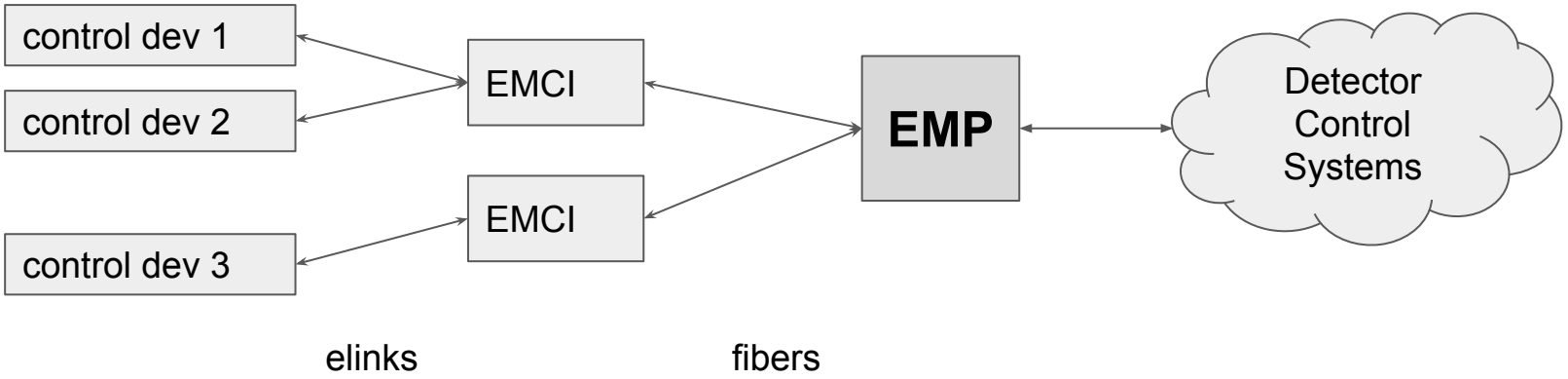
AirHdl and PuzzledLizardWizard for register map handling

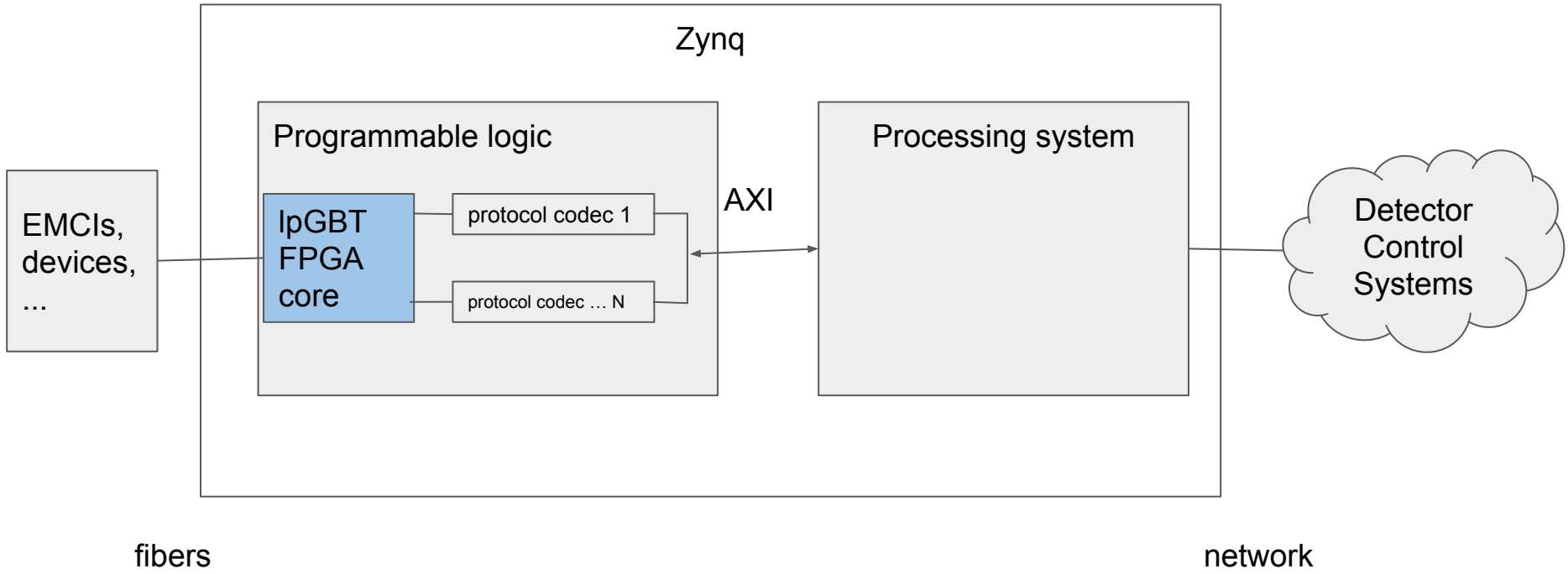
Piotr P. Nikiel for the EMP/EMCI team
(D. Blasco Serrano, M. Kristensen, P. Moschovakos, P. Nikiel, V.Ryjov, S.Schlenker)

EMP - Embedded Monitoring Processor

- Primary idea:
  - EMP: a device to integrate elink-based control & monitoring devices into detector control systems
  - EMP (current prototype) is a Zynq-based device



EMP architecture (simplified) - FPGA-SoC view



Common vs "uncommon" parts

- lpGBT core common, but individual elink access well exposed
- such individual elinks have different higher layer protocols thus individual codecs, e.g.
  - lpGBT IC channel elink,
  - Endeavour-encoded elink,
  - TileCal SIN sensor over elink,
  - ...
- these codecs are by different working groups - *challenge A*: not everyone can/wants to write/maintain AXI bus HDL (and ensure it is valid…!)
- these codecs need processing system software to be talked to and handled - *challenge B*: not everyone can/wants to write/maintain OS or user-level software to handle AXI-based components

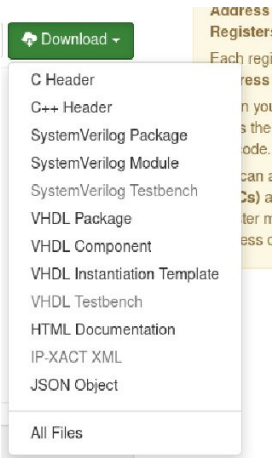
AirHdl (1) and solving challenge A

- AirHdl (https://airhdl.com) found to address challenge A
- it's a web-based tool to create AXI components based on register maps, e.g.:

Registers

Offset	Name	Description	Type	Access	Actions
0x0	magic	Magic identifier -- corresponds to empI	REG	READ_ONLY	
0x4	control	–	REG	WRITE_ONLY	
0x8	status	–	REG	READ_ONLY	
0xC	data_rx	–	REG	READ_ONLY	
0x10	data_tx	–	REG	READ_WRITE	
0x14	register_addr	The address of the register on the lpGBT of which we wish to read/write.	REG	READ_WRITE	
0x18	lpGBT_addr	The address of the lpGBT device you are interacting with.	REG	READ_WRITE	
↕ address gap: 228 bytes					
0x100	counter_lhc_clock	–	REG	READ_WRITE	

+ Register + Array + Memory Duplicate



- from that description, a number of artifacts can be generated.

AirHdl (2)

- basic mode: AirHdl generates input/output signals for all entered registers (of different types, access settings, bit-widths… etc.)
  - e.g. for the previous slide, we'd have: magic, control (with subfields), status (with subfields) std_logic_vectors
- prime advantage: you can easily create AXI4-compliant register-based components without much competence in AXI signalling
- identified trouble:
  - basic licensing not convenient for collaborations (but each collaboration can possibly work on their own components independently). Licensing schemas are under investigation.

Challenge B

- so you have your AXI components in the firmware (base address configured etc…) and Linux running on your PS … what's next?
- for basic hacking: devmem or hand-crafted mmaping tools can help
- you can also do some Userspace I/O yourself using the generic uio kernel driver (see this)
- but let's have something more serious

PuzzledLizardWizard and solving challenge B

- PuzzledLizardWizard:
  - takes AirHdl JSON description (basically same as your regmap seen on the Web)
  - generates software libraries that enable very easy integration of the component in the higher-level software
  - is totally Free + Open Source, *produit artisanale* from CERN (AOC).
- Basic example: for regmap as few slides before, the generated library has function/method "read_status" for register "status", etc.
- PuzzledLizardWizard has two versions:
  - original, generates C software libs (loadable via Python *ctypes*)
  - new, under development (not publicly released), generates C++ software libs and the Python wrapper using Boost.Python

PuzzledLizardWizard-based components

- the components are basically data-model wrappers on top of libuio (NOTE: there are at least two distinct libs called such, one from Digilent, another one which is better) - see this
- libuio features component enumeration (by name or id) from the device tree - as long as your DTB is correct, you get nice tools like "lsuio" plus corresponding software components from PuzzledLizardWizard
- the support for reading and writing is generally there, support for IRQs - not yet.

Final notes

- PuzzledLizardWizard is a companion tool for AirHdl
- using both products enables simplified workflow for simplified PL-PS interfacing using AXI4 (and you don't have to learn AXI)
- in the EMP team it speeds up development so that experts on particular elink protocols can focus on their job rather than on AXI signalling or software companion
- both products are versatile and universal
- AirHdl seems not widely known at CERN - thus it was worth sharing.