X2O ATCA IPMC and control solution: Update

D. Acosta, M. Bachtis, D. Campos, A. Greshilov, A. Jelisijevic,
E. Juska, J. Konigsberg, A. Madorsky, V. Rekovic, A. Tan

2023-10-03

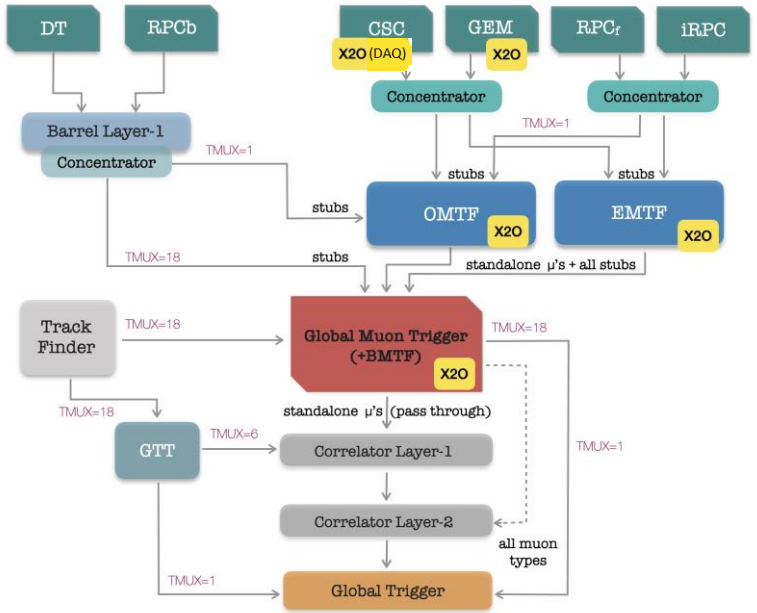
Hardware overview

- ATCA form factor.
- Currently using VU13P-2 A2577 FPGA.
  - Dual KU15P modules available.
- Platform can be easily adapted to other FPGAs.
- Supports 120 optical links up to 25 Gb/s. Boards fully loaded with optics support future extensions of the system and flexibility with SLR routing.
  - Backward-compatible with cheaper 10Gbps QSFP modules.
- Sufficient signal integrity in both the electrical and optical domains, BER much better than $10^{-12.}$
- Optics provide sufficient optical margin with a receiver sensitivity better than -6 dBm to ensure operability at end of life (as laser degrades).
- System management through an on-board linux system. Use the Xilinx Kria SOM in all the boards.
- IPMC fully conforming to IPMI protocol, running on the same Kria SOM. No dedicated IPMC hardware.
- Connectivity required for TCDS2 is available.
- Software and firmware framework for board management and subsystem development.

X2O for Phase-2

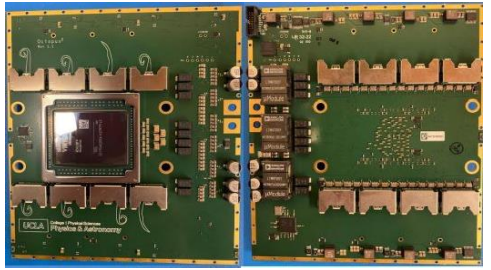
The control system based on X2O platform will be applied for upcoming Phase-2 Upgrade of L1 Trigger subsystems:

- EMTF
- OMTF
- GEM
- GMT
- CSC (DAQ system upgrade)



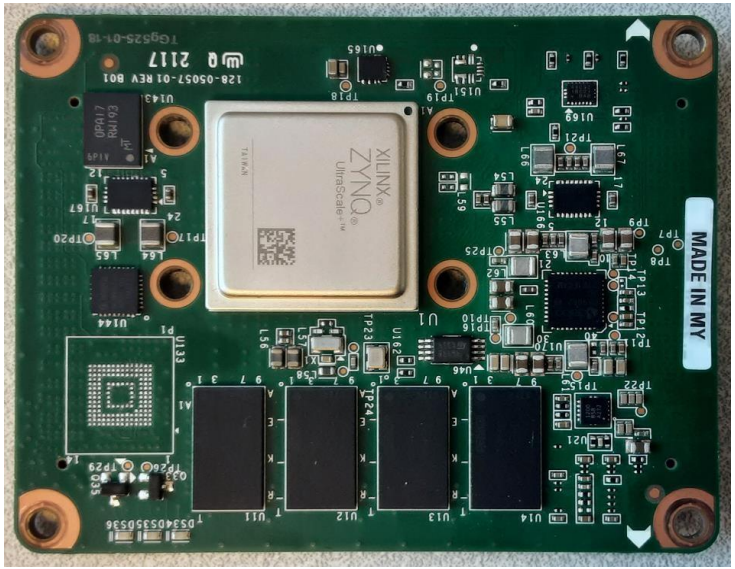
X2O platform update

- **X2O pilot production received.**
- **Octopus FPGA module rev 2:**
  - Halogen-free
  - 1x VU13P
  - New TI clock synthesizer - tested by EP-ESE
  - Improved safety and interlock system with a lattice small FPGA
- **Power module with Kria rev 3:**
  - SD3.0 (clock running at 200 MHz)
  - 10GbE (sustained speed about 5.14 Gbps)
  - DMA-JTAG chain (fast bitstream uploading to FPGA: 20 Mb/sec sustained).
  - Custom I2C core fw module (I2C master for IPMB transfers)
  - IPMC running as an application on Kria



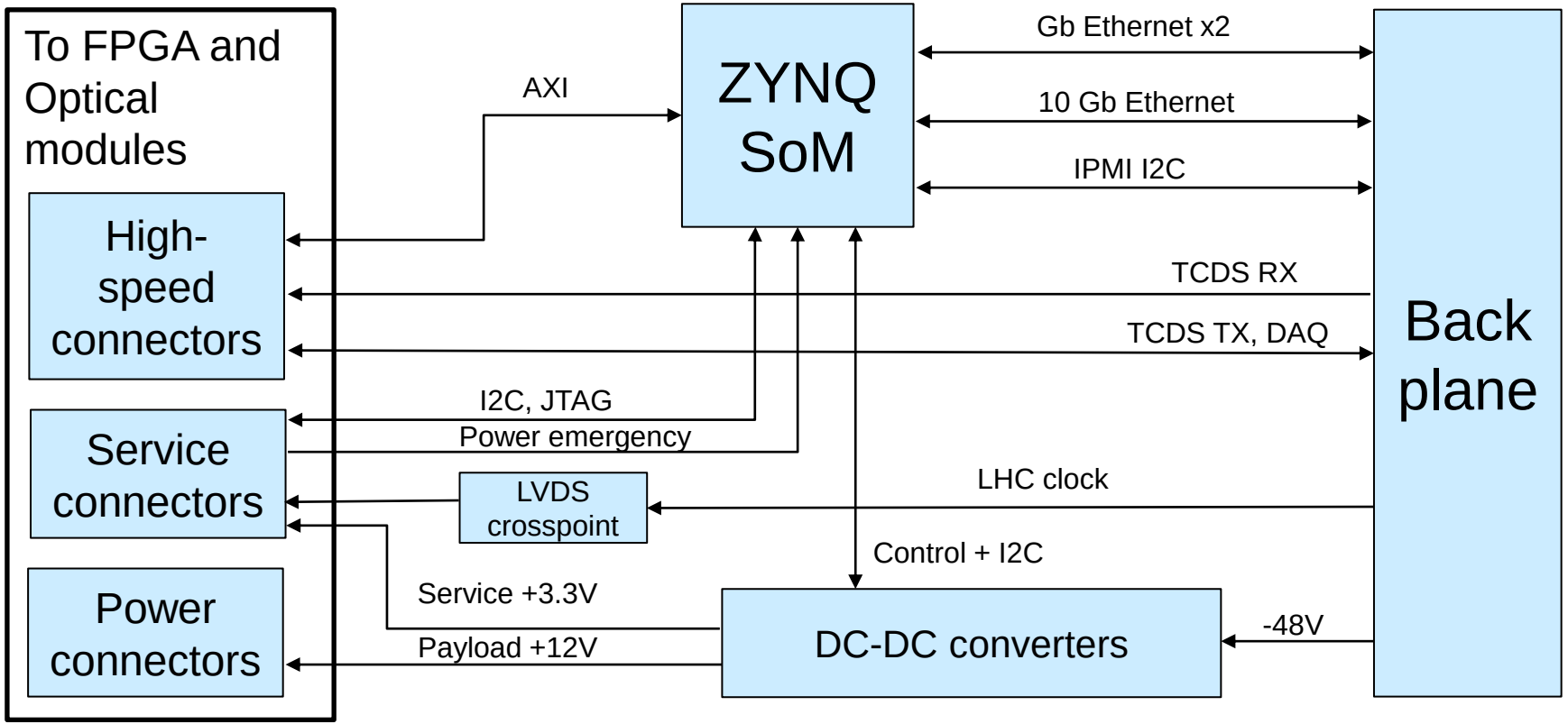
KRIA for X2O

Power module rev 3:

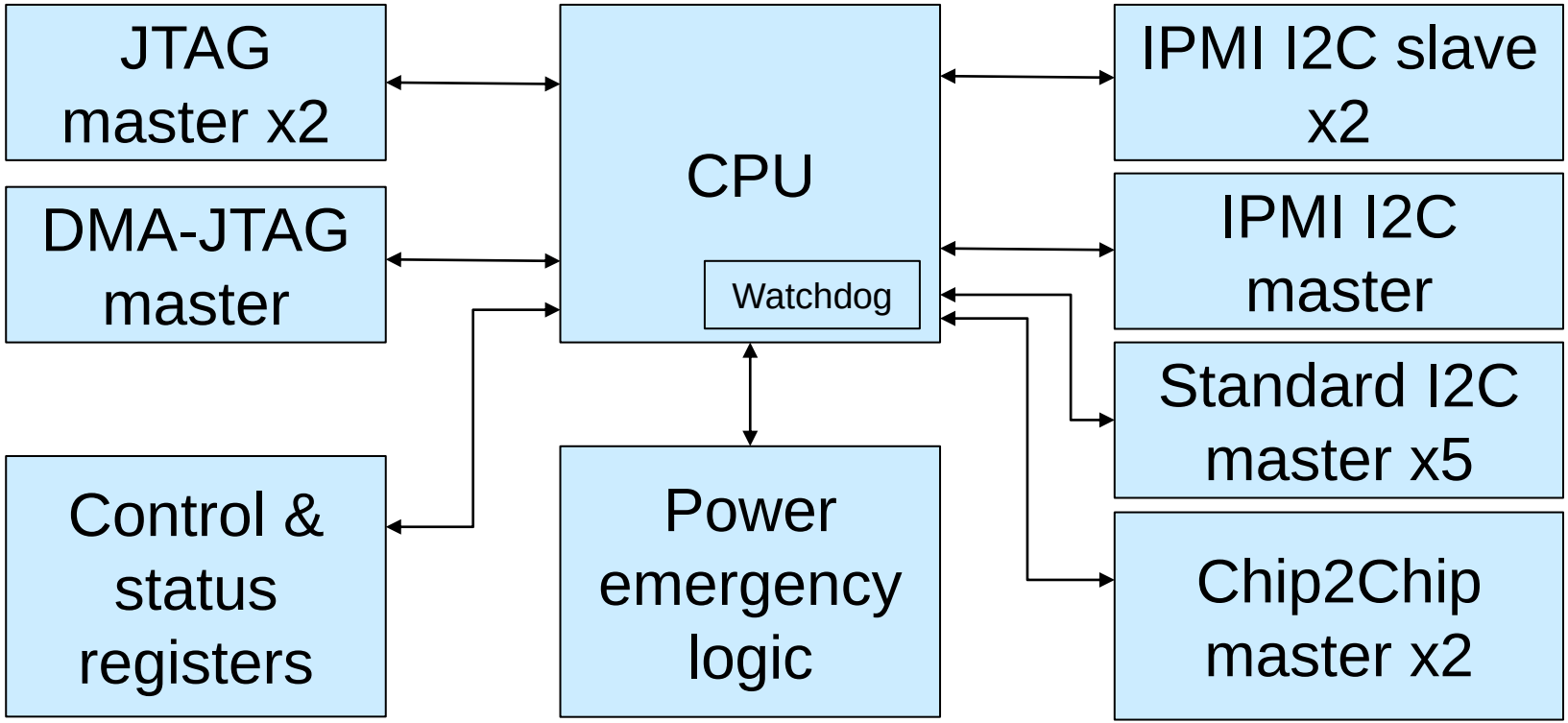


- Upgrade ZYNQ module to US+ family.
- Selected Xilinx Kria K26 SoM as optimal candidate:
  - Low cost: $300
  - 4 GB RAM
  - 4 GTH links 12.5 Gbps
- Faster AXI links to FPGA modules:
  - 7.8125 Gbps
  - Max bit rate using CPLL
  - Both QPLLs in quad are needed for TCDS2
  - Rev 2: 3.75 Gbps max
- 10G Ethernet connection:
  - In addition to 2x1G

X2O Power Module block schematics



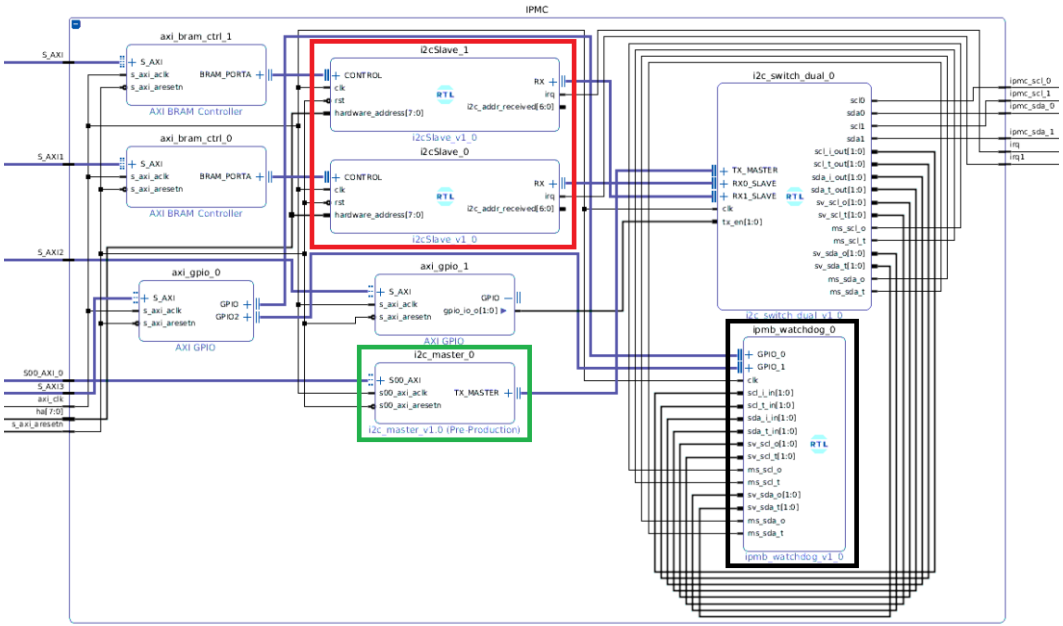
X2O Power Module ZYNQ firmware



X2O Power Module firmware[1] modules

IP Module	Description
IPMI I2C slave	Customized I2C slave receiver capable of buffering long transfers. Does not pose any timing requirements to IPMC software.
IPMI I2C master	Customized I2C master transmitter on common multimaster IPMB-0 bus that manages the IPMI transfers from IPMC to ATCA SHELF.
Standard I2C master	Standard AXI I2C master modules used to control I2C devices on Power, FPGA, and Optical modules.
Chip2Chip master	Regular Chip2Chip master, used to extend AXI bus to FPGA modules.
JTAG master	JTAG masters are used for firmware downloading and Xilinx Virtual Cable (XVC) debugging of the FPGA modules.
DMA-JTAG master	DMA-JTAG master is used for fast firmware downloading.
Power emergency logic	Payload modules can signal emergency conditions (such as overvoltage) using dedicated lines. Emergency logic then shuts down Payload power in 10 ns or less.
Watchdog	The dedicated watchdog hard IP is used to monitor the health of the IPMC software. If IPMC software stops polling the watchdog, the watchdog resets the CPU and shuts down the Payload power.

UF IPMC FW



- Custom I2C receiver (slave) modules
- Operate independently of CPU
- No real-time software handling needed

- Custom I2C transmitter (master) module
- Glitch filter
- Arbitration for multi-master IPMB-0 bus

+ IPMB watchdog

UF IPMC SW

Historical reference: UF IPMC project started in 2020 for ZYNQ 7000 (32-bit ARM).

UF IPMC[2] is an Intelligent Platform Management Controller (IPMC) that resides on ATCA boards (X2O) in an embedded ZYNQ[3] device (KRIA). UF IPMC is responsible for the communication with Shelf Manager.

- **Built 64-bit ARM Embedded Linux system for KRIA module within the X2O boards:**
  - **Petalinux kernel version 2022.2.2.**
  - **CentOS 8.**
- **Main points:**
  - **Based on the IPMI & PICMG cores of the coreIPM open-source project[4].**
  - **Provides full basic functionality[5] for sensor monitoring.**
  - **Provides easy customization of USER sensors:**
    - **Successful integration with Octopus user module (UCLA for X2O rev.2).**
    - **Successful integration with Octopus user module (UCLA for X2O rev.3).**
  - **Supports the use of various sensor readout interfaces, including I2C, System Monitor.**
- **Current status:**
  - **(done) UF IPMC sw version for VU13P with QSFP-DD (rev.2).**
  - **(done) UF IPMC sw version for VU13P with QSFP 30-cage module (rev.3).**

NOTE: No separate IPMC hardware module is needed.

Normal CPU usage of UF IPMC process is 1-3%.

```
90: FRU # 0
    Entity: (0xa0, 0x60)
    Hot Swap State: M4 (Active), Previous: M7 (Commu
    Device ID String: "UF IPMC"
    Site Type: 0x00, Site Number: 11
    Current Power Level: 0x02, Maximum Power Level:

# clia sensordata 90 5

Pigeon Point Shelf Manager Command Line Interpreter

90: LUN: 0, Sensor # 5 ("T:QSFP MAX")
    Type: Threshold (0x01), "Temperature" (0x01)
    Belongs to entity (0xa0, 0x60): FRU # 0
    Status: 0xc0
        All event messages enabled from this sensor
        Sensor scanning enabled
        Initial update completed
    Raw data: 0 (0x00)
    Processed data: 0.000000 degrees C
    Current State Mask: 0x00

# clia sensordata 90 9

Pigeon Point Shelf Manager Command Line Interpreter

90: LUN: 0, Sensor # 9 ("T:FPGA")
    Type: Threshold (0x01), "Temperature" (0x01)
    Belongs to entity (0xa0, 0x60): FRU # 0
    Status: 0xc0
        All event messages enabled from this sensor
        Sensor scanning enabled
        Initial update completed
    Raw data: 32 (0x20)
    Processed data: 32.000000 degrees C
    Current State Mask: 0x00

# clia sensordata 90 10

Pigeon Point Shelf Manager Command Line Interpreter

90: LUN: 0, Sensor # 10 ("T:LTM4638 MAX")
    Type: Threshold (0x01), "Temperature" (0x01)
    Belongs to entity (0xa0, 0x60): FRU # 0
    Status: 0xc0
        All event messages enabled from this sensor
        Sensor scanning enabled
        Initial update completed
    Raw data: 44 (0x2c)
    Processed data: 44.000000 degrees C
    Current State Mask: 0x00
```

ATCA shelf manager recognizing some of our temperature sensors (rev.3)

IPMC user code implementation (rev.3)

- **TEMPLATES are provided.**
- **UF IPMC Manual is provided.**
- **Separate USER code space (for easy update).**
- **Aarch64 RPMs build within the CERN Gitlab CI flow.**
- **X2O EOS storage for distribs[6].**

Power ON

```
void user_module_payload_on( void )
{
  lock(1);

  std::string config = "/usr/local/share/octopus.toml";
  int verbose = 1;
  octopus oct(&octo_bus);
  oct.configure(true, config);

  unsigned int payload_rw;
  unsigned int val_plrty_bot;
  unsigned int val_plrty_top;
  unsigned int val_plrty_qsfp;
  unsigned int val_pok_en_bot;
  unsigned int val_pok_en_top;
  unsigned int val_pok_en_qsfp;

  val_plrty_bot = (unsigned int) polarity_bot << 20;
  val_plrty_top = (unsigned int) polarity_top << 21;
  val_plrty_qsfp = (unsigned int) polarity_qsfp << 22;

  val_pok_en_bot = (unsigned int) pok_en_bot << 23;
  val_pok_en_top = (unsigned int) pok_en_top << 24;
  val_pok_en_qsfp = (unsigned int) pok_en_qsfp << 25;

  payload_rw = reg_read(devmem_ptr, qbv_on_off);
  payload_rw |= (val_plrty_bot | val_plrty_top | val_plrty_qsfp);
  reg_write(devmem_ptr, qbv_on_off, payload_rw);

  payload_rw = reg_read(devmem_ptr, qbv_on_off);
  payload_rw |= (val_pok_en_bot | val_pok_en_top | val_pok_en_qsfp);
  reg_write(devmem_ptr, qbv_on_off, payload_rw);

  payload_rw = reg_read(devmem_ptr, qbv_on_off);
   payload_rw |= 0x20;
   reg_write(devmem_ptr, qbv_on_off, payload_rw);

  if (prototype)
  {
    usleep(200000);

    val_pok_en_bot = ((unsigned int) pok_en_bot | 0x1) << 23;
    val_pok_en_top = ((unsigned int) pok_en_top | 0x1) << 24;
    val_pok_en_qsfp = ((unsigned int) pok_en_qsfp | 0x1) << 25;

    payload_rw = reg_read(devmem_ptr, qbv_on_off);
    payload_rw |= (val_pok_en_bot | val_pok_en_top | val_pok_en_qsfp);
    reg_write(devmem_ptr, qbv_on_off, payload_rw);
  }

  usleep(200000);

  oct.configure(false, config);
  oct.power_up(verbose);

  payload_timeout_init = lbolt;
  power_up_done = 1;
  logger("PAYLOAD", "On");

  unlock(1);
}
```

Power OFF

```
void
user_module_payload_off( void )
{
  lock(1);

  std::string config = "/usr/local/share/octopus.toml";
  octopus oct(&octo_bus);

  unsigned int payload_rw;
  unsigned int val_pok_en_bot;
  unsigned int val_pok_en_top;
  unsigned int val_pok_en_qsfp;

  oct.configure(false, config);
  oct.power_down();

  val_pok_en_bot = (unsigned int) pok_en_bot << 23;
  val_pok_en_top = (unsigned int) pok_en_top << 24;
  val_pok_en_qsfp = (unsigned int) pok_en_qsfp << 25;

  payload_rw = reg_read(devmem_ptr, qbv_on_off);
  payload_rw |= (val_pok_en_bot | val_pok_en_top | val_pok_en_qsfp);
  reg_write(devmem_ptr, qbv_on_off, payload_rw);

    payload_rw = reg_read(devmem_ptr, qbv_on_off);
    payload_rw &= ~0x20;
    reg_write(devmem_ptr, qbv_on_off, payload_rw);

  power_up_done = 0;
    logger("PAYLOAD", "Off");

  unlock(1);
}
```

Sensor enable

Upper non-recoverable threshold exceeded (Power OFF)

Upper critical threshold exceeded (Fans speed - UP)

Back to normal (Fans speed - DOWN)

Sensor disable

```
/*=============================================================*/
/*      Temp FPGA Sensor                                       */
/*=============================================================*/
void read_sensor_temp_FPGA(void) {
    lock(1);

    //  Sensor Data Record
    u8 sensor_N = 9;

    if (check_power_up()) {
        std::string config = "/usr/local/share/octopus.toml";
        octopus octopus(&octo_bus);
        octopus.configure(false, config);
        monitor = octopus.monitor(0);
        float temp_f = monitor["VIRTEXUPLUS"].t1_remote;

        //  Convert float to byte and get precision
        u8 temp_b = (u8)(temp_f);

        sd[sensor_N].last_sensor_reading = temp_b;
        sd[sensor_N].sensor_scanning_enabled = 1;
        sd[sensor_N].event_messages_enabled = 1;
        sd[sensor_N].unavailable = 0;

        static int first_time = 1;
        static int up_noncrt_assert = 0;

        if (first_time) {
            first_time = 0;
        } else if (sd[sensor_N].last_sensor_reading >= sdr[sensor_N].upper_non_recoverable_threshold) {
            // Transition to M6 for non-recoverable
            sd[sensor_N].current_state_mask = 0xE0;
            unlock(1);
            picmg_m6_state(fru_inventory_cache[0].fru_dev_id);
            logger("WARNING","Non-recoverable threshold crossed for FPGA temperature sensor");
            lock(1);
        } else if (up_noncrt_assert == 0 && sd[sensor_N].last_sensor_reading >= sdr[sensor_N].upper_critical_threshold) {
            sd[sensor_N].current_state_mask = 0xD0;
            // Assertion message for shelf manager
            FRU_TEMPERATURE_EVENT_MSG_REQ msg;
            msg.command = 0x02;
            msg.evt_msg_rev = 0x04;
            msg.sensor_type = 0x01;
            msg.sensor_number = sensor_N;
            msg.evt_direction = 0x01;
            msg.evt_data2_qual = 0x01;
            msg.evt_data3_qual = 0x01;
            msg.evt_reason = 0x07;
            msg.temp_reading = sd[sensor_N].last_sensor_reading;
            msg.threshold = sdr[sensor_N].upper_critical_threshold;

            ipmi_send_event_req(( unsigned char * )&msg, sizeof(FRU_TEMPERATURE_EVENT_MSG_REQ), 0);
            up_noncrt_assert = 1;
        } else if (up_noncrt_assert == 1 && sd[sensor_N].last_sensor_reading < sdr[sensor_N].upper_critical_threshold) {
            sd[sensor_N].current_state_mask = 0xC0;
            // Deassertion message for shelf manager
            FRU_TEMPERATURE_EVENT_MSG_REQ msg;
            msg.command = 0x02;
            msg.evt_msg_rev = 0x04;
            msg.sensor_type = 0x01;
            msg.sensor_number = sensor_N;
            msg.evt_direction = 0x01;
            msg.evt_data2_qual = 0x01;
            msg.evt_data3_qual = 0x01;
            msg.evt_reason = 0x07;
            msg.temp_reading = sd[sensor_N].last_sensor_reading;
            msg.threshold = sdr[sensor_N].upper_critical_threshold;

            ipmi_send_event_req(( unsigned char * )&msg, sizeof(FRU_TEMPERATURE_EVENT_MSG_REQ), 0);
            up_noncrt_assert = 0;
        }
    } else {
        sd[sensor_N].last_sensor_reading = 0;
        sd[sensor_N].sensor_scanning_enabled = 0;
        sd[sensor_N].event_messages_enabled = 0;
        sd[sensor_N].unavailable = 1;
        sd[sensor_N].current_state_mask = 0;
    }
    unlock(1);
}
```

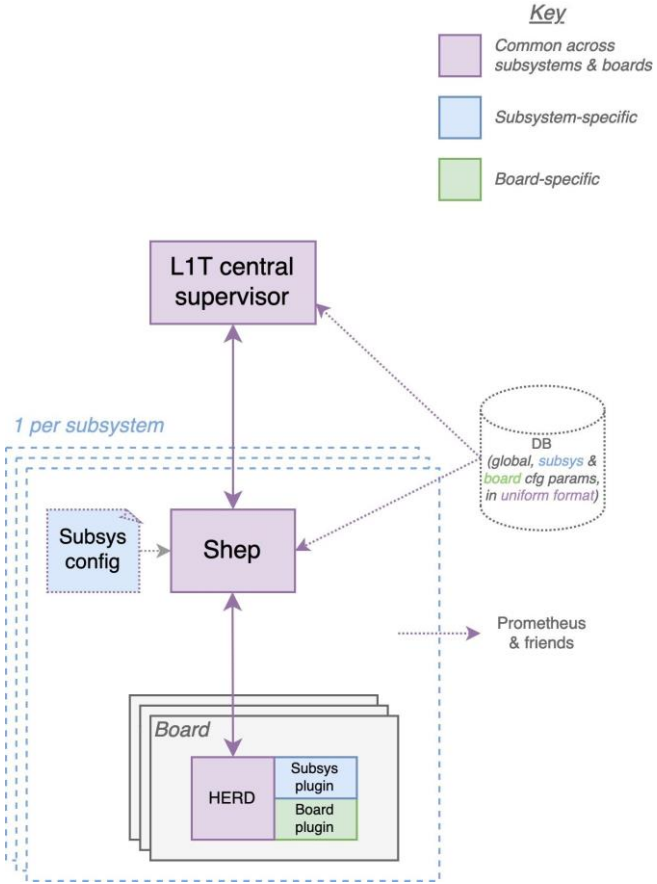
Reaction on Power emergency

- Quick payload shutdown required in case of power emergency, specifically overvoltage.
- Dedicated Power emergency signal is used for shutting down the payload power via firmware logic:
  - No CPU participation.
  - Reaction time as fast as a few nanoseconds.
- IPMC eventually lets the Shelf Manager know that a failure took place, but there is no software timing limitation.
- Shutting down the payload power as a result of the sensor readout via I2C (or another interface) is also possible:
  - Reaction time much longer than Power emergency logic in firmware.
  - Used for thermal shutdown.

Control solution

High-level design (proposed by phase-2 online SW group* (K. Lannon, T. Williams, A. Akpinar, D. Gastler, R. Knowlton, A. Mitra, D. Monk, J. Sweet):

- **HERD:**
  - **On-board application providing run control & monitoring interface (1 per board).**
  - **Builds on SWATCH library from phase-1 L1T.**
- **SHEP (short for shepherd):**
  - **Off-board application — each instance supervises a single subsystem.**

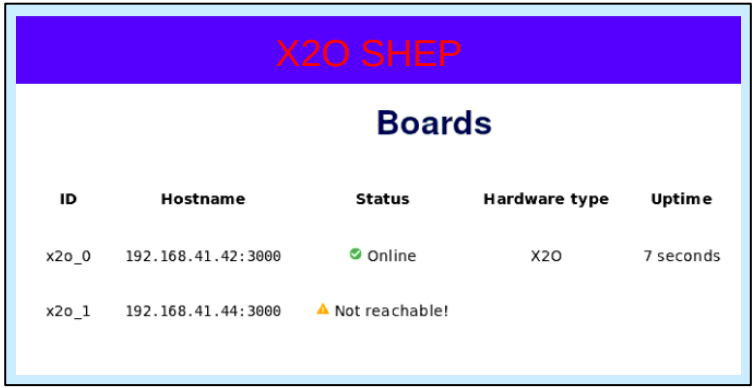


*https://indico.cern.ch/event/1099319/contributions/4625725/attachments/2359823/4027959/L1TPhase2OnlineSoftware_20211206.pdf (see from slide 13)

X2O HERD

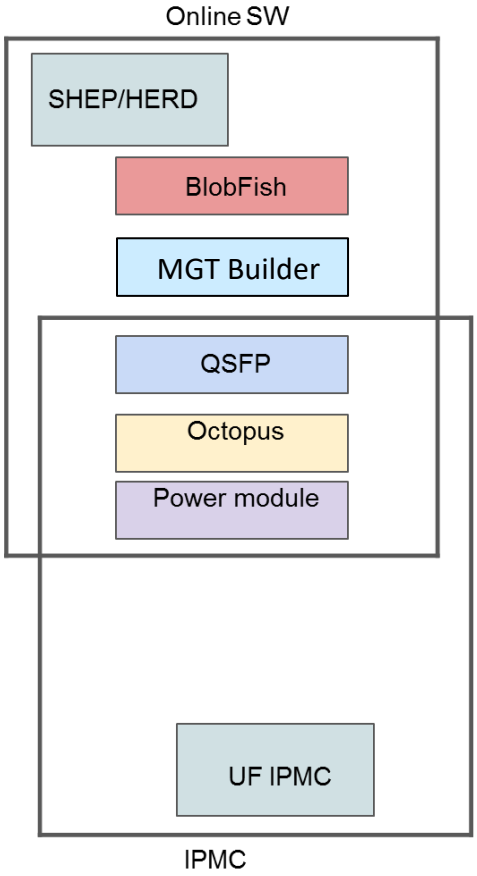
Initial version of X2O HERD plugin[7] :

- **MGT configurator (tested for VU13P).**
- **DMA-JTAG chain for fast FPGA programming (tested for KU15P, VU13P):**
  - **Bitstream decoder**
  - **JTAG FW Programmer**
  - **DMA-proxy driver**
  - **DMA-JTAG fw core**
- **Utilities (I2C tool, devmem tool, semaphores, config parsers).**
- **Octopus module (tested for VU13P).**
- **(in progress) BlobFish fw module support (for VU13P).**



Online software and IPMC

- **Unified software approach:**
  - Build C++ modules for all objects and integrate them both in online SW and IPMC
- **Recent work:**
  - New Octopus C++ library translated from python:
    - Power on/off, monitoring, clock configuration.
  - Power module software:
    - Fast FPGA config, payload control and monitoring , XVC server.
  - Optical module:
    - Monitoring and configuration of optics.
  - New version of IPMC with Octopus and QSFP optical module ready.
  - C++ library for BlobFish control:
    - Ready to be integrated.



MGT builder purposes

MGT builder[8] is a set of software tools and firmware modules.

Main functionality:

- Automatic generation of firmware structure for complex FPGA designs with MGTs.
- Support for arbitrary MGT configurations, including:
  - Different bit rates and encodings in RX and TX parts of the same MGT.
  - Using CPLL and QPLL as needed for each MGT.
  - Automatic routing and assignment of reference clocks.
  - Grouping MGTs into interfaces with friendly names and indexes.
  - Automatic generation of all constraints.
- Configuration via Excel spreadsheets.
- Easy and fast reworking of MGT locations and protocols .
- Used in current EMTF system at P5 since 2020.
- Design for Phase-II EMTF is ready.
- Used in both PC-based and embedded systems.

MGT Builder: generated firmware

- Firmware is generated in SystemVerilog.
- EMTF top level module is shown as an example.
- All serial links are grouped into interfaces:
  - As prescribed by source file.
- DRP bus is connected to all MGTs and COMMON modules.
- MGT and COMMON modules' ports that don't require dynamic signaling are also accessible via DRP-like interface.
- TX clocks are grouped according to MGT locations.
- Generated firmware is based on bare MGT library primitives:
  - Does not contain Transceiver Wizard Ips.
  - Uses absolute minimum of FPGA logic resources.

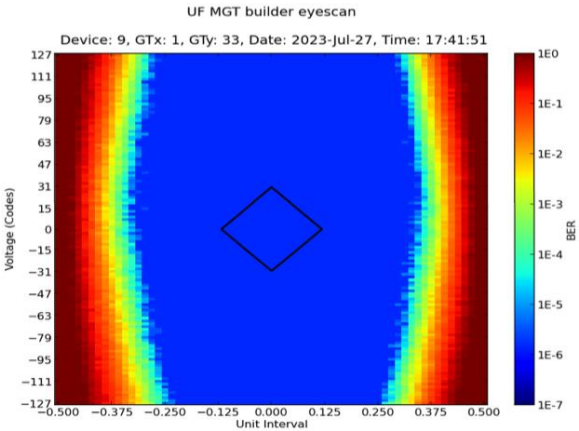
Top-level module header from generated EMTF firmware

```
module emtf_serial_io
(
    drp.in drp_fif,
    input drpclk,
    input [15:0] refclk_p,
    input [15:0] refclk_n,
    mgt_rx.in mpc2_rx [7:0],
    mgt_rx.in ge11_rx [6:0],
    mgt_rx.in mpc3_rx [7:0],
    mgt_rx.in mpc4_rx [7:0],
    mgt_rx.in cppf_rx [6:0],
    mgt_rx.in mpc1_rx [7:0],
    mgt_rx.in mpc0_rx [7:0],
    mgt_rx.in mpcn_rx [8:0],
    mgt_tx.in gmt_tx [0:0],
    mgt_tx.in daq_tx [0:0],
    output DAQ_0_mmcm_clk,
    output CPPF_7_mmcm_clk,
    output clk125_nob
);
```

MGT builder: software

- mgt_configurator tool reads the source files.
- Programs all MGTs and COMMON modules with settings according to protocols:
  - All DRP parameters.
  - All ports that need static settings from software.
- Initially, power of all MGTs and COMMON modules is off.
- mgt_configurator enables power according to configuration source:
  - RX, TX, CPLL, QPLL separately, only where needed.
- Executes reset procedure for each COMMON module and MGT.
- Currently, no reset logic in firmware at all:
  - All resets done under software control.
  - Saves FPGA resources.
- Extra functions such as BER eye scans.

Real Eye scan made by MGT builder at P5 EMTF system during data taking



Frequently Asked Questions

The most frequently asked questions about X2O:

- **What happens if Linux and/or IPMC software crashes?**

  ZYNQ provides a built-in hardware watchdog. If IPMC software stops polling the watchdog, the watchdog resets the CPU and shuts down the Payload power.

- **What about SD card reliability?**

  SD card will be used only as bootable system storage. Operations that can potentially corrupt the SD card, such as logging, will be avoided.

  Additionally, we plan to use high-reliability SD cards specifically designed for these purposes:

  https://www.westerndigital.com/products/memory-cards/industrial-microsd#SDSDQAF3-008G-I
  https://www.westerndigital.com/products/memory-cards/sandisk-max-endurance-uhs-i-microsd#SDSQQVR-032G-GN6IA

- **How do we plan to use 10G ethernet connection?**

  10G Ethernet is an optional feature that can be useful when large amounts of data are transferred regularly to/from the device. If needed, the connection details have to be discussed with CMS IT. You would also need a 10G switch in your ATCA chassis.

References

1. X2O firmware project: https://github.com/madorskya/apex
2. UF IPMC: https://gitlab.cern.ch/x2o/UF_IPMC
3. UG1085 Zynq-UltraScale Device Technical Reference Manual: https://docs.xilinx.com/r/en-US/ug1085-zynq-ultrascale-trm/Zynq-UltraScale-Device-Technical-Reference-Manual
4. coreIPM open-source project: http://www.coreipm.com/
5. IPMI specification: https://www.intel.com/content/www/us/en/products/docs/servers/ipmi/ipmi-second-gen-interface-spec-v2-rev1-1.html
6. X2O EOS storage: https://cms-x2o.web.cern.ch/
7. X2O HERD plugin: https://gitlab.cern.ch/cms-cactus/phase2/software/plugins/x2o
8. FPGA MGT Builder: https://github.com/madorskya/mgt_builder
9. FRU info storage specification: https://www.intel.com/content/www/us/en/servers/ipmi/ipmi-platform-mgt-fru-infostorage-def-v1-0-rev-1-3-spec-update.html

Backup

UF IPMC Tests

- **UF IPMC implements all functionality necessary for normal operation of the device in the ATCA chassis. It was tested in three different ways:**
  - **In the CMS-standard ATCA chassis**
  - **In the COMTEL ATCA chassis**
  - **On the Polaris Compliance test stand in CERN**
- **All errors detected by Polaris Compliance test stand are expected due to either implementation features, or non-critical functionality not currently implemented in the UF IPMC.**
- **No unexplained errors have been detected. None of the detected errors preclude UF IPMC from normal operation in ATCA chassis.**

Thanks a lot to CERN EP-ESE team for the assistance with compliance tests!

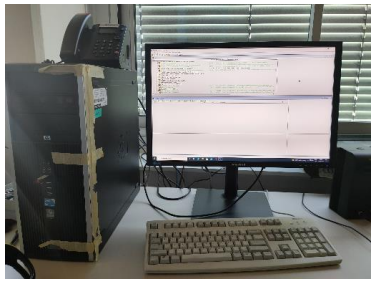
UF IPMC Polaris Compliance test results

UF IPMC tasks:

- Mandatory:
  - 58 (passed)
  - 17 (failed) – not critical
- Optional:
  - 18 (HPM.x) - not implemented
- Debug:
  - 30 (skipped) - not important

ELMA IPMC tasks:

- Mandatory:
  - 42 (passed)
  - 31 (failed)
- Optional:
  - 18 (HPM.x) - not implemented
- Debug:
  - 32 (skipped)

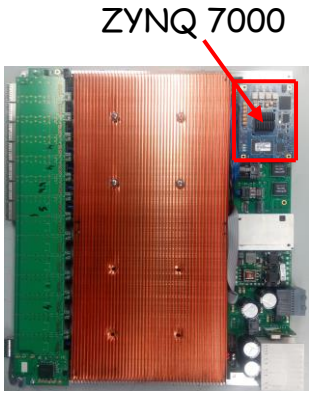


HOST with Polaris tester

IPMI LAN

ATCA Shelf

IPMB



X2O Board

UF IPMC Polaris passed mandatory fields (58 tasks)

All required basic functionality for correct IPMC operation is implemented and tested. The following test have passed:

- IPMC state transition commands (M1, M2, M3, M4, M5, M6)
- Monitoring “Criteria Met” conditions within insertion/extraction procedures
- FRU info commands (mandatory header, can be fully completed depending on the Hardware Platform used)
- SDR commands (implemented with Human-Readable .toml format files as Full Sensor Record Type 01h that could be changed at any time without recompilation of project)
- FRU Hot Swap sensor
- IPMB-0 state sensor
- FRU Handle Switch sensor
- Dummy custom sensors (USERS can easily implement their own custom sensors)
- Sensor monitoring functionality
- Power Management commands
- Event Generation functionality:
  - Hot Swap Event messages within IPMC state transitions
  - Hot Swap Event messages within Abnormal Operation Stage
  - Hot Swap Event messages within IPMB-0 state monitoring
- Power faults handling functionality
- Fans Speed Up reaction to the exceeded thresholds within Temperature sensor implementation (USER defined functionality)
- LED commands (commands are present, but not used. In the current implementation on X2O platform doesn’t need to have this functionality. If needed USERs can add it as required)
- Reset functionality (commands are present, but not used. Currently not required in X2O platform implementation).

UF IPMC Polaris failed mandatory fields (17 tasks)

Task count	Description	Reason for error
1	Get SEL command	Not implemented: Not required because UF IPMC is using regular log files
1	Max FRU Device ID in the Get PICMG Properties response is 0	It's sufficient to use FRU Device ID=0
2	Stopping at intermediate IPMC states	Not allowed in UF IPMC
3	Read FRU Data[8] failed – 8 bytes instead of 15	Expected. Only mandatory header (8 bytes) is implemented in UF IPMC
2	Multirecord Info Area is not present in FRU Information	Not implemented
1	Product Info Area is not present in FRU Information	Not implemented
6	Temperature Event Messages fail: USER defined fields	UF IPMC does not allow changing Temperature sensor records via Shelf Manager commands. They should only be modified by updating corresponding SDR configuration files.
1	Watchdog Timer Commands Support failed	~~Not implemented~~ (Fixed by using ZYNQ built-in watchdog system timer)