ZYNQ-IPMC

IPMC Workshop – ATLAS Upgrade

Marcelo Vicente (University of Wisconsin-Madison)
on behalf of the ZYNQ-IPMC team

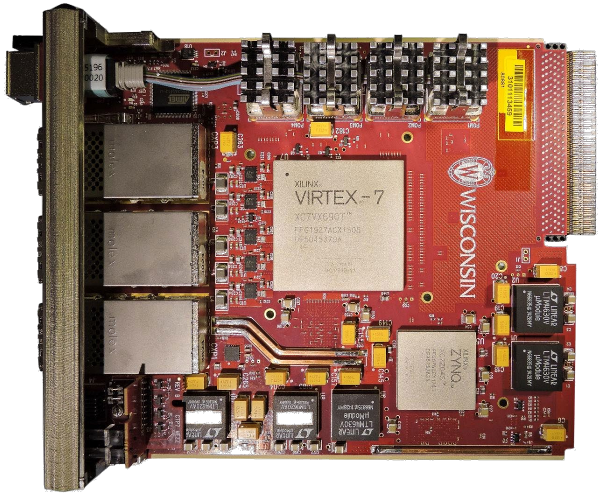
Outline

- **Introduction & Motivation**
- **The ZYNQ-IPMC Project**
  - Hardware
  - Software Framework
- **Features**
- **IPMI and SDRs**
- **Environment**
- **Configurability**
- **Summary**

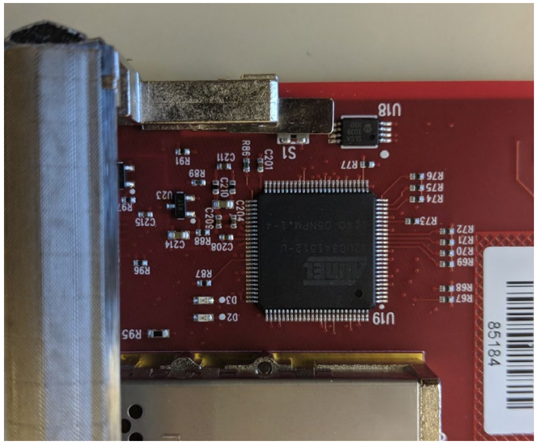
US CMS ATCA APx Consortium

- Pooling of efforts in ATCA Processor hardware, firmware and software development
  - Present efforts include CMS Phase-2 Calorimeter, Correlator and Muon Triggers; ECAL Barrel, CSC and GEM readouts.
- Multiple ATCA processors and mezzanine board types
- **University of Wisconsin-Madison** is responsible for the design of the **Advanced Processor (APx)** ATCA board and the **ZYNQ-IPMC**

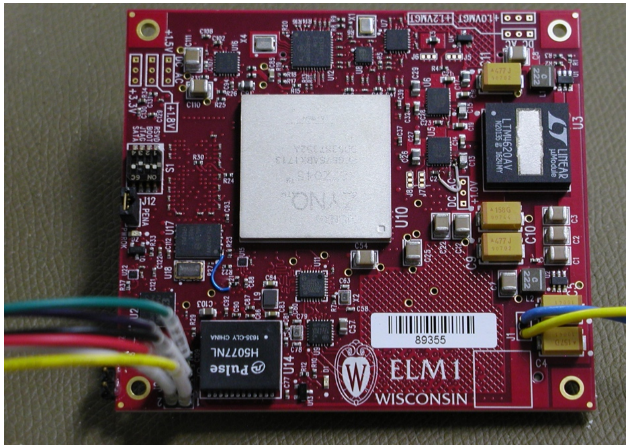
Related US-CMS UW HW Projects



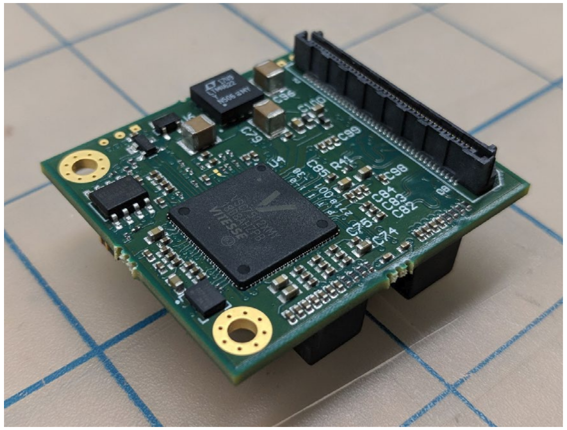
CMS Phase-1 Calo Trigger (2015)



UW-MMC (2013)



Linux Embedded Mezzanine – ELM (2017)



Ethernet Switch Mezzanine – ESM (2018)

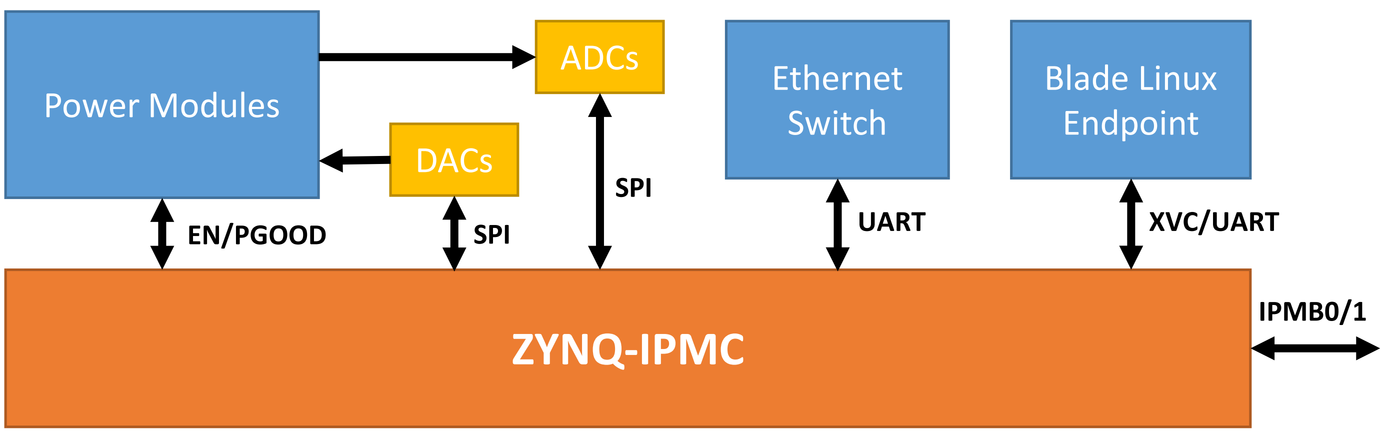
Why a fresh IPMC solution

- Available IPMC solutions lack critical features to safe-guard expensive electronics in ATCA in case of faults
- Exploit software experience designing other IPMI components: MMC and System Manager in CMS use

- Goal: design a fresh IPMC solution with:
  1. **Sub-millisecond response to critical board conditions**
     - Reduce chances of electrical damage in case of hardware fault
     - e.g. over-current, DCDC converter failure, short-circuits, etc.
  2. **Recoverability and monitoring**
     - No operational disturbances during IPMC soft-failures or maintenance
     - Monitor ATCA board metrics (temperatures, voltages, etc.) over time
  3. **Hardware and software flexibility**
     - Facilitate ATCA board design and cut down development time
  4. **Open-source – free of the hassle of dealing with NDAs and licenses**

The ZYNQ-IPMC project

- Develop a highly versatile IPMC platform for ATCA applications based on Xilinx ZYNQ SoC:
  - Exploit fast ARM Cortex-A9 processor and high FPGA parallelism for time-critical decisions
  - Support for a high number of peripherals (power controllers, fast ADCs, DACs, temperature sensors, RTMs, JTAG, etc.)
  - Previous experience with ZYNQ designs

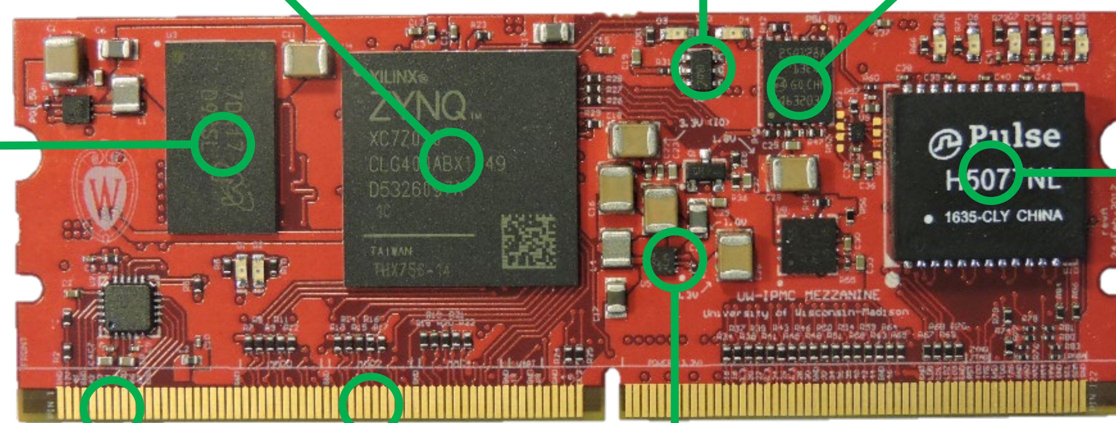


ZYNQ-IPMC Hardware (1)

- Specifications:
  - **Xilinx ZYNQ-7014S System-on-Chip**:
    - Single-core 32-bit ARM Cortex-A9 @ 666.6 MHz
    - 34.4 kLUT and 68.8 kFF Artix-7 FPGA core
    - FPGA takes care of critical tasks, offloading the CPU

  - **256MB DDR3-1066** / 64MB QSPI Flash / 256 Kb EEPROM
    - Frees developer from memory and storage constraints
  - 16 dedicated 16-bit ADC inputs up to **10 kSamples/s/ch**
    - Allows for very fast response times when paired with FPGA logic
  - 109 GPIOs accessible from FPGA logic
    - Allows native implementation of must standard protocols
    - Provides flexibility when routing ATCA boards

ZYNQ-IPMC Hardware (2)

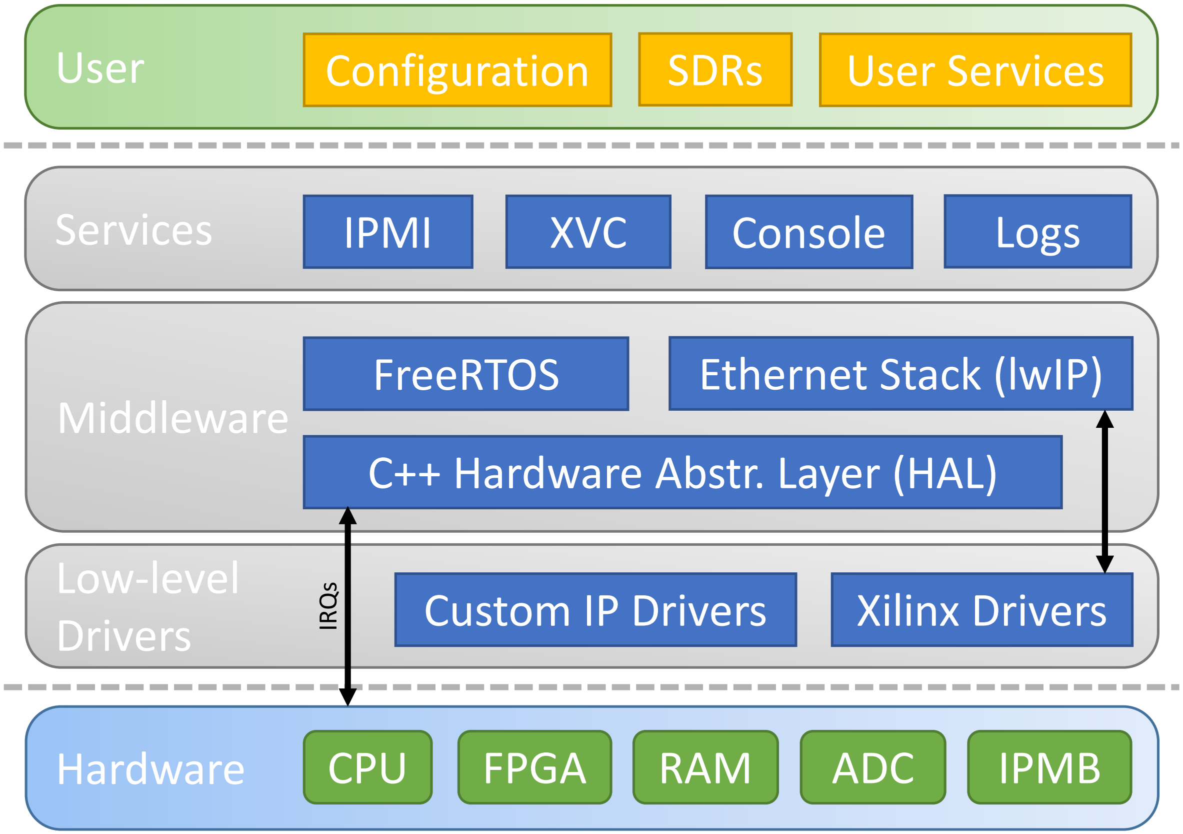


- 244-pin LP miniDIMM form factor (82mm x 30mm, 1mm thickness)
  - Mounted on a miniDIMM socket with a 22.5 degree angle to meet ATCA height specifications – components can still be mounted underneath
- Pinout similar to LAPP and CERN IPMC
  - **NOT** fully pin compatible, but migration fairly easy to do

ZYNQ-IPMC Framework (1)

- ZIOS – **Z**YNQ-**I**PMC **O**pen-source **S**tack
  - **Services**: Service based framework allowing coordination and management of many varied tasks
  - **FreeRTOS**: Real-time operating system providing clean per-service thread separation and prioritization of critical tasks – also well established!
  - **Drivers**: Interrupt based drivers integrated with FreeRTOS prevents system lock down and frees CPU cycles
  - **C++11**: Object-oriented programming allowing encapsulation of different modules and easy customization

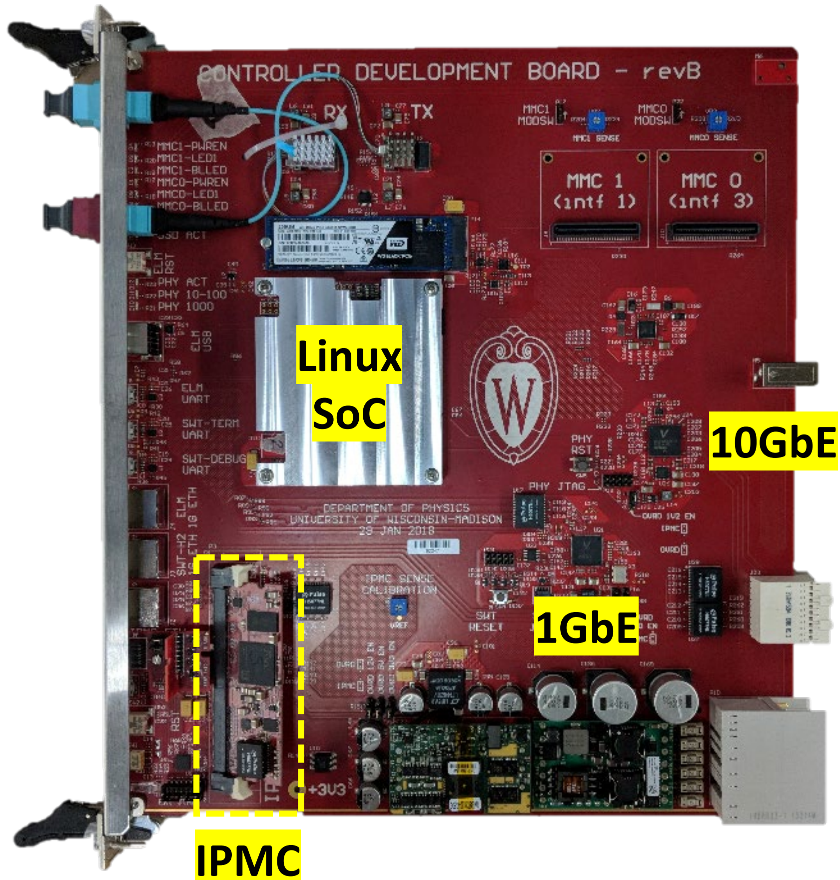
ZYNQ-IPMC Framework (2)



* Not every component illustrated

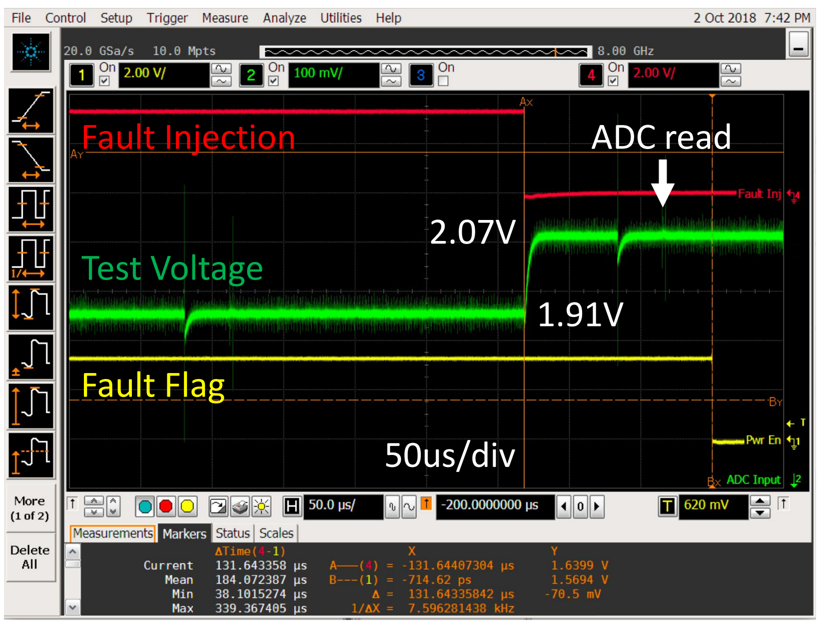
Test platform

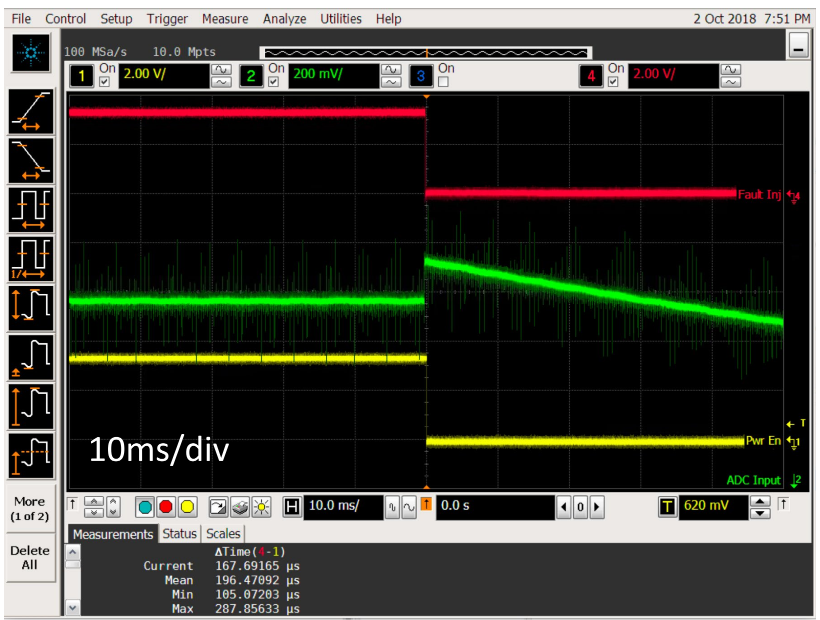
- **Controller Development Board (CDB)**
  - Development platform for several key technologies that will be used on other blades
  - Test platform for IPMCs and MMCs with Shelf Manager connectivity via the backplane
  - Several IPMC peripherals available:
    - IPMB-A and IPMB-B
    - I2C, SPI, MDIO, JTAG
    - Ethernet Switch Module
    - etc..



Fault Detection & Handling (1)

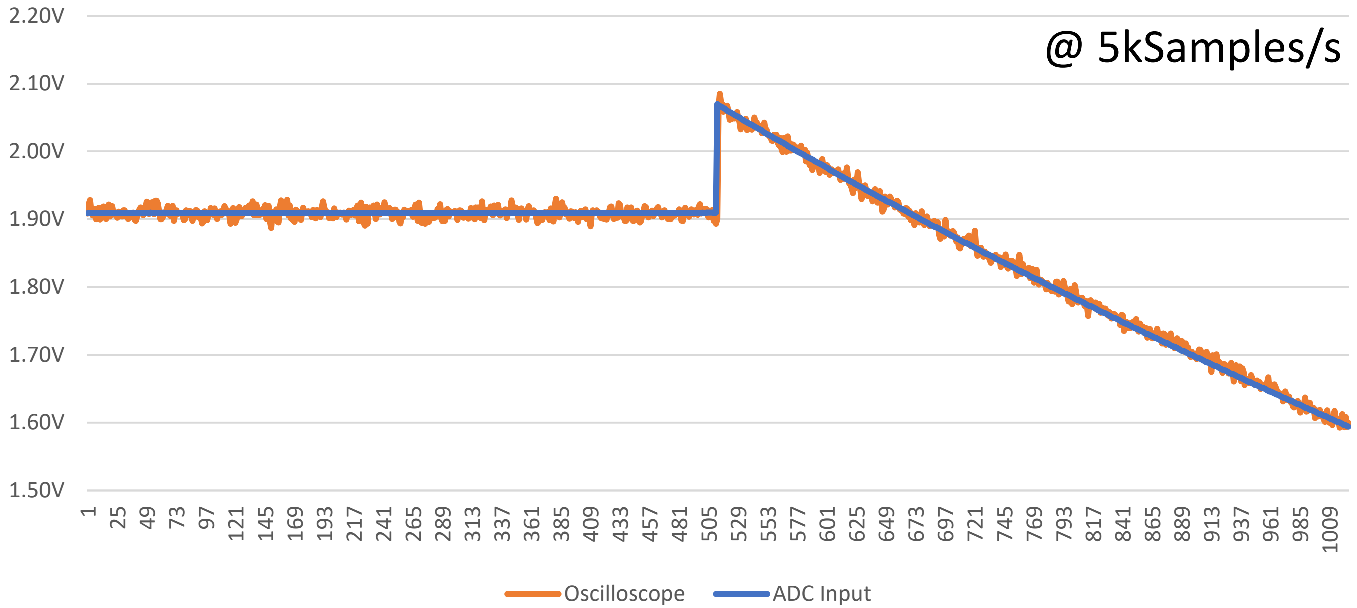
- **Fault detection processed in ZYNQ FPGA fabric – fast!**
  - Individual ADC channels checked in firmware (approx. 500ns)
  - Also supports additional external multi-channel ADCs
  - Thresholds tied to SDR definitions (non-critical, critical, non-recoverable)
  - IPMI service informed by interrupts
- At 3.3 kS/s/ch: **38μs < $t_{response}$ < 339μs**





Fault Detection & Handling (2)

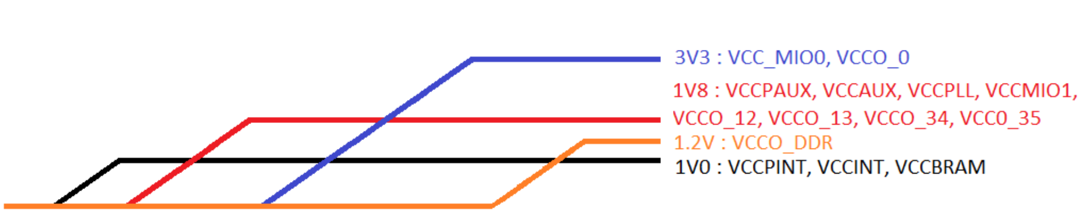
- ADC inputs are recorded into memory for post-mortem analysis of fault conditions on ATCA blades, facilitating debugging
  - Configurable pre-trigger point and capture length using DDR3 buffering
- Feature always running while in use – immediate captures supported as well



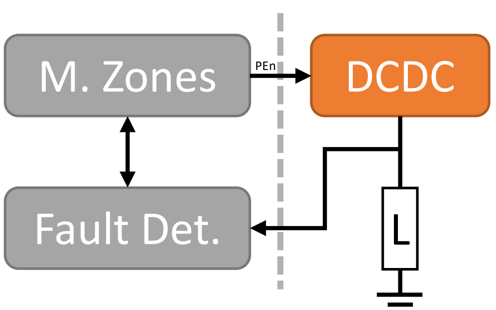
- “Essentially a multi-channel DSO”

Power Management Zones

- Allows timed and **sequential power-up/down of power rails** by having tight control of power enable pins – also in FPGA fabric
  - Reduces the need for external sequencing logic
  - e.g. FPGA power-up and proper sequencing
- Allows creation of multiple independent power groups (zones) and their operation defined in software
  - **Prevents sneak path currents if configured properly**
- Integration with Fault Detection & Handling
  - **Associates sensors with zones and defines proper behavior in case of non-recoverable faults**



Example of XC7Z030 power sequence, Xilinx Forums



Recoverability

- Internal fault detection of sensitive code by **watchdog**
  1. Output signals (e.g. power enables) hold by external circuitry on main board as reset takes place
  2. Outputs and memory are then readout to understand previous state
  3. A state recovery jump takes place if the previous state was clearly understood
     - External circuitry releases output lines if recovery fails

- **Transparent recovery since no power cut takes place**
  - Fault still gets logged for analysis
- External hold circuit will be available as reference design

HL-LHC US CMS

Ethernet Interface

THE UNIVERSITY of WISCONSIN MADISON

- **lwIP 2.x with on-board Gigabit Ethernet PHY and magnetics**
- Unique MAC addresses assigned per card (EEPROM MAC)
  - Stored in EEPROM MAC - can be overwritten if required
- IP assignment through **DHCP** or **statically set**
  - DHCP server can be configured to provide geographical addresses
- TCP/IP **iperf server** on the IPMC
  - Example:
    1. Control PC
    2. Lab Switch
    3. ATCA 1/10GbE Switch (ATC807)
    4. On-board Switch (ESM)
    5. IPMC

```
[redacted]$ iperf -c 192.168.250.252 -i 5 -t 60 -w 64k
------------------------------------------------------------
Client connecting to 192.168.250.252, TCP port 5001
TCP window size:  125 KByte (WARNING: requested 62.5 KByte)
------------------------------------------------------------
[  3] local 192.168.1.8 port 46468 connected with 192.168.250.252 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 5.0 sec   316 MBytes   531 Mbits/sec
[  3]  5.0-10.0 sec   316 MBytes   531 Mbits/sec
[  3] 10.0-15.0 sec   316 MBytes   529 Mbits/sec
[  3] 15.0-20.0 sec   317 MBytes   531 Mbits/sec
[  3] 20.0-25.0 sec   316 MBytes   531 Mbits/sec
[  3] 25.0-30.0 sec   304 MBytes   510 Mbits/sec
[  3] 30.0-35.0 sec   316 MBytes   531 Mbits/sec
[  3] 35.0-40.0 sec   316 MBytes   531 Mbits/sec
[  3] 40.0-45.0 sec   316 MBytes   531 Mbits/sec
[  3] 45.0-50.0 sec   316 MBytes   531 Mbits/sec
[  3] 50.0-55.0 sec   316 MBytes   531 Mbits/sec
[  3] 55.0-60.0 sec   316 MBytes   531 Mbits/sec
[  3]  0.0-60.0 sec  3.69 GBytes   529 Mbits/sec
```

(1 min test)

- **SNTP** used for time synchronization

Console Interface (1)

- Serial console via **UART** or **Telnet** – same user interface
- Feature packed command line interface (CLI):
  - Command history
  - Auto-complete
  - Color tagging
  - etc.
- Real-time log messages are mirrored to all interfaces

```
[NOTI] Our MAC address is [redacted]
[NOTI] Our IPMB0 address is E6h
[NOTI]
********************************************************************************

ZYNQ-IPMC - Open-source IPMC hardware and software framework
HW revision : 1
SW revision : 45cf121
HW serial   : 256
Build date  : Thu Oct 11 09:36:16 CEST 2018
Build host  : [redacted]
Build conf  : Debug
OS version  : FreeRTOS V10.0.1

 M Vivado/ipmc_zynq_vivado.sdk/ipmc_bd_wrapper_hw_platform_0/.project
 M Vivado/ipmc_zynq_vivado.sdk/ipmc_standalone_bsp/ps7_cortexa9_0/lib/libxilffs.a
 M Vivado/ipmc_zynq_vivado.xpr

********************************************************************************

[NOTI] Network interface is UP
[NOTI] DHCP request success
Address: 128.141.188.206
Netmask: 255.255.255.0
Gateway: 128.141.188.1

> ps
PID Name                BasePrio CurPrio StackHW State CPU% CPU
  2 IDLE                       0       0     146     R  99% 10116511
  3 tmrd                       5       5     392     B  <1% 3890
 12 linkdet                    0       0     390     B  <1% 2082
  5 PersistentFlush            1       1    1767     B  <1% 1267
  7 ipmb0                      4       4    1497     B  <1% 728
 11 _lwipd                     3       3     856     B  <1% 662
 14 xemacifd                   3       3    1944     B  <1% 216
  8 console                    2       2    1365     *  <1% 196
 15 _dhcpd                     3       3    1668     B  <1% 140
  4 PS_WDT                     6       6    1920     B  <1% 46
 17 telnetd                    3       3    1852     B  <1% 28
 20 ftpd:21                    3       3    1876     B  <1% 21
 18 lwiperfd                   3       3    1500     B  <1% 18
 19 xvcserver:2542             1       1    1872     B  <1% 18
 21 statd                      1       1     737     B  <1% 11
 16 influxdbd                  1       1    1948     B  <1% 6
> help date
date

Print the current system time. Updated by SNTP and kept by FreeRTOS.
> date
Thu Oct 11 07:37:55 2018
>
```

Console Interface (2)

- Services and drivers can register new commands dynamically to the console
  - Example for the 'date' command:

```
/// A "date" console command.
class ConsoleCommand_date : public CommandParser::Command {
public:
    virtual std::string get_helptext(const std::string &command) const {
        return stdsprintf(
                "%s\n\n"
                "Print the current system time. Updated by SNTP and kept by FreeRTOS.\n", command.c_str());
    }

    virtual void execute(std::shared_ptr<ConsoleSvc> console, const CommandParser::CommandParameters &parameters) {
        timeval tv;
        gettimeofday(&tv, NULL);
        if (tv.tv_sec == 0) {
            console->write("Time information unavailable.\n");
        } else {
            struct tm *timeinfo = localtime(&tv.tv_sec);
            console->write(std::string(asctime(timeinfo))); // GMT
        }
    }
};
```

  - Command is then registered in the command parser:

```
console_command_parser.register_command("date", std::make_shared<ConsoleCommand_date>());
```

Logging

- **Granular and per-component configurable logging provides verbosity when needed**
  - Several levels of logging severity provided
    - e.g.: Critical, error, warning, notice, etc.
  - Hierarchical facility tree allows per-component filtering
    - e.g. 'ipmc.console' has sub-component 'ipmc.console.uart'
  - Console used to visualize logs in **real-time**
    - Severity filtering prevents overwhelming of information
    - Multiple console instances with different filter settings
  - Log buffer **backed up by large DDR memory**, can be read at any time by **JTAG** or **Console**

```
[NOTI] Request resent on ipmb0:    0.8c -> 0.20: 0a.11 (seq 40) [01 08 00 02]  (retry 6)
[NOTI] Unexpected response received on ipmb0 (erroneous retry?): 0.20 -> 0.8c: 0b.11 (seq 00) [00 02 01 04]
[NOTI] Request resent on ipmb0:    0.8c -> 0.20: 0a.11 (seq 40) [01 08 00 02]  (retry 7)
[NOTI] Unexpected response received on ipmb0 (erroneous retry?): 0.20 -> 0.8c: 0b.11 (seq 00) [00 02 01 04]
[NOTI] Request resent on ipmb0:    0.8c -> 0.20: 0a.11 (seq 40) [01 08 00 02]  (retry 8)
[NOTI] Request resent on ipmb0:    0.8c -> 0.20: 0a.11 (seq 40) [01 08 00 02]  (retry 9)
[NOTI] Unexpected response received on ipmb0 (erroneous retry?): 0.20 -> 0.8c: 0b.11 (seq 00) [00 02 01 04]
[WARN] Retransmit abandoned on ipmb0: 0.8c -> 0.20: 0a.11 (seq 40) [01 08 00 02]
```

Metric Gathering

- Real-time publication of metrics to **InfluxDB**:
  - Temperatures, voltages, currents, logs, etc.
  - Internal and external sources supported (ADCs, DCDCs, etc.)

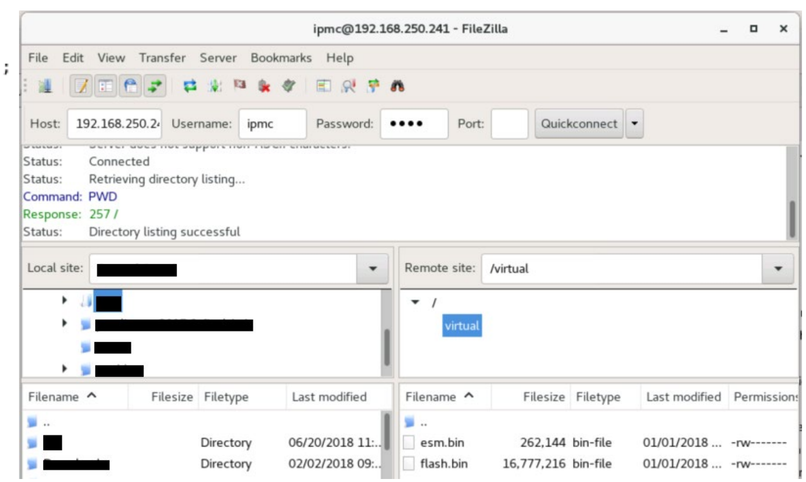
```
ADC::Channel::Factor tmp36 = [](float r) -> float  { return (r - 0.5) * 100.0; };
ADC::Channel tCDBbot  (*adc[1], 0, tmp36);  // Temperature channel w/ custom factor
ADC::Channel v3p3mp   (*adc[0], 7, 1.600);  // Voltage channel w/ x1.6 factor
ADC::Channel v1p0eth  (*adc[1], 2);         // Voltage channel w/ default factor (x1.0)

const InfluxDB::Timestamp timestamp = InfluxDB::getCurrentTimestamp();
influxdb->write("cdb.temperature", {{"ipmc", ipmc}, {"name", "bot"}}, {{"value", (float)tCDBbot}}, timestamp);
influxdb->write("cdb.voltage", {{"ipmc", ipmc}, {"name", "v3p3mp"}}, {{"value", (float)v3p3mp}}, timestamp);
influxdb->write("cdb.voltage", {{"ipmc", ipmc}, {"name", "v1p0eth"}}, {{"value", (float)v1p0eth}}, timestamp);
```

- **Grafana** used for real-time & offline metric visualization

Upgradability

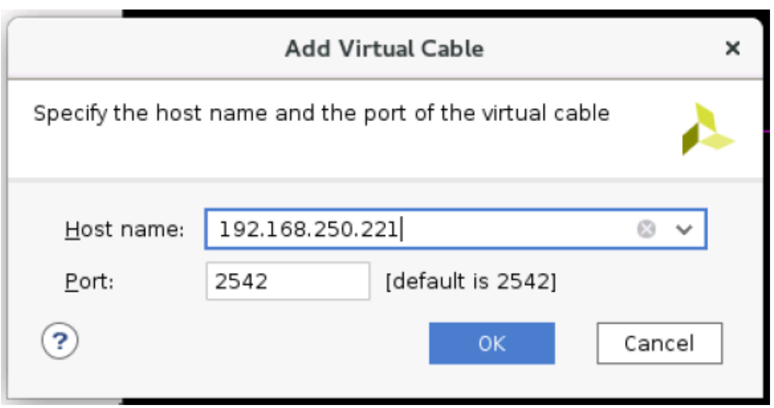
- Via **File Transfer Protocol (FTP)**
  - Virtual file system
    - Each file provides read/write software handlers
    - Handlers used to process and push uploaded files to wired devices (e.g. QSPI flash, EEPROMs, external mezzanines, etc.)
    - Allows validation and post-programmability verification
  - Supports authentication
- Also via **Console**

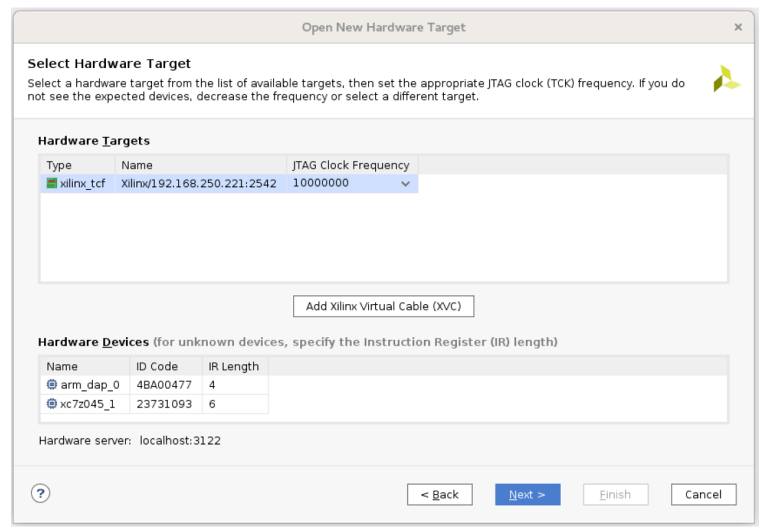
```
FTPServer::addFile("virtual/flash.bin", FTPFile(flash_read, flash_write, 16 * 1024 * 1024));

size_t flash_write(uint8_t *buf, size_t size) {
    // Validate the bin file before writing
    BootFileValidationReturn r = validateBootFile(buf, size);
    if (r != BFV_VALID) {
        // File is invalid!
        printf("Received bin file has errors: %s. Aborting firmware update.",
                getBootFileValidationErrorString(r));
        return 0;
    } else {
        printf("Bin file is valid, proceeding with update.");
    }

    // Write the buffer to flash
```



Xilinx Virtual Cable (XVC)

- **JTAG over PL GPIOs – firmware IP (Xilinx XAPP1251)**
- One interface dedicated in IPMC pinout
  - But multiple interfaces are supported
- 12.7MB image in 25 seconds (chain at 10 MHz)





- GUI support since Vivado 2018.1

Persistent Storage

- Automatic modular storage allocation simplifies persistent storage for diverse modules
  - Streamlines the process of **storing card and service based configurations**
  - **Automatic synchronization** between volatile and non-volatile storages
- Allocations example:

```
namespace PersistentStorageAllocations {
    /* Vendor 0: RESERVED */
    PERSISTENT_STORAGE_ALLOCATE(0x0000, RESERVED_END_OF_INDEX); ///< A marker to internally denote the end of the index.
    /* Vendor 1: University of Wisconsin */
    PERSISTENT_STORAGE_ALLOCATE(0x0101, WISC_SDR_REPOSITORY);   ///< The SDR repository.
    PERSISTENT_STORAGE_ALLOCATE(0x0102, WISC_INFLUXDB_CONFIG);  ///< InfluxDB configuration
    PERSISTENT_STORAGE_ALLOCATE(0x0103, WISC_NETWORK_AUTH);     ///< Auth configuration for network services
};
```

- Usage example:

```
// Get the current hashes from persistent storage
uint16_t secver = persistent_storage->get_section_version(PersistentStorageAllocations::WISC_NETWORK_AUTH);
HashPair *nvHashes = (HashPair*)persistent_storage->get_section(PersistentStorageAllocations::WISC_NETWORK_AUTH, 1, sizeof(HashPair));
if (secver == 0) {
    // Default password and user
    ResetCredentials();
}

// Compare keys
if (memcmp(nvHashes->pass, provPassHash, SHA_VALBYTES) == 0) {
    // Password is valid!
    return true;
} else {
    // Password is incorrect!
    return false;
}
```

Sensor Data Records (SDRs)

In accordance with the IPMI specification, but with human-comprehensible field naming

```
SensorDataRecord01 plyd3v3;
plyd3v3.entity_id(0x0);
plyd3v3.entity_instance(0x60);
plyd3v3.events_enabled_default(true);
plyd3v3.scanning_enabled_default(true);
plyd3v3.sensor_auto_rearm(true);
plyd3v3.sensor_hysteresis_support(2); // Readable/Settable
plyd3v3.sensor_threshold_access_support(2); // Readable/Settable
plyd3v3.sensor_event_message_control_support(0); // Full-Granular
plyd3v3.sensor_type_code(0x02); // Voltage
plyd3v3.event_type_reading_code(0x01); // Threshold
plyd3v3.assertion_lower_threshold_reading_mask(0x7fff); // All events supported & assertions enabled.
plyd3v3.deassertion_upper_threshold_reading_mask(0x7fff); // All events supported & deassertions enabled
plyd3v3.discrete_reading_setable_threshold_reading_mask(0x3fff); // All thresholds are configurable.
plyd3v3.units_numeric_format(SensorDataRecord01::UNITS_UNSIGNED);
plyd3v3.units_base_unit(4); // Volts
// IPMI Specifies a linearization function of: y = L[(Mx + (B * 10^(Bexp) ) ) * 10^(Rexp) ]
// Our settings produce a valid range of 2000 mV to 4540 mV with 10 mV granularity.
plyd3v3.conversion_m(1);
plyd3v3.conversion_b(20);
plyd3v3.b_exp(1);
plyd3v3.r_exp(1);
plyd3v3.sensor_direction(SensorDataRecordReadableSensor::DIR_INPUT);
plyd3v3.nominal_reading_specified(true);
plyd3v3.nominal_reading_rawvalue(130); // 3300 mV
plyd3v3.threshold_unr_rawvalue(147); // 3470 mV
plyd3v3.threshold_ucr_rawvalue(142); // 3420 mV
plyd3v3.threshold_unc_rawvalue(137); // 3370 mV
plyd3v3.threshold_lnc_rawvalue(123); // 3230 mV
plyd3v3.threshold_lcr_rawvalue(118); // 3180 mV
plyd3v3.threshold_lnr_rawvalue(113); // 3130 mV
plyd3v3.hysteresis_high(1); // +10 mV
plyd3v3.hysteresis_low(1); // -10 mV
device_sdr_repo.add(plyd3v3);
```

- Sensors are updated in individual tasks at fixed rates set at the sensor level
  - e.g. pyld3v3.update_value(3300.0)
  - Threshold processing, conversions, events, etc. are executed internally
- **Toolset under development to automate SDR generation**

IPMB Interface

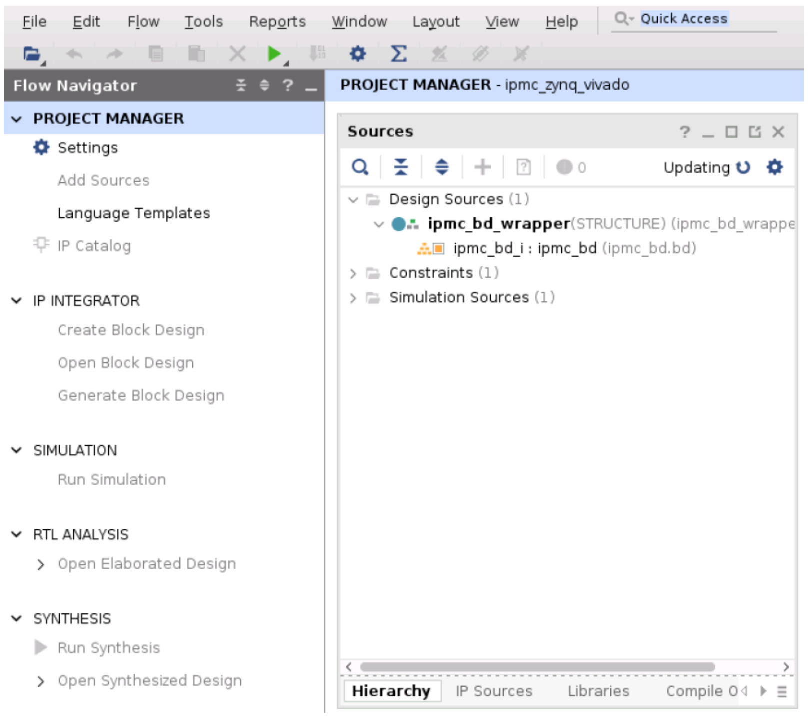
- IPMB0 and IPMB1 connected to processor fabric
  - IPMI service handles requests and arbitration when needed
  - Ability to send and receive raw IPMI commands from Console
- Example: dump FRU information from other IPMI controllers

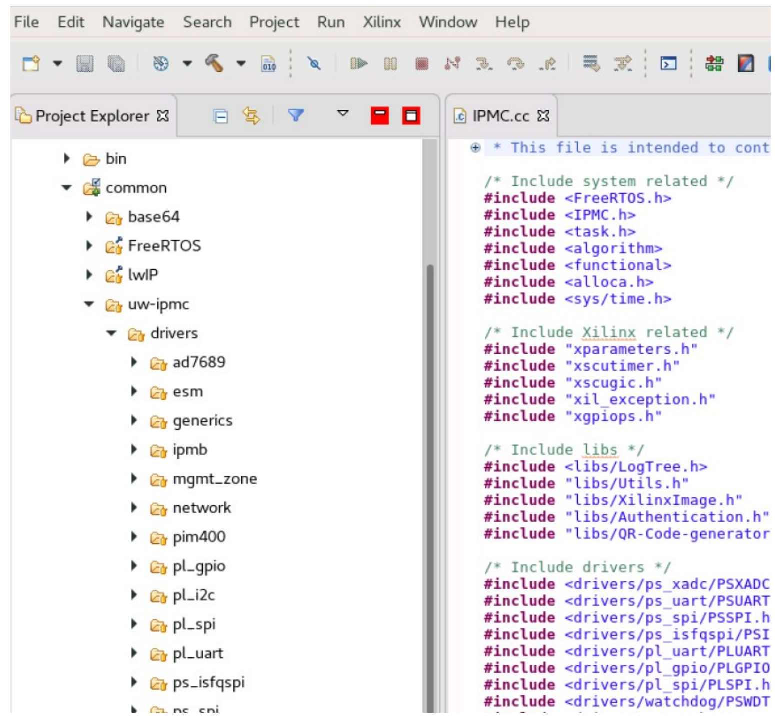
```
> ipmb0.dump_fru_storage 0x20 0
Board Area:
  Board Area Version: 1
  Language Code:      0x19 "en English"
  Mfg. Date:          2015-04-25 05:20:00
  Manufacturer:       "Pigeon Point System"
  Product Name:       "Pigeon Point Shelf Manage"
  Serial Number:      "<NULL>"
  Part Number:        "<NULL>"
  FRU File ID:        "<NULL>"
Product Info Area:
  Product Info Area Version: 1
  Language Code:      0x19 "en English"
  Manufacturer:       "Pigeon Point System"
  Product Name:       "Pigeon Point Shelf Manage"
  Product Part/Model: "<NULL>"
  Product Version:    "3.4.2."
  Serial Number:      "<NULL>"
  Asset Tag:          "<NULL>"
  FRU File ID:        "<NULL>"
```

```
> ipmb0.dump_fru_storage 0x20 1
Chassis Information Area:
  Info Area Version:  1
  Chassis Type:       0x17 "Rack Mount Chassis"
  Part Number:        "470-07000-17."
  Serial Number:      "4516GV-000"
Board Area:
  Board Area Version: 1
  Language Code:      0x19 "en English"
  Mfg. Date:          2016-10-22 08:32:00
  Manufacturer:       "Comtel Electronics Gmb"
  Product Name:       "ATCA 14 Slot Full Mesh 100G Speed Backplan"
  Serial Number:      "0701001A0001549002    "
  Part Number:        "370-07010-1.A0"
  FRU File ID:        "                                "
  Custom Info:
    "                                "
Product Info Area:
  Product Info Area Version: 1
  Language Code:      0x19 "en English"
  Manufacturer:       "Comtel Electronics Gmb"
  Product Name:       "ATCA 14U 14 slot C014G4R shel"
  Product Part/Model: "370-07004    "
  Product Version:    "000"
  Serial Number:      "0700400A1001625026          "
  Asset Tag:          "                              "
  FRU File ID:        "470-07000-17(Rev2"
  Custom Info:
    "ATCA C014G4 Power Backplane (Radial IPMB"
Multi-Record Area: Present
```

Programming Environment

- Using Xilinx Vivado 2018.2 and XSDK (Eclipse)

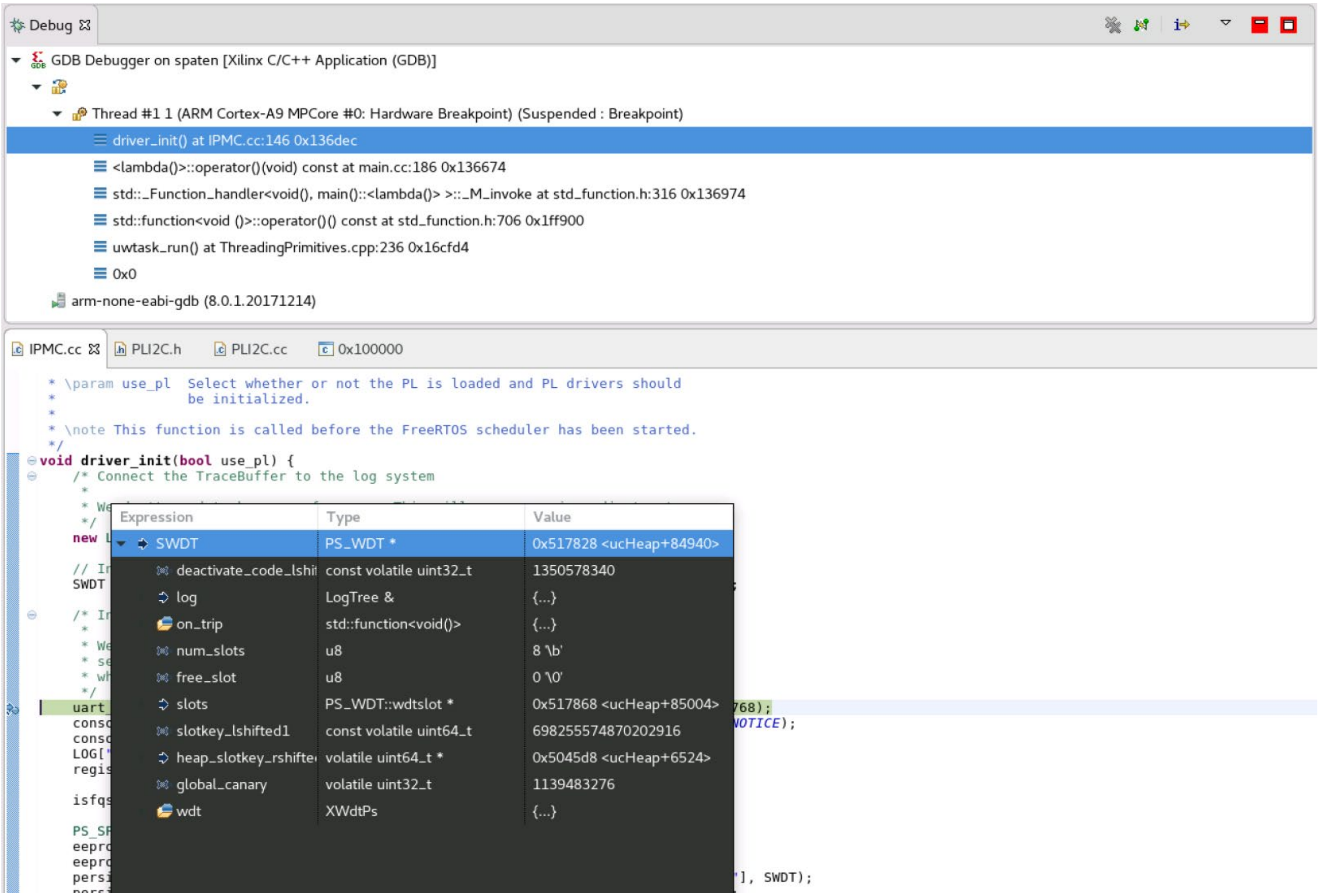




- Supports step-by-step debugging and programming

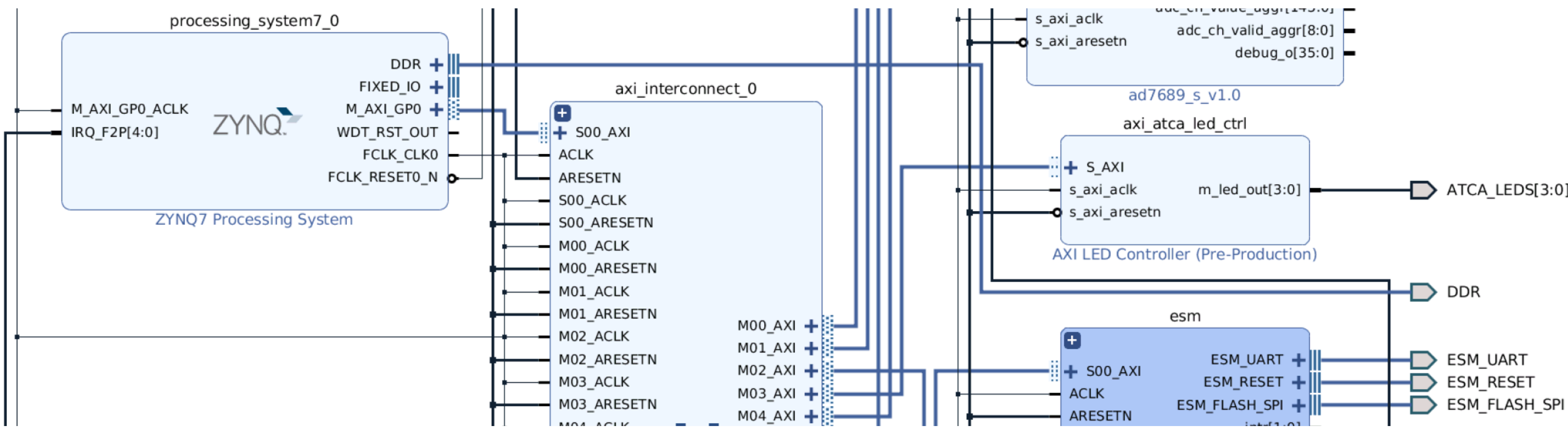
Debugging

- Via Xilinx JTAG dongle and integrated in XSDX



Configurability (1)

- Firmware and software configuration go hand-in-hand:
  - Using Xilinx IP packager in Vivado for AXI integration
  - IPs include firmware blocks and low-level C drivers
    - Using both Xilinx standard and custom developed IPs
  - **Leveraging Xilinx Vivado Block Design automation to speed up development time**



Configurability (2)

- Object-oriented & inheritance facilitates programming and reconfiguration of similar drivers and services
- Example of SPI inheritance between CPU and FPGA drivers:

```
/**
 * An abstract SPI master driver.
 * Chain operations supported by AddressableAtomicitySupport::atomic.
 */
class SPIMaster : public AddressableAtomicitySupport {
public:
     * This function will perform a SPI transfer in a blocking manner.
    virtual bool transfer(u8 chip, const u8 *sendbuf, u8 *recvbuf, size_t bytes, TickType_t timeout = portMAX_DELAY) = 0;
```

↑ ↑

```
class PS_SPI : public SPIMaster, protected InterruptBasedDriver {
public:
    /**
     * Constructs the PS SPI driver.
     * @param DeviceId AXI SPI device ID.
     * @param IntrId The interrupt ID associated with the IP.
     */
    PS_SPI(uint16_t DeviceId, uint32_t IntrId);
    ~PS_SPI();
```

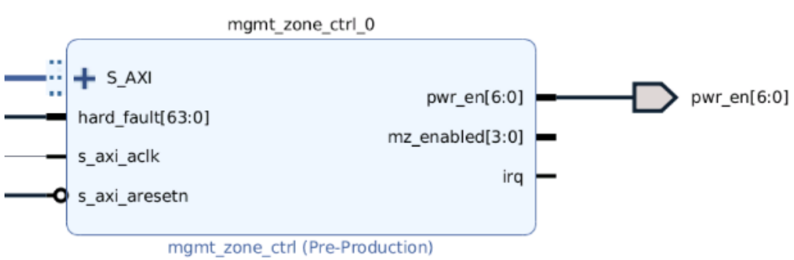
```
class PL_SPI : public SPIMaster, protected InterruptBasedDriver {
public:
    /**
     * Constructs the PL QSPI driver.
     * @param DeviceId AXI QSPI device ID of target IP.
     * @param IntrId The interrupt ID associated with the IP.
     */
    PL_SPI(uint16_t DeviceId, uint32_t IntrId);
    ~PL_SPI();
```

- Example of driver propagation between software layers:

```
PS_SPI *ps_spi0 = new PS_SPI(XPAR_PS7_SPI_0_DEVICE_ID, XPAR_PS7_SPI_0_INTR);
EEPROM *eeprom_data = new SPI_EEPROM(*ps_spi0, 0, 0x8000, 64);
EEPROM *eeprom_mac = new SPI_EEPROM(*ps_spi0, 1, 0x100, 16);
persistent_storage = new PersistentStorage(*eeprom_data, LOG["persistent_storage"], SWDT);
persistent_storage->register_console_commands(console_command_parser, "eeprom.");
```

Configuration Workflow (1)

- 1. Add firmware IPs or custom VHDL in block design
  - Example of custom IP: Management Zone Control
  - Aggregates and manages related power trees
  - IP is configurable through GUI, driver will automatically adapt
  - Same procedure for Xilinx IPs and other custom IPs



Configuration Workflow (2)

- 2. Use provided C++ drivers in framework or use expandable templates
  - Example C++ driver for Management Zone Control

```
/**
 * A single management zone.
 */
class MGMT_Zone {
public:
    const u16 DeviceId; ///< The DeviceId of the controller hosting this zone.
    const u32 MZNo; ///< The MZ number within the MZ Controller
    MGMT_Zone(u16 DeviceId, u32 MZNo);
    virtual ~MGMT_Zone();

    virtual void set_power_state(PowerAction action);
    virtual bool get_power_state(bool *in_transition=NULL);

    /**/
};
```

  - Drivers are initialized on dedicated function in framework

```
void driver_init(bool use_pl) {

    /**/

    // Create required instances for  management zones
    for (int i = 0; i < XPAR_MGMT_ZONE_CTRL_0_MZ_CNT; ++i)
        mgmt_zones[i] = new MGMT_Zone(XPAR_MGMT_ZONE_CTRL_0_DEVICE_ID, i);

    /**/
}
```

Opportunities to collaborate

- **Open-source hardware and software**
  - Full rights to customize, reuse or modify
  - No commercial licensing or NDAs required

- Preliminary documentation already available:

  https://drive.google.com/open?id=1LBVPM_l-X3dnLe49_n24LOrSW49aN5_s

Interested peers are welcome to get in touch with us:

marcelo.vicente@cern.ch

Thank you!

Related Peripheral: Ethernet Switch Mezzanine

- Compact **6-port 1Gb Ethernet Switch** module (35mm x 40mm)
  - Four 1000BASE-T interfaces with **on-board magnetics**
  - Two additional SGMII interfaces available
- **Targeted for ATCA blades where Ethernet connectivity is required**
  - Unmanaged with low to no maintenance required
  - Singly 3.3V supply (e.g. from management power)
  - Easily connects IPMC, SoC, HUB 1, HUB 2 and FPGAs together
  - Programmed through the ZYNQ-IPMC by using SPI or UART