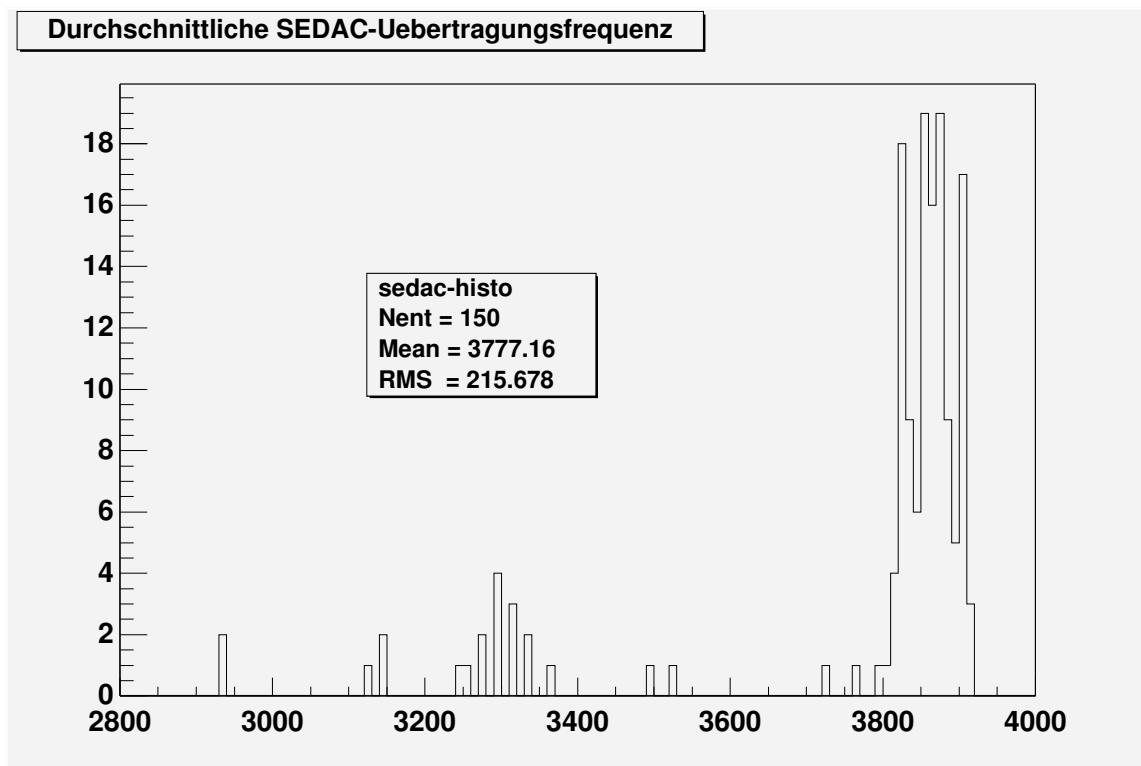
Maximum transmission speed (1 SEDAC line):

WL 93 : 4381.2 ± 1.6 Hz

WL 718 : 4140 ± 10 Hz

Maximum transmission speed (2 SEDAC lines):

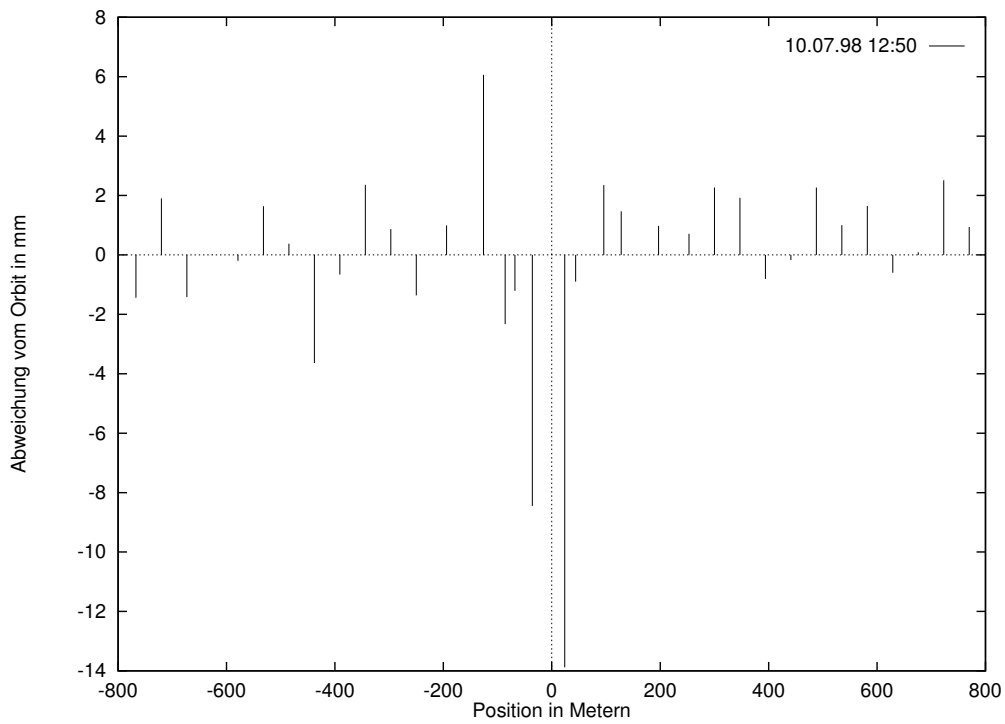
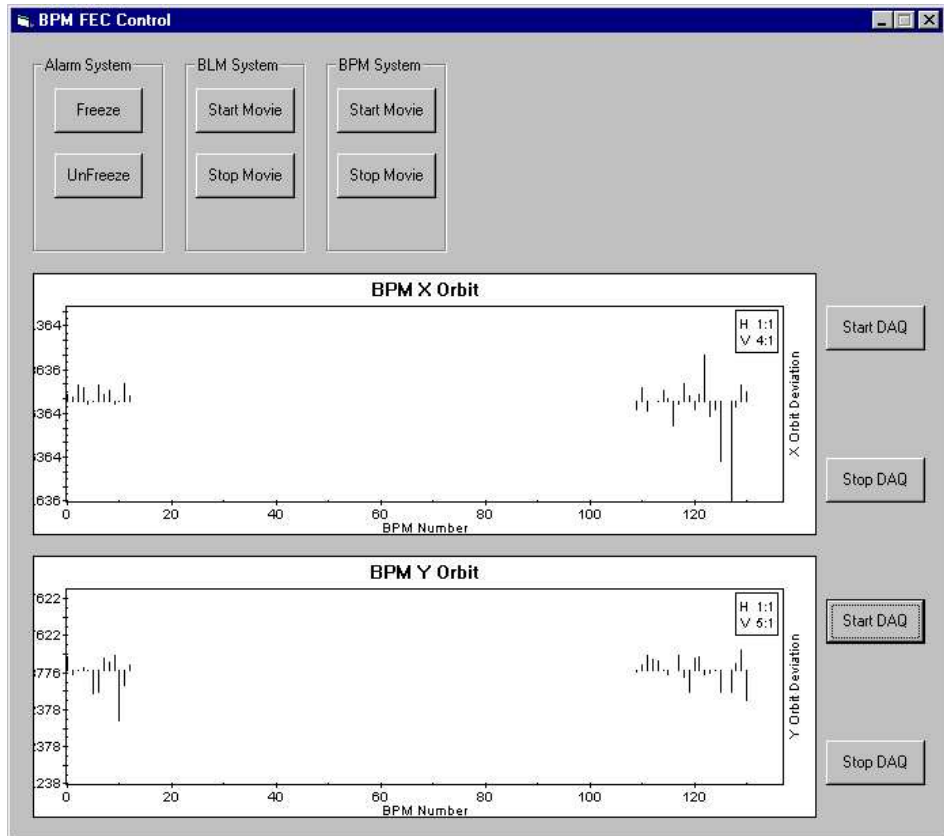
WL 93 : 4027.5 ± 1.0 Hz



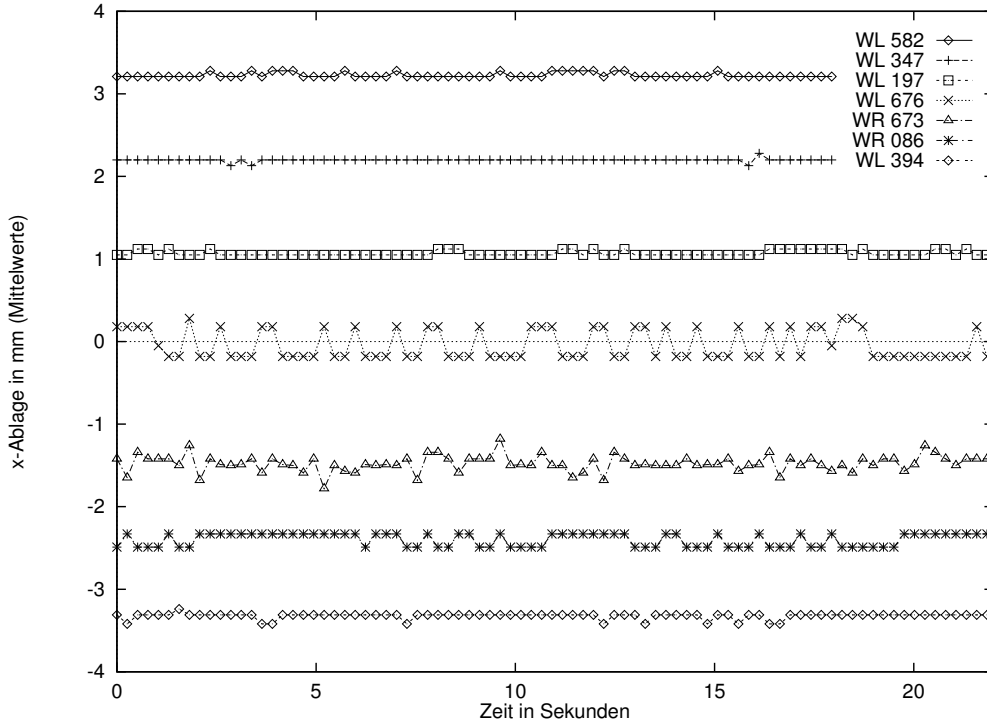
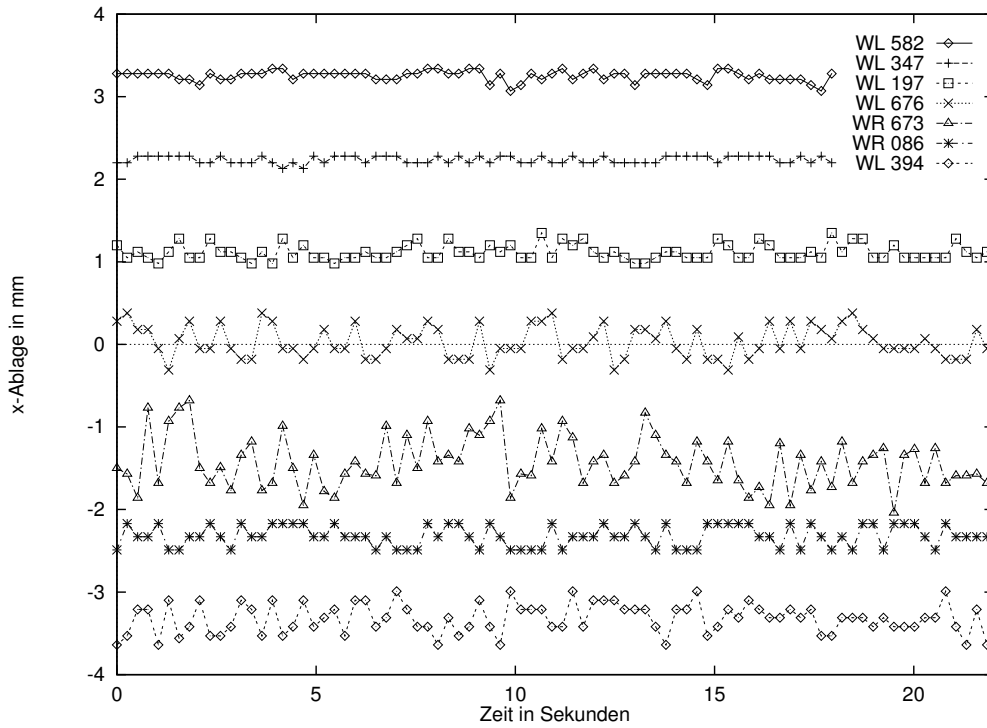
Average transmission speed (2 SEDAC lines):

3777 ± 216 Hz

Orbit Mode Test

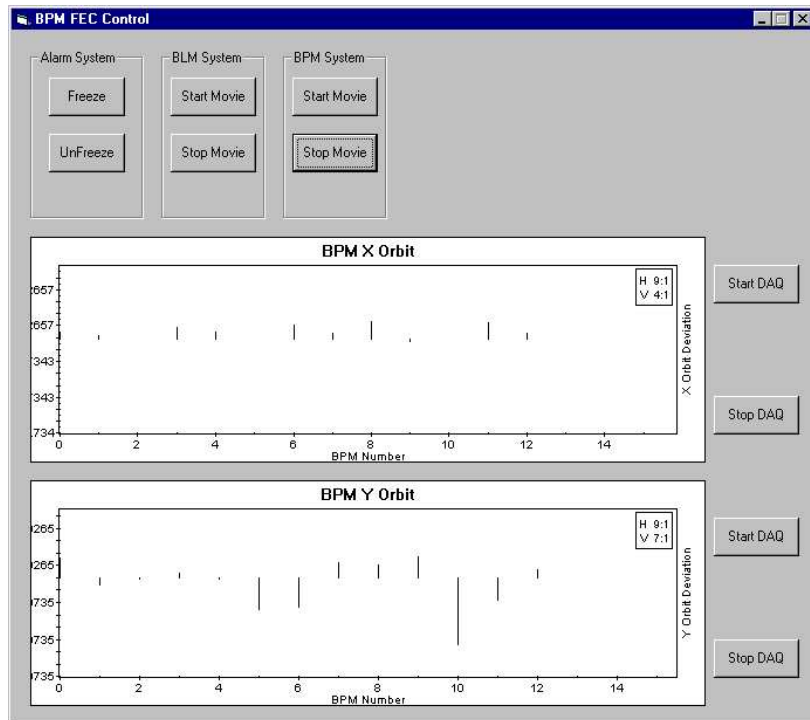
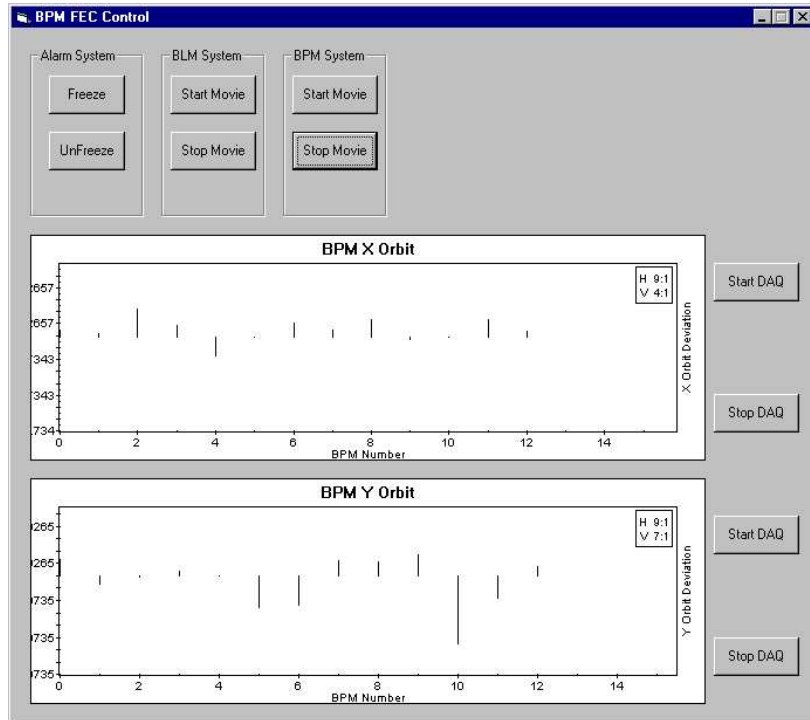


Time variation

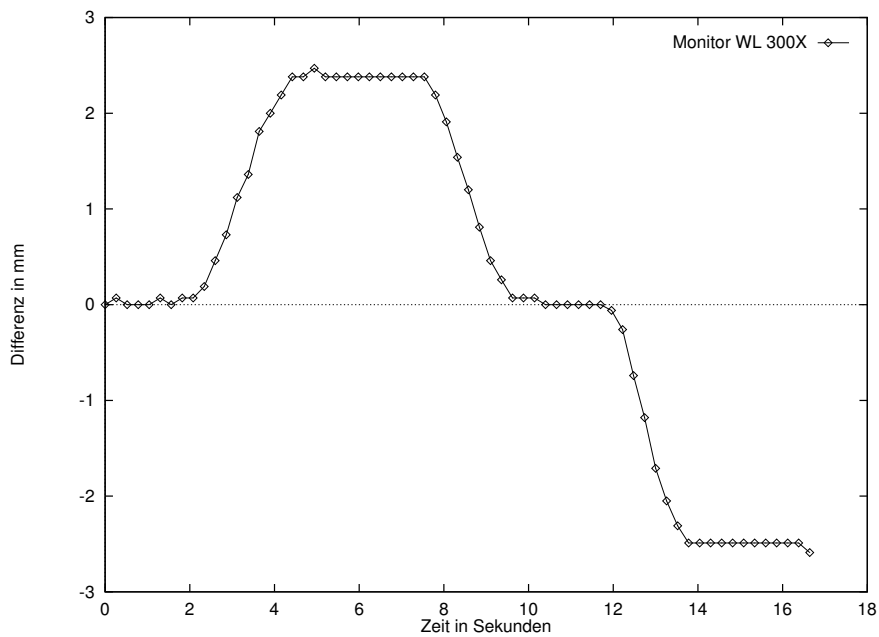
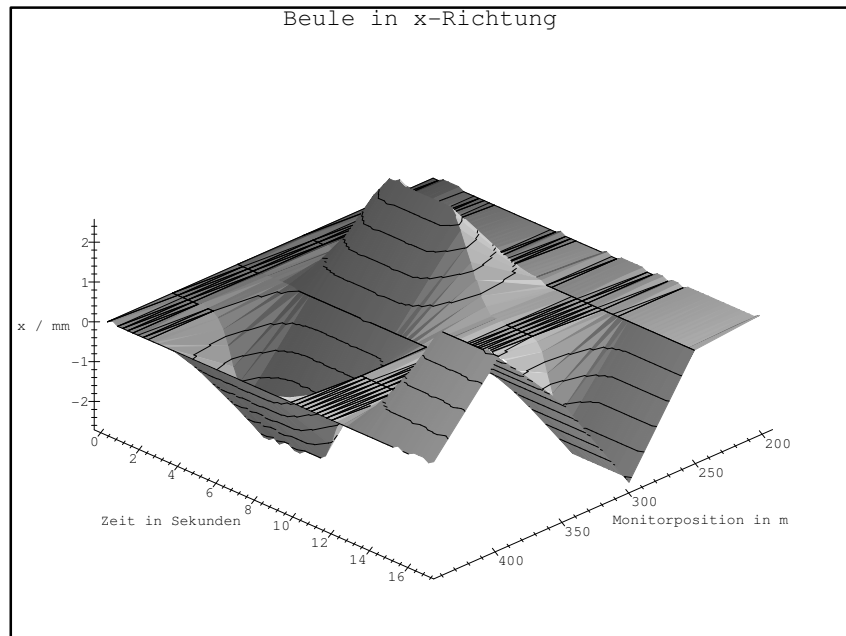


- Movie feature used

Local Orbit bump (x) I

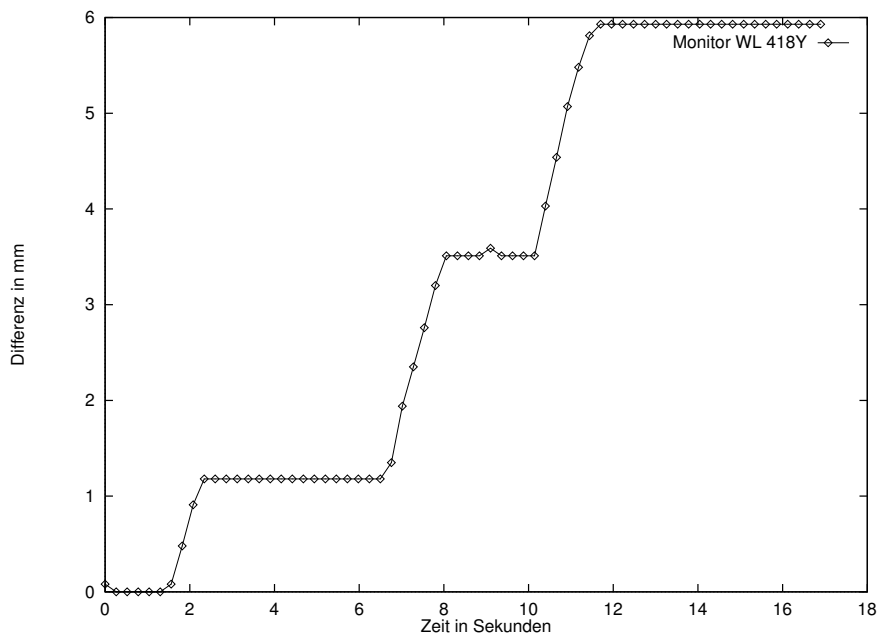
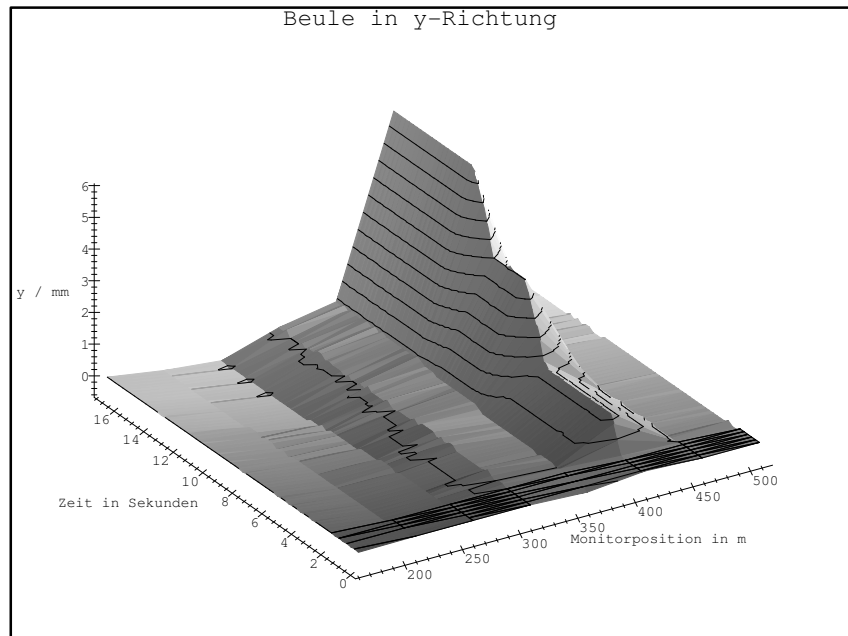


Local Orbit bump (x) II



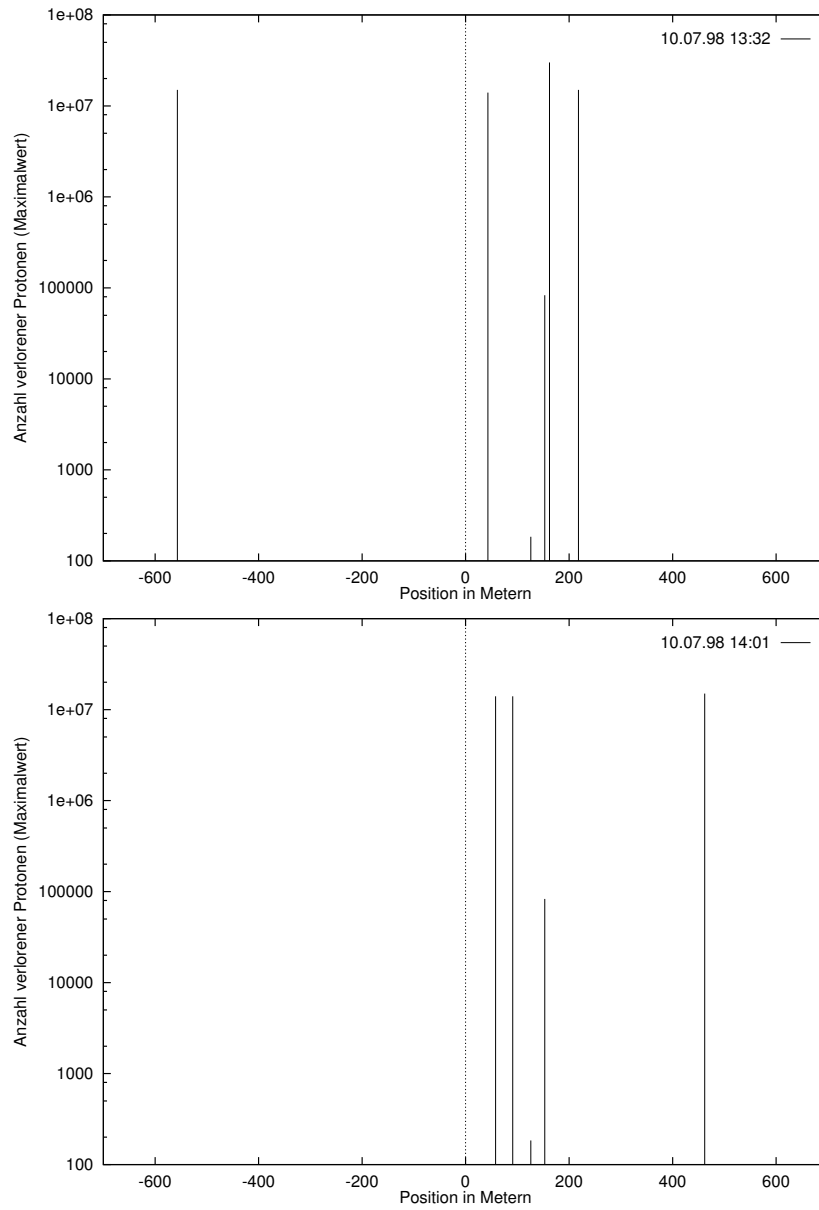
- Movie feature used

Local Orbit bump (y)



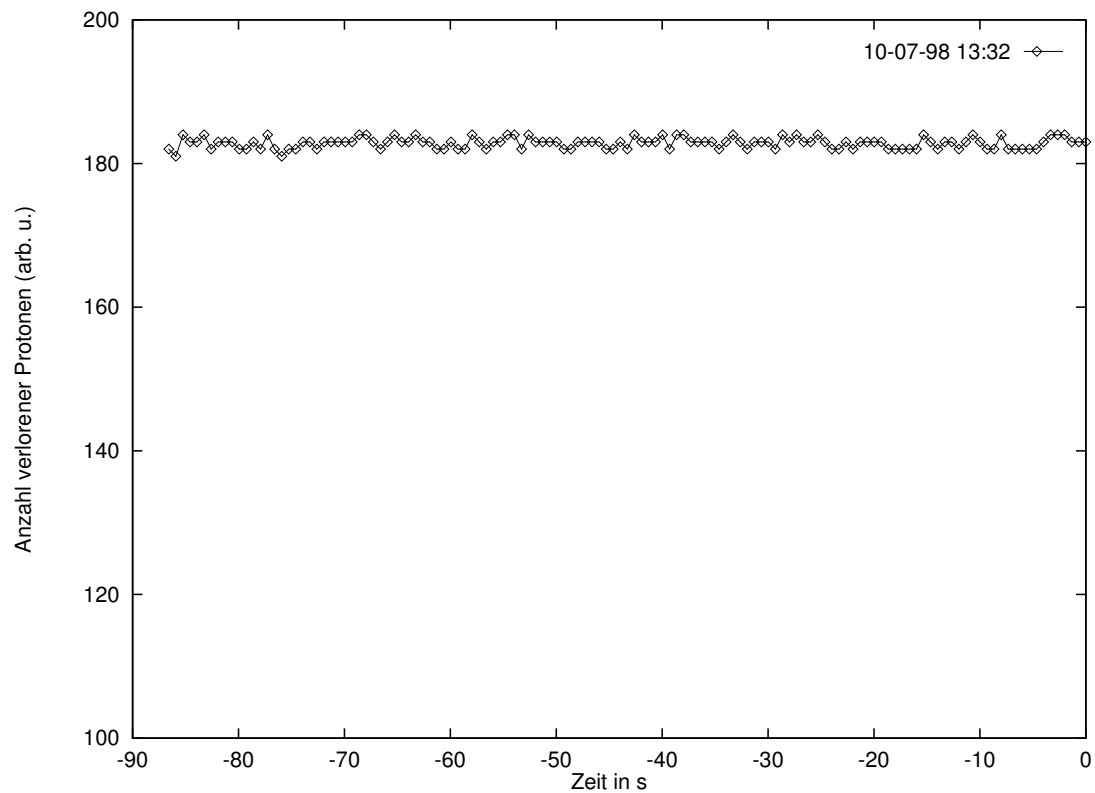
- Movie feature used

BLM Test Result



- very few data at 39 GeV
- no statistical significance
- archive feature used

BLM Test Result II



- CERN loss monitor
- archive feature used

ALM Test Result

```
ALM: DAQ Task run every 361 milliseconds...
ALM: DAQ Task run every 361 milliseconds...
ALM: FREEZE detected with 14 Alarms
BLM: DAQTask waiting for unfreeze.
BLM: DAQTask waiting for unfreeze.
Reading timeout between 4 and 4 ms.
PPD: Polling PPD is frozen.
PPD: Polling PPD is frozen.
ALM: DAQ Task run every 28 milliseconds...
Reading timeout between 0 and 1 ms.
PPD: Polling PPD is frozen.
ALM: DAQ Task run every 31 milliseconds...
ALM: DAQ Task run every 4 milliseconds...
Reading timeout between 0 and 1 ms.
PPD: Polling PPD is frozen.
Reading timeout between 0 and 1 ms.
PPD: Polling PPD is frozen.
ALM: DAQ Task run every 3 milliseconds...
PPD: Polling PPD is frozen.
ALM: DAQ Task run every 7 milliseconds...
ALM: DAQ Task run every 4 milliseconds...
```

- ALM signals FREEZE
- BLM and PPD stop DAQ and wait for UNFREEZE
- Speed boost of ALM and PPD

Conclusion

$$\Delta t_{ALM} = 411 \pm 19 \text{ ms}$$

$$\Delta t_{BLM} = 1500 \pm 300 \text{ ms}$$

$$\Delta t_{PPD} = 260 \pm 2 \text{ ms}$$

$$\nu_{avg} = 3777 \pm 216 \text{ Hz}$$

- ALM and BLM DAQ works well
- BPM Orbit Mode ok, others untested
- long term stability (two crashes during test)
- Missing: Four server interface