Control servers for ATCA based LLRF system

Piotr Pucyk - DESY, Warsaw University of Technology
Jaroslaw Szewinski – Warsaw University of Technology

Agenda

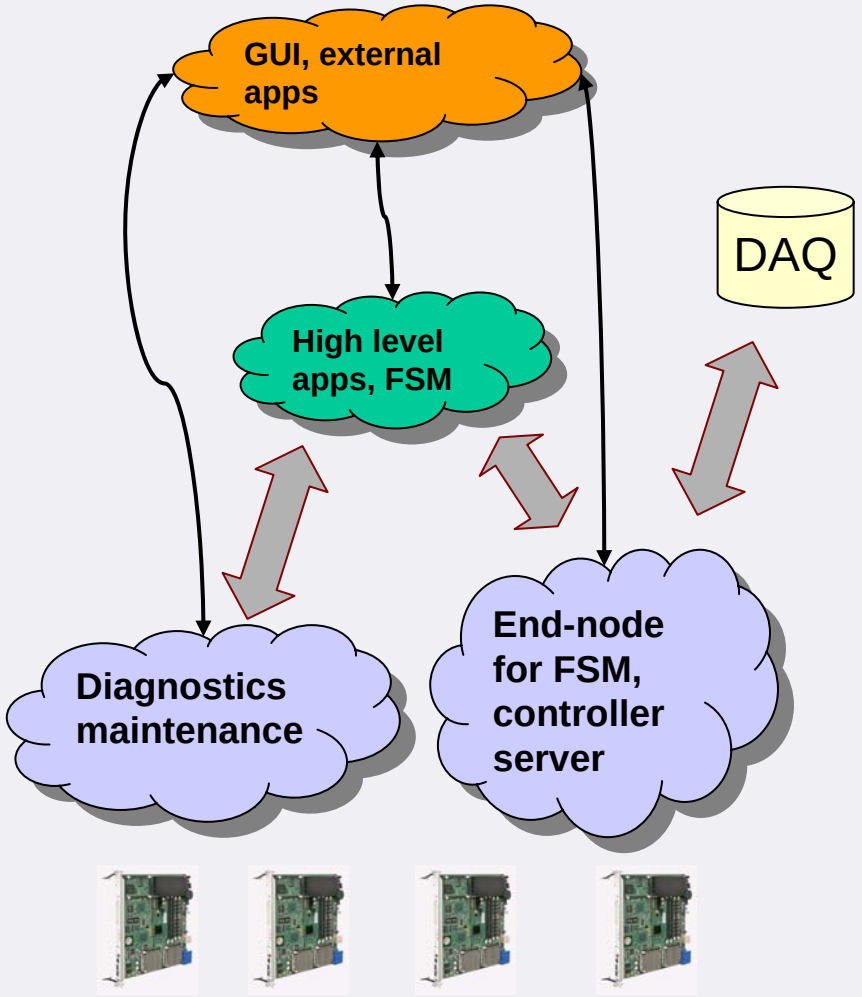
- Requirements
- LLRF servers classification
- ATCA computation power for LLRF servers
- Servers topology in ATCA based LLRF system
- Possible control systems
- Development environment
- Time schedule and summary

Requirements

- What servers should do ?
  - Should make possible remote access to the applications (update parameters, readout data, etc.)
  - Should support hardware in computation with given timing constraints (recalculate tables between pules, etc.)
  - Servers should guarantee reliable data transfers to the history storage
  - Should provide, for local applications, interface and interaction with external systems (other servers, foreign hardware, etc.)
  - Should provide management, control and synchronization of multiple subsystems (procedures, FSMs, automation)

What control software do we need?

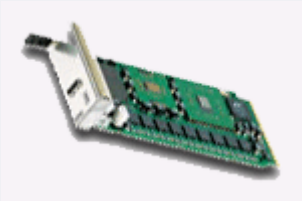
- Finite state machines for automation
- Front-end servers for hardware maintenance, configuration and diagnostics
- Front-end servers for controller and low level applications (execution nodes for state machines)
- Middle layer servers for high level applications
- DAQ servers or interfaces to DAQ
- GUI panels, interfaces to Matlab, C, etc.



Different CPUs in ATCA based LLRF system

Embedded processors	AMC module processors	ATCA blades	mainframes

Computation power



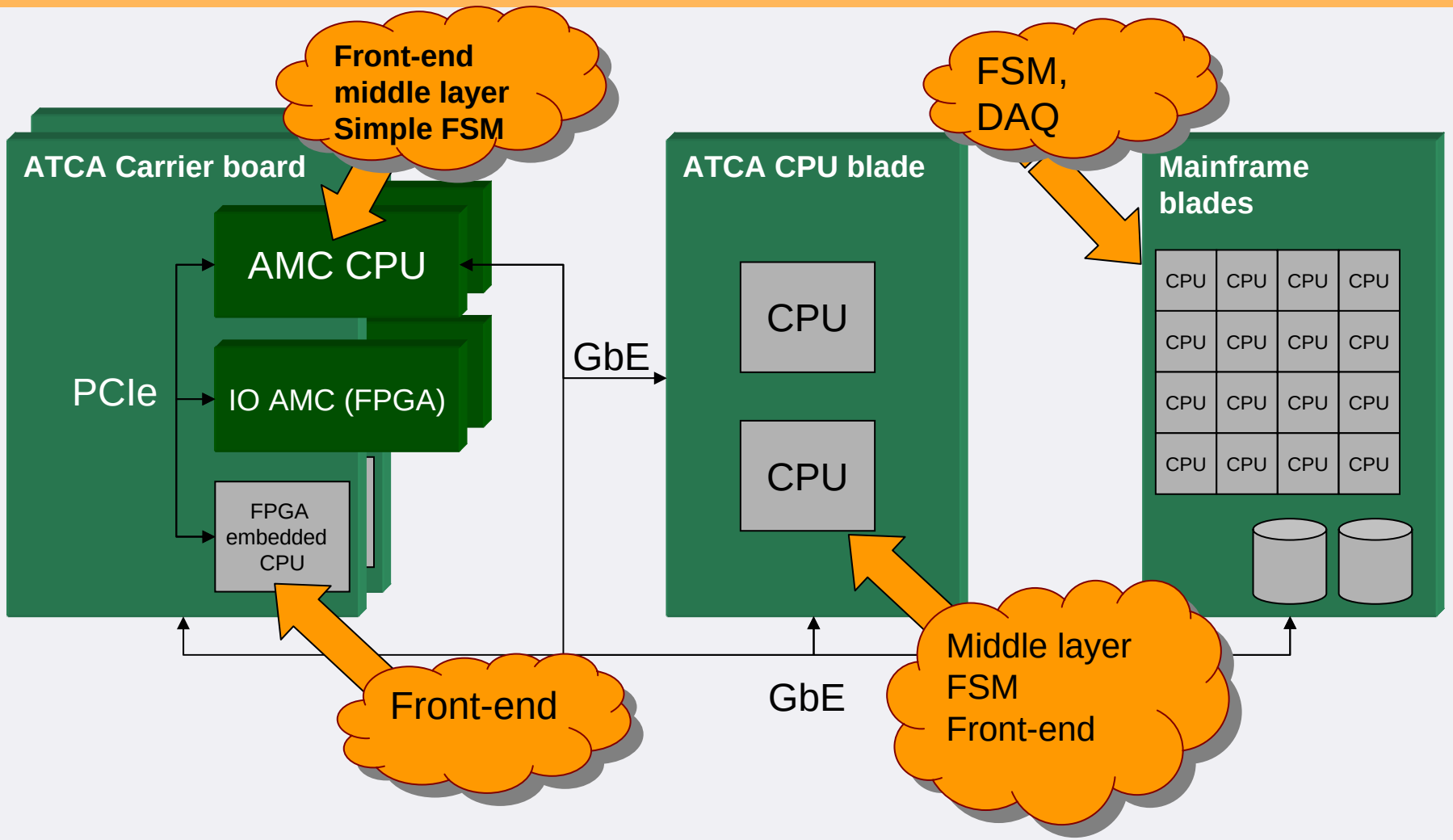
- Low processing power
- FPGA resources
+ close to the hardware

+ serious processing power
- uses AMC slot on carrier
+ PCIe, GbE

+ server class processing power
- No PCIe until now

• Huge processing power
• DAQ, storage
• Post processing
• GbE

Possible topology of LLRF servers



Possible control systems

- DOOCS
  - Used for the FLASH control
- EPICS
  - Widely used, low hardware requirements
- TINE
  - Heterogeneous/portable, runs on different platforms (DOS, Windows, various UNIXes, VxWorks, etc), can use different network protocols (TCP/IP, IPX/SPX, etc.)
- Dedicated software
  - custom servers, often embedded
- Hybrid configurations
  - Standard control system server (on ATCA CPU board) as gateway for small, lightweight custom embedded servers in different ATCA facilities (AMC, Carrier, etc.)

What we can reuse, what we have to develop

- Reuse
  - Old communication scheme (one software interface – many hardware drivers), memory description map
  - Tools for debugging and configuration
  - Some existing server's source code

- New development
  - Diagnostics, maintenance interfaces (inc. IPMI, crate management)
  - FSM framework for high level apps
  - New communication channels

Development environment & tools

- Platforms - preferred free, open systems like Linux or FreeBSD. Linux is widely used on various systems (from embedded to mainframe and cluster – can run in different facilities of ATCA). It has good hardware support - excellent platform for development.
- Usage of real time (commercial) systems (like VxWorks) – only where Linux can not be used.
- Technologies - 'keep it simple when possible ' - preferred well known, widely **portable** technologies like:
  - C/C++
  - BSD sockets + TCP/IP
  - Pthreads
  - etc.

Schedule, manpower

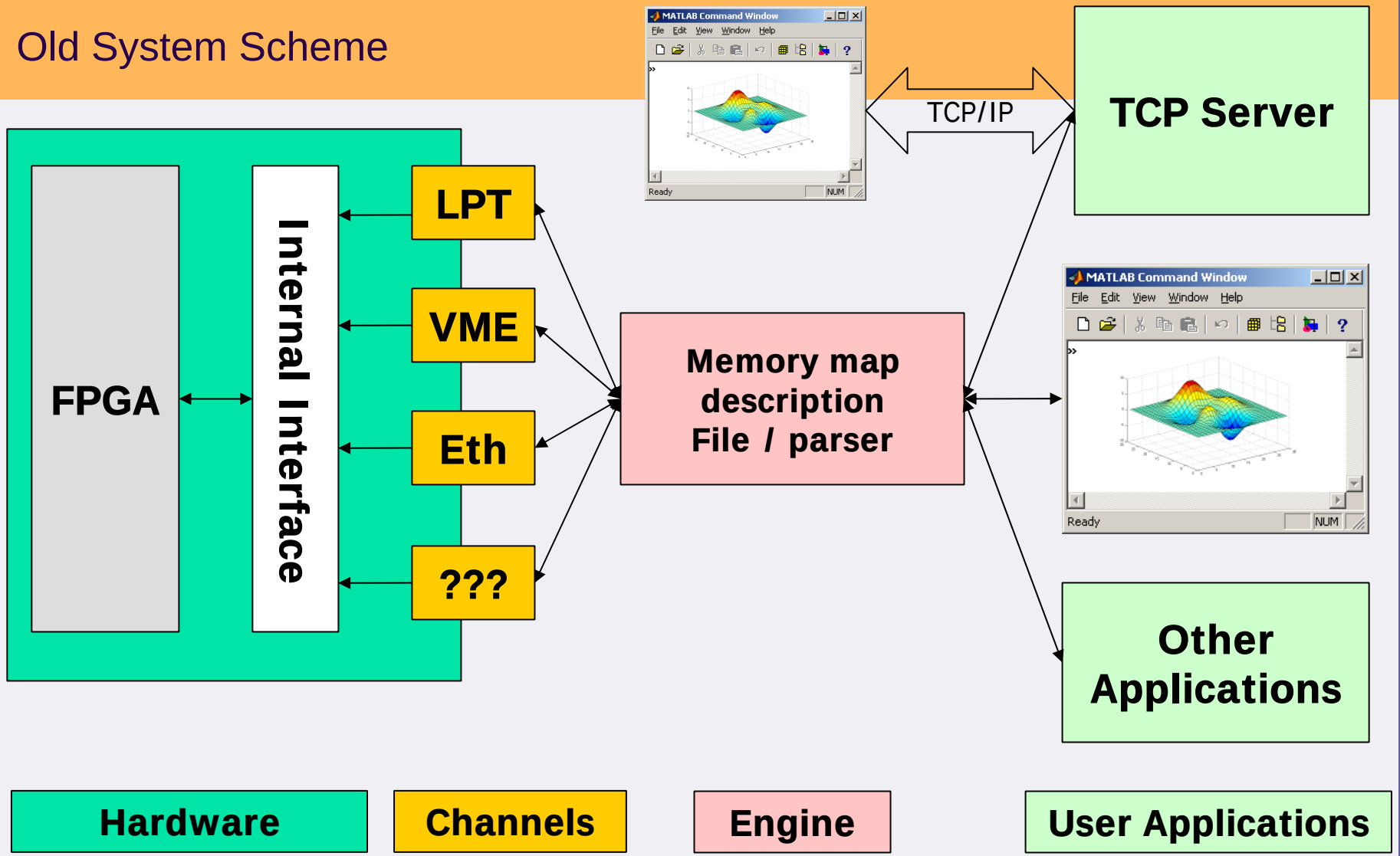
- Before we start:
  - Development environment, control system, communication libraries, FSM framework.
- Development schedule strongly depends on other tasks
  - Configuration and maintenance servers (when carrier board and at least one AMC is debugged and ready for firmware implementation - 3-5.2008 ?)
  - Finite state machines and procedures – starting from 1.2008 ?
  - Controller server, and low level applications interface servers (parallel to controller development and low level applications)
- Minimum Manpower
  - 1 fulltime programmer for front-ends, 1 fulltime for FSM and high level apps, 1-2 students for help
  - A lot of support from MCS group !!!

Backup slides

Development & *Design guidelines*

- Usage open-source, free tools when possible
- Projects organized by ***Makefiles****, to enable automatic build of all sub-components, including cross-builds for different targets (CPUs on AMCs, DSPs, PPCs in FPGA, ATCA CPU boards, etc.)*
- *Codes organized by version control tools (**CVS**, **SVN**)*
- *Codes documented by tools like **Doxygen***
- *Common coding conventions & policy*
- *Separated **mechanism** (platform dependent) and **policy** (platform independent)*
- Avoid platform depended, hardly portable solutions
- Avoid fancy external tools/libraries/technologies which development or support my be stopped one day.

Old System Scheme

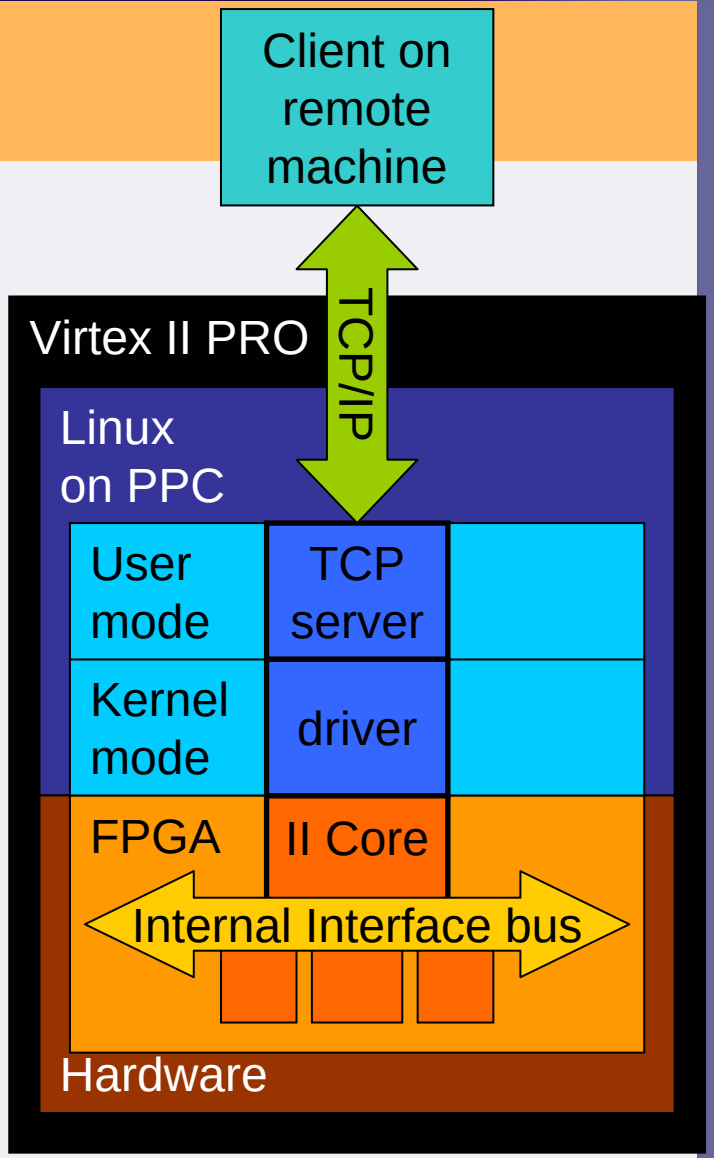


Linux on PowerPC

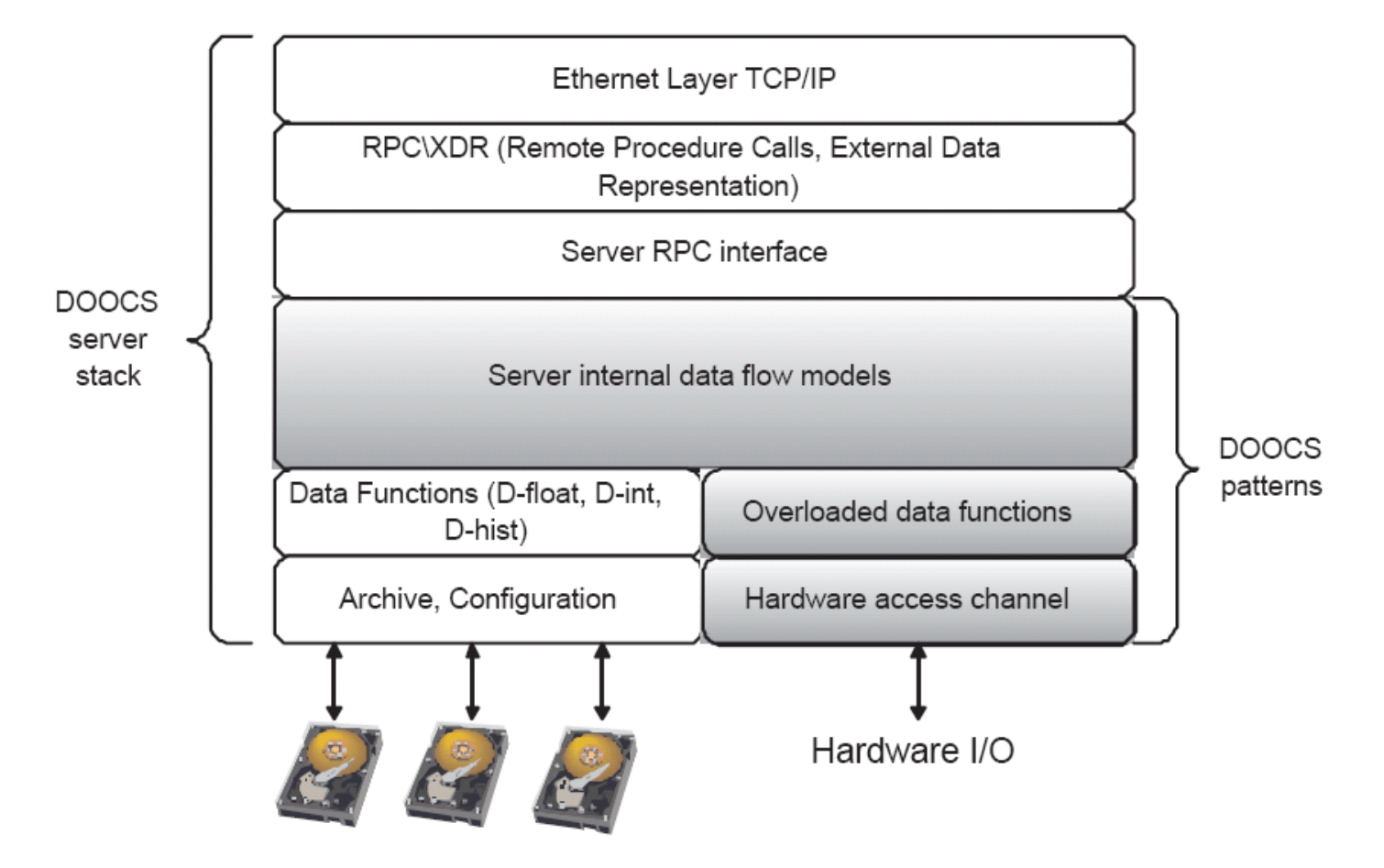
User applications access hardware through the driver

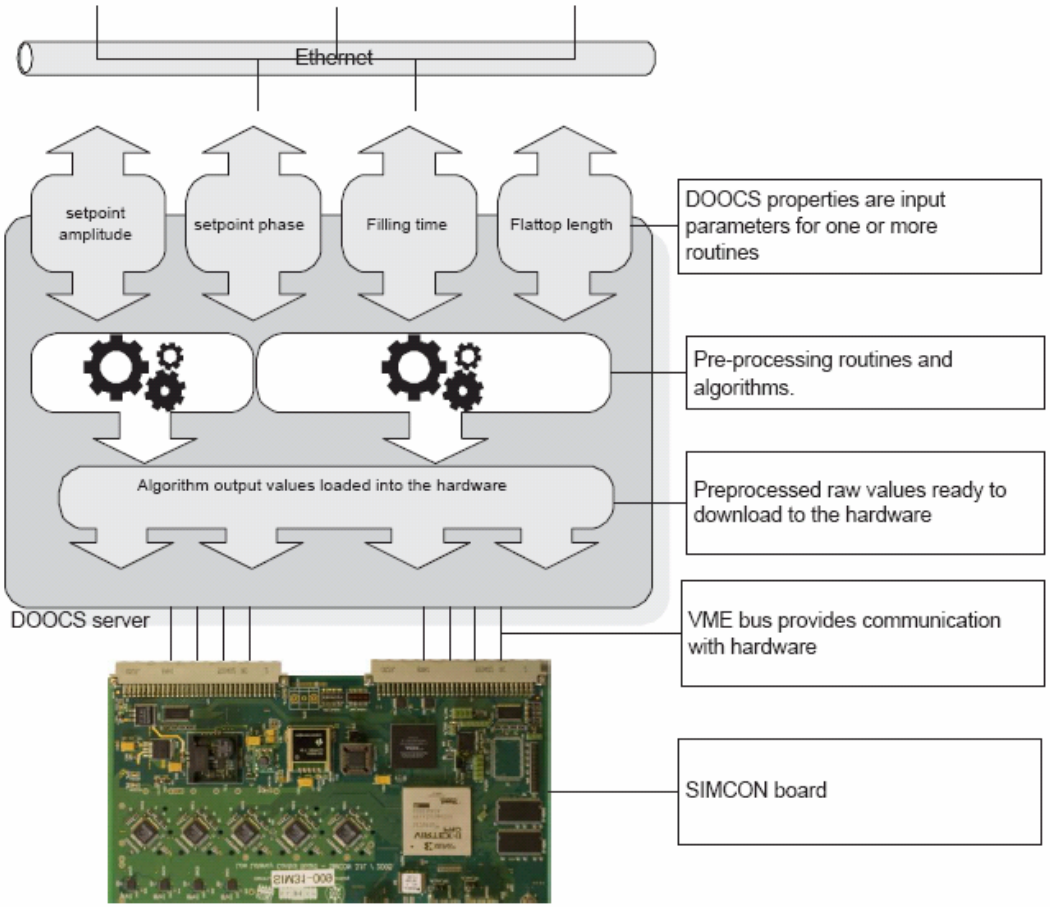
Kernel mode driver has access to the FPGA

FPGA has defined hardware interface



DOOCS patterns



http://flash.desy.de/sites/site_vuvfel/content/e403/e1644/e1136/e1137/infoboxContent1796/tesla2006-10.pdf

DOOCS patterns (2)