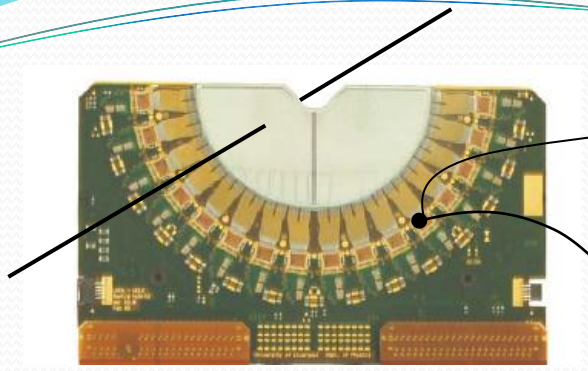
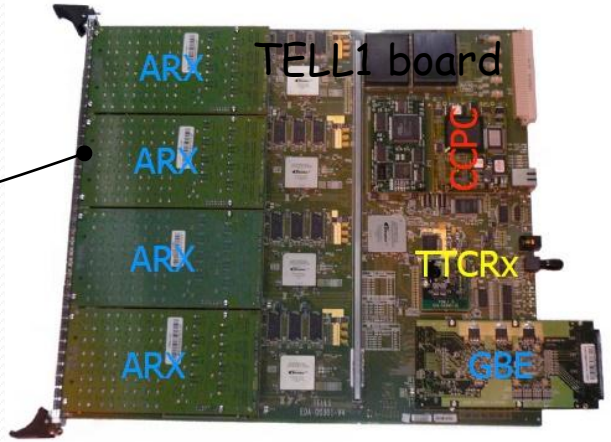
On-chip zero suppression – processing algorithms

Outline

- How we do this now (VELO & ST)
- On-chip zero suppression procedure for the 40 MHz read-out



VELO front-end



Common electronic acquisition read-out board – **TELL1**

Digital/analogue input (interfaced to gigabit Ethernet)

FPGA based – Altera Stratix

Main goal – synchronisation, buffering and the data zero suppression (factor ~ 200)

From non-zero suppressed (2048 /sensor)

To zero suppressed (clusters only)

Technically it is a farm of parallel stream processors

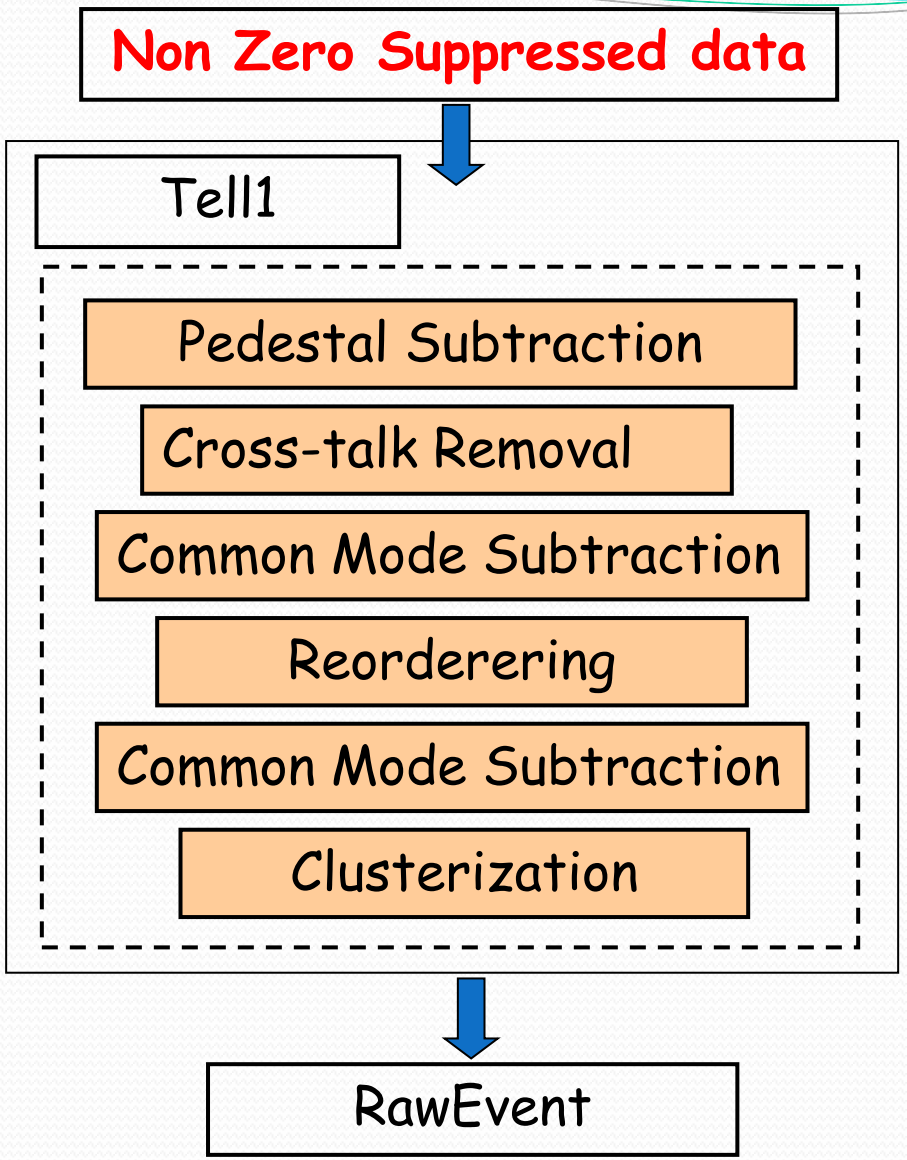
Processing 36 threads at the same time

↓

Raw cluster buffer

↓

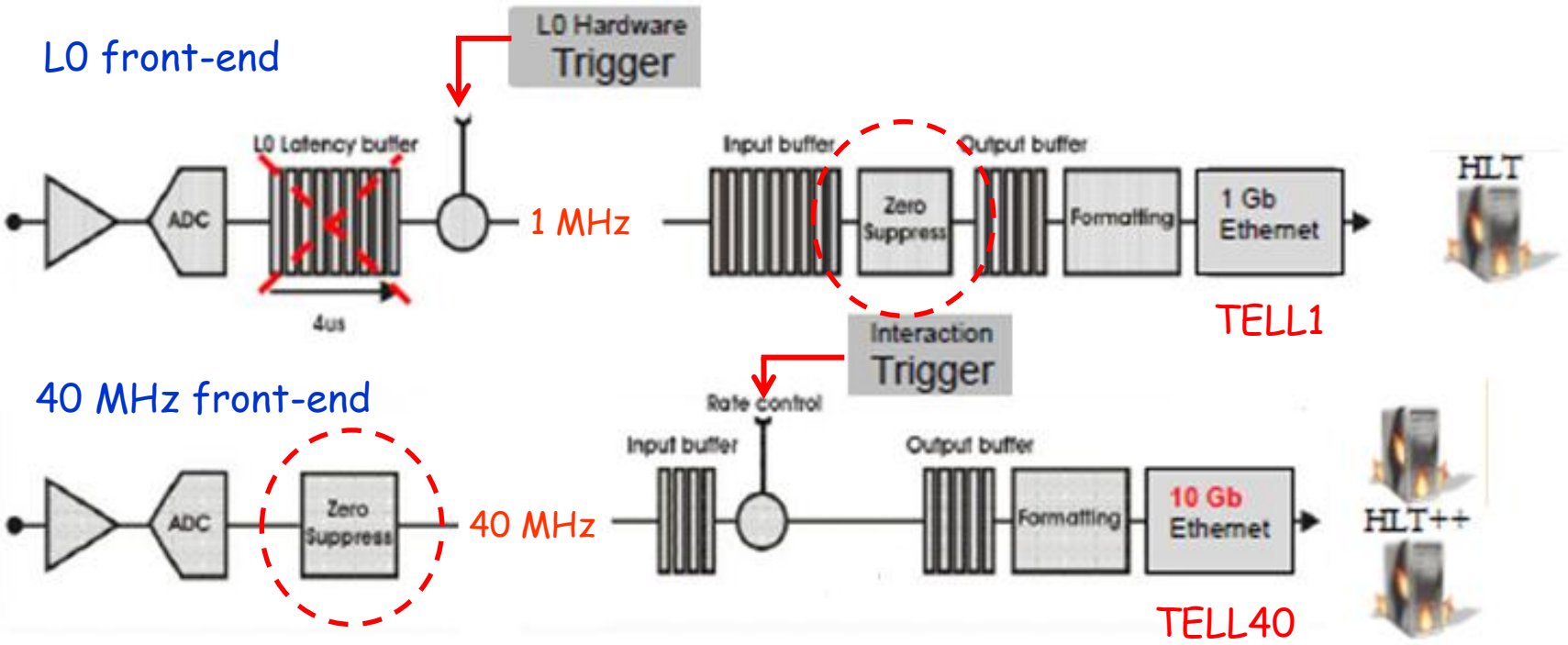
Reconstruction


Non Zero Suppressed data
Tell1
Pedestal Subtraction
Cross-talk Removal
Common Mode Subtraction
Reorderering
Common Mode Subtraction
Clusterization
RawEvent

- not a trivial task
- each processing channel needs to be tuned separately using calibration data (e.g. for the VELO we need ~ 1 M parameters)
- processing runs in real time using hundreds of threads
- huge amount of data

  88 sensors x 2048 channels x 10 bits x 1 MHz

Zero suppression in the new read-out scheme



New front-end chip for the pixel/strip option
New electronic acquisition board – TELL40

Base-line zero suppression procedure has been defined in our internal 'specs' document

- digital processing (6-bit samples)
- pedestal subtraction/following
- CM correction (constant/linear)
- Clusterisation

The actual list will depend on the power consumption

Can we estimate now how much power do we need? (Critical)

Do we need to perform dedicated studies?

If any problems what is the absolute minimum?

Pedestal subtraction/following - present

- this one was chosen as optimal for the VELO/ST
- actually we do not run it!
- pedestals are taken form the Tell1 memory
- fixed in off-line emulation
- need to use raw data for calibration
- dead/noisy channels masked before pedestal subtraction
- our experience is that pedestals are rather stable (~weeks)

Pedestal subtraction - NEW

- in principle we can have the same running average approach
- in my opinion – we should also run subtraction without following
- we need enough memory to upload parameters (pedestals)
- the same goes for the dead/noisy strips
- need raw data for calibration
- optimal use of the dynamic range

Common Mode correction – present (complicated situation…)

- done in two stages (the VELO design)
- first performed in electronic channel domain - MCMS
- second in strip channel domain - LCMS (this is actually inactive)
- both perform superbly with minimum bias events
- initial design not robust against busy events
- eventually they were corrected
- still a lot of trouble with large charge deposit (base line shift)

Common Mode correction - NEW

- no reordering: chip channel -> strip channel, so, one CM correction only
- how complicated do we want to be?
- the base line is LCMS (our current experience is that CM is small – but could get worse with radiation damage)
- we need to make it robust against busy events from the start
- detailed simulation can help (how many hits per chip)
- smart hit rejection

Clusterisation - present

- very complicated for VELO (R/Phi specific)
- using two sets of thresholds
- seeding thresholds – hit detection
- inclusion thresholds – to construct a cluster
- up to 4 strips per cluster
- cluster centre estimate (3 bit resolution)
- charge deposit on each strip contributing to a cluster available for off-line
- transport protocol optimised for on-line track reconstruction
- has some basic intelligence regarding cluster shape detection
- boundaries problem (mainly VELO)

Clusterisation - NEW

- no reordering for VELO and TT (should be generic for both)
- hit detection done the same way
- do we need inclusion threshold per channel?
- we proposed to have up to 5 strips per cluster
- critical – define the transport protocol
- ADC measured for each contributing strip available off-line
- optimised for Tell40 pre-processing
- re-clusterisation over the boundaries
- hit centre estimated on the chip or within Tell40?

Spillover

- shaping – see Jan's talk
- we will run with 25 ns scheme
- need to be aware that this will produce visible effects
- the base-line idea is to store information on channels that were hit (together with measured charge) for BX „i" and subtract from charge measured in consecutive BX „i+1"

How do we go abut it?

- having the current suite of algorithms as a „step 0" seems reasonable
- produce C „engines" first and then start implementing in VHDL
- in parallel prepare LHCb specific code for emulation using the engines
- the „big issue" here is that we need to implement the processing on the chip – once done it's done (past the PSR)
- hardware emulation using FPGA processors
- need dedicated manpower for each subdetector
- common VHDL/C/C++ project

Spares

New electronic acquisition board TELL40

- 24 GBT serial input streams
- FPGA based (Altera Stratix V/VI)
- Input data
  - Data packets identification (output data format optimisation helps!)
  - Time ordering for VeloPix (packets come in random order)
  - Decoding super pixels to individual pixels (ToT and address)
- Processing
  - Zero-suppression moved to front-end
  - Any complex algorithm will be resource intensive (40 MHz)
  - Clusterization for pixels
  - Interpolated centre position for strips
  - Hits spatial ordering (for patter recognition)
- Output (not VELO specific)
  - Event building and formatting
  - Storage and filtering
  - Ethernet frames building

Considerable work has been done for the pixels – 90% of the firmware is ready
Some of the code should be reusable for strips