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

2023-05-24

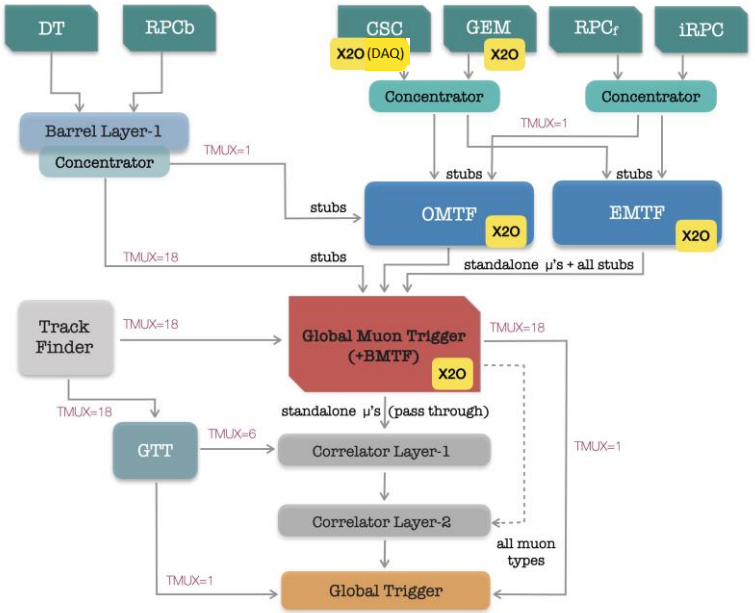
Hardware overview

- 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

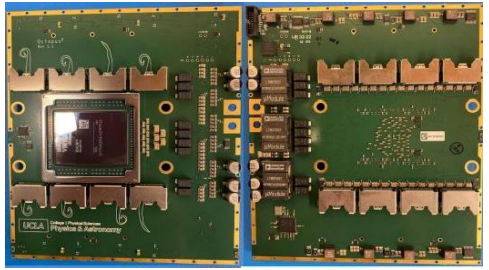
The control system based on X2O platform will be applied for upcoming Phase-2 Upgrade of L1 Trigger subsystems:

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



X2O platform update

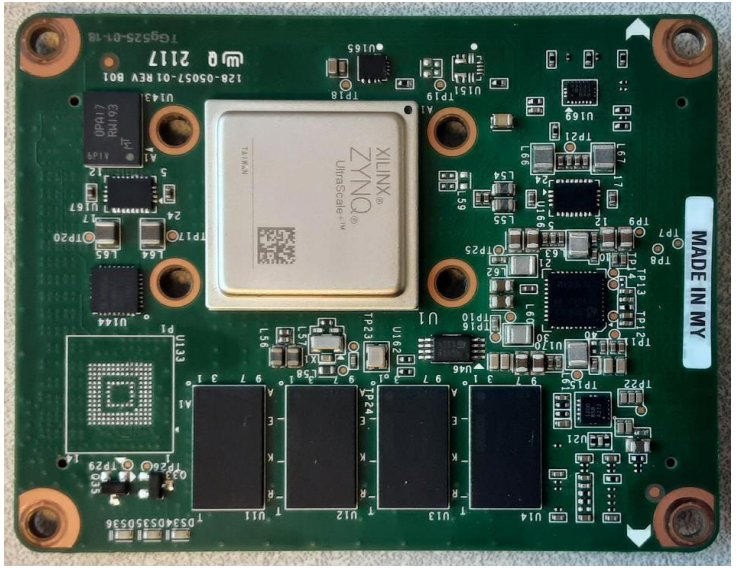
- X2O pilot production received
- Octopus FPGA module rev 2
  - Halogen-free
  - 1x VU13P
  - New TI clock synthesizer - tested by EP-ESE
  - Improved safety and interlock system with a lattice small FPGA
- Power module with Kria rev 3
  - SD3.0 (clock running at 200 MHz)
  - 10GbE (sustained speed about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



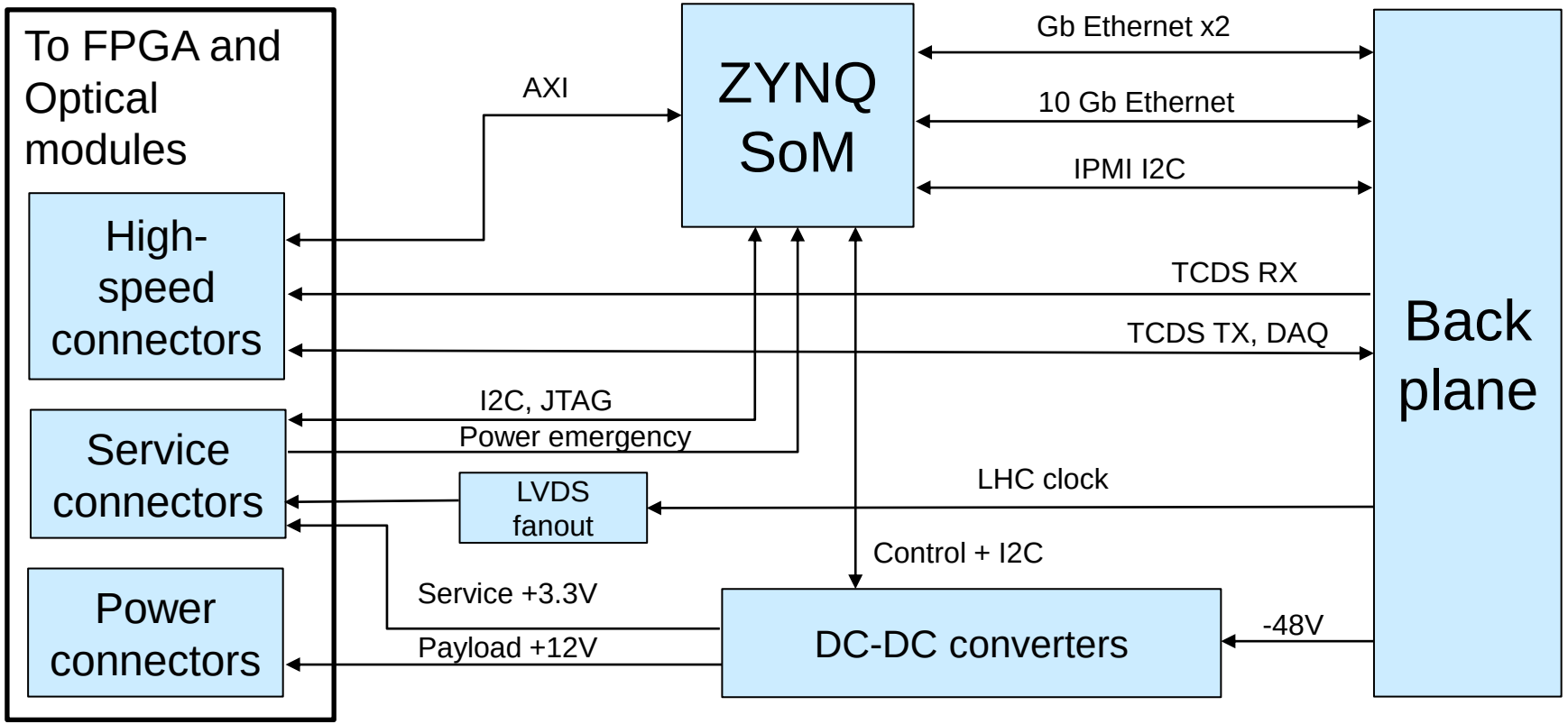
KRIA for X2O

- Power module rev 3

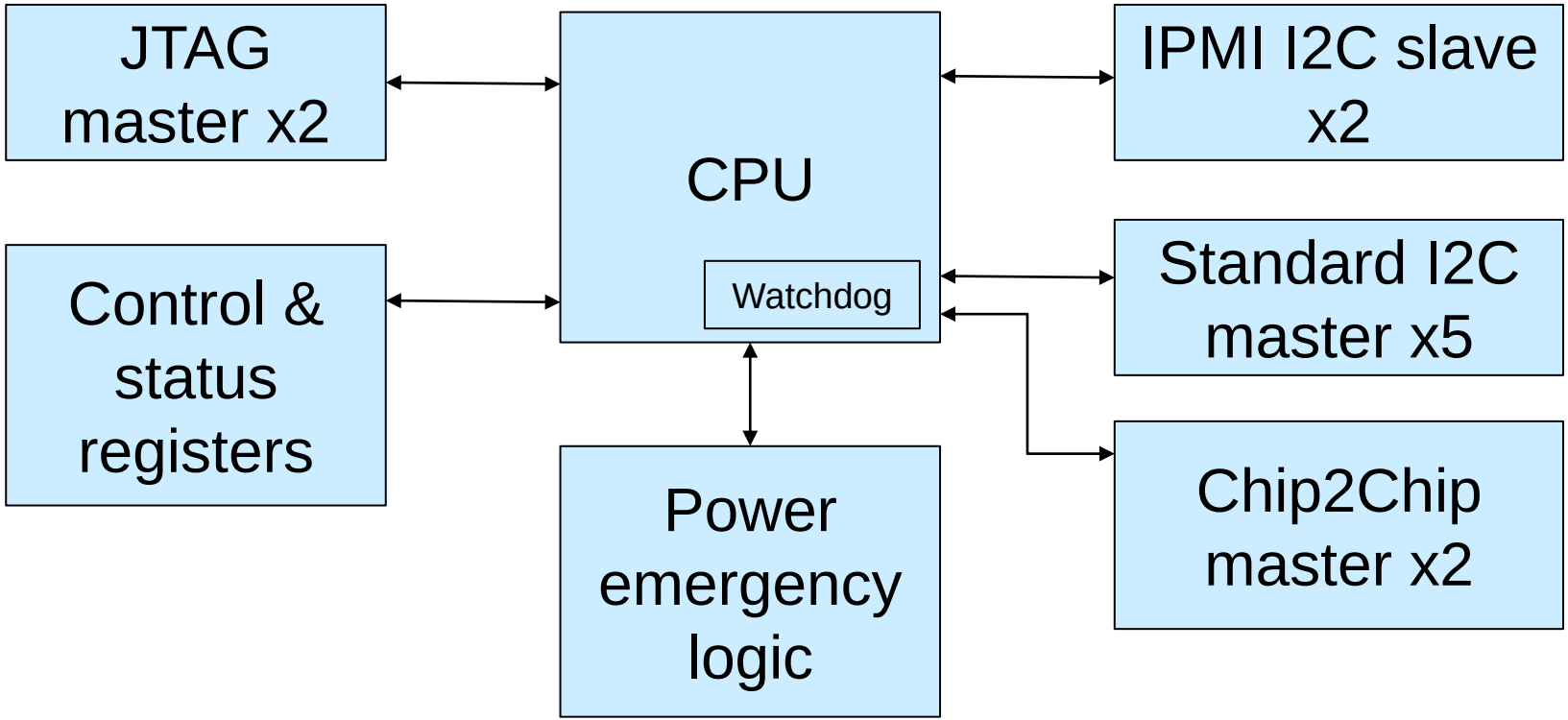
- Upgrade ZYNQ module to US+ family
- Selected Xilinx Kria K26 SoM as optimal candidate
  - Low cost: $300
  - 4 GB RAM
  - 4 GTH links 12.5 Gbps
- Faster AXI links to FPGA modules
  - 7.8125 Gbps
  - Max bit rate using CPLL
  - Both QPLLs in quad are needed for TCDS2
  - Rev 2: 3.75 Gbps max
- 10G Ethernet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



X2O Power Module block schematics



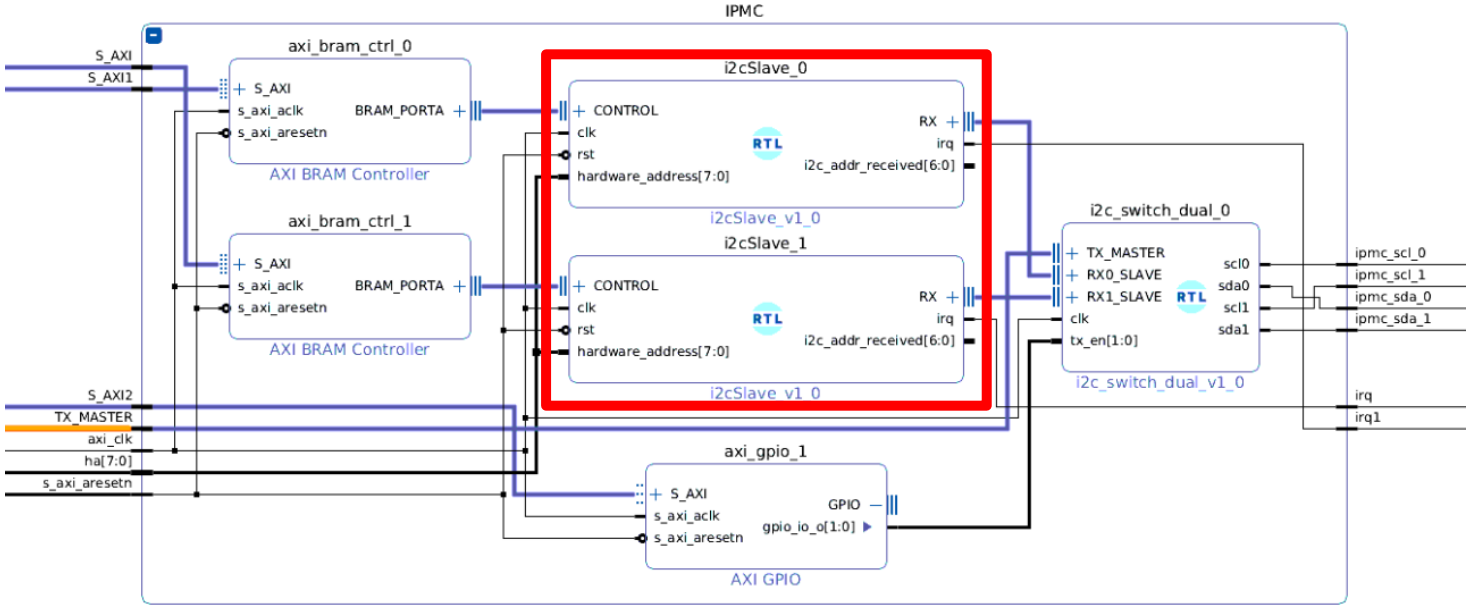
X2O Power Module ZYNQ firmware



X2O Power Module firmware[1] modules

IP Module	Description
IPMI I2C slave	Customized I2C slave transceiver capable of buffering long transfers. Does not pose any timing requirements to IPMC software.
Standard I2C master	Standard 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
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 Firmware



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

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 on KRIA module within X2O boards:

- Petalinux kernel version 2022.2
- 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
  - UCLA team has successfully implemented and tested custom Octopus board sensors (rev.2).
- Supports the use of various sensor readout interfaces, including I2C

Current status:

- (done) UF IPMC sw version for VU13P with QSFP-DD (rev.2)
- (in progress) 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 **2-3%**

ATCA shelf manager recognizing some of our temperature sensors (rev.2)

```
8c: LUN: 0, Sensor # 9 ("T:QSFPDD")
    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: 31 (0x1f)
    Processed data: 31.000000 degrees C
    Current State Mask: 0x60
       At or Above Upper Non-Recoverable Threshold

8c: LUN: 0, Sensor # 7 ("T:2V7_I+ Rmt")
    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: 68 (0x44)
    Processed data: 68.000000 degrees C
    Current State Mask: 0x07
       At or Below Lower Non-Critical Threshold
       At or Below Lower Critical Threshold
       At or Below Lower Non-Recoverable Threshold

8c: LUN: 0, Sensor # 5 ("T:V+ Rmt")
    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: 29 (0x1d)
    Processed data: 29.000000 degrees C
    Current State Mask: 0x1e
       At or Below Lower Critical Threshold
       At or Below Lower Non-Recoverable Threshold
       At or Above Upper Non-Critical Threshold
       At or Above Upper Critical Threshold
```

IPMC user code implementation (rev.2)

- TEMPLATES are provided
- UF IPMC Manual is provided
- Separate USER code space (for easy update)

Power ON

Power OFF

```
void user_module_payload_on( void )
{
  unsigned int payload_read;

  lock(1);
  payload_read = reg_read(devmem_ptr, qbv_on_off);
  payload_read |= 0x20;
  reg_write(devmem_ptr, qbv_on_off, payload_read);
  payload_timeout_init = lbolt;
  power_up_octopus(i2c_fd_snsr[1]);
  power_up_qsfpdd_module(i2c_fd_snsr[1]);
  logger("PAYLOAD", "On");
  power_up_done = 1;
  unlock(1);

}

void
user_module_payload_off( void )
{
  lock(1);
  unsigned int payload_read;
  payload_read = reg_read(devmem_ptr, qbv_on_off);
  payload_read &= ~0x20;
  power_down_octopus(i2c_fd_snsr[1]);
  power_down_qsfpdd_module(i2c_fd_snsr[1]);
  power_up_done = 0;
  reg_write(devmem_ptr, qbv_on_off, payload_read);
  logger("PAYLOAD", "Off");
  unlock(1);

}
```

Sensor enable

Upper non-recoverable threshold exceeded (Power OFF)

Upper critical threshold exceeded (Power OFF)

Upper non-critical threshold exceeded (Fans speed - UP)

Back to normal (Fans speed - DOWN)

Sensor disable

```
/*=============================================================*/
/*  QSFPDD Temp Sensor                                         */
/*=============================================================*/
void read_sensor_temp_qsfpdd(void) {
  lock(1);

  //  Wrapper parameters
  u8 i2c_ch = 0x01;

  //  Sensor Data Record
  u8 sensor_N = 9;

  if (check_power_up()) {
      //  Read the temp
      float temp_f = qsfpddTemperature(i2c_fd_snsr[i2c_ch]);

      //  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
          unlock(1);
          picmg_m6_state(fru_inventory_cache[0].fru_dev_id);
          logger("WARNING","Non-recoverable threshold crossed for QSFPDD temperature sensor");
          lock(1);
      } else if (sd[sensor_N].last_sensor_reading >= sdr[sensor_N].upper_critical_threshold) {
          // Transition to M6 for upper critical
          unlock(1);
          picmg_m6_state(fru_inventory_cache[0].fru_dev_id);
          logger("WARNING","Critical threshold crossed for QSFPDD temperature sensor");
          lock(1);
      } else if (up_noncrt_assert == 0 && sd[sensor_N].last_sensor_reading >= sdr[sensor_N].upper_non_critical_threshold) {
          // 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_non_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_non_critical_threshold) {
          // 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 = 0x81;
          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_non_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;
  }

  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.

- UF IPMC implements all functionality necessary for normal operation of the device in the ATCA chassis. It was tested in three different ways:
  1. In the CMS-standard ATCA chassis
  2. In the COMTEL ATCA chassis
  3. 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.**

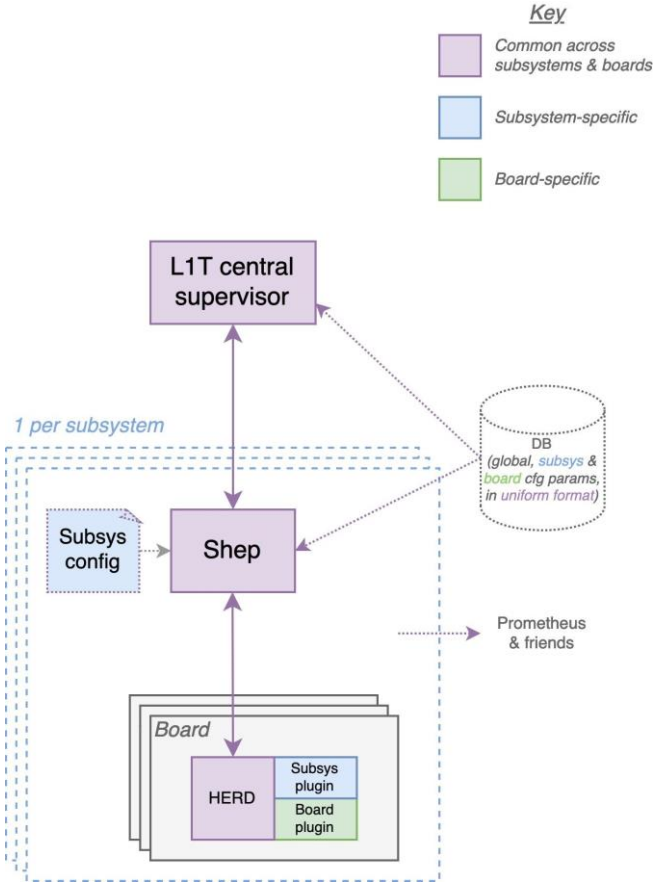
Thanks a lot to CERN EP-ESE team for the assistance with compliance tests!

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):

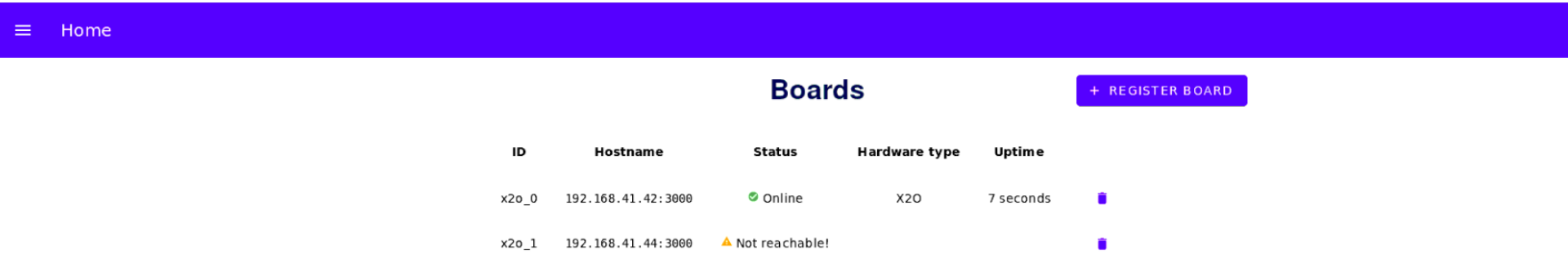
1. HERD
   - On-board application providing run control & monitoring interface (1 per board)
   - Builds on SWATCH library from phase-1 L1T
2. SHEP (short for shepherd)
   - Off-board application — each instance supervises a single subsystem

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



X2O SWATCH6

- **X2O/ Phase-2/SWATCH:**
  - Installed and tested SHEP.
  - Installed and tested initial version of HERD plugin:
    - (done) MGT configurator (tested for VU13P).
    - (done) DMA-proxy driver for bitstream uploading to FPGA via DMA-JTAG chain (tested for KU15P, VU13P).
    - (done) FPGA programming from HOST (tested for KU15P, VU13P).
    - (done) Added utilities: I2C tool, devmem tool, semaphores, configs parsers.
    - (done) Added clock(sync & async) synthesizer configuration (tested for KU15P, VU13P).
  - (in progress) X2O HERD plugin for Octopus (VU13P module).



FPGA MGT builder7

FPGA MGT builder is a set of software tools with the following functionality:

- Automatic generation of firmware structure with support for arbitrary MGT configurations. This includes:
  - Different bit rates and encodings in RX and TX parts of the same MGT
  - Using CPLL and QPLL as needed for each MGT, programmable separately for RX and TX parts (within the constraints of the particular FPGA architecture)
  - Automatic routing and assignment of available reference clocks for each MGT
  - Grouping MGTs into interfaces with programmable names and indexes, which makes using them in the top-level design much easier
  - Automatic generation of all constraints related to MGTs
    - Reference clock location, timing, and grouping
    - MGT location
    - User clock timing
  - The software is designed to make porting a project to different board design or FPGA an easy task.
  - Targeting lowest-latency serial links, by disabling RX and TX buffers
- Software framework:
  - Reads configuration settings and programs all DRP registers and port settings in each MGT and COMMON modules
  - Customizable reset procedures, with main functionality provided in the example design
  - Written in portable C++, can be adapted for nearly any system, including embedded processors
  - Does not need rework if the MGT configuration is changed
- Configuration sources:
  - All source configuration files are kept in Excel XLSX format
    - Makes working with them much easier
  - Data format is optimized for direct copying from Xilinx manuals and example source code, with minimal manual rework
  - A Python script is provided for exporting configuration files into plain text

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. X20 firmware project repository: https://github.com/madorskya/apex
2. UF IPMC github repository: https://gitlab.cern.ch/x2o/UF_IPMC
3. UG - 585 ZYNQ – 7000 SoC 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 HERD plugin: https://gitlab.cern.ch/cms-cactus/phase2/software/plugins/x2o
7. FPGA MGT Builder repository: https://github.com/madorskya/mgt_builder
8. 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

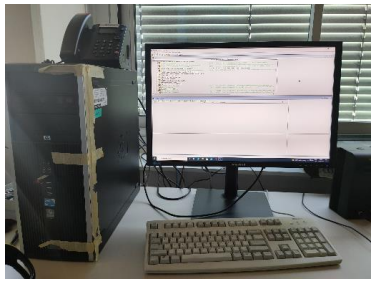
UF IPMC Polaris Compliance test results

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)

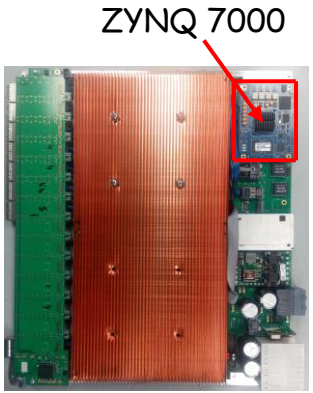


HOST with Polaris tester

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)