DUNE Timing System – Single Phase monitoring via DAQ application

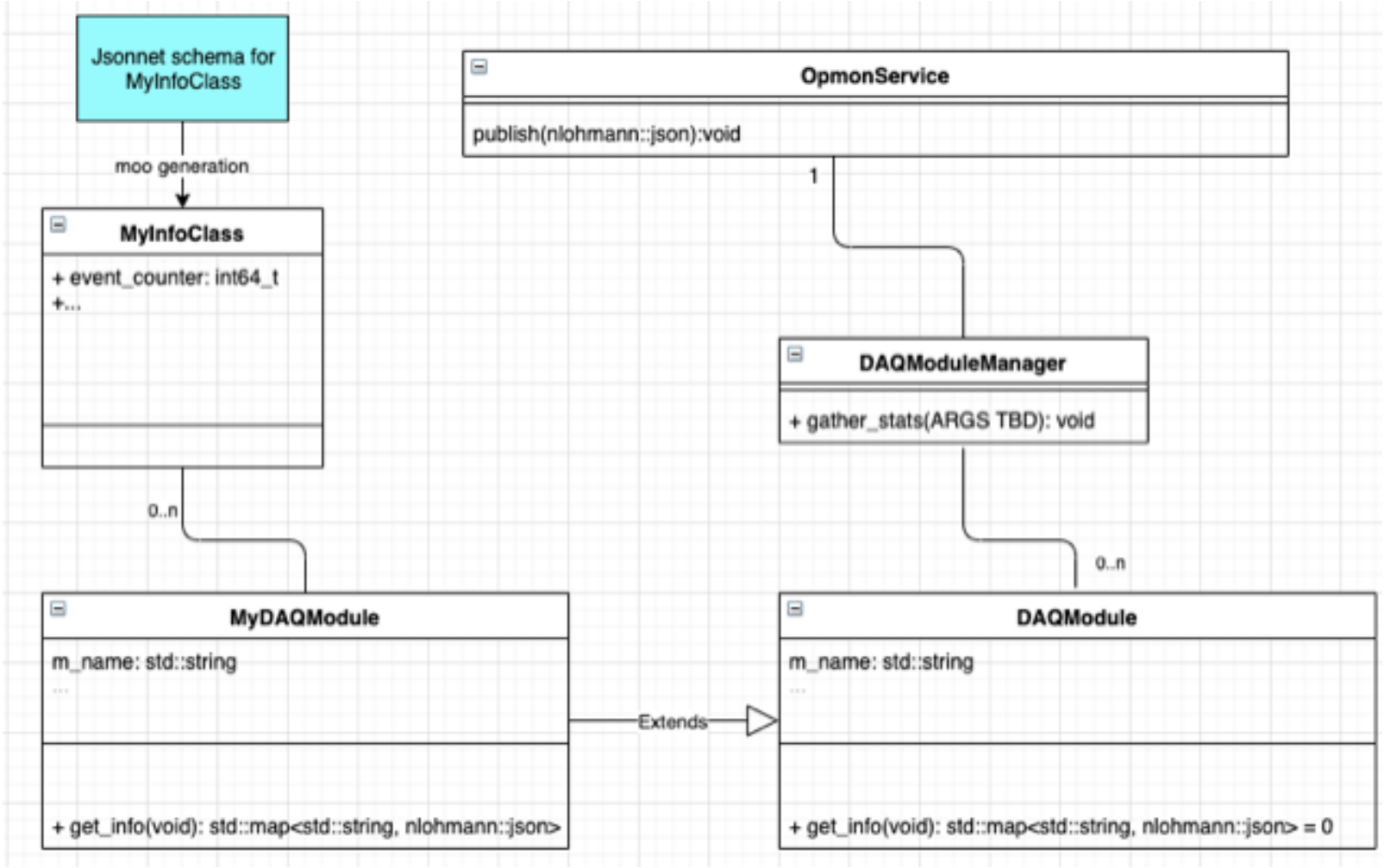
Stoyan Trilov

04/02/2021

University of BRISTOL

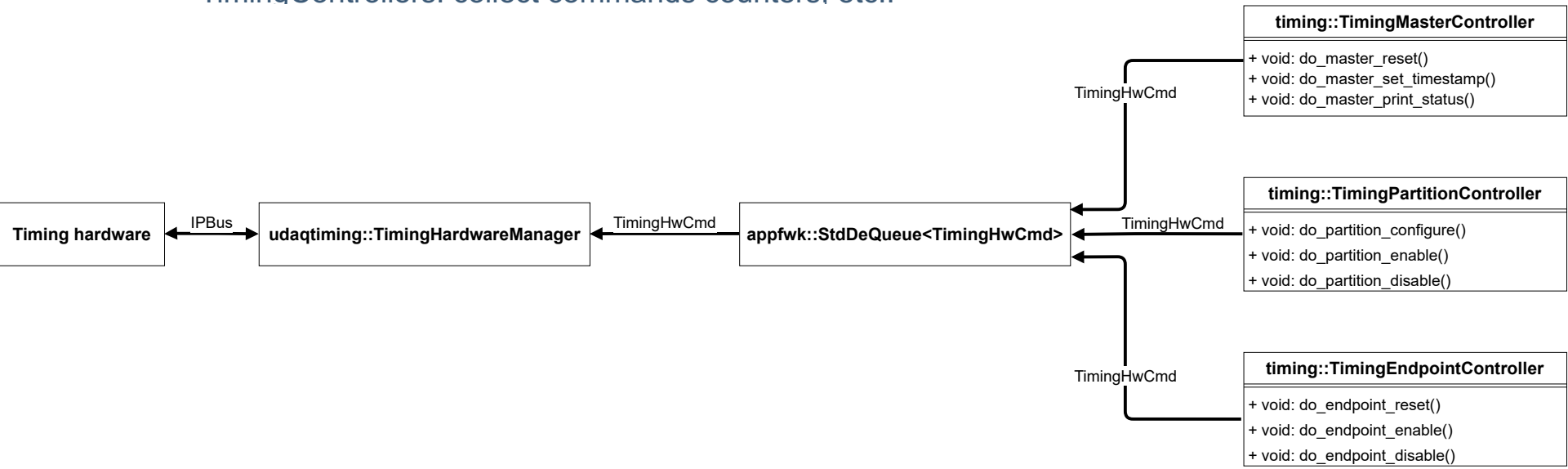
DUNE

Monitoring in DUNE DAQ



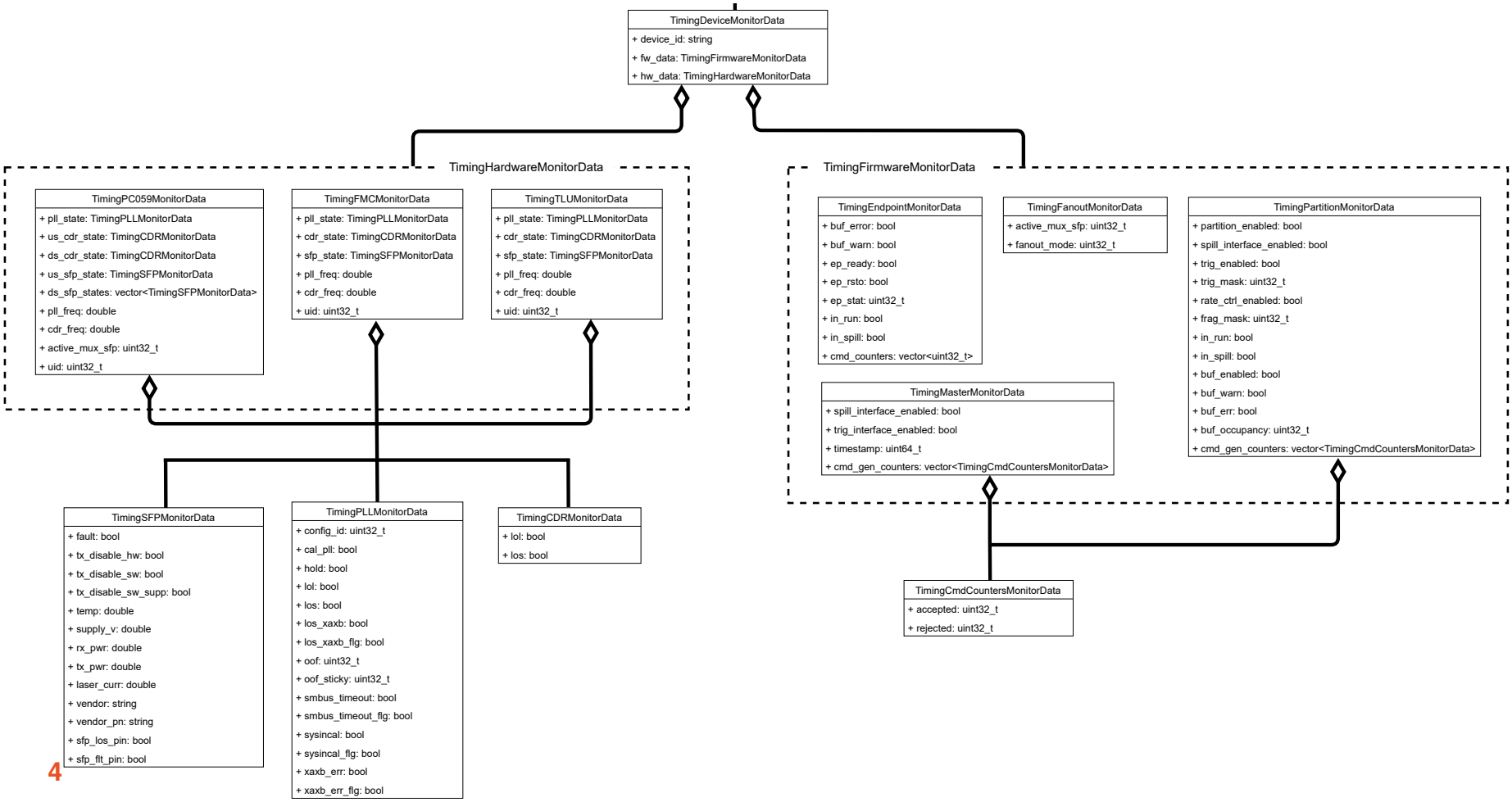
Monitoring in timing DAQModules

- Two types of timing DAQModules implemented so far
  - TimingHardwareManager – makes use of the timing core library to send messages to the hardware
  - TimingController – receives external high-level commands and sends appropriate lower-level commands to TimingHardwareManager. One instance of a controller exists for each object to be controlled, e.g. timing master, endpoint, or partition.
- Function of get_info() in each timing DUNEDAQ module
  - TimingHardwareManager: and collect all hardware information there, along with information relating to TimingHardwareManager itself, e.g. command counters.
  - TimingControllers: collect commands counters, etc..



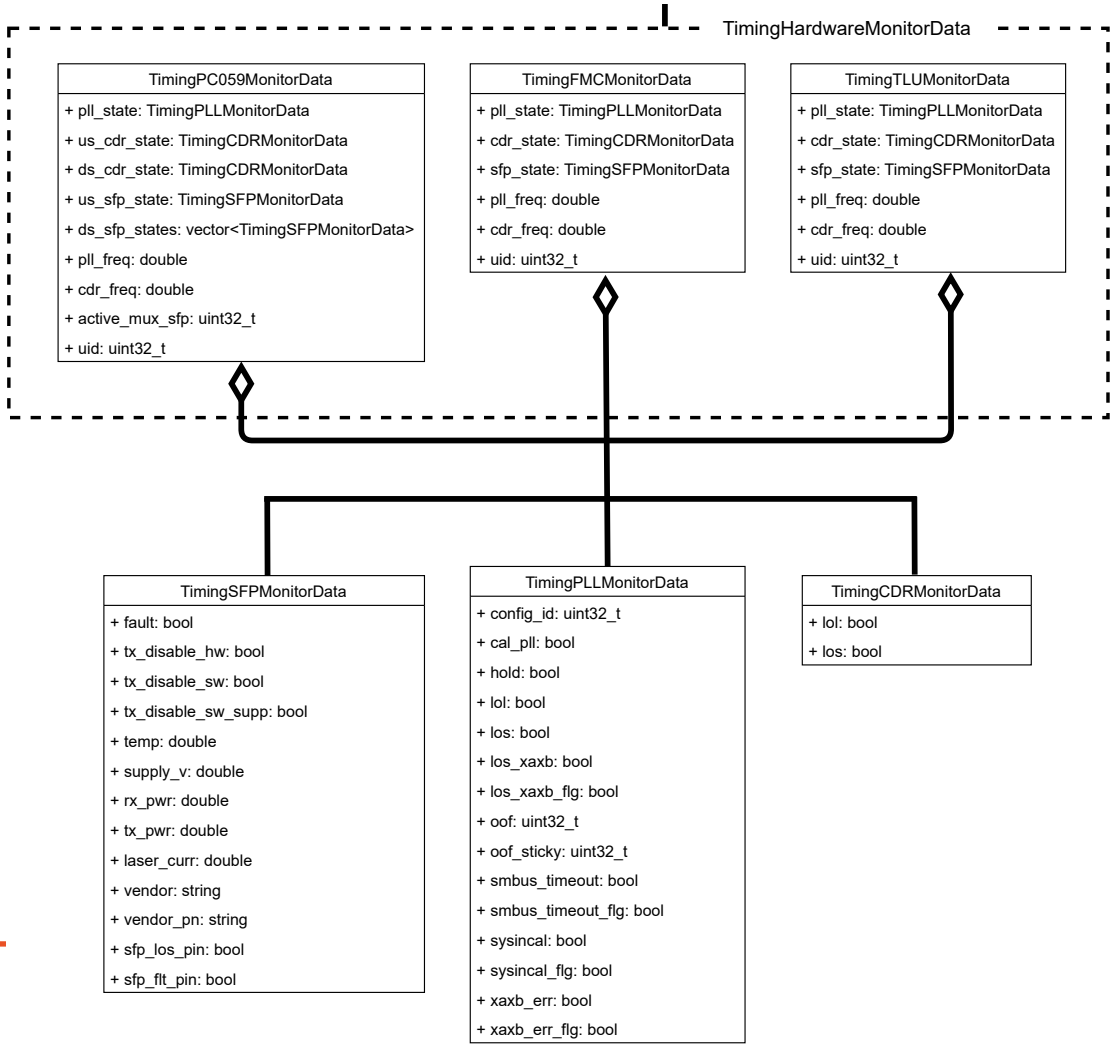
Monitoring of timing devices

- Main timing hardware for MiniDAQ will be the timing FMC
  - Some setups will make use of TLU
  - Model timing monitoring structures on inherent modularity of timing hardware and firmware



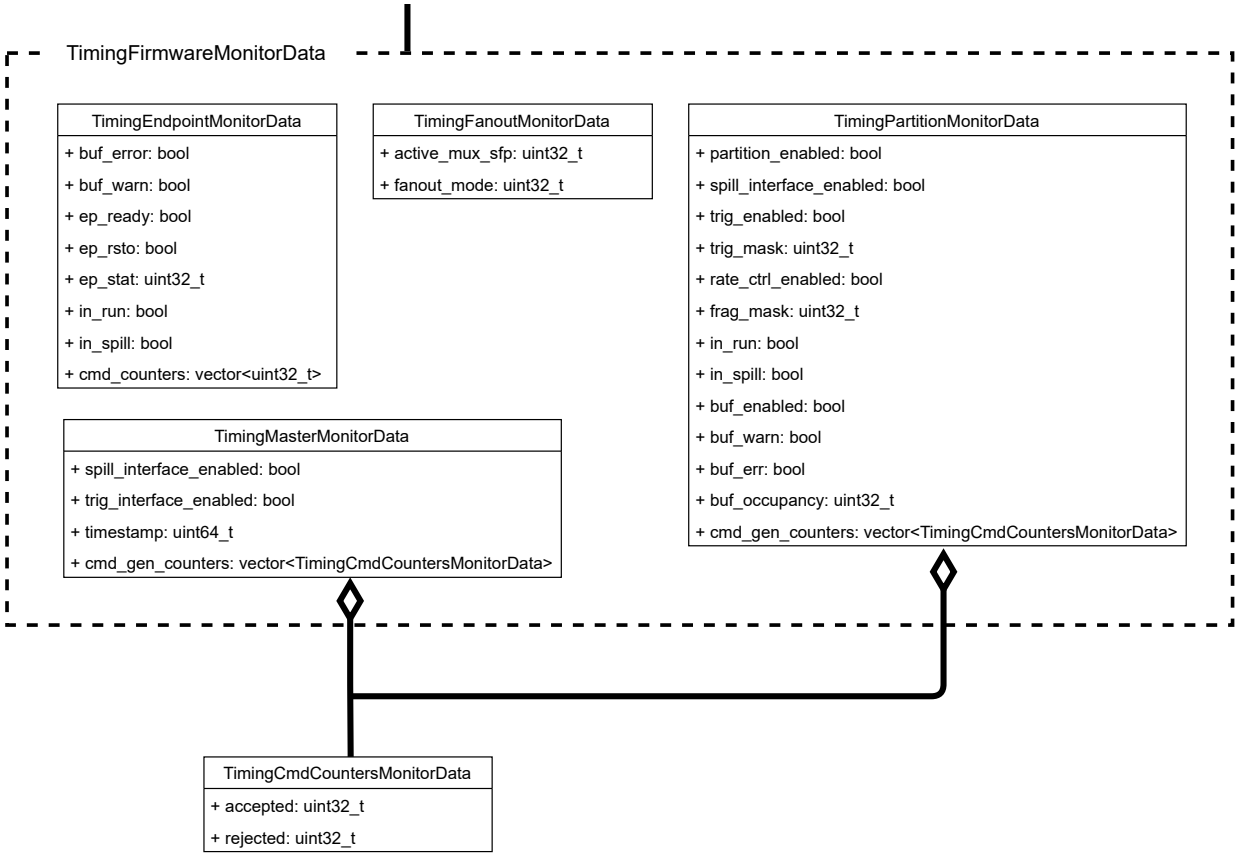
Timing hardware monitoring structures

- Timing hardware boards share a number of common physical components
  - CDR chip, PLL chip, SFP(s), clocks



Timing firmware monitoring structures

- Timing firmware designs share a number of common firmware blocks
  - A master design usually has at least one partition block and at least endpoint block
  - An endpoint design will have at least one endpoint node



Timing monitoring structure generation

Monitoring structures' schema described in jsonnet file

```
pll_config_id: s.string("PLLConfigID", moo.re.ident_only,
    doc="A string field"),

pll_reg_val: s.number("PLLRegValue", "u2",
    doc="A PLL register bit(s) value"),

timing_pll_mon_data: s.record("TimingPLLMonitorData",
[
    s.field("config_id", self.pll_reg_val,
            doc="PLL config ID"),
    s.field("cal_pll", self.pll_reg_val,
            doc="Cal pll"),
    s.field("hold", self.pll_reg_val,
            doc="Holdover flag"),
    s.field("lol", self.pll_reg_val,
            doc="Loss of lock flag"),
    s.field("los", self.pll_reg_val,
            doc="Loss of signal flag"),
    s.field("los_xaxb", self.pll_reg_val,
            doc="Loss of signal flag XAXB"),
    s.field("los_xaxb_flg", self.pll_reg_val,
            doc="Loss of signal flag XAXB stricky"),
    s.field("oof", self.pll_reg_val,
            doc="Out of frequency flags"),
    s.field("oof_sticky", self.pll_reg_val,
            doc="Out of frequency flags sticky"),
    s.field("smbus_timeout", self.pll_reg_val,
            doc="SMBUS timeout"),
    s.field("smbus_timeout_flg", self.pll_reg_val,
            doc="SMBUS timeout sticky"),
    s.field("sysincal", self.pll_reg_val,
           doc="In calibration flag"),
    s.field("sysincal_flg", self.pll_reg_val,
            doc="In calibration flag sticky"),
    s.field("xaxb_err", self.pll_reg_val,
            doc="XA-XB error flag"),
    s.field("xaxb_err_flg", self.pll_reg_val,
            doc="XA-XB error flag sticky"),
],
doc="Timing PLL monitor structure"),
```

Generate C++ header file from jsonnet file

Generated C++ header file

```
// @brief A string field
using PLLConfigID = std::string;

// @brief A PLL register bit(s) value
using PLLRegValue = uint16_t;

// @brief Timing PLL monitor structure
struct TimingPLLMonitorData {

    // @brief PLL config ID
    PLLRegValue config_id = 0;

    // @brief Cal pll
    PLLRegValue cal_pll = 0;

    // @brief Holdover flag
    PLLRegValue hold = 0;

    // @brief Loss of lock flag
    PLLRegValue lol = 0;

    // @brief Loss of signal flag
    PLLRegValue los = 0;

    // @brief Loss of signal flag XAXB
    PLLRegValue los_xaxb = 0;

    // @brief Loss of signal flag XAXB stricky
    PLLRegValue los_xaxb_flg = 0;

    // @brief Out of frequency flags
    PLLRegValue oof = 0;

    // @brief Out of frequency flags sticky
    PLLRegValue oof_sticky = 0;

    // @brief SMBUS timeout
    PLLRegValue smbus_timeout = 0;

    // @brief SMBUS timeout sticky
    PLLRegValue smbus_timeout_flg = 0;

    // @brief In calibration flag
    PLLRegValue sysincal = 0;

    // @brief In calibration flag sticky
    PLLRegValue sysincal_flg = 0;

    // @brief XA-XB error flag
    PLLRegValue xaxb_err = 0;

    // @brief XA-XB error flag sticky
    PLLRegValue xaxb_err_flg = 0;
};
```

Monitoring next steps

- Capture all monitoring structures in schema jsonnet file
  - This will provide the C++ structures

- Implement code to fill these structures
  - Code in timing core library and hardware interface DUNEDAQ module to retrieve data and fill structures
  - get_info() code in timing DUNEDAQ modules

- Test calling get_info()
  - How?