# Specification and Prototyping for Design of a Control System for Proton Computer Tomography.

A thesis by

Aleksei Kuleshov

for degree of

Master of Science in Physics

Department of Physics and Technology
Faculty of Mathematics and Natural Sciences

November 2019

# Abstract

Proton Computer Tomography (pCT) is a new imaging modality. It based on proton with matter interaction. Treatment with protons is used already against cancer, as it offers the possibility to give a high dose to the tumour, while at the same time deposit a very low dose to the surrounding tissue. Imaging with protons improves the accuracy of the dose planning. However, there are no commercial pCT scanners available in the market today.

At the Department of Physics and Technology at the University of Bergen, a new generation of a Digital Tracking Calorimeter (DTC) is designed. It is a significant part of the pCT system. It is a highly complex device designed with 41 layers of ALPIDEs, a Monotholitic Active Pixel Sensor. It is more than 4000 ALPIDEs in the design, and they will be read out and controlled by 41 custom readout units. A control system is required, both to configure the system and to monitor vital health parameters such as temperatures, current and voltages during operation. During this thesis, prototyping has been done to find the best solution. At the time of writing, the DTC is still under development, so the work covers tests of communication protocols and prototyping of a monitoring system with randomly generated data.

Various options of communication protocols for the pCT control system has been tested in this thesis.The work of this thesis has shown that the best option for the pCT control system will be the use of OPC UA communication protocol as it is a standard industrial protocol of communication. An OPC UA information model for the pCT has been developed. The XML design file has been imported to a commercial SDK.

# Acknowledgments

First of all, it was an excellent opportunity to come to the Department of Physics and Technology at the University of Bergen. I have a physical background from Saint-Petersburg's State University and perfect memories about people and the atmosphere around science. I'm more than satisfied with the time spent on my studies. I wished to do valuable work for science and society. I've learned a lot as it was my goal.

I want to thank associate professor Johan Alme and professor Kjetil Ullaland for their supervision, Ola Grøttvik , Magnus Rentsch Ersdal, Simon Voigt Nesbø, and Ganesh Jagannath Tambave and my fellow students.

I appreciate my family and sibling for support during my thesis. Without them, this would not be possible.

# Contents

# Glossary

| | |
|---|---|
| **ALPIDE** | Alice Pixel Detector |
| **ALICE** | A Large Ion Collider Experiment |
| **ADC** | Analog to Digital Converter |
| **CT** | Computer Tomography |
| **CMU** | Control Management Unit |
| **DAC** | Digital to Analog Converter |
| **DB** | Data Base |
| **DCS** | Detector Control System |
| **DMU** | Data Management Unit |
| **DTC** | Digital tracking colorimeter |
| **DTU** | Data Transmission Unit |
| **FROMU** | Framing and Management Unit |
| **FSM** | Finite State Machine |
| **ICS** | Industrial Control Systems |
| **IoT** | Internet of Things |
| **IP** | Internet Protocol |
| **ITS** | Inner Tracking System |
| **FPGA** | Field-Programmable Gate Array |
| **GUI** | Graphical User Interface |
| **HMI** | Human-Machine Interface |
| **HU** | Hounsfield Unit |
| **LET** | Linear Energy Transfer |
| **LHC** | Large Hadron Collider |
| **LS2** | Long Shutdown two(in LHC) |
| **MEB** | Multi Event Buffer |
| **OS** | Operating System |
| **pCT** | proton Computer Tomography |
| **PID** | Proportional-Integral-Derivative |
| **PLC** | Programmable Logic Controller |
| **pRU** | proton Readout Unit |
| **RSP** | Relative Stopping Power |
| **RTOS** | Real Time Operating System |
| **RU** | Readout Unit |
| **SCADA** | Supervisory Control and Data Acquisition |

| | |
|---|---|
| **SOBP** | Spread Out Bragg Peak |
| **UiB** | University of Bergen |

# Chapter 1

# Introduction

## 1.1 Proton therapy and proton CT

Proton therapy is a way of irradiation treatment of diseased tissue for medical purposes, most often in regards to cancer. This is a particular case of charged particle irradiation when protons are used. The main advantage of proton and other ion therapy over traditional radiotherapy using gamma radiation is that the most dose of a charged particle is deposited in a narrow range in the tissue. This region is known as the Bragg peak after William Henry Bragg who discovered it in 1903.

The Bragg Curve is a graph of the energy loss rate, or Linear Energy Transfer (LET) as a function of the distance through a stopping medium. For protons, the peak occurs immediately before the particles come to rest. This peak is about two-three times higher than the LET at the entering of a stopping medium. Heavier ions have a longer tail after the peak [1]. It makes protons superior due to sharp transition from the peak of LET to zero LET at the depth which is proportional to the initial energy of the proton beam.

In Figure 1.1 the difference between 10 MeV photon and a single proton beam can be observed. A typical treatment planning for proton therapy is initiated by the determination of a tumor area. In Figure 1.1 this is between dotted lines, depth from approximately 55 to 145 mm. On the vertical axis dose in percentage, where 100% is the desired dose in the tumor area. In proton treatment the deposition of dose has a maximum near the surface, with a slow dacey with depth. Proton beam of a given energy, instead, has peak and sharp decay (pristine peak on Figure 1.1). For achieving a 100 % dose in the desired area, several proton beams with different energies are used (blue lines under the pristine peak in Figure 1.1, so combined dose from these beams gives spread out Bragg Peak (SOBP) region which covers tumor area. SOBP keeps sharp decay of the single proton beam, which makes it superior in treatment near to vital organs.

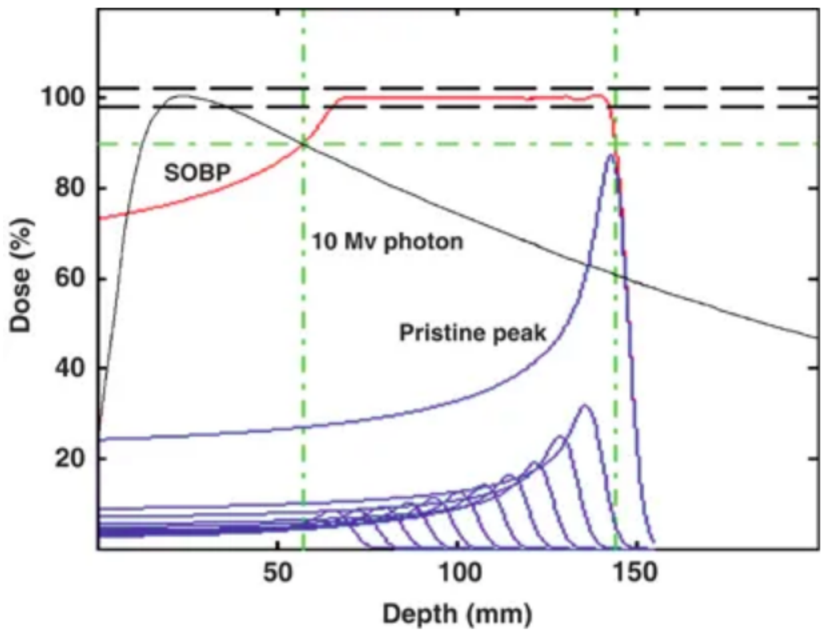


Figure 1.1: A comparison of 10 MeV photon (black) with spread-out Bragg peak (SOBP, red)[2]

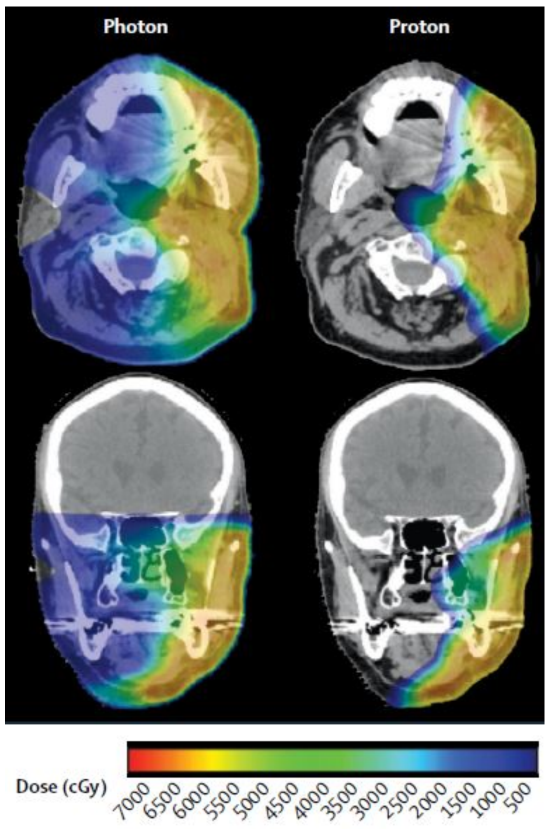


Figure 1.2: A comparison of treatment plans for head cancer. Photon is on the left, protons on the right. [5]

Studies shows that the mean and integral doses were lower for protons[3]. Thus fewer side effects are expected for proton therapy. The biological effect on cancerous cells is two to three times higher [4]. Treatment plans with protons have trade-offs between keeping low unwanted damage in shallow tissues and reaching the desired dose it tumor area. Often less than the desired dose is used in photon treatment to avoid damage in healthy tissues and avoid the side effects of radiation treatment.

Figure 1.2 shows a cross-section of a head for photon and proton treatment plans. The blue regions in the proton treatment plan are significantly smaller than for photons. This indicates that the rest of the body will get close to zero dose. It means that transition in dose between healthy tissue and tumor can be made very steep in proton treatment. Thus the ultimate goal of cancer treatment to keep margins as small as possible can be achieved. Proton therapy today is performed by precalculated dose based on X-ray computer tomography (CT) images. The X-ray CT images reflect the patient's anatomy in therm of photon interaction with matter, not protons.

The Relative Stopping Power (RSP) for protons in tissue is needed to calculate the proton range during dose calculations. The RSP maps can be obtained by converting attenuation of x-rays, represented by Hounsfield Unit (HU) maps in tissue, to RSP maps in the same tissue. Figure 1.3 shows conversion curves which are non-linear and obtained by end-to-end tests using an anthropomorphic phantom containing tissue-equivalent materials and dosimeters. This conversion procedure introduces range uncertainties in the order of 2–3%, corresponding to 4–6 mm at treatment depth 20 cm into the patient [6].

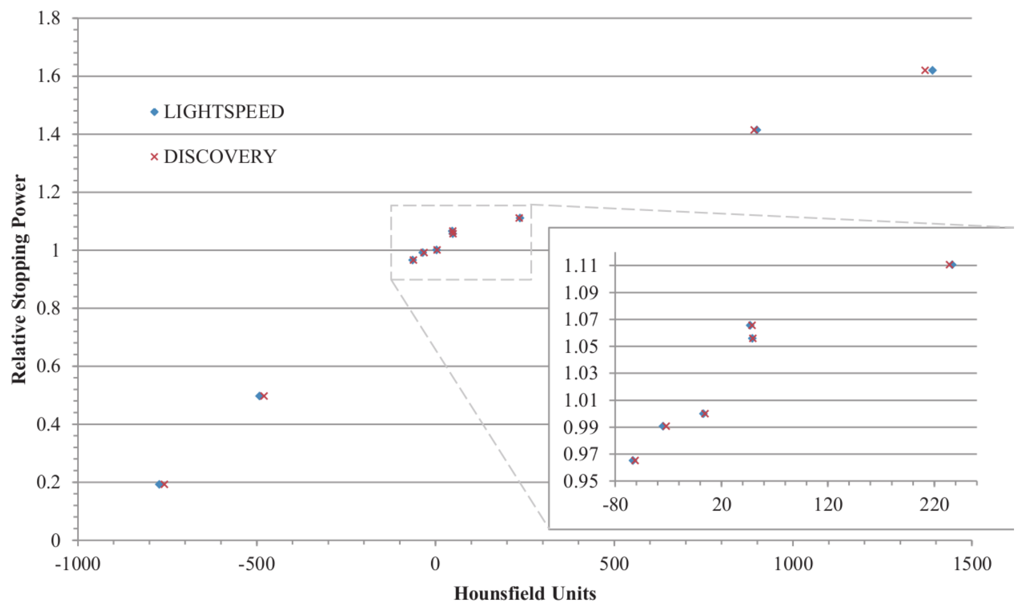


Figure 1.3: Relative Stopping Power to Hounsfield Unit calibration curve for LightSpeed and Discovery CT Scanners [7]. The zoomed area highlights non-linear behavior of calibration points which introduce uncertainties in the conversion from HU to RSP.

Proton CT addresses the problem of the conversion between HU and RSP. Currently the patient has to move from photon scanner to proton treatment. It would be beneficial to perform proton scan and irradiation at the same place. The difference between proton CT scans and proton therapy is only the energy of the protons in the beam. Thus proton CT images would eliminate positioning uncertainties from conversion between HU maps to RSP maps. To solve this task, a special detector is needed to measure the energy and direction of each proton passed through tissue.

## 1.2 Proton CT at UiB

At the Department of Physics and Technology at the University of Bergen a new generation of the Digital Tracking Calorimeter (DTC) is designed, which is a major part of the proton CT system. A proof-of-concept of the DTC for proton CT has already been made with the previous generation of sensor chips[8]. A complete proton CT system will consist of the DTC, readout electronics, cooling, power, and control systems. The design of the DTC will only be briefly described here. Mechanical drawing is shown in Figure 1.4.

### The digital Tracking Calorimeter (DTC)

The proof-of-concept of the DTC was originally dedicated to the ALICE-FoCal project, where the reconstruction of high-energy electromagnetic showers is needed. The advantages of DTC for proton CT usage was recognized later. The DTC allows not only to reconstruct individual proton tracks but also to measure the energy of protons after passing through an object. It is also substantially faster than using traditional calorimeters.

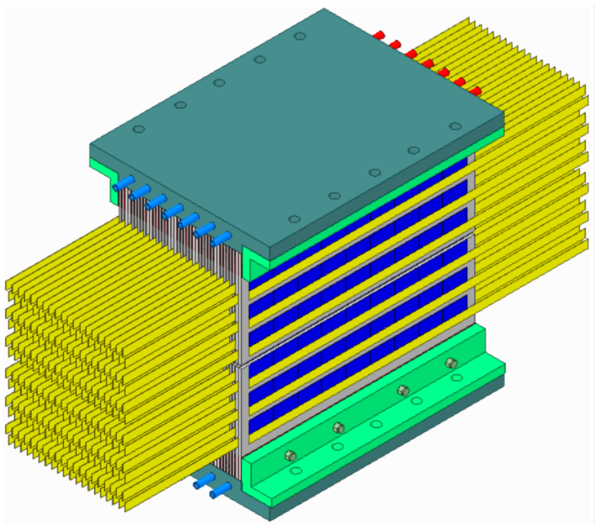


Figure 1.4: 3D view of the DTC [14].

It has 41 layers. Each layer has divided horizontally into two parts and has nine horizontal oriented ALPIDEs in a row is stacked in 12 columns. Sensor chips - ALPIDEs (dark blue) arranged in staves with a cable (yellow). The staves altered on both sides of the absorber and all staves on the front surface have cables on the right. Staves of the rear surface have cables on the left. At the top of the DTC, there are cooling inlets (light blue) and outlets (red). There is an air gap 2 mm between layers [14].

## 1.3 The ALPIDE

The ALPIDE chip is a state-of-art detector chip for particle detection. It makes the ALPIDE an ideal candidate for use in proton CT. It is not so many special chips for particle detection, so newest ALPIDE developed for CERN stand out it term of specifications among them.

### 1.3.1 The ALICE experiment and ITS upgrade

The ALICE(A Large Ion Collider Experiment) is a general-purpose heavy-ion detector at the CERN LHC. It is designed to study the physics of strongly interacting matter and the quark-gluon plasma at the high temperature and density in nucleus-nucleus collisions. The first ideas for the ALICE detector were formulated during a workshop in 1990. The design was approved in 1997[15].

ALICE Inner Tracking System (ITS) is the subdetector closes to the Interaction Point. Its next upgrade is planned on during 2019-2020 (second long shutdown of LHC). It is currently being upgraded and the new ITS will consist of: Inner Barrel (IB) - three layers and Outer Barrel (OB) - four layers. The new layout is shown in Figure 1.5. The detector layers will be installed on distances from 22 mm to 400 mm around interaction point. All layers are using ALPIDE as the sensor chip. The only difference in readout connection. The further layer from collision point, the less occupancy or particle detection is expected. After start of LHC in 2021 statistics will increase by factor of 50-100 [17].

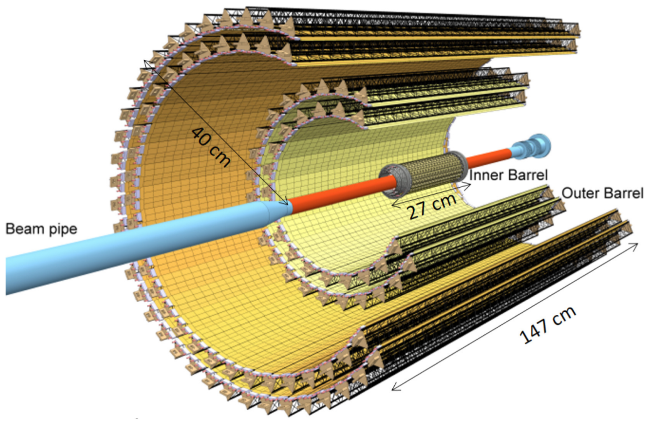


Figure 1.5: ALICE new ITS layout. [16].

## 1.4 The need of a control system

Control systems are playing a key role in complex systems nowadays, where data interaction is essential. Data exchange is what drives today's progress of the Internet of Things (IoT) and Industry 4.0. It will allow a system to make decisions (change behavior) according to incoming data. Design of a control system covers the development and planing of the data path for all sensors and subsystems. In many cases it is also influences the hardware. A control system commands or regulates the behavior of other devices or subsystems. Analysis of all parameters in real-time is also a task of a control system.

Almost every chip or microcontroller needs configuration. This configuration can be stored localy on an onboard flash or can be loaded at the power-on cycle. In systems with many similar components, it is common to store configuration in a database. It allows to easily update the configurable parameters. To get a device or just one chip in a ready state, a special procedure should be followed to avoid damage to the electronics inside a device.

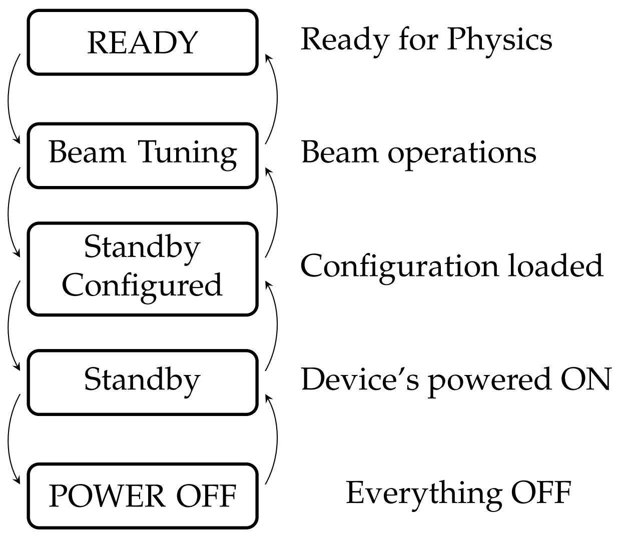


Figure 1.6: State diagram of ALICE DCS[9].

A general state diagram for ALICE detector shown on Figure 1.6. In the such complex system, verification must be done before state transitions. Each arrow on Figure 1.6 represents some actions and verification that this state is safe. During power on it can be: starting cooling system, power supply voltage check, etc. After the configuration is loaded and tests passed device is ready for operation. In radiation environments is common to update/reload configuration at the background to protect from accidental memory changes. For pCT has been chosen to write the whole configuration before each measurement, because errors are expected to happen very often. On the fault conditions (fatal error), action has to be done immediately to come to the safe state. It can be Standby or Power off.

Complex systems, like the ALICE detector, are so big, so it not possible to have an operator who has professional knowledge about all subsystems. Thus the ultimate goal for the development of a such control system is to create it such that an non-expert can run it after some training. ALICE Detector Control System (DCS) is made in this way. To achieve this, ALICE DCS has an alert handling where actions on different alert levels have specified and are part of the control system. In a situation where expert knowledge is required, a part of the system can be delegated to an expert to control under special conditions [9].

## 1.5 Objective of this thesis

The proton CT prototype developed at the Department of Physics and Technology at the University of Bergen will require a dedicated control system. The prototype has been described in detail in Section 1.2. The detector has 41 layers with 108 ALPIDEs on each. Every layer has own Readout Unit (RU). Now, in the development phase, the RU is prototyped on a Xilinx VCU118 development board. In an earlier master thesis [22], FreeRTOS system has been installed on a soft-core processor programmed inside the FPGA. The read, write operations and readout of a simulated hit from a single ALPIDE have been tested.

The specification of a control system is the next logical step forward to complete the pCT prototype. During this thesis, requirements for the complete system is formulated, various technologies are examined, and one or more solutions are proposed and validated. The final control system has to be able to perform configuration and monitoring tasks primarily for the ALPIDE chips and front-end electronics. Ability to connect future parts into pCT and easy scaling of the system has to be possible without fundamentally changing og the software or hardware.

A sketch of the relation between the general components in the monitoring system is shown in Figure 1.7. ALPIDEs are unique in the sense of bad pixels masks, ADC calibration coefficients, etc, so a configuration Data Base (DB) is needed. Logging of monitored parameters allows analyzing historical data. An example: to find typical operational values and adjust alarm and warning thresholds in monitoring software.

A state machine diagram for the pCT detector with ALPIDEs is very similar to Figure 1.6. Monitoring software has to control transition between states, read-write monitoring information, and make sure that RU with all ALPIDES is in safe operational mode. Some protection can be done on the RU locally to satisfy the required reaction speed. In addition to the main software, it would be beneficial to be able to run other copies on remote computers for monitoring purposes with restricted or closed access for the write operation to ALPIDEs and RU.

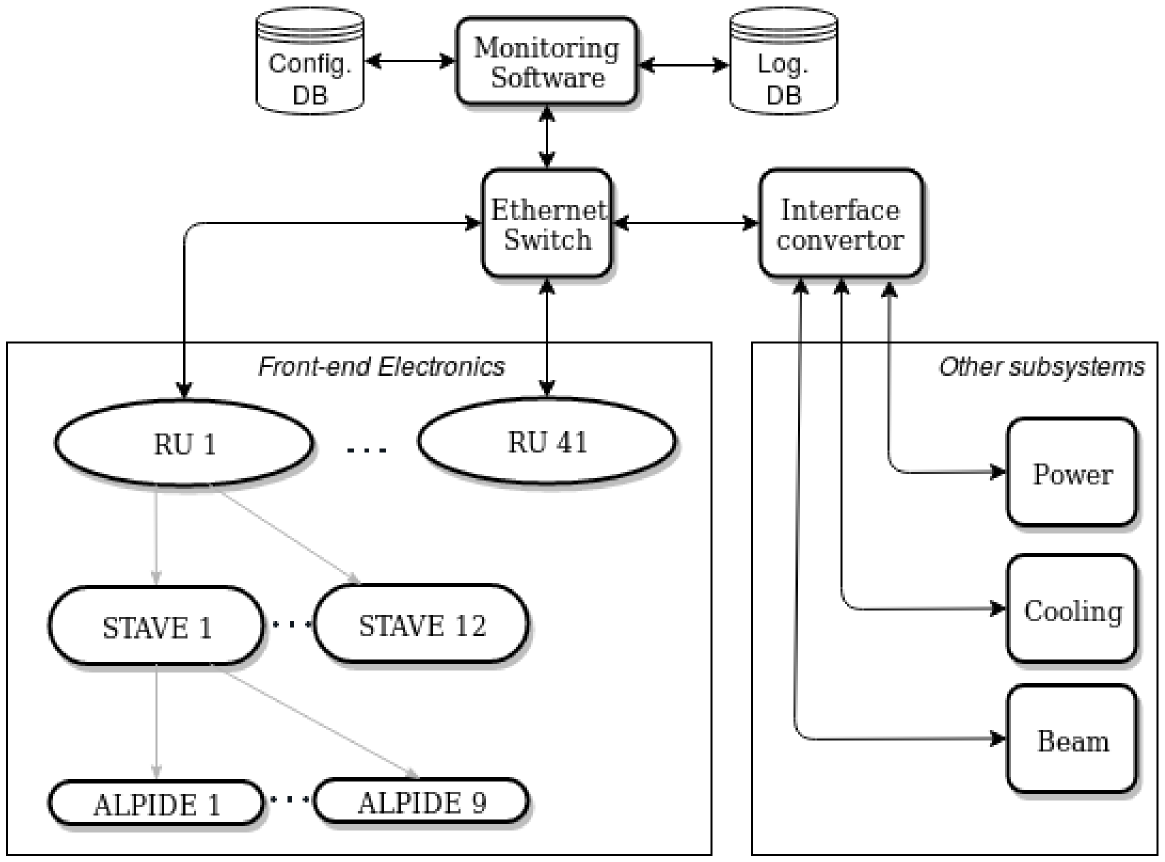


Figure 1.7: A sketch of the relations in the pCT monitoring system.

## 1.6 Overview (Thesis structure)

**Chapter 1 Introduction** is covering the background for this thesis and describe some technical aspects in details needed for understanding this work. Objective of this thesis is given at the end of the chapter.

**Chapter 2 Control Systems** highlight technologies which are possible candidates for monitoring system and have been evaluated to make basis for choice of the solution. Finally, the chapter will conclude with an example of an existing control system, the ALICE Detector Control System.

**Chapter 3 Proton CT Readout Unit** describes the proton CT Readout Unit, the ALPIDE chip and FreeRTOS. Proposal of the whole pCT electronics system is given as block diagram.

**Chapter 4 Requirements for a pCT Control System** Here requirements are formulated for finding proper candidates and making decision later.

**Chapter 5 Prototyping** Consist of description development of prototypes for a control system for pCT. The chapter covers implementations of MQTT and OPC UA thechnologies.

**Chapter 6 Proposal of Production Version Control System** In this chapter possible solutions for pCT have been described. MQTT and OPC UA have considered separetly for design of control system for pCT. An option have been discussed where both MQTT and OPC UA is used.

**Chapter 7 Conclusion and future work** summarize all result and gives proposal for what can be done next.

# Chapter 2

# Control Systems

In this section, control systems in different applications will be described. It will give a good overview of modern technologies and make requirements for a proposal for the control system of the pCT. Finally, the chapter will conclude with an example of an existing control system, the ALICE Detector Control System.

## 2.1 SCADA

The SCADA concept was developed as a universal means of remote access to a variety of local control modules, which could be from different manufacturers, allowing access through standard automation protocols. In practice, large SCADA systems have grown to become very similar to distributed control systems in function. They can operate large-scale processes that can include multiple sites and work over large distances.

Supervisory control and data acquisition (SCADA) covers the need to monitor and control remote gear since the 1960s. A SCADA control system architecture uses computers, networked data communications, and graphical user interfaces (GUI) for high-level process supervisory management and other peripheral devices such as programmable logic controller (PLC) and discrete proportional– integral–derivative (PID) controllers to interface with the process plant or machinery. The SCADA system for ALICE experiment is shown in Figure 2.1.

SCADA, according to *IEEE Standard for SCADA and Automation Systems*, is a concept for a control system. The standard covers most of the aspects which designer will face during the development of a SCADA system. However, "The designer/specifier shall specify the protocol(s) required...". The word "designer/specifier" is used many times in SCADA specification to point attention to that it is the designer's responsibility to choose technology, protocols, physical connections, functional elements, and the logical operation of the whole system. The standard does not force to use specific technology or protocol. Choice of technology is the key to success.Scalability and integrability play a crucial role, allow to connect hundreds or thousands of devices and smoothly exchange data between them. Also, efforts are used to make maintenance and future upgrades easy.

Typically SCADA systems are used in Industrial Automation for lighting, power, oil and gas industry, traffic control and lights, pharmaceutic and food production, water processes, etc. Prominent vendors like ABB and Siemens offer their own SCADA products (MicroSCADA and WinCC, respectively). These give designers and users an abstraction from communication protocols and simplify development and usage. It can be an advantage in some cases, but in pCT with a lot of custom hardware and software it will probably not provide such an advantage.

In the process industry, a production process typically automated. The control system concept covers everything from the way humans interact with the system to data formats in all internal communications. Control systems can be spread over big areas. Then often, local nodes have some automation logic and measurement data sent to the central control room. For example, the oil and gas industry has many cites where no people are present. With technology progress, data communication over distances became more affordable, and the Internet of Things (IoT) lead to growth in distributed systems and remote control. It allows to concentrate at one place information (data from all sensors, process parameters, and variables) and one or more operators.

### 2.1.1 Industrial Control systems

The key factor for industrial control systems (ICS) is the ability to use standardized elements to build a complete custom system. Time to market and reliable operation is essential for Industry. Due to the industry progress, all elements that are made for control have become more flexible and programmable. Time to market is vital, so the availability of the components "off the shelf" is one of the requirements. Nevertheless, the complexity of the total system is growing, and it has to be solved by programming and configuring.

The main benefits of an efficient process control system are energy savings - energy wastage is reduced and improved safety - control systems automatically warn about abnormalities. These benefits can be easily converted in money and evaluated for each particular case.

Ability to collect and store a big amount of data gives now opportunity to statistical analysis, so makes it a control system provide better performance, quality of the product, easier to predict failures. Also, analysis of collected data is used to improve the system by finding subsystems which limit performance or quality.

## 2.2 Distributed Control Systems

In addition to the SCADA concept, term Distributed Control Systems (DCS) is often used. Distributed Control Systems[1] are similar to SCADA. The difference is that DCS means more control function is laid on region units, when in SCADA system control is more centralized. At the beginning of the SCADA, all control was centralized, but later with the progress of electronics control loops of process variable became independent, and only set points are controlled remotely. In Industry SCADA and DCS are synonymous, because a top-level control unit always exists for human interaction.

DCS, however, is solely used when no central control is present. It means that every unit control their own behavior based on logic inside and some communication or sensing environment around. Examples from nature are birds, fish, and other animal's behavior in the absence of a leader. In automation, some analogy can be found in load balancing during computation.

## 2.3 OPC UA

OPC UA is a communication protocol for industrial automation. It has been built to satisfy needs in machine to machine communications, robotics and sophisticated automation. OPC UA is more than just standard of communication. It has its own binary protocol but can use other transport protocols like MQTT and UDP. It has object-oriented ideology and is OS agnostic. OPC UA protocol is often used for communication in SCADA systems.

OPC UA is a major upgrade of OPC classic. OPC classic was developed for Microsoft Windows. It has specifications for Data Assess, Alarms & Events and Historical Data Access. The OPC UA protocol, however, is quite complex: the primary documentation is in 15 parts, and over 1000 pages. The core idea is to have an OPC UA server near to a source of information and client that can read, write to variables, run methods on the OPC UA server and get results, subscribe on value changes and get notifications about events and alarms. OPC UA gives much more semantic than classic OPC and other transport protocols.

An OPC UA server has address space (*AddressSpace*) where all information is organized by hierarchical and non-hierarchical links. The idea to present information with meta-information together with semantics that a client without prior knowledge can utilize data by using a discovery service.

Table 2.1 gives an overview over node types in OPC UA. Nodes can be objects, data types, methods, events or references. The references are used to describe relations and organize a system. Events can be different types. A hierarchical type is used to make hierarchy between objects when they are defined separately or even

[1]DCS will be used here, please note that ALICE DCS is used for ALICE Detector Control System

in different OPC UA servers. Objects, data types, methods and events represents functionality of an physical object.

| **Node type** | **Defines** | **Examples** |
|---|---|---|
| Object | A physical entity or application | Robot, Motor, Pump |
| Data Type | A property of the Object | Position, Speed, Setpoint |
| Methods | Commands that are exposed to Clients | Start, Run, Reset, Stop |
| Events | Events that Clients can access | Alarm, Error, Fault |
| References | The relations between Nodes | "HasComponent", "HasEventSource" |

Table 2.1: OPC UA Node Types

## 2.4 MQTT

MQTT is a Client-Server publish-subscribe messaging transport protocol. It is very lightweight and binary protocol, and due to its minimal packet overhead, MQTT is excellent when transferring data over the wire in comparison to protocols like HTTP.

The MQTT protocol was invented in 1999 by Andy Stanford-Clark (IBM) and Arlen Nipper (Arcom, now Cirrus Link). They needed a protocol for minimal battery loss and minimum bandwidth to connect with oil pipelines via satellite. The two inventors specified several requirements for the future protocol[2]:

- Simple implementation
- Quality of Service data delivery
- Lightweight and bandwidth efficient
- Continuous session awareness

These goals are still at the core of MQTT. However, the primary focus of the protocol has changed from proprietary embedded systems to open Internet of Things (IoT) use cases.

These characteristics make it ideal for use in many situations, including constrained environments such as for communication in Machine to Machine (M2M) and Internet of Things (IoT) contexts where a small code footprint is required or network bandwidth is at a premium[3].

Another critical aspect of the protocol is that MQTT is extremely easy to implement on the client-side. Ease of use was a key concern in the development of MQTT and

[2]https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt/
[3]the official MQTT 3.1.1 specification

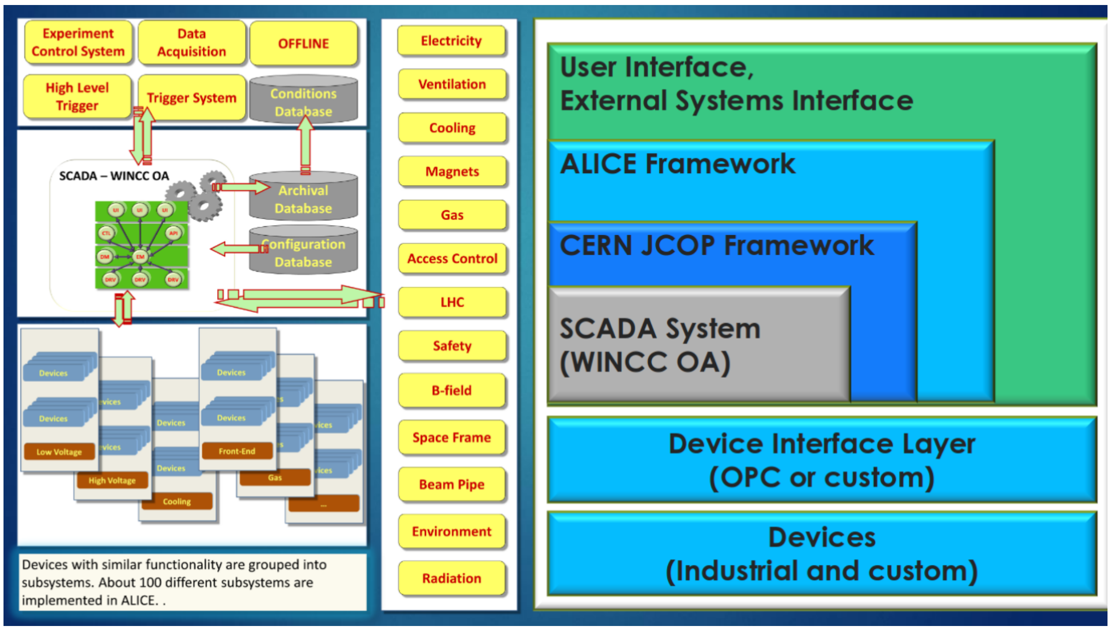


Figure 2.1: ALICE SCADA Detector Control System[13].

makes it a perfect fit for constrained devices with limited resources today. In fact, OPC UA can use MQTT as a transport protocol, which shows that MQTT can be beneficial in some cases.

## 2.5 ALICE Detector Control System

The ALICE DCS is responsible for the stable operation of the experiment. It has been built partially by experts of the detector teams. The core of the SCADA system is the commercial WINCC OA, which is used in all major CERN experiments and several technical infrastructure controls (electricity, gas, cooling, cryo). In the WINCC OA communication goes over custom protocol on top of TCP/IP[26].

The ALICE detector is much more complex than the pCT detector. However, it has similarities with the need to control and monitor detector electronics, cooling system etc. This makes the analysis of a such system useful to gain knowledge about how large systems are built and take advantage of it in planning a control system for pCT. This section is based on presentations [9], [25] and [26] to highlight new features after LS2.

OPC UA is used as a communication standard where it is possible. WINCC OA has a native OPC UA client embedded. Each detector in ALICE has its own distributed system. ALICE control layer is built as a distributed system consisting of autonomous distributed systems. ALICE DCS priorities use industrial components and standards, standardized components, commercial products, and CERN-made devices. To connect smaller devices into the system, CERN-made protocol DIM is used. ALICE DCS Architecture is shown in Figure 2.1 [9].

### 2.5.1 ALICE DCS Operation

ALICE DCS designed to be operated by a single operator. It has 24/7 shift coverage and a high turnaround of operators. ALICE central team is responsible for shifters training. ALICE hierarchy has one top DCS node, nineteen detector nodes, hundred subsystems, five hundred logical devices. It is impossible for one person to remember all operational sequences and dependencies, which are often too complex. Sub-detector developers make alerts and related instructions for their subsystems. Reactions to DCS alerts is one of the main tasks of the DCS operator. [9]

**Emergency handling and partitioning** In case of alert or critical periods, there is an option to delegate control over failing part to remote experts (they are available during running periods). Then the failure part is removed from the DCS hierarchy. To avoid potentially dangerous situations, there is the automatic check of all critical parameters for excluded components. This option has been introduced in response to needs to operate at the not normal, but safe conditions, which are typically known only by experts. [9]

### 2.5.2 DIM protocol

The Distributed Information Management (DIM) system has been designed for the DELPHI experiment at CERN. It is based on the publish-subscribe paradigm but has an advantage over the Remote Procedure Call (RPC) system. In the RPC, clients have to poll data and use one-to-one connections. DIM allows one-to-many communication and handles recovery from errors at the client and server level. The DIM system has a name server where servers register services only once at startup, and clients subscribe to them directly. It allows to save network bandwidth and reduce communication time since clients will be notified about a data change. Figure 2.2 illustrate the concept [11].

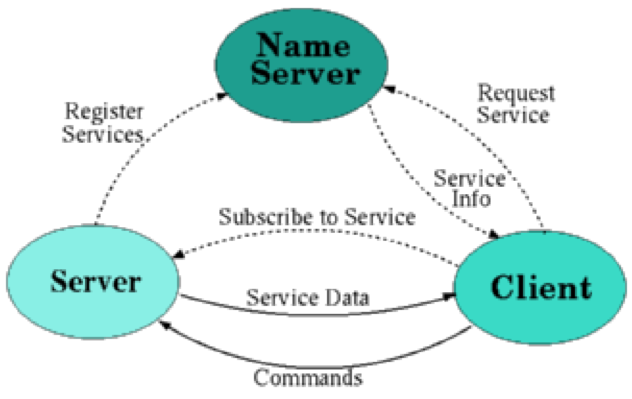


Figure 2.2: Interactions in a DIM system[11].

In addition to the industrial protocol OPC UA, DIM is used as a lighter alternative at ALICE and some other CERN experiments. DIM is used for devices without

OPC functionality[10].

DIM can be used under the GPL License, but the connection to WINCC OA is part of the JCOP framework, which is developed and maintained for CERN use. A request has been sent to use JCOP in the pCT project.

### 2.5.3 Summary

The ALICE DCS is a highly sophisticated control system and relies heavily on the OPC UA industrial protocol. However, it also shows the need for a lighter alternative for communication when custom components are involved. Proton CT will use industrial power supplies and other components where applicable. The custom devices in pCT have to be connected by the OPC UA technology or by a lighter alternative.

# Chapter 3

# Proton CT Readout Unit

The proton CT Readout Unit (pRU) is currently under development, and during the work done in this thesis, all tests and design have been done on a test setup mimicking the final system. This chapter describes the ALPIDE and its ADC in details as it main source of information for control and monitoring at the current stage.

## 3.1 VCU118

The current test setup is based on the Xilinx UltraScale+ VCU118 Evaluation Board. The choice of this board was due to the big FPGA fabric, and the amount and speed of LVDS lines [21]. The FPGA fabric allows connecting 108 ALPIDEs from one layer of the DTC. Inside the FPGA fabric, a soft-core processor (MicroBlaze)[1] is used to control communication between peripherals on AXI bus. The MicroBlaze processor is a 32-bit RISC microprocessor designed for Xilinx FPGAs. It was a natural choice for the pRU. Performance of MicroBlaze can be chosen by using one of three processors presets: Microcontroller, Real-Time Processor and Application Processor [29]. For the development of the pRU the Real-Time Processor preset has been chosen to running FreeRTOS.

### 3.1.1 FreeRTOS

FreeRTOS is one of the markets leading Real-Time Operating System (RTOS) [2]. It has more than 15 years of development time and the current major version is 10.2.1. FreeRTOS was started by Richard Barry in 2003 and was later developed and maintained by Richard's company, Real Time Engineers Ltd. In 2017 Real Time Engineers

[1]https://www.xilinx.com/products/design-tools/microblaze.html

[2]https://www.embedded.com/electronics-blogs/embedded-market-surveys/4458724/2017-Embedded-Market-Survey

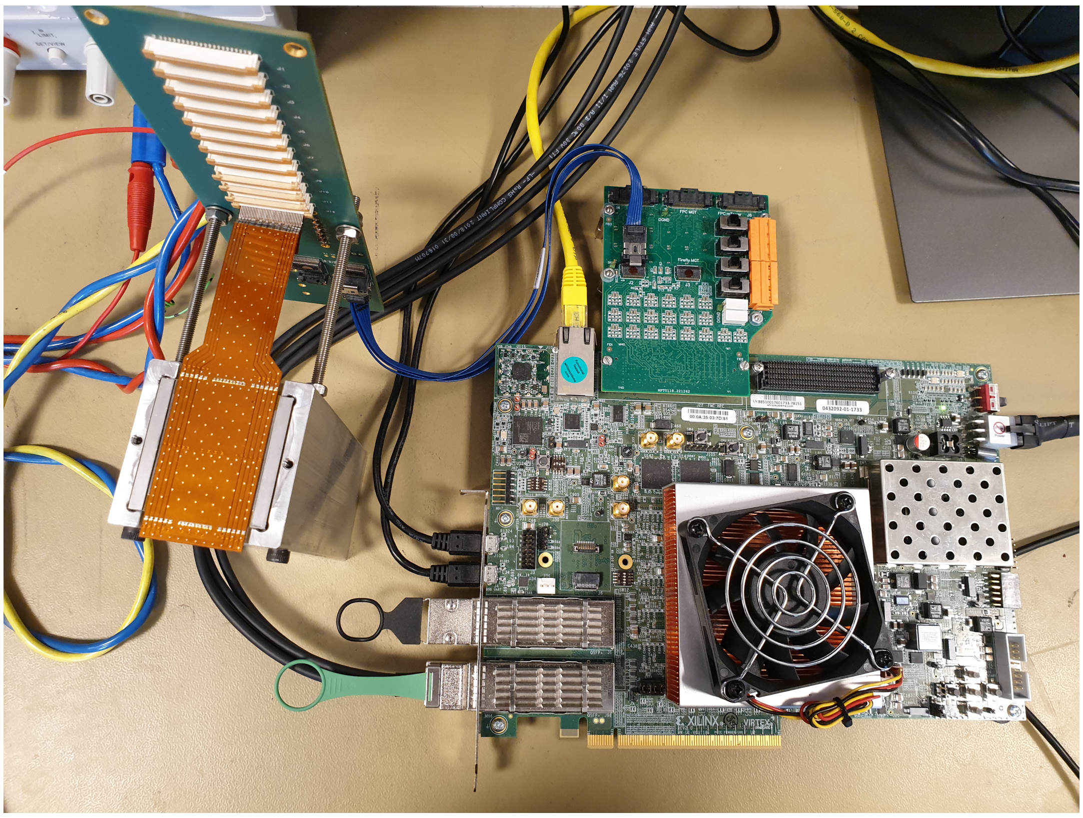


Figure 3.1: Xilinx VCU118 board with two ALPIDE chips connected.

Ltd. passed stewardship of the FreeRTOS project to Amazon Web Services (AWS)[3].

A RTOS gives the following advantages: predictable execution, multitasking (essential for low-cost microcontrollers), easy development of complex projects (by splitting into small tasks), and more[4]. FreeRTOS occupies a niche between low cost, low bit microcontrollers, and desktop processors. It is where a user can get benefits from RTOS.

FreeRTOS uses a "task" for each separate thread existing inside it. It is possible to run many copies of the same task with different priorities and input parameters. The key to master FreeRTOS is to understand the scheduler, which rules switching between tasks.

### Scheduler

The scheduler is the main part of the FreeRTOS kernel. It allows running multiple tasks according to their priority on the single-core processor by switching between tasks. Figure 3.2 shows how tasks with different priorities run and Figure 3.3 represent valid state transitions.

[3]http://www.freertos.org/RTOS.html
[4]http://www.freertos.org/FAQWhat.html#WhyUseRTOS

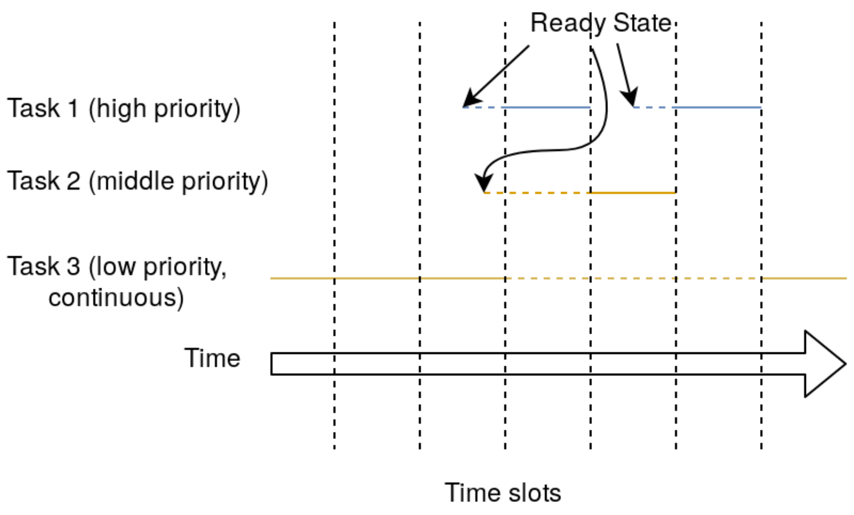


Figure 3.2: FreeRTOS scheduler work.

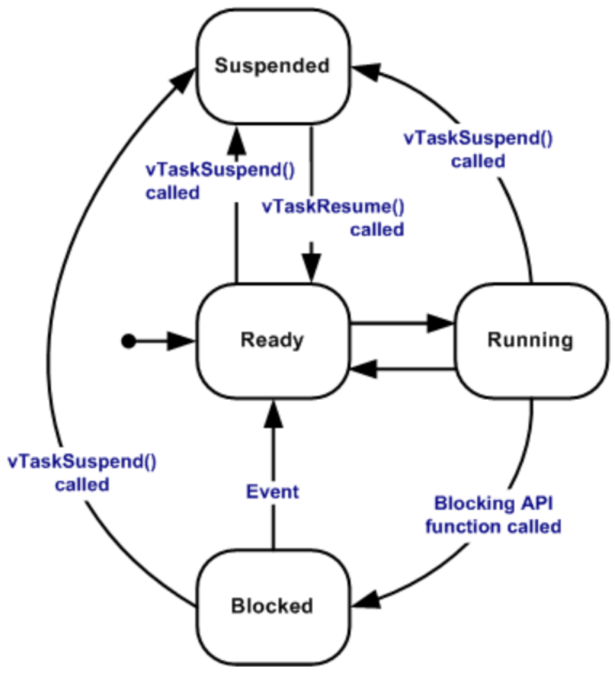


Figure 3.3: FreeRTOS valid state transitions [a]
[a] https://www.freertos.org/RTOS-task-states.html

**Ready State** Task can be executed, but waiting when another task with higher priority will finish execution.

**Running State** Task is running and currently using CPU.

**Blocked State** Task is waiting for an event.

**Suspended State** Task can enter or leave suspended state when they are explicitly commanded to do so.

It has to be at least one idle task with the lowest priority zero to be able to execute low priority, background, and continuous processing tasks.

### 3.1.2 Other subsystems in the pCT

At the moment of writing this thesis, only DTC and readout electronics are specified to some extent. In Figure 3.4 a design proposal of the whole pCT electronic system have shown. It is advised to have an external micro-controller instead soft-core. Systems like cooling, power, control system are not clarified in detail yet.

A control system should be able to connect all subsystem in one, exchange data between them, monitor all parameters, make alerts and warnings. It means that communication has to be standardized and good documented in case custom software or protocols.

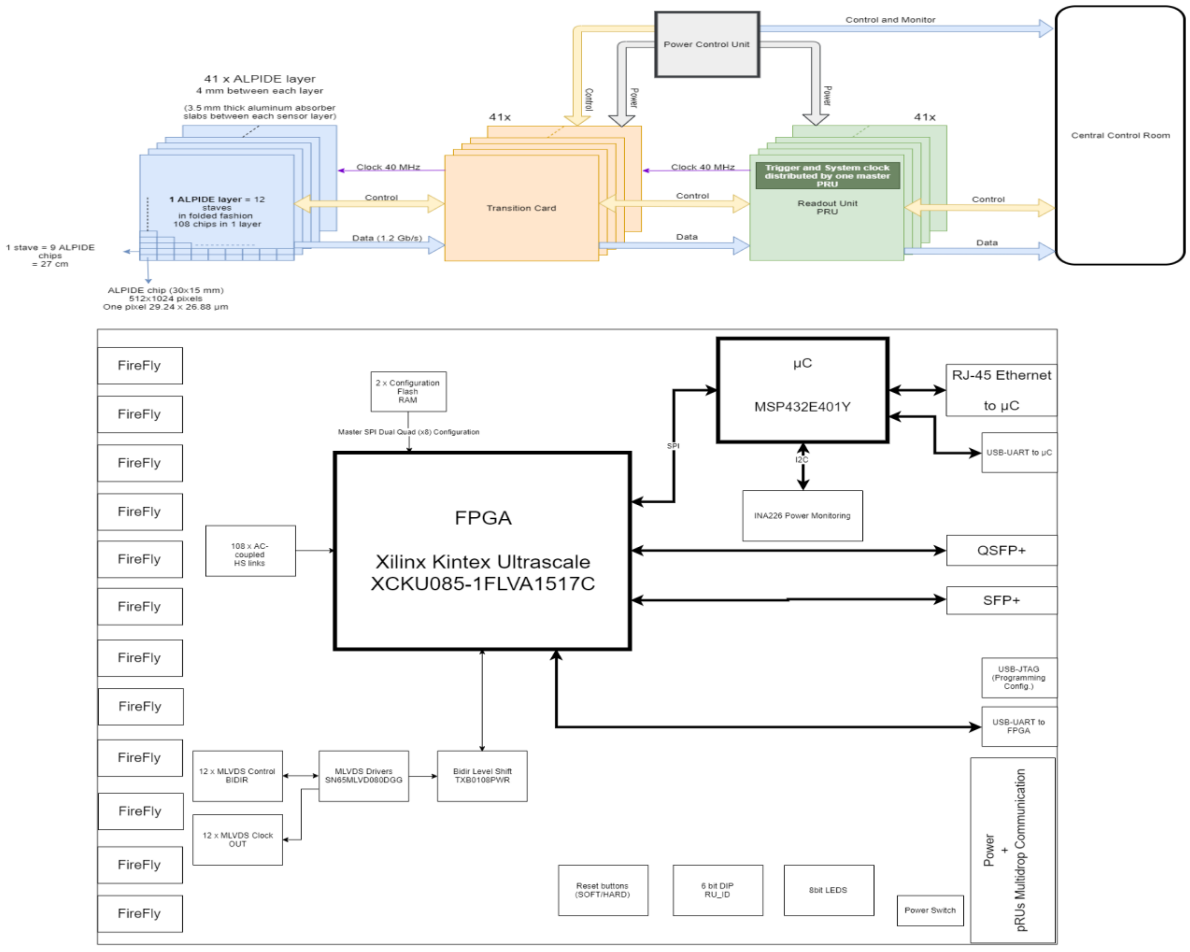

41 x ALPIDE layer
4 mm between each layer
(3.5 mm thick aluminum absorber slabs between each sensor layer)
1 ALPIDE layer = 12 staves in folded fashion 108 chips in 1 layer
1 stave = 9 ALPIDE chips = 27 cm
ALPIDE chip (30x15 mm) 512x1024 pixels One pixel 29.24 x 26.88 µm
Clock 40 MHz
Control
Data (1.2 Gb/s)
41x
Transition Card
Control
Power
Power Control Unit
Control and Monitor
Power
41x
Clock 40 MHz
Control
Data
Trigger and System clock distributed by one master PRU
Readout Unit PRU
Control
Data
Central Control Room
FireFly
FireFly
FireFly
FireFly
FireFly
FireFly
FireFly
FireFly
FireFly
FireFly
FireFly
FireFly
2 x Configuration Flash RAM
Master SPI Dual Quad (x8) Configuration
108 x AC-coupled HS links
FPGA
Xilinx Kintex Ultrascale XCKU085-1FLVA1517C
SPI
µC
MSP432E401Y
I2C
INA226 Power Monitoring
RJ-45 Ethernet to µC
USB-UART to µC
QSFP+
SFP+
USB-JTAG (Programming Config.)
USB-UART to FPGA
12 x MLVDS Control BIDIR
MLVDS Drivers SN65MLVD080DGG
Bidir Level Shift TXB0108PWR
12 x MLVDS Clock OUT
Reset buttons (SOFT/HARD)
6 bit DIP RU_ID
8bit LEDS
Power Switch
Power + pRUs Multidrop Communication

Figure 3.4: Block diagram of the whole pCT electronics system [20].

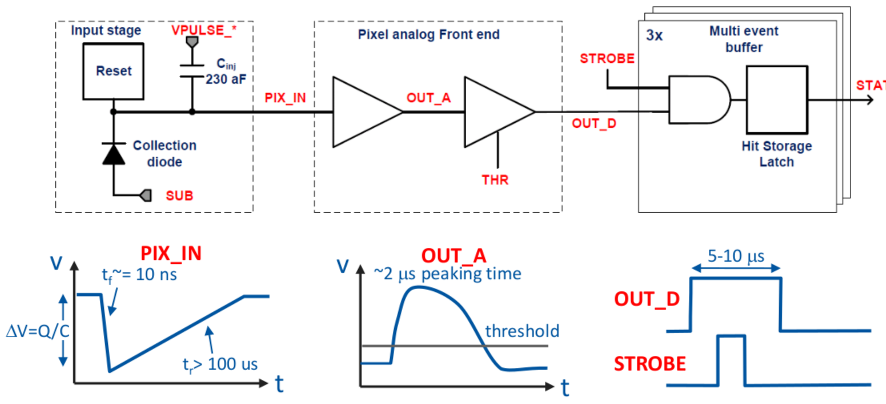


Figure 3.5: Pixel cell block diagram of the ALPIDE chip[19].

## 3.2 The ALPIDE

The ALPIDE chip is a state-of-art detector chip for particle detection. It makes the ALPIDE an ideal candidate for use in proton CT. It is not so many special chips for particle detection, so newest ALPIDE developed for CERN stand out it term of specifications among them. The ALPIDE chip has dimensions 15 mm by 30 mm with sensitive matrix 512 $\times$ 1024 pixels. Each pixel is 29,24 µm $\times$ 26,88 µm.This section is based on the ALPIDE Operational Manual and provides the details needed for understanding this thesis [19].

### 3.2.1 Pixel's front-end

Each pixel has a sensing diode, a front-end amplifying and shaping stage, a discriminator and a digital section Figure 3.5. Hit Storage Latch in the digital section is designed as a Multi Event Buffer (MEB), a pulsing logic and a pixel masking register[19].

All pixels have a common threshold level. A hit will be latched in one of the cells in MEB if OUT_A signal above threshold while STROBE is asserted. The STROBE signal can be initiated by the internal sequencer or by an external command (TRIGGER). The STROBE pulses have a programmable duration. A pulse injection capacitor is placed in every pixel allow to inject test charge in analog front-end.

### 3.2.2 Priority Encoder

Pixel cells are organized in double columns for readout Figure 3.6. 1024 columns divided into 32 regions having 16 double columns each. Each region is connected to Region Readout unit (Figure 3.7).

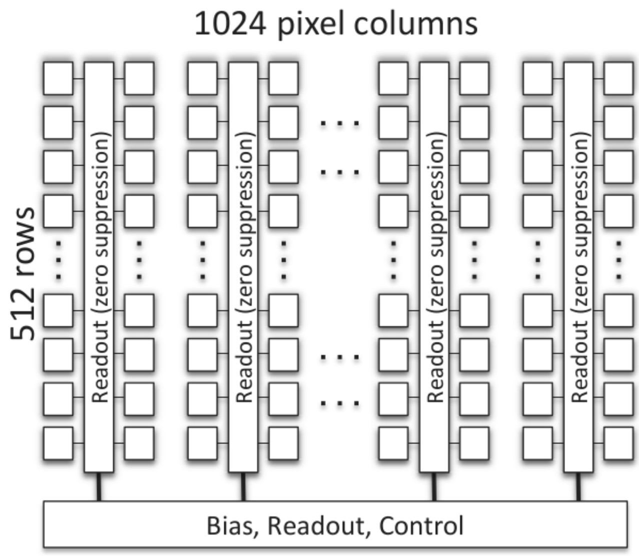

1024 pixel columns
512 rows
Readout (zero suppression)
Readout (zero suppression)
Readout (zero suppression)
Readout (zero suppression)
Bias, Readout, Control

Figure 3.6: General architecture of the ALPIDE chip[19].

### 3.2.3 ALPIDE block diagram

The ALPIDE has a Differential Control Port running at 40 Mbps for control interface with a custom protocol. IB and OB Master chips using this port and OB Slaves connected through Single Ended Control Port. OB Masters acts as a hub on the control bus.

The data Readout from the ALPIDE chip in IB and OB Master mode is going through the hi-speed serial port with rate 1200 or 400 Mbps, respectively. OB Slaves are using Parallel Data Port shared with other slaves in half-stave.

### 3.2.4 ALPIDE ADC

The ALPIDE chip has built-in 11-bit ADC for measurement of quasi-static internal analog signals. It can measure power supplies, grounds, temperature, Bandgap voltage, and outputs of internal DAC. Global schematics of the ADC is given on Figure 3.8. The ALPIDE ADC is the primary source of the monitoring information.

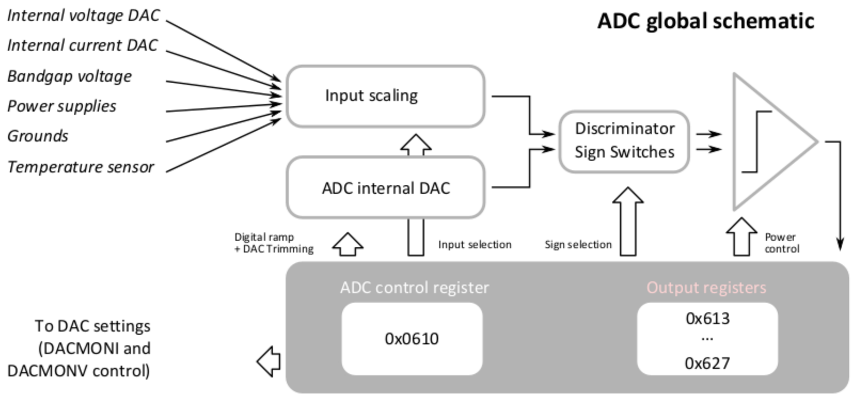


Figure 3.8: ADC global schematics[19].

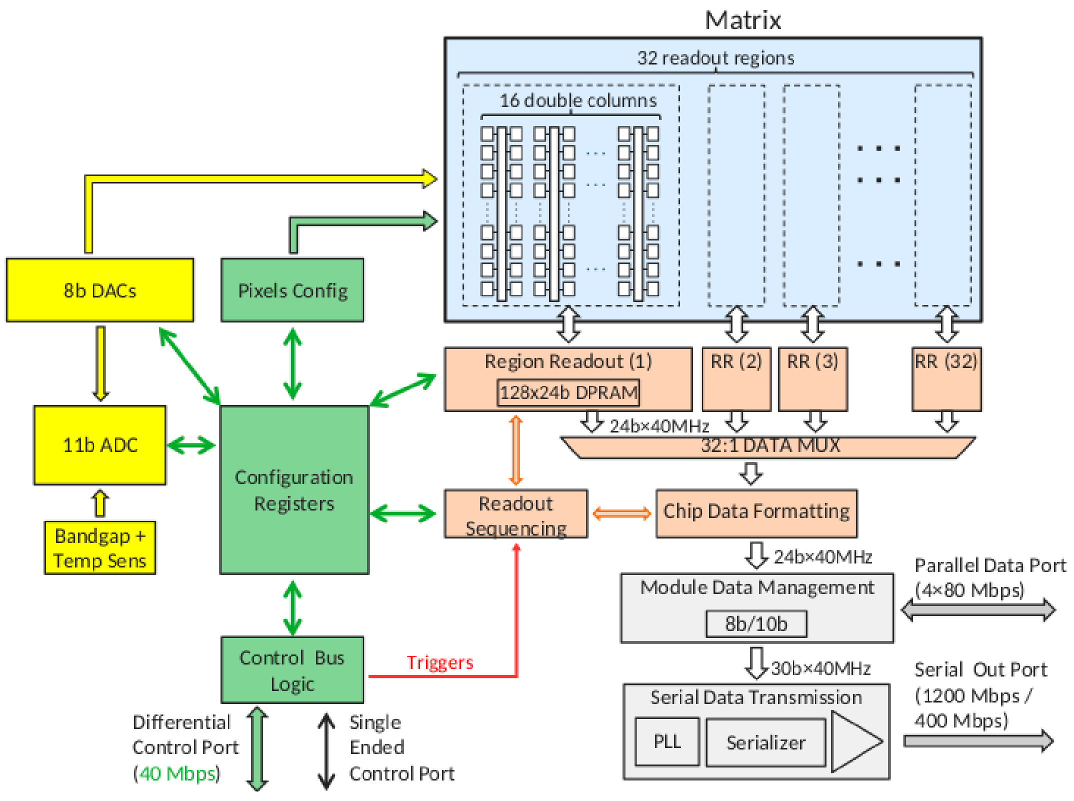


Figure 3.7: The block diagram of the ALPIDE chip[19].

Before the ADC can be used, a calibration procedure described in [19] has to be done. The following formulas for conversion of the ADC readings are described in [19] as given below:

## Voltage measurements

$$Value_{read} = \frac{V_{mV}}{2 \cdot 1.068} + offset \tag{3.1}$$

## Current measurements

$$Value_{read} = \frac{5 \cdot I_{\mu A}}{1.068} + offset \tag{3.2}$$

## Temperature measurements

$$Value_{read} = \frac{Temp_{degCelsius} + 51.5}{0.147} + offset \tag{3.3}$$

However, during test, the values converted by equation 3.1 and 3.3 was not realistic since temperature (at least in the first moment after startup) and voltage was known. ALPIDE Operational manual available at that time was from July 25, 2016, and it was still in draft state. After a research another formulas which give realistic results have been found in *pct/alpide-sw* repository in file TAlpide.cpp with comment *first approximation*[5]. The formulas from ALPIDE.cpp in the form similar to ALPIDE Operations Manual are given below for comparison:

**Voltage measurements**

$$Value_{read} = \frac{V_{mV}}{1.644} + offset \tag{3.4}$$

**Current measurements**

$$Value_{read} = \frac{I_{\mu A}}{0.1644} + offset \tag{3.5}$$

**Temperature measurements**

$$Value_{read} = \frac{Temp_{degCelsius} - 6.8}{0.1281} + offset \tag{3.6}$$

The temperature equation has the most significant difference. Temperature measurements were the main reason why the formulas from [19] were not accepted. The voltage tests have been done to verify that absolute value, sign, and scaling factor are appropriate. During the test, supply voltages of the ALPIDE were decreased by 50 mV, and the ADC readings of the analog and digital power supplies decreased accordingly. The absolute errors for temperature and voltage were under 10%, which is acceptable for monitoring. Another group has done current measurements during DAC scan, where values were written to the DAC registers and then read back by the ADC. The goal of the test was verification of the behavior of the ADC and DAC of the ALPIDE[6]. Extensive ADC tests and tuning of formulas were outside of the scope of this thesis as it requires detailed knowledge of the ALPIDE. Information about the formulas has been sent to the authors of the ALPIDE Operational Manual.

In Table 3.1 overview over the ADC inputs is given. Most of the inputs are dedicated to the monitoring of the analog Front-end. In Appendix A.2 Front-end schemes for the ALPIDE pixel have given. For the monitoring purpose, only temperature and analog and digital ground and supply are relevant. There are two temperature sensors on the ALPIDE chip and both included in monitoring. However, in manual mode only direct measurement is possible (see Table 3.2). It is parameter *TEMPERATURE* in Table 3.1 and formula 3.6 is used for convertion. For *VTEMP* temperature is not in degrees Celsius, but in mV and formula 3.4 is used.

The ADC has manual and automatic measurement modes. The duration of the automatic measurement of all channels of the ADC is 100 ms, but one channel can be measured in manual mode for 5 ms. Channels available for the manual ADC measurement is given in Table 3.2. Inputs which belongs to analog front-end are typically used for debugging and not included in manual measuring mode of ADC.

[5]ADC reading functions can be found in Appendix A.1

[6]More information about ADC and DAC registers in section 4.3.9

Using manual mode and only five channels gives a shorter time and less power consumption. A dedicated procedure has to be implemented on pRU to avoid delays on transmission lines in this case.

| Parameter | Description | Monitoring | Debug |
|---|---|---|---|
| *AVSS* | Analog ground | ☑ | |
| *DVSS* | Digital ground | ☑ | |
| *AVDD* | Analog power supply | ☑ | |
| *DVDD* | Digital power supply | ☑ | |
| *VCASN* | Analog Front-end, Figure A.1 | | ☑ |
| *VCASP* | Analog Front-end, Figure A.1 | | ☑ |
| *VPULSEH* | VPULSEH - VPULSEL = VPLULSE_* [a] | | ☑ |
| *VPULSEL* | | | ☑ |
| *VRESETP* | PMOS reset, Figure A.2 | | ☑ |
| *VRESETD* | Reset voltage for pix-in, Figure A.1 | | ☑ |
| *VCASN2* | Analog Front-end, Figure A.1 | | ☑ |
| *VCLIP* | Analog Front-end, Figure A.1 | | ☑ |
| *VTEMP* | Temperature | ☑ | |
| *IRESET* | Bias current, Figure A.2 | | ☑ |
| *IAUX2* | Analog Front-end | | ☑ |
| *IBIAS* | Analog Front-end, Figure A.1 | | ☑ |
| *IDB* | Analog Front-end, Figure A.1 | | ☑ |
| *IREF* | The value is not correct for this input [b] | | ☑ |
| *ITHR* | Analog Front-end, Figure A.1 | | ☑ |
| *BANDGAP* | Band gap | | ☑ |
| *TEMPERATURE* | Temperature | ☑ | |

Table 3.1: ADC inputs description
[a] See Figure A.1, [b] According to ALPIDE Operation Manual Section 3.10.5[19]

| Value | Selected Input |
|---|---|
| 0 | AVSS |
| 1 | DVSS |
| 2 | AVDD |
| 3 | DVDD |
| 4 | V band-gap through voltage scaling |
| 5 | DACMONV |
| 6 | DACMONI |
| 7 | Bandgap (direct measurement) |
| 8 | Temperature (direct measurement) |

Table 3.2: ADC inputs selection in manual mode[19].

Monitoring the power supplies of each ALPIDE serves the purpose of making sure that every chip has voltage within recommended limits. However, tracking a single channel at the rate of $200Hz$ is possible in manual mode. It has to be done on board of the proton Readout Unit (pRU) as a dedicated procedure to do not introduce extra latency between ADC reading.

Chapter 4

# Requirements for a pCT Control System

In this chapter, requirements for the monitoring system of pCT will be outlined. It will make the basis for decisions further when system configuration is discussed together with technology choice for protocols. The system for pCT is given in Figure 1.7.

## 4.1 General requirements

The pCT monitoring system has to be able to:

- Run on soft-core processor or a standard micro-controller.
- Support RTOS and embedded Linux.
- Allow to update all monitoring parameters at least once per second
- Allow to read and write to ALPIDE registers during monitoring
- Provide monitoring and control when the system is powered on
- Technology used and has to be stable, well documented and supported
- Easy to master, maintain and change

## 4.2 Graphical interface requirement

A Graphical User Interface (GUI) is an essential part of a monitoring system. It should give a good overview of the whole pCT detector in an easy and informative way together with access to detailed information and parameters. Functions which are used often in typical usage pattern have to be easy to access and intuitive to understand name or icons of buttons. GUIs for a control system usually go through many stages of modification before they reach the final look and functionality. So here only the main concepts will be proposed.

## 4.3 Technical requirements

It is already decided that the PRU will have an Ethernet interface for data upload and configurations of ALPIDE chips and PRUs. So most important is what type of protocol will be used on top of the TCP-IP stack to fulfill the monitoring task. As described in [22], to control PRU The Serial Protocol for Arbitrary Data (SPAD) has been used. In that work, emphasis has been made on making communication between two hosts reliable even if link does not error-free and in the implementation of communication protocol a protection from garbage data is used.[1] It means that packets coming out of order lead to closing of connection. This feature is good to secure that both sender and receiver are synchronized in terms of packet interaction between them. For monitoring purposes, when data can be requested from many hosts, it is hard to implement because it needs to keep track of all active sessions. Monitored data is usually sent periodically and the loss of one packet should not make a critical situation.This is an argument to find a lighter alternative to supply external hosts with monitoring data.

### 4.3.1 Loss of packet with monitoring information

During a run, a control system can get monitoring information irregularly due to packet loss, error on the hardware while receiving data. These losses are not frequent in normal operating conditions and should not lead to the closing of connection or generation of an alarm. To cope with it, an internal check has to track incoming information and verifies that no data is outdated. It can be implemented by setting timeout a couple of times higher than update time. However, a connection lost must generate an alarm immediately. A timestamp has to be attached to every packet of monitoring information.

### 4.3.2 Protection from erroneous data

The write operation to the ALPIDE should have a protection from the misinterpretation of incoming commands. Operational codes of the ALPIDE command register have a long hamming distance, but register address and value have to be protected separately. This protection will work in case of losing the end of the packet in addition to a cyclic redundancy check of a protocol.

[1]File software/embedded/app_vcu118/src/tcp_ctrl.c in wp3 repository line 158 se below:
if (packet[payload_len + TCP_CTRL_PKT_OVRHD - 1] != sender_seq_num_tcp) { //Sender sending garbage data; close connection

### 4.3.3 Monitoring protocol

The monitoring protocol should be a robust protocol and able to give access to information from many observers. Robustness meants that it should have tolerance to erroneous data format on input and re-synchronization if losing the end of the packet sequence.

Using additional protocol along with an already existing communication protocol described in [22] are benificial for monitoring purposes. The data offload is not polluted with monitoring data, several monitoring systems can be run at the same time if the monitored data is published autonomously by pRU. In case some data have been requested by a master monitoring software, another software can be running and receiving the same information by having appropriate configuration. This gives an advantage at development, commissioning and running stage because several operators can observe, analyze the data and follow the alarms and warnings.

### 4.3.4 On-board monitoring software

A task in FreeRTOS has already been implemented that checks specified registers if they exceed a maximum threshold. This is sufficient when all ALPIDEs are error-free and operational parameters are very similar between chips. Also, it would be practical to use minimum and maximum thresholds as well as the ability to change them "on the fly" without spending time on compilation and programming of FPGA on PRU. However the monitoring information has to be sent to the control system where a threshold check is done. So it is an optional feature. A task on the pRU side should be able to read and send all predefined parameters without external request. Registers involved in monitoring, as well as a suitable format for sending these data, should be specified.

### 4.3.5 Schedule ADC scan

Using the ADC of the ALPIDE is power-consuming operation. To guarantee stable power voltage and, as a result, better precision and repeatability ADC measurements can be scheduled. Anyway in a stave control lines are shared and starting of ADC measurments synchronously is not possible. It will, however, introduce time difference between measurements, but a small delay is not critical for monitoring purposes. At the moment of the writing this thesis there is no prototype available with one whole layer of ALPIDEs to test this approach and verify if it has implications to the power stability. It is recommended to test this when the necessary hardware is in place.

### 4.3.6 Timestamp for monitored parameters

During monitoring the control data will not change significantly in standby condition. It may even, can be the same over many monitoring cycles. To distinguish packets with monitored data, a sequence number or timestamp has to be used. This will serve to verify the validity of the measurement. In case package loss the last successfully received packet will be a reference point for alarm triggering. The alarm has to be triggered after loss of several expected messages.

### 4.3.7 Cross-platform protocol

FreeRTOS is currently selected as the OS for pRU, but it has been considered to move to a Linux OS where programming is more comfortable. Due to this reason, a solution and especially protocol have to be platform-independent. A Linux distribution requires much more operative memory that is available inside FPGA fabric, so external memory has to be used.

### 4.3.8 Open-source technology implementation

The pCT project is still in an early stage, and some subsystems will be introduced later. Besides, the major part, like the operating system, can change. All these are reasons to test open-source technology implementations first. Further, this requirement can become optional when the system has been finalized, and the use of a commercial version will be worth it.

### 4.3.9 Estimation of amount of monitored data

To estimate the amount of monitored data we will go through ALPIDE's address space and assign definition to each register one of the following: monitor, debug or configure. This section is based on [19]. The results will be summarized in Tables 4.1-4.3.

Registers in range 0x0000-0x001B of the ALPIDE are periphery control registers. The command register is a dedicated register for the execution of internal operations or sequences. This register accept only predefined operational codes. Among them are chip global reset, pixel matrix reset and start ADC measure. This is a configuration register.

Mode control register is responsible for enabling readout and clustering, matrix readout speed and serial link speed. Region disable registers 1 and 2 is used for masking of individual regions. Framing and Management Unit (FROMU) Configuration Register 1 controls pixel Multi Event Buffer, internal strobe generation, busy monitoring, test pulse mode, enable test strobe, trigger delay. FROMU Configuration Register 2-3 control duration of the strobe pulses and gap between them. FROMU Pulsing Register 1-2 controls delay of the pulse signal to the strobe signal and duration of the pulse signal. These are configuration registers.

FROMU Status Registers 1-5 are counters. They have values of the FROMU trigger counter, the strobe counter, the matrix readout counter and the frame counter respectively. These are typically used for debugging and will not be involved in

| **Address** | **Mode** | **Register or memory** | **Monitor** | **Debug** | **Configure** |
|---|---|---|---|---|---|
| 0x0000 | R/W | Command Register | | | ☑ |
| 0x0001 | R/W | Mode Control register | | | ☑ |
| 0x0002 | R/W | Disable of regions 0-15 | | | ☑ |
| 0x0003 | R/W | Disable of regions 16-31 | | | ☑ |
| 0x0004 | R/W | FROMU Configuration Register 1 | | | ☑ |
| 0x0005 | R/W | FROMU Configuration Register 2 | | | ☑ |
| 0x0006 | R/W | FROMU Configuration Register 3 | | | ☑ |
| 0x0007 | R/W | FROMU Pulsing Register1 | | | ☑ |
| 0x0008 | R/W | FROMU Pulsing Register 2 | | | ☑ |
| 0x0009 | R | FROMU Status Register 1 | | ☑ | |
| 0x000A | R | FROMU Status Register 2 | | ☑ | |
| 0x000B | R | FROMU Status Register 3 | | ☑ | |
| 0x000C | R | FROMU Status Register 4 | | ☑ | |
| 0x000D | R | FROMU Status Register 5 | | ☑ | |
| 0x000E | R/W | DAC settings for DCLK and MCLK I/O | | | ☑ |
| 0x000F | R/W | DAC settings for CMU I/O | | | ☑ |
| 0x0010 | R/W | CMU and DMU Configuration Register | | | ☑ |
| 0x0011 | R | CMU and DMU Status Register | | ☑ | |
| 0x0012 | R | DMU Data FIFO [15:0] | | ☑ | |
| 0x0013 | R | DMU Data FIFO [23:16] | | ☑ | |
| 0x0014 | R/W | DTU Configuration Register | | | ☑ |
| 0x0015 | R/W | DTU DACs Register | | | ☑ |
| 0x0016 | R | DTU PLL Lock Register 1 | | ☑ | |
| 0x0017 | R/W | DTU PLL Lock Register 2 | | | ☑ |
| 0x0018 | R/W | DTU Test Register 1 | | ☑ | |
| 0x0019 | R/W | DTU Test Register 2 | | ☑ | |
| 0x001A | R/W | DTU Test Register 3 | | ☑ | |
| 0x001B | R/W | BUSY min width | | | ☑ |

Table 4.1: Periphery Control Registers

regular monitoring due the fact that they expected to change only during the particle registration.

Two DAC settings registers set drivers and receivers currents for input and output buffers, and Control Management Unit (CMU) and Data Management Unit (DMU) configuration register are configuration registers.

CMU and DMU status register contains error counters which under normal conditions should remain zero, these are debug register. DMU Data First In First Out (FIFO) last significant bit and most significant bit registers are for reading DMU data or busy FIFO. These are used in data readout so they are debug registers.

Data Transmission Unit (DTU) configuration register has Phase-Locked Loop (PLL) settings. DTU DACs register contains different current setting of DTU. These are configuration registers.

DTU PLL Lock Register 1 is a read only register of output flags and signals of PLL related logic. This is used for debugging. DTU PLL Lock Register 2 controls the behavior of the PLL monitoring state machine and it is configuration register. DTU Test Register 1-3 provides access to built-in DTU tests therefore they are debug registers. The BUSY min width register controls minimum length of BUSY signal. This is a configuration register.

Registers in range 0x600-0x0627 are DAC, ADC and monitoring control registers. Analog monitor and override register is for DAC voltage and current selection, and for DAC monitoring and overriding selection, and for reference current selection. It is configuration register. Registers in range 0x0601-0x060E are DAC's setting registers. They are dedicated for setting in analog front-end.[2] They are configuration registers. The buffer bypass register controls bypassing of buffer cell for particular voltage or current and therefore is configuration register.

The ADC control register controls ADC setting and stores ADC calibration coefficients. The ADC DAC input value register used for reference value selection for the DAC. They are configuration registers.

The ADC AVSS/manual value register is used in ADC manual mode and stores read value. In auto mode it stores analog ground supply (AVSS) value. Registers in range 0x0614-0x0627 provide data after an auto measurement procedure[3]. Registers in ranges 0x0617-0x061E and 0x0620-0x0626 are dedicated to measurements of analog front-end which is set by DAC registers[4] These are debug registers. For monitoring have chosen analog and digital power supplies and grounds(AVSS, DVSS, AVDD and DVDD) as well as temperatures ADC VTEMP and ADC T2V.

[2]See Appendix A.2 for the ALPIDE front-end.
[3]Described in [19] in section 3.10.5
[4]See Appendix A.2.

| **Address** | **Mode** | **Register or memory** | **Monitor** | **Debug** | **Configure** |
|---|---|---|---|---|---|
| 0x0600 | R/W | Analog Monitor and Override Register | | | ☑ |
| 0x0601 | R/W | VRESETP | | | ☑ |
| 0x0602 | R/W | VRESETD | | | ☑ |
| 0x0603 | R/W | VCASP | | | ☑ |
| 0x0604 | R/W | VCASN | | | ☑ |
| 0x0605 | R/W | VPULSEH | | | ☑ |
| 0x0606 | R/W | VPULSEL | | | ☑ |
| 0x0607 | R/W | VCASN2 | | | ☑ |
| 0x0608 | R/W | VCLIP | | | ☑ |
| 0x0609 | R/W | VTEMP | | | ☑ |
| 0x060A | R/W | IAUX2 | | | ☑ |
| 0x060B | R/W | IRESET | | | ☑ |
| 0x060C | R/W | IDB | | | ☑ |
| 0x060D | R/W | IBIAS | | | ☑ |
| 0x060E | R/W | ITHR | | | ☑ |
| 0x060F | R/W | Buffer Bypass Register | | | ☑ |
| 0x0610 | R/W | ADC Control Register | | | ☑ |
| 0x0611 | R/W | ADC DAC input value | | | ☑ |
| 0x0612 | R | ADC Calibration Value Register | | | ☑ |
| 0x0613 | R | ADC AVSS/Manual Value Register | ☑ | | |
| 0x0614 | R | ADC DVSS Value Register | ☑ | | |
| 0x0615 | R | ADC AVDD Value Register | ☑ | | |
| 0x0616 | R | ADC DVDD Value Register | ☑ | | |
| 0x0617 | R | ADC VCASN Value Register | | ☑ | |
| 0x0618 | R | ADC VCASP Value Register | | ☑ | |
| 0x0619 | R | ADC VPULSEH Value Register | | ☑ | |
| 0x061A | R | ADC VPULSEL Value Register | | ☑ | |
| 0x061B | R | ADC VRESETP Value Register | | ☑ | |
| 0x061C | R | ADC VRESETD Value Register | | ☑ | |
| 0x061D | R | ADC VCASN2 Value Register | | ☑ | |
| 0x061E | R | ADC VCLIP Value Register | | ☑ | |
| 0x061F | R | ADC VTEMP Value Register | ☑ | | |
| 0x0620 | R | ADC ITHR Value Register | | ☑ | |
| 0x0621 | R | ADC IREF Value Register | | ☑ | |
| 0x0622 | R | ADC IDB Value Register | | ☑ | |
| 0x0623 | R | ADC IBIAS Value Register | | ☑ | |
| 0x0624 | R | ADC IAUX2 Value Register | | ☑ | |
| 0x0625 | R | ADC IRESET Value Register | | ☑ | |
| 0x0626 | R | ADC BG2V Value Register | | ☑ | |
| 0x0627 | R | ADC T2V Value Register | ☑ | | |

Table 4.2: DACs, ADCs and Monitoring Control Registers

| Address | Mode | Register or memory | Monitor | Debug | Configure |
|---|---|---|---|---|---|
| 0x0700 | R | SEU Error Counter | | ☑ | |
| 0x0701 | R/W | Test Control Register | | ☑ | |
| 0x0702 | R | BMU Debug Stream | | ☑ | |
| 0x0703 | R | DMU Debug Stream | | ☑ | |
| 0x0704 | R | TRU Debug Stream | | ☑ | |
| 0x0705 | R | RRU Debug Stream | | ☑ | |
| 0x0706 | R | FROMU Debug Stream | | ☑ | |
| 0x0707 | R | ADC Debug Stream | | ☑ | |

Table 4.3: Test and Debug Control Registers

The ALPIDE ADC channels are desirable to monitor on a rate about one per second in normal conditions. This rate is acceptable because high-speed protection against latch-up will be implemented at the FPGA level. Cooling system in the pCT consists of water cooling to the edges of the absorber frames and then convection cooling through the material of the absorder and air circulation in gap between neighbor layers. This gives a high temperature constant of the system. In case water circulation stops temperature is not expected to rise quickly.

The last group in addressing space of the ALPIDE is eight test and debug registers and five status registers in peripheral control registers. These contains error counters, debug steams for shadows registers. Test control register forces local bus or FROMU to a specific value. The whole group is debug registers.

To summarize it is needed to monitor analog and digital power supplies - ADC AVSS, ADC DVSS, ADC AVDD, ADC AVSS and the temperatures - ADC VTEMP and ADC T2V, on every chip. In addition we need to monitor on pRU - board temperature and current; on FPGA VCCINT,VCCAUX, junction temperature. The minimum amount of registers that we would like to read is six for each ALPIDE. So estimation will be:

$$(6_{monitoring}) * 108_{ALPIDE\ per\ layer} + 2_{pRU} + 3_{FPGA} = 653\ Registers$$

We can round up to 1000 parameters per second for one layer to give a room for additional registers if needed.

### 4.3.10 Requirements summary

Requirements described above are given in Table 4.4, with a check whether it is a must or an option for the pCT control system prototype. The table is giving an overview of all formulated requirements. Every requirement has its reference that will allow easy refer and discuss it later.

R.1 has an optional requirement due to the proposal for of electronic system shown

| **Reference** | **Requirement** | **Must** | **Optional** |
|---|---|---|---|
| R.1 | Run on soft-core processor | | ☑ |
| R.2 | Update 1000 parameters once per second | ☑ | |
| R.3 | Read and write to ALPIDE during monitoring | ☑ | |
| R.4 | Solution is stable, documented and supported | ☑ | |
| R.5 | Technology is easy to master, maintain and change | ☑ | |
| R.6 | Monitoring on board of pRU | ☑ | |
| R.7 | Schedule the ADC measurement | | ☑ |
| R.8 | Timestamp for monitored parameters | ☑ | |
| R.9 | Cross-platform | ☑ | |
| R.10 | Open-source technology implementation | ☑ | |
| R.11 | Monitor always when data is nor acquiring | ☑ | |

Table 4.4: Summary table for requirements

in Figure 3.4. R.3 means that monitoring does not have to be stopped to read and write operations to an ALPIDE. R.4 requires mature and supported protocols and solutions. R.5 takes into account that pCT is under development in the University and people involved in the project might change. R.6 requires that monitoring information must be sent without request. R.7 is an optional requirement that ADC measurements are schedulered to get better repeatability and precision. R.11 means that monitoring is always running when pCT neither in the picture nor in the treatment modes. The rest of requirements are self-explained.

# Chapter 5

# Prototyping

This section describes the prototyping that has been done to verify MQTT and OPC UA technologies for use in a the pCT control system. Both technologies will be explained in more detail than what was given in Chapter 2. Emphasis is made on realization and revealing the strengths and weaknesses of each technology so that it will be possible to compare them and come up with a recommendation for pCT in the next chapter. During prototyping the solution must be based on open-source tools (see requirement R.11).

## 5.1 MQTT

An open-source broker Eclipse Mosquitto has been chosen to test MQTT. In the MQTT, a broker is piece of software that receives all messages and then forwards them according to the subscriptions of clients. To give flexibility in terms of programming language for client, the Eclipse Paho client implementation is selected. The programming language for testing MQTT functionality is Python.

The MQTT protocol has a package forwarding method based on so called topics. A topic is a text preamble or header of a message in MQTT. The text format for topics is used in MQTT for easier setup and debugging. In MQTT, all communication goes through the broker. All clients have equal status, and there is no master in communication. Clients can publish messages. The broker delivers them according to subscription. To receive a message a client has to subscribe to topics it is interested in.

A topic is a mandatory part of every message. A client can publish on any topic even if nobody have subscribed to it. So the function of a broker is to receive messages from connected clients and then send it to clients according to their subscription. The subscription of every client can be understood as a filter that is applied to all messages inside a broker before they will be sent to a client. Figure 5.1 illustrates

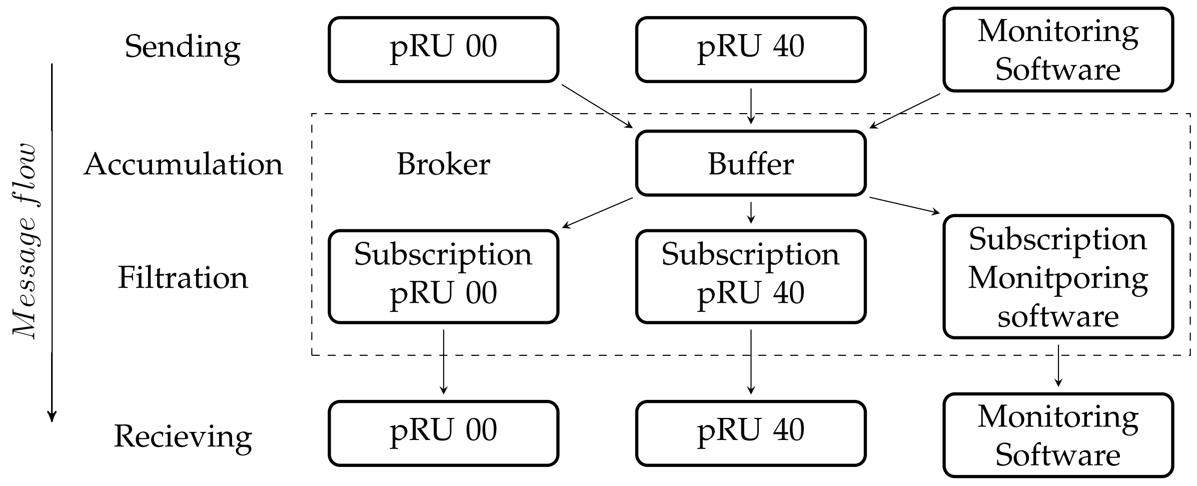


Figure 5.1: MQTT message flow.

the concept.

## 5.1.1 Topic structure for pCT

The MQTT messaging protocol provides abstraction from IP addresses. So it is essensial to make structure of topics in a way that the monitoring communication is easy to implement. It means that topics are a part of the monitoring system. MQTT has built-in wild-cards that gives the opportunity to make high-level subscriptions and get all publications on all underlying levels. It is similar to a folder structure: having a folder means having all subfolders. The pCT has a hierarchy made up of pRUs, staves and ALPIDEs, so it is natural to use these as MQTT topics.

The logical geometry, which is representing the hierarchy of the pCT detector, is shown in Figure 5.2. The first layer with ID=00 is facing the beam. The ALPIDE chips are grouped in horizontal staves of nine chips. The first stave with ID=00 is on the top of the layer. Behind each layer is the absorber plate followed by an air gap, except for the first two layers that are the tracking layers, which does not have an absorber plate. Dimensions of the absorber plate and the air gap is not in scale.

To test the MQTT functionality, a topic structure shown in Table 5.1 have been designed. The special character slash "/" is used to set the hierarchy in a topic. It is a built-in feature of the MQTT protocol. It will also be used for managing subscriptions in Section 5.1.3. Currently, one pRU will be connected to one layer, so it means that LAYER_ID is pRU's id number.

| | | | | | | |
|---|---|---|---|---|---|---|
| LAYER_ID | / | STAVE_ID | / | CHIP_ID | / | OPERATION_TYPE |

Table 5.1: A general topic structure for a massage towards a pRU.

To avoid mixing layer, stave, and chip IDs, a letter is used in addition to the ID.

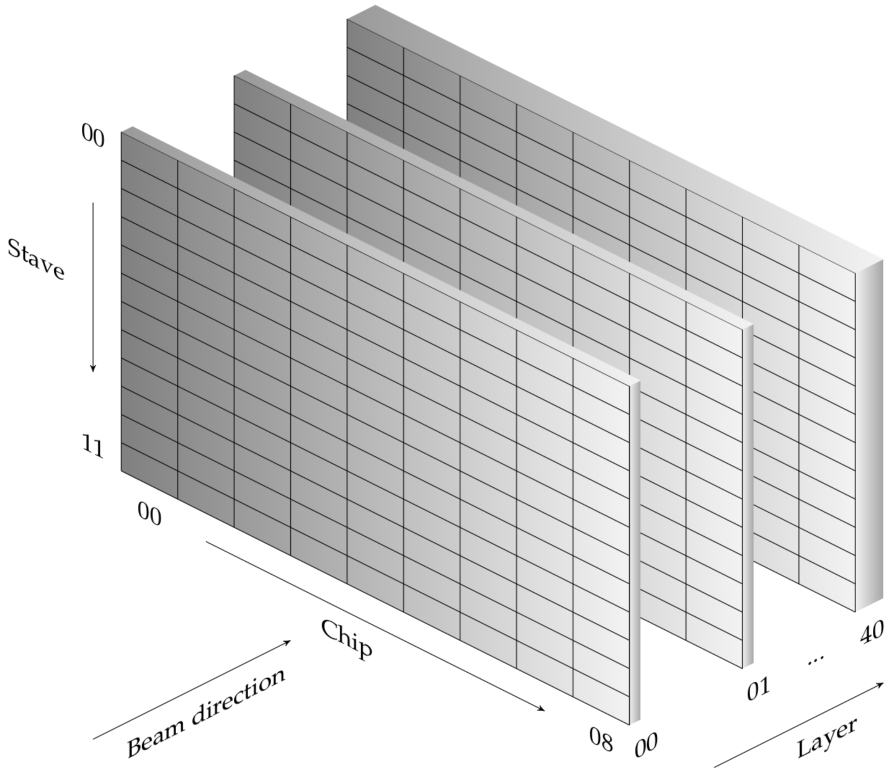


Figure 5.2: Logical geometry of pCT detector.
The first surface has LAYER_ID=L00. Stave's ID is along the y axis pointing down, CHIP's ID is along the x axis pointing to the right. ID number of the pRU increases along beam direction.

LAYER_ID gets leading "L", to mark layer field. It is better to use "L" when we are using layer terminology, than "R" for pRU, because it can be misinterpreted as the reading operation. A stave gets "S" and chip "C". Example: message to layer "01", stave "02" and chip "03", a write operation topic is shown in Table 5.2.

| | | | | | | |
|---|---|---|---|---|---|---|
| L01 | / | S02 | / | C03 | / | WR |

Table 5.2: A topic for a write operation to layer "01", stave "02" and chip "03"

To separate the outgoing flow of commands dedicated to a pRU from the other information received by the monitoring software, all topics on which a pRUs are publishing on have to begin with the "R" letter except event messages. Explicit "R" allows making subscription and filtration easier. In Table 5.3 shows topic structure of an message from pRU.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| R | / | LAYER_ID | / | STAVE_ID | / | CHIP_ID | / | OPERATION_TYPE |

Table 5.3: A general topic structure for a message from pRU.

### 5.1.2 Broadcasting of messages

The DTC contains 41 layers, 108 ALPIDE chips on each [14], so to address one chip on each layer, a broadcast topic structure is beneficial to speed up configuration and monitoring tasks. The broadcasting shortens message sequences also. There are several scenarios for broadcasting:

- Message to all layers, all staves, and all chips
- Message to all staves and chips on one layer
- Message to all chips on one stave
- Message to certain chip on all layers and staves
- Message to certain chip and stave on all layers

**Broadcasting to all layers, all staves and all chips** In this case, LAYER_ID, STAVE_ID and CHIP_ID numbers are replaced by "XX" and a write operation is defined as:

| | | | | | | |
|---|---|---|---|---|---|---|
| LXX | / | SXX | / | CXX | / | WR |

Table 5.4: A broadcast message to cover all layers, all staves and all chips

To utilized this functionality on the pRU, a double loop has to be implemented to go through all staves and chips. It will give significant speed-up in writing configuration as it will eliminate parsing messages to all staves and ALPIDEs on one pRU. An additional subscription has to be made to receive the broadcasting message on the topic "LXX/SXX/CXX/WR". It is a subscription to "LXX/#". A full overview of subscriptions in pCT will be given in section 5.1.3.

**Broadcasting to all staves and chips on one layer** In this case STAVE_ID and CHIP_ID numbers are replaced by "XX" and a write operation on layer "01" is defined as:

| | | | | | | |
|---|---|---|---|---|---|---|
| L01 | / | SXX | / | CXX | / | WR |

Table 5.5: A broadcast message for a specific layer. In this case layers "01".

On the pRU, we need the same as before except for the additional subscription.

**Broadcasting to certain chip on all layers and staves** In this case, LAYER_ID and STAVE_ID numbers are replaced by "XX" and a write operation for chip "02" is defined as:

| | | | | | | |
|---|---|---|---|---|---|---|
| LXX | / | SXX | / | C02 | / | WR |

Table 5.6: A broadcast message to a specific chip on all layers and staves. In this case chip id is "02".

This requires only one loop to go through all staves and the subscriptions to "LXX/#".

**Broadcasting to certain chip and stave on all layers** In this case, LAYER_ID numbers are replaced by "XX" and a write operation on stave "01" and chip "02" is defined as:

| | | | | | | |
|---|---|---|---|---|---|---|
| LXX | / | S01 | / | C02 | / | WR |

Table 5.7: A broadcast message to certain chip and stave on all layers. Stave with id "01" and chip with is "02" in this example.

A subscription on the topic "LXX/#" is needed, and there is no need for a loop on pRU.

### 5.1.3 MQTT subscriptions in pCT

Subscriptions in the MQTT allow receiving only messages on topics of interest. The MQTT topic structure described above, together with the broadcasting method, allows subscription on separate message flows in the pCT. In the MQTT, a subscription can be specified explicitly or with wildcards. There are two types of wildcards: + and #. The + wildcard is used for a single level of the hierarchy. For example, subscription on "R/+/S01/C02" will give response messages from stave 01 and chip 02. The + wildcard can be used many times in one subscription. For example, "R/+/+/C01" will give responses from chips with ID 01. The # wildcard is for all remaining levels of hierarchy. E.g., subscription on "R/#" will give messages from all pRUs.

#### Monitoring software subscription

The monitoring software has a subscription on "R/#" an on "EVENT/#'". It allows for having logging software and additional monitoring software to be run on other locations.

#### pRU subscription

One pRU will handle all staves on one layer. In this case, the pRU with given Layer_ID=N1 has to subscribe on the following topics: "LXX/#", "LN1/#". In case more than one pRU would be needed to one layer, the subscription will be the same, but on the pRUs, filtration has to be done according to the connected staves.

**Receiving of broadcast messages** To utilize a broadcast subscription, the pRU layer "01" will have to subscribe both on topics "L01/#" and "LXX/#". A subscription on broadcast topics uses only one extra line of code on pRU side but allows receiving a broadcast message. Any publishing on topics starting either with "L01" or "LXX" will be received by pRU with ID "01". In case of broadcasting to all chips on one layer, a function has to be written on pRU side.

**Receiving broadcast messages for one chip on each layer** Broadcasting to certain CHIP_IDs on all layers does not need any changes in software on pRU side, since pRU will perform only one operation per ALPIDE chip.

### 5.1.4 MQTT message flows in pCT

Making use of the topic structure described above, the monitoring and logging tasks can be separated. It is an advantage to have a logging module run on a dedicated machine due to the filtration and analysis that has to be done before saving to the database. All communication in MQTT goes through the broker. Figure 5.3 is illustrating the concept. An additional monitoring software can be introduced for the investigation of problems. Other subsystems are omitted of the scope of this concept. The complete solution for the pCT system will be described in chapter 6.

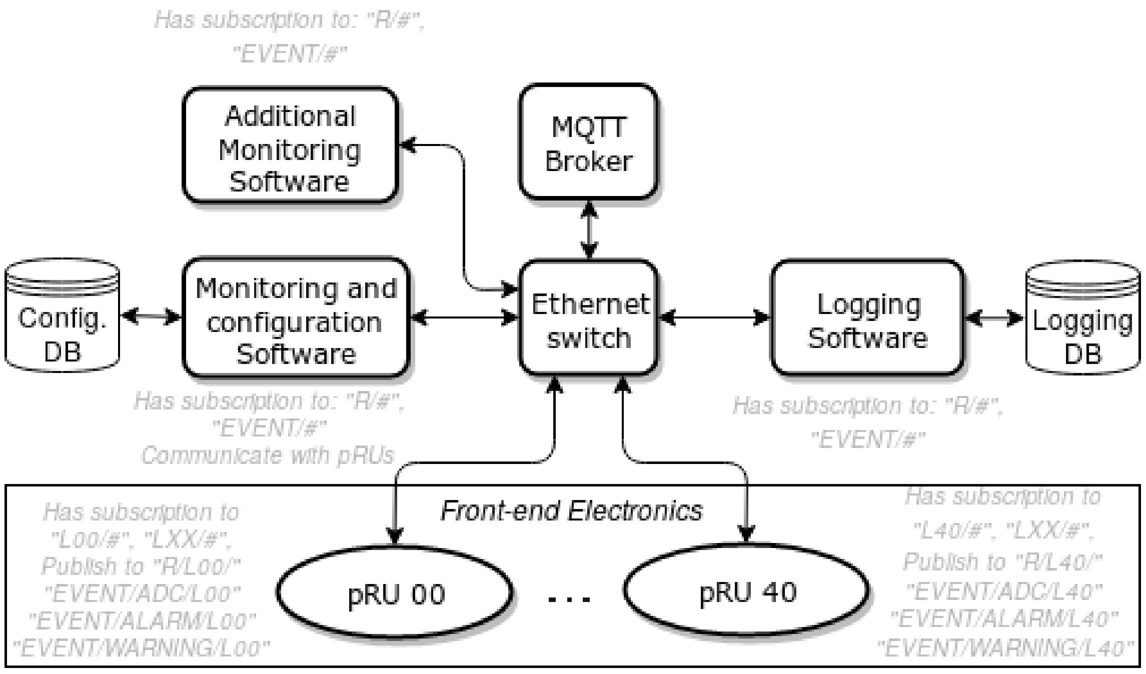

Has subscription to: "R/#", "EVENT/#"
Additional Monitoring Software
MQTT Broker
Config. DB
Monitoring and configuration Software
Ethernet switch
Logging Software
Logging DB
Has subscription to: "R/#", "EVENT/#" Communicate with pRUs
Has subscription to: "R/#", "EVENT/#"
Front-end Electronics
Has subscription to "L00/#", "LXX/#", Publish to "R/L00/" "EVENT/ADC/L00" "EVENT/ALARM/L00" "EVENT/WARNING/L00"
pRU 00
...
pRU 40
Has subscription to "L40/#", "LXX/#", Publish to "R/L40/" "EVENT/ADC/L40" "EVENT/ALARM/L40" "EVENT/WARNING/L40"

Figure 5.3: The pCT message flow with MQTT broker. Staves and ALPIDEs are not shown for simplicity. Connection of the subsystems will be discussed later in chapter 6.

#### A topic for event-based messages

To utilize the asynchronous communication, so called event messages can be sent. It allows being not bounded to polling. The EVENT topic has subtopics: Alarm, Warning, ADC, Info. The Alarm and the Warning topics are used to notify about not normal status of the operation. The ADC topic is for publishing readings of ALIDEs ADCs. Info is dedicated for logging and debug information when coordination between part of the system has to be verified.

### 5.1.5 Test framework for MQTT

A test framework is designed to visualize data coming from pRUs and to navigate through the layers of pCT to analyze the running parameters of all ALPIDEs. The data flow for the test is shown in Figure 5.4. It has a random data generator written in Python, a bash script to run multiple instances of the random data generator, MQTT broker, and a graphical interface to visualize incoming data written in Python. During the test, the MQTT broker and random data generation have been running on one machine while the test monitoring software on the other.

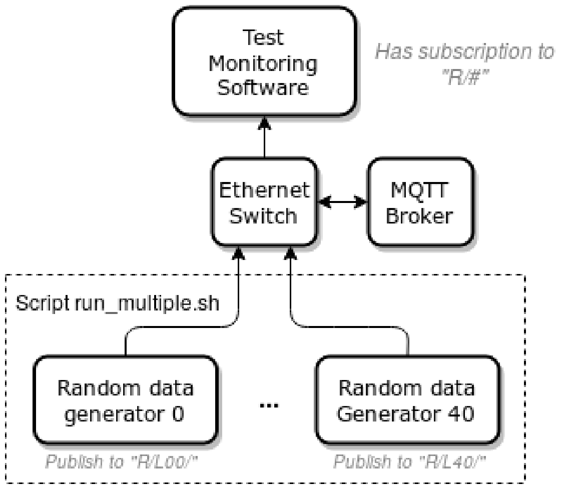


Figure 5.4: Block diagram for mqtt test with random data generators.

#### Random data generation

MQTT_pRU_simulation.py is written for a generation of random data to test the MQTT protocol with part of the proposed subscriptions. Only one direction of data flow have been tested because the full system is not implemented yet. The request-response time test have been done on the proton CT readout unit.

If no input parameters are specified for MQTT_pRU_simulation.py then it will be run in monitoring mode. It will print all messages on all topics in the command line. So it can be used for debugging. Filtration can be easily implemented in Python code if needed.

At the moment, it generates three sets of data. These sets represent temperature and voltage of ALPIDE chips on one layer and current measurements which will be available from transition boards. The transition board is under development, and it will manage the power for staves.

The ALPIDE's ADC is 10 bit, but only 8-bit values are used for random data generation. All simulated data are 8 bit, but the variation zone is different for temperature, voltage, and current, so that in the visualization program it will be possible to verify that the data is displayed correctly.

**Temperature random data** is in range from 0 to ration between a LAYER_ID and total number of layers. It is described by the formula:

$$255 * \frac{Layer\_ID}{number\_of\_simulated\_layers}$$

**Voltage random data** is in range:

$$63 * \frac{Layer\_ID}{number\_of\_simulated\_layers} + 64$$

It is colored in shadows of gray from black to white on visualization.

**Current random data** is in range:

$$63 * \frac{Layer\_ID}{number\_of\_simulated\_layers} + 128$$

The range will colored in shadows of gray from black to white on visualization.

### Content of run_multiple.sh

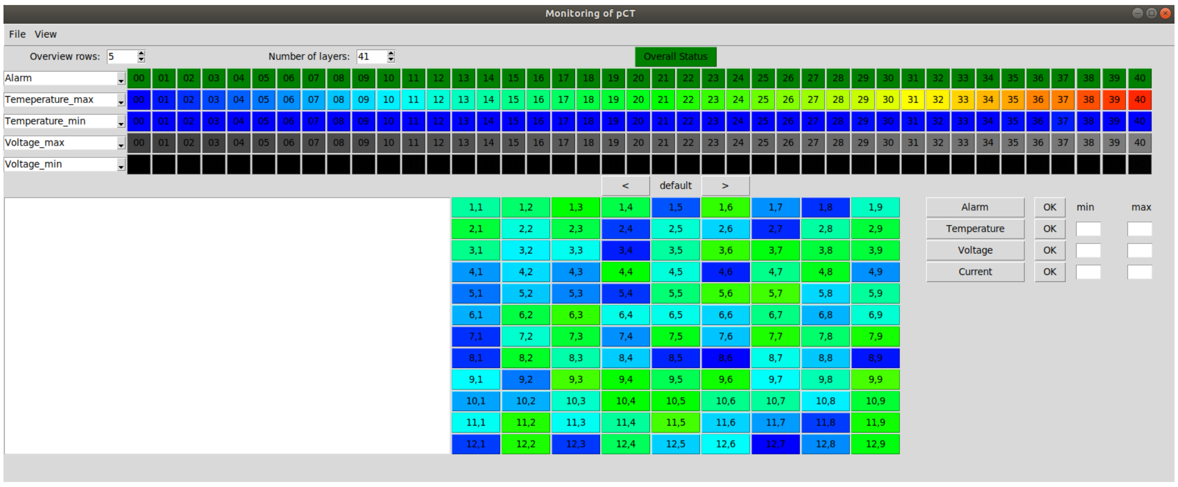
A bash script is written to run multiple Python files for data generation (see Listing 5.1). It takes as input parameters the version of Python, the filename, and number of instances to run. Then it runs separated processes and passes LAYER_ID to them. The LAYER_ID is used to calculate a proportional delay to spread the publishing through the monitoring period, which is one second for this test.

Listing 5.1: Content of run_multiple.sh

```
#Command line example below
#./run_multiple.sh python3 MQTT_RU_simulation.py 40
#bash script       py_ver. One_layer_simulation  ...
    number_of_simulated_layers
echo "Simulating $3 layers"

# Allow to find and close processes when terminal window is closed
trap "pkill -P $$; kill -INT $$" INT

while read n
do
# $n is Layer_ID, $3 is number_of_simulated_layers
"$1" "$2" $n "$3" &
done < <(seq $3)
wait
```

### Graphical interface for visualization of randomly generated data

To visualize randomly generated data in the MQTT test, a GUI has been developed (see Figure 5.5). The GUI is made with overview rows at the upper part. The number of rows and parameters for a row can be chosen. Alarm status in the first row



Overview rows: 5

Number of layers: 41

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Alarm | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| Temeperature_max | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| Temperature_min | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| Voltage_max | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| Voltage_min | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1,1 | 1,2 | 1,3 | 1,4 | 1,5 | 1,6 | 1,7 | 1,8 | 1,9 |
| 2,1 | 2,2 | 2,3 | 2,4 | 2,5 | 2,6 | 2,7 | 2,8 | 2,9 |
| 3,1 | 3,2 | 3,3 | 3,4 | 3,5 | 3,6 | 3,7 | 3,8 | 3,9 |
| 4,1 | 4,2 | 4,3 | 4,4 | 4,5 | 4,6 | 4,7 | 4,8 | 4,9 |
| 5,1 | 5,2 | 5,3 | 5,4 | 5,5 | 5,6 | 5,7 | 5,8 | 5,9 |
| 6,1 | 6,2 | 6,3 | 6,4 | 6,5 | 6,6 | 6,7 | 6,8 | 6,9 |
| 7,1 | 7,2 | 7,3 | 7,4 | 7,5 | 7,6 | 7,7 | 7,8 | 7,9 |
| 8,1 | 8,2 | 8,3 | 8,4 | 8,5 | 8,6 | 8,7 | 8,8 | 8,9 |
| 9,1 | 9,2 | 9,3 | 9,4 | 9,5 | 9,6 | 9,7 | 9,8 | 9,9 |
| 10,1 | 10,2 | 10,3 | 10,4 | 10,5 | 10,6 | 10,7 | 10,8 | 10,9 |
| 11,1 | 11,2 | 11,3 | 11,4 | 11,5 | 11,6 | 11,7 | 11,8 | 11,9 |
| 12,1 | 12,2 | 12,3 | 12,4 | 12,5 | 12,6 | 12,7 | 12,8 | 12,9 |

Figure 5.5: User interface prototype for pCT.

in GUI should accumulate the status for each layer and represent alarm or warning color code. The status row for temperatures, voltages, and currents have the option of either showing the maximum value or the minimum value of a layer. It allows so by choosing Temperature_max to get the maximum for the whole layer in the corresponding cell. According to the pattern of data generation of temperature, the minimum is zero, but the maximum is proportional to the LAYER_ID. In real operation, both maximum and minimum for each layer is essential as it will represent cooling efficiency. The distribution of a layer can be seen in the center of GUI. Buttons can do navigation through layers with signs "<" and ">" or by clicking on a cell of the status row, then the corresponding layer will be displayed.

A special conversion procedure has been programmed (see Listing 5.2), to visualize the temperatures, voltages, and currents. It converts a value in the range from 0 to 1023 to RGB color code for temperature or in shadows of grey for others. It is colored from blue (cold) to red (hot) on visualization. The transitions for the temperature is shown in Figure 5.6.

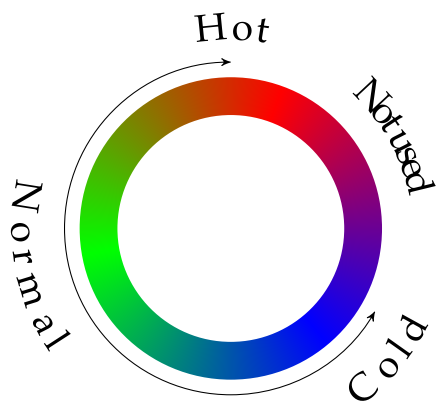


Figure 5.6: Color range for the ALPIDE temperature.

The goal of this algorithm is to give a color representation of a value. It allows to spot anomalies easily. When working limits for a parameter will be known, the algorithm can be changed to have variation starting at the minimum and continue to the maximum threshold. It allows having a better color resolution in the interested range. Voltage and current data use the shades of the grey output of the algorithm. They also have different ranges to verify proper parsing in GUI (see Figure 5.5).

Listing 5.2: Value to color convertoin algorithm. Part of the file MQTT_GUI.py

```
def convert_value_to_rgb_hex(value, param=-1):
""" 1024 values to variation from Blue to Red through Green.
#1 Goes from Blue to Green by incresing Green.
#2 Decrease  Blue to get pure Green
#3 Increase  Red to get Orange
#4 Decrease  Green to get pure Red
Suggest 255(8bit) max on value
"""
if param == 0:
value=value*4 #Suggest 255 max on value. We have 1024 max
if value < 256: #1
color = '#%02x%02x%02x' % (0,value,255)
elif value ≥ 768: #4
color = '#%02x%02x%02x' % (255,1023-value,0)
elif value < 512: #2
color = '#%02x%02x%02x' % (0,255,511-value)
elif value ≥512: #3
color = '#%02x%02x%02x' % (value-512,255,0)
if param ≥ 1:
#Shadows of grey output
color = '#%02x%02x%02x' % (value,value,value)
return color
```

### 5.1.6 MQTT request-response time test.

Time used on request-response turnaround (loop) is essential for estimating delay in monitoring in a control system. This also gives the reaction time in closed-loop regulation. To find out the minimum response time with the MQTT broker showed in Figure 5.7 was used. The test is using a single Gigabit link when the MQTT broker and test software is running on the same PC. It is a simplified configuration, but results will give a minimum possible delay.

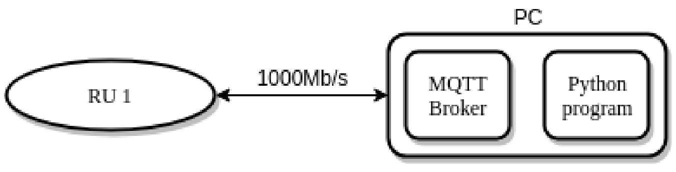


Figure 5.7: Request-response time test configuration.

Based on the tests made request-response time is 5 ms. Given that we want to monitor approximately 1000 values per second, a request response time of 5 ms is too slow. It is better to send commands grouped in a packet to achieve maximum speed.

### 5.1.7 Summary

The test framework has verified that the MQTT broker can handle the needed amount of data, and the data can easily be visualized. A color representation of data has been developed. The topic structure described above has been tested on the way to the control software prototype. The request-response time achieved implies that it is beneficial to pack more configuration commands in one packet, if possible. It requires a busy flag implementation on the pRU, which will let the control software know if the limit for incoming commands is reached.

## 5.2 OPC UA

For testing OPC UA technology, Open62541 has[1] was selected. Open62541 is an open-source, cross-platform client and server implementation of OPC UA written in C99. It is developed by three German universities and some commercial companies. Open62541 has commercial support offerings from two partners.

The latest release version of Open62541 is 1.0. Documentation is available online for releases and development branches. The work with the repository has shown that the Publish-Subscribe functionality for the client API is not finished. This feature is required for the asynchronous reception of data from the server. However, Open62541 gives insights on how an OPC UA system is working. Further steps with OPC UA will be discussed in section 7.2.

A server build from Open62541 repository has been tested with a free tool of a commercial SDK, - UaExpert from Unified Automation. UaExpert is a cross-platform OPC UA client.

### 5.2.1 Information modeling

Information modeling is the first step in the development of an OPC UA system. At this step, low-level functionalities are planned to meet top-level requirements. The information model expresses the semantic of the physical system in objects, data types, methods, events, and references. It requires knowledge and hands-on experience of OPC UA standard. OPC UA standard allows creating an information model in the way that it represents the semantics of the system very closely and will make communication efficient. OPC UA has an object-oriented approach.

The file format for an information model is XML. It contains references to namespaces, which are used. In our case a new namespace is created for the special pCT

[1]open62541.org

related objects, methods, classes etc (see Appendix B.2). At the end of the XML file, objects are defined to verify the structure by the UaModeler tool from Unified Automation.

To verify the information model file, UA-ModelCompiler from OPC Foundation is used. It checks consistency and integrity and generates a set of files: NodeSet2.xml, Types.bsd, Nodes.csv, and some other files. The first three files are used for import to the Open62541 OPC UA stack. NodeSet2.xml file is the official format for a custom information model. Depending on the design, the NodeSet2 file can be substantially larger than the XML source file.
Nodeset2.xml file is imported in UaModeler for visual verification. The set of files generated by UA-ModelCompiler can be imported to any OPC UA SDK or server. It means that objects described in the information model can be created inside an OPC UA server. The information model XML file is a design file for an OPC UA system. In Figure 5.8 OPC UA workflow is illustrated.

The design of an information model can be done by some GUI-based tools, but in practice, a manually written XML file is more convenient. It is more readable, and it is more comfortable to add new functionality. The manual approach requires an understanding of OPC UA and OPC UA specific XML language.

### 5.2.2 OPC UA Information model for pCT control system

The information model for pCT has to include object classes for pRU, power, cooling, and beam systems. Creation of a new object type require a reference to the *BaseType* which is *"OpcUa:BaseObjectType"*. "OpcUa" in front of "BaseObjectType" is the namespace. A pCT objects and object types will have "pCT" as the namespace. It allows to avoid mixing the same names in different namespaces if they exist. In the next sections, OPC UA models developed for pCT will be described.

#### The power, cooling and beam subsystems

For managing power, cooling and beam system, a new base class is needed with two methods: Turn_ON, Turn_OFF; and property Is_ON. In Figure 5.9 the beam system represents an object of "SubsystemObjectType"[2]. This minimum functionality allows to turn on and off, and read the status Is_ON. Methods can have input and output parameters, but at the moment only output parameters are used to return the result of calling a method.

The cooling system in Figure 5.10 inherits from "SubsystemObjectType" and have in addition parameters: Temperature_In, Temperature_Out and Flow. These are needed for the calculation of cooling performance. Power system with one output will require methods "Set_Voltage" and "Set_Current_Limit" and parameters "Voltage" and "Current" for reading values back. See Figure 5.11

[2]Appendix B.2 line 20

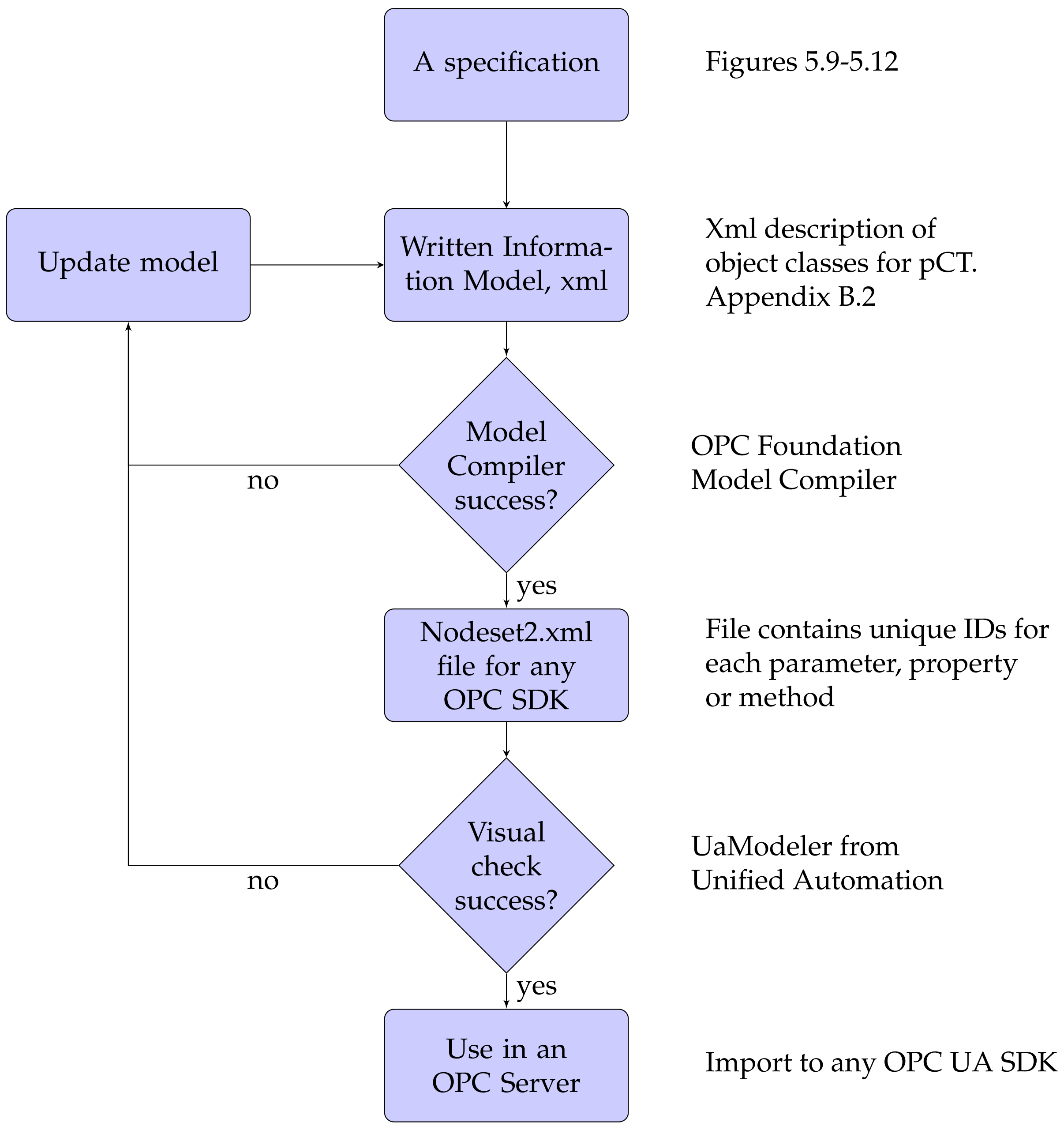

A specification
Figures 5.9-5.12
Update model
Written Information Model, xml
Xml description of object classes for pCT. Appendix B.2
Model Compiler success?
no
OPC Foundation Model Compiler
yes
Nodeset2.xml file for any OPC SDK
File contains unique IDs for each parameter, property or method
Visual check success?
no
UaModeler from Unified Automation
yes
Use in an OPC Server
Import to any OPC UA SDK

Figure 5.8: OPC UA workflow

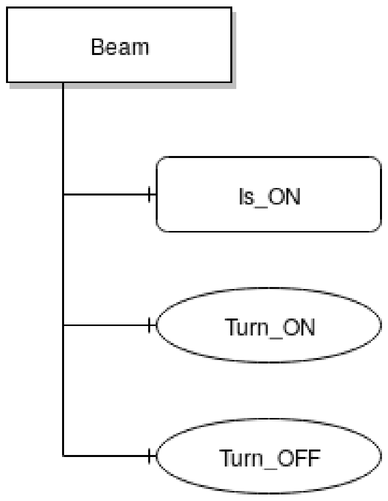


Figure 5.9: OPC UA Information Model of the beam systems[a].
[a] See for OPC UA graphical notations in Appendix B.1

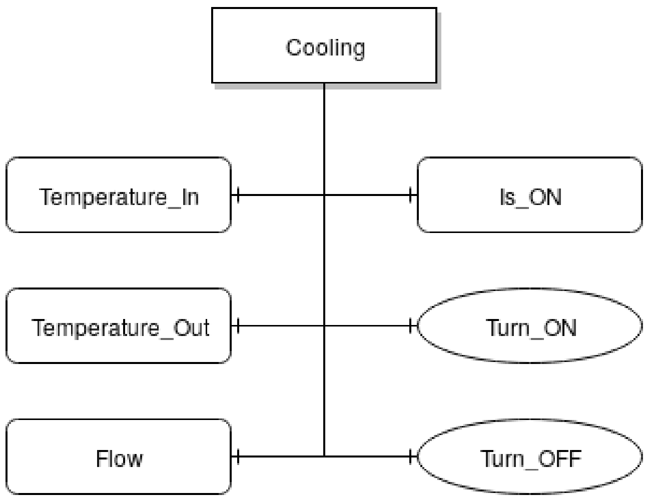


Figure 5.10: OPC UA Information Model of the cooling systems.

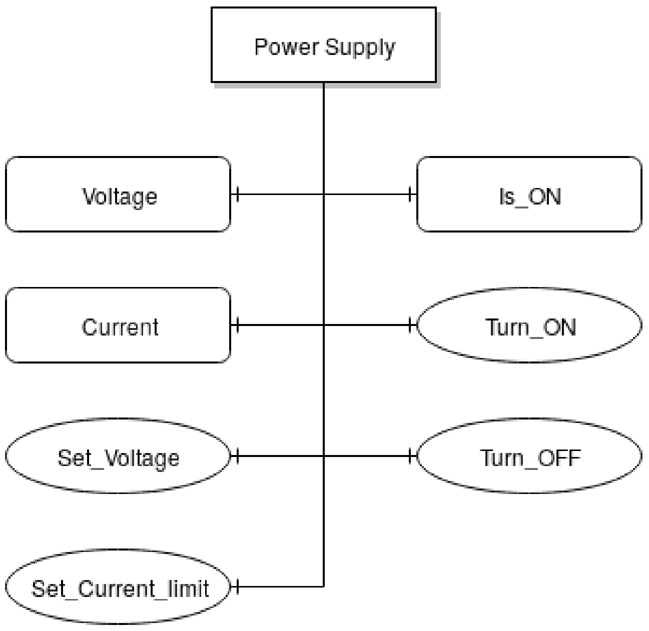


Figure 5.11: OPC UA Information Model of the power systems.

### The pRU

The pRU is the most complex OPC UA object in the pCT instrument. It represents the structure of the pRU with submodules like FPGA, staves and ALPIDEs by using OPC UA terminology (see Figure 5.12). To design a complete object type, we need all subtypes. It means that the design will go from the bottom upwards. To model the ALPIDE chip, two methods are required: Register_Write and Register_Read. Access to the ADC registers inside the ALPIDE is by reading and writing to registers. However, it is more convenient to have ADC as a subtype of ALPIDE. Then the ADC can have methods for calibrations and parameters for the status of ADC. Results of the ADC measurements have to be part of the topology to allow subscribe on changes. In Figure 5.12 only temperature, digital (DVDD) and analog (AVDD) power supplies, digital (DVSS) and analog (AVSS) grounds is shown for simplicity. To provide results of the ADC measurements, an event can be generated to send data. It is shown by reference "HasEventSource" from the ALPIDE object to the ADC object.

During operation of the pCT when the beam is on and the system register particles, the Scan_Is_ON property is used. The reference "HasEffect" from Scan_Is_On to ALPIDE, ADC and Register_Write and is meaning that the functionality of the ALPIDE is limited. The ADC will not start a measurement Scan_Is_ON is true. Result of calling the Register_Write method will not return success. The reason for this is that the ALPIDE control bus might be busy issuing triggers to initiate data readout.

All object types for sub objects of the pRU have to be defined once. Modeling rules are used to mark the parameter, object, or method is required when an object is instantiated. It allows to specify the amount of ALPIDEs in each stave and the number of staves in the pRU. Then a pRU object can be created in few lines of code with all submodules (see Appendix B.2 line 159). For the UaModeller view of the pCT see Appendix B.3.

### 5.2.3 Summary

During the OPC UA development and testing phase, a more structured approach was considered. The pRU object was designed relatively fast by using OPC UA terminology. The rich semantic of the OPC UA allows for developing an information model in a way that it can be controlled by a high-level software.

## 5.3 Top-level control system

That all monitored data are stored at one place is not enough. To operate the pCT instrument requires a Finite State Machines (FSM). This is the place where control logic is concentrated. In Figure 5.13 an example of FSM is given that utilizes the OPC UA functionality described in section 5.2. The complete set of finite state machines for the pCT has not been designed as it was outside of the scope of this thesis.
The GUI development for MQTT tests had a top-level control system in mind (see Figure 5.5). It structured hierarchically with overall status at the top. The color representation of a values are dedicated to the detection of erroneous situations. Status rows have been introduced to highlight the maximum and minimum values of each layer. A mouse click on a cell of any status row will change the active layer in the middle to appropriate layer and parameter. On the bottom-left is a logging window for events, alarms, and warnings. At the right side from layers view, there are buttons for navigation between monitored parameters with text windows for showing limits for alarm generations. A similar control system interface can be envisaged no matter if the underlying protocol is MQTT or OPC UA.

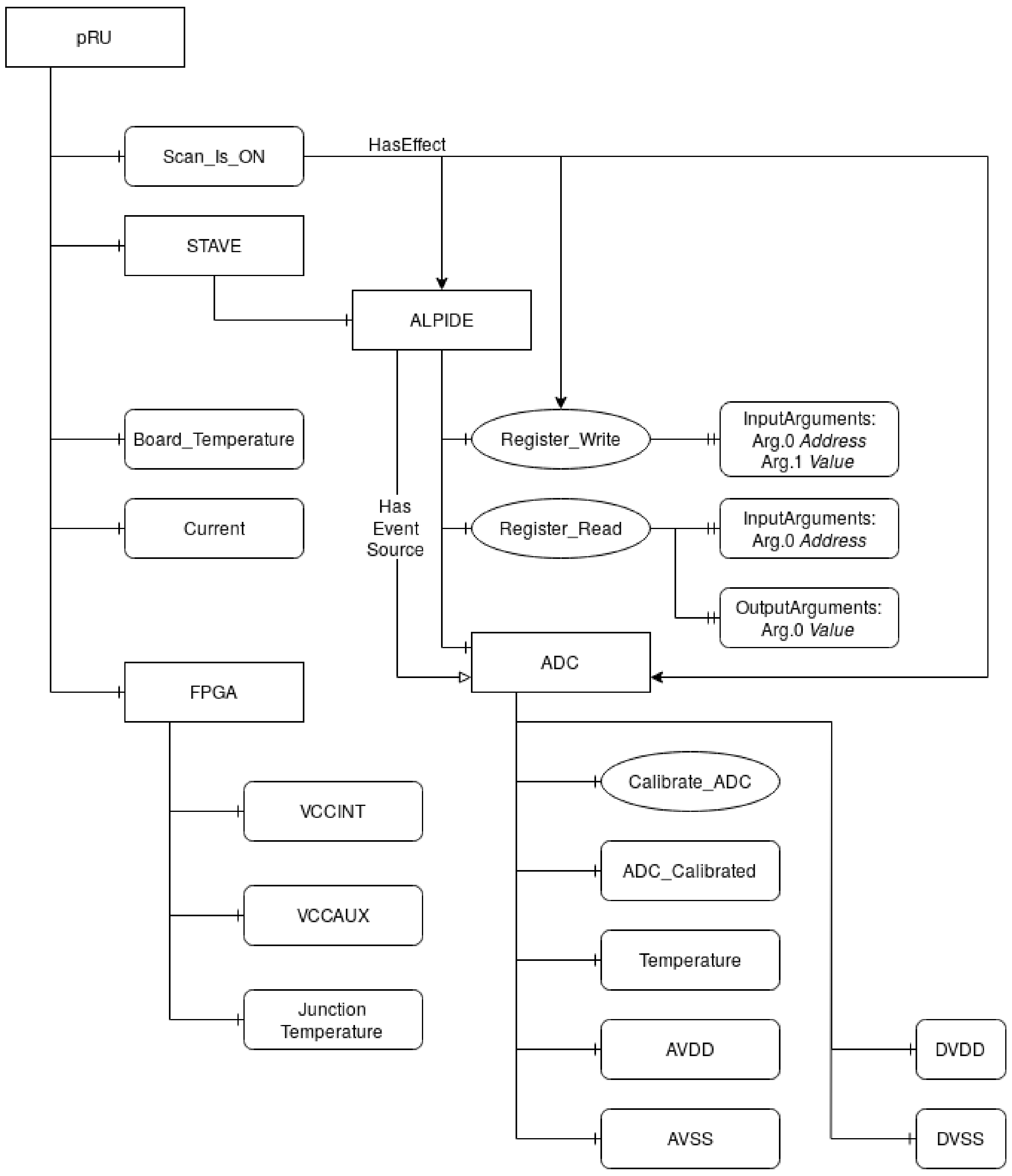


Figure 5.12: OPC UA Information Model of the pCT.

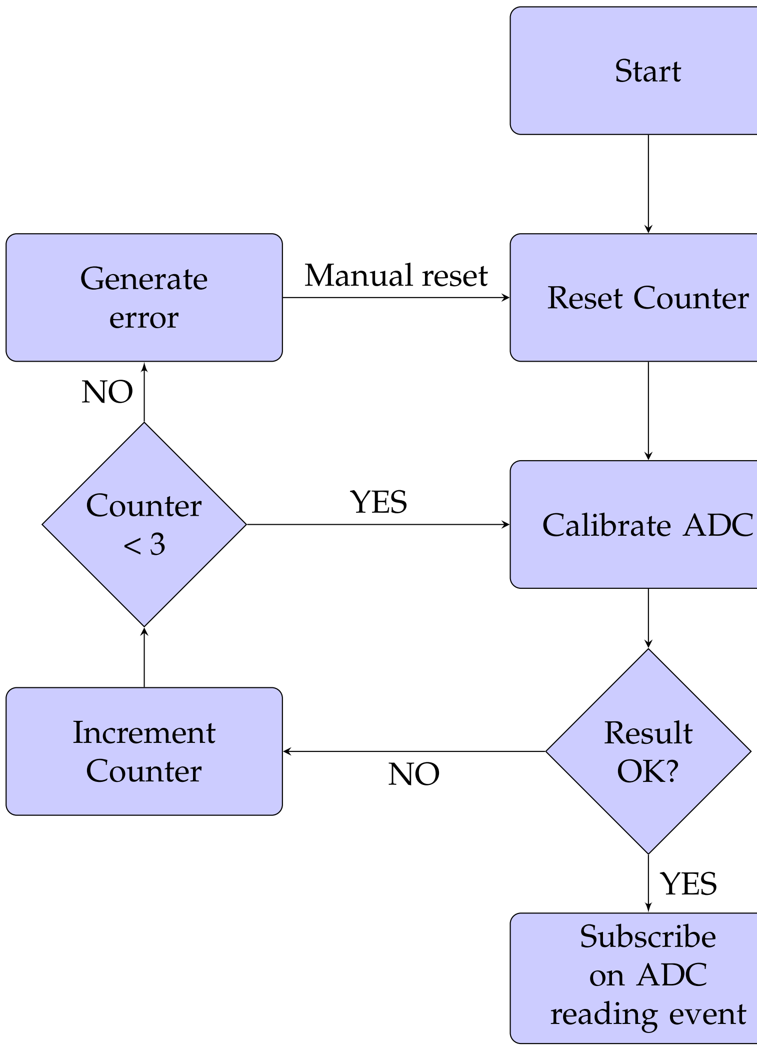


Figure 5.13: An example of FSM for receiving of ADC measurements

# Chapter 6

# Proposal of Production Version Control System Architecture

In this chapter, the pros and cons will be weighed against the requirements described in chapter 4. The primary candidates for the control system communication protocol are OPC UA and MQTT. The DIM protocol is not chosen as it is more complicated than MQTT. The DIM is also not an industry-proven protocol and has no commercial support. In this section, three scenarios will be described: MQTT only, OPC UA only, and a combination of both.

This section is based on that the design proposal of pRu will be accepted (see Figure 3.4, [20]). It means that transition cards are custom made for the transfer of data, control, and power to the ALPIDE layers. Power monitoring on transition cards is expected per stave. The Ethernet interface on the transition cards has not been specified. It means that it will probably be controlled via the FPGA on the pRU. The pRU will then be responsible for the control of the transition cards. The power control unit have not been specified in detail in [20].

## 6.1 MQTT only

The MQTT protocol has no graphical notations like OPC UA due to its simplicity, but the object-oriented approach can be used here. It means that the OPC UA information model developed for the pCT can be used as the specification, and the topic structure has to be made to allow to call appropriate methods and receive replies. The event-based messages can be published on the dedicated topic "EVENT".[1] The cooling and the beam systems can be managed by one micro-controller and use one connection. The power system most likely will be an industrial component, and to communicate with it an interface converter will be needed. A MQTT client located

[1]More detail about the MQTT subscriptions in section 5.1

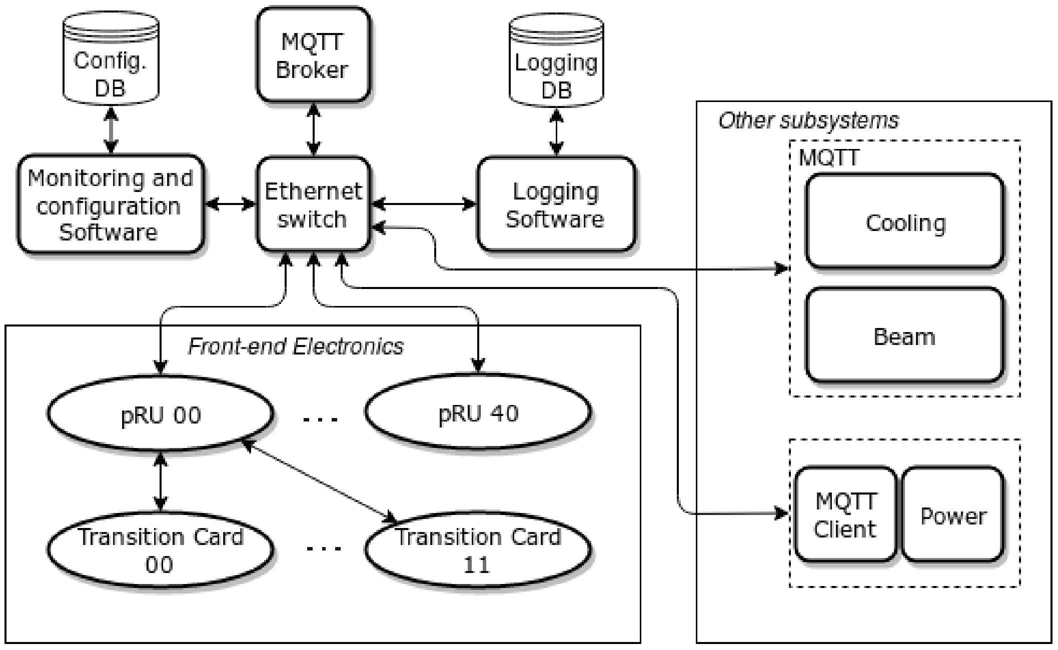


Figure 6.1: Control connetions in pCT with MQTT as communication protocol.

next to power supply to control it. In Figure 6.1 the concept is shown.

The framework that has been developed to visualize data and described in section 5.1 can be used in this case without modifications. The MQTT protocol was proven to be easy to implement as well as transparent. The MQTT protocol has been implemented on the PRU test setup described in section 3. At the current stage with many unknowns in the pCT system, the MQTT will give an advantage in the integration of custom devices into the pCT control system.

## 6.2 OPC only

When considering OPC as a solution for all components in the pCT, OPC UA servers have to be placed near all sources of information or in need of control. An aggregating server can be introduced to concentrate information distributed over front-end electronics at one place if communication with an external system is needed. It will "unload" OPC UA servers on pRUs. It is possible with one-to-one and one-to-many architectures due to the built-in subscription functionality in the OPC UA.

In Figure 6.2 the OPC UA topography for the pCT is shown. In the OPC UA, there is no broker as in MQTT. During prototyping in section 5, OPC UA has not been tested with FreeRTOS. To estimate possibility to monitor parameters chosen in section 4 metric from Matrikon SDK is used. The CPU utilization with OPC server is shown in Table 6.1. At the first line, hardware similar to a soft-core MicroBlaze processor is shown. I.e. sampling and reporting at a rate about 1000 tags per second can be expected in the current setup. This satisfies the R.2 requirement.

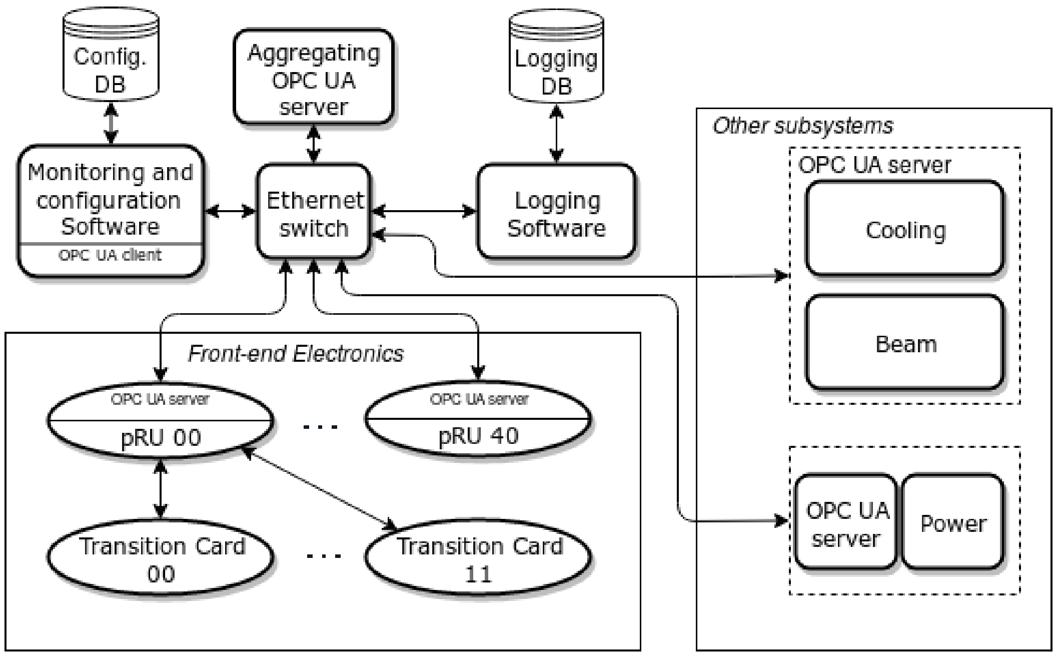


Figure 6.2: OPC UA topography for the pCT.

| Test | Conditions | Hardware | CPU Utilization (%)[a] |
|---|---|---|---|
| 100 continuously changing tags | Sampling and reporting every 100 ms | ARM Cortex-M4F (STM32F407) @ 168MHz | 12.5 |
| 1000 continuously changing tags | Sampling and reporting every 100 ms | ARM Cortex-A8 (AM3359) @ 1GHz | 31.0 |

Table 6.1: OPC server performance for Matrikon SDK[28].
[a] Metrics obtained using GCC-03

In this case, the power system will require an OPC UA server. Industry components are getting native OPC UA support now. An appropriate power supply with OPC UA functionality will likely be found for the pCT needs. There is a software framework for generating OPC UA servers developed at CERN. The results of this development will be available commercially in the market[2].

Publish/subscribe functionality has not been tested due to the fact that the API in Open62541 is not finished. It is required for sending monitoring information asynchronously from pRUs. So it is currently not possible to implement a full-scale OPC UA control system with Open62541.

[2]https://kt.cern/success-stories/quasar-project

## 6.3 OPC and MQTT

An option will be considered here for pCT where both MQTT and OPC UA is used (see Figure 6.3). This is an attempt to use the best of both. There are several motivations to use MQTT technology. First is transparency - all communications between all clients can be observed. It is essential for debugging. MQTT is also easy to implement, has small memory footprint and CPU utilization.

Reasons to include OPC UA in this solution is that industrial power supplies are expected to be with native OPC UA support and communication with an external system will most probably use OPC UA as it become a standard transport protocol for SCADA systems(WinCC Siemens is an example).

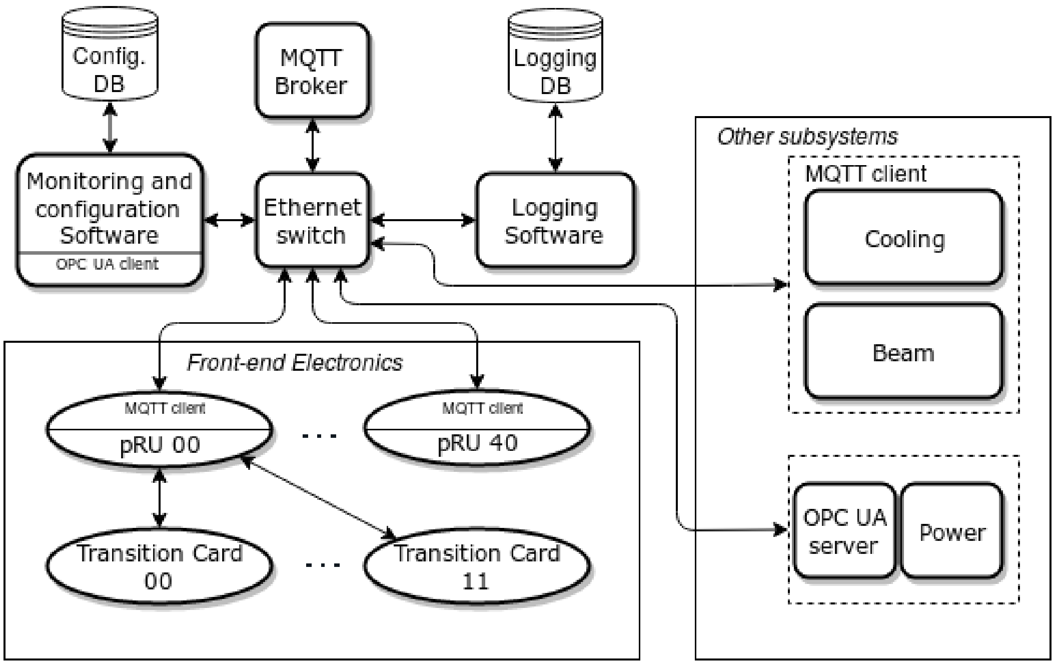


Figure 6.3: Both MQTT and OPC is used for the pCT. Layers and staves are not shown.

Despite the use of two technologies the total efforts on implementation of the working system is expected to be less that option with OPC UA only. A power supply for the pCT will be product "off the shelf" with OPC UA functionality and efforts on communication with it is less that implementation the whole OPC UA system.

# Chapter 7

# Conclusion and future work

## 7.1 Conclusion

Hands-on experience with the two technologies suggested in a previous master thesis[22] has been gained. It reveals the strengths and weaknesses of both. MQTT is very light and protocol that is easy to implement. It allows to set-up one-to-many communication by subscribing to interesting topics, however, the data formats and package structure have to be developed. Taking in account the MQTT message flow shown in Figure 5.1, all information has to go through the MQTT broker. It is a disadvantage for high load situation.

OPC UA is a sophisticated object-oriented protocol but in concept easy to port from one server implementation to another by using XML format for information model. However, the publish-subscribe feature in OPC UA was released in February 2018. Due to this reason, it is still work in progress in Open62541 on publish-subscribe client API, and the desired asynchronous patter of communication has not been tested. A commercial SDK instead of Open62541 has to be considered for the pCT. Some tools from Unified Automation SDK for OPC UA have been tested, and it advised to be used.

An OPC UA information model for the pCT has been developed. The XML design file has been imported to a commercial SDK . The design of the OPC UA information model helps to structure the specification towards the final system.

The option based on MQTT is suitable for the current state of the pCT project due to that all communication in MQTT can be logged easily, in OPC UA not. However, the integration of power supplies will require an MQTT driver to control it. The use of OPC UA communication protocol is the best option for the final system considering that more industrial power supplies and SCADA systems coming with native OPC UA support. The development of the OPC UA servers require more knowledge and efforts in comparison to MQTT, but the communication in OPC UA is highly standardized. It will lead to fewer issues during start-up of the control system.

## 7.2 Future work

If OPC UA is selected as a communication protocol for the pCT, the OPC UA SDK has to be chosen. It will be beneficial to have training implementation of particular SDK. The information model developed in this thesis has to be adapted to and tested on a full-scale prototype when it will be available. In the next steps, the design of the relevant FSM must be done. A framework for the development of the FSM has to be chosen as well. A SCADA system is needed to incorporate FSM and functionalities of the pCT system. Due to the lack of personal experience with SCADA systems, it is difficult to suggest an option, but overview over open-source SCADA systems with OPC UA functionality have not shown well supported candidates. A commercial SCADA system have to be considered. It has to be a member of OPC Foundation to offer compatibility with all features of the OPC UA protocol.

# Bibliography

[1] Brookhaven National Laboratory *"NSRL User Guide: Bragg Curves and Peaks"* https://www.bnl.gov/nsrl/userguide/bragg-curves-and-peaks.php

[2] W. P. Levin et al, 2005 *"Proton beam therapy"* British Journal of Cancer. 93 (8): 849–854.

[3] S. Chera et al, 2009 *"Proton Therapy for Maxillary Sinus Carcinoma"* American Journal of Clinical Oncology, Vol.32(3), pp.296-303

[4] S. E. Combs, 2019 *"The Benefits of Particle Therapy in Cancer Treatment"* Lecture

[5] J.E. Leeman et al, 2017 *"Proton therapy for head and neck cancer: expanding the therapeutic window"*

[6] H.E. Seime Pettersen et al, 2017 *"Proton tracking in a high-granularity Digital Tracking Calorimeter for proton CT purposes"* Nuclear Inst. and Methods in Physics Research, A 860.C : 51-61. Web.

[7] E. Schnell et al, 2016 *"Commissioning of a relative stopping power to Hounsfield unit calibration curve for a Mevion proton radiation treatment unit"* Article

[8] H.E. Seime Pettersen, 2018 *"A Digital Tracking Calorimeter for Proton Computed Tomography"*

[9] P. Chochula et al, 2018 *"ALICE Detector Control System Management and Organization"* Presentation

[10] P. Chochula et al, 2010 *"The ALICE Detector Control System"* IEEE Transactions on Nuclear Science, Vol.57(2), pp.472-478

[11] C. Gaspar et al, 2001 *"DIM, a portable, light weight package for information publishing,data transfer and inter-process communication"* Computer Physics Communications, Vol.140(1), pp.102-109

[12] *"DIM - Distributed Information Managment System"* https://dim.web.cern.ch/

[13] P. Chochula, 2018 *"The evolution of the ALICE Detector Control System"* Presentation

[14] H. Shafiee, 2019 *"Prototyping of a Tracking Calorimeter for Computed Tomography in Proton Therapy"* Presentation

[15] ALICE Collaboration, 2008 *"The ALICE Experiment at the CERN LHC"* Journal of Instrumentation 3.08 : S08002. Web.

[16] A.D. Mauro, 2019 *"The New Inner Tracking System for the ALICE Upgrade at the LHC"* Nuclear Inst. and Methods in Physics Research, A 936: 625-29. Web.

[17] P. Gasik, 2018 *"Upgrade of the ALICE Central Barrel Tracking Detectors: ITS and TPC"* Web.

[18] B Abelev et al, 2014 *"Upgrade of the ALICE Experiment: Letter Of Intent"*

[19] ALICE ITS ALPIDE development team, 2016 *"ALPIDE operational manual"*

[20] O. Grottvik et al, 2019 *"Design proposal for the pCT Readout Unit (pRU)"*

[21] H.A. Schaug, 2017 *"Proton computed tomography readout testing and detector design"*

[22] K.E. Sandvik Bohne, 2018 *"Ethernet-Based Control System and Data Readout for a Proton Computed Tomography Prototype"*

[23] EEE Standards, 2008 *"IEEE Standard for SCADA and Automation Systems"* Piscataway, USA: IEEE. Web.

[24] Embedded Market Survey, 2017 https://www.embedded.com/electronics-blogs/embedded-market-surveys/4458724/2017-Embedded-Market-Survey

[25] P. Chochula, 2018 *"The evolution of the ALICE Detector Control System"* presentation

[26] A. Augustinus et al, 2018 *"ALICE DCS preparation for Run 3"* presentation by Alexander Kurepin

[27] R.A. Light, 2017 *"Mosquitto: server and client implementation of the MQTTprotocol"* The Journal of Open Source Software, vol. 2, no. 13, DOI: 10.21105/joss.00265

[28] MatrokinOPC, 2015
https://opcfoundation.org/wp-content/uploads/2015/03/Keys-To-Developing-an-Embedded-UA-Server_Whitepaper_EN.pdf

[29] https://www.xilinx.com/support/documentation/white_papers/wp501-microblaze.pdf

[30] S. Chacon et al, 2018 *"Pro Git"* Version 2.1.84

[31] D. Kim et al, 2016 *"Front end optimization for the monolithic activepixel sensor of the ALICE Inner Tracking Systemupgrade"*

[32] B. Chaudhary, 2013 *"Tkinter GUI application development hotshot"*

[33] B. Franek et al, 2005 *"SMI++ Object Oriented Framework for Designing and Implementing Distributed Control Systems"* IEEE Transactions on Nuclear Science, Vol.52(4), pp.891-895

# Appendix A

# ALPIDE

## A.1 ADC readings functions in TAlpide.cpp

Here a part of *pct/alpide-sw/TAlpide.cpp* file is listed to show implemented procedures for reading temperature, voltage and current for comparison with description in ALPIDE Operation manual[19].

```
/* ------------------------------------------------------
* Reads the temperature sensor by means of internal ADC
*
* Returns  : the value in Celsius degree.
*
* Note  : if this was the first measure after the chip
*         configuration phase, a calibration will be
*         automatically executed.
*
*/
float TAlpide::ReadTemperature() {

uint16_t theResult = 0;
float theValue;
if(fADCBias == -1) { // needs calibration
CalibrateADC();
}

SetTheDacMonitor(Alpide::REG_ANALOGMON); // uses the ...
   RE_ANALOGMON, in order to disable the monitoring !
usleep(5000);
SetTheADCCtrlRegister(Alpide::MODE_MANUAL, ...
   Alpide::INP_Temperature, Alpide::COMP_296uA, Alpide::RAMP_1us);
fReadoutBoard->SendOpCode ( Alpide::OPCODE_ADCMEASURE,  ...
   fConfig->GetChipId());
usleep(5000); // Wait for the measurement > of 5 milli sec
ReadRegister( Alpide::REG_ADC_AVSS, theResult);
theResult -=  (uint16_t)fADCBias;
theValue =  ( ((float)theResult) * 0.1281) + 6.8; // first ...
```

```
    approximation
return(theValue);
}

/* ----------------------------------------------------
* Reads the output voltage of one DAC by means of internal ADC
*
* Parameter : the Index that define the DAC register
*
* Returns  : the value in Volts.
*
* Note   : if this was the first measure after the chip
*          configuration phase, a calibration will be
*          automatically executed.
*
*/
float TAlpide::ReadDACVoltage(Alpide::TRegister ADac) {

uint16_t theResult = 0;
float theValue;
if(fADCBias == -1) { // needs calibration
CalibrateADC();
}

SetTheDacMonitor(ADac);
usleep(5000);
SetTheADCCtrlRegister(Alpide::MODE_MANUAL, ...
    Alpide::INP_DACMONV, Alpide::COMP_296uA, Alpide::RAMP_1us);
fReadoutBoard->SendOpCode ( Alpide::OPCODE_ADCMEASURE,  ...
    fConfig->GetChipId());
usleep(5000); // Wait for the measurement > of 5 milli sec
ReadRegister( Alpide::REG_ADC_AVSS, theResult);
theResult -=  (uint16_t)fADCBias;
theValue =  ( ((float)theResult) * 0.001644); // V scale first ...
    approximation
return(theValue);
}

/* ----------------------------------------------------
* Reads the output current of one DAC by means of internal ADC
*
* Parameter : the Index that define the DAC register
*
* Returns  : the value in Micro Ampere.
*
* Note   : if this was the first measure after the chip
*          configuration phase, a calibration will be
*          automatically executed.
*
*/
float TAlpide::ReadDACCurrent(Alpide::TRegister ADac) {

uint16_t theResult = 0;
float theValue;
if(fADCBias == -1) { // needs calibration
CalibrateADC();
}
```

```

    SetTheDacMonitor(ADac);
    usleep(5000);
    SetTheADCCtrlRegister(Alpide::MODE_MANUAL, ...
       Alpide::INP_DACMONI, Alpide::COMP_296uA, Alpide::RAMP_1us);

    fReadoutBoard->SendOpCode ( Alpide::OPCODE_ADCMEASURE,  ...
       fConfig->GetChipId());
    usleep(5000); // Wait for the measurement > of 5 milli sec
    ReadRegister( Alpide::REG_ADC_AVSS, theResult);
    theResult -= (uint16_t)fADCBias;
    theValue =  ( ((float)theResult) * 0.164); // uA scale   first ...
       approximation
    return(theValue);
    }
```

## A.2 ALPIDE's Front-end

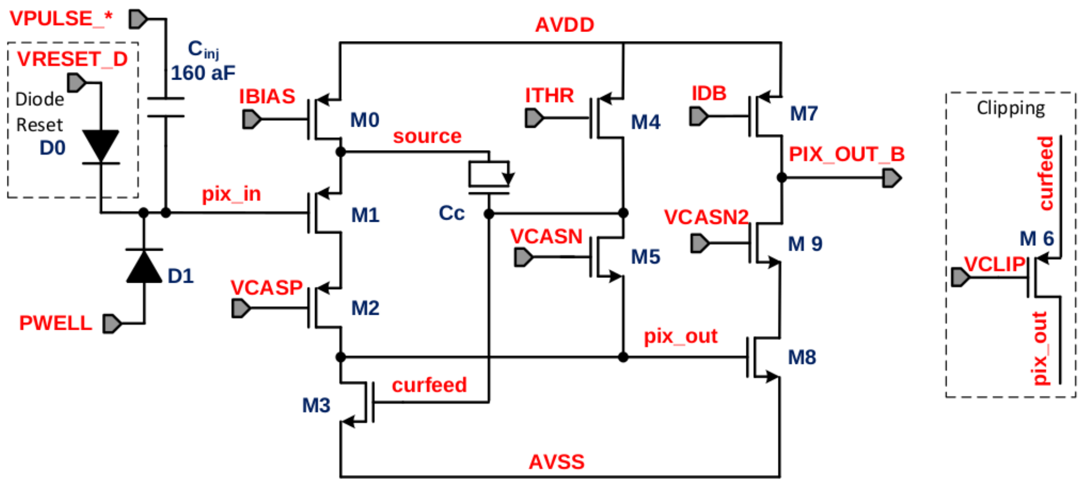


Figure A.1: ALPIDE Front-end scheme[19].

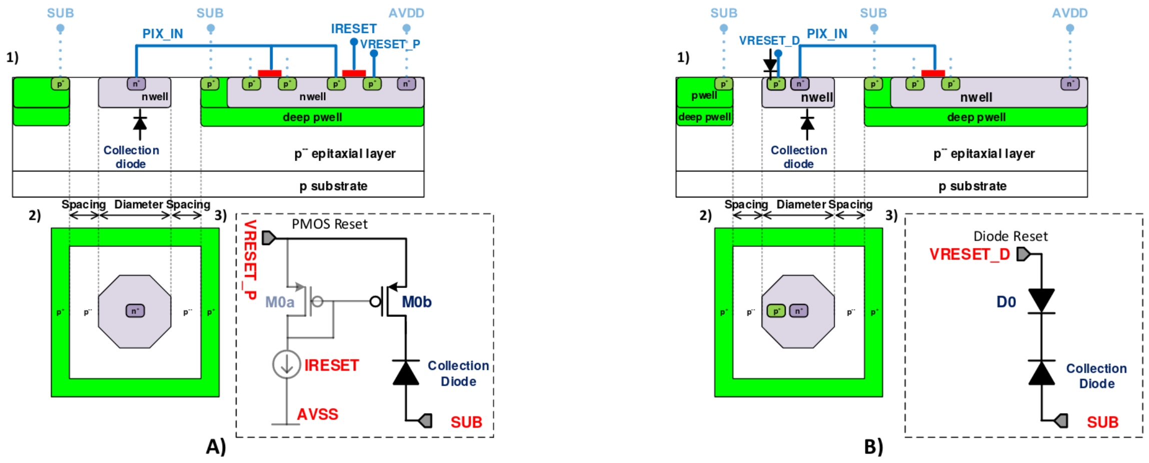


Figure A.2: ALPIDE input stage[31].

# Appendix B

# OPC UA

## B.1 Graphical notations

OPC UA has own graphical notation different from UML language to be able represent functionality and semantic of Information Model and Address Space.

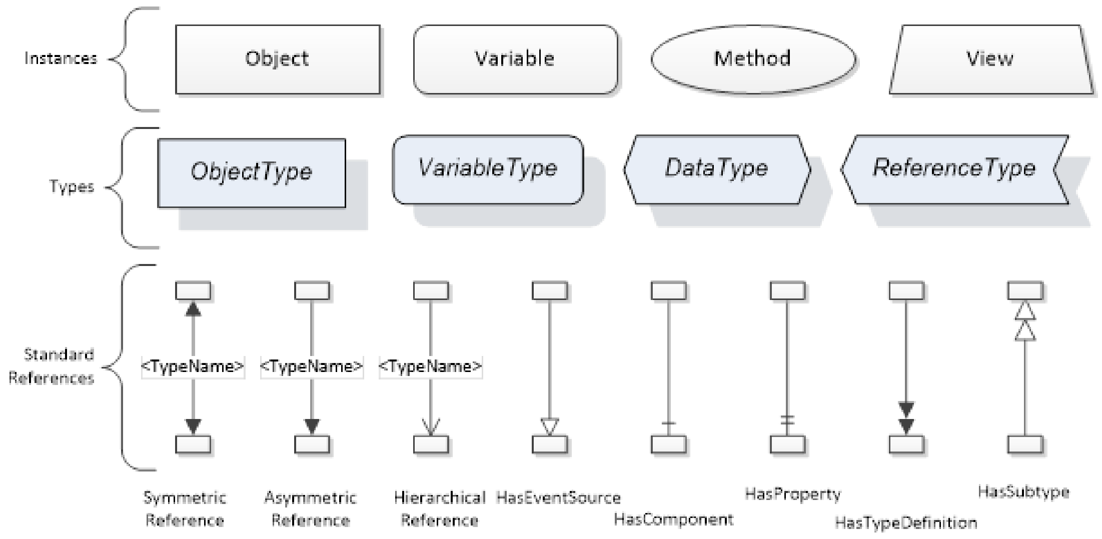


Figure B.1: OPC UA Notations.

## B.2 XML file for pCT Information model

```
<?xml version="1.0" encoding="utf-8"?>
<ModelDesign xmlns:uax="http://opcfoundation.org/UA/2008/02/Types.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:OpcUa="http://opcfoundation.org/UA/"
xmlns:pCT="http://uib.no/UA/pCT/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
TargetNamespace="http://uib.no/UA/pCT/"
TargetXmlNamespace="http://uib.no/UA/pCT/"
TargetVersion="0.01"
TargetPublicationDate="2019-10-01T00:00:00Z"
xmlns="http://opcfoundation.org/UA/ModelDesign.xsd">

<Namespaces>
<Namespace Name="Bergen_pCT" Prefix="Bergen.Ua.pCT" ...
    XmlNamespace="http://uib.no/UA/pCT/Types.xsd" ...
    XmlPrefix="pCT">http://uib.no/UA/pCT/</Namespace>
<Namespace Name="OpcUa" Prefix="Opc.Ua" ...
    InternalPrefix="Opc.Ua.Server" ...
    XmlNamespace="http://opcfoundation.org/UA/2008/02/Types.xsd" ...
    XmlPrefix="OpcUa">http://opcfoundation.org/UA/</Namespace>
</Namespaces>

<!-- ### Object Types ###-->
<!--A generic Subsytem ObjectType-->
<ObjectType SymbolicName="pCT:SubsystemObjectType" ...
    BaseType="OpcUa:BaseObjectType" IsAbstract="false">
<Description>A base type for subsystems</Description>
<Children>
<Property SymbolicName="pCT:IsON" DataType="OpcUa:Boolean" ...
    ModellingRule="Mandatory" />
<Method SymbolicName="pCT:TurnOn" ...
    TypeDefinition="pCT:TurnOnOffMethod" ...
    ModellingRule="Mandatory"></Method>
<Method SymbolicName="pCT:TurnOff" ...
    TypeDefinition="pCT:TurnOnOffMethod" ...
    ModellingRule="Mandatory"></Method>
</Children>
</ObjectType>

<!-- An ADC ObjectType-->
<ObjectType SymbolicName="pCT:ADCObjectType" ...
    BaseType="OpcUa:BaseObjectType" IsAbstract="false">
<Description>ADC object type</Description>
<Children>
<Method SymbolicName="pCT:CalibrateADC" ...
    TypeDefinition="pCT:CalibrateADCMethod" ...
    ModellingRule="Mandatory"></Method>
<Property SymbolicName="pCT:AdcCalibrated" ...
    DataType="OpcUa:Boolean" ModellingRule="Mandatory" />
<Property SymbolicName="pCT:Temperature" DataType="OpcUa:Float" ...
    ModellingRule="Mandatory"/>
<Property SymbolicName="pCT:VDD" DataType="OpcUa:Float" ...
    ModellingRule="Mandatory"/>
```

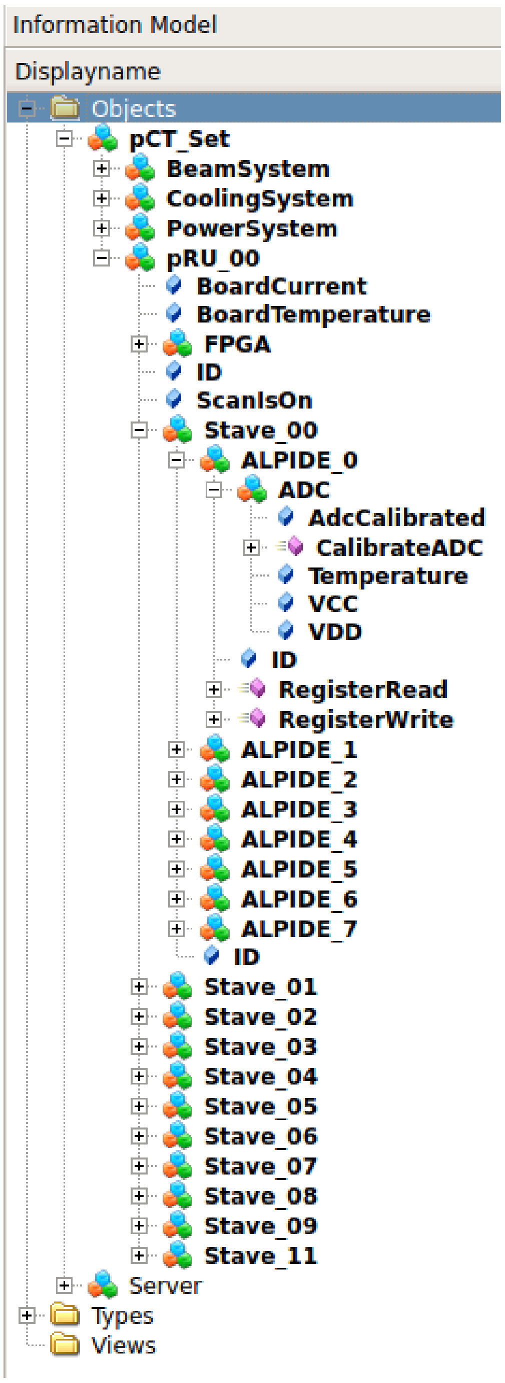
```
<Property SymbolicName="pCT:VCC" DataType="OpcUa:Float" ...
    ModellingRule="Mandatory"/>
</Children>
</ObjectType>

<!-- An ALPIDE ObjectType-->
<ObjectType SymbolicName="pCT:ALPIDEObjectType" ...
    BaseType="OpcUa:BaseObjectType" IsAbstract="false">
<Description>A base type for ALPIDE object</Description>
<Children>
<Property SymbolicName="pCT:ID" DataType="OpcUa:Int32" ...
    ModellingRule="Mandatory" />
<Object SymbolicName="pCT:ADC" TypeDefinition="pCT:ADCObjectType" ...
    ModellingRule="Mandatory" />
<Method SymbolicName="pCT:RegisterWrite" ...
    TypeDefinition="pCT:ALPIDERegisterWriteMethod" ...
    ModellingRule="Mandatory" />
<Method SymbolicName="pCT:RegisterRead" ...
    TypeDefinition="pCT:ALPIDERegisterReadMethod" ...
    ModellingRule="Mandatory" />
</Children>
</ObjectType>

<!-- A stave ObjectType-->
<ObjectType SymbolicName="pCT:StaveObjectType" ...
    BaseType="OpcUa:BaseObjectType" IsAbstract="false">
<Description>A base type for stave object</Description>
<Children>
<Property SymbolicName="pCT:ID" DataType="OpcUa:Int32" ...
    ModellingRule="Mandatory" />
<Object SymbolicName="pCT:ALPIDE_0" ...
    TypeDefinition="pCT:ALPIDEObjectType" ModellingRule="Mandatory" />
<Object SymbolicName="pCT:ALPIDE_1" ...
    TypeDefinition="pCT:ALPIDEObjectType" ModellingRule="Mandatory" />
<Object SymbolicName="pCT:ALPIDE_2" ...
    TypeDefinition="pCT:ALPIDEObjectType" ModellingRule="Mandatory" />
<Object SymbolicName="pCT:ALPIDE_3" ...
    TypeDefinition="pCT:ALPIDEObjectType" ModellingRule="Mandatory" />
<Object SymbolicName="pCT:ALPIDE_4" ...
    TypeDefinition="pCT:ALPIDEObjectType" ModellingRule="Mandatory" />
<Object SymbolicName="pCT:ALPIDE_5" ...
    TypeDefinition="pCT:ALPIDEObjectType" ModellingRule="Mandatory" />
<Object SymbolicName="pCT:ALPIDE_6" ...
    TypeDefinition="pCT:ALPIDEObjectType" ModellingRule="Mandatory" />
<Object SymbolicName="pCT:ALPIDE_7" ...
    TypeDefinition="pCT:ALPIDEObjectType" ModellingRule="Mandatory" />
</Children>
</ObjectType>

<!-- A FPGA ObjectType-->
<ObjectType SymbolicName="pCT:FPGAObjectType" ...
    BaseType="OpcUa:BaseObjectType" IsAbstract="false">
<Description>A base type for FPGA object</Description>
<Children>
<Property SymbolicName="pCT:VCCINT" DataType="OpcUa:Float" ...
    ModellingRule="Mandatory" />
<Property SymbolicName="pCT:VCCAUX" DataType="OpcUa:Float" ...
```

```
    ModellingRule="Mandatory" />
<Property SymbolicName="pCT:JunctionTemperature" ...
    DataType="OpcUa:Float" ModellingRule="Mandatory" />
</Children>
</ObjectType>

<!-- pRU ObjectType-->
<ObjectType SymbolicName="pCT:pRUObjectType" ...
    BaseType="OpcUa:BaseObjectType" IsAbstract="false">
<Description>A base type for pRU object</Description>

<Children>
<Property SymbolicName="pCT:ID" DataType="OpcUa:Int32" ...
    ModellingRule="Mandatory" />
<Property SymbolicName="pCT:ScanIsOn" DataType="OpcUa:Boolean" ...
    ModellingRule="Mandatory" />
<Property SymbolicName="pCT:BoardTemperature" ...
    DataType="OpcUa:Float" ModellingRule="Mandatory" />
<Property SymbolicName="pCT:BoardCurrent" DataType="OpcUa:Float" ...
    ModellingRule="Mandatory" />
<Object SymbolicName="pCT:FPGA" ...
    TypeDefinition="pCT:FPGAObjectType" ModellingRule="Mandatory" />
<Object SymbolicName="pCT:Stave_00" ...
    TypeDefinition="pCT:StaveObjectType" ModellingRule="Mandatory" />
<Object SymbolicName="pCT:Stave_01" ...
    TypeDefinition="pCT:StaveObjectType" ModellingRule="Mandatory" />
<Object SymbolicName="pCT:Stave_02" ...
    TypeDefinition="pCT:StaveObjectType" ModellingRule="Mandatory" />
<Object SymbolicName="pCT:Stave_03" ...
    TypeDefinition="pCT:StaveObjectType" ModellingRule="Mandatory" />
<Object SymbolicName="pCT:Stave_04" ...
    TypeDefinition="pCT:StaveObjectType" ModellingRule="Mandatory" />
<Object SymbolicName="pCT:Stave_05" ...
    TypeDefinition="pCT:StaveObjectType" ModellingRule="Mandatory" />
<Object SymbolicName="pCT:Stave_06" ...
    TypeDefinition="pCT:StaveObjectType" ModellingRule="Mandatory" />
<Object SymbolicName="pCT:Stave_07" ...
    TypeDefinition="pCT:StaveObjectType" ModellingRule="Mandatory" />
<Object SymbolicName="pCT:Stave_08" ...
    TypeDefinition="pCT:StaveObjectType" ModellingRule="Mandatory" />
<Object SymbolicName="pCT:Stave_09" ...
    TypeDefinition="pCT:StaveObjectType" ModellingRule="Mandatory" />
<Object SymbolicName="pCT:Stave_11" ...
    TypeDefinition="pCT:StaveObjectType" ModellingRule="Mandatory" />

</Children>

</ObjectType>

<!-- ### Object Instances ###-->
<!--An Object to collect all pCT related instances at one place by ...
    reference to it-->
<Object SymbolicName="pCT:pCT_Set" ...
    TypeDefinition="OpcUa:BaseObjectType">
<Description>Contains all instances of Bergen pCT</Description>
<References>
<Reference IsInverse="true">
```

```
<ReferenceType>OpcUa:Organizes</ReferenceType>
<TargetId>OpcUa:ObjectsFolder</TargetId>
</Reference>
</References>
</Object>

<Object SymbolicName="pCT:BeamSystem" ...
    TypeDefinition="pCT:SubsystemObjectType">
<Description>Beam system control</Description>
<References>
<Reference IsInverse="true">
<ReferenceType>OpcUa:Organizes</ReferenceType>
<TargetId>pCT:pCT_Set</TargetId>
</Reference>
</References>
<Children>
</Children>
</Object>

<Object SymbolicName="pCT:CoolingSystem" ...
    TypeDefinition="pCT:SubsystemObjectType">
<Description>Cooling system control</Description>
<References>
<Reference IsInverse="true">
<ReferenceType>OpcUa:Organizes</ReferenceType>
<TargetId>pCT:pCT_Set</TargetId>
</Reference>
</References>
<Children>
<Property SymbolicName="pCT:TemperatureIn" DataType="OpcUa:Float"  />
<Property SymbolicName="pCT:TemperatureOut" DataType="OpcUa:Float" ...
     />
<Property SymbolicName="pCT:Flow" DataType="OpcUa:Float"  />
</Children>
</Object>

<Object SymbolicName="pCT:PowerSystem" ...
    TypeDefinition="pCT:SubsystemObjectType">
<Description>Power system control</Description>
<References>
<Reference IsInverse="true">
<ReferenceType>OpcUa:Organizes</ReferenceType>
<TargetId>pCT:pCT_Set</TargetId>
</Reference>
</References>
<Children>
<Property SymbolicName="pCT:Voltage" DataType="OpcUa:Float"  />
<Property SymbolicName="pCT:CurrentLimit" DataType="OpcUa:Float"  />
<Method SymbolicName="pCT:SetVoltage" ...
    TypeDefinition="pCT:SetVoltageCurrentMethod" ...
    ModellingRule="Mandatory" />
<Method SymbolicName="pCT:SetCurrent" ...
    TypeDefinition="pCT:SetVoltageCurrentMethod" ...
    ModellingRule="Mandatory" />
</Children>
</Object>

```

```
<Object SymbolicName="pCT:pRU_00" TypeDefinition="pCT:pRUObjectType">
<Description>An instance of pRU </Description>
<References>
<Reference IsInverse="true">
<ReferenceType>OpcUa:Organizes</ReferenceType>
<TargetId>pCT:pCT_Set</TargetId>
</Reference>
</References>
<Children>

</Children>
</Object>

<!-- ### Methods ###-->
<Method SymbolicName="pCT:TurnOnOffMethod">
<OutputArguments>
<Argument Name="TurnOnOffStatus" DataType="OpcUa:Int32" />
</OutputArguments>
</Method>

<Method SymbolicName="pCT:CalibrateADCMethod">
<OutputArguments>
<Argument Name="CalibrateStatus" DataType="OpcUa:Int32" />
</OutputArguments>
</Method>

<Method SymbolicName="pCT:SetVoltageCurrentMethod">
<InputArguments>
<Argument Name="SetPoint" DataType="OpcUa:Float" />
</InputArguments>
<OutputArguments>
<Argument Name="ReturnStatus" DataType="OpcUa:Int32" />
</OutputArguments>
</Method>

<Method SymbolicName="pCT:ALPIDERegisterReadMethod">
<InputArguments>
<Argument Name="Address" DataType="OpcUa:Int32" />
</InputArguments>
<OutputArguments>
<Argument Name="Value" DataType="OpcUa:Int32" />
</OutputArguments>
</Method>
<Method SymbolicName="pCT:ALPIDERegisterWriteMethod">
<InputArguments>
<Argument Name="Address" DataType="OpcUa:Int32" />
<Argument Name="Value" DataType="OpcUa:Int32" />
</InputArguments>
</Method>
</ModelDesign>
```

## B.3 UaModeler view of the pCT instrument



Information Model

Displayname

- Objects
  - pCT_Set
    - BeamSystem
    - CoolingSystem
    - PowerSystem
    - pRU_00
      - BoardCurrent
      - BoardTemperature
      - FPGA
      - ID
      - ScanIsOn
      - Stave_00
        - ALPIDE_0
          - ADC
            - AdcCalibrated
            - CalibrateADC
            - Temperature
            - VCC
            - VDD
          - ID
          - RegisterRead
          - RegisterWrite
        - ALPIDE_1
        - ALPIDE_2
        - ALPIDE_3
        - ALPIDE_4
        - ALPIDE_5
        - ALPIDE_6
        - ALPIDE_7
        - ID
      - Stave_01
      - Stave_02
      - Stave_03
      - Stave_04
      - Stave_05
      - Stave_06
      - Stave_07
      - Stave_08
      - Stave_09
      - Stave_11
  - Server
- Types
- Views