Userspace Hardware Abstraction Layer for PCIe devices

Érico N. Rolim

MicroTCA Workshop 2022

CNPEM
Centro Nacional de Pesquisa em Energia e Materiais

LNLS sirius
Brazilian Synchrotron Light Laboratory

Target Hardware

- The Sirius Beam Position Monitors (**BPM**) and Fast Orbit Feedback (**FOFB**) systems use MicroTCA crates:
  - The backplane interconnect is set up with a PCIe bus between the AMC boards and the host CPU
  - The AMC boards in use are the AFCv4 (presented earlier today) and AFCv3.1 boards

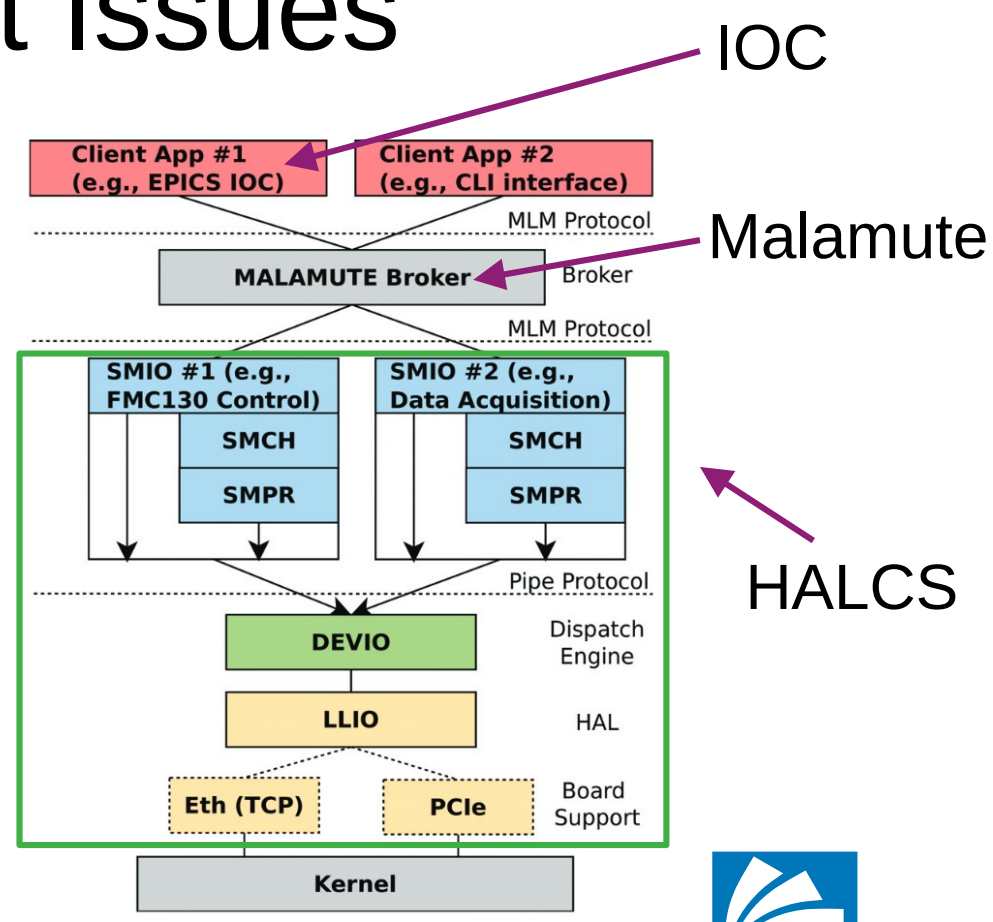
Software Requirements

- **Interface** with the AFC boards via the **PCIe** bus
  - BAR4 exposes hardware register maps for each FPGA core via Wishbone, which we use as our main interface
  - We get the addresses for the register maps from a Self Describing Bus (**SDB**) filesystem in address 0 of BAR4
  - Register map header and structs provided by Cheby (code generator from register map definitions)

- **So why are we looking for a new solution?**
- Should be…
  - Easy to **debug**
  - **Maintainable** in the long term

Current Issues

- There are 3 main kinds of *daemons* responsible for the hardware interaction
- They use RPC – **ZeroMQ**, specifically – between themselves
- These 3 kinds are:
  - Hardware Abstraction Layer for Control Systems (**HALCS**) – *in-house* development
  - **Malamute**
  - EPICS **IOCs** – *in-house* development



HALCS

- Motivation was to expose the hardware **safely**, enabling **multiple** simultaneous clients which don’t interface directly with the hardware

- Uses a custom kernel driver to talk to the hardware: makes it harder to **update our host distribution**
- The chosen internal architecture and abstractions ended up being **too complex**: makes it **hard to debug** issues
- The complexity also lead to verbosity: adding new functionality requires making changes in **multiple files**, with **no automatic verification**
- Not enough abstraction: mainly accesses register fields

Malamute

- A communication *broker* between HALCS instances and their clients

- **Abandoned** project (breaks the ZeroMQ aspect of *not* having a broker)
- Has **memory leaks** that used to make our crates **unresponsive** until we added a CGroups memory limit

```
● malamute.service - malamute service
     Loaded: loaded (/usr/lib/systemd/system/malamute.service; enabled; vendor preset: disabled)
     Active: active (running) since Ter 2022-11-22 12:07:32 -02; 1 weeks 5 days ago
   Main PID: 1638 (malamute)
     Memory: 1023.9M (limit: 1.0G)
     CGroup: /system.slice/malamute.service
             └─1638 /usr/bin/malamute -f /usr/etc/malamute/malamute.cfg
```

EPICS IOC

- Implemented with the EPICS Asyn module together with the HALCS client library
- Includes a state machine for high speed acquisitions

- Implemented as a **monolithic** driver, making code organization and reuse harder

New Design

- The new design is based on the issues that have been observed during development and operation

- Don’t use a custom kernel driver: **userspace only**
- Remove RPC layer: don’t duplicate the IOC’s role
- More **complete abstractions**: deal with signed and fixed-point values

- The initial implementation of this design is open source and has been called **µHAL**

μHAL

- **C++17 library** (making use of language advancements)
- Implementation is split between **decoders** and **encoders**
  - **Decoders** store register field values in a hash map using strings as identifiers (+ indexes for registers that are organized in “channels”)
  - **Encoders**/controllers can therefore perform **sanity checks** for inputted values
    - Deal with field length, hard-coded max values, fixed-point decimal point position (determined via a special register – allows changes to valid value range without redeploying software)
- Checks register ABI version, to make sure it’s supported

```
Core::Core(struct pcie_bars &bars):
    RegisterDecoder(bars, {
        /* normal wishbone registers */
        PRINTER("CC_ENABLE", "Enable CC module", PrinterType::enable),
        PRINTER("TFS_OVERRIDE", "Timeframe start override", PrinterType::boolean, "override, use external signal", "normal, use internal signal"),
        PRINTER("TOA_RD_EN", "Enable Time of Arrival module for reading", PrinterType::enable),
        PRINTER("TOA_DATA", "Time of Arrival data", PrinterType::value),
        PRINTER("RCB_RD_EN", "Enable Received Buffer module for reading", PrinterType::enable),
        PRINTER("RCB_DATA", "Received Buffer data", PrinterType::value),
        /* RAM registers (r/w) */
        PRINTER("BPM_ID", "BPM ID for sending packets", PrinterType::value),
        PRINTER("TIME_FRAME_LEN", "Time frame length in clock cycles", PrinterType::value),
        PRINTER("MGT_POWERDOWN", "Transceiver powerdown", PrinterType::enable),
        PRINTER("MGT_LOOPBACK", "Transceiver loopback", PrinterType::enable),
        PRINTER("TIME_FRAME_DLY", "Time frame delay", PrinterType::value),
        PRINTER("RX_POLARITY", "Receiver polarity", PrinterType::value),
        PRINTER("PAYLOAD_SEL", "Payload selection", PrinterType::value),
        PRINTER("FOFB_DATA_SEL", "FOFB data selection", PrinterType::value),
        /* RAM registers (ro) */
        PRINTER("FIRMWARE_VER", "Firmware version", PrinterType::value_hex),
        PRINTER("SYS_STATUS", "System status", PrinterType::value),
        PRINTER("LINK_UP", "Link status", PrinterType::enable),
        PRINTER("TIME_FRAME_CNT", "Total time frame count", PrinterType::value),
        PRINTER("FOD_PROCESS_TIME", "Forward or Discard process time", PrinterType::value),
        PRINTER("BPM_CNT", "BPM devices count", PrinterType::value),
        /* RAM registers (ro) - channels */
        PRINTER("LINK_PARTNER", "Link partner ID", PrinterType::value),
        PRINTER("HARD_ERR_CNT", "Hard error count", PrinterType::value),
        PRINTER("SOFT_ERR_CNT", "Soft error count", PrinterType::value),
        PRINTER("FRAME_ERR_CNT", "Frame error count", PrinterType::value),
        PRINTER("RX_PCK_CNT", "Received packet count", PrinterType::value),
        PRINTER("TX_PCK_CNT", "Transmitted packet count", PrinterType::value),
    }),
    regs_storage(new struct fofb_cc_regs),
    regs(*regs_storage)
{
    read_size = sizeof regs;
    read_dest = &regs;

    device_match = device_match_fofb_cc;
}
```

```
void Core::decode()
{
    fixed_point = extract_value<uint32_t>(regs.fixed_point_pos, WB_FOFB_PROCESSING_REGS_FIXED_POINT_POS_VAL_MASK);
    add_general("FIXED_POINT_POS", fixed_point);

    data_order_done = true;

    /* we don't use number_of_channels here,
     * because there aren't 'normal' per channel variables */
    coefficients.resize(MAX_NUM_CHAN);

    auto convert_fixed_point = [this](uint32_t value) -> float {
        return (double)(int32_t)value / (1 << fixed_point);
    };

    /* finish conversion */
    unsigned i;
    auto add_bank = [this, &i, &convert_fixed_point](const auto &ram_bank) {
        const size_t elements = sizeof ram_bank/sizeof *ram_bank;
        coefficients[i].resize(elements);

        size_t u = 0;
        std::generate(coefficients[i].begin(), coefficients[i].end(),
            [&](){ return convert_fixed_point(ram_bank[u++].data); });
    };

    for (i = 0; i < MAX_NUM_CHAN; i++) switch (i) {
        case 0: add_bank(regs.ram_bank_0); break;
        case 1: add_bank(regs.ram_bank_1); break;
        case 2: add_bank(regs.ram_bank_2); break;
        case 3: add_bank(regs.ram_bank_3); break;
        case 4: add_bank(regs.ram_bank_4); break;
        case 5: add_bank(regs.ram_bank_5); break;
        case 6: add_bank(regs.ram_bank_6); break;
        case 7: add_bank(regs.ram_bank_7); break;
        case 8: add_bank(regs.ram_bank_8); break;
        case 9: add_bank(regs.ram_bank_9); break;
        case 10: add_bank(regs.ram_bank_10); break;
        case 11: add_bank(regs.ram_bank_11); break;
        default: throw std::logic_error("there are only 12 RAM banks");
    }
}
```

```
void ControllerV2::encode_config()
{
    static const std::unordered_map<std::string_view, int> mode_options({
        {"open-loop-dac", 0},
        {"open-loop-square", 1},
        {"closed-loop-pi_sp", 2},
        {"closed-loop-square", 3},
        {"closed-loop-external", 4},
    });

    int mode_option;
    if (mode_numeric) mode_option = *mode_numeric;
    else try {
        mode_option = mode_options.at(mode);
    } catch (std::out_of_range &e) {
        throw std::runtime_error("mode must be one of " + list_of_keys(mode_options));
    }

    if (channel > NUM_CHAN-1)
        throw std::runtime_error("there are only 12 channels");

    bar4_read_v(&bars, addr + WB_RTMLAMP_OHWR_REGS_CH + channel * CHANNEL_DISTANCE, channel_regs.get(), CHANNEL_DISTANCE);

    clear_and_insert(channel_regs->ctl, mode_option, WB_RTMLAMP_OHWR_REGS_CH_CTL_MODE_MASK);
    insert_bit(channel_regs->ctl, amp_enable, WB_RTMLAMP_OHWR_REGS_CH_CTL_AMP_EN);

    if (trigger_enable) {
        if (devinfo.abi_ver_minor >= TRIGGER_ENABLE_VERSION)
            insert_bit(channel_regs->ctl, *trigger_enable, WB_RTMLAMP_OHWR_REGS_CH_CTL_TRIG_EN);
        else if (*trigger_enable)
            throw std::runtime_error("this core doesn't support trigger_enable");
    }

    if (pi_kp) clear_and_insert(channel_regs->pi_kp, *pi_kp, WB_RTMLAMP_OHWR_REGS_CH_PI_KP_DATA_MASK);
    if (pi_ti) clear_and_insert(channel_regs->pi_ti, *pi_ti, WB_RTMLAMP_OHWR_REGS_CH_PI_TI_DATA_MASK);
    if (pi_sp) clear_and_insert(channel_regs->pi_sp, (uint16_t)*pi_sp, WB_RTMLAMP_OHWR_REGS_CH_PI_SP_DATA_MASK);

    if (dac) clear_and_insert(channel_regs->dac, (uint16_t)*dac, WB_RTMLAMP_OHWR_REGS_CH_DAC_DATA_MASK);

    if (limit_a) clear_and_insert(channel_regs->lim, (uint16_t)*limit_a, WB_RTMLAMP_OHWR_REGS_CH_LIM_A_MASK);
    if (limit_b) clear_and_insert(channel_regs->lim, (uint16_t)*limit_b, WB_RTMLAMP_OHWR_REGS_CH_LIM_B_MASK);

    if (cnt) clear_and_insert(channel_regs->cnt, *cnt, WB_RTMLAMP_OHWR_REGS_CH_CNT_DATA_MASK);
}
```

µHAL

- Decoders have the added advantage of automatically generating a register field printer. Calling the class's **print()** method in a command line utility is enough.

```
if (mode == "decode") {
    auto type = args.get<std::string>("-q");
    std::unique_ptr<RegisterDecoder> dec;
    if (type == "acq") {
        dec = std::make_unique<LnlsBpmAcqCore>(bars);
    } else if (type == "lamp") {
        dec = std::make_unique<lamp::CoreV1>(bars);
        /* if v1 can't be found, try v2;
         * assumes only one version of the device will be available */
        if (!read_sdb(&bars, dec->device_match, dev_index)) {
            dec = std::make_unique<lamp::CoreV2>(bars);
        }
    } else if (type == "fofb_processing") {
        dec = std::make_unique<LnlsFofbProcessing>(bars);
    } else {
        fprintf(stderr, "Unknown type: '%s'\n", type.c_str());
        return 1;
    }

    dec->channel = args.present<unsigned>("-c");

    if (auto d = read_sdb(&bars, dec->device_match, dev_index)) {
        if (verbose) {
            fprintf(stdout, "Found device in %08jx\n", (uintmax_t)d->start_addr);
        }
        dec->set_devinfo(*d);
        dec->read();
        dec->print(stdout, verbose);
    }
}
```

μHAL – Implementation

- Depends on **Linux** specific functionality

- Find the device for a **slot** via the */sys/bus/pci/slots/<number>/address* file
- **Memory map** each BAR via */sys/bus/pci/devices/<device>/resource** files

- Currently hard-codes the addressing scheme used by LNLS PCIe devices:
  - Paging registers in BAR0 for BAR2 (RAM – acquisition data) and BAR4 (Wishbone – registers)
  - Weird strides in BAR4 addressing

New Design – Trade-offs

- As any design choice does, this new architecture deals with some trade-offs

- We can’t use DMA at the moment, since the software is userspace only. This has been partially worked around with a custom data copy function using SSE 4.2 intrinsics
- There can only be one user of the hardware via this library at a time, as the hardware access isn’t thread safe (paging register). This isn’t a deal breaker because the hardware functionality is exposed via the EPICS IOC (or whichever other control system is used)

(New) EPICS IOC

- Switch to modular architecture for each FPGA module
  - One Asyn port for each module
  - Can simply instantiate each port as needed
  - The eventual plan is to perform dynamic instantiation
- A single IOC source code repository for all our MicroTCA devices
- Overall decrease in code size (LoC for a given module in µHAL + IOC are half the LoC for the same module in HALCS)

- The expectation is that as a new module is developed, its EPICS module will be developed in tandem

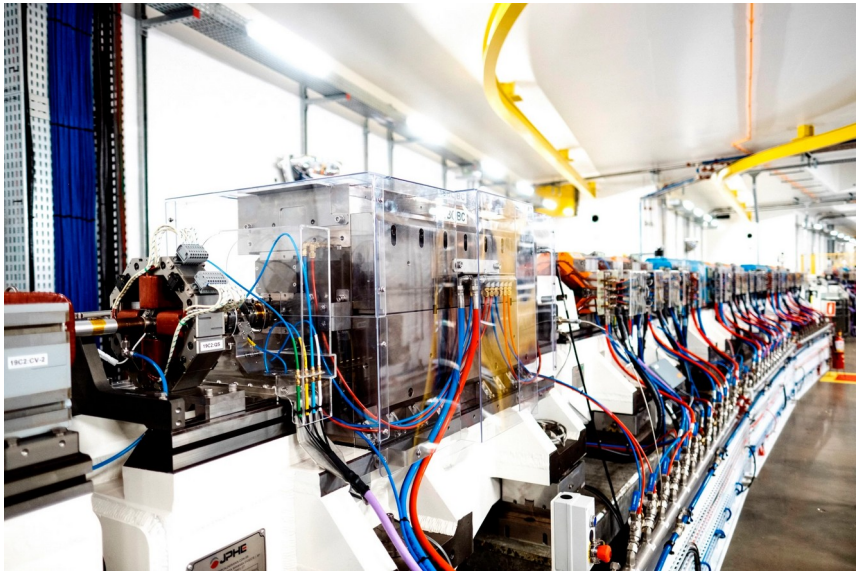
Next steps

- Consider integration with Linux VFIO module
  - Still in userspace, but DMA-capable
- Implement missing modules for a full deployment with our FOFB boards
- Full deployment for Timing and BPM boards

Thank you!

µHAL: https://github.com/lnls-dig/uhal
IOC: https://github.com/lnls-dig/erics

Contact: erico.rolim@lnls.br



Backup slides

Self Describing Bus

```
0000000000000651:e6a542c9 @ 00000000-0fffffff WB4-Crossbar-GSI
0000000000000651:eef0b198 @ 01000000-01001fff WB4-Bridge-GSI
0000000000000651:e6a542c9 @ 01000000-01001fff     WB4-Crossbar-GSI
0000000000000651:aa7bfb3c @ 01000000-01000000     WB4-MSI-Bridge-GSI
1000000000001215:af1a1c3e @ 01000000-010000ff     LNLS_AFC_BASE_REGS
0000000000000651:eef0b198 @ 01001000-01001fff     WB4-Bridge-GSI
0000000000000651:e6a542c9 @ 01001000-010017ff         WB4-Crossbar-GSI
000000000000ce42:8a5719ae @ 01001000-010010ff         CERN_SIMPLE_UART
0000000000000651:35aa6b95 @ 01001100-010011ff         GSI_GPIO_32
0000000000000651:35aa6b95 @ 01001200-010012ff         GSI_GPIO_32
000000000000ce42:fdafb9dd @ 01001300-0100130f         CERN_TICS_COUNTER
000000000000ce42:123c5443 @ 01000100-010001ff     WB-I2C-Master
000000000000ce42:00000013 @ 01000200-010002ff     WB-VIC-Int.Control
1000000000001215:51954750 @ 01000300-010003ff     LNLS_AFCDIAG
1000000000001215:bcbb78d2 @ 01000400-010007ff     LNLS_TRIGGER_IFACE
0000000000000651:eef0b198 @ 08000000-0fffffff WB4-Bridge-GSI
0000000000000651:e6a542c9 @ 08000000-0803ffff     WB4-Crossbar-GSI
1000000000001215:4519a0ad @ 08002000-08002fff     LNLS_BPM_ACQ_CORE
1000000000001215:4519a0ad @ 08003000-08003fff     LNLS_BPM_ACQ_CORE
1000000000001215:4519a0ad @ 08004000-08004fff     LNLS_BPM_ACQ_CORE
1000000000001215:84b6a5ac @ 08000000-080003ff     LNLS_TRIGGER_MUX
1000000000001215:84b6a5ac @ 08000400-080007ff     LNLS_TRIGGER_MUX
1000000000001215:84b6a5ac @ 08000800-08000bff     LNLS_TRIGGER_MUX
1000000000000d15:4a1df147 @ 08010000-0801ffff     DLS_DCC_REGS
1000000000000d15:4a1df147 @ 08020000-0802ffff     DLS_DCC_REGS
1000000000001215:a1248bec @ 08005000-08005fff     LNLS_RTM_LAMP_REGS
1000000000001215:49681ca6 @ 08008000-0800ffff     FOFB_PROC_REGS
```

Other developments

- ChimeraTK: not as flexible for what we wanted, didn’t address the needs of our acquisition modules
- Most other solutions use some sort of kernel driver, which is a kind of dependency we want to avoid

µHAL

app	decode-reg: error out if decode query isn't found.	2 months ago
etc	Add udev rule for devices.	3 months ago
modules	util: use more descriptive macro for printer.	2 months ago
subprojects	Replace docopt with argparse.	3 months ago
third_party	Replace docopt with argparse.	3 months ago
util	util: use more descriptive macro for printer.	2 months ago
.gitmodules	Replace docopt with argparse.	3 months ago
LICENSE	Add license information.	10 months ago
README.md	Rename project to µHAL.	3 months ago
meson.build	Add pcie_opt meson option.	2 months ago
meson_options.txt	Add pcie_opt meson option.	2 months ago