SHiP DAQ demonstrator Status

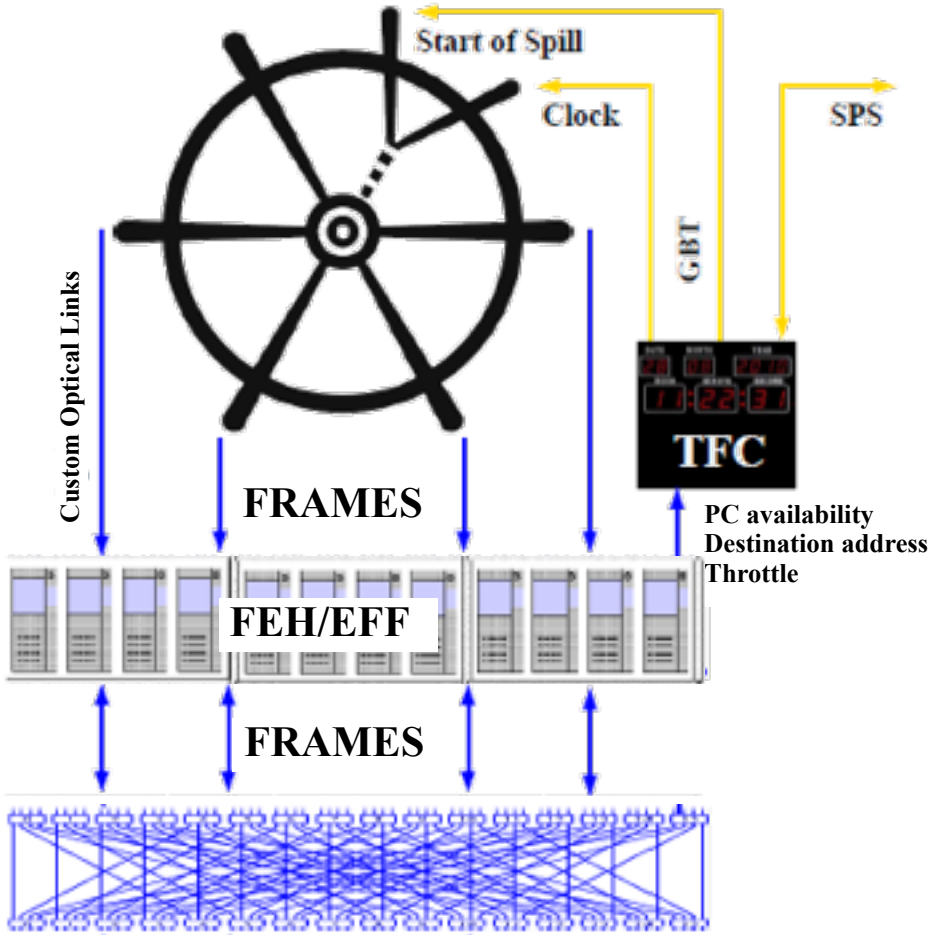
Richard Brenner Uppsala U
Mario Campanelli UCL
Hans Dijkstra CERN
Michael Jonker CERN/TE
Erik Van Herwijnen CERN
Stefania Xella Copenhagen U

System architecture:

Original TP:
- Direct Ethernet links FE to EFF switch
- Transfer by Multi Event Packages

Updated TP:
- Allow custom Optical links (interfaced to FEH)
- Processors have dual role FEH/EFF
- Transfer data in Time-Frames (FE sends a “Frame”, data from a FE-board covering 1.6 μs)

Further modifications & options :
- EFF data also as Time-Frames



SHiP DAQ demonstrator objective:

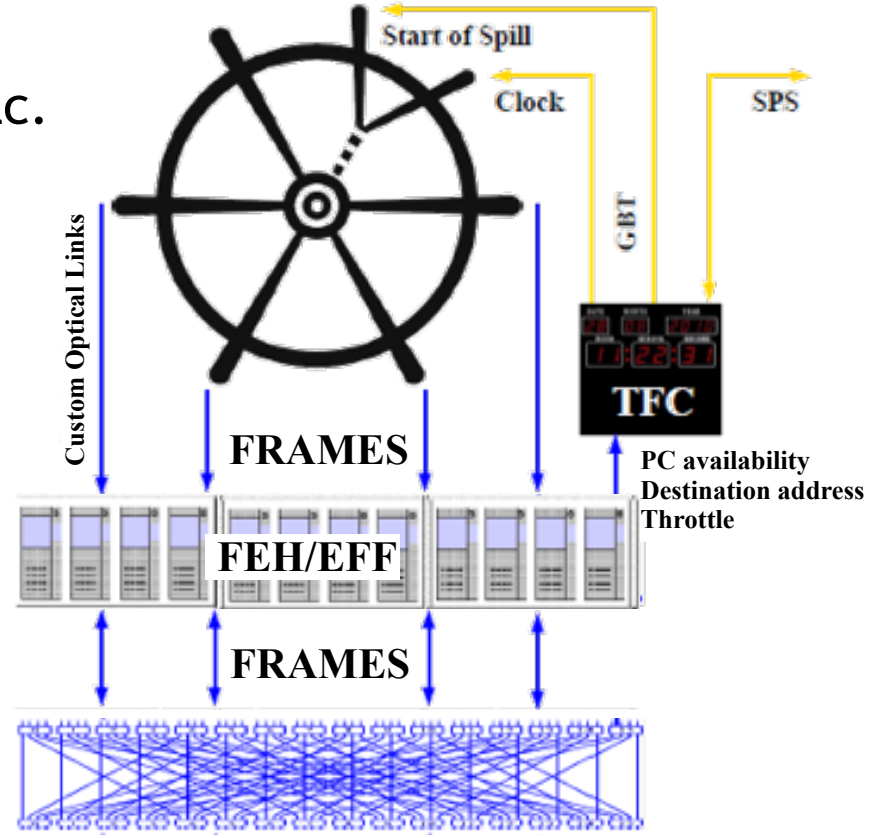
Create a scaled-down version of the SHiP DAQ system to

- demonstrate the feasibility
- study performance bottlenecks
- provide implementation details on data format, flow control, etc.

Components

- Frond End (FE) electronic data producer cards
- TFC (timing controller)
- Front End Host process (FEH)
- GBT: Giga Bit Transceiver optical link
- Event Filter Farm (EFF) process with network switch.

  NB: FEH and EFF may share the same hardware



FEH/EFF

for the first time:
going into details, start implementation

Work mainly from Michael Jonker CERN/TE, Hans Dijkstra CERN

Event Filter Farm process

Key component of EFF: receives data frames from the FEH

- Store data frames of one spill in a spill-buffer.
  - Check for data consistency, missing frames, etc.
  - Save data to temporary local disk if needed (e.g. high EFF occupancy, absence of calibration data)
- Sort hit data into event candidates, build (FAIR)ROOT data structures
  - Subject event candidates to Software Trigger for data selection
  - Write selected events to output data stream

Frame data organisation

Subdetectors can be divided in independent partitions

Independent data frame stream per partition

Frame packing either by FE electronics, or by FE host computer (FEH).

Data frames from a partition form a continuous uninterrupted data stream.

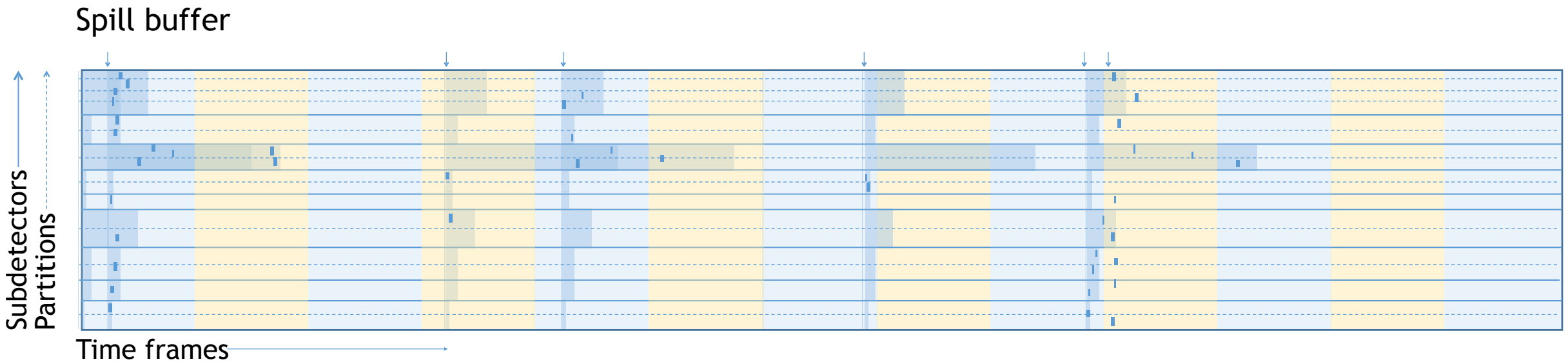
→ Many empty frames & network packets

Flexible Data Frames

Cycle: total cycle of the SPS, lasts ~ 7 s.
Spill: part of Cycle with beam on SHiP target, ~ 1 s.

frame header format common to all detectors. Contains start and time duration. Either comes like this from FE or is formatted so on FEH

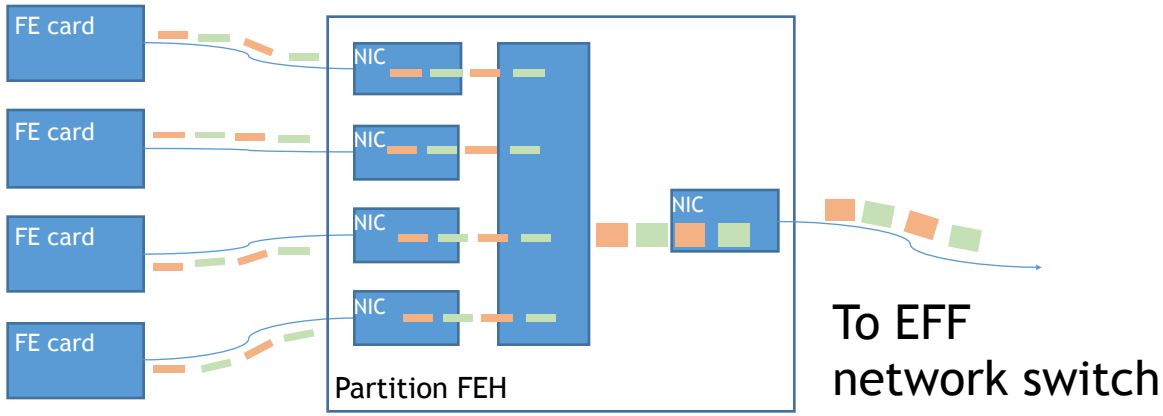
Frame-size (2B)	Partition ID (2B)	Spill Number (4B)	Start time (4B)	Time extend (2B)	Flags (2B)



Front End Host:

- Collect data from Front End cards
- Prepare data (if not already done) in data frames
  In practise this may involve unpacking/sorting/repacking of data
- Communicate with the ‘elected’ EFF computer that handles the data for a given spill.
- Local multi-spill buffer handling (if required).
- Prototype to be developed in collaboration with FE developers.

- Optional task for FEH computer:
  - perform local data quality monitoring
  - participate in EFF processing



Current Status:

- EFF process (spill handler V0.0) is available

EFF process functional list:
✓Frame management
- Frame reception
- ❖ UDP acknowledge
- Consistency checks
- Ordering of frames
- Checking of missing frames

• Sliding event mask over data frames
- Locate next events based on smallest hit time.
- Hit extraction based on partition specific time window
- ❖ Handling of overlaid events
- ❖ Event building (need agreement on data format)

❖Event filter framework.
- ❖ Physics event filters
- ❖ Output selected events on output stream (ROOT?)

✓Data provided by a single **FEH emulator**.
- Simple no handshake UDP data source
- Hit and Frame creation
- Occasional shuffling of frame order

Other items:
✓Partition configuration definition
❖Communication handshake
❖Interaction with EFF manager

❖Investigate raw Ethernet communication

EFF process implementation V0:
800 lines of C++ code *).
no external libraries/templates.

3 major functional classes
- DataStreamReader
- PartitionData

- DataStreamManager

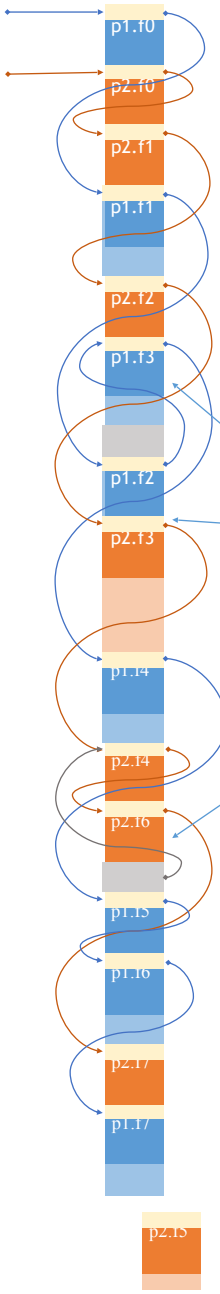
2 data container classes
- DataFrame

- Hit

A few linkage-, iterator-, and configuration classes, main()

FEH emulation process:
190 lines of C++ code *)

*) nb: XXL line length



SpillBuffer DataFrame organisation
shown for two partitions

Partition 1
Offset to following frame
Frame header
Frame data
Partition 2
Frame discontinuity marker (part of linked list not shown here)

Delayed reception of event frame 2, partition 1.

Delayed reception of event frame 5, partition 2.

Frames are aligned to int32_t addresses by padding the frame data where needed. (As hits are int16_t data items, it may happen that a frame does not end at an int32_t boundary).

Offsets in the Spill buffer are stored in int32_t locations and expressed as int32_t item offsets. Hence maximum spill buffer span is 8 G byte.

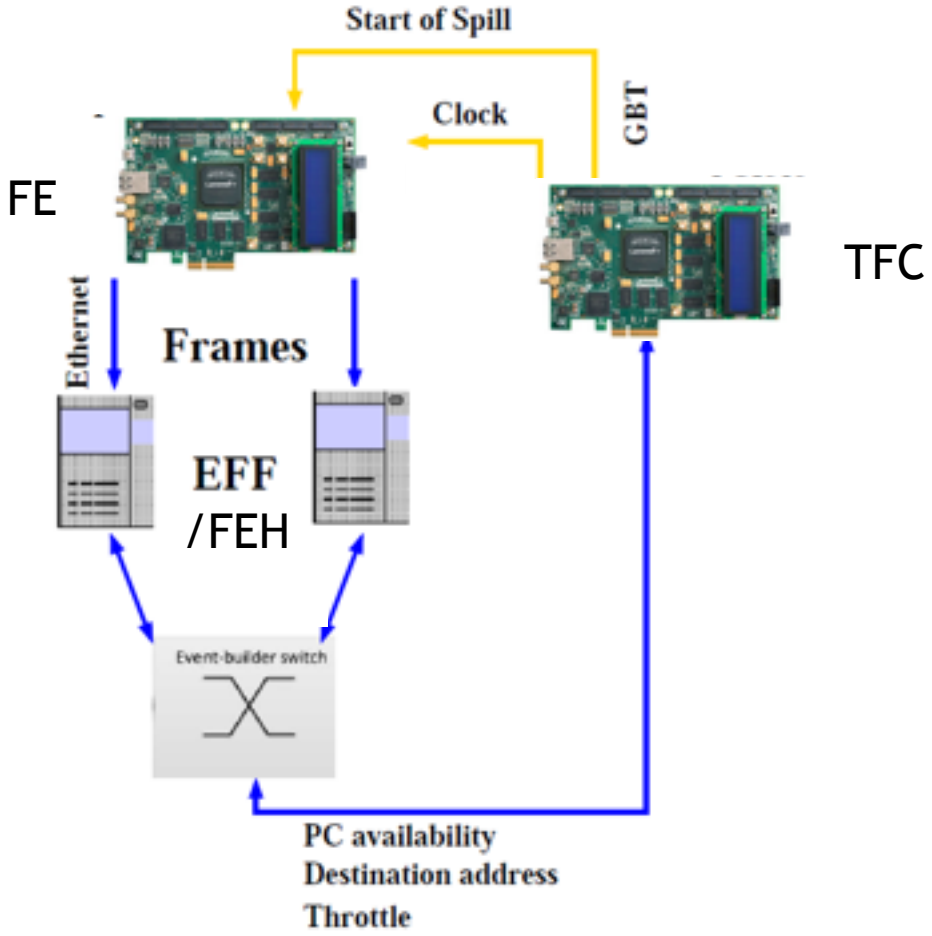
Use of absolute addresses can be considered, but in this case an int64_t object is needed and data frames should be aligned to int64_t addresses. Moreover, the use of offsets (which are position independent data references) allow concurrent processing of the incoming spill data through a shared data segment.

TDAQ demonstrator

Work mainly from Richard Brenner's group, Uppsala U

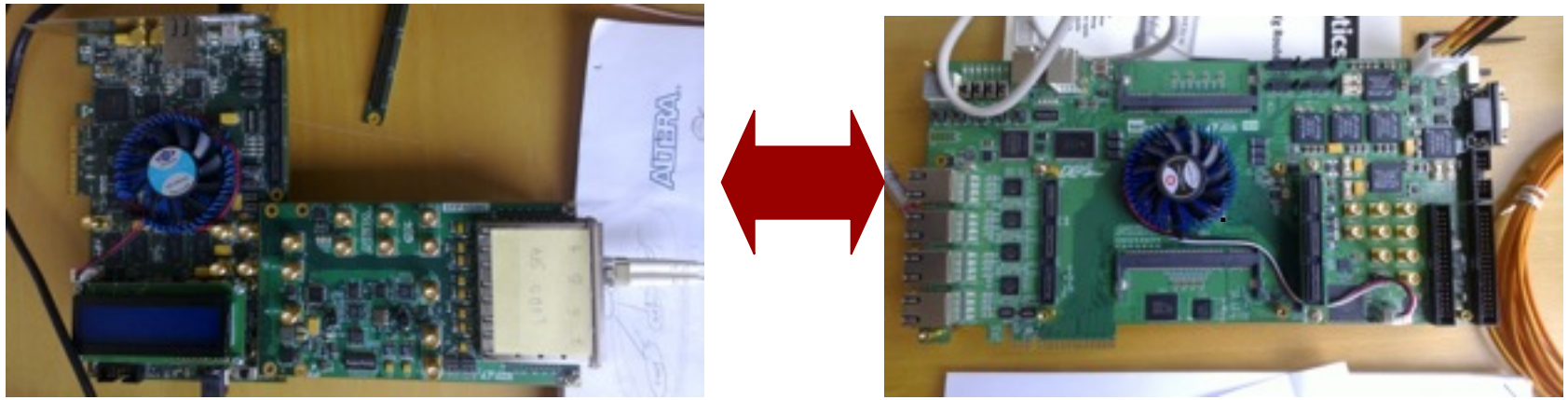
Demonstrator for TDAQ system

- Demonstrate distribution of timing and control with GBT.
- Exercise transmission of time stamped FE data on Ethernet to EFF
- No timing from bunch structure in SHIP requires good timing on-detector.
- Use mid-range FPGA hardware to build an affordable system.



Status of demonstrator

- Work on SHIP TDAQ demonstrator ongoing in Uppsala
- Two master students (Filos Vasilelos and Jiheng Chen) just finished project on communication between two FPGAs (TFC ↔ FE) using Transceiver Toolkit combined in the Quartus II software.
- System built using two different ALTERA FPGA families Cyclone V GT and Stratix IV GX FPGAs from Terasic purchased by Copenhagen and Uppsala for the project
- Evaluation of communication using ETH and GBT protocols completed



Summary & Next steps

Work on SHiP DAQ demonstrator has advanced a lot since last collaboration meeting.

- we have a system (2 FPGA boards, 2 PC and optical links) to test things
- we have advanced in the ideas for FE data format and FEH/EFF desired functionality

Plan is to use the TDAQ demonstrator in Uppsala and TDAQ simulation to go further into details and test system.

- A detailed note is in preparation: "*Readout Control Specifications for the Front-End and Back-End of SHiP*". Aiming for first EFF/FEH in a networked environment in the course of 2017.

- start collecting input/guidelines needed on physics analysis code to use, and on event data format for analysis and start a discussion needed for a coherent proposals for raw data formats.
- skeleton program to emulate the DAQ exists, written by Kris Korcyl. Eric van Herwijnen expressed interest to pursue this further, once we have a reasonable understanding of the demonstrator. We can then study the scalability to bigger system.