



Development of a time-stamped Distributed Data Acquisition System for K=600 Spectrometer

by

BERTRAM M. LOSPER

A Thesis submitted in fulfillment of the requirements for the degree

Master of Engineering: Electrical, Electronic and Computer Engineering

in the Faculty of Engineering and the Built Environment

at the Cape Peninsula University of Technology

Supervisor: Dr BBJ Groenewald

Co-supervisor: Dr V Balyan

Bellville:

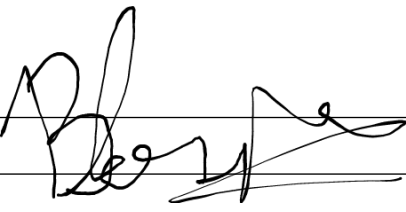
September 2021

CPUT copyright information

The dissertation/thesis may not be published either in part (in scholarly, scientific or technical journals), or as a whole (as a monograph), unless permission has been obtained from the University

DECLARATION

I, Bertram Marinus Losper declare that the contents of this dissertation/thesis represent my own unaided work, and that the dissertation/thesis has not previously been submitted for academic examination towards any qualification. Furthermore, it represents my own opinions and not necessarily those of the Cape Peninsula University of Technology.

		15/03/22
Signed		Date

ABSTRACT

Abstract- This study discusses the development of a new prototype Distributed Data Acquisition System with optional event timestamp that can be used with the K600 spectrometer facility at iThemba LABS. This work is part of an ongoing project known as the Dolosse data acquisition system and will focus on the development of a new data acquisition system with timestamping capabilities on the spectrometer's crate controller single-board computer that is based on the project's architecture. Direct memory access transfer and fragmented readout are proposed to allow efficient data transmission using an open-source stream-processing software platform as an external event buffer. Using this new data acquisition system an event timestamp can be added to the system's read-out pipeline while still maintaining the real-time requirements of the system. To make sure the project is successful, structured programming practices and modular decomposition development methodologies are used to enhance the software quality of project. It is shown that the new data acquisition readout software system fulfills the functional requirements by adding a timestamp to physics event, and the readout of said event data using open source streaming technologies.

ACKNOWLEDGEMENTS

I want to thank:

- My family for their support throughout the duration of the Project
- Dr. BBJ Groenewald and Dr. Vipin Balyan my supervisors at Cape Peninsula University of Technology (CPUT) for their support throughout the whole research project. Without them this wouldn't have been possible.
- Mr. Sehlabaka Qhobosheane my internal supervisor at iThemba LABS for his technical support.
- Dr. Retief Neveling for his valuable insight to the operation of the detector electronics and advice on the k600 DAQ.
- The Software Engineering Division at iThemba LABS for their technical assistance.

The financial assistance of the National Research Foundation towards this research is acknowledged. Opinions expressed in this thesis and the conclusions arrived at, are those of the author, and are not necessarily to be attributed to the National Research Foundation.

DEDICATION

For All people living with special needs or mental disorders.

TABLE OF CONTENTS

DECLARATION.....	ii
ABSTRACT.....	iii
ACKNOWLEDGEMENTS.....	iv
TABLE OF CONTENTS.....	vi
Nomenclature.....	xiv
CHAPTER ONE.....	1
INTRODUCTION.....	1
1.1 Background.....	2
1.2 The Research Problem.....	3
1.3 Research Objectives.....	4
1.4 Limitations of the Study.....	5
1.5 Thesis Organization.....	5
CHAPTER TWO.....	7
LITERATURE REVIEW.....	7
2.1 Background.....	7
2.2 Overview of the Data Acquisition Systems.....	7
2.3 The Composition of a Physics DAQ.....	8
2.3.1 Triggering.....	8
2.3.2 Readout.....	8
2.3.3 Event Building.....	9
2.3.4 Run Control and Event Storage.....	9
2.3.5 Monitoring and Slow control.....	9
2.4 VME Background.....	9
2.5 Data Acquisition Software Systems.....	10
2.6 Overview of Current Spectrometer DAQ.....	11
2.6.1 Current Spectrometer DAQ Hardware Overview.....	12
a. Frontend Electronics.....	12
b. Experiment Host.....	12
c. Analysis Computers.....	12
2.6.2 Current K600 DAQ Software Overview.....	13
2.6.3 Identification of the Deficiencies of the Current Spectrometer DAQ.....	14
2.6.4 Dolosse DAQ.....	15
2.7 Data Acquisition systems used at other Scientific institutions.....	17
2.8 Conclusion.....	19
CHAPTER THREE.....	20
METHODOLOGY.....	20

3.1 DAQ Design Specifications and Requirements.....	20
3.1.1 Design Constrains.....	21
3.2 Proposed Data Acquisition System for Spectrometer.....	22
3.3 Hardware Description.....	24
3.3.1 V792/N Charge to Digital Converter (QDC).....	24
3.3.2 V1495 FPGA.....	24
3.3.3 V785 Analog to Digital Converter (ADC).....	24
3.3.4 V1190A Time to Digital Converter.....	25
3.3.5 V830 Scalar.....	26
3.4 DAQ Software Design.....	26
3.4.1 Overview of the Readout Software Architecture.....	26
a. Kafka Computing Cluster.....	28
b. Kafka Interface Manager on the ROC.....	28
c. User Computer and Kafka Interface.....	29
3.4.2 Communication Topics.....	29
a. Management Topics.....	30
b. Feedback Topics.....	30
c. Data and Event Topics.....	31
d. Control Topics.....	31
e. Topics Structure used for New DAQ.....	32
3.4.3 Software System Design.....	32
a. System Startup.....	35
b. Configuration and Setup of DAQ.....	35
c. Control and Monitoring.....	37
d. Data Readout.....	38
3.4.4 Readout Transfer Methods.....	40
3.4.5 Readout Algorithm Implementation.....	41
3.5 VME Event Builder.....	43
3.5.1 VME Event Builder Architecture.....	43
3.5.2 Event Builder Operation.....	43
a. Event Builder Algorithm.....	44
3.6 Conclusion.....	46
CHAPTER 4.....	47
TIMESTAMP DEVELOPMENT.....	47
4.1 Timestamp HW Design.....	47
4.2 VME Bridge and User FPGA's.....	49
4.2.1 Local Bus.....	49
4.2.2 Timestamp Configuration Registers.....	50
4.2.3 Timestamp Data Registers.....	51
4.3 Timestamp Firmware Development.....	52
4.3.1 V1495_Timestamp Entity.....	52
4.3.2 Timestamp Entity.....	53
4.3.3 Timestamp Multi Event Memory.....	54
4.4 Synchronize and Timestamp Firmware Design.....	54
4.4.1 Timestamp Functional Design.....	55
a. Timestamp Event Counter Process.....	55
b. Timestamp Event Latch.....	56
c. Timestamp Counters.....	57
4.4.2 Timestamp Readout.....	57

a. Single Event Readout.....	57
b. Multi-Event Readout.....	58
4.5 Timestamp Interface Software Development.....	59
4.6 Conclusion.....	61
CHAPTER 5.....	62
System Verification and Experimental Results.....	62
5.1 Configuration and Communication with Other Subsystems.....	62
5.2 Readout Test and Measurements.....	64
5.2.1 Readout Process.....	64
5.2.2 Readout Component Intervals.....	65
a. ADC/QDC Busy Signals.....	65
b. Readout and Data Transmission.....	66
5.2.3 Test Configuration and Measurement.....	67
5.2.4 Measurement Techniques.....	67
a. Readout Interval Timing Techniques.....	67
b. Data Readout Measurements.....	68
c. Measurement Techniques Used.....	69
d. Data Readout Rate.....	70
5.3 Event Builder Evaluation.....	71
5.3.1 Event Rate Measurement Techniques.....	71
5.4 Results Obtained.....	72
5.5 Timestamp Platform Measurements.....	74
5.5.1 Timestamp Event Readout.....	74
a. Timestamp Tests Using Single Event Readout.....	75
b. Timestamp Multi-event Buffer Tests.....	76
c. Metrics of Timestamp Platform.....	78
5.6 Conclusion.....	82
CHAPTER 6.....	83
Conclusion and Discussion of Results.....	83
6.1 Future Work.....	84
BIBLIOGRAPHY.....	85

List of Figures

Figure 1.1: K600 facility at IThemba LABS (iThemba LABS. n.d).....	1
Figure 1.2: Schematic overview of the K600 magnetic spectrometer (Neveling et al., 2008).....	2
Figure 1.3: Outline of Thesis.....	6
Figure 2.1: Block diagram of a DAQ (Emilio, 2013:2).....	7
Figure 2.2: VME Hardware and crate (Emilio, 2013: 96).....	10
Figure 2.3: Example VME bus Software data acquisition system architecture (Emilio, 2013: 7).....	11
Figure 2.4: Current Hardware Architecture of K600 DAQ.....	11
Figure 2.5: MIDAS DAQ readout for the k=600 (Pool et al., 2018).....	13
Figure 2.6: Dolosse High level Architecture (Dolosse n.d.).....	15
Figure 2.7: Diagram of the generic IO32 Field Programmable Gate Array logic. See the text for further explanation (Christian et al., 2014).....	18
Figure 2.8: Digital DAQ for 24 Compton suppressed clover detectors (Palit et al., 2012).....	19

Figure 3.1: High level architecture of the Dolosse DAQ for the k=600 Spectrometer.....	22
Figure 3.2: System Data flow.....	27
Figure 3.3: Architecture of VME Kafka manager.....	28
Figure 3.4: Dolosse VME Message.....	29
Figure 3.5: Functional block diagram of Dolosse Message design.....	32
Figure 3.6: Flow chart of the VME frontend Software system.....	33
Figure 3.7: System startup Flow chart.....	35
Figure 3.8: DAQ configuration flow chart.....	36
Figure 3.9: System monitor and control.....	37
Figure 3.10: Software Application readout flowchart.....	39
Figure 3.11: DMA buffer layout (Concurrent Technologies n.d.).....	40
Figure 3.12: Readout algorithm diagram.....	41
Figure 3.13: Distinct Data sources producing to Kafka.....	42
Figure 3.14: Block diagram depicting how Event builder works.....	43
Figure 3.15: Event builder multi-threaded flow chart.....	44
Figure 3.16: Event construction.....	45
Figure 4.1: Description of V1495 Hardware and I/O registers.....	48
Figure 4.2: Configuration Register for Timestamp platform.....	50
Figure 4.3: Timestamp platform Event timestamp registers.....	51
Figure 4.4: Block diagram of timestamp platform in VME crate.....	55
Figure 4.5: Single timestamp event state diagram.....	58
Figure 4.6: FIFO memory for Multi-event timestamp.....	58
Figure 4.7: Multi-event timestamp state diagram.....	59
Figure 4.8: VME API block diagram.....	60
Figure 5.1: DAQ simple test setup.....	62
Figure 5.2: DAQ Configuration handshaking between ROC and Run Control System.....	63
Figure 5.3: Trigger Timeline and Deadtime (Pool et al, 2018).....	64
Figure 5.4: Trigger timeline when event is above threshold (Pool et al, 2018).....	65
Figure 5.5: QDC busy time output signal (green).....	66
Figure 5.6: Trigger (yellow) with QDC busy time (green).....	66
Figure 5.7: DAQ test evaluation setup.....	67
Figure 5.8: Depiction of k600 DAQ with signal modules.....	68
Figure 5.9: Event builder test setup.....	71
Figure 5.10: Histogram of event rate vs occurrence for event-by-event readout.....	72
Figure 5.11: Histogram of event rate vs occurrence for multi-event readout.....	72
Figure 5.12: Energy loss paddle1 vs paddle 2.....	73
Figure 5.13: QDC RAW values.....	73
Figure 5.14: Pulse height vs time of Flight.....	73
Figure 5.15: Single Periodic Pulsar connected to DAQ.....	75
Figure 5.16: Address space block diagram (CAEN, 2010a).....	77
Figure 5.17: Times stamp test configuration.....	78
Figure 5.18: Plot showing small values readout between consecutive events.....	79
Figure 5.19: Plot showing biggest values between consecutive events.....	79
Figure 5.20: Histogram plot of time difference between consecutive events.....	80
Figure 5.21: Histogram plot of time difference between consecutive events with 70us veto signal.....	81
Figure 22: Histogram plot of time difference between consecutive events using 400us veto signal.....	81

LIST OF TABLES	
Table 2.1: Comparison of MIDAS and Dolosse DAQ	16
Table 5.1: Statistic results with QDC and 7 TDC's in coincidence	69
Table 5.2: QDC and V1495 Timestamp statistics	75
Table 5.3: Consecutive Results recorded	76
Table 5.4: Results of readout data from multi-event buffer on timestamp	77
Table 5.5: Measurements of timestamp	79

APPENDICES	
Appendix A: VME bus specification	88
Appendix B: Time stamp Register Interface	90
Appendix C: V1495_timestamp entity	91
Appendix D: Firmware and Block diagram of Timestamp	92
Appendix E: Firmware and Block diagram of Timestamp	101

BIBLIOGRAPHY	85
---------------------	-----------

GLOSSARY

<u>Abbreviations</u>	<u>Definition/Explanation</u>
ADC	Analog-To-Digital Converter
COIN	Coincidence Logic
CTP	Central Trigger Processor
DAQ	Data Acquisition System
DPP	Digital Pulse Processing
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GUI	Graphical User Interface
HDL	Hardware Design/Description Language
I/O	Input/Output
LABS	Laboratory for Accelerator Based Sciences
LAN	Local Area Network
LVDS	Low-Voltage Differential Signalling
MCU	Microcontroller Unit
NIM	Nuclear Instrumentation Modules
PCI	Peripheral Component Interconnect
PLL	Phase-Locked Loop
QDC	Charge-To-Digital Converter
RAM	Random Access Memory
ROD	Read Out Driver
ROM	Read Only Memory
SBC	Single Board Computer
TDC	Time-To-Digital Converter
TTL	Transistor-Transistor Logic
VMEbus	Versa Modular Eurocard bus

VHDL

**Very High Speed Integrated
Circuit Hardware Description
Language**

Nomenclature

Bridge FPGA - A FPGA used on the V1495 input/out board for the interface to the VMEbus and a 2nd FPGA called the "User" FPGA through a proprietary local bus developed by CAEN.

CAEN – Costruzioni Apparecchiature Elettroniche Nucleari

Dolosse – Is a Scalable and maintainable data acquisition system that can be used in physics

Field Programmable Gate Array (FPGA) — A FPGA is a user configurable semiconductor integrated circuit (IC) that is designed to provide flexibility for a specific application.

Hardware Description Language (HDL) — A text based programming language that describes the behavior of digital logic circuits.

Phase-Locked Loop(PLL) 's – Is a frequency control process configured as multipliers, demodulators, tracking generators, or clock recovery circuits that produce an output signal whose phase is related to the phase of an input signal.

Versa Module Eurocard (VME) — A flexible computing systems data bus architecture that consists of electrical specifications and mechanical specifications for describing the backplane, bus connector board sizes and enclosures.

VHsic Hardware Description Language (VHDL) — Is a commonly hardware description language that is used to write text models that describe a digital logic circuit.

V1495 – is a multipurpose input/output VME 6U board, 1U wide, suitable for various digital applications, which can be directly customised by the User. The management of the module is handled by two programmable FPGAs.

CHAPTER ONE

INTRODUCTION

This dissertation focuses on the development of a real-time, software reconfigurable Data Acquisition System (DAQ) with timestamp for use in High Energy Physics (HEP) experiments at iThemba Laboratory for Accelerator Based Sciences (iThemba LABS) to capture experimental data and archiving it for later analysis. The new DAQ uses open-source streaming technologies that run on a high-performance computing cluster to provide a reliable and robust data readout path. With the latest developments in streaming platforms like Kafka, the throughput of data processing has increased and it has become well suited to use with HEP experiments. The timestamp module for the DAQ will run on a Field Programmable Gate Array (FPGA) platform that will allow the data-streams read out from an external digital DAQ to be merged with the read out data from the K600 DAQ using the generated timestamp value.

iThemba Laboratory for Accelerator Based Sciences (iThemba LABS) is a group of multi-disciplinary research and educational laboratories that are administered by the National Research Foundation (NRF) of South Africa. iThemba LABS has two facilities that are based in Gauteng and Western Cape provinces. At iThemba LABS, the subatomic physics department consist of various active experimental vaults, and one such facility is the K600 magnetic spectrometer as seen in Figure 1.1. This study is going to focus on the development of a new DAQ with timestamp for the K600 magnetic spectrometer.



Figure 1.1: K600 facility at IThemba LABS (iThemba LABS. n.d)

This chapter provides the background information of the K600 magnetic spectrometer and the motivation for the research study for this project. The next section to follow

will be the research problem description. Research objectives followed by the thesis overview are the next and last sections discussed.

1.1 Background

The iThemba LABS K600 magnetic spectrometer is a high resolution detector system and it is only one of two facilities worldwide that can measure inelastically scattered particles and reactions at small angles that includes zero degrees (iThemba LABS n.d.). The K600 was first commissioned at iThemba LABS (previously known as the National Accelerator Centre) in October 1991 and it was designed to be the same as the K=600 that was built at the Indiana University Cyclotron Facility (IUCF) in United states of America (USA), Indiana. It was built to have the same size and average flight path for a particle from target to detector (Neveling et al., 2017). The magnetic spectrometer consists of five active elements namely two dipoles bending magnets, a quadrupole used at the spectrometer's entrance, and two trim coils (K and H) inside the two dipole magnets, see Figure 1.2. By varying the ratio of the two dipole magnets, DIPOLE1 and DIPOLE2 it allows for the diffusion of momentum. The quadrupole magnet located at the entrance is used for vertical focusing at the focal plane, while the two trim coils are used to achieve the final focusing at the focal-plane. The magnetic spectrometer's focal plane detector consists of multiple plastic scintillating paddles which act as the experiment trigger. It also measures the energy lost by a particle that traveled through the drift chambers and the paddles themselves for position determination (Adsley et al., 2017).

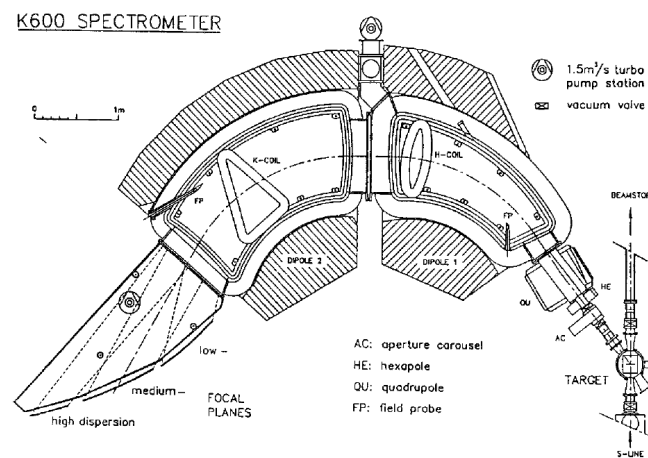


Figure 1.2: Schematic overview of the K600 magnetic spectrometer (Neveling et al., 2008)

The trigger signal from the paddles is used to initiate the data acquisition process by activating the experiment electronics (DAQ) for the magnetic spectrometer which is located in an electronics tower next to the focal plane detector.

The quality and reliability of the computational analysis of inelastically scattered particles and reactions at extreme forward angles within the K600 facility are directly related to the quality of the underlying basic readout nuclear data. Data Acquisition Systems (DAQ's) are generally used to gather data about these nuclear interactions (González et al., 2012). The K600 facility uses a data acquisition system that consists of a Versa Module EuroCard (VME) crate that houses different modules that are used for the readout of data that is used in physics experiments.

As a result of the need for an extensible and scalable DAQ with timestamp that is reliable and still maintains lossless data readout that will ensure accuracy of the computational analysis of nuclear reactions at extreme forward angles within the K600 facility, it has become an urgent need to research and investigate the latest technologies available to use to be able to develop such a system.

The real-time DAQ presented in this dissertation is introduced to address the need for these improvements mentioned in the previous section. A new readout method is introduced that consists of a runtime process running on the DAQ's Single Board Computer (SBC), that moves the data from the detector electronics captured by the readout cards (e.g. ADC) into the memory of the SBC. The read-out data is produced using a streaming platform to further processes in the readout chain (e.g. event processing and data archiving).

1.2 The Research Problem

For a successful nuclear physics experiment to be performed with the K600, there is currently a need to be able to reliably set up the DAQ on the fly if necessary. The current DAQ setups involve time-consuming manual processes of setting up the variables and software compilation, implying lengthy waits for the user before experiments and inefficient resource usage.

The purpose of this study is to investigate using open-source streaming technologies to develop a prototype reconfigurable time-stamped DAQ for the K600 facility at iThemba LABS for robust and reliable data transmission and monitoring and control. The event timestamping functionality for the DAQ will be developed using the CAEN

V1495 Field Programmable Gate Array (FPGA) module (CAEN, 2019). The new timestamp added to the DAQ will allow the users to do more advanced experiments with the K600 in coincidence with the full AFRODITE array, or the full ALBA array, or the combination of LaBr and HPGe arrays that is called GAMKA (iThemba LABS n.d.).

The key requirement in this R&D project and dissertation is the development and implementation of a new functional prototype DAQ with timestamp for the K600 spectrometer. The prototype will act as the foundation for the continuous development and improvements to be made to the magnetic spectrometer's DAQ. The requirements that need to be fulfilled, forms the critical part of the development. These requirements are discussed in the following section.

1.3 Research Objectives

The overall aim of this research is to develop an extensible and scalable DAQ with time-stamp for the Spectrometer facility at iThemba LABS.

The objectives of the research are:

- i **Identify** a DAQ architecture that uses industry-standard open-source streaming tools to fulfill the stringent requirements for data transmission that will reduce the reliance on heavy processing servers/computers. The architecture should also be able to provide a platform for continuous improvement of the K600 DAQ.
- ii **Investigate** the use of FPGA platforms to develop and add an event timestamp to the DAQ readout chain.
- iii **Create** a new simplified readout algorithm and interface that uses a fragmented method for data extraction that can be used with the streaming platform.
- iv **Develop** a DAQ that is capable of reading out single or multiple (hardware buffered) events from the signal modules. The development methods followed should simplify the maintenance of the DAQ software.
- v **Make** provisions for a time-stamp readout in software.
- vi **Replace** the method of DAQ configuration. The new DAQ should allow the user to configure the frontend electronics and not only the DAQ administrator.
- vii **Test** the performance of the newly developed DAQ prototype against stringent physics requirements.

1.4 Limitations of the Study

This study will focus on the first phase of the development of new readout software for the DAQ and use it at triggers with frequencies not higher than 2 kHz due to DAQ trigger Veto inhibit limitation.

The following exclusions will not be discussed:

- DAQ Hardware design and modules that were used because the new readout system is software-based.
- DAQ Trigger system design. This was developed by the physicists at iThemba LABS and does not form part of this study.

1.5 Thesis Organization

- Chapter 1: Introduction.

A summary of the research is outlined in sections 1.1 and 1.2. In section 1.3 the research objectives will be discussed.

- Chapter 2: Literature Review.

This chapter will describe the Literature review of previous studies and describe the current K600 DAQ. This section will discuss open-source DAQ systems and the improvements it has compared to the current DAQ. It will also describe the process of data acquisition, event coincidence, and communication buses and protocols used in nuclear physics Data acquisition.

- Chapter 3: Methodology

In this chapter, the implementation of the Data Acquisition System, communication between different software modules using Dolosse, and the methods used to develop the system are presented.

- Chapter 4: Timestamp Implementation

In this chapter the design and implementation of timestamp on a Field Programmable Gate Array (FPGA's), will be described.

- Chapter 5: Evaluation, Results, and Discussion

This chapter describe the experimental setup protocol, and the evaluation of

the system are discussed. Results from tests using new readout software system with timestamp in a controlled laboratory environment using two pulsars simulating real world triggers are discussed.

- Chapter 6: Conclusion and Future Work

This is the final chapter of this document, and it is based on experimental results and future work is recommended.

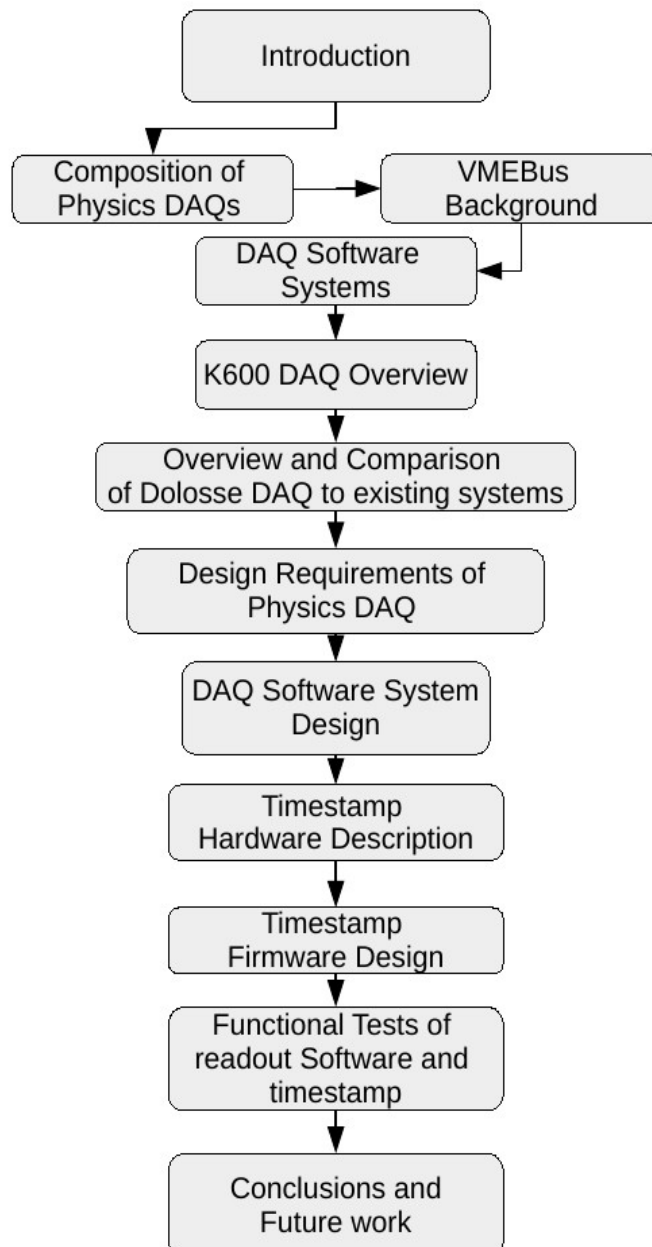


Figure 1.3: Outline of Thesis

CHAPTER TWO

LITERATURE REVIEW

2.1 Background

To successfully develop new software for a Data Acquisition System, it is imperative to understand the background of the different subsystems and modules used for the system to operate optimally. This chapter gives a brief overview of the hardware and software needed to develop such a system and provides reasons why we undertook this development. Furthermore, deficiencies in the operation of the current system are identified and describe how these areas will be improved, and how it will be a benefit to the system. The author will also be looking at previous data acquisition system used and discuss its shortcomings.

2.2 Overview of the Data Acquisition Systems

The process of data acquisition is the reading of information in real-time from one or multiple sensors (González et al., 2012). The read-out data is then processed and stored for later analysis that can be done offline. The process of data acquisition can be viewed as four different activities, namely; acquisition/input (detector), processing (triggering), data conversion (readout, event building, and control), and output (González et al., 2012; Emilio, 2013: 6).

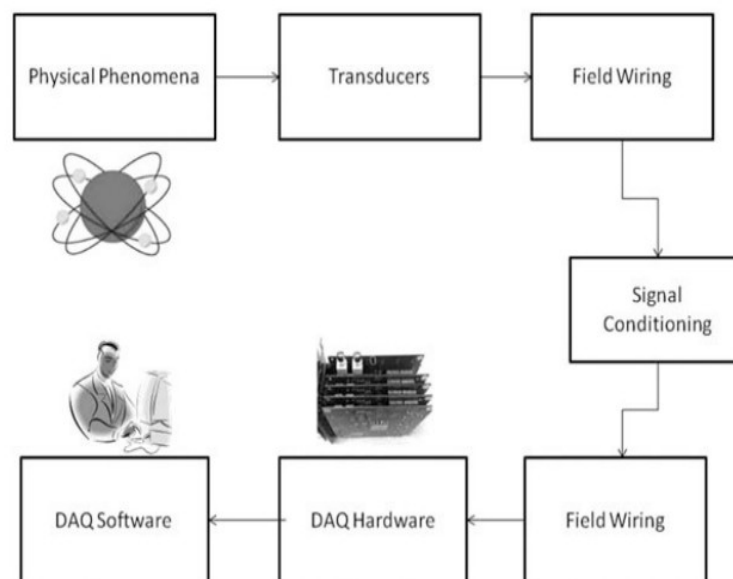


Figure 2.1: Block diagram of a DAQ (Emilio, 2013:2)

A block diagram in Figure 2.1 shows the different elements that a DAQ consists of. Detectors/transducers are used to measure a physical incident which can be anything from small electrical signals, mechanical or radiant energy (Emilio, 2013: 1). The output of the detector/transducer signal is connected to what is known as the frontend electronics, aptly named that way because it is closest to the experiment (González et al., 2012; Wielers, 2015; U.S. Department of Energy Office of Scientific and Technical Information n.d.). The frontend electronics amplifies and shapes the signal pulse (signal conditioning) from the detectors and the signal is then converted to digital format using an Analog-to-Digital Converter (ADC) (González et al., 2012; Emilio, 2013: 6). The digital data is then read out and transferred to a computer for display, storage, and analysis.

2.3 The Composition of a Physics DAQ

A physics DAQ consists of: Triggering, Readout, Event building, Event analysis, event storage, run control, monitoring, and slow control (De Robertis, 2018; Wielers, 2015). In the following sections, what these processes are will be expanded on.

2.3.1 Triggering

In sub-atomic physics, a trigger is a signal that indicates that an interesting event has occurred (De Robertis, 2018). This trigger gets activated when the amplitude of the input signal from the detector is above a certain set threshold, and the trigger also acts as a filter to block out any unwanted signals that shouldn't be digitized for analysis. The trigger system should be able to efficiently react to the event of interest to inform the DAQ that the Frontend electronics (FE) should digitize the input signal(s) (Wielers, 2015).

2.3.2 Readout

When a trigger occurs, the analog signals from the detectors are first signal-conditioned by the pre-amplifiers and then digitized by the FE. The FE normally consists of multiple interconnected modules that each have a specific task, and each of these modules can be seen as a fragment of the event of interest. The conditioned signal(s) of each module is digitized and the digital output is produced to a central

system that processes the digitized data (González et al., 2012).

2.3.3 Event Building

Event formatting or event building is the processing of the raw data that is read out from different modules (González et al., 2012). The event builder is responsible for combining data from the different transducers and/or their abstractions. The event builder is also responsible for extracting the meaningful information from the data that was read out from the transducers for example identifying the module, channel, etc. The built event should be as compact as possible so that data can be easily analyzed.

2.3.4 Run Control and Event Storage

Run control is used to start, stop and configure different runs on the FE electronics during an experiment. Event storage is the process of archiving important information (data and status) from the FE. This data from the FE can be stored in different formats, that can be used later for analysis. Before data is stored, it is required that a data archiving method should be implemented that can verify the integrity of the data (Wielers, 2015).

2.3.5 Monitoring and Slow control

The control software of the DAQ should have the capacity to recover from any instrument component failures and power disruption without losing data (De Robertis, 2018, Wielers, 2015). Data Acquisition software should be able to reconfigure the hardware for a specific experiment and it should be able to handle tasks such as real-time monitoring and control of your application (González et al., 2012; De Robertis, 2018).

2.4 VME Background

The DAQ hardware modules used for the K600 are Versa Module Eurocard (VME) modules. VME is a computing systems architecture that is based on the Eurocard standard (González et al., 2012; Emilio, 2013: 95; Joos, M., 2010). The standard describes the electrical specifications for the data bus and mechanical specifications for the backplane, bus connector board sizes, and enclosures as seen in Figure 2.2.

VME systems use a multi-master, parallel and asynchronous bus (see Appendix A) with a unique arbiter, and the standard is the widest standard used in physics experiments because it provides a backplane that enables fast data transfer, allowing for an increase in the amount of data being transferred (González et al., 2012; Emilio, 2013: 95; Istituto Nazionale di Fisica Nucleare, 2020). The increased data rate permits for an increase of the channel count coming from the front-end electronics (González et al., 2012.; Ahmed, S. N., 2007).

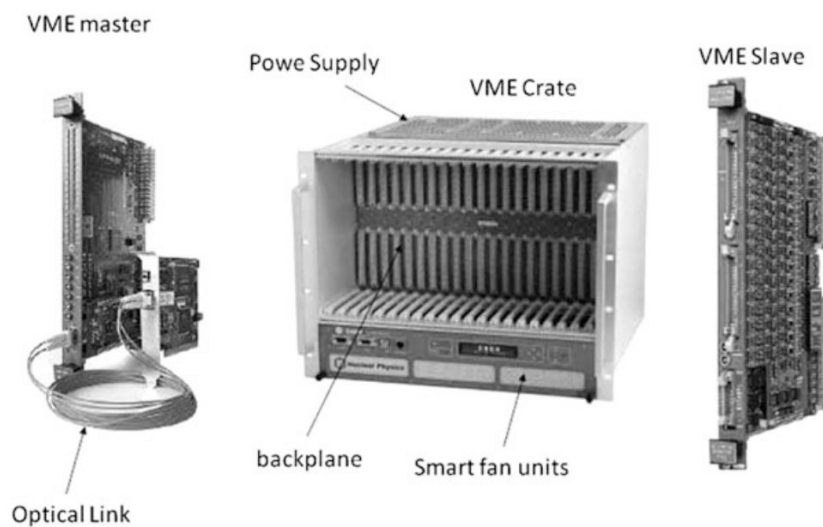


Figure 2.2: VME Hardware and crate (Emilio, 2013: 96)

2.5 Data Acquisition Software Systems

The central component of a DAQ is the data readout software and this software is required for the DAQ FE electronics to work and interface with a Personal Computer (Emilio, 2013: 7; Joos, 2010).

Software for DAQs (seen in Fig. 2.3) is written in a range of different programming languages (e.g. The C programming language, Python, C++, etc.) (Emilio, 2013: 7).

Most manufacturers of instrumentation hardware provide some sort of framework (XIA, CAEN, etc) to write applications and alternatively, several proprietary/open-source DAQ software packages that are available to use (see National Instruments with Labview or MIDAS) (Emilio, 2013:7; Wielers, 2015).

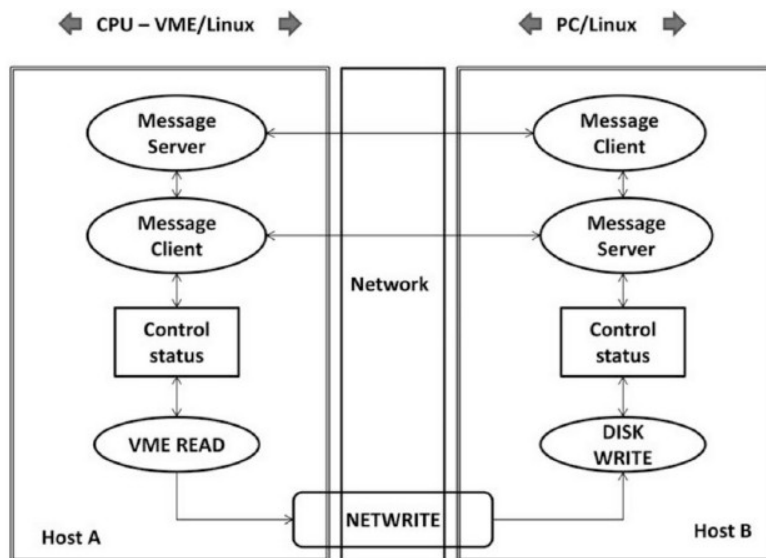


Figure 2.3: Example VME bus Software data acquisition system architecture (Emilio, 2013: 7)

2.6 Overview of Current Spectrometer DAQ

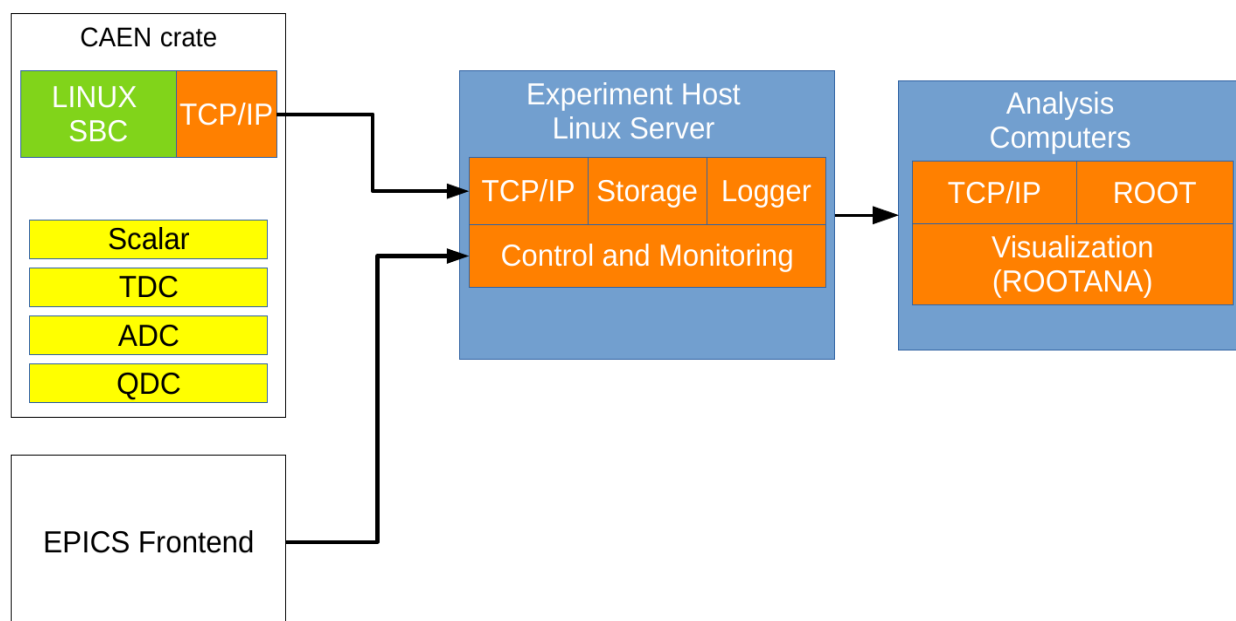


Figure 2.4: Current Hardware Architecture of K600 DAQ

The current spectrometer DAQ Architecture is shown in Figure 2.4. The DAQ frontend electronics consists of a VME crate (CAEN) with several different sensor modules. Each of these sensors is used as a fragment of the event of interest. A Linux Server is used as the experiment host and analysis computers are used as the name suggests to analyze the data from the sensor modules as shown in Figure 2.4.

2.6.1 Current Spectrometer DAQ Hardware Overview

This section will describe the current DAQ that is used for the K600 experiment facility. There will be a discussion on the function of the frontend electronics, the experiment host, and the mentioned data analysis PCs. The k600 DAQ utilises the Trigger and Dataquisition system, where the trigger electronic system “selects” the event of interest that should be aquired by the DAQ electronics (Tlou, 2020:1). The current DAQ software is based on the Maximum Integrated Data Acquisition System (MIDAS) which is used for the readout of event and status data from the sensor modules.

a. Frontend Electronics

The Frontend Electronics (FE) in the DAQ is responsible for the acquisition of event data. The FE used for the K600 is a VME64x crate with multiple interconnected modules which consists of a charge-to-digital converter, several Time-to-Digital (TDC's), Analog-to-Digital Converters (ADC's), and Scalars. As mentioned in the previous section; each module in the crate is used as a fragment of the event of interest. A 24-bit hardware event counter on the QDC is used to count the number of events per run and a maximum of 4M events can be acquired (CAEN, 2010b.).

b. Experiment Host

As the name suggests, the Linux server hosts (remotely) all MIDAS experiment-specific software. It has access to all the hardware required for a specific experiment. It is used to store event data, process event data, and host the monitoring and control web interface (MIDAS n.d). All experiment-related commands are handled and executed by the host, and the experiment host is also responsible for control and monitoring of the FE, and data storage.

c. Analysis Computers

The function of the Analysis Computers that are used is to process and visualise the physics events in real-time. The online data analysis software is implemented using ROOTAna which is a part of the MIDAS framework (MIDAS n.d).

2.6.2 Current K600 DAQ Software Overview

The current software system used for DAQ at iThemba LABS' magnetic spectrometer facility is the MIDAS software that is used with VME Hardware modules. MIDAS is a general-purpose software DAQ, that can be used for small to medium size experiments and can be run on any Linux, VMS, VxWorks, and Windows PC that supports TCP/IP sockets (MIDAS n.d). MIDAS DAQ is written in C/C++ and supports experiments that range from test systems, where a single computer is connected to a CAMAC or VME Crate, in order to experiment with several front-end systems and analysis nodes (MIDAS n.d).

MIDAS DAQ has various drivers written for CAMAC, VME, Fastbus, High Voltage Crates, GBIB, and several different PC plug-in DAQ boards (MIDAS n.d). It also has a built-in control (slow control) system that contains a fast online database (ODB) and a history system (MIDAS n.d).

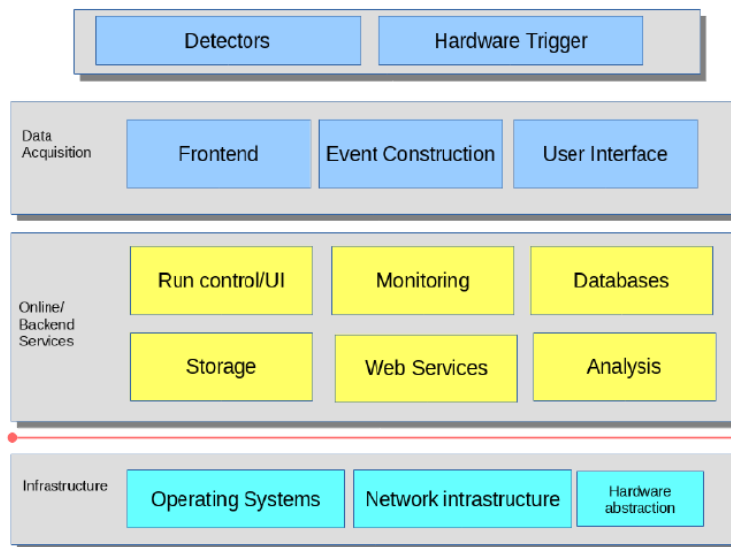


Figure 2.5: MIDAS DAQ readout for the k=600 (Pool et al., 2018)

Figure 2.5 shows a depiction of the current readout for the DAQ used at the spectrometer experimental facility. The block diagram describes the subsystems of the whole readout chain, from the detector/transducer which is connected to the frontend electronics to the data storage. The readout chain works as follows: data is readout from VMEbus modules and the physics event is built on the ROC which then transmits that data via TCP to the online/backend services where data is processed, stored, and sent to the analysis systems where data is visualized in real-time. This is all done through the online service which is the MIDAS ODB. The ODB is used to

communicate between the different MIDAS applications and contains the system configuration run an experiment.

MIDAS is a stable DAQ that is still being actively developed by engineers at PSI, CERN, and Triumf. The reason for moving away from MIDAS at iThemba LABS is to bring all-new DAQ developments in-house.

2.6.3 Identification of the Deficiencies of the Current Spectrometer DAQ

Software DAQs are important for not only retrieving physics event data but also monitoring the status information that helps to determine the health of the overall system. It is therefore important that the acquired data and status information are transmitted and received correctly without errors so that the health and status of the experiment can be monitored accurately. If there are errors present in the read-out data there should be at least a way to warn the user that there could be a presence of such errors to prevent the use of erroneous information.

The components that make up the current DAQ for the spectrometer facility, the data transmission path, and configuration are shown in Figure 2.4 and 2.5 respectively, and are summarized below:

The hardware systems - For the data transmission path the same sensor modules and system computers will be used as with the current DAQ to avoid additional cost issues and to keep the development as simple as possible. But because the project for this thesis is the development of new readout software, this path does not form part of the work in this thesis but will be described in later chapters.

The data readout and control - The data readout path in Figure 2.5 on the other hand uses an algorithm that builds the event of interest on the ROC. This makes the way data is read out an interesting focus area for improvement because the new DAQ adds an external memory buffer and event builder to the readout chain. As part of this investigation into the readout path, the need for time-stamping of events will also be investigated.

Configuration and monitoring - Another area that needs to be investigated is the way the DAQ is configured for experiments. Currently, the configuration of the system for

an experiment can only be done by the DAQ administrator so this would also be an interesting area of investigation. The new DAQ will be able to be configured by the user by sending the configuration data to the ROC using open-source stream-processing software.

2.6.4 Dolosse DAQ

The DAQ architecture that will be investigated to use for the new prototype DAQ is known as Dolosse. Dolosse DAQ is a distributed DAQ architecture whose main objective is to create a new DAQ using open, supported software solutions as a framework (Dolosse n.d.). The Dolosse project is an Open Source project aimed at the development of a physics data acquisition and management system using the Open Source Apache Kafka framework (Kafka n.d.-a). It is the brainchild of Dr. Stanley Paulauskas from Project Science who subsequently collaborated with iThemba LABS and some universities to develop it.

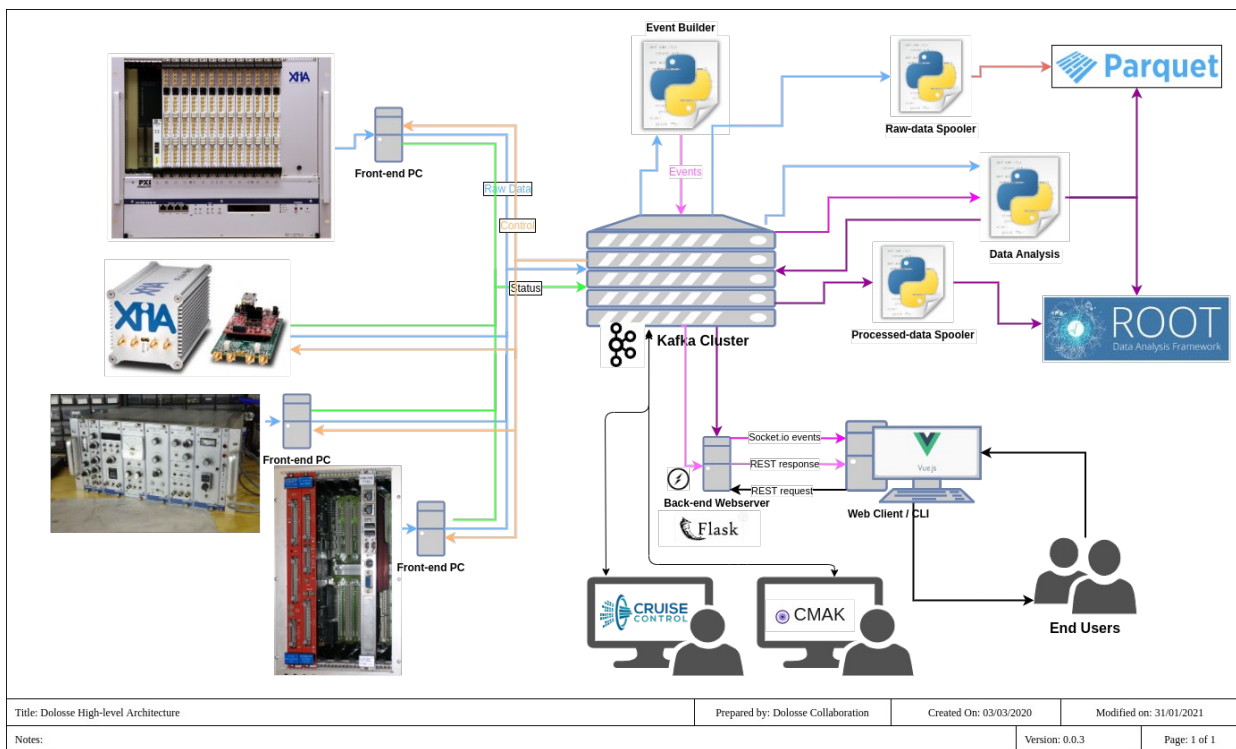


Figure 2.6: Dolosse High level Architecture (Dolosse n.d.)

Dolosse is being developed to address the requirements of a modern data acquisition system, which are to (Dolosse n.d.):

- Provide a scalable DAQ that should be able to integrate analog and digital systems.
- Provide a data visualization framework using modern technologies that do not limit concurrent operations.

- Provide the storage of data that does not depend on a specific data format.

Dolosse DAQ uses central messaging software to transport messages from producers to consumers. Using the messaging system removes the need for an experiment host as data sources and consumers do not depend on the central host for communication between them. The message service that was selected for Dolosse is Apache Kafka which is an open-source distributed messaging and streaming platform that allows one to build real-time streaming applications to manage communication between different data sources (Dolosse n.d.; Kafka n.d.-a). Kafka was selected because it is open source and has a much greater throughput compared to other messaging platforms (Kafka n.d.-b). Through Kafka, the Dolosse DAQ consists of a multi-producer, multi-consumer model which allows real-time data collection and analysis (Dolosse n.d.).

A producer in Kafka is a data source that publishes a message to one or more Kafka topics and a consumer is a piece of software that reads (consumes) the data from Kafka topics. A Kafka topic is a category name given to the messages that is produced to that specific category managed by the Kafka computing cluster (Dolosse n.d.).

As seen in Figure 2.6, Dolosse has data producers which can be previously data sources that can be used, like for example the existing VME FE electronics. Data produced from the FE electronics is consumed from the Kafka cluster by, among others, a piece of software called the event builder. The event builder creates collated events from the data and produces it back into the cluster, ready to be consumed by other data consumers such as those that store the data to disk and those that analyse the data.

Table 2.1: Comparison of current MIDAS DAQ vs Dolosse

	Dolosse	MIDAS
Architecture	Distributed	Centralized/Distributed
Programming Language	Python, C/C++, EPICS, Vue.js, Labview	Python, C/C++, EPICS
Support	IThemba LABS, Project Science	PSI, TRIUMF
Platform	Linux, Windows	Linux, Windows
Software release	Development	Mature, stable
Based on	Open source messaging software	Propriety developed software
Concurrent operations on data	Yes	No
Storage on multiple locations/Brokers	Yes	No

Using Kafka topics in Dolosse provides several advantages over the current MIDAS DAQ as can be seen in Table 2.1. Dolosse using Kafka allow multiple concurrent

consumers to ingest data so that multiple analysis streams or storage applications can work at the same time without worrying about missing data (Dolosse n.d.). Second, data can be duplicated on multiple locations for redundancy. This protects the data in the event if one gets corrupted. Lastly, the topics in Kafka can aggregate data from several data sources (Dolosse n.d.).

Dolosse's biggest advantage is that new user modules can be added to the system without modifying the core data acquisition system i.e. a user will be able to add new modules into the readout pipeline in parallel with other modules without interfering with the system operation. This makes the system simpler to maintain compared to the current MIDAS based DAQ at the K600 facility. Using industry standardized software provides an incredibly solid foundation and new members of the team can get to work sooner using maintained documentation and community support (Dolosse n.d.).

2.7 Data Acquisition systems used at other Scientific institutions

Apart from the DAQs currently operational at iThemba LABS, different accelerator facilities that have implemented DAQs with timestamp system that operate similar to DAQs at iThemba LABS. One such facility is TRIUMF in Canada known as the DRAGON recoil mass separator that exists to study radiative proton and alpha capture reactions (Christian et al., 2014:1). This DAQ is used for the identification of coincidences between the "head" (γ -ray) and "tail" (heavy-ion) detectors, and the original DRAGON DAQ was designed to trigger on single events from either detector systems while also identifying coincidences from hardware gating (Christian et al., 2014:2). The DRAGON DAQ had a trigger logic that was complicated and required a large amount of hardware changes when changing the detector setup (Christian et al., 2014:2). Because of this reconfiguration, the system created a potential for logic problems due to human error or faulty modules to be quite high, and could result in the possibility of wasted beam time or otherwise non-optimal data sets (Christian et al., 2014:2). It is because of these problems that the team at TRIUMF set out to design and upgrade their current DAQ to a new DAQ that would determine coincidence through timestamps using an FPGA (as seen in Figure 2.7) rather than using hardware gating. With the upgraded DAQ the head and tail systems are triggered when an interesting event occurs and read out independently, and coincidences are then determined at the analysis stage (Christian et al., 2014:2).

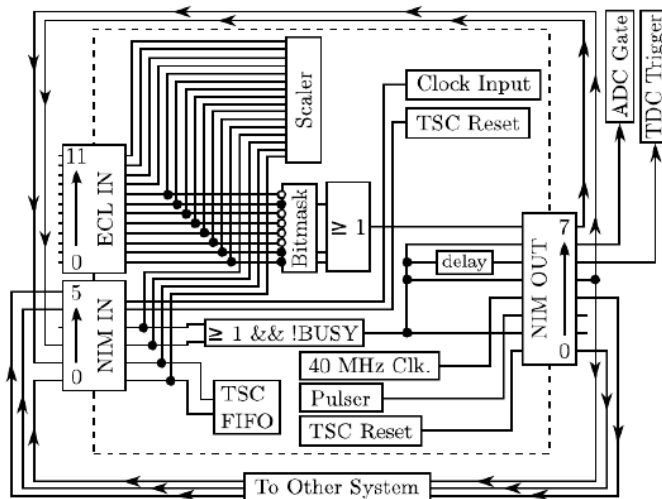


Figure 2.7: Diagram of the generic IO32 Field Programmable Gate Array logic. See the text for further explanation (Christian et al., 2014)

Another facility that was looked at is the Indian National Gamma Array (INGA) at Tata Institute of Fundamental Research where they use a high-speed digital data acquisition system, the TIFR-BARC which uses the Pixie-16 Digital Gamma Finder modules by XIA LLC for their digital data acquisition system (DDAS) for the Compton suppressed clover detector array (Palit et al., 2012:90). The use of the DDAS at TIFR-BARC has improved the rate handling capabilities for the clover array (Palit et al., 2012:92). DDAS provides higher throughput, better energy resolution, and stability compared to Analog systems for the multi-detector Compton suppressed clover array. Previous analog readout systems of INGA wasn't able to keep up with high count rates and was limited due to long term temperature gain drift (Palit et al., 2012:90,92).

The facility uses a DDAS that has six Pixie-16 modules, two LVDS level translator modules, and one controller in one CompactPCI/PXI crate as seen in Figure 2.8. For any trigger on a channel that is validated by the external trigger and not vetoed by the channel veto signal, a time-stamp will be latched and written to the event header (Palit R. et al., 2012:92).

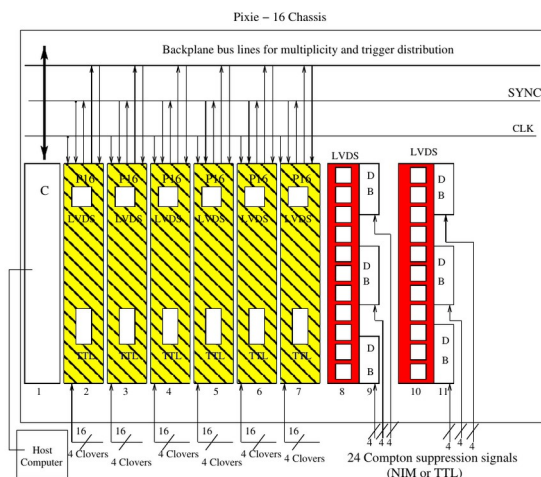


Figure 2.8: Digital DAQ for 24 Compton suppressed clover detectors (Palit et al., 2012)

All the modules in the PXI crate are required to synchronize clocks either internally or externally for the coincidence measurement (Palit R. et al., 2012:91-92).

For the new spectrometer DAQ a different approach is followed to develop the timestamp compared to the DAQs mentioned above. An FPGA module will be added to the frontend electronics and Firmware will be developed that is able to store the timestamp of when an event occurred for a single event readout and it also supports Multi-event readout of the timestamp, where the timestamp is stored in emulated Random Access Memory (RAM) on the FPGA that can be read out via VME.

2.8 Conclusion

This chapter discussed the background of Physics data acquisition systems, the different software DAQs that exist, DAQ that exists at other research facilities, and gave an introduction to the Dolosse DAQ. The chapter also highlighted the differences between Dolosse and the current MIDAS DAQ software used for the magnetic spectrometer and the reason why the facility has decided to go forward with Dolosse.

CHAPTER THREE

METHODOLOGY

This chapter provides detailed guidelines of the development and design process of the new readout software for the k600 data acquisition system.

Before the commencement of the study, requirements were accumulated and refined. The software design for this project makes use of an incremental and iterative approach to design, development, and testing of the system under discussion. The methodology for this project is divided into 5 different sections.

The five main sections of this methodology are:

- Description of the proposed system
- Hardware description
- Data Acquisition Software design
- Hardware Timestamp design
- Prototype, Evaluation and testing process

In the following sections, the researcher will elaborate more on these phases by starting with section 3.1, explaining the design requirements followed by section 3.2 discussing the proposed software system and different microservices. This chapter proceeds with hardware description in section 3.3, followed by the software design in section 3.4, and finishing off with the event builder design in section 3.5.

3.1 DAQ Design Specifications and Requirements

The design of the readout software system for the spectrometer facility stems from the experiences gained from the current K600 MIDAS DAQ and the need to improve the functionality of the DAQ compared to the previous readout system.

The requirement was to develop a reliable new data acquisition software system with a timestamp that could be used at the magnetic spectrometer facility at iThemba LABS, that is able to efficiently transfer physics data from detector to storage.

As mentioned in 2.6.3 the decision was to keep the current VME HW to avoid any additional cost issues. This provides the advantage of keeping the development of new system as simple as possible.

The specifications for the timestamp DAQ were obtained from researching multiple sources. The sources that were consulted were the K600 DAQ and user manual, consulting with the Physicists doing experiments at iThemba LABS, and reading research theses by post-graduate engineering and Physics students. An investigation was launched to identify areas that needed improvement on the current DAQ and using these resources multiple areas were identified which will be discussed in the subsequent sections.

For the design of the new readout software, a top-down design approach was followed for readout. The physics DAQ software will be using the following stepwise approach:

1. Reading out the event fragments from the VMEBbus modules that were generated by the detector source and producing said fragments to the external event buffer, which will be the Kafka messaging system.
2. The event building using a downstream data consumer to the external event buffer, and Real-time analysis of Data. This includes visualization of event data graphically that was read out by the ROC from the signal digitizers.
3. Archiving of event data, to storage that can be a hard disk, tape drive, Kafka messaging system, or removable storage.

3.1.1 Design Constrains

The current version of the HW that is used to do experiments is based on the VME standard as discussed in chapter 2. The scope of the DAQ development doesn't cover HW trigger design as that was developed by the Physicists at IThemba LABS and the HW modules and their configuration will be kept the same as developed by Pool L, et al., (2018).

The current readout system used for the spectrometer buffers and aggregates the event fragments on the ReadOut Computer (ROC) before transmitting the data packets to the experiment host where the event of interest can be accessed for analysis and storage. This process of event construction will be substituted on the new Dolosse ReadOut Software System (ROSS) by buffering binary event fragments in the

Kafka Messaging system and the process of collating and formatting the data on a different workstation. Another area that needs to be scrutinized is the new ROSS should be able to maintain the same readout rate as the current DAQ. The current rate of the DAQ is that it can read out data at a trigger rate of 2kHz thereafter, the time between consecutive events that can be read out starts to increase exponentially. The implementation of the new DAQ readout software should be able to accommodate future improvements in readout rate of the K600 DAQ.

3.2 Proposed Data Acquisition System for Spectrometer

The overall functional block diagram of the proposed DAQ with a timestamp for the Spectrometer facility is shown in Figure 3.1.

The new DAQ for the spectrometer system consists of readout electronics which include the ROC, a Kafka computing cluster, and personal computers.

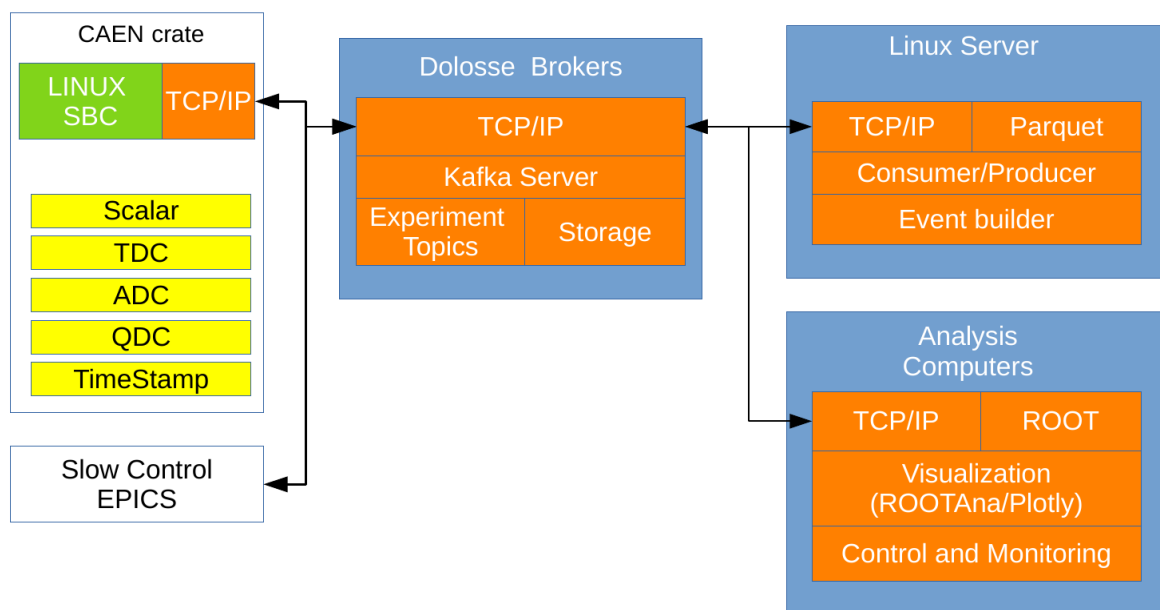


Figure 3.1: High level architecture of the Dolosse DAQ for the k=600 Spectrometer

The new prototype DAQ can be divided into three specialised groups:

1. *Instrumentation/Digitiser Devices* - which is the frontend electronics, and these devices provide services such as the digitization logic. These are VME slave devices and they perform their tasks according to the orchestration of the control and management messages that are being passed to it from the user's frontend computer.

2. *Dolosse Brokers (Kafka Messaging system)* - Multiple Kafka brokers added to the readout chain to form the high-performance computing Kafka messaging system. This messaging system performs the function of event buffer of non-formatted and formatted data and provides the medium of communication between the different software modules. A Kafka broker is defined as a Kafka server (computer) that receives data from data sources and commits it to storage on disk (Narkhede N. et al., 2017: 10).

3. *System Computers* - These devices perform user services, such as the control and monitoring of the instrumentation, real-time data analysis, and the storage of analyzed data. The workstations also function as hosts the new experiment-specific software services such as event-builder, EPICS interface and data archiver software modules. All systems which includes the ROC and Linux workstations are implemented on the PC architecture.

The data acquisition electronics for the spectrometer consists of the existing VME hardware used at the k600 experimental facility. The only change that was introduced to the readout electronics is the addition of a CAEN V1495 FPGA module within the VME crate to implement the event time-stamping for the DAQ.

The new prototype DAQ for the spectrometer is based on a distributed computing system as described by van Steen M. et al., (2016), and it provides a distinct advantage over the current DAQ because there is no experiment host or central processing software, and all the software modules that compromise the DAQ operate as microservices (Nadareishvili, I. et al., 2016: 6). Using the microservice architecture provides the benefit of extending the software system by adding additional software modules without having to modify the existing core readout software. With the use of the messaging system and microservices architecture, no data is lost if a microservice in the readout pipeline crashes due to power loss or losses communication. The module can start receiving from the last offset committed to hard-disk when it is restarted, i.e. start consuming from the last message that was read-out.

3.3 Hardware Description

As mentioned in the previous chapters the k600 DAQ is a VME based data acquisition system and this section will describe the different modules used and what their functions are.

3.3.1 V792/N Charge to Digital Converter (QDC)

The charge-to-digital converter used for the spectrometer is the V792N which is a 16-channel (V792 32-channel) module that integrates the charge on input channels during active level of the GATE input signal (CAEN, 2010b). This is the first fragment to be read out from the DAQ and the signal is then converted into digital format (it takes about $\sim 5.7 \mu\text{s}$ for all channels to be converted) using 12 bit ADC. The converted value from the ADC is stored in the module's Multi Event Buffer (MEB) where these stored event data is accessible via the VME bus (CAEN, 2010b). The QDC used in this data acquisition system is externally triggered with a GATE signal that provides the integration period from the detector electronics. The QDC also serves the function of readout controller, i.e if there is an event on the QDC then all modules will be read out sequentially (CAEN, 2010b; Neveling, R. et al., 2008).

3.3.2 V1495 FPGA

The module that is used for event timestamping is the V1495 FPGA module from CAEN. It is a 6U VME card that provides on its front panel two 32 channel ports of LVDS/ECL/PECL Robertson Nugent P50E-068-P1-SR1-TG multi-pin inputs, one port of 32 channel LVDS outputs, 2 LEMO NIM/TTL that is selectable as input or output which will be used for trigger input and a user programable dual colour LED (CAEN, 2019). The V1495 additionally has three mezzanine slots available, which can be expanded with by either 32-channel LVDS/ECL/PECL inputs, 32-channel LVDS output, 8-channel NIM/TTL LEMO input/output daughter boards. Onboard the VME module there are two FPGAs, one for VME bridge interface and an empty user-programmable Intel Cyclone I EP1C20 FPGA with 20, 000 Logical elements (LE's) (CAEN, 2019).

3.3.3 V785 Analog to Digital Converter (ADC)

The ADC used in the K600 DAQ is the CAEN V785 VME module. It is a 32-input channel

12 bit 62.5 MSPS simultaneous analog-to-digital converter (CAEN, 2012a). The module has an input range of 4V with 1 mV LSB. It has a memory buffer that can store 192k samples per channel which can be divided into up to 1024 separate events (CAEN, 2012a).

This is the third fragment to be read out from the DAQ. Every channel on the V785 is able to detect the maximum value of the analog signal and convert the signal into a digital word (CAEN, 2012a; Neveling, R. et al., 2008).

This digitizer can be triggered internally by any channel exceeding a threshold value, or externally the module can be triggered by an active low or active high GATE signal from the detector (CAEN, 2012a). When the module receives a trigger, all channels that were enabled at configuration are sampled, converted into digital format, and stored in the event output buffer, where the data can be read out through VME bus (CAEN, 2012a). For the magnetic spectrometer, an external active-low GATE trigger is used on the ADC that will initiate data conversion before data is ready to be read.

3.3.4 V1190A Time to Digital Converter

The measurement of time intervals in nearly all of the applications in High Energy Particle physics (HEP), will be a primary task (González et al., 2012). The way time intervals are measured, is based on the start and stop signals. These start and stop signals are usually given by discriminator circuits from the detector (González et al., 2012; De Robertis, 2018). Then, the value between the start and stop signals are digitized which is proportional to the time interval (González et al., 2012).

The last event fragment to be read out is the V1190A Time-To-Digital Converter (TDC). The module contains 4 High-Performance TDC application-specific integrated circuits (ASIC's), that were developed by CERN/ECP-MIC division. Resolution for the V1190A TDC can be set at 100 ps, 200 ps (19 bit dynamics, 104 μ s FSR) or 800 ps (17 bit dynamics, 104 μ s FSR) (CAEN, 2012b).

The data acquisition for the V1190A TDC can be programmed to operate in trigger matching mode with a programmable time window where only the hits belonging to the trigger window will be recorded or in continuous storage mode (CAEN, 2012b; Neveling, R. et al., 2008). The TDC used with the spectrometer is used in trigger matching mode (Neveling, R. et al., 2008).

3.3.5 V830 Scalar

The V830/820 VME Latching Scaler and Multievent Latching Scaler modules are 1-unit wide, 6U high, and are typically used as a pulse counter in particle physics (CAEN, 2007; Neveling, R. et al., 2008). The modules contain 32 independent counting channels, these inputs are accesable via the front panel of the device. Each channel has 32 bit counting depth with a maximum input frequency of 250 Mhz. The value of the counters' can be accessed via VMEbus while the module is busy with acquisition without interfering with the data acquisition process. V830 scaler modules can be triggered by an external NIM/ECL signal, by a VME read request, or by an internal trigger that can be generated by a programmable internal timer (CAEN, 2007).

This module used in the spectrometer by programming the internal timer to generate a 1Hz signal so the counters are read out once a second (Neveling, R. et al., 2008).

3.4 DAQ Software Design

This section discusses the new DAQ software design. The new prototype readout software for the DAQ was developed to run across three computers and a Computing cluster. Communication between different computing systems, software modules, and startup handshaking will be discussed in the following sections.

3.4.1 Overview of the Readout Software Architecture

In this section, the architecture (see Figure 3.2) of the readout software will be discussed. The ReadOut Computer (ROC) that is used, is used to run the Dolosse readout software that communicates with the Apache Kafka messaging system and transfers data from the VMEbus modules. Based on the software architecture shown in Figure 3.2 a single Kafka producer to multiple topic design for the ROC was selected for the new prototype software on the Dolosse runtime application. This was selected because the multi-threaded approach to the readout software and using a single producer is much faster than using multiple instances of a producer as described by Kafka n.d.-b.

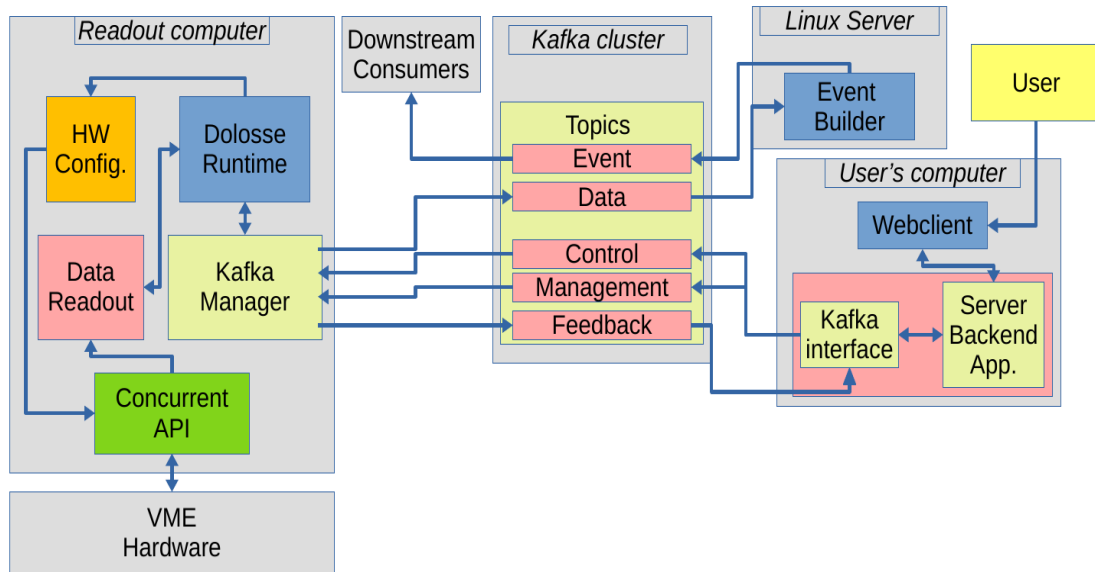


Figure 3.2: System Data flow

The readout software can be classified as having multiple components which are designed to operate independently of each other. As mentioned in section 3.2 the software is based on the microservice architecture and it can be divided into two main categories; there are the data producers, which are also known as data sources and consumers of data for example the event builder microservice running on the Linux server. The readout software was designed so that all software modules in the readout pipeline operate independently around operation on data in specific topics used within Kafka that is described using the experiment's approved number in the form of PRxxx. Multiple topics were selected for communication due to the different services subscribes only to specific messages and ignore the others. Using these multiple topics the throughput of communication between services is improved when compared to using a single topic.

Each software service in the readout pipeline uses the experiment number to configure itself so that it knows which topics in the cluster it should subscribe and produce to as described in Figure 3.2. This experiment number is passed to the modules via a command-line argument when the software applications are started.

a. Kafka Computing Cluster

The Apache Kafka messaging system in the readout chain is used as the external event buffer and communication medium between the different microservices. The system provides a long-term record of all messages that were produced and therefore data in the system can be replayed/retransmitted as needed (Narkhede N., et al., 2017: 4). As seen in Figure 3.2 the communication topics in Kafka range from control and configuration to buffering of the raw data that was produced from the ROC.

b. Kafka Interface Manager on the ROC

To enable the ROC to communicate with the Kafka messaging system a VME producer and VME consumer Class was developed using librdkafka (librdkafka n.d.). This software interface included in Appendix E is developed in C/C++ and is used to interface with the Kafka cluster as shown in Figure 3.3. Since all communication in Dolosse uses JavaScript Object Notation (JSON) messages, a JSON Parser was developed using the JSONCPP library with the VME Consumer Class so the readout application was able to consume and extract the important data that was intended for it (JsonCpp n.d).

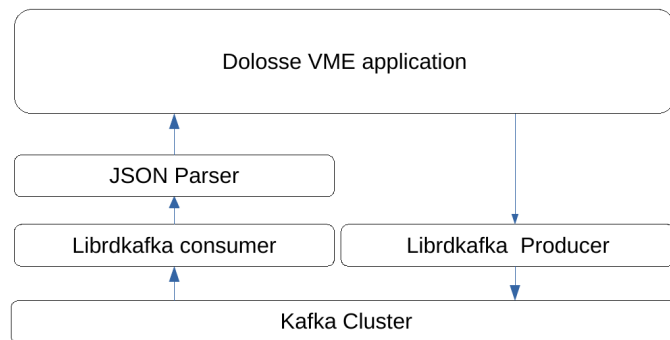


Figure 3.3: Architecture of VME Kafka manager

The new readout software system consists of multiple interconnected modules through the messaging system that each have a specific task. It was designed to be flexible and the software is developed so that is able to adapt over time. By using the C++ consumer and producer classes with external memory buffer new modules could be added to the readout chain and ensure that the software is able to expand with the arrival of new technologies or new requirements.

c. User Computer and Kafka Interface

The software that is used on the User's computer is responsible for monitoring and communicating with the ROC. This software will provide the Graphical User Interface (GUI) for the user to interface with the DAQ and its operation. The web-based GUI or command line application is used to send the commands to the backend server application that maintains a collection of interfaces to various topics in the Kafka messaging system that is used for controlling the experimental run and monitoring the status outputs from the ROC and the event builder application.

3.4.2 Communication Topics

This section will discuss the message structure that was designed for communication between different services through the different topics in the messaging system. The different services that make up the DAQ readout chain communicate using simple instructions that are structured in a straightforward manner using JSON as seen in Figure 3.4.

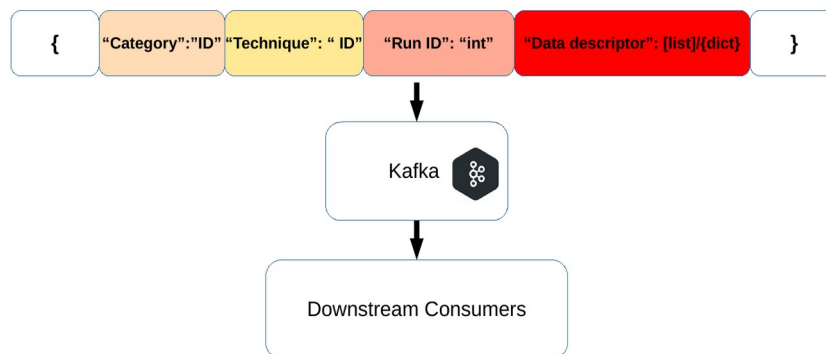


Figure 3.4: Dolosse VME Message

The JSON message that is used to transmit information between different services consists of the following fields :

- Category – describes the type of message that is being produced. It can be measurement, control, management, or feedback.
- Technique – describes what type of system that is being communicated with
- Run ID (optional) – is the current run number being executed in the experiment
- Data – is the interesting information that that is being transmitted. This data

can be of type list or dictionary that contains the non-formatted and formatted data from the ROC.

These JSON messages are used to pass critical information between the different microservices and the data contained in these messages can be the raw transducer data, collated events, feedback, control, or configuration information that is passed through to the different subsystems using Kafka topics. The different categories of topics that are used for communication will be elaborated on in the following sections expanding on each message structure and function within the DAQ.

a. Management Topics

The acquisition software on the ROC will be written in such a way that it should be able to receive configuration data (see illustration 3.1) by polling the Kafka topic that will be used to send configuration data. When the configuration data is available the configuration message is parsed by the software on the ROC to extract the configuration data that is used to set up the different modules used in the VME crate. Previously the DAQ software had to be recompiled by the DAQ administrator with the new configuration for that experiment.

```
{
  "category": "config",
  "data": [
    { "name": "module_name",
      "num.channels": 12,
      "base.address": [201326592],
      "module.number": 1,
      "highwater": 1
    }
  ]
}
```

Illustration 3.1: Example of a configuration Dolosse Message

b. Feedback Topics

Feedback topics or messages are used to convey status and error information to the back-end software regarding the status of Frontend Electronics as seen in illustration 3.2.

```
{
  "category": "feedback",
  "daq": {
    "run_state": true,
    "evt_rate": 0,
    "kB_rate": 0, "events": 0}
}
```

Illustration 3.2: Dolosse Feedback Message

This data message sends the status regarding the experimental run (start, stop or pause), warnings, and any errors that might occur during a run.

c. Data and Event Topics

Equipment (data or event) topics are used to send digitized transducer data to the event builder. The data readout from different modules will be in raw binary format (fragments) described in (CAEN, 2010b; CAEN, 2012a; CAEN, 2012b). These fragments will be aggregated and collated in the event builder and formatted into a JSON string which will be produced back into the Kafka cluster where the events are consumed by the data Analyser for visualization. An example of a built event is shown in illustration 3.3.

```
{
  'category': 'measurement',
  'technique': 'k600',
  'run number': 2021,
  'Event number': 866,
  'Event': [{ 'name': 'module_0', 'data': [0x400144df, 0x8000015f]},
            { 'name': 'module_1', 'data': [0x400144ff, 0x8000015f]}]
}
```

Illustration 3.3: Dolosse Data message example

d. Control Topics

The control topics as the name suggests are responsible for run control of an experiment. Data is read from control topics by the ROC which will then react to the command specified. These commands are used to send the start and stop command for a given experimental run as seen in illustration 3.4.

```
{
  'category': 'control',
  'technique': 'k600',
  'run': { "action": "start", "run_number": 0, "date" : "2017-01-
          12T14:12:06.000-0500" }
}
```

Illustration 3.4: Dolosse Control Message Example

e. Topics Structure used for New DAQ

The new ROSS will be implemented using the topic structure shown in Figure 3.5. The ROC subscribes to management and control topics and it provides equipment and feedback topics, which the Linux workstation running the experiment backend server will subscribe to.

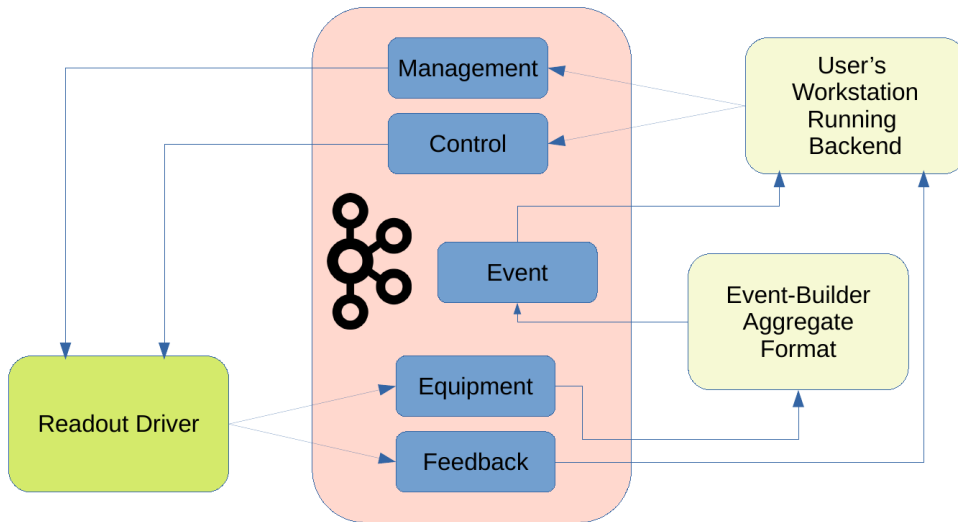


Figure 3.5: Functional block diagram of Dolosse Message design

3.4.3 Software System Design

This section outlines the design of the new software system for the k600 spectrometer. We will be highlighting the execution of the different software modules that runs on the ROC and how each module executes its tasks. It is organized as follows; the section begins with a discussion of the architectural design of the runtime application in the current subsection. The readout design of the K600 spectrometer is illustrated in section 3.4.5 and in section 3.5 showing what role the messaging system and event builder plays in the readout pipeline (included in Appendix E).

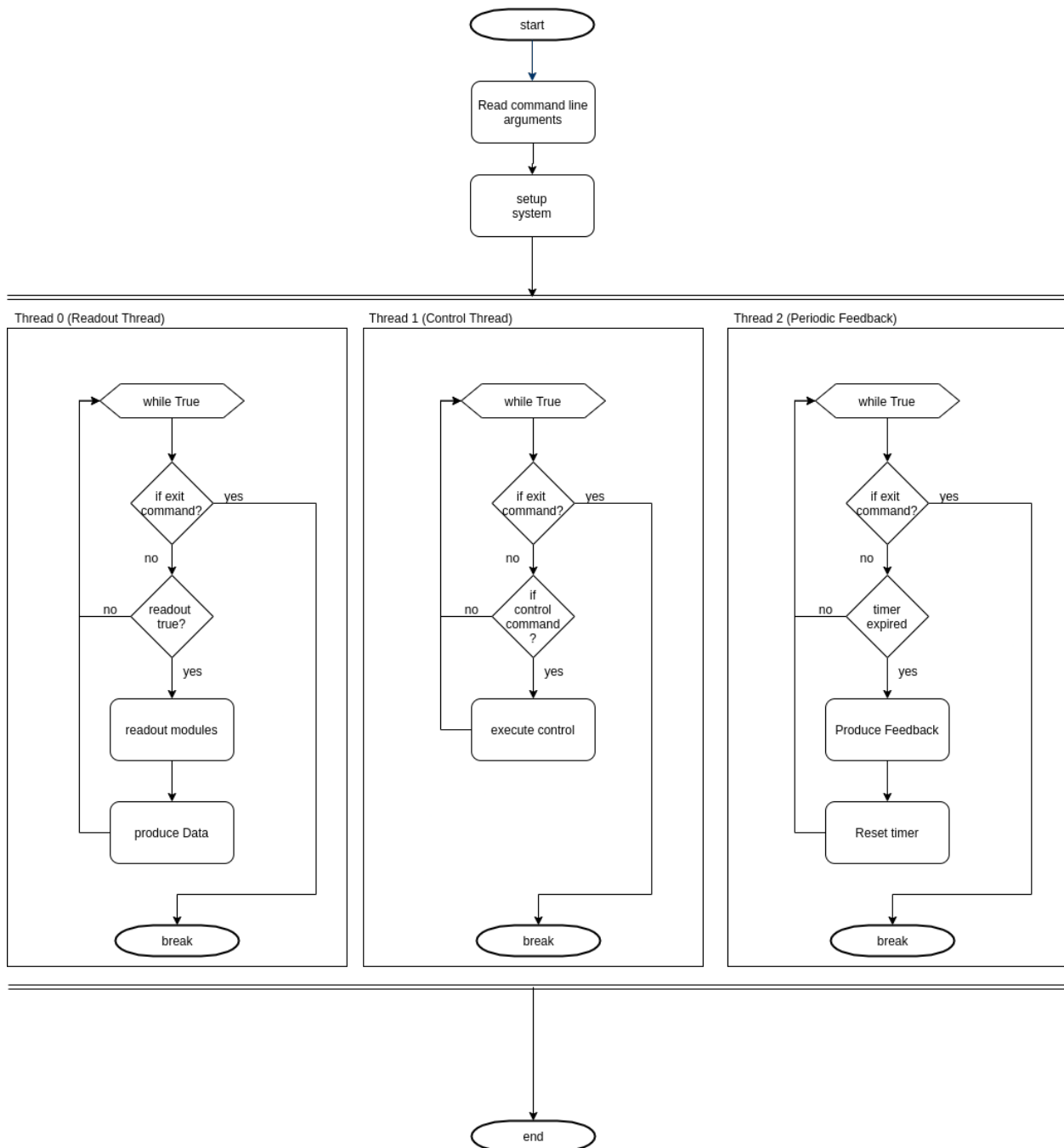


Figure 3.6: Flow chart of the VME frontend Software system

The software takes a multi-threaded approach to simplify the readout. The software consists of three threads and these threads are:

- Readout thread - This thread is used to read out the data from the different signal modules and produce the data from the ROC to the external memory buffer within the Kafka cluster.
- Periodic status thread - This thread is used to periodically produce every second the run information from the ROC to the backend. The feedback message can

be event information (the number of events on each module) or any internal errors in the system or system modules.

- Control thread – The control thread continuously polls the control topic for any run-control messages that have been produced by the backend and executes the commands.

The different threads are employed because the system is designed to operate asynchronously. This asynchronous approach to the ROSS helps so that there is no interference with other processes, like for example the control commands being received don't interfere with the event readout which can severely inhibit the readout rate.

The Dolosse runtime ROSS is written in C/C++ and was extended with the Concurrent Technologies' VME API driver (Concurrent Technologies n.d.) which serves the function of communicating with the Universe II PCI-to-VME converter. The SW also uses librdkafka++ client library to communicate with the kafka cluster (librdkafka n.d.).

For the program to start it requires the following command-line arguments:

Experiment Number: This is the Program Advisory Committee (PAC) approved number for the experiment and it has the naming convention of PRxxx, where xxx is the number given to the experiment.

Path to configuration file: The configuration file contains the Internet Protocol (I.P) address to the Kafka bootstrap server.

When the ROC's Software system is started it consists of the following states:

- startup
- Configuration
- Control
- Data Readout

a. System Startup

System startup is a simple state that is used to reset the runtime application on the ROC, see Figure 3.7.

It transitions from startup where it reads default variables from the configuration file to initialise network and sets up the Kafka consumer and producer topics.

The program is started by passing the experiment name via command line, and the ROSS will setup the Consumer and Producer topics according to the experiment name.

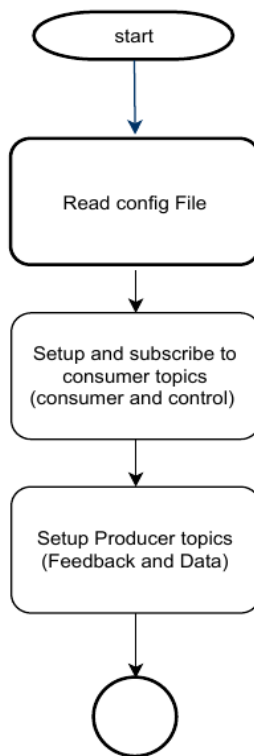


Figure 3.7: System startup Flow chart

b. Configuration and Setup of DAQ

The configuration state allows the ROSS to become ready to receive configuration information from the backend. Once the ROSS is in the configuration state the system polls the control topic within Kafka for 20 seconds for configuration data. During this state the user will use the WebClient together with the backend server application to send a configuration message that contains the details on how the DAQ hardware should be configured for example the number of modules to setup, the number of events to readout, the number of channels used, the base address of the modules and

the name of the module. When the configuration data has been received from the backend the ROSS on the ROC will proceed to configure the signal digitisers to the predetermined setup.

With the new implementation, the system can be considered ready once all Signal modules have been configured and indicated to ROSS by reading the status registers that they are active and ready to start the data collection process. After the configuration process has been completed the ROSS will inform the user by sending a JSON message to the backend to indicate that configuration and HW setup has been completed.

Once the HW modules have been configured, the ROSS will set up the control, readout, and Feedback threads, and the system will enter the main program loop within the control thread which is known as the control and monitoring state.

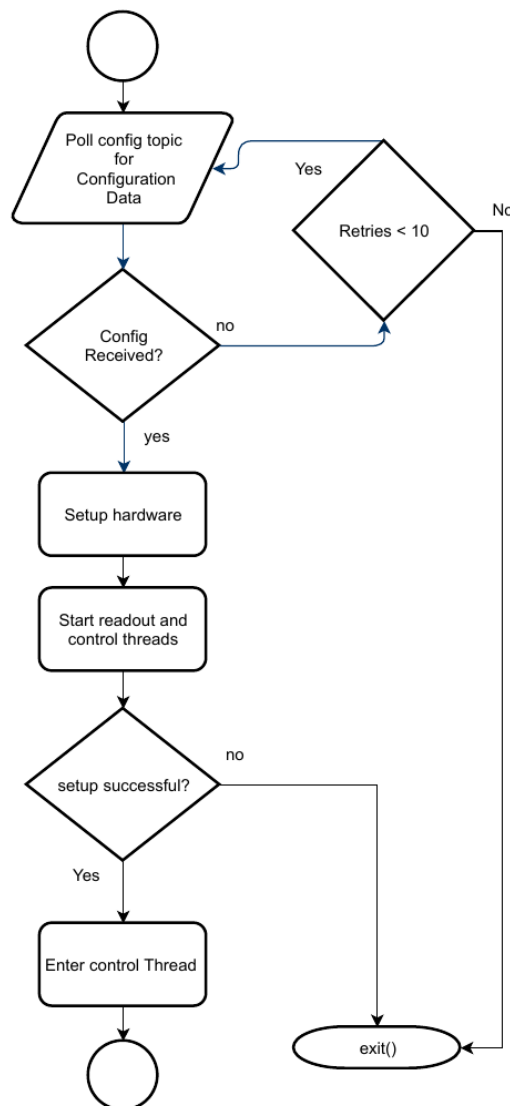


Figure 3.8: DAQ configuration flow chart

The previous DAQ was designed so that only the System Administrator (SA) could set up what modules are to be used for the experiment. This introduced a single point of failure issue because if the SA becomes ill there would be no way that the user of the system could configure the DAQ.

c. Control and Monitoring

Once configuration and setup have been completed the program will enter the control thread and will remain in this loop and repeat until an exit command was received from the Kafka messaging system that was issued by the user.

The control loop's function is to execute all run-related functions of the DAQ as can be seen in Figure 3.9.

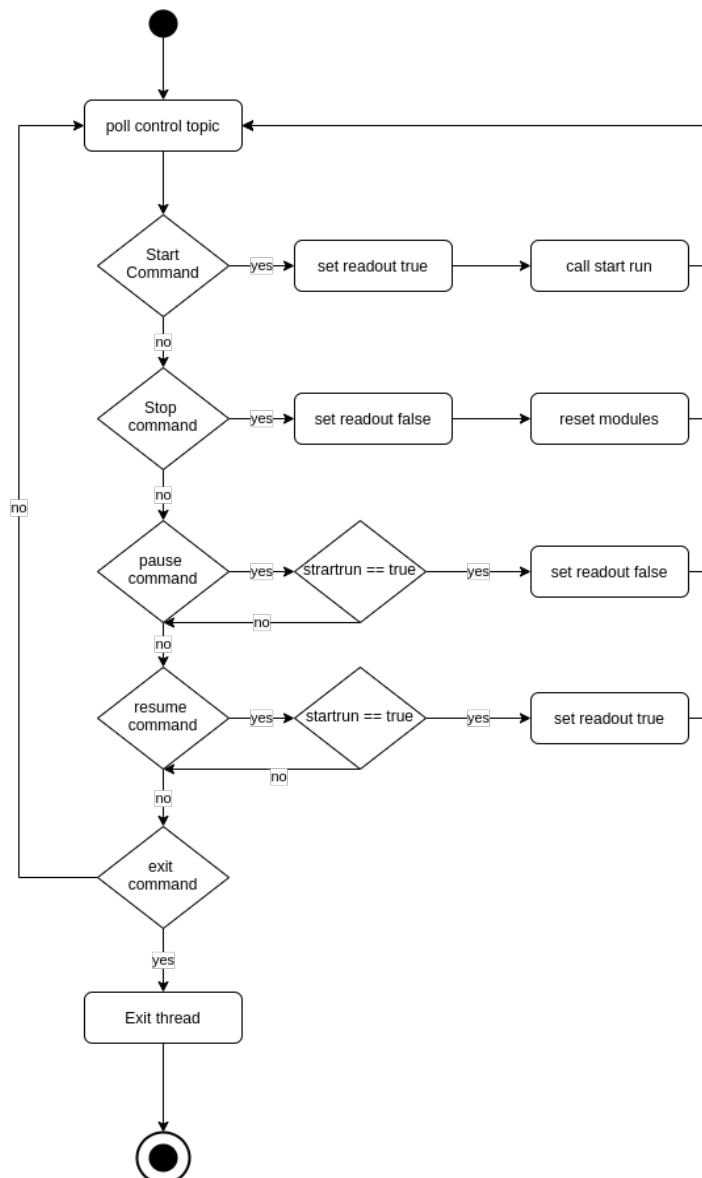


Figure 3.9: System monitor and control

The control loop functions as follows; it polls the experiment control topic for any new control commands that are sent by the user of the DAQ. The different commands that currently can be issued to the control topic are:

- start of run - This command message is used to set the system in an active readout state. The function “start_run” is called to reset the modules’ event counters so that each run will start from event zero. A boolean value called “readout” is set to true. This variable is used to set the event thread to an active state so the system can start reading out data from the event modules.
- pause of run - This command sets the boolean “readout” to false. This will put the system in a state where all readout from the modules is paused.
- stop run - This command calls the function stop_run. The function stop_run sets the value of “readout” to false, and resets the modules’ event counter, and sets the system back into idle state or default state. All acquisition at this point is stopped.
- exit program - This command is used to exit all running threads, and call functions to cleanup dynamically created variables so the program can exit gracefully.

d. Data Readout

Data readout is the process of reading the event data from the signal digitizers. When a run is active and an interesting event occurs on the detector, a trigger is generated which is received by the VME modules (González et al., 2012; De Robertis, 2018). During readout, the ROSS continuously polls the QDC module to see if the DataReady bit is set in the status register which means that there is data in the output buffer that is ready to be read out as can be seen in Figure 3.10. In multi-event readout mode, the ROSS will read the v792N_EVRDYRead register after the status register to determine if the number of events specified at configuration are in the output buffer of the module before readout can occur.

When data is available, the ROSS initiates the readout process to retrieve the data from different modules using Direct Memory Access (DMA) transfer. The received data is then validated to make sure that the ROC received digitized data from the signal modules and not filler words as described in the module’s user manuals (CAEN, 2010b; CAEN, 2012a; CAEN, 2012b). If it is determined that the data is valid, it is then

produced to the Kafka cluster into their respective topics. This type of readout is known as fragmented readout where each module in the crate can be seen as a fragment of the event.

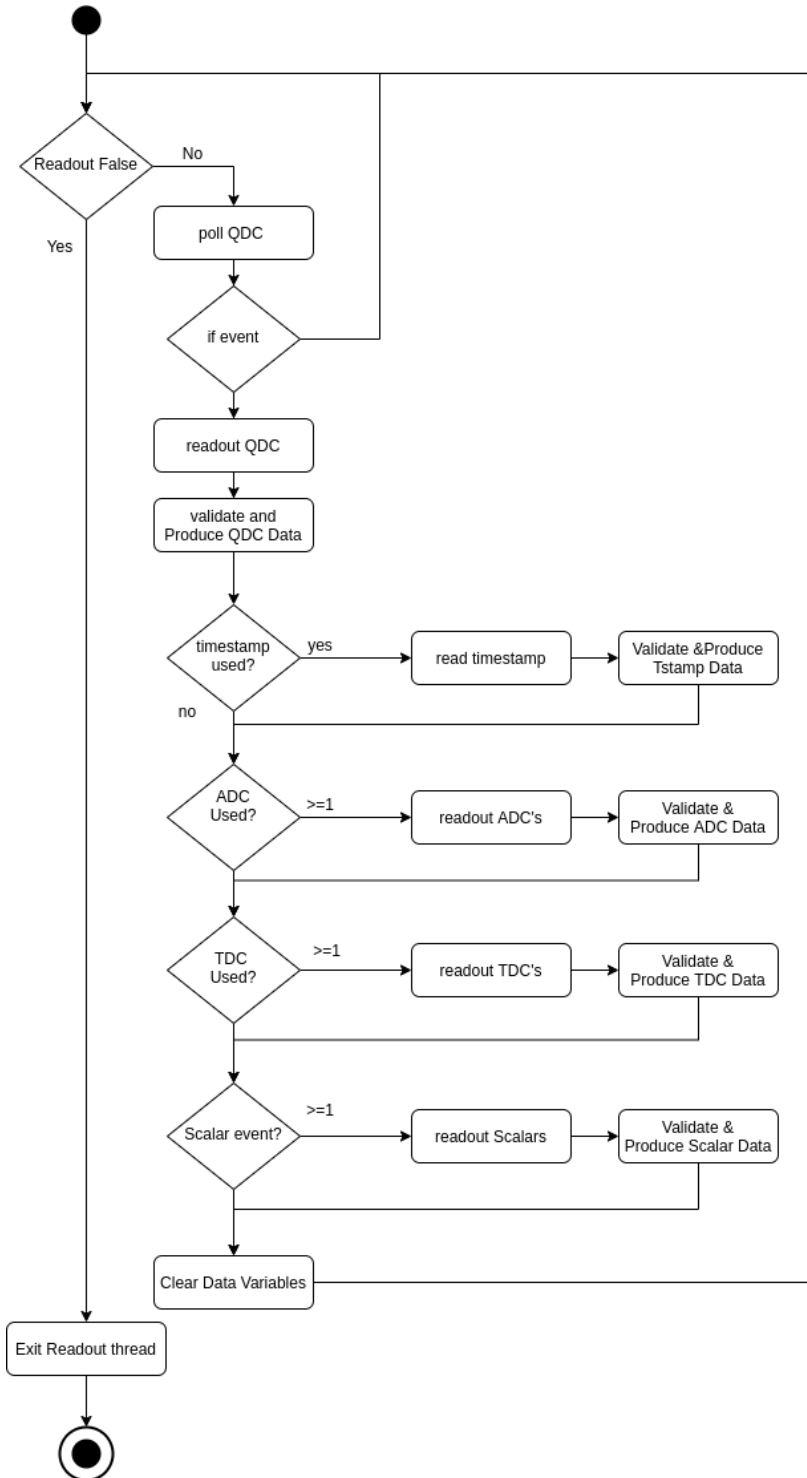


Figure 3.10: Software Application readout flowchart

3.4.4 Readout Transfer Methods

The ROSS is a multi-threaded application as mentioned in section 3.5.2. For the new DAQ different readout methods were evaluated to determine which is the best method to use when reading interesting data from the signal digitisers and producing data to the Kafka Messaging service. A readout thread is utilized to readout data from the digitisers by polling the QDC VMEbus module to read out any available new data as mentioned in section 3.4.3.4 when an experimental run was started.

When a trigger is received by the QDC and the “DataReady” bit in the status register indicates that there is data available, the program will start to initiate data transfer from the different signal digitizers. This trigger is distributed from the detector electronics to all the modules used in the experiment to ensure all the modules that are read out received the same trigger and are on the same event. The data from the digitizers are transferred to the ROC which is designed to support direct memory access (DMA) single event readout or DMA multi-event readout mode mediated by the concurrent vme driver as seen in Figure 3.11 (Concurrent Technologies n.d.).

The readout methods that was investigated are both based on DMA direct transfer 32-bit and 64-bit.

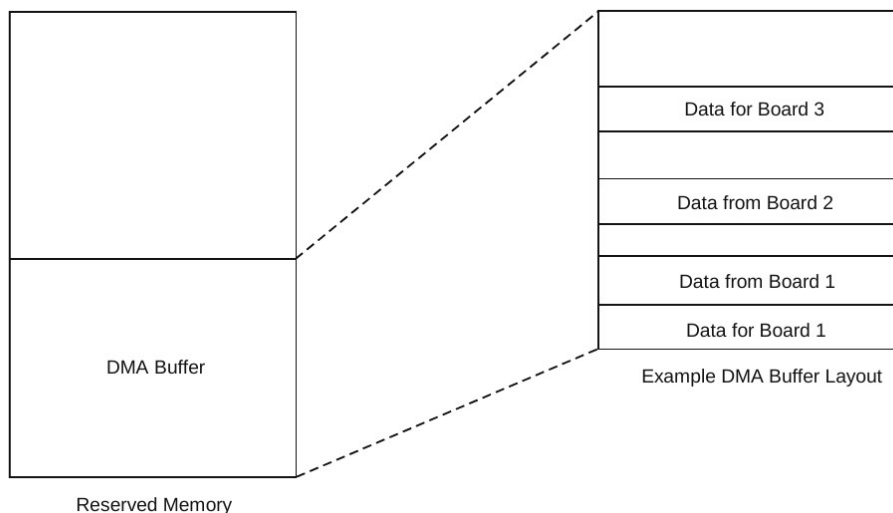


Figure 3.11: DMA buffer layout (Concurrent Technologies n.d.).

When using direct mode operation, a single block of data is transferred from each VMEbus module at a time (IDT, 2010; Concurrent Technologies n.d.). Before the block of data is readout the user sets up the data structure and passes the information about the transfer using the Concurrent technologies API (Concurrent Technologies

n.d.). Once the structure has been setup the data can be transferred from the digitiser boards by calling the “vme_read” function to initiate the transfer by programming the Universe II DMA registers (IDT, 2010).

With direct mode operation, the user/developer has more control over the data transfer, and it is simpler to configure and use with the new readout software system.

3.4.5 Readout Algorithm Implementation

The new DAQ software is designed to read out the data fragments from the digitisers sequentially starting with the QDC followed by the timestamp, ADC, and TDC using 32-bit DMA transfer during an experimental run (IDT, 2010; Concurrent Technologies n.d.). Because of the fragmented data read out, the algorithm used to readout as seen in Figure 3.12., is a modified and simplified version of multi-sensor readout described by González et al., (2012). Using fragmented readout it frees up processing resources on the ROC compared to readout used in the k600 MIDAS DAQ described in 2.6.2 that builds and buffers the events of interest on the ROC.

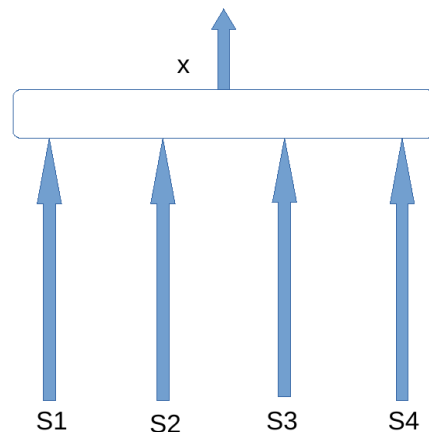


Figure 3.12: Readout algorithm diagram

S1 to S4 in the readout process (see Figure 3.12) represents the VMEbus modules that should be readout as described in section 3.4. The four readings that needs to processed are carried out sequentially in four phases: $F1 = (s1)$, $F2 = (s2)$, $F3 = s3*N$ and $F4 = s4*N$ where x represents the total data read out, F represents the function of reading and producing data from the ROC to Kafka, and N is the number of modules that will be readout. The final output for event readout is $x = \{F1, F2, F3, F4\}$ as described in equation 1.

$$X = \{F1, F2, F3, F4\}$$

Equation 1: Readout Algorithm

When used in multi-event readout the number of events that will be read out from a module is specified when DMA transfer is initiated and the readout algorithm remains the same as shown in equation 1.

Upon completion of a DMA transfer from signal modules to ROC, the raw data is validated against the CAEN binary format that is described in in the VMEbus module's user manuals, by checking that there weren't any errors during readout of the data. After the data from each module was validated it is extracted from the DMA readout buffer, and produced from the ROC to the Kafka messaging system, where the Kafka messaging system functions as First-In-First-Out (FIFO) data buffer for the event fragments using the data topics as shown in Figure 3.13 (CAEN, 2010b; CAEN, 2012a; CAEN, 2012b).

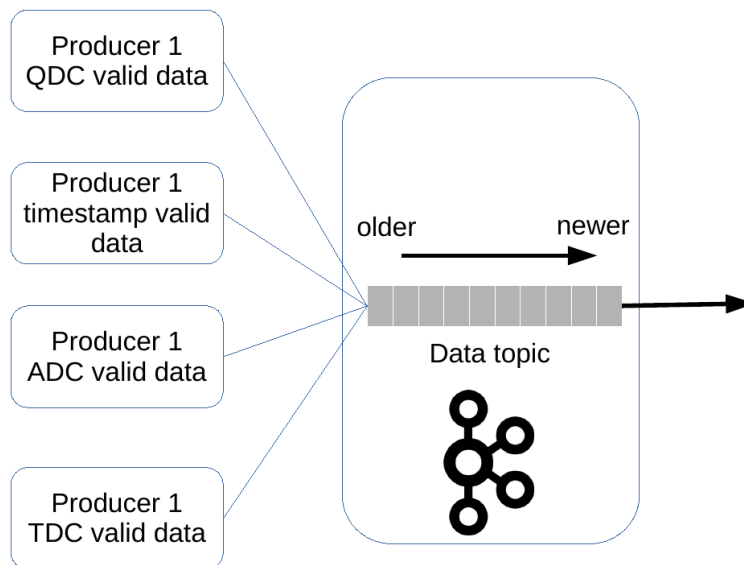


Figure 3.13: Distinct Data sources producing to Kafka

3.5 VME Event Builder

3.5.1 VME Event Builder Architecture

The event builder is a software service that is written in Python which is used to aggregate and collate interesting events from the event fragments that were produced from the ROC. The newly developed event builder is designed so that it is able to process a single event readout from the module or buffered (multi) events.

The event builder is the next essential step in the readout process, it is downstream of the messaging system and is responsible for creating the event structure as shown in Figure 3.14.

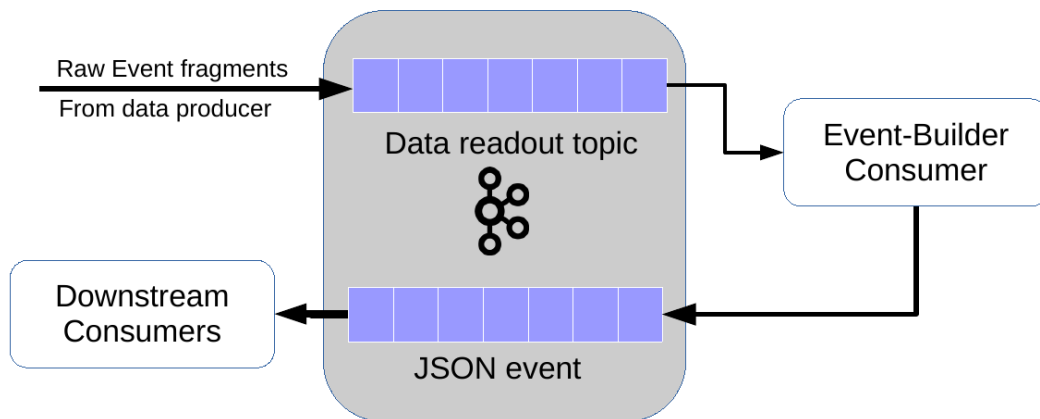


Figure 3.14: Block diagram depicting how Event builder works

3.5.2 Event Builder Operation

The new VME event builder uses the same multithreaded design approach as the Dolosse runtime application on the ROC as seen in Figure 3.15.

The software for the event-builder consists of three threads viz., the event construction, control and feedback threads as described below:

- Event thread - This thread polls the event topic(s) in the cluster for any event data and reads out the different event topic(s) to create an interesting event.
- Control thread - This thread polls the topic for any run-control-related data that has been produced and executes the commands.
- Periodic run status thread - this thread is used to periodically produce status feedback information regarding the number of events and event rate back into the cluster where it can be accessed by the backend.

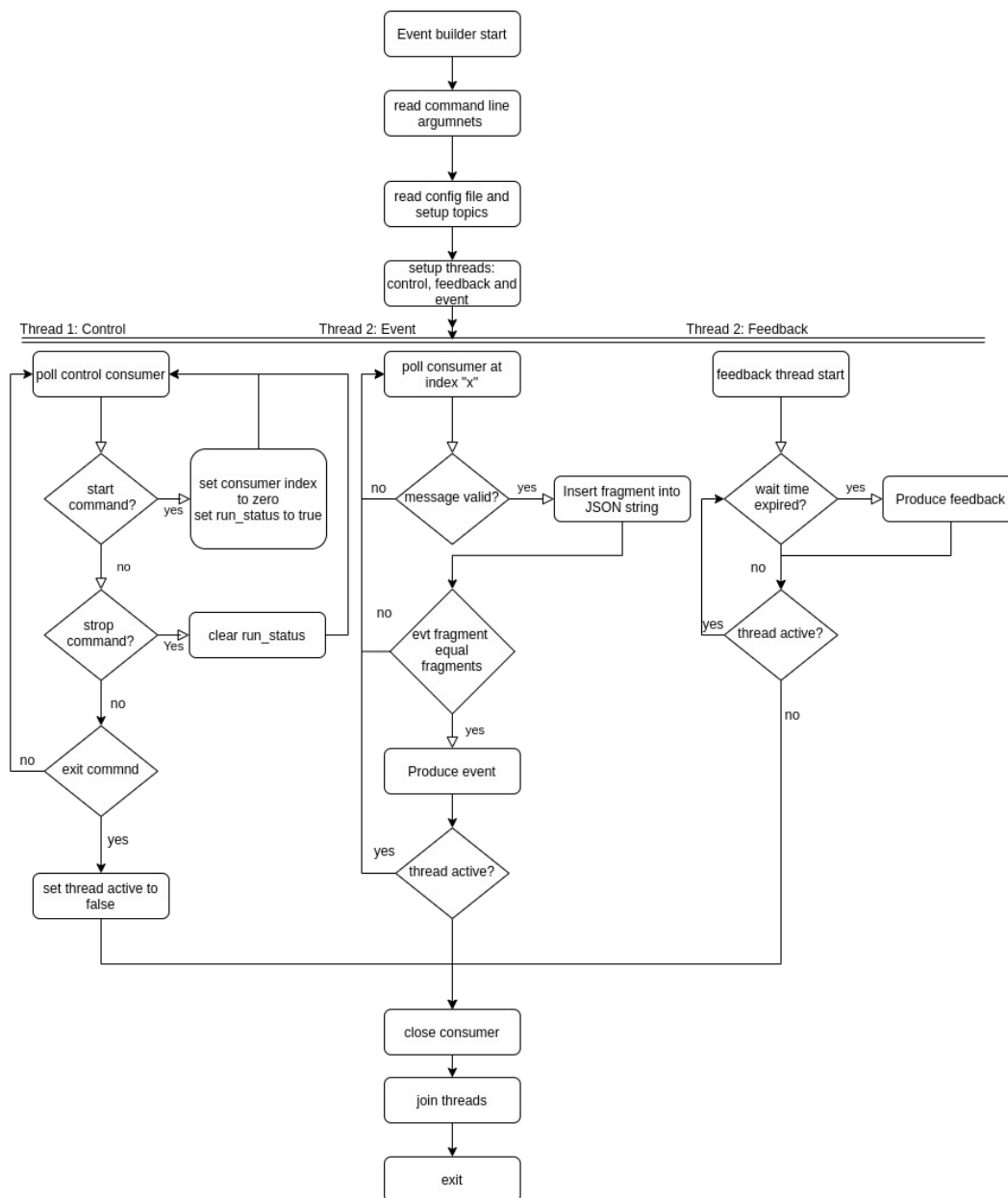


Figure 3.15: Event builder multi-threaded flow chart

a. Event Builder Algorithm

Though the event fragments that were buffered in the Kafka topic are usable, it would be very difficult for the user to use the data in this binary format. To assist the user with analysis the event builder is used to consume the binary CAEN data from the messaging system, extract the event number from each data fragment and store this

event in a python dictionary with the event number as key as seen in Figure 3.16 and illustration 3.6 (Real Python n.d.).

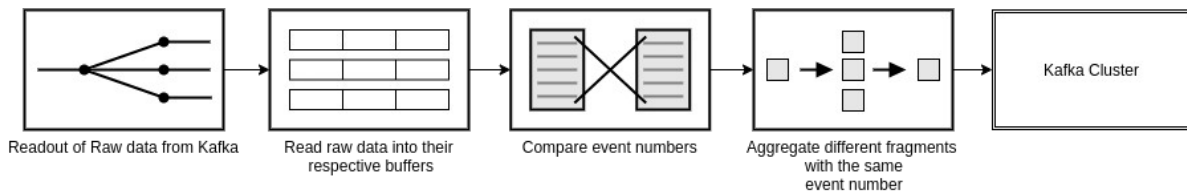


Figure 3.16: Event construction

The data in the dictionaries are used to compare the event numbers of the different event fragments from the different modules. When there's a match between event numbers those fragments are collated and reformatted into a usable JSON string as an example is shown in illustration 3.5. This structured event is then produced back into the cluster and buffered using a different topic where it can be accessed by any software module that consumes from that topic as for example the analysis software.

```

{
  'category': 'measurement',
  'technique': 'k600',
  'run number': 1,
  'Event number': 209,
  'Event': [
    {'name': 'qdc_0',
     'data': ['0xfa001000', '0xf8004952', '0xf810406e', '0xf8024a39',
              '0xf8124083', '0xf8044990', '0xf814406e', '0xf80649ad',
              '0xf8164076', '0xf8084065', '0xf818406c', '0xf80a4065',
              '0xf81a4089', '0xf80c4079', '0xf81c4084', '0xf80e406e',
              '0xf81e408d', '0xfc0000c7']},
    {'name': 't_stamp', 'data': ['0xc8', '0x24e4ccc', '0xa']},
    {'name': 'tdc_0', 'data': ['0xf8000000', '0x400018ff', '0xc9599', '0x49cd0',
                              '0x8000009f']}]
}
  
```

Illustration 3.5: Built event from Event-Builder

The event building algorithm (see illustration 3.6) was developed to determine coincidence and align the event fragments from the different modules. It was developed this way so that it is able to aid in recovery if there was event misalignment between modules during the readout process, for example if TDC events read is greater than the number of events for the QDC which was observed during testing.

```

IF data available = 'True' THEN
  IF qdc_used = 'True' THEN
    CALL add_qdc buffer (event data)
    FOR number of events read
      extract event number from data and use as dictionary key
      create dictionary using event number and data
    ENDFOR
  ENDIF
  IF adc_used = 'True'
    CALL add_adc buffer (event data)
    create dictionary with new module number
    FOR number of events in list
      IF Trailer found
        extract event number from data and use as dictionary key
        create dictionary using event number and data
      ELIF Module ID found
        create new dictionary with module number
      ENDIF
    ENDFOR
  ENDIF
  IF tdc_used = 'True'
    CALL add_tdc buffer (event data)
    FOR number of events in list
      IF Trailer found
        extract event number from data and use as dictionary key
        create dictionary using event number and data
      ELIF Module ID found
        create new dictionary with module number and data
      ENDIF
    ENDFOR
  ENDIF

```

Illustration 3.6: Event builder Algorithm

3.6 Conclusion

This chapter discussed the current design of the Dolosse DAQ. The different readout methods with the design of the new readout chain was illustrated and the incremental design process followed, was showed.

CHAPTER 4

TIMESTAMP DEVELOPMENT

This chapter discusses how the CAEN V1495 FPGA module is used as the timestamping platform and how it is interconnected to the rest of the DAQ. The timestamp platform is used to mark each interesting event with a unique timestamp. With the introduction of this timestamp, the events readout from the K600 DAQ *can* be used to merge data streams when the k600 system is used in coincidence with another DAQ. In the first section of this chapter, the FPGA module and its expansion boards will be discussed, the second will focus on the proprietary bus by CAEN and the final sections contain the interface software and Firmware design of the timestamp.

An appropriate design of the timestamp platform will provide an event timestamp every time a trigger is received from the detector electronics and is able to provide a synchronous signal that can be used to time synchronize with a external DAQ system.

4.1 Timestamp HW Design

The VME 6U V1495 board which is produced by nuclear instrumentation manufacturer CAEN is a general-purpose user-programmable input/output (I/O) board (CAEN, 2019). The board has four digital I/O sections which can be seen in Figure 4.1. These sections are input section Ports A and B, output section Port C, input or output section Port G, and the expansion section Ports D, E, and F. The three expansion interface sections can be expanded by adding up to three different mezzanine boards, which come in 4 different types as shown in Figure 4.1. The first two LVDS/ECL ports A and B are fixed 32 channel input channels and Port C is a fixed LVDS/ECL 32 channel output port. The two channels of Port G, G0, and G1 are two programable bidirectional channels which is also NIM/TTL level user selectable.

The mezzanine expansion boards can be mounted on the interface Ports F, E, and D. The mezzanine expansion boards includes an 8-channel 16-bit resolution analog output board (A395E), 32-channel LVDS/ECL/PECL (A395a input, A395b output, and A395c output) and a 8-channel bidirectional NIM/TTL (A395D) board.

The configuration selected for the timestamp platform was to install one A395D board used as 8-channel TTL output and one A395D board used as an 8-channel TTL input. Information detailing the V1495 module and its expansion boards can be found in the user manual (CAEN, 2019).

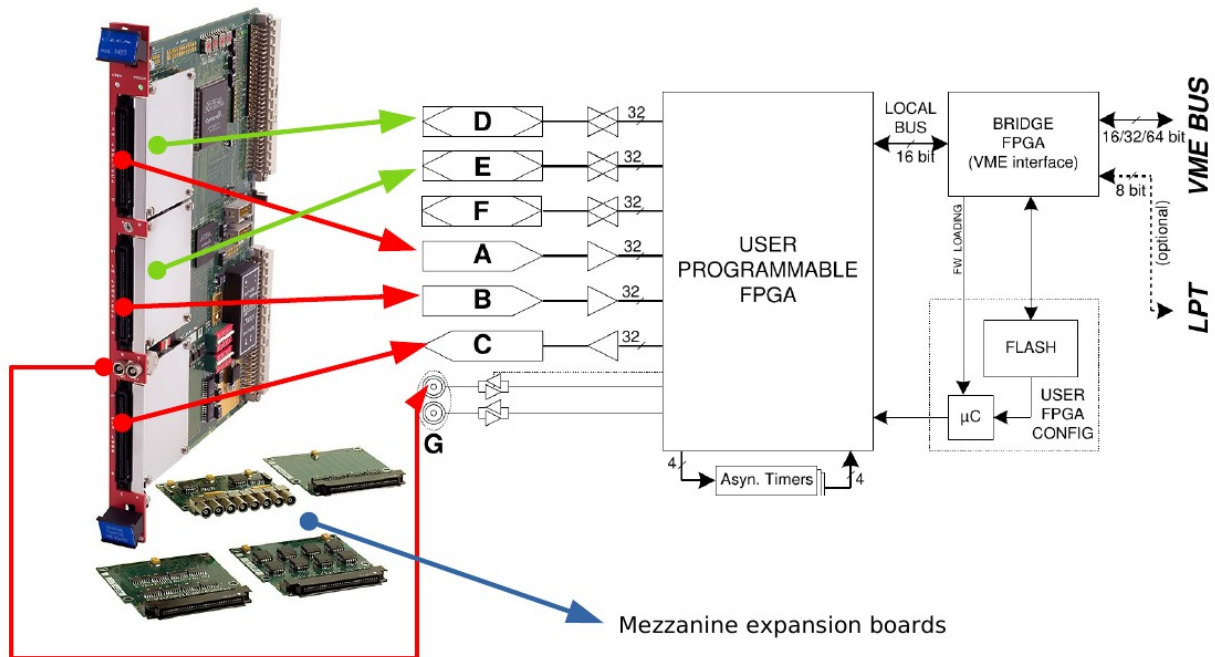


Figure 4.1: Description of V1495 Hardware and I/O registers

The V1495 timestamp platform provides the following specifications which are outlined below:

- V1495 Timestamp Board

- VME board is 6U high and 1U wide
- I/O Ports of V1495 support: NIM, TTL, ECL, and LVDS.
- The User FPGA is a cyclone I with maximum 20K logic elements (LE) [34]
- 64 inputs, expandable to 162 using mezzanine boards via Port D, E, and F (with 32 outputs)
- 32 outputs, expandable to 130 via Port D, E, and F (with 64 inputs)
- Support for PLL synthesis up to 405 MHz maximum frequency

- Interface registers which are provided by CAEN:

- VME addresses and registers are 32-Bit read and/or write registers

The V1495 VME module can be used to perform various applications, which are handled by two FPGAs mounted on the module's motherboard which will be discussed in the subsequent sections.

4.2 VME Bridge and User FPGA's

As shown in section 4.1 the V1495 has two FPGA's, the VME bridge FPGA and the user FPGA. The bridge FPGA is used to communicate with the VME bus and to program the user FPGA with custom firmware as seen in Figure 4.1. Accordingly, the user FPGA is used to manage all external input and output signals as seen in Figure 4.1 showing the schematic diagram of the bridge and user FPGAs and various I/O sections. Communication between the user FPGA and the VME bridge FPGA is being done through CAEN's own proprietary bus firmware which is known as the Local Bus (CAEN, 2019).

4.2.1 Local Bus

To be able to use the local bus the developer should take note of the signals used to interface with the bridge FPGA. The signals that are used between the bridge and user FPGAs are shown in illustration 4.1.

```
entity lb_int is
port(
  -- Local Bus in/out signals
  nLBRES  : in  std_logic;
  nBLAST  : in  std_logic;
  WnR     : in  std_logic;
  nADS    : in  std_logic;
  LCLK    : in  std_logic;
  nREADY  : out std_logic;
  nINT    : out std_logic;
  LAD     : inout std_logic_vector(15 downto 0);
  REG_STATUS : in      std_logic_vector(31 downto 0);

  -- FIFO in/out signals
  meb_dto  : in  std_logic_vector(31 downto 0);
  meb_rd   : out std_logic;
  meb_rdepty: in  std_logic;
  meb_rusedw: in  std_logic_vector (11 downto 0);

  -- Internal Registers Read Only Registers
  REG_R1 - REG_R6

  -- Internal Registers Read/RW Registers
  --REG_RW1 - REG_RW6
);
end lb_int ;
```

Illustration 4.1: CAEN Local Bus entity

This entity performs the task of accessing all I/O ports and allows the registers on the V1495 User FPGA to be read and written to from the ROC. There are two kinds of

registers that are used for all data communication and are organized in writing and reading accesses to specific registers which are accessible through the VMEbus interface. The local bus supports single read, single write, BLock Transfer (BLT) read which is accessed via 32-bit VME Access Mode (AM) (CAEN, 2019). It also provides communication between the user FPGA, the bridge FPGA, and the different entities. This local bus entity that was used for the timestamp is a modified version of the example entity that was provided by CAEN (CAEN, 2010a).

All registers of the local bus that are used and their functions are outlined in detail in the following sections. A complete listing of these registers and their addresses can be found in Appendix B.

4.2.2 Timestamp Configuration Registers

The development of Firmware for the timestamp system uses a modified version of the local bus module from CAEN to incorporate more readable and writable registers. One of the important registers created is the configuration register. The configuration register contains the information that is needed to configure the timestamp modules for timestamp and synchronization output of the timestamp platform. A description of this register can be seen in Figure 4.2.

Configuration Register																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HR																		CM	CE					SR	DM	E		Port Level Set			

Figure 4.2: Configuration Register for Timestamp platform

The different bits in the configuration register are described below:

Port level set - These bits are used to set the direction and the type of logic levels (TTL or NIM) for the ports D, E, F and G.

E - Enable. This bit is used to enable or disable the timestamp counters

DM - Data Mode, this bit is used to select either Single transfers or Block transfer mode

SR - Is the software reset of the system which is used to reset the internal timestamp counter registers

CE - Clock enable is used to enable the clock/sync out function of the timestamp module, in other words, used to produce an external TTL signal used for synchronization.

CM - Clock/sync mode, this bit is used to set the frequency of the output clock/sync.

4.2.3 Timestamp Data Registers

The timestamp event consists of 3 32-bit words as seen in Figure 4.3. These registers are the Event Count Register (ECR), Nanosecond Counter Register (NCR), and Second Count Register (SCR).

Data is written to these registers when the module receives the leading edge of the external trigger signal. Whenever a trigger is received the event counter increments by a value of 1 and the value of the nanosecond counter and second counter is written into their respective registers for single event read and written to Random Access Memory (RAM) when BLT is used.

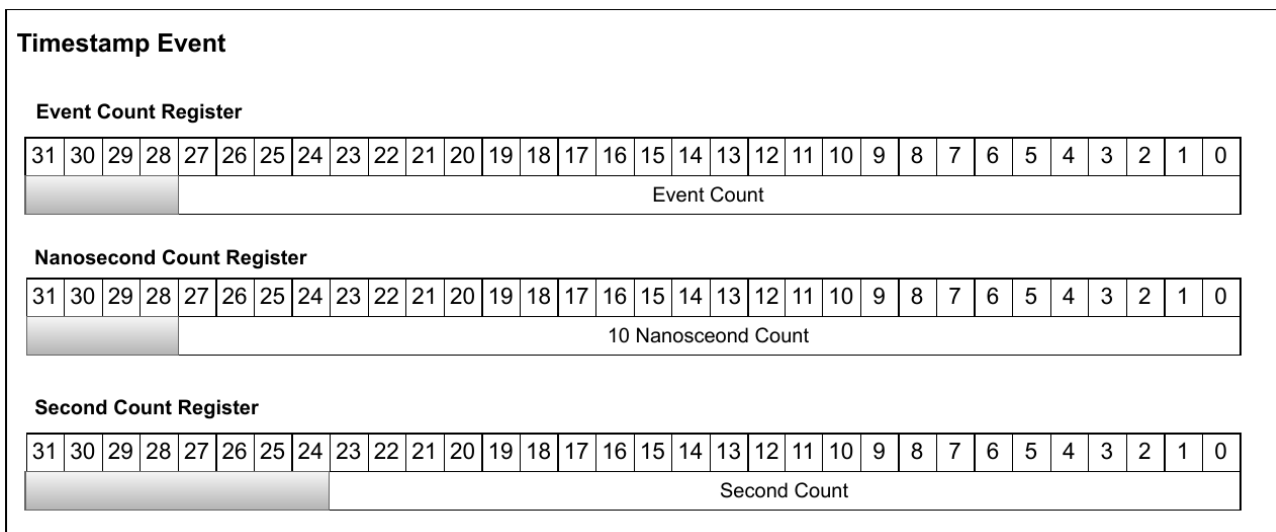


Figure 4.3: Timestamp platform Event timestamp registers

4.3 Timestamp Firmware Development

4.3.1 V1495_Timestamp Entity

The main entity for the timestamp platform is v1495_Timestamp_Logic. It provides access to all I/O ports of the board as seen in illustration 4.2. Appendix C provides the summary of all I/O signals that are being used for entity V1495_Timestamp.

```
entity V1495_Timestamp is
port(
  -- Front Panel Ports
  A   : IN   std_logic_vector (31 DOWNTO 0); -- In A (32 x LVDS/ECL)
  B   : IN   std_logic_vector (31 DOWNTO 0); -- In B (32 x LVDS/ECL)
  C   : OUT  std_logic_vector (31 DOWNTO 0); -- Out C (32 x LVDS)
  D   : INOUT std_logic_vector (31 DOWNTO 0); -- In/Out D (I/O Expansion)
  E   : INOUT std_logic_vector (31 DOWNTO 0); -- In/Out E (I/O Expansion)
  F   : INOUT std_logic_vector (31 DOWNTO 0); -- In/Out F (I/O Expansion)
  GIN  : IN   std_logic_vector ( 1 DOWNTO 0); -- In G - LEMO (2 x NIM/TTL)
  GOUT : OUT  std_logic_vector ( 1 DOWNTO 0); -- Out G - LEMO (2 x NIM/TTL)

  -- Port Output Enable (0=Output, 1=Input)
  nOED : OUT  std_logic;          -- Output Enable Port D (only for A395D)
  nOEE : OUT  std_logic;          -- Output Enable Port E (only for A395D)
  nOEF : OUT  std_logic;          -- Output Enable Port F (only for A395D)
  nOEG : OUT  std_logic;          -- Output Enable Port G

  -- Port Level Select (0=NIM, 1=TTL)
  SELD : OUT  std_logic;          -- Output Level Select Port D (only for A395D)
  SELE : OUT  std_logic;          -- Output Level Select Port E (only for A395D)
  SELF : OUT  std_logic;          -- Output Level Select Port F (only for A395D)
  SELG : OUT  std_logic;          -- Output Level Select Port G

  -- Expansion Mezzanine Identifier:
  -- 000 : A395A (32 x IN LVDS/ECL)
  -- 001 : A395B (32 x OUT LVDS)
  -- 010 : A395C (32 x OUT ECL)
  -- 011 : A395D (8 x IN/OUT NIM/TTL)
  IDD  : IN   std_logic_vector (2 DOWNTO 0); -- Slot D
  IDE  : IN   std_logic_vector (2 DOWNTO 0); -- Slot E
  IDF  : IN   std_logic_vector (2 DOWNTO 0); -- Slot F

  -- LED drivers
  nLEDG : OUT  std_logic;          -- Green (active low)
  nLEDR : OUT  std_logic;          -- Red (active low)

  -- Spare
  SPARE : INOUT std_logic_vector (11 DOWNTO 0);

  -- Local Bus in/out signals
  ...
);
END V1495_Timestamp ;
```

Illustration 4.2: Timestamp main entity

The signals that are used to configure the direction and the type of logic levels for ports D, E, F, and G are read from the configuration register. Signals nOED, nOEE,

nOEF, and nOEG are used to select if the ports will be used as input or output. The signals IDD, IDE, and IDF which is 3 bit wide selects the type of expansion boards used on ports D, E, and F. The expansion boards are hardcoded and not read from the configuration register. The signals nLEDG and nLEDR are output signals that are used to set the two LEDs (green and red respectively) of the board.

4.3.2 Timestamp Entity

The Timestamp entity implements the programmed asynchronous timestamp.

The implemented timestamp instantiates all required functions to be able to determine at what time during an experimental run an interesting event occurred and the readout of that time value.

The port definition of the time_stamp entity is shown together with the used input and output ports of the time_stamp entity which is described in illustration 4.3. The output buses tStamp, sec_counter, and tstamp_header are the nanosecond count, second count and tstamp_header holds the timestamp event counter.

```

ENTITY time_stamp is
  port(
    clk :    in      std_logic; -- time-stamp clock input
    A : in  std_logic_vector(31 downto 0); -- in a (32 x lvds/ecl)
    B : in  std_logic_vector(31 downto 0); -- in b (32 x lvds/ecl)
    output_C : out std_logic_vector(31 downto 0);
    GIN : in  std_logic_vector( 1 DOWNT0 0); -- In G - LEMO (2 x NIM/TTL)
    nLBRES : in  std_logic;
    areset : buffer std_logic;
    meb_wrfull : in  std_logic;
    meb_wr : out std_logic;
    meb_wrusedw : in  std_logic_vector(11 downto 0);

    ctrlreg : in  std_logic_vector(31 downto 0);
    ndiv_length : in  std_logic_vector(31 downto 0);
    reg_status : out std_logic_vector(31 downto 0);

    --time stamp specific variables
    tStamp : out std_logic_vector(31 downto 0); -- don't think this is needed
    sec_counter : out std_logic_vector(31 downto 0);-- one second counter
    tstamp_header : out std_logic_vector(31 downto 0); -- event counter

    clk_status : out std_logic_vector(31 downto 0); -- clock status
    clk_out : out std_logic;
    clk_rst : out std_logic
  );
END time_stamp ;

```

Illustration 4.3: timestamp entity

4.3.3 Timestamp Multi Event Memory

```
USE ieee.std_logic_1164.all;

LIBRARY altera_mf;
USE altera_mf.all;

ENTITY mb_fifo IS
    PORT
    (
        aclr          : IN STD_LOGIC := '0';
        data          : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
        rdclk         : IN STD_LOGIC ;
        rdreq         : IN STD_LOGIC ;
        wrclk         : IN STD_LOGIC ;
        wrreq         : IN STD_LOGIC ;
        q             : OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
        rdempty: OUT STD_LOGIC ;
        rdusedw: OUT STD_LOGIC_VECTOR (11 DOWNTO 0); --- number of words stored in the FIFO
    (read)
        wrfull      : OUT STD_LOGIC ;
        wrusedw    : OUT STD_LOGIC_VECTOR (11 DOWNTO 0) --words stored in the FIFO (write)
    );
END mb_fifo;
```

Illustration 4.4: Event storage entity

This entity (shown in illustration 4.4) was generated by Quartus II web edition software and it is used as a Memory module providing the Multi-event timestamp. The entity creates a First-In-First-out (FIFO) queue that holds the 3 values for each timestamp and is connected to the instantiated time_stamp component.

The queue is backed by the internal memory of the FPGA which emulates RAM (intel n.d.; CAEN tools for discovery n.d.). The values in the FIFO are ordered by the oldest first.

4.4 Synchronize and Timestamp Firmware Design

The time stamping platform is used for providing the event timestamp for the VME DAQ and for time synchronization by providing the synchronize signal (see Figure 4.4) which can be used by the other DAQ to synchronize it's internal counters based on the received signal. This is achieved by generating a pulse every 65535 counts at 100 MHz.

The module uses an external 40 MHz crystal oscillator which is then Phase lock looped to create an internal clock of 100 MHz. Using this 100MHz internal clock as a free-running 10ns counter that increments the 32 bit 10-nanosecond counter register and rolls over when it has counted up to 1-second, thereafter the 32 bit 1 second register will be incremented. By using the three registers describe 4.3.3 with the timing

resolution being at a maximum of 100 MHz (10 ns) the timer can be run continuously for 194 days, before roll-over.

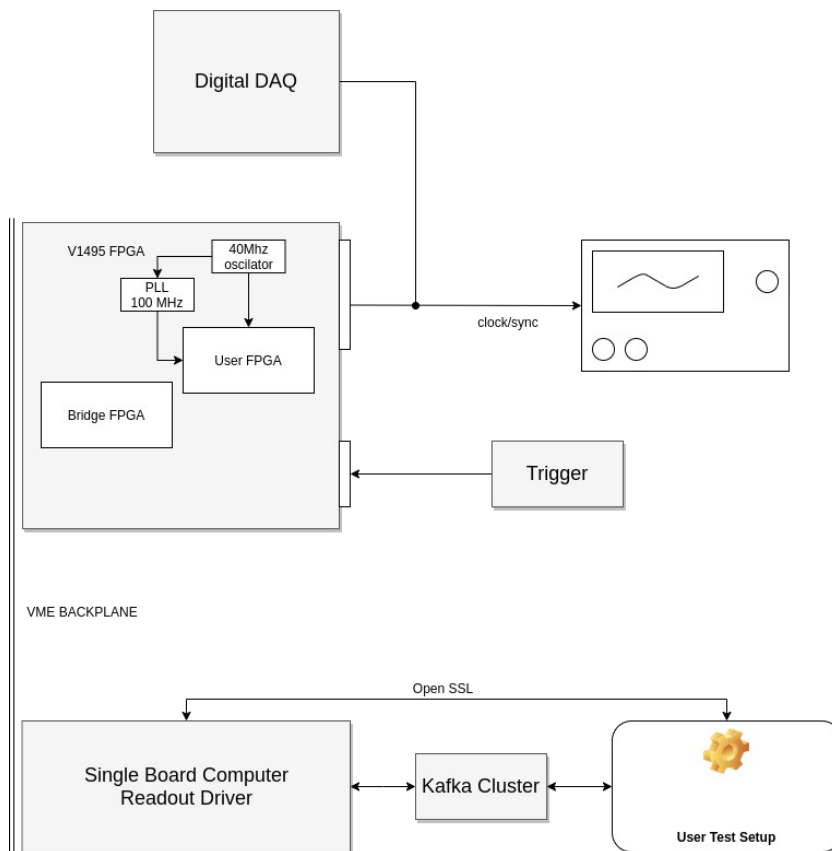


Figure 4.4: Block diagram of timestamp platform in VME crate

The timestamp implementation will be discussed in detail in the following sections.

4.4.1 Timestamp Functional Design

This section describes the development of a repetitive increasing timestamp for the K600 DAQ and it covers the behavioral description of the entity that was discussed in 4.3.2

a. Timestamp Event Counter Process

The event counter of the timestamp is used to determine that the correct timestamp value was readout for a given trigger when compared to the event numbers of the other VMEbus modules. The event counter for the timestamp is implemented using a process statement that increments the event counter. This asynchronous statement gets executed when a trigger signal is received on the trigger input port G input 0. This statement increments the event register by 1 and the data is written into the

internal event counter storage register of the timestamp platform as illustrated by illustration 4.5.

```
-- event counter latch
evt_count: process(areset_1, start_g1_in)
begin
  if areset_1 = '1' then
    evt_counter <=(others => '0'); -- clear event counter reset
    evt_flag <= '0';
  elsif start_g1_in'event and start_g1_in = '0' then
    if(time_stamp_enable = '1')then -- divide 100Mhz downto 20Mhz for external time sync
      evt_counter <= evt_counter + '1';
      evt_flag <= '1'; -- set event flag
    end if; -- start_g1
  end if;
end process evt_count;
```

Illustration 4.5: Counter latch

b. Timestamp Event Latch

The latching of the time counters is performed by using a process statement called `data_latch` that gets executed every time there is a change on the trigger input (illustration 4.6).

```
-- data latch
data_latch: process(start_g1_in, areset_1)
begin
  if areset_1 = '1' then
    scl_rod <= '0';
    meb_rod <= '0';
    latch_time_stamp <=(others => '0');
    latch_sec_timer <= (others => '0');
    evt_header <= (others => '0');

  elsif start_g1_in = '0' then
    if(time_stamp_enable = '1')then -- divide 100Mhz downto 20Mhz for external time sync
      if (read_mode = '1')then -- blt mode
        if (meb_wrusedw < conv_std_logic_vector(4087, 12) - rec_length) then
          latch_time_stamp <= time_stamp_counter;
          latch_sec_timer <= sec_timer;
          meb_rod <= '1';
        end if; -- start_g1_in

        elsif (read_mode = '0') then -- sct mode
          latch_time_stamp <= time_stamp_counter;
          Latch_sec_timer <= sec_timer;

        end if; -- readmode
      end if; -- timestamp enable
    elsif start_g1_in = '1' then
      scl_rod <= '0';
      meb_rod <= '0';
    end if; -- event
  end process data_latch;
```

Illustration 4.6: timestamp data latch

It latches the data of the counters into their respective storage registers on the leading edge of the trigger signal. Before data is written to storage registers the function first checks which readout mode the platform is configured for by reading the control register and set the storage of the time and event counters accordingly.

c. Timestamp Counters

The timestamp counters are implemented using an asynchronous process statement called counters. The process triggers on the internal 100 Mhz clock and increments the 10ns counter register until it reaches 100 Million counts. After the 10ns counter reaches 100M it is then cleared and the 1 second counter is incremented by 1 (illustration 4.7).

```
counters: process(clk, areset_1)
begin
    if areset_1 = '1' then
        sec_timer <=(others => '0'); -- clear second counter reset
        time_stamp_counter <=(others => '0'); -- clear nanosecond counter reset
    elsif clk'event and clk = '1' then
        if(time_stamp_enable = '1')then
            time_stamp_counter <= time_stamp_counter + '1'; -- increment nanosecond count
            if(time_stamp_counter = conv_std_logic_vector(100000000, 32))then -- 1 second @ 100 MHZ
                sec_timer <= sec_timer + '1';
                time_stamp_counter <= (others => '0'); -- reset the ns timer counter
            end if; -- timestamp counter
        end if; -- timestamp enable
    end if;-- clk event
end process counters;
```

Illustration 4.7

4.4.2 Timestamp Readout

The readout of the timestamp firmware (Appendix D) was developed to operate in two different modes; it can operate in single event readout mode where the timestamp is read from the V1495 module on an event-by-event basis when a trigger signal is received. The second mode of operation is that the V1495 can be set up to store timestamps of multiple events before using block transfer mode to read out said events. Both methods of readout can be configured through the control register by setting or clearing bit 7.

a. Single Event Readout

With the V1495 configured to operate in single event readout mode the event data for the timestamp is written to storage registers as can be seen in Figure 4.5. This event of the timestamp is immediately available to be read out via VMEbus. These storage

registers were described in detail in section 4.2.3.

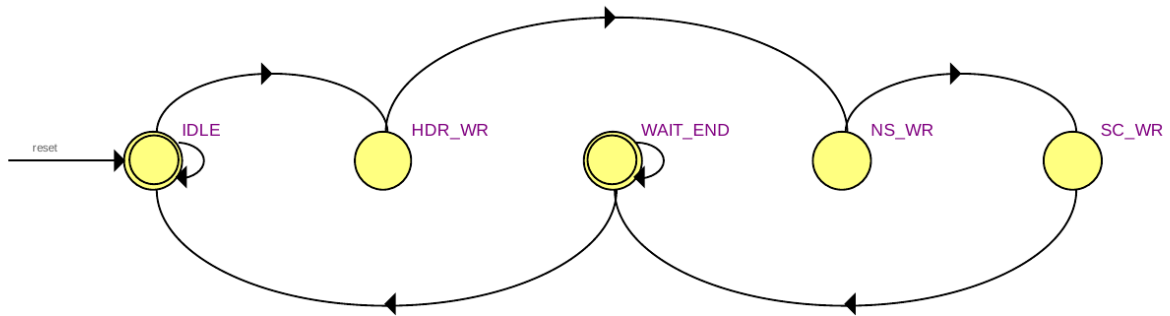


Figure 4.5: Single timestamp event state diagram

b. Multi-Event Readout

When the timestamp platform is configured to operate in multi-event mode the data for each timestamp is stored in a Multi-Event Buffer (MEB) as shown in Figure 4.5. BLT transfer is used to transfer a block of events from the user FPGA to the bridge FPGA DMA buffer for data readout. The MEB was implemented to be 3 x 32-bit deep and 32-bit wide. The latched timestamp data is written to the MEB every time the module received a trigger signal as shown in Figure 4.6.

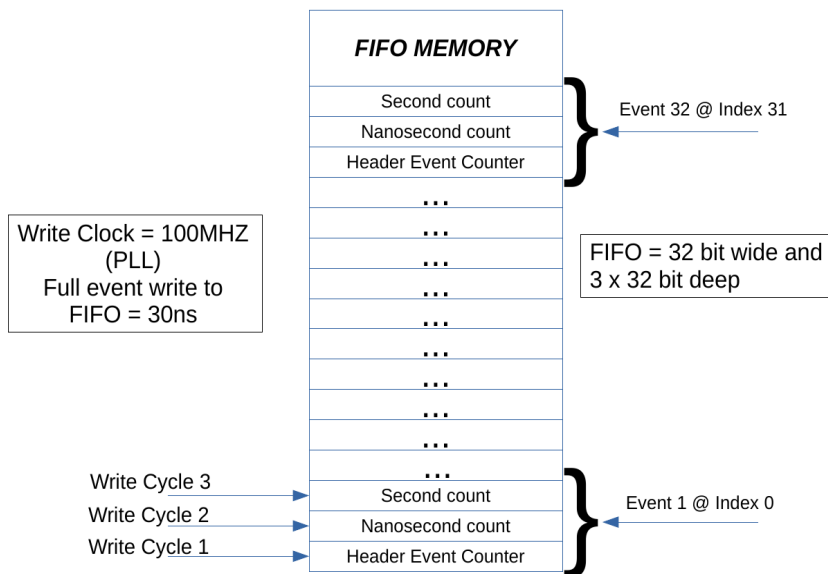


Figure 4.6: FIFO memory for Multi-event timestamp

To facilitate the operation of multi-event mode the following state machine was developed as seen in Figure 4.7. The state machine operates in a loop and transitions

through these different states based on the data write clock period which is equal to the global clock, LCLK (100MHz).

- a) Idle - During idle state, the system waits for an external trigger, and turns on both output Light Emitting Diode's (LED's) on the module's front panel.
- b) header write - The latched event counter is written to RAM.
- c) nanosecond count write - The latched nanosecond count is written to RAM and turns on both LED's on the front panel of the modules
- d) second count write - The latched second count is written to RAM.
- e) wait for write complete - Turns the LEDs off to indicate the write completed, and set the Finite State Machine (FSM) back to idle state

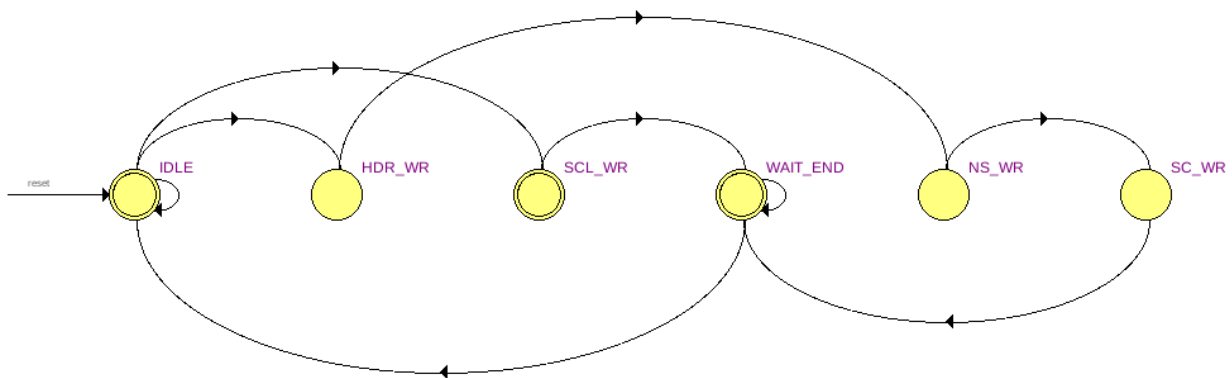


Figure 4.7: Multi-event timestamp state diagram

4.5 Timestamp Interface Software Development

The readout software has been updated to accommodate the added timestamp. A software module is added to the readout software to accommodate the configuration and readout of the V1495 FPGA module. The current implementation of the timestamp can either read a single timestamp event or read N number of events from the MEB on the VME Timestamp platform. The V1495 software interface was developed using C/C++ based routines to access the timestamp platform via VMEbus as shown in Figure 4.8.

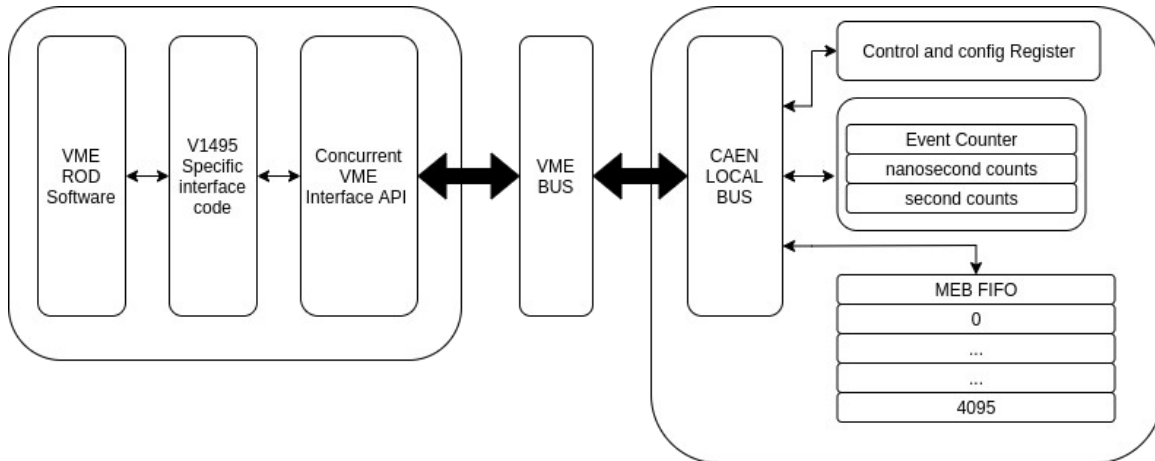


Figure 4.8: VME API block diagram

Using the Concurrent VME API (Concurrent Technologies n.d.) the software module was developed for the timestamp platform. The software interface contains all the VME access functions that are used to to read device status, write configuration and readout of timestamp events.

The software was modified to enable the user to easily interact and configure the timestamp platform. Also, it was ensured that the software interface together with the VME API provided by concurrent technologies, would let a user accomplish the task of reading events from the timestamp VME platform effectively. The FPGA functionality and structure of the interface software that is used to access the different registers within the FPGA module are presented below:

`v1495_DataRead(vme_interface *vme, uint32_t vme_addr, DWORD* dest, DWORD readMode, int16_t entries)`

- *What* : Read the timestamp values from the FPGA module when an event occurred
- *How*: Sets up the transfer mode that needs to be executed using the readout structure `vme_interface`. After the readout structure has been setup for single event or multi-event readout, the timestamp events are readout from the timestamp platform.
- *Result*: Returns success if the data have been successfully read or error.

`v1495_reset_clk(vme_interface *vme, uint32_t vme_addr)`

- *What* : Reset all the TimeStamp Counters (TSC) within the FPGA.

- *How:* Sets up the vme_interface structure in write mode. The function then sets the reset bit in the control register of the timestamp platform after which the reset bit is cleared.
- *Result:* Returns if the reset bit has been successfully set or error.

v1495_stop(vme_interface *vme, uint32_t vme_addr)

- *What:* Stops the TSC within the FPGA
- *How:* Sets up the vme_interface structure in write mode. The function then clears the enable bit in the control register of the timestamp platform
- *Result:* Returns if the bit has been successfully cleared or error.

v1495_start_clk(vme_interface *vme, uint32_t vme_addr)

- *What:* Starts the TSC within the FPGA.
- *How:* Sets up write transfer mode in the vme_interface structure. The enable bit is then set in the timestamp platform's control register.
- *Result:* Returns success if the TSC has been successfully enabled or error.

v1495_init(vme_interface *vme, uint32_t vme_addr, DWORD ctrl, DWORD num_events)

- *What:* Initialises the timestamp platform
- *How:* Sets up the transfer mode that needs to be executed using the readout structure vme_interface. The function first resets the TSC then writes the configuration bits to the platform configuring all I/O direction and levels.
- *Result:* Returns successful if the FPGA has been successfully configured or write error.

4.6 Conclusion

This chapter discussed the development of the timestamp. The system makes use of the V1495 FPGA Module from CAEN that runs a simple application that captures the event timestamp when it receives an external trigger from the detector electronics. The timestamp platform then latches the timestamp and writes it to memory where it can be accessed via VME Bus.

CHAPTER 5

System Verification and Experimental Results

This chapter describes the tests that were performed and the results acquired to verify the operation of the new DRSS and timestamp. The results obtained from tests are then analyzed to determine the usability and effectiveness of the new data acquisition system. Using the results obtained from the various functional tests will set the scene for further recommendations and conclusions for the discussed DAQ.

5.1 Configuration and Communication with Other Subsystems

These tests represent the integration testing of the new data acquisition system against the functional requirements of the producer and consumers. This system level tests were conducted using messages with input values that fall within the expected range and messages that are out of the expected range. Since the other subsystems for the iThemba LABS Dolosse DAQ is still under development, as for example the user's WebClient used to configure the DAQ, the testing workbench methodology was followed. That is to say that all tests described in this section and chapter were carried out by emulating the JSON data strings that would be produced through Kafka to the frontend electronics' single board computer. This means python test scripts were created that produces JSON strings that simulate the control and configuration interfaces that is currently under development. These test programs were launched and operated from a separate computer connected to the Kafka computing system, but on the same network to evaluate the system with network communications identical to those expected from the actual DAQ subsystems.

Figure 5.1 illustrates the simple test setup consisting of a VME crate with the readout system computer, Kafka cluster, and the test computer hosting the simulation programs.

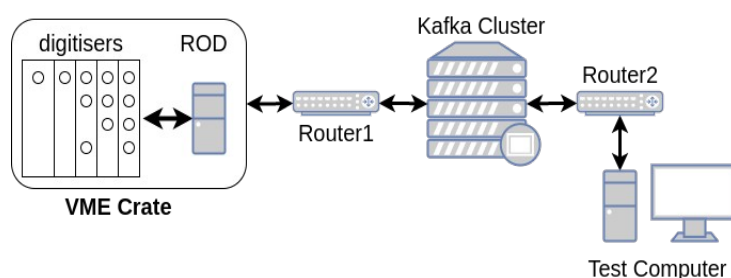


Figure 5.1: DAQ simple test setup

Figure 5.2 shows the startup handshaking between the data acquisition system and the backend server system where configuration information is being sent to the DAQ. A custom JSON data structure is used to hold the configuration information sent by the backend server system to Kafka which is consumed by the readout computer. When the system has been configured successfully it returns a simple JSON string stating that the system configuration was successful and the DAQ is in ready state. In this state the DAQ is ready to receive control information to start acquiring data.

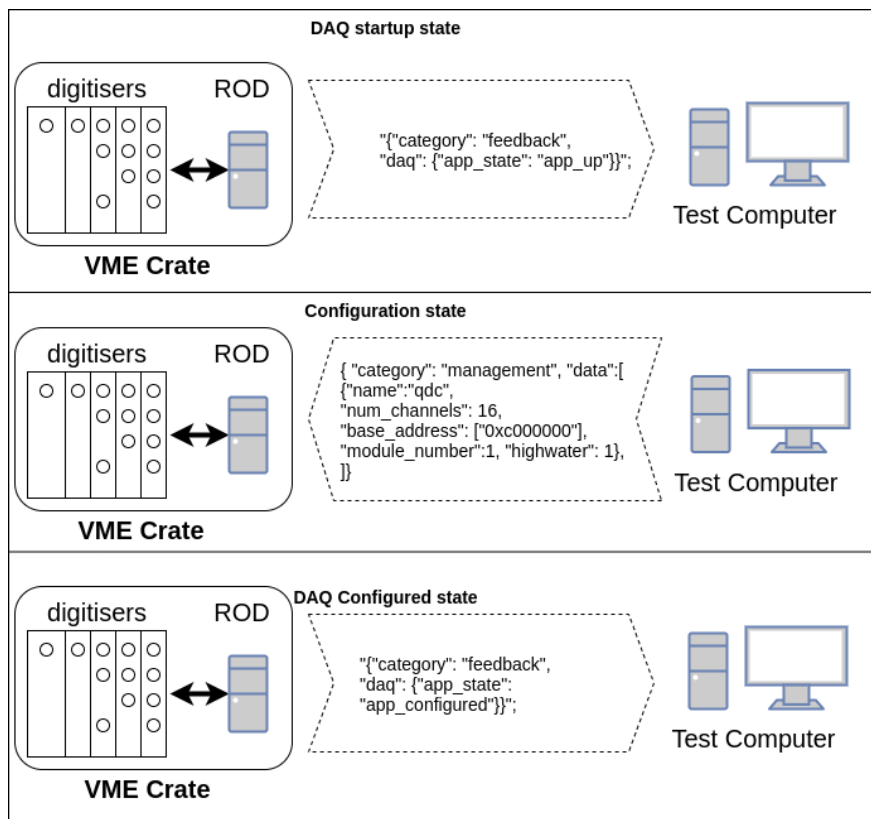


Figure 5.2: DAQ Configuration handshaking between ROC and Run Control System

These configuration tests were repeated multiple times using proper configuration data and “corrupted” configuration data. When tested with the expected configuration data the configuration went through successfully, and when tested with “corrupted” configuration data it resulted in the program terminating the application and notifying the user of the failure.

5.2 Readout Test and Measurements

This section describes and discusses the equipment used and techniques employed to acquire operational and benchmark measurements of the new readout software for the data acquisition system. These measurement tests include the system readout time using the different readout methods employed and the periods of which it is composed. The intervals that constitute the readout time are: the trigger veto, software readout delays, and data transfer durations as shown in Figure 5.2. The readout duration is known as dead time of the system and it can be defined as the time interval between events in which the DAQ is unable to process another event (Neveling R., et al).

The readout tests were designed to measure the intercommunication between event digitization, the ROC, Kafka computing cluster, and the event builder. These interactions are critical to the operation of data acquisition software and such it will be closely scrutinized.

5.2.1 Readout Process

The fundamental operations of the K600 data acquisition system are the digitization of detector output, data readout, and transmission. These operations are executed repeatedly consecutively and as can be seen in Figure 5.3. These critical operations determine the performance of the data acquisition system and such will determine the suitability of using data streaming technologies as a medium for data readout.

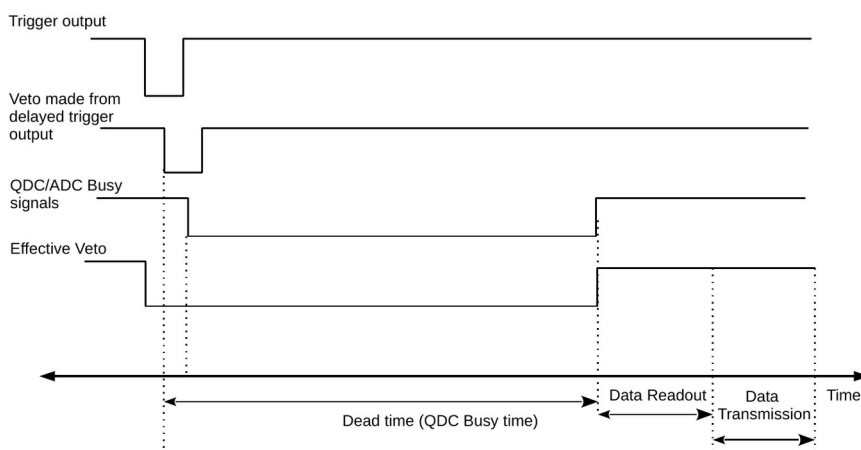


Figure 5.3: Trigger Timeline and Deadtime (Pool et al, 2018)

The current DAQ readout algorithm supports both single event readout, meaning it reads out data event-by-event, and it supports multi-event readout by utilizing the digitizer module's internal Multi-Event Buffers (MEB) before reading out the data. Using these readout methods the new readout software satisfies the requirements of reading out data event-by-event and future developments using the digitizers' MEB. This means the current implementation of the DAQ reads out a single event from each module and buffers the data in the Kafka computing cluster by producing the data to a unique topic.

To prevent the DAQ to respond to any new incoming triggers the busy signals from the QDC/ADC in combination are used to veto the trigger.

5.2.2 Readout Component Intervals

The following sections describe the operations involved in the readout process which determines the DAQ's performance and ultimate deadtime of the system.

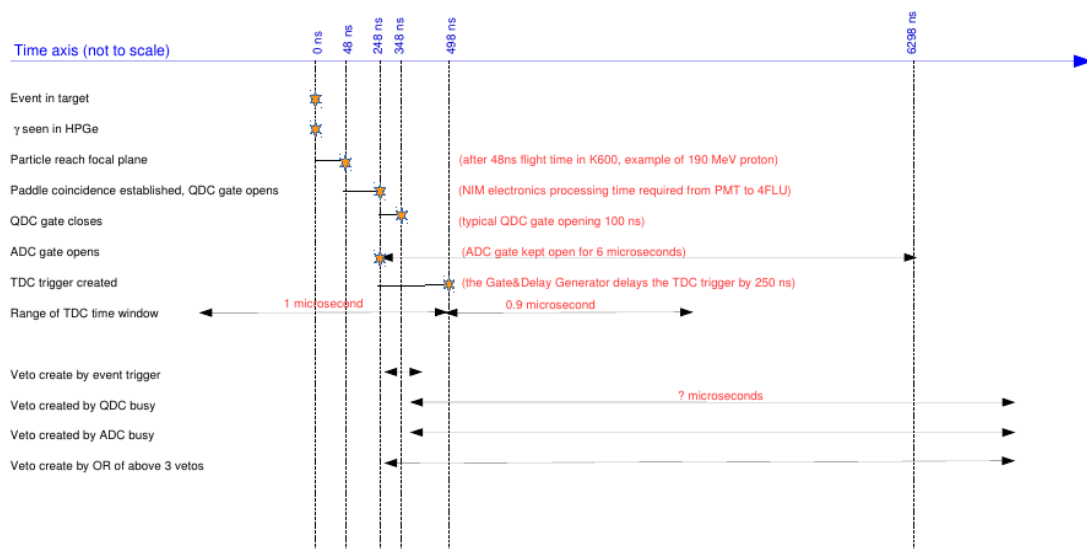


Figure 5.4: Trigger timeline when event is above threshold (Pool et al, 2018)

a. ADC/QDC Busy Signals

This is the period of the GATE signal input on the QDC and/or ADC and the time it takes to complete the digitization after the trigger input. The GATE signal can be defined as the integration period of the input signal. This period of the QDC veto was measured using an oscilloscope and it was found to be $\sim 6\mu\text{s}$ as seen in Figure 5.5. This confirms the results that were stated in the V792 QDC Datasheet. The total period is $\sim 6\mu\text{s}$ because the gate signal on the QDC is 100ns (as seen in Figure 5.4) and the

time it takes for the QDC to convert all its channels is 6 μ s as can be seen in CAEN, 2010b



Figure 5.5: QDC busy time output signal (green)



Figure 5.6: Trigger (yellow) with QDC busy time (green)

b. Readout and Data Transmission

The time period to read out the data from the signal digitizers is determined by the amount of data that needs to be transferred to the external memory buffer and ultimately the storage or Analysis computers.

For the DAQ under discussion, the digitized data is transferred from the digitizer modules, it is then validated to make sure that the data received from the modules is valid and not filler words as described in (CAEN, 2010b; CAEN, 2012a, CAEN, 2012b). This validated data is then transmitted from the ROC to the external memory buffer in Kafka into their unique topic(s). The time it takes to transfer data from the digitizers to the ROC is dependent on the transfer rate of the VMEbus and the mode of transfer selected (data width, single or block data transfer).

5.2.3 Test Configuration and Measurement

To verify the new data acquisition system and determine its performance the DAQ was evaluated using two uncorrelated Pulsars as seen in Figure 5.7.

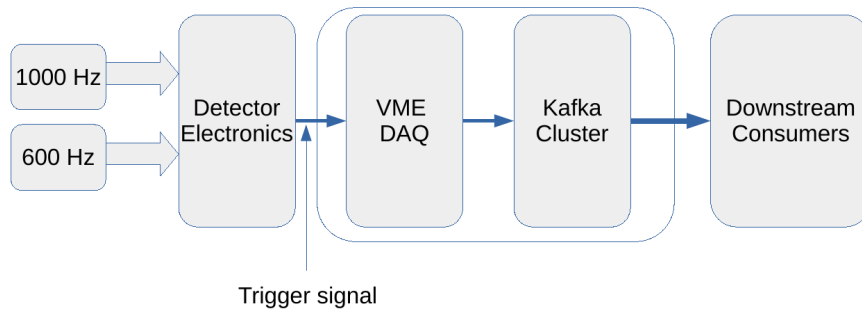


Figure 5.7: DAQ test evaluation setup

Two uncorrelated pulsars were used to simulate detector output that is above threshold to generate triggers from the detector electronics. These tests were created and it forms part of the system integration tests with the downstream data consumers.

The system tests that were conducted were divided into measuring four metrics namely;

1. Readout integration tests using DMA direct transfer and MEB DMA direct transfer.
2. Event builder evaluation.
3. Functional tests of DAQ with timestamp.
4. Performance measurements.

The results that are achieved from these DAQ tests are then analyzed to verify the system. This is done to form conclusions based on the analyzed results, so that future work can be recommended.

5.2.4 Measurement Techniques

a. Readout Interval Timing Techniques

All the measurements that will be discussed in the next section requires a method to determine the time intervals of readout functions. These intervals include the time it takes to readout the event of interest from the VME modules to the transmission of data to the external memory buffer. These timing tests are important

because it will inform that the DAQ meets the real-time requirements. Timing methods are required to warrant meaningful results and the mechanism used to determine the readout rate of the data acquisition system is the monotonic clock that will be discussed in the subsequent sections.

b. Data Readout Measurements

The primary purpose of the readout tests was to test attributes of the system under investigation that included communication between the VMEbus modules to the ROC and between the ROC and Kafka cluster over the network. Tests at this level are to ensure the integrity of the data readout and that the readout software as a complete entity fulfills its operational requirements.

All measurements that are discussed in the subsequent sections were done with the DAQ setup with V792 QDC, and 7x V1190A TDC’s in coincidence (as seen in Figure 5.8) connected to the Kafka cluster. The Scaler events were read out periodically every second and are not discussed as part of these tests.

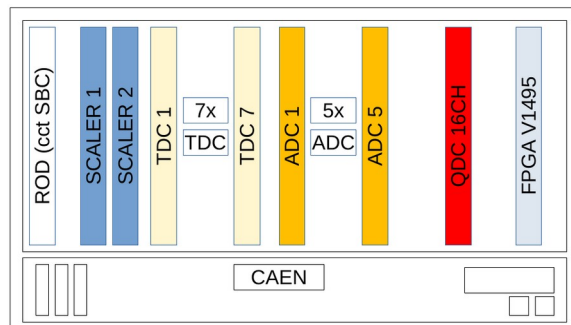


Figure 5.8: Depiction of k600 DAQ with signal modules

Various fixed event sizes were read out from the VMEbus modules to test that the readout software is able to scale with changing event sizes. The DAQ was tested by performing multiple readout tests to ensure that the software components meet their functional requirements. Statistical results of the readout metrics obtained are shown in Table 5.1 and is provided in Appendix E.

Table 5.1: Statistic results with QDC and 7 TDC's in coincidence

Parameter	Single Event Readout	Multi-event Readout (7 events)
Wirechamber Events (VMEcrate)	45948	45947
Total Run length (s)	30	30
Average QDC readout and data transmission times	41 us	56 us
Average TDC readout and data transmission times	262 us	749 us
Instantaneous Readout time (s)	495,028 us	1255,33 us
Data rate (kB/s)	146	133

The data for the key statistical parameters provides the mean of the readout and data transmission periods that were calculated using the following formula:

$$x = \frac{\left(\sum_{i=0}^{N-1} x \right)}{N}$$

Equation 2: Mean Calculation

c. Measurement Techniques Used

This section demonstrates how readout and timing measurements on the data producer were performed. Measurement and analysis were conducted for all functions of data readout on the data acquisition system which includes the transmission of data from VMEbus modules to ROC, the data validation process, and data transmission to the external memory buffer (Kafka Cluster). All tests (as seen in Table 5.1) conducted was done by using to 2 uncorrelated pulsars at frequencies 600 Hz and 1.00 kHz which gives a total trigger rate of 1.6 kHz.

The readout routines and data transmission routines were monitored using the “clock_gettime” function which is a POSIX function provided by the Linux Operating System (clock_gettime (2) - Linux Man Pages n.d.). The readout functions is enclosed with a call to function clock_gettime to determine the start and end times of the function of interest. Using the start and end timestamps the interval time can be calculated using the difference between the two timestamps $\Delta t_f = (ts1 - ts2)$. To transform this timestamp into standard units of time the difference between the two timestamps Δt_f are multiplied by a constant to derive the time measured in seconds as seen in illustration 5.1. The call to the “clock_gettime” function supplies an inexpensive method of determining the readout and data transmission periods.

```

#include <time.h>
#define CONSTANT 1E9L

struct timespec begin, end;
uint64_t time_diff;
clock_gettime(clock_id, &begin);
/* function call */
clock_gettime(clock_id, &end);

time_diff = (end.tv_sec - begin.tv_sec) * CONSTANT + end.tv_nsec - begin.tv_nsec;

```

Illustration 5.1: retrieval of function execution interval

The different readout methods that were evaluated included 32-bit single event DMA direct transfer and Multi-event transfers from the different modules all using address mode A32. Using the measurement technique described in the previous paragraph, throughput for these data transfer and produced functions was determined. The timestamp values that were obtained from calling clock_gettime function to determine the start and end of the readout routines provides the duration required to transfer the specified number of bytes from the VME modules to the ROC, and from the ROC to the external memory buffer.

d. Data Readout Rate

The measurement of this system metric is perhaps the most elementary process. This test was created to determine if Kafka would be suitable for the data rate at which the ROC would read out and produce data. The transfer rate of data was measured between the ROC and the VMEbus modules to determine the amount of data that is transferred over a period during an experimental run. This readout test also consisted of comparing different DMA readout methods of single and multi-event block transfers and their data rates to find the best and most reliable method possible for the transmission of data across the VMEbus.

During the test, data rates were continuously recorded by using a storage variable called WirechamberData within the ROC's software, which maintains the total number of bytes readout from the VMEbus modules during an experimental run. A periodic thread was created that expires every second and in this way, it was possible to determine the amount of data readout per second during the acquisition process for a given trigger rate. The total number of bytes that were recorded over the experimental run gets stored in another storage variable called prevWirechamberData everytime the thread is executed. The difference between and the WirechamberData

and prev WirechamberData, provides the number of bytes that get transmitted.

The results from these experimental runs determined as shown in table 5.1 that Kafka is more than suitable to be used as transport medium because the readout rate of the current DAQ is 146 and 133 kb/s for single and multi-event respectively. These results provided evidence that using Kafka as external memory is more than capable for the data rate of the ROC, because Kafka was benchmarked at 605 MB/s as it was demonstrated in Confluent n.d.

5.3 Event Builder Evaluation

This section discusses the measurements and results that were obtained using the new event builder application that was added to the readout chain as shown in Figure 5.9. Tests were conducted with varied fixed size event fragments that were produced to the external memory buffer from the ROC. This is done in order to assess the scaling behavior of the event builder, where the number of events read out from the VMEbus modules is varied and set to 1, 2, 4, or 7 for evaluation purposes.

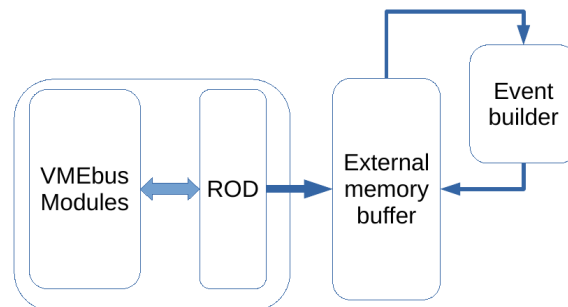


Figure 5.9: Event builder test setup

The raw event format produced to the external memory buffer is not conducive for easy analysis. A thread was created in the event builder application that polls the data topics in the Kafka cluster for any available event fragments that were produced by the ROC. The available data is then consumed from the cluster and parsed, converted into a usable format, and produced back into the cluster.

5.3.1 Event Rate Measurement Techniques

To measure the performance of the event builder a periodic time-out thread was employed by using the Python class `threading.Timer`. When the timer expires it

calculates and sends the status and performance metrics to Kafka where it can be used by downstream data consumers. The event builder performance was measured in events per second that were collated and parsed in a run which is given by the following formula:

$$x = \frac{1}{T} \left(\left(\sum_{i=0}^{\infty} Et \right) - Ep \right)$$

Equation 3: Event rate calculation

The number of events per period was determined from the total sum of events per run and the difference between the total events and the event count in the previous period, divided by the period of the thread timeout.

Figures 5.10 and 5.11 show the distribution of the number of events per second that was created with the event builder for single and multi-event readout respectively. Statistics that were collected shows the number of events that were parsed and collated per second by the event builder. The discrete nature of the event builder statistics is because how the event rate was calculated, and how the data was collected.

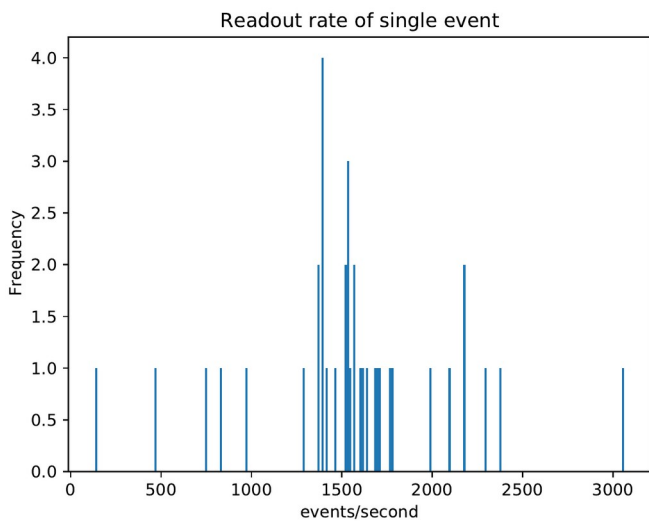


Figure 5.10: Histogram of event rate vs occurrence for event-by-event readout

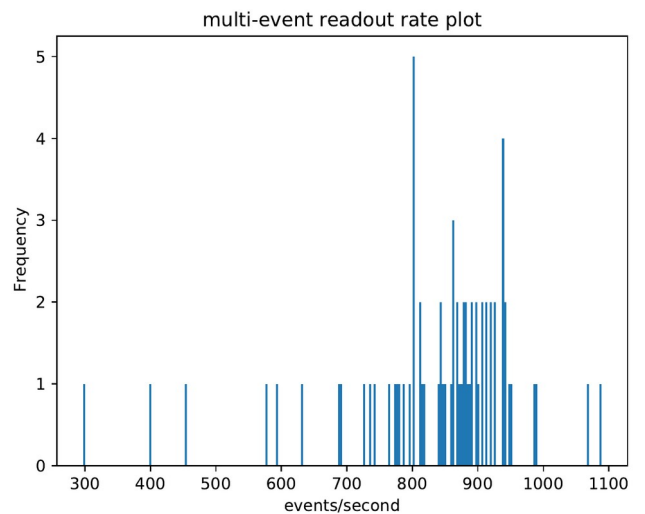


Figure 5.11: Histogram of event rate vs occurrence for multi-event readout

5.4 Results Obtained

To verify the integrity of the data readout from the signal digitizers and collated events that was created by the event builder, a downstream Kafka consumer was created and used with a modified version of the K600 RootAna analyzer (Neveling, R. et al.; RootAna n.d). The analyzer was used to consume the data from the cluster by

polling the event topic every 100ms for data availability. Using the consumed data from the cluster the data was visualized by creating the following plots with the RootAna Analyser:

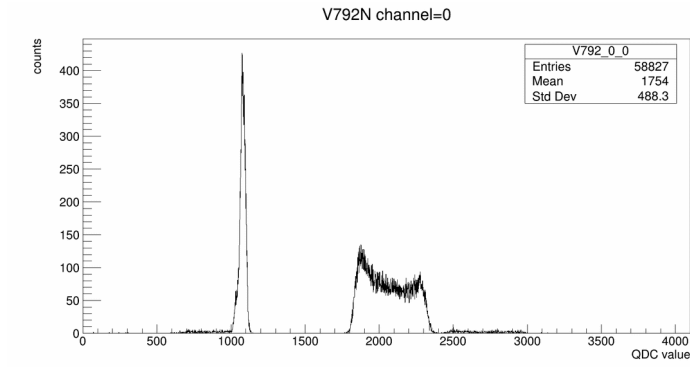


Figure 5.13: QDC RAW values

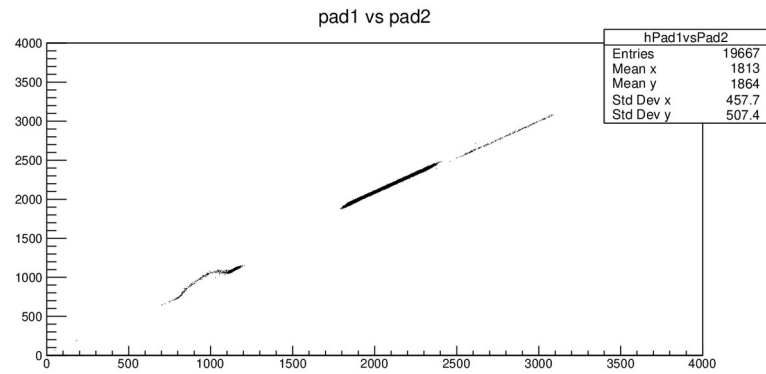


Figure 5.12: Energy loss paddle1 vs paddle 2

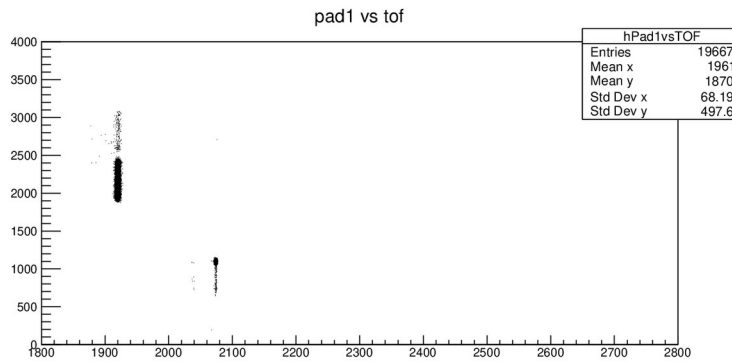


Figure 5.14: Pulse height vs time of Flight

Figure 5.13 is showing the RAW ADC data from the QDC and Figure 5.12 is a plot of energy loss in paddle 1 versus the energy loss in paddle 2 in the focal plane detector. The last plot in Figure 5.14 shows the plot for particle identification (PID), this is done by plotting the pulse-height of paddle 1 against the Time Of Flight (TOF) of the particle.

This test was done using the configuration mentioned in 5.2.3 using the two uncorrelated pulsars that simulate trigger signals from the focal plane detector.

5.5 Timestamp Platform Measurements

All test that is discussed in the following sections were done with the V792 QDC and V1495 in coincidence connected to the Kafka cluster to verify that the software and hardware interface of the timestamp platform operates according to the requirements. The same test setup with two uncorrelated pulsars was used as described in 5.2.3 but at lower frequencies and the two different operation modes of the timestamp were evaluated.

To test the system and verify that the timestamp platform is operating according to requirements a test setup was created using the VME crate with QDC and FPGA module.

The test that is described in this section is the evaluation of the timestamp generation within the FPGA module and the data transfer from the VMEbus module to the ROC.

5.5.1 Timestamp Event Readout

Tests at this level ensure that the interface software to the FPGA module, as a sub-entity complies with its operational requirements.

The timestamp packets are read out every time the V1495 received a copy of the trigger. The event number in the header data field from each event fragment of the FPGA VMEbus module is used to compare with the event number from the QDC to determine coincidence and that the timestamp is operating correctly. When there was an event match the timestamp packet is produced to the external memory buffer, where the event builder added the timestamp to the event structure.

Experimental runs was completed multiple times and below is an example of the timestamp data collected using a periodic pulsar at 1kHz and the statistical metrics of the timestamp platform:

```
{'category': 'measurement', 'technique': 'k600', 'run number': 1, 'Event number': 22060, 'Event': [{'name': 'qdc_0', 'data': ['0xfa001000', '0xf800458b', '0xf8104083', '0xf80249aa', '0xf812409a', '0xf8044cfe', '0xf8144084', '0xf8064d0a', '0xf816408b', '0xf8084071', '0xf8184083', '0xf80a4073', '0xf81a409f', '0xf80c408f', '0xf81c4098', '0xf80e4083', '0xf81e40a5', '0xfc005616']}, {'name': 'tstmp_0', 'data': ['0xfe005617', '0x5b56aa3', '0x15']}]}
```

```
{'category': 'measurement', 'technique': 'k600', 'run number': 1, 'Event number': 22061, 'Event': [{'name': 'qdc_0', 'data': ['0xfa001000', '0xf80045c7', '0xf8104083', '0xf80249f8', '0xf812409a', '0xf8044d2a', '0xf8144084', '0xf8064d37', '0xf816408b', '0xf8084072', '0xf8184083', '0xf80a4072', '0xf81a409f', '0xf80c408f', '0xf81c4098', '0xf80e4084', '0xf81e40a5', '0xfc005617']}, {'name': 'tstmp_0', 'data': ['0xfe005618', '0x5b6efe0', '0x15']}]}
```

The event packets shown contain a system timestamp which consists of a header with the event counter, 10 ns counter and, 1 second counter.

Table 5.2: QDC and V1495 Timestamp statistics

Parameter	Single Event Readout	Multi-event Readout (7 events)
Total number of Events	20597	20591
Total Run length (s)	30	30
Average timestamp data Transmission times	33 us	54us
Total Readout time with QDC	47 us	82us

These statistical metrics as shown in Table 5.2, were obtained by using the same process as described in section 5.2.1. The timestamp readout function was encapsulated in the two timestamps and the difference will give the readout time. Adding the time it takes to readout timestamp to the total readout time will increase on the readout time by an average of 33 us for single event and 54 us for MEB readout of 7 events, so the timestamp does have a slight impact on the readout time. The timestamp though provides minimal changes in readout and because of this, the timestamp can be used with the DAQ in its current form and still maintain the real-time requirements of the system but at lower trigger rates.

a. Timestamp Tests Using Single Event Readout

The timestamp platform was first evaluated using a single periodic 1kHz pulsar as shown in Figure 5.15. During these tests multiple runs were done evaluate the functional operation of the timestamp platform.

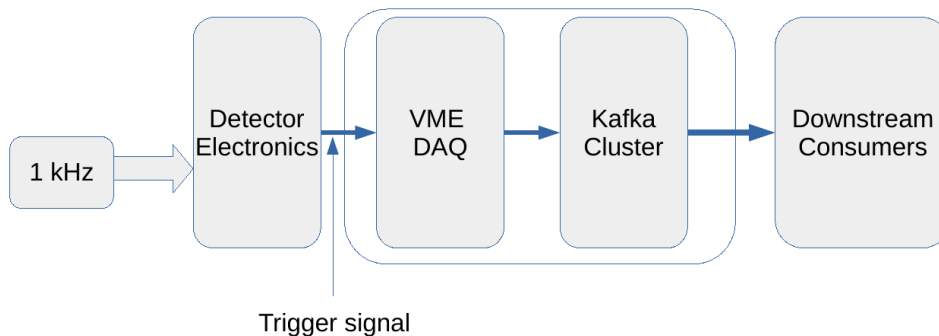


Figure 5.15: Single Periodic Pulsar connected to DAQ

The timestamp system must record correct data continuously. Table 5.3 shows consecutive values of timestamps that was readout from the timestamp platform using the function “v1495_DataRead” and single 1kHz pulsar. The values are readout are in raw hexadecimal format. The hexadecimal value “0xfe” value in the header is prepended to the event counter as timestamp identifier and can be ignored when the timestamp data is analyzed. The period between consecutive readouts was calculated and found to be 1003.512 Hz.

Table 5.3: Consecutive Results recorded

Event number (Header)	10ns Counter	1 second counter
0xfe005617	0x5b56aa3	0x15
0xfe005618	0x5b6efe0	0x15
0xfe005619	0x5b87509	0x15
0xfe00561a	0x5b9fa3f	0x15
0xfe00561b	0x5bb7f86	0x15
0xfe00561c	0x5bd04ab	0x15
0xfe00561d	0x5be89dc	0x15
0xfe00561e	0x5c00f21	0x15
0xfe00561f	0x5c19468	0x15
0xfe005620	0x5c319aa	0x15

b. Timestamp Multi-event Buffer Tests

The multi-event buffer (MEB) and BLT transfer of the timestamp platform was evaluated using the same single periodic 1kHz pulsar as shown in Figure 5.15 and using the function “v1495_DataRead”. Multiple runs of readout of 7 events from the MEB was performed to establish that the MEB functions as per functional requirements. All data transfers between the timestamp platform, ROC and external memory buffer (Kafka) were successful when tested as shown by the results achieved in Table 5.4. The snapshot of the experimental runs can be found in Appendix E and only a snippet of consecutive results are shown to improve readability of the thesis.

Table 5.4: Results of readout data from multi-event buffer on timestamp

Event number (Header)	10ns Counter	1 second counter
0xfe0027a4	0xa392f7	0xa
0xfe0027a5	0xa51804	0xa
0xfe0027a6	0xa69d14	0xa
0xfe0027a7	0xa8221d	0xa
0xfe0027a8	0xa9a72a	0xa
0xfe0027a9	0xab2c4a	0xa
0xfe0027aa	0xacb173	0xa
0xfe0027ab	0xae3687	0xa
0xfe0027ac	0xafbba1	0xa
0xfe0027ad	0xb140d9	0xa
0xfe0027ae	0xb2c5f1	0xa
0xfe0027af	0xb44af9	0xa
0xfe0027b0	0xb5d001	0xa
0xfe0027b1	0xb75526	0xa

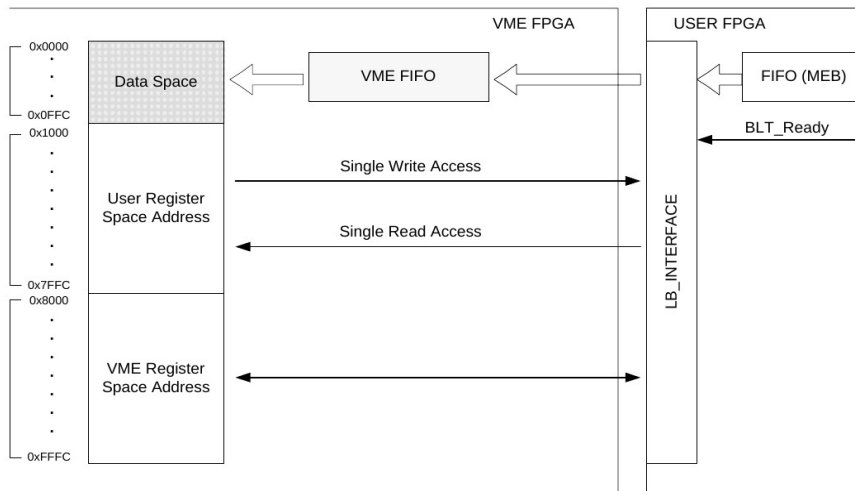


Figure 5.16: Address space block diagram (CAEN, 2010a)

The period between consecutive events was calculated to be 1004.0463 Hz. However it was established during these tests that the MEB readout doesn't meet functional requirements. During consecutive experimental runs (as shown below) it was found that the readout using CAEN LB_INTERFACE BLT that all data was not readout from the VME buffer, and some data of the previous run remained. This is due to the data being transferred to the VME buffer from the User MEB FIFO during readout before it is

transferred to the ROC as seen in Figure 5.16. This indicates that data transfer between User FPGA and the VME-bridge FPGA in combination with the read request was slower than expected.

```
data in buffer = {85: ['0xfe000055', '0xbaec84', '0x0'], 86: ['0xfe000056', '0xbd7db8', '0x0'], 87: ['0xfe000057', '0xbe6555', '0x0'], 88: ['0xfe000058', '0xc00eeb', '0x0'], 89: ['0xfe000059', '0xc2a01f', '0x0'], 90: ['0xfe00005a', '0xc53153', '0x0'], 91: ['0xfe00005b', '0xc7c288', '0x0'], 92: ['0xfe00005c', '0xca53bb', '0x0'], 93: ['0xfe00005d', '0xcce4ef', '0x0'], 94: ['0xfe00005e', '0xcd4f21', '0x0'], 95: ['0xfe00005f', '0xcf7623', '0x0'], 96: ['0xfe000060', '0xd20757', '0x0'], 97: ['0xfe000061', '0xd4988a', '0x0'], 98: ['0xfe000062', '0xd729be', '0x0'], 99: ['0xfe000063', '0xd9baf2', '0x0'], 100: ['0xfe000064', '0xdc383d', '0x0'], 1: ['0xfe000001', '0x1790', '0x0'], 2: ['0xfe000002', '0x1e13b', '0x0'], 3: ['0xfe000003', '0x4726d', '0x0']}
```

c. Metrics of Timestamp Platform

These test were done to simulate real random trigger signals from the detector electronics as shown in Figure 5.17. Two uncorrelated pulsars at low rates were used to evaluate the timestamp platform.

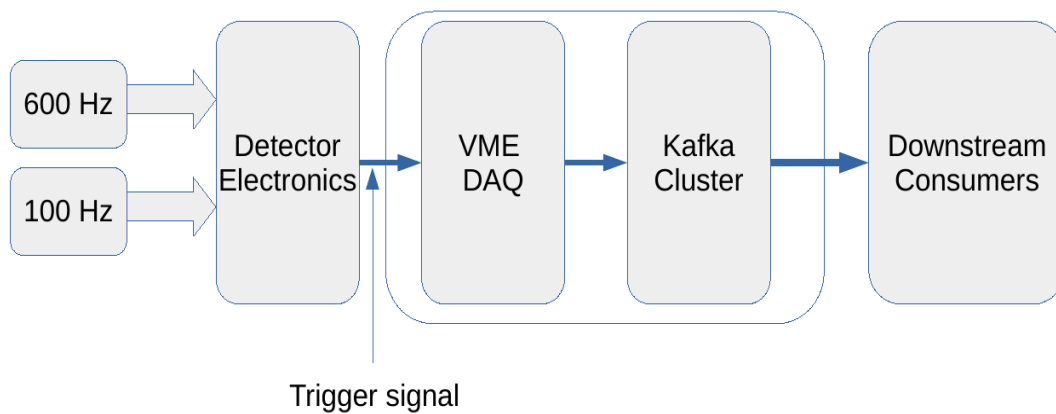


Figure 5.17: Times stamp test configuration

This test configuration will only focus on single event readout because that is the how the readout for DAQ is currently configured. As provided in Appendix E, multiple test runs were done to evaluate the performance of the timestamp in coincidence with the QDC. As seen in Figure 5.18 using the uncorrelated pulsars the biggest value readout between consecutive events approached the period of 600Hz (events 673 and 674) and smaller values that was read out (events 3120 and 3121) approached 218us or 4587Hz as can be seen in Figure 5.19.

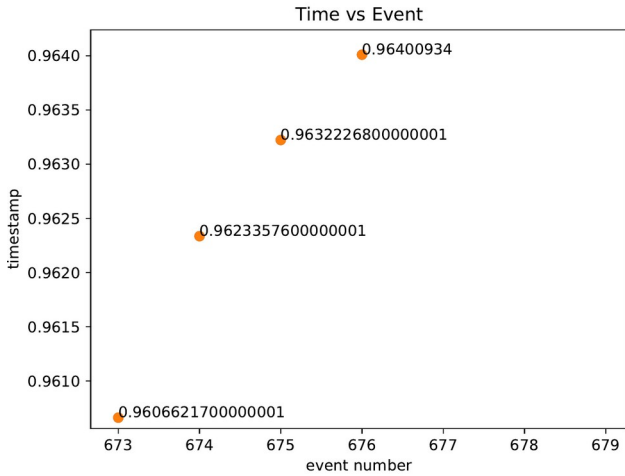


Figure 5.19: Plot showing biggest values between consecutive events

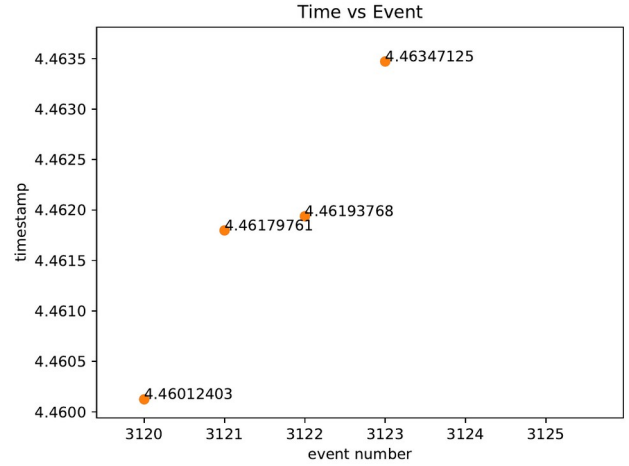


Figure 5.18: Plot showing small values readout between consecutive events

During these empirical tests it was found for single event readout of the timestamp, using the V1495 FPGA module, that on random intervals it had loss of timestamp events as seen in table 5.5. However the new developed event builder was able to recover from the data loss using the recovery protocol described in section 3.6.2.1 but it resulted in a loss of time-series data of >1% at low trigger rate.

Table 5.5: Measurements of timestamp

Run number	Wirechamber events	Event-builder events	Loss %
1	9789	9774	0,15
2	6183	6172	0,18
3	6083	6078	0,18

Figure 5.20 show a histogram plot of of a test run with 6083 events to identify the maximum rate at which the timestamp could operate and what effects the missed events has on the readout. This histogram plots the time difference between two consecutive events that were calculated using the formula: ***timestamp(x+1) - timestamp(x)*** and this was done for all x events, where x is the total number of events in a run.

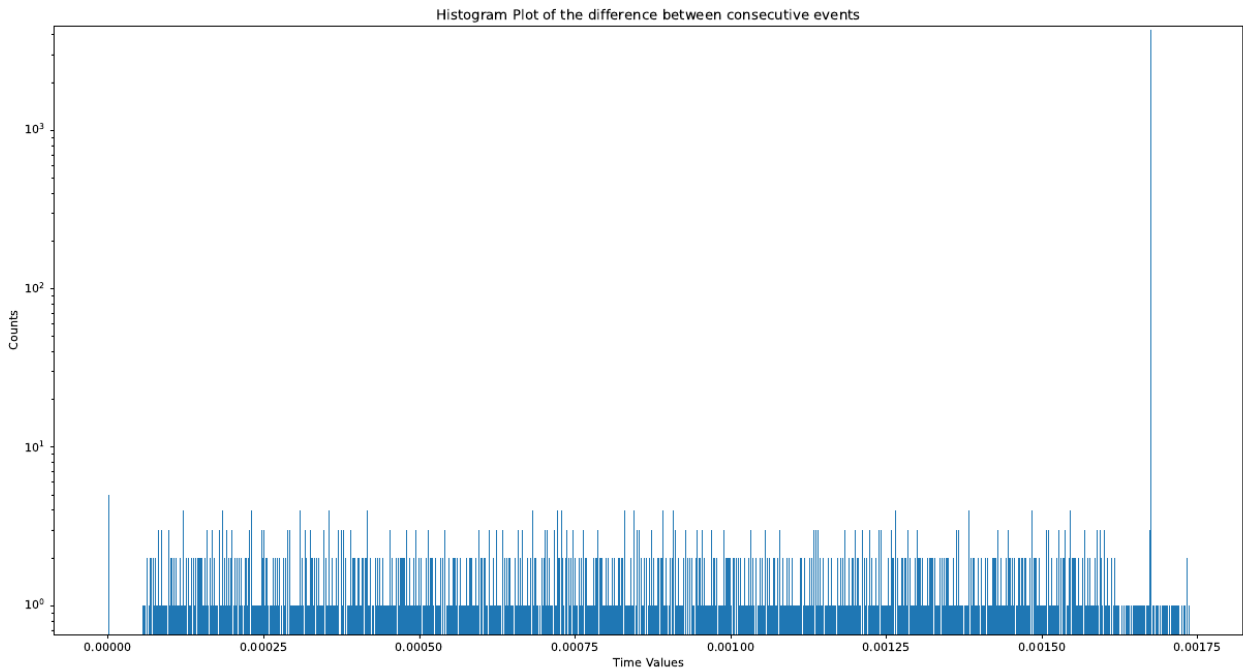


Figure 5.20: Histogram plot of time difference between consecutive events

The results show that the minimum time between 2 consecutive components are approximately 55 microseconds and the maximum time is 1.735 milliseconds. The strong component sits at 1.669 msec, which represents a trigger rate of 599 Hz, because that is the frequency one of the input pulsars for this run which was set at ~600 Hz. The readout time of 1.735 msec which is 576 Hz is due to the fact that there were 5 events with a zero time difference. These are the "skipped" events that has a time difference of 0 seconds.

Investigating the origin of the missed events, forced Veto signals of 70us and 400us (as provided in Appendix E) from the detector electronics was employed to determine if the data transfer between User FPGA, VME-bridge FPGA, and readout software was slower than expected.

Figure 5.21 shows the plot representing the difference between consecutive events with a forced veto signal of 70us. This was achieved by setting up the detector electronics to create a veto signal so no other events can be received by the DAQ before the 70us has expired. The results show that for a run of 8312 events using 70us Veto signal that there was only one missed event.

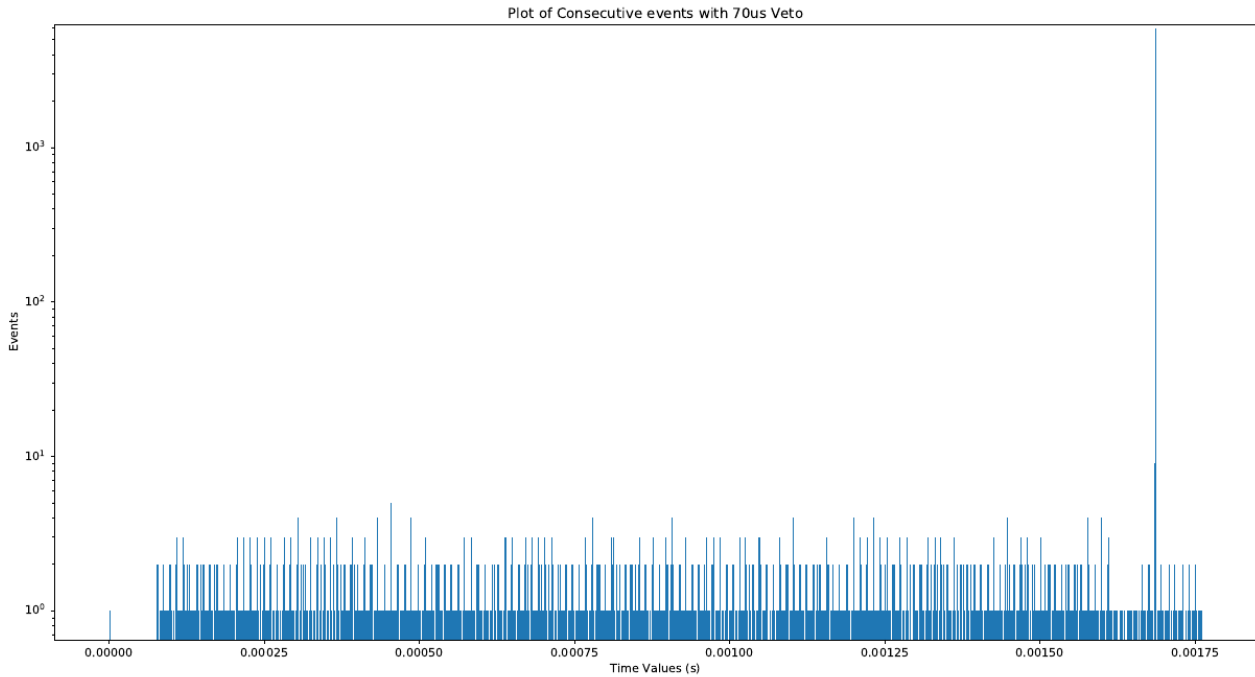


Figure 5.21: Histogram plot of time difference between consecutive events with 70us veto signal

The sample readout and histogram plot from the timestamp using a 400us veto signal is shown in Figure 5.22. The strong component sits at 599 Hz for a run of 9127 events using a 400us Veto signal, and there were no loss of any time events, indicating that data transmission between the V1495 VMEbus User FPGA and VME-Bridge FPGA was slower than expected affecting both single and multi event readout.

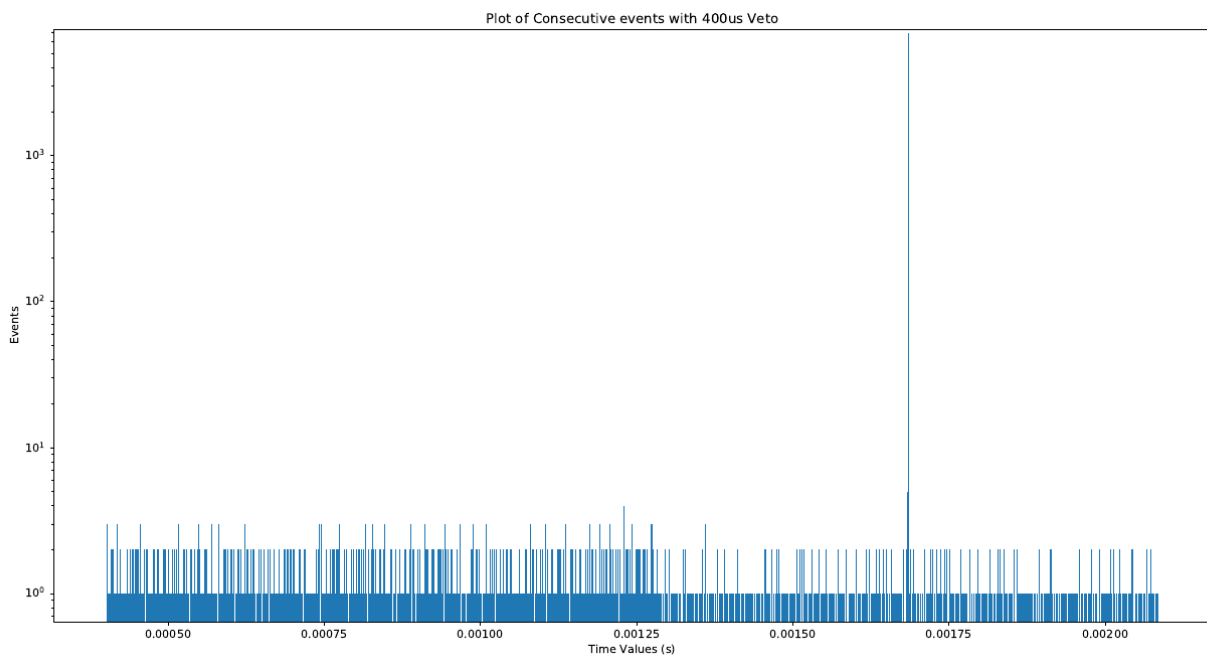


Figure 22: Histogram plot of time difference between consecutive events using 400us veto signal

Multiple tests was run on the system using the V1495 FPGA and timestamps were captured. The timestamp platform was found to operate according to the functional requirements. However the timestamp can only be used at low trigger rates using two uncorrelated pulsars if no veto signal is used as shown by the results in Figure 5.20.

5.6 Conclusion

The new readout software system with timestamp has been evaluated in terms of its ability to readout of event data and generate timestamps in response to trigger input signals. The timestamp is accurately captured but it was found that readout of the timestamp was slower than expected at higher trigger rates.

The event data was read out successfully and data was visualized using a modified existing Analyzer.

CHAPTER 6

Conclusion and Discussion of Results

The prototype version of the new K600 data acquisition system with timestamp for iThemba LABS has been designed, implemented, and tested with custom-made testing scripts that represent the complete data acquisition system. A prototype version of readout software was tested using two uncorrelated pulsars and the requirements were met in terms of functionality or reading out events on an event-by-event basis and reading out buffered events from the VMEbus modules.

The Dolosse DAQ Architecture developed to meet dataflow requirements for physics. Data Acquisition systems was successfully implemented and extended as the readout system for the K600 DAQ which is based on VMEbus. The implemented system has been shown to meet the operational requirements of data acquisition and using the timestamping platform, events were captured with timestamps added to the event string using the CAEN V1495 FPGA module with internal 100 MHz clock.

The timestamp firmware was successfully developed on the VME V1495 FPGA general-purpose board. The firmware implements a timestamp when a trigger is received from the detector electronics which can either be written to the platform's multi-event buffer when configured in multi-event buffer-mode, or be read out via VMEbus when configured for single event readout.

Communication between front-end electronics and the other subsystems using JSON messages through Kafka was used to configure the VMEbus modules with experimental parameters to ensure that the correct modules, number of modules, and number of events that should readout are set for the experiment. Also, Kafka has proved to provide a robust method of communication for transmitting run control information to and from the ROC.

The new distributed readout software system will be the first step in the development of the K600 DAQ. The new modular design of the DAQ software that is based on open source streaming technologies serves as a universal platform to be used for any future developments of the DAQ. This allows that in different scenarios using the technique of creating plugins new functions can be added by only applying minimal changes to components of the DAQ.

6.1 Future Work

Due to the buffered readout of data via VME being so slow, that method of readout appears to not be useful for use with the DAQ in its current configuration. If the readout time could be reduced it will result in shorter readout delays. This will increase the readout rate and assist in more accurate readout results. Because currently the trigger veto as shown in 5.2.2.2 is only 6 μ s (13 μ s when ADC included in the readout chain) and the total readout time as discussed in 5.4.2 is >1ms for multi-event readout, it can cause event misalignment between modules due to the veto being released before data was readout from all the VMEbus modules. Creating a trigger inhibit circuit with the V1495 FPGA while the modules are being read out will insure that no other triggers are generated by the detector electronics while data is being read out and produced to the Kafka cluster. This will assist in improving a more accurate readout of data and reading data at a faster trigger rate.

Results show that captured timestamps from the V1495 FPGA module, operated according the functional requirements for single event readout. It however didn't meet the operational requirements because empirically measured timestamp losses were higher than expected at low trigger rates. If one were to consider MEB readout and using the timestamp platform at higher trigger rates, then the Firmware of the timestamp platform would have to be carefully scrutinised to facilitate the increased trigger rate and further research into the low trigger rate and MEB readout is required. The recommended inhibit circuit will assist with the timestamp readout, because it will Veto any incoming events until all modules has been read out.

BIBLIOGRAPHY

- González, V., Barrientos, D., Blasco, J.M., Carrió, F., Egea, X. and Sanchis, E., 2012. Data Acquisition in Particle Physics Experiments. In *Data Acquisition Applications*. [online] IntechOpen. Available at <https://www.intechopen.com/books/data-acquisition-applications/data-acquisition-in-particle-physics-experiments> [8 August 2020]
- Emilio, M. 2013. *Data Acquisition Systems From Fundamentals to Applied Design*. London: Springer
- De Robertis G., 2018 *Data Acquisition is Prarticle Physics Experiments*. [online] INFN Bari. available at <https://agenda.infn.it/event/15138/contributions/28606/attachments/20405/23148/DAQ.pdf> [20 August 2020]
- iThemba LABS. n.d *Subatomic Physics - K=600 Magnetic Spectrometer* <https://tlabs.ac.za/subatomic-physics/k600-magnetic-spectrometer/> [18 August 2020]
- Joos, M., 2010. Introduction to VMEbus. [online] available at <https://indico.cern.ch/event/68278/contributions/1234555/attachments/1024465/1458672/VMEbus.pdf> [18 August 2020]
- VITA, 1995. American National Standard for VME64. [online] available at: <https://www.ge.infn.it/~musico/Vme/Vme64.pdf> [18 August 2020]
- Wielers M., 2015. Introduction to Trigger and Data Acquisition. [online] available at https://indico.cern.ch/event/518474/contributions/1198683/attachments/1268107/1878054/TDAQ_RAL_Lecture_040516.pdf [21 August 2020]
- MIDAS n.d *MIDAS Documentation* https://midas.triumf.ca/MidasWiki/index.php/Midas_documentation [1 September 2020]
- Dolosse n.d. *Modernizing Nuclear Physics Data Processing* <https://dolosse.org/modernizing-nuclear-physics-data-processing/> [24 August 2020]
- Kafka n.d.-a *Introduction, Everything you need to know about Kafka in 10 minutes* <https://kafka.apache.org/intro> [2 September 2020]
- Tlou, H., 2020, December. Implementation of the DAQ software in the ALTI Module of the ATLAS TileCal. In *Journal of Physics: Conference Series* (Vol. 1690, No. 1, p. 012055). IOP Publishing. [online] available at <https://iopscience.iop.org/article/10.1088/1742-6596/1690/1/012055/pdf> [27 October 2021]
- Christian, G., Akers, C., Connolly, D., Fallis, J., Hutcheon, D., Olchanski, K. and Ruiz, C., 2014. Design and commissioning of a timestamp-based data acquisition system for the DRAGON recoil mass separator. *The European Physical Journal A*, 50(4), pp.1-11. [online] Available at https://www.researchgate.net/publication/260836515_Design_and_commissioning_of_a_timestamp-based_data_acquisition_system_for_the_DRAGON_recoil_mass_separator [September 2020]
- Palit, R., Saha, S., Sethi, J., Trivedi, T., Sharma, S., Naidu, B.S., Jadhav, S., Donthi, R., Chavan, P.B., Tan, H. and Hennig, W., 2012. A high speed digital data acquisition system for the Indian National Gamma Array at Tata Institute of Fundamental Research. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 680, pp.90-96.

[online] Available at https://www.researchgate.net/publication/224008018_A_high_speed_digital_data_acquisition_system_for_the_Indian_National_Gamma_Array_at_Tata_Institute_of_Fundamental_Research [September 2020]

Ahmed, S. N. (2007) Physics and Engineering of Radiation Detection. Academic Press, Elsevier.

U.S. Department of Energy Office of Scientific and Technical Information n.d. *Particle and nuclear physics instrumentation and its broad connections* <https://www.osti.gov/pages/servlets/purl/1346724> [13/02/2021]

van Steen, M., Tanenbaum, AS., (2016) A brief introduction to distributed systems [online] <https://link.springer.com/article/10.1007/s00607-016-0508-7> [14/02/2021]

Real Python n.d. *Dictionaries in Python* <https://realpython.com/python-dicts/> [5 June 2021]

Confluent n.d. *Benchmarking Apache Kafka, Apache Pulsar, and RabbitMQ* <https://www.confluent.io/blog/kafka-fastest-messaging-system/> [5 September 2020]

Kafka n.d.-b *Class KafkaProducer* <https://kafka.apache.org/20/javadoc/org/apache/kafka/clients/producer/KafkaProducer.html> [7 March 2021]

CAEN, 2010a. V1495 USER Demo - Pattern Recorder. [Online] available at: <https://www.caen.it/download/?filter=V1495#> [8 March 2021]

intel n.d Cyclone Family datasheet. [online] available at: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ds/ds_cyc.pdf [08 March 2021]

CAEN tools for discovery n.d. *A395x Piggyback boards for Vx495 and DT5495* <https://www.caen.it/products/a395/> [9 March 2021]

CAEN, 2010b. V792 Technical Information Manual. [online] available at: <https://www.caen.it/?downloadfile=4021> [9 March 2021]

CAEN, 2012a. V785 Technical Information Manual <https://www.caen.it/?downloadfile=3976> [9 March 2021]

CAEN, 2012b. V1190A/B-2eSST 128 Channel Multihit TDC Technical Information Manual. [online] available at: <https://www.caen.it/?downloadzip=4024-4660> [9 March 2021]

CAEN, 2007. V830 32 Channel Latching Scaler Technical Information Manual. [Online] available at: <https://www.caen.it/?downloadfile=4019> [9 March 2021]

Neveling, R., Smit, F.D., Fujita, H. & Newman, R.T. 2008. K600 user manual, iThemba LABS, Somerset west. Unpublished.

CAEN, 2019. V1495 Technical Information Manual. [online] available at: <https://www.caen.it/?downloadfile=4962> [10 March 2021]

IDT, 2010. Universe IID/IIB User Manual <http://www1.futureelectronics.com/doc/Renesas/CA91C142D-33IE.pdf> [10 September 2020]

Istituto Nazionale di Fisica Nucleare, 2020. Introduction to VME. [online] available at: https://wwwusers.ts.infn.it/~rui/univ/Acquisizione_Dati/Lezioni/IX%20-%20VME%20Standard%20-%20Introduction/lezione_vme.pdf [9 September 2020]

Concurrent Technologies n.d. *CD LNX/BSn - Linux Board Support Package*
<https://www.gocct.com/product/linux-board-support-package/> [6 August 2020]

librdkafka n.d. *The Apache Kafka C/C++ client library*
<https://docs.confluent.io/platform/current/clients/librdkafka/html/index.html> [6 August 2020]

Adsley, P., Neveling, R., Papka, P., Dyers, Z., Brümmer, J.W., Diget, C.A., Hubbard, N.J., Li, K.C.W., Long, A., Marin-Lambarri, D.J. and Pellegrini, L., 2017. CAKE: the coincidence array for K600 experiments. *Journal of Instrumentation*, 12(02), p.T02004.

Pool, L., Neveling, R., Adsley, P., Pesudo, V. 2018. K600 Data-Acquisition Manual, iThemba LABS, Somerset west. Unpublished.

RootAna n.d. *MIDAS Documentation*
<https://midas.triumf.ca/MidasWiki/index.php/ROOTANA> [10 January 2021]

clock_gettime (2) - Linux Man Pages n.d. *SysTutorials*
https://www.systutorials.com/docs/linux/man/2-clock_gettime/ [10 January 2021]

Narkhede N., Shapira G., Palino T. 2017. *Kafka: The Definitive Guide*. Sebastopol: O'Reilly Media

JsonCpp n.d. *JsonCpp* <https://en.wikibooks.org/wiki/JsonCpp> [6 August 2020]

Nadareishvili, I., Mitra, R., McLarty, M. and Amundsen, M., 2016. *Microservice architecture: aligning principles, practices, and culture*. " O'Reilly Media, Inc."

APPENDIX A: VME bus specification

A.1 VME Standard Background

Versa Module Europa (VME) bus is a flexible open-ended computer bus system that is physically based on the Eurocard standard² (VITA, 1995). The VME bus specification was introduced by Motorola, Phillips, Thompson, and Mostek in 1981, and is defined by the IEEE 1014-1987 standard. VME bus architecture is a Multi-Master, parallel, asynchronous and TTL-based bus (VITA, 1995; Emilio, 2013: 95). The bus was developed to be a flexible environment supporting computing intensive tasks, and to assist scientists and Engineers standardize the protocol in the computer industry. VME card crates contain 21 slots, the first position in the crate must be reserved as a system controller or readout driver (González et al., 2012 ; Emilio, 2013: 95).

A.1 VME Data transmission

The VMEBus is made up of four different buses as seen in Figure A.1. These buses are Data Transfer Bus (DTB), the Data Transfer Arbitration Bus (DTAB), Priority Interrupt Bus (PIB) and Utility Bus (UB).

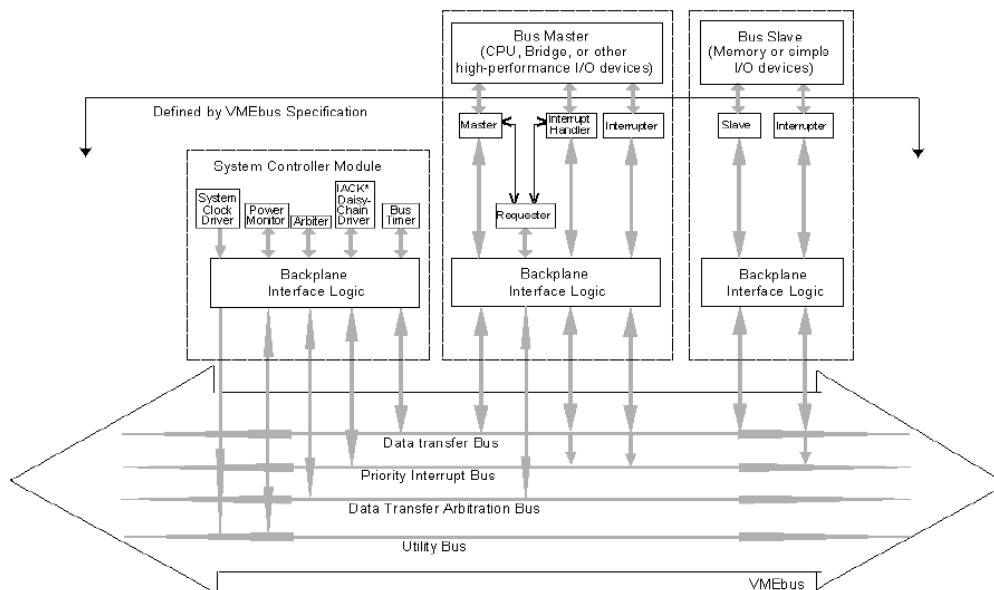


Figure A.1: VMEbus elements

The function of each bus is as follow:

DTB - The data transfer bus consist of the signal lines; 32 data line (D0 - D31) which contains the actual data during data transfer, 32 address lines (A1 - A32) are monitored by the slave module, the address strobe (AS) which is driven by the master

to indicate there is a valid address on the bus, the Address Modifiers (AM0 - AM5) indicates the kind of data cycle and length of the address, WRITE line that is used to indicate if the current cycle is read or write, data strobe lines (DS0, DS1) that indicates valid data and the size of transfer, data transfer acknowledge, (DTACK) is driven by the slave device to indicate the data transfer was completed and Bus ERRor (BERR) that indicates an error on the bus or timeout (VITA, 1995; Emilio, 2013: 96).

DTAB - The bus master uses the DTAB to write the busy signal (BBSY) by driving it low to indicate the bus is in use. If the BBSY is not low the arbiter in the System Controller determines which module will be granted bus access (VITA, 1995).

PIB - The priority interrupt bus contains the interrupt signal lines (IRQ7 - IRQ1) for handling all interrupt where IRQ7 has the highest priority, the daisy chain controller Interrupt Acknowledge (IACK) which initiates the activity on the Interrupt Acknowledge IN (IACKIN) and Interrupt Acknowledge OUT (IACKOUT) during an IACK cycle (VITA, 1995; Emilio, 2013: 96-97).

UB - The utility bus is a collection of signals that monitors the power pins (5V, +12V and -12V), System Reset (SYSRESET) used to reset the VMEbus, System Clock (SYSCLK), System Failure (SYSFAIL) a utility that is used for system diagnostics (VITA, 1995).

APPENDIX B: Time stamp Register Interface

VME Address	Name	Description	Size	Type R/W
Base+0x100C	REG_R6	Firmware release	32-bit	R
Base+0x1018	REG_RW3	Config. Register	32-bit	R/W
Base+0x1030	REG_R1	Nanosecond counter	32-bit	R
Base+0x1034	REG_R2	Second Counter	32-bit	R
Base+0x1038	REG_R3	Event Counter	32-bit	R
Base+0x1042	REG_R4	Output Clock Status	32-bit	R
Base+0x0000-0x0FFC	n/a	USER FPGA Block transfer	32-bit	R

APPENDIX C: V1495_timestamp entity

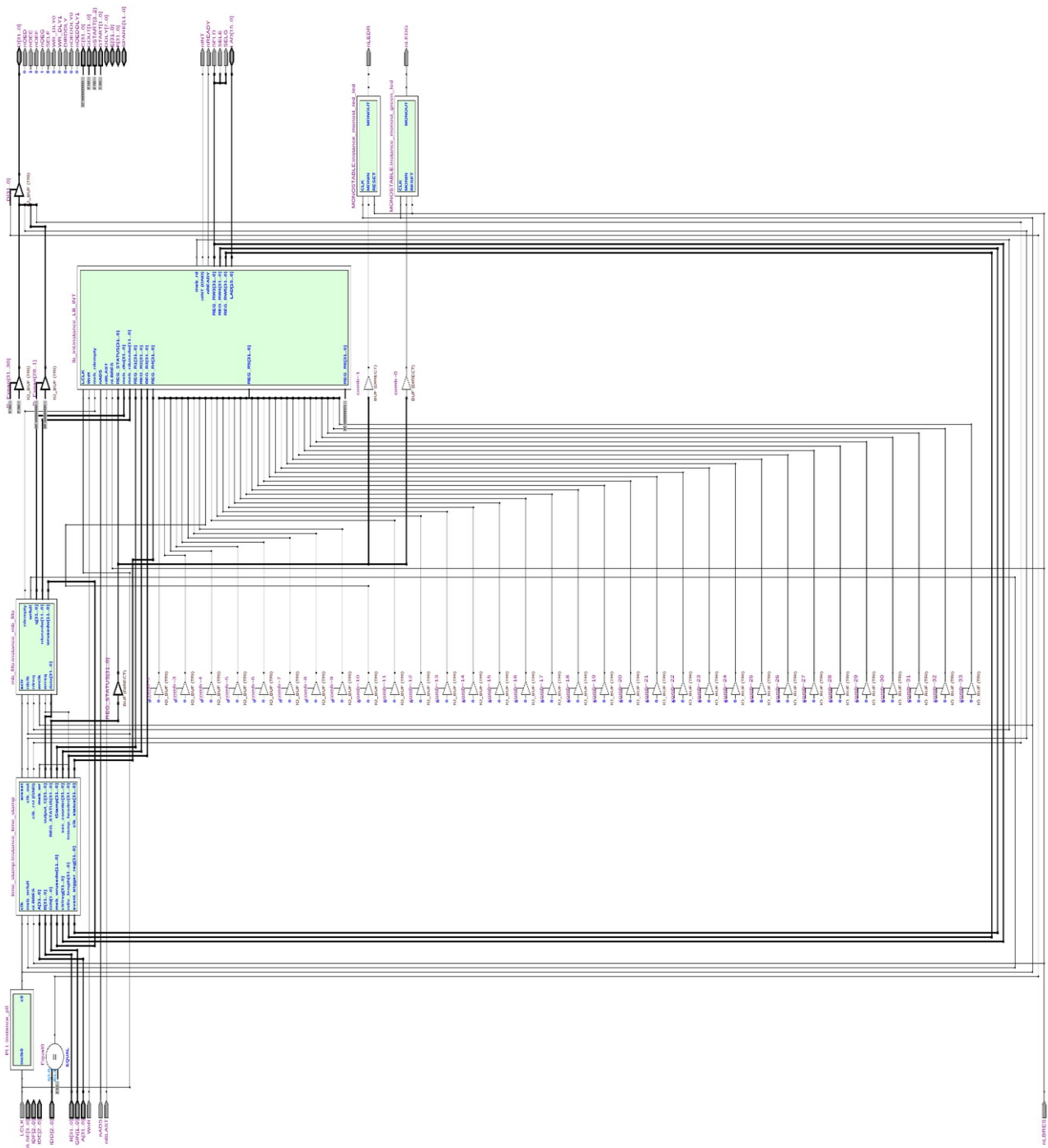
Port	Width	Description	I/O
Port A	32-bit	Input section of on the front panel of the V1495 board	Input
Port B	32-bit	Input section of on the front panel of the V1495 board	Input
Port C	32-bit	Output section of on the front panel of the V1495 board	output
Port D	32-bit	Bi-directional expansion section of on the front panel of the V1495 board	Input/Output
Port E	32-bit	Bi-directional expansion section of on the front panel of the V1495 board	Input/Output
Port F	32-bit	Bi-directional expansion section of on the front panel of the V1495 board	Input/Output
Port GIN	2-bit	Input section of on the front panel of the V1495 board	Input
Port GOUT	2-bit	Output section of on the front panel of the V1495 board	Output

IDD	3-bit	Expansion board select: 000: A395A; 001: A395B 010: A395C; 011: A395D	Input
IDE	3-bit	Expansion board select: 000: A395A; 001: A395B 010: A395C; 011: A395D	Input
IDF	3-bit	N/A Not used	N/A

nOED	1-bit	Output/Input enable of Port D: 0= Output, 1= Input	Output
nOEE	1-bit	Output/Input enable of Port D: 0= Output, 1= Input	Output
nOEF	1-bit	Output/Input enable of Port D: 0= Output, 1= Input	Output
nOEG	1-bit	Output/Input enable of Port D: 0= Output, 1= Input	Output

SELD	1-bit	Level select Port D: 0= NIM, 1= TTL	Output
SELE	1-bit	Level select Port D: 0= NIM, 1= TTL	Output
SELF	1-bit	Level select Port D: 0= NIM, 1= TTL	Output
SELG	1-bit	Level select Port D: 0= NIM, 1= TTL	Output

APPENDIX D: Firmware and Block diagram of Timestamp



```
--
=====
=====
```

```
-- *****
```

```
-- Company:    CAEN SpA - Viareggio - Italy
-- Model:      V1495 - Multipurpose Programmable Trigger Unit
-- Device:     ALTERA EP1C4F400C6
-- Author:     Eltion Begteshi, Bertram Losper
-- Date:       April 13th, 2009
-- modified:   August, 2021
```

```
-----
```

```
-- Module:     time_stamp
-- Description: data producer
```

```
-- *****
```

```
library ieee;
use IEEE.Std_Logic_1164.all;
use IEEE.Std_Logic_arith.all;
use IEEE.Std_Logic_unsigned.all;
```

```
ENTITY time_stamp is
```

```
port(
```

```
    clk : in std_logic; -- time-stamp clock input
    A : in std_logic_vector(31 downto 0); -- in a (32 x lvds/ecl)
    B : in std_logic_vector(31 downto 0); -- in b (32 x lvds/ecl)
    output_C : out std_logic_vector(31 downto 0);
    GIN : in std_logic_vector( 1 DOWNTO 0); -- In G - LEMO (2 x NIM/TTL)
    nLBRES : in std_logic;
    areset : buffer std_logic;
    meb_wrfull : in std_logic;
    meb_wr : out std_logic;
    meb_wrusedw : in std_logic_vector(11 downto 0);

    ctrlreg : in std_logic_vector(31 downto 0);
    ndiv_length : in std_logic_vector(31 downto 0);
    event_trigger_reg : in std_logic_vector(31 downto 0);
```

```

REG_STATUS : out std_logic_vector(31 downto 0);

--time stamp specific variables
tStamp: out std_logic_vector(31 downto 0);
sec_counter: out std_logic_vector(31 downto 0);
tstamp_header : out std_logic_vector(31 downto 0);

clk_status : out std_logic_vector(31 downto 0); -- clock status
clk_out : out std_logic;
in_trig2 : in std_logic; -- PORT E7 alternative input for trigger
in_trig : in std_logic -- PORT E0 alternative input for trigger

);

END time_stamp ;

ARCHITECTURE RTL of time_stamp is

type ACQ_STATE_TYPE is (IDLE, HDR_WR, NS_WR, SC_WR, WAIT_END, TRIG_WAIT, TRIG_SET); --
states of state machine
signal STATE : ACQ_STATE_TYPE := IDLE;

alias start_g1_in : std_logic is GIN(0); -- change trigger input to G0
signal latch_time_stamp : std_logic_vector(31 downto 0):=(others => '0');
signal time_stamp_counter : std_logic_vector(31 downto 0) := (others => '0');
signal sync_counter : std_logic_vector(31 downto 0):=(others => '0');

-- 1 second counter variables
signal sec_timer : std_logic_vector(31 downto 0):=(others => '0');
signal latch_sec_timer : std_logic_vector(31 downto 0):=(others => '0');

--event counter
signal evt_counter : std_logic_vector(31 downto 0):=(others => '0'); -- 24 bit event counter

```

```

signal evt_header : std_logic_vector(31 downto 0):=(others => '0'); -- 24 bit event counter
signal header : std_logic_vector(31 downto 0):=(others => '0'); -- 24 bit event counter

signal reset_start : std_logic;
signal areset_1 : std_logic;

alias soft_clear : std_logic is ctrlreg(8);
alias time_stamp_enable : std_logic is ctrlreg(5); --
alias read_mode : std_logic is ctrlreg(7); -- check what readout mode is being done
alias sync_clk: std_logic is ctrlreg(11);
alias ndiv : std_logic_vector( 3 downto 0) is ndiv_length( 3 downto 0);;
alias blt_ready : std_logic is REG_STATUS(0);
alias nLEDG : std_logic is REG_STATUS(1);
alias nLEDR : std_logic is REG_STATUS(2);
signal ext_clear : std_logic;
signal ext_clear_probe : std_logic;

signal clk_div : unsigned(4 downto 0);

signal evt_wr_cnt : std_logic_vector(31 downto 0):=(others => '0');
signal state_tr : std_logic_vector(31 downto 0):=(others => '0'); -- state transistion
signal trig_reset : std_logic := '0';
begin
    areset <= soft_clear or ext_clear ;
    ext_clear_probe <= ext_clear;

    -- Updating reset
    ext_clear <= ctrlreg(31);
    areset_1 <= areset or reset_start;

    --Sync signal
    sync_sig: process(areset, clk)
    begin
        if areset = '1' then

```

```

        sync_counter <=(others => '0'); -- clear nanosecond counter reset
    elsif clk'event and clk = '1' then

        if (sync_clk = '1') then -- if bit 11 is set and bit 12 is cleared
            if(time_stamp_enable = '1')then -- if enable is set in SW
                sync_counter <= sync_counter + '1';
                if(sync_counter >= conv_std_logic_vector(65535, 32))then
                    --set sync_signal
                    sync_counter <= (others => '0'); -- reset the sync. counter
                    clk_out <= '1';

                else
                    --clear sync_signal
                    clk_out <= '0';
                end if; -- timestamp counter
            end if; -- timestmap enable
        end if; -- if contrlreg
    end if;-- clk event

end process sync_sig;

--counters
counters: process(clk, areset)
begin
    if areset = '1' then
        sec_timer <=(others => '0'); -- clear second counter reset
        time_stamp_counter <=(others => '0');
    elsif clk'event and clk = '1' then
        if(time_stamp_enable = '1') then
            time_stamp_counter <= time_stamp_counter + '1';
            if(time_stamp_counter >= conv_std_logic_vector(100000000, 32))then
                sec_timer <= sec_timer + '1';
                time_stamp_counter <= (others => '0');

            end if; -- timestamp counter
        end if; -- timestmap enable
    end if;
end process;

```

```

        end if;-- clk event

    end process counters

-- event counter latch
evt_count: process(areset, start_g1_in, clk)
begin
    if areset = '1' then
        evt_counter <= x"FE000000";
    elsif rising_edge(start_g1_in) then
        if(time_stamp_enable = '1' )then --
            evt_counter <= evt_counter + '1';
        end if; -- start_g1
    end if;
end process evt_count;

-- data latch
data_latch: process(start_g1_in, areset, clk) --
begin
    if areset = '1' then
        latch_time_stamp <=(others => '0'
        latch_sec_timer <= (others => '0');
    elsif falling_edge(start_g1_in) then
        if(time_stamp_enable = '1')then --
            latch_time_stamp <= time_stamp_counter;
            latch_sec_timer <= sec_timer;

        end if; -- timestmap enable
    end if;-- event
end process data_latch;

--sync processs
trigger_sync3: process(start_g1_in, areset, trig_reset) --
begin

```

```

if areset = '1' then
    state_tr <= (others => '0');
elsif trig_reset = '1' then
    state_tr(0) <= '0';
elsif rising_edge(start_g1_in) then -- = '1' then
    if(read_mode = '1') then -- if blt mode
        state_tr(0) <= '1';
    end if;
end if;
end process trigger_sync3;

--write out timestamp
tstamp_header <= evt_counter; -- populate event counter
tStamp <= latch_time_stamp; -- load nanosecond count;
sec_counter <= latch_sec_timer; -- load second count

-- FSM
fsm_process: process(clk, areset)
begin
    if (areset = '1') then
        STATE <= IDLE;
        meb_wr <= '0';
        nLEDG <= '1'; -- GREEN OFF
        nLEDR <= '0'; -- RED ON
        reset_start <= '1';
        trig_reset <= '0';
        evt_wr_cnt <= (others => '0');

    elsif(clk'event and clk = '1') then -- rising edge of clock input
        meb_wr <= '0';
        case STATE is
            when IDLE =>
                reset_start <= '0';
                nLEDG <= '0'; -- GREEN ON

```

```

nLEDR <= '1'; -- RED OFF
blt_ready <= '0';

if(state_tr(0) = '1')then
    if(read_mode = '1') then
        trig_reset <= '1';
        STATE <= HDR_WR; -- MEB including Header
    end if;
end if; -- meb_wrout

when HDR_WR => --write header
    output_C <= evt_counter; -- populate event counter;
    nLEDG <= '0'; -- GREEN ON
    nLEDR <= '0'; -- RED ON

    meb_wr <= '1'; -- enable meb write
    STATE <= NS_WR;

when NS_WR => --write nanosecond
    output_C <= latch_time_stamp; -- load nanosecond count
    nLEDG <= '0'; -- GREEN ON
    nLEDR <= '0'; -- RED ON

    meb_wr <= '1'; -- enable meb write
    STATE <= SC_WR;

when SC_WR => --write second
    output_C <= latch_sec_timer; -- load second count

    nLEDG <= '0'; -- GREEN ON
    nLEDR <= '0'; -- RED ON

    meb_wr <= '1';
    evt_wr_cnt <= evt_wr_cnt + '1';

    if(evt_wr_cnt >= evt_wr_cnt )then

```



```

        STATE <= WAIT_END;
        blt_ready <= '1';    -- start BLT request r
        evt_wr_cnt <= (others => '0');
    else
        STATE <= TRIG_SET;
    end if;

when TRIG_SET =>
    nLEDG <= '0'; -- GREEN ON
    nLEDR <= '0'; -- RED ON
    STATE <= TRIG_WAIT;

when TRIG_WAIT => -- wait for trigger to go idle
    nLEDG <= '0'; -- GREEN ON
    nLEDR <= '0'; -- RED ON

    if(start_g1_in = '1') then -- trigger go back to idle
        STATE <= IDLE;
        trig_reset <= '0';
    end if;

when WAIT_END => --wait for data to be readout
    nLEDG <= '1'; -- GREEN OFF
    nLEDR <= '1'; -- RED OFF

    if (meb_wrusedw = conv_std_logic_vector(0, 12)) then
        STATE <= IDLE;
        trig_reset <= '0'; -- clear trigger set
    end if;

end case;

end if;

end process fsm_process;

END RTL;

```

APPENDIX E: Resources used

This is included on a Flash/cloud drive which is submitted together with this dissertation.

