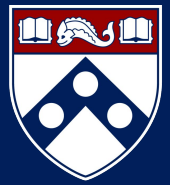
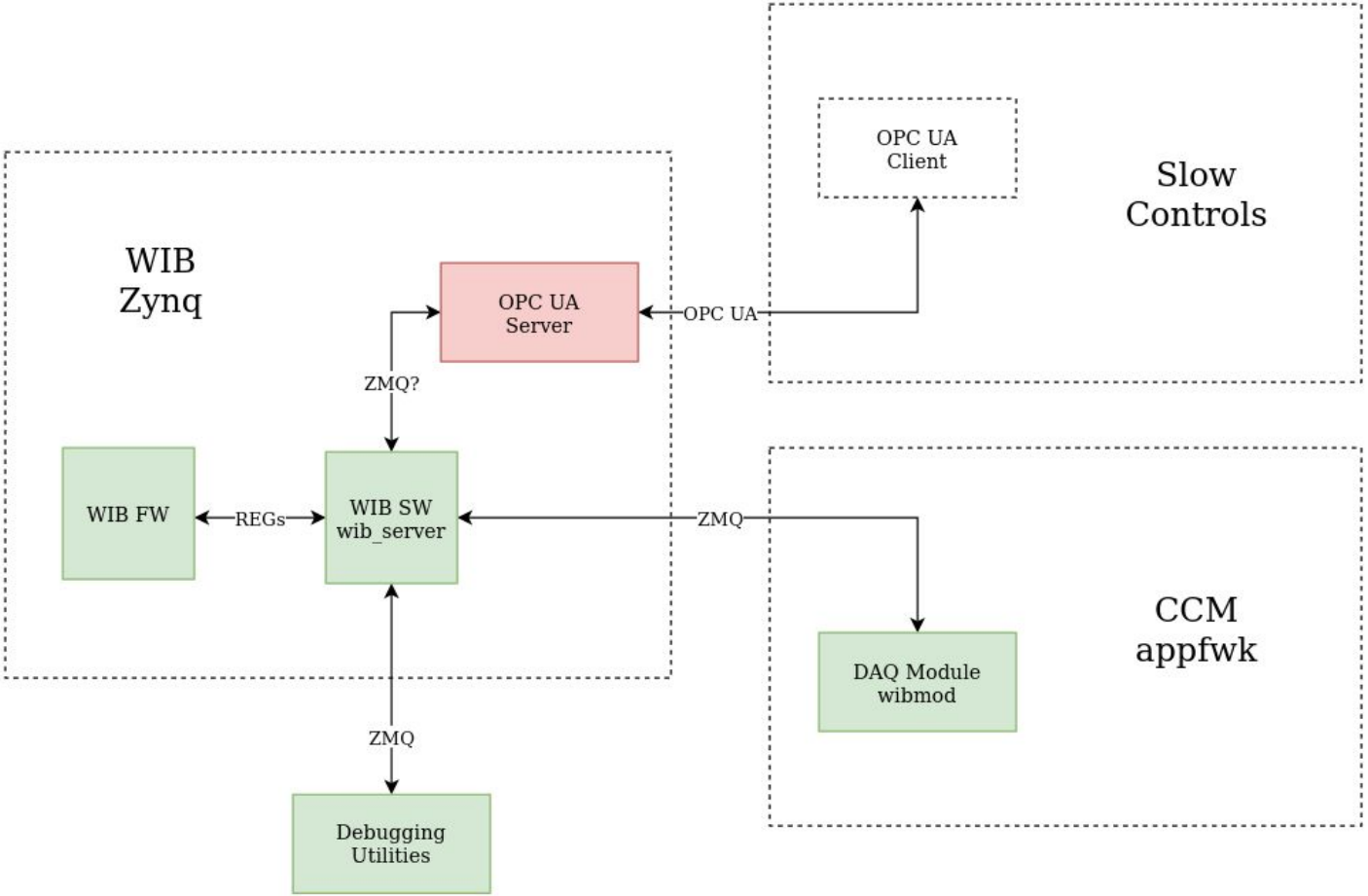


Penn
University of Pennsylvania

WIB OPC UA Server

Ben Land
July 20, 2021

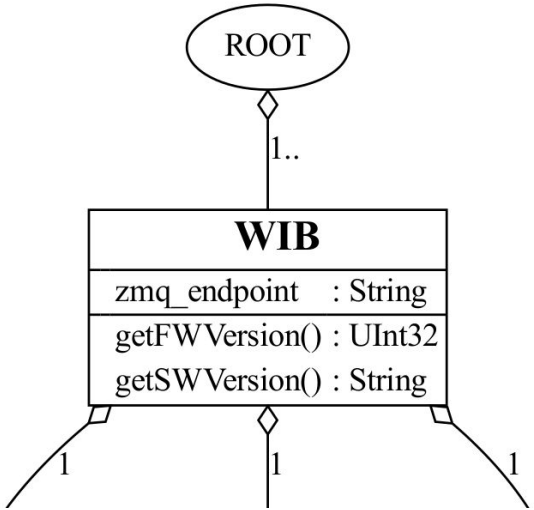

OPC UA
Client
Slow
Controls
WIB
Zynq
OPC UA
Server
OPC UA
ZMQ?
WIB FW
REGs
WIB SW
wib_server
ZMQ
CCM
appfwk
DAQ Module
wibmod
ZMQ
Debugging
Utilities

Overview

- OPC UA is a protocol for clients (control software) and servers (hardware) to interface for commands and status updates
- Giovanna recommended QUASAR for generating OPC UA servers
  - Wraps open62541 (most popular free / open source OPC UA implementation)
  - Used (exclusively) by CERN / one primary developer
- Quasar has some quirks, but easier than using open62541 directly
  - Only serious project documentation is a series of youtube videos on basic topics
  - Complex build system (Python+CMake) that isn't well documented
  - Significant code generation → relatively simple interface to implement
- Generates skeleton of a OPC UA server from XML schema
  - Define "classes" encapsulating different subsystems
  - Define "variables" and "methods" for each class
  - Implement logic in generated C++ code

Server Layout

- WIB class interfaces with wib_server via ZMQ
  - zmq_endpoint specified in the configuration
- Server supports any number of WIBs
  - ROOT represents a server process
  - If running on the WIB, probably just one WIB object
  - If running off the WIB, one process could be several
- Has methods for querying WIB metadata
  - Software and firmware versions
- Contains three child classes
  - FEMBPower, Sensors, and TimingEndpoint

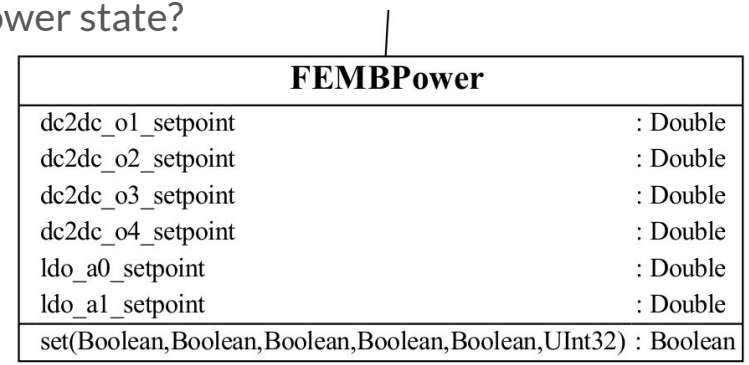


```
<d:class name="WIB">
    <d:devicelogic></d:devicelogic>
    <d:hasobjects instantiateUsing="configuration" class="FEMBPower"
        maxOccurs="1" minOccurs="1">
    </d:hasobjects>
    <d:hasobjects instantiateUsing="configuration" class="Sensors"
        maxOccurs="1" minOccurs="1">
    </d:hasobjects>
    <d:hasobjects instantiateUsing="configuration"
        class="TimingEndpoint" maxOccurs="1" minOccurs="1">
    </d:hasobjects>
    <d:configentry dataType="UaString" name="zmq_endpoint"
        storedInDeviceObject="true">
    </d:configentry>

    <d:method name="getFWVersion"
        executionSynchronicity="synchronous">
        <d:returnvalue dataType="OpcUa_UInt32" name="version"></d:returnvalue>
        <d:documentation>
            Return the firmware build timestamp register
        </d:documentation>
    </d:method>

    <d:method name="getSWVersion"
        executionSynchronicity="synchronous">
        <d:returnvalue dataType="UaString" name="version"></d:returnvalue>
        <d:documentation>
            Return the software git commit hash
        </d:documentation>
    </d:method>

    <d:documentation>
        Interface to the wib_server control software
    </d:documentation>
</d:class>
```

```
/* delegators for methods */
UaStatus DWIB::callGetFWVersion (
    OpcUa_UInt32& version
)
{
    wib::GetTimestamp req;
    wib::GetTimestamp::Timestamp rep;
    if (wib.send_command(req,rep,10000)) {
        version = rep.timestamp();
        return OpcUa_Good;
    } else {
        return OpcUa_Bad;
    }
}
UaStatus DWIB::callGetSWVersion (
    UaString& version
)
{
    wib::GetSWVersion req;
    wib::GetSWVersion::Version rep;
    if (wib.send_command(req,rep,10000)) {
        version = rep.version().c_str();
        return OpcUa_Good;
    } else {
        return OpcUa_Bad;
    }
}
```

Server Layout

- FEMBPower class controls FEMB power regulators
- Contains variables for each regulator setpoint
  - Read/Write by OPC UA clients
- Contains a “set” method to change power state of FEMBs
  - Loads setpoints, changes FEMB power state (4 boolean parameters)
  - Also sets the “warm” or “cold” parameter sets (last boolean parameter)
  - Runs wib_server’s init sequences for different power-on stages (uint32)
- TODO: current power state?

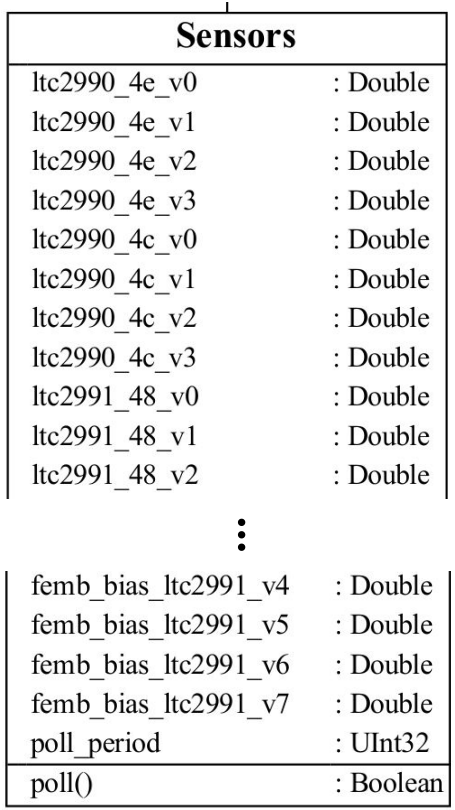


```
<d:class name="FEMBPower">
    <d:devicelogic></d:devicelogic>
    <d:cachevariable initializeWith="configuration"
        dataType="OpcUa_Double" name="dc2dc_o1_setpoint"
        nullPolicy="nullForbidden" addressSpaceWrite="regular">
        <d:documentation>
            Voltage setpoint for DC2DC O1 to be used for next set command
        </d:documentation>
    </d:cachevariable>
    <d:cachevariable initializeWith="configuration"
        dataType="OpcUa_Double" name="dc2dc_o2_setpoint"
        nullPolicy="nullForbidden" addressSpaceWrite="regular">
        <d:documentation>
            Voltage setpoint for DC2DC O2 to be used for next set command
        </d:documentation>
    </d:cachevariable>
    <d:cachevariable initializeWith="configuration"
        dataType="OpcUa_Double" name="dc2dc_o3_setpoint"
        nullPolicy="nullForbidden" addressSpaceWrite="regular">
        <d:documentation>
            Voltage setpoint for DC2DC O3 to be used for next set command
        </d:documentation>
    </d:cachevariable>
    <d:cachevariable initializeWith="configuration"
        dataType="OpcUa_Double" name="dc2dc_o4_setpoint"
        nullPolicy="nullForbidden" addressSpaceWrite="regular">
        <d:documentation>
            Voltage setpoint for DC2DC O4 to be used for next set command
        </d:documentation>
    </d:cachevariable>
    <d:cachevariable initializeWith="configuration"
        dataType="OpcUa_Double" name="ldo_a0_setpoint"
        nullPolicy="nullForbidden" addressSpaceWrite="regular">
        <d:documentation>
            Voltage setpoint for LDO A0 to be used for next set command
        </d:documentation>
    </d:cachevariable>
    <d:cachevariable initializeWith="configuration"
        dataType="OpcUa_Double" name="ldo_a1_setpoint"
        nullPolicy="nullForbidden" addressSpaceWrite="regular">
        <d:documentation>
            Voltage setpoint for LDO A0 to be used for next set command
        </d:documentation>
    </d:cachevariable>
    <d:method name="set" executionSynchronicity="synchronous">
        <d:argument dataType="OpcUa_Boolean" name="femb0_on"></d:argument>
        <d:argument dataType="OpcUa_Boolean" name="femb1_on"></d:argument>
        <d:argument dataType="OpcUa_Boolean" name="femb2_on"></d:argument>
        <d:argument dataType="OpcUa_Boolean" name="femb3_on"></d:argument>
        <d:argument dataType="OpcUa_Boolean" name="cold"></d:argument>
        <d:argument dataType="OpcUa_UInt32" name="stage"></d:argument>
        <d:returnvalue dataType="OpcUa_Boolean" name="success"></d:returnvalue>
        <d:documentation>
            Set the power state of the FEMBs on the WIB using the
            current voltage settings
        </d:documentation>
    </d:method>
    <d:documentation>
        Control the FEMB power subsystem on the WIB
    </d:documentation>
</d:class>
```

```
/* delegators for methods */
UaStatus DFEMBPower::callSet (
    OpcUa_Boolean femb0_on,
    OpcUa_Boolean femb1_on,
    OpcUa_Boolean femb2_on,
    OpcUa_Boolean femb3_on,
    OpcUa_Boolean cold,
    OpcUa_UInt32 stage,
    OpcUa_Boolean& success
)
{
    auto *as = getAddressSpaceLink();
    wib::ConfigurePower conf_req;
    conf_req.set_dc2dc_o1(as->getDc2dc_o1_setpoint());
    conf_req.set_dc2dc_o2(as->getDc2dc_o2_setpoint());
    conf_req.set_dc2dc_o3(as->getDc2dc_o3_setpoint());
    conf_req.set_dc2dc_o4(as->getDc2dc_o4_setpoint());
    conf_req.set_ldo_a0(as->getLdo_a0_setpoint());
    conf_req.set_ldo_a1(as->getLdo_a1_setpoint());
    wib::Status conf_rep;
    if (getParent()->wib.send_command(conf_req,conf_rep,10000)) {
        if (!conf_rep.success()) return OpcUa_Bad;
    } else {
        return OpcUa_Bad;
    }

    wib::PowerWIB req;
    req.set_femb0(femb0_on);
    req.set_femb1(femb1_on);
    req.set_femb2(femb2_on);
    req.set_femb3(femb3_on);
    req.set_cold(cold);
    req.set_stage(stage);
    wib::Status rep;
    if (getParent()->wib.send_command(req,rep,10000)) {
        success = rep.success();
        return OpcUa_Good;
    } else {
        return OpcUa_Bad;
    }
}
```

Server Layout

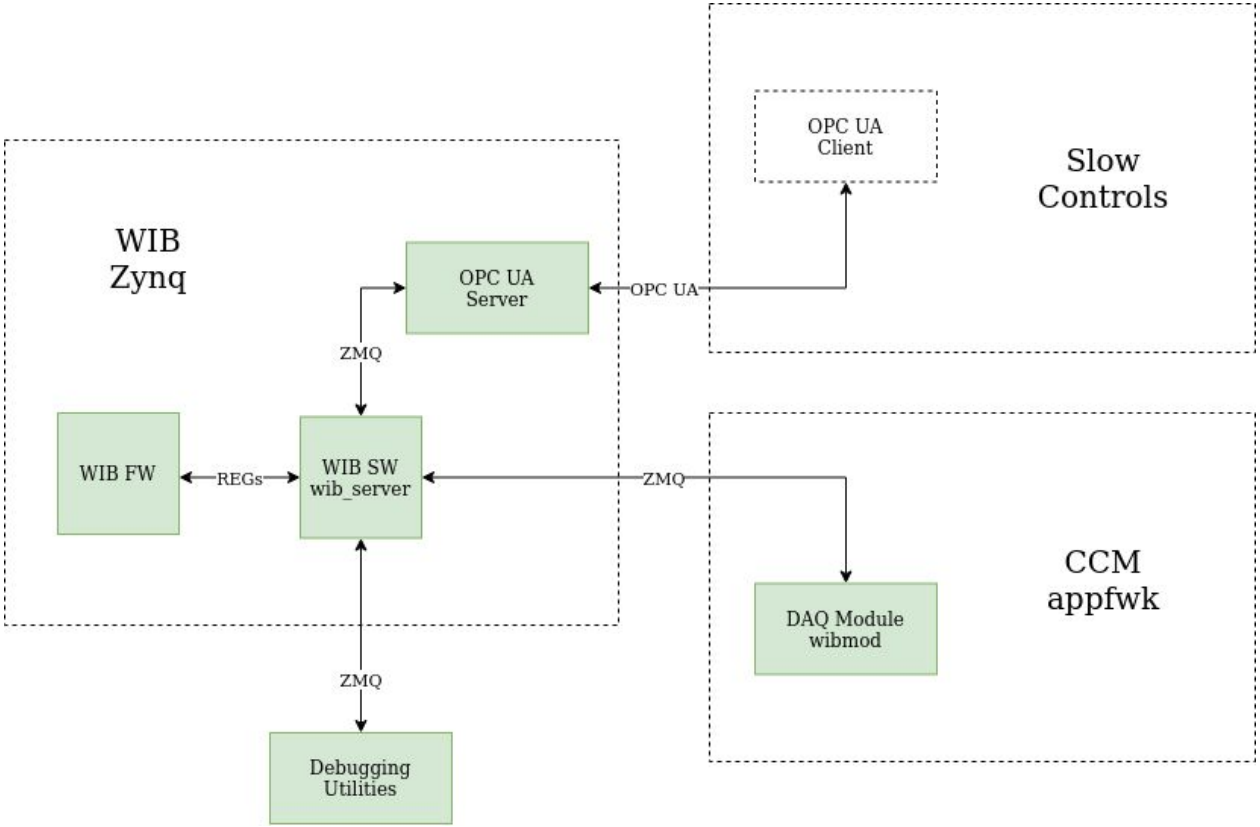
- Sensors class provides access to onboard monitoring
- Contains variables for each sensor value
  - Typically in raw volts
    - i.e. before/after sense resistors
  - Quasar supports “calculated variables”
    - To add scale factors, compute current
    - Only useful if SC needs this
    - Clients can calculate their own scaled values
- Values updated by polling the i2c sensors via wib_server
  - By calling “poll” method as needed
  - By setting an auto poll period (handled by server)



PetaLinux Integration

- QUASAR documentation suggested this worked out-of-the-box
  - Realistically, hadn't been tested in years, and dev doesn't use it
  - Quasar builds and runs nicely within its development environment, not much baked in support for actually installing it
- Spent a couple of days trial/error debugging build system
  - Stripped out several unnecessary parts of Python build code
  - Added several non-existent required dependencies in Petalinux (Yocto)
  - Kludged together an installer script
- Now working on my ARM64 emulator with WIB linux distro
  - Will deploy to UPenn WIB later this week (or soon…)
  - For now, testing with simulated wib_server on x86_64 host

Live Demo!


OPC UA
Client
Slow
Controls
WIB
Zynq
OPC UA
Server
OPC UA
ZMQ
WIB FW
REGs
WIB SW
wib_server
ZMQ
DAQ Module
wibmod
CCM
appfwk
ZMQ
Debugging
Utilities