UNIVERSITÄT
HEIDELBERG

Belle 2 PXD DCS

Schaltungstechnik
und Simulation

Michael Ritzert

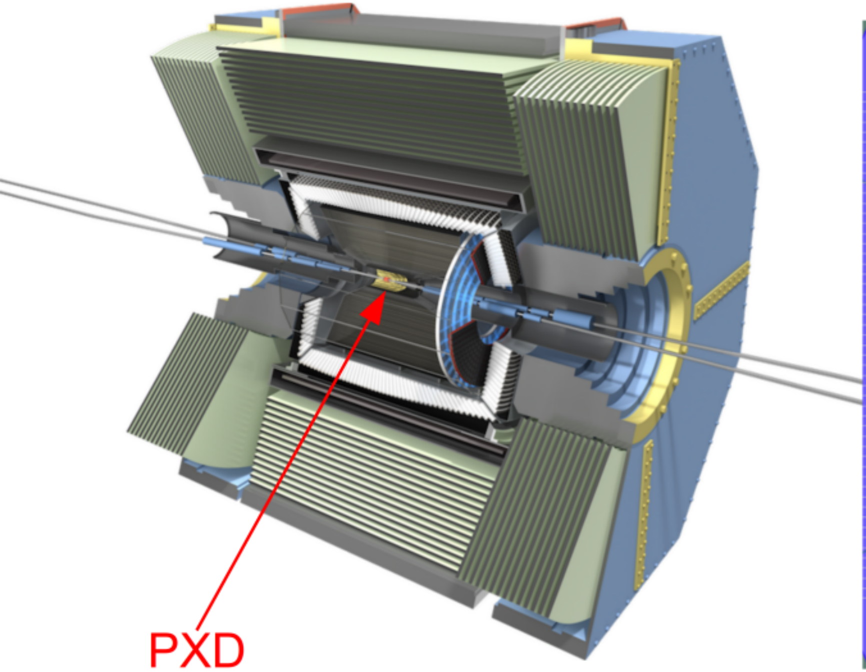
michael.ritzert@ziti.uni-heidelberg.de

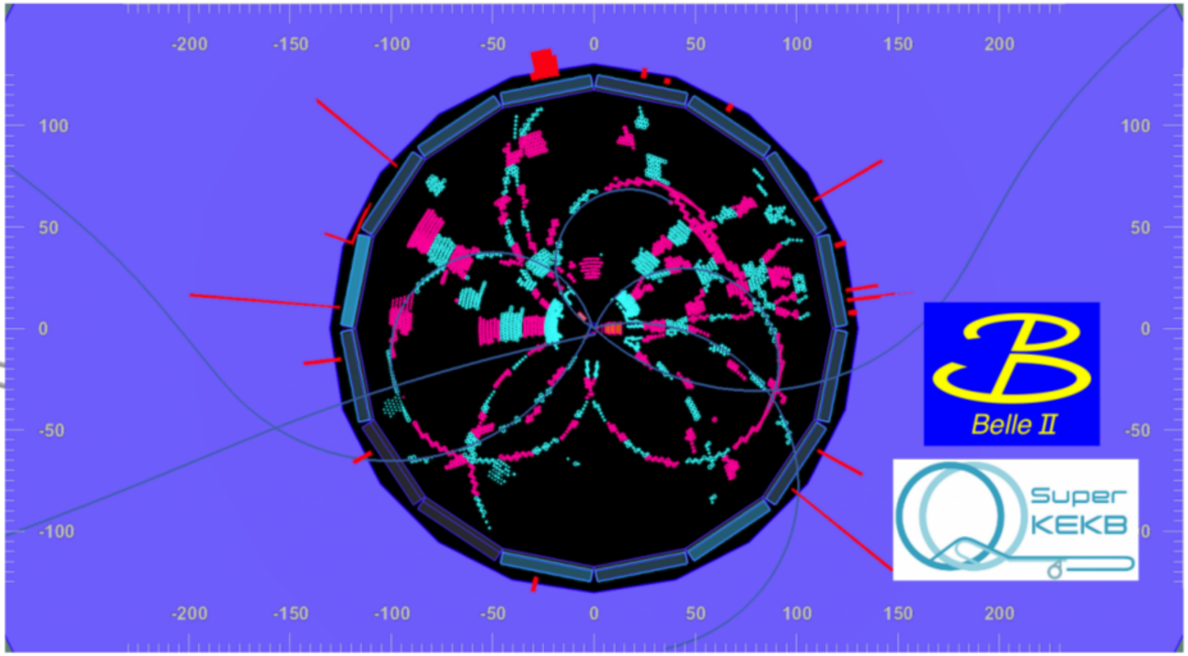
Joint CBM / PANDA Topical Workshop
on Detector Control Systems Aspects

Darmstadt

20.06.2018

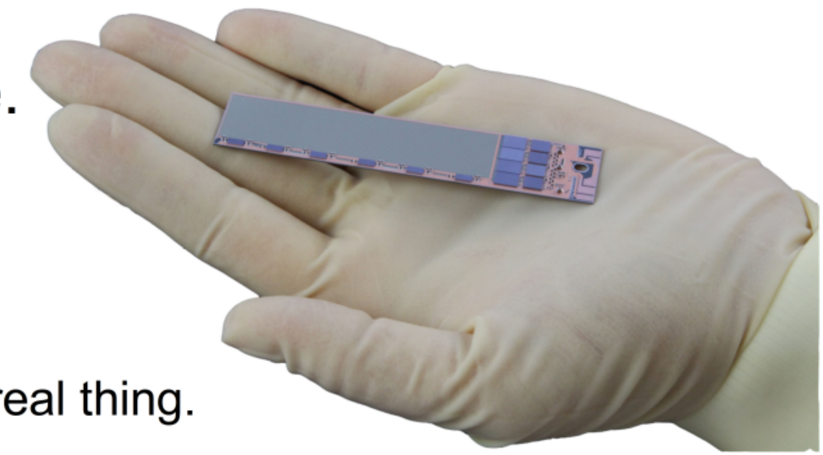
- Sole experiment at the SuperKEKB B-factory in Tsukuba, Japan.
- Target: $8\times10^{35}/cm^2/s$ instantaneous luminosity, $50ab^{-1}$ integrated.
- First collisions on April 25th. Currently commissioning @ $2.5\times10^{33}/cm^2/s$.





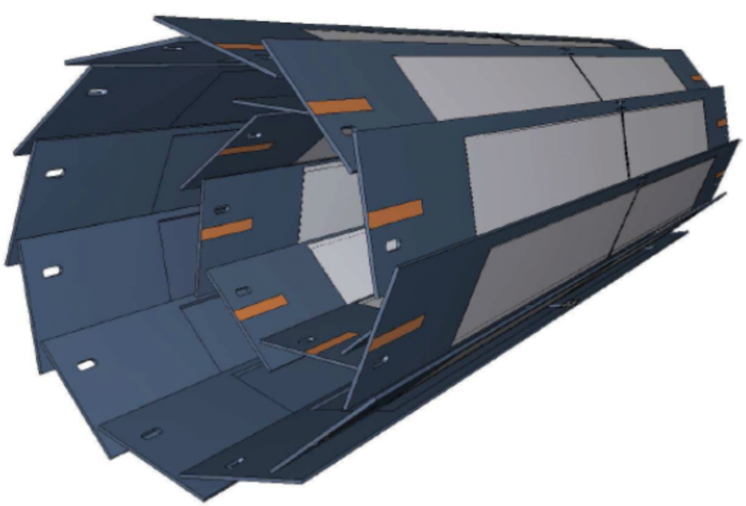
Belle 2 Pixel Detector (PXD)

- German contribution: PXD detector based on DEPFET sensors.
- Two layers @ r=1.4 cm, r=2.2 cm.
- 40 modules, 16M pixels.
  - Four modules during SuperKEKB commissioning phase.
- Thinned to 75 µm in the active area.
- 50 kHz frame rate, ~ 1 GB/s to storage.
  ⇒ ~⅓ of all of Belle 2's data.



The real thing.

„Artist's illustration"



Many Protocols

- The Challenge: Lots of devices, each communicating in a different way.

An incomplete list:

Hardware	Protocol
Cooling Plant	modbus
Power Supply Units	custom protocol
„DHH“, „DATCON“ DAQ systems	IPbus (CERN)
ATCA crates	IPMI
fibre optical sensors	TCP-based custom protocol
„ONSEN“ DAQ system	EPICS on FPGA PowerPC
Belle 2	NSM2 (KEK)
temperature sensors	i²c
primary power supplies	GPIB-over-TCP

And the number is still growing…

- The unifying standard: **EPICS** as the common protocol.

Overall Organization

- Well-defined SC installations, based on Scientific Linux.
- Centralized, automatic, build system for RPMs for EPICS base, modules + our own IOCs.
- Centralized build of a CSS (an EPICS GUI) version preconfigured with PXD settings.
- IOC (~ device driver) for „finished“ hardware — mostly off-the-shelf products — done by a small group of core developers.
- IOC development for own hardware by the hardware developers.
  - Provide a basic IOC with all required protocols implemented.
  - But have them fill in and maintain all the hardware details.
    ⇒ Faster turnaround. Adding a register be done within one group.
    ⇒ Easier for them to develop with an always-up-to-date SC system.
- More or less „free“ development within a (wide) set of guidelines, e.g. GUI design, use of the EPICS PV namespace.

- Two Belle 2-wide systems to integrate the PXD SC into:

Power Supply Control

Sensor Power

Power Off

⇅

LV On, Injection Allowed

⇅

Sensors Active, Injection Prohibited

+ error states

Run Control

DAQ Operation

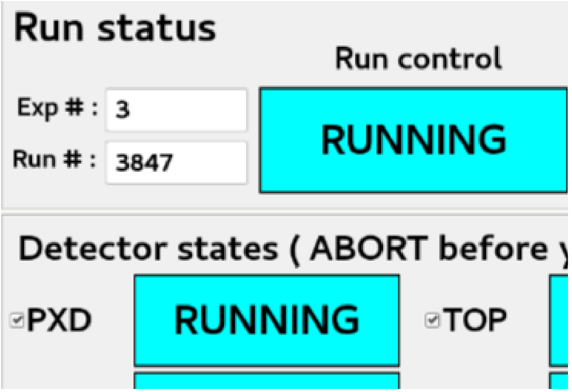
Off

⇅

Standby for Run

⇅

Running



- Higher radiation levels during beam injection.
  ⇒ Protect the detectors by switching off HV during injection.
- Some discussions were required to fit the Belle operations scheme to PXD, esp. for the Power Supply Control.
  - Most other subdetectors have some sort of HV voltage.
  - PXD: Complex sequence (O(100) steps) to power and configure the ASICs on the modules, and the modules themselves.
    ⇒ Digital configuration in step 1, module power in step 2.

Detector Control Implementation

- Several levels in the hierarchies:
  - Belle 2
  - Subdectors: PXD, SVD, CDC, …, Trigger (RC), Storage (RC), …
  - @PXD: individual modules (PSC) / DAQ compomenents (RC)
- On each level, modules from the lower level can be excluded.

  ⇒ Local control of these excluded components.
- @PXD: Implemented using the EPICS seq module.
  - One IOC per level in the hierarchy.
  - NSM2-to-EPICS bridge for connection to Belle 2 level.
- Typical problems encountered during development:
  - Recovery procedure from error states.
    ⇒ Discussion with Belle 2 SC responsibles.
  - Hardware interfaces to other systems controlled outside PXD SC.
    ⇒ Make no assumptions about anything outside your control.
    ⇒ Provide clear information, which side is currently failing.
  - Undetected changes in the actual hardware status.
    ⇒ best practice: WHEN_ERROR macro in each state.

```
#define WHEN_ERROR \
  when (PSoff) state ERROR; \
  when (detector_burning) state FIRE;
```

So That Was the Plan…

What could have gone better?

And what Lessons have been learned?

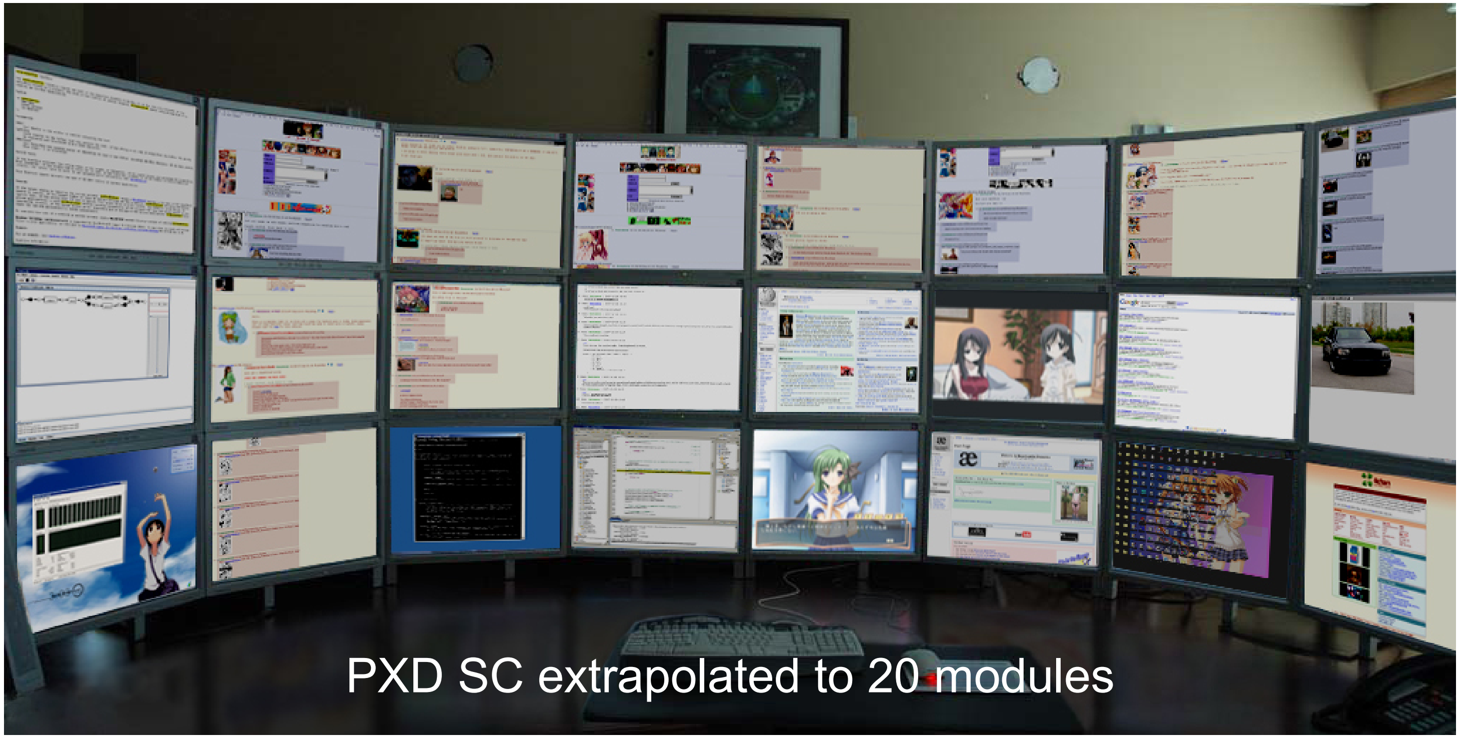
- There won't be any recommendations you are not aware of.
- But I'll try to add a new aspect to some of them.

Off-the-Shelf Devices Integrate Easier

- Of course, it's not always possible to buy standard hardware.
- But my experience is, standard devices use standard, „easy“ protocols, own developments often don't.
- Looking at the IOCs (device drivers) we are running, the most complex ones are the ones for PXD-developed hardware.
  - Partly because it's highly specialized hardware with special requirements.
  - But also partly because little-used protocols are implemented. E.g. IPbus (no EPICS driver) vs. Modbus (good EPICS driver).
- ⇒ Try to follow standards where applicable.
- This may require compromises on the device side, but I'd say a good case has to be presented to use anything else.

Old Habits Die Hard

- Change is only accepted slowly.
- „OK, we can do it like this for now, but we must be aware that this won't scale for the final system.“
  - A very bad sentence to say.
  - People will go to big lengths to *make* it scale, no matter how inconvenient.



Even when you do finally offer a better solution.

- ⇒ Insist on a scalable solution from day 1.

Documentation!

Documentation is **always** a good thing.

- But it's not a one-time thing.
- Make sure it is up to date.
  - Outdated, but still „similar“ documentation can very easily mislead the reader.

One more very important aspect to it:

- Key personel can move on at any time, often on very short notice. I heard “Next month/week(!) I'll be gone“ two times.
- When the day comes
  - Option a – No / outdated documentation: Hastily arrange a handover meeting.
  - Option b – Up to date documentation: Use the time to throw a goodbye party.

A System is Never Finished

- Any „complete“ list (requirements, hardware, …) will always be amended…
- E.g. even after installation, (small) new devices will creep in.
  - Latest example: A remote-controlled main power switch for hardware in the radiation exclusion zone.
  - They need network connectivity, aka. a switch port.
    ⇒ Always have some spares!
  - They need slow-control integration.
    ⇒ Be prepared to invent something on short notice.
- Side-aspect: You gotta love ~~pie~~ the Raspberry Pi.
  - Need a „PC“ to control a JTAG programmer? ⇒ use a Pi.
  - Need a „PC“ to do RS232 serial? ⇒ use a Pi.
  - i²c? ⇒ use a Pi…
  - **But:** Can you trust them? The quick fix is easy and cheap.
    But finding anything long-term can be really cumbersome.

Test Everything, or the True Story of a Firmware

- Task: Develop the µC firmware and PC control code (no EPICS at that time) for a power supply unit.
- 2011: Development outsourced.
- Since: Used during PS development.
  - Some small issues, but mostly OK.
- 2016: More than one unit in operation at the same time, in the same place.
  - Problems show up immediately.
  - Patched around in the PC-side code.
- 2017: It becomes clear that scalability is a major issue.
  - Detailed analysis of code and network traffic starts.

- They wouldn't… or would they…?

```
command_data_t *data =
    (command_data_t *)(xme_hal_sharedPtr_getPointer(dataHandle) + offset);
// Check if the command received is for the specific node or not.
if ((data->nodeNumber == node_Number) || (data->nodeNumber == 40)) {
  switch (data->commandType) {
  case START_UP_COMMAND_TYPE:
```

- Any command for one unit is sent (via TCP) to all units.
  ⇒ The traffic goes with the number of units **squared**.
  - A factor of 1600 for 40 units (one per PXD module).
  - The poor µC certainly can't handle that.
- 2018: All of the communication layer rewritten from scratch.
- At some sites, observed disconnections due to lost Ethernet frames.
  - Why didn't this happen before? Ah, it happened (a lot).
    The code just has very relaxed constraints.
  - What's the cause? Hardware issues? The jury is still out on that ATM.

Problem Analysis

So where did this go wrong?

- The coding was outsourced, but requirements were misunderstood.
  - „We need to be able to send a command to all units.“
    became
    „Send all command to all units.“
- The company produces a „Middleware for Cyber-Physical Systems“.
  - It (barely) fit the job, so they just used it.
  - But there's no knowledge about the framework outside that company. The one person somewhat knowledgeable in the project has long left (→ Documentation!).
- Hardware testing was one-unit-at-a-time.
  - The multi-unit issues remained undiscovered for quite a while.
- Possibly: „Fail-safe“ code hid hardware problems.
  - Failures happened all the time, but were automatically handled and never noticed.

Chain of Failures

⇒ A badly designed, overly complex system was developed.
- For comparison: Old code 230kB, new code 70kB binary size.

⇒ The code quality was never doubted. It works, so it must be OK!
- It worked when it was developed.
- An EPICS wrapper could be designed, so that also worked.
- Nobody expected problems in scalability.
  This is a middleware for distributed systems…

⇒ Problems were only discovered late.
- First signs were misinterpreted.
  After all they went away with a small patch.

⇒ Eventually, this problem significantly affected PXD operation until very recently.
- Only one out of four modules in Belle II could be ramped at a time.
- A shifter had to be present at all times. 24/7.
  Now: Normally **no night shift**, only on-call availability.

⇒ Originating in 2011, an undiscovered problem caused a lot of people a lot of headache in 2018.

Acknowledgments

This work has been sponsored by the German Federal Ministry of Education and Research (BMBF) under Grant Identifier 05H12VHH.

Federal Ministry of Education and Research

Thank you!