Cape Peninsula
University of Technology

IEC 61850-9-2 BASED SAMPLED VALUES AND IEC 61850-8-1 GOOSE MESSAGES MAPPING ON AN FPGA PLATFORM

by

ALEXANDER MANDLENKOSI NCUBE

Thesis submitted in fulfilment of the requirements for the degree

Master of Engineering: Electrical Engineering

in the Faculty of Engineering

at the Cape Peninsula University of Technology

Supervisor: Mr C Kriger

Co-supervisors: Prof P Petev

Bellville Campus

Date submitted: June 2016.

DECLARATION

I, Alexander Mandlenkosi Ncube, declare that the contents of this 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.

Signed	**Date**

ABSTRACT

Electricity substation monitoring and control systems have evolved over the years from simple systems capable of achieving minimalistic functions to autonomous, self-healing smart grid schemes (Farhangi, 2010). The migration of technology to networked smart grid systems was driven by the need for standardisation of communication networks, system configuration and also the reduction of system implementation costs and engineering time.

Before the introduction of a uniform communication standard, legacy (non-standardised) communication protocols, for example, the Distributed Network Protocol (DNP3) were used by Remote Terminal Units (RTUs) for information exchange (Luwaca, 2014). These communication protocols could not provide a standard naming convention or data semantics since the data/information was accessed using an address-based system. The implementation of automation systems based on legacy protocols and RTUs was expensive because of parallel copper wiring required to connect instrument transformers and circuit breakers to multiple RTUs for protection and monitoring functions (Iloh *et al.,* 2014).

Legacy systems refer to Supervisory Control and Data Acquisition (SCADA) systems implemented using RTUs and legacy communication protocols. Legacy systems tended to be vendor specific because devices from different vendors did not support the same communication protocol. These issues led to the introduction of the IEC 61850 standard. The IEC 61850 standard for “communication networks and systems in a substation” provides standardised naming convention, data semantics, standardised device configuration and also device interoperability and interchangeability in some instances. The IEC 61850 standard provides a solution to expensive parallel copper wiring and standardisation issues experienced with legacy protocols.

In as much as the introduction of the IEC 61850 standard addresses problems experienced with legacy system there is still a need to provide inexpensive access to IEC 61850-compliant devices and effective knowledge transfer to facilitate implementation of automation systems based on this standard. The development of an IEC 61850-compliant device requires a specialised skillset and financial investment for research and industrialisation therefore only a few vendors manufacture these devices resulting in an increase in production and manufacturing costs.

For this reason this research project develops VHDL modules for mapping IEC 61850-9-2 Sampled Value (SV) messages and IEC 61850-8-1 Generic Object Oriented Substation

Event (GOOSE) messages on a Field Programmable Gate Array (FPGA) platform. Sampled values are used for transmitting current and voltage transformer (CT and VT) measurements to protection devices while GOOSE messages exchange information/commands between primary equipment (CT, VT and circuit breaker) and protection devices over an Ethernet network known as the process bus.

The VHDL modules developed in this research project for mapping SV and GOOSE messages on an FPGA platform combined with an analogue signal interface device will produce an inexpensive Merging Unit prototype (MU) and GOOSE enabled remote monitoring node that can be used in substation automation systems. This research project provides laboratory test procedures for evaluating the developed MU prototype and GOOSE-enabled remote node and also discusses the obtained results. The structure of the sampled value messages published by the MU is evaluated using SVScout and Wireshark software applications and the results validate the implemented mapping VHDL module.

The accuracy of the sampled values from the developed MU under fault/abnormal conditions is investigated by making use of four case studies where faults are simulated using the CMC 256*plus* test set and injected into the developed MU. The sampled value messages are then captured using SVScout and the waveforms compared to the injected signals' waveform. The four case studies reported are the voltage-sag, voltage-swell, frequency variations and harmonic disturbances. The results show that the MU prototype publishes accurate signals even under fault conditions.

The GOOSE messages published by the GOOSE message-mapping VHDL module are evaluated using IEDScout and Wireshark software applications and the results obtained validate the implemented mapping VHDL module. The GOOSE client state machine implemented in the GOOSE message mapping VHDL module is evaluated to determine whether it follows the one defined in the IEC 61850-8-1 standard. A voltage-sag fault case study is conducted to evaluate the accuracy of the injected signal's root mean square value calculation implemented by the GOOSE message mapping VHDL module. The results show that this research project presents successful development of the GOOSE and the SV message mapping VHDL modules which meet the requirements of the IEC 61850 standard. This thesis document contains an extensive literature review, design flow charts, VHDL code snippets, hardware interface diagrams for the development and evaluation of the IEC 61850-9-2 SV and IEC 61850-8-1 GOOSE message mapping VHDL modules.

Key words: *GOOSE messages, Sampled Value messages, IEC 61850 standard, FPGA, CSAEMS,* VHDL modules *Substation Automation Systems, Merging Unit, legacy protocols, instrument transformers.*

ACKNOWLEDGEMENTS

I am indeed grateful to Mr C. Kriger, Prof. R. Tzoneva and Adj. Prof P Petev for providing great technical and administrative support.

I am grateful to Adeyemi Adewole, Marcus Mthunzi, Tafadzwa Shamu, Nontobeko Gulu and all my friends for their support throughout my study.

To those I have left out unknowingly please accept my gratitude and know that your assistance pushed me closer and closer towards the finish line.

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.

Alexander Mandlenkosi Ncube

Bellville, June 2016

DEDICATION

To my **Father** who taught me how to dream.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

APPENDICIES

LIST OF ABBREVIATIONS

1PPS	One Pulse Per Second
ACSI	Abstract Communication Service Interface
ADC	Analogue to Digital Converter
AFE	Analogue Front-End
APDU	Application Protocol Data Unit
APPID	Application ID
ASDU	Application Service Data Unit
ASN.1	Abstract Syntax Notation One
CDC	Common Data Class
CID	Configured IED Description
CSAEMS	Centre for Substation Automation and Energy Management Systems
CT	Current Transformer
DC	Direct Current
DUT	Device under Test
FC	Functional Constraint
FPGA	Field Programmable Gate Array
GOCB	GOOSE Control Block
GOOSE	Generic Object Oriented Substation Event
GPS	Global Positioning System
HMI	Human Machine Interface
ICD	IED Capability Description
IEC	International Electrotechnical Commission
IED	Intelligent Electronic Device
IEEE	Institute of Electrical and Electronic Engineers
LAN	Local Area Network
LD	Logical Device
LN	Logical Node
MAC	Media Access Controller
MII	Media Independent Interface
ms	Milli-Second(s)
MSB	Most Significant Bit
MU	Merging Unit
MV	Medium Voltage
OLTC	On-Load Tap Changer
OSI	Open System Interconnect

PC	Personal Computer
PD	Physical Device
PDU	Protocol Data Unit
PMOD	Peripheral Module Connector
PSL	Programmable Scheme Logic
RMS	Root Mean Square
RTDS	Real Time Digital Simulator
RTU	Remote Terminal Unit
s	Second(s)
SAS	Substation Automation System
SCADA	Supervisory Control and Data Acquisition
SCL	Substation Configuration Language
SCSM	Specific Communication Service Mapping
SPI	Serial Peripheral Interface
SV	Sampled Value(s)
SVCB	Sampled Value Control Block
svID	Sampled Value Identification
UCAIug	Utility Communications Architecture International Users Group
UML	Unified Mark-up Language
VLAN	Virtual Local Area Network
VT	Voltage Transformer
XML	eXtensible Mark-up Language

CHAPTER ONE
INTRODUCTION

1.1 Introduction

Substation monitoring and control is vital to ensure the provision of quality electricity to consumers and protection of infrastructure in case faults occur, to this end, Supervisory Control and Data Acquisition (SCADA) systems were introduced. SCADA systems introduced in the early 1990s used Remote Terminal Units (RTUs) and legacy communication protocols for information exchange and control (Luwaca, 2014). This information was exchanged between RTUs in the same sub-station or between a station controller and the control centre. Distributed Network Protocol (DNP3), MODBUS and the IEC 60870-5-101 are some of the non-standardised legacy protocols used for communication (Luwaca, 2014).

Legacy protocols were byte-based and the information was accessed using an address or tag, this meant that there were no semantics attached to a value/information transmitted. Legacy protocols lacked a standard naming convention of devices, functions and data. These legacy protocols were also based on physical communication media (RS-232/RS-485) only capable of exchanging less than 1 Mbps of information and this had an effect on data access and command execution time. The use of these protocols resulted in reliable communication networks only if all communicating devices supported the same protocol, this architecture often lead to costly single-vendor proprietary solutions which had zero to minimal interoperability and interchangeability prospects.

Since legacy communication protocols could not provide data semantics or a standardised naming convention, devices supporting different protocols, for example, IEC 60870-5-101 or DNP3 could not communicate with each other without a protocol translator. A protocol translator is a device which translates messages from one protocol to another to facilitate communication between devices supporting different protocols. The use of protocol translators increased the complexity and implementation costs of the SCADA system.

Another shortfall of conventional automation systems was the use of expensive parallel copper wiring between primary plant equipment and the protection relays as illustrated in Figure 1.1. Following this communication architecture, legacy SCADA systems tended to be complex and expensive to implement and maintain. Therefore, there was a need for a standard for communication networks and systems in substations for protection, monitoring, control and automation purposes thus the introduction of the IEC 61850 standard (Mackiewicz, 2011).

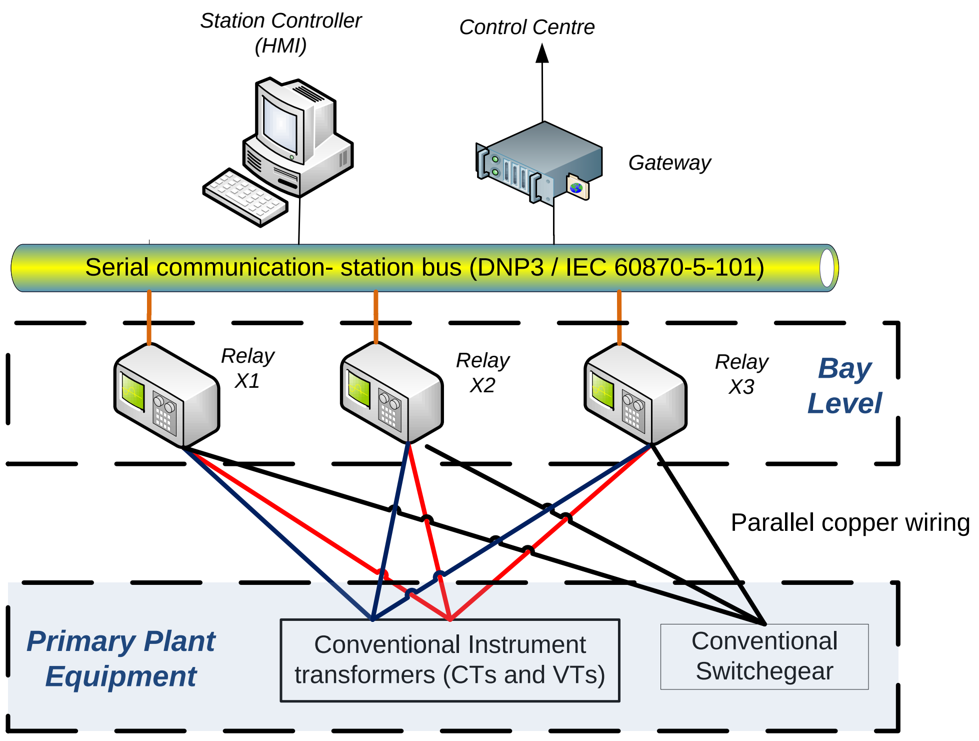


Figure 1.1: **Communication networks in conventional substation automation systems (Adapted from Iloh *et al.*, 2014)**

The first edition of the IEC 61850 standard was introduced in 2002 and has been designed to offer interoperability, standard naming convention, data semantics, standard design process, system setup procedures and interchangeability prospects. The IEC 61850 standard provides protocols for communication between different levels in the substation from process (primary plant equipment), bay level (protection relays) and station level devices.

The first edition of the IEC 61850 standard defines a communication interface between process equipment (circuit breakers, current and voltage transformers) and bay level Intelligent Electronic Devices (IEDs) known as the process bus. The IEC 61850 process bus presents a better way of exchanging primary plant information using new devices capable of making this information available over an Ethernet network. An example of such a device is the Merging Unit (MU) which serves as an interface between instrument transformers and bay-level IEDs thereby eliminating parallel copper wiring and reducing system implementation costs.

This research project centres on the development of two methods and VHDL modules for mapping sampled value and GOOSE messages conforming to the IEC 61850-9-2 and the IEC 61850-8-1 standards. The SV and GOOSE message-mapping VHDL modules are developed for the Xilinx Spartan 6 FPGA. This research project produces a cost-effective limited-function Merging Unit and GOOSE monitoring node prototypes.

The current research project is developed under the Centre for Substation Automation and Energy Management Systems (CSAEMS) at the Cape Peninsula University of Technology (CPUT).

1.2 Problem definition

The IEC 61850 standard was introduced to solve problems experienced with the use of legacy communication protocols in conventional substation automation systems. These problems include the lack of data semantics, standardised naming convention, interoperability between devices (IEDs) from different vendors and the use of expensive parallel copper wiring between primary plant equipment and the protection relays. According to Iloh *et al.,* (2014), the IEC 61850 standard has received substantial patronage from vendors and electrical supply utility operators over the years because of these technical capabilities.

According to the IEC 61850 standard, communication networks within a substation are composed of three layers; the station, bay and process levels (International Electrotechnical Commission, 2003-2004). Data is transmitted between devices residing in these layers through communication interfaces numbered 1 to 10 in Figure 1.2.

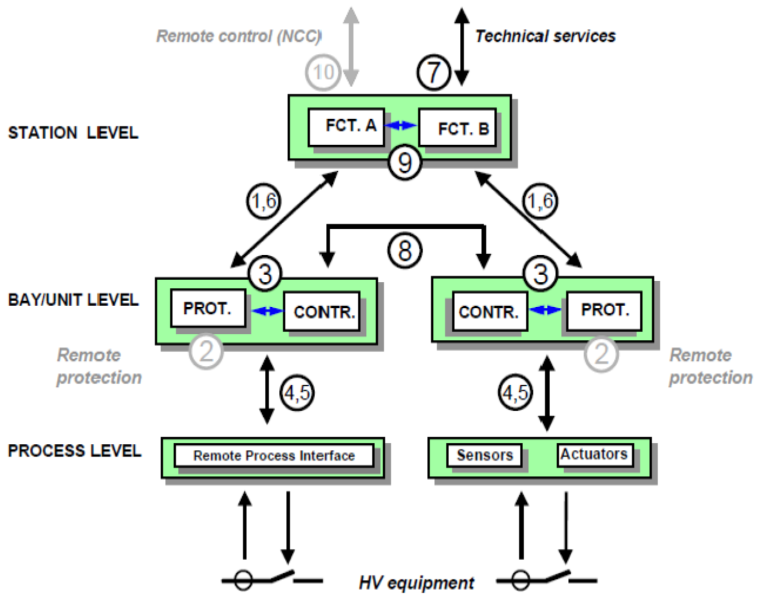


Figure 1.2: **Communication hierarchy in an IEC 61850 standard-based SAS (Adapted from International Electrotechnical Commission, 2003-2004)**

In Figure 1.2, interfaces 4 and 5 constitute the IEC 61850 process bus; these interfaces are used for the exchange of Sampled Values (SV) and Generic Object Oriented Substation Event (GOOSE) messages respectively. This process bus allows current, voltage, breaker status and other measured variables to be transmitted in real-time over an Ethernet network to bay level IEDs. These status and sampled values are made available on the process bus by Actuators and Merging Unit as shown in Figure 1.2.

Most actuators and Merging Units developed by different vendors are expensive because of the specialised skillset, monetary and time investments required when designing these IEC 61850 standard-compliant devices. Many utilities especially in less developed countries are still reluctant or unable to implement IEC 61850-based automation systems due to the lack of expertise or the high cost of IEC 61850 standard-compliant IEDs. Most academic institutions do not possess the expertise to train engineers who are able to design IEC 61850 standard-compliant devices or implement substation automation systems based on this standard (Luwaca, 2014; Retonda-Modiya, 2012).

This research project provides theory on the application of specific communication service mapping of substation data/information to GOOSE and SV messages, this thesis can be used for academic purposes or the development of certified IEC 61850 standard-compliant IEDs. This research project produces a low-cost GOOSE monitoring node and a limited-function Merging Unit prototype. These prototypes can be industrialised for use in IEC 61850 standard-based automation systems.

1.3 Research aim and objectives

1.3.1 Research aim

This research project aims to develop VHDL modules for mapping and publishing IEC 61850-9-2 SV and IEC 61850-8-1 GOOSE messages on an FPGA platform. These developed VHDL modules implemented on an FPGA will be combined with an Analogue Front-End (AFE) to produce a limited-function FPGA-based Merging Unit (MU) and a GOOSE monitoring node prototype. The developed MU and GOOSE monitoring node prototypes will be tested in the laboratory to validate their conformance to the IEC 61850-9-2 implementation guideline and the first edition of the IEC 61850-8-1 standard.

1.3.2 Research objectives

In order to achieve the aims of this research project the following objectives will be conducted:

- Literature review: History of substation automation systems and the introduction of the IEC 61850 standard.
- Literature review: To analyse the IEC 61850 standard data modeling techniques and communication service mapping for GOOSE and SV messages.
- To conduct a review of literature on software algorithms and VHDL models for mapping and publishing IEC 61850-9-1/IEC 61850-9-2 and IEC 61850-8-1 standard-based SV and GOOSE messages on embedded platforms.
- Methodology: To conduct a comparative analysis of different embedded platforms for mapping and publishing GOOSE and SV messages.
- Methodology: To design VHDL modules for mapping and publishing GOOSE and SV messages as specified in the IEC 61850-8-1 standard and

the Utility Communications Architecture International users group (UCAIug) IEC 61850-9-2 implementation guideline (IEC 61850-9-2LE) respectively.

- Methodology: To integrate the AFE with the GOOSE and SV message-mapping VHDL modules to produce a GOOSE monitoring node and a limited-function Merging Unit prototype.
- Evaluation: To develop laboratory test benches for evaluating the developed GOOSE monitoring node and Merging Unit prototype.
- Evaluation: To validate the structure of SV messages published by the developed MU prototype against that defined in the IEC 61850-9-2LE using Wireshark and SVScout software applications.
- Evaluation: To conduct an interoperability test between the developed limited-function Merging Unit prototype with an IEC 61850-compliant IED.
- Evaluation: To conduct an interoperability test between the developed GOOSE monitoring node prototype with an IEC 61850-compliant IED
- Evaluation: To validate the structure of GOOSE messages published by the developed GOOSE monitoring node against that defined in the IEC 61850-8-1 standard using Wireshark and IEDScout software applications
- Evaluation: To evaluate the accuracy of sampled values published by the MU prototype using the CMC 256*plus* test set to simulate five case studies covering power system normal and fault conditions.
- Evaluation: To validate the GOOSE client state machine implemented in the GOOSE message mapping VHDL module against that defined in IEC 61850-8-1 standard.
- Evaluation: To evaluate the accuracy of the root mean square calculation of the voltage injected into the GOOSE monitoring node by the CMC 256*plus* test set.
- Conclusion: To analyse the results from the conducted case studies and provide a conclusion of this research project.

1.4 Project assumptions and delimitation

1.4.1 Project assumptions

The following assumptions are made before embarking on the research project:

- The Omicron CMC 256*plus* test set and the Test Universe software can be used to validate the functionality of the SV and GOOSE message-mapping VHDL modules.
- The CSAEMS at the Cape Peninsula University of Technology will provide an Analogue Front-End (AFE) device that generates time synchronized digital samples from the instrument transformers.
- The ratio between the primary and secondary current/voltage magnitudes is 100:1.
- Interoperability between devices from different manufacturers is possible.
- Interchangeability between devices from different vendors is possible.
- The Xilinx Spartan 6 FPGA contains sufficient resources for implementing a GOOSE and SV messages publisher.
- The Ethernet switch used in the substation will be under moderate load conditions and there are no packet delays.

1.4.2 Project delimitations

The following tasks fall outside the scope of this research:

i. For SV message-mapping this research project will not:
   - Develop the current and voltage transformer (CT and VT) sampling and synchronisation module but will receive ADC codes of the input voltages and currents from an AFE provided by the CSAEMS.
   - SV messages will be published at a fixed rate of 80 samples per cycle using the PhsMeas1 dataset defined in the IEC 61850-9-2LE.

ii. For GOOSE message publishing the following tasks fall outside the scope of this research:
   - The GOOSE monitoring node developed in this research project will only report the status of a circuit breaker and the root mean square voltage injected by the CMC 256*plus* test set.
   - The GOOSE messaging device will not be used as an actuator.
   - The transfer time will not be evaluated for messages published by the GOOSE message-mapping VHDL module.
   - Remote control and data access using the Manufacturing Message Specification (MMS) will not be supported.

1.5 Research methodology and technique

This research project focuses on the design and development of VHDL modules for mapping and publishing IEC 61850 SV and GOOSE message on an FPGA platform. To achieve this goal, three major steps will be conducted and these are the literature review, software development and experimental evaluation.

1.5.1 Literature review

In order to design an IEC 61850-compliant device it is imperative that a clear understanding of the IEC 61850 standard be acquired and this information will be gained from past research papers and projects from different institutions. The literature review will also serve as an initial design step by comparing the different designs of GOOSE and SV message publishers. This literature review process will also discuss communication hierarchy and components defined in the IEC 61850 standard.

1.5.2 VHDL development and hardware integration

Once the information models and communication service mappings to GOOSE and SV messages have been identified the next step will be to develop VHDL modules to achieve this functionality. These VHDL modules will use state machine modeling techniques to define the behaviour of an SV and GOOSE message publishing device. State machines are useful for defining functionality of devices which have clearly defined states when operational.

This part of the research project will also focus on integrating the AFE provided by the CSAEMS and the developed GOOSE and SV message-mapping VHDL modules. The integration of these two entities (AFE and GOOSE/SV message-mapping VHDL modules) will produce a limited-function GOOSE monitoring node and a Merging Unit prototype. The GOOSE and SV message-mapping VHDL modules will then use the ADC codes from the AFE to calculate the power system voltages and currents.

1.5.3 Experimental evaluation

In this step, the developed limited-function Merging Unit and GOOSE monitoring node will be evaluated in the laboratory to validate their conformance to the IEC 61850-9-2LE and the IEC 61850-8-1 standard respectively. These tests will also evaluate the accuracy of the published sampled values during normal and abnormal power system conditions. These power system conditions (normal and fault conditions) will be simulated using the CMC 256*plus* test set.

1.6 Significance of the study

This research is aimed at the process bus implementation in substations for IEC 61850 standard-based automation systems. This process bus implementation was introduced by the IEC 61850 standard to cater for data communication between the primary substation equipment and bay controllers. This research project also aims at providing information on the implementation of this process-bus to assist utilities in the paradigm shift from hardwired and legacy communication protocols to IEC 61850 standard-based communication architectures. This research project will provide effective knowledge transfer to facilitate the development of IEC 61850-compliant devices, test methods for evaluating these devices and a greater understanding of the IEC 61850 standard.

This research project also highlights the advantages of using the IEC 61850 process bus over legacy protocols and traditional hardwired communication architectures.

1.7 Thesis organisation

This thesis comprises of six chapters stated below:

Chapter One introduces the trends in communication networks in substation automation systems and presents the problem statement. This chapter also highlights the aims and objectives of this research project in terms of providing a solution to the problems encountered.

Chapter Two provides a literature review of the application of GOOSE and SV messages in substation automation systems (SAS). This section also reviews past research projects on the design of Personal Computer (PC) based or embedded GOOSE and SV message publishers.

Chapter Three discusses the ten-part IEC 61850 standard with the main focus on part 8-1 and part 9-2 for mapping and publishing GOOSE and SV messages respectively. This chapter discusses the IEC 61850 standard’s data modelling techniques, data semantics, communication interfaces and the communication service mappings of substation information for transmission.

Chapter Four discusses the VHDL code design and implementation of the GOOSE and SV message-mapping VHDL modules conforming to the IEC 61850-8-1 standard and the IEC 61850-9-2LE. This chapter provides state machine models, VHDL code snippets, equations and block diagrams used to implement functional VHDL modules for mapping and publishing SV and GOOSE messages.

Chapter Five details test procedures conducted in the CSAEMS laboratory at CPUT for evaluating the developed GOOSE and SV messages mapping VHDL modules. The laboratory tests are conducted using IEC 61850-compliant software packages and an Omicron test set.

Chapter Six provides a conclusion to this research project and mentions challenges encountered and possible future work for improving and industrializing the developed GOOSE monitoring node and Merging Unit prototype. References used and appendices are presented immediately after Chapter Six.

1.8 Conclusion

This chapter presented the awareness of the need for the development and application of GOOSE-enabled IEDs and Merging Units in substation automation systems. The questions, aims and objectives, deliverables, methodology and contributions of this research project have been presented in this chapter. This section has also provided an outline of the chapters of this thesis.

Chapter Two presents an extensive literature review of research projects conducted on the development of embedded IEDs and PC-based software algorithms capable of publishing GOOSE and SV messages. This literature review chapter also discusses the introduction of the IEC 61850 standard, the IEC 61850 process bus and its application in substation automation systems to serve as theoretical background.

CHAPTER TWO
LITERATURE REVIEW

2.1 Introduction

Substation Automation Systems (SAS) have evolved over the decades from simple bang-bang control mechanisms to fully automated systems relying on the exchange of high priority information between components in a distributed manner. This evolution was and is still driven by the need for a high quality electricity supply, safety considerations for the field workers/consumers and also the reduction of the overall industry (electricity generation) carbon footprint.

This chapter is organised as follows: section 2.2 provides the bulk of the literature review and the conclusion is drawn in section 2.5. Section 2.2 is sub-divided into sections 2.2.1 and 2.2.2, section 2.2.1 provides a brief history on old communication technologies which were employed in substations and the introduction of the IEC 61850 standard. This section also presents a review of literature related to the application of the IEC 61850 process-bus in substations, the design and the evaluation of IEC 61850-compliant devices.

Section 2.3 and 2.4 focus on IEC 61850-8-1 Generic Object Oriented Substation Event (GOOSE) and IEC 61850-9-2 Sampled Value (SV) messages respectively. These sections, sections 2.3 and 2.4 conduct a comparative review of literature pertaining to the mapping, publishing, evaluation and application of GOOSE and SV messages in a Substation Automation System (SAS) respectively. These two sections also compare and analyse hardware designs and software algorithms if any applied in the construction of GOOSE or SV messages enabled devices.

2.2 Review of related literature

This research project focuses on the design and development of two separate VHDL modules for mapping GOOSE and SV messages on a Field Programmable Gate Array (FPGA). This GOOSE and SV message-enabled FPGA device is then combined with an Analogue Front-End (AFE) module to produce a limited-function GOOSE monitoring node and Merging Unit (MU) prototypes respectively. In order to develop an IEC 61850-compliant device which satisfies the aims and objectives of this research project established in Chapter One, a number of research topics are considered and are listed below:

- Evolution of device technology used in substation automation systems.
- Evolution of communication protocols and standards in substation communication networks.
- Understanding of the IEC 61850 standard, and its impact on substation device (IED) design, evaluation and conformance certification.
- Review of research theses and papers on the application of the IEC 61850 process bus.
- Review of research papers on the development of GOOSE and SV message-mapping algorithms on embedded or PC platforms.
- Review of literature on the mapping, application and benefits of using IEC 61850 GOOSE and SV messages in a SAS.

The literature review provided in this thesis is discussed in the sections listed below:

- Communication technology in the substation and the introduction of the IEC 61850 standard.
- The implementation and benefits of using the IEC 61850 process-bus over hardwired and legacy communication techniques.
- The application and performance evaluation of IEC 61850 GOOSE messages.
- Review of research projects on the development of GOOSE-enabled devices.
- The application and performance evaluation of IEC 61850 SV messages
- Review of literature on the development of MUs or computer algorithms for mapping SV messages.

The next section discusses communication techniques used in Supervisory Control and Data Acquisition (SCADA) systems and their shortfalls which led to the introduction of the IEC 61850 standard.

2.2.1 Communication technology in the substation and the introduction of the IEC 61850 standard

Communication networks are vital components in substation automation systems (SAS) with telephone networks being used to report status and provide controls for a few digital points in early control and protection schemes (Sun *et al*., 2012). In the years that followed, digital communications became possible and protection engineers introduced Data Acquisition Systems (DASs) in substations to monitor and control points. At that stage, communication systems only offered limited bandwidth therefore the communication protocols used were optimised for low bandwidth channels (Luwaca, 2014; Retonda-Modiya, 2012; Sun *et al*., 2012).

Over the years that followed, research was conducted to improve the reliability of DASs by improving the communication channel bandwidth. Following the development in communication channel bandwidth, microprocessors were needed to improve control and data acquisition from the substation. This in turn led to rapid development of present day software-configurable microprocessor-based relays known as Intelligent Electronic Devices (IEDs) (Sun *et al*., 2012).

These IEDs combine monitoring, protection, control and logging capabilities in one logical device. Microprocessor-based IEDs present the protection engineer with an opportunity to integrate multiple functions in one device thereby reducing automation system implementation cost and engineering time. With IEDs it is possible to modify the SAS without the need to re-design the device’s hardware.

Legacy communication protocols used in old data acquisition systems prior to the introduction of the IEC 61850 standard and the microprocessor IED specified the byte-order for message transmission between communicating devices. This communication architecture introduced interoperability issues between devices and no interchangeability prospects. This is due to the fact that there was no standard used for data presentation, information modelling and device behaviour within the communication network.

This lack of interoperability and standardised data presentation coupled with manufacturers using proprietary communication protocols led to an estimated US$82 billion being channelled into the development of complicated protocol translators and integration costs in 1998 (Sidhu & Gangadharan, 2005). Therefore it was important that an international standard be introduced to cater for integration issues, standardisation of communication and provide interoperability between devices from different vendors (Sun *et al*., 2012).

Efficient electricity distribution and energy conservation with the aim of reducing the overall industry carbon footprint had to be addressed which led to the introduction of smart grid systems (Farhangi, 2010). According to Konka *et al*.,(2011), smart grid systems are fully inter-operable; communications-enabled electrical systems which aim at revolutionising traditional power systems through the use of condition monitoring and intelligence.

One such standard, which was introduced tackle interoperability issues experienced with legacy protocols in substation communications and has since been adopted to ensure a future with smart grid systems is the IEC 61850 standard. The IEC 61850 standard edition one was introduced in 2002 by Working Group 10 (WG 10) of the International Electrotechnical Commission (IEC) Number 57 (IEC TC 57). The function of the IEC TC 57 is to develop and maintain international standards for power system automation functions. Power system automation functions include distribution automation, teleprotection, remote control and data exchange of real and non-real time information.

The IEC 61850 standard introduces communication interfaces between functions/components which replace hardwired communication links which simplifies substation design and integration. This communication architecture improves the automation system’s expandability and reduces implementation and maintenance costs (Apostolov, 2010). These interfaces divide the substation communication system into three layers; the station, bay and process levels as illustrated in Figure 2.1 (International Electrotechnical Commission, 2003-2004).

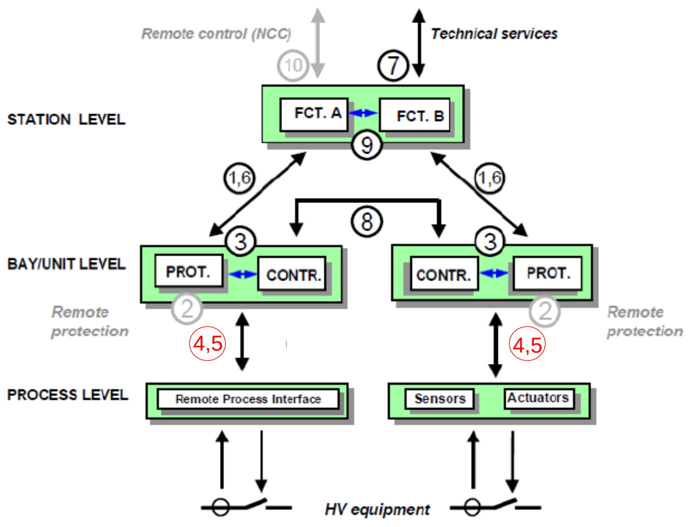


Figure 2.1: **Communication interfaces between components in an IEC 61850-based SAS (Adapted from International Electrotechnical Commission, 2003-2004)**

The numbered interfaces 1 through to 10 in Figure 2.1 represents the type of data and how this data is communicated between functions/devices as discussed below:

- IF1: protection data exchange between the bay and station level functions/IEDs.
- IF2: protection data exchanged between bay level IEDs and remote protections system
- IF3: inter-bay data exchange
- IF4: Voltage and current sampled values published by Merging Units directed towards bay-level IEDs/functions (Sampled Value messages).
- IF5: measurement, status or control data exchange between process level actuators and bay IEDs (GOOSE messages)
- IF6: this interface is for the exchange of control information between bay and station level devices.
- IF7: information exchange between an engineering platform and the station level devices.
- IF8: fast real-time data exchange between bays for real-time functions (GOOSE messages).
- IF9: station level information exchange.
- IF10: this interface is for data exchange between station level device and a remote control center.

Seamless data communication between devices in three substation levels (station, bay and process levels) is made possible by making use of these communication interfaces. Some of these interfaces do not fall within the scope of the first edition of the IEC 61850 standard. For instance, communication interfaces IF7 and IF10 are for communicating with remote control functions and control station respectively.

Bay level IEDs communicate with the station level devices through a specific communication service mapping to the Manufacturing Message Specification (MMS) in which one device is the master and the other is a slave. This communication interface is used for configuration purposes and remote access to retrieve data from bay level devices.

Peer-to-peer communication between bay level and process level devices for real-time applications and transmission of voltage and current samples from process devices (Merging Units) are the two main focus areas of this research project. The exchange of information between process and bay level IEDs using communication interfaces IF4 and IF5 is known as the IEC 61850 process bus. These communication interfaces are achieved by making use of specific communication service mapping of information models and data to an ISO 8802-3 Ethernet frame resulting in GOOSE and SV messages.

With reference to the communication level abstraction of the IEC 61850 standard shown in Figure 2.1; IEDs fall into three categories: the station, bay and process level IEDs. Station-level IEDs reside in the station level of the substation communication hierarchy and they act as Human Machine Interfaces (HMIs) allowing users to execute functions and/or view status of the underlying bay and/or process levels. This IED is capable of communicating with a control centre and/or station level IED in another substation (inter-substation communication). Bay-level IED(s) perform protection, control and measurement functions and exchange real-time information amongst themselves using GOOSE messages represented by interface IF8 as shown in Figure 2.1. These IEDs communicate with process level IEDs through the process bus (communication interface IF4 and IF5).

Process-level IEDs are simpler in design and function compared to bay and station level IEDs. These IEDs use the process bus (GOOSE and SV messages) to transmit sampled values of voltage and current measurements and digital status from primary plant equipment (current and voltage transformers and circuit breakers). Figure 2.2 shows IEDs typically found in different levels of the IEC 61850 substation communication hierarchy.

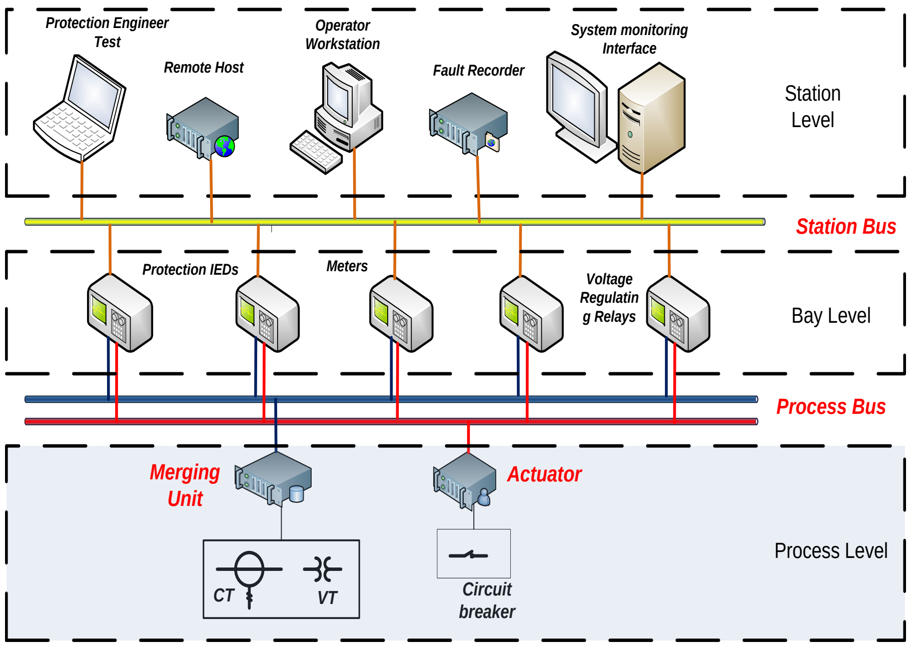


Figure 2.2: **IEDs typically found in different levels of the IEC 61850 substation (Adapted from Luwaca, 2014)**

The following section reviews literature on the IEC 61850-8-1 GOOSE and IEC 61850-9-2 SV messages, their application in automation systems and the development of IEDs capable of publishing these messages.

2.2.2 The IEC 61850 process bus

The IEC 61850 process bus is a data communication interface between process and bay level devices. This interface was introduced in the IEC 61850-8-1 and IEC 61850-9-2 standards and is represented by communication interfaces IF5 and IF4 in the illustration in Figure 2.1 respectively. This high speed communication interface is based on the ISO 8802-3 Ethernet frame for the transmission of voltage and current sampled values and also digital signals e.g. circuit breaker status.

The IEC 61850 process bus is highly advantageous compared to hardwired communication techniques because it:

- Reduces parallel copper wiring between devices thereby reducing the implementation cost of an automation system
- Eliminates current transformer saturation by reducing the lead resistance. The lead resistance is small because Merging Units (MUs) have very small input impedance.
- Improves the reliability of a system by eliminating open current circuit conditions (Apostolov, 2010).

The IEC 61850 process bus consists of both GOOSE and SV messages, for this reason GOOSE and SV messages are referred to as process bus protocols. Analogue and digital data can be transmitted using GOOSE messages, for example, circuit breaker status and root mean square voltages for a 3-phase power system. The following section (section 2.3), will be dedicated to reviewing literature on the application of GOOSE messages in automation systems, performance evaluation of GOOSE-enabled devices and the development of IEDs capable of publishing these GOOSE messages. Section 2.4 is dedicated to a comparative review of literature related to the application of SV messages and the development of Merging Units.

2.3 Application, performance evaluation and development of IEDs capable of publishing IEC 61850-8-1 GOOSE messages

The IEC 61850 standard introduces a number of interfaces to facilitate communication between IEDs in different levels and components within a substation automation system. One such communication interface represented by IF5 and IF8 in Figure 2.1 is the Generic Object Oriented Substation Event (GOOSE) message. This interface (GOOSE messaging) maps information models and data objects into an ISO 8802-3 Ethernet frame (Retonda-Modiya, 2012).

GOOSE messages are fast real-time, peer-to-peer messages transmitted regularly or when there is a change of status/value of any of the objects in the dataset. GOOSE messages can be used for reporting analogue and or binary information to any subscriber device. Implementing an automation system using GOOSE messages can reduce wiring and engineering costs because the information (binary and/or analogue data) is readily available on the Ethernet network (process bus) (Sidhu & Gangadharan, 2005). GOOSE messages are highly advantageous because they allow substation automation systems to expand without major redesigns of the existing architecture or infrastructure.

GOOSE messages can be used for exchanging information/commands between functions (inter-bay communication) or between primary plant equipment (CTs, VTs and circuit breakers) and bay level IEDs (International Electrotechnical Commission, 2003-2004). Analogue measurements required by protection applications can be transmitted using GOOSE messages to the bay level IEDs; for instance, current and voltage measurements from the CTs and VTs to the Voltage Regulating Relays (VRRs) as illustrated in Figure 2.3 (Bowe, 2014).

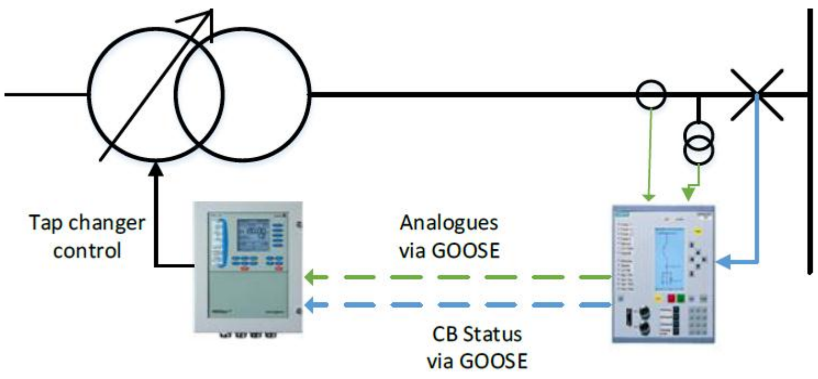


Figure 2.3: Transformer voltage regulation using GOOSE messages (Adapted from Bowe, 2014)

GOOSE messages are classified into two types -Type 1A and Type 1B, Type 1A is for GOOSE trip messages whereas type 1B is for the GOOSE Block message (International Electrotechnical Commission, 2003-07). GOOSE trip messages fall into two performance classes and these are P1 and P2/P3. For P1 class the total transmission time of 10 ms is defined and for performance class P2/P3 the total transmission time of 3 ms is defined in the IEC 61850-5 standard (International Electrotechnical Commission, 2003-07; Ito & Ohashi, 2008).

GOOSE messages are mapped directly to layer 2 of the Open Systems Interconnect (OSI) stack to reduce protocol overhead thereby guaranteeing fast transmission times. GOOSE messaging supports multicasting so that all subscriber

IEDs can receive information published by a device in the network. The IEC 61850-8-1 GOOSE messaging scheme does not require receipt acknowledgments to be sent to the publisher once the subscriber receives a message, this communication scheme guarantees fast peer-to-peer message transmission (Ali, 2012).

There are many advantages of using GOOSE messages over legacy protocols; these advantages listed below make this protocol flexible and ideal for high speed communications:

i. They reduce point-to-point copper wiring of signals from IED to the next thereby reducing the system engineering time and commissioning costs (Fernandes *et al*., 2014).
ii. They are used to achieve fully integrated distributed substation automation systems by exchanging IED information in real-time (Ali, 2012).
iii. GOOSE messages can be multicast to multiple subscriber IEDs on the process bus network by making use of the multicast address defined in the IEC 61850-8-1 standard (Mackiewicz, 2011).
iv. The GOOSE protocol does not support acknowledgements for any messages received and this greatly reduces the amount of traffic in the network and guarantees high delivery rates (Ali, 2012).
v. The GOOSE messaging scheme allows the transmission of multiple data objects in a dataset. The use of datasets reduces the protocol overhead in each message and thus provides high message transmission rates and proper communication bandwidth utilisation.
vi. GOOSE messages implement a retransmission algorithm which allows the information to be published after a certain time interval such that all the IEDs in the SAS are updated of the other IED's status. These messages can also be used as heartbeat message to diagnose communication problems in a network (International Electrotechnical Commission, 2004-2005).
vii. They support VLAN tagging according to IEEE 802.1Q to separate time critical messages from less time constrained station configuration data and thus increasing delivery rate and reliability (Fernandes *et al.,* 2014).

As can be seen from the points listed above, IEC 61850 GOOSE messages are ideal for implementing communication systems for substation automation purposes.

The following section discusses different case studies that show how GOOSE messaging is used in substation automation systems.

2.3.1 Application of GOOSE messages in substation automation systems

GOOSE messages are used for communications in substation automation systems. One such application of GOOSE messages is presented in a research paper by Gajic *et al.,* (2014) where On Load Tap Changer (OLTC) of parallel transformers is implemented using these messages. OLTC is implemented to regulate the electricity network voltage in case it drops when the supplied load increases. In their paper, Gajic *et al.,* (2014) configures the test system such that the measured voltage and current values are exchanged between two IEDs using GOOSE messages. The authors conclude that GOOSE messages can be used to achieve this functionality by taking advantage of IEC 61850 data models and information exchange (Gajic *et al.*, 2014).

Similarly to the research paper by Gajic *et al*., (2014), Bowe, (2014) presents a GOOSE message-based parallel transformer switch mode using A.Eberle REG-DA voltage regulating relays. In this application, two incoming transformers are controlled by A.Eberle REG-DAs; these REG-DAs receive incoming feeder and bus section circuit breaker and voltage measurements from three Siemens SIPROTEC IEDs. This paper demonstrates the versatility of GOOSE messages by using this protocol for the exchange of analogue and binary information thereby eliminating the use of parallel copper wiring in a substation.

The GOOSE messaging communication interface can also be used in load shedding schemes, in which the system continuously calculates the generation reserve and the shed-able load in real-time by making use of power measurements transmitted via this interface. The Fast Load Shedding Scheme (FLSC) conducted in a paper by Wester *et al.,* (2014) takes advantage of GOOSE message mechanism like VLAN tagging to ensure high message delivery rates by separating low priority traffic from time-critical messages.

The FLSC scheme also relies on data retransmission attribute of GOOSE messages to ensure that all the aggregators within this scheme receives commands from the master controller. The authors conclude that the FLSC proposed will be reliable and will prevent total system failure should the generation reserve fail to meet the load requirements (Wester *et al.*, 2014).

Similarly to the papers published by Gajic *et al.,* (2014) and Wester *et al.,* (2014), Cabrera *et al.,* (2012) illustrates the use of IEC 61850-8-1 GOOSE messages in protection schemes within a substation. An arc-flash protection scheme is implemented in case of current flashover from one conductor to another due to insulation breakdown and poor maintenance. Arc-flash detectors detect this fault and are connected to protection relays which initiate circuit breaker tripping. Bus-bar protection is initiated when arcing is detected which trips the bus (Cabrera *et al*., 2012).

The implemented solution comprises of arc-flash detectors installed in the breaker, current transformer and bus bar chambers of the different feeders. These detectors are connected to an IED dedicated to the feeder bay and these IEDs exchange arc detection status using multicast GOOSE messages. The results show that GOOSE messaging can be used to provide for reliable and interoperable cost effective methods for bus protection (Cabrera, *et al*., 2012).

IEC 61850 GOOSE messages can be used for a variety of applications and the examples discussed in this section are some of the common applications of GOOSE messages within IEC 61850 standard-based SAS. Table 2.1 catalogues papers found in literature discussing the application of the GOOSE communication protocol in an IEC 61850 standard-based substation automation system.

Table 2.1: **Application of GOOSE messages in an IEC 61850 based SAS**

Paper	Application	Method/Simulation	Type of Data	Evaluation and Comments
Bowe, (2014).	Voltage regulation.	Two transformers are controlled in a parallel switch mode using two A.Eberle REG-DA with three Siemens SIPROTEC IEDs which publish the voltage measurements and circuit breakers statuses at the incoming feeders and bus section	Analogue and Digital	Transformer parallel switch mode can be achieved using GOOSE messages with great reduction in complex wiring and improving system flexibility.
Cabrera *et al.*, (2012).	Bus protection using Arc flash detection in four bays.	The system is made up of multiple arc flash detectors in the CT, breaker and bus bar chamber connected to the dedicated IEDs. The IEDs exchange the arc flash detector status and implement bus protection scheme.	Analogue and Binary	The protection scheme was tested in a real power station and the results prove that IEC 61850 GOOSE SCSM can be used for real-time protection schemes.
Wester *et al.*, (2014).	Fast Load Shedding Scheme (FLSS).	The system is made up of a single controller and one or more aggregators which inform the main controller of the current system power flows. The controller and the aggregators communicate via GOOSE messages. The main controller decides which load(s) to shed depending on the measurands received from the aggregators.	Analogue and Binary	GOOSE messages can be delivered reliably by making use of VLAN tags. This proposed Load shedding scheme is reliable and can prevent system failures because of the real-time nature of the GOOSE protocol.
Gajic *et al.*, (2014).	On Load Tap Changer Control (OLTC).	The system consists of two transformers and each transformer has a dedicated IED for protection and control. The transformer data is modelled using the YLTC and the ATCC logical node classes and is exchanged using GOOSE messages between these two IEDs. The circulating current method is used which relies on a circulating current flowing if two parallel transformers are out of step.	Analogue and Binary	GOOSE messages can be delivered reliably by making use of VLAN tagging and periodical retransmission. Using GOOSE messages ensures system flexibility and interoperability.

2.3.2 GOOSE message performance evaluation

Part 5 of the IEC 61850 standard defines message transmission times for different types and classes of messages (International Electrotechnical Commission, 2003-07). GOOSE messages are classified as type 1 and 1A messages and the standard defines a transmission time of 3 ms for performance P2/P3 messages (Mackiewicz, 2012).

All devices claiming conformance to the IEC 61850 standard must be able to send messages of distinct types/classes which are supported according the transmission times defined in Part 5 of the standard. The IEC 61850 standard is still in the implementation phase and practical experimentation is required to detect efficiency and applicability problems experienced so that efficient solutions can be proposed. To this end, research into the efficiency of GOOSE messages has been conducted and is reviewed in this section.

Netto *et al.*, (2012) investigates the effect of Denial of Service (DoS) overloading of an IED bandwidth on the GOOSE transfer time. In this test, two time-synchronised IEDs are setup wherein IED1 is the publisher and IED2 is the subscriber. The published and subscribed messages are time-stamped and recorded using the State Events Recorder (SER). IED2's bandwidth was flooded with UDP packets generated by the DoS. The transfer time is calculated as the time difference between subscriber time-stamped SER and publisher SER.

The results show that the GOOSE transfer time is not violated for up to 15% DoS overload and all GOOSE messages were delivered. For 15%-95% DoS overload the GOOSE mean transfer time ranges from 4.43 ms to 103.01 ms and 46.95% of GOOSE message published by IED1 were not delivered to IED2.

When designing IEC 61850-8-1 GOOSE-compliant devices, it is important that the message transmission time be less than 3 ms as defined in the IEC 61850-5 standard (International Electrotechnical Commission, 2003-07). Gonzalez-Redondo *et al.,* (2013) discusses methods for measuring the GOOSE message transfer time for device in the development stage to ensure that they adhere to the timing requirements for the transmission of IEC 61850-8-1 messages.

There are three time measurement methods and these are the round-trip, ping-pong test and rally test. Round trip time defined in the IEC 61850-10 standard can be used to determine how fast the transmitting device is; in this case the subscribed message is retransmitted back to the publisher. The second method is the ping-pong test where one device, the Device under Test (DUT) publishes a message to a subscriber device. When the subscribing device receives the message it transmits a GOOSE message that is subscribed to by the DUT. The third test is a rally test where both devices publish GOOSE messages which are subscribed to by the other device. Upon receiving a message the device publishes its own message subscribed to by the other device therefore both devices are in a continuous excitation state.

Experimental test benches are implemented in a laboratory environment using two DK60 development boards from Beck IPC with a built–in IEC 61850 stack. In this setup one device is the server and the other a client. The results of this experimental setup show that the maximum round time trip for a GOOSE message is 1.536 ms which is less than the 3 ms required for the transmission time (Gonzalez-Redondo *et al.*, 2013).

Fernandes *et al.,* (2014) presents two methods for measuring the transfer and round trip times of GOOSE messages published by an IED in an automation system. These test methods comprises of two IEDs, network switch, configurator PC and the Omicron CMC test set. For measuring the round trip time, the publisher IED transmits a dataset containing the binary input (BI1) information using the GGIO logical node to the subscriber IED2 which then maps this data to user binary output (BO2) connected to the CMC test set. A GOOSE message is transmitted automatically once the CMC test set injects a voltage into the binary input of IED1. The state sequencer tool calculates the time between voltage injection into IED1 and voltage detected from BO2 of IED2.

The GOOSE transfer time is calculated by setting up one IED as the publisher, the Omicron CMC test set as the voltage source and GOOSE message subscriber. The IED publishes GOOSE messages once the voltage on binary input (BI1) connected to the Omicron CMC test set AUX output changes. The Omicron test set calculates transfer time as the time from voltage injection to GOOSE message reception from the IED.

The round-trip time and the publisher time are measured to be 18.50 ms and 6.20 ms which include the input debounce, IED operation time and the binary input sensing time Fernandes *et al.,* (2014). According to Fernandes *et al.,* (2014) the number of objects published in the dataset is directly proportional to the GOOSE transfer time, the fewer the objects in the dataset the faster the transmission time.

Daboul *et al.,* (2015) presents the performance and timing performance of GOOSE messages over hardwired communication techniques in a substation automation system. In this evaluation, two IEDs were configured to communicate via GOOSE messages and hardwired signals exchange; the results show that the GOOSE round-trip time is approximately 2.5 ms whereas it took 20 ms for the same data to be exchanged via hardwired signals. This paper proves that GOOSE messaging can be used in substation automation systems for information exchange without degrading the performance of the system.

Table 2.2 shows a catalogue of reviewed papers on the measurement of the GOOSE transfer and round-trip times. This catalogue also highlights the factors affecting the GOOSE message transfer and round-trip times.

Table 2.2: **Comparison of performance evaluation of GOOSE messages**

Paper	**Type of diagnosis**	**Test setup**	**Test Equipment**	**Comments/ Outcomes**
Netto *et al*., (2012)	Effect of DOS overload on the GOOSE transfer time	Yes	GOOSE transfer time measurement	GOOSE transfer time is affected by network traffic, for less than 15% DoS overload all GOOSE messages published are delivered to the subscriber and the 3 ms transfer time is honoured. For more than 15% overload the transfer time is violated.
Gonzalez-Redondo *et al*., (2013)	Measuring GOOSE transfer time using the Round trip time, Ping pong and the rally test. One of the development boards is the client whilst the other is the server.	Yes	• two DK60 development boards	There are some factors to be considered when measuring the time. Is the time measured within the application code before the API function? (Efficiency of the stack to be considered). Measurement of time after the stack which reflects the network time And lastly time measurement when an event occurs e.g. change in breaker status.
Fernandes *et al.,* (2014)	Measurement of the GOOSE transfer time and round-trip time using two IEDs and the Omicron CMC test set.	Yes	• Two IEDs • Omicron CMC test set • Industrial switch	The GOOSE transfer time is directly proportional to the number of data objects in the dataset. The GOOSE publisher time measured in these experiments was more than the 3 ms defined in the standard because it included the input debounce and relay function time.
Daboul *et al.,* (2015)	Measurement of the IEC 61850-8-1 standard GOOSE round trip time.	Yes	• Two ABB REF 615 IEDs • Configuration PC • Ethernet switch	This test shows that GOOSE messages are faster and more flexible compared to conventional hardwired communication techniques

These past research projects prove that GOOSE messages can be used for communicating time-critical messages using an Ethernet network with relatively low traffic of up to 15%. The DK60 board used by Gonzalez-Redondo *et al.,* (2013) is based on an FPGA and embedded Beck-IPC web controller, this test shows that it is possible to map and publish GOOSE messages conforming to the timing requirements of the IEC 61850 standard using an embedded platform.

The next section presents a review of literature on the design and development embedded device for mapping and publishing GOOSE messages.

2.3.3 GOOSE message-mapping on embedded platforms

There are a number of factors to be considered when developing IEC 61850 standard-compliant devices for GOOSE messaging. These factors depend mostly on the application of the device in a substation automation system and also the timing constraints defined in the IEC 61850-5 standard. Some of the factors to be considered are:

i. Does the device need analogue inputs?
ii. Does the device require digital inputs?
iii. How much processing power and memory does the device require?
iv. What are the effects of the environment on the operation of the device?
v. Should the device implement the MMS stack?

This section discusses previous research projects on designing GOOSE-compliant devices. The findings are catalogued in chronological order and compared according to application in Table 2.3.

Fan *et al.,* (2011) details the design of a device to measure the steady, transient and dynamic states of a power system. This device outputs phasor, power and frequency measurements from the substation to a higher degree of accuracy with a percentage error of 0.2%. This device also records the system parameters for fault analysis. In this paper the authors highlight the advantages of using FPGAs in designing IEC 61850 standard-compliant devices; these advantages include high precision for time-keeping and synchronisation (Fan *et al.,* 2011).

This device presented in the paper by Fan *et al*., (2011) is a multi-function device and possesses time synchronisation capabilities, analogue inputs for system voltages and currents from the instrument transformers. This device publishes and subscribes to GOOSE messages for locking functions and the results prove that it is possible to design a multi-functional IEC 61850 standard-compliant device using a high speed microprocessor and an FPGA.

MingCai *et al.,* (2012) presents the design of a device to control high voltage SF6 circuit breakers in a 126 kV substation using an STM32F103ZET6 processor and an Altera Cyclone II EP2C8Q208C8 FPGA. The terminal device receives commands from a secondary IED and sends actuating signals to the breaker. This device also transmits all circuit breaker status, alarms and analogue values using GOOSE messages.

The design is based on an Advanced R*ISC (Reduced Instruction Set Computers)* Machines (ARM) and an FPGA, the processor is responsible for data processing and GOOSE transmission while the FPGA is used for extended decoding and processing functions because of its high speed input/output, high processing speed and the multi-threading capabilities (MingCai *et al.*, 2012). The results show that the design can meet the real-time requirements of the IEC 61850 standard even though it was not tested using IEC 61850 standard-compliant equipment to verify its compliance and interoperability.

Retonda-Modiya, (2012) outlines the design of an IEC 61850-8-1 standard compliant circuit breaker actuator. This actuator also publishes the circuit breaker status to subscribing IEDs using GOOSE messages (Retonda-Modiya, 2012). This GOOSE monitoring node presented by Retonda-Modiya, (2012) opens/closes the circuit breaker once it receives actuating commands mapped onto the MMS according to the IEC 61850-8-1 standard from a server device.

This actuator is based on the DK60 platform which has a built-in Ethernet controller from Beck-IPC running an IEC 61850 communication stack for mapping GOOSE and MMS messages. This actuator is tested in a laboratory setup using an Omicron CMC 256*plus* injection test set and a Siemens IED to implement an overcurrent protection scheme. The CMC 256*plus* injection test set is configured to generate and inject voltage and current signals emulating an overcurrent fault into the Siemens IED. Once the fault is detected the Siemens IED publishes a command instructing the actuator to close the circuit breaker.

After a number of experimental overcurrent injection tests, the average GOOSE transfer time is measured to be 0.95 ms which is less than 3 ms defined for Type 1A performance class P2/P3 messages. This proves that IEC 61850-compliant devices can be developed using the DK60 development platform.

Gonzalez-Redondo *et al.,* (2013) developed two GOOSE-enabled devices based on the DK60 development board similarly to the one used by Retonda-Modiya, (2012). These devices were used for evaluating the methods for measuring the GOOSE message transfer time for devices in the development stage to ensure their conformance to the timing requirements defined in the IEC 61850-5 standard. Device testing especially time and performance measurement are important in IEC 61850 standard-compliant device design. Retonda-Modiya, (2012) uses a variation of the round-trip test to evaluate the developed GOOSE actuator’s transfer time.

Fan *et al.,* (2011) and MingCai *et al.,* (2012) outline the advantages of using FPGAs for designing IEC 61850 standard-compliant devices so as to meet the timing constraints for type 1A messages. In these two papers, the authors use a processor for communication and data processing and an FPGA for timing and data I/O because the FPGA has fast I/O ports.

The development of an IEC 61850-8-1 standard GOOSE message-mapping VHDL module conducted in this research project will use an FPGA for data I/O, processing and communication as opposed to projects presented by Fan *et al.,* (2011), MingCai *et al.,* (2012), Retonda-Modiya, (2012) and Gonzalez-Redondo *et al.,* (2013). The FPGA is capable of performing these tasks without the assistance of a processor because it contains a vast number of logic blocks which can be used for any logical functions executed in parallel.

Figure 2.4 shows the number of research papers on the development of embedded IEDs capable of communicating via GOOSE messages.

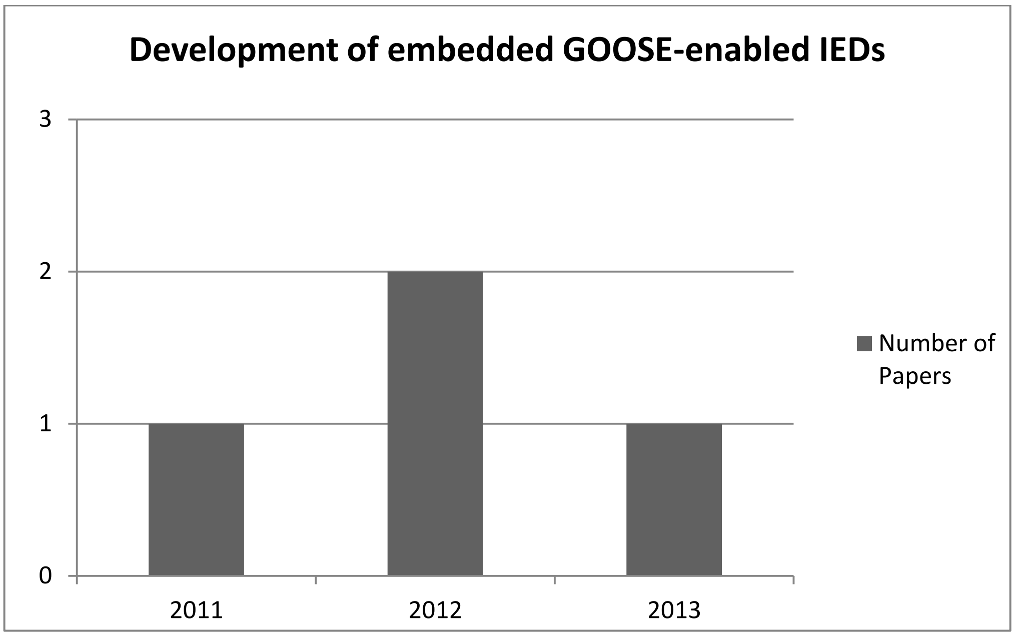


Figure 2.4: **Number of papers reviewed on the development of embedded GOOSE-enabled IEDs**

Table 2.3 shows a comparative analysis of embedded IEC 61850-8-1 standard GOOSE message-enabled devices found in literature. This catalogue shows the intended application of the devices, performance evaluation and the development platforms used.

Table 2.3: **Comparing GOOSE-enabled embedded devices**

Paper	Research Objectives	Device Application	Target Platform	Performance and Evaluation	Outcomes
Fan *et al.*, (2011).	Development of an IEC 61850-compliant device for steady-state, transient and dynamic state system measurement using phasor and instantaneous measurements.	Phasor measurement, GOOSE publisher and system recording.	• POWERPC • FPGA • PNM module for mapping GOOSE and IEC 61850-9-2 SV messages	N/A	Highlights the advantages of using an FPGA for timing synchronisation and sampling owed to its parallel processing capabilities, high operating frequency and fast input/output ports.
MingCai *et al.*, (2012).	Design and development of an actuator to control the high voltage SF6 circuit breaker using MMS commands. This actuator publishes breaker status and other analogue measurements from the breaker using GOOSE messages.	High voltage SF6 actuator.	• ARM processor (STM32F103ZET6) • FPGA (Altera Cyclone II EP2C8Q208C8) • Ethernet MAC and PHY (DM9000AE)	Yes	This paper highlights the advantages of using an FPGA in applications with stringent time constraints.
Retonda-Modiya, (2012).	Design of a low cost IEC 61850-compliant circuit breaker actuator using GOOSE and MMS messaging.	Circuit breaker actuator.	• DK60 development board with a Beck IPC web controller chip (SC143-IEC-LF)	Yes	GOOSE delivery time of 0.975ms was achieved which meets the requirements of IEC 61850 P1 class messages.
Gonzalez-Redondo *et al.,* (2013)	Development of methods for evaluating the GOOSE message transfer time for devices in the development stage.	GOOSE transfer time evaluation.	• DK60 development board with a Beck IPC web controller chip (SC143-IEC-LF)	Yes	The traffic on the Ethernet network and programming APIs affect the GOOSE message transfer time.

The following section conducts a review of literature on the application of SV messages in an automation system, performance evaluation of SV-based automation systems and lastly the development of IEC 61850-9-2 SV-compliant devices.

2.4 Application, performance evaluation and development of IEDs capable of publishing IEC 61850-9-2 Sampled Value (SV) messages

The Merging Unit is defined in the IEC 60044-7 and IEC 60044-8 technical documents on Electronic Voltage Transformers (EVTs) and Electronic Current Transformers (ECTs) respectively. The MU is also mentioned in the IEC 61850-9-1 and IEC 61850-9-2 standard for the transmission of current and voltage samples from the primary plant equipment. IEC 61850-9-1 standard and IEC 61850-9-2 standard Merging Units transmit SV messages to bay level IEDs through the serial link and the Ethernet process bus respectively.

These SV messages contain instantaneous voltage and current samples of the power system sampled by the MU at a specific rate. Sampled value messages received by the bay-level IEDs are used for metering and for implementing various protection functions. Bay-level IEDs subscribe to the these published SV messages using the abstract communication services defined in the IEC 61850-7-2 and IEC 61850-9-2 standards (International Electrotechnical Commission, 2002-2007; International Electrotechnical Commission, 2004).

The MU illustrated in Figure 2.5 accepts analogue input signals from CTs and VTs through an Analogue to Digital Converter (ADC) and binary inputs from primary plant equipment. This information is mapped onto an IEC 60044-8 standard or IEC 61850-9-1 standard SV frame and published to bay controllers using a serial unidirectional multi-drop point-to-point link.

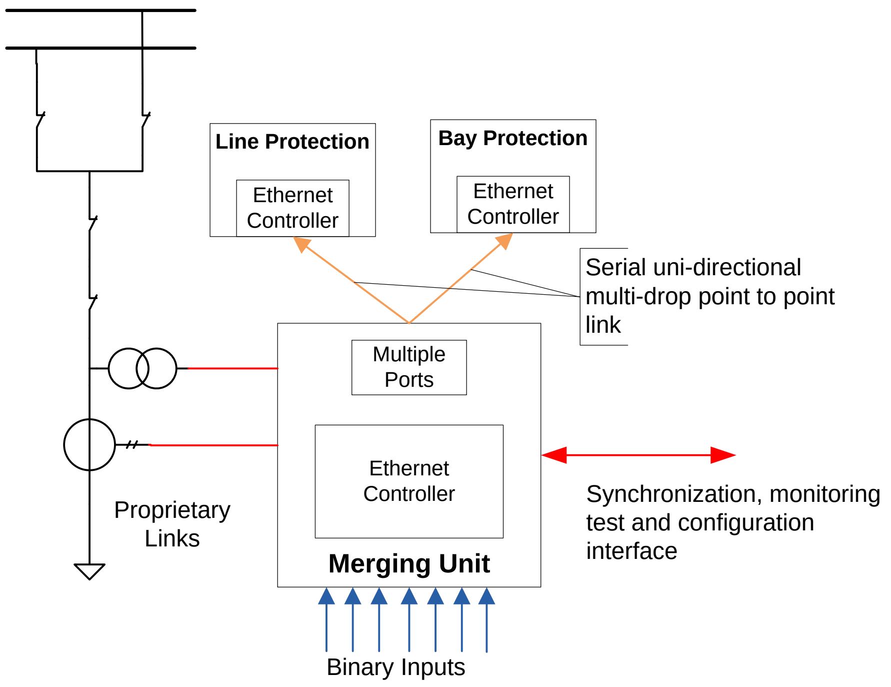


Figure 2.5: **Application of the IEC 61850-9-1 standard-based Merging Unit (Adapted from Baigent *et al.,* 2004)**

The illustration in Figure 2.5 shows a MU which is capable of publishing sampled value messages containing digital information from digital inputs as per the IEC 61850-9-1 standard. This research project focuses only on sampled value messages published as per the Utility Communications Architecture International Users Group (UCAIug) implementation guideline of the IEC 61850-9-2 standard known as the IEC 61850-9-2LE (Light Edition). Figure 2.6 shows an internal block diagram and analogue connections of a typical IEC 61850-9-2LE Merging Unit.

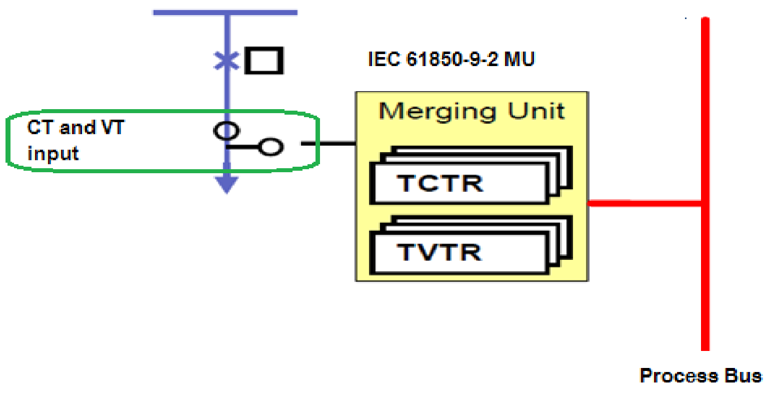


Figure 2.6: **Typical IEC 61850-9-2LE-compliant Merging Unit**

Figure 2.6 shows a typical IEC 61850-9-2LE Merging Unit, this MU receives current and voltage signals from CTs and VTs which are modelled using TCTR and TVTR logical node classes respectively. The difference between the two MU designs shown in Figure 2.5 and Figure 2.6 is that the latter does not support the mapping of digital inputs to the sampled value messages. The IEC 61850-9-2LE defines a dataset known as *PhsMeas1* containing a total of eight instantaneous sampled values: four current and four voltage samples published at either 80 or 256 samples per cycle for protection or metering functions respectively.

The IEC 61850-9-2LE was published in 2004 to eliminate implementation ambiguity of MUs that is to reduce design and implementation complexity of IEC 61850-9-2 standard-based Merging Units (Luwaca, 2014). This is because the IEC 61850-9-2 standard is broad and extensive (Apostolov, 2010). The IEC 61850-9-2LE defines two standard control blocks which define how and when the sampled values are published to the bay level IEDs.

2.4.1 IEC 61850-9-2 Merging Unit testing, performance and evaluation

Performance, reliability, availability and timing characteristics of the IEC 61850-9-2 standard process bus in a substation automation system must be evaluated since it is set to be the backbone of all communications between the process and the bay level devices.

Haude, (2010) proposes a method for evaluating sampled value messages transmission under different power system conditions. This test procedure focuses on the sample count, accuracy of samples and completeness of samples by using an Omicron CMC test set as a reference. Haude, (2010) also compares the sampled value transmission and accuracy between Non-Conventional Instrument Transformers (NCIT) namely the Rogowski coils and conventional instrument transformers.

In the proposed test setup, two MUs from different vendors are used; one MU has a Rogowski Coil and the other a conventional current transformer and they are connected to the same master time source with the CMC test set. The CMC injection set was then configured to inject current signals into these MUs and also publish sampled value messages. The sampled value messages published by these three devices (two MUs and the CMC test set) were then compared to check for completeness of samples and accuracy.

Starck *et al*., (2013) investigates the effects of using IEC 61850-9-2 standard and IEC 61850-8-1 standard communication service mappings on the availability, performance and reliability of a substation automation system. Additionally, Starck *et al*., (2013) compares the availability, reliability and performance factors between a non-conventional and conventional instrument transformers-based single bus bar configuration distribution substation. Each section of the bus bar contains one incoming feeder and eight outgoing feeders with the measuring equipment on the cable side. From the calculated Mean Time to Failure (MTTF) values, Starck *et al*., (2013) established that protection systems based on the conventional measurement techniques are costly to implement, less reliable and less flexible compared to IEC 61850-9-2 and NCIT-based protection systems. IEC 61850-9-2 standard-based protection systems exhibit high availability and reliability compared to conventional systems because the IEC 61850 standard specifies a network redundancy scheme based on the Parallel Redundancy Protocol (PRP) and High availability Seamless Redundancy Protocol (HSRP).

Adewole and Tzoneva, (2014) evaluate the impact of the SV process bus on operating performance of protection IEDs. The factors investigated included the speed, dependability and security capabilities of the concerned sampled value process bus. Furthermore, Adewole & Tzoneva, (2014) considered the performance of two IEDs in a distance protection scheme by comparing a hardware-in-the-loop implementation to a conventional-hardwired setup. The hardware-in-the-loop SV protection scheme was configured using a Real Time Digital Simulator (RTDS), an IED, GPS clock and an industrial network switch. To measure the performance, the protection system was subjected to various fault locations, fault resistance and fault inceptions with different Source Impedance Ratios (SIR). The IEC 61850-9-2 standard process bus communication network was subjected to random noise/delay so as to evaluate its security and dependability characteristics (Adewole & Tzoneva, 2014).

The results of the tests conducted in the laboratory prove that the IEC 61850-9-2 standard process bus can be used instead of the conventional hardwired communication techniques between protection IEDs and instrument transformers because both communication techniques have similar response and tripping times. The advantage that the SV process bus has over hardwired systems is the reduction of parallel copper wires between CTs, VTs and the protection IEDs since sample values are published onto the Ethernet network by the RTDS. The IEC 61850 process bus has also proven to be dependable and secure as witnessed by successful protection scheme demonstrations. Test conducted by Adewole & Tzoneva, (2014) demonstrates that the IEC 61850 process bus can be used instead of conventional protection schemes without affecting the security and functionality of the system.

To facilitate the verification of SV messages published by a MU, Sumec, (2014) developed a software application that monitors traffic on the process bus to detect SV messages. This application is capable of decoding the SV frame, plotting the received information on a real-time graph and exporting the sample values to a Comma Separated Values (CSV) file for further analysis. This software is able to decode messages from an arbitrarily configured MU using a Substation Configuration language (SCL) file.

This software is capable of detecting up to 20 errors in the received SV messages including invalid sample count, message length and ASN.1 Tag errors. Sumec, (2014) evaluated the decoding algorithm by publishing sampled value messages from a CMC 256*plus* test set and from a MU then comparing the decoded signals on a plot within the developed software application.

Table 2.4 shows a comparative analysis of research projects in evaluating the performance of SV messages for use in substation automation systems.

Table 2.4: **SV message performance review**

Paper	Investigation	Test Equipment	Method	Physical Test Bench	Comments
Haude, (2010)	Sampled value message transmission under different network condition	• Merging Unit • NCIT • conventional instrument transformer	Comparing time behaviour data transmission, completeness of samples, sample counter and accuracy of samples between SV messages published by NCIT connected MU, CMC test set and Merging Unit connected to conventional instrument transformers.	Yes	The resolution of the test set is not high enough to evaluate the sample tolerance of 4 µs.
Starck *et al.*, (2013)	Effects of IEC 61850-9-2 sampled values and Non-Conventional measurements on availability performance of an IEC 61850 based SAS.	ABB UniGear ZS1 in single bus-bar configuration with two bus sections each with one incomer and eight outgoing feeders.	Comparing availability, reliability and performance attributes of protection systems based on conventional instrument transformers and NCIT with IEC 61850-9-2 process bus.	No	Protection systems based on NCIT (Non-conventional instrument transformers) on a redundant Ethernet network have a higher availability compared to their conventional counter-parts.
Adewole & Tzoneva, (2014)	The main objective of this research was to compare performance, speed, security and dependability of an IEC 61850-9-2 process bus-based distance protection system with a hardwired protection systems.	• RTDS with a GTAO card and GTNET-SV cards. • Conventional instrument transformers • Circuit breaker	Conventional System Connected to a circuit breaker and to the instrument transformer through the RTDS GTA0 card. IEC 61850-9-2 SV system. Connected to a circuit breaker and to the instrument transformer through the RTDS's GTNET-SV card	Yes	The tests verify that the protection system is not affected due to the noise/delay of up to 3 sample period. The research paper also proves that IEC 61850-9-2 sampled value process bus can be used in protection schemes without affecting the overall performance, speed and reliability but greatly reduces the amount copper wiring required for implementation.

Paper	Investigation	Test Equipment	Method	Physical Test Bench	Comments
(Sumec, 2014	Verification of Sampled value messages published by MU.	• CMC 256*plus* test set • Merging Unit	The CMC 256*plus* test set is configured to generate analogue signals and publish Sampled value messages. The signals generated by the CMC 256*plus* test set are injected into a MU. The SV streams published by both the CMC 256*plus* test set and the MU are captured using the SV verification software and compared.	Yes	The captured signals plotted by the SV verification software are not compared to signals captured and plotted by any other software tool e.g. SVScout.

2.4.2 Sampled Value (SV) messages mapping on embedded platforms

This section provides a review of literature on sampled value message-mapping algorithms and MU designs. This review section focuses on MU designs and SV message-mapping algorithms as specified in the IEC 61850-9-1 and IEC 61850-9-2 standards so as to provide a solid background into hardware designs and mapping techniques.

Yin & Liu, (2004) highlight the importance of MUs by listing the disadvantages of using conventional instruments transformers and highlighting the advantages of Electronic Instrument transformers (ECTs and EVTs). Contrary to conventional instrument transformers, ECTs and EVTs have a wide dynamic range and a linear output (Yin & Liu, 2004). Yin & Liu, (2004) mention the IEC 60044-8 and the IEC 61850-9-1 standards.

In their research, Yin & Liu, (2004) used the 1 Pulse per Second (1PPS) pulse from a Global Positioning System (GPS) for the FPGA-based MU. The multi-channel analogue signals from the electronic instrument transformer are sampled synchronously by the ADC and the sampled value messages are then transmitted using the universal dataset defined in IEC 61850-9-1 standard.

The sampled value control block is preconfigured to publish 80 samples per nominal period so therefore on a 50 Hz network, 4000 sampled value messages are published. Yin & Liu, (2004) conclude by stating that it is possible to design a merging unit to interface electronic instrument transformers with bay level IEDs using an FPGA and still conform to the IEC 61850-9-1 standard.

In 2007, Liu *et al*., (2007) utilised a 32-bit ARM processor with a 10/100 Mbps Ethernet Media Access Controller (MAC) to develop a Merging Unit. The solution uses a 1PPS synchronisation signal from a GPS master clock and its accuracy is tested using the processor’s timer input capture method. The MU uses a multi-channel ADC which samples the analogue signals from the CTs and VTs and sends the data to the processor via a Serial Peripheral Interface (SPI) port. The use of the multi-channel ADC is innovative because only one ADC is used compared to using an ADC for each channel and the SPI port reduces the number of connection tracks between the ARM processor and the ADC.

Figure 2.7 shows a block diagram of hardware components which make up the MU developed by Liu *et al.*, (2007).

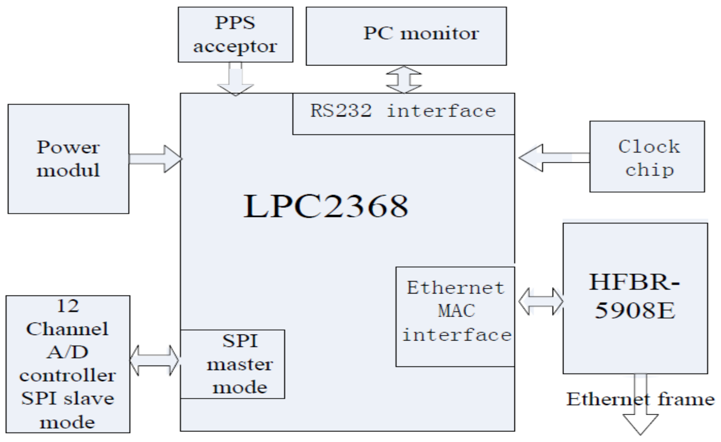


Figure 2.7: ARM processor-based Merging Unit (Adapted from Liu *et al*., 2007)

This developed MU transmits both the Manchester encoded FT3 frame and the SV packets specified in the IEC 60044-8 and the IEC 61850-9-1 standards respectively (Liu *et al.,* 2007).

The PHY controller is a transceiver chip representing the physical layer of the OSI stack and is responsible for analogue data transmission and reception of Ethernet frames (Liu *et al.,* 2007). The PHY used is the HFBR 5908E which is a high performance transceiver for optical fibre communications systems (Liu *et al*., 2007). Liu *et al.,* (2007) used the Microcontroller Operating system II (µC/OS-II) kernel as the core operating system and the MU applications are implemented as tasks.

These tasks are assigned execution priorities from 0-63. In this design there were three mains tasks, for receiving multi-channel data from the ADC, packing the SV packet and lastly sending the data onto the Ethernet network. In the implementation of the MU by Liu *et al.,* (2007), time synchronisation is interrupt-based and does not run off the main OS kernel. The results from the conducted test procedures prove that the developed microcontroller-based MU can be used in IEC 61850 substation automation systems (Liu *et al*., 2007).

Lee *et al.,* (2008) published a research paper detailing the required technique for designing a Merging Unit based an embedded processor and a real-time Linux kernel. Contrary to the design by Liu *et al.,* (2007), the MU design produced by Lee *et al.,* (2008), uses the IRIG-B protocol for synchronising the ADC sampling.

Lee *et al*., (2008) summarises that MUs have the following functions:

- Processing of signals from all sensors.
- Time synchronisation of measurements.
- Providing an interface between conventional/non-conventional instrument transformers and bay level IEDs.
- Transmission of SV frames to bay level controllers.

According to Lee *et al*., (2008), the MU sends data to SV subscribers using the SV multi-cast service, which is called the SendMSVMessage. The frames are captured and analysed using the MMS Ethereal software and the results prove that designing a MU using an embedded processor and the Linux real-time kernel is feasible and also conforms to the requirements of IEC 61850 standard.

Wei-ming *et al.,* (2011) conducted research into the design and development of a MU based on the Altera Cyclone III EPC 3C25Q240C8 FPGA. The code for the FPGA was written using the hardware description language for very-high speed integrated circuits (V*HSIC*-HDL). The developed Merging Unit has three functional modules; the time synchronisation, sampled value processing and the data frame transmission modules similar to the design by Yin & Liu, (2004).

Similarly to the approach by Yin & Liu, (2004) and Liu *et al.,* (2007); Wei-ming *et al.,* (2011) developed an FPGA-based MU synchronised using a 1PPS pulse from a GPS clock. The SV messages are framed and published as specified in the IEC 61850-9-2LE using the *MSVCB01* control block that defines a publishing sample rate of 80 samples per cycle and a dataset that contains four voltage and four current samples Wei-ming *et al.,* (2011).

The FPGA drives the LAN9215 PHY controller connected to an RJ-45 port and publishes SV packets at 100 Mbps full duplex mode (Wei-ming *et al.*, 2011). This MU design also has a fibre optic port controlled by the FPGA through the ICS 1889 PHY and HFBR 5103 fibre transceiver. According to Wei-ming *et al.,* (2011) the MMS Ethereal software is used to analyse the captured data frames and the results show that by taking advantage of the speed and modularisation of the FPGA a reliable IEC 61850-9-2 Merging Unit can be realised.

Weiss *et al.,* (2011) discuss the advantages of making conventional instrument transformer signals available on the process bus by making use of an IEC 61850-9-2 Merging Unit. The MU discussed in this paper generates and publishes SV messages as specified in the IEC 61850-9-2LE. Similarly to the research project by Wei-ming *et al.,* (2011), the MU detailed in the paper by Weiss *et al.,* (2011) publishes SV messages using the MSVCB01 control block but it can also generate and publish messages using MSVCB02 control block.

According to Weiss *et al.,* (2011), a MU solves the predefined loading characteristic problems experienced with conventional current transformers. Loading characteristics of a particular sensor means that only a limited number of IEDs can be connected to it without compromising its accuracy. This problem is solved by using Merging Units in an automation system because current and voltage samples are made available to any number of IEDs through the process bus with a burden of less than 1 VA applied to the instrument transformer (Weiss *et al.,* 2011).

Weiss *et al.,* (2011) suggest a number of recommendations which include the implementation of IEEE 1588 for time synchronisation instead of 1PPS because in the latter method (1PPS) an extra fibre cable is required to connect the master clock to the Merging Unit. Redundancy in a network is suggested where the network is ring-configured such that data can be sent in both directions. This setup guarantees a higher degree of confidence that SV messages will reach a subscriber IED even if one of the two communication links break. Lastly, Weiss *et al.,* (2011) recommend the development of Gigabit networks to reduce data loading on the process bus and increase transmission speeds.

A feasibility study was conducted by Jing *et al.,* (2011) into the application of MUs in intelligent distribution substations. This study was based on a hardware device designed to conform to the IEC 61850-9-1 standard. This research led to the design of a MU which interfaces to both conventional and electronic instrument transformers. The hardware platform for this Merging Unit is founded on an M4 Cortex-based 32 bit microprocessor from the STMicroelectronics STM32 family.

The Merging Unit designed presented in the paper by Jing *et al.,* (2011) uses the 1PPS signal from a GPS clock for synchronisation similar to the design by Yin & Liu, (2004), Liu *et al*., (2014), Wei-ming *et al*., (2011) and Weiss *et al.*, (2011).

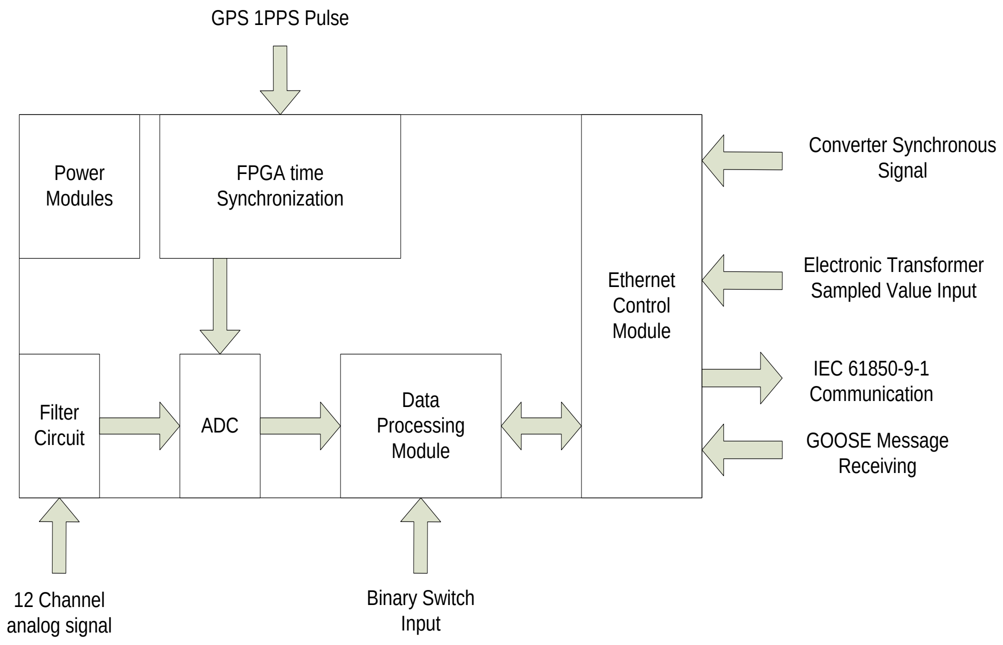


Figure 2.8: Merging Unit design block diagram (Adapted from Jing *et al.,* 2011)

Figure 2.8 shows a block diagram of the Merging Unit developed by Jing *et al.,* (2011). Table 2.5 provides a catalogue of research projects found in literature of the development of IEC 61850-9-1 and IEC 61850-9-2 Merging Units.

Table 2.5: **Comparison of different SV message-mapping algorithms and Merging Unit designs**

Paper	Summary	Standard Conformance	Advantages/Disadvantages	Development Platform
Yin & Liu, (2004).	Uses an FPGA platform to design a Merging Unit based on the IEC 61850-9-1 and the IEC 60044-8 standard to interface ECTs and EVTs with protection IEDs.	IEC 61850-9-1 IEC 60044-8	Reduction of complicated and parallel wiring used for connecting conventional CTs and VTs to the protection IEDs. Use of ECTs and EVTs compared to conventional instrument transformers.	FPGA
Liu *et al*., (2007).	In this paper a Merging Unit based on an ARM processor (LPC2368 MCU) is designed to interface ECTs and EVTs with protection IEDs	IEC 61850-9-1 IEC 60044-8	Advantages: • ECTs and EVTs are better than conventional CTs and VTs the latter device have magnetic saturation and insulation problems. • Reduction of parallel wiring between the bay level and the process level. • Compared to conventional instrument transformers, ECTs and EVTs have a wide dynamic range, are smaller in size and can be adapted easily to digital systems.	LPC 2368 ARM-based MCU
Xiaobin *et al.*, (2008).	Sampled value communication service mapping on a serial point-to-point link defined in IEC 61850-9-1 and IEC 60044-8 is analysed in this paper. Conventional instrument transformers are used.	IEC 60044-8 IEC 61850-9-1	Outcomes of the research: • This paper proposes the use of a Merging Unit to sample and transmit process information from the instrument transformers simultaneously. • The Merging Unit will reduce cabling costs in the substation and improve system reliability. • This Merging Unit provides remote control of equipment and also transmits digital status.	N/A
Schmid & Kunde (2009).	This paper investigates the use of Electronic instrument transformers in high voltage equipment. This paper also reviews communication standards defined for interfacing primary plant equipment with bay levels.	IEC 61850-9-1 IEC 60044-8	Advantage of using new Electronic Transformers: • Higher accuracy of 0.1% compared to their counterparts. • They can measure the transient short circuit current • Reduction in parallel copper wires between process and bay level. • They have a wider linearity and small core sections-thus smaller footprint.	N/A

Paper	Summary	Standard Conformance	Advantages/Disadvantages	Development Platform
Wei-ming *et al.,* (2011).	This paper analyses the structures of MUs and develops a merging unit capable publishing 1PPS time synchronized IEC 61850-9-2 sampled value messages from an FPGA.	IEC 61850-9-2	Advantages: • Use of electronic instrument transformers achieves better system performance compared to conventional instrument transformers in high voltage networks.	Altera Cyclone III EP3C25Q240C8 FPGA
Jing *et al.,* (2011).	This research focuses on the introduction of the IEC 61850 standard to improve data communication between the process level and the bay level. The authors design a Merging Unit which samples analogue signals from the instrument transformer and transmits them to bay level equipment as per the IEC 61850-9-1 and the IEC 60044-8 standards.	IEC 61850-9-1 IEC 60044-8	Advantages of using a merging unit for process level data communication are: • SV packets can be used for development of advanced functions for condition monitoring and protection functions. • Disconnection fault detection of conventional instrument transformer disconnection can be guaranteed.	STM32F103ZET6 MCU
Weiss *et al.,* (2011).	This paper discusses the advantages of using Merging Units for sending digital voltage and current samples to bay level IEDs.	IEC 61850-9-2	Advantages of using Merging Units: • Reduced wiring complexity for connecting instrument transformers to the IEDs because the data is available on the process bus network. • Increase in the number of IEDs to the same instrument transformers since there are maximum load issues.	N/A
Konka *et al.,* (2011).	In this research the authors focus on developing software that is used in the IEC 61850 standard-based measurement and testing product development.	IEC 61850-9-2	Advantages of this software: • Quickens product development and testing. • Reduces implementation time during commissioning by allowing engineers to use flexible software for defining different data models and acquisition techniques.	N/A
Zhengyang *et al.,* (2011).	In this paper the authors design a Merging Unit using an Altera FPGA with three main parts signal sampling, analysis of signal synchronisation and adjustable analogue sampling frequency.	IEC 60044-8 IEC 61850-9-1	Outcomes: • Advantages of using an FPGA for real-time design include high speed I/O and parallel threading capabilities.	Altera EP1C3T144 FPGA

Paper	Summary	Standard Conformance	Advantages/Disadvantages	Development Platform
Zhao, (2012).	This paper discusses the use of the process bus in a substation for transferring current and voltage samples from the CTs and VTs.	IEC 61850-9-2	Advantages of the SV message mapping: • Reduced complexity of wiring from instrument transformers to the control IEDs. • Reduces system engineering and configuration time.	Linux PC.
(Baranov *et al.*, (2013).	In this paper software for emulating SV packets according to IEC 61850-9-2 and IEC 61850-9-2LE is designed. This software transmits SV packets at 80 and 256 samples per cycle.	IEC 61850-9-2.	Advantages of emulating software: • Used to test equipment for development and commissioning purposes. • Provides insight into the IEC 61850-9-2 communication service mapping.	PC platform.
Luwaca, (2014)	Development of virtual sensor publishing IEC 61850-9-2LE SV message using Scapy	IEC 61850-9-2	Disadvantages: • The virtual sensor node does not have an analogue front-end for connection to a real-power system therefore the sampled values are simulated.	Linux PC

Figure 2.9 shows the number of research papers found in literature detailing the development of a Merging Unit as specified in both the IEC 61850-9-1 and the IEC 61850-9-2 standard for a period between 2004 and 2015. In this chart it can been seen that most developments were for Merging Units supporting the IEC 61850-9-1 standard and after 2012 all papers reviewed were on MUs supporting the IEC 61850-9-2LE.

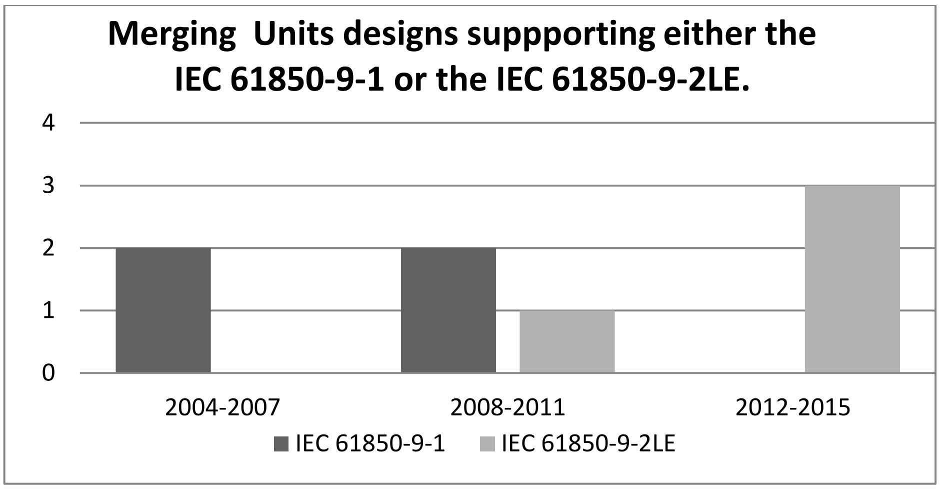


Figure 2.9: **Supported standards in the developed MU prototypes**

Form literature reviewed in the previous sections it was found out that before 2008 most Merging Units were based on the FPGA platform followed by microcontroller based units. This information is plotted in the chart in Figure 2.10.

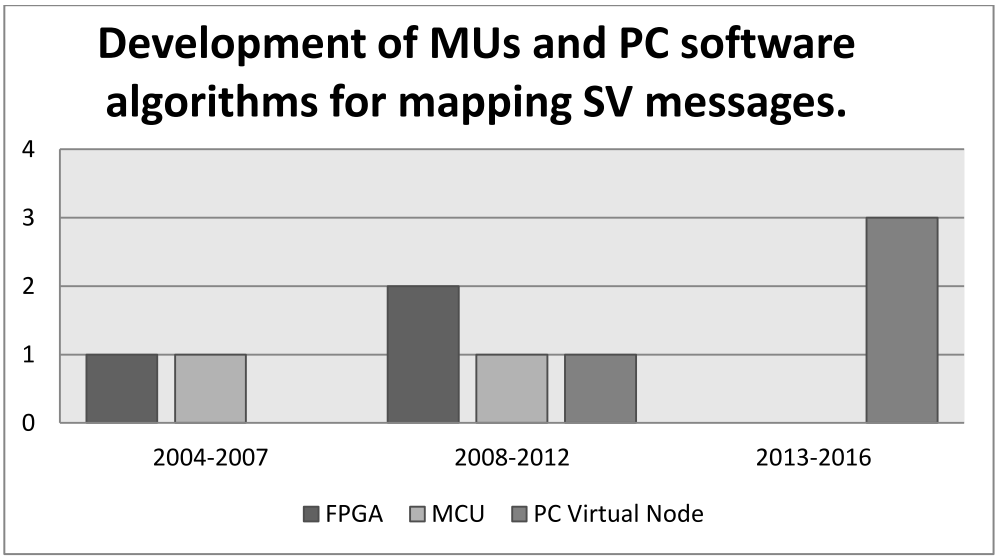


Figure 2.10: **Platforms used for developing Merging Units**

With reference to Figure 2.9 and Table 2.5, most MU designs found in literature supported the IEC 61850-9-1 standard which is less popular and is not supported by most protection IEDs compared to the IEC 61850-9-2 standard. Therefore unlike the research projects conducted by Yin & Liu, (2004), Liu *et al.,* (2007), Xiaobin *et al.,* (2008), Jing *et al.,* (2011) and Zhengyang *et al.,* (2011), this research project will develop a VHDL module for mapping sampled value messages as specified in the IEC 61850-9-2LE.

MUs developed in the past research projects were not tested for interoperaility with other IEC 61850-9-2-compliant IEDs but were tested using software applications. Therefore this research project will evaluate whether the SV messages published by the developed SV message mapping VHDL module can be subscribed to by the MiCOM P444 IED. This interoperability test will demonstrate the developed SV message mapping VHDL module is compliant to the IEC 61850-9-2LE.

This section has completed a comparative study of Merging Unit designs found in literature highlighting the different standards to which the MU generated and published the SV messages, different embedded platforms used and the test procedures conducted to evaluate the accuracy of samples or conformance to the referenced standard.

2.5 Conclusion

This literature review section has provided theoretical background information into this research project on the development of VHDL modules for mapping and publishing GOOSE and SV messages on an FPGA. This review section also looked at the application of the IEC 61850 process bus for the communication of data between entities in an automation system. This literature review chapter also highlighted advantages that the IEC 61850 process bus has over legacy protocols and/or hardwired communication architectures which include reduced system engineering costs and time.

According to Retonda-Modiya, (2012) and Luwaca, (2014) there are some electricity supply utilities in less developed countries whose substation automation systems are based on legacy protocols and hardwired communication architectures. This literature review has provided benefits that these utilities can take advtange of should they shift to the IEC 61850 process bus paradigm.

IEC 61850 standard-compliant IEDs are available off the shelf from different vendors and can be integrated into any substation automation system but developing such a device requires in-depth knowledge of this standard. The development of an embedded IED which can map and publish GOOSE and SV messages also requires knowledge of software development lifecycles, integrated development environments (IDEs) and IEC 61850 standard testing procedures for conformity checks.

From the literature reviewed, there are a number of research projects on the application of GOOSE and SV messages in substation automation systems compared to those on the development of IEDs capable of mapping and publishing these messages. With reference to Figure 2.4 and Figure 2.10, only four and nine research projects were conducted on the development of embedded GOOSE-enabled IEDs and SV message publishers respectively. Of the nine research projects on the development of SV message publishers, four of these projects were PC-based publishers capable of only publishing simulated sampled values and not real world values from the process equipment. Therefore this thesis will contribute towards the effective dissemination of information on the IEC 61850 standard and also on the design of embedded IEDs capable of publishing IEC 61850-9-2 SV and IEC 61850-8-1 GOOSE messages.

Chapter Three of this thesis discusses the IEC 61850 standard in great detail focusing on the communication service mapping of information into GOOSE and SV messages.

CHAPTER THREE

OVERVIEW OF THE IEC 61850 STANDARD

3.1 Introduction

The IEC 61850 standard was introduced in a bid to standardise communications in Substation Automation Systems (SAS) (Konka *et al*., 2011), and this standard has been extended for communications targeted at smart grid systems (Von Dollen, 2009). According to Zhao, (2012), the IEC 61850 standard has many advantages over hardwired and legacy communication techniques which include reduced system engineering time, standard data modelling providing data semantics and reduced implementation costs (Netto *et al*., 2012).

In order to design a Sampled Value (SV) message sand Generic Object Oriented Substation Event (GOOSE) message mapping VHDL modules on a Field Programmable Gate Array (FPGA) platform, in-depth knowledge of the IEC 61850 standard is required. This chapter provides an overview of the IEC 61850 standard suite with extensive focus on data modelling techniques and communication service mappings. This chapter will also focus on the application of the IEC 61850 standard in an SAS specifically for GOOSE and Sampled Value messaging.

This chapter is broken down into the following sections; section 3.2 provides a history of communication techniques used in legacy Supervisory Control and Data Acquisition (SCADA) systems and the advent of the IEC 61850 standard. Section 3.3 provides an overview of the ten parts of the IEC 61850 standard. Section 3.4 discusses the application of the IEC 61850 process bus and also the communication service mapping of data to SV and GOOSE messages. Section 3.5 concludes this chapter by highlighting the advantages of using the IEC 61850 standard in SAS over legacy communication protocols and hardwired techniques.

3.2 Introduction of the IEC 61850 standard

The IEC 61850 standard is a product of various initiatives initiated by players in the industry towards providing standardised communication systems in substations for automation purposes. Non-standardised communication techniques have been used in SCADA systems from the early 1990s' but they lacked interoperability and interchangeability between devices/RTUs. This lack of interoperability was due to the fact that different RTUs from different vendors did not support the same protocol. With such communication systems, the RTUs were single function devices which made it hard to implement distributed functions.

With legacy SCADA systems, expensive protocol translators were required to facilitate communications between RTUs supporting different protocols. The most commonly used protocols in substations towards the end of the 20th century were DNP3 and IEC 60870-5-101 (Luwaca, 2014). Another shortfall of hardwired communications and legacy protocols was that process equipment (circuit breakers and instrument transformers) were connected to multiple RTUs using copper wires. This resulted in expensive parallel copper wiring between RTUs and equipment (control relays and primary equipment) which led to high system engineering costs and time. Therefore there was a need to introduce a new standardised communication platform for devices used in the automation system.

The first edition of the IEC 61850 standard was then introduced in 2002 to cater for communication networks in substations. The IEC 61850 standard is an object oriented standard which resulted from a profile of recommended protocols for various layers of the International Standards Organisation (ISO) and the Open Systems Interconnect (OSI) communication models (Sun *et al*., 2012). IEC 61850 standard-based automation systems are easily configurable and expandable (Netto *et al.*, 2012).

The IEC 61850 standard for "Communication networks and systems in substations" provides:

- Standard naming convention,
- Standard meaning of data (semantics),
- Standard abstract services,
- Standard device behaviour models,
- Standard device configuration language and
- Mapping of data/information to various protocols using specific communication services (Mackiewicz, 2011).

Therefore the IEC 61850 standard provides device interoperability and simpler integration of power system functions in a distributed and cooperative manner (Mackiewicz, 2011). The following section discusses the ten parts of the IEC 61850 standard drawing attention to parts 8-1 and 9-2 related to GOOSE and SV messages respectively.

3.3 Overview of the IEC 61850 standard

The IEC 61850 standard suite has ten major parts which define different communication aspects, nomenclature, data models and timing requirements for data exchange in a substation automation system as illustrated in Figure 3.1.

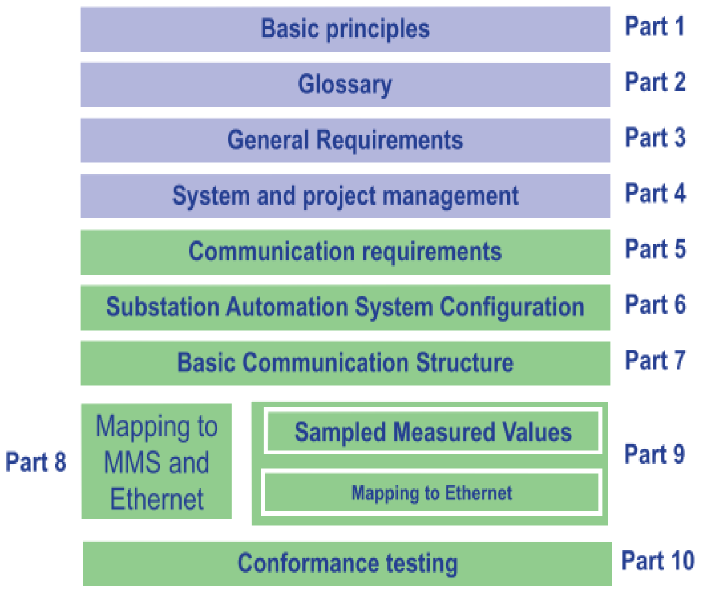


Figure 3.1: **The IEC 61850 standard suite (Adapted from Mackiewicz, 2012)**

The ten parts of the IEC 61850 standard are discussed in the following sections from section 3.3.1 to section 3.3.8.

3.3.1 Part 1 and Part 2

Part 1 and part 2 introduces nomenclature used throughout the rest of the standard and provides an overview of the IEC 61850 standard.

3.3.2 Part 3 and Part 4

These two parts of the IEC 61850 standard define specific function requirements directed to system integrators and protection engineers for communications in a substation automation system. These requirements include application profiles and the underlying lower level protocols.

3.3.3 Part 5

Part 5 of the IEC 61850 standard defines performance classes for the different types of messages used for mapping information (data objects and attributes) to specific protocols. Five communication profiles are defined in this standard of which three of these profiles are for time-critical messages known as link-layer communication services (Layer 2 of the OSI stack). These communication profiles are shown in Figure 3.2.

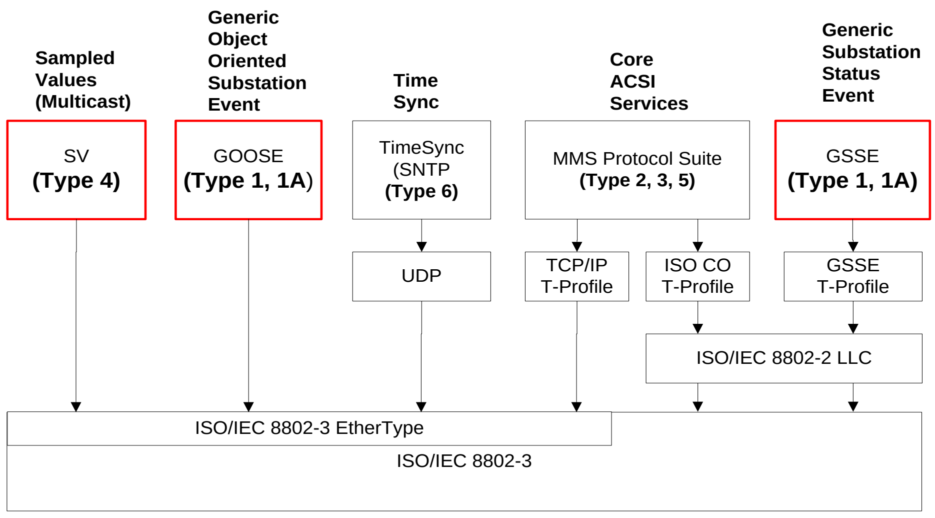


Figure 3.2: IEC 61850 standard message types and performance class (Adapted from Konka *et al.*, 2011)

The three time-critical communication services highlighted in red blocks in Figure 3.2 are the GOOSE, Generic Substation State Event (GSSE) and Sampled Value (SV) messages. These communication services are mapped directly to the data-link layer to reduce protocol overhead and therefore increase performance (Zhao, 2012). The remaining two communication profiles defined in this standard are the device time synchronisation messages (Time Sync) based on the Simple Network

Time Protocol (SNTP) and the Manufacturing Message Specification (MMS) profile for management of substation devices.

Of particular interest to this research project are the Type 1, Type 1A and Type 4 time-critical messages which are mapped to a specific EtherType registered to the Institute of Electrical and Electronic Engineers (IEEE). The IEC 61850-5 standard defines delivery times for different types of messages, all IEDs capable of publishing either one or multiple types of messages as defined in this standard should adhere to the defined delivery times.

Test procedures are provided in part 10 of this standard for evaluating the delivery times according to message type and performance class for the development of IEC 61850-compliant devices. The next section provides a summary of part 6 of the IEC 61850 standard.

3.3.4 Part 6

This part of the IEC 61850 standard introduces and discusses the Substation Configuration Language (SCL). The SCL is an eXtensible Markup Language (XML) based configuration language which allows system engineers to define abstract models of primary and secondary substation equipment, communication mechanisms between equipment and their relationship (Apostolov, 2010). The Unified Modelling Language (UML) is the base modelling platform used by this configuration language.

This configuration language allows IED configuration and settings to be passed to a system configuration tool or another IED. Four types of files are defined in the IEC 61850-6 standard for system engineering and these are the: System Specific Description (SSD), IED Capability Description (ICD), Station Configuration Description (SCD) and the Configured IED Description (CID) documents (International Electrotechnical Commission, 2003-2004; Mackiewicz, 2011; Sun *et al.*, 2012).

The SSD file describes the single line diagram of the distribution substation and contains logical nodes which define the functional model of the SAS. Functional models define the behaviour of the automation systems in carrying out a function e.g. protection, automation or control functions. The ICD file defines the default capability of the IED before configuration of its name and address while the CID file represents a configured IED. The difference between ICD and CID files is that the IED has been assigned a specific name and address. The SCD file represents a fully configured communication section for entity (IED) or sub-system interaction. This file contains all the IEDs available in the distribution substation which make up the automation system (Apostolov, 2010).

The next section summaries the four sub-parts of the IEC 61850-7 standard.

3.3.5 Part 7

This part of the IEC 61850 standard is sub-divided into four parts and these are 7-1, 7-2, 7-3 and 7-4. Part 7-1 provides the basic models and principles of communication between substation equipment in an IEC 61850 standard-based automation system. Part 7-2 discusses how data is exchanged between entities using abstract communication service interfaces. Part 7-3 defines structures for describing common data attributes for a specific data object; these structures are known as Common Data Classes (CDCs) (Mackiewicz, 2011).

Part 7-4 defines logical node classes with their compatible common data classes that can be mapped to a specific communication service mapping for information exchange between entities. Section 3.3.5.1 to 3.3.5.3 discuss the different aspects of part 7 of the IEC 61850 standard under the subheadings abstract communication services, data modelling and the IEC 61850 standard naming convention.

3.3.5.1 Abstract Communication Services

Part 7-2 of the IEC 61850 standard defines the Abstract Communication Service Interface (ACSI). The ACSI defines how data/information is communicated between devices within a substation automation system, for example, how the voltage and current samples are published by a Merging Unit (Wei-ming *et al*., 2011). The ACSI is divided into two parts which define two communication models and these are the client-server and the peer-to-peer models. The information/data exchanged using these communication services is modelled following the data modelling techniques defined in part 7-3 and part 7-4 of this standard.

In a peer-to-peer communication model, one device is a publisher (transmits data) and the other a subscriber (listens for information); this model allows information transmitted by the publisher to be received by one subscriber using a unicast address or multiple subscribers using a multicast address. The client-server model is used mostly for device configuration and remote access while the peer-to-peer model is used for the exchange of time-critical messages. Figure 3.3 illustrates the two different communication models between devices/functions.

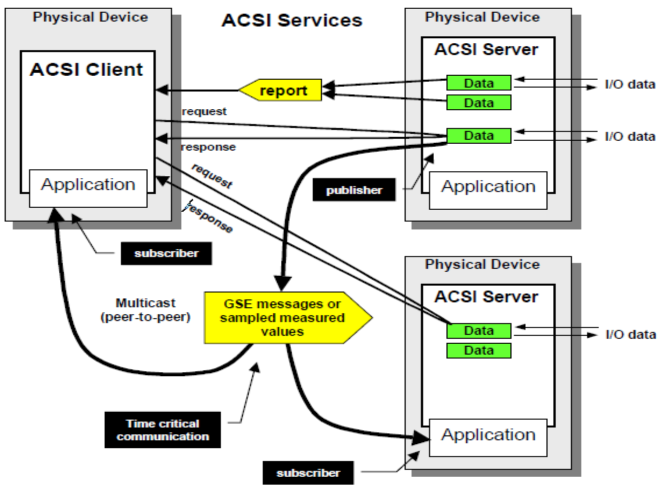


Figure 3.3: **ASCI services for device/distributed function communication (Adapted from International Electrotechnical Commission, 2003-2007)**

The modelled information/data is mapped to well-known communication protocols like TCP/IP, MMS and ISO 8802-3 Ethernet frame using Specific Communication Service Mappings (SCSMs). SCSMs define the underlying communication protocols and standards used for the exchange of application data between devices in either a client-server or peer-to-peer communication model. An example of a SCSM is defined in part 9-2 of the IEC 61850 standard; this definition is for the transmission of sampled value messages using an ISO 8802-3 Ethernet frame.

The next section discusses the data modelling of substation information using logical nodes and common data classes.

3.3.5.2 Data modelling

Data models used in the IEC 61850 standard are representations of real analogue power system equipment, these data models are developed through a process of virtualization. Virtualization is a process of providing a view of the aspects of a real device that are of interest to the automation system. For example, the position and command block status of the circuit breaker are some of the important attributes that are needed in the implementation of a circuit breaker monitoring and control scheme. Figure 3.4 shows an illustration of a circuit breaker virtualization process.

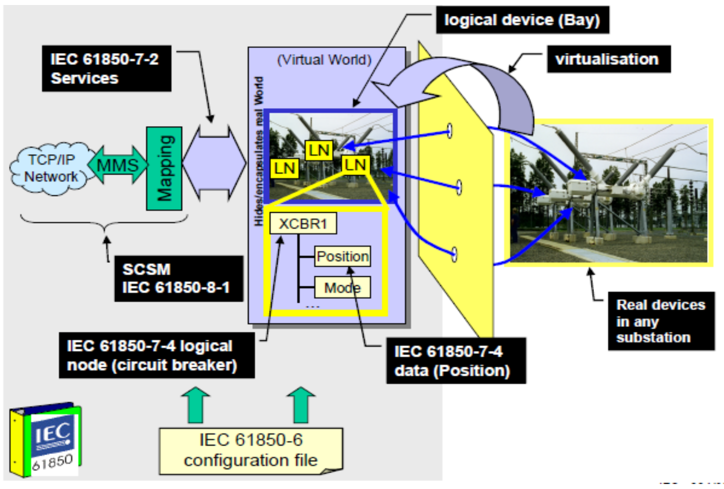

logical device (Bay)
IEC 61850-7-2 Services
(Virtual World)
virtualisation
TCP/IP Network
MMS
Mapping
Hides/encapsulates real World
LN
LN
LN
XCBR1
Position
Mode
SCSM IEC 61850-8-1
Real devices in any substation
IEC 61850-7-4 logical node (circuit breaker)
IEC 61850-7-4 data (Position)
IEC 61850-6 configuration file
IEC 61850

Figure 3.4: **Data modelling through virtualization**

(Adapted from International Electrotechnical Commission, 2003-2004)

Through this virtualization process, standard logical nodes are defined (e.g. XCBR) representing power system equipment (e.g. circuit breaker) and the corresponding data objects (e.g. circuit breaker position).

As illustrated in Figure 3.4, the IEC 61850 standard allows for modularisation of application functions into LNs and then using them to facilitate communication between devices. Logical nodes consist of Data Objects (DOs) which are instances of Common Data Classes (CDCs) which in turn consists of Data Attributes (DA). Logical Nodes (LNs) allows an IEC 61850 standard-based automation system to support distributed protection and communication functions (International Electrotechnical Commission, 2003-2004).

The IEC 61850 standard defines ready to use LNs for common functions, for example, circuit breaker control and measurement functions. Data modelling methods coupled with standardised communication techniques offer interoperability between devices from different manufactures with the prospect of interchangeability. Thirteen LN groups are defined in the standard that extends over ninety-two logical node classes and these groups are shown in Table 3.1.

Table 3.1: Logical Node groups in the IEC 61850-7-1 standard (Adapted from International Electrotechnical Commission, 2003-05)

Logic Node groups	Number of Logic Nodes
System Logic Nodes	3
Protection Function	28
Protection related function	10
Supervisory control	5
Generic references	3
Interfacing and archiving	4
Automatic control	4
Metering and measurement	8
Sensor and monitoring	4
Switchgear	2
Instrument transformer	2
Power transformer	4
Further power system equipment	15
Total number of logical nodes	92

Using the example of the circuit breaker virtualization, the XCBR logical node contains the Pos and BlkOpn data objects which represent the position and operation block flag respectively of the power plant circuit breaker.

Figure 3.5 shows the data modelling of a circuit breaker as defined in the IEC 61850 standard.

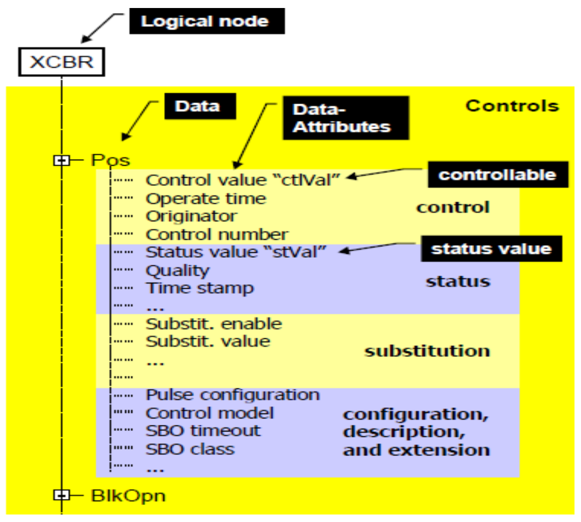


Figure 3.5: **Illustration of the data model structure using XCBR logical node (Adapted from International Electrotechnical Commission, 2003-2007)**

The Pos and BlkOpn data objects in the XCBR LN are instances of the Controllable Double Point (DPC) and Controllable Single Point (SPC) common data classes defined in IEC 61850-7-3 standard.

The data model can be further explained using Figure 3.6, this illustration shows an object oriented view of the IEC 61850 standard data modelling technique. With reference to Figure 3.6, a physical network addressable device contains a logical device which in turn contains multiple logical nodes for different functions and in this case, circuit breaker monitoring/control and single phase current measurement.

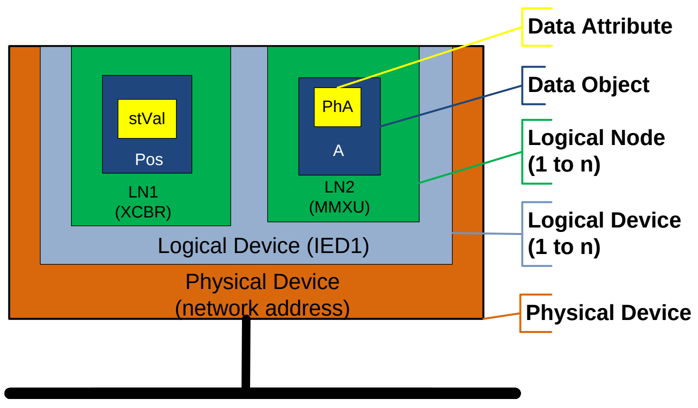


Figure 3.6: **Data model layers defined in the IEC 61850 standard (Adapted from Sun *et al.*, 2012)**

In Figure 3.6 the data model is layered with standardised names; the layers are described as follows:

- ***Physical Device***: identifies the actual network addressable device (IED) in an automation system.
- ***Logical Device***: groups of related Logical Nodes within a physical device.
- ***Logical Node***: This refers to a grouping of related data and associated services to produce some power system function e.g. circuit breaker monitoring and control represented by the XCBR logical node.
- ***Data Object****:* this is the actual data that is being measured, monitored or controlled in an automation system; this object is an instance of the common data class (CDC) or a user defined class.
- ***Data Attribute***: this refers to a characteristic of the data object being monitored or measured for instance, the status value (stVal) indicating the position of Pos data object of the XCBR logical node.

The IEC 61850 standard follows an object-oriented modelling approach based on the Unified Modelling Language (UML). Once a class or function has been defined, the user does not need to redefine the object or function but can create an instance of that class, an object. Figure 3.7 shows the UML class diagram of the IEC 61850 data model.

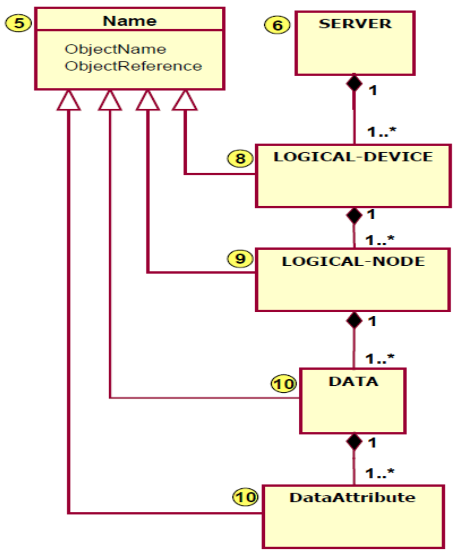


Figure 3.7: UML class diagram of the IEC 61850 data model

(Adapted from International Electrotechnical Commission, 2003-05)

The IEC 61850 standard is not a rigid as it allows for expansion of existing data classes by following the naming conventions and the virtualization process. This research project will use standard logical nodes and these are the TCTR, TVTR, XCBR and the MMXN for current transformers, voltage transformers, circuit breakers and single phase measurements respectively.

The next section defines the naming convention introduced by the IEC 61850 standard for identifying devices and data.

3.3.5.3 The IEC 61850 standard naming convention

Another important factor introduced by the IEC 61850 standard is the standard naming convention for devices, logical nodes, objects and data attributes. The naming convention eliminates ambiguity of named devices and/or objects within a system. For instance, using the example discussed before of the XCBR logical node, the attribute stVal of the Pos data object can be referenced as shown in Figure 3.8.

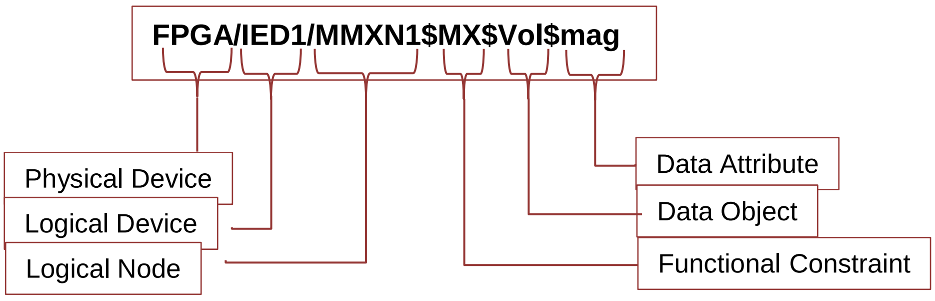


Figure 3.8: Naming convention defined in the IEC 61850 standard (Adapted from Retonda-Modiya, 2012)

The name of the Logical Device (LD) is not defined within the IEC 61850 standard so therefore it can be assigned locally by the user. Functional Constraints (FCs) are used to group objects according to function and in the example in Figure 3.8 the constraint is MX which means that the data attribute is a measured value.

In conclusion, Part 7 of the IEC 61850 standard defines information models, communication services and data classes for data/information communication between devices in an IEC 61850 standard-based system. The next section discusses part 8-1 of the IEC 61850 standard; this part defines the specific communication service mapping of application data to Manufacturing Message Specification (MMS) and ISO 8802-3 GOOSE messages.

3.3.6 Part 8-1

This section of the IEC 61850 standard defines the specific communication service mapping (SCSM) of the ACSI and the information models to a specific protocol. Part 8-1 focuses on mapping data objects and ACSI to MMS (ISO 9506-1 and ISO 5906-2) and also to an ISO 8802-3 Ethernet frame (International Electrotechnical Commission, 2004-2005). The mapping of ACSI and application data to ISO 8802-3 frame results in a time-critical peer-to-peer transmission service called Generic Object Oriented Substation Event (GOOSE) messages.

Figure 3.9 shows how the application data modelled in part 7 of the IEC 61850 standard is exchanged using specific communication service mappings defined in parts 8-1, 9-1 and 9-2. The mapping of data and information to MMS covers the seven layers of the OSI stack while on the ISO 8802-3 frame the data is mapped directly to the OSI link-layer (Ethernet). Figure 3.9 shows the relationship between the specific communication service mappings, underlying protocols and the data models in terms of the IEC 61850 standard.

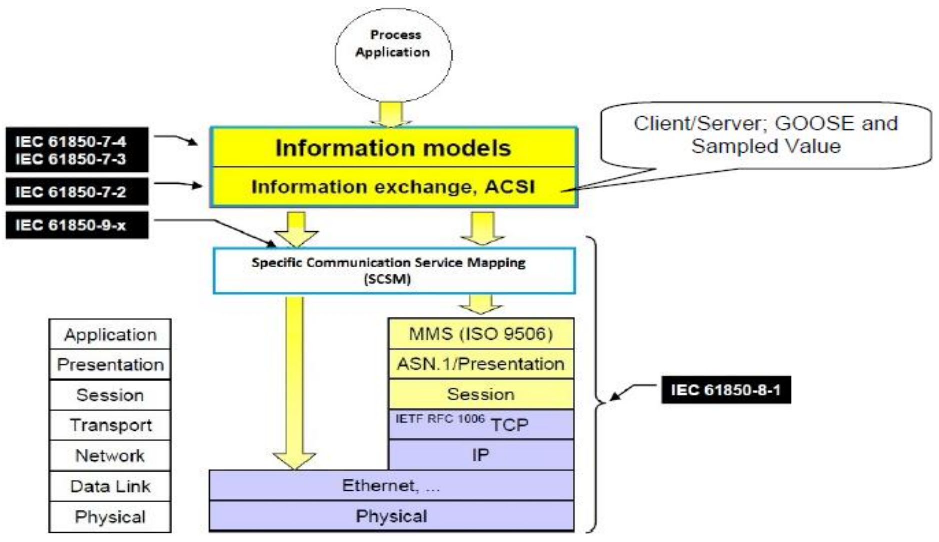


Figure 3.9: **Mapping of data models and information to SCSM (Adapted from International Electrotechnical Commission, 2003-05)**

GOOSE messages are mapped directly to the data link layer to reduce protocol overhead and message delivery times. GOOSE messages mapped to the ISO 8802-3 frame use a specific EtherType registered to the Institute of Electrical and Electronic Engineers (IEEE). The structure and application of GOOSE message will be discussed in detail in section 3.4.1.

The next section introduces the mapping of application for the transmission of current and voltage samples from primary plant equipment.

3.3.7 Part 9-1 and 9-2

This section of the IEC 61850 standard defines specific communication service mapping (SCSM) for transmitting current and voltage samples over a serial point-to-point link or over an Ethernet process bus to bay level IEDs. Part 9 is sub-divided into two parts, part 9-1 and 9-2; part 9-1 defines mapping of information over a serial uni-directional multi-drop point to point link while part 9-2 defines mapping onto an ISO 8802-3 Ethernet frame. The mapping process of the information model is similar to that defined in part 8-1 of the standard illustrated in Figure 3.9.

Similar to GOOSE messages, mapping of voltage and current samples to the ISO 8802-3 Ethernet frame results in a time-critical peer-to-peer transmission service called Sampled Value messages (SV).

Part 9-1 of the IEC 61850 standard differs from part 9-2 of the same standard not only in the underlying interface but also in the information exchanged. Part 9-1 encapsulates both digital inputs and voltage/current measurements in the same frame. Part 9-2 only allows measured voltage and current samples in the published frame because GOOSE messages are already made available for digital status communication. These two parts of the IEC 61850-9 standard introduce a process level device known as the Merging Unit (MU), this MU measures and transmits voltage and current samples from the voltage and current transformers to bay level devices.

The dataset specified in the IEC 60044-8 is adopted as the preconfigured dataset in an IEC 61850-9-1 standard-based Merging Unit, this dataset contains three phase voltage samples, the bus voltage, the neutral voltage, two status words (16 binary inputs), three phase currents for protection purposes and also three phase current samples for measurement (International Electrotechnical Commission, 2002-07).

Part 9-2 of the IEC 61850 standard is restricted to the transmission of sampled value (SV) packets without binary inputs statuses contrary to Part 9-1. In the context of the IEC 6180-9-2 standard, the dataset is configurable using the SCL defined in part 6 of the standard (International Electrotechnical Commission, 2004; Sun *et al*., 2012). Due to the complexity and ambiguity of the IEC 61850-9-2 standard, the UCAIug implementation guideline was created by major vendors in industry to reduce the implementation ambiguity and time to market of MUs by defining the sample rates, VLAN tag priority, sampled value control blocks and the dataset etc. This implementation guideline is known as the IEC 61850 Light-Edition (IEC 61850-9-2LE) (Ingram *et al.,* 2012).

This research project will only focus on sampled value message-mapping as defined in IEC 61850-9-2LE. The VHDL module developed in this research will publish IEC 61850-9-2LE SV messages because most IEDs can subscribe to these messages. The concept of the Merging Unit, IEC 61850-9-2LE sampled value message structure, mapping and transmission will be discussed in section 3.4.2

The following section gives a brief overview of part 10 of the IEC 61850 standard.

3.3.8 Part 10

This section of the IEC 61850 standard defines testing methods to determine device conformance to numerous protocol definitions and timing constraints. The evaluation techniques defined in this part will allows for proper use and easy integration of IEDs into substation automation applications as intended (International Electrotechnical Commission, 2005-05).

The next section introduces the application of the IEC 61850 standard in an automation system focusing on process bus communication.

3.4 Application of the IEC 61850 standard in a Substation Automation System

The IEC 61850 standard has been introduced to standardise communication systems in a distribution substation automation system. This research project focuses on the application of the IEC 61850 standard in substation automation systems and in particular the process bus. As discussed in Chapter Two, the process bus is made up of two peer-to-peer protocols for real-time data transmission and these are the GOOSE and the Sampled Value (SV) messages defined in Part 8-1 and Part 9-2.

According to the IEC 61850 standard, an SAS is split into three distinct levels that is the station, bay and the process levels as illustrated in Figure 3.10.

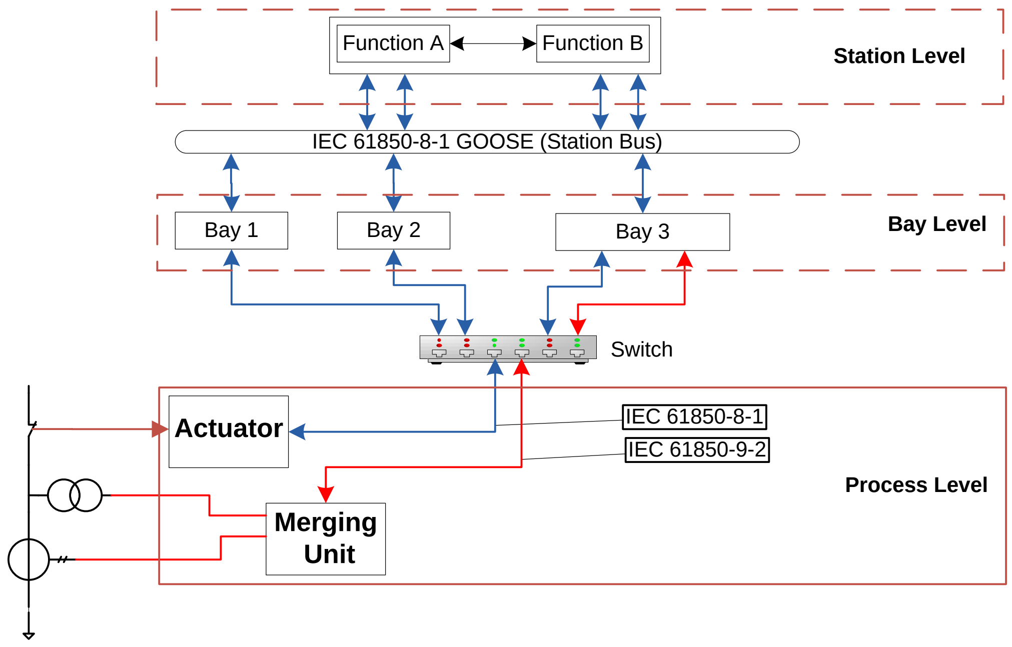


Figure 3.10: **Hierarchy within a Substation Automation System (Adapted from Zhao, 2012)**

In legacy systems, expensive parallel copper wiring was required to connect primary plant equipment to multiple RTUs for protection and measurement functions; this setup was costly to implement, unreliable and rigid. Wiring analogue signals from the primary plant equipment was also an issue because some constraints had to be considered for example the maximum length of the copper wire before the signal is attenuated, size of the burden in case of instrument transformers and the noise level in the environment.

The IEC 61850 process bus allows primary plant equipment measurements and status to be transmitted to bay level IEDs via an Ethernet network. The Ethernet technology has been developed from the older CSMA/CD (Carrier Sense Multiple Access with Collision Detection) paradigm to a native switched-base mechanism suitable for real-time data transmission. This native switched-base mechanism is almost collision free and can be used with deterministic transmission times (Sun *et al.,* 2012).

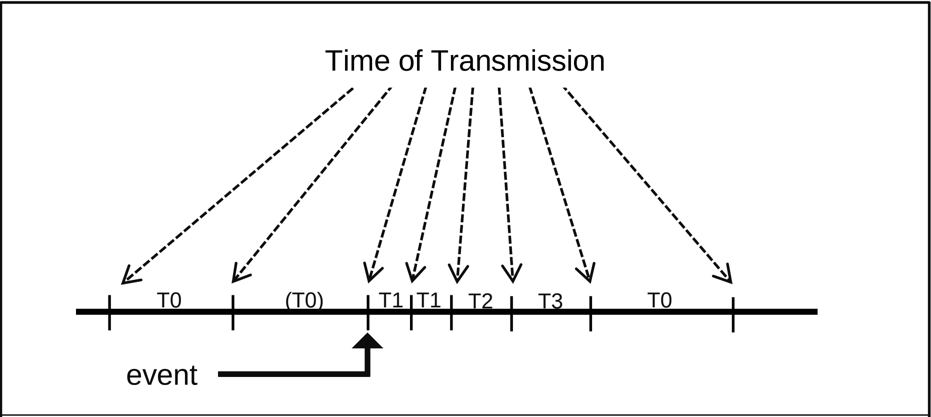
The IEC 61850 process bus also minimises the calibration of devices and system maintenance costs by reducing the complex wiring and in turn reducing the system engineering time; this process bus makes the automation system more reliable, flexible and easily expandable. With this process bus, analogue measurements can be transmitted long distances using SV messages without worrying about signal attenuation.

The following sections, section 3.4.1 and section 3.4.2 discuss the GOOSE and the SV messages structure and transmission procedures in detail.

3.4.1 IEC 61850-8-1 standard GOOSE Messages

Generic Object Oriented Substation Event (GOOSE) messages are exchanged between IEDs using a peer-to-peer communication model; these messages can be multicast to multiple IEDs or directed to a specific IED using a unicast address. GOOSE messages report the instantaneous status/value of data objects monitored or measured by an IED. These GOOSE messages can be used to relay digital and/or analogue information between sub-systems/functions, for example, in a protection scheme the circuit breaker status and voltage measurements are exchanged between functions.

GOOSE messages are transmitted continuously by a GOOSE client after a specific interval, *MaxTime*. *MaxTime* is a period configured in the client device (IED) which prompts the goose control block (GoCB) to publish messages when this period expires, this message retransmission scheme acts as a heartbeat message for subscriber IEDs. When a substation event occurs, that is a change in the status/value of one or more data objects in the GOOSE dataset, a GOOSE message is published and the retransmission time is reduced to its minimum (*MinTime*). GOOSE messages are retransmitted when the interval time (*MinTime*) expires; the interval time (*MinTime*) is gradually increased until *MaxTime* is reached or another event occurs. This repetitive process is illustrated in Figure 3.11.



T0	retransmission in stable conditions (*MaxTime)*
(T0)	retransmission in stable conditions may be shortened by an event
T1	shortest retransmission time after the event (*MinTime*)
T2, T3	retransmission times until achieving the stable conditions time

Figure 3.11: **Repetitive transmission of GOOSE messages (Adapted from International Electrotechnical Commission, 2004-2005)**

The *MaxTime* and *MinTime* values are not defined in the IEC 61850-8-1 standard which means that they can be defined locally by the user or manufacturer. The IEC 61850-8-1 standard caps the maximum value of *MaxTime* to 60 seconds (International Electrotechnical Commission, 2004-2005). With reference to Figure 3.11, *MaxTime* and *MinTime* are represented by the times T0 and T1 respectively.

GOOSE message services and datasets are controlled by the GOOSE Control Block. Control blocks define the rate at which data is communicated and also how it is communicated between IEDs using abstract communication services.

The GOOSE Control Block (GoCB) defines a reference to a dataset that contains data objects to be transmitted; this GOOSE Control Block (GoCB) class is shown in Table 3.2.

Table 3.2: **IEC 61850-8-1 GOOSE Control Block (GoCB) class (Adapted from International Electrotechnical Commission, 2003-05)**

GoCB Class				
Attribute name	**Attribute type**	**FC**	**TrgOp**	**Value/Value range**
GoCBName	ObjectName	GO	-	Instance name of an instance of GoCB
GoCBRef	ObjectReference	GO	-	Path-name of an instance of GoCB
GoEna	BOOLEAN	GO	dchg	Enabled (TRUE) | disabled (FALSE)
AppID	VISIBLE STRING65	GO	-	System wide identification
DatSet	ObjectReference	GO	dchg	
ConfRev	INT32U	GO	dchg	
NdsCom	BOOLEAN	GO	dchg	
Services SendGOOSEMessage GetGoReference GetGOOSEElementNumber GetGoCBValues SetGoCBValues				

The *DatSet* attribute in the GOOSE control block references a combination of data objects grouped together into a dataset whose values or status are published after an interval or when they change. The ACSI services defined for GOOSE messaging are mapped to the Specific Communication Service (SCSM) for MMS stack or ISO 8802-3 Ethernet frame. Of particular interest to this research project is the SendGOOSEMessage service which is mapped onto the ISO 8802-3 frame. The SendGOOSEMessage service publishes the data in an unsolicited manner once the GOOSE control block is enabled by setting the *GoEna* attribute to TRUE (International Electrotechnical Commission, 2004-2005).

GOOSE messaging is a reliable multicasting messaging scheme which does not require data acknowledgements from the receiver (International Electrotechnical Commission, 2004-2005). The GOOSE publisher/client maintains a finite state machine which controls the transition of the state number (*stNum*) and sequence number (*sqNum*) of the GOOSE messages.

This GOOSE publisher/client state machine is shown in Figure 3.12.

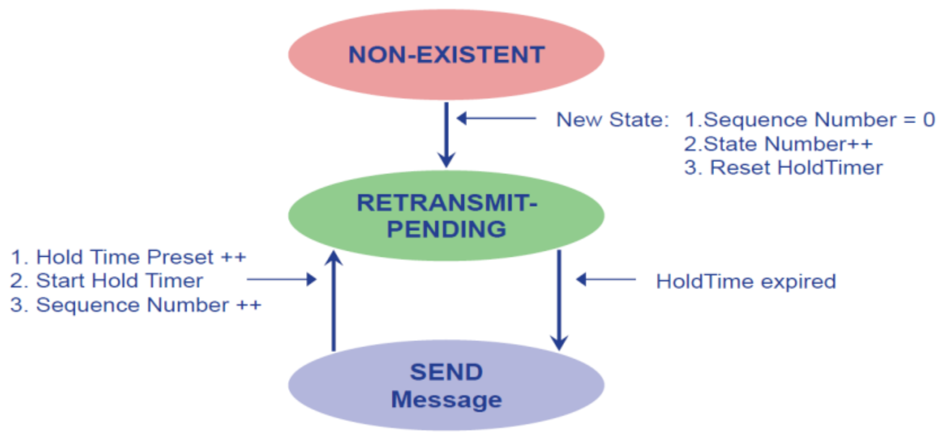


Figure 3.12: GOOSE client state machine

(Adapted from Mackiewicz, 2011)

The next section discusses the structure of GOOSE messages as defined in IEC 61850-8-1 standard.

3.4.1.1 GOOSE Message Structure.

GOOSE message are mapped onto the ISO 8802-3 Ethernet frame and the Protocol Data Unit (PDU) is embedded onto data payload section of this Ethernet frame, the structure of the ISO 8802-3 Ethernet frame used is shown in APPENDIX B. The ISO 8802-3 frame contains fields discussed in the following list:

- ***MAC Destination address:*** This is the MAC address of the destination device. In multicast addressing, this value should range from 01-0C-CD-01-00-00 to 01-0C-CD-01-01-FF. The first 3 bytes are assigned by IEEE and the fourth byte, 01 is set aside for GOOSE message transmission (Mackiewicz, 2012).
- ***MAC Source Address***: this is the MAC address of the GOOSE publisher/client.
- ***VLAN Tag***: GOOSE frames are tagged using the IEEE 802.1Q to separate time critical messages from low priority data. The Tag Protocol Identifier (TPID) is set 0x8100 for identifying IEEE 802.1Q tagged frames from normal frames. Similar to SV messages, GOOSE messages are assigned a default

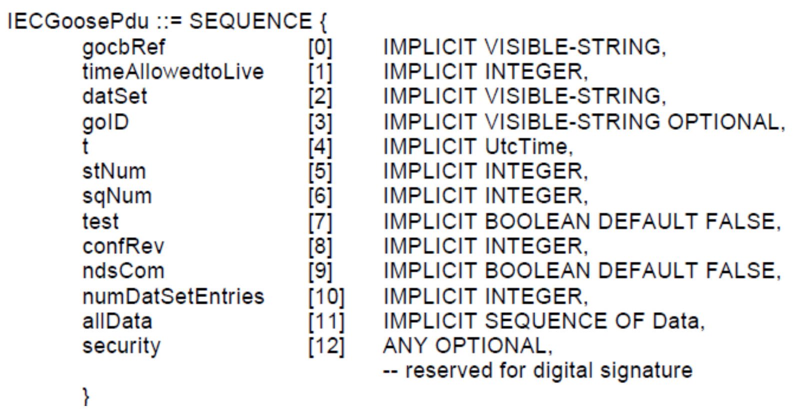
priority of 4 and a VLAN ID (VID) of 0. The structure of the tag header is defined in Table 3.3.

Table 3.3: **Structure of the IEEE 802.1Q Tag header**
Adapted from (International Electrotechnical Commission, 2004-2005)

Octets		8	7	6	5	4	3	2	1
0	TPID	0x8100							
1									
2	TCI	User priority			CFI	VID			
3		VID							

- ***EtherType information***: The EtherType for GOOSE messages is 0x88B8 and the APPID by default is 0x0000. The Application ID (APPID) is used to distinguish between Ethernet frames containing SV, GOOSE, GSSE and other messages. According to the IEC 61850-8-1 standard the GOOSE APPID can be any value between 0x0000 to 0x3FFF (International Electrotechnical Commission, 2004-2005).

The IEC 61850-8-1 standard defines a GOOSE PDU which contains message identifiers and the dataset embedded onto the payload section of the ISO 8802-3 frame. Similarly to the SV APDU, the GOOSE message PDU is encoded using the basic rules of the ASN.1 standard, this PDU is illustrated in Figure 3.13



```
IECGoosePdu ::= SEQUENCE {
        gocbRef                 [0]     IMPLICIT VISIBLE-STRING,
        timeAllowedtoLive       [1]     IMPLICIT INTEGER,
        datSet                  [2]     IMPLICIT VISIBLE-STRING,
        goID                    [3]     IMPLICIT VISIBLE-STRING OPTIONAL,
        t                       [4]     IMPLICIT UtcTime,
        stNum                   [5]     IMPLICIT INTEGER,
        sqNum                   [6]     IMPLICIT INTEGER,
        test                    [7]     IMPLICIT BOOLEAN DEFAULT FALSE,
        confRev                 [8]     IMPLICIT INTEGER,
        ndsCom                  [9]     IMPLICIT BOOLEAN DEFAULT FALSE,
        numDatSetEntries        [10]    IMPLICIT INTEGER,
        allData                 [11]    IMPLICIT SEQUENCE OF Data,
        security                [12]    ANY OPTIONAL,
                                        -- reserved for digital signature
        }
```

Figure 3.13: **GOOSE PDU as defined in the IEC 61850-8-1 standard**
(Adapted from International Electrotechnical Commission, 2004-2005)

These GOOSE PDU fields are described in the following paragraphs:

i. ***gocbRef***

This is a visible string identifier which contains a reference to the GOOSE control block controlling the publication of these GOOSE messages.

ii. ***timeAllowedtoLive (TAL)***

For every GOOSE message published, the *timeAllowedtoLive* contains the time in milliseconds for which the subscribing device has to wait for the next GOOSE message. If a message hasn't been received after this time has elapsed then the subscriber assumes that the publisher association has been lost (International Electrotechnical Commission, 2004-2005).

iii. ***t***

This field was mapped from EntryTime field in the IEC 61850-7-2 GOOSE message field to Timestamp field in the IEC 61850-8-1 GOOSE message. This field contains the Universal Co-ordinated Time (UTC) encoded according to RFC-1305 of the time the GOOSE message was generated. The structure of the *Timestamp* field is illustrated in Table 3.4.

Table 3.4: Structure of the Timestamp field
(Adapted from International Electrotechnical Commission, 2003-05)

TimeStamp type definition			
Attribute Name	**Attribute type**	**Value/value range/explanation**	**M/O**
SecondsSinceEpoch	INT32	(0 … MAX)	M

FractionOfSecond	INT24U	Value = SUM from i=0 to 23 of bi*2**-(i+1);
		Order = b0, b1, b2, b3, …
			M

TimeQuality	TimeQuality		M

The SecondsSinceEpoch is a 32 bit integer representing the number of complete whole seconds since Epoch (1st of January 1970 at 00:00:00) and the FractionOfSecond field is a 24 bit field which represents the fractions of a second that have elapsed since the last whole second. The maximum resolution of the FractionOfSecond field is calculated using Equation 3.1.

$$\mathbf{resolution}(s) = \frac{\mathbf{1}}{\mathbf{2^{24}}} = \mathbf{60\ ns}$$

Equation 3.1

Even if the FractionOfSecond field allows for resolutions of up to 60 ns the resolution of the GOOSE publisher's timestamp relies solely on the device manufacturer and is outside the scope of the IEC 61850-7-2 and IEC 61850-8-1 standards (International Electrotechnical Commission, 2003-2005; International Electrotechnical Commission, 2004-05).

Lastly, the Timestamp field contains the TimeQuality field that reports the publisher IED's time source as shown in Table 3.5.

Table 3.5: IEC 61850-8-1 standard TimeQuality definition
(Adapted from International Electrotechnical Commission, 2003-05)

TimeQuality definition			
Attribute Name	**Attribute type**	**Value/value range/explanation**	**M/O**
	PACKED LIST		
LeapSecondsKnown	BOOLEAN		M
ClockFailure	BOOLEAN		M
ClockNotSynchronised	BOOLEAN		O
TimeAccuracy	CODED ENUM	Number of significant bits in FractionOfSecond	M

iv. stNum

This integer variable represents the state number of the client's state machine. This value is incremented every time an event occurs.

v. sqNum

This integer value represents the sequence number for message retransmission after the occurrence of an event. After an event has occurred, this value is incremented until the next event occurs or until the retransmission time equals the stable retransmission time. Transition of *stNum* and *sqNum* is illustrated in Figure 3.14.

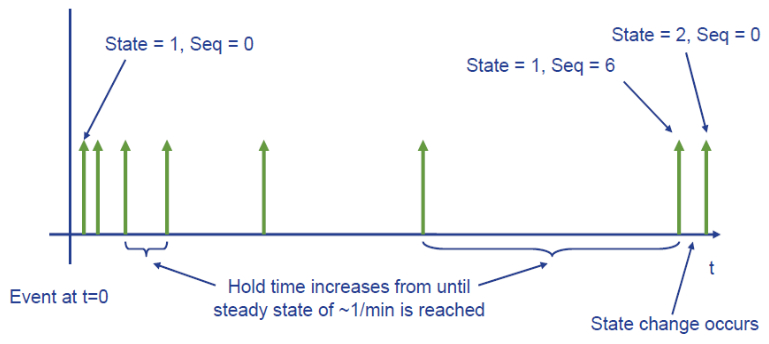


Figure 3.14: **Operation of *sqNum* and *stNum***

(Adapted from Mackiewicz, 2012)

vi. Simulation/Test

This is a BOOLEAN flag which represents whether the messages published are from a valid application or they are generated from a test operation. This flag informs subscribers not to use these GOOSE messages for any functions.

vii. confRev

This value refers to the configuration revision number of the GOOSE control block at the time of GOOSE message transmission (International Electrotechnical Commission, 2004-2005; Mackiewicz, 2011). This value will be incremented when the data elements within the dataset are re-ordered, removed or added.

viii. ndsComm

This flag indicates whether the GOOSE client/publisher requires commissioning or not.

ix. numDataSetEntries

This is the number of data object entries in the dataset to be mapped into the GOOSE message.

x. DataSet

Datasets are organised grouping of data objects or data attributes called dataset members. This grouping of data or data attributes into dataset is for convenience of the publisher because all the dataset elements can be transmitted in one message thus efficiently utilising the communication bandwidth.

The next section discusses the SV message structure and transmission procedures defined in the IEC 61850-9-2 standard.

3.4.2 IEC 61850-9-2 SV messages according to IEC 61850-9-2LE

Part 9-2 of the IEC 61850 standard defines a user configurable SV message frame; this frame can be configured using the substation configuration language. The UCAIug implementation guideline, commonly referred to as the IEC 61850-9-2 Light Edition simplifies the implementation of the IEC 61850-9-2 communication service mapping by defining datasets, ASCI, physical connections and sample rates (Ingram *et al*., 2013).

This research project focuses on sampled value message-mapping and transmission procedures as specified in the IEC 61850-9-2LE. IEC 61850-9-2LE SV messages are mapped to an ISO 8802-3 Ethernet frame which consists of a header and the SV Application Protocol Data Unit (APDU) as illustrated in Figure 3.15.

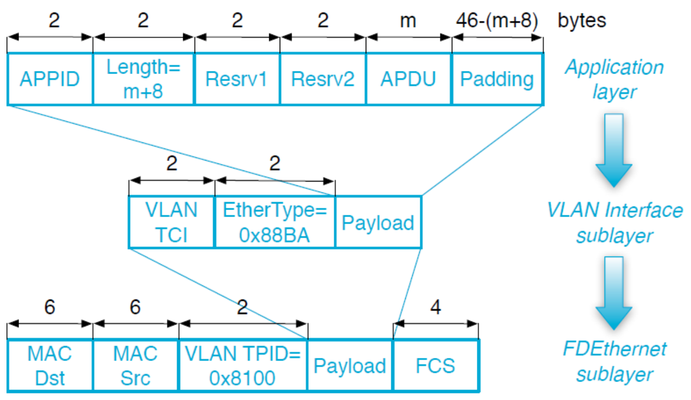


Figure 3.15: Structure of an IEC 61850-9-2 SV message (Adapted from Konka *et al.*, 2011)

The Ethernet frame header comprises of the information listed below:

- ***MAC destination address***: IEC 61850-9-2 performs Media Access Controller (MAC) filtering to meet the time demands of SV messages. In multicast mode the destination address should range from 01-0C-CD-04-00-00 to 01-0C-CD-04-01-FF. The first three bytes are assigned by IEEE and 04 is set aside for

Sample Value message transmission (International Electrotechnical Commission, 2004; UCA International Users Group, 2004).

- ***MAC source address***: this is the source device's MAC address.
- ***VLAN Tag***: SV frames are tagged according to IEEE 802.1Q to separate time critical messages from low priority data. The Tag Protocol Identifier (TPID) should be set to 0x8100 for identifying IEEE 802.1Q tagged Ethernet frames. Sampled values messages are assigned a default priority of 4 and a VLAN ID (VID) of 0. The structure of the VLAN tag is shown in Table 3.3.
- ***EtherType information***: this refers to an IEEE registered Ethernet frame reserved for SV messages. The EtherType for IEC 61850-9-2 SV packets is 0x88BA. Application IDs (APPIDs) are used to distinguish between Ethernet frames containing SV, GOOSE, GSSE messages and other frames. For SV frames, the APPID must range from 0x4000 to 0x7FFF of which a default value of 0x4000 is defined if the APPID is not configured.
- ***SV APDU***: This is the sampled value APDU which contains the application data packed using Basic Encoding Rules (BER) of the Abstract Syntax Notation One (ASN.1). The BER syntax consists of TLV (Tag, Length and Value) triplets where all fields are series of octets. It is possible that the value field in a TLV triplet encapsulate another TLV triplet.

The Tag-Length-Value triplets of the basic encoding rules are illustrated in Figure 3.16.

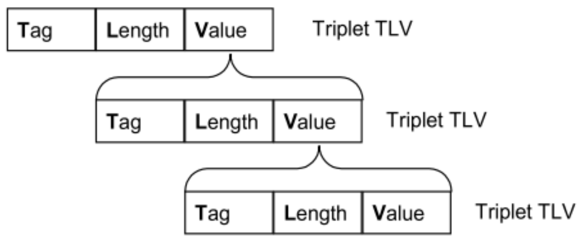


Figure 3.16: BER TLV triplets
(Adapted from International Electrotechnical Commission, 2004)

Figure 3.17 shows an example of a sampled value APDU encoded using the basic encoding rules of the ASN.1 standard.

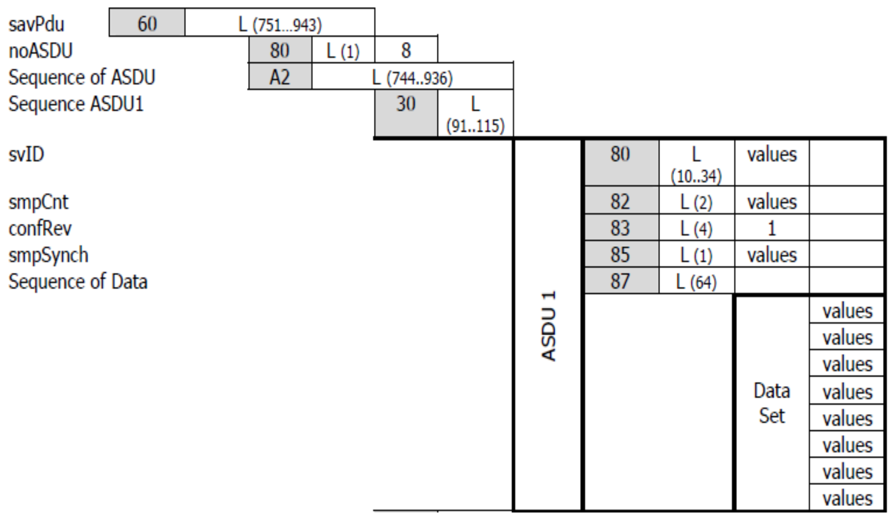


Figure 3.17: ASN.1 encoded SV APDU

(Adapted from International Electrotechnical Commission, 2004)

The sampled values APDU in Figure 3.17 shows a list of Application Service Data Unit (ASDU) attributes for identification and to provide specific information to the subscriber. These attributes/fields are the svID, smpCnt, confRev, smpSynch and the sequence of data. Given below is a list and a summary of each attribute/field in the ASDU:

i. svID

This attribute is of data type VISIBLE STRING made up of up to 35 ASCII characters and is a unique identification of the sampled value buffer.

ii. ***smpCnt***

This is a counter which increments every time a new sample is acquired from the instrument transformer. This count value starts from 0 and rolls over at a maximum number according to application. The maximum sample count (*smpCnt*) is equal to the number of SV messages published per cycle. The maximum sample count (*smpCnt*) is related to the power system frequency, sample rate (*smpRate)* and the number of ASDUs concatenated into a single APDU according to Equation 3.2.

$$\mathbf{smpCnt} = \frac{\mathbf{Power\ system\ frequency} \times \mathbf{smpRate}}{\mathbf{noASDU}}$$

Equation 3.2

iii. ***confRev***

This integer number is the number of times the SV control block configurations were edited. These editions include addition of datasets, re-ordering of members etc., with reference to the IEC 61850-9-2LE, the confRev number must always be 1 (UCA International Users Group, 2004).

iv. ***smpSynch***

This BOOLEAN flag indicates whether the sampled values are synchronised to any clock source by setting it either TRUE of FALSE.

v. ***Sequence of data and dataset***

This field contains the dataset to be transmitted in the SV packet, this dataset can be configured using SCL according to the IEC 61850-9-2 standard but for simplicity reasons the dataset is preconfigured in the IEC 61850-9-2LE to consist of four voltage and four current measurements with quality flags. The IEC 61850-9-2LE data model is discussed in detail in section 3.4.2.1.

The IEC 61850-9-2LE defines two optional attributes the refresh time and the sample-rate flags which when set to TRUE means that the SV buffer contains the RefrTm (refresh time) and the smpRate (sample rate) values.

The next section discussed the IEC 61850-9-2LE data model.

3.4.2.1 IEC 61850-9-2LE data model

The IEC 61850-9-2LE provides a pre-defined data model and dataset for the transmission of current and voltage samples. This data model contains four TVTR and four TCTR logical nodes representing voltage and current transformers respectively, these logical nodes are shown in Appendix C.3 and Appendix C.4. The TVTR and the TCTR logical node classes contain the Vol and Amp data objects for the voltage and current samples respectively. These data objects are instances of the sampled value (SAV) common data class defined in IEC 61850-7-3 standard.

The IEC 61850-9-2LE uses an instance of the SAV common data class supporting only the attributes with the MX (measurement) functional constraint to reduce implementation complexity. The resulting common data class used for the transmission of the current and voltage samples is shown in Table 3.6.

Table 3.6: SAV common data class defined in IEC 61850-9-2LE

(Adapted from UCA International Users Group, 2004)

Attribute Name	Attribute type	Comment
instMag.i	INT32	
q	Quality	Sampled value quality
sVC.scaleFactor	FLOAT32	0.001 for current; 0.01 for voltage
sVC.offset	FLOAT32	Always 0

Each sampled value published in an IEC 61850-9-2LE sampled value message contains the instantaneous magnitude (*instMag.i*) value and the quality flags (q) discussed below:

- ***instMag.i***: this is the mandatory instantaneous magnitude value of the analogue voltage/current (International Electrotechnical Commission, 2003-2005). This instantaneous magnitude value is represented using the integer data type.
- ***q***: this is the quality attribute of the sampled value, this quality attribute contains flags which are set by the source to inform the receiving device/function about the validity and other quality related issues of the sample. The structure of the quality attribute is shown in Table 3.7.

Table 3.7: Quality attribute of the SAV CDC

(Adapted from International Electrotechnical Commission, 2003-2005)

Quality Type Definition				
Attribute Name		Attribute Type	Value/ Value Range	M/O/C
		PACKED LIST		
validity		CODED ENUM	good | invalid reserved | questionable	M
detailQual		PACKED LIST		M
	overflow	BOOLEAN		M
	outOfRange	BOOLEAN		M
	badReference	BOOLEAN		M
	oscillatory	BOOLEAN		M
	failure	BOOLEAN		M
	oldData	BOOLEAN		M
	inconsistent	BOOLEAN		M
	inaccurate	BOOLEAN		M
source		CODED ENUM	process | substituted DEFAULT PROCESS	M
test		BOOLEAN	DEFAULT FALSE	M
operatorBlocked		BOOLEAN	DEFAULT FALSE	M

The identifiers in the sampled value/data quality attribute are discussed below:

i. ***Validity***: The data can be marked as *good*, *invalid*, *reserved* or *questionable*; data is marked *good* if there are no abnormalities detected, *questionable* if a supervision function detects an abnormal behaviour and *invalid* if an abnormal event has occurred.

ii. ***detailQual***: data attributes marked *questionable* and *invalid* are described using the identifiers under the detailQual flags as illustrated in Table 3.8.

Table 3.8: detailQual identifiers of the Quality attribute (Adapted from International Electrotechnical Commission, 2003-2005)

DetailQual	**Invalid**	**Questionable**
Overflow	X	
Out of Range	X	X
Bad Reference	X	X
Oscillatory	X	X
Failure	X	
Old Data		X
Inconsistent		X
Inaccurate		X

iii. ***Source* –** this flag can be set to *process* or *substituted*, process data is measured while substituted data is calculated.

iv. ***Test***: this identifier is used to notify the client/subscriber that the received information is test data and must not be used for operational purposes.

v. ***operatorBlocked***: This identifier is used to signify that the data will not be updated because it was stopped by an operator.

The resulting data model for IEC 61850-9-2LE sampled value messages is shown in Figure 3.18.

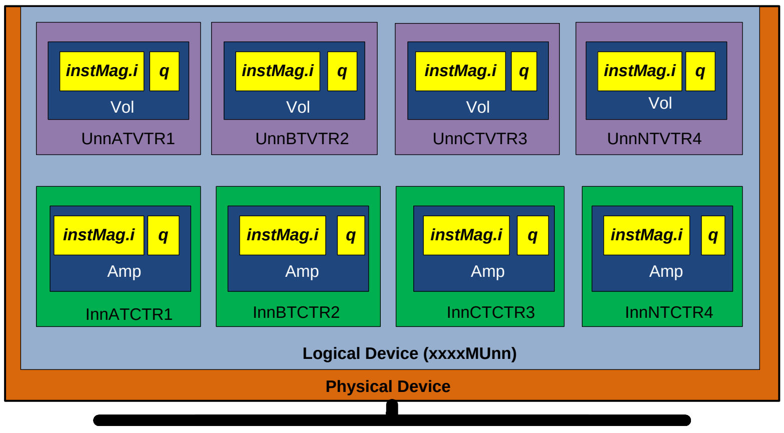


Figure 3.18: **IEC 61850-9-2LE data model**

This data model shows a total of eight logical node instances four of which are instances of the TCTR LN class and the rest are instances of the TVTR class. APPENDIX A shows the dataset defined in the IEC 61850-9-2LE for the transmission of SV messages.

The next section concludes this chapter by highlighting the advantages of using the IEC 61850 standard for communication networks over legacy protocols and hardwired communication systems.

3.5 Conclusion

After discussing the modelling techniques and the application of the IEC 61850 standard in substation automation system it can be concluded that the suite presents a number of advantages over legacy and hardwired systems. These advantages make the IEC 61850 standard an ideal choice when considering communications networks within systems. The advantages are:

i. VLAN tagging: the use of VLAN tagging is an ingenious mechanism for ensuring an intelligent and efficient use of Ethernet switches. Priority tagging allows Ethernet switches to send time-critical messages to the intended destination with minimal delays.

ii. Virtualization: this allows the standard to use the data models (logical nodes, common data classes, device behaviour and abstract service) to represent real power system equipment.

iii. Standard naming convention: the IEC 61850 standard offers standardised naming of data by using descriptive strings. This standardised naming system makes it easy for systems engineers to identify components of the power system without the need to identify index locations and registers as it was with legacy communication protocols in SCADA systems.

iv. Self-descriptive devices: with the IEC 61850 standard-based application, connected devices are able to obtain a description of data supported by the other device therefore eliminating manual configuration and reducing system engineering time. This allows for easy expansion of the SAS and reduces system rigidity.

v. Substation Configuration Language (SCL): this language enables substation devices to be configured using a language based on XML. The use of SCL eliminates purchase of the wrong equipment because the engineer can be able to identify required device functionality using the ICD document.

vi. Lower installation and integration costs: the IEC 61850 standard enables seamless integration of devices within an SAS through the GOOSE and SV process bus interface. This implementation reduces wiring cost by utilising the Ethernet bandwidth for signals instead of traditional wiring, ducting, trenching and installing conduits for different wires from the multiple transducers.

vii. Reduced engineering time: through the SCL, manual configuration of substation automation systems is drastically reduced thereby reducing configuration errors and improving system efficiency.

viii. Interchangeability: in an IEC 61850 based automation system, different devices from different vendors can be interchanged with none to minimal configuration changes because of the virtualization process of real power system devices and a standard naming convention.

ix. Interoperability: the IEC 61850 standard allows seamless integration of devices from multiple vendors through standard naming conventions, configuration language and specific communication service mappings of data.

x. Distributed functionality: The IEC 61850 standard allows the design and implementation of a decentralised SAS architecture by allowing functions in different physical devices to perform functions and communicate to achieve a single function.

The first edition of the IEC 61850 standard is limited to communication within a substation and covers information modelling, communication service mapping for the transmission of data, testing and evaluation procedures for IEDs. This chapter serves as the information base for the development of the IEC 61850-8-1 GOOSE and IEC 61850-9-2 SV messages-mapping VHDL modules.

The development of the proposed VHDL modules for mapping GOOSE and SV messages will be based on the IEC 61850-8-1 standard and the IEC 61850-9-2LE guideline respectively. The IEC 61850-8-1 standard and the UCAIug IEC 61850-9-2 implementation guideline defined the communication service mapping of data/information to the ISO 8802-3 Ethernet frame for GOOSE and SV messaging respectively.

The mapping and publishing of GOOSE and SV messages as defined in the IEC 61850-8-1 standard and IEC 61850-9-2LE specific communication service mappings (SCSM) will refer to the four-part IEC 61850-7 standard (IEC 61850-7-1 to IEC 61850-7-4 standards). The four-part IEC 61850-7 standard defines the data modelling and communication services to be employed for data transmission. The mapping of the GOOSE and SV messages will use control blocks, instances of common data classes and logical nodes defined in the IEC 61850-7-2, IEC 61850-7-2 and IEC 61850-7-4 standards.

The next chapter, Chapter Four discusses the design and implementation of VHDL modules for mapping and publishing GOOSE and SV messages as specified in the IEC 61850-8-1 standard and the IEC 61850-9-2 standard in conjunction with the UCAIug IEC 61850-9-2 implementation guideline respectively. Chapter Four also provides finite state machine models, VHDL code snippets, equations and block diagrams used to implement functional VHDL modules for publishing SV and GOOSE messages. Hardware design using VHDL and integration methods are also presented in Chapter Four for combining the developed GOOSE and SV messages mapping VHDL modules with the Analogue-Front End (AFE) to produce a GOOSE monitoring node and limited-function Merging Unit prototype respectively.

CHAPTER FOUR

DESIGN AND IMPLEMENTATION OF GOOSE AND SV MESSAGE MAPPING HARDWARE USING THE XILINX SPARTAN 6 FPGA

4.1 Introduction

This chapter presents the design of VHDL modules and hardware integration method for mapping IEC 61850-8-1 Generic Object Oriented Substation Event (GOOSE) messages and IEC 61850-9-2LE Sampled Value (SV) messages in V*HSIC*-Hardware Description Language (VHDL) on a Field Programmable Gate Array (FPGA).

This chapter is subdivided into sections 4.2 to 4.5. Section 4.2 consists of two sub-sections, section 4.2.1 and 4.2.2 which provide the design scope of the GOOSE and SV messages-mapping VHDL modules respectively. Section 4.3 provides a comparative analysis of FPGAs, microcontrollers and PCs for the selection of an ideal platform for mapping and publishing GOOSE and SV messages. Furthermore, it provides a brief overview of the Analogue Front-End (AFE) module, the integration of this AFE with the GOOSE and SV message-mapping VHDL modules implemented on the FPGA to produce a GOOSE monitoring node and Merging Unit prototypes respectively.

Section 4.4 details the development of VHDL modules for mapping and publishing GOOSE and SV messages on an FPGA platform through flow charts, state diagrams and VHDL code snippets. Section 4.5 concludes this chapter and highlights the challenges encountered during the development of VHDL code for mapping GOOSE and SV messages on an FPGA.

4.2 Research project scope in the context of the IEC 61850 standard

This research project focuses on the development of two separate VHDL modules for mapping and publishing SV and GOOSE messages as defined by the IEC 61850 standard on an FPGA platform. In the context of the first edition of the IEC 61850 standard, process level IEDs act as interfaces between the primary plant equipment (instrument transformers and circuit breakers) and bay level IEDs. These process IEDs monitor, control and measure variables in the power system and communicates this information to bay level IEDs through the process bus.

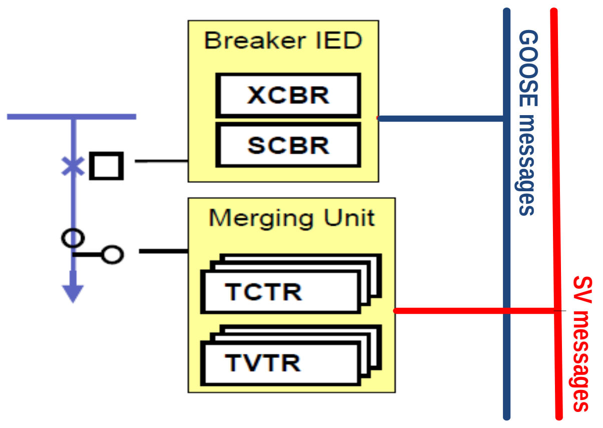


Figure 4.1: Process level Intelligent Electronic Devices (IED)

In the illustration shown in Figure 4.1, the breaker IED uses GOOSE messages to transmit circuit breaker information modeled using the XCBR logical node to bay level IEDs. In the same illustration in Figure 4.1, the Merging Unit (MU) uses SV messages to transmit voltage and current samples modeled using the TVTR and the TCTR logical nodes respectively to bay level IEDs. These two devices (breaker IED and MU) are known as process level IEDs and the messages published by these devices (GOOSE and SV messages) make up the IEC 61850 process bus.

In order to design an IED, the designer must have a clear understanding of the device's intended operations/functions so that a fitting hardware platform is used. Once the hardware platform has been selected, the next step is to develop methods/algorithms to achieve these functions. After that, software must be developed to implement this algorithm following a set language supported by the hardware platform. The last step of any design process is to evaluate the final prototype in order to ascertain whether it meets all the design criteria.

The design evaluation documented in Chapter Five of this thesis will determine whether the developed VHDL modules for publishing GOOSE and SV messages meets the requirements of the IEC 61850-8-1 standard and the IEC 61850-9-2LE respectively. Commercial software applications like Wireshark, TransView, IEDScout and SVScout will be used to validate the structure and accuracy of the published GOOSE and SV messages against the requirements of the referenced standards. If the developed VHDL modules fail to meet the requirements of the referenced standards, the VHDL code will be edited and re-evaluated in a repetitive manner until the requirements are satisfied.

The following sections, section 4.2.1 and 4.2.2 discusses the design and implementation scope for GOOSE and SV messages mapping VHDL modules.

4.2.1 The design scope for GOOSE message-mapping

Real-time data communication between different functions and/or devices is vital in any automation system; the IEC 61850 standard defines a real-time, object oriented communication protocol known as GOOSE messages. With reference to Figure 2.1, GOOSE messages are used for peer-to-peer communication between bay-level IED (interface IF8) or between process level actuators/devices and bay IEDs (interface IF5). Process equipment use this communication interface (IF5) to relay primary plant information to bay-level IEDs, for example, GOOSE messages on the process bus are used for transmitting instantaneous voltages to voltage regulating relays for on-load tap changing.

This research project will demonstrate the concept of communicating via GOOSE messages by developing a VHDL module which will enable an FPGA to publish analogue and binary values using the GOOSE protocol defined in the IEC 61850-8-1 standard. The VHDL module to be developed in this research project will publish GOOSE messages containing the status of a circuit breaker (XCBR$ST$Pos$stVal) and the measured root mean square voltage (MMXN$MX$Vol$mag). The circuit breaker status will be derived from the position of a switch on the Nexys 3 development board while the magnitude of the voltage will be injected into the developed GOOSE monitoring node prototype through the AFE using the CMC 256*plus* test set.

This research project demonstrates how GOOSE messaging can be used as an alternative to copper wiring for communicating digital status and analogue values in a substation automation system. GOOSE messages can be multicast to multiple devices connected to the process bus thereby reducing parallel copper wires and implementation costs compared to hardwired and legacy communication protocol-based systems.

Figure 4.2 shows hardware components and logical nodes that will be used in the implementation of the IEC 61850-8-1 GOOSE message-mapping VHDL module on an FPGA.

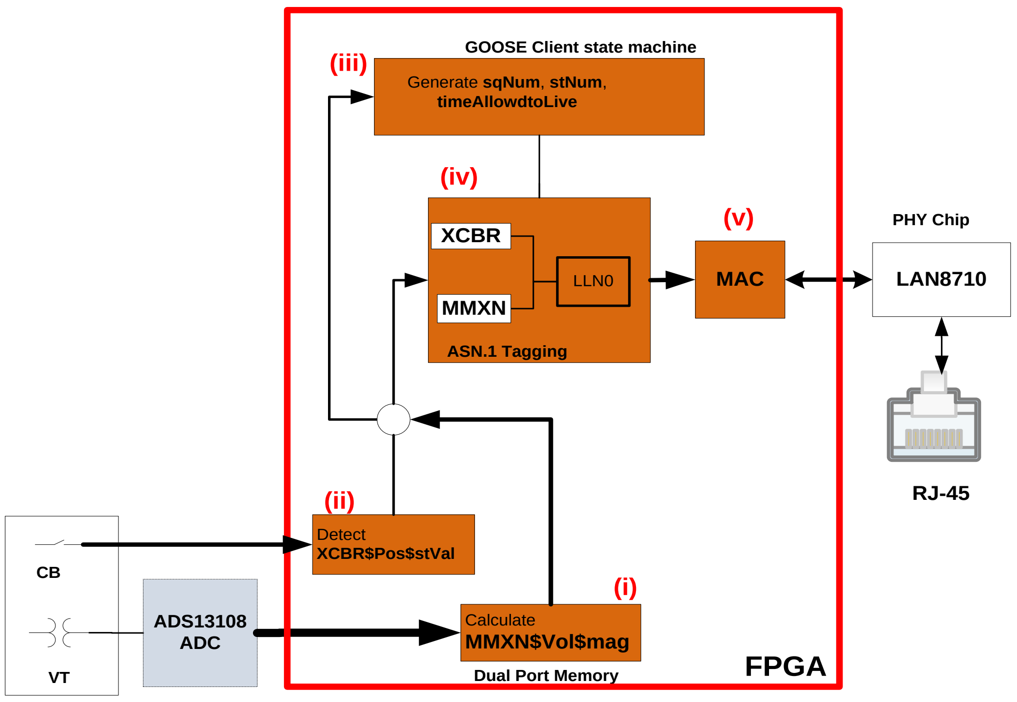


Figure 4.2: **Mapping GOOSE messages on an FPGA platform**

In Figure 4.2, the red block represents the FPGA device and the numbered blocks inside represents hardware modules implemented in VHDL for:

i. calculating the RMS value of the injected voltage (MMXN$MX$Vol$mag.i),
ii. detecting the status change of the emulated circuit breaker (XCBR$ST$Pos$stVal object),
iii. implementing the GOOSE client state machine as defined in the IEC 61850-8-1 standard,
iv. generating an ASN.1 tagged GOOSE message and
v. sending the GOOSE message to the PHY for transmission through the MAC.

The next section discusses the design scope for the IEC 61850-9-2LE sampled value message-mapping VHDL module.

4.2.2 The design scope for Sampled Value (SV) message-mapping

Sampled value messages contain instantaneous current and voltage samples from current and voltage transformers (CTs and VTs). Merging Units periodically publish sampled value messages onto an Ethernet network (process bus) to subscriber IEDs. In this research project, the developed VHDL module must be able to map and publish sampled value messages following the rules and requirements defined in the UCAIug implementation guideline commonly known as the IEC 61850-9-2LE (Light Edition).

The Analogue Front-End (AFE) module developed at the Center for Substation Automation and Energy Management (CSAEMS) at the Cape Peninsula University of Technology (CPUT) samples voltage and currents signals from CTs and VTs. This AFE consists of an eight-channel Analogue to Digital Converter (ADC) and an FPGA module connected through a Serial Peripheral Interface (SPI) bus. This FPGA module synchronises the ADC samples to a 1PPS pulse and also reads the output codes for the eight channels. The AFE's internal structure and communication interface is discussed in detail in section 4.3.2 on page 93.

The IEC 61850-9-2LE defines two control blocks and a dataset for the transmission of sampled values as discussed in section 3.4.2 in Chapter Three. In this research project, the MSVCB01 control block and the PhsMeas1 dataset are supported which means that the developed VHDL module will enable the FPGA to publish SV messages at 80 samples per cycle. The PhsMeas1 dataset contains four TCTR and four TVTR logical node instances for current and voltage transformer measurements respectively.

Figure 4.3 shows a block diagram of hardware components, functional modules and logical nodes required for mapping and publishing SV messages from an FPGA platform.

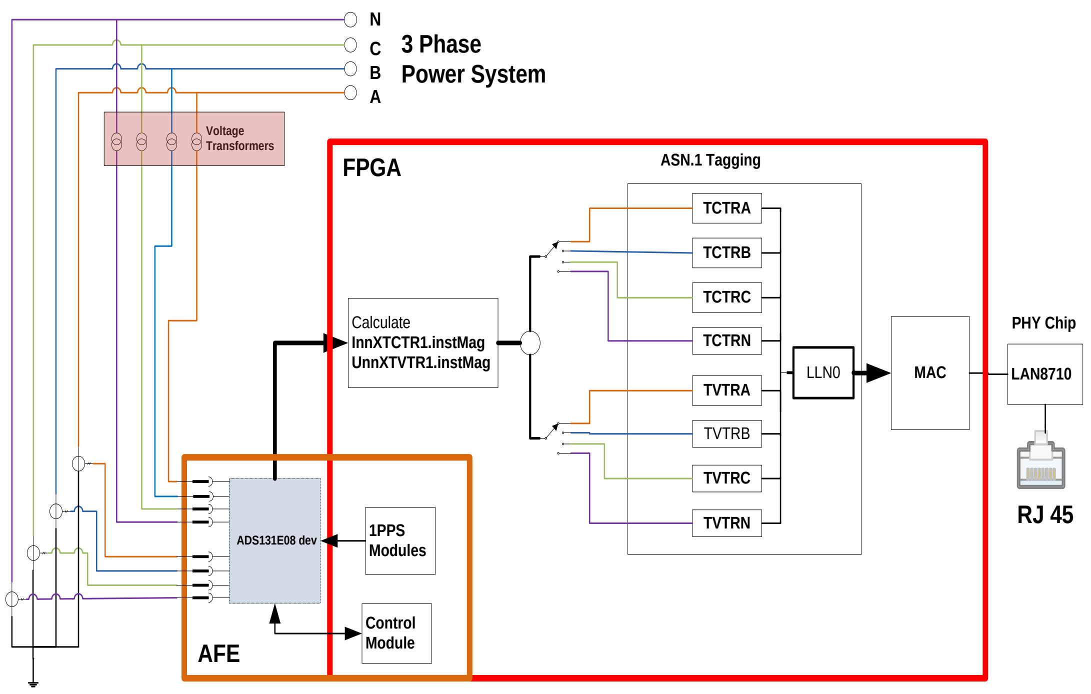


Figure 4.3. Mapping Sampled Value (SV) messages on an FPGA

Figure 4.3 shows the AFE as an **orange** block consisting of an ADC development board (ADS131E08 EVM) and the VHDL modules implemented in the FPGA for synchronising and reading the ADC output codes. In Figure 4.3, besides the AFE, the FPGA is represented by the red block while the calculation and SV message-mapping functional modules are shown as blocks inside the FPGA.

The next section compares hardware platforms found in literature that were used in the design of GOOSE enabled devices and Merging Units.

4.3 Hardware platform

The limited-function process level IED prototypes to be developed in this research project (for publishing GOOSE and SV messages) have to meet the functionality described in sections 4.2.1 and 4.2.2 respectively. The functionality described in the referenced sections is useful in the VHDL hardware module design phase and partly for hardware platform selection. To select the proper hardware to support the required functions a number of design questions are presented below:

- How much processing power is required by the device?
- Which platform is suitable for this design between a Windows PC, microcontroller and an FPGA?
- Is there need to use an IEC 61850 communication stack from a vendor?

- Which communication interface is required by the process level IED?
- How will the IED connect to the primary plant equipment?

Process level IEDs capable of publishing GOOSE and/or SV messages must be able to perform high-speed calculations, processing and communication to meet the requirements of the IEC 61850 standard in terms of the message performance class and delivery times. A Windows/Linux PC platform using Scapy or NS3 open source network simulator or LabVIEW is capable of generating SV and/or GOOSE messages adhering to the timing constraints defined in the IEC 61850-5 standard.

The down side with Windows/Linux PCs is that they are expensive compared to microcontrollers and FPGA platforms and they require an AFE module to interface with primary plant equipment. Another disadvantage of implementing limited-function process-level IED on Windows/Linux PC platforms is that high-end graphic and storage resources will be under-utilised.

Another option is to use a microcontroller or an FPGA platform for mapping and publishing GOOSE and SV messages. These two platforms are on average cheaper and smaller than industrialised PCs. Microcontrollers which possess the processing, Ethernet communications and timing capabilities for mapping GOOSE and SV messages are on average 5 to 10 times cheaper than an FPGA module with same capabilities.

With microcontroller-based platforms, IEC 61850 standard-communication libraries are available to quicken the development process unlike with FPGA-based platforms. For example, Beck-IPC controllers use an IEC 61850 library developed by SystemCorp, this stack supports GOOSE messaging and MMS (Retonda-Modiya, 2012). The *openIEC61850* and the *libIEC61850* stacks are example of libraries based on Java and C available for microprocessor-based (PCs) and microcontroller-based platforms.

FPGAs are inherently faster than microcontrollers even though the clock frequency of any general purpose microcontroller is normally up to twenty times more than that of FPGAs. FPGAs are faster because of the memory data streaming technique, reduced instructions and also the overlap of control (Guo *et al.,* 2004). The FPGA's memory data streaming technique allows for faster memory read cycles thereby improving device efficiency.

The iteration parallelism of an FPGA is more than twice that of a microcontroller, this factor together with control overlapping and data streaming from memory accounts for an efficiency factor, which can be more than 6 to 47 times more than that of a microcontroller (Guo *et al.,* 2004). Moreover, the iteration parallelism of FPGAs allows the device to execute different processes in parallel other than sequentially.

VHDL is a full-bodied language for FPGA-based systems development which allows electrical signal attributes to be described so as to mimic exact timing characteristics which any other language cannot do. This feature makes VHDL the ideal language for developing application for real-time status or value change detection. Table 4.1 shows the differences between FPGA, microcontroller and PC based platforms.

Table 4.1: Comparing FPGA, microcontroller and PC platforms

Factor	FPGA	Microcontroller	Windows/Linux PC
System clock speed	Fast	Fast	Very Fast
Multi-threading	No	No	Yes
Parallel computing	Yes	No	No
Reduced instruction set	Yes	Yes	Yes
IEC 61850 library support	No	Yes	Yes
Price per unit	Moderate	Cheap	Expensive

With reference to the differences listed in the previous paragraphs the FPGA is ideal because it is inherently faster than the microcontroller/processor platforms and it is moderately expensive. For this research project, the Xilinx Spartan-6 XL16 FPGA is used. The following sections discuss the features of this FPGA and the Nexys 3 development board.

4.3.1 Nexys 3 development board

The Nexys 3 development board is a ready to use design platform for digital circuits based on the Xilinx Spartan-6 XL16 FPGA. In addition to the Xilinx FPGA, the Nexys 3 development board offers a collection of peripherals including non-volatile RAM, 10/100 Ethernet PHY controller, 16 MB of cellular RAM and a USB to UART port (Digilent Inc, 2013).

The Xilinx Spartan-6 XL16 FPGA features 2278 slices each with 6 Look-Up Tables (LUTs), 576 Kbits of Random Access Memory (RAM), a Digital Clock Management (DCM) block and Phase Locked Loop (PLL). The DCM and the PLL allow the FPGA clock to run at 500 MHz or more, which accounts to a cycle time of 2 ns.

The FPGA has a multitude of digital I/O pins that can be used as simple general Input /Outputs (I/O) ports, or to function as specialised peripherals. For this design, some digital I/O pins of the Xilinx FPGA will be used as an SPI port and a Media Independent Interface (MII) port for communicating with the AFE and the PHY chip respectively.

Figure 4.4 shows a block diagram and interconnections between the Xilinx FPGA and the components/peripherals on the Nexys 3 development board.

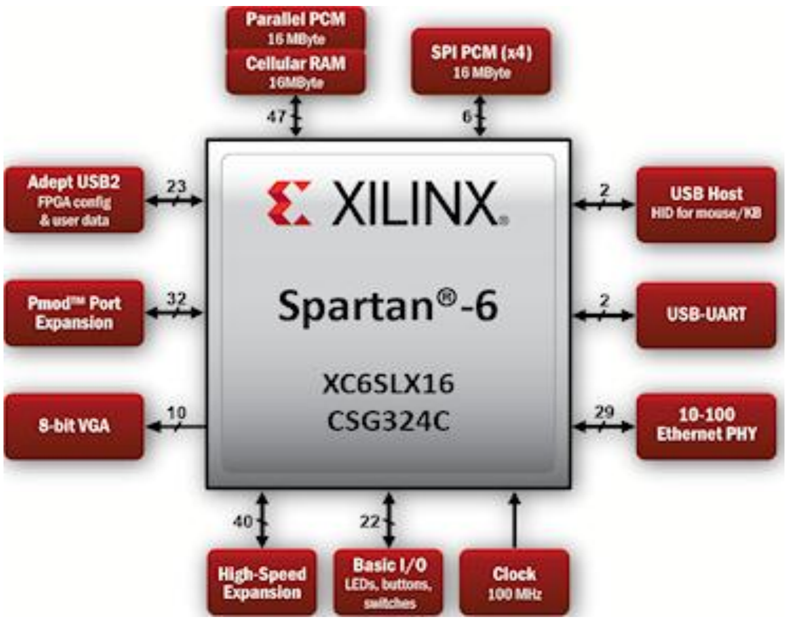


Figure 4.4: **Nexys 3 development board block diagram**
(Adapted from Digilent Inc, 2013)

The Nexys 3 development board includes the LAN 8710 chip from Microchip which is a 10/100 Base TX PHY controller (Digilent Inc, 2013). This PHY chip is responsible for encoding and decoding messages transmitted and received from the physical medium (copper wires). The LAN 8710 PHY controller connects to the FPGA through the Media Independent Interface (MII). The MII bus consists of four receive and four transmit connections between the Media Access Controller (MAC) and the PHY clocked at the 25 MHz to achieve 100 Mbps Ethernet speed. The Xilinx FPGA does not have a dedicated MAC so therefore a custom MAC will be implemented in VHDL.

This Nexys 3 development board has eight slide-switches, five pushbuttons, eight LEDs and four Peripheral Module (PMOD) connectors connected to the FPGA I/O pins. The PMOD connectors are designed by Digilent Incorporated which allows data acquisition, input/output or communication modules to be connected to the Nexys development boards (Digilent Inc, 2013). Two PMOD connectors will be used to communicate with the Analogue Front-End module (AFE) and the slide-switch (SW2) will be used to simulate the status of a circuit breaker.

The next section discusses the AFE module developed by the CSAEMS for sampling the analogue signals from CTs and VTs and how it is integrated to the Nexys 3 development board.

4.3.2 Analogue Front-End (AFE) device

The AFE consists of a VHDL module implemented in the Xilinx FPGA (main processing unit), eight step-down transformers and the ADS131E08 EVM Rev A ADC development kit as shown in Figure 4.6. The ADS131E08 EVM Rev A development kit is manufactured by Texas Instrument for developing industrialised power system monitoring and measurement applications based on the ADS131E08 Analogue to Digital Converter (ADC) (Texas Instruments, 2012-2013).

Figure 4.5 shows how current and voltage signals from CTs and VTs are connected to the eight-channel ADC through the step-down transformers. In this figure (Figure 4.5), the **red** transformers step-down the CT and VT analogue output signals to voltages within the specified ADC input range.

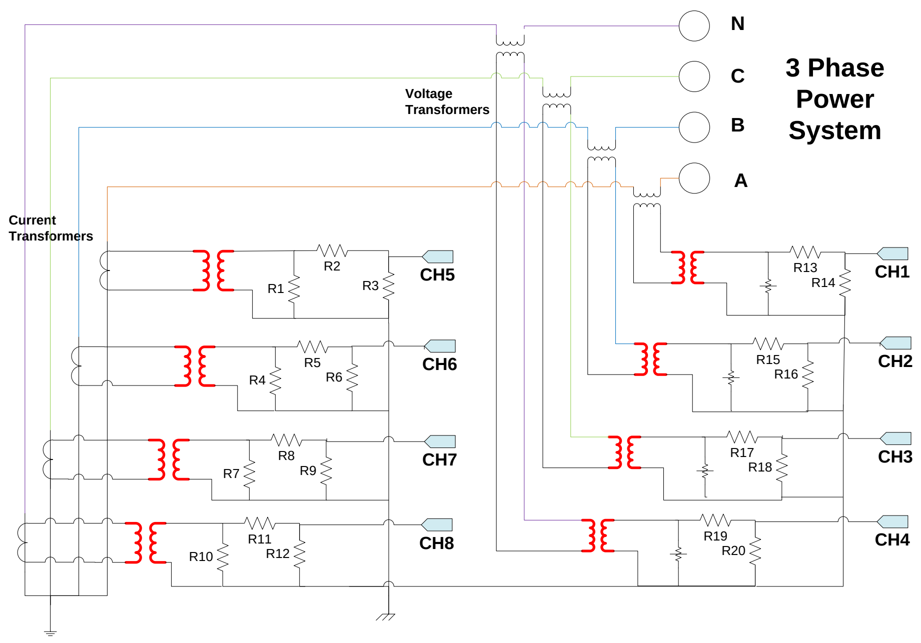


Figure 4.5: CT and VT analogue signal interface

The AFE step-down transformers turns ratio was not known so therefore it was determined empirically using the CMC 256*plus* test set. Voltage and current signals were injected into to the primary side of the step-down transformers and the output measured on the ADC input marked CH1 to CH8 in Figure 4.5 above. Table 4.2 shows the relationship between step-down transformer primary and the ADC input voltage.

Table 4.2: Relationship between step-down transformer primary and secondary voltage measurements

Voltage Injection (V_{RMS})	**Step-down transformer output (mV)**	**Comment**
1	28.567113	
33	1334	Nominal Input
84.0126897	2400	V_{MAX}

The AFE's maximum voltage input is limited by the ADC's maximum voltage input equal to 2.4 V, so therefore according to Table 4.2 the maximum voltage that can be injected without damaging the ADC is approximately 84.01 V_{RMS}.

Table 4.3 shows the relationship between the step-down transformer's primary current injection and its secondary voltage measurements. The AFE's maximum current injection is limited by the ADC's maximum input voltage equal to 2.4 V, so therefore according to Table 4.3, the maximum current that can be injected without damaging the ADC is approximately 16.48 A_{RMS}.

Table 4.3: Relationship between injected current signals and step-down transformer voltage outputs

Current Injection (A_{RMS})	**Step-down transformer output (mV)**	**Comment**
0	0	
1	145.664	Nominal input
16.47627417	2400	I_{MAX}

VTs and CTs step-down power-system voltage and current signals to 110 V and 1 A/5 A respectively under nominal conditions (Khuraam, 2012). In this research project the CMC 256*plus* test set is used as a voltage and current source instead of VTs and CTs in a controlled environment in the laboratory.

The relationship between CT/VT primary and secondary current and voltage signals is based on Equation 4.1.

$$\frac{N_P}{N_S} = \frac{V_P}{V_S} = \frac{I_S}{I_P}$$

Equation 4.1

In Equation 4.1, the subscripts P and S refer to the primary and secondary sides of the instrument transformer and the N symbol represents the number of turns in the transformer windings. In order to calculate the primary voltage and current measurements the value of the CT/VT turns ratio must be known. In this research project, the GOOSE/SV message-mapping VHDL modules will be evaluated in the laboratory using the CMC 256*plus* test set so therefore for calculation purposes the VT and CT primary to secondary turns ratio is chosen to be 100:1.

Table 4.4 and Table 4.5 show the relationship between primary power system voltage/current measurements and the ADS131E08 ADC input voltages. The values were calculated using the CT/VT ratio of 100:1 and the scaling factors of the step-down transformers in the AFE presented in Table 4.2 and Table 4.3.

Table 4.4: Relationship between CMC 256*plus* generated voltage and ADC input

Primary Voltage (V_{RMS})	CMC 256*plus* output (V_{RMS})	ADC Input (mV)	Comment
0	0	0	
100	1	28.567113	
3300	33	1334	Nominal Voltage
8401.26897	84.0126897	2400	V_{MAX}

Table 4.5: Relationship between CMC 256*plus* generated current and ADC input

Primary Current (A_{RMS})	CMC 256*plus* output (A_{RMS})	ADC input (mV)	Comment
0	0	0	
100	1	145.664	Nominal Current
1647.63	16.4763	2400	I_{MAX}

In this research project, the nominal line-to-neutral voltage and current measurements are assumed to be 3.3 kV_{RMS} and 100 A_{RMS} respectively equal to secondary voltage and current measurements of 33 V_{RMS} and 1 A_{RMS} respectively. These two values, V_{MAX} and

I_{MAX} will be used in the calculation of the instantaneous voltage and currents in section 4.4 respectively.

The output voltages of the AFE step-down transformers are then connected to the eight channels of the ADS131E08 ADC. The ADS131E08 is a 24 bit delta-sigma (ΔΣ), eight channel simultaneous sampling ADC. This ADC has an adjustable gain and sampling rate configurable through the serial peripheral interface. The ADS131E08 ADC is connected to the Xilinx FPGA through the PMOD connector serial peripheral interface for configuration and ADC output code retrieval. The data is written to and read from the ADC by the AFE VHDL module at 12.5 Mbps (Megabits per second).

Figure 4.6 shows the AFE block diagram with main emphasis on the connections between the ADS13108 evaluation kit and the AFE VHDL module in the Xilinx FPGA.

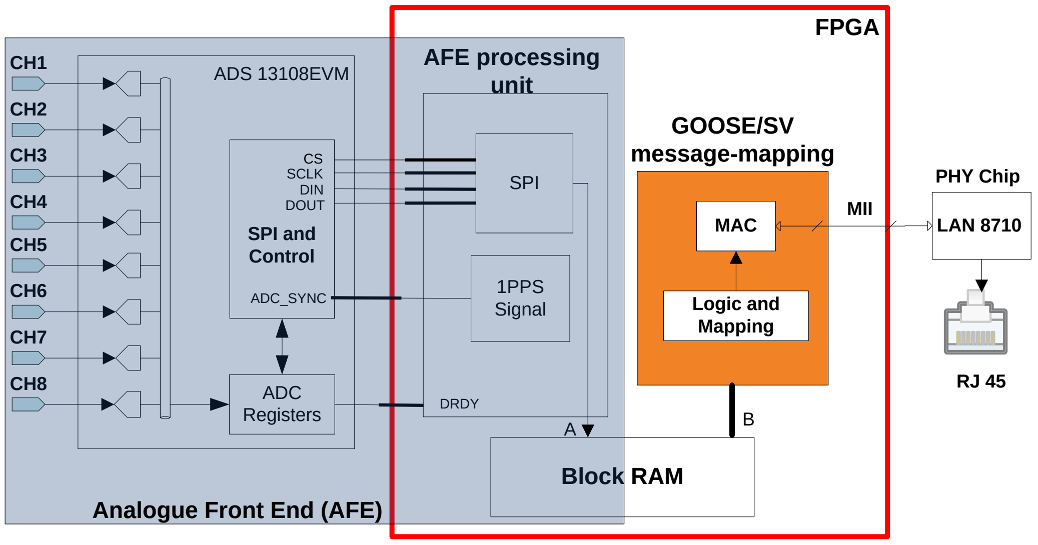


Figure 4.6: **Communication between the AFE and the SV/GOOSE message-mapping VHDL modules**

With reference to Figure 4.6, the Xilinx FPGA is represented by the red block and the blue-shaded area shows the components which make up the AFE excluding the step down transformers shown in Figure 4.5.

The AFE's processing unit is a VHDL top-module implemented in the Xilinx FPGA for configuring, synchronising and reading conversion data from ADS13108 ADC. The FPGA synchronises the ADC using an internally generated 1PPS signal. This internally generated 1PPS signal will suffice for evaluating the performance and accuracy of the

SV message-mapping VHDL module but this means that the developed limited-function MU prototype will not be synchronised to the CMC 256*plus* test set.

On startup, the AFE processing unit configures the ADS131E08 ADC to sample the eight analogue channels at 4 kSPS (4000 samples per second). The ADS131E08 ADC is also configured to use an internally generated reference voltage (V_{REF}) of 2.4 V to produce conversion data (output codes) for the eight channels. The ADC output codes are in binary twos' complement format, the ADC outputs 0x7FFFFF and 0x800000 for positive and negative full-scale deflections ($\pm V_{REF}$).

Table 4.6 shows the relationship between the ADC's output codes and the input voltage.

Table 4.6: Relationship between the ADC output code and the input voltage (Adapted from Texas Instruments, 2012-2013)

Input Signal, V_{IN}	**Ideal Output Code**
$\geq V_{REF}$	0x7FFFFF
$+V_{REF}/(2^{23}-1)$	0x000001
0	0x000000
$-V_{REF}/(2^{23}-1)$	0xFFFFFF
$\leq -V_{REF}(2^{23}/2^{23}-1)$	0x800000

The ratio of the input voltage to the reference voltage (V_{REF}) can be expressed using Equation 4.2.

$$\frac{\mathbf{V_{in}}}{\mathbf{V_{REF}}} = \frac{\textbf{ADC OUTPUT CODE}}{\mathbf{0x7FFFFF}}$$

Equation 4.2

Substituting the value of V_{REF} and re-arranging Equation 4.2 results in Equation 4.3 for calculating the input voltage from the ADC output code.

$$\mathbf{v(t)_{in}} = \frac{\textbf{ADC OUTPUT CODE}}{\mathbf{0x7FFFFF}} \times \mathbf{2.4V}$$

Equation 4.3

The ADC is fixed into continuous-read mode, in this mode, the eight channels are sampled simultaneously every 250 µs (4000 samples per second) and it uses an active-low DRDY output pin to indicate the availability of output codes (conversion data). Once the $\overline{\mathrm{DRDY}}$ output pin has been set LOW, the processing unit reads the ADC codes and writes them to the dual-port RAM to be accessed by the GOOSE/SV message-mapping VHDL modules.

The read process is illustrated in Figure 4.7 showing the transitions of the $\overline{\mathrm{DRDY}}$, SCLK, $\overline{CS}$ and the DOUT pins.

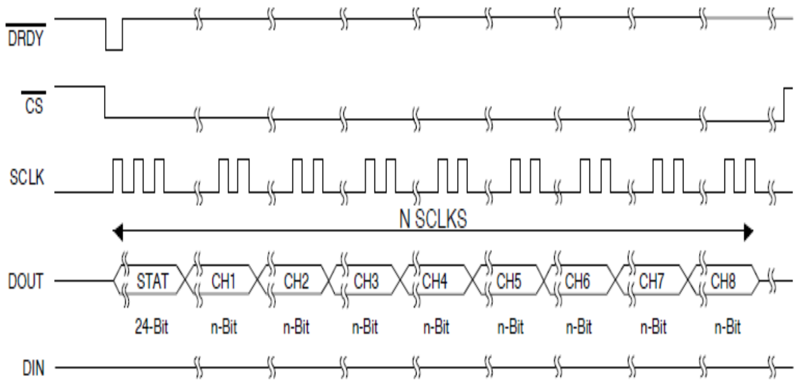


Figure 4.7: **Reading ADC conversion data using SPI when $\overline{DRDY}$ pin goes LOW (Adapted from Texas Instruments, 2012-2013)**

A total number of 216 bits consisting of a 24 bit status-word plus 24 bits per channel for the eight channels (192 bits) are read from the ADC once $\overline{\mathrm{DRDY}}$ pin is activated. Data is read starting with the Most Significant Bit (MSB) of ADC status word then channel one to channel eight output codes in sequential order. The AFE processing module appends a sample count (*smpCnt*) value to the output codes read from the ADC before the data is stored in the dual-port RAM. The dual-port RAM is an ideal storage facility implemented in VHDL for storing information that has to be accessed by two separate applications. This RAM allows the AFE to store the ADC output codes using port A and the GOOSE/SV message-mapping VHDL modules to read these ADC codes using port B as shown in Figure 4.6. These two ports are independent of each other meaning that data can be written or read from whichever port irregardless of any operation on the other port.

The next section (section 4.4) discusses the development of VHDL code for mapping IEC 61850-9-2LE-based SV and IEC 61850-8-1 standard-based GOOSE messages on the Xilinx Spartan 6 FPGA.

4.4 VHDL Hardware Development

V*HSIC*-HDL is a hardware description language developed in the mid-1980s by the USA's Department of Defense and the Institute of Electrical and Electronic Engineers (IEEE) for the development of very-high-speed integrated circuits (Van der Spiegel, 2001). VHDL is a full-featured language suitable for describing the behavior of electronic components or even systems ranging from simple logical gates to custom chips e.g. Media Access Controllers (MAC) for Ethernet communications (Altium, 2008). Different electrical signal attributes can be descried using VHDL, for example, pulse rise/fall times and signal delays.

VHDL is a parallel programming paradigm whereby the system behavior is expressed in parallel format unlike in sequential programming languages e.g. C, C++ in which functions execute one-after-the-other. VHDL process statements are executed in a parallel manner as soon as the value of the input in the sensitivity-list changes; this is similar to logic gates whereby the output changes as soon as one or more inputs change. Due to this fact, VHDL-based digital systems are inherently faster than microcontroller-based systems. The Xilinx ISE design suite was used to generate VHDL designs and programming files for Xilinx Spartan 6 FPGA.

4.4.1 VHDL module design process

The design and development of the GOOSE message and the SV message-mapping VHDL modules is based on the agile software development life cycle which breaks-up the development process into small build increments provided in iteration. The Agile software development model combines iterative and incremental processes focusing on meeting design requirements and rapid delivery of working software products. The iterative method of the agile development model ensures that working software is delivered after each build before the new build increment commences. The build tasks are assigned completion time frames and the developed software is tested before moving on to the next task.

The design and development of VHDL code for mapping GOOSE and SV messages is a repetitive process executed until valid messages conforming to the IEC 61850-8-1 standard and the IEC 61850-9-2LE are published by the FPGA respectively. It is worth noting that even though the structure and transmission procedures for GOOSE and SV messages are different, the VHDL hardware designs will share common digital circuits e.g. MAC and dual-port RAM. Sections 4.4.1.1 and section 4.4.1.2 discuss the developed VHDL modules for mapping and publishing GOOSE and SV messages using a Xilinx Spartan 6 FPGA respectively.

The following section details the development of a VHDL hardware module which will enable the Xilinx Spartan 6 FPGA to publish GOOSE messages as defined in the IEC 61850-8-1 standard.

4.4.1.1 GOOSE message mapping VHDL module

The proposed GOOSE message-mapping VHDL module is based on a finite state machine, this finite state machine allows a system with distinct states to be modeled using a state diagram to clearly show state transitions and trigger events. The finite state model for the IEC 61850-8-1 standard-based GOOSE message-mapping VHDL module is shown in Figure 4.8.

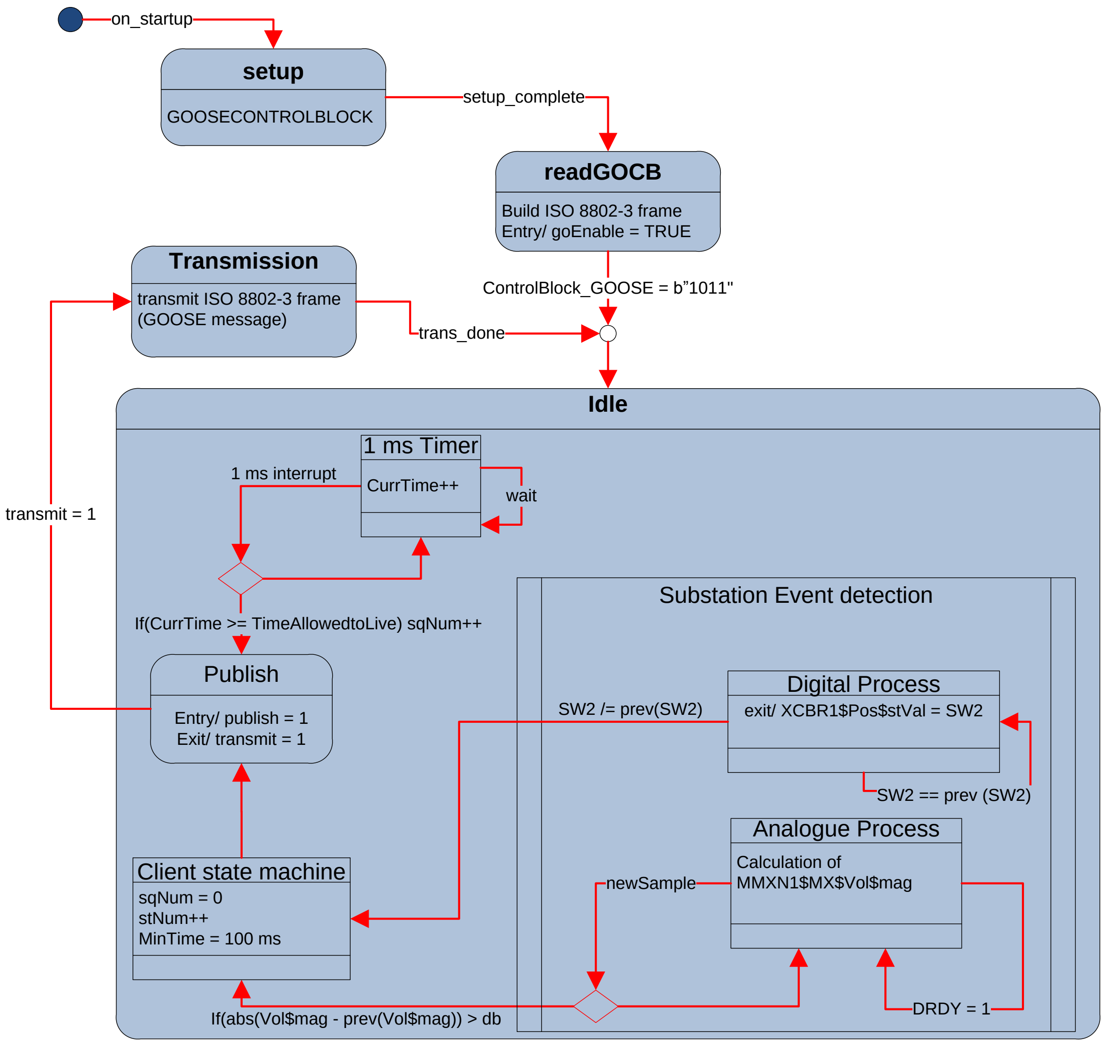


Figure 4.8: **GOOSE message-mapping VHDL module finite state model**

The finite state model shown in Figure 4.8 consists of four main states connected by trigger events and these are the *setup*, *readGOCB*, *Idle* and *Transmission* states. The VHDL entities for mapping GOOSE messages are implemented as sub-modules in the AFE's processor unit's VHDL model. This hierarchical approach in VHDL reduces code complexity and produces an easily maintainable design. The GOOSE message-mapping VHDL module also follows a hierarchical approach by implementing a top module with multiple sub-modules for different functions.

On power cycle, finite state machine of the GOOSE message-mapping VHDL module commences at the *setup* state where it creates a byte array to hold the GOOSE ISO 8802-3 Ethernet frame. In the *setup* state, the VHDL model assigns values to the Ethernet frame header fields which remain static until device reconfiguration. The fields which remain static for the remainder of the device operation include the source and destination MAC addresses, VLAN tags, EtherType and the APPID. These fields are assigned to the ISO 8802-3 frame (byte array) using the default values defined in the IEC 61850-8-1 standard. Table 4.7 shows the value assigned to the frame as defined in the standard.

Table 4.7: ISO 8802-3 frame values according to the IEC 61850-8-1 standard

Parameter	Value	Comments
Destination MAC Address.	01:0C:CD:01:00:00	Multi-cast address for all destination subscribers.
VLAN Tag.	0x8100	VLAN tags are used to separate high priority information data based on IEEE 802.1Q.
VID	0	VLAN ID not used
Priority	4	Default priority of GOOSE messages
EtherType.	0x88B8	IEEE registered EtherType for GOOSE messages.
APPID.	0x0000	Default application ID for GOOSE messages.

A VHDL record is used for generating the GOOSE message PDU because this PDU is a composite data entity consisting of multiple fields as shown in Figure 3.13 on page 71. This GOOSE PDU consists of Tag-Length-Value triplets and using this VHDL record the tag and length fields are assigned as shown Table 4.8.

In Table 4.8, the values of the *gocbRef*, *datSet* and *allData* attributes of the GOOSE PDU are extracted from the Configured IED Description (CID) file and the data model discussed in sections 4.4.1.1.1 and 4.4.1.1.2.

Table 4.8: **Values assigned to GOOSE PDU fields**

PDU Attribute	Tag	Value
gocbRef	0x80	FPGA/IED1/LLN0$GO$GSE_CB_GOOSE
timeAllowedtoLive	0x81	MinTime – MaxTime
datSet	0x82	FPGA/IED1/LLN0$GOOSE_Eval.
goID	0x83	GOOSEID
t	0x84	UTC time
stNum	0x85	0-2^{32}
sqNum	0x86	0-2^{32}
test	0x87	TRUE/FALSE
confRev	0x88	1
ndsComm	0x89	FALSE
numDataSetEntries	0x8A	2
allData	0xAB	MMXN1$MX$Vol$mag.i XCBR1$ST$Pos$stVal

Figure 4.9 shows how an ASN.1 encoded GOOSE message PDU is generated and stored in a VHDL record before it is copied into the ISO 8802-3 Ethernet frame (byte array) for transmission.

```
--this is the GOOSE Control Block
goosePdu.gocbRef(0) <= x"80";                -- Goose ControlBlock - TAG 0 -
conv_int := std_logic_vector(to_signed(GOOSE_REF'right, 32));
goosePdu.gocbRef(1) <= conv_int(24 to 31);           -- Lenght of gocbRef
for i in 1 to GOOSE_REF'right loop
   goosePdu.gocbRef(1 + i) <= GOOSE_REF(i);
end loop;
--Tag 1-
-- this is the timeAlllowedtoLive for a message sent
goosePdu.timeAllowedtoLive(1) <= x"81";
goosePdu.timeAllowedtoLive(2) <= x"04";                  -- Number of bytes
--Tag 2 -
-- Goose Data set
goosePdu.datSet(0) <= x"82";
conv_int := std_logic_vector(to_signed(len_datSet, 32));  -- length of GOOSE
goosePdu.datSet(1) <= conv_int(24 to 31);                -- length of dataset
for i in 1 to GOOSE_DATASET'right loop
   goosePdu.datSet(1 + i) <= GOOSE_DATASET(i);                        --
end loop;
```

Figure 4.9: **ASN.1 BER encoded GOOSE APDU in VHDL**

After assigning values to the ISO 8802-3 frame header fields and creating the VHDL record for the GOOSE PDU, the GOOSE message-mapping finite state machine transitions to the *readGOCB* as shown in Figure 4.8. Once in the *readGOCB* state the

VHDL record containing the GOOSE PDU is then copied to the Ethernet frame (byte array). Figure 4.10 shows a snippet of the VHDL code for copying data from the VHDL record into the Ethernet frame (byte array).

```
-- datSet ---------------------------------------------------
when b"0010" =>
    goPDUattr <= b"0010";
       if byteCnt < len_DATSET + 2 then
          gooseframe(index) <= goosePdu.datSet(byteCnt);
          byteCnt <= byteCnt + 1;  -- count number of bytes
          count <= count + 1;
       else
          byteCnt <= 0;
          goPDUattr <= b"0011";   -- handle next attribute
       end if;
-- goID -----------------------------------------------------
---
when b"0011" =>
    goPDUattr <= b"0011";           -- stay
    if byteCnt < len_GOID + 2 then
       gooseframe(index) <= goosePdu.goID(byteCnt);
       byteCnt <= byteCnt + 1;     -- count number of bytes
       count <= count + 1;
     else
        byteCnt <= 0;
        goPDUattr <= b"0100";
     end if;
```

Figure 4.10: **Copying the GOOSE PDU from the VHDL record into the Ethernet frame before transmission**

According to the IEC 61850-8-1 standard, GOOSE messages are published once the GOOSE control block is enabled by setting the *goEnable* flag TRUE. Before exiting the *readGOCB* state, the mapping VHDL module enables the GOOSE control block by setting this aforementioned flag. Once GOOSE message transmission is enabled, the finite state machine transitions to the *Idle* state. In the *Idle* state the VHDL module is on standby waiting for status/value change (substation event) or retransmission time to occur according to the GOOSE client state machine defined in the IEC 61850-8-1 standard.

The *Idle* state implements the GOOSE client logic for transmitting GOOSE messages as defined in the IEC 61850-8-1 standard. In this state, GOOSE messages are transmitted every time the status or value of any of the objects in the dataset changes (substation event) or when the message retransmission time occurs. When this event occurs, new object values are copied into the pre-configured Ethernet frame (byte array) together with the *timeAllowedtoLive*, *stNum*, *sqNum* and the *timestamp* (*T*). The values of the *stNum* and *sqNum* attributes are calculated using the GOOSE client state machine shown in Figure 3.12 in page 70.

The developed *Idle* state logic is shown in the flowchart in Figure 4.11.

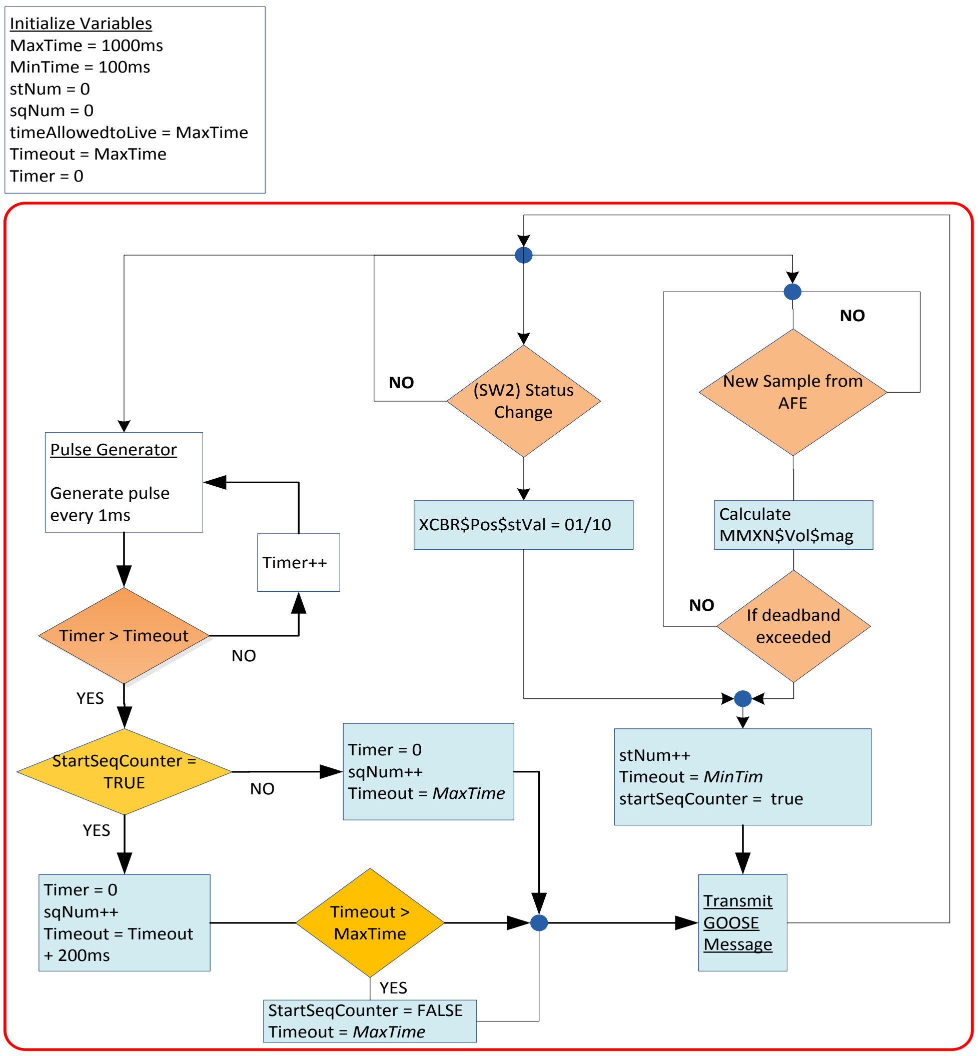


Figure 4.11: **GOOSE client state machine implemented in the *Idle* state**

The flowchart in Figure 4.11 shows how the GOOSE client state machine defined in the IEC 61850-8-1 standard is implemented in the GOOSE message-mapping VHDL module. This flowchart highlights how the GOOSE message state and sequence numbers (*stNum* and *sqNum*) are calculated every time a substation event or message retransmission time occurs.

Figure 4.12 shows a VHDL process in the developed *publishSigGen.vhd* module which implements the GOOSE client state machine as described in the flow chart in Figure 4.11.

```vhdl
-- Process : PUBLISH
-- Publishes GOOSE messages after after MaxTime has expired or after MinTime has
expired before or after an event respectively
PUBLISH : process (clk, goEnable, msPulse, TriggerEvent)
variable timeCount : integer := 0;
variable currPubRate : integer := 0;
begin
-- reset stNum and sqNum
    if clk'event and clk = '1' then -- if there is a rising edge on the clock
        publishSig <= '0';
        if TriggerEvent = '1' then                                     -- SET --
            publishSig <= '1';        -- publish new value immediately
            afterEvent <= '1';         -- set Flag after value of data set changed
            currPubRate := MinTime;            -- start at the minimum time
            timeCount := 0;
            stNum <= stNum + 1;
            sqNum <= 0;
        else
            if goEnable = '1' then  -- is goEnable ?
               if msPulse = '1' then                           -- after 1ms --
                   timeCount := timeCount + 1;
                   if afterEvent = '1' then
                        if timeCount >= currPubRate then
                            timeCount := 0;
                            sqNum <= sqNum + 1;     -- sqNum
                            currPubRate := currPubRate + 200;
                            publishSig <= '1';            -- publish
                            if currPubRate >= MaxTime then
                                afterEvent <= '0';
                            end if;
                        end if;
                    else
                        if timeCount >= MaxTime then
                            sqNum <= sqNum + 1;
                            currPubRate := MaxTime;    -- make timeAllowedtoLive
                            timeCount := 0;
                            publishSig <= '1';
                         end if;
                     end if;
                end if;
            end if;
         end if;
      timeAllowedtoLive <= std_logic_vector(to_unsigned(currPubRate, 32));
-- convert the time allowed to live and send it with the GOOSE Message
    end if;
end process PUBLISH;
```

Figure 4.12: **GOOSE client state machine logic implemented in VHDL**

The normal retransmission time (*MaxTime*) and the fast retransmission time (*MinTime*) of GOOSE messages are specified as manufacturer local issues in the IEC 61850-8-1 standard. As such, the MaxTime and MinTime values are set to 1000 ms and 100 ms in this VHDL module respectively. Also, the fast retransmission time increment (*MinTime* increment) is not defined in the IEC 61850-8-1 standard so for this research project the

MinTime is incremented by 200 ms until the normal retransmission time (*MaxTime*) is reached or exceeded as shown in Figure 4.12.

The GOOSE messages published by the developed VHDL module contain an analogue value measured through the AFE and the emulated circuit breaker status. This dataset contains an instance of the MMXN and XCBR logical nodes for analogue measurements and circuit breaker control respectively. These two logical nodes will suffice for demonstrating the capabilities of GOOSE messaging in replacing hardwiring in a substation automation system.

The following sections (section 4.4.1.1.1 and section 4.4.1.1.2) discus the data modeling for the analogue and digital information published in the GOOSE message dataset.

4.4.1.1.1 Analogue GOOSE modelling using the MMXN LN

The MMXN logical node (LN) is used for the calculation of the voltage, frequency, current and other attributes of single-phase systems. The structure of the MMXN logical node is shown in Appendix C.1, this LN consists of a number of instances of different common data classes but of particular interest are the measured values (MV) objects specifically the *Vol* data object. Data objects in the MMXN logical node are optional and are selected depending on the device's application in an automation system. For the purposes of this research project, the *Vol* object will be used to report the Root Mean Square (RMS) value calculated from the input voltage measured by the voltage transformer (VT).

The *Vol* object is an instance of the Measured Value (MV) common data class, this class is shown in Appendix D.2. In this class, there are three mandatory data attributes and these are the *mag* (measured value), *q* (quality) and *t* (timestamp) which should be available in all instances of this object. In this research project, only the *mag* attribute will be mapped to the dataset for transmission.

The mag data attribute in the Vol object referenced as the MMXN$Vol$mag is the calculated RMS voltage of the injected voltage; this mag attribute is an instance of the AnalogueValue data type defined in the IEC 61850 standard. With the mag attribute, the measured or calculated value can be represented as an integer (i) or floating point value (f). According to the IEC 61850 standard, integer and floating point values are represented by data types INTEGER and FLOAT32 respectively (International Electrotechnical Commission, 2003-2005). For the purposes of this research project the calculated RMS voltage will be transmitted as an integer value referenced as the MMXN1$MX$Vol$mag.i.

The resulting data model for the analogue data measurements is shown in Figure 4.13.

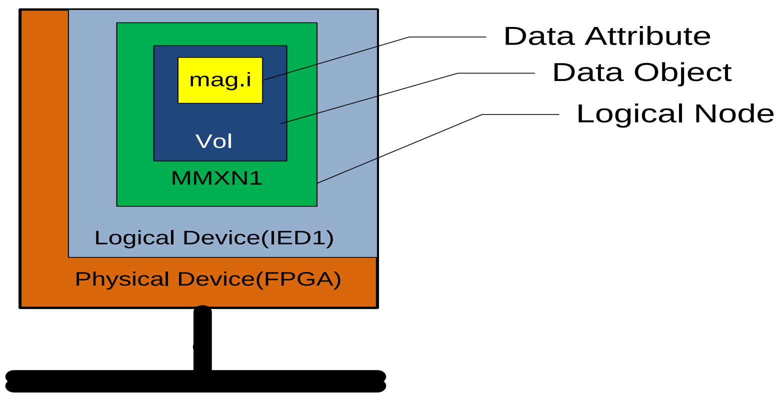


Figure 4.13: **Data Model for analogue value transmission (MMXN1$MX$Vol$mag)**

Using the standard naming convention of the IEC 61850 standard the RMS voltage published by the FPGA-based GOOSE monitoring node is referenced as FPGA/IED1/MMXN1$MX$Vol$mag. The RMS value of an alternating current (AC) signal is defined as the square root of the mean (average) of the squared function of the instantaneous values. The root mean square value is calculated using Equation 4.4.

$$\mathbf{RMS} = \sqrt{\frac{1}{n}\sum_{n=0}^{n} v(t)^2}$$

Equation 4.4

In Equation 4.4, n represents the number of mid-ordinates used in the RMS calculation; the higher the number of ordinates used to more accurate the resultant RMS value (Basic Electronics Tutorials, 2016). Equation 4.5 below is normally used in theoretical calculations of the root mean square value of an AC signal; this formula is correct for purely sinusoidal waveforms but incorrect for distorted sinusoidal signals. The RMS voltage for the MMXN1$MX$Vol$mag object will be calculated using Equation 4.4.

$$v_{RMS} = \frac{v_p}{\sqrt{2}}$$

Equation 4.5

The instantaneous voltages are calculated using the ADC output code of the single phase voltages injected into channel 1 of the AFE's analogue digital converter. The instantaneous primary voltage is calculated using the ADC maximum deflection (0x7FFFFF) and the maximum primary input voltage (V_{MAX}) using Equation 4.6. The value of V_{MAX} used in Equation 4.6 is equal to 8.401 kV_{RMS} and provided in Table 4.4.

$$\mathbf{v(t)} = \frac{\mathbf{ADC\_OUTPUT\_CODE}}{\mathbf{0x7FFFFF}} \times \mathbf{V_{MAX}}$$

Equation 4.6

Substituting Equation 4.6 into Equation 4.4, the resulting formula for calculating the RMS value of the injected single phase voltage is shown in Equation 4.7.

$$\mathbf{V}_{\boldsymbol{RMS}} = \sqrt{\frac{\mathbf{1}}{\mathbf{n}} \sum_{\mathbf{n=0}}^{\mathbf{n}} \left(\frac{\mathbf{ADC\_OUTPUT\_CODE}}{\mathbf{0x7FFFFF}} \times \mathbf{V_{MAX}} \right)^{\mathbf{2}}}$$

Equation 4.7

The RMS value of the injected voltage (MMXN1$MX$Vol$mag attribute) is calculated over one complete cycle, for this calculation the sample count (*smpCnt*) value appended to the ADC output code is used. The RMS calculation VHDL module calculates the instantaneous voltage using Equation 4.6 as soon as new ADC output code is available on the dual-port RAM. The calculated instantaneous voltages are used as mid-ordinates in the summation shown in Equation 4.7.

The *SQ_VOLT_CALC* and *RET_RMS* VHDL processes shown in Figure 4.14 are used for calculating the RMS value of the injected voltage. The *SQ_VOLT_CALC* process in Figure 4.14 executes every 250 µs when new ADC conversion data is available, in this process (*SQ_VOLT_CALC*) the instantaneous magnitude is calculated and summated. The RMS value is evaluated after 3999 mid-ordinates and returned in integer format.

The VHDL module for calculating the RMS of the injected voltage signal (*Calculation.vhd*) is shown in Figure 4.14.

```
-- calculate the mean square voltage using ADC code over a full cycle
SQ_VOLT_CALC : process(clk, newSample, VinA, SampleCount)
    variable ordinates, Vin_sq : unsigned( 31 downto 0) := x"00000000";
    variable Sum_VinSq : unsigned( 47 downto 0) := x"000000000000";
    begin
       if clk = '1' and clk'event then
          newRMS <= '0';
          if newSample = '1' then
             ordinates := ordinates + 1;
             Vin_sq := Calc_VOLTS(VinA);
             Sum_VinSq := Sum_VinSq + Vin_sq;
             if ordinates = x"00000F9F" then
                 newRMS <= '1';
                 VRMS <= Sum_VinSq/ordinates;
                 Sum_VinSq := x"000000000000";
                 ordinates := x"00000000";
             end if;
           end if;
        end if;
    end process;
    ----------------------------------------------------------------
    -- Calculate the square root of the mean square value
    ----------------------------------------------------------------
RET_RMS : process( clk, newRMS, VRMS)
    variable tempVal : unsigned(15 downto 0);
    begin
        if clk = '1' and clk'event then
            ValueChange <= '0';
            if newRMS = '1' then
                tempVal := unsigned_sqrt(VRMS(31 downto 0));
                Root <= std_logic_vector(tempVal);
                ValueChange <= '1';
            end if;
       end if;
   end process;
```

Figure 4.14: **VHDL calculation routine for MMXN$MX$Vol$mag value**

For magnitude measurement, the value of MMXN1$MX$Vol$mag is updated to the calculated instantaneous RMS voltage only when the deadband parameter is exceeded as illustrated in Figure 4.15.

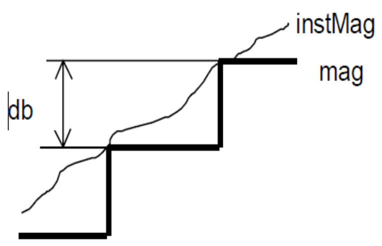


Figure 4.15: **Relationship between instantaneous magnitude (instMag), magnitude (mag) and deadband (db) value for MV objects**

(Adapted from International Electrotechnical Commission, 2003-05)

Figure 4.16 shows a VHDL process (*ANALOG_CHANGE*) used for determining whether the deadband parameter was exceeded between successive RMS calculations. In this snippet the deadband (db) parameter is hardcoded to 10 V_{RMS}. The deadband calculation process runs every time the instantaneous RMS value is available.

```vhdl
ANALOG_CHANGE : process(clk, ValueChange, Root)
    variable db : signed(15 downto 0);
    variable diff : signed(15 downto 0);
    variable vrms : signed(15 downto 0);
    begin
        if clk = '1' and clk'event then
           AnaChange <= '0';
           if ValueChange = '1' then
              db := x"000A";              -- set deadband to 500V
              vrms := signed(Root);
              prevVRMS <= vrms;
              diff := abs(vrms - prevVRMS);
              if diff > db then         -- dead band violated
                 AnaChange <= '1';        -- change in the Analogue signal
                 VArms(15 downto 0) <= Root(15 downto 0);
              end if;
            end if;
         end if;
     end process;
```

Figure 4.16: VHDL implementation of MMXN1$MX$Vol$mag deadband exceed detection

If the deadband parameter is exceeded the fast re-transmission procedure of the GOOSE client is executed is illustrated in the flowchart in Figure 4.11. The following section discusses the modeling and implementation of the XCBR1$ST$Pos$stVal object in the GOOSE message mapping VHDL module.

4.4.1.1.2 Binary GOOSE modelling using the XCBR LN

In this research project the XCBR logical node is used to report the status of an emulated circuit breaker, this logical node class is shown Appendix C.2. The XCBR class consists of multiple mandatory and optional data objects; from these, the *Pos, BlkOpn* and the *BlkCls* objects are required for any instance of this logical node class. This research project only maps *Pos* object for the status of the circuit breaker while the *BlkOpn* and *BlkCls* objects will be supported in future developments.

The *Pos* object represents the position of a substation switch/circuit breaker and it is an instance of Double Point Controllable (DPC) shown in Appendix D.1. This DPC class contains multiple mandatory attributes including the *stVal* (status), *q* (quality) and *t* (timestamp) but for the purposes of this research project only the *stVal* attribute will be mapped to the dataset for transmission. According to the IEC 61850-8-1 standard, the stVal attribute is a CODED ENUM data type which is represented by a two-bit bit-string. This two-bit bit-string symbolises the status of the circuit breaker as shown in Table 4.9.

Table 4.9: **Interpretation of the status of the circuit breaker using stVal values**

stVal bit-string	Interpretation of status
00	Intermediate state
01	Off / Open state
10	On / Close state
11	Invalid State

The resulting data model for reporting the status of the circuit breaker is shown in Figure 4.17.

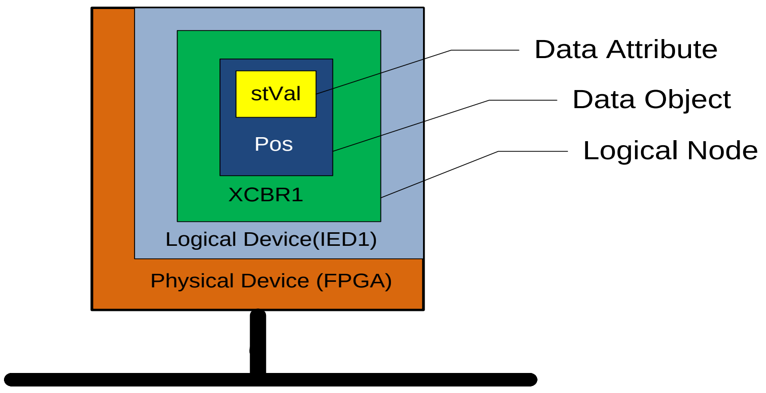


Figure 4.17: **Data model for binary value (XCBR1$ST$Pos$stVal) transmission**

The resulting GOOSE data model for the GOOSE message-mapping VHDL module is shown in Figure 4.18.

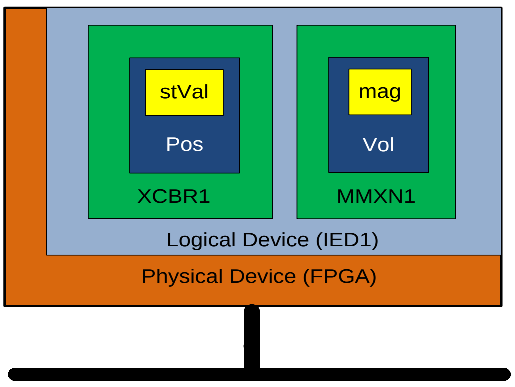
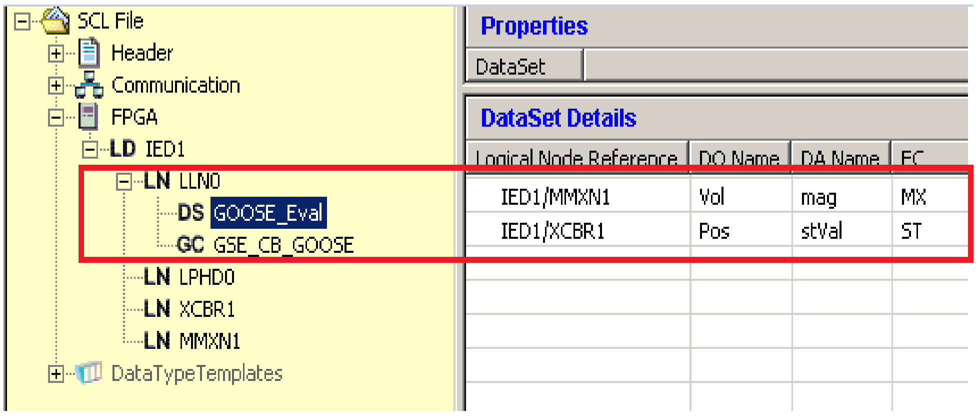


Figure 4.18: **Data model for the IEC 61850-8-1 standard GOOSE message-mapping VHDL module**

Figure 4.19 shows in the red rectangle the dataset containing the XCBR1$ST$Pos$stVal and the MMXN1$MX$Vol$mag data attributes published by the GOOSE message-mapping VHDL module as configured in the Configured IED Description (CID) given in APPENDIX G.



```
SCL File
  Header
  Communication
  FPGA
    LD IED1
      LN LLN0
        DS GOOSE_Eval
        GC GSE_CB_GOOSE
      LN LPHD0
      LN XCBR1
      LN MMXN1
  DataTypeTemplates
```

Properties

DataSet

DataSet Details

Logical Node Reference	DO Name	DA Name	FC
IED1/MMXN1	Vol	mag	MX
IED1/XCBR1	Pos	stVal	ST

Figure 4.19: **GOOSE dataset published by the GOOSE message-mapping VHDL module**

The references to the dataset and the GOOSE control block for this GOOSE message-mapping algorithm are *FPGA/IED1/LLN0$GOOSE_Eval* and *FPGA/IED1/LLN0$GO_GSE_CB_GOOSE*.

Figure 4.20 shows the VHDL code snippet implementing the EVENT_TRIG_BOOL process. The EVENT_TRIG_BOOL process detects the change in position of SW2 on the Nexys 3 development board which emulates a circuit breaker. In this figure (Figure 4.20), the *Boolin* signal is the position of SW2 after de-bouncing.

```vhdl
-- check whether the BOOLEAN state has changed
EVENT_TRIG_BOOL : process(Boolin, clk)
begin
    if RST = '1' then
        BoolChange <= '0';
        prevDigitalState <= '0';
    elsif clk = '1' and clk'event then
        prevDigitalState <= Boolin;
        BoolChange <= '1';         -- change in
        if Boolin = prevDigitalState then    -- static
            BoolChange <= '0';
        end if;
    end if;
end process EVENT_TRIG_BOOL;
```

Figure 4.20: **SW2 (XCBR1$ST$Pos$stVal) position change detection process**

Once the position of SW2 changes or the deadband of the calculated RMS value is exceeded (substation event) or the GOOSE message retransmission time occurs, the dataset elements (MMXN1$MX$Vol$mag and XCBR1$ST$Pos$stVal) are inserted into the protocol data unit for transmission.

For GOOSE message transmission, the values in the dataset are encoded into the PDU using the MMS data value encoding rules (Retonda-Modiya, 2012). The MMS data encoding rules defines tags which should be used for encoding different values of different data types as shown in Table 4.10.

Table 4.10: GOOSE data value encoding using MMS tags (Adapted from Retonda-Modiya, 2012)

MMS Data Value Encoding			
Data Type	**MMS Tag (HEX)**	**Example Value**	**Encoding (HEX)**
array	81	int[2] = {0, 1}	81 06 85 01 00 85 01 01

structure	82	Struct{
		int 0,
		Bool TRUE
		}
			82 06 85 01 00 83 01 01

boolean	83	TRUE	83 01 01
bit-string	**84**	**1010_2**	**84 02 04 A0**
integer	**85**	**255**	**85 04 00 00 00 FF**
unsigned	86	255	86 01 FF
floating point	87	1.0	87 05 08 3F 80 00 00 (IEEE)
octet string	89	01 02	89 02 01 02
visible String	8A	“ab”	8A 02 61 62
timeofday	8C	1:00:05	8C 04 .. ..
bcd	8D	09 09 09	8D 02 03 E7
boolean array	8E	1010	8E 02 04 A0

Value

Padding

The MMS data value encoding shown in Table 4.10 follows the Tag-Length-Value triplet rule. This encoding rule contains a Tag which identifies the type of data, the Length which is the number of bytes that make up the data and the Value which is the actual data. According to the MMS encoding rules, bit-string information is padded so that the value forms a complete octet(s) as shown in Table 4.10. With reference to the bit-string encoding example shown in Table 4.10, the value “A0” consists of the bit-string “1010” with the four least significant bits padded with zeros.

The integer and bit-string tags are used for mapping IEC 61850 standard values of data types INTEGER and CODED ENUM respectively. The calculated RMS voltage (MMXN1$MX$Vol$mag) of type INTEGER is mapped using the integer tag and the circuit breaker status (XCBR1$ST$Pos$stVal) of type CODED ENUM is mapped using the bit-string types.

Figure 4.21 shows a VHDL process (*ALLDATA*) which copies the calculated RMS value of the injected voltage into the ISO 8802-3 Ethernet frame. This process also sets the value of XCBR1$ST$Pos$stVal to 0x80 and 0x40 if the emulated circuit breaker (SW2) is on and off respectively. The values 0x40 and 0x80 are derived from the bit strings values “01” and “10” which represent the XCBR$ST$Pos$stVal off and on positions respectively. The encoded values 0x40 and 0x80 are equal to bit strings “01” and “10” with the six least significant bits padded with zeros.

```
-- Runs when the value of Vrms or the state of the boolean changes
ALLDATA : process (TriggerEvent, clk, publish)
begin
   if RST = '1' then
       element(0 to 11) <= (others=>x"00");
    elsif clk = '1' and clk'event then
       if publish = '1' then   -- if there is a status change in one
of the elements in the dataset then
          element(0) <= x"AB";
          element(1) <= x"0A";           -- length of list
          element(2) <= x"85";     -- Analogue Value using the MMXU LN
          element(3) <= x"04"       -- INT32 value with 4 bytes
          element(4) <= VArms(31 downto 24);
          element(5) <= VArms(23 downto 16);
          element(6) <= VArms(15 downto 8);
          element(7) <= VArms(7 downto 0);
                -- start the bit string for breaker position
          element(8) <= x"84";
          element(9) <= x"02";
          element(10) <= x"06";          -- padding
          if Boolin = '1' then
              element(11) <= x"80";       -- ON
          else
              element(11) <= x"40";       -- OFF
          end if;
       end if;
    end if;
end process ALLDATA;
```

Figure 4.21: **Adding dataset objects into the GOOSE frame for transmission**

The state machine transitions to the *Transmission* state in which the MAC shifts data out to the PHY chip for transmission onto the Ethernet network. Once the GOOSE message has been transmitted, the finite state machine of the GOOSE message-mapping VHDL module transitions back to the *Idle* state and waits for a substation event or message retransmission time to occur. The state machine runs continuously following the procedure defined until the device is powered off.

The implementation of the Ethernet MAC is given in APPENDIX E (VHDL source). The following section provides the VHDL code design and development for mapping SV messages on an FPGA platform.

4.4.1.2 Sampled Value (SV) message-mapping VHDL module design

The proposed Sampled Value (SV) message-mapping VHDL module is modeled using the finite state model. The finite state model is conceived from a machine that can only be in a specific state at a particular point in time and it transitions between states when trigger events occur. Figure 4.22 shows a finite state model for the mapping sampled value messages on the Xilinx FPGA.

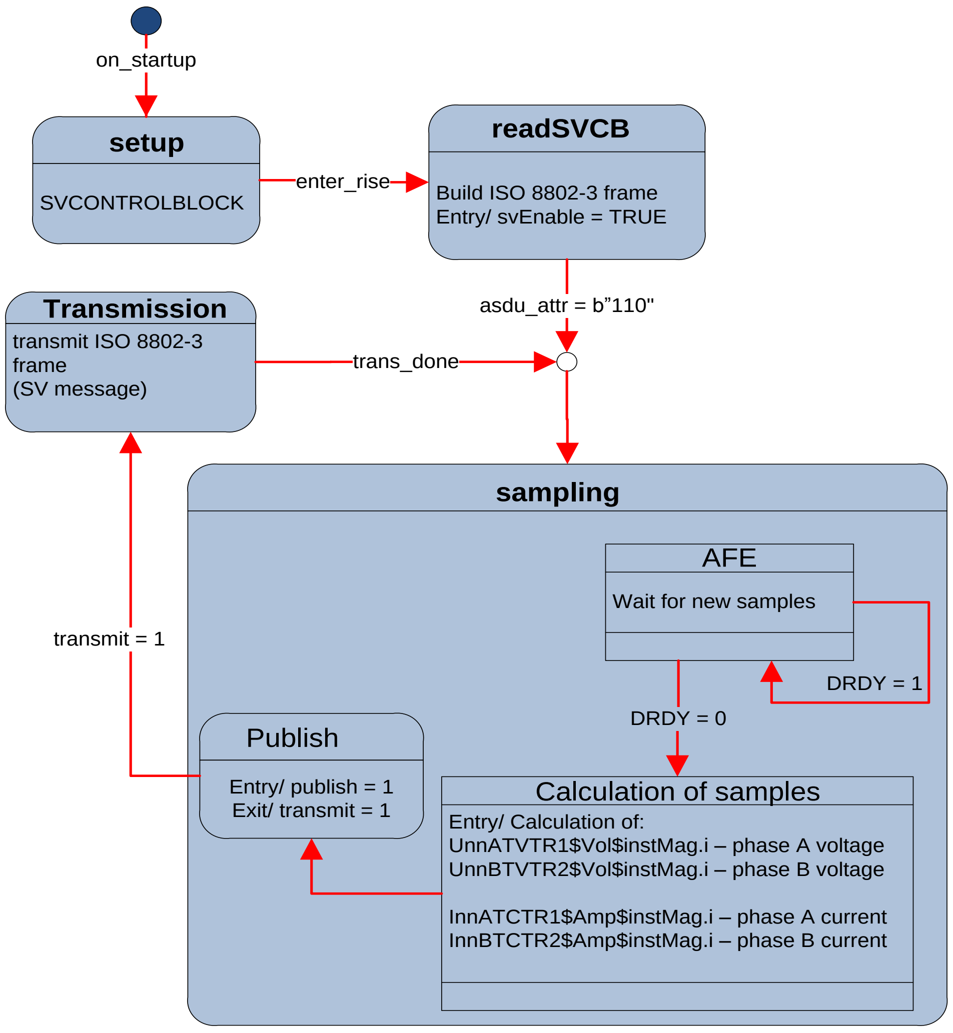


Figure 4.22: **Sampled Value message mapping VHDL module finite state model**

The finite state model shown in Figure 4.22 consists of four main states the *setup*, *readSVCB*, *sampling* and *transmission* states. This finite state model in Figure 4.22 is translated into a hardware description language (HDL) for the Xilinx Spartan 6 FPGA. The VHDL modules for mapping and publishing SV messages is implemented as a sub-module inside the analogue front-end device's VHDL processing unit's VHDL model.

The SV message-mapping VHDL module begins at the *setup* state were the ISO 8802-3 Ethernet frame header fields are assigned. These fields are the destination and source MAC addresses, VLAN tag, frame length, ASN.1 *savPDU* tag and frame length. Default values defined in the IEC 61850-9-2LE are assigned to the header except for the source MAC address. Table 4.11 shows the default values assigned to the ISO 8802-3 Ethernet frame header fields.

Table 4.11: **SV frame fields**

Parameter	Value	Comments
Destination MAC Address	01:0C:CD:04:00:01	Multi-cast address for all destination subscribers.
VLAN Tag	0x8100	VLAN tags are used to separate high priority information data based on IEEE 802.1Q
VID	0	Not used
Priority	4	Default priority 4 defined in the IEC 61850-9-2LE
EtherType	0x88BA	IEEE registered EtherType for sampled value messages
APPID	0x4000	Default APPID according to IEC 61850-9-2LE
Length	m + 8	m represents the length SV APDU

The length of the SV Ethernet frame depends on the length of the SV APDU which in turn depends on the length of the fields encapsulated within. The length of some fields of the SV PDU are fixed, for example, the PhsMeas1 dataset is 64 bytes long as defined in the IEC 61850-9-2LE and the svID for this sampled value stream is hardcoded to "Ncube_MUcput0001". The length of the SV PDU and the Ethernet frame are calculated using Equation 4.8 to Equation 4.11.

$$\mathbf{l_{seq\_ASDU1} = l_{svID} + l_{dataset} + 17}$$

Equation 4.8

$$\mathbf{l_{seq_{ASDUs}} = l_{seq_{ASDU1}} + 2}$$

Equation 4.9

$$\mathbf{l_{savPDU} = l_{seq\_ASDUs} + 5}$$

Equation 4.10

$$\mathbf{l_{sv\_frame} = l_{savPDU} + 8}$$

Equation 4.11

With reference to Equation 4.8 to Equation 4.11, the constants refer to the fixed number of bytes of the tag and value fields encapsulated in the parent field. For example, using Figure 3.17 on page 77 and Equation 4.10, the value of $\mathbf{l_{savPDU}}$ is equal to $\mathbf{l_{seq\_ASDUs} + 5}$, the constant (5) is the number of bytes which make up the *noASDU* field (3 bytes), the *sequence of ASDU* tag and length (2 bytes).

After the header fields of the ISO 8802-3 Ethernet frame are set, the finite state machine transitions to the *readSVCB* state. In this state, the sampled value APDU is configured by making use of a VHDL record. Records are powerful VHDL constructs for instantiating composite packages containing variables of different data types similar to the *struct* construct in the C language. The VHDL record generates a package containing all the SV APDU fields as illustrated in Figure 4.23.

```
--Sampled value APDU record
type ASDU is
    record
        seqASDU : byte_array(integer range 0 to 1);       -- Sequence
        svID : byte_array (integer range 0 to 20);        -- TAG |
        smpCnt : byte_array (integer range 0 to 3);       -- 4 byte
        confRev : byte_array (integer range 0 to 5);   -- 6 byte field
        smpSynch :byte_array (integer range 0 to 2);   -- 3 byte field
        seqData :  byte_array (integer range 0 to 1);     -- 2 byte -
    end record ASDU;
    type APDU is
    record
        savAPDU : byte_array (integer range 0 to 3);      -- 4 byte
        noASDU : byte_array (integer range 0 to 2);       -- 3 byte
        seqofASDU : byte_array ( integer range 0 to 3);     -- 4 byte
        asdu : ASDU;
    end record APDU;
```

Figure 4.23: **sampled value APDU generated using a VHDL record**

The sampled value PDU is encoded using the Basic Encoding Rules (BER) of the ASN.1 standard. In this *readSVCB* state the BER tags of the SV PDU attributes are assigned using the values defined in the IEC 61850-9-2LE. The lengths of the attributes are assigned using the hardcoded length in case of the smpCnt, confRev, smpSynch and the seqData attributes while the lengths of the other fields were calculated using Equation 4.8 to Equation 4.11.

Table 4.12 shows the default and calculated values assigned to the sampled value APDU.

Table 4.12: **SV APDU fields and their values**

Parameter	Tag	Length	Value
savPDU	0x60	l_{savPDU}	Contains TLV triplets
noASDU	0x80	1	1
Sequence of ASDUs	0xA2	$l_{seq_{ASDUs}}$	Contains TLV triplets
Sequence of ASDU1	0x30	$l_{seqASDU1}$	Contains TLV triplets
svID	0x80	16	Ncube_MUcput0001
smpCnt	0x82	4	0-3999
confRev	0x83	4	1
smpSynch	0x85	1	TRUE
Sequence of Data	0x87	64	InnATCTR1.Amp.instMag.i InnATCTR1.Amp.q InnBTCTR2.Amp.instMag.i InnBTCTR2.Amp.q InnCTCTR3.Amp.instMag.i InnCTCTR3.Amp.q InnNTCTR4.Amp.instMag.i InnNTCTR4.Amp.q UnnATVTR1.Vol.instMag.i UnnATVTR1.Vol.q UnnBTVTR2.Vol.instMag.i UnnBTVTR2.Vol.q UnnCTVTR3.Vol.instMag.i UnnCTVTR3.Vol.q UnnNTVTR4.Vol.instMag.i UnnNTVTR4.Vol.q

In Table 4.12, the values of $l_{seqASDU1}$, $l_{seqASDUs}$ and l_{savPDU} are calculated using Equation 4.8 to Equation 4.10 respectively.

Figure 4.24 shows a snippet of VHDL code for assigning values for the SV APDU fields in the instantiated record.

```
apdu.asdu.seqASDU(0)<=x"30";          -- sequence of ASDU tag 0x30
conv_int := std_logic_vector(to_signed(len_seqASDUn, 32));      -
-
-- length of seqASDU(n)
apdu.asdu.seqASDU(1) <= x"62"; -- length of sequence of ASDUs
apdu.asdu.svID(0) <= x"80";       -- svID tag 0x80
conv_int := std_logic_vector(to_signed(SV_ID'right + 1, 32)); --
apdu.asdu.svID(1) <= conv_int(24 to 31);                -- assign
for i in 0 to SV_ID'right loop
       apdu.asdu.svID(i + 2) <= SV_ID(i);           -- copy the
end loop;
apdu.asdu.smpCnt(0) <= x"82";
apdu.asdu.smpCnt(1) <= x"02";
apdu.asdu.smpCnt(2) <= x"00";
apdu.asdu.smpCnt(3) <= x"00";
apdu.asdu.confRev(0) <= x"83";
apdu.asdu.confRev(1) <= x"04";
apdu.asdu.confRev(2) <= x"00";
apdu.asdu.confRev(3) <= x"00";
apdu.asdu.confRev(4) <= x"01";
apdu.asdu.confRev(5) <= x"01";
apdu.asdu.smpSynch(0) <= x"85";
apdu.asdu.smpSynch(1) <= x"01";
apdu.asdu.smpSynch(2) <= x"00";
apdu.asdu.seqData(0) <= x"87";
apdu.asdu.seqData(1) <= x"40";
setup_Complete <= '1';
```

Figure 4.24: **Building sampled value APDU using a VHDL record**

Once the sampled value APDU is generated using a VHDL record and the data assigned, this data is copied from this record onto the Ethernet frame (byte array) and then the state machine transitions to the *sampling* state. In this state, the state machine remains idle, waiting for time synchronised digital samples of the eight analogue inputs from the AFE. As stated previously the AFE reads the ADC output every 250 µs and appends a sample counter (*smpCnt*) value before storing this information in a block RAM in the FPGA. The mapping VHDL module reads the stored data and uses it to calculate the voltage and current samples.

The instantaneous primary voltage and current values are calculated using the ADC output code for the specific channel, the maximum primary voltage/current (V_{MAX}/I_{MAX}) and the ADC output code corresponding to the reference voltage (V_{REF}) using Equation 4.12 and Equation 4.13. The ADC output code corresponding to the reference voltage (V_{REF}) is equal 0x7FFFFF and the values of V_{MAX} and I_{MAX} are given Table 4.4 and Table 4.5 respectively.

$$\mathbf{UnnTVTR_{A,B,C,N}} = \frac{\mathbf{ADC\_OUTPUT\_CODE}}{\mathbf{0x7FFFFF}} \times (\mathbf{V_{MAX}} \times \mathbf{100})$$

Equation 4.12

$$\mathbf{InnTCTR_{A,B,C,N}} = \frac{\mathbf{ADC\_OUTPUT\_CODE}}{\mathbf{0x7FFFFF}} \times (\mathbf{I_{MAX}} \times \mathbf{1000})$$

Equation 4.13

The IEC 61850-9-2LE guideline defines the *sVC.scaleFactor* of 0.001 and 0.01 for current and voltage measurements, this therefore means that the current and voltage measurements published by an IEC 61850-9-2LE MU must be factored by 1000 and 100 respectively. To meet the *sVC.scaleFactor* requirement, the measured current and voltage samples are factored by 1000 and 100 respectively as shown in Equation 4.12 and Equation 4.13.

The instantaneous voltage and current measurements in the sampled value message dataset are represented as integer values as defined in the IEC 61850-9-2LE but for calculation purposes a floating-point library is required. Floating-point libraries are not available in VHDL-93 so therefore library files designed for VHDL-2008 have been used. Even though these file are designed for the VHDL-2008, a VHDL package compatible with VHDL-93 is available at the VHDL site (VHDL, 2015). Compared to fixed-point arithmetic, floating point arithmetic take up to three times the hardware occupied by the former but maintains the accuracy of results to a wider range (Bishop, 2006).

For the purposes of evaluating the accuracy and functionality of the developed SV message mapping VHDL module, the 32-bit fields of each of the quality flags of the eight voltage and current sampled values is hardcoded to 0x00000000. With reference to Table 3.7 in Chapter Three and APPENDIX A, this value (0x00000000) means that the sampled values are good (no errors) and are measured from the process and not substituted.

Figure 4.25 shows an extraction of the VHDL code for calculating the instantaneous voltage and current samples using Equation 4.12 and Equation 4.13 respectively.

```
    -------------------------------------------------------------------
-- Calculate Amps sample value using 24 bit input, Ref (Maximum
Deflection), input current
    ----------------------------------------------------------------
function Calc_AMPS  (signal Sample : in std_logic_vector(23 downto 0))
return std_logic_vector is
    variable TCTR : std_logic_vector(31 downto 0) := x"00000000";
    variable signedVal : signed(23 downto 0) := x"000000";
    variable Ref, tctr_mag,inTCTR : float(8 downto -23) := x"00000000";
    begin
        signedVal := signed(Sample);
        inTCTR := to_float(signedVal, Ref);
        Ref := to_float(0.282989445, Ref);
        tctr_mag := inTCTR * Ref;
        TCTR(31 downto 0) := std_logic_vector(to_signed(tctr_mag, 32));
        return TCTR;
    end Calc_AMPS;
    --------------------------------------------------------------
    -- Calculate Volts sample value using 24 bit input, Ref (Maximum
Deflection), input current
    ----------------------------------------------------------------
function Calc_VOLTS  (signal Sample : in std_logic_vector(23 downto 0))
return std_logic_vector is
    variable TVTR : std_logic_vector(31 downto 0);
    variable signedVal : signed(23 downto 0);
    variable Ref, tvtr_mag,inTVTR : float(8 downto -23);
    begin
        signedVal := signed(Sample);
        inTVTR := to_float(signedVal, Ref);
        Ref := to_float(0.146151345, Ref);                        --
        tvtr_mag := inTVTR * Ref;                                   -
        TVTR(31 downto 0) := std_logic_vector(to_signed(tvtr_mag, 32));
        return TVTR;
    end Calc_VOLTS;
```

Figure 4.25: **Instantaneous sample voltage and current calculation**

After calculating the voltage and current samples, the SV message-mapping VHDL module copies these samples into the ISO 8802-3 Ethernet frame and transitions to the *transmission* state. In this *transmission* state, The SV frame bytes are shifted out to the PHY chip at 100 Mbps using a limited-function MAC implementation shown in Figure 4.26.

```
when Transmitting =>        -- Frame transmission state
    if crc_clk = '1' then
       if byte_num < 7 then          -- Send preamble
           m2p_TDATA<=x"AA";
       end if;
       if byte_num = 7 then                      -- Send SFD
          m2p_TDATA<=x"AB";
       end if;
       if byte_num >= PREAMBLE_SOF_LEN and byte_num <= len_svpacket + PREAMBLE_SOF_LEN then           -- Send Data LSB first
          if byte_num >= SampleIndex + PREAMBLE_SOF_LEN and byte_num < byte_num + SAV_CDC_SIZE + PREAMBLE_SOF_LEN then
             for I in 0 to 7 loop
                m2p_TDATA(I) <= cdcSamples(byte_num -(SampleIndex + PREAMBLE_SOF_LEN))(7-I);
             end loop;
          else
             for I in 0 to 7 loop
                m2p_TDATA(I) <= sv_frame(byte_num - 8)(7-I);
             end loop;
          end if;
       end if;
       if byte_num > len_svpacket+8 and byte_num < len_svpacket+13 then            --send CRC MSB first
          m2p_TDATA <= not crc(byte_num - len_svpacket-9);
       end if;
       -- Manage CRC computation
       if byte_num > 2 and byte_num < 7 then
          crc_en <= '1';
          crc_is_msb <= '1';
          crc_data_in <= not sv_frame(byte_num-3);
       end if;
-------------------------------------------------------------------------
       if byte_num > 6 and byte_num < len_svpacket + 4 then
          crc_en <= '1';
          crc_is_msb <= '1';
          if byte_num >= SampleIndex + PREAMBLE_SOF_LEN and byte_num < byte_num + SAV_CDC_SIZE + PREAMBLE_SOF_LEN then
             crc_data_in <= cdcSamples(byte_num -(SampleIndex + PREAMBLE_SOF_LEN));
          else
             crc_data_in <= sv_frame(byte_num-3);
          end if;
    crc_clk<=not crc_clk;
end case;
```

Figure 4.26: MAC implementation in VHDL for ISO 8802-3 Ethernet frame transmission

The finites state machine is designed to return to the *sampling* state after transmitting the SV message. This calculation and transmission process executes continuously until the device is power cycled. The sampled value messages published by the developed mapping VHDL module were checked using SVScout and Wireshark and no errors were detected. The resulting MU prototype was tested in the laboratory to determine the accuracy of the published sampled values and its conformance to the IEC 61850-9-2LE.

The next section concludes this chapter and highlights challenges encountered during the VHDL module development and hardware integration for mapping and GOOSE and SV messages on the Xilinx Spartan 6 FPGA platform.

4.5 Conclusion

This chapter presented the design scope for mapping GOOSE and SV messages according to the IEC 61850-8-1 standard and IEC 61850-9-2LE on an FPGA. The Xilinx Spartan 6 FPGA has been selected as the platform for implementing the mapping VHDL modules because of its data streamlining and parallel processing capabilities. The data models and datasets of the information published in the GOOSE and SV messages were presented in thesis document.

This chapter also discussed the AFE module provided by the CSAEMS at the CPUT to facilitate the sampling of analogue signals from the instrument transformers. The VHDL code developed for mapping GOOSE and SV messages was discussed and is also made available in APPENDIX E and APPENDIX F.

The Xilinx Spartan 6 FPGA does not have sufficient primitive 18 Kb dual-port block random access memory (RAMB16BWER) for calculating and storing the primary voltage and current samples for the four voltage and four current phases. Due to the inadequacy of the Xilinx Spartan 6 FPGA, the developed MU prototype only calculates phase A and B voltage and current samples while phase C and N sampled values are set to zero. The step-down transformers in the AFE device introduced a 180º phase shift between the injected analogue signals and published sampled values.

The combination of the IEC 61850 process bus message-mapping VHDL modules with the AFE produces a limited-function GOOSE monitoring node and a Merging Unit prototype. These two limited-function process level IED prototypes were evaluated in the laboratory in order to determine their performance and accuracy characteristics. IEC 61850 standard-compliant software programs (SVScout, IEDScout and Wireshark) and the CMC 256*plus* test sets were used in the validation and evaluation processes.

Chapter Five presents laboratory tests conducted to evaluate the GOOSE monitoring node and limited-function MU prototypes resulting from the combination of the AFE and the developed GOOSE and SV message mapping VHDL modules respectively. Chapter Five details the conformance test of the GOOSE monitoring node and limited-function MU prototype to the IEC 61850-8-1 standard and IEC 61850-9-2LE respectively. Chapter Five also details tests conducted in the laboratory using the CMC 256*plus* test, Wireshark, IEDScout and SVScout software applications for evaluating the accuracy of the measured and calculated values published by the MU prototype and GOOSE monitoring node under nominal and fault conditions of the power system.

CHAPTER FIVE
EVALUATION OF THE GOOSE AND SV MESSAGE MAPPING HARDWARE DESIGNS

5.1 Introduction

The first edition of the IEC 61850 standard defines communication interfaces for data exchange between devices and/or functions in a substation automation system (SAS) as illustrated in Figure 1.2 on page 4. This standard also introduces a process bus to facilitate communication between primary plant equipment and bay level IEDs. This process bus consists of both GOOSE and Sampled Values (SV) messages defined in parts 8-1 and 9-2 of the IEC 61850 standard respectively.

This research project focused on the development of two process level IEDs namely the GOOSE monitoring node and the limited-function Merging Unit (MU) prototype. Chapter Four of this thesis discussed the development of VHDL modules for mapping and publishing GOOSE and SV messages on an FPGA platform. Chapter Four also presented the integration of the Analogue Front-End (AFE) to the GOOSE/SV message-mapping FPGA-based VHDL modules to produce the limited-function GOOSE monitoring node and Merging Unit prototype.

This chapter details the test procedures conducted in the laboratory using the Test Universe software, CMC 256*plus* test set, IEDScout and SVScout to determine the accuracy and performance characteristics of the developed GOOSE monitoring node and limited-function MU prototype. Wireshark software was used to validate the structure of the GOOSE and SV messages published by the GOOSE monitoring node and MU prototype against that defined in the IEC 61850-8-1 standard and in the IEC 61850-9-2LE respectively.

Section 5.3 presents test procedures conducted in the laboratory for evaluating the accuracy and performance characteristics of the developed GOOSE monitoring node. The first test conducted in section 5.3.1 compares the structure of the

GOOSE messages published by the developed GOOSE monitoring node against that defined in the IEC 61850-8-1 standard.

The next test validates the GOOSE client state machine implemented by the GOOSE message-mapping VHDL module against that defined in the IEC 61850-8-1 standard. For this test, the client state machine is validated by observing the transition of the GOOSE message state and sequence numbers (*stNum* and *sqNum*) during a simulated a voltage-swell fault.

Section 5.3.4 evaluates the profile of the calculated RMS voltage published by the GOOSE monitoring node when a voltage-sag fault is simulated using the CMC 256*plus* test set. The profile of the RMS voltage published by the GOOSE monitoring node will be compared to that of the analogue voltage injected by the CMC 256*plus* test set. Section 5.3.5 presents a laboratory interoperability test in which GOOSE messages published by the GOOSE monitoring node are subscribed to by the MiCOM P546 IED. This test will validate whether messages published by the developed GOOSE monitoring node can be used by any IEC 61850 standard-compliant IED.

On the other hand, section 5.4 details tests procedures conducted to evaluate Sampled Value (SV) messages published by the developed limited-function MU prototype using Wireshark and SVScout. In this section, the structure of SV messages published by the developed FPGA-based MU prototype is validated against that defined in the UCAIug implementation guideline of the IEC 61850-9-2 standard (IEC 61850-9-2LE).

Section 5.4 also presents performance and accuracy tests conducted by injecting distorted current and voltage signals to emulate common power system disturbances. These tests are crucial because the power grid is prone to disturbances/faults caused by natural elements, earth faults and or harmonics. Section 5.4.2 assesses the accuracy of the sampled values published by the developed limited-function MU prototype by comparing these samples to those published by the CMC 256*plus* test set.

Section 5.4.4 provides an interoperability test conducted in the laboratory using a MiCOM P444 IED to determine whether the SV messages published by the MU prototype can be subscribed to by any IEC 61850-9-2LE-compliant IED.

5.2 Outline of Evaluation

To facilitate this research project, an Analogue Front-End (AFE) device was provided by the Centre for Substation Automation and Energy Management Systems (CSAEMS). The AFE handles the synchronisation and sampling of the analogue signals from the instrument transformers (CTs and VTs). Combining this AFE and the FPGA-based SV and GOOSE message-mapping VHDL modules as discussed in Chapter Four produces two limited-function process levels IEDs; these are the GOOSE monitoring node and the MU prototypes.

For the following laboratory tests, the CMC 256*plus* test set was used as a voltage and current source instead of voltage and current transformers (VTs and CTs) connected to a real power system. The CMC 256*plus* test set is an advanced IEC 61850 IED tester and calibration tool that can function as both a current/voltage signal source and a sampled value message publisher (Rudigier & Steinhauser, 2015).

The CMC 256*plus* test set was configured using the Omicron Test Universe package, this package contains function-oriented test modules for generating different waveforms through the CMC 256*plus* test set. The software modules under the Test Universe package used for evaluation purposes include the *QuickCMC*, *Ramping*, *Harmonics* and *Control Center* modules.

The test bench for evaluating the GOOSE monitoring node and Merging Unit prototypes is similar except that for the former, the CMC 256*plus* test set is configured to generate single phase voltages and three phase signals (current and voltage) for the latter.

Figure 5.1 shows a connection block diagram for evaluating the GOOSE monitoring node and MU prototype using the CMC 256*plus*, Ruggedcom industrial Ethernet switch and two configuration PCs. Figure 5.1 shows two computers, one for running the Test Universe package for configuring the CMC 256*plus* test set and the other computer for capturing the published GOOSE and SV messages.

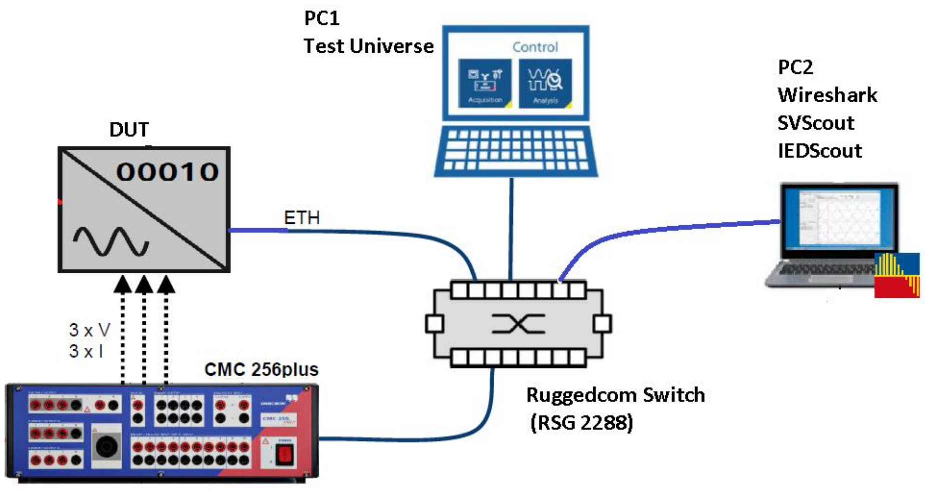


Figure 5.1: **Laboratory connection block diagram for evaluating the DUT**

In Figure 5.1, the Device Under Test (DUT) refers to the Nexys 3 development board implemented as the GOOSE message mapping hardware (GOOSE monitoring node) or SV message mapping hardware (Merging Unit prototype). The evaluation procedures are detailed in either sections 5.3 or section 5.4 for the GOOSE monitoring node and MU prototype respectively. In Figure 5.1, PC1 is utilized for analysing the captured COMTRADE files using the Omicron *TransView* module. PC2 runs the Wireshark, IEDScout and the SVScout software applications for capturing and analysing the structure of the published GOOSE/SV message.

Figure 5.2 shows the laboratory setup of the test equipment and the DUT according to the block diagram shown in Figure 5.1.

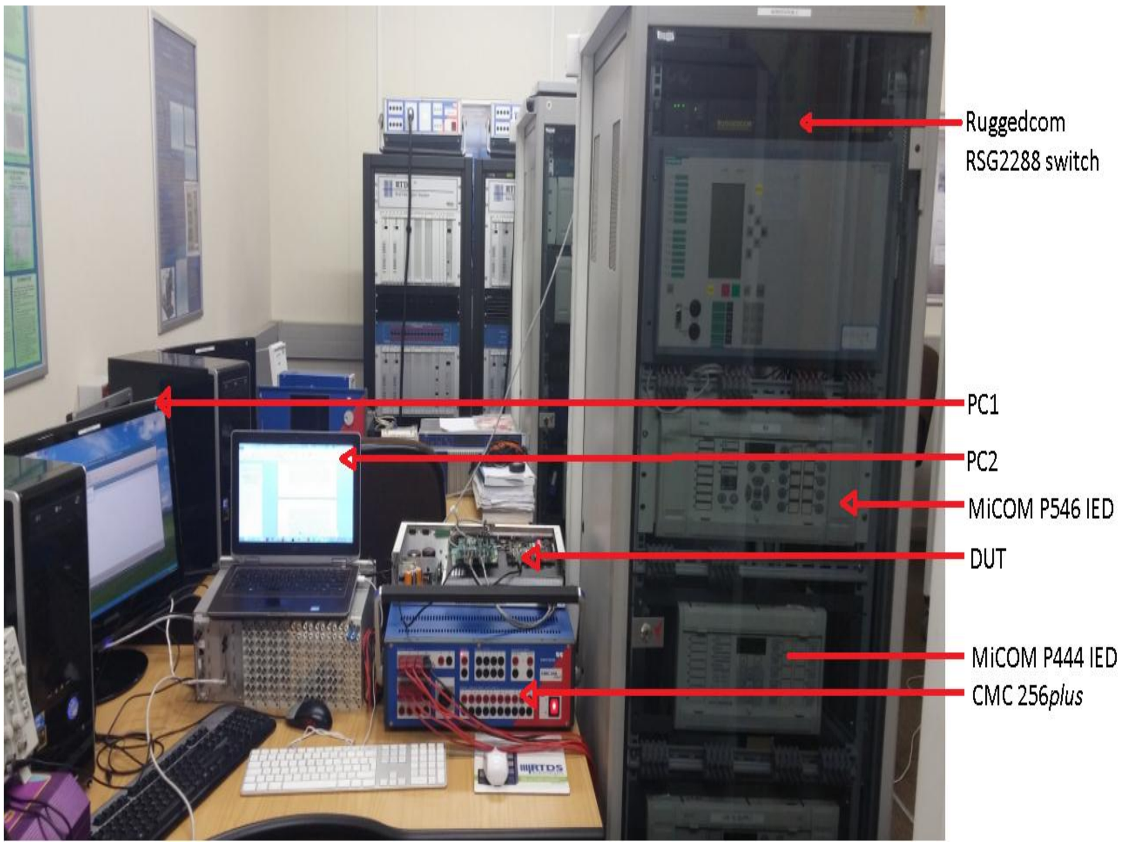


Figure 5.2: **Setup of equipment in the laboratory**

Figure 5.3 shows how the voltage and current signals generated by the CMC 256*plus* test are connected to the step-down transformers of the AFE. This figure also shows the serial peripheral interface between the Nexys 3 development board and the ADS131E08 ADC development board.

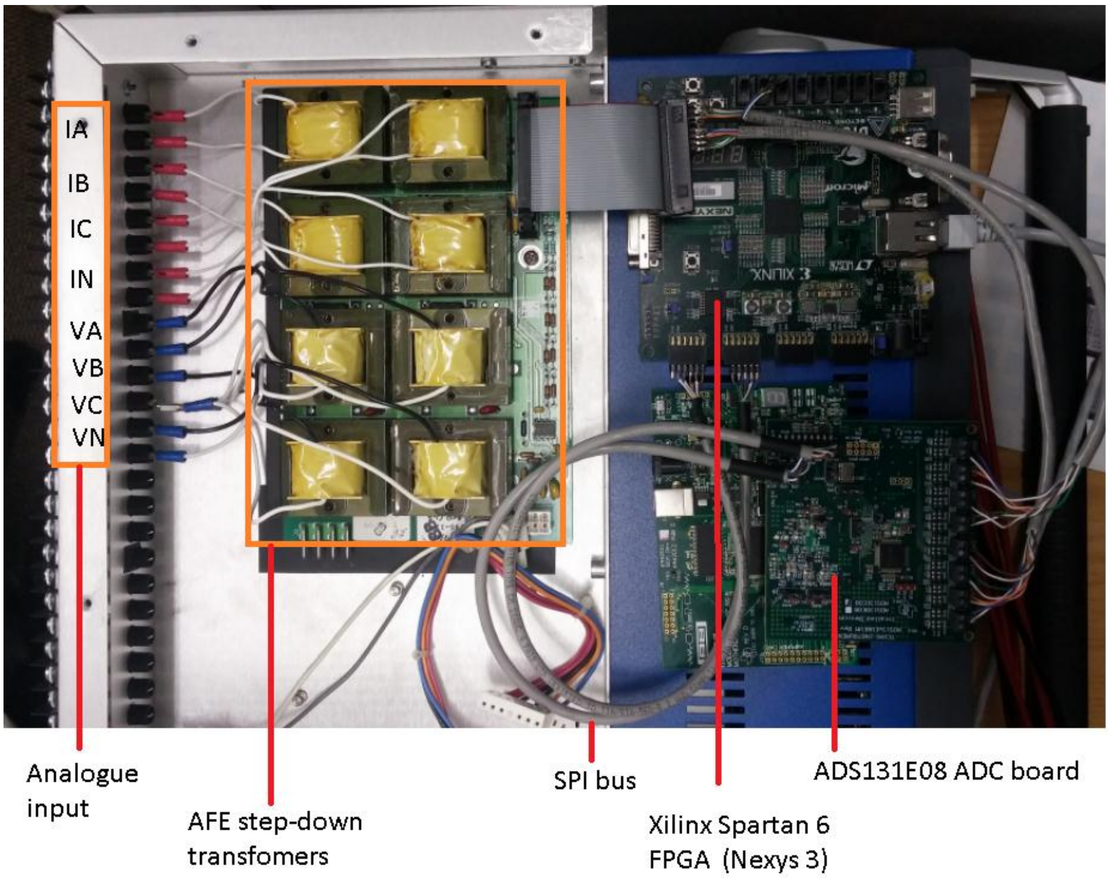


Figure 5.3: **Connection between the CMC 256*plus* test set and the AFE through the step-down transformers**

As stated previously in Chapter Four, the nominal power system line to neutral voltages and currents are chosen to be equal to 3.3 kV$_{RMS}$ and 100 A$_{RMS}$. Using the primary to secondary signal ratio of 100:1, under nominal conditions the CMC 256*plus* generates line to neutral voltages and currents equal to 33 V$_{RMS}$ and 1 A$_{RMS}$. For all the tests conducted in this chapter, the phase angle between the three phases is 120º equivalent to that of balanced power systems.

All voltage and current signals generated using by CMC 256*plus* test set to emulate power system faults are arbitrarily chosen to evaluate the performance and accuracy of the GOOSE monitoring node and MU prototype under such conditions. The power system faults emulated using the CMC 256*plus* test set are the voltage-sag, voltage-swell, frequency variation and harmonic distortion using the *Ramping*, *QuickCMC* and *Harmonics* modules of the Test Universe software.

The following section discusses the test procedures conducted to evaluate the accuracy and performance characteristics of the developed GOOSE monitoring node.

5.3 GOOSE monitoring node laboratory evaluation

This section details the test procedures for evaluating the developed FPGA-based GOOSE monitoring node. This first test procedure in section 5.3.1 validates the structure of GOOSE messages published by the mapping VHDL module against that defined in the IEC 61850-8-1 standard.

The FPGA/IED1/LLN0$GOOSE_Eval dataset published by the GOOSE monitoring node contains the IED1/XCBR1$ST$Pos$stVal and the IED1/MMXN1$MX$Vol$mag data attributes to represent digital and analogue information respectively as discussed in Chapter Four. The Configured IED Description (CID) file for the limited-function GOOSE monitoring node is given in APPENDIX G.

In order to evaluate the GOOSE message-mapping VHDL module implemented on an FPGA platform, the AFE is used to provide ADC samples of the analogue signal injected by the CMC 256*plus* for the MMXN1$MX$Vol$mag attribute. The circuit breaker status (XCBR1$ST$Pos$stVal) was emulated using SW2 switch on the Nexys 3 development board. The GOOSE monitoring node will be referred to as the Device under Test (DUT) in this section and the underlying sub-sections.

GOOSE messages published by the DUT were captured using IEDScout and the information exported to a COMTRADE file for analysis. The exported COMTRADE file contains the published dataset, state and sequence numbers (*stNum* and *sqNum*). The exported COMTRADE file was then used to validate the GOOSE client state machine against that defined in the IEC 61850-8-1 standard

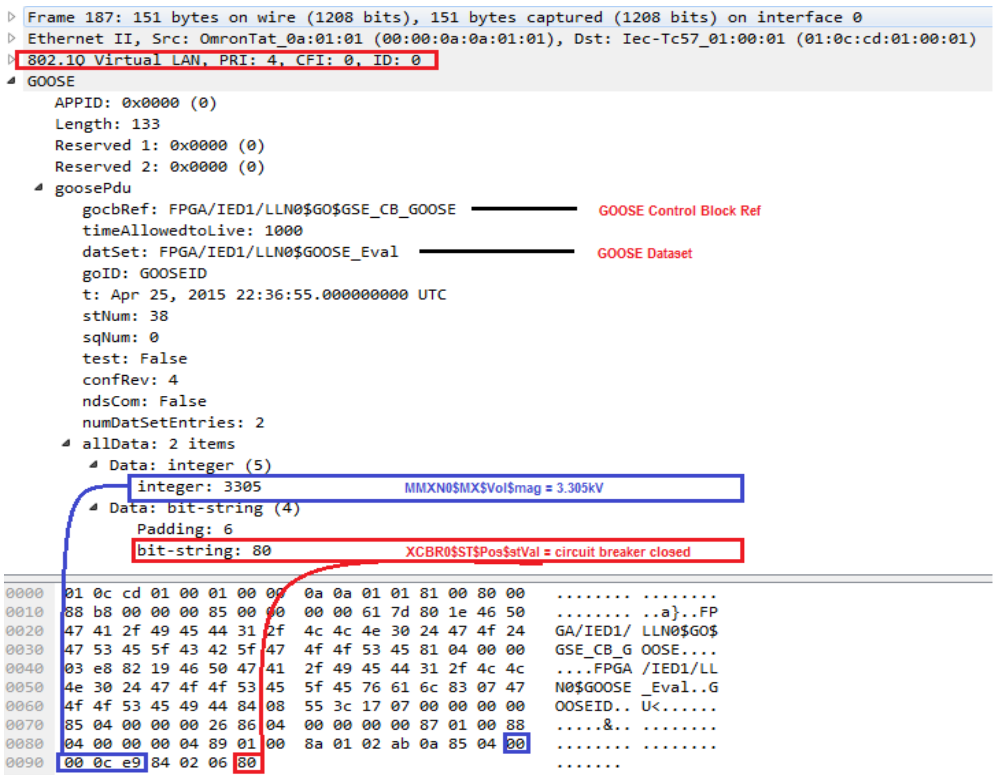
Power networks are prone to disturbances caused by types of loads, configuration of control equipment and natural events; it is therefore important to monitor the value of the MMXN1$MX$Vol$mag when distorted signals are injected. This research project in sections 5.3.3 and 5.3.4 analyses the magnitude profile of the RMS voltage published by the DUT during a voltage-swell and voltage-sag fault simulation respectively. This is analysis is conducted by comparing the profile of the analogue signals generated by the CMC 256*plus* test set to that of RMS values (MMXN1$MX$Vol$mag attribute) published by the DUT.

Sections 5.3.1 to 5.3.5 details the tests conducted in the laboratory for evaluating the developed FPGA-based GOOSE message publisher.

5.3.1 GOOSE message structure evaluation

Wireshark was used for capturing and analysing the GOOSE messages published by the DUT. Omicron's IEDScout software was used to monitor GOOSE traffic generated by the DUT. IEDScout is a powerful tool used to simulate a GOOSE subscriber IED using the publisher IED's CID file or by sniffing GOOSE messages from the Ethernet network. IEDScout also allows the GOOSE message attributes (dataset, *stNum* and *sqNum*) to be exported to a COMTRADE file for further analysis.

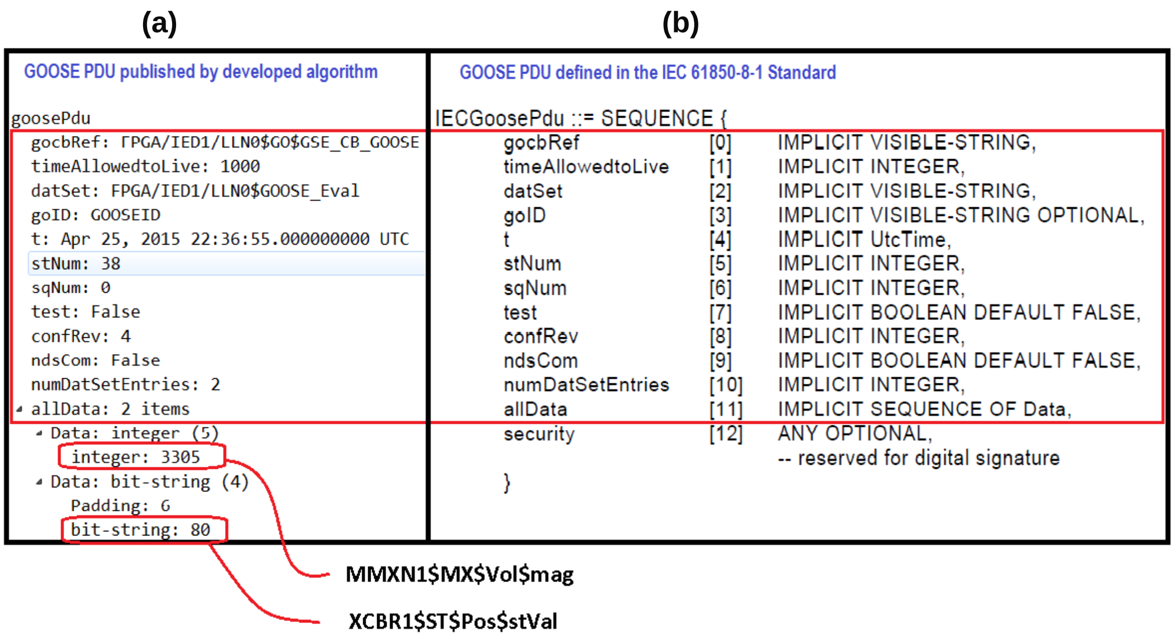
Figure 5.4 below shows a Wireshark capture of a GOOSE message published by the DUT. The GOOSE message shown in Figure 5.4 is based on an ISO 8802-3 Ethernet frame that consists of two sections, the Ethernet header and the GOOSE PDU. The captured GOOSE message in Figure 5.4, shows that the destination MAC address is set to 01:0C:CD:01:00:01 which is the multicast address for publishing messages to multiple subscribers. The APPID and the VLAN tags are set to their default values defined in the IEC 61850-8-1 standard and discussed in section 4.4.1.1.


```
▷ Frame 187: 151 bytes on wire (1208 bits), 151 bytes captured (1208 bits) on interface 0
▷ Ethernet II, Src: OmronTat_0a:01:01 (00:00:0a:0a:01:01), Dst: Iec-Tc57_01:00:01 (01:0c:cd:01:00:01)
▷ 802.1Q Virtual LAN, PRI: 4, CFI: 0, ID: 0
◢ GOOSE
    APPID: 0x0000 (0)
    Length: 133
    Reserved 1: 0x0000 (0)
    Reserved 2: 0x0000 (0)
  ◢ goosePdu
      gocbRef: FPGA/IED1/LLN0$GO$GSE_CB_GOOSE
      timeAllowedtoLive: 1000
      datSet: FPGA/IED1/LLN0$GOOSE_Eval
      goID: GOOSEID
      t: Apr 25, 2015 22:36:55.000000000 UTC
      stNum: 38
      sqNum: 0
      test: False
      confRev: 4
      ndsCom: False
      numDatSetEntries: 2
    ◢ allData: 2 items
      ◢ Data: integer (5)
          integer: 3305
      ◢ Data: bit-string (4)
          Padding: 6
          bit-string: 80

0000  01 0c cd 01 00 01 00 00  0a 0a 01 01 81 00 80 00   ........ ........
0010  88 b8 00 00 00 85 00 00  00 00 61 7d 80 1e 46 50   ........ ..a}..FP
0020  47 41 2f 49 45 44 31 2f  4c 4c 4e 30 24 47 4f 24   GA/IED1/ LLN0$GO$
0030  47 53 45 5f 43 42 5f 47  4f 4f 53 45 81 04 00 00   GSE_CB_G OOSE....
0040  03 e8 82 19 46 50 47 41  2f 49 45 44 31 2f 4c 4c   ....FPGA /IED1/LL
0050  4e 30 24 47 4f 4f 53 45  5f 45 76 61 6c 83 07 47   N0$GOOSE _Eval..G
0060  4f 4f 53 45 49 44 84 08  55 3c 17 07 00 00 00 00   OOSEID.. U<......
0070  85 04 00 00 00 26 86 04  00 00 00 00 87 01 00 88   .....&.. ........
0080  04 00 00 00 04 89 01 00  8a 01 02 ab 0a 85 04 00   ........ ........
0090  00 0c e9 84 02 06 80                               .......
```
GOOSE Control Block Ref
GOOSE Dataset
MMXN0$MX$Vol$mag = 3.305kV
XCBR0$ST$Pos$stVal = circuit breaker closed

Figure 5.4: Captured GOOSE message

The PDU encapsulated in the GOOSE message published by the DUT is shown on the left side (a) of Figure 5.5 and the right side (b) shows the PDU defined in the IEC 61850-8-1 standard.


(a)

GOOSE PDU published by developed algorithm

```
goosePdu
  gocbRef: FPGA/IED1/LLN0$GO$GSE_CB_GOOSE
  timeAllowedtoLive: 1000
  datSet: FPGA/IED1/LLN0$GOOSE_Eval
  goID: GOOSEID
  t: Apr 25, 2015 22:36:55.000000000 UTC
  stNum: 38
  sqNum: 0
  test: False
  confRev: 4
  ndsCom: False
  numDatSetEntries: 2
◢ allData: 2 items
  ◢ Data: integer (5)
      integer: 3305
  ◢ Data: bit-string (4)
      Padding: 6
      bit-string: 80
```

(b)

GOOSE PDU defined in the IEC 61850-8-1 Standard

```
IECGoosePdu ::= SEQUENCE {
      gocbRef            [0]   IMPLICIT VISIBLE-STRING,
      timeAllowedtoLive  [1]   IMPLICIT INTEGER,
      datSet             [2]   IMPLICIT VISIBLE-STRING,
      goID               [3]   IMPLICIT VISIBLE-STRING OPTIONAL,
      t                  [4]   IMPLICIT UtcTime,
      stNum              [5]   IMPLICIT INTEGER,
      sqNum              [6]   IMPLICIT INTEGER,
      test               [7]   IMPLICIT BOOLEAN DEFAULT FALSE,
      confRev            [8]   IMPLICIT INTEGER,
      ndsCom             [9]   IMPLICIT BOOLEAN DEFAULT FALSE,
      numDatSetEntries   [10]  IMPLICIT INTEGER,
      allData            [11]  IMPLICIT SEQUENCE OF Data,
      security           [12]  ANY OPTIONAL,
                               -- reserved for digital signature
      }
```

Figure 5.5: Comparison between the GOOSE PDU published by the DUT and that defined in the IEC 61850-8-1 standard

From this illustration (Figure 5.5), it can be noted that the captured frame encapsulates a GOOSE PDU that matches the one defined in the IEC 61850-8-1 standard. This test proves that the structure of GOOSE messages published by the DUT conforms to the IEC 61850-8-1 standard.

The CMC 256*plus* test set was configured to inject 33 V_{RMS} into the GOOSE monitoring node and with reference to Figure 5.5, the calculated RMS voltage modeled as the IED1/MMXN1$MX$Vol$mag is equal to 3.305 kV_{RMS}. The calculated RMS voltage has a percentage error of 0.15% with respect to the 3.3 kV_{RMS} expected from the voltage injected. This measurement and calculation accuracy will suffice for protection and metering functions.

With reference to Figure 5.5, the emulated circuit breaker position (SW2) mapped to the IED1/XCBR1$ST$Pos$stVal is equal to 0x80. This MMS encoded value 0x80 represents the IED1/XCBR1$ST$Pos$stVal bit-string value “10” encoded using the MMS rules as described in section 4.4.1.1.2 in Chapter Four. The status of the circuit breaker is 0x80 and 0x40 when SW2 is toggled between ON and OFF positions as expected.

The following section details experiments conducted in the laboratory for evaluating the GOOSE client state machine. This procedure focuses on the transition of the state and sequence numbers (*stNum* and *sqNum*) of the published GOOSE messages.

5.3.2 GOOSE client state machine evaluation

The IEC 61850-8-1 standard defines a state machine which controls the functions of a GOOSE client IED. This client state machine defines the transitions of state and sequence numbers (*stNum* and *sqNum*) when the status/value of object(s) in the referenced dataset changes. The GOOSE client state machine is illustrated in Figure 3.12 in page 70.

This section evaluates the client state machine implemented by the GOOSE monitoring node against that defined in the IEC 61850-8-1 standard. In order to evaluate the client state machine, the CMC 256*plus* test set was configured to generate and inject a single phase voltage-swell into the DUT. The dataset, state and sequence numbers (*stNum* and *sqNum*) in the GOOSE message published by the DUT were then captured and recorded into a COMTRADE file.

Figure 5.6 shows the transition of *stNum* and *sqNum* attributes in the captured GOOSE messages when the injected voltage increased from nominal (33 V_{RMS}) to peak at 49.5 V_{RMS} equating to primary voltages of 3.3 kV_{RMS} and 4.95 kV_{RMS} respectively.

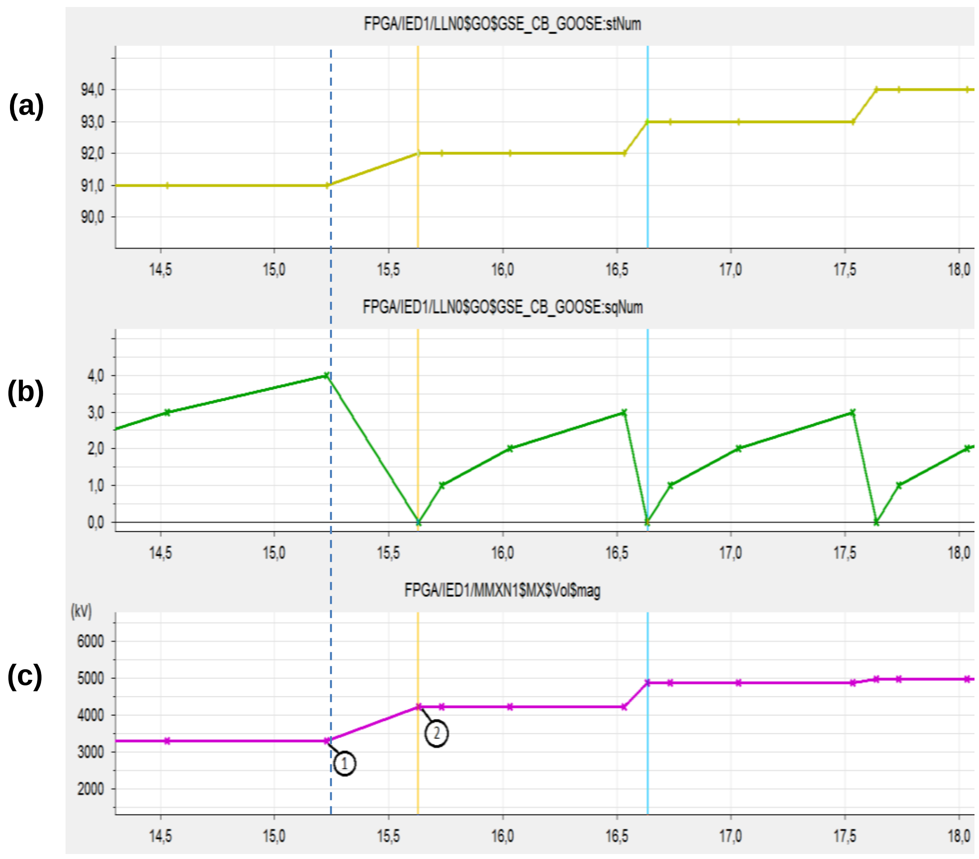


Figure 5.6: **Transition of *stNum* and *sqNum* in GOOSE messages published by the DUT**

With reference to the chart in Figure 5.6, the point number 1 on graph (c) shows the corresponding *stNum* and *sqNum* of a GOOSE message published before the occurrence of a substation event in graphs (a) and (b) respectively. This substation event stimulated by the value of MMXN1$MX$Vol$mag exceeding the deadband marked as point number 2 causes the value of *sqNum* (green graph) to be set to zero and that of *stNum* (yellow graph) to be incremented to 92. Referring to Figure 5.6, the value of *stNum* remains constant at 92 after the substation event and that of *sqNum* increments from zero up until the measured RMS value (MMXN1$MX$Vol$mag) exceeds the deadband parameter.

It can also be noted that after the occurrence of the substation event, marker number 2 in Figure 5.6, the GOOSE message retransmission time starts from 100 ms and increments by 200 ms with every consecutive transmission. The transition of *stNum* and *sqNum* follows this suite during and after the next substation event. This test shows that the client state machine implemented in the GOOSE monitoring node conforms to the definition in the IEC 61850-8-1 standard as illustrated in Figure 3.11 on page 68. The next section analyses the profile of the calculated RMS values published by the DUT during a voltage-swell simulation.

5.3.3 Voltage-swell injection

Voltage-swell faults are common phenomena in power systems which may be caused by switching off of heavy loads. The procedure detailed in this section compares the calculated RMS voltage published by the DUT against the analogue signals generated by the CMC 256*plus* test. For this test, the injected voltage was ramped from nominal (33 V_{RMS}) to peak at 49.5 V_{RMS} using the Test Universe's *Ramping* module. These voltages injected by the CMC 256*plus* test set corresponds to the primary values equal to 3.3 kV_{RMS} and 4.95 kV_{RMS} according to the primary to secondary ration of 100:1. Figure 5.7 shows the voltage-swell fault as simulated by the CMC 256*plus* test set expressed as primary values.

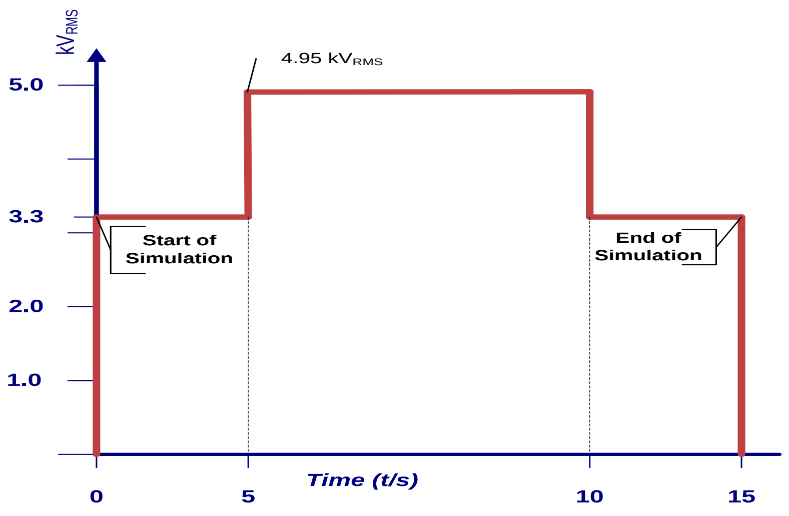


Figure 5.7: **Waveform of voltage signal generated by the CMC 256*plus* test set**

Figure 5.8 shows the transition of *stNum* and *sqNum* and the calculated RMS voltage (MMXN1$MX$Vol$mag) during the voltage-swell fault injection in graphs (a), (b) and (c) respectively.

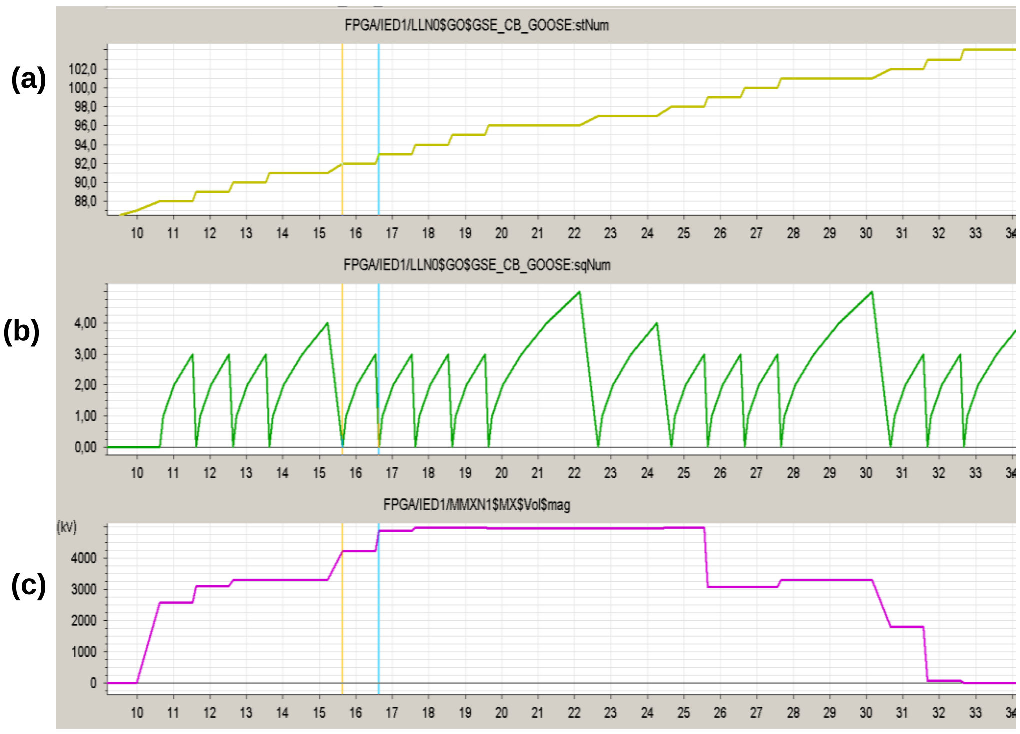


Figure 5.8: *sqNum* and *stNum* transition during voltage-swell injection

With reference to Figure 5.8, it can be noted that the magnitude-time profile of the MMXN1$MX$Vol$mag follows that of the analogue signal injected into the DUT. From Figure 5.8, the nominal and peak voltages published by the DUT before and during the voltage-swell fault injection are equal to 3.305 kV_{RMS} and 4.96kV_{RMS} respectively. The calculated RMS voltage (MMXN1$MX$Vol$mag) is 99.85% accurate.

The next section evaluates the RMS voltage (MMXN1$MX$Vol$mag) published by the DUT during a simulated voltage-sag fault.

5.3.4 Voltage-sag injection

Voltage-sag faults are common phenomena in power systems and they may be caused by phase to ground faults or switching on of heavy load. The procedure detailed in this section compares the calculated RMS voltage published by the DUT against the analogue signals generated by the CMC 256*plus* test. Using the Test Universe’s *Ramping* module, the CMC 256*plus* test set was configured to generate a single phase voltage with a magnitude of 33 V_{RMS} and after 5 seconds the voltage was dropped to 16.5 V_{RMS} for 10 seconds as shown in Figure 5.7. In Figure 5.7 the voltage magnitudes are expressed as primary values using the 100:1 primary to secondary voltage ratio.

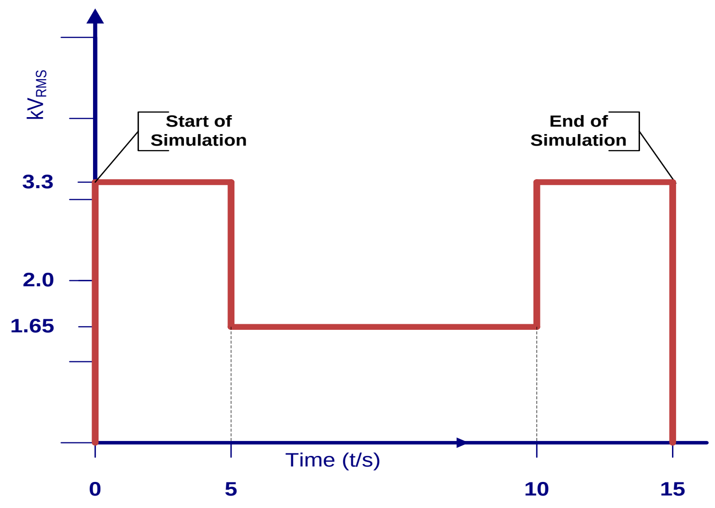


Figure 5.9: **Voltage-sag simulation using the *Ramping* module and the CMC 256*plus* test set**

The voltage signals injected into the DUT are sampled and the calculated root mean square value is published as the MMXN1$MX$Vol$mag object. Graphs (a), (b) and (c) in Figure 5.10 show the transition of *stNum*, *sqNum* and the calculated RMS voltage respectively.

In Figure 5.10, graph (c) show the magnitude of the MMXN1$MX$Vol$mag object (RMS voltage) published by the DUT during the simulation of the voltage-sag fault. This graph is similar to that of the analogue voltage injected into the AFE by the CMC 256*plus* test set shown in Figure 5.9.

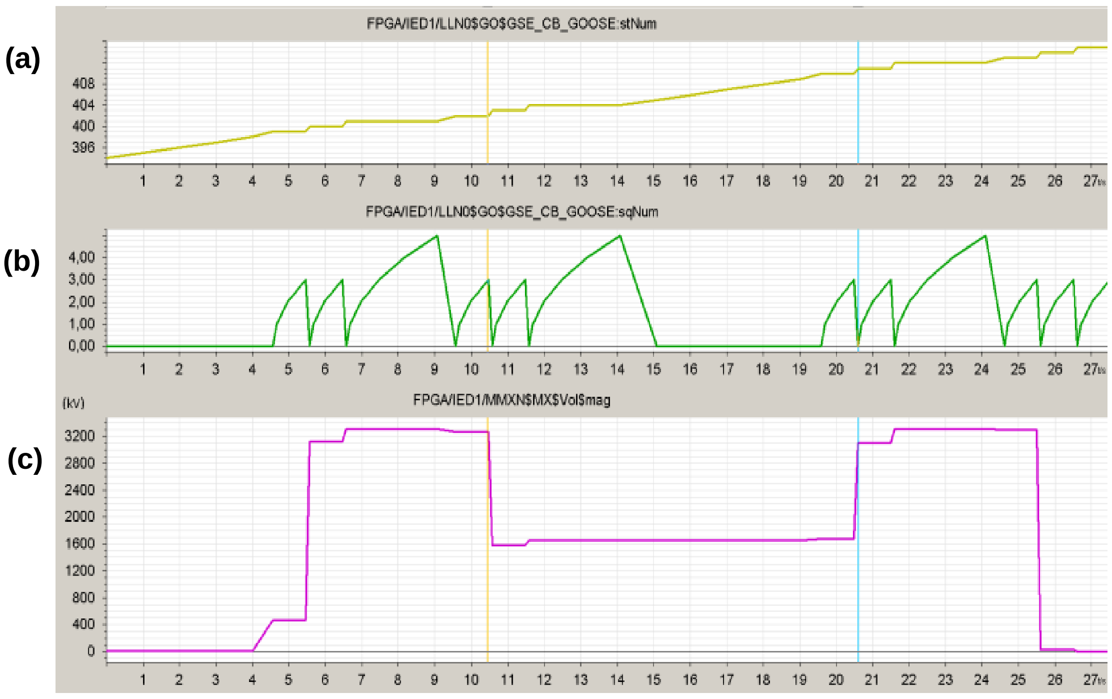


Figure 5.10: **MMXN1$MX$Vol$mag value published by the DUT**

Figure 5.10 also shows the transition of *sqNum* and *stNum* against the value of MMXN1$MX$Vol$mag object, as expected from the client state machine *stNum* increments on occurrence of a substation event (change in MMXN1$ST$Vol$mag) while *sqNum* increments on GOOSE message-retransmission. The RMS voltage before and during the voltage-sag fault was calculated to 0.15% error.

This test also shows that the developed FPGA-based GOOSE monitoring node can be used in protection schemes of real power systems affected by voltage-sag and swell faults. The next section provides the results of an interoperability test of the developed GOOSE monitoring node with an IEC 61850-compliant IED.

5.3.5 GOOSE monitoring node interoperability test

The IEC 61850 standard (Edition one) allows for interoperability between devices from different vendors and for this reason an interoperability test between the developed GOOSE monitoring node and an IEC 61850 certified IED was conducted. For this evaluation, the MiCOM P546 IED was configured to subscribe to GOOSE messages published by the DUT as shown in Figure 5.11.

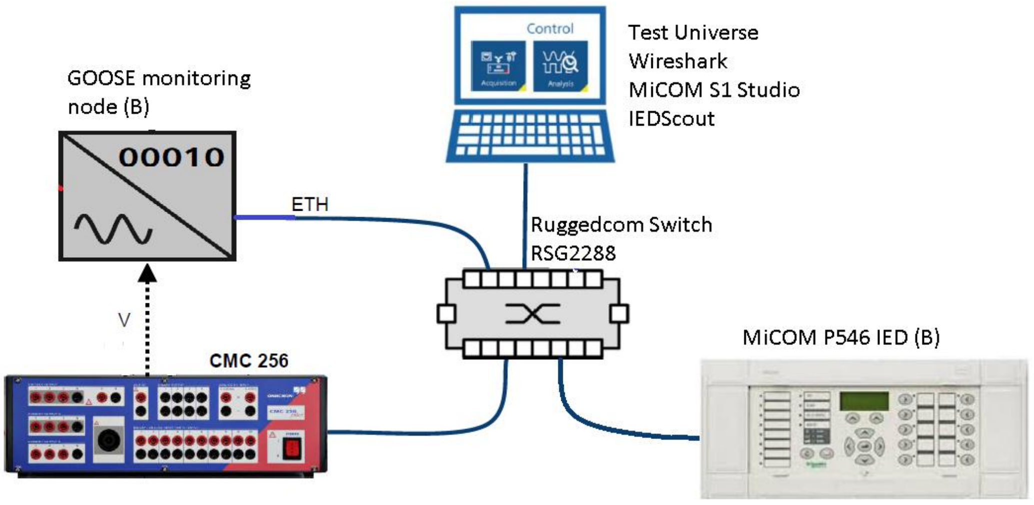


Figure 5.11: **FPGA-based GOOSE monitoring node interoperability test**

In this experiment, the MiCOM P546 IED was configured to display the Boolean status of a virtual input on its status LED (LED1) and at the same time publish the status of this LED using another GOOSE message. The Boolean status of this virtual input (Virtual Input 1) was derived from the FPGA-based GOOSE monitoring node circuit breaker status (FPGA/IED1/XCBR1$ST$Pos$stVal). The GOOSE messages published by the MiCOM P546 were then subscribed to and recorded onto a COMTRADE file by IEDScout. This test setup is a variation of the ping-pong test whereby device A (DUT) publishes messages subscribed to by device B (MiCOM P546 IED) which in turn publishes messages subscribed to by IEDScout.

This experiment will prove that the DUT can publish GOOSE messages that can be subscribed to by an IEC 61850-compliant IED. The MiCOM P546 IED is configured as follows:

1. Create a new system in MiCOM S1 Studio (Schneider Electric, 2016).

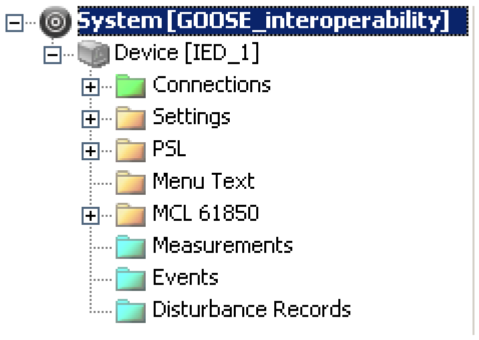
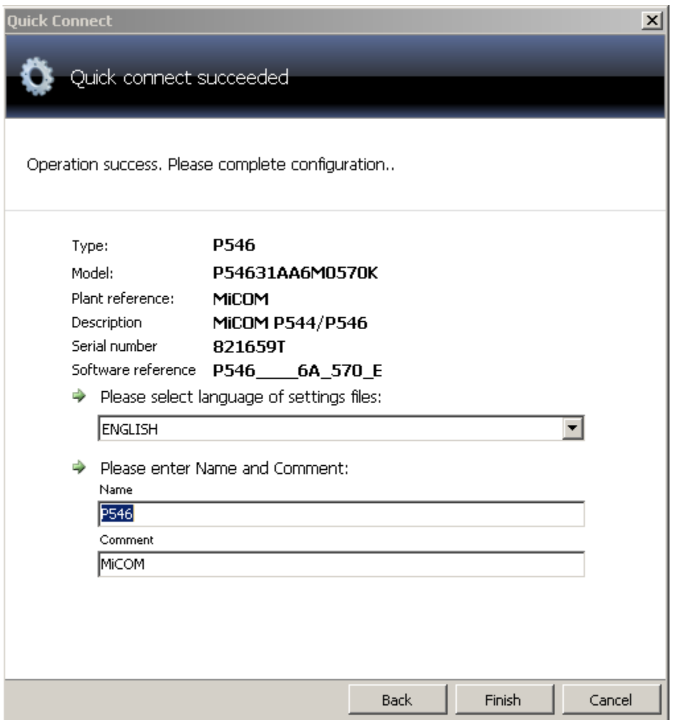


Figure 5.12: **Creating a new system in MiCOM S1 Studio**

2. Creating devices in the system using the **Quick Connect** command which connects to the IED attached via a serial cable. After a successful connection, the Quick Connect command retrieves the IED type, model, serial number and software reference as shown in Figure 5.43.



Operation success. Please complete configuration..

Type:	P546
Model:	P54631AA6M0570K
Plant reference:	MiCOM
Description	MiCOM P544/P546
Serial number	821659T
Software reference	P546____6A_570_E

Please select language of settings files:
ENGLISH

Please enter Name and Comment:
Name
P546
Comment
MiCOM

Figure 5.13: **Connecting to the MiCOM P546 IED using the *Quick Connect* command**

3. Configuring the Programmable Scheme Logic (PSL) of the MiCOM P546 IED to update the status of IED1 using the status of the Virtual Input 1 derived from the FPGA based GOOSE monitoring node circuit breaker status.

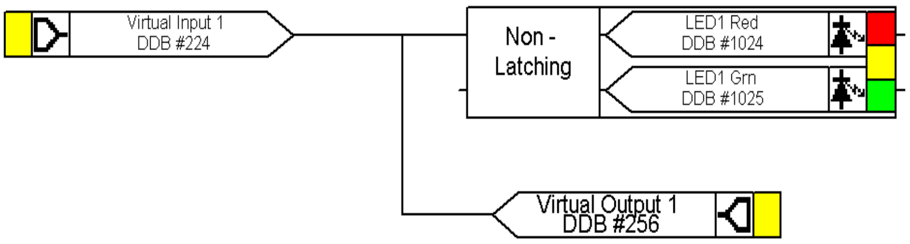
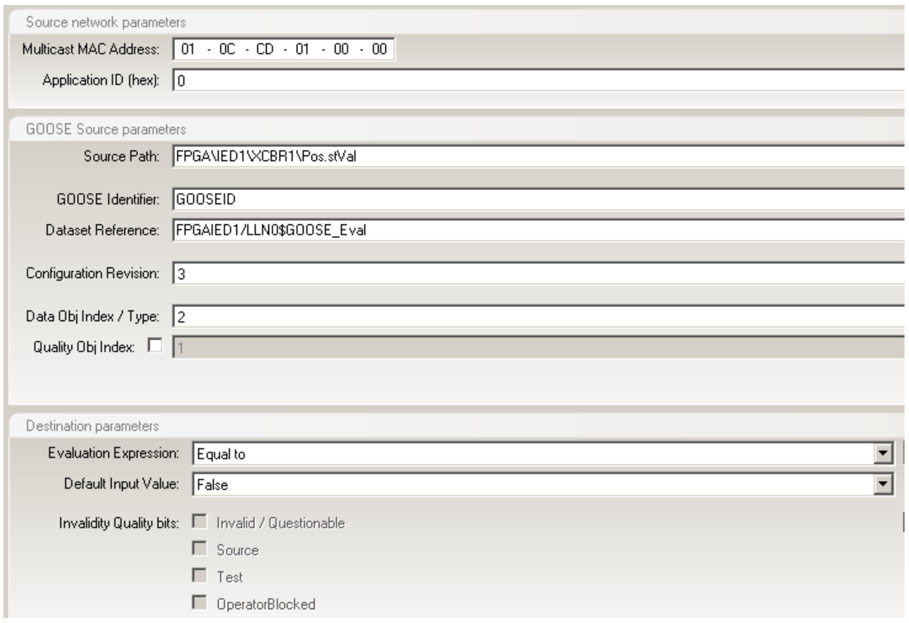


Figure 5.14: PSL configuration for updating status of LED1 using the Virtual Input 1 status

4. Configuring the MiCOM P546 IED to subscribe to GOOSE messages published by the developed GOOSE monitoring node and map the FPGA/IED1/XCBR1$ST$Pos$stVal to Virtual Input 1 by importing the DUT's IID file.



Source network parameters

Multicast MAC Address:	01 - 0C - CD - 01 - 00 - 00
Application ID (hex):	0

GOOSE Source parameters

Source Path:	FPGA\IED1\XCBR1\Pos.stVal
GOOSE Identifier:	GOOSEID
Dataset Reference:	FPGAIED1/LLN0$GOOSE_Eval
Configuration Revision:	3
Data Obj Index / Type:	2
Quality Obj Index: ☐	1

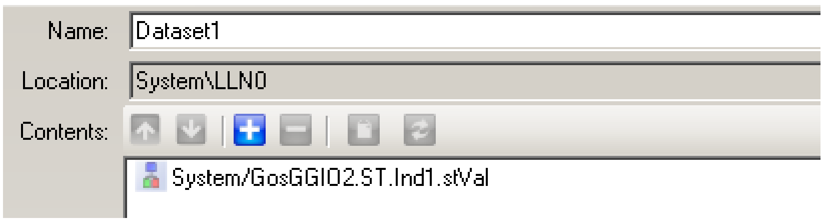
Destination parameters

Evaluation Expression:	Equal to
Default Input Value:	False
Invalidity Quality bits:	☐ Invalid / Questionable
	☐ Source
	☐ Test
	☐ OperatorBlocked

Figure 5.15: Configuring the MiCOM P546 IED to subscribe to GOOSE messages published by the DUT

With reference to Figure 5.15, the status of Virtual Input 1 of the MiCOM P546 IED is by default set to false and is only set to true when the subscribed FPGA/IED1/XCBR1$ST$Pos$stVal attribute (circuit breaker status) is ON.

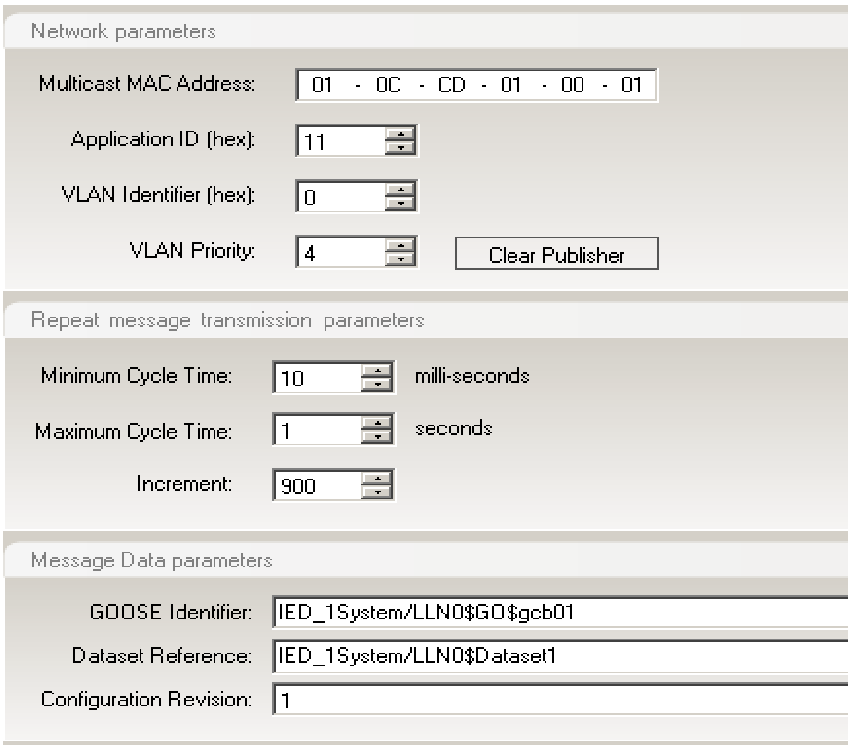
5. Creating a dataset containing the status of LED1 mapped to Virtual Output 1 as shown in the PSL in Figure 5.14. The status of LED1 is mapped to the System/GosGGIO2$ST$Ind1$stVal Boolean data attribute.



Name:	Dataset1
Location:	System\LLN0
Contents:	System/GosGGIO2.ST.Ind1.stVal

Figure 5.16: **Dataset containing the status of the LED1**

6. Configure the MiCOM P546 to publish GOOSE messages containing the dataset configured in step 5.



Network parameters

Multicast MAC Address:	01 - 0C - CD - 01 - 00 - 01
Application ID (hex):	11
VLAN Identifier (hex):	0
VLAN Priority:	4

Repeat message transmission parameters

Minimum Cycle Time:	10 milli-seconds
Maximum Cycle Time:	1 seconds
Increment:	900

Message Data parameters

GOOSE Identifier:	IED_1System/LLN0$GO$gcb01
Dataset Reference:	IED_1System/LLN0$Dataset1
Configuration Revision:	1

7. Downloading the settings to the IED.

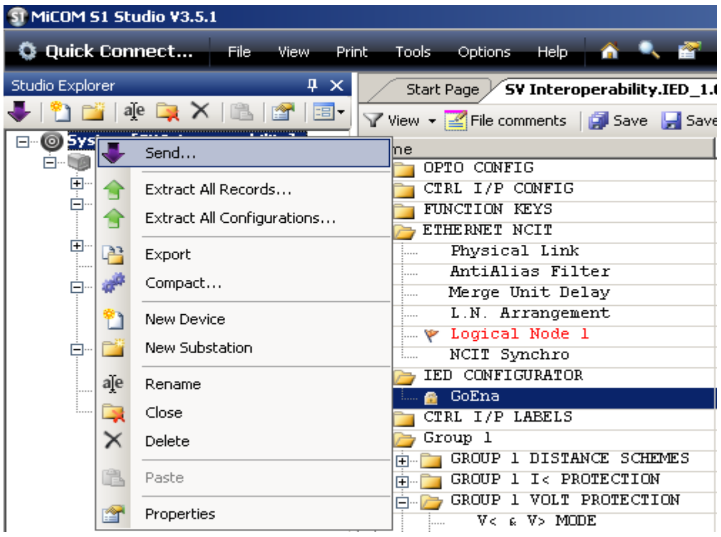


Figure 5.17: Downloading settings to the MiCOM P546 IED

With reference to the PSL and GOOSE message subscription configuration of the MiCOM P546 IED shown in Figure 5.14 and Figure 5.15, the boolean status of Virtual Input 1 is false by default and only true when the subscribed FPGA/IED1/XCBR1$ST$Pos$stVal data attribute is ON. The IED’s LED1 is expected to be ON when Virtual Input 1 is true and OFF otherwise.

Figure 5.18 shows the transition of the status of the MiCOM P546 Virtual Output 1 (System/GosGGIO2$ST$Ind1$stVal) between true (orange blocks) and false states during the interoperability test. In this test, SW2 on the Nexys 3 development board was toggled between ON and OFF positions. In Figure 5.18, the status of Virtual Output 1 is true (orange block) when the position of SW2 (FPGA/IED1/XCBR1$ST$Pos$stVal) is ON.

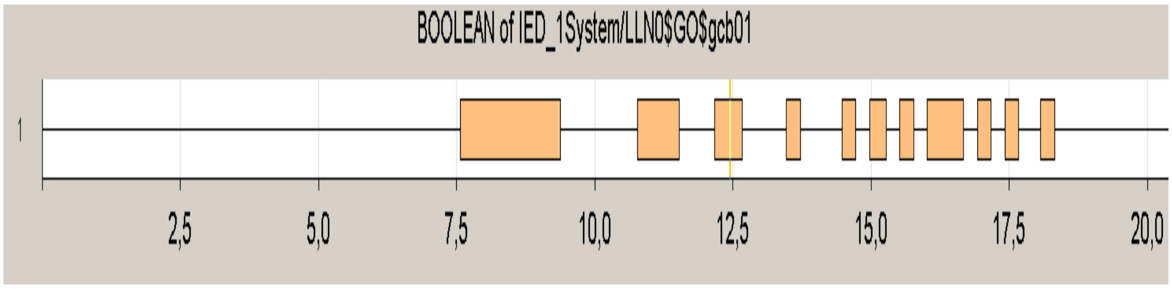


Figure 5.18: Transition of the MiCOM P546 IED’s Virtual Output 1 during the interoperability test

As expected from the PSL configuration of the MiCOM P546 IED, LED1 is ON when the status of the circuit breaker (FPGA/IED1/XCBR1$ST$Pos$stVal) is ON and OFF when the circuit breaker is on the OFF position. This evaluation procedure proves that the developed FPGA-based GOOSE monitoring node publishes GOOSE messages which can be subscribed to by any IEC 61850-compliant IED to implement any automation system.

The next section evaluates the developed limited Merging Unit prototype.

5.4 Limited-function Merging Unit prototype laboratory evaluation

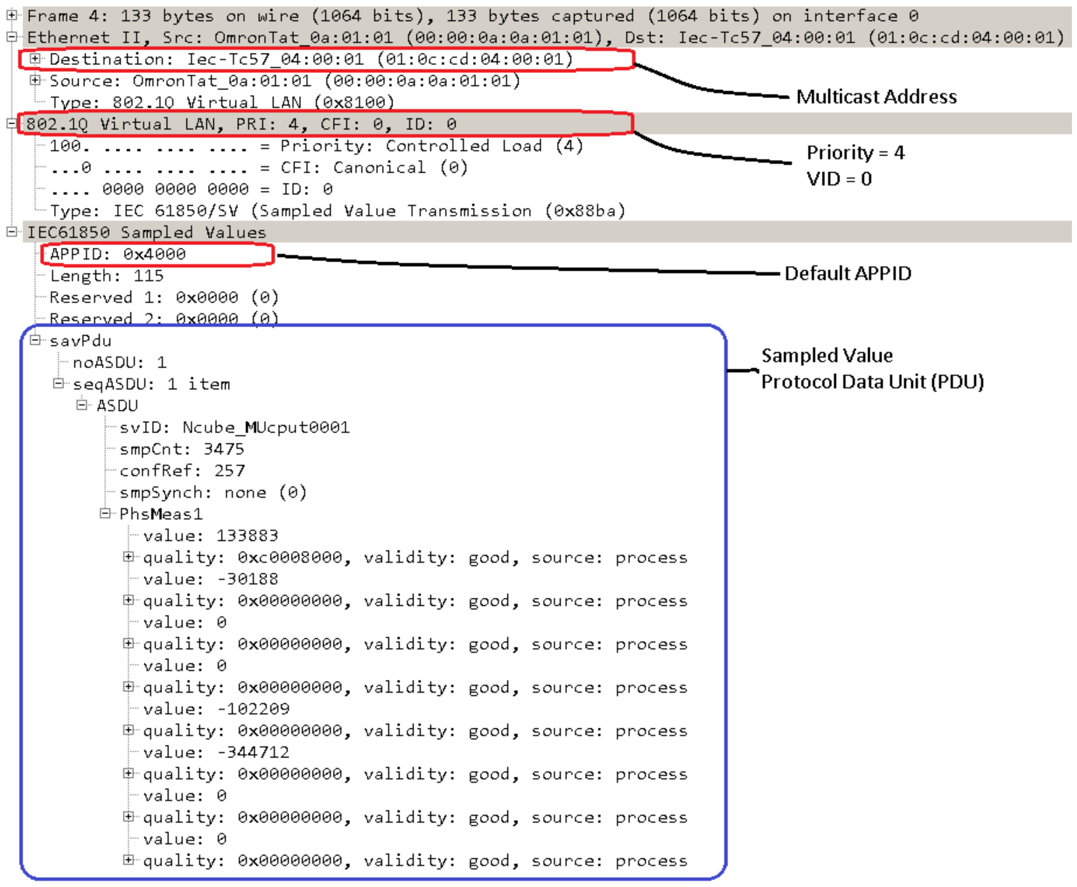
This section provides details of laboratory tests conducted to validate and evaluate the accuracy and performance characteristics of the developed limited-function MU prototype. In section 5.4.1, the structure of SV messages published by the developed MU prototype is compared to that defined in the IEC 61850-9-2LE. The structure of the sampled value messages is captured and analysed using Wireshark and SVScout software applications. Wireshark is a Windows and UNIX systems network analysis software based on the "*pcap*" libraries.

In section 5.4.2 the accuracy of sampled values published by the developed MU prototype is determined by comparing the extracted phasors to those extracted from SV messages published by the Omicron CMC 256*plus* test set. For this test, the CMC 256*plus* test set was configured to generate and inject analogue signals into the developed MU prototype while simultaneously publishing SV messages containing samples of the generated signals.

Test three detailed in section 5.4.3 investigates the accuracy of samples published by the MU prototype when distorted signals are injected. For power system fault/disturbance simulation, the *Harmonics*, *Ramping* and *QuickCMC* modules of the Test Universe package were used to configure the CMC 256*plus* test set. APPENDIX H on page 262 provides a detailed configuration manual for setting up the CMC 256*plus* test set through the Test Universe software. In this section and also including the underlying sub-sections, in some instances, the developed MU prototype will be referred to as the Device under Test (DUT).

5.4.1 Evaluation of Sampled Value message structure

This section validates the structure of sampled value messages published by the developed MU prototype against that defined in IEC 61850-9-2LE. Figure 5.19 shows a Wireshark capture of a sampled value message published by the developed MU prototype when nominal voltage and current signals are injected.

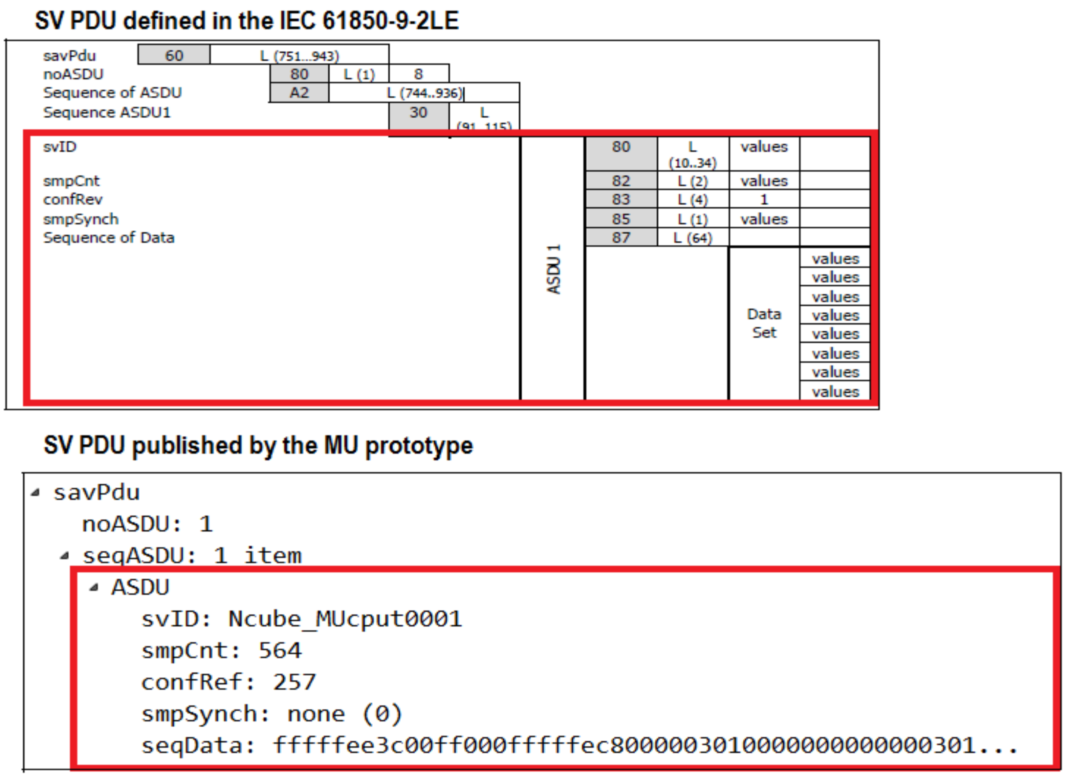


```
⊞ Frame 4: 133 bytes on wire (1064 bits), 133 bytes captured (1064 bits) on interface 0
⊟ Ethernet II, Src: OmronTat_0a:01:01 (00:00:0a:0a:01:01), Dst: Iec-Tc57_04:00:01 (01:0c:cd:04:00:01)
   ⊞ Destination: Iec-Tc57_04:00:01 (01:0c:cd:04:00:01)
   ⊞ Source: OmronTat_0a:01:01 (00:00:0a:0a:01:01)
     Type: 802.1Q Virtual LAN (0x8100)
⊟ 802.1Q Virtual LAN, PRI: 4, CFI: 0, ID: 0
     100. .... .... .... = Priority: Controlled Load (4)
     ...0 .... .... .... = CFI: Canonical (0)
     .... 0000 0000 0000 = ID: 0
     Type: IEC 61850/SV (Sampled Value Transmission (0x88ba)
⊟ IEC61850 Sampled Values
     APPID: 0x4000
     Length: 115
     Reserved 1: 0x0000 (0)
     Reserved 2: 0x0000 (0)
   ⊟ savPdu
        noASDU: 1
      ⊟ seqASDU: 1 item
         ⊟ ASDU
              svID: Ncube_MUcput0001
              smpCnt: 3475
              confRef: 257
              smpSynch: none (0)
            ⊟ PhsMeas1
                 value: 133883
               ⊞ quality: 0xc0008000, validity: good, source: process
                 value: -30188
               ⊞ quality: 0x00000000, validity: good, source: process
                 value: 0
               ⊞ quality: 0x00000000, validity: good, source: process
                 value: 0
               ⊞ quality: 0x00000000, validity: good, source: process
                 value: -102209
               ⊞ quality: 0x00000000, validity: good, source: process
                 value: -344712
               ⊞ quality: 0x00000000, validity: good, source: process
                 value: 0
               ⊞ quality: 0x00000000, validity: good, source: process
                 value: 0
               ⊞ quality: 0x00000000, validity: good, source: process
```

Figure 5.19: **Structure of sampled value message published by the developed MU**

This sampled value message shown in Figure 5.19 consists of two main parts, the Ethernet header and the Protocol Data Unit (PDU) in the blue block. The Ethernet header fields in this message are set to default values as defined in the IEC 61850-9-2LE except for the source MAC address which is manufacturer specific. Referring to Figure 5.19, the destination MAC address is set to the multicast address 01:0C:CD:04:00:01 registered to the IEEE and the VLAN ID is set to default value 0. The captured SV message's APPID field is set to 0x4000 as defined in the IEC 61850-9-2LE.

Figure 5.20 shows the comparison between the captured SV PDU published by the DUT and the one defined in the IEC 61850-9-2LE guideline.



```
◢ savPdu
    noASDU: 1
  ◢ seqASDU: 1 item
    ◢ ASDU
        svID: Ncube_MUcput0001
        smpCnt: 564
        confRef: 257
        smpSynch: none (0)
        seqData: fffffee3c00ff000fffffec8000003010000000000000301...
```

Figure 5.20: **Comparing between SV PDU published by MU prototype to that defined in the IEC 61850-9-2LE**

The protocol data unit shown in Figure 5.20 consists of one ASDU which in turn encapsulates the *svID*, *smpCnt*, confRev *smpSynch* and the dataset (sequence of data) fields. The *smpSynch* flag in the APDU is set to FALSE because the AFE uses an internally generated 1PPS clock instead of using a 1PPS signal from an external GPS clock.

The *seqData* field in Figure 5.20 is the "*PhsMeas1*" dataset which contains the instantaneous current and voltage samples of the analogue signals injected by the CMC 256*plus* test set. With reference to Figure 5.20, it can be noted that the structure of the SV messages published by the developed MU prototype matches that defined in the IEC 61850-9-2LE guideline.

The following section assesses the accuracy of MU prototype's sample calculation VHDL module by comparing the root mean square values extracted from the sampled value messages published by the developed MU prototype to those published by the CMC 256*plus* test set.

5.4.2 Sampled Value accuracy evaluation

Merging Units (MUs) are important devices used IEC 61850 process bus-based protection schemes. These MUs should be tested to ensure that they publish accurate current and voltage samples under nominal and abnormal/fault conditions.

For this test procedure, the CMC 256*plus* test set was configured to generate current and voltage signals and also publish sampled value messages containing samples of the generated signals. These analogue signals were then injected into the developed MU prototype. The sampled value messages published by the CMC 256*plus* and the MU prototype were captured using SVScout and the exported to a COMTRADE file. The magnitudes of the voltage and current signal generated by the CMC 256*plus* device are equal to 33 V_{RMS} and 1 A_{RMS} equating to primary line to neutral magnitudes of 3.3 kV_{RMS} and 100 A_{RMS} respectively.

The sampled value messages published by the developed MU prototype and the Omicron CMC 256*plus* test set were captured using SVScout. SVScout subscribes to the sampled value message streams published by both devices (CMC 256*plus* test set and MU prototype) and plots the voltage and current samples in an oscilloscope view and extracts the phasors for the different phases (OMICRON, 2014).

Figure 5.21 shows an SVScout sinusoidal plot of phase A and B voltage and current sampled values published by the developed MU prototype under nominal conditions (3.3 kV_{RMS} and 100 A_{RMS}).

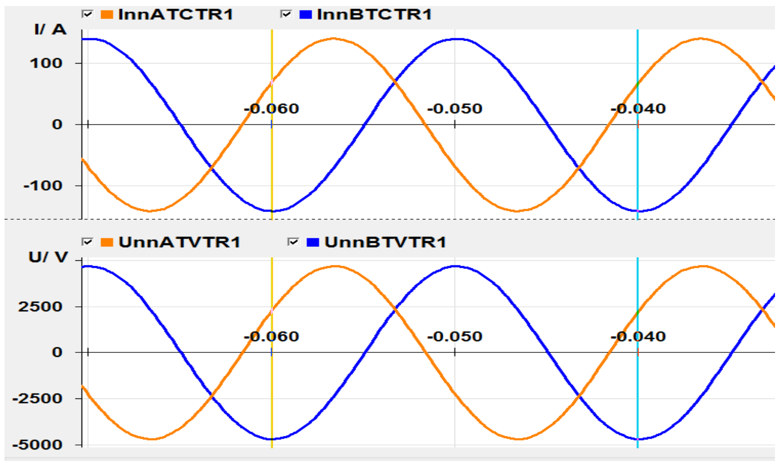
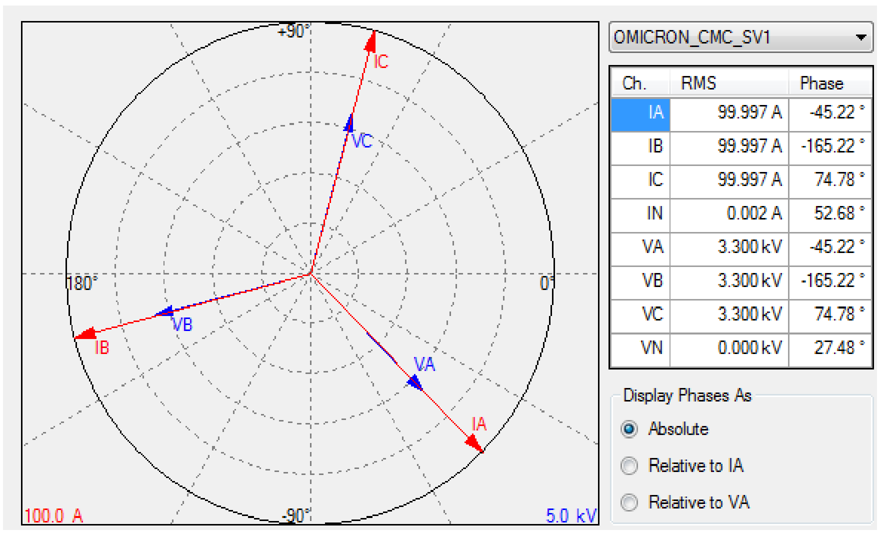


Figure 5.21: **SV messages published by the developed MU prototype**

Figure 5.22 and Figure 5.23 shows the phasors of sampled values published by the CMC 256*plus* test set and the developed MU prototype respectively.



OMICRON_CMC_SV1

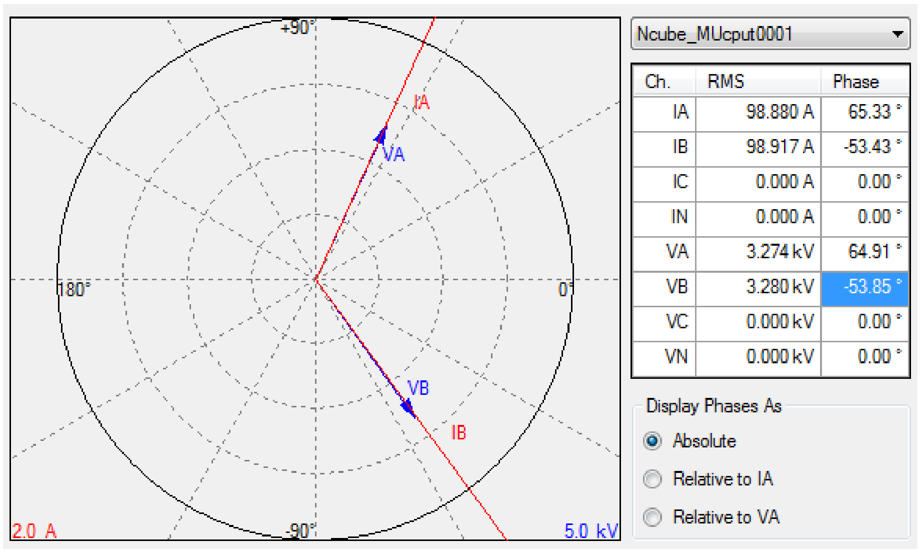
Ch.	RMS	Phase
IA	99.997 A	-45.22 °
IB	99.997 A	-165.22 °
IC	99.997 A	74.78 °
IN	0.002 A	52.68 °
VA	3.300 kV	-45.22 °
VB	3.300 kV	-165.22 °
VC	3.300 kV	74.78 °
VN	0.000 kV	27.48 °

Display Phases As

[✓ Absolute]
Relative to IA
Relative to VA

Figure 5.22: **Phasors of sampled values published by the CMC 256*plus* test set**

With reference to Figure 5.22 above, the line to neutral voltage and current samples published by the CMC 256*plus* test set for phases A and B are equal to 3.3 kV$_{RMS}$∠-45.22º, 3.3 kV$_{RMS}$∠-165.22º and 99.997 A$_{RMS}$∠-45.22º, 99.997 A$_{RMS}$∠-165.22º respectively. The angle between phase A and B phasors is equal to 120º as expected.



Ncube_MUcput0001

Ch.	RMS	Phase
IA	98.880 A	65.33 °
IB	98.917 A	-53.43 °
IC	0.000 A	0.00 °
IN	0.000 A	0.00 °
VA	3.274 kV	64.91 °
VB	3.280 kV	-53.85 °
VC	0.000 kV	0.00 °
VN	0.000 kV	0.00 °

Display Phases As

[✓ Absolute]
Relative to IA
Relative to VA

Figure 5.23: **Phasors of sampled values published by the developed MU prototype**

Figure 5.23 shows phasors of sampled values published by the developed MU prototype; the phase to neutral voltage and current samples for phases A and B are equal to 3.274 kV$_{RMS}$∠-64.91º, 3.28 kV$_{RMS}$∠-53.81º and 98.880 A$_{RMS}$∠-65.33º, 98.917 A$_{RMS}$∠-53.43º respectively. The angles between the phase A and B samples of the injected voltages and currents is equal to 118.72º and 118.76º and respectively.

Therefore from Figure 5.22 and Figure 5.23, the voltage and current magnitude percentage error of sampled values published by the developed MU prototype is 0.61% and 1.11% respectively. The percentage phase angle error between the A and B phases for the voltage and current sampled values is 1.07% and 1.03% and respectively. The developed MU prototype's voltage and current magnitude measurement percentage error is reasonable since most IEC 61850 standard-compliant IEDs have a typical percentage error of ±1% (Alstom, 2013; Alstom, 2014; Schneider Electric, 2010). These results show that the developed Merging Unit prototype can publish accurate samples of the injected analogue voltages and current under nominal conditions.

The next section assesses the response and sampled value accuracy of the MU prototype when subjected to laboratory simulated power system disturbances/signal distortions.

5.4.3 Merging Unit prototype response evaluation

The electrical power system is prone to disturbances which affect the quality of the supplied electricity. These disturbances are caused by a variety of factors including weather conditions (lightning strikes, strong winds), equipment failure, non-linear loads, over-loaded wiring and large load swing (We Energies, 2015), (D & B Power Associates, 2015), (Seymour, 2011).

According to Seymour, (2011), the seven most common disturbances experienced in power systems are transients, interruptions, sag/under-voltage, swell/over-voltage, waveform distortion, voltage fluctuations and frequency variations. The Merging Unit being the primary measuring device in an IEC 61850 standard process bus-based protection scheme must tested in order to ascertain its sample accuracy and response to distorted current and voltage injections.

For the following simulations, the CMC 256*plus* test set was configured through the Test Universe modules to inject distorted current and voltage signals into the developed MU prototype emulating four types of disturbances. The response and sample accuracy of the developed MU prototype was evaluated for the voltage-sag, voltage-swell, waveform distortions and frequency variation disturbances. For waveform distortion, the *Harmonics* module of the Test Universe package was used for generating voltage and current signals superimposed with harmonic components.

The analogue signals generated by the CMC 256*plus* test set were compared to the sampled values published by the developed MU prototype. The MU publishes primary values calculated from the secondary voltage and current signals injected by the CMC 256*plus* test set using the 100:1 (primary to secondary) ratio. For disambiguation purposes, graphs of voltage and current signals generated by the CMC 256*plus* test set will be quoted in primary values, that is, actual CMC 256*plus* analogue output factored by 100. The following sections, (section 5.4.3.1 to section 5.4.3.4) detail and discuss the disturbance simulations and the results thereof.

5.4.3.1 Voltage-sag fault performance evaluation

A voltage-sag is a short duration (half cycle to 3 seconds) decrease in system voltage, these voltage-sag faults are usually caused by system faults e.g., phase to ground faults or switching on of equipment with heavy start-up currents (Seymour, 2011). Voltage-sag faults can cause electronic equipment damage or data loss.

Figure 5.24 shows an illustration of a voltage-sag fault.

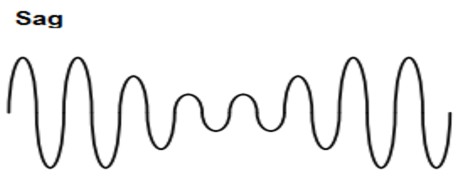


Figure 5.24: **Illustration of a voltage-sag fault on a power system (Adapted from Seymour, 2011)**

For simulating a voltage-sag disturbance, the CMC 256*plus* test set was configured using the *Ramping* module of the Test Universe software. The test set generated voltage and current signals simulating a voltage-sag fault by appending five ramp signals as shown in Figure 5.25. In this simulation, the analogue voltage and currents outputs of the CMC 256*plus* test set begins at 33 V_{RMS} and 1 A_{RMS} before dipping to 16.5 V_{RMS} and 0.5 A_{RMS}. Time graphs (a) and (b) in Figure 5.25 shows the voltage and current signals injected into the MU prototype by the CMC 256*plus* quoted as primary values.

Phase B voltage and current magnitudes remain steady at 33 V_{RMS} (3.3 kV_{RMS}) during this voltage-sag fault simulation as shown in Figure 5.25.

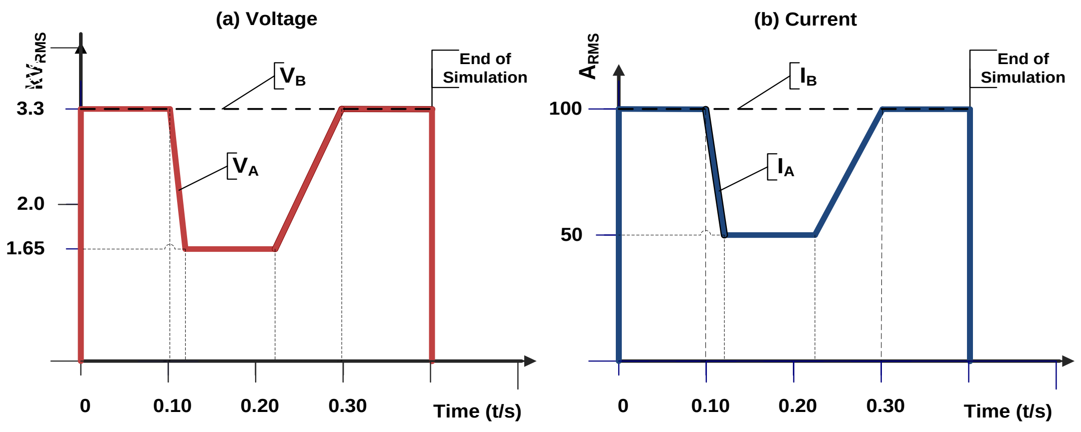


Figure 5.25: **Voltage-sag simulation using the CMC 256*plus* test set and the Test Universe**

Figure 5.26 shows a graphical representation of current and voltage root mean square values extracted from sampled value messages published by the developed MU prototype during this voltage-sag fault simulation.

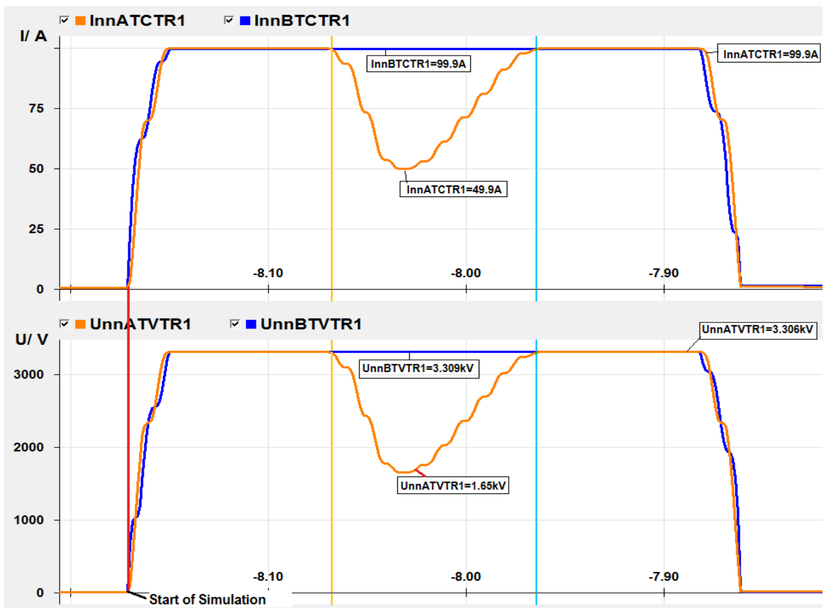


Figure 5.26: **Current and voltage RMS values measured by the MU prototype during a voltage-sag simulation**

With reference to Figure 5.26, the pre-fault and post-fault line to neutral voltages and currents measured on phases A and B are approximately 3.306 kV_{RMS} and 99.9 A_{RMS} as expected from the injection shown in Figure 5.25. In Figure 5.26, 1.65 kV_{RMS} and 49.9 A_{RMS} phase A voltage and currents are measured by the developed MU prototype during the voltage-sag simulation as expected from the generated signals illustrated in Figure 5.25.

Figure 5.27 shows waveforms of current and voltage signals extracted from sampled value messages published by the developed MU prototype during the voltage-sag injection test.

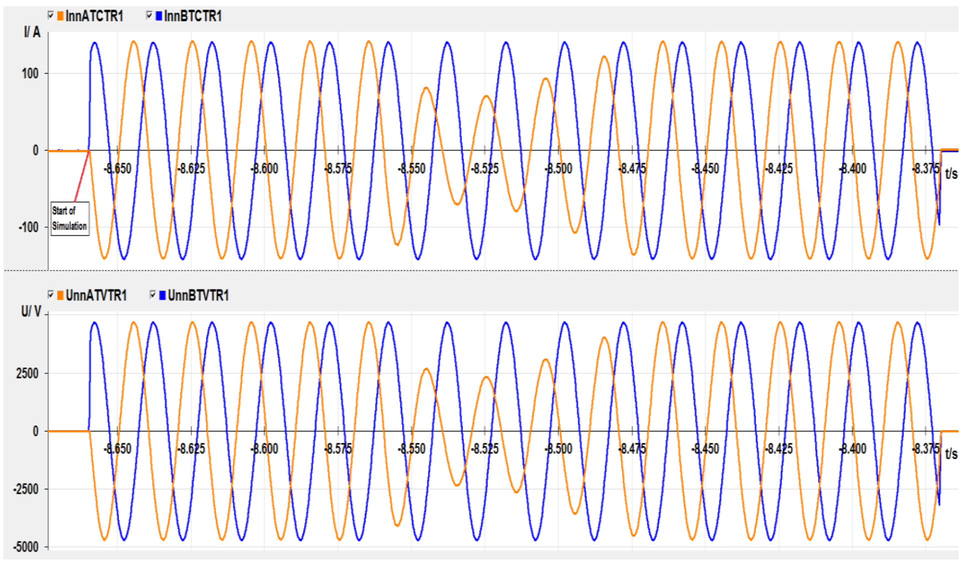


Figure 5.27: Waveforms extracted from SV messages published by the MU prototype during voltage-sag injection test

The voltage and current magnitude pattern of the injected signals shown in Figure 5.25 and that of sampled values published by the MU prototype shown in Figure 5.26 match. These two graphs (Figure 5.25 and Figure 5.26) prove that the developed MU prototype can be used in a real power system and can produce accurate measurement even under voltage-sag conditions.

5.4.3.2 Voltage-swell fault performance evaluation

A voltage-swell disturbance is characterised by a voltage increase from the nominal value for a short period usually half a cycle to 3 seconds. Voltage-swells are caused by sudden disconnection of large loads or phase-to-phase faults in three phase systems. This disturbance may cause electronic component damage, degradation of insulation and electrical contacts. Voltage swells are common in power grids with incorrectly configured tap changer transformer which causes the voltage to swell whenever a load is disconnected (We Energies, 2015).

Figure 5.28 shows as illustration of a voltage-swell fault in power systems.

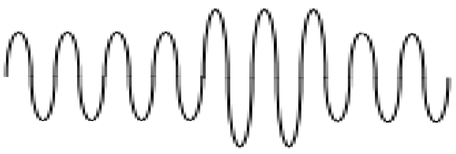


Figure 5.28: Illustration of a voltage-swell disturbance (Adapted from Seymour, 2011)

The CMC 256*plus* test set was configured using the Test Universe *Ramping* module to generate voltage and current signal emulating a voltage-swell condition. Using the *Ramping* module, the CMC 256*plus* test set's phase A analogue output voltage and current signals were ramped from nominal values of 33 V_{RMS} and 1 A_{RMS} to peak at 49.5 V_{RMS} and 1.5 A_{RMS} respectively. Time graphs (a) and (b) in Figure 5.29 shows the voltage and current signals injected into the MU prototype by the CMC 256*plus* quoted as primary values.

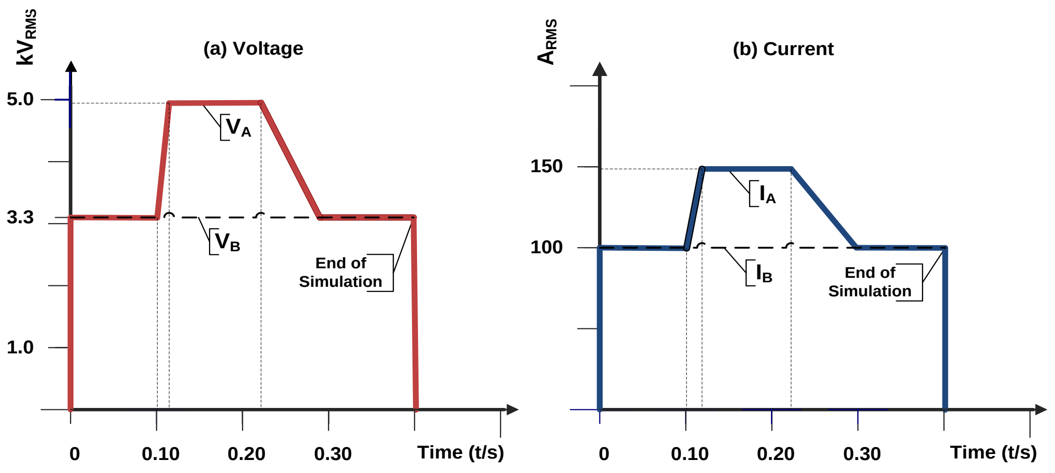


Figure 5.29: Voltage-swell simulation using the CMC 256*plus* test set and the Omicron Test Universe

The phase B voltage and current magnitudes remain steady at 33 V_{RMS} (3.3 kV_{RMS}) as shown in Figure 5.29. The injected signals are sampled by the MU prototype and published to the process bus; the oscillographic information of the sampled values was exported into a COMTRADE file using SVScout.

Figure 5.30 shows a graphical representation of current and voltage root mean square values published by the MU prototype during the voltage-swell simulation.

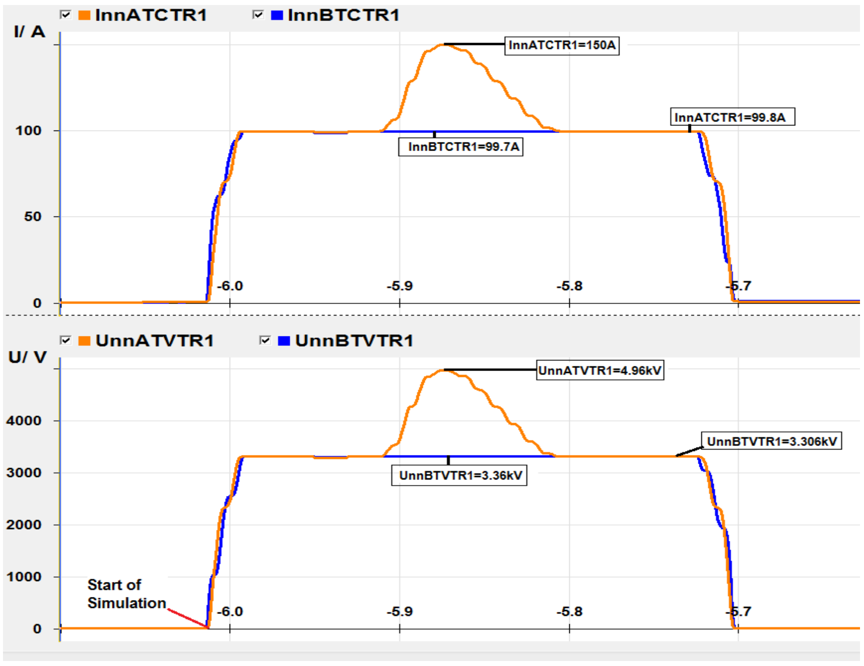


Figure 5.30: **Voltage and current RMS values measured by the MU prototype during a voltage-swell simulation**

Figure 5.31 shows a waveform of sampled values published by the developed MU prototype during a voltage-swell injection test.

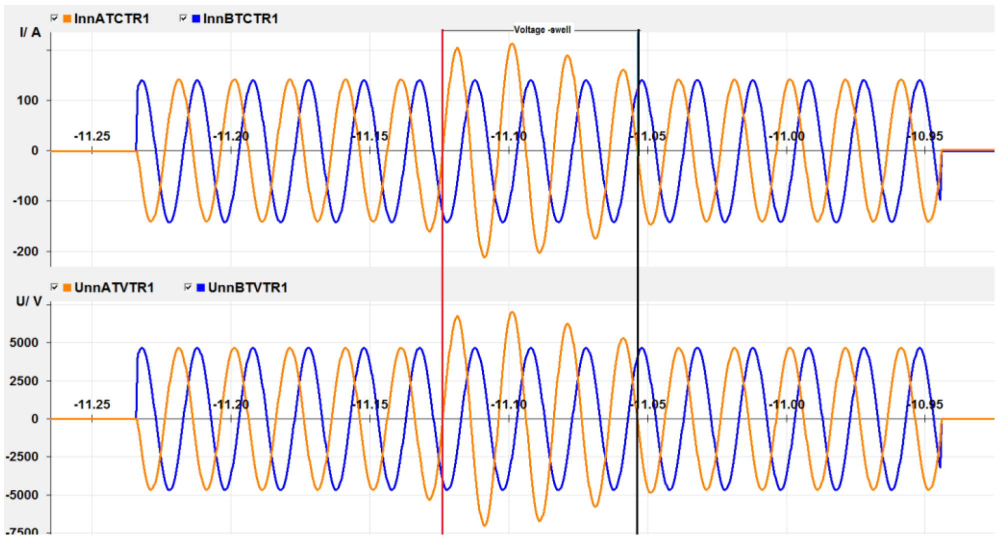


Figure 5.31: Voltage and current sampled values published by the MU prototype during a voltage-swell injection test

With reference to Figure 5.30 and Figure 5.31, the phase A and B voltage and current samples published by the developed MU prototype begin at nominal values of approximately 3.306 kV$_{RMS}$ and 99.8 A$_{RMS}$. During the voltage-swell simulation, phase A voltage and current samples rise to peak at 4.96 kV$_{RMS}$ and 150 A$_{RMS}$. Comparing the injected signals (Figure 5.29) and the published sampled values (Figure 5.30 and Figure 5.31) it can be seen that the magnitude profile of the published sampled values and that of the generated signals match. The peak voltage and current magnitudes published by the developed MU prototype are 150 A$_{RMS}$ and 4.96 kV$_{RMS}$ as expected from the injected signals. This section proves that the developed MU prototype is capable of publishing accurate sampled value even under voltage-swell faults.

The next section discusses the sample accuracy evaluation and the results thereof of the MU prototype power system frequency variation test.

5.4.3.3 Frequency variations tests

In electrical power networks, the measured frequency deviates from the rated system frequency due to in-balances between the generated power and the load. As such, the frequency measurement accuracy and range of the developed MU prototype was evaluated using the CMC 256*plus* test set. For this test, the magnitude of the generated voltages and currents were set to the nominal values equal to 33 V_{RMS} and 1 A_{RMS} respectively while the frequency was varied between 10 Hz, 50 Hz and 100 Hz.

In this test, the current and voltages signal generated by the CMC 256*plus* test set were sampled and published by the MU prototype. Similar to the previous simulations, SVScout was used for capturing the sampled value messages and exporting the oscillographic information to a COMTRADE file for further analysis. The power system variation tests were conducted by injecting signals generated at 10 Hz, 50 Hz (nominal) and 100 Hz as shown in Figure 5.32 to Figure 5.34.

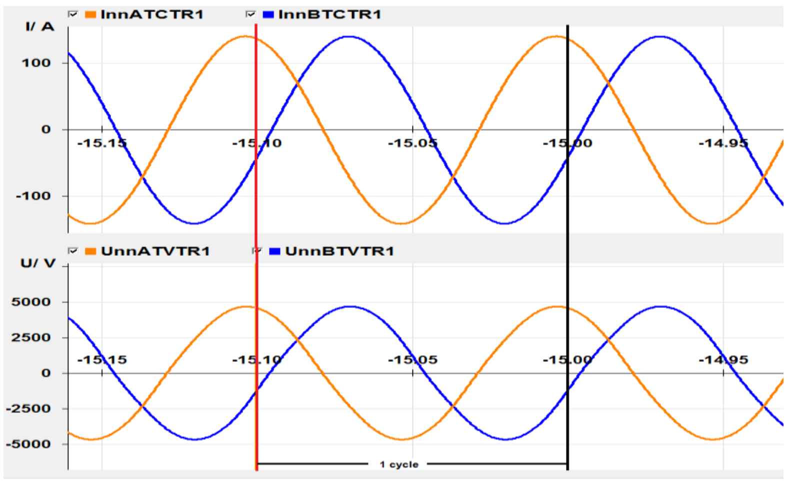


Figure 5.32: **Sampled value messages at 10 Hz system frequency**

Figure 5.32 above shows voltage and current waveforms extracted from sampled value messages published by the MU prototype when the CMC 256*plus* test set was configured to generate signals at 10 Hz system frequency. The waveforms displayed in Figure 5.32 are sinusoidal with a period of 0.1 s corresponding to a frequency of 10 Hz as expected.

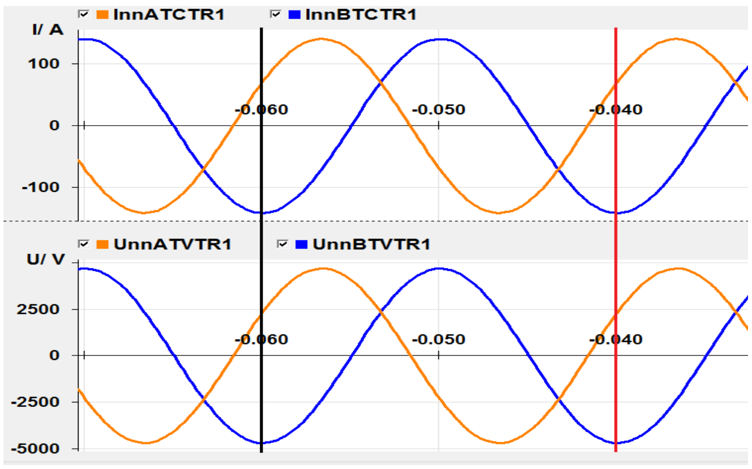


Figure 5.33: **Sampled values published at 50 Hz (nominal) system frequency**

Figure 5.33 above shows voltage and current waveforms extracted from sampled value messages published by the MU prototype when the CMC 256*plus* test set was configured to generate signals with a 50 Hz system frequency. The period of the sinusoidal waveforms in Figure 5.33 is 0.02 seconds corresponding to a frequency of 50 Hz.

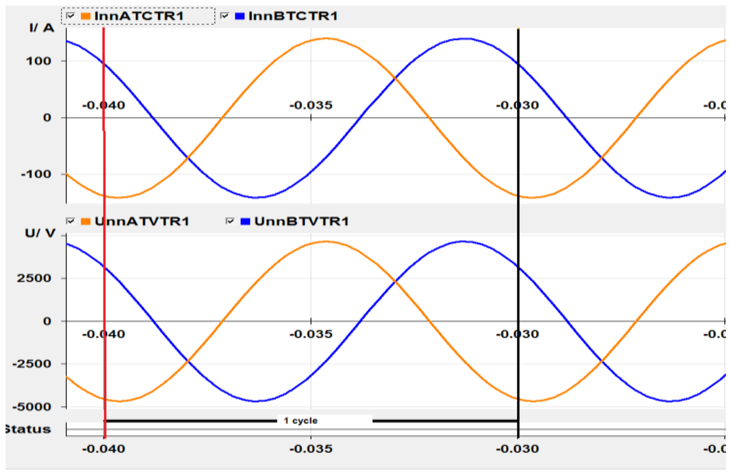


Figure 5.34: **Sampled value messages published at 100 Hz system frequency**

Figure 5.34 shows voltage and current waveforms extracted from sampled value messages published by the MU for an injection test at 100 Hz system frequency. With reference to Figure 5.34, the captured waveform is sinusoidal with one cycle spanning over 0.01 s as expected for a system frequency of 100 Hz.

In Figure 5.35, two plot areas (a) and (b) are shown which contain sampled value line graphs of current and voltage injections, respectively, generated at 10 Hz, 50 Hz and 100 Hz by the CMC 256*plus* test set. Chart (a) in Figure 5.35 shows sampled values line graphs of phase A current signals generated at 10 Hz, 50 Hz and 100 Hz represented by the orange, red and green plots respectively. Similarly to chart (a), chart (b) in the same figure (Figure 5.35) shows the sampled value line graphs of phase A voltage signals generated at 10 Hz, 50 Hz and 100 Hz represented by the orange, red and green plots respectively.

Figure 5.35 shows the relationship between signals generated at different frequencies with the same magnitude.

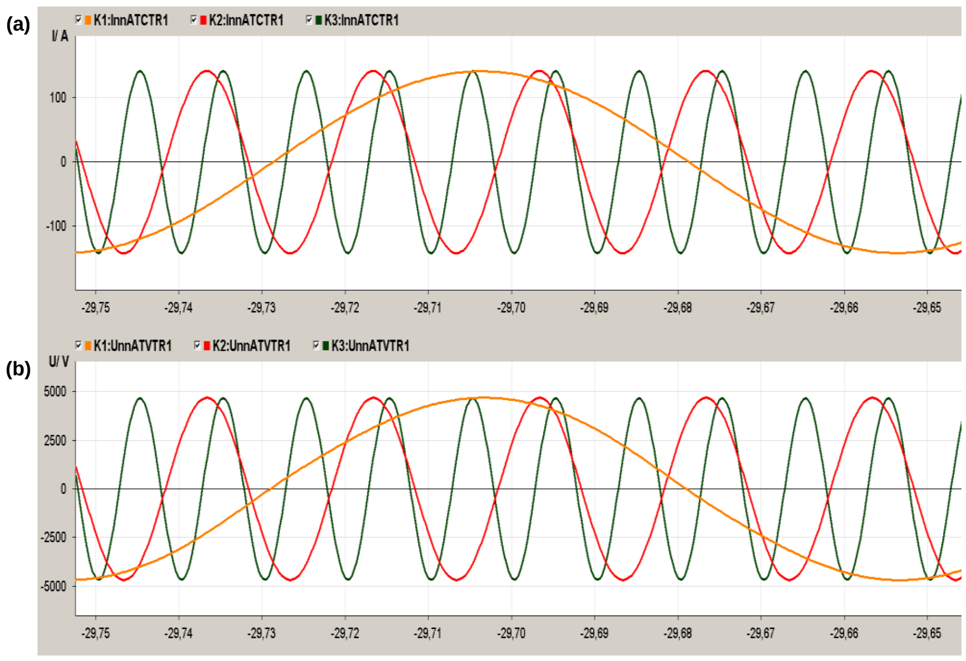


Figure 5.35: **Sampled value plot of voltage and current signals generated at different frequencies**

With reference to Figure 5.32 to Figure 5.34, the developed MU prototype has a wide dynamic input frequency range as witnessed by its ability to accurately measure the injected analogue signals with varying system frequencies. The frequencies calculated from the sampled value plots correspond to those of the analogue signals injected by the CMC 256*plus* test set.

5.4.3.4 Evaluation of sample accuracy during Harmonic distortions

This section of the research project assesses the effects of harmonic distortion on the accuracy of the sampled values published by the developed MU prototype. Harmonics in alternating current signals are sinusoidal components whose frequencies are multiples of the fundamental frequency (Csanyi, 2015). Harmonics are caused by non-linear loads on the power grid drawing non-sinusoidal current from a voltage source.

The resulting signal is a sum of a number of super-imposed harmonics as illustrated in Figure 5.36.

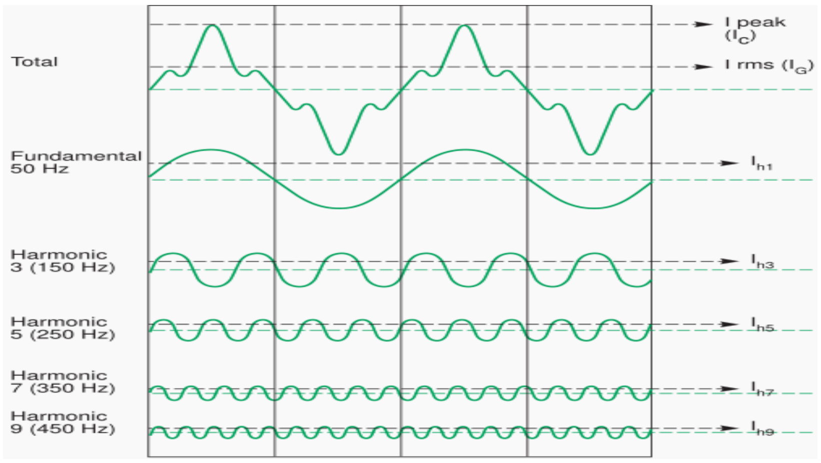


Figure 5.36: **Harmonics in an electrical signal (Adapted from Csanyi, 2015)**

Harmonic disturbances can cause heating of transformer and capacitor banks and in some cases cause mal-operation of electronic equipment (Ellis, 2001). The CMC 256*plus* was configured to generate voltage and current signals containing 2nd and 3rd order harmonic components. The fundamental voltage and current signals generated by the CMC 256*plus* test set were set to 33 V_{RMS} and 1 A_{RMS} respectively. The 2nd and 3rd order harmonic components have a magnitude of 30% and 15% of the fundamental signal respectively and are in phase with their respective fundamental signals. The magnitude and phase values of the 2nd and 3rd order harmonic components are arbitrarily chosen for evaluation purposes.

The *Harmonics* module setup is shown in Table 5.1.

Table 5.1: **2nd and 3rd order harmonic component simulation setup using the *Harmonics* Module**

	VL1-E, IL1		VL2-E, IL2		VL3-E, IL3	
Order	**Mag**	**Phase**	**Mag**	**Phase**	**Mag**	**Phase**
2	30%	0º	30%	-120º	30%	120º
3	15%	0º	15%	-120º	15%	120º

Chart (a) and (b) in Figure 5.37 shows RMS current and voltage magnitude-time graphs respectively. These graphs were exported from the *Harmonics* module of the Test Universe software into a COMTRADE file. The orange and blue line graphs in charts (a) and (b) represents sampled values of phases A and B current and voltage injections respectively. In Figure 5.37 and Figure 5.38, the current and voltage magnitudes of analogue signals generated by the CMC 256*plus* test set are quoted as primary values.

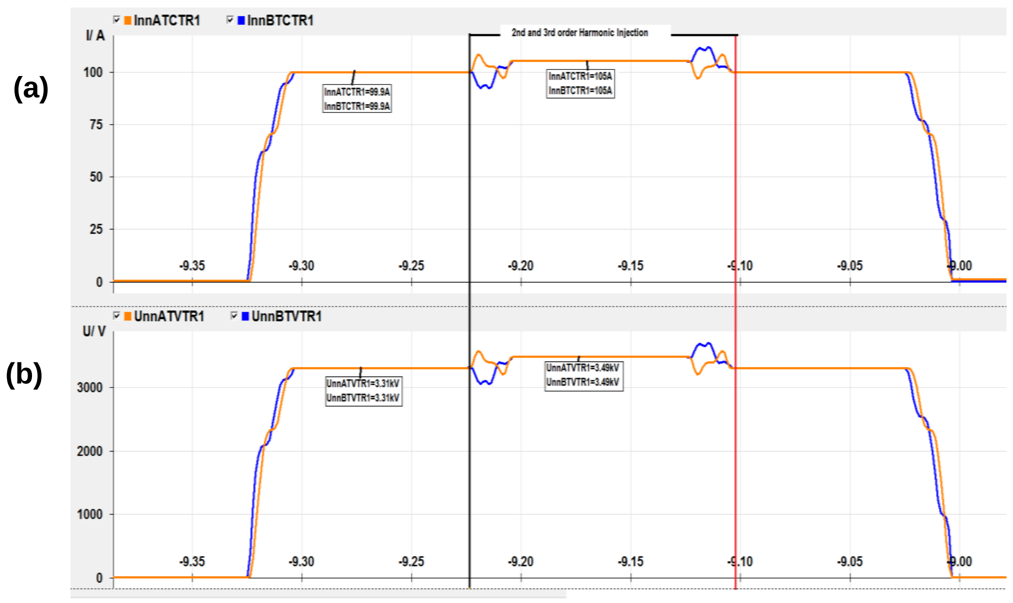


Figure 5.37: **Voltage and current RMS values of signals generated by the CMC 256*plus* test set during harmonic disturbance injection test**

Chart (a) and (b) in Figure 5.38 shows amplitude-time graphs of current and voltage analogue signals generated by the CMC 256*plus* test set during the harmonic disturbance injection respectively.

(a)

(b)

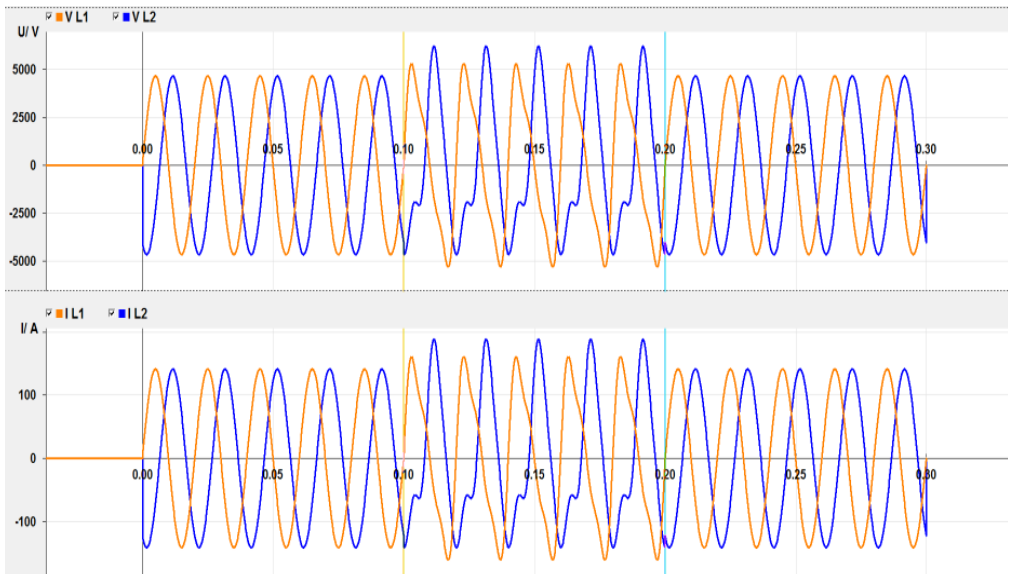


Figure 5.38: **Harmonic distorted signals generated by the CMC 256*plus* test set**

The SV messages published by the developed MU prototype were captured using SVScout and the oscillographic data exported to a COMTRADE file. Chart (a) and (b) in Figure 5.39 shows RMS magnitude-time graphs of currents and voltages extracted from sampled values published by the MU prototype during the harmonic distortion simulation respectively.

In this Figure 5.39, the voltages and current RMS magnitude-time plots for phases A and B are represented by the orange and blue line charts respectively.

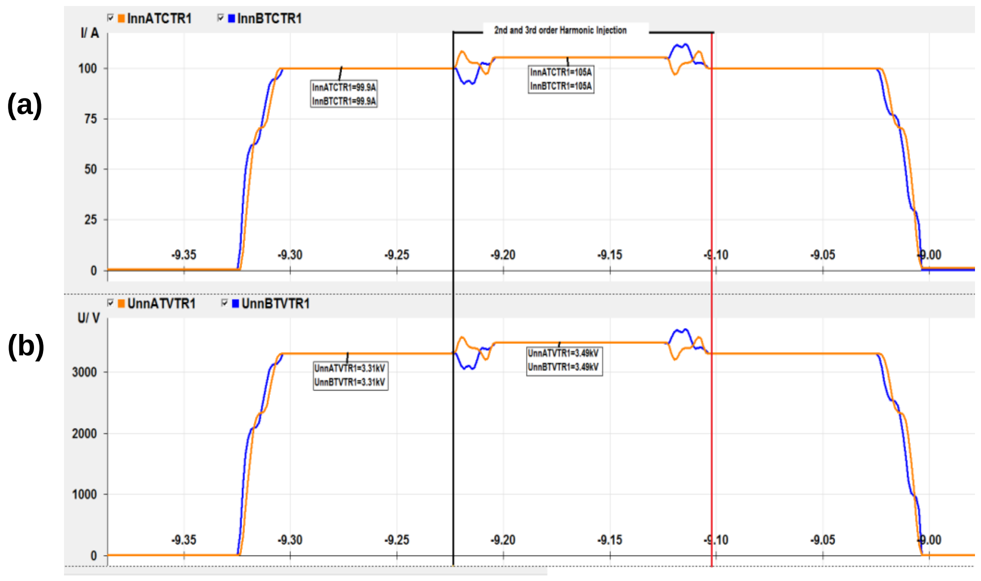


Figure 5.39: **RMS values extracted from sampled values published by the MU prototype during the Harmonic injection test**

Chart (a) and (b) in Figure 5.40 shows the voltage and current magnitude-time waveforms extracted from SV messages published by the MU prototype during the harmonic distortion simulation.

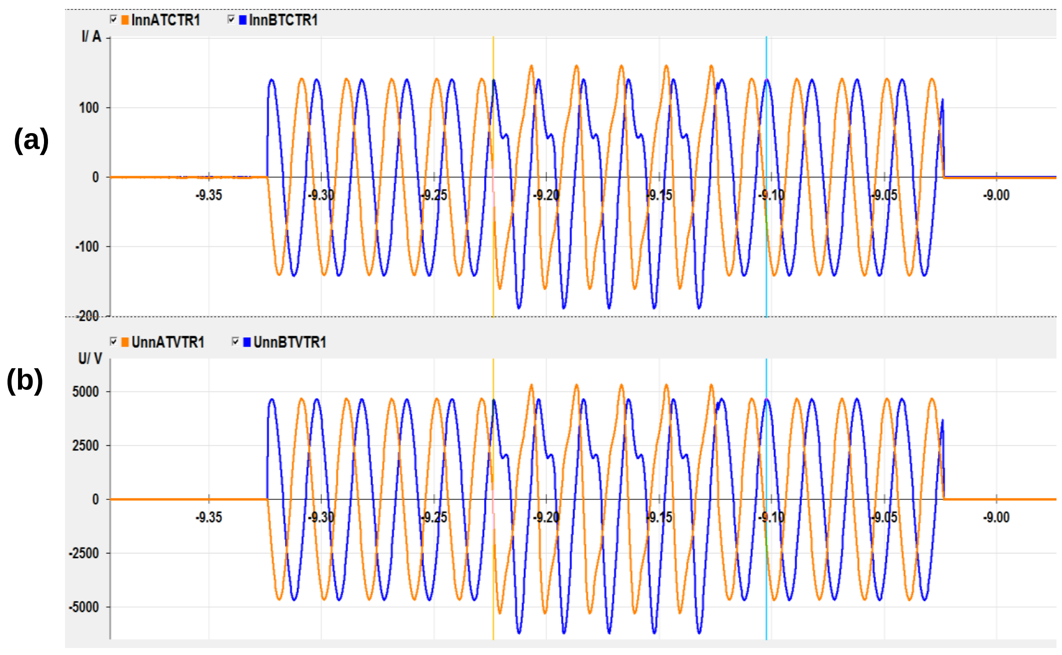


Figure 5.40: **Harmonic distortion in published sampled value messages**

By comparison, the RMS voltage and current magnitudes published by the developed MU prototype shown in Figure 5.39 are equal to the RMS magnitudes of analogue signals generated by the CMC 256*plus* test set shown Figure 5.37. The magnitude-time waveforms of signals generated by the CMC 256*plus* test set and those published by the MU prototype shown in Figure 5.38 and Figure 5.40, respectively, are 180˚ out of phase with each other, this is due to the phase shift introduced by the step-down transformers of the AFE.

This test proves shows that the developed MU prototype is capable of publishing accurate samples analogue voltage and current signals during harmonic disturbances in an electrical power system. The next section presents an interoperability test of the developed MU prototype with an IEC 61850-9-2LE-compliant IED.

5.4.4 Merging Unit prototype interoperability test

IEC 61850-9-2LE-compliant IEDs must be able to subscribe to the SV messages published by the developed MU prototype and therefore an interoperability test was conducted between this the MU prototype and the MiCOM P444 IED. In this interoperability evaluation, the CMC 256*plus* test set was configured to generate and inject nominal voltage and current signals into the developed MU prototype. The sampled value messages published by the MU prototype were then subscribed to by the MiCOM P444 IED.

Figure 5.41 shows equipment setup in the laboratory for evaluating the developed MU prototype's interoperability with the MiCOM P444 IED.

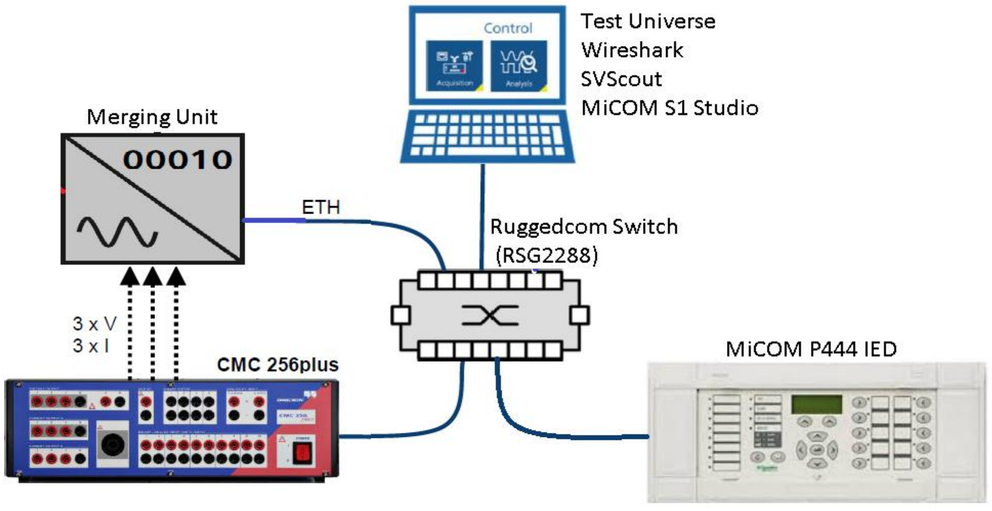


Figure 5.41: FPGA-based Merging Unit prototype interoperability test

The MiCOM P444 IED was configured to subscribe to and extract voltage and current measurements from the sampled value messages published by the FPGA-based MU prototype. These measurements extracted by the MiCOM IED were compared to those extracted by SVScout shown in Figure 5.23 in section 5.4.2 on page 150.

The MiCOM P444 IED is configured as follows:

1. Create a new system in MiCOM S1 Studio.

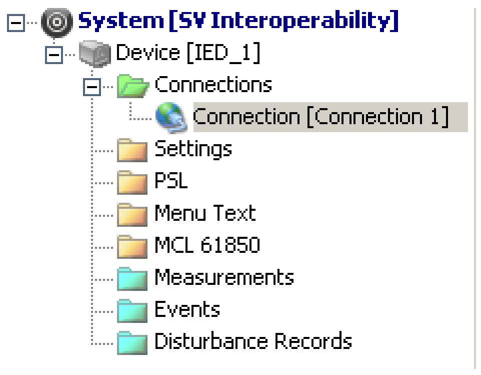
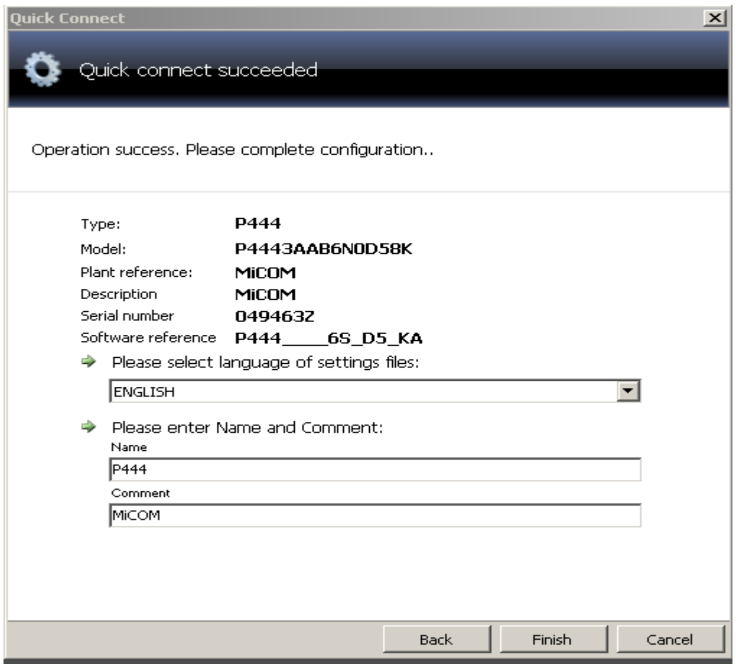


Figure 5.42: Creating a new system in MiCOM S1 Studio

2. Creating devices in the system using the **Quick Connect** command which connects to an IED attached to the PC through a serial RS-232 cable. This **Quick Connect** command retrieves the IED type, model, serial number and software reference numbers as shown in Figure 5.43.



Operation success. Please complete configuration..

Type:	P444
Model:	P4443AAB6N0D58K
Plant reference:	MiCOM
Description	MiCOM
Serial number	0494632
Software reference	P444____6S_D5_KA

Please select language of settings files: ENGLISH

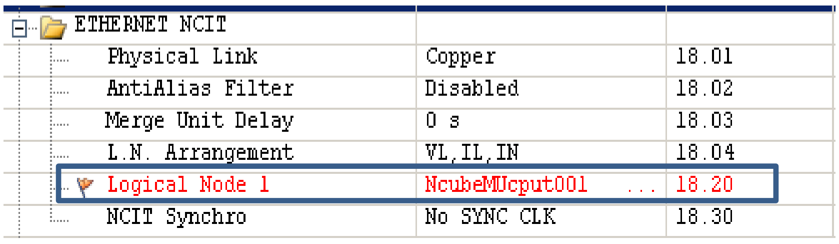
Please enter Name and Comment:

Name: P444

Comment: MiCOM

Figure 5.43: **Connecting to the MiCOM P444 IED using the *Quick Connect* command**

3. Configuring the MiCOM P444 IED to subscribe to SV messages published by the developed MU prototype. This is achieved by setting the NCIT logical node value of the MiCOM IED to the *svID* (sampled value ID) of sampled value messages published by the MU prototype. This configuration is shown in Figure 5.44



ETHERNET NCIT		
Physical Link	Copper	18.01
AntiAlias Filter	Disabled	18.02
Merge Unit Delay	0 s	18.03
L.N. Arrangement	VL,IL,IN	18.04
Logical Node 1	NcubeMUcput001 ...	18.20
NCIT Synchro	No SYNC CLK	18.30

Figure 5.44: **Configuration of the NCIT settings for SV subscription**

4. Downloading the settings to the IED as shown in Figure 5.45.

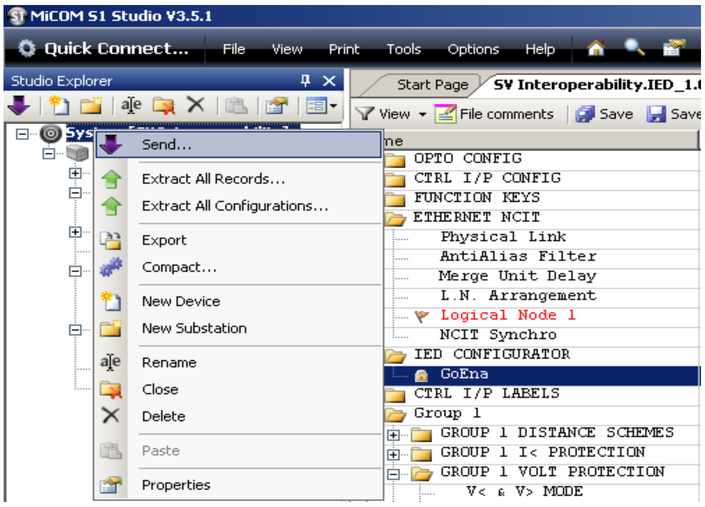
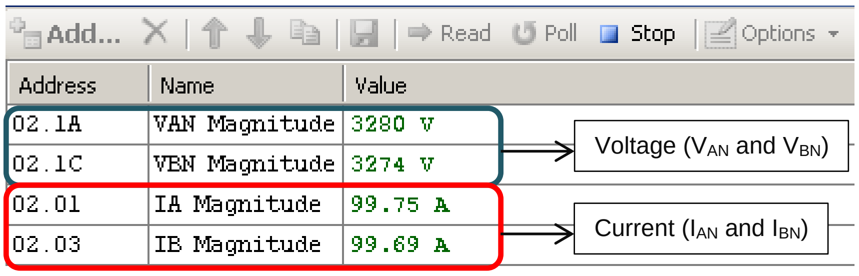


Figure 5.45: Downloading settings to the MiCOM P444 IED

Figure 5.46 shows phases A and B voltage and current magnitude values extracted by the MiCOM P444 IED from the SV messages published by the developed MU prototype.


Add...
Read
Poll
Stop
Options
Voltage (V_{AN} and V_{BN})
Current (I_{AN} and I_{BN})

Address	Name	Value
02.1A	VAN Magnitude	3280 V
02.1C	VBN Magnitude	3274 V
02.01	IA Magnitude	99.75 A
02.03	IB Magnitude	99.69 A

Figure 5.46: Phase A and B voltage and current magnitude values extracted from SV messages published by the DUT

The phase A and B for voltage and current magnitude values shown in Figure 5.46 are equal to the magnitude values for the same phases shown published by the DUT in Figure 5.22.

This evaluation procedure proves that the current and voltage measurements published by the developed MU prototype can be used by IEC 61850-compliant IEDs for implementing protection schemes. The results attained from this experiment prove that the developed MU prototype is interoperable with different IEDs from different vendors according to the IEC 61850 standard.

The following section concludes this chapter.

5.5 Conclusion

Chapter Five has presented in section 5.3 and section 5.4 test procedures conducted in the laboratory for evaluating the developed FPGA-based GOOSE monitoring node and limited-function MU prototypes respectively. In these two sections, the results obtained from the tests conducted were presented and discussed to show that the developed GOOSE and SV message-mapping VHDL modules are compliant with the referenced IEC 61850 standards.

Section 5.3 presents simulations and test conducted in the laboratory to evaluate the FPGA-based GOOSE monitoring node. This section started off by validating the structure of the GOOSE messages published by the developed GOOSE message-mapping VHDL module against that defined in the IEC 61850-8-1 standard. The results of this analysis show that the structure of the GOOSE messages published by the developed VHDL module conforms to the IEC 61850-8-1 standard.

Section 5.3.2 focused on the GOOSE client state machine of the developed VHDL module, in this section the client state machine of the developed VHDL module was compared to that defined in the IEC 61850-8-1 standard. From the analysis of the transition of the state and sequence numbers (*stNum* and *sqNum*) it can be concluded that the client state machine implemented in the GOOSE message-mapping VHDL module follows the definition in the IEC 61850-8-1 standard.

Section 5.3.3 and 5.3.4 presented laboratory test procedures for evaluating the accuracy of the calculated RMS value (MMXN1$MX$Vol$mag) of the injected voltage signal. For this test a voltage-sag and voltage-swell disturbance was simulated and the calculated RMS voltages (MMXN1$MX$Vol$mag) were compared to those generated by the CMC 256*plus* test set. The results prove that the developed GOOSE monitoring node can be used in an IEC 61850 standard-based automation system.

Section 5.4.3.1 to section 5.4.3.4 focused on the evaluation of the accuracy of sampled values published by the MU prototype when current and/voltage distortions/disturbances were introduced. With reference to the injection tests conducted and the results obtained, it can be concluded that the published sampled values equal the signals injected into the MU prototype by the CMC 256*plus* test set. From the tests conducted and the results discussed in this chapter it can be concluded that the developed sampled value and GOOSE message-mapping VHDL modules are compliant to the IEC 61850 standard.

Chapter Six provides an overall conclusion of this research project and also discusses the deliverables, drawbacks, future work and also the application of results.

CHAPTER SIX
CONCLUSION AND FUTURE WORK

6.1 Introduction

Since its introduction, the first edition of the IEC 61850 standard for "communication networks and systems in substations" has progressively been adopted into automation systems in electrical substations. This standard introduces a communication interface referred to as the process bus which allows primary plant equipment to communicate with bay-level devices within substations using an Ethernet network. This communication interface reduces parallel copper wiring, system engineering complexity and implementation time.

This process bus consists of both IEC 61850-8-1 GOOSE and IEC 61850-9-2 SV messages. Due to the extensive and complex nature of the IEC 61850-9-2 standard, an implementation guideline known as the IEC 61850-9-2LE was published in 2004 by the UCA International Users Group (UCAIug). This IEC 61850-9-2LE is a result of co-operation between major vendors in the industry to reduce ambiguity and reduce time-to-market for Merging Units (MUs). Since the inception of the implementation guideline, a host of testing software and devices has been developed to evaluate MUs claiming conformance to this implementation guideline of the IEC 61850-9-2 standard.

These devices tend to be very expensive and information on the development of such devices is generally not available in the public domain. Presently there are companies which develop hardware independent IEC 61850 standard-compliant communication stacks, for example, Triangle MicroWorks, SISCO and SystemCORP, however, these communication stacks come at a price.

Similarly to Merging Units, devices capable of monitoring primary plant equipment (current/ voltage transformers and circuit breakers) and publishing analogue measurements and digital statuses are expensive. This is due to specialised skillset and funding for research, development and industrialisation of such devices.

This research project aims to disseminate information on the introduction and application of the IEC 61850 standard, application of the IEC 61850 process bus and the development of IEDs conforming to this standard. Chapter Five of this thesis provided test methods for evaluating the accuracy and performance of the developed limited-function Merging Unit and GOOSE monitoring node in the laboratory. In the detailed test procedures the CMC 256*plus* test set, Wireshark, IEDScout and SVScout software applications were used to validate the structure of the published GOOSE and SV messages against the structure defined in the IEC 61850-8-1 standard and the IEC 61850-9-2LE respectively.

6.2 Aims and objectives

As stated in Chapter One of this document, this research project focused on the development of VHDL modules for mapping and publishing IEC 61850-9-2 SV and IEC 61850-8-1 GOOSE messages on an FPGA platform. These developed VHDL modules implemented on an FPGA were combined with an Analogue Front-End (AFE) to produce a limited-function FPGA-based Merging Unit (MU) and a GOOSE monitoring node prototype. The developed MU and GOOSE monitoring node prototypes were tested in the laboratory to validate their conformance to the IEC 61850-9-2LE and the first edition of the IEC 61850-8-1 standard respectively. The test procedures conducted during the evaluation of the GOOSE monitoring node and Merging Unit prototype are documented in Chapter Five of this thesis.

In order to achieve the aims of this research project the following objectives were stated:

- Literature review: History of substation automation systems and the introduction of the IEC 61850 standard.
- Literature review: Analysis of the IEC 61850 standard data modeling techniques and communication service mapping for GOOSE and SV messages.
- Literature Review: Discussion and comparison of software algorithms and VHDL models for mapping and publishing IEC 61850-9-1, IEC 61850-9-2 SV messages and IEC 61850-8-1 standard-based GOOSE messages on embedded platforms.
- Methodology: Comparative analysis of different embedded platforms for mapping and publishing GOOSE and SV messages.

- Methodology: Implementation of VHDL modules for mapping and publishing GOOSE and SV messages as specified in the IEC 61850-8-1 standard and the UCA International users group IEC 61850-9-2 implementation guideline (IEC 61850-9-2LE) respectively.
- Methodology: Integration of the AFE with the GOOSE and SV message-mapping VHDL modules to produce a GOOSE monitoring node and a limited-function Merging Unit prototype.
- Evaluation: Development of laboratory test benches for evaluating the developed GOOSE monitoring node and Merging Unit prototype.
- Evaluation: Validation of the structure of SV messages published by the developed MU prototype against that defined in the IEC 61850-9-2LE using Wireshark and SVScout software applications.
- Evaluation: Conduct an interoperability test between the developed limited-function Merging Unit prototype and an IEC 61850-compliant IED.
- Evaluation: Conduct an interoperability test between the developed GOOSE monitoring node prototype and an IEC 61850-compliant IED.
- Evaluation: Validation of the structure of GOOSE messages published by the developed GOOSE monitoring node against that defined in the IEC 61850-8-1 standard using Wireshark and IEDScout software applications
- Evaluation: Evaluation of the accuracy of sampled values published by the MU prototype using the CMC 256*plus* test set to simulate five case studies covering power system nominal and fault conditions.
- Evaluation: Validation of the GOOSE client state machine implemented in the GOOSE message mapping VHDL module against that defined in IEC 61850-8-1 standard.
- Evaluation: Evaluation of the accuracy of the root mean square calculation of the voltage injected into the GOOSE monitoring node by the CMC 256*plus* test set.

6.3 Thesis deliverables

6.3.1 Literature review

The literature review section presented in Chapter Two provides background information on communication networks used in Supervisory Control and Data Acquisition (SCADA) systems before the advent of the IEC 61850 standard. In this review, shortfalls of hardwired systems and legacy protocols which prompted the development of this IEC 61850 standard were discussed. This literature review section discussed practical applications of the IEC 61850 process bus for data communication between primary plant equipment and bay-level control and protection IEDs.

Chapter Two of this thesis also presented a review of software algorithms and hardware designs of Merging Units conforming to both the IEC 61850-9-1 and IEC 61850-9-2 standards developed in past research projects. Additionally, a review of methodologies for designing devices capable of publishing IEC 61850-8-1 GOOSE messages was also presented.

6.3.2 Overview of the IEC 61850 standard

Chapter Three of this thesis document provided an overview of the first edition of the IEC 61850 standard by discussing each part of the standard from part 1 to part 10. The main objective of this chapter was to lay background information on the IEC 61850 standard's data modeling techniques, communication service interfaces, device/data naming convention and the communication service mappings of information to specific protocols for transmission.

The discussion of the IEC 61850 standard centered on part 8-1 and part 9-2, these two parts define the nomenclature, attributes and methods for mapping and publishing substation data using GOOSE and SV messages respectively. The mapping and publishing of GOOSE and SV messages as defined in the IEC 61850-8-1 standard and IEC 61850-9-2LE references the four-part IEC 61850-7 standard (IEC 61850-7-1 to IEC 61850-7-4 standards). The four-part IEC 61850-7 standard defines the data modelling and communication services to be employed for data transmission. The mapping of the GOOSE and SV messages will use control blocks, instances of common data classes and logical nodes defined in the IEC 61850-7-2, IEC 61850-7-2 and IEC 61850-7-4 standards.

Lastly, Chapter Three highlighted the advantages of using the IEC 61850 standard in a substation automation system over legacy protocols and hardwired communication systems.

6.3.3 Development of VHDL modules for mapping SV and GOOSE messages

The main aim of this research project was to develop VHDL modules for mapping and publishing GOOSE and SV messages defined in the IEC 61850-8-1 standard and the UCAIug IEC 61850-9-2 standard implementation guideline. To achieve this aim the following tasks in sections 6.3.3.1 to 6.3.3.3 were conducted.

6.3.3.1 Development of VHDL code for publishing IEC 61850-8-1 GOOSE messages

Development of finite state models and VHDL code for mapping and publishing GOOSE messages from the Xilinx Spartan 6 FPGA was presented in section 4.4.1.1 in Chapter Four. The developed VHDL modules are shown in Table 6.1 and provided in APPENDIX E.

Table 6.1: **Developed VHDL modules for mapping and publishing IEC 61850-8-1 GOOSE messages on the Xilinx Spartan 6 FPGA**

VHDL Module	Function
goose_frame.vhd	GOOSE message mapping VHDL code top entity. Implements the main state machine and mapping functions for generating GOOSE messages
fifo_core.vhd	IP Core for sending data from MAC to the LAN 8710 PHY chip
MAC2PHY4IR.vhd	Media Access Controller (MAC) implementation
publishSigGen.vhd	Implementation of the GOOSE client state machine as defined in the IEC 61850-8-1 standard
sigSynchronizer.vhd	Synchronizing signals to the clock pulse
utcTime.vhd	Time of Day module
btn_deb.vhd	De-bouncing circuit for SW2 (XCBR1$ST$Pos$stVal)
ethernet_frame.vhd	VHDL library containing formulas and records (GOOSE PDU)
Calculations.vhd	Calculation of the value of MMXN1$MX$Vol$mag.i (RMS voltage)
ethcrc32.vhd	Calculation of the Ethernet frame 32 bit CRC value

6.3.3.2 Development of VHDL code for publishing IEC 61850-9-2LE SV messages

Section 4.4.1.2 in Chapter Four details the development of finite state models and VHDL modules for mapping and publishing Sampled Value messages from the Xilinx Spartan 6 FPGA. The developed VHDL modules are shown in Table 6.2 and provided in APPENDIX F.

Table 6.2: **Developed VHDL modules for mapping and publishing IEC 61850-9-2LE SV messages on the Xilinx Spartan 6 FPGA**

VHDL Module	Function
hello_frame.vhd	SV message mapping VHDL code top entity. Implements functions for calculating the sampled values and mapping functions for generating SV messages. Implements a MAC for transmission of Ethernet frames
fifo_core.vhd	IP Core for sending data from MAC to the LAN 8710 PHY chip
MAC2PHY4IR.vhd	MAC to FIFO module
sigSynchronizer.vhd	Synchronizing signals to the clock pulse
ethernet_frame.vhd	VHDL library containing formulas and records (SV PDU)
ethcrc32.vhd	Calculation of the Ethernet frame 32 bit CRC value

6.3.3.3 Integration of the Analogue Front-End (AFE) to the GOOSE and SV message mapping VHDL modules

The AFE was integrated into the GOOSE and SV message mapping VHDL modules to produce a GOOSE monitoring node and limited-function Merging Unit prototype. This information is documented in section 4.3.2 in Chapter Four and the following tasks were conducted:

- Determining the maximum voltage and current signals that can be injected into the AFE and using these values in calculation algorithms.
- Integrating the GOOSE and SV messages mapping VHDL modules into the AFE VHDL model to extract time synchronized ADC output codes for input voltage and current calculation.

6.3.4 Evaluation of the limited-function MU and GOOSE monitoring node prototypes

The limited-function GOOSE monitoring node and MU prototypes developed in this research project were evaluated in the laboratory using the CMC 256*plus* test set, Wireshark, IEDScout and SVScout software applications. The results of the evaluation of the GOOSE monitoring node and MU prototype are presented in various sections of Chapter Five: The following tasks were conducted:

- Development of test-benches for evaluating the GOOSE monitoring node and MU prototype as shown in Figure 5.1.
- Validation of the structure of GOOSE messages published by the developed GOOSE monitoring node in section 5.3.1.
- Validation of the structure of SV messages published by the developed MU prototype in section 5.4.1.

- Evaluation of the accuracy of the sampled values published by the developed MU prototype under normal power system conditions presented in section 5.4.2.
- Evaluation of the accuracy of sampled values and MU prototype performance during voltage-sag, voltage-swell, power system frequency variation and harmonic faults simulations. These tests and results thereof are presented in sections 5.4.3.1 to 5.4.3.4.
- Evaluation of the accuracy of the root mean square value (MMXN1$MX$Vol$mag) of the injected voltage published by the developed GOOSE message-mapping VHDL module presented in sections 5.3.3 and 5.3.4.
- Validation of the GOOSE client state machine implementation in the developed GOOSE message-mapping VHDL module against that defined in the IEC 61850-8-1 standard. This validation test is presented in section 5.3.2.
- Interoperability test between the developed Merging Unit prototype and an IEC 61850-9-2LE-compliant IED (MiCOM P444 IED). This interoperability test is presented in section 5.4.4.
- Interoperability test between the developed GOOSE monitoring node prototype and an IEC 61850-8-1 standard-compliant IED (MiCOM P546 IED). This interoperability test is presented in section 5.3.5.

6.4 Challenges encountered

There have been some challenges encountered during the development of the IEC 61850-9-2 sampled value and the IEC 61850-8-1 GOOSE message-mapping VHDL modules on an FPGA platform. These include:

i. There are few papers available in the public domain which details the implementation of an algorithm for mapping sampled value messages on processor or FPGA platform, compared to GOOSE messages. This information is proprietary to vendors and is not made available in the public domain for educational purposes.

ii. There are few software tools available within the public domain to facilitate the development and testing of GOOSE and sampled value messages.

iii. Development of VHDL-93 code is challenging because it doesn't support some basic functions for floating point arithmetic which prompted the use of a floating package requiring three times the space for fixed point arithmetic, which resulted in additional memory usage within the FPGA. It must be noted that there are other versions of VHDL, for instance, VHDL-2008 which support floating point arithmetic and are therefore recommended for future work.

iv. The Analogue Front-End provided by the Centre for Substation Automation and Energy Management Systems (CSAEMS) at the CPUT is synchronised using an internally generated 1PPS signal, therefore it could not be synchronised with the CMC 256*plus* test set.

6.5 Application of results

This research covers the design, development and implementation of a GOOSE and sampled value message publisher on an FPGA platform. The results presented in this research provide many opportunities to research institutions and other interested parties which are listed below:

1. The sampled value message-mapping VHDL module developed in this research can be used to publish PMU measurement conforming to the C37.11 standard using the sampled values published by this device.
2. The developed VHDL modules for both sampled value and GOOSE messages can be used as a low cost Merging Unit or IED respectively for educational purposes.
3. The GOOSE message-mapping VHDL design can be used for other analogue or digital status communication, for example, circuit breaker SF6 measurement.
4. The developed GOOSE and SV message-mapping VHDL modules will be integrated into a process interface device currently under development at the CSAEMS at CPUT.

6.6 Future research work

The research project has been successful in achieving its stated objectives but there is some extra work that can still be conducted to improve the design and incorporate some other aspects which were not in the scope of this research project. These include:

1. GOOSE message-mapping VHDL module: this VHDL module can be improved by adding a GOOSE subscriber to subscribe to GOOSE messages published by other devices in the substation automation system.
2. GOOSE message-mapping VHDL module: An MMS stack can be developed and incorporated into this design to allow for remote device configuration.
3. SV message-mapping VHDL module: this hardware design can be further developed to publish sampled value messages at 256 samples per cycle for metering functions.
4. SV message-mapping VHDL module: Investigate the effects of over-current, over-voltage and under-voltage conditions on the published sampled value messages.
5. Investigate using the IEEE 1588v2 synchronisation technique, its advantage over the 1PPS method and its effect on the published sampled value messages.

REFERENCES

Adewole, A.C. & Tzoneva, R., 2014. Impact of IEC 61850-9-2 Standard Based Process Bus on the Operating Performance of Protections IEDS: Comparative Study. In *19th World Congress of The International Federation of Automatic Control*. Cape Town, 2014. IFAC.

Ali, I., 2012. High-speed Peer-to-peer Communication based Protection Scheme Implementation and Testing in Laboratory. *International Journal of Internation Applications*, 38(4), pp.16-24.

Alstom, 2013. *MiCOM Px4x-92LE*. [Document] Saint Ouen: General Electric Company (1.1) Available at: https://www.gegridsolutions.com/alstomenergy/grid/Global/Grid/Resources/Documents/Automation/SAS/Px4x-92LE-TM-EN-1.1-epslanguage=en-GB.pdf [Accessed 29 April 2016].

Alstom, 2014. *MiCOM P40 Agile P442, P444*. [Document] Saint-Ouen: General Electric Company (1) Available at: https://www.gegridsolutions.com/alstomenergy/grid/TechnicalManuals/P44x_EN_M_HA6.pdf [Accessed 25 April 2016].

Altium, 2008. *VHDL Language Reference*. [Document] Boston: Altium (2.0) Available at: http://valhalla.altium.com/Learning-Guides/TR0114%20VHDL%20Language%20Reference.pdf [Accessed 6 February 2015].

Amin, M., 2014. *IEEE Smart Grid, The Self Healing Grid : A concept Two decades in the making*. [Online] Available at: http://smartgrid.ieee.org/march-2013/813-the-self-healing-grid-a-concept-two-decades-in-the-making [Accessed 30 July 2014].

Apostolov, A., 2010. IEC 61850 9-2 process bus applications and benefits. In *10th IET International Conference on Developments in Power System Protection (DPSP)*. Manchester, 2010. IET.

Apostolov, A., 2010. IEC 61850 Substation Configuration Language and its Impact on the Engineering of Distribution Substation Systems. In *CIDEL*. Buenos Aires, 2010. CIDEL.

Baigent, D., Adamiak, M. & Mackiewicz, R., 2004. *IEC 61850 communication networks and systems in substations: an overview for users*. San Jaose: SISCO Systems.

Baranov, F.P. et al., 2013. Software for Emulating the Sampled Values Transmission in Accordance with IEC 61850 Standard. In *2nd International Symposium on Computer, Communication, Control and Automation(3CA 2013)*. Pulau Ujong, 2013. Atlantic Press.

Basic Electronics Tutorials, 2016. *RMS Voltage Tutorial*. [Online] Available at: http://www.electronics-tutorials.ws/accircuits/rms-voltage.html [Accessed 11 April 2016].

Bishop, D., 2006. *VHDL-2008 Support Library*. [Online] Carlifornia: VHDL Available at: http://www.vhdl.org/fphdl/ [Accessed 1 August 2015].

Bowe, N., 2014. *Analogue and Binary GOOSE transfer in an A.Eberle REG-DA Voltage Regulating Relay*. [Document] Auckland: HV Power Measurements & Protection Ltd (2.0) Available at: www.hvpower.co.nz/TechnicalLibrary/A-Eberle/GOOSE_transfer.pdf [Accessed 15 September 2015].

Brunner, C. & Apostolov, A., 2010. Functional Testing of IEC 61850 based systems. *Protection, Automation and Control World (PACWorld)*, 1 December. pp.1-5.

Cabrera, C., Chiu, S. & Nair, N.K.C., 2012. Implementation of arc-flash protection using IEC 61850 GOOSE messaging. In IEEE, ed. *International Conference on Power System Technology (POWERCON)*. Auckland, 2012. IEEE.

Csanyi, E., 2015. *Electrical Engineering Portal*. [Online] Available at: http://electrical-engineering-portal.com/definition-of-harmonics-and-their-origin [Accessed 2 October 2015].

D & B Power Associates, 2015. *Understanding Power Disturbances*. [Document] South Carolina: D & B Power Associates Incoporated Available at: http://www.dbpowerinc.com/resources/Understanding%20Power%20Disturbances.pdf [Accessed 5 March 2015].

Daboul, M., Wasserbauser, V. & Orsagova, J., 2015. Laboratory testing of the communication based protection relays. In *21st Conference student EEICT*. Brně, 2015. Fakulta elektrotechniky a komunikačních technologií.

Digilent Inc, 2013. *Nexys 3 Board Reference Manual*. [Document] Pullman: Xilinx (1) Available at: http://www.xilinx.com/support/documentation/university/XUP%20Boards/XUPNexys3/documentation/Nexys3_rm.pdf [Accessed 3 June 2014].

Ellis, R.G., 2001. *Power system harmonics: A reference guide to causes, effects and corrective measures*. [Document] Canada: Allen-Bradley (1) Available at: literature.rockwellautomation.com/idc/groups/./mvb-wp011_-en-p.pdf [Accessed 18 May 2016].

Fan, C. et al., 2011. The development of tri-status measure and control device used in smart substation of china. In IEEE, ed. *Asia-Pacific in Power and Energy Engineering Conference (APPEEC)*. Wuhan, 2011. IEEE.

Farhangi, H., 2010. The path of the smart grid. *IEEE Power and Energy Magazine*, VIII(1), pp.18-28.

Fernandes, C., Borkar, S. & Gohil, J., 2014. Testing of GOOSE Protocol of IEC 61850 Standard in Protection IED. *International Journal of Computer Applications*, 93(16), pp.30-35.

Gajic, Z. et al., 2014. Using IEC 61850 analogue GOOSE messages for OLTC control of parallel transformers. In IET, ed. *10th IET International Conference on Developments in Power System Protection (DPSP)*. Manchester, 2014. IET.

Gonzalez-Redondo, M.J. et al., 2013. IEC 61850 GOOSE transfer time measurement in development stage. *IEEE Symposium on Industrial Electronics*, pp.1-6.

Guo, Z., Vahid, F., Najjar, W. & Vissers, K., 2004. A quantitative analysis of the speedup factors of FPGAs over processors. In ACM, ed. *ACM/IEEE Conference on Field Programmable Gate Array*. Carlifornia, 2004. ACM.

Gurbiel, M. et al., 2009. Merging Unit accuracy testing. In IEEE, ed. *Power & Energy Society General Meeting (PES)*. Alberta, 2009. IEEE.

Haude, J., 2010. Testing of Sampled Value Transmission over IEC 61850 Process Bus. Amprion, 2010. OMICRON.

Honeth, N., Khurram, Z.A., Zhao, P. & Nordstrom, L., 2013. Development of the IEC 61850-9-2 software Merging Unit IED test and training platform. In *PowerTech*. Grenoble, 2013. IEEE.

Hui, C., Jiong-cong, C., Xiao-bing, L. & Nan-hua, Y., 2010. Development of Digital Protective Relay Tester under IEC 61850. In IEEE, ed. *China International Conference on Electricity Distribution (CICED)*. Nanjing, 2010. IEEE.

Iloh, J.P.I., Mbachu, C.B. & Uzhede, G.O., 2014. An improved Merging Unit model for substation automation system based on IEC 61850. *Interational Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, III(11), pp.13054-63.

Ingram, D.M., Campbell, D.A., Schaub, P. & Ledwich, G., 2011. Test and evaluation system for multi-protocol sampled value protection schemes. In IEEE, ed. *PowerTech*. Trondheim, 2011. IEEE.

Ingram, D.M., Schuab, P., Taylor, R.R. & Campbell, D.A., 2012. Performance Analysis of IEC 61850 Sampled Value Process Networks. *IEEE Transactions on Industrial Informatics*, IX(3), pp.1445-54.

International Electrotechnical Commission, 2002-07. *IEC 60044-8 : Instrument Transformers - Electronic Current Transformers*. 1st ed. Geneva: IEC.

International Electrotechnical Commission, 2003-05. *Part 7-2 : Basic Communication structure for substation and feeder equipment: Abstract Communication Service Interface.* 1st ed. Geneva: IEC.

International Electrotechnical Commission, 2003-07. *Part 5: Communication Requiremements for functions and device models*. 1st ed. Geneva: IEC.

International Electrotechnical Commission, 2003-2004. *Part 1: Introduction and overview*. 1st ed. Geneva: IEC.

International Electrotechnical Commission, 2003-2004. *Part 6 : Configuration description language for communication in electrical substations related to IEDs*. 1st ed. Geneva: IEC.

International Electrotechnical Commission, 2003-2004. *Part 7-1: Basic Communication for substation and feeder equipment - Principles and models*. 1st ed. Geneva: IEC.

International Electrotechnical Commission, 2003-2005. *Part 7-3: Basic Communication Structure for Substation and Feeder Equipment - Common Data Classes*. 1st ed. Geneva: IEC.

International Electrotechnical Commission, 2003-2005. *Part 7-4: Basic communication structure for substation and feeder equipment - Compatible logical node classes and data classes*. 1st ed. Geneva: IEC.

International Electrotechnical Commission, 2004-2005. *Part 8-1: Specific Communication Service Mapping (SCSM) - Mappings to MMS (ISO 9506-1 and ISO 9506-2) and to ISO/IEC 8802-3*. 1st ed. Geneva: IEC.

International Electrotechnical Commission, 2004. *Part 9-2 : Specific Communication Service Mapping (SCSM) - Sampled values over ISO/IEC 8802-3*. 1st ed. Geneva: IEC.

International Electrotechnical Commission, 2005-05. *Part 10: Conformance testing*. 1st ed. Geneva: IEC.

Ito, H. & Ohashi, K., 2008. Implementation of High Perfomance Protection and Relay Testing: IEC 61850 GOOSE. *Protection, Automation and Control World (PACWorld)*, 1 February. pp.40-47.

Jing, L. et al., 2011. The feasibility study of advanced fuction of Merging Unit in the intelligent digital substation. In IEEE, ed. *The International Conference on Advanced Power System Automation and Protection (APAP)*. Beijing, 2011. IEEE.

Khuraam, Z.A., 2012. *Interface between Process Equipment and Process Bus for Light Weight Testing of Protection Functions*. Masters Thesis. Stockholm: KTH Royal Institute of Technology KTH Royal Institute of Technology.

Kirrmann, H., 2004. *Introduction to IEC 61850 substation communication standard*. Research Paper. Zurich: ABBCH-RD ABB Switzerland Ltd.

Konka, J.W., Aurthur, C.M., Garcia, F.J. & Atkinson, R.C., 2011. Traffic generation of IEC 61850 sampled values. In IEEE, ed. *IEEE First International Workshop on Smart Grid Modeling and Simulation (SGMS)*. Glasgow, 2011. IEEE.

Kriger, C., Behardien, S. & Retonda-Modiya, J.C., 2013. Analysis of GOOSE and Sampled Value Message Structure for educational Purposes. *International Journal of Computers Communications & Control*, VIII(5), pp.708-21.

Lee, H.H., Kim, G.S., Lee, J.H. & Kim, B.J., 2008. Real-Time Communications on IEC 61850 Process Bus Based Distributed Sampled Measured Values Applications in Merging Unit. In Springer-Verlag, ed. *Advanced Intelligent Computing Theories and*

Applications. With Aspects of Theoretical and Methodological Issues. Heidelberg, 2008. Springer-Verlag.

Liu, J., Li, K. & Yang, H., 2007. The design of a Merging Unit of electronic transfomers based on ARM. In IEEE, ed. *42nd International Universities Power Engineering Conference (UPEC)*. Brighton, 2007. IEEE.

Luwaca, E., 2014. *Virtualization of a sensor node to enable the simulation of IEC 61850 based sampled value messages*. MTech Thesis. Cape Town: CPUT Cape Peninsula University of Technology.

Mackiewicz, R., 2011. IEC 61850 Technical Overview and Summary of Other Related IEC Standards. Michigan, 2011. SISCO Inc.

MingCai, K., Tong, S., Qian, B.Z. & Ping, Z.X., 2012. Designation and development of Hardware platform for Intelligent Terminal. In IEEE, ed. *China International Conference on Electricity Distribution (CICED)*. Shanghai, 2012. IEEE.

National Energy Regulator of South Africa, 2008. *The South African Grid Code: Network Code*. [Document] Pretoria: National Energy Regulator of South Africa (7.0) Available at: http://www.nersa.org.za/Admin/Document/Editor/file/Electricity/Compliance%20Monitoring/SAGC%20Network%20Version%207_March%202008.pdf [Accessed 2 March 2016].

NettedAutomation, 2007. *Previews of IEC 61850 standards: Communication networks and systems for power utility automation*. [Online] Available at: http://www.nettedautomation.com/news/n_72.html [Accessed 17 September 2014].

Netto, U.C., de Castro G, D., Lonel, I.D. & Coury, D.V., 2012. A behaviour evaluation of network traffic in a power substation concerning GOOSE messages. In IEEE, ed. *Power and Energy Society General Meeting*. California, 2012. IEEE.

Nick, S., 2014. *An Investigation approach to test Protection Intelligent Electronic Devices (IEDs) in IEC 61850 based substation automation systems (SAS) at station Level.* Master's Thesis. Brisbane: Queensland University of Technology Queensland University of Technology.

NIST, 2010. *NIST Framework and Roadmap for Smart Grid Interoperability*. [Document] Gaithersburg: ational Institute of Standards and technology (3.0) Available at: http://www.nist.gov/smartgrid/upload/NIST-SP-1108r3.pdf [Accessed 9 September 2015].

OMICRON, 2014. *IEC 61850 testing tools*. [Document] California: Omicron (1) Available at: https://www.omicronenergy.com/fileadmin/user_upload/pdf/literature/IEC-61850-Testing-Tools-ENU.pdf [Accessed 5 June 2015].

Retonda-Modiya, J.C., 2012. *Development of an embedded system actuator node for integration into an IEC 61850 based substation automation application*. MTech Thesis. Cape Town: CPUT CPUT.

RTDS, 2013. *RTDS Application*. [Online] Available at: http://www.rtds.com/applications/applications.html.

Rudigier, M. & Steinhauser, F., 2015. *Testing Sampled Values publishers with Daneo-400*. [Document] Stafford: Omicron (1.0) Available at: https://www.google.co.za/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0ahUKEwisoJyOt5HNAhXEJ8AKHT1UBnMQFggbMAA&url=https%3A%2F%2Fwww.omicronenergy.com%2Ffileadmin%2Fuser_upload%2Fpdf%2Fappnotes%2FDANEO-400-AppNote-Testing-SV-Publishers-2015 [Accessed 2015 May 2].

RuggedCom Inc, 2011. *Evolution of Substation Communication*. [Online] Available at: http://www.ruggedcom.com/applications/electric-utilities/evolution/ [Accessed 2013].

Schmid, J. & Kunde, K., 2011. Application of non conventional voltage and currents sensors in high voltage transmission and distsribution systems. In *International Conference on Smart Measurements for Future Grids (SMFG)*. Bologna, 2011. IEEE.

Schneider Electric, 2010. *[Protection Relay] MiCOM P543 P546*. [Brochure] Schneider Electric (1.0) Available at: http://download.schneider-electric.com/files?p_File_Id=683083615&p_File_Name=P54d_ds_1317.pdf [Accessed 1 May 2016].

Schneider Electric, 2011. *MiCOM P54x: P543, P544, P545 & P546 Current Differential Relay*. [Document] Rueil-Malmaison: Schneider Electric (1.0) Available at: http://www.schneider-electric.co.kr/documents/Catalogue/MiCOM_P543to546.pdf [Accessed 5 May 2016].

Schneider Electric, 2016. *Schneider Electric Library*. [Online] Schneider Electric Available at: http://www.schneider-electric.co.za/library/SCHNEIDER_ELECTRIC/SE_LOCAL/APS/211350_4DC2/MiCOM_S1_Studio_-_Get_Started.pdf [Accessed 14 August 2016].

Seymour, J., 2011. *The Seven Types of Power Problems*. [Document] Rueil-Malmaison: Schneider Electric (1) Available at: https://www.google.co.za/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0ahUKEwj8g9fRuJHNAhWFBsAKHaFPDMsQFggbMAA&url=http%3A%2F%2Fwww.apcmedia.com%2Fsalestools%2FVAVR-5WKLPK%2FVAVR-5WKLPK_R1_EN.pdf&usg=AFQjCNH_HqCnlrP0e5GvSlJMbvClweEOFw&sig2= [Accessed 2 May 2015].

Sidhu, T.S. & Gangadharan, P.K., 2005. Control and automation of power system substation using IEC 61850 communication. In IEEE, ed. *Conference on Control Applications (CCA)*. Toronto, 2005. IEEE.

Starck, J. et al., 2013. Switchgear optimization using IEC 61850-9-2. In IET, ed. *22nd International Conference and Exhibition on Electricity Distribution (CIRED 2013)*. Stockholm, 2013. IET.

Sumec, S., 2014. Software tool for verification of sampled values transmission via IEC 61850-9-2 protocol. In *15th International Scientific Conference on Electric Power Engineering (EPE)*. Brno, 2014. IEEE.

Sun, X., Redfern, M. & Aggarwal, P., 2012. *Protection Performance Study for Secondary Systems with IEC 61850 Process Bus Architecture*. PhD Thesis. Bath: University of Bath University of Bath.

Texas Instruments, 2013. *Analogue Front-End for Power Monitoring, Control and Protection*. [Document] Dallas: Texas Instruments (B) Available at: www.ti.com/lit/gpn/ads131e08 [Accessed 25 March 2015].

UCA International Users Group, 2004. *Implementation Guideline for Digital Interface to Instrument Transformers using IEC 61850-9-2*. North Carolina: UCA.

Van der Spiegel, J., 2001. *VHDL Tutorial*. [Online] (1) Available at: http://www.seas.upenn.edu/~ese171/vhdl/vhdl_primer.html [Accessed 6 August 2014].

VHDL, 2015. *Fixed and Floating Point support page*. [Online] Available at: http://www.vhdl.org/fphdl/vhdl.html [Accessed 25 August 2015].

Von Dollen, D., 2009. *Report to NIST on the smart grid interoperability standards*. [Document] Gaithersburg: National Institute of Standards and technology (10) Available at: http://www.nist.gov/smartgrid/upload/Report_to_NIST_August10_2.pdf [Accessed 2015 June 4].

We Energies, 2015. *Disturbance Types*. [Online] Available at: https://www.we-energies.com/powerquality/disturbance_type.htm [Accessed 12 October 2015].

Wei-ming, W., Xiong-ying, D. & Yan, L., 2011. Research and Development of an Intelligent Merging Unit based on IEC 61850-9-2. In IEEE, ed. *2011 4th International Conference on Electric Utility Deregulation and Restructuring and Power Technologies (DRPT)*. Weihai, 2011. IEEE.

Weiss, S., Graeve, P. & Andersson, A., 2011. Benefits of converting conventional instrument transformer data into smart grid capable process data utilizing IEC 61850 merging unit. In *21st International Conference on Electricity Distribution (CIRED)*. Frankfurt, 2011. CIRED.

Wester, C., Smith, T., Theron, J. & McGinn, D., 2014. Developments in Fast Load Shedding. In *67th Annual Conference for Protective Relay Engineers*. College Station, 2014. IEEE.

Xiaobin, L. et al., 2008. Development of a New Kind of Merging Unit Based On IEC 61850. *CICED: Protection, control, communication and automation of distribution network*, III(III), pp.1-5.

Xilinx, 2011. *Spartan-6 FPGA Block RAM Resources User Guide*. [Document] California: Xilinx (1.5) Available at: www.xilinx.com/support/documentation/user_guides/ug383.pdf [Accessed 25 July 2015].

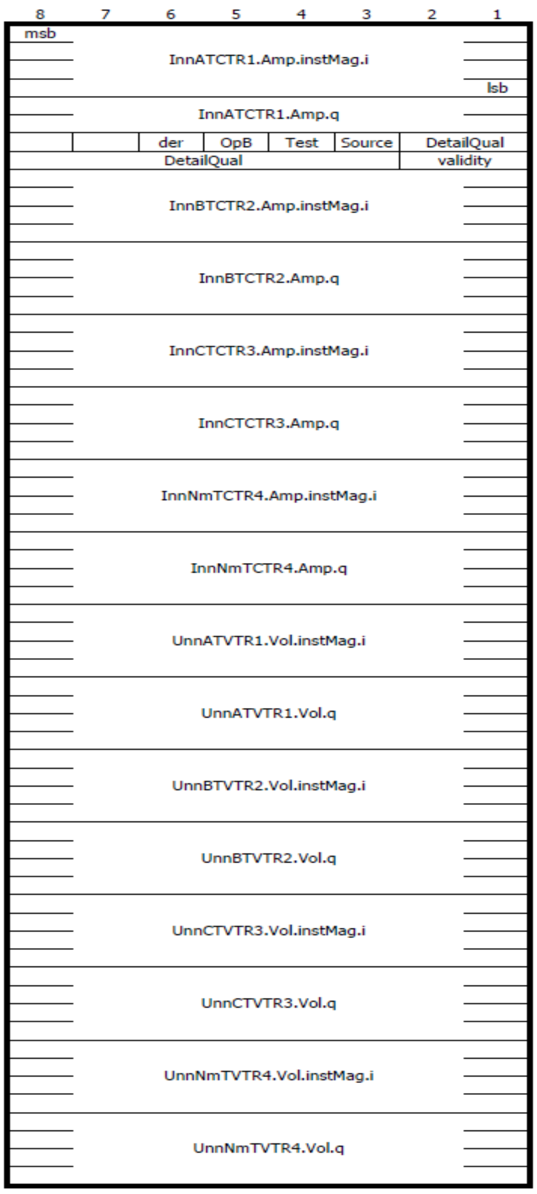
Yin, Z.L. & Liu, W.S., 2004. A Novel FPGA-Based Method to Design the Merging Unit Following IEC 61850. *International Conference on Power System Technology (PowerCon)* , I, pp.260-63.

Zhao, P., 2012. *IEC 61850-9-2 Process Bus Communication Interface for Light Weight Merging Unit Testing Unit Testing*. Master's Degree. Stockholm: KTH Engineering KTH Royal Institute of Technology.

Zhengyang, Z. et al., 2011. The design of the merging unit of real time and synchronicity based on EP1C3T144 chip. *The International Conference on Advanced Power System Automation and Protection*, III, pp.2341-45.

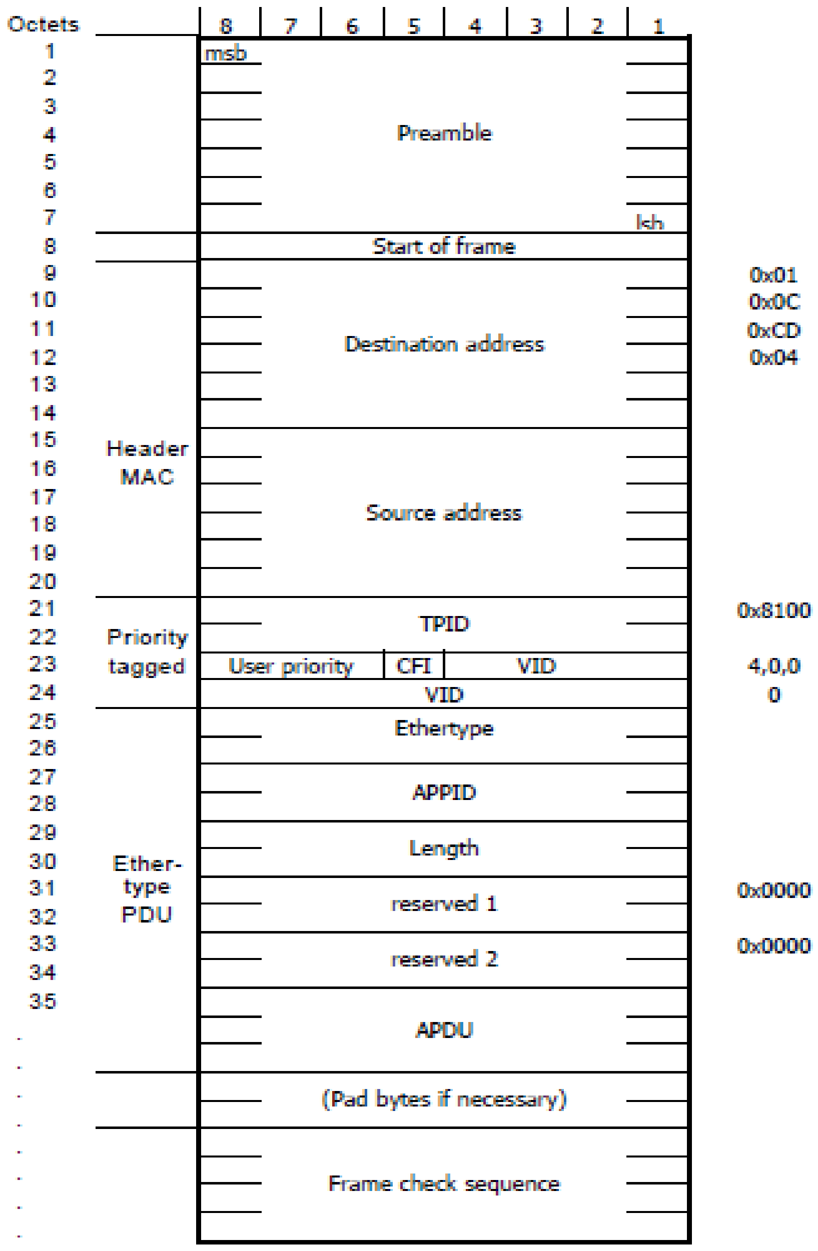
APPENDIX A. IEC 61850-9-2LE dataset

This is the PhsMeas1 dataset defined in the UCAIug IEC 61850-9-2 implementation guideline for the transmission of sampled values. This dataset consists of four TCTR and four TVTR logical node instances for the transmission of current and voltage samples of the A, B, C and N phases.



APPENDIX B. Contents of an ISO 8802-3 Ethernet frame

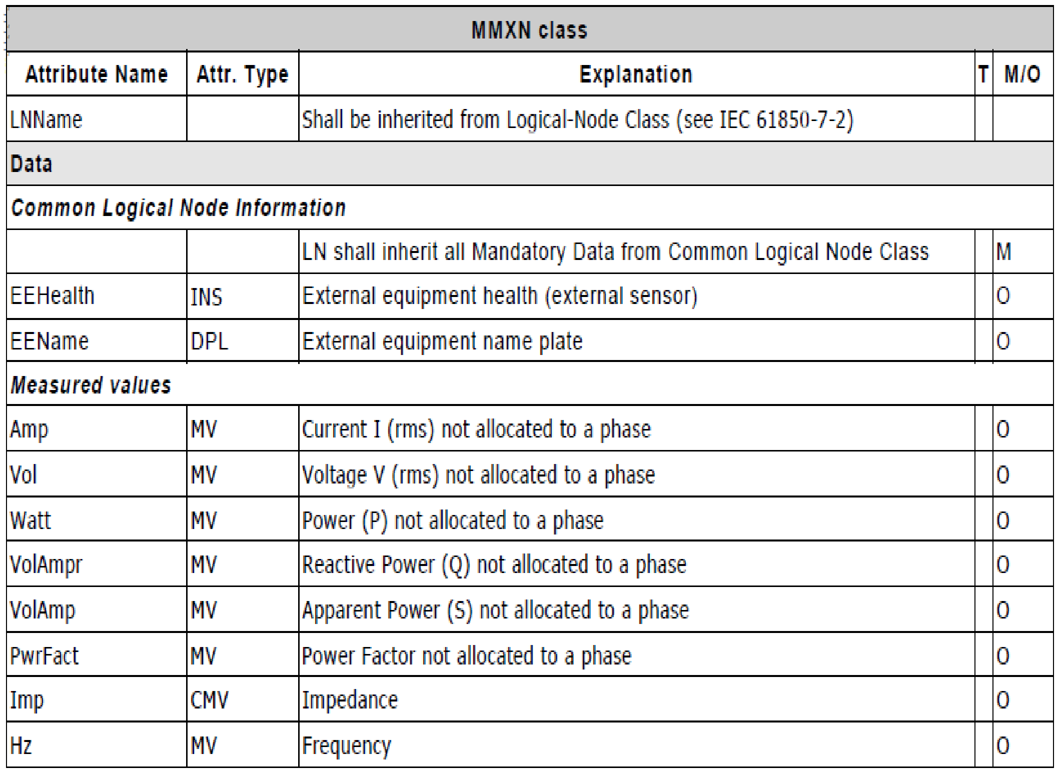
This is the ISO 8802-3 Ethernet frame used for the transmission of GOOSE and Sampled Value messages as defined in the IEC 61850-8-1 and IEC 61850-9-2 standards (International Electrotechnical Commission, 2004-2005), (UCA International Users Group, 2004).



APPENDIX C. : Logical Node Classes

Appendix C.1 : MMXN Logical Node

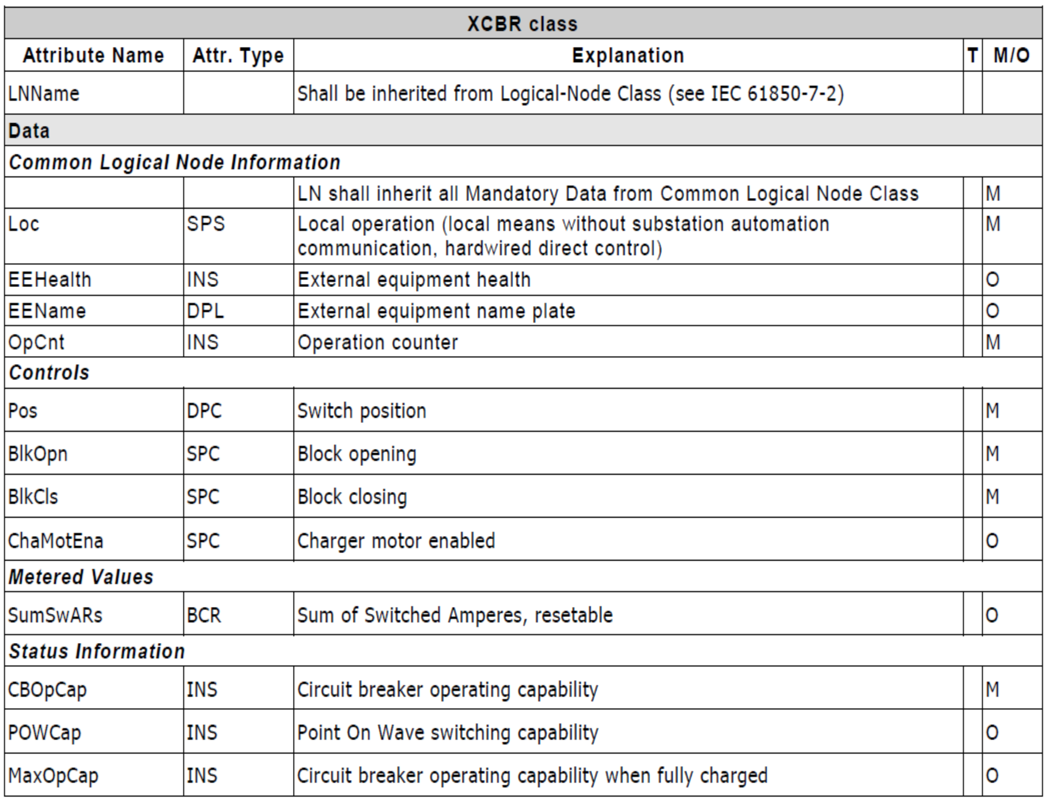
This logical node class is used for the calculation of attributes of single phase system. The calculated attributes are the power, currents, frequency, impedances and voltages.



MMXN class				
Attribute Name	Attr. Type	Explanation	T	M/O
LNName		Shall be inherited from Logical-Node Class (see IEC 61850-7-2)		
Data				
Common Logical Node Information				
		LN shall inherit all Mandatory Data from Common Logical Node Class		M
EEHealth	INS	External equipment health (external sensor)		O
EEName	DPL	External equipment name plate		O
Measured values				
Amp	MV	Current I (rms) not allocated to a phase		O
Vol	MV	Voltage V (rms) not allocated to a phase		O
Watt	MV	Power (P) not allocated to a phase		O
VolAmpr	MV	Reactive Power (Q) not allocated to a phase		O
VolAmp	MV	Apparent Power (S) not allocated to a phase		O
PwrFact	MV	Power Factor not allocated to a phase		O
Imp	CMV	Impedance		O
Hz	MV	Frequency		O

Appendix C.2 : XCBR Logical Node

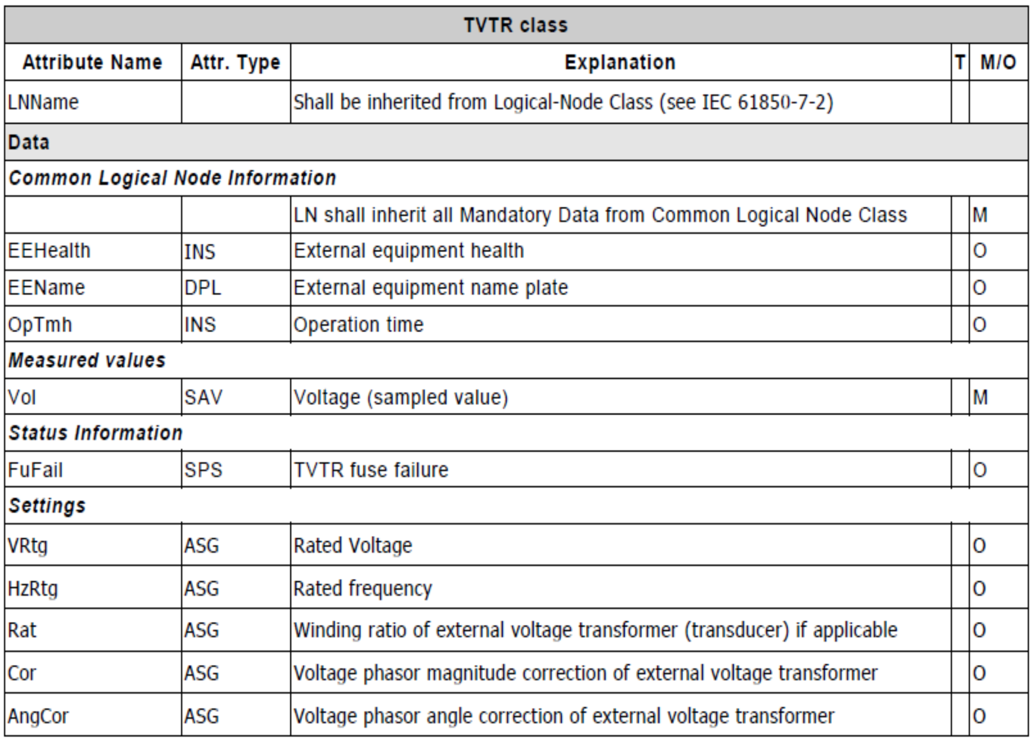
This logical node is used for modelling of substation switches with short circuit breaking capabilities, e.g., circuit breakers.



XCBR class				
Attribute Name	**Attr. Type**	**Explanation**	**T**	**M/O**
LNName		Shall be inherited from Logical-Node Class (see IEC 61850-7-2)		
Data				
Common Logical Node Information				
		LN shall inherit all Mandatory Data from Common Logical Node Class		M
Loc	SPS	Local operation (local means without substation automation communication, hardwired direct control)		M
EEHealth	INS	External equipment health		O
EEName	DPL	External equipment name plate		O
OpCnt	INS	Operation counter		M
Controls				
Pos	DPC	Switch position		M
BlkOpn	SPC	Block opening		M
BlkCls	SPC	Block closing		M
ChaMotEna	SPC	Charger motor enabled		O
Metered Values				
SumSwARs	BCR	Sum of Switched Amperes, resetable		O
Status Information				
CBOpCap	INS	Circuit breaker operating capability		M
POWCap	INS	Point On Wave switching capability		O
MaxOpCap	INS	Circuit breaker operating capability when fully charged		O

Appendix C.3 : TVTR Logical Node

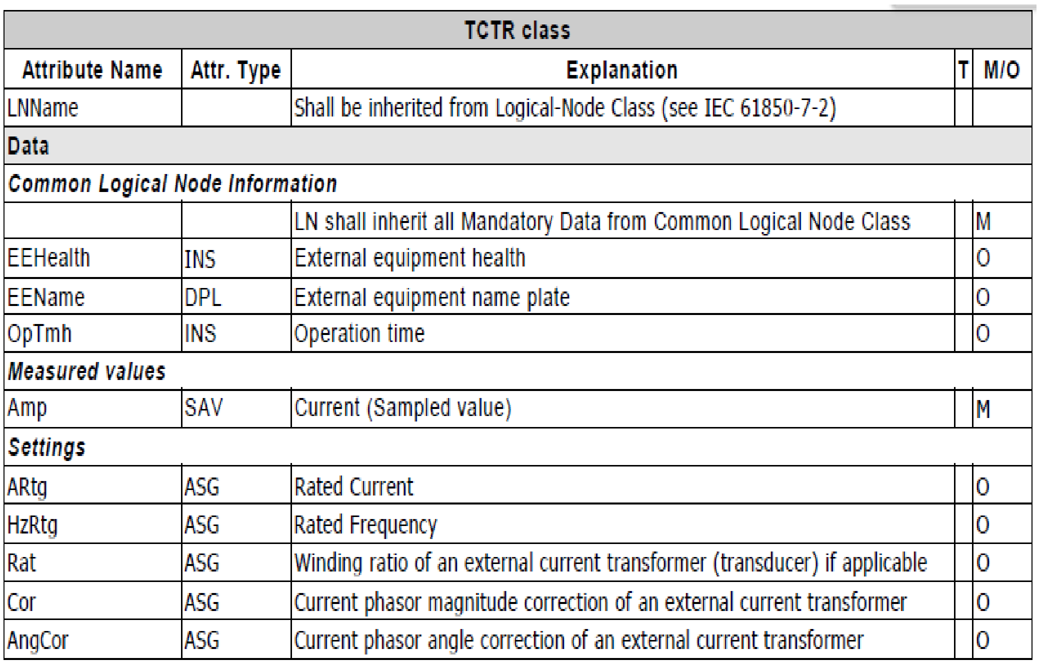
This logical node is defined in the IEC 61850-7-4 standard for the modelling of Voltage transformers (VTs). The measured voltage is transmitted as sampled values according to the specific communication service mapping defined in the IEC 61850-9-2 standard.



TVTR class				
Attribute Name	**Attr. Type**	**Explanation**	**T**	**M/O**
LNName		Shall be inherited from Logical-Node Class (see IEC 61850-7-2)		
Data				
Common Logical Node Information				
		LN shall inherit all Mandatory Data from Common Logical Node Class		M
EEHealth	INS	External equipment health		O
EEName	DPL	External equipment name plate		O
OpTmh	INS	Operation time		O
Measured values				
Vol	SAV	Voltage (sampled value)		M
Status Information				
FuFail	SPS	TVTR fuse failure		O
Settings				
VRtg	ASG	Rated Voltage		O
HzRtg	ASG	Rated frequency		O
Rat	ASG	Winding ratio of external voltage transformer (transducer) if applicable		O
Cor	ASG	Voltage phasor magnitude correction of external voltage transformer		O
AngCor	ASG	Voltage phasor angle correction of external voltage transformer		O

Appendix C.4 : TCTR Logical Node

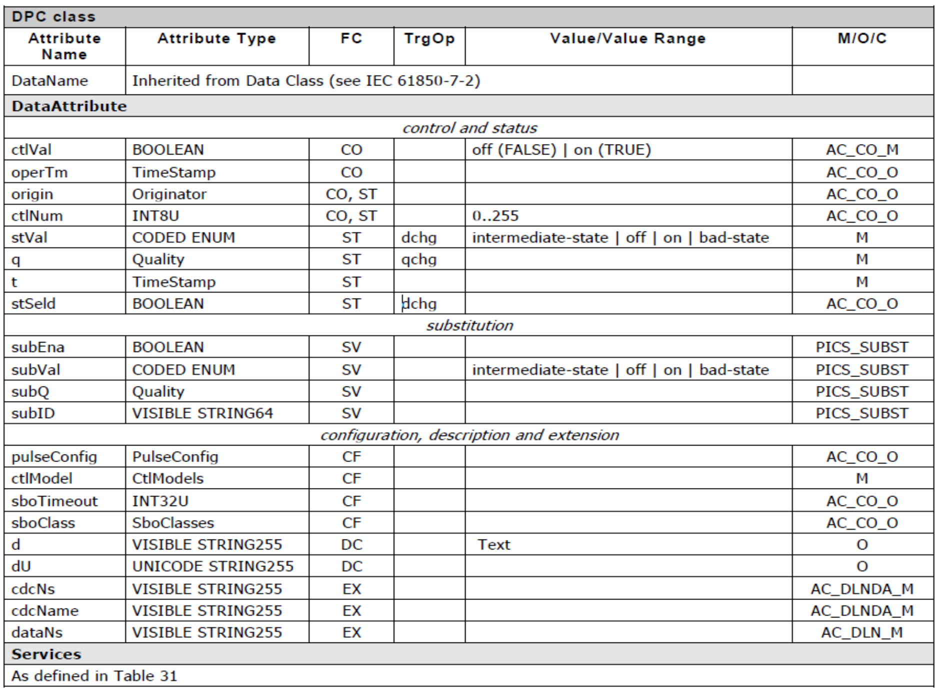
This logical node is defined in the IEC 61850-7-4 standard for the modelling of Current transformers (VTs). The measured current is transmitted as sampled values according to the specific communication service mapping defined in the IEC 61850-9-2 standard.



TCTR class				
Attribute Name	**Attr. Type**	**Explanation**	**T**	**M/O**
LNName		Shall be inherited from Logical-Node Class (see IEC 61850-7-2)		
Data				
Common Logical Node Information				
		LN shall inherit all Mandatory Data from Common Logical Node Class		M
EEHealth	INS	External equipment health		O
EEName	DPL	External equipment name plate		O
OpTmh	INS	Operation time		O
Measured values				
Amp	SAV	Current (Sampled value)		M
Settings				
ARtg	ASG	Rated Current		O
HzRtg	ASG	Rated Frequency		O
Rat	ASG	Winding ratio of an external current transformer (transducer) if applicable		O
Cor	ASG	Current phasor magnitude correction of an external current transformer		O
AngCor	ASG	Current phasor angle correction of an external current transformer		O

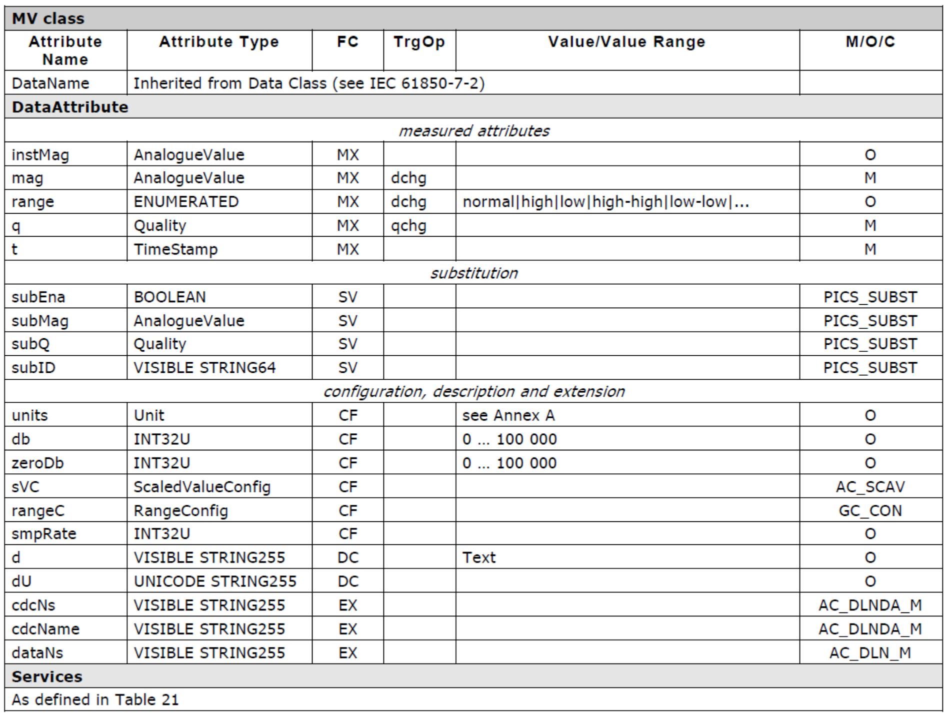
APPENDIX D. Common Data Classes

Appendix D.1 : Double Point Controllable (DPC) class



DPC class					
Attribute Name	**Attribute Type**	**FC**	**TrgOp**	**Value/Value Range**	**M/O/C**
DataName	Inherited from Data Class (see IEC 61850-7-2)				
DataAttribute					
				control and status	
ctlVal	BOOLEAN	CO		off (FALSE) | on (TRUE)	AC_CO_M
operTm	TimeStamp	CO			AC_CO_O
origin	Originator	CO, ST			AC_CO_O
ctlNum	INT8U	CO, ST		0..255	AC_CO_O
stVal	CODED ENUM	ST	dchg	intermediate-state | off | on | bad-state	M
q	Quality	ST	qchg		M
t	TimeStamp	ST			M
stSeld	BOOLEAN	ST	dchg		AC_CO_O
				substitution	
subEna	BOOLEAN	SV			PICS_SUBST
subVal	CODED ENUM	SV		intermediate-state | off | on | bad-state	PICS_SUBST
subQ	Quality	SV			PICS_SUBST
subID	VISIBLE STRING64	SV			PICS_SUBST
				configuration, description and extension	
pulseConfig	PulseConfig	CF			AC_CO_O
ctlModel	CtlModels	CF			M
sboTimeout	INT32U	CF			AC_CO_O
sboClass	SboClasses	CF			AC_CO_O
d	VISIBLE STRING255	DC		Text	O
dU	UNICODE STRING255	DC			O
cdcNs	VISIBLE STRING255	EX			AC_DLNDA_M
cdcName	VISIBLE STRING255	EX			AC_DLNDA_M
dataNs	VISIBLE STRING255	EX			AC_DLN_M
Services					
As defined in Table 31					

Appendix D.2 : Measured Value (MV) class



MV class					
Attribute Name	**Attribute Type**	**FC**	**TrgOp**	**Value/Value Range**	**M/O/C**
DataName	Inherited from Data Class (see IEC 61850-7-2)				
DataAttribute					
				measured attributes	
instMag	AnalogueValue	MX			O
mag	AnalogueValue	MX	dchg		M
range	ENUMERATED	MX	dchg	normal|high|low|high-high|low-low|…	O
q	Quality	MX	qchg		M
t	TimeStamp	MX			M
				substitution	
subEna	BOOLEAN	SV			PICS_SUBST
subMag	AnalogueValue	SV			PICS_SUBST
subQ	Quality	SV			PICS_SUBST
subID	VISIBLE STRING64	SV			PICS_SUBST
				configuration, description and extension	
units	Unit	CF		see Annex A	O
db	INT32U	CF		0 … 100 000	O
zeroDb	INT32U	CF		0 … 100 000	O
sVC	ScaledValueConfig	CF			AC_SCAV
rangeC	RangeConfig	CF			GC_CON
smpRate	INT32U	CF			O
d	VISIBLE STRING255	DC		Text	O
dU	UNICODE STRING255	DC			O
cdcNs	VISIBLE STRING255	EX			AC_DLNDA_M
cdcName	VISIBLE STRING255	EX			AC_DLNDA_M
dataNs	VISIBLE STRING255	EX			AC_DLN_M
Services					
As defined in Table 21					

APPENDIX E. IEC 61850-8-1 GOOSE message-mapping VHDL module

Appendix E.1 goose_frame.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
--use ieee.std_logic_arith.all;
--use ieee.std_logic_unsigned.all;
use work.ethernet_frame.all;                                -- using package

entity goose_frame is
    Port (    -- General purpose ports
          RST : in  std_logic;                                          -- BTND -- C9
          NUM : in STD_LOGIC_VECTOR(7 downto 0);
          --PHY ports
          SampleCnt : in std_logic_vector(11 downto 0);
          Volt : in std_logic_vector(23 downto 0);               -- voltage input
          --General purpose
          MDIO : inout  std_logic;
          MDC : out  std_logic;
          RESETN : out  std_logic;
          COL : in  std_logic;
          CRS : in  std_logic;
          --TX
          TXD :out  std_logic_vector (3 downto 0);
          TXER : out  std_logic;
          TXEN: out std_logic;
          TXCLK : in  STD_LOGIC;
          --RX
          RXD : in  std_logic_vector (3 downto 0);
          RXER : in  std_logic;
          RXDV : in  std_logic;
          RXCLK : in  std_logic;
          clk : in std_logic);
end goose_frame;
```

```
architecture Behavioral of goose_frame is
   --Constants and type

   Type possible_state is (Idle,Transmitting);

   --Components
   component ethcrc32 is
     port (    clk         : in std_logic;
               rst     : in std_logic;
               en          : in std_logic;
               is_msb  : in std_logic;
               data_in     : in std_logic_vector (7 downto 0);
               crc_out     : out std_logic_vector (31 downto 0));
   end component;

   component mac2phy4IR is
        Port (
                 -- General purpose ports
                 RSTN : in  std_logic;
                 --PHY ports
                 --General purpose
                 MDIO : inout  std_logic;
                 MDC : out  std_logic;
                 RESETN : out  std_logic;
                 COL : in  std_logic;
                 CRS : in  std_logic;
                 --TX
                 TXD :out  std_logic_vector (3 downto 0);
                 TXER : out  std_logic;
                 TXEN: out std_logic;
                 TXCLK : in  STD_LOGIC;
                 --RX
                 RXD : in  std_logic_vector (3 downto 0);
                 RXER : in  std_logic;
                 RXDV : in  std_logic;
                 RXCLK : in  std_logic;
                 --MAC Ports
                 --General purpose
                 COLLISION: out std_logic;
                 CARRIER: out std_logic;
                 CLK : out std_logic;
```

```
                    --TX
                    TDATA: in std_logic_vector (7 downto 0);
                    TXVALID: in std_logic;
                    --RX
                    RDATA : out std_logic_vector (7 downto 0);
                    RXVALID: out std_logic);
    end component;

    -- ADC signal
    signal VArms : std_logic_vector(31 downto 0);                                              -- measured root
mean square value
    signal prevDigitalState, BoolChange, AnaChange, TriggerEvent : std_logic := '0';



    --GP signals
    signal gooseframe : byte_array(0 to 150) :=(others=>x"00");                    --ISO 8802-3 frame;
    signal not_reset: std_logic:='1';
    signal local_clk: std_logic:='0';

    --Counter of transmitted/received bytes
    signal byte_num: integer range 0 to 200 := 0;                       -- Maximum length of Ethernet frame is 1521
    --Keep the state of the entity
    signal state, succ_state : possible_state := Idle;                                -- MAC implementation FSM

    --Shit register to bufferize the received data for crc computation purpose
    signal data_received : byte_array(3 downto 0):=(others=>x"00");

    --Received frame counter
    signal nb_received_frame: integer range 0 to 15 :=0;
    signal nb_error_frame: integer range 0 to 15 :=0;

    signal trans_klaar, trans_done : std_logic := '0';                                -- done transmitting data
    signal rec, goEnable : std_logic := '0';
    signal goPDUattr : std_logic_vector(0 to 3) := b"0000";
    type state_machine is (setup, readGOCB, Idle ,Transmission);                   -- GOOSE message mapping state machine
    signal curr_state, next_state : state_machine := setup;                         -- state machine variables
    signal byteCnt : integer := 0;
    signal publish, msPulse, ControlBlock_GOOSE : std_logic := '0';          -- these flag used when setting up Record at
start up   signal frame : eth_frame;                                          -- create a signal for holding the SV frame
```

```
    signal goosePdu : APDU;
    signal UTCTimestamp : STD_LOGIC_VECTOR(0 to 55);
    signal timeAllowedtoLive : STD_LOGIC_VECTOR(31 downto 0);                   -- time allowed to Live

    -- Synchronizer module outputs
    signal smpCount : std_logic_vector(11 downto 0);
    signal newSample : std_logic := '0';

    signal enter_rise : std_logic := '0';
    signal sample_rise : std_logic := '0';
    signal transmit_rise : std_logic := '0';
    signal transmit : std_logic := '0';



    signal tempState: byte_array(0 to 3):=(others=>x"00");
    signal tempSeq: byte_array(0 to 3):=(others=>x"00");
    signal tempTL: byte_array(0 to 3):=(others=>x"00");
    signal tempUTC: byte_array(0 to 7):=(others=>x"00");
    signal element: byte_array(0 to 11):=(others=>x"00");

    signal  GoosePDU_StartIndex: integer range 0 to 100 := 0;            -- this is the start index of the Data set
    signal len_svpacket, count : integer range 0 to 1521:= 0;       -- maximum length of Ethernet packet ( IEC 61850-8-
1)
    signal testIndex, ndsComIndex, timeIndex, sqNumIndex, stNumIndex, tatlIndex, allDataIndex : integer := 0;

    -- Constants ----------------------------------------------------------------------------------------
    --crc_core signals
    signal crc: byte_array(0 to 3):=(others=>x"00");
    signal crc_crc_out: std_logic_vector(31 downto 0);
    signal crc_data_in : std_logic_vector(7 downto 0):=x"00";
    signal crc_en: std_logic := '0';
    signal crc_is_msb: std_logic:='1';
    signal crc_rst: std_logic:='1';
    signal crc_clk: std_logic:='0';

    -- Debounce circuit port map signals -----------------------------------------------------
    signal ndsCom, boolIn, testFlag : std_logic := '0';

    signal setup_Complete : std_logic := '0';
-- setup state complete
```

```
--mac2phy4IR signals
signal  m2p_COLLISION:  STD_LOGIC;
signal  m2p_CARRIER:  STD_LOGIC;
signal  m2p_CLK :  STD_LOGIC;
signal  m2p_TDATA:  STD_LOGIC_VECTOR (7 downto 0):=x"00";
signal  m2p_TXVALID:  STD_LOGIC:='0';
signal  m2p_RDATA :  STD_LOGIC_VECTOR (7 downto 0);
signal  m2p_RXVALID:  STD_LOGIC;

-- GOOSE Messages status and sequence numbers
signal stNum : STD_LOGIC_VECTOR(31 downto 0);
signal sqNum : STD_LOGIC_VECTOR(31 downto 0);

-- button debounce module
component btn_deb is
Port (clk    : in std_logic;
        nds    : in STD_LOGIC;
        test   : in STD_LOGIC;
        inBool: in STD_LOGIC;
        ndsCom : out STD_LOGIC;
        testFlag : out STD_LOGIC;
        boolIn : out STD_LOGIC);
end component;

-- signal Synchronizer component
component sigSynchronizer is
Port ( clk : in  STD_LOGIC;
          enter : in STD_LOGIC;
          transmit : in  STD_LOGIC;
          publish : in STD_LOGIC;
          trans_klaar : in STD_LOGIC;
          enter_rise : out STD_LOGIC;
          transmit_rise : out STD_LOGIC;
          sample_rise : out STD_LOGIC;
          trans_done : out STD_LOGIC);
end component;

-- Root mean square calculation module
component Calculations is
Port ( VinA : in  STD_LOGIC_VECTOR (23 downto 0);
        VArms : out  STD_LOGIC_VECTOR (31 downto 0);
           SampleCount : in std_logic_vector(11 downto 0);
           AnaChange : out STD_LOGIC;
           newSample : in std_logic;
           RST : in std_logic;
        clk : in  STD_LOGIC);
end component;

-- publish signal generator

component publishSigGen is
Port ( clk : in  STD_LOGIC;
       goEnable : in STD_LOGIC;
       publishSig : out  STD_LOGIC;
       timeAllowedtoLive : out STD_LOGIC_VECTOR(31 downto 0);                       -- time allowed to Live
       stateNum : out STD_LOGIC_VECTOR(31 downto 0);
       seqNum : out STD_LOGIC_VECTOR(31 downto 0);
       TriggerEvent : in  STD_LOGIC;
       millisPulse : out STD_LOGIC);
end component;

-- UTC Timestamp function
component utcTime is
Port ( clk : in  STD_LOGIC;
       msPulse : in STD_LOGIC;
       UTCTimestamp : out STD_LOGIC_VECTOR(0 to 55));
end component;

begin
    -- Mapping signal from the root mean calculation
        VrmsCalculator :  Calculations port map ( VinA => Volt,
                                                  RST => RST,
                                                  AnaChange => AnaChange,
                                                  SampleCount => SampleCnt,
                                                  VArms => VArms,
                                                  newSample => newSample,
                                                  clk => clk);

    -- Mapping signals from the utcTime module

    utcTimeModule : utcTime port map ( clk => clk,
                                   msPulse => msPulse,
                                   UTCTimestamp => UTCTimestamp);

-- Mapping the signals from sigSynchronizer to the top module

sigSync : sigSynchronizer port map ( clk => clk,
                                     enter => setup_Complete,
                                     transmit => transmit,
                                     publish => publish,
                                     trans_klaar => trans_klaar,
                                     enter_rise => enter_rise,
                                     transmit_rise => transmit_rise,
                                     trans_done => trans_done,
                                     sample_rise => sample_rise);

-- Mapping the signals from publishSigGen to the top Module

sigGenPublish : publishSigGen port map (clk => clk,
                                        goEnable => goEnable,
                                        publishSig => publish,
                                        timeAllowedtoLive => timeAllowedtoLive,
                                        stateNum => stNum,
                                        seqNum => sqNum,
                                        TriggerEvent => TriggerEvent,
                                        millisPulse => msPulse);
--Mapping signals from the debounce Module to the Top VHD module

btn_debounce : btn_deb port map (clk => msPulse,                  -- use 1 ms to debounce the switch
                                 nds => Num(7),
                                 test => Num(6),
                                 inBool => Num(2),                -- emulated circuit breaker status
                                 ndsCom => ndsCom,
                                 testFlag => testFlag,
                                 boolIn => boolIn);
-- Component instantiation
crc_core : ethcrc32 port map (crc_clk,crc_rst,crc_en,crc_is_msb,crc_data_in,crc_crc_out);
m2p      : mac2phy4IR port map(not_reset,MDIO,MDC,RESETN,COL,CRS,
                                         TXD,TXER,TXEN,TXCLK,
                                         RXD,RXER,RXDV,RXCLK,
                                         m2p_COLLISION,m2p_CARRIER,m2p_CLK,m2p_TDATA,m2p_TXVALID,
                                         m2p_RDATA,m2p_RXVALID);
```

```
--Trivial mapping
not_reset <= not rst;
local_clk <= m2p_CLK;
crc(0)<= crc_crc_out(31 downto 24);
crc(1)<= crc_crc_out(23 downto 16);
crc(2)<= crc_crc_out(15 downto 8);
crc(3)<= crc_crc_out(7 downto 0);

----------------------------------------------------------
-- state transition controller for the MAC
----------------------------------------------------------
SYNC_FSM_TRANS : process (clk, succ_state)
begin
    state <= succ_state;                             -- go to the next state
end process SYNC_FSM_TRANS;

----------------------------------------------------------------
-- state transition controller for the Main Finite State Machine
----------------------------------------------------------------
SYNC_FSM_MAIN : process (clk, next_state)
begin
    curr_state <= next_state;
end process SYNC_FSM_MAIN;

-----------------------------------------------------------------------------------------
-- Main FSM state transition logic
-----------------------------------------------------------------------------------------
FSM : process( enter_rise, curr_state, trans_done , clk, transmit_rise, goEnable)
begin
    if clk'event and clk = '1' then
        case (curr_state) is
            when setup =>
                next_state <= setup;                         -- stay on setup state
                if enter_rise = '1' then
                    next_state <= readGOCB;                   -- on rising edge of ENTER go to sampling state
                end if;
            when readGOCB =>                                 -- when reading the sampled value control block
                next_state <= readGOCB;
                if goEnable = '1' then                        -- if the control block has been set then continue
                    next_state <= Idle;                       -- start sampling data
                end if;
```

```
                    when Idle =>
                        next_state <= Idle;
                        if transmit_rise = '1' then      -- start transmitting data as soon as it is copied onto the frame
                            next_state <= Transmission;
                        end if;
                    when Transmission =>
                        next_state <= Transmission;                         -- stay on setup state
                        if trans_done = '1' then
                            next_state <= Idle;      -- if transmission is complete then shift the ASDUs asdu(n) = asdu(n-1)
                        end if;
                end case;
            end if;
        end process FSM;


------------------------------------------------------------------------------------
-- Finite state machine outputs
-- sv_frame
-- frame (record)
-- succ_state
------------------------------------------------------------------------------------
FSM_OUTPUT : process (clk, curr_state, sample_rise, trans_done, transmit_rise, publish)

    variable conv_int : std_logic_vector(0 to 31) := x"00000000";
-- temporary array to hold converted integer
    variable index : integer range 0 to 1521 := 0;
    -- goosePDU handling signals
    variable len_gocbref : integer range 0 to 66 := 0;                      -- VISIBLE STRING65 + 2
    variable len_datset : integer range 0 to 66 := 0;                       -- VISIBLE STRING65 + 2
    variable len_GOID : integer range 0 to 66 := 0;                         -- VISIBLE STRING65 + 2
    variable len_IEE802 : integer range 0 to 500 := 0;                      -- length of the Ethernet frame
    variable len_goosepdu : integer range 0 to 500 := 0;                    -- length of seqASDU(n)
    variable tempIndex : integer range 0 to 100 := 0;                       -- length of seqASDU(n)
    begin
--        if RST = '1' then
--            gooseframe(0 to 100) <= (others=>x"00");
--            goPDUattr <= b"0000";
--            len_GOID := 0;
--            len_GOCBREF := 0;
--            len_datSet := 0;
--            len_GOOSEPDU := 0;
--          len_IEE802 := 0;
--          len_svpacket <= 0;
--          goEnable <= '1';
--          byteCnt <= 0;
--          transmit <= '0';
--          transmit <= Idle;
        if clk = '1' and  clk'event then                            -- state transitions happen at rising edge of the
clock
            succ_state <= Idle;                                         -- Transmission FSM defaults at Idle state
            goEnable <= '1';                                                -- default vale for svEnable
            transmit <= '0';                                                -- start transmit of the frame
            case (curr_state) is
                when setup =>
                    goEnable <= '0';
                    --  Caluculate lengths

                    len_GOID := GOOSE_ID'right;                             -- total length of  gocbRef+TAG+L
                    len_GOCBREF := GOOSE_REF'right;                        -- length of the        goID+TAG+L
                    len_datSet := GOOSE_DATASET'right;                    -- length of            datSet+TAG+L
                    len_GOOSEPDU := (len_GOCBREF + 2) + goosePdu.timeAllowedtoLive'right + (len_datSet + 2) + (len_GOID
+ 2) + goosePdu.t'right + goosePdu.stNum'right + goosePdu.sqNum'right + goosePdu.test'right + goosePdu.confRev'right +
goosePdu.ndsCom'right + goosePdu.numDataSetEntries'right;
                    len_GOOSEPDU := len_GOOSEPDU + goosePdu.allData'right + 2;

                    len_IEE802 := len_GOOSEPDU + 8;                     -- total length of the IEC 802.3 frame
                    len_svpacket <= len_GOOSEPDU + 25;

            -- Build frame and Record Information ----------------------------------------------------

                    for i in 0 to 5 loop
                        gooseframe(i) <= DEST_ADDRESS(i);                      -- Destination MAC address
                        gooseframe(6 + i) <= SOURCE_ADDRESS(i);            -- Source address (MAC Add of the device)
                    end loop;

                    for i in 0 to 3 loop
                        gooseframe(12 + i) <= IEEE_VLANtag(i);
                    end loop;
                    for i in 0 to 1 loop
                        gooseframe(16 + i) <= IEEE_ETHERTYPE(i);                            -- IEC 61850 ether-type
                        gooseframe(18 + i) <= IEEE_APPID(i);                                -- SV packets APP ID
                    end loop;
                    conv_int := std_logic_vector(to_signed(len_IEE802, 32));
                    gooseframe(20) <= conv_int(16 to 23);
                    gooseframe(21) <= conv_int(24 to 31);                                   -- Frame length

                -- Start of APDU
                    gooseframe(26) <= x"61";
                    conv_int := std_logic_vector(to_unsigned(len_GOOSEPDU, 32));
                    tempIndex := 27;                                        -- start of temp Index after savPDU tag
                    if len_GOOSEPDU > 127 then                              -- if the length is greater than 128 we need 2
bytes to hold the length value
                        if len_GOOSEPDU < 256 then                              -- if the length is a 1 byte value
                            gooseframe(tempIndex) <= x"81";                     -- length more than 128 and in need of 2
extra bytes to represent the value
                            gooseframe(tempIndex + 1) <= conv_int(24 to 31);    -- length of SV frame with 2 ASDUs
                            tempIndex := tempIndex + 2;                         -- next index
                        elsif len_GOOSEPDU > 255 then               -- if value has to be stored in a 2 byte value
                            gooseframe(tempIndex) <= x"82";
                            gooseframe(tempIndex + 1) <= conv_int(16 to 23);
                            gooseframe(tempIndex + 2) <= conv_int(24 to 31);
                            tempIndex := tempIndex + 3;
                        end if;
                    else
                        gooseframe(tempIndex) <= conv_int(24 to 31);                    -- length of goosePdu
                        tempIndex := tempIndex + 1;
                    end if;
                    GoosePDU_StartIndex <= tempIndex;
                -- Copy constants into the Record according to ASN.1
                    -- Tag 0 -
                    --this is the GOOSE Control Block
                    goosePdu.gocbRef(0) <= x"80";                           -- Goose ControlBlock - TAG 0 -
                    conv_int := std_logic_vector(to_signed(GOOSE_REF'right, 32));
                    goosePdu.gocbRef(1) <= conv_int(24 to 31);                      -- Lenght of gocbRef
                    for i in 1 to GOOSE_REF'right loop
                        goosePdu.gocbRef(1 + i) <= GOOSE_REF(i);            -- gocbRef onto the record
-- length of the data in the gocbRef
                    end loop;

                    --Tag 1-
                    -- this is the timeAlllowedtoLive for a message sent
                    goosePdu.timeAllowedtoLive(1) <= x"81";
```

```
                    goosePdu.timeAllowedtoLive(2) <= x"04";                                      -- Number of bytes
                    --Tag 2 -
                    -- Gose Data set
                    goosePdu.datSet(0) <= x"82";
                    conv_int := std_logic_vector(to_signed(len_datSet, 32));                   -- length of GOOSE dataset
                    goosePdu.datSet(1) <= conv_int(24 to 31);                                        -- length of dataset
                    for i in 1 to GOOSE_DATASET'right loop
                        goosePdu.datSet(1 + i) <= GOOSE_DATASET(i);                              -- gocbRef onto the record
-- length of the data in the gocbRef
                    end loop;

                    -- Tag 3 -
                    -- this is is the GOOSE ID
                    goosePdu.goID(0) <= x"83";
                    conv_int := std_logic_vector(to_signed(GOOSE_ID'right, 32));
                    goosePdu.goID(1) <= conv_int(24 to 31);                          -- Lenght of goose ID
                    for i in 1 to GOOSE_ID'right loop
                        goosePdu.goID(1 + i) <= GOOSE_ID(i);                         -- goID onto the record
                    end loop;
                    --Tag 4 -
                    -- UTC timestamp for the published GOOSE message
                    goosePdu.t(1) <= x"84";                                        -- Goose Timestamp values - TAG 4-
                    goosePdu.t(2) <= x"08";
                    -- Tag 5 -
                    -- This is the status number
                    goosePdu.stNum(1) <= x"85";
                    goosePdu.stNum(2) <= x"04";

                    -- Tag 6-
                    -- sequence number for the sent GOOSE message
                    goosePdu.sqNum(1) <= x"86";
                    goosePdu.sqNum(2) <= x"04";

                    -- Tag 7 -
                    -- Test Flag set to TRUE/FALSE
                    goosePdu.test(1) <= x"87";
                    goosePdu.test(2) <= x"01";
                    -- Tag 8 -
                    -- Configuaration Revision
                    goosePdu.confRev(1) <= x"88";
                    goosePdu.confRev(2) <= x"04";
```

```
    goosePdu.confRev(3) <= x"00";
    goosePdu.confRev(4) <= x"00";
    goosePdu.confRev(5) <= x"00";
    goosePdu.confRev(6) <= x"04";                                         -- configuration revision 4
    -- Tag 9 -
    -- Device needs commissioning
    goosePdu.ndsCom(1) <= x"89";
    goosePdu.ndsCom(2) <= x"01";
    -- Tag 10 -
    -- Number of data set entries
    goosePdu.numDataSetEntries(1) <= x"8A";
    goosePdu.numDataSetEntries(2) <= x"01";
    goosePdu.numDataSetEntries(3) <= x"02";                   -- 2 elements in the dataSET

    -- IMPLICIT SEQUENCE OF DATA
    goosePdu.allData(1) <= x"AB";
    goosePdu.allData(2) <= x"0D";                                     -- length of list
    goosePdu.allData(3) <= x"85";                                     -- Analogue Value using the MMXU LN
    goosePdu.allData(4) <= x"04";                                     -- INT32 value with 4 bytes
    goosePdu.allData(9) <= x"84";                                     -- XCBR.stVal.Pos
    goosePdu.allData(10) <= x"02";
    goosePdu.allData(11) <= x"06";
    goosePdu.allData(12) <= x"C0";                                    -- bad state

    ControlBlock_GOOSE <= '0';
    byteCnt <= 0;
    setup_Complete <= '1';   -- start transmitting once GOOSE control block has been configured
when readGOCB =>
    goEnable <= '0';                                                  -- Goose publishing disbled
    succ_state <= Idle;    -- stay on the transmission Idle state until we have set up the GOCB
    case (ControlBlock_GOOSE) is
        when '0' =>                               -- before data is copied onto the gooseframe
            index :=  GoosePDU_StartIndex + count;               -- start index value
            case (goPDUattr) is
                -- gocbRef --------------------------------------------------
                when b"0000" =>
                    goPDUattr <= b"0000";            -- stay
                    if byteCnt < len_GOCBREF + 2 then
                        gooseframe(index) <= goosePdu.gocbRef(byteCnt);       -- REMOVE index + count
                        byteCnt <= byteCnt + 1;                           -- count number of bytes
                        count <= count + 1;
                    else
                        byteCnt <= 0;
                        goPDUattr <= b"0001";                         -- handle next attribute
                    end if;

                -- timeAllowedtoLive ------------------------------------------------
                when b"0001" =>
                    goPDUattr <= b"0001";           -- stay
                    if byteCnt < goosePdu.timeAllowedtoLive'right  then
                        gooseframe(index) <= goosePdu.timeAllowedtoLive(byteCnt + 1);
                        if byteCnt = 2 then                   -- copy TAG and Length bytes
                            tatlIndex <= index;
                        end if;
                        byteCnt <= byteCnt + 1;                   -- count number of bytes
                        count <= count + 1;
                    else
                        byteCnt <= 0;
                        goPDUattr <= b"0010";                     -- handle next attribute
                    end if;

                -- datSet -----------------------------------------------------------------------
                when b"0010" =>
                    goPDUattr <= b"0010";           -- stay
                    if byteCnt < len_DATSET + 2 then
                        gooseframe(index) <= goosePdu.datSet(byteCnt);
                        byteCnt <= byteCnt + 1;                   -- count number of bytes
                        count <= count + 1;
                    else
                        byteCnt <= 0;
                        goPDUattr <= b"0011";                     -- handle next attribute
                    end if;

                -- goID -------------------------------------------------------------------------------
                when b"0011" =>
                    goPDUattr <= b"0011";           -- stay
                    if byteCnt < len_GOID + 2 then
                        gooseframe(index) <= goosePdu.goID(byteCnt);
                        byteCnt <= byteCnt + 1;                    -- count number of bytes
                        count <= count + 1;
                    else
```

```
            byteCnt <= 0;
            goPDUattr <= b"0100";                     -- handle next attribute
        end if;

    -- t --------------------------------------------------------------------------
    when  b"0100" =>
        goPDUattr <= b"0100";               -- stay
        if byteCnt < goosePdu.t'right then
            gooseframe(index) <= goosePdu.t(byteCnt + 1);
            if byteCnt = 2 then                         -- copy TAG and Length bytes
                timeIndex <= index;
            end if;
            byteCnt <= byteCnt + 1;                     -- count number of bytes
            count <= count + 1;
        else
            byteCnt <= 0;
            goPDUattr <= b"0101";                      -- handle next attribute
        end if;

    -- stNum  ---------------------------------------------------------------------
    when b"0101" =>
        goPDUattr <= b"0101";               -- stay
        if byteCnt < goosePdu.stNum'right then
            gooseframe(index) <=  goosePdu.stNum(byteCnt + 1);
            if byteCnt = 2 then
                stNumIndex <= index;
            end if;
            byteCnt <= byteCnt + 1;                       -- count number of bytes
            count <= count + 1;
        else
            byteCnt <= 0;
            goPDUattr <= b"0110";                        -- handle next attribute
        end if;

    -- sqNum ------------------------------------------------------------------------

    when b"0110" =>
        goPDUattr <= b"0110";               -- stay
        if byteCnt < goosePdu.sqNum'right then
            gooseframe(index) <= goosePdu.sqNum(byteCnt + 1);   -- REMOVE index + count
            if byteCnt = 2 then
```

```
                sqNumIndex <= index;
            end if;
            byteCnt <= byteCnt + 1;                         -- count number of bytes
            count <= count + 1;
        else
            byteCnt <= 0;
            goPDUattr <= b"0111";                            -- handle next attribute
        end if;

-- test ---------------------------------------------------------------------------------
when b"0111" =>
        goPDUattr <= b"0111";           -- stay
        if byteCnt < goosePdu.test'right then
            if byteCnt < 2 then                             -- copy TAG and Length bytes
                gooseframe(index) <= goosePdu.test(byteCnt + 1);
            else
                testIndex <= index;                          -- location of the Test Flag
            end if;
            byteCnt <= byteCnt + 1;                         -- count number of bytes
            count <= count + 1;
        else
            byteCnt <= 0;
            goPDUattr <= b"1000";                           -- handle next attribute
        end if;

-- confRev ------------------------------------------------------------
when b"1000" =>
        goPDUattr <= b"1000";           -- stay
        if byteCnt <  goosePdu.confRev'right then
            gooseframe(index) <= goosePdu.confRev(byteCnt + 1);
            byteCnt <= byteCnt + 1;                     -- count number of bytes
            count <= count + 1;
        else
            byteCnt <= 0;
            goPDUattr <= b"1001";                            -- handle next attribute
        end if;
-- ndsCom -----------------------------------------------
when b"1001" =>
        goPDUattr <= b"1001";           -- stay
        if byteCnt < goosePdu.ndsCom'right then
            if byteCnt < 2 then                               -- copy TAG and Length bytes
```

```
                                        gooseframe(index) <= goosePdu.ndsCom(byteCnt + 1);
                                    else
                                        ndsComIndex <= index;                           -- index of test Flag
                                    end if;
                                    byteCnt <= byteCnt + 1;                             -- count number of bytes
                                    count <= count + 1;
                                else
                                    byteCnt <= 0;
                                    goPDUattr <= b"1010";                                -- handle next attribute
                                end if;

                            -- numDataSetEntries ---------------------------------------------------------------
                            when b"1010" =>
                                goPDUattr <= b"1010";           -- stay
                                if byteCnt < goosePdu.numDataSetEntries'right then
                                    gooseframe(index) <= goosePdu.numDataSetEntries(byteCnt + 1);
                                    byteCnt <= byteCnt + 1;                               -- count number of bytes
                                    count <= count + 1;
                                else
                                    byteCnt <= 0;
                                    goPDUattr <= b"1011";
                                    allDataIndex <= index;                       -- index of the Element in the dataSet
                                end if;
                            -- all attributes copied onto the frame -------------------------------------------

                            when b"1011" =>         -- if the ASDU attribute is 110 or something not defined
                                byteCnt <= 0;
                                goPDUattr <= b"0000";
                                count <= 0;             -- go to the next index when starting off with the next ASDU
                                ControlBlock_GOOSE <= '1';                          -- enable goose publishing
                            when others =>
                                ControlBlock_GOOSE <= '1';
                        end case;
                    when '1' =>
                        goEnable <= '1';                                         -- enable the goose publisher
                        ControlBlock_GOOSE <= '0';
                    when others =>
                        goEnable <= '1';                                       -- synthesis report other=> clause not
reached
                end case;
            when Idle =>
```

```
                    succ_state <= Idle;
                    if publish = '1' then
                        succ_state <= Transmitting;                -- start sending the bytes to the process bus
                        transmit <= '1';
                    end if;
                when Transmission =>
                    succ_state <= Transmitting;                    -- stay on the Transmitting State (MAC)
                    if trans_done = '1' then    -- at the end of the transmission cycle go back to the Idle state
                        succ_state <= Idle;                        -- MAC idle state
                        count <= 0;
                        byteCnt <= 0;                             -- reset counters for Multiple ASDU handling
                    end if;
            end case;
        end if;
    end process FSM_OUTPUT;

    -------------------------------------------------------------------------
    -- Update the value of the published state number
    -------------------------------------------------------------------------
    process(stNum, clk, publish)
    begin
        if RST = '1' then
            tempState(0 to 3) <= (others=>x"00");
        elsif clk = '1' and clk'event then
            if publish = '1' then
                tempState(0) <= stNum(31 downto 24);
                tempState(1) <= stNum(23 downto 16);
                tempState(2) <= stNum(15 downto 8);
                tempState(3) <= stNum(7 downto 0);
            end if;
        end if;
    end process;

    -----------------------------------------------------------------------------
    -- Update the value of the published sequence number
    -----------------------------------------------------------------------------
    process(sqNum, publish,clk)
    begin
        if RST = '1' then
            tempSeq(0 to 3)<= (others=>x"00");
        elsif clk = '1' and clk'event then
            if publish = '1' then
                tempSeq(0) <= sqNum(31 downto 24);
                tempSeq(1) <= sqNum(23 downto 16);
                tempSeq(2) <= sqNum(15 downto 8);
                tempSeq(3) <= sqNum(7 downto 0);
            end if;
        end if;
    end process;

    -----------------------------------------------------------------------------
    -- process runs everytime the time allowed to live changes
    -----------------------------------------------------------------------------
    process(timeAllowedtoLive, publish, clk)
    begin
        if RST = '1' then
            tempTL(0 to 3) <= (others=>x"00");
        elsif clk = '1' and clk'event then
            if publish = '1' then
                tempTL(0) <= timeAllowedtoLive(31 downto 24);
                tempTL(1) <= timeAllowedtoLive(23 downto 16);
                tempTL(2) <= timeAllowedtoLive(15 downto 8);
                tempTL(3) <= timeAllowedtoLive(7 downto 0);
            end if;
        end if;
    end process;

    -----------------------------------------------------------------------------
    --process runs everytime the UTC time changes
    -----------------------------------------------------------------------------
    process(UTCTimestamp, publish, clk)
    begin
        if RST = '1' then
            tempUTC(0 to 6) <= (others=>x"00");
        elsif clk = '1' and clk'event then
            if publish = '1' then
                tempUTC(0) <= UTCTimestamp(0 to 7);
                tempUTC(1) <= UTCTimestamp(8 to 15);
                tempUTC(2) <= UTCTimestamp(16 to 23);
                tempUTC(3) <= UTCTimestamp(24 to 31);                         -- SecondsSinceEpoch
                tempUTC(4) <= UTCTimestamp(32 to 39);
                tempUTC(5) <= UTCTimestamp(40 to 47);
                tempUTC(6) <= UTCTimestamp(48 to 55);
            end if;
        end if;
    end process;

    ----------------------------------------------------------------------------
    -- Runs when the value of Vrms or the state of the boolean changes
    ----------------------------------------------------------------------------
    ALLDATA : process (clk, publish)
    begin
        if RST = '1' then
            element(0 to 11) <= (others=>x"00");
        elsif clk = '1' and clk'event then
            if publish = '1' then   -- if there is a status change in one of the elements in the dataset then
                element(0) <= x"AB";
                element(1) <= x"0A";                                             -- length of list
                element(2) <= x"85";                                             -- Analogue Value using the MMXU LN
                element(3) <= x"04";                                             -- INT32 value with 4 bytes

                element(4) <= VArms(31 downto 24);
                element(5) <= VArms(23 downto 16);
                element(6) <= VArms(15 downto 8);
                element(7) <= VArms(7 downto 0);

                -- start the bit string for breaker position
                element(8) <= x"84";
                element(9) <= x"02";
                element(10) <= x"06";                   -- padding
                if Boolin = '1' then
                    element(11) <= x"80";               -- ON
                else
                    element(11) <= x"40";               -- OFF
                end if;
            end if;
        end if;
    end process ALLDATA;

    --------------------------------------------------------------------------------
    -- check whether the BOOLEAN state has changed
    --------------------------------------------------------------------------------
    EVENT_TRIG_BOOL : process(boolIn, clk)
```

```
begin
    if RST = '1' then
        BoolChange <= '0';
        prevDigitalState <= '0';
    elsif clk = '1' and clk'event then
        prevDigitalState <= boolIn;
        BoolChange <= '0';                                            -- change in
        if boolIn /= prevDigitalState then            -- if the current state of SW2 is equal to its previous state
            BoolChange <= '1';
        end if;
    end if;
end process EVENT_TRIG_BOOL;

-----------------------------------------------------------------------------------
-- if the Boolean or the Analog Event changed then Trigger should be HIGH else LOW
-- This process drives the publisher signal of the GOOSE FSM
-----------------------------------------------------------------------------------
TRIGGER_EVENT : process(BoolChange, AnaChange, clk)
begin
    if RST = '1' then
        TriggerEvent <= '0';
    elsif clk = '1' and clk'event then
        TriggerEvent <= '0';
        if BoolChange = '1' or AnaChange = '1' then
            TriggerEvent <= '1';                        -- trigger event is HIGH then publish GOOSE Message
        end if;
    end if;
end process TRIGGER_EVENT;

---------------------------------------------------------------------------------------
-- Process detects a change in the input signals from the AFE and generates
-- sample signal to force the publisher to the transmit state
-- on LOW-HIGH transition of sample signal sample values are generated
--------------------------------------------------------------------------------------
MAP_SV : process(SampleCnt, RST, clk)
variable curr_cnt, prevVal : signed(11 downto 0) := x"000";
begin
    if RST = '1' then
        curr_cnt := x"000";
        prevVal := x"000";
        newSample <= '0';
        elsif clk = '1' and clk'event then                                          -- synchronous
            curr_cnt := signed(SampleCnt);
            prevVal := signed(smpCount);
            smpCount <= SampleCnt;
            newSample <= '0';
            if curr_cnt /= prevVal then
                newSample <= '1';
            end if;
        end if;
    end process;

end Behavioral;
```

Appendix E.2 MAC2PHY4IR.vhd

```
--------------------------------------------------------------------------
--This file defines an entity that simplifies the MII interface
--with an Ethernet Physical layer. It particularly fits with the
--Nexys3 development board from Digilent.
--The entity have several types of ports:
--General purposes (RSTN)
--Physical ports to manage the communication with the MII Interface on
--the physical layer side
--Mac ports to communicate with MAC layer
------------------------------------------------------------------------
--IMPORTANT:
--Because the physical interface has two clocks domains (RXCLK, TXCLK)
--This entity uses a FIFO component specific to the SPARTAN6 FPGA of NEXYS3 board
--As a result, the entity provide only one clock signal (CLK) to the mac layer.

---------------------------------------------------------------------------
--MAC PORTS/INTERFACE COMMUNICATION:
-- COLLISION is set to '1' by the entity when the PHY layer has detected a collision
-- CARRIER is set to '1' by the entity when the PHY layer has detected a carrier
-- CLK clock given to the mac layer, based on the TXCLK of the PHY layer (25Mhz)
-- TDATA: 8 bits std_logic_vector that the MAC layer must provide to be transmitted
-- TDATA value must be hold for 80 ns (2 periods clock)
-- TXVALID: A pulse of 40ns (1 period clock) must be hold when valid data is available on TDATA
-- RDATA: 8 bits std_logic_vector that the entity provide to the mac layer
-- its value changes every 40ns (1 period clock). The mac layer must look for
-- the SFD by checking every clock rising edge. Once the SFD is found, the
-- MAC layer only needs to check every two clock rising edges to have a valid value.
-- RXVALID: The entity set this flag to '1' when the receiver is synchronized, (ie) SFD is
--              about to be visible on RDATA.
---------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity mac2phy4IR is
    Port (
           -- General purpose ports
           RSTN : in  STD_LOGIC;
           --PHY ports
           --General purpose
           MDIO : inout  STD_LOGIC;
           MDC : out  STD_LOGIC;
           RESETN : out  STD_LOGIC;
           COL : in  STD_LOGIC;
           CRS : in  STD_LOGIC;
           --TX
           TXD :out  STD_LOGIC_VECTOR (3 downto 0);
           TXER : out  STD_LOGIC;
           TXEN: out STD_LOGIC;
           TXCLK : in  STD_LOGIC;
           --RX
           RXD : in  STD_LOGIC_VECTOR (3 downto 0);
           RXER : in  STD_LOGIC;
           RXDV : in  STD_LOGIC;
           RXCLK : in  STD_LOGIC;
           --MAC Ports
           --General purpose
           COLLISION: out STD_LOGIC;
           CARRIER: out std_logic;
           CLK: out STD_LOGIC;
           --TX
           TDATA: in STD_LOGIC_VECTOR (7 downto 0);
           TXVALID: in STD_LOGIC;
           --RX
           RDATA : out STD_LOGIC_VECTOR (7 downto 0);
           RXVALID: out STD_LOGIC);
end mac2phy4IR;
```

```
architecture Behavioral of mac2phy4IR is
    signal first_nibble: Boolean :=true;
    signal nTXD : std_logic_vector(3 downto 0) :=(others=>'0');
    signal RXDbuf: std_logic_vector(7 downto 0):=(others=>'0');
    signal noDATA: std_logic:='1';
    signal fifoOUT : std_logic_vector(3 downto 0);
    signal rst : std_logic:='1';
    signal RXen : std_logic:='0';

    component fifo_core is
        Port (
         rst : IN STD_LOGIC;
         wr_clk : IN STD_LOGIC;
         rd_clk : IN STD_LOGIC;
         din : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
         wr_en : IN STD_LOGIC;
         rd_en : IN STD_LOGIC;
         dout : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
         full : OUT STD_LOGIC;
         empty : OUT STD_LOGIC);
    end component;
begin

    -- General purpose signals
    RESETN<=RSTN;
    COLLISION<=COL;
    CARRIER<=CRS;
    CLK<=TXCLK;
    --Pre cabled signals
    MDIO     <='1';
    MDC  <='0';
    TXER     <='0';
    --Rx signals
    RDATA<= RXDbuf;
    RXVALID<=RXDV;
    --fifo signals
    RXen<= not noDATA;
    rst<= not RSTN;
    fifoRX : fifo_core port map (rst,RXCLK,TXCLK,RXD,RXDV,RXen,fifoOUT,open,noDATA);            -- positional mapping of
signals in fifo_core to the local signals
    --Interfacing process
    process(TXCLK,RSTN) is begin
        if (RSTN='0') then
                first_nibble<=true;
                TXD<=(others=>'0');
                RXDbuf<= (others=>'0');
        elsif (rising_edge(TXCLK)) then
                RXDbuf<= RXDbuf(3 downto 0) & fifoOUT;     -- receive a nibble
                TXEN    <= TXVALID;
                if first_nibble and TXVALID='1' then
                    TXD<= TDATA(7 downto 4);
                    nTXD<= TDATA(3 downto 0);
                    first_nibble<=false;
                elsif first_nibble=false then
                    TXD<=nTXD;
                    first_nibble<=true;
                end if;
        end if;
    end process;
end Behavioral;
```

Appendix E.3 publishSigGen.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_signed.all;
entity publishSigGen is
    Port ( clk : in  STD_LOGIC;
              goEnable : in STD_LOGIC;
           publishSig : out  STD_LOGIC;
              timeAllowedtoLive : out STD_LOGIC_VECTOR(31 downto 0);                         -- time allowed to Live
              stateNum : out STD_LOGIC_VECTOR(31 downto 0);
              seqNum : out STD_LOGIC_VECTOR(31 downto 0);
           TriggerEvent : in  STD_LOGIC;
              millisPulse : out STD_LOGIC);
end publishSigGen;

architecture Behavioral of publishSigGen is
    constant MaxTime: integer := 1000;
    constant MinTime: integer := 100;                                             -- GOOSE message publish max
and Min Times
    signal stNum : integer := 1;
    signal sqNum, stepInt : integer := 0;
    signal afterEvent, msPulse : STD_LOGIC := '0';
begin

    stateNum <=  std_logic_vector(to_unsigned(stNum, 32));
    seqNum  <=  std_logic_vector(to_unsigned(sqNum, 32));

   ---------------------------------------------------------------------------
    -- Process : PULSE_GEN
    -- This process generates a 1 ms pulse from the SYS_CLK
    ---------------------------------------------------------------------------
    PULSE_GEN : process (clk)
    variable pulseCount : integer range 0 to 110000:= 0;
    begin
        if clk'event and clk = '1' then                          -- synchronous FF
            if goEnable = '1' then      --Enable--      -- only generate 1 ms pulse when Goose Control block is enabled
                pulseCount := pulseCount + 1;
                msPulse <= '0';                                  -- Pulse HIGH for 10ns and always OFF
                if pulseCount >= 100000 then                                            -- 100 000 x 10ns equal to 1ms
                    msPulse <= '1';
                    pulseCount := 0;
                end if;
            end if;
        end if;
    end process PULSE_GEN;

    millisPulse <= msPulse;
    ------------------------------------------------------------------------------------------
    -- Process : PUBLISH
    -- Publishes GOOSE messages after after MaxTime has expired or after MinTime has expired
    -- before or after an event respectively
    ------------------------------------------------------------------------------------------

    PUBLISH : process (clk, goEnable, msPulse, TriggerEvent)
    variable timeCount : integer := 0;
    variable currPubRate : integer := 0;
    begin                                                            -- reset stNum and sqNum
```

```
        if clk'event and clk = '1' then                                  -- if there is a rising edge on the clock signal
then
            publishSig <= '0';
            if TriggerEvent = '1' then                                   -- SET --
                publishSig <= '1';                                       -- publish new value immediately
                afterEvent <= '1';                                      -- set Flag after value of data set changed
                currPubRate := MinTime;                                 -- start at the minimum time
                timeCount := 0;
                stNum <= stNum + 1;
                sqNum <= 0;
            else
                if goEnable = '1' then                         -- only generate a publish timer when the Goose is enabled
                    if msPulse = '1' then                                  -- after 1ms --
                        timeCount := timeCount + 1;
                        if afterEvent = '1' then
                            if timeCount >= currPubRate then
                                timeCount := 0;
                                sqNum <= sqNum + 1;                   -- sequence counter for the message sent within the
the same stNum
                                currPubRate := currPubRate + 200;     -- add 200ms to the current publishing rate
                                publishSig <= '1';                        -- publish
                                if currPubRate >= MaxTime then            -- when the normal rate of publishing has been
reached then
                                    afterEvent <= '0';
                                end if;
                            end if;
                        else
                            if timeCount >= MaxTime then
                                sqNum <= sqNum + 1;                     -- next state
                                currPubRate := MaxTime;                 -- make timeAllowedtoLive to MaxTime
                                timeCount := 0;
                                publishSig <= '1';
                            end if;
                        end if;
                    end if;
                end if;
            end if;
            timeAllowedtoLive <= std_logic_vector(to_unsigned(currPubRate, 32));             -- convert the time allowed
to live and send it with the GOOSE Message
        end if;
    end process PUBLISH;
end Behavioral;
```

Appendix E.4 sigSynchronizer.vhd

```
entity sigSynchronizer is
    Port ( clk : in  STD_LOGIC;
              enter : in STD_LOGIC;
              transmit : in STD_LOGIC;
              publish : in STD_LOGIC;
              trans_klaar : in STD_LOGIC;
              transmit_rise : out STD_LOGIC;
              sample_rise : out STD_LOGIC;
              enter_rise : out STD_LOGIC;
              trans_done : out STD_LOGIC);
end sigSynchronizer;

architecture Behavioral of sigSynchronizer is

begin
    SIGNAL_SYNCH : process(clk)
        variable enter_sync : std_logic_vector(1 to 3);                                    -- temp signals for
synchronization
        variable transmit_sync : std_logic_vector(1 to 3);
        variable sample_sync : std_logic_vector(1 to 3);
        variable trans_sync : std_logic_vector(1 to 3);

        begin
        if rising_edge(clk) then
            enter_rise <= enter_sync(2) and not enter_sync(3);
            transmit_rise <= transmit_sync(2) and not transmit_sync(3);
            sample_rise <= sample_sync(2) and not sample_sync(3);
            trans_done <= trans_sync(2) and not trans_sync(3);

            enter_sync := enter & enter_sync(1 to 2);
            transmit_sync := transmit & transmit_sync(1 to 2);
            sample_sync := publish & sample_sync(1 to 2);
            trans_sync := trans_klaar & trans_sync(1 to 2);
        end if;
    end process SIGNAL_SYNCH;
end Behavioral;
```

Appendix E.5 utcTime.vhd

```
entity utcTime is
    Port ( clk : in  STD_LOGIC;
              msPulse : in STD_LOGIC;
              UTCTimestamp : out STD_LOGIC_VECTOR(0 to 55));
end utcTime;

architecture Behavioral of utcTime is

begin
    UTC_TIME : process (clk)
        variable FractionofSeconds : integer := 0;
        variable SecondsSinceEpoch : integer := 1430001415;
    begin
        if clk'event and clk = '1' then                                   -- Synchronous Flip-flop
            if msPulse = '1' then       -- Enable --
                FractionofSeconds := FractionofSeconds + 16777;           -- fraction of a seconds resolution of 1ms
                if FractionofSeconds >= 16777000 then
                    FractionofSeconds := 0;
                    SecondsSinceEpoch := SecondsSinceEpoch + 1;           -- next second
                end if;
            end if;
        end if;
        UTCTimestamp(0 to 31) <= std_logic_vector(to_signed(SecondsSinceEpoch, 32));   -- Seconds since 1 January since
1970 @ 00:00:00
        UTCTimestamp(32 to 55) <= std_logic_vector(to_signed(FractionofSeconds, 24));   -- Fraction of a second from 1-
2^24 within 1 second
    end process UTC_TIME;
end Behavioral;
```

Appendix E.6 btn_deb.vhd

```
entity btn_deb is
    Port ( clk : in  STD_LOGIC;
              nds : in STD_LOGIC;
              test : in STD_LOGIC;
              inBool : in STD_LOGIC;
              ndsCom : out STD_LOGIC;
```

```
                testFlag : out STD_LOGIC;
                boolIn : out STD_LOGIC);
end btn_deb;

architecture Behavioral of btn_deb is
    Signal ndsCom_delay1, ndsCom_delay2, ndsCom_delay3 : STD_LOGIC;
    Signal test_delay1, test_delay2, test_delay3 : STD_LOGIC;
    Signal in1_delay1, in1_delay2, in1_delay3 : STD_LOGIC;
begin
    process (clk, inBool)
        begin
            if clk'event and clk = '1' then
                ndsCom_delay1 <= nds;                                          -- needs Commissioning
                ndsCom_delay2 <= ndsCom_delay1 ;
                ndsCom_delay3 <= ndsCom_delay2 ;

                test_delay1 <= test;                                               -- test Flag
                test_delay2 <= test_delay1 ;
                test_delay3 <= test_delay2 ;

                in1_delay1 <= inBool;                                          -- Input Boolean value
                in1_delay2 <= in1_delay1 ;
                in1_delay3 <= in1_delay2 ;
            end if;
    end process;
    ndsCom <= ndsCom_delay1 and ndsCom_delay2 and ndsCom_delay3;            -- debounced signal for ndsCom
    testFlag <= test_delay1 and test_delay2 and test_delay3;
    boolIn <= in1_delay1 and in1_delay2 and in1_delay3;
end Behavioral;
```

Appendix E.7 ethernet_frame.vhd

```
library ieee_proposed;
use ieee_proposed.float_pkg.all;
use ieee_proposed.fixed_float_types.all;

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.numeric_std.all;
```

```
package ethernet_frame is
    type byte_array is array (integer range <> ) of std_logic_vector(7 downto 0);

    -- Constants

    constant DEST_ADDRESS: byte_array (0 to 5) := (x"01", x"0C", x"CD", x"01", x"00", x"01");
-- IEC 61850-8-1 MAC Addresses defined by IEEE
    constant SOURCE_ADDRESS: byte_array (0 to 5) := (x"00", x"00", x"0A", x"0A", x"01", x"01");
    constant IEEE_VLANtag: byte_array (0 to 3) := (x"81", x"00", x"80", x"01");
-- Constant declaration of VLAN Tag
    constant IEEE_ETHERTYPE: byte_array (0 to 1) := (x"88", x"B8");
-- Constant declaration of the Ethertype
    constant IEEE_APPID: byte_array (0 to 1) := (x"00", x"04");
-- Constant declaration of the Ethertype
    constant GOOSE_ID: byte_array (1 to 7) := (x"47",x"4F",x"4F",x"53",x"45",x"49",x"44");                          --
Goose ID -- GOOSEID --
    constant GOOSE_REF: byte_array (1 to 30) :=
(x"46",x"50",x"47",x"41",x"2F",x"49",x"45",x"44",x"31",x"2F",x"4C",x"4C",x"4E",x"30", x"24",x"47",
x"4F",x"24",x"47",x"53",x"45",x"5F",x"43",x"42",x"5F",x"47",x"4F",x"4F",x"53",x"45");                -- GooseCBRef
FPGA/IED1/LLN0$GO$GSE_CB_GOOSE
    constant GOOSE_DATASET: byte_array (1 to 25) :=
(x"46",x"50",x"47",x"41",x"2F",x"49",x"45",x"44",x"31",x"2F",x"4C",x"4C",x"4E",x"30", x"24",x"47", x"4F",x"4F",
x"53",x"45",x"5F",x"45", x"76",x"61", x"6C");                -- datSET     FPGA/IED1/LLN0$GOOSE_Eval


    -- Goose PDU aacording to IEC 61850-8-1

    type APDU is
    record
        gocbRef : byte_array(integer range 0 to 31);                              -- field - 0x80 | Length | VISIBLE STRING129
|-
        timeAllowedtoLive: byte_array(integer range 1 to 6);              -- 4 byte field - 0x81 | Length | Time in ms
        datSet : byte_array(integer range 0 to 30);                       -- field - 0x82 | Length | VISIBLE STRING64 |-
        goID : byte_array(integer range 0 to 15);                      -- field - 0x83 | Length | VISIBLE STRING64 |-
        t : byte_array(integer range 1 to 10);                    -- 10 byte field - 0x84 | 0x08 | TIMESTAMP UTC |-
        stNum : byte_array (integer range 1 to 6);                  -- 3 byte field - 0x85 | Length | Status Number
        sqNum : byte_array (integer range 1 to 6);                  -- 3 byte field - 0x86 | Length | Sequence Number
        test : byte_array (integer range 1 to 3);                     -- 3 byte field - 0x87 | Length | TEST CASE :
TRUE/FALSE
        confRev : byte_array( integer range 1 to 6);                -- 3 byte field - 0x88 | Length | Configuration Revision
        ndsCom : byte_array (integer range 1 to 3);                 -- 3 byte field - 0x89 | Length | Needs Comm TRUE/FALSE
        numDataSetEntries : byte_array(integer range 1 to 3);   -- 3 byte field - 0x8A | Length | Num of DATASET entries
        allData : byte_array(integer range 1 to 12);
    end record APDU;

    function Calc_VOLTS  (signal Sample : in std_logic_vector(23 downto 0)) return unsigned;
    function  unsigned_sqrt ( d : UNSIGNED ) return UNSIGNED;
end ethernet_frame;

package body ethernet_frame is
    ---------------------------------------------------------------------------------------------------
    -- Calculate Volts sample value using 24 bit input, Ref (Maximum Defelction), input current
    ---------------------------------------------------------------------------------------------------
    function Calc_VOLTS  (signal Sample : in std_logic_vector(23 downto 0)) return unsigned is
        variable TVTR : unsigned(31 downto 0);
        variable signedVal : signed(23 downto 0);
        variable Ref, tvtr_mag,inTVTR, sqVolt : float(8 downto -23);
        begin
            signedVal := signed(Sample);
            inTVTR := to_float(signedVal, inTVTR);
--          Ref := to_float(0.00147536272, Ref);                                  -- (Voltage)/Max_Pos_Deflection
(12376.23804V/8388607) 12376.23804 V is the maximum voltage
            Ref := to_float(0.00146151345, Ref);
            tvtr_mag := inTVTR * Ref;                                                 -- calculate feeder current
            sqVolt := tvtr_mag * tvtr_mag;
            TVTR := to_unsigned(sqVolt, 32);                                            -- return the scaled value
        return TVTR;
    end Calc_VOLTS;
    ---------------------------------------------------------------------------------------------------
    -- Calculate the square root of an 32 bit unsigned value and return a 16 bit value
    ---------------------------------------------------------------------------------------------------

    function  unsigned_sqrt ( d : UNSIGNED ) return UNSIGNED is
    variable a : unsigned(31 downto 0):= d;  --original input.
    variable q : unsigned(15 downto 0):=(others => '0');  --result.
    variable left,right,r : unsigned(17 downto 0):=(others => '0');  --input to adder/sub.r-remainder.
    variable i : integer:=0;

    begin
        for i in 0 to 15 loop
            right(0):='1';
            right(1):=r(17);
```

```
                right(17 downto 2):=q;
                left(1 downto 0):=a(31 downto 30);
                left(17 downto 2):=r(15 downto 0);
                a(31 downto 2):=a(29 downto 0);  --shifting by 2 bit.
                if ( r(17) = '1') then
                    r := left + right;
                else
                    r := left - right;
                end if;
                q(15 downto 1) := q(14 downto 0);
                q(0) := not r(17);
            end loop;
            return q;
        end unsigned_sqrt;
    end ethernet_frame;
```

Appendix E.8 Calculations.vhd

```
    library ieee_proposed;
    use ieee_proposed.float_pkg.all;
    use ieee_proposed.fixed_float_types.all;

    library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;
    use work.ethernet_frame.all;                              -- using package

    entity Calculations is
        Port ( VinA : in  STD_LOGIC_VECTOR (23 downto 0);
                  SampleCount : in std_logic_vector(11 downto 0);
               VArms : out  STD_LOGIC_VECTOR (31 downto 0);
                  AnaChange : out std_logic;
                  newSample : in std_logic;
                  RST : in std_logic;
               clk : in  STD_LOGIC);
    end Calculations;

    architecture Behavioral of Calculations is

        signal VRMS : unsigned( 47 downto 0);
```

```
signal prevVRMS : signed( 15 downto 0);
signal newRMS : std_logic := '0';
signal SumVolt : signed(31 downto 0) := x"00000000";
signal calc_RMS : std_logic := '0';
signal ValueChange : std_logic := '0';
signal Root : std_logic_vector(15 downto 0);

begin
-------------------------------------------------------------------------
-- Trigger message publishing when the dead bad is exceeded
-------------------------------------------------------------------------
ANALOG_CHANGE : process(clk, ValueChange, Root)
    variable db : signed(15 downto 0);
    variable diff : signed(15 downto 0);
    variable vrms : signed(15 downto 0);
    begin
        if clk = '1' and clk'event then
            AnaChange <= '0';
            if ValueChange = '1' then
                db := x"000A";                                  -- set deadband to 20V
                vrms := signed(Root);
                prevVRMS <= vrms;
                diff := abs(vrms - prevVRMS);
                if diff > db then                               -- dead band violated
                    AnaChange <= '1';                           -- signal the change in the Analogue signal
                    VArms(15 downto 0) <= Root(15 downto 0);    -- only assign whn deadband violated
                end if;
            end if;
        end if;
    end process;

----------------------------------------------------------------------
-- SQ_VOLT_CALC
-- calculate the mean square voltage using ADC code over a full cycle
----------------------------------------------------------------------
SQ_VOLT_CALC : process(clk, newSample, VinA, SampleCount)
    variable ordinates, Vin_sq : unsigned( 31 downto 0) := x"00000000";
    variable Sum_VinSq : unsigned( 47 downto 0) := x"000000000000";
    begin
        if clk = '1' and clk'event then
            newRMS <= '0';
```

```
                if newSample = '1' then        -- only calculate the square of the input voltage whenever a new sample
is received
                    ordinates := ordinates + 1;          -- count number of ordinates for the mean square value
                    Vin_sq := Calc_VOLTS(VinA);
                    Sum_VinSq := Sum_VinSq + Vin_sq;
                    if ordinates = x"00000F9F" then              -- if this is the last sample of the cycle 0-3999
                        newRMS <= '1';
                        VRMS <= Sum_VinSq/ordinates;
                        Sum_VinSq := x"000000000000";
                        ordinates := x"00000000";                -- clear the variables and calculate the rms value
                    end if;
                end if;
            end if;
        end process;

    ------------------------------------------------------------------------
    -- Calculate the square root of the mean square value
    -----------------------------------------------------------------------
    RET_RMS : process( clk, newRMS, VRMS)
    variable tempVal : unsigned(15 downto 0);
    begin
        if clk = '1' and clk'event then
            ValueChange <= '0';
            if newRMS = '1' then
                tempVal := unsigned_sqrt(VRMS(31 downto 0));
                Root <=     std_logic_vector(tempVal);
                ValueChange <= '1';
            end if;
        end if;
    end process;

end Behavioral;
```

Appendix E.9 ethcrc32.vhd

```
    ---------------------------------------------------------------------------------
    --This file defines an entity that compute a CRC for 802.3 procotol
    --The computation is done by group of 8 bits.
    --The reset is asynchronous.
    --Computation are synchronous to clk and enabled when 'en' equals 1
```

```
--data_in is an 8bits input port for data to consider for a crc computation
--is_msb equals '1' indicates that data_in need to be reverted (msb to lsb) before computation.
------------------------------------------------------------------------------
--IMPORTANT :
--To be used in a real Ethernet case, user must take considerations
--which are:
--1) The four first byte must be Ones' complemented
--2) data_in only considers some fields (ie not the preamble nor the SFD, see norm)
--3) data_in must be given in the same sens than transmission:
--      * LSB first for data
--      * MSB first for CRC in the case of reception
--      * ALL zero for CRC in the case of transmission
--4a) crc_out must be complemented before transmission
--4b) received crc must be complemented before given to data_in for
--    error checking in reception
--5) In the case of error checking in reception, if the received frame is correct
-- crc_out equals x"00000000"
------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity ethcrc32 is
    port (clk: in STD_LOGIC;                    -- Input clock
         rst: in STD_LOGIC;                 -- Asynchronous reset
         en : in STD_LOGIC;             -- Assert to compute calculations
         is_msb: in STD_LOGIC;              -- Assert to indicate the sens of data_in
           data_in: in STD_LOGIC_VECTOR(7 downto 0);         -- Data to compute
         crc_out: out STD_LOGIC_VECTOR (31 downto 0)     -- CRC output
    );
end ethcrc32;
------------------------------------------------------------------------------
architecture nano of ethcrc32 is
-- The Generator polynomial is
--  32   26   23   22   16   12   11   10   8   7   5   4   2
-- x   + x  + x  + x  + x  + x  + x  + x  + x + x + x + x + x + 1
constant GENERATOR : STD_LOGIC_VECTOR := X"04C11DB7";
begin
    process (clk,rst) is
        variable crc_buf : STD_LOGIC_VECTOR (31 downto 0):=x"00000000";
    begin
```

```
        if rst = '1' then   -- reset signals to values
            crc_buf := (others => '0');
        elsif rising_edge(clk) then  -- operate on positive edge
            if (en='1') then
                if is_msb='1' then
                        for I in data_in'reverse_range loop
                            crc_buf := (crc_buf(30 downto 0) & data_in(I)) XOR (GENERATOR AND (0 to 31=>crc_buf(31)));
                        end loop;
                else
                        for I in data_in'range loop
                            crc_buf := (crc_buf(30 downto 0) & data_in(I)) XOR (GENERATOR AND (0 to 31=>crc_buf(31)));
                        end loop;
                end if;
            end if;
        end if;
        crc_out<=crc_buf;
    end process;
end nano;
```

APPENDIX F. IEC 61850-9-2LE SV message mapping VHDL module

Appendix F.1 hello_eth.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.ethernet_frame.all;                             -- using package

entity hello_eth is
    Port (    -- General purpose ports
                  RST : in  std_logic;
                  NUM : in STD_LOGIC_VECTOR(7 downto 0);
                  --PHY ports
                  --General purpose
                  MDIO : inout  std_logic;
                  MDC : out  std_logic;
                  RESETN : out  std_logic;
                  COL : in  std_logic;
                  CRS : in  std_logic;
                  --TX
                  TXD :out  std_logic_vector (3 downto 0);
                  TXER : out  std_logic;
                  TXEN: out std_logic;
                  TXCLK : in  STD_LOGIC;
                  -- Samples
                  sampleCTVT : in std_logic_vector(215 downto 0);
                  SampleCnt : in std_logic_vector(11 downto 0);
                  --RX
                  RXD : in  std_logic_vector (3 downto 0);
                  RXER : in  std_logic;
                  RXDV : in  std_logic;
                  RXCLK : in  std_logic;
                  clk : in std_logic);
end hello_eth;
```

```
architecture Behavioral of hello_eth is
    --Constants and type

--  Type possible_state is (Idle,Receiving,Decoding_crc,Transmitting);
Type possible_state is (Idle,Transmitting);

    --Components
    component ethcrc32 is
      port (    clk         : in std_logic;
                rst     : in std_logic;
                en          : in std_logic;
                is_msb  : in std_logic;
                data_in     : in std_logic_vector (7 downto 0);
                crc_out     : out std_logic_vector (31 downto 0));
    end component;

    component mac2phy4IR is
         Port (
                  -- General purpose ports
                  RSTN : in  std_logic;
                  --PHY ports
                  --General purpose
                  MDIO : inout  std_logic;
                  MDC : out  std_logic;
                  RESETN : out  std_logic;
                  COL : in  std_logic;
                  CRS : in  std_logic;
                  --TX
                  TXD :out  std_logic_vector (3 downto 0);
                  TXER : out  std_logic;
                  TXEN: out std_logic;
                  TXCLK : in  STD_LOGIC;
                  --RX
                  RXD : in  std_logic_vector (3 downto 0);
                  RXER : in  std_logic;
                  RXDV : in  std_logic;
                  RXCLK : in  std_logic;
                  --MAC Ports
                  --General purpose
                  COLLISION: out std_logic;
```

```
                    CARRIER: out std_logic;
                    CLK : out std_logic;
                    --TX
                    TDATA: in std_logic_vector (7 downto 0);
                    TXVALID: in std_logic;
                    --RX
                    RDATA : out std_logic_vector (7 downto 0);
                    RXVALID: out std_logic);
    end component;

    --GP signals
    signal sv_frame : byte_array(0 to 70) :=(others=>x"00");                           -- my frame;
    signal not_reset: std_logic:='1';
    signal local_clk: std_logic:='0';

    --Counter of transmitted/received bytes
    signal byte_num: integer range 0 to 150 := 0;                               -- Maximum length of Ethernet frame is 1521
    --Keep the state of the entity
    signal state, succ_state : possible_state:=Idle;

    --Shit register to bufferize the received data for crc computation purpose
--  signal data_received : byte_array(3 downto 0):=(others=>x"00");

    --Received frame counter
--  signal nb_received_frame: integer range 0 to 15 :=0;
--  signal nb_error_frame: integer range 0 to 15 :=0;
    signal enter_deb : std_logic := '0';
    signal trans_klaar, trans_done : std_logic := '0';                                    -- done transmitting data
    signal  svEnable : std_logic := '0';
    signal asdu_attr : std_logic_vector(0 to 2) := b"000";
    type state_machine is (setup, sampling, readSVCB, Transmission);
    signal curr_state, next_state : state_machine := setup;                           -- state machine variables
    signal byteCnt : integer := 0;
    signal sample : std_logic := '0';                                                        -- these flag used when setting
up Record at start up
    signal no_asdu : integer range 0 to 2 := 0;                                    -- asdu counters (maximum number of ASDU is
8)
--  signal NumSampleCycle : integer range 0 to 10000 := 0;                            -- Number of samples in a seconds

    signal apdu : APDU;
    signal enter_rise : std_logic := '0';
    signal sample_rise : std_logic := '0';
    signal transmit_rise : std_logic := '0';
    signal transmit : std_logic := '0';
--  signal COUNT_VAL :integer range 0 to 500000 := 0;
    signal ASDU_START_INDEX, SampleIndex: integer range 0 to 100 := 0;              -- ASDU starts at 26 + number of
bytes for (savPDU, noASDU and seqASDUs)
    signal len_svpacket, count : integer range 0 to 150:= 0;       -- maximum length of Ethernet packet ( IEC 61850-9-2)

    -- Constants ---------------------------------
    constant PREAMBLE_SOF_LEN : integer := 8;                               -- total combined length of preamble and SOF
    constant SAV_CDC_SIZE : integer := 64;                                  -- SAV CDC made up of 64 bytes
    constant NO_ASDUS : integer := 1;                                       -- Integer made up of 4 bytes
    constant DEST_ADDRESS: byte_array (0 to 5) := (x"01", x"0C", x"CD", x"04", x"00", x"01");
-- IEC 61850-9-2 MAC Addresses defined by IEEE
    constant SOURCE_ADDRESS: byte_array (0 to 5) := (x"00", x"00", x"0A", x"0A", x"01", x"01");
    constant IEEE_VLANtag: byte_array (0 to 3) := (x"81", x"00", x"80", x"00");     -- Constant declaration of VLAN Tag
    constant IEEE_ETHERTYPE: byte_array (0 to 1) := (x"88", x"BA");      -- Constant declaration of the Ethertype
    constant IEEE_APPID: byte_array (0 to 1) := (x"40", x"00");          -- Constant declaration of the Ethertype
    constant SV_ID: byte_array (0 to 16) :=
(x"4E",x"63",x"75",x"62",x"65",x"5F",x"4D",x"55",x"63",x"70",x"75",x"74",x"30",x"30",x"30",x"31",x"00");    -- svID is
Ncube_MUcput0001\0

    --crc_core signals
    signal crc: byte_array(0 to 3):=(others=>x"00");
    signal crc_crc_out: std_logic_vector(31 downto 0);
    signal crc_data_in : std_logic_vector(7 downto 0):=x"00";
    signal crc_en: std_logic:='0';
    signal crc_is_msb: std_logic:='1';
    signal crc_rst: std_logic:='1';
    signal crc_clk: std_logic:='0';

    -- sample values per channel
    signal TCTRA, TCTRB, TCTRC, TCTRN, TVTRA, TVTRB, TVTRC, TVTRN : std_logic_vector(31 downto 0):=x"00000000";-- CDC
Values
    signal qualTCTRA, qualTCTRB, qualTCTRC, qualTCTRN, qualTVTRA, qualTVTRB, qualTVTRC, qualTVTRN : std_logic_vector(15
downto 0);
    signal cdcSamples : byte_array(0 to 63);                             -- data for all the 8 channels
    --mac2phy4IR signals
    signal  m2p_COLLISION:  STD_LOGIC;
    signal  m2p_CARRIER:  STD_LOGIC;
    signal  m2p_CLK :  STD_LOGIC;
    signal  m2p_TDATA:  STD_LOGIC_VECTOR (7 downto 0):=x"00";
    signal  m2p_TXVALID:  STD_LOGIC:='0';
    signal  m2p_RDATA :  STD_LOGIC_VECTOR (7 downto 0);
    signal  m2p_RXVALID:  STD_LOGIC;
    signal smpCount : std_logic_vector(11 downto 0);
    signal setup_Complete : std_logic := '0';

     component sigSynchronizer is
    Port ( clk : in  STD_LOGIC;
              enter : in STD_LOGIC;
              transmit : in STD_LOGIC;
              sample : in STD_LOGIC;
              trans_klaar : in STD_LOGIC;
              transmit_rise : out STD_LOGIC;
              sample_rise : out STD_LOGIC;
              enter_rise : out STD_LOGIC;
              trans_done : out STD_LOGIC;
           RST : in  STD_LOGIC);
    end component;
    ---------------------------------------------------

begin
    -- synchronizer module instantiation ------------------------------------------
         synchronizer : sigSynchronizer port map ( clk => clk,
                                                         enter => setup_Complete,
                                                         transmit => transmit,
                                                         sample => sample,
                                                         trans_klaar => trans_klaar,
                                                         transmit_rise => transmit_rise,
                                                         sample_rise => sample_rise,
                                                         enter_rise => enter_rise,
                                                         trans_done => trans_done,
                                                         RST => RST);


    -- Component instantiation
    crc_core : ethcrc32 port map (crc_clk,crc_rst,crc_en,crc_is_msb,crc_data_in,crc_crc_out);
    m2p     : mac2phy4IR port map(not_reset,MDIO,MDC,RESETN,COL,CRS,
                                         TXD,TXER,TXEN,TXCLK,
                                         RXD,RXER,RXDV,RXCLK,
                                         m2p_COLLISION,m2p_CARRIER,m2p_CLK,m2p_TDATA,m2p_TXVALID,
                                                    m2p_RDATA,m2p_RXVALID);
    --Trivial mapping
    not_reset<=not rst;
    local_clk<=m2p_CLK;
    crc(0)<= crc_crc_out(31 downto 24);
    crc(1)<= crc_crc_out(23 downto 16);
    crc(2)<= crc_crc_out(15 downto 8);
    crc(3)<= crc_crc_out(7 downto 0);


    --
------------------------------------------------------------------------------------
-- driven signals
---------------------
-- fsm_init
-- next_state
-- state
-- input signals
---------------------
-- curr_state
-- sample
    SYNC_FSM : process (clk, RST, next_state, succ_state)
    begin
--      curr_state <= curr_state;  infers a latch
        if(RST = '1') then
            curr_state <= setup;                                    -- go to reset state on RST rising edge
            state <= Idle;
        elsif clk = '1' and  clk'event then                 -- state transitions happen at rising edge of the clock
            curr_state <= next_state;
            state <= succ_state;                                    -- go to the next state
        end if;
    end process SYNC_FSM;


    FSM : process( enter_rise, curr_state, trans_done , RST , clk, transmit_rise, svEnable)
    begin
        case (curr_state) is
            when setup =>
                next_state <= setup;                            -- stay on setup state
                if enter_rise = '1' then
                    next_state <= readSVCB;                     -- on rising edge of ENTER go to sampling state
                end if;
            when readSVCB =>                                          -- when reading the sampled value control block
                next_state <= readSVCB;
                if svEnable = '1' then                                -- if the control block has been set then continue
                    next_state <= sampling;                           -- start sampling data
                end if;
            when sampling =>
                next_state <= sampling;                               -- default value of next state in the Samplimg State
                if transmit_rise = '1' then            -- start transmitting data as soon as it is copied onto the frame
                    next_state <= Transmission;
                end if;
            when Transmission =>
                next_state <= Transmission;                           -- stay on setup state
                if trans_done = '1' then
                    next_state <= sampling;    -- if transmission is complete then shift the ASDUs asdu(n) = asdu(n-1)
                end if;
        end case;
    end process FSM;


----------------------------------------------------------------------------------------------------------
-- Finite state machine
-- this is the main FSM for the packing algorithm, this FSM executes the following steps
-- 1. begins bulding the header of ISO/MAC 8802-3 frame  (MAC dest and source address, VLAN tag, length
-- 2. builds a record to store the SV APDU
-- 3. copies the record into the sv_frame buffer to begin tranmission and waits in the sampling state
--  4. When new samples are received the FSM moves to the transmit state and pushes out the ethernet frame to the PHY
-- 5. After transmission FSM goes back to the sampling state and waits for new messages
----------------------------------------------------------------------------------------------------------

FSM_OUTPUT : process (clk, RST, curr_state, sample_rise, trans_done, transmit_rise, no_asdu)

    variable conv_int : std_logic_vector(0 to 31) := x"00000000";
-- temporary array to hold converted integer
    variable index : integer range 0 to 150 := 0;
    -- ASDU handling signals
    variable asdu_asdu_offset : integer range 0 to 150 := 0;
    variable len_seqASDUs : integer range 0 to 150 := 0;                  -- length of the seqASDUs
    variable len_savPDU :integer range 0 to 150 := 0;                     -- length of savPDU
    variable len_apdu : integer range 0 to 150 := 0;                      -- total length of the APDU
    variable len_IEE802 : integer range 0 to 150 := 0;                    -- length of the Ethernet frame
    variable len_seqASDUn : integer range 0 to 150 := 0;                                  -- length of seqASDU(n)
    variable tempIndex : integer range 0 to 150 := 0;                                     -- length of seqASDU(n)
--  variable SmpCnt : integer := 0;                                                       -- sample count variable
    variable cdc_index : integer := 0;
    begin
        if(RST = '1') then
            sv_frame(0 to 70) <= (others=>x"00");                                 -- default all sv_frame bytes to x"00"
            succ_state <= Idle;                                                       -- Do not transmit any data (wait state)
            no_asdu <= 0;
            svEnable <= '0';
            count <= 0;
            index := 0;
            len_svpacket <= 0;
            tempIndex := 0;
            transmit <= '0';                                                          -- start transmit of the frame
            asdu_attr <= b"000";
            byteCnt <= 0;                                                             -- reset all signals on RST
number of SV packets per cycle/second

        elsif clk = '1' and  clk'event then                                   -- state transitions happen at rising edge of the
clock
            succ_state <= Idle;                                                   -- Transmission FSM defaults at Idle state
            svEnable <= '1';                                                          -- default vale for svEnable
            transmit <= '0';                                                          -- start transmit of the frame
            setup_Complete <= '0';
            case (curr_state) is
                when setup =>
                    svEnable <= '0';
                    no_asdu <= 0;
--                  asdu <= 0;                                                            -- force these variables to start from
0
                    --  Calculate lengths
                    cdc_index := 15 + (SV_ID'right+1) + 2;                                                      -- sample index
                    len_seqASDUn := SAV_CDC_SIZE + 15 + (SV_ID'right+1) + 2;         -- calculation of length of
seqASDU(n)
                    --seqASDUs
                    len_seqASDUs := asdu_asdu_offset * NO_ASDUS;                          -- length of all ASDUs
                    --savPDU
                    len_savPDU := len_seqASDUs + 5;
                    cdc_index := cdc_index + 5;
                    len_apdu := len_savPDU + 2;                      -- 1 byte representation
                    cdc_index := cdc_index + 2;
                    len_IEE802 := len_apdu + 8;                         -- total length of the IEC 802.3 frame
                    len_svpacket <= len_apdu + 25;
                    cdc_index := cdc_index + 26;                            -- this where the cdc samples start
                    for i in 0 to 5 loop
                        sv_frame(i) <= DEST_ADDRESS(i);                     -- Destination MAC address
                        sv_frame(6 + i) <= SOURCE_ADDRESS(i);                -- Source address (MAC Add of the device)
                    end loop;

                    for i in 0 to 3 loop
                        sv_frame(12 + i) <= IEEE_VLANtag(i);                                    -- Copy data to frame
                    end loop;

                    for i in 0 to 1 loop
                        sv_frame(16 + i) <= IEEE_ETHERTYPE(i);                                  -- IEC 61850 ether-type
                        sv_frame(18 + i) <= IEEE_APPID(i);                                      -- SV packets APP ID
                    end loop;

                    conv_int := std_logic_vector(to_signed(len_IEE802, 32));
                    sv_frame(20) <= conv_int(16 to 23);
                    sv_frame(21) <= conv_int(24 to 31);                                         -- Frame length


                --
                -- Copy constants into the Record according to ASN.1
                --
                    apdu.savAPDU(0) <= x"60";                                           -- savAPDU tag value
                    apdu.noASDU(0) <= x"80";                                            -- No of ASDU tag
                    apdu.noASDU(1) <= x"01";                                            -- 1 byte values field
                    apdu.noASDU(2) <= x"01";                                          -- 8 ASDU in an APDU
                    apdu.seqofASDU(0) <= x"A2";

                    -- Start of APDU
                    -- savPDU
                    conv_int := std_logic_vector(to_unsigned(len_savPDU, 32));
                    tempIndex := 27;                                                -- start of temp Index after savPDU tag
                    sv_frame(26) <= x"60";
                    -- noASDU
--      conv_int := std_logic_vector(to_signed(NO_ASDUS, 32));
                    sv_frame(tempIndex) <= x"80";
                    sv_frame(tempIndex + 1) <= x"01";
```

```
                    sv_frame(tempIndex + 2) <= x"01";              -- number of ASDUS in the message
                    tempIndex := tempIndex + 3;                   -- skip to appropriate index

                    --seqofASDUs
                    conv_int := std_logic_vector(to_signed(len_seqASDUs, 32));
                    sv_frame(tempIndex) <= x"A2";
                    tempIndex := tempIndex + 1;
                    ASDU_START_INDEX <= tempIndex;                  -- store the ASDU start address
                    SampleIndex <= cdc_index;
                    apdu.asdu.seqASDU(0) <= x"30";                 -- sequence of ASDU tag 0x30
                    conv_int := std_logic_vector(to_signed(len_seqASDUn, 32));      -- convert integer value
(len_seqASDUn)

                    apdu.asdu.seqASDU(1) <= x"62";                             -- length of sequence of ASDUs

                    -- svID Data
                    apdu.asdu.svID(0) <= x"80";                                -- svID tag 0x80
                    conv_int := std_logic_vector(to_signed(SV_ID'right + 1, 32));     -- get the length of svID
                    apdu.asdu.svID(1) <= conv_int(24 to 31);        -- assign the length of the array as the svID
length --

                    for i in 0 to SV_ID'right loop                                          -- copy sample to Record
                        apdu.asdu.svID(i + 2) <= SV_ID(i);                -- copy the latest time stamp the structure
                    end loop;
                    -- smpCnt
                    apdu.asdu.smpCnt(0) <= x"82";                  -- sample count TAG
                    apdu.asdu.smpCnt(1) <= x"02";                  -- sample count 2 bytes long
                    apdu.asdu.smpCnt(2) <= x"00";
                    apdu.asdu.smpCnt(3) <= x"00";
                    -- confRev
                    apdu.asdu.confRev(0) <= x"83";             -- configuration revision TAG
                    apdu.asdu.confRev(1) <= x"04";             -- configuration revision length (INTEGER )
                    apdu.asdu.confRev(2) <= x"00";             -- initial ConfRev 0
                    apdu.asdu.confRev(3) <= x"00";
                    apdu.asdu.confRev(4) <= x"01";
                    apdu.asdu.confRev(5) <= x"01";
                    --smpSynch
                    apdu.asdu.smpSynch(0) <= x"85";            -- synchronized samples TAG
                    apdu.asdu.smpSynch(1) <= x"01";            -- synchronized samples Length ( BOOLEAN )
                    apdu.asdu.smpSynch(2) <= x"00";            -- FALSE -- sample not synchronised
                    --samples
                    apdu.asdu.seqData(0) <= x"87";                             -- start of data values
                    apdu.asdu.seqData(1) <= x"40";
```

```
                        setup_Complete <= '1';

                    when readSVCB =>
                        svEnable <= '0';
                        succ_state <= Idle;             -- stay on the transmission Idle state until we have set up the SVCB
--                      if asdu < NO_ASDU then
--
--                          asdu <= asdu + 1;
--                      end if;
--                      if asdu = NO_ASDU then
                            if no_asdu < NO_ASDUS then
--                              index := (no_asdu * asdu_asdu_offset) + ASDU_START_INDEX + count;
                                index := ASDU_START_INDEX + count;                      -- start index value
                                case (asdu_attr) is
                                    -- seqASDUS ----------------------------------------------------------------------
                                    when b"000" =>
                                        asdu_attr <= b"000";            -- stay
                                        if byteCnt < apdu.asdu.seqASDU'right + 1 then
                                            sv_frame(index) <= apdu.asdu.seqASDU(byteCnt);          -- REMOVE index + count
                                            byteCnt <= byteCnt + 1;                     -- count number of bytes
                                            count <= count + 1;
                                        else
                                            byteCnt <= 0;
                                            asdu_attr <= b"001";                        -- handle next attribute
                                        end if;
                                    -- svID --------------------------------------------------------------------------
                                    when b"001" =>
                                        asdu_attr <= b"001";            -- stay
                                        if byteCnt < SV_ID'right + 3  then
                                            sv_frame(index) <= apdu.asdu.svID(byteCnt);
                                            byteCnt <= byteCnt + 1;                  -- count number of bytes
                                            count <= count + 1;
                                        else
                                            byteCnt <= 0;
                                            asdu_attr <= b"010";                      -- handle next attribute
                                        end if;
                                    -- smpCnt ------------------------------------------------------------------------
                                    when b"010" =>
                                        asdu_attr <= b"010";            -- stay
                                        if byteCnt < apdu.asdu.SmpCnt'right + 1  then
                                            sv_frame(index) <=  apdu.asdu.SmpCnt(byteCnt);
```

```
        byteCnt <= byteCnt + 1;                  -- count number of bytes
        count <= count + 1;
    else
        byteCnt <= 0;
        asdu_attr <= b"011";                     -- handle next attribute
    end if;
-- confRev ----------------------------------------------------------------------------
when b"011" =>
    asdu_attr <= b"011";            -- stay
    if byteCnt < apdu.asdu.confRev'right + 1 then
        sv_frame(index) <=  apdu.asdu.confRev(byteCnt);
        byteCnt <= byteCnt + 1;      -- count number of bytes
        count <= count + 1;
    else
        byteCnt <= 0;
        asdu_attr <= b"100";                 -- handle next attribute
    end if;
-- smpSynch -------------------------------------------------------------------------
when b"100" =>
    asdu_attr <= b"100";            -- stay
    if byteCnt < apdu.asdu.smpSynch'right + 1 then
        sv_frame(index) <=  apdu.asdu.smpSynch(byteCnt);
        byteCnt <= byteCnt + 1;             -- count number of bytes
        count <= count + 1;
    else
        byteCnt <= 0;
        asdu_attr <= b"101";               -- handle next attribute
    end if;
-- seqData ---------------------------------------------------------------------------
when b"101" =>
    asdu_attr <= b"101";            -- stay
    if byteCnt < apdu.asdu.seqData'right + 1 then
        sv_frame(index) <=  apdu.asdu.seqData(byteCnt);
        byteCnt <= byteCnt + 1;        -- count number of bytes
        count <= count + 1;
    else
        byteCnt <= 0;
        asdu_attr <= b"110";            -- handle next attribute
    end if;
-- allData --------------------------------------------------------------------------
when b"110" =>               -- if the ASDU attribute is 110 or something not defined
```

```
                                        byteCnt <= 0;
                                        asdu_attr <= b"000";
                                        count <= 0;             -- go to the next index when starting off with the next ASDU
                                        no_asdu <= no_asdu + 1;
                                    when others =>
                                        byteCnt <= 0;
                                        asdu_attr <= b"000";
                                        count <= 0;        -- go to the next index when starting off with the next ASDU
-- clear that so that it starts from0 for new ASDU data
                                        no_asdu <= no_asdu + 1;
                                end case;
                            end if;
                            if no_asdu = NO_ASDUS then
                                svEnable <= '1';                        -- SV enabled
                            end if;
--                      end if;
                    when sampling =>
                        transmit <= '0';                                -- start transmit of the frame
                        if sample_rise = '1' then
                            sv_frame(cdc_index - 12) <= SampleCnt(7 downto 0);
                            sv_frame(cdc_index - 13)(3 downto 0) <= SampleCnt(11 downto 8);             -- Phase N quality
                            succ_state <= Transmitting;                 -- move to Transmission state (MAC)
                            transmit <= '1';                            -- start transmit of the frame
                        end if;
                    when Transmission =>
                        succ_state <= Transmitting;                 -- stay on the Transmitting State (MAC)
                        if trans_done = '1' then           -- at the end of the transmission cycle go back to the Idle state
                            succ_state <= Idle;                                         -- MAC idle state
                        end if;
                end case;
            end if;
        end process FSM_OUTPUT;

        ---------------------------------------------------------------------------------------
        -- This process copied the calculated samples into the cdcsample buffer for transmission
        ---------------------------------------------------------------------------------------
    COLLECT_SAMPLES : process (TCTRA, TCTRB, TCTRC, TCTRN, TVTRA, TVTRB, TVTRC, TVTRN, transmit_rise)
        begin
            if RST = '1' then
                cdcSamples(0 to 63) <= (others=>x"00");
            end if;
            if clk = '1' and clk'event then
                    if transmit_rise = '1' then
                    -------------- TCTRA ------------------------------------------------------------
                        cdcSamples(0) <= TCTRA(31 downto 24);                --TCTRA(31 downto 24);
                        cdcSamples(1) <= TCTRA(23 downto 16);
                        cdcSamples(2) <= TCTRA(15 downto 8);
                        cdcSamples(3) <= TCTRA(7 downto 0);                     -- Current Values for Phase A
                        cdcSamples(4) <= sampleCTVT(215 downto 208);
                        cdcSamples(5) <= sampleCTVT(207 downto 200);
                        cdcSamples(6) <= sampleCTVT(199 downto 192);

    --                  cdcSamples(6) <= qualTCTRA(15 downto 8);
    --                  cdcSamples(7) <= qualTCTRA(7 downto 0);             -- Phase A quality

                    -------------- TCTRB ------------------------------------------------------------
                        cdcSamples(8) <= TCTRB(31 downto 24);
                        cdcSamples(9) <= TCTRB(23 downto 16);
                        cdcSamples(10) <= TCTRB(15 downto 8);
                        cdcSamples(11) <= TCTRB(7 downto 0);                     -- Current Values for Phase B


                        cdcSamples(14) <= qualTCTRB(15 downto 8);
                        cdcSamples(15) <= qualTCTRB(7 downto 0);                -- Phase B quality

                    -------------- TCTRC ------------------------------------------------------------
                        cdcSamples(16) <= TCTRC(31 downto 24);
                        cdcSamples(17) <= TCTRC(23 downto 16);
                        cdcSamples(18) <= TCTRC(15 downto 8);
                        cdcSamples(19) <= TCTRC(7 downto 0);                     -- Current Values for Phase C


                        cdcSamples(22) <= qualTCTRC(15 downto 8);
                        cdcSamples(23) <= qualTCTRC(7 downto 0);                -- Phase C quality

                    -------------- TCTRN ------------------------------------------------------------
    --                  cdcSamples(24) <= TCTRN(31 downto 24);
    --                  cdcSamples(25) <= TCTRN(23 downto 16);
    --                  cdcSamples(26) <= TCTRN(15 downto 8);
    --                  cdcSamples(27) <= TCTRN(7 downto 0);                     -- Current Values for Phase N current

        --              cdcSamples(28) <= qualTCTRN(31 downto 24);
        --              cdcSamples(29) <= qualTCTRN(23 downto 16);
        --              cdcSamples(30) <= qualTCTRN(15 downto 8);
        --              cdcSamples(31) <= qualTCTRN(7 downto 0);                 -- Phase N quality

                    ------------------------------------------------------------------------------
                    -------------- TVTRA -----------------------------------------------------------
                        cdcSamples(32) <= TVTRA(31 downto 24);
                        cdcSamples(33) <= TVTRA(23 downto 16);
                        cdcSamples(34) <= TVTRA(15 downto 8);
                        cdcSamples(35) <= TVTRA(7 downto 0);                      -- Voltage Values for Phase N


                        cdcSamples(38) <= qualTVTRA(15 downto 8);
                        cdcSamples(39) <= qualTVTRA(7 downto 0);                 -- Phase A quality

                    -------------- TVTRB -----------------------------------------------------------
                        cdcSamples(40) <= TVTRB(31 downto 24);
                        cdcSamples(41) <= TVTRB(23 downto 16);
                        cdcSamples(42) <= TVTRB(15 downto 8);
                        cdcSamples(43) <= TVTRB(7 downto 0);                      -- Voltage Values for Phase B


                        cdcSamples(46) <= qualTVTRB(15 downto 8);
                        cdcSamples(47) <= qualTVTRB(7 downto 0);                 -- Phase B quality

                    -------------- TVTRC -----------------------------------------------------------
                        cdcSamples(48) <= TVTRC(31 downto 24);
                        cdcSamples(49) <= TVTRC(23 downto 16);
                        cdcSamples(50) <= TVTRC(15 downto 8);
                        cdcSamples(51) <= TVTRC(7 downto 0);                      -- Voltage Values for Phase C


                        cdcSamples(54) <= qualTVTRC(15 downto 8);
                        cdcSamples(55) <= qualTVTRC(7 downto 0);                 -- Phase C quality

                    -------------- TVTRN -----------------------------------------------------------
    --                  cdcSamples(56) <= TVTRN(31 downto 24);
    --                  cdcSamples(57) <= TVTRN(23 downto 16);
    --                  cdcSamples(58) <= TVTRN(15 downto 8);
    --                  cdcSamples(59) <= TVTRN(7 downto 0);                      -- Voltage Values for Phase N
    --
```

```
--              cdcSamples(60) <= qualTVTRN(31 downto 24);
--              cdcSamples(61) <= qualTVTRN(23 downto 16);
--              cdcSamples(62)(3 downto 0) <= SampleCnt(11 downto 8);              -- Phase N quality
--              cdcSamples(63) <= SampleCnt(7 downto 0);

            end if;
    end if;
end process COLLECT_SAMPLES;

-------------------------------------------------------------------------------------------
-- This process calculates the current sample value of the input from the ADC(AFE)
-- using this formula
--
--  Sample = (Input (24 bit) / 0x7FFFFF) * Current / Voltage
--
--------------------------------------------------------------------------------------------
CALC_SAMPLES : process (clk, sample_rise)
begin
    if RST = '1' then
        TCTRA <= x"00000000";
        TCTRB <= x"00000000";
        TCTRC <= x"00000000";
        TCTRN <= x"00000000";
        TVTRA <= x"00000000";
        TVTRB <= x"00000000";
        TVTRC <= x"00000000";
        TVTRN <= x"00000000";
    elsif clk = '1' and clk'event then
        if sample_rise = '1' then
--          TCTRN(31 downto 0) <= Calc_AMPS(sampleCTVT(23 downto 0));               -- channel 8
--          TCTRC(31 downto 0) <= Calc_AMPS(sampleCTVT(47 downto 24));              -- channel 7
            TCTRB(31 downto 0) <= Calc_AMPS(sampleCTVT(71 downto 48));              -- channel 6
            TCTRA(31 downto 0) <= Calc_AMPS(sampleCTVT(95 downto 72));
--          TVTRN(31 downto 0) <= Calc_VOLTS(sampleCTVT(119 downto 96));            -- channel 4
--          TVTRC(31 downto 0) <= Calc_VOLTS(sampleCTVT(143 downto 120));           -- channel 3
            TVTRB(31 downto 0) <= Calc_VOLTS(sampleCTVT(167 downto 144));           -- channel 2
            TVTRA(31 downto 0) <= Calc_VOLTS(sampleCTVT(191 downto 168));           -- channel 1
        end if;
    end if;
end process;
```

```
    ----------------------------------------------------------------------------------------
    -- Process detects a change in the input signals from the AFE and generates
    -- sample signal to force the publisher to the transmit state
    -- on LOW-HIGH transition of sample signal sample values are generated
    ---------------------------------------------------------------------------------------
    MAP_SV : process(SampleCnt, RST)
    variable curr_cnt, prevVal : signed(11 downto 0) := b"000000000000";
    begin
        if RST = '1' then
            curr_cnt := b"000000000000";
            prevVal := b"000000000000";
            sample <= '0';
        elsif clk = '1' and clk'event then                                                  -- synchronous
            curr_cnt := signed(SampleCnt);
            prevVal := signed(smpCount);
            smpCount(11 downto 0) <= SampleCnt(11 downto 0);
            sample <= '0';
            if curr_cnt /= prevVal then
                sample <= '1';
            end if;
        end if;
    end process;

end Behavioral;
```

Appendix F.2 MAC2PHY4IR.vhd

```
----------------------------------------------------------------------------
--This file defines an entity that simplifies the MII interface
--with an Ethernet Physical layer. It particularly fits with the
--Nexys3 development board from Digilent.
--The entity have several types of ports:
--General purposes (RSTN)
--Physical ports to manage the communication with the MII Interface on
--the physical layer side
--Mac ports to communicate with MAC layer
--------------------------------------------------------------------------
--IMPORTANT:
--Because the physical interface has two clocks domains (RXCLK, TXCLK)
--This entity uses a FIFO component specific to the SPARTAN6 FPGA of NEXYS3 board
```

```vhdl
--As a result, the entity provide only one clock signal (CLK) to the mac layer.

---------------------------------------------------------------------------
--MAC PORTS/INTERFACE COMMUNICATION:
-- COLLISION is set to '1' by the entity when the PHY layer has detected a collision
-- CARRIER is set to '1' by the entity when the PHY layer has detected a carrier
-- CLK clock given to the mac layer, based on the TXCLK of the PHY layer (25Mhz)
-- TDATA: 8 bits std_logic_vector that the MAC layer must provide to be transmitted
-- TDATA value must be hold for 80 ns (2 periods clock)
-- TXVALID: A pulse of 40ns (1 period clock) must be hold when valid data is available on TDATA
-- RDATA: 8 bits std_logic_vector that the entity provide to the mac layer
-- its value changes every 40ns (1 period clock). The mac layer must look for
-- the SFD by checking every clock rising edge. Once the SFD is found, the
-- MAC layer only needs to check every two clock rising edges to have a valid value.
-- RXVALID: The entity set this flag to '1' when the receiver is synchronized, (ie) SFD is
--              about to be visible on RDATA.
--------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity mac2phy4IR is
    Port (
           -- General purpose ports
           RSTN : in  STD_LOGIC;
           --PHY ports
           --General purpose
           MDIO : inout  STD_LOGIC;
           MDC : out  STD_LOGIC;
           RESETN : out  STD_LOGIC;
           COL : in  STD_LOGIC;
           CRS : in  STD_LOGIC;
           --TX
           TXD :out  STD_LOGIC_VECTOR (3 downto 0);
           TXER : out  STD_LOGIC;
           TXEN: out STD_LOGIC;
           TXCLK : in  STD_LOGIC;
           --RX
           RXD : in  STD_LOGIC_VECTOR (3 downto 0);
           RXER : in  STD_LOGIC;
           RXDV : in  STD_LOGIC;
           RXCLK : in  STD_LOGIC;
           --MAC Ports
           --General purpose
           COLLISION: out STD_LOGIC;
           CARRIER: out std_logic;
           CLK: out STD_LOGIC;
           --TX
           TDATA: in STD_LOGIC_VECTOR (7 downto 0);
           TXVALID: in STD_LOGIC;
           --RX
           RDATA : out STD_LOGIC_VECTOR (7 downto 0);
           RXVALID: out STD_LOGIC);
end mac2phy4IR;

architecture Behavioral of mac2phy4IR is
    signal first_nibble: Boolean :=true;
    signal nTXD : std_logic_vector(3 downto 0) :=(others=>'0');
    signal RXDbuf: std_logic_vector(7 downto 0):=(others=>'0');
    signal noDATA: std_logic:='1';
    signal fifoOUT : std_logic_vector(3 downto 0);
    signal rst : std_logic:='1';
    signal RXen : std_logic:='0';

    component fifo_core is
        Port (
         rst : IN STD_LOGIC;
         wr_clk : IN STD_LOGIC;
         rd_clk : IN STD_LOGIC;
         din : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
         wr_en : IN STD_LOGIC;
         rd_en : IN STD_LOGIC;
         dout : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
         full : OUT STD_LOGIC;
         empty : OUT STD_LOGIC);
    end component;
begin

    -- General purpose signals
    RESETN<=RSTN;
    COLLISION<=COL;
    CARRIER<=CRS;
    CLK<=TXCLK;
    --Pre cabled signals
    MDIO      <='1';
    MDC   <='0';
    TXER      <='0';
    --Rx signals
    RDATA<= RXDbuf;
    RXVALID<=RXDV;
    --fifo signals
    RXen<= not noDATA;
    rst<= not RSTN;
    fifoRX : fifo_core port map (rst,RXCLK,TXCLK,RXD,RXDV,RXen,fifoOUT,open,noDATA);            -- positional mapping of
signals in fifo_core to the local signals
    --Interfacing process
    process(TXCLK,RSTN) is begin
        if (RSTN='0') then
                first_nibble<=true;
                TXD<=(others=>'0');
                RXDbuf<= (others=>'0');
        elsif (rising_edge(TXCLK)) then
                RXDbuf<= RXDbuf(3 downto 0) & fifoOUT;    -- receive a nibble
                TXEN    <= TXVALID;
                if first_nibble and TXVALID='1' then
                    TXD<= TDATA(7 downto 4);
                    nTXD<= TDATA(3 downto 0);
                    first_nibble<=false;
                elsif first_nibble=false then
                    TXD<=nTXD;
                    first_nibble<=true;
                end if;
        end if;
    end process;
end Behavioral;
```

Appendix F.3 sigSynchronizer.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;
-- Uncomment the following library declaration if instantiating
```

```vhdl
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity sigSynchronizer is
    Port ( clk : in  STD_LOGIC;
              enter : in STD_LOGIC;
              transmit : in STD_LOGIC;
              sample : in STD_LOGIC;
              trans_klaar : in STD_LOGIC;
              transmit_rise : out STD_LOGIC;
              sample_rise : out STD_LOGIC;
              enter_rise : out STD_LOGIC;
              trans_done : out STD_LOGIC;
           RST : in  STD_LOGIC);
end sigSynchronizer;

architecture Behavioral of sigSynchronizer is

begin
    ------------------------------------------------------------------------
    -- This process synchronizes the signals to the SYS clock
    SIGNAL_SYNCH : process(clk)
        variable enter_sync : std_logic_vector(1 to 3);    -- temp signals for synchronization
        variable transmit_sync : std_logic_vector(1 to 3);
        variable sample_sync : std_logic_vector(1 to 3);
        variable trans_sync : std_logic_vector(1 to 3);
        begin
        if rising_edge(clk) then
            enter_rise <= enter_sync(2) and not enter_sync(3);
            transmit_rise <= transmit_sync(2) and not transmit_sync(3);
            sample_rise <= sample_sync(2) and not sample_sync(3);
            trans_done <= trans_sync(2) and not trans_sync(3);

            enter_sync := enter & enter_sync(1 to 2);
            transmit_sync := transmit & transmit_sync(1 to 2);
            sample_sync := sample & sample_sync(1 to 2);
            trans_sync := trans_klaar & trans_sync(1 to 2);
        end if;
    end process SIGNAL_SYNCH;
    --------------------------------------------------------------------------
```

```
end Behavioral;
```

Appendix F.4 ethernet_frame.vhd

```
--  Package File Template
--  Purpose: This package defines supplemental types, subtypes,
--  constants, and functions
--   To use any of the example code shown below, uncomment the lines and modify as necessary
library ieee_proposed;
use ieee_proposed.float_pkg.all;
use ieee_proposed.fixed_float_types.all;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.numeric_std.all;

package ethernet_frame is
    type byte_array is array (integer range <> ) of std_logic_vector(7 downto 0);
     -- --------------------------------------------------------------------
    --Sampled value APDU record
    type ASDU is
    record
        seqASDU : byte_array(integer range 0 to 1);              -- Sequence of ASDU (2 bytes )  -- 0x30 | ASDU Length |
Length byte
--      svID : byte_array (integer range 0 to 36);               -- TAG | Length | Visible string of 35 bytes
        svID : byte_array (integer range 0 to 20);               -- TAG | Length | Visible string of 35 bytes
        smpCnt : byte_array (integer range 0 to 3);              -- 4 byte value - 0x82 | L | CountH | CountL |-
        confRev : byte_array (integer range 0 to 5);         -- 6 byte field - 0x83 | L(0x04)| 4 byte Config Value| -
        smpSynch :byte_array (integer range 0 to 2);         -- 3 byte field - 0x84 | L(0x01) | SYNCH | -
        seqData :  byte_array (integer range 0 to 1);            -- 2 byte - | 0x87 | L  |
    end record ASDU;

    type APDU is
    record
        savAPDU : byte_array (integer range 0 to 3);             -- 4 byte field - 0x60 | 0x82 | apduLH | apduLL |-
        noASDU : byte_array (integer range 0 to 2);              -- 3 byte field - 0x80 | L(0x01) | numberofASDU |-
        seqofASDU : byte_array ( integer range 0 to 3);          -- 4 byte field - 0xA2 | 0x82 | L (length of ASDUs)-
        asdu : ASDU;
    end record APDU;

-- Function definition ---------------------------------------------
```

```
    function Calc_AMPS  (signal Sample : in std_logic_vector(23 downto 0))  return std_logic_vector;
    function Calc_VOLTS  (signal Sample : in std_logic_vector(23 downto 0)) return std_logic_vector;

end ethernet_frame;

package body ethernet_frame is

    -- ------------------------FUNCTION BODY -----------------------


    -------------------------------------------------------------
    -- Calculate Amps sample value using 24 bit input, Ref (Maximum Deflection), input current
    -----------------------------------------------------------------
    function Calc_AMPS  (signal Sample : in std_logic_vector(23 downto 0))  return std_logic_vector is
        variable TCTR : std_logic_vector(31 downto 0) := x"00000000";
        variable signedVal : signed(23 downto 0) := x"000000";
        variable Ref, tctr_mag,inTCTR : float(8 downto -23) := x"00000000";
        begin
            signedVal := signed(Sample);
            inTCTR := to_float(signedVal, Ref);
            Ref := to_float(0.282989445, Ref);                              -- (Current * 1000)/Max_Pos_Deflection
(9600A * 100/8388607) 9600 is the over current measurement
            tctr_mag := inTCTR * Ref;       -- calculate feeder current
            TCTR(31 downto 0) := std_logic_vector(to_signed(tctr_mag, 32)); -- return the scaled value
        return TCTR;
    end Calc_AMPS;

    --------------------------------------------------------------
    -- Calculate Volts sample value using 24 bit input, Ref (Maximum Deflection), input current
    -------------------------------------------------------------
    function Calc_VOLTS  (signal Sample : in std_logic_vector(23 downto 0)) return std_logic_vector is
        variable TVTR : std_logic_vector(31 downto 0);
        variable signedVal : signed(23 downto 0);
        variable Ref, tvtr_mag,inTVTR : float(8 downto -23);
        begin
            signedVal := signed(Sample);
            inTVTR := to_float(signedVal, Ref);
            Ref := to_float(0.146151345, Ref);      -- (Voltage * 100)/Max_Pos_Deflection (11881.18852V * 100/8388607)
            tvtr_mag := inTVTR * Ref;      -- calculate feeder current
            TVTR(31 downto 0) := std_logic_vector(to_signed(tvtr_mag, 32)); -- return the scaled value
        return TVTR;
    end Calc_VOLTS;
end ethernet_frame;
```

Appendix F.5 ethcrc32.vhd

```
-----------------------------------------------------------------------
--This file defines an entity that compute a CRC for 8802.3 procotol
--The computation is done by group of 8 bits.
--The reset is asynchronous.
--Computation are synchronous to clk and enabled when 'en' equals 1
--data_in is an 8bits input port for data to consider for a crc computation
--is_msb equals '1' indicates that data_in need to be reverted (msb to lsb) before computation.
------------------------------------------------------------------------------
--IMPORTANT :
--To be used in a real Ethernet case, user must take considerations
--which are:
--1) The four first byte must be Ones' complemented
--2) data_in only considers some fields (ie not the preamble nor the SFD, see norm)
--3) data_in must be given in the same sens than transmission:
--       * LSB first for data
--       * MSB first for CRC in the case of reception
--       * ALL zero for CRC in the case of transmission
--4a) crc_out must be complemented before transmission
--4b) received crc must be complemented before given to data_in for
--    error checking in reception
--5) In the case of error checking in reception, if the received frame is correct
-- crc_out equals x"00000000"
------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity ethcrc32 is
    port (clk: in STD_LOGIC;                        -- Input clock
         rst: in STD_LOGIC;                     -- Asynchronous reset
         en : in STD_LOGIC;                 -- Assert to compute calculations
         is_msb: in STD_LOGIC;          -- Assert to indicate the sens of data_in
         data_in: in STD_LOGIC_VECTOR(7 downto 0);          -- Data to compute
         crc_out: out STD_LOGIC_VECTOR (31 downto 0)      -- CRC output
    );
```

```
end ethcrc32;
-------------------------------------------------------------------------------
architecture nano of ethcrc32 is
-- The Generator polynomial is
--  32   26   23   22   16   12   11   10   8   7   5   4   2
-- x  + x  + x  + x  + x  + x  + x  + x  + x + x + x + x + x + x + 1
constant GENERATOR : STD_LOGIC_VECTOR := X"04C11DB7";
begin
    process (clk,rst) is
        variable crc_buf : STD_LOGIC_VECTOR (31 downto 0):=x"00000000";
    begin
        if rst = '1' then   -- reset signals to values
            crc_buf := (others => '0');
        elsif rising_edge(clk) then  -- operate on positive edge
            if (en='1') then
                if is_msb='1' then
                        for I in data_in'reverse_range loop
                            crc_buf := (crc_buf(30 downto 0) & data_in(I)) XOR (GENERATOR AND (0 to 31=>crc_buf(31)));
                        end loop;
                else
                        for I in data_in'range loop
                            crc_buf := (crc_buf(30 downto 0) & data_in(I)) XOR (GENERATOR AND (0 to 31=>crc_buf(31)));
                        end loop;
                end if;
            end if;
        end if;
        crc_out<=crc_buf;
    end process;
end nano;
```

APPENDIX G. FPGA-based GOOSE-enabled remote monitoring node CID file

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SCL xmlns="http://www.iec.ch/61850/2003/SCL" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Private type="SchneiderElectric-SFT-Key">BA0F63FC6D573EFBE7319E867B65F755</Private>
  <Private type="SchneiderElectric-SFT-EditTime">2016-05-01 03:01:11</Private>
  <Private type="SchneiderElectric-SFT-Version">2.1.0</Private>
  <Header id="My Project Id" nameStructure="IEDName" revision="0" toolID="CET850 v2.0" version="2">
    <History>
      <Hitem revision="0" version="V1" what="Draft 1" when="2016-05-01" who="CET850Config"/>
    </History>
  </Header>
  <Communication>
    <SubNetwork name="NONE">
      <ConnectedAP apName="AP1" iedName="FPGA">
        <Address>
          <P type="IP" xsi:type="tP_IP">169.254.0.10</P>
          <P type="IP-SUBNET" xsi:type="tP_IP-SUBNET">255.255.0.0</P>
          <P type="IP-GATEWAY" xsi:type="tP_IP-GATEWAY">0.0.0.0</P>
          <P type="OSI-PSEL" xsi:type="tP_OSI-PSEL">00000001</P>
          <P type="OSI-SSEL" xsi:type="tP_OSI-SSEL">0001</P>
          <P type="OSI-TSEL" xsi:type="tP_OSI-TSEL">0001</P>
        </Address>
        <GSE cbName="GSE_CB_GOOSE" ldInst="IED1">
          <Address>
            <P type="MAC-Address" xsi:type="tP_MAC-Address">01-0C-CD-01-00-00</P>
            <P type="APPID" xsi:type="tP_APPID">0000</P>
            <P type="VLAN-ID" xsi:type="tP_VLAN-ID">000</P>
            <P type="VLAN-PRIORITY" xsi:type="tP_VLAN-PRIORITY">4</P>
          </Address>
          <MinTime multiplier="m" unit="s">100</MinTime>
          <MaxTime multiplier="m" unit="s">1000</MaxTime>
        </GSE>
      </ConnectedAP>
    </SubNetwork>
  </Communication>
  <IED configVersion="1" desc="FPGA based IEC 61850-8-1 GOOSE message publisher" manufacturer="CSAEMS_NCUBE" name="FPGA"
type="RTUType">
    <Services>
      <ConfDataSet max="0" maxAttributes="0" modify="false"/>
      <GOOSE max="10"/>
    </Services>
    <AccessPoint clock="false" name="AP1" router="false">
      <Server timeout="30">
        <Authentication none="true"/>
        <LDevice inst="IED1">
          <LN0 desc="Logical node zero" inst="" lnClass="LLN0" lnType="LLN0_0">
            <DataSet desc="" name="GOOSE_Eval">
              <FCDA daName="mag" doName="Vol" fc="MX" ldInst="IED1" lnClass="MMXN" lnInst="0"/>
              <FCDA daName="stVal" doName="Pos" fc="ST" ldInst="IED1" lnClass="XCBR" lnInst="0"/>
            </DataSet>
            <GSEControl appID="GOOSEID" confRev="2" datSet="GOOSE_Eval" name="GSE_CB_GOOSE" type="GOOSE">
              <Private type="SchneiderElectric-IED-GseRef">
                <GseRef xmlns="http://www.schneider-electric.com/IEC61850/XMLSchema" cbName="GSE_CB_GOOSE"
ldInst="IED1">
                  <Address>
                    <P type="MAC-Address">01-0C-CD-01-00-00</P>
                    <P type="APPID">0000</P>
                    <P type="VLAN-ID">000</P>
                    <P type="VLAN-PRIORITY">4</P>
                  </Address>
                  <MinTime multiplier="m" unit="s">100</MinTime>
                  <MaxTime multiplier="m" unit="s">1000</MaxTime>
                </GseRef>
              </Private>
            </GSEControl>
          </LN0>
          <LN desc="Physical device information" inst="0" lnClass="LPHD" lnType="LPHD_0" prefix=""/>
          <LN desc="Circuit breaker" inst="1" lnClass="XCBR" lnType="XCBR_0" prefix=""/>
          <LN desc="Non phase related Measurement" inst="1" lnClass="MMXN" lnType="MMXN_0" prefix=""/>
        </LDevice>
      </Server>
    </AccessPoint>
  </IED>
  <DataTypeTemplates>
    <LNodeType id="LPHD_0" lnClass="LPHD">
      <DO desc="Device name plate" name="PhyNam" type="DPL_0"/>
      <DO desc="Enumerated status" name="PhyHealth" type="ENS_0"/>
      <DO desc="Single point status" name="Proxy" type="SPS_0"/>
    </LNodeType>
    <LNodeType id="LLN0_0" lnClass="LLN0">
      <DO desc="Controllable enumerated status" name="Mod" type="ENC_1"/>
      <DO desc="Enumerated status" name="Beh" type="ENS_0"/>
      <DO desc="Enumerated status" name="Health" type="ENS_1"/>
      <DO desc="Logical Node name plate" name="NamPlt" type="LPL_0"/>
    </LNodeType>
    <LNodeType id="MMXN_0" lnClass="MMXN">
      <DO desc="Controllable enumerated status" name="Mod" type="ENC_0"/>
      <DO desc="Enumerated status" name="Beh" type="ENS_0"/>
      <DO desc="Enumerated status" name="Health" type="ENS_0"/>
      <DO desc="Logical Node name plate" name="NamPlt" type="LPL_0"/>
      <DO desc="Measured value" name="Vol" type="MV_0"/>
    </LNodeType>
    <LNodeType id="XCBR_0" lnClass="XCBR">
      <DO desc="Controllable enumerated status" name="Mod" type="ENC_0"/>
      <DO desc="Enumerated status" name="Beh" type="ENS_0"/>
      <DO desc="Enumerated status" name="Health" type="ENS_0"/>
      <DO desc="Logical Node name plate" name="NamPlt" type="LPL_0"/>
      <DO desc="Single point status" name="Loc" type="SPS_0"/>
      <DO desc="Integer status" name="OpCnt" type="INS_0"/>
      <DO desc="Controllable double point" name="Pos" type="DPC_0"/>
      <DO desc="Controllable single point" name="BlkOpn" type="SPC_0"/>
      <DO desc="Controllable single point" name="BlkCls" type="SPC_0"/>
      <DO desc="Integer status" name="CBOpCap" type="INS_0"/>
    </LNodeType>
    <DOType cdc="DPL" desc="Device name plate" id="DPL_0">
      <DA bType="VisString255" fc="DC" name="vendor"/>
    </DOType>
    <DOType cdc="SPS" desc="Single point status" id="SPS_0">
      <DA bType="BOOLEAN" dchg="true" fc="ST" name="stVal"/>
      <DA bType="Quality" fc="ST" name="q" qchg="true"/>
      <DA bType="Timestamp" fc="ST" name="t"/>
    </DOType>
    <DOType cdc="SPC" desc="Controllable single point" id="SPC_0">
      <DA bType="Enum" fc="CF" name="ctlModel" type="ctlModel"/>
    </DOType>
    <DOType cdc="DPC" desc="Controllable double point" id="DPC_0">
      <DA bType="Dbpos" dchg="true" fc="ST" name="stVal" type="Dbpos"/>
      <DA bType="Quality" fc="ST" name="q" qchg="true"/>
      <DA bType="Timestamp" fc="ST" name="t"/>
      <DA bType="Enum" fc="CF" name="ctlModel" type="ctlModel"/>
    </DOType>
    <DOType cdc="LPL" desc="Logical Node name plate" id="LPL_0">
      <DA bType="VisString255" fc="DC" name="vendor"/>
      <DA bType="VisString255" fc="DC" name="swRev"/>
      <DA bType="VisString255" fc="DC" name="d"/>
    </DOType>
    <DOType cdc="ENS" desc="Enumerated status" id="ENS_0">
      <DA bType="Enum" dchg="true" fc="ST" name="stVal" type="Mod"/>
      <DA bType="Quality" fc="ST" name="q" qchg="true"/>
      <DA bType="Timestamp" fc="ST" name="t"/>
    </DOType>
    <DOType cdc="ENC" desc="Controllable enumerated status" id="ENC_0">
      <DA bType="Enum" dchg="true" fc="ST" name="stVal" type="Mod"/>
      <DA bType="Quality" fc="ST" name="q" qchg="true"/>
      <DA bType="Timestamp" fc="ST" name="t"/>
      <DA bType="Enum" dchg="true" fc="CF" name="ctlModel" type="ctlModel"/>
    </DOType>
    <DOType cdc="ENS" desc="Enumerated status" id="ENS_1">
      <DA bType="Enum" dchg="true" fc="ST" name="stVal" type="Health"/>
      <DA bType="Quality" fc="ST" name="q" qchg="true"/>
      <DA bType="Timestamp" fc="ST" name="t"/>
    </DOType>
    <DOType cdc="ENC" desc="Controllable enumerated status" id="ENC_1">
      <DA bType="Enum" dchg="true" fc="ST" name="stVal" type="Mod"/>
      <DA bType="Quality" fc="ST" name="q" qchg="true"/>
      <DA bType="Timestamp" fc="ST" name="t"/>
      <DA bType="Enum" fc="CF" name="ctlModel" type="ctlModel"/>
    </DOType>
    <DOType cdc="MV" desc="Measured value" id="MV_0">
      <DA bType="Struct" dchg="true" fc="MX" name="mag" type="mag_0"/>
      <DA bType="Quality" fc="MX" name="q" qchg="true"/>
      <DA bType="Timestamp" fc="MX" name="t"/>
    </DOType>
    <DOType cdc="INS" desc="Integer status" id="INS_0">
      <DA bType="INT32" dchg="true" fc="ST" name="stVal"/>
      <DA bType="Quality" fc="ST" name="q" qchg="true"/>
      <DA bType="Timestamp" fc="ST" name="t"/>
    </DOType>
```

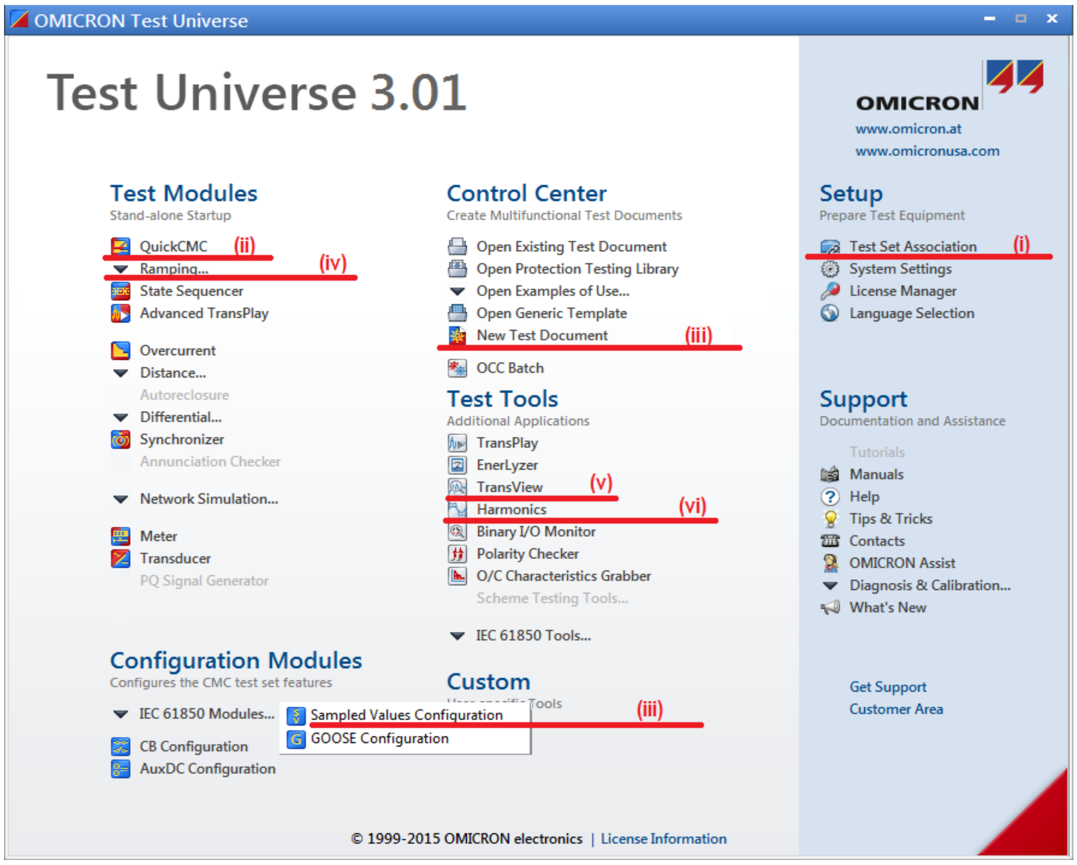
```
    <DAType id="mag_0">
      <BDA bType="INT32" name="i"/>
    </DAType>
    <EnumType id="Dbpos">
      <EnumVal ord="0">intermediate</EnumVal>
      <EnumVal ord="1">off</EnumVal>
      <EnumVal ord="2">on</EnumVal>
      <EnumVal ord="3">bad</EnumVal>
    </EnumType>
    <EnumType id="ctlModel">
      <EnumVal ord="0">status-only</EnumVal>
      <EnumVal ord="1">direct-with-normal-security</EnumVal>
      <EnumVal ord="2">sbo-with-normal-security</EnumVal>
      <EnumVal ord="3">direct-with-enhanced-security</EnumVal>
      <EnumVal ord="4">sbo-with-enhanced-security</EnumVal>
    </EnumType>
    <EnumType id="Health">
      <EnumVal ord="1">Ok</EnumVal>
      <EnumVal ord="2">Warning</EnumVal>
      <EnumVal ord="3">Alarm</EnumVal>
    </EnumType>
    <EnumType id="Mod">
      <EnumVal ord="1">on</EnumVal>
      <EnumVal ord="2">blocked</EnumVal>
      <EnumVal ord="3">test</EnumVal>
      <EnumVal ord="4">test/blocked</EnumVal>
      <EnumVal ord="5">off</EnumVal>
    </EnumType>
  </DataTypeTemplates>
</SCL>
```

APPENDIX H. CMC 256*plus* test set configuration

The CMC 256*plus* test set is an advanced IEC 61850 IED tester and calibration tool which was used as both a current/voltage signal source and sampled value message publisher. This test set was used in conjunction with Omicron Test Universe modules to generate voltage and current signals which are injected into the GOOSE monitoring node and limited-function Merging Unit (MU) prototypes. The GOOSE monitoring node an the MU prototypes are a combination of GOOSE/SV message mapping devices based on the Xilinx FPGA and an Analogue Front-End (AFE) module.

In Chapter Five, multiple injection tests were conducted to evaluate the performance and measurement accuracy of the GOOSE monitoring node and MU prototypes. The voltage and current signals generated by the CMC 256*plus* test set emulated secondary signals from voltage and current transformers (VTs and CTs) connected to a power system under nominal and faults conditions. To achieve this, the *Ramping*, *Control Centre*, *QuickCMC*, sampled value configuration and *Harmonics* modules of Test Universe software were used. The *TransView* module of the Test Universe software was used for viewing the captured COMTRADE files for accuracy and performance evaluation.

Figure_Appendix H.1 shows Test Universe modules used for configuring the CMC 256*plus* test set.



Figure_Appendix H.1: **Test Universe modules used for testing the developed MU and GOOSE publisher VHDL modules**

In Figure_Appendix H.1, the modules/functions marked from (i) to (vi) are discussed below.

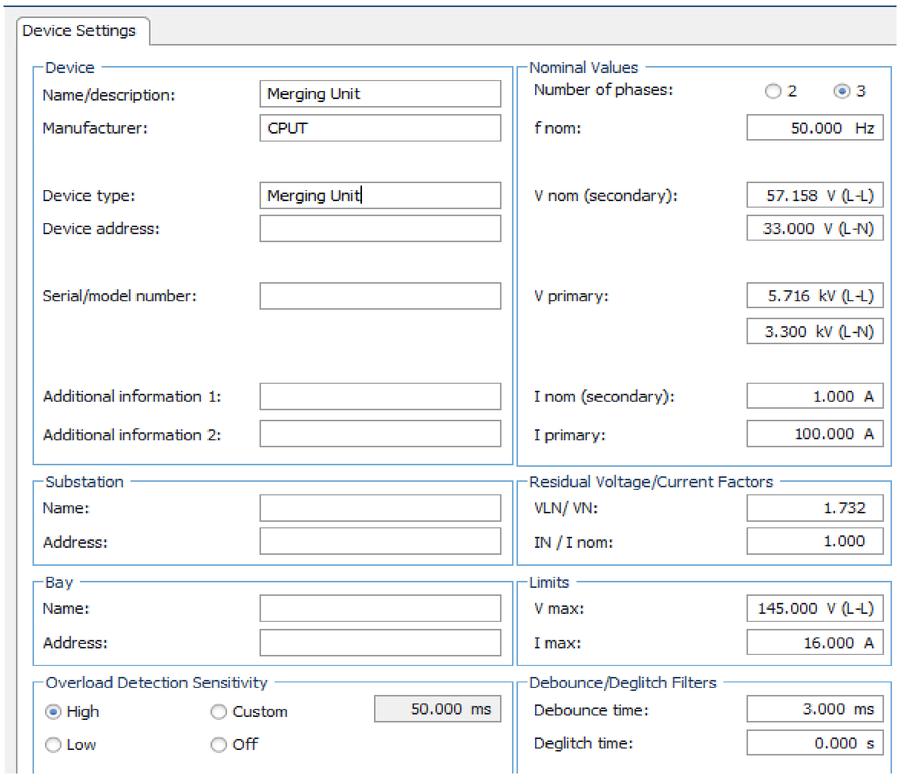
i. Test Set Association: this is the initial step required in order to associate the CMC 256*plus* test set to the configuration PC.

ii. *QuickCMC*: This module is used for generating nominal voltage and current signals (33 V_{RMS} and 1 A_{RMS} at 50 Hz) for the following:
   a. MU prototype sampled value accuracy evaluation.
   b. Validating the structure of sampled value messages published by the MU prototype.
   c. Validating the structure of GOOSE messages published by the GOOSE monitoring node prototype.
   d. GOOSE monitoring node prototype RMS voltage calculation accuracy evaluation
   e. Evaluating the frequency measurement range and accuracy of the developed limited-function MU prototype.

iii. *Control Center*: This creates multi-function test procedures by combining different modules. The control center will be used for configuring the CMC 256*plus* test set to generate voltage and current signals and publish sampled value messages by combining the *QuickCMC* and the *sampled value configuration* modules.

iv. *Ramping*: This module will be used for generating multiple voltage and current signal ramps to emulate power system faults. This module was used for emulating the following:
   a. Voltage-sag,
   b. Voltage-swell and
   c. Frequency variation faults
v. *TransView*: this tool was used for viewing COMTRADE files.
vi. *Harmonics*: this module was used for configuring the CMC 256*plus* test to generate voltage and current signals with harmonic components to emulate harmonic distortions.

When generating current/voltage signals using the Test Universe and the CMC 256*plus* test set, nominal and maximum voltage and current values can be set before the injection test to avoid damaging the device under test (DUT). For the test procedures discussed in this research project, the nominal and maximum voltage and current signals for the DUT are set to 33 V_{RMS}/1 A_{RMS} and 84 V_{RMS}/16 A_{RMS} respectively. The DUT hardware configuration is shown in Figure_Appendix H.2.


Device Settings

Device

Name/description:	Merging Unit
Manufacturer:	CPUT
Device type:	Merging Unit
Device address:	
Serial/model number:	
Additional information 1:	
Additional information 2:	

Nominal Values

Number of phases:	2 [✓ 3]
f nom:	50.000 Hz
V nom (secondary):	57.158 V (L-L)
	33.000 V (L-N)
V primary:	5.716 kV (L-L)
	3.300 kV (L-N)
I nom (secondary):	1.000 A
I primary:	100.000 A

Substation

Name:	
Address:	

Residual Voltage/Current Factors

VLN/ VN:	1.732
IN / I nom:	1.000

Bay

Name:	
Address:	

Limits

V max:	145.000 V (L-L)
I max:	16.000 A

Overload Detection Sensitivity

[✓ High] Custom 50.000 ms
Low Off

Debounce/Deglitch Filters

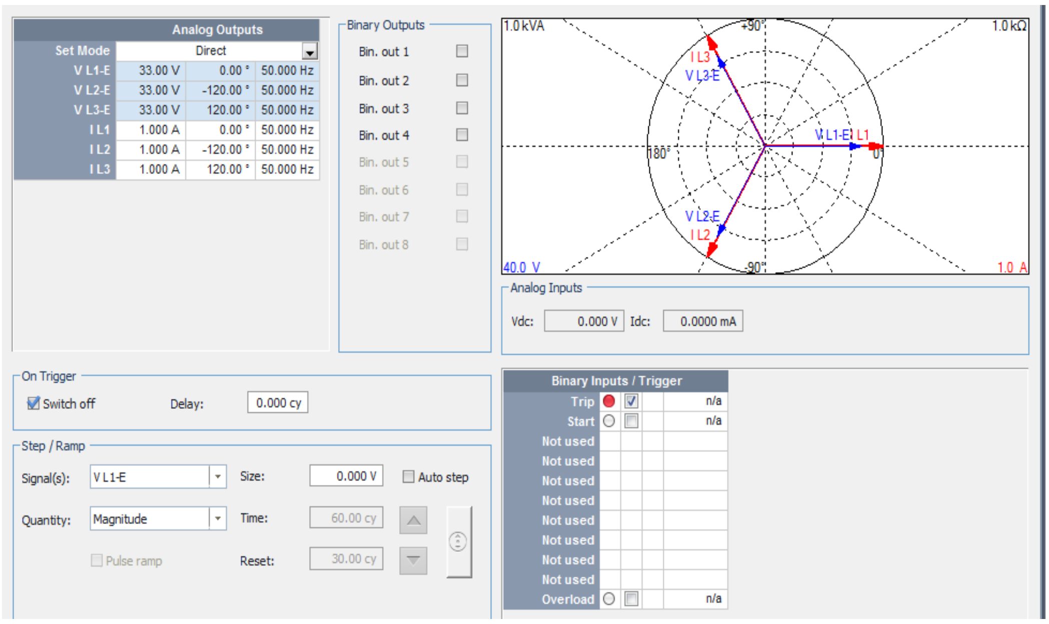
Debounce time:	3.000 ms
Deglitch time:	0.000 s

Figure_Appendix H.2: **Setting device under test parameters**

In the following sections; Appendix H.1 to Appendix H.6 shows how the different Test Universe modules were used to generate current/voltage signals emulating different power system conditions.

Appendix H.1 : Nominal signals injection

The CMC 256*plus* test set was configured to generate nominal voltages and current using the *QuickCMC* module as shown in Figure_Appendix H.3.



Analog Outputs			
Set Mode	Direct		
V L1-E	33.00 V	0.00 °	50.000 Hz
V L2-E	33.00 V	-120.00 °	50.000 Hz
V L3-E	33.00 V	120.00 °	50.000 Hz
I L1	1.000 A	0.00 °	50.000 Hz
I L2	1.000 A	-120.00 °	50.000 Hz
I L3	1.000 A	120.00 °	50.000 Hz

Binary Outputs: Bin. out 1 ☐, Bin. out 2 ☐, Bin. out 3 ☐, Bin. out 4 ☐, Bin. out 5 ☐, Bin. out 6 ☐, Bin. out 7 ☐, Bin. out 8 ☐

Analog Inputs: Vdc: 0.000 V Idc: 0.0000 mA

On Trigger: ✓ Switch off Delay: 0.000 cy

Step / Ramp: Signal(s): V L1-E Size: 0.000 V ☐ Auto step; Quantity: Magnitude Time: 60.00 cy; ☐ Pulse ramp Reset: 30.00 cy

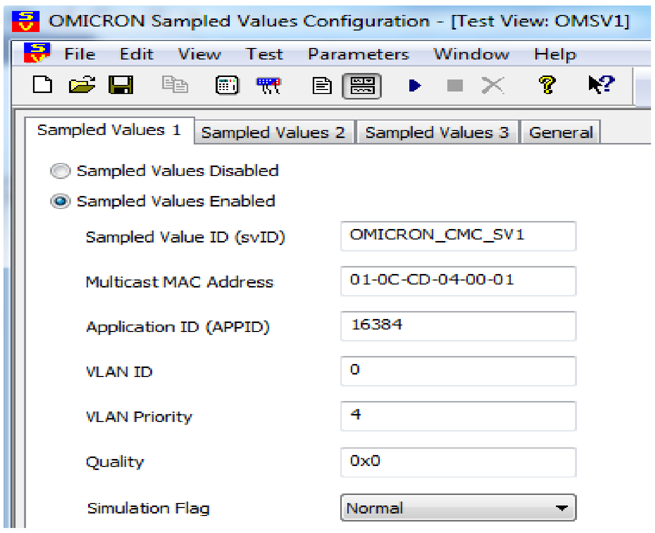
Binary Inputs / Trigger				
Trip	●	✓		n/a
Start	○	☐		n/a
Not used				
Not used				
Not used				
Not used				
Not used				
Not used				
Not used				
Not used				
Overload	○	☐		n/a

Figure_Appendix H.3: *QuickCMC* configuration for generating nominal voltage and current signals

Appendix H.2 : CMC 256*plus* test set sampled values publisher

The CMC 256*plus* test set was configured to generate voltage/current signals and at the same time publish sampled value messages containing samples of the generated signal. This multi-function test setup was used for evaluating the magnitude accuracy and phase angles of sampled values published by the developed limited-function MU prototype. For this function the Control Centre module was used for combining the sampled value configuration and the *QuickCMC* functions. The *QuickCMC* module configured in Appendix H.1 is adopted into this multi-functional test setup.

The IEC 61850 sampled value module was configured to generate one SV stream as shown in Figure_Appendix H.4.

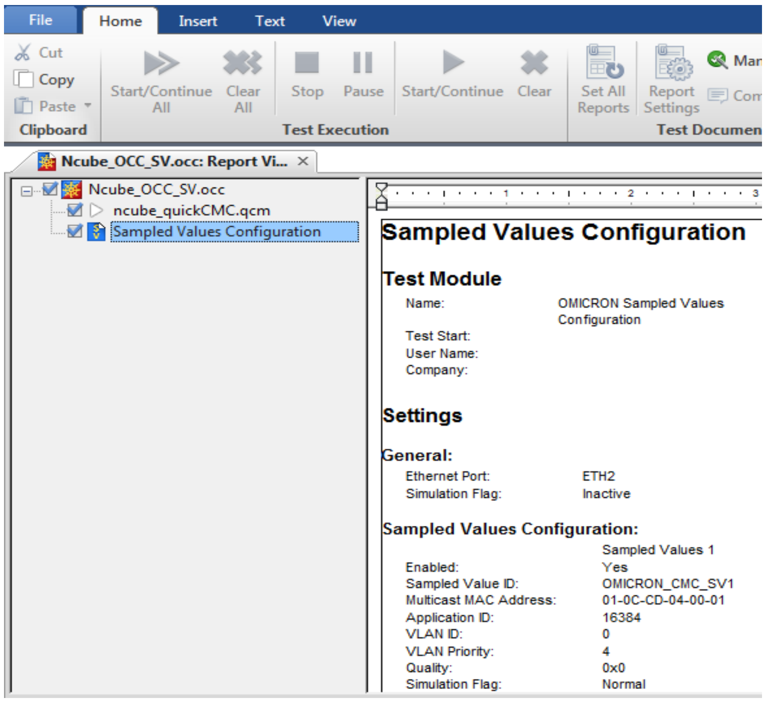

OMICRON Sampled Values Configuration - [Test View: OMSV1]
File Edit View Test Parameters Window Help
Sampled Values 1
Sampled Values 2
Sampled Values 3
General

Sampled Values Disabled
(●) Sampled Values Enabled

Sampled Value ID (svID)	OMICRON_CMC_SV1
Multicast MAC Address	01-0C-CD-04-00-01
Application ID (APPID)	16384
VLAN ID	0
VLAN Priority	4
Quality	0x0
Simulation Flag	Normal

Figure_Appendix H.4: **CMC 256*plus* IEC 61850 sampled value module**

The resulting multi-functional module is shown in Figure_Appendix H.5



Sampled Values Configuration

Test Module

Name:	OMICRON Sampled Values Configuration
Test Start:	
User Name:	
Company:	

Settings

General:

Ethernet Port:	ETH2
Simulation Flag:	Inactive

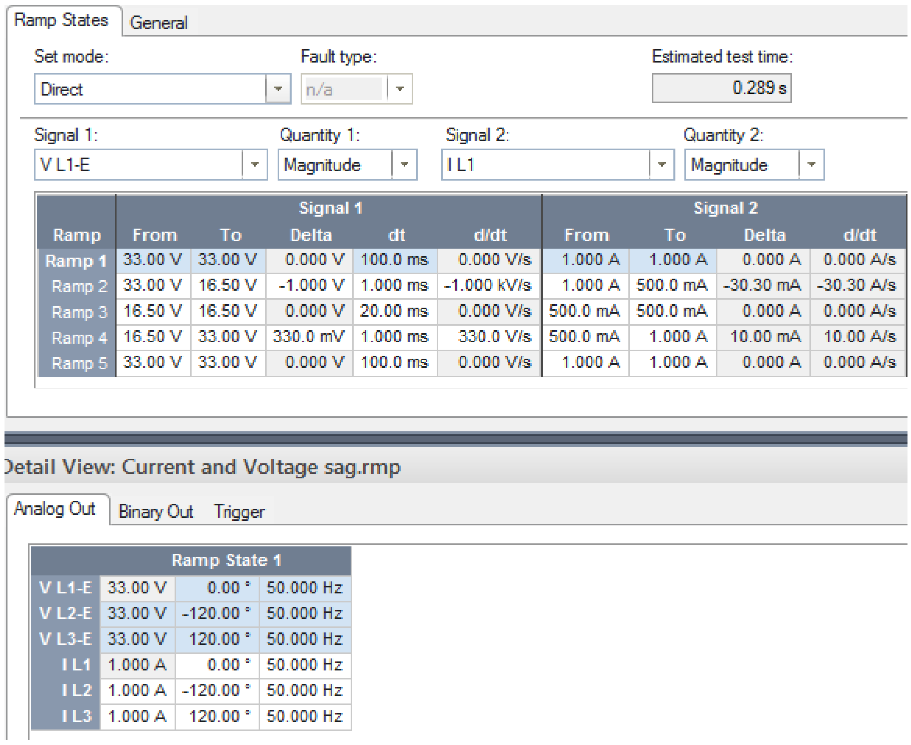
Sampled Values Configuration:

	Sampled Values 1
Enabled:	Yes
Sampled Value ID:	OMICRON_CMC_SV1
Multicast MAC Address:	01-0C-CD-04-00-01
Application ID:	16384
VLAN ID:	0
VLAN Priority:	4
Quality:	0x0
Simulation Flag:	Normal

Figure_Appendix H.5: ***Control Centre* configuration for *QuickCMC* and *SV configuration* multi-function test**

Appendix H.3 : Voltage-sag fault simulation

A voltage-sag fault condition is simulated using the Test Universe *Ramping* module which when configured can cause the generated voltage and current signal to ramp up or down at rates faster than 1 ms. This is achieved by executing five ramp functions consecutively as shown in Figure_Appendix H.6. During this simulation, phase B and C voltage and current signals are kept constant at nominal values while phase A values dip to 16.5 V_{RMS} and 0.5 A_{RMS} respectively.



Set mode: Direct
Fault type: n/a
Estimated test time: 0.289 s

Signal 1: V L1-E
Quantity 1: Magnitude
Signal 2: I L1
Quantity 2: Magnitude

	Signal 1					Signal 2			
Ramp	From	To	Delta	dt	d/dt	From	To	Delta	d/dt
Ramp 1	33.00 V	33.00 V	0.000 V	100.0 ms	0.000 V/s	1.000 A	1.000 A	0.000 A	0.000 A/s
Ramp 2	33.00 V	16.50 V	-1.000 V	1.000 ms	-1.000 kV/s	1.000 A	500.0 mA	-30.30 mA	-30.30 A/s
Ramp 3	16.50 V	16.50 V	0.000 V	20.00 ms	0.000 V/s	500.0 mA	500.0 mA	0.000 A	0.000 A/s
Ramp 4	16.50 V	33.00 V	330.0 mV	1.000 ms	330.0 V/s	500.0 mA	1.000 A	10.00 mA	10.00 A/s
Ramp 5	33.00 V	33.00 V	0.000 V	100.0 ms	0.000 V/s	1.000 A	1.000 A	0.000 A	0.000 A/s

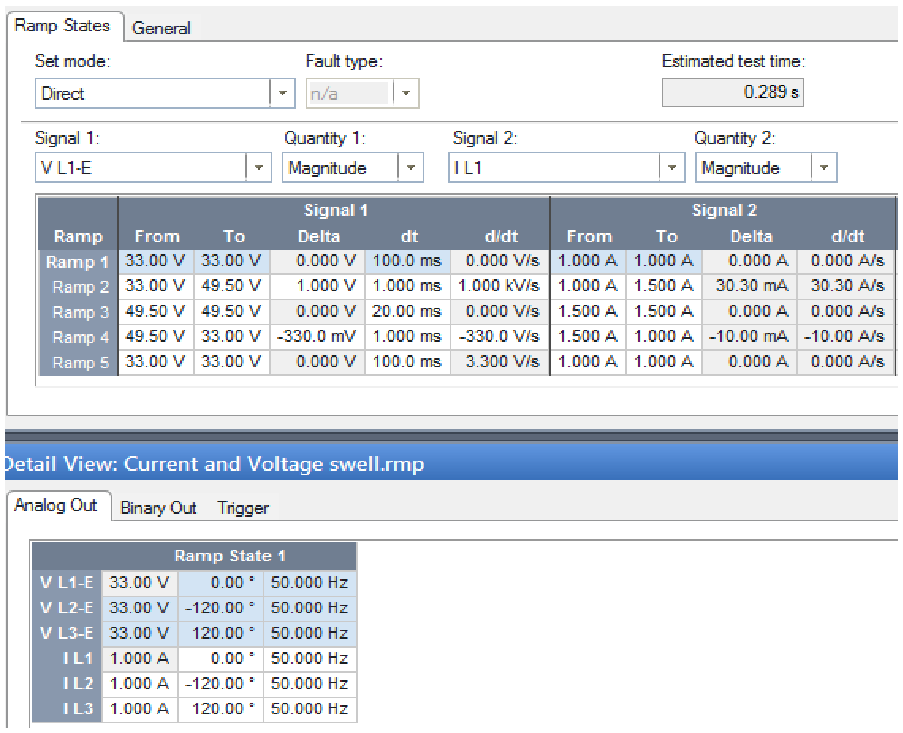
Detail View: Current and Voltage sag.rmp

	Ramp State 1		
V L1-E	33.00 V	0.00 °	50.000 Hz
V L2-E	33.00 V	-120.00 °	50.000 Hz
V L3-E	33.00 V	120.00 °	50.000 Hz
I L1	1.000 A	0.00 °	50.000 Hz
I L2	1.000 A	-120.00 °	50.000 Hz
I L3	1.000 A	120.00 °	50.000 Hz

Figure_Appendix H.6: Voltage-sag fault emulation using the CMC 256*plus* test set

Appendix H.4 : Voltage-swell fault simulation

Similarly to the voltage-sag fault simulation, the CMC 256*plus* test was configured to generate voltage and current signals simulating a voltage-swell fault using the *Ramping* module. For this experiment, the phase A voltage and current signals were configured to swell and peak of 49.5 V_{RMS} and 1.5 A_{RMS} respectively while phases B and C signals remain constant as shown in Figure_Appendix H.7.



Set mode: Direct

Fault type: n/a

Estimated test time: 0.289 s

Signal 1: V L1-E

Quantity 1: Magnitude

Signal 2: I L1

Quantity 2: Magnitude

	Signal 1					Signal 2			
Ramp	From	To	Delta	dt	d/dt	From	To	Delta	d/dt
Ramp 1	33.00 V	33.00 V	0.000 V	100.0 ms	0.000 V/s	1.000 A	1.000 A	0.000 A	0.000 A/s
Ramp 2	33.00 V	49.50 V	1.000 V	1.000 ms	1.000 kV/s	1.000 A	1.500 A	30.30 mA	30.30 A/s
Ramp 3	49.50 V	49.50 V	0.000 V	20.00 ms	0.000 V/s	1.500 A	1.500 A	0.000 A	0.000 A/s
Ramp 4	49.50 V	33.00 V	-330.0 mV	1.000 ms	-330.0 V/s	1.500 A	1.000 A	-10.00 mA	-10.00 A/s
Ramp 5	33.00 V	33.00 V	0.000 V	100.0 ms	3.300 V/s	1.000 A	1.000 A	0.000 A	0.000 A/s

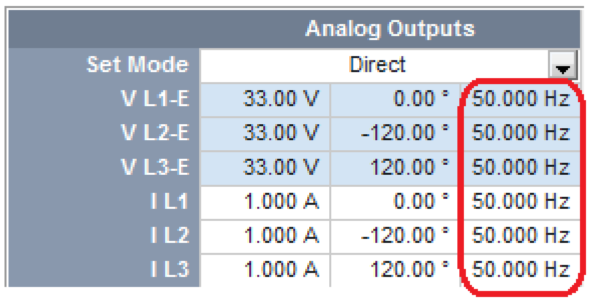
Detail View: Current and Voltage swell.rmp

	Ramp State 1		
V L1-E	33.00 V	0.00 °	50.000 Hz
V L2-E	33.00 V	-120.00 °	50.000 Hz
V L3-E	33.00 V	120.00 °	50.000 Hz
I L1	1.000 A	0.00 °	50.000 Hz
I L2	1.000 A	-120.00 °	50.000 Hz
I L3	1.000 A	120.00 °	50.000 Hz

Figure_Appendix H.7: Voltage/current swell simulation using the CMC 256*plus* test set

Appendix H.5 : Power system frequency variation simulation

The Test Universe *QuickCMC* module was used to generate voltage and current signals with the system frequency changed between 10 Hz, 50 Hz and 100 Hz for each injection test. Figure_Appendix H.8 show the *QuickCMC* module setting changed to produce voltage and current signals at different frequencies

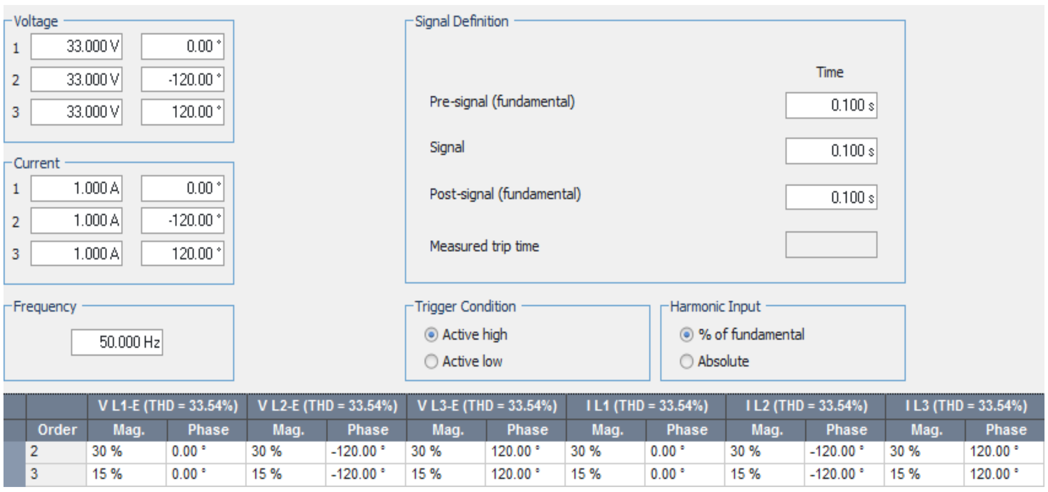


	Analog Outputs		
Set Mode	Direct		
V L1-E	33.00 V	0.00 °	50.000 Hz
V L2-E	33.00 V	-120.00 °	50.000 Hz
V L3-E	33.00 V	120.00 °	50.000 Hz
I L1	1.000 A	0.00 °	50.000 Hz
I L2	1.000 A	-120.00 °	50.000 Hz
I L3	1.000 A	120.00 °	50.000 Hz

Figure_Appendix H.8: Power system frequency variation using the *QuickCMC* module

Appendix H.6 : Harmonics distortion simulation

Figure_Appendix H.9 shows the CMC 256*plus* test set setup for generating voltage and current signals with harmonic distortions emulating a condition is power network whereby load draws a non-linear current. This test setup shown in Figure_Appendix H.9 configures the CMC 256*plus* test set to generate voltage and current signals emulating harmonic disturbances in power systems.



Voltage

1	33.000 V	0.00 °
2	33.000 V	-120.00 °
3	33.000 V	120.00 °

Current

1	1.000 A	0.00 °
2	1.000 A	-120.00 °
3	1.000 A	120.00 °

Frequency: 50.000 Hz

Signal Definition

	Time
Pre-signal (fundamental)	0.100 s
Signal	0.100 s
Post-signal (fundamental)	0.100 s
Measured trip time	

Trigger Condition: [✓ Active high] Active low

Harmonic Input: [✓ % of fundamental] Absolute

	V L1-E (THD = 33.54%)		V L2-E (THD = 33.54%)		V L3-E (THD = 33.54%)		I L1 (THD = 33.54%)		I L2 (THD = 33.54%)		I L3 (THD = 33.54%)	
Order	Mag.	Phase	Mag.	Phase	Mag.	Phase	Mag.	Phase	Mag.	Phase	Mag.	Phase
2	30 %	0.00 °	30 %	-120.00 °	30 %	120.00 °	30 %	0.00 °	30 %	-120.00 °	30 %	120.00 °
3	15 %	0.00 °	15 %	-120.00 °	15 %	120.00 °	15 %	0.00 °	15 %	-120.00 °	15 %	120.00 °

Figure_Appendix H.9: ***Harmonics*** **module setup for 2nd and 3rd order harmonic injection**