Software for the main functionality

Rafał Kiełbik & Wojciech Jałmużna

About Us

Wojciech Jałmużna

Ph.D. in Electrical Engineering

- Role
  - RFPI project contractor
  - FPGA/software engineer
  - Hardware Designer
- Relevant experience
  - FLASH/XFEL LLRF Systems (2003-2012): HW/FW designer
  - Various electronics systems for HEP (2012-2018): HW/FW designer
  - ESS-ERIC: FPGA/software engineer (since 2018)
  - POLFEL LLRF and Piezo Control: HW/FW designer

About Us

Rafał Kiełbik

Ph.D. in Electrical Engineering

- Role
  - RFPI project contractor
  - FPGA/software engineer
- Relevant experience
  - ESS-ERIC: FPGA/software engineer (since 2018)
  - ARUZ Large-scale, FPGA-based Analyzer of Real Complex Systems: project coordinator, FPGA/software engineer (since 2005)

Agenda

- The main requirements,
- The PoC version specification and scope,
- The functionality and design details,
- Implementation,
- Test results discussion,
- Full scale design plans,
- Summary

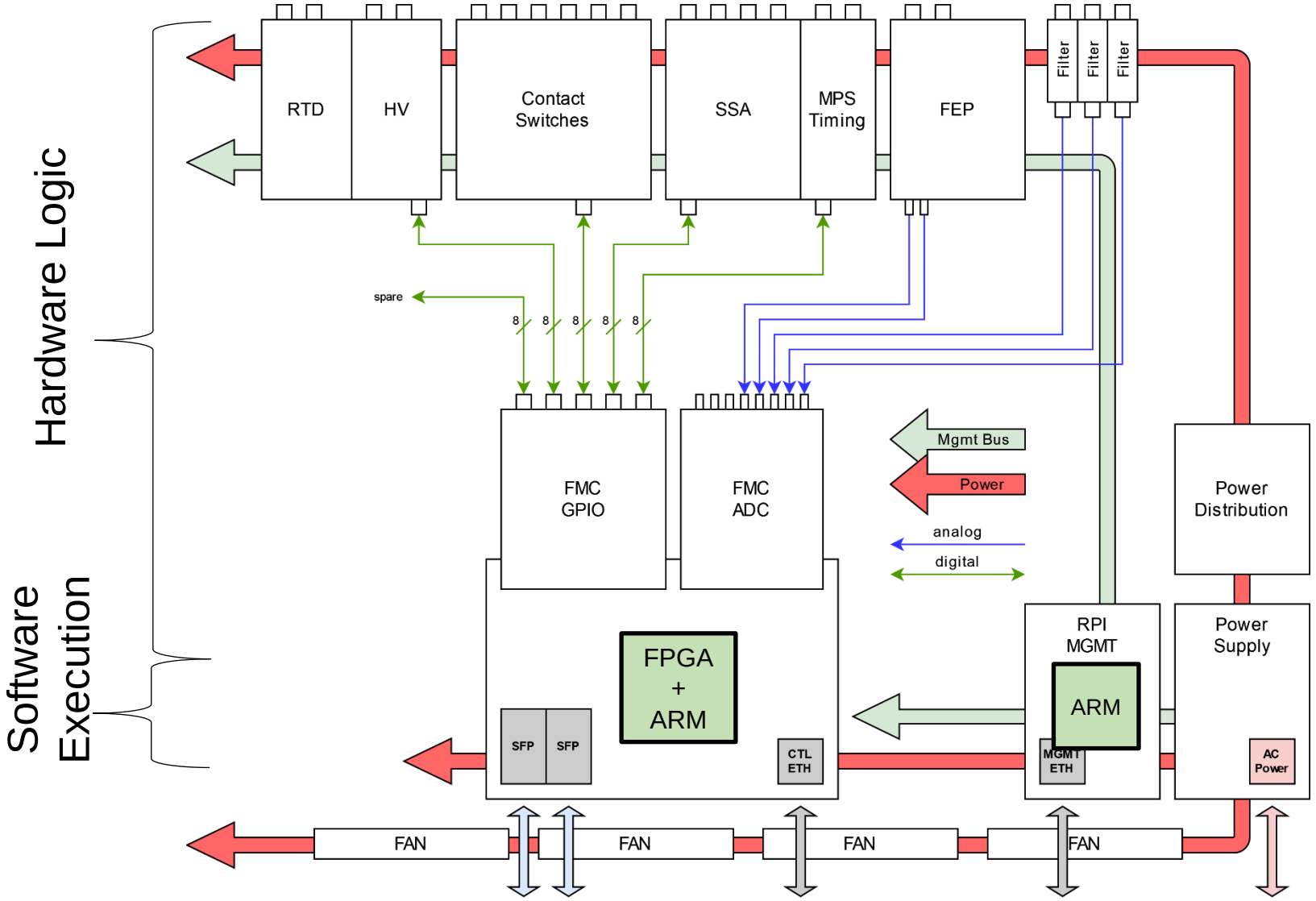
The main requirements

- Firmware(+hw) Layer must provide real-time response to all events within the time specified in the signal tables
  - Firmware is treated as low level hardware layer
- Software Layer must provide interface to fully control system parameters and provide monitoring and archiving of all critical data
- Execution of protection function must be independent of non-critical software components
  - No parts of protection function are running on SW level
  - Software can be started and stopped without disturbance to system functions
- In case of critical software fault, appropriate statuses must be reported and inhibit signals raised

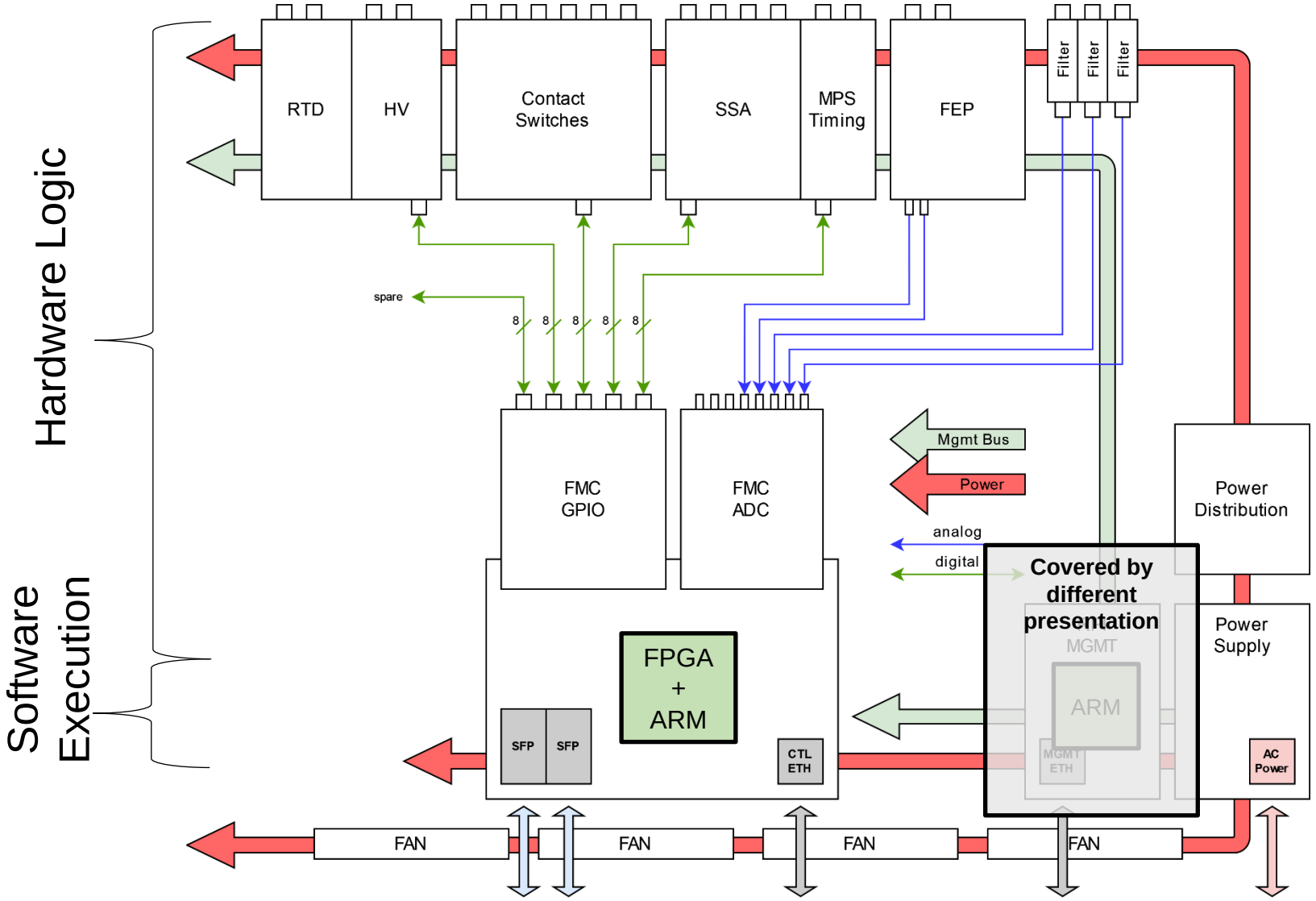
The PoC version specification and scope

- Main purpose of the initial POC fw/sw is to test hardware layer and ensure that all low-level hardware requirements can be met
- POC implementation is focused on easy to use, efficient tools to perform various actions on hardware
- EPICS layer was added to demonstrate possibilities and evaluate performance, but it is not intended to be final control application

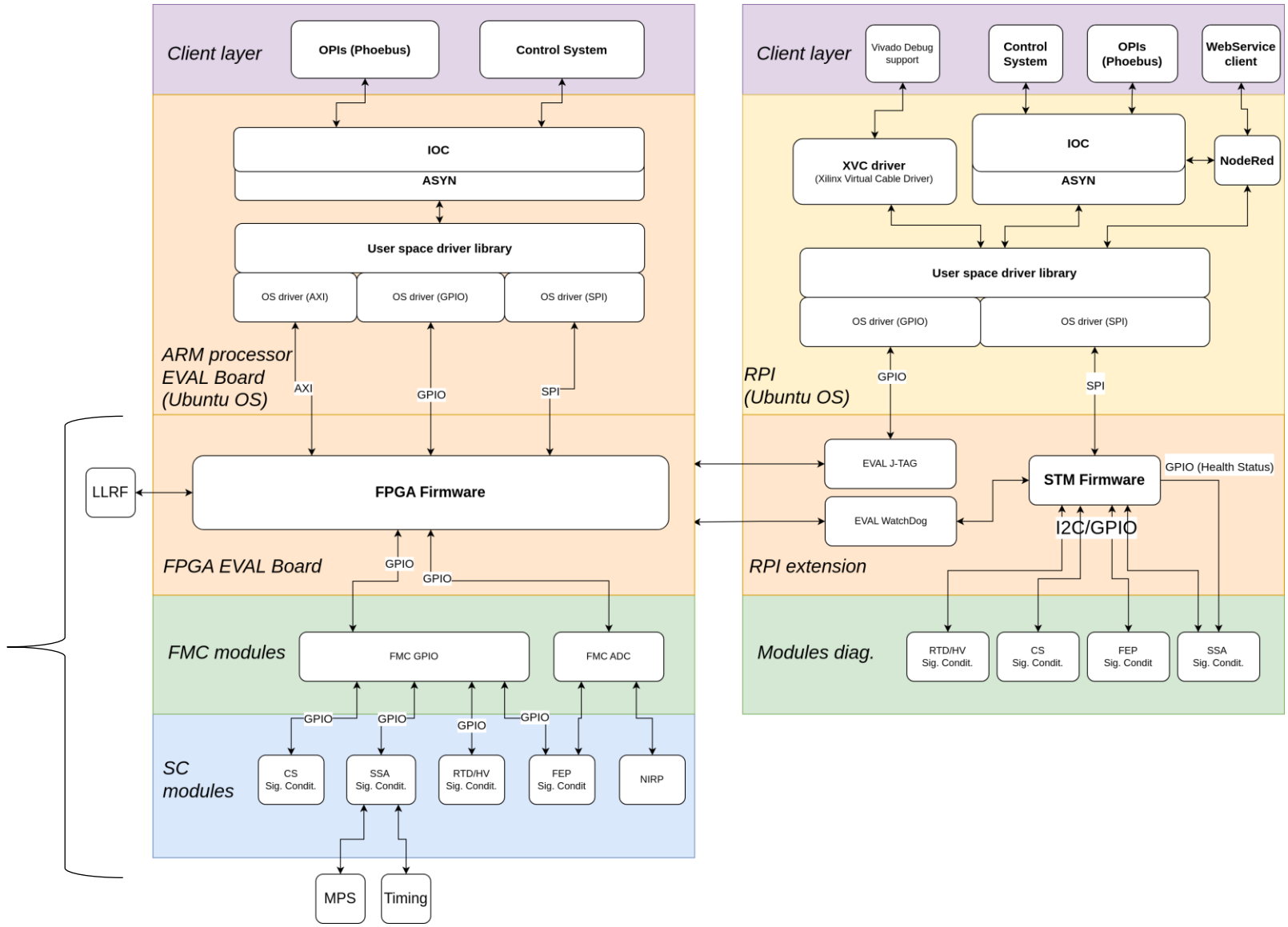
The functionality and design details (1)



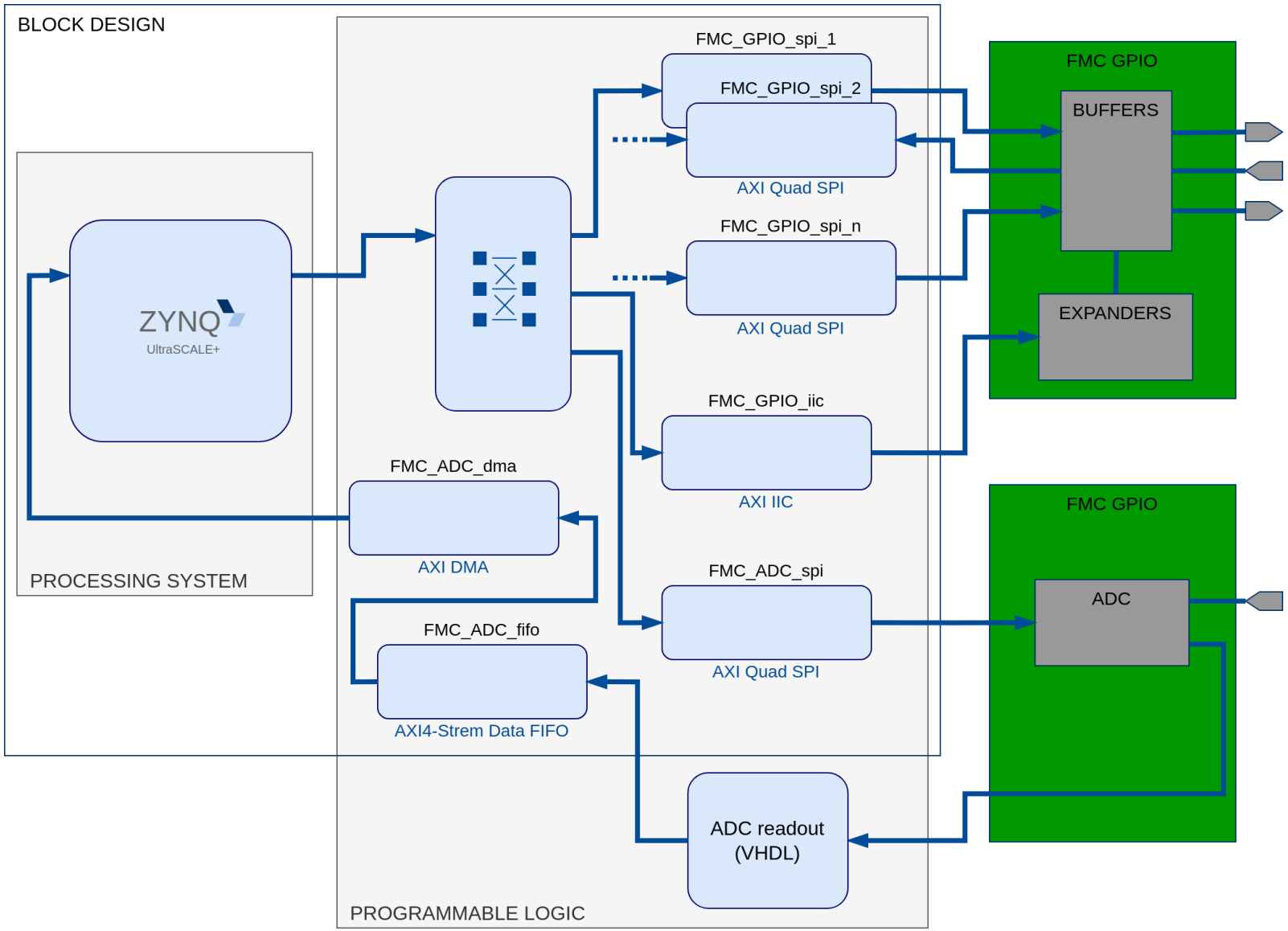
The functionality and design details (1)



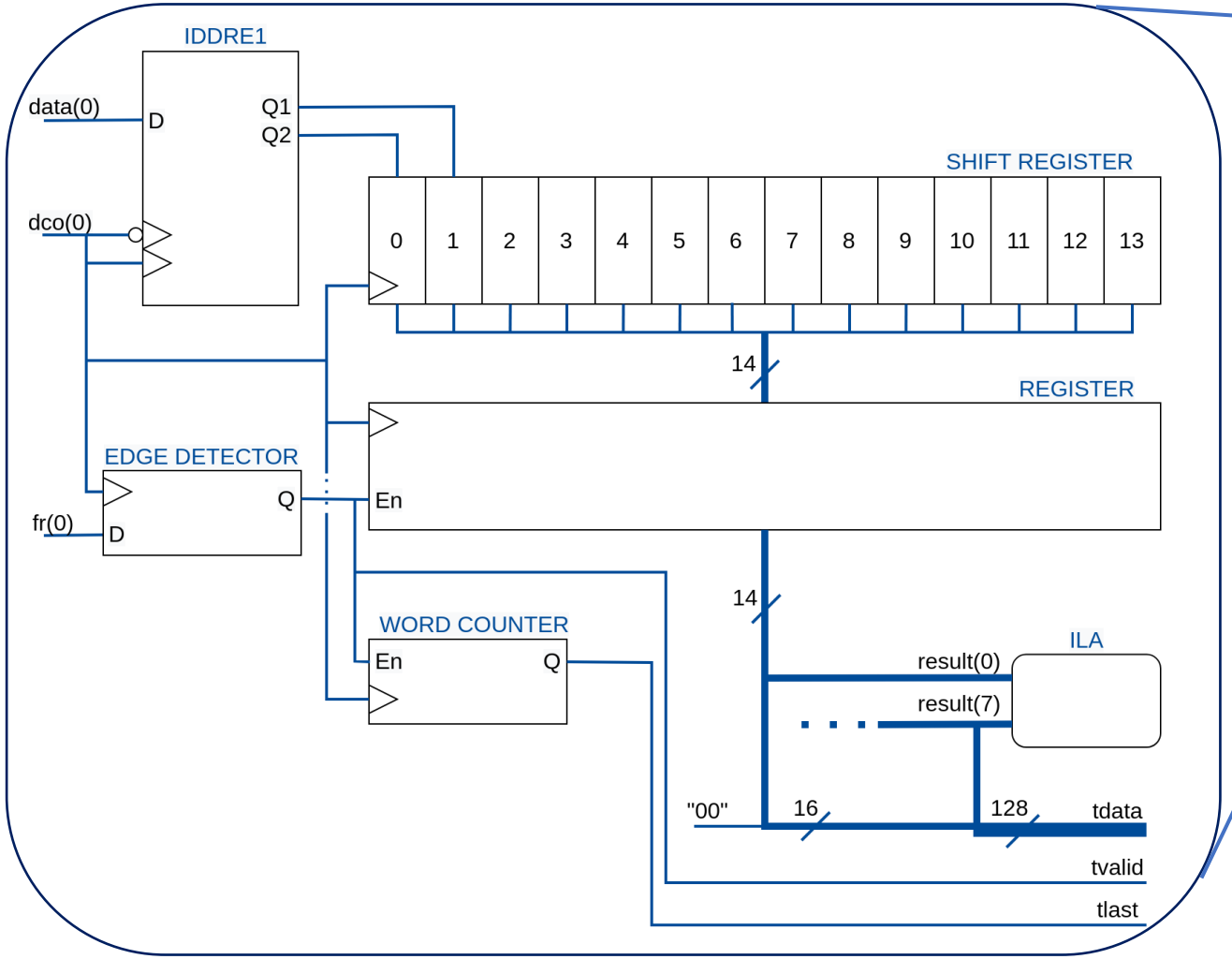
The functionality and design details (2)

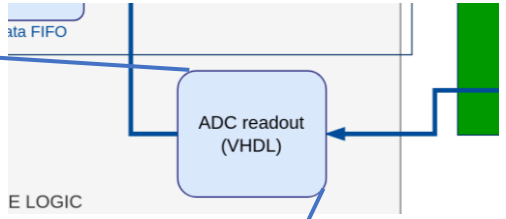


Implementation - Firmware Architecture



Implementation - Firmware Architecture





Implementation - Flow

- Management of hardware platform for Ubuntu is performed using:
  - xlnx-config command line tool
  - PAC (Platform Assets Container) folder
- Process automation is obtained by means of dedicated bash and TCL scripts for:
  - Exporting Vivado project to TCL
  - Recovering Vivado project from TCL
  - Generating bitstream
  - Exporting hardware platform
  - Creating Vitis platform for DT (Device Tree)
  - Compiling DT
  - Transferring generated files (bitstream, DT) to PAC folder (on ZCU106)
  - Updating Ubuntu configuration with xlnx-config command (on ZCU106)

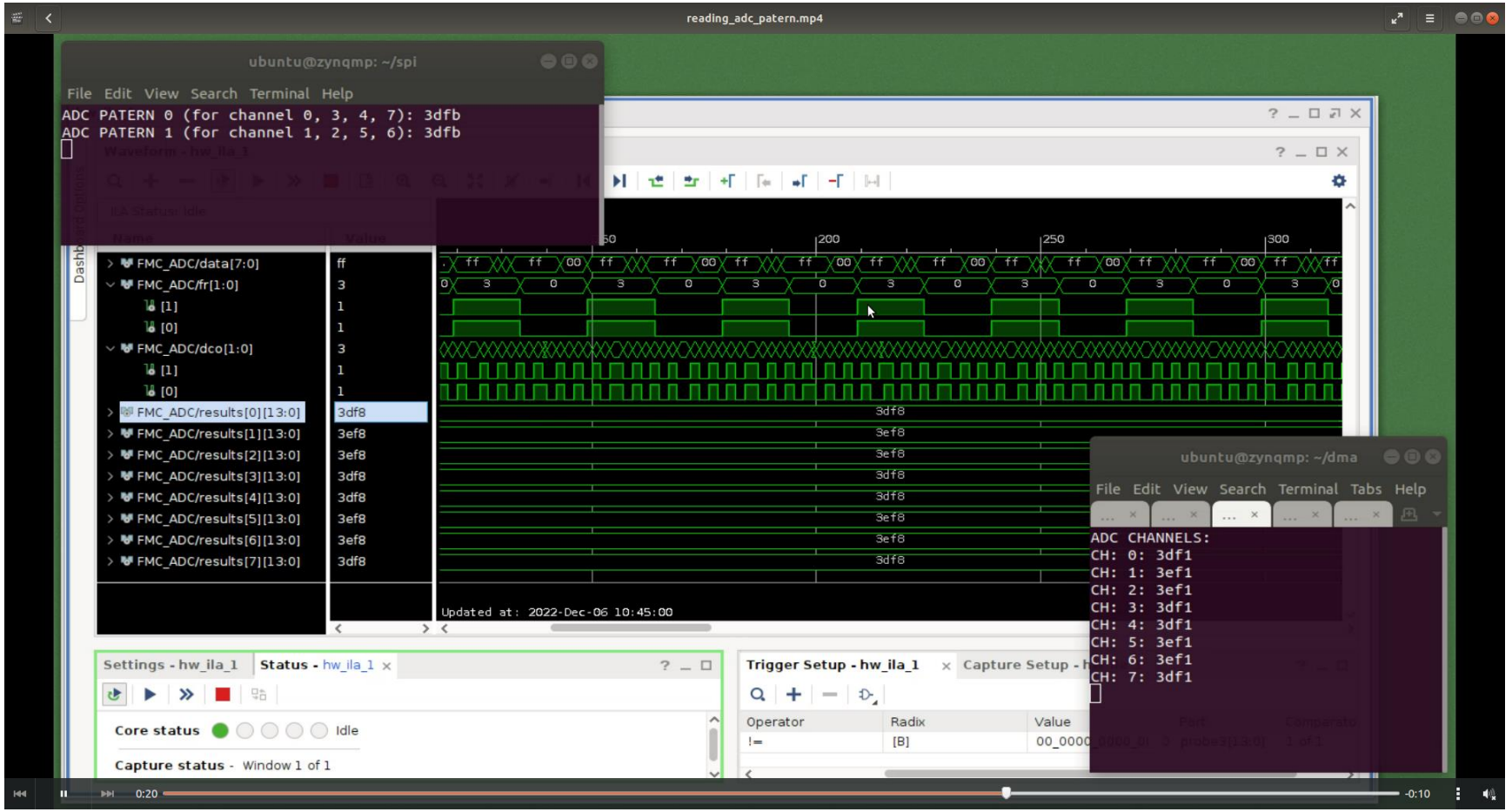
Implementation - Drivers used

- SPI
  - spi-xilinx (kernel space only, built-in)
  - spidev (user space, built-in, kernel reconfiguration required)
- IIC
  - i2c-xiic (user space, built-in)
- DMA
  - xilinx_dma (kernel space only, built-in)
  - dma-proxy (user space, loadable, compilation required)
- UIO (Userspace IO)
  - used to handle address space access (for example register modules on fw)

Low Level Applications Prepared

- FMC FRU manipulation (via I2C)
  - eeprom_clean – erasing FRU FLASH
  - eeprom_dump – reading FRU FLASH
  - eeprom_write – witing FRU on FLASH
- Expanders configuration (via I2C)
  - gpio_loopback – setting all buffers of FMC GPIO in loopback mode
  - spi_expander_test – setting some buffers of FMC GPIO for SPI transfer
- Expanders configuration (via SPI)
  - spi_expander_test – setting expanders of conditioning boards
- ADC configuration (via SPI)
  - adc_config – reseting ADC, setting ADC mode (and pattern, if required)
- ADC readout (via DMA)
  - basic_test – testing single DMA transfer
  - dma_transfer – performing continuous DMA transfer and printing averaged ADC readouts, multi-threaded

Demo: Setting and Reading ADC Patterns

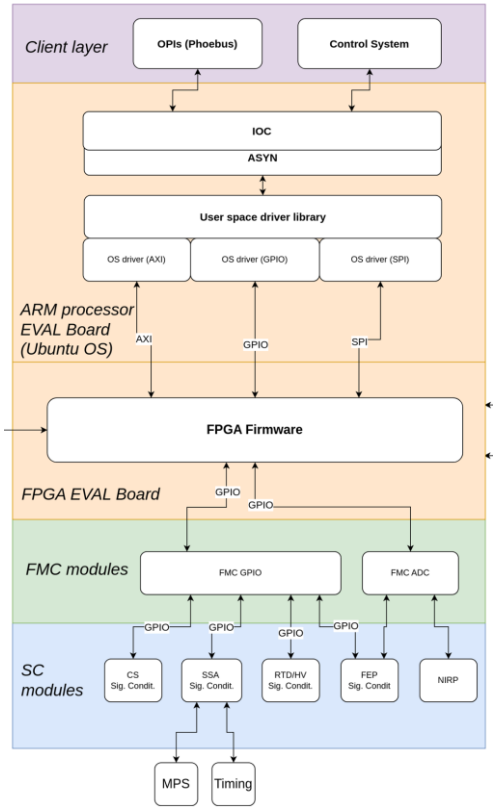


Implementation – Higher Level Software Layers

- User Space HW Library
- Command Line Tools
- EPICS IOC
- XVC Driver

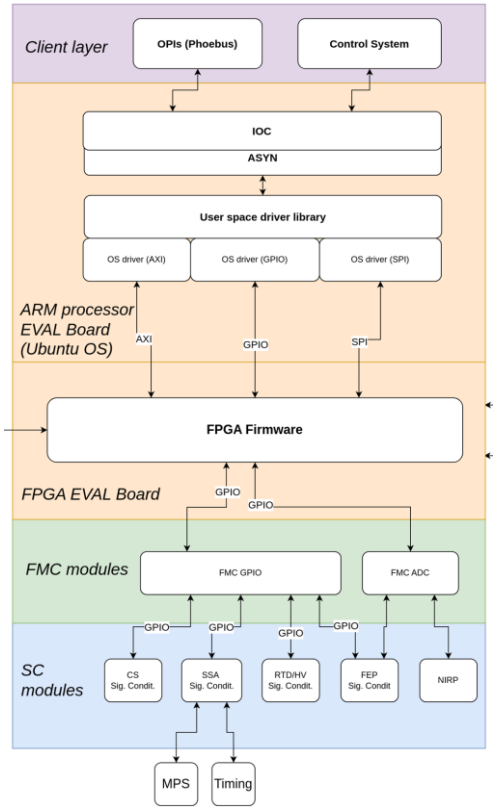
Implementation – Higher Level Software Layers

- **User Space HW Library**
  - C++
  - Software reflection of all hw/fw parts
  - Abstraction layers and easy to use API
    - Low level API can be used when needed
- Command Line Tools
- EPICS IOC
- User Interface



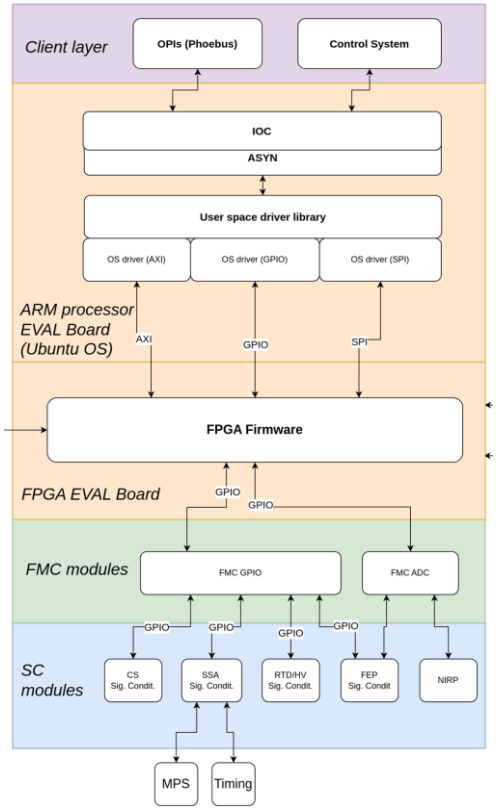
Implementation – Higher Level Software Layers

- User Space HW Library
- **Command Line Tools**
  - They are just interface to HW Lib
  - Give quick and easy access to low level hw features
  - Can be used for quick testing of various components without the need of running of the whole control system
- EPICS IOC
- User Interface



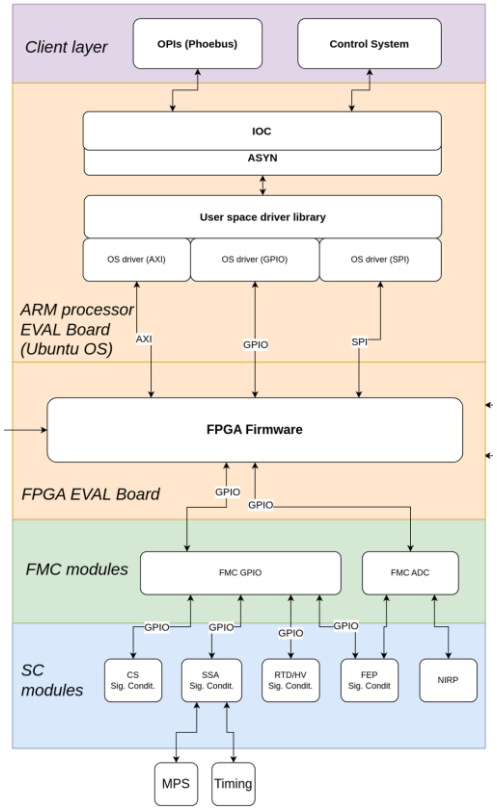
Implementation – Higher Level Software Layers

- User Space HW Library
- Command Line Tools
- **EPICS IOC**
  - IOC is using ASYN driver to interface with the lower layers of sw stack
  - ASYN interfaces the same SW lib as command line tools
  - IOC is not implementing complex logic
- User Interface



Implementation – Higher Level Software Layers

- User Space HW Library
- Command Line Tools
- EPICS IOC
- **User Interface**
  - User Interface is implemented using CSS



Summary

- Prototype Version of RFPI software tools is implemented
  - Main purpose: tests of hw layer and performance evaluation
- Current implementation can be base for further development
- General Conclusions move us towards final shape of the system

Thank you !