EIC Controls

Rapid Device Support Development Using Hardware Abstraction Server and P4P

Andrei Sukhanov, application architect, C-AD, BNL

Electron Ion Collider – EIC at BNL

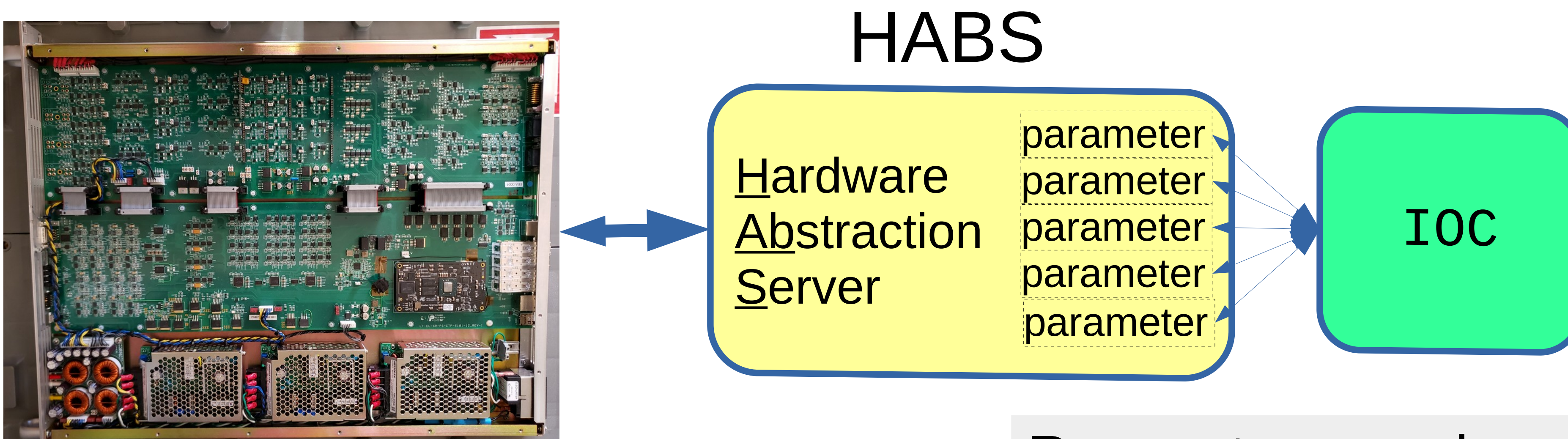
BROOKHAVEN NATIONAL LABORATORY

U.S. DEPARTMENT OF ENERGY | Office of Science

Motivation

- Provide set of C/C++ functions to build simple server for a device.
- Point-to-point communication to a client.
- The IOC (PVAccess) should be automatically generated.
- Full introspection of device parameters.
- The full system should be able to run standalone (on Raspbery Pi).

Rapid Device Support Development Using Hardware Abstraction Server



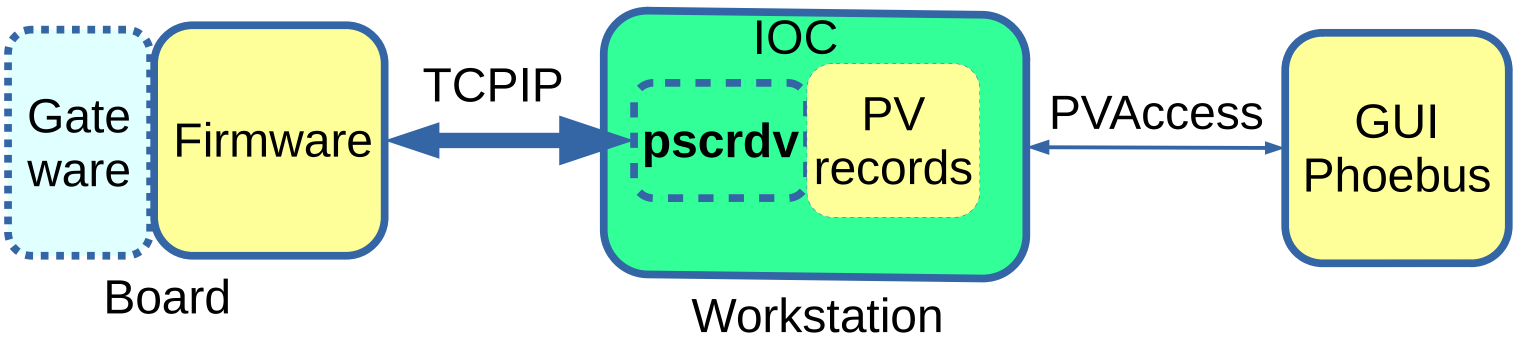
Parameters properties:
- Value (scalar, vector, …)
- Type
- Timestamp
- Feature bits
- Description
- ...

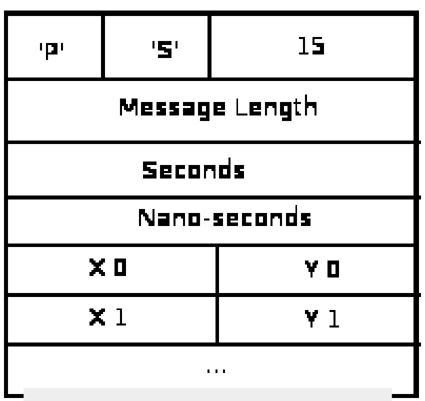
Feature bits: **R**eadable, **W**ritable, **E**ditable, **M**easurement

Requests served:
- `info` — Full introspection
- `get`
- `set`
- `run` — To start/stop continuous measurements

Traditional approach

pscdrv toolchain





Message frame

```
record(waveform, “wf:X”) {
  field(“DTYP”,”PSC Block I16 In”)
  field(“SCAN”,”I/O Intr”)
  field(“FTVL”,”DOUBLE”)
  field(“NELM”,”1024”)
  field(“INP”,”NAME 15 8 4”)
  info(“TimeFromBlock”,”0”)
}
```

Record example

Requires in-depth knowledge of

- EPICS device drivers,
- database records,
- IOC toolchain and
- EPICS GUI

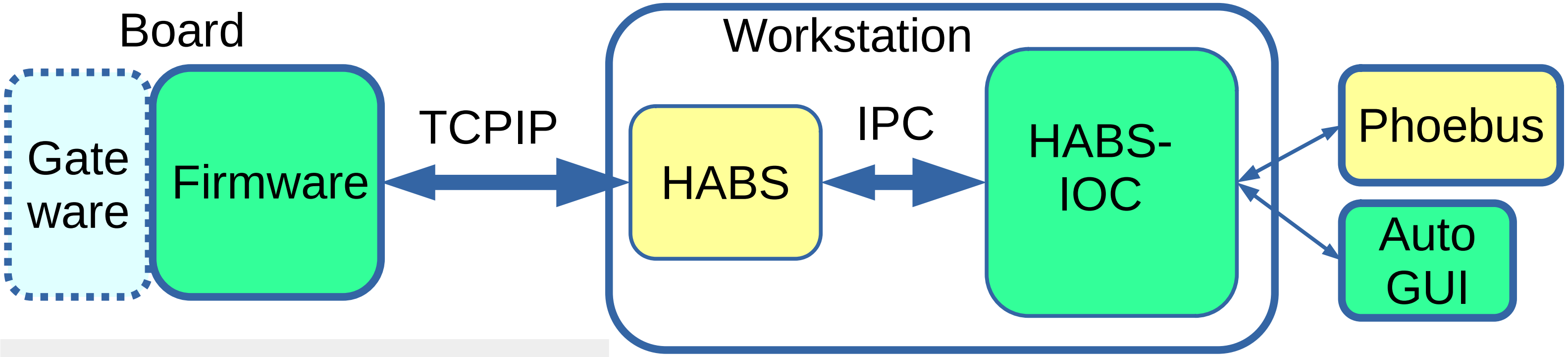
Color coding:

Developer's code

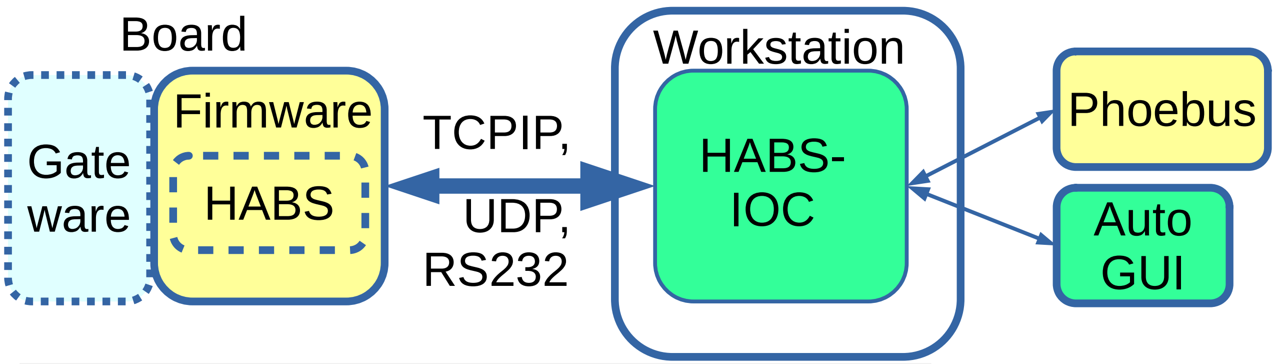
Provided code

HABS toolchain

Two types of implementation.



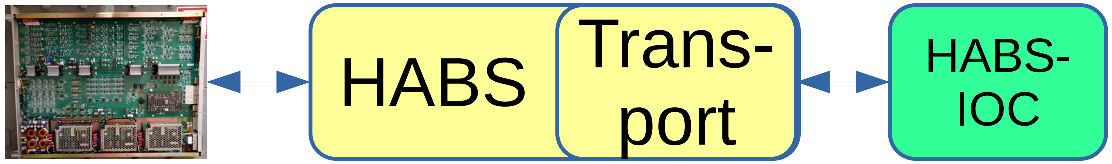
Option1: Closed firmware, off-the-shelf devices



Option2: Open firmware, microcontrollers

Color coding:
Developer's code
Provided code

HABS



Transport protocols supported:
✔ IPC Message Queue

In development:
- RS232
- UDP
- TCP-IP

HABS provides set of C++ functions and headers.

- **`init()`**: to initialize the set of published parameters and bind them to program variables.
- **`update()`**: called in the main loop to update parameters and respond to requests.

Data format of client communication is CBOR:
- binary JSON,
- self describing,
- very compact,
- standardized (RFC 8949),
- widely used in IOT,

Software components

Headers:

- pv.h - creation of parameters

Functions:

- p2plant.cpp - message parsing
- transport.cpp - communication with client

Library:

- libtinycbor.a - CBOR encoding. Public release from Intel.

Demo: https://github.com/Asukhanov/P2Plant

- make

Code Example

Create 3 parameters: **run**, **adc_reclen,** 2D vector **adcs,** and bind them to program variables

```
// Mandatory PVs
static PV pv_run =  {"run",
    "Start/Stop the streaming of measurements", T_str, F_WED};
pv_run.legalValues = (char*)"start,stop";
pv_run.set("stop");

// ADC-related PVs
#define ADC_Max_nSamples 2000
static PV pv_adc_reclen = {"adc_reclen",
    "Record length. Number of samples in each channel", T_u2, F_WE};
pv_adc_reclen.opLow = 10;
pv_adc_reclen.opHigh = 10000;
pv_adc_reclen.set(ADC_Max_nSamples);

static uint16_t adc_samples[ADC_Max_nChannels*ADC_Max_nSamples];
static PV pv_adcs = {"adcs",
    "Two-dimentional array[adc#][samples]",T_u2ptr, F_RM, "counts"};
pv_adcs.set(adc_samples);
```

Python interface to HABS

It simplifies debugging of the device

```
>>> from p2plantaccess import Access as dev
>>> dev.init(); dev.start()
>>> info =  dev.request(['info', ['*']])
>>> reply = dev.request([
            'get', ['version','run'],
            'set',[('adc_gains',[0.5,1,2,4]),
                   ('run',['start']),
            'get',['adc_amplitude[2]'],
        ])
```

Several requests could be combined in one transaction

```
>>> print(info)
{'*':{
   'version': {'desc': 'simulatedADCs version',
               'type': 'char*', 'shape': [1], 'fbits':'R'},
   'run': {'desc': 'Start/Stop the streaming of   measurements',
               'type': 'char*', 'shape': [1], 'fbits': 'WRDsrE',
               'legalValues': 'start,stop'},
   'debug': {'desc':
   …
}
```

Any request returns a dictionary

HABS-IOC

Universal softIocPVA, communicating to any HABS, written in python using p4p wrapper for PVAccess.

Module: p2plant_ioc, to install:

```
pip install p2plant_ioc
```

To run

```
python -m p2plant_ioc -l
```

Companion modules:
pypeto: to automatically create tabular control page,
pvplot: plotting tool,
apstrim: logging of time-series data.

Summary

Development of device support is greatly simplified using HABS.

➔ No EPICS expertise required, only:
- basic C/C++
- basic Python (for device debugging).

➔ HABS could be embedded in device firmware (microcontroller or FPGA soft core), or run at a separate host.

➔ Multiple transport protocols: IPC, RS232, UDP, TCP-IP.

➔ Full introspection.

➔ IOC is generated automatically.

➔ Binary object streaming is based on well-established standard (CBOR).

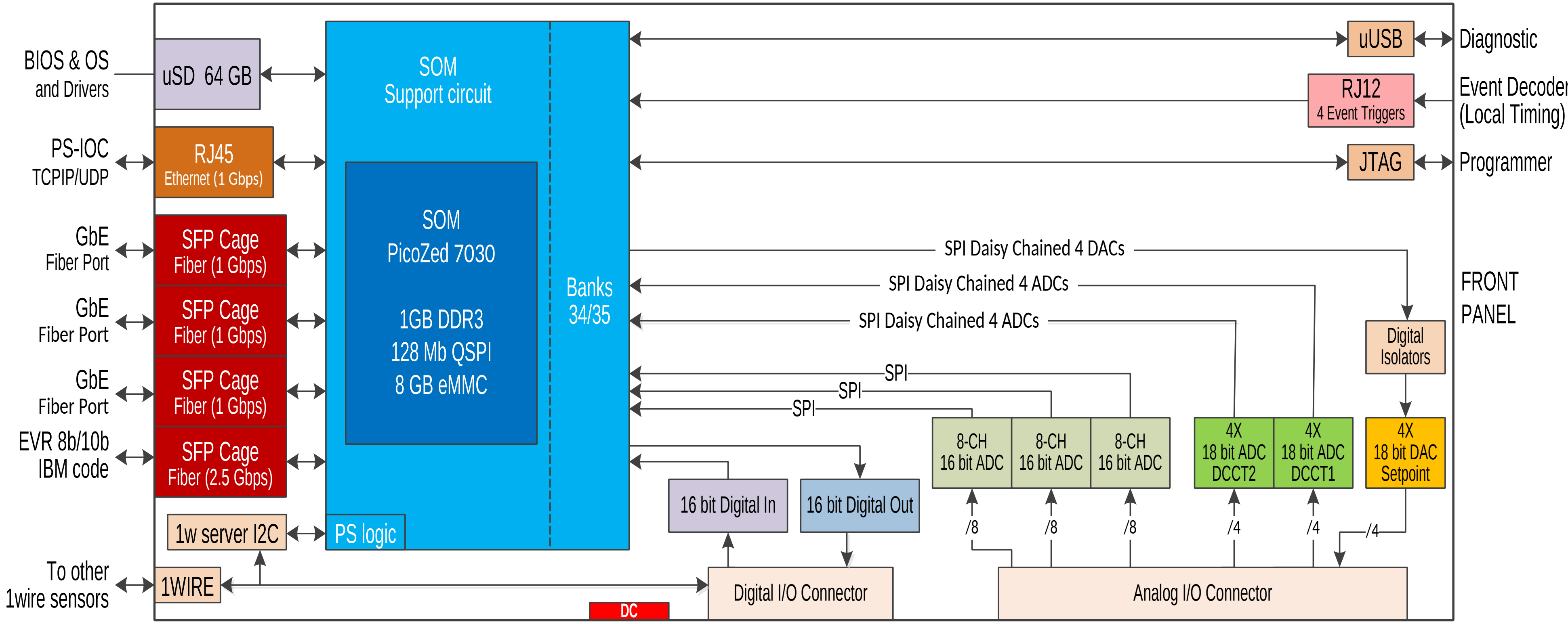
➔ Network traffic is significantly reduced using combined requests.

➔ Simplicity: point-to-point communication, only 4 request types, fixed subscriptions.

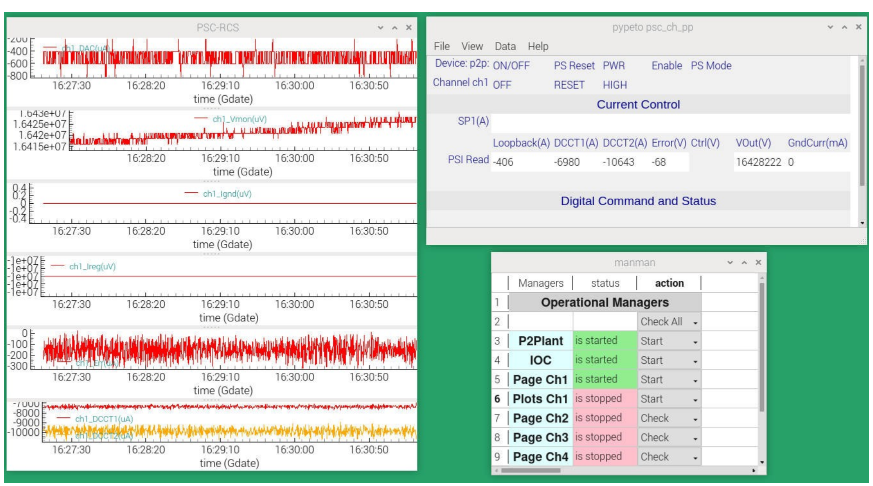
Resources:

HABS:	https://github.com/ASukhanov/P2Plant
HABS-IOC:	https://github.com/ASukhanov/P2Plant_ioc
Python access:	https://github.com/ASukhanov/p2plantAccess

Implementations



Test/Development of ALS-U Power Supply Controllers.
Very little documentation was available.



EPICS support was developed in 1 week.
The whole system is running standalone on **Raspberry 5**.