IEC 61850-9-2 Process Bus Communication Interface for Light Weight Merging Unit Testing Environment

PENGCHENG ZHAO

KTH Electrical Engineering

Master's Degree Project
Stockholm, Sweden
August, 2012

XR-EE-ICS 2012:016

Dedicate my master thesis to my dear parents:
Mr. Lianmo Zhao and Mrs. Guiping Peng

献给我敬爱的父亲赵连默和母亲彭桂平

Life is short and tough, I appreciate everything that life has given and will give to me. I hope every one could enjoy his/her life. Thanks to every one I have ever met and talked. It is my honour to meet and talk to you.

Abstract

IEC 61850 is an international standard for communication networks for substation [1]. As a part of the IEC 61850 series, IEC 61850 Part 9-2 was introduced in 2004 and updated to a stable version in 2011 [2]. Part 9-2 defines the specific communication service mapping for the transmission of sampled values from process equipment [3]. The interface between process equipment and bay level devices is named as Merging Unit (MU) [4].

The 9-2 standard is a newly updated standard which has not yet been implemented in large scale. Therefore it is new or unfamiliar to many engineers working with substation automation systems. A project which is aimed to develop a light weight IEC 61850-9-2 MU testing environment is conducted in ICS department in KTH. The project can be also referred as "soft" MU project and consists of two parts. Part A is to implement conventional power system model, instrument transformer, and analog to digital converter [5]. This thesis project is Part B of the light weight merging unit (MU) testing tool project.

This project is focusing on developing the 9-2 process bus communication interface for the "soft" MU project. The digitalized measurements such as current and voltage from Project A are encoded into 9-2 sampled values (SV) Ethernet stream by the process bus interface. The project is executed in the following steps. Firstly, the interface is programmed by C language under Linux environment. Secondly, the process bus communication interface is embedded into Matlab so that Part A and Part B can be combined as one system. To evaluate the light weight IEC 61850-9-2 MU testing environment, the "soft" MU is connected to a process bus network with one IED. The IED used in this thesis project is RET 670 which is a product of ABB Substation Automation. The four steps over current protection and two Windings transformer differential protection in RET 670 is tested. The evaluation work is also demonstrated in this report.

Acknowledgements

First of all, I would like give my biggest thanks to my parents. Without their support on all aspects, I could not have so much wonderful experience in my life and this master thesis report could never be finished. Their love always gives me the strength to go through all difficulties in my life. Here I must give my special thanks to my mother whose greatness keeps inspiring me. My father had been suffered from the Non-Hodgkin lymphoma for more than thirteen years, during which, my mother takes over all the responsibilities of our family and brings me grown-up. My father had passed away during the report writing. May my graduation comfort his soul in the heaven.

Thanks my supervisor, Ph.D. Nicholas Honeth, who provided great help on both technical and administration issues during my master thesis project. Thanks to him for dedicating his working time to my project. I have learned a lot of knowledge from him. His professional suggestions always lead me to the right direction.

I also want to thank my examiner, Prof. Lars Nordstrm. The course held by him, Computer Application in Power System, introduced me to the field of ICT in power system and developed my academic interest in this field. It is him and my supervisor who make this master thesis project possible.

Thanks the industry reviewers in ABB, Mr. Johan Salj and Mr. Klas Koppari. This project benefited a lot from their industrial experience. I would also give my thanks to them for arranging meeting and contacting technical persons for Zeeshan and I when we visited ABB Substation Automation. Thanks all the people who had given their help in ABB Substation Automation, Västerås.

What is more, thanks all the persons I worked with in the Industrial information & Control Systems (ICS), KTH. The time I worked with you is the best working time I have spent in Sweden.

Last but not least, I would like to thank my project partner, Mr. Zeeshan Ali Khurram. He is awesome. Thanks him for sharing a lot of his great experience with me. It is a great experience for me to listen to his talking.

Table of Contents

List of Figures

List of Tables

Acronyms

1PPS One Pulse Per Second.

ADC Analogue to Digital Converter.

APDU Application Protocol Data Unit.

ARP Address Resolution Protocol.

ASCII American Standard Code for Information Interchange.

ASDU Application Service Data Unit.

ASN.1 Abstract Syntax Notation One.

GOOSE Generic Object Oriented Substation Event.

HMI Human Machine Interface.

ICT Information and Communication Technology.

IED Intelligent Electronic Device.

IP Internet Protocol.

IPv4 Internet Protocol version 4.

LED Light-Emitting Diode.

MAC Media Access Control.

MU Merging Unit.

OSI Open Systems Interconnection.

PC Personal Computer.

RMS Root Mean Square.

RTDS Real Time Digital Simulator.

SAS Substation Automation System.

SV Sampled Values.

UDP User Datagram Protocol.

Chapter 1

Introduction

The power system has been greatly developed since the invention of electricity. The architecture developed in 20^{th} century can not fulfil the increasing requirement now. The architecture should be changed from the bottom. [11] A multilayer information and control system architecture is proposed for the future power grid. The power transmission layer is an important part. [1] In the power transmission system, the substation plays a significant role since it transfers the voltage from low to high or the other way around by utilizing the power transformer. In the substation, the monitoring, control, protection and other automation functions are provided by the Substation Automation System (SAS). [12] IEC 61850 defines the communication network for the protection and automation in the substation [13]. The communication network within the substation is divided into three levels: process, bay and station. The communication network between primary equipment and bay level is referred as the process bus. [6] This thesis contributes effort on the communication aspect especially the process bus Sampled Values (SV) service in the SAS.

In order to make the readers familiar with this topic, the terms related to this topic and the scope of this project are introduced in this chapter.

1.1 Smart Grid

The demand of electricity generation is increasing largely. At the same time, the environment issues such as reducing carbon dioxide emissions and minimizing the effects of climate change are more and more concerned by public as well. As a result, the renewable and clear energy is preferred and a lot of green energy power plant is under construction. The increasing of demand and more distributed generation requires a more intelligent power grid in the future which can control and distribute the energy flow efficiently. The smart grid is the vision of future power grid.[14]

The smart grid is a fully automated power delivery grid which allows bidirectional flow of the information and electricity. It has distributed intelligence and equipped with broadband communications and automated control systems which enables the real-time management. The smart grid also seamless interfaces among all the actors in power grid which include the industrial users, end-user costumers, building automation system, industrial plants, energy storage installations and the electric network. [15, 16]

1.2 Substation Automation System

The electricity is generated by the power plant (e.g. nuclear power plant). The power generated needs to be delivered to the consumers. The substation is an important part of the power distribution grid [12]. The early technology used the fuse to protect the power transformer in the substation. As the technology developing, the electromechanical relay is introduced to replace the fuse. In a modern substation, the old electromechanical relay is replaced by the microprocessor-based relay which is one type of Intelligent Electronic Devices (IEDs). IEDs can provide more features and capabilities like protection, monitoring, data logging etc. than the devices based on the old technology. [17] To protect the power equipment, a lot of parameters such as the current and voltage needs to be measured from the system and the IED should be able to get the measurement to perform the protection functions. Furthermore, the bay level IEDs in the substation need to communicate with each other in order to execute interlock functions and other complicate tasks. In the substation the SAS provides monitoring, control, protection, communication and other automation functions [12].

1.3 IEC 61850

The IEC 61850 standard mainly defines the information, information exchange and configuration for protection, monitoring and control in the substation. [1] In [16], the IEC 61850 is listed as a related standard for achieving smart grid vision. With the IEC 61850, the modern Information and Communication Technology (ICT) is utilized as the communication solution. For instance, the fibre optic cables, network switches and routers are introduced and the amount of copper wires are

greatly reduced. By applying the IEC 61850, the information in a substation is transferred in digital form so that a lot of microcomputer-based devices and automation solutions can be applied in the substation.

There are considerable benefits by installing the IEC 61850 technology. Figure 1.1 below shows the comparison between a traditional copper wired technology and the IEC 61850 installation.

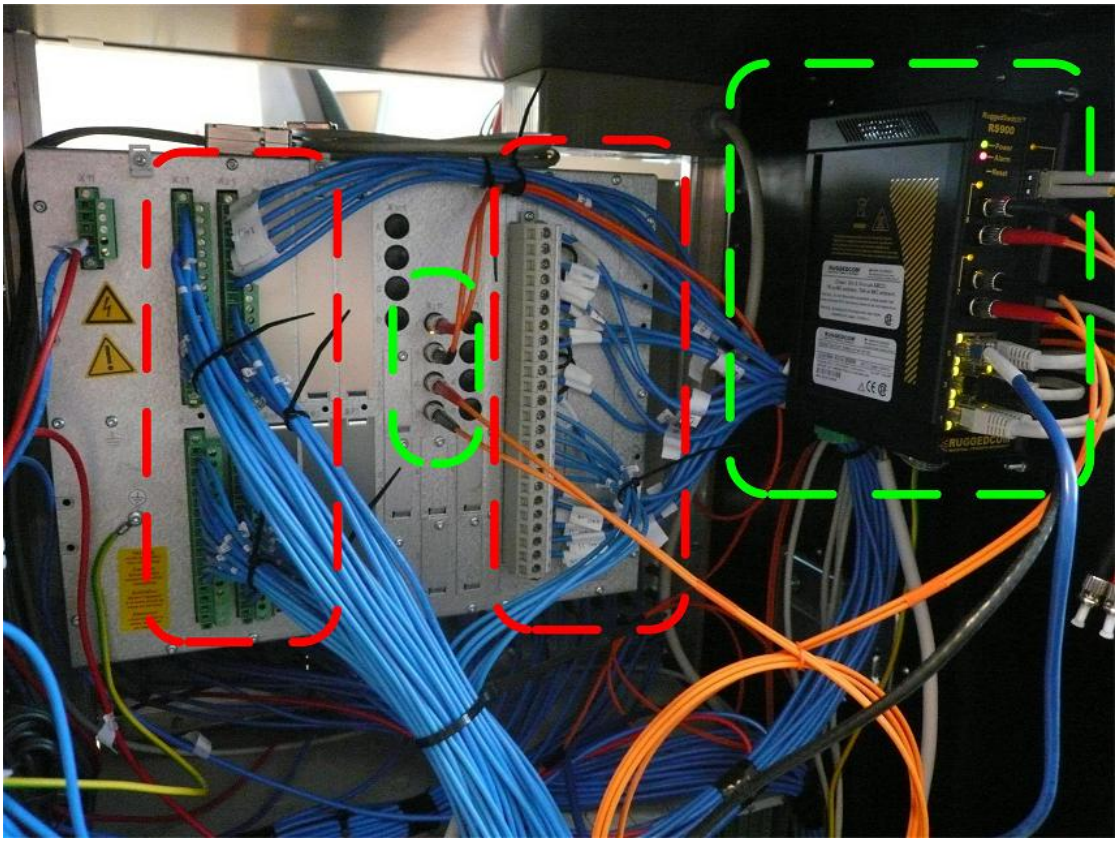


Figure 1.1: The back view of RET 670

The RET 670 in the lab is configured with both copper wired technology and IEC 61850. The parts marked with green color in Figure 1.1 are the fibre optical cables and the Ruggedcom switch which are the IEC 61850 installation. The red color parts with blue copper cables are the conventional copper wired solution for the protection relay. It is obvious that the IEC 61850 technology saves a lot of the copper wires and easy to map.

1.4 IEC 61850-9-2 Process Bus and Merging Unit (MU)

The communication network within the substation is divided into three levels: process, bay and station. The process level includes the switchgear equipment, actuators and sensors. [6] The process bus is the communication network between the process level and bay level IEDs. [6]

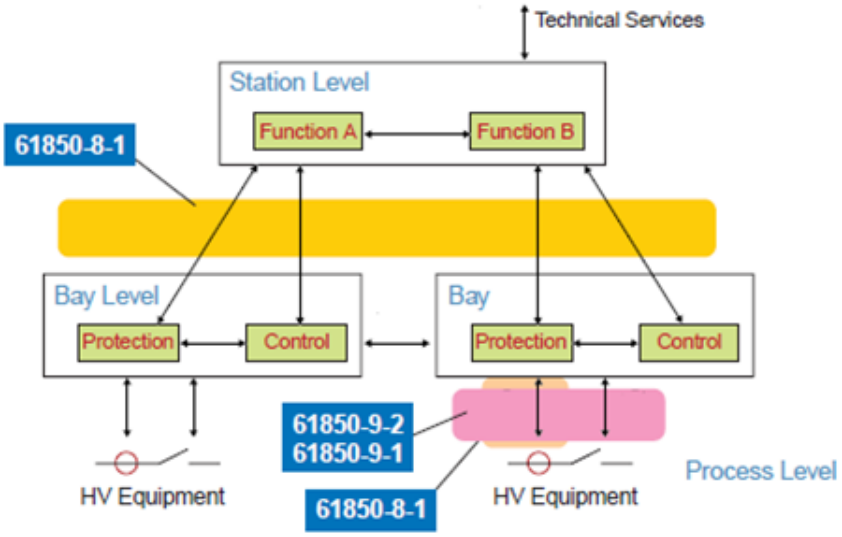


Figure 1.2: The three level hierarchy of SAS according to IEC 61850 [6]

Although in general the IEC 61850 is being widely used, the IEC 61850 Part: 9-2 process bus standard has only been applied in some pilot installations. The benefits of introducing the process bus are obvious. There will be a great reduction of the copper wiring requirement to interface to the process equipment compared with the traditional approach. In addition, with the digitalized information system, automated testing is possible which makes the implementation procedure easier. [18]

As mentioned in [12], the circuit breaker and current transformers are process level devices. To exchange information between electronic current transformer and the bay level equipment, Merging Unit is introduced. It is first defined in IEC 60044-8 as shown in Figure 1.3. [4]

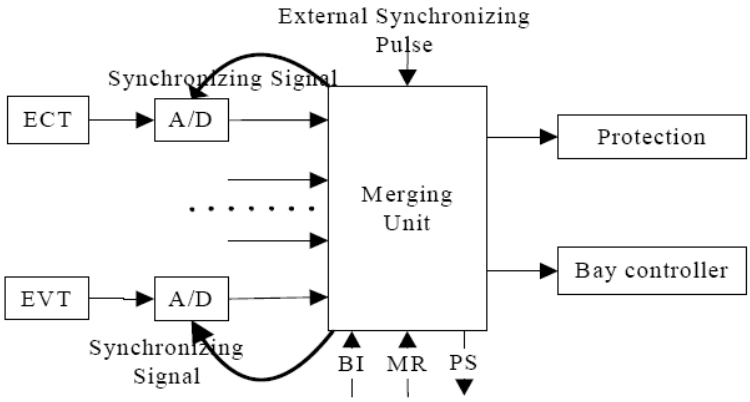


Figure 1.3: The definition of MU in IEC 60044-8 [4]

As shown in Figure 1.3, the analogue data from the Electronic Current/Voltage Transformer (ECT/EVT) is converted to digital form first and the MU is

responsible for collecting the digital data synchronously and transmitting these measurement to bay level devices [4]. In this project, the synchronization is not implemented.

1.5 Need for Testing

The testing here does not mean the testing for process bus communication itself. It means the testing for the protection through the process bus. In [19], the importance of testing is emphasized. The protective relays need to be thoroughly tested before they can operate in a real system. An evaluation tool is wanted by the vendors to validate the design of the protection logic. At the same time, the field engineers need a tool to calibrate relay settings and perform troubleshooting. [19] To evaluate the protection logic and relay settings through process bus with Merging Unit (MU), there are several options, such as the physical MU as shown in [7], the MU model in real-time digital simulator (such as Opal-RT), the bench-top testing tool like the Omicron tool etc. However, all of them are either expensive or inflexible. This point motivates the proposal of this project.

However, to develop a industry level MU testing tool is a too ambitious goal for a master thesis. As a result, the feasibility is taken into account and a manageable goal is set and it is presented in next section.

1.6 Scope

It is already mentioned in the Section 1.4, the MU is the interface between the process equipment and process bus. Motivated by the need for testing, this project is proposed as that it is aim to develop a light weight MU testing environment for the educational usage. The testing environment can be used as a demonstration tool for the process bus concept. Also it can perform basic simulation test to roughly prove the design idea on process bus. The light weight is the core idea. The testing environment should be easily accessible and not require high performance, complicate and costly hardware. The detailed scope is defined in the following sections.

1.6.1 The Light Weight MU Testing Environment Project

As mentioned in the abstract, this project is the Part B of the light weight MU testing environment project or "soft" MU project for short. The scope of the light weight MU testing environment project is illustrate in Figure 1.4.

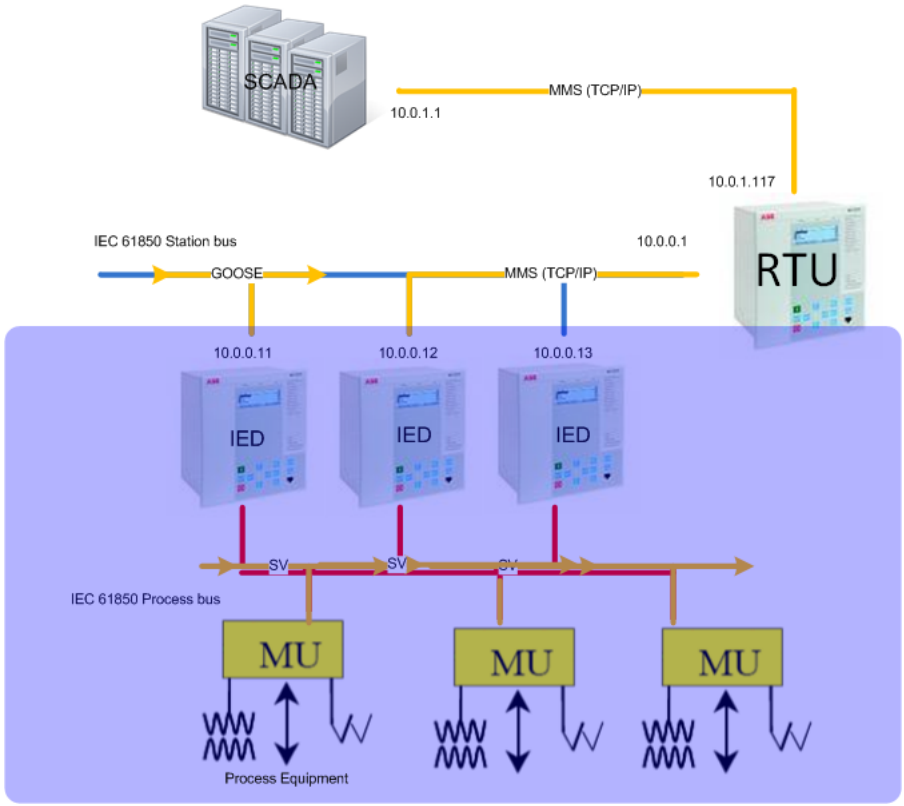


Figure 1.4: The substation architecture

As shown in Figure 1.4, the communication in a substation begins from the process equipment. The MU is the interface between the process equipment and process bus. The bay level devices acquire the measurement from the process bus communication. The station bus is used to connect the bay level devices and other systems like SCADA together. The "soft" MU project simulates the parts before the station bus. The area with light blue color indicates the parts related to the "soft" MU project. The light weight MU testing environment includes the simulated process equipment, model of MU and the process bus network. The MU model generate the SV traffic which is fed in the process bus network so that the IEDs can collect the measurement. In this project, the IED is a physical device RET 670 which is manufactured and supported by ABB. The "soft" MU project is divided into two sub-projects Project A and B. Project A focuses on the designing the process equipment model and the Analogue to Digital Converters (ADCs)in MU model which is reported in [5].

1.6.2 Scope of Project B

The components in a MU model are shown in Figure 1.5 below. The scope of Project B is more concentrated on the process bus communication interface of the MU model and the process bus network set-up.

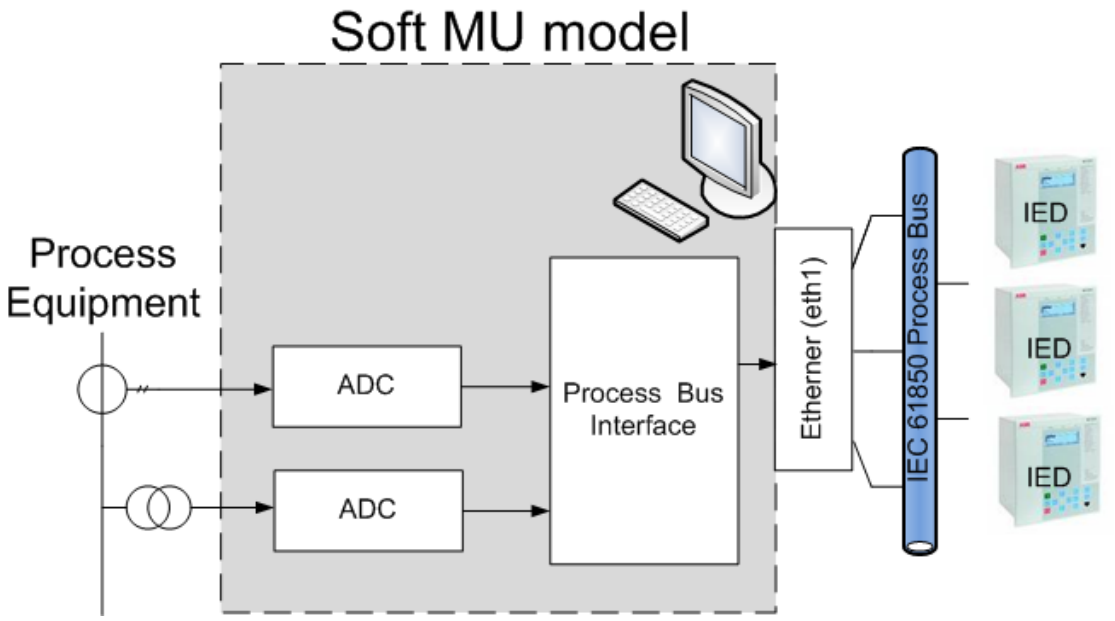


Figure 1.5: The components in the MU model

Figure 1.5 shows clearly that the analogue measurement from the process equipment first is digitalized. The digitalized data then is encapsulated as SV frame by the process bus interface. The SV frame is transferred into the process bus. The goal of this project is to develop a software interface that can complete the work of acquiring the digitalized data from Project A, encapsulating SV frame and transferring them to the process bus. That is to say, the process bus interface shown in Figure 1.5 is to be developed.

Another task of this project is to evaluate the process bus interface. To evaluate the process bus interface, a physical process bus network is established. The configurations of the process bus network and the bay device (RET 670) are done together with my project partner Zeeshan Ali Khurram who is responsible for Project A.

1.6.3 Limitation

This "soft" MU testing environment is designed to be used for educational purpose or act as a process bus concept demonstration tool. It can roughly test the protection logic in the substation which means it can prove the logic of the design but not provide the detailed performance analysis such as the response time of the protection function, the delay time due to communication hardware, etc.

The synchronization between the MU and IEDs is not studied in this project. It is recommended as a continuous work of this project.

In addition, this project is a practical implementation of the process bus technology. The performance analysis of the process bus is out of the scope of this project as well.

Chapter 2

Background

In this chapter, background information is presented. First, the related work has been done on this topic is briefly discussed. Second, some background theories which are necessary to know to understand this report are explained.

2.1 Related Work

Since the process bus has been introduced fairly recently, there is not much material about implementation experience in this field and it is even more difficult to find the material about the process bus testing tool. Most of the literature can be found is either the illustration of process bus standard or the process bus performance evaluation in the laboratory.

2.1.1 Evaluation of 9-2 Process Bus

As the IED is developing, there are more and more digital systems introduced in power system and the conventional protection system is replace with the digital system. The testing for conventional protection system is already well performed while the evaluation of digital systems in power industry is still ongoing. [7] Several feasibility tests are demonstrated in [20]. Those tests includes the delay time of SV on Ethernet network, performance of IED's CPU when it is loaded with SV stream. In the tests, the synchronization accuracy is $\pm 20\mu$s. The maximum delay time for SV (from primary equipment to the receiver) is 3.520ms. That fulfils the real-time requirement of control and protection for substation automation. In [20], it is also mentioned that the advanced Ethernet equipment should be utilized to achieve correct operation.

Similar results are get in [13]. It evaluates the process bus from the Ethernet communication point of view. Dynamic communication simulation tool, OPNET, is used to simulate the communication network. A process bus for 345/230kV substation is simulated. According to IEC 61850, the allowed communication delay for time-critical packet such as process bus communication (including GOOSE and SV) is 3 to 4 ms. The results in [13] show that the delay time of SV (sample rate 1920-4800Hz) is within the range by using 100 Mb/s fibre optical communication link. In this project, the sample rate is 4000Hz.

One of the most significant advantages of IEC 61850 is the compatibility and interoperability which means different digital devices from different vendors can be utilized in one system and the system will perform properly. [19] Paper [19] evaluates the performance of IEC 61850-9-2 on those aspects. Positive results are given in this paper which indicates that the compatibility and interoperability can be achieved by this novel standard.

2.1.2 Implementation Example of 9-2

Since there are few practical implementations in the industry, the major vendors together in power industry provide an implementation guidance for IEC 61850-9-2 in order to have a uniform interpretation of the IEC standard. This guidance is named as "Implementation Guidance for Digital Interface to Instrument Transformers Using IEC 61850-9-2" [10]. This is also used as an important reference in this project.

A pilot substation is already implemented in USA based on process bus. In [21], this pilot project is reported. The project is to replace parts of the substation with merging units and process bus communication. So the pilot substation has both conventional system and process bus working together. In this project, it was found that the project was easy to execute since the process bus is easy to configure and the architecture was straightforward. The key challenges are data sharing and synchronization. The process bus works with conventional system without any problems. Also, the signal transferred by process bus is very closed to the results given by conventional systems. This project practically proves the 9-2 process bus technology is a better communication solution for substation compared with the conventional systems.

2.1.3 Existing Testing Method

The case reported in [21] is a replacement project for a substation. In that project, the physical MU is used in the testing. Similarly, in [7], the physical MU is utilized in its experimental set-up which is shown in Figure 2.1 below.

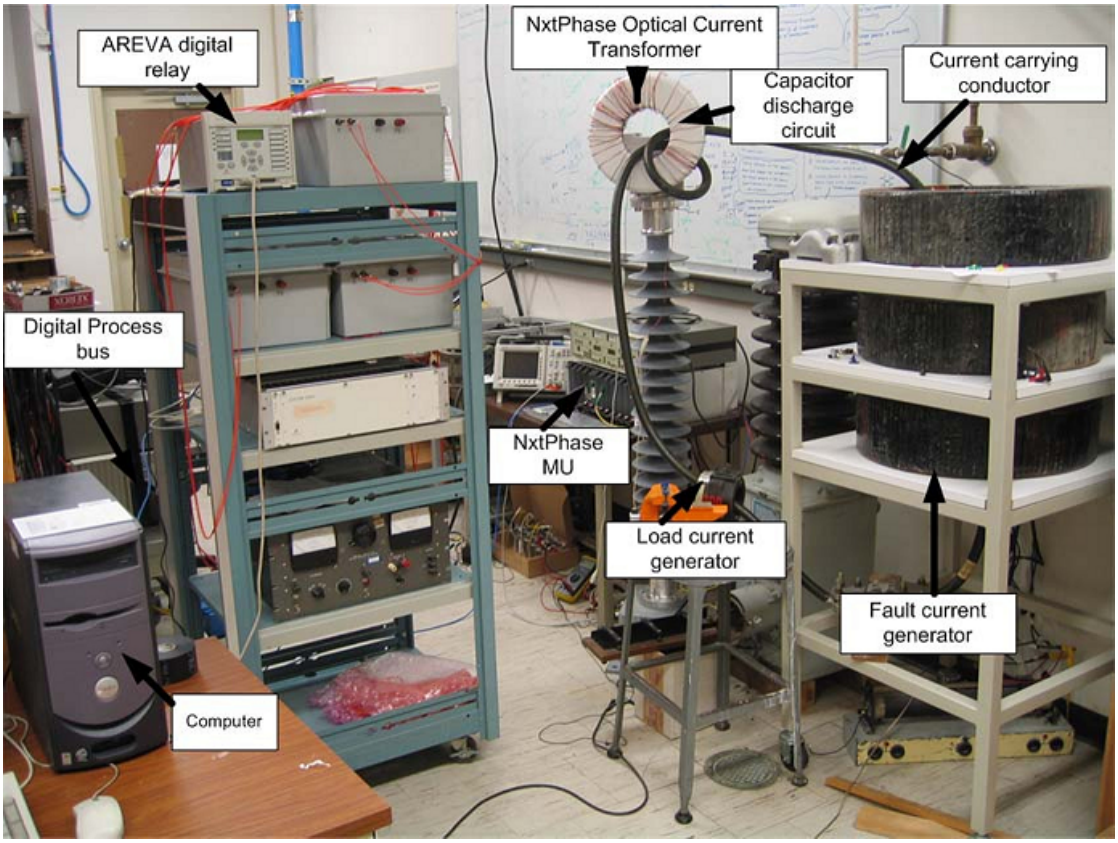


Figure 2.1: Experimental set-up with physical MU in [7]

With professional tools and physical MU shown in Figure 2.1, a detailed IED protection function evaluation can be given in [7]. But using physical MU is not feasible when large amount MUs are needed in a test. It will be too expensive and inflexible. Simulation and software models of MU are preferred in large scale test

in order to make the test executable. Furthermore, for the educational purpose such as concept demonstration, the detailed evaluation report is not needed and the convenience of the testing tool is preferred.

Another testing method is to use simulator. In some simulation tool such as Opal-RT and Real Time Digital Simulator (RTDS), they provide the build-in MU model. The users can connect the MU models to their power system and the SV traffic will be generated from the Ethernet port of the simulator. However, those models are commercial products. It is expensive to have a real time simulator. In addition, the model send out SV traffic is not fully opened. There are only limited parameters can be manipulated by users. This project is also motivated by this point.

2.2 Theory

In this section, the theories related to this project are presented. The process bus interface is programmed by C language in Linux environment and works with Simulink model from Project A. The related theories include C language, some aspects of Linux operating system and Matlab environment. The basic knowledge of Open Systems Interconnection (OSI) model is also presented in this chapter since the process bus is part of the OSI model.

2.2.1 C Language

C language is a commonly used programming language. There are sufficient examples and tutorials resources on the Internet so it is easy to learn and performing troubleshooting. Also, the C language can easily access the low-level operation system functionalities. For instance, in Java programming environment, it runs on a virtual machine which may cause indeterminate run-time behaviour. This is why C language programming is selected in this project.

2.2.2 Linux Operating System

Linux is an open source operating system. It is a Personal Computer (PC) version of Unix system. Unix was designed to fulfil the vary demand of researchers. Linux inherits many advantages of Unix like the speed, efficiency, scalability, and flexibility. [22]

Similar as the reason why C language is selected, the Linux operating system is widely used by researchers especially programmers, so there are huge amount of well documented tutorials and examples on developing C program in Linux environment. Like Linux most of the literatures are open source an easy to access as well.

2.2.3 OSI Layer Communication Model

The OSI model is a commonly applied model in the communication industry. As an international standard for substation communication, the IEC 61850 standard is no exception.

The OSI model aims to provide a common network system model so that the products from all the vendors and communication software developers have interoperability. The OSI model uses a vertical layer structure. In the OSI model, there are seven layers and each layer is responsible for a set of function. The higher layer can utilize the service provided by lower layer. Table 2.1 below lists all the seven layers and their functions. [8]

Layers	Function
1. Physical	Transmission of an unstructured bit stream over the physical medium
2. Data Link	Reliable transmission of frames over a single network connection
3. Network	End-to-end communication across one or more sub-networks
4. Transport	Reliable and transparent transfer of data between end points
5. Session	Control structure and management of sessions between applications
6. Presentation	Data representation (encoding) during transfer
7. Application	Information processing and provision of services to end users

Table 2.1: The seven layers of the OSI model [8]

The lower layers include **Physical**, **Data Link**, and **Network**. The rest of layers are higher layers. The responsibility of lower layers is to transfer the data between end systems while the higher layers is to provides services to users of end system. The information flow between OSI layers is presented in Figure 2.2 below. [8]

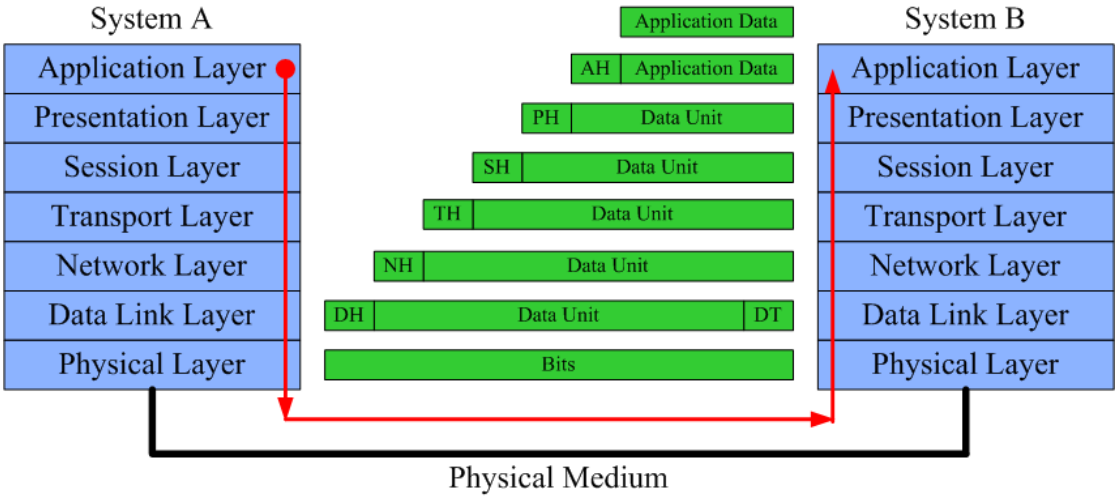


Figure 2.2: Information flow and encapsulation[8]

As shown in Figure 2.2, the information is encapsulated from higher layers to lower layers and each layer adds a header to the data. The header indicates the protocol information in that layer. The receiver side unpack the data from bottom to top.

2.2.4 The SV Service (Data Link Layer)

The process bus SV service is based on data link layer [3]. The data should be encapsulated from top layer down to data link layer. All the headers to be added in each layer are defined in [3]. The programming for the encapsulation is documented in Chapter 4.

Chapter 3

Methodology

In Section 1.6, the goals of this project is defined. To reach those goals, several key problems need to be solved. In this chapter, the key problems are defined. In addition, the method used to solve those problems is introduced.

3.1 Problems Definition

The functional parts in the soft MU model are already shown in Figure 1.5. The process bus interface in that figure is the goal of this project. By observing the Figure 1.5, three problems can be identified.

- The process bus interface programming - Data link layer programming
- Working together with Project A
- Evaluation method of the results

Obviously, the core function of the process bus interface is to generate process bus traffic so the priority should be given to the data link layer programming. All the measurements which need to be sent to process bus are from Project A so the interface with Project A is also needed to be solved. By solving the two problems mentioned above, the project is almost done. The final step is to validate the "soft" MU testing environment before it is finished. Thus, the evaluation work for both Project A and B is the last problem to be solved. The following subsections are going to define all the three problems.

3.1.1 Data Link Layer Programming

As mentioned in Section 2.2.4, the process bus communication belongs to the data link layer. To generate the SV traffic, the data link layer programming should be studied. According to the definition of OSI layer communication model, the data link layer is Layer 2 which is just one layer above the physical layer and deals with raw data through the Ethernet. To program the data link layer traffic, it is more difficult than the high level such as transport layer. For instance, the User Datagram Protocol (UDP) protocol is a transport layer protocol. In C language, there are commands for the UDP protocol. The command can encapsulate the protocol structure automatically. The programmer can just need to assign the data to be sent to the argument of the command and the UDP packet will be generated automatically by the command. However, the SV protocol needs to be programmed from the raw packet structure. As shown in Figure 2.2, the program should encapsulate down to link layer and it needs high level permission to have access to send out raw packet.

What is more, normally, programming is not considered as the professional field of an electric power engineer so the programming technique is not the key point to consider in this project. More attention should be paid on getting familiar with the 9-2 standard and how to interpret the standard during programming.

3.1.2 Working Together with Project A

The output of Project A is the output of the ADC. It is the digitalized measured values from the Simulink power system model. This project will encapsulate the digitalized data from Project A into SV packet and feed them into the process bus

network. How to interface with or embed C code application into Simulink model needs to be figured out. Also, the way of passing data from Simulink model to the process bus interface should be found. The data format from the Simulink model could be different from the data format required by SV 9-2 standard. Data type transformation could be necessary if they do not fit each other.

3.1.3 Evaluation Method of the Results

The outcome of Project A and B should be evaluated in a way which can be executed easily and repeatable. The evaluation results should be able to prove that the MU testing environment could roughly simulate the SV service as the real MU provides. Since the process bus is used in substation automation industry, it is better that the light weight MU testing environment could be evaluated by the substation automation devices. If the automation devices operate normally under the MU environment, it can be concluded that the MU environment can be used as a simulation of the real MU. The evaluation work will be presented in Chapter 5

3.2 Solution - Iterative Method

To solve the problems defined in last section, the iterative research pattern is applied as the research method in this project.

3.2.1 Introduction

The iterative research method is defined in [9] which is shown in Figure 3.1 below.

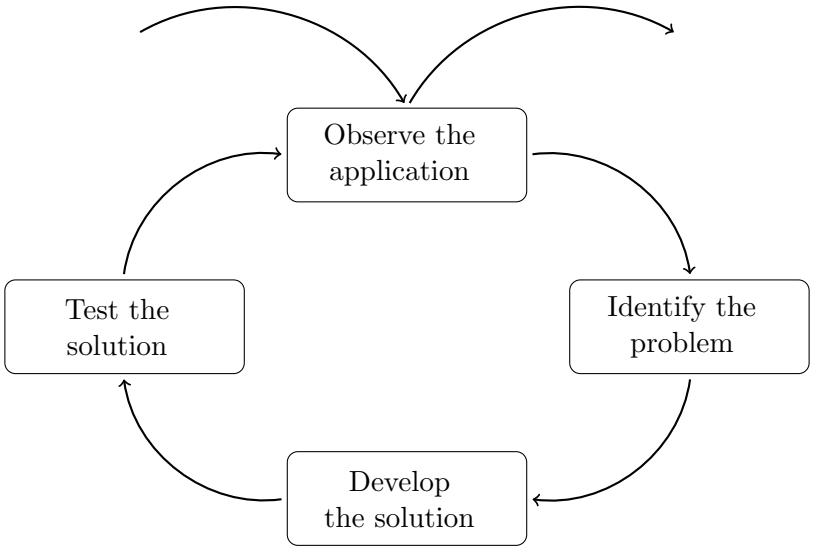


Figure 3.1: Diagram of the iterative research method [9]

It can be seen from Figure 3.1 that the progress starts with the observation which means to observe the requirements of the task. There might be already some results from previous work so the first step could be the observation of existed application. The second step is to identify the problem based on the observation. In the first iteration, the second step may conclude several possible topics. As it is moved into later iterations, this step will focus on the tests which is executed in previous loop. The next step is to solve the problems addressed in the second step. The final step is testing the new work from the development step. Based on the testing results, it can be decided if further iteration is needed to refine or improve the application. [9]

3.2.2 Solution

The iterative method is utilized for solving the programming and interface problems. The solution to the evaluation is presented in Chapter 5. Based on the iterative research pattern, the iterative research method for those specific problems can be designed as follow.

Since there is few previous work can be observed on the MU testing environment, the first step, observation, is simplified. The simplified first step is that observe the related topics of the project and define the scope. The scope of the project is already explained in Section 1.6. The related topics of the project include data link layer programming, SV traffic programming and the interface with Matlab. According to this observation, the project can be accomplished by three iterations:

- 1st Iteration - Program for Data Link Layer Traffic
- 2nd Iteration - 9-2 SV Protocol Implementation
- 3rd Iteration - Interface with Matlab

The 1st iteration solves the data link layer programming problem. A simple data link layer traffic generation application will be developed during this loop. The 2nd iteration refines the code from the 1st iteration. The data link layer traffic will be formatted according to the 9-2 standard. Up to the 2nd iteration, the process bus interface programming problem will be solved. The last iteration is to solve the interface with Matlab issue. The Matlab interface will be added into the 9-2 interface application which is the outcome of the 2nd iteration. All the three iterations are interconnected. The later iteration refines or improves the previous iteration. The programming work of each iteration is designed to be executable by an electric power engineer so that every later iteration can begin from a working program instead of a complicate bugging program.

The detailed implementation procedures are introduced in Chapter 4.

Chapter 4

Implementation

In this chapter, the iterative implementation procedures are documented. Since the iterative pattern is being used in this thesis project, the sections in this chapter follow the order of iterations. There are three iterations in total. In the 1st iteration, data link layer programming is studied. The later iterations are either improvement or extension of the 1st iteration. The second iteration is programming SV data frame according to IEC 61850-9-2 standard. In the 3rd iteration, the process bus interface is embedded into Matlab environment so that it can work with Project A.

The complete source code of the interface is not included in this report. For complete source code, please contact the author by email: zblackhao@gmail.com

4.1 1st Iteration - Program for Data Link Layer Traffic

This iteration is done in several steps. First, the Ethernet communication programming is studied in the beginning phase of the 1st iteration since all the communication traffic is sent and received through the Ethernet port in a PC. The simple UDP sending and receiving example in [23] is used as a reference in order to understand how to send out information through the Ethernet port. However, the UDP is a transport layer protocol while the SV service is based on data link layer [3]. The data link programming needs to be understood before the SV service can be programmed. To learn the data link layer programming, an data link layer protocol-Address Resolution Protocol (ARP) is implemented based on the example in [24] after the first step. The outcome of the 1st iteration is an ARP sending application programmed in C language. With the ARP sending application, the 2nd iteration can be easily done by just modifying the code based on the 1st iteration.

4.1.1 UDP Server and Client Sample Code

UDP is a transport layer protocol which enable application programs to send messages to each others. In [23], there are two separate applications which are named as "client" and "server". The "client" sends out UDP message to "server" through the Ethernet port. The complete code can be found on [23]. Since the aim of the 1st iteration is to program for **sending out** data link layer traffic, the concentration is put on the "client" (sending) program. The flow chart of [23] can be drawn as follow in Figure 4.1.

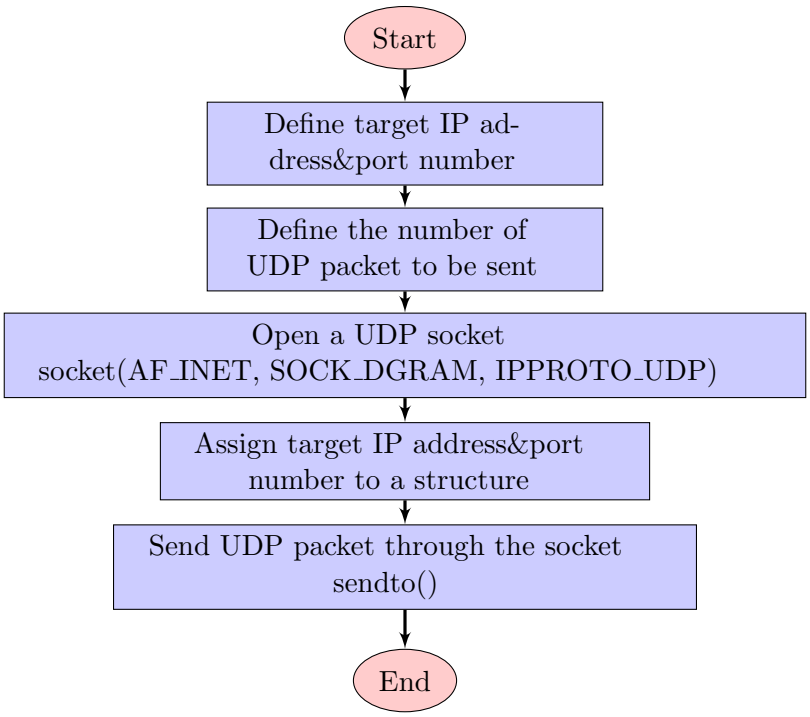


Figure 4.1: Flow Chart of UDP sending

In the "client" (sending) code, commands "socket()" and "sendto()" are the key commands to send out UDP message through Ethernet port.

"socket()" Command [25] [1]

The call of **socket()** will open a socket for the sake of communication and return the file descriptor of the new socket. If an error happened during executing this command, -1 will be returned. The format of calling is as following:

```
1 int socket(int domain, int type, int protocol);
```

The *domain* argument specifies the communication protocol family which will be used. In the UDP "client" example, **AF_INET** is used, which indicates that an Internet Protocol version 4 (IPv4) Internet protocol is used in the socket. The *type* argument defines what kind of communication semantic is to be used. **SOCK_DGRAM** is specified in this example which means datagrams type. The third argument *protocol* indicates the specific protocol which will be used by the socket. In this case, **IPPROTO_UDP**, which represents UDP protocol. Detailed description for the socket() command and the constants for different Internet Protocols (IPs) can be found in [25].[1] [2]

"sendto()" Command [25] [1] [3]

The format of calling sendto() is as following:

```
1 ssize_t sendto(int sockfd, const void *buf, size_t len,
2     int flags, const struct sockaddr *dest_addr,
3     socklen_t addrlen);
```

The **sendto()** call will send the data through a socket. The *sockfd* is file descriptor of the socket through which the data will be sent out. The *buf* pointer points to a buffer where the data to be sent is stored. The *len* indicates the length of the data to be sent. The *flag* specifies the type of message transmission; it is set to zero in the UDP example. The *dest_addr* points to a **sockaddr** structure where the destination address is stored. The *addrlen* specifies the length of the destination **sockaddr**. On success, The return value of **sendto()** call is the number of characters sent. Similar as the **socket()** call, -1 will be returned on error.

With the combination of *socket()* and *sendto()*, the UDP message can be sent out through the Ethernet port. The Ethernet traffic can be captured by the software Wireshark. The UDP packets capture is presented in Figure 4.2.

[1] http://linux.die.net/man/2/socket
[2] http://linux.die.net/include/netinet/in.h
[3] http://linux.die.net/man/3/sendto

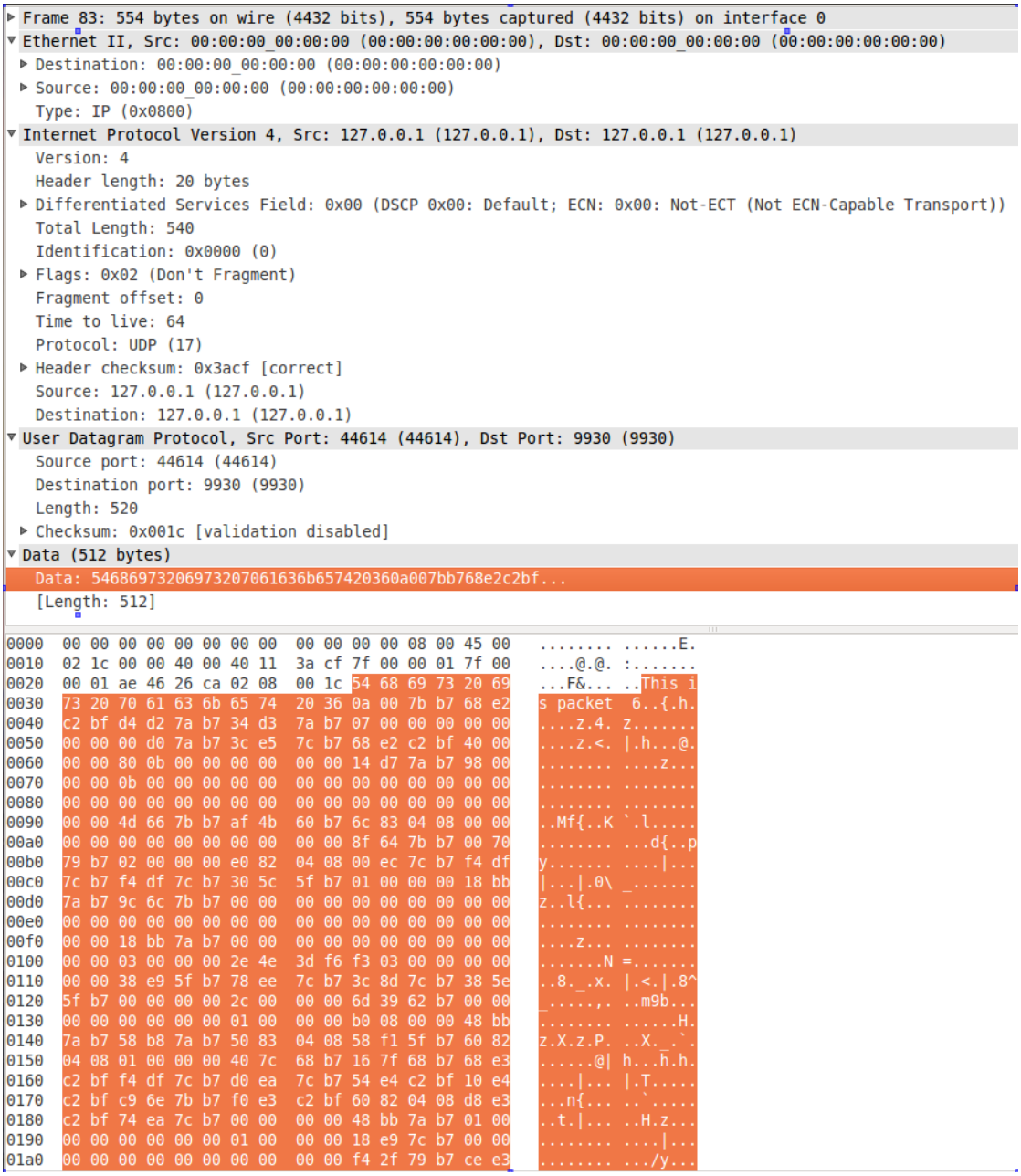


Figure 4.2: The Wireshark capture of UDP sending

An important point to be noticed from Figure 4.2 is that a complete UDP packet includes not only the message to be sent but also other data like Ethernet header and protocol tag. The *sendto()* can encapsulate all the other headers for the UDP protocol since the UDP is a transportation layer protocol. But for data link layer protocol, the *sendto()* command can not do it automatically. As a result, the data link layer encapsulation needs to be studied. An ARP sending example [24] then is studied.

4.1.2 ARP Protocol Implementation

To be able to send out data link layer traffic out, all necessary data should be encapsulated before the calling of packet sending commands **socket()** and **sendto()**. An ARP spoofing sample code [24] is studies in order to understand the way of encapsulating data link layer packet. Based on the sample code in [24], the ARP sending program is finally implemented in this iteration. The flow chart of the ARP sending program is shown in Figure 4.3 below.

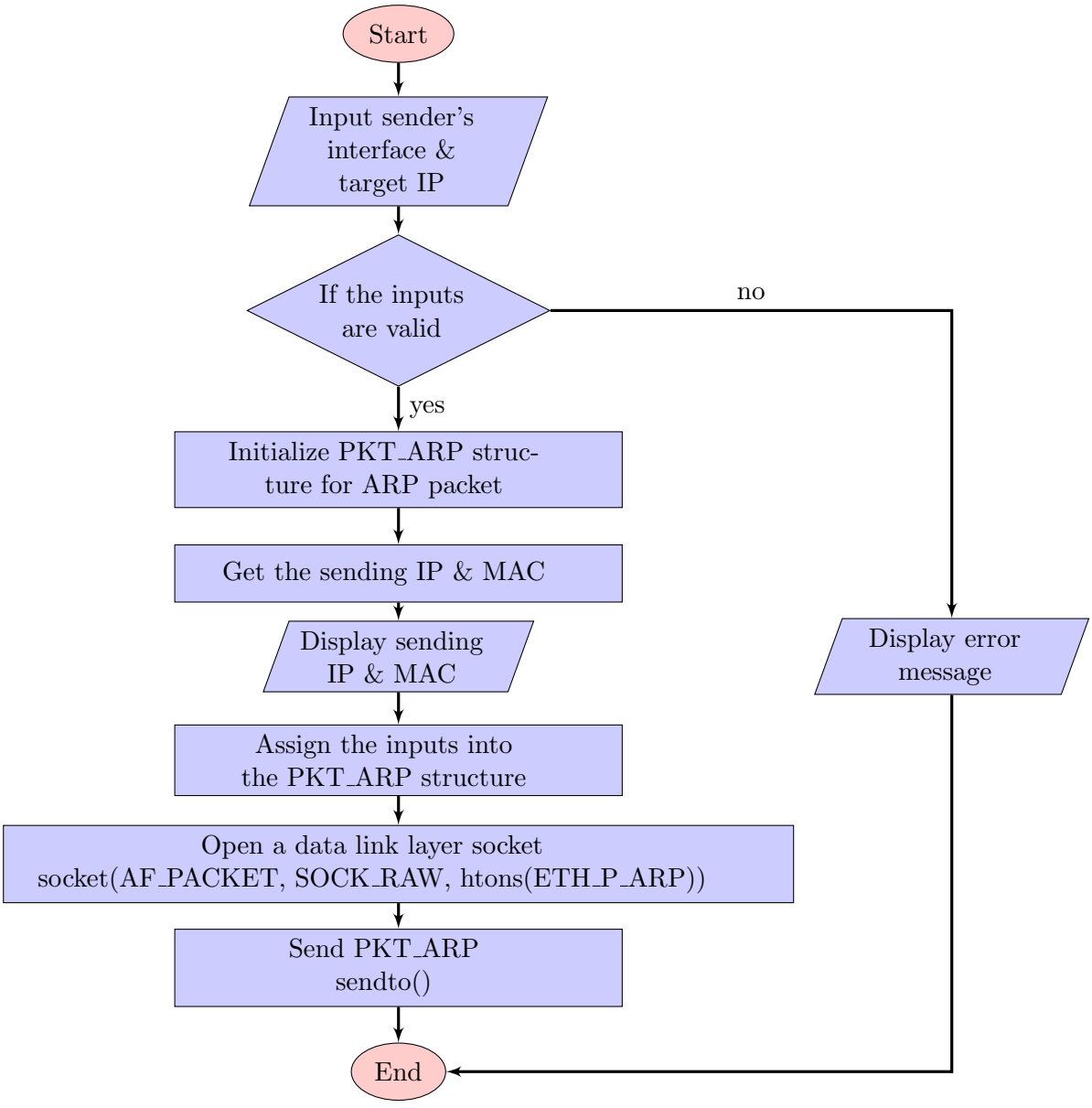


Figure 4.3: Flow Chart of ARP spoofing

Comparing the flow chart of UDP sending and ARP sending, it can be found that the major differences are the structure **PKT_ARP** and the arguments in the **socket()** command.

The PKT_ARP Structure

Unlike the UDP sending program, the **sendto()** command can not encapsulate the Ethernet packet for ARP protocol. As a result, the ARP packet needs to be encapsulated manually according to the ARP protocol which is standardized in [26]. In [26], the ARP packet format is defined as follow [26]:

Ethernet transmission layer (not necessarily accessible to the user):
48.bit: Ethernet address of destination
48.bit: Ethernet address of sender
16.bit: Protocol type = ether_type$ADDRESS_RESOLUTION
Ethernet packet data:
16.bit: (ar$hrd) Hardware address space (e.g., Ethernet, Packet Radio Net.)
16.bit: (ar$pro) Protocol address space. For Ethernet hardware, this is from the set of type fields ether_typ$<protocol>.
8.bit: (ar$hln) byte length of each hardware address
8.bit: (ar$pln) byte length of each protocol address
16.bit: (ar$op) opcode (ares_op$REQUEST — ares_op$REPLY)
nbytes: (ar$sha) Hardware address of sender of this packet, n from the ar$hln field.
mbytes: (ar$spa) Protocol address of sender of this packet, m from the ar$pln field.
nbytes: (ar$tha) Hardware address of target of this packet (if known).
mbytes: (ar$tpa) Protocol address of target.

The structure **PKT_ARP** is defined exactly the same as it is defined in the standard:

```
1  struct PKT_ARP {
2    //Ethernet Header
3    uint8_t dest_mac[6];/*Ethernet address of destination*/
4    uint8_t src_mac[6]; /*Ethernet address of sender*/
5    uint16_t ether_type;/*Protocol type = ether_type$ADDRESS_RESOLUTION
         Ethernet packet data:*/
6    //ARP Structure
7    uint16_t ar_hrd;  /* 16.bit: (ar$hrd) Hardware address space (e.g.,
       Ethernet,Packet Radio Net.)*/
8    uint16_t ar_pro;  /* 16.bit: (ar$pro) Protocol address space. For
       Ethernet hardware, this is from the set of type fields
       ether_typ$<protocol>.*/
9    uint8_t ar_hln;   /* 8.bit: (ar$hln) byte length of each hardware
       address */
10   uint8_t ar_pln;   /* 8.bit: (ar$pln) byte length of each protocol
       address */
11   uint16_t ar_op;   /* 16.bit: (ar$op)  opcode (ares_op$REQUEST |
       ares_op$REPLY)*/
```

```
12   uint8_t ar_sha[6];  /* nbytes: (ar$sha) Hardware address of sender
        of this packet, n from the ar$hln field.*/
13   uint32_t ar_spa;  /* mbytes: (ar$spa) Protocol address of sender of
         this packet, m from the ar$pln field.*/
14   uint8_t ar_tha[6];  /* nbytes: (ar$tha) Hardware address of target
        of this packet (if known).*/
15   uint32_t ar_tpa;  /* mbytes: (ar$tpa) Protocol address of target.*/
16 } _attribute_((_packed_));/*this specifies that one element in this
      structure is stored immediately after the previous one in memory*/
```

To send out one ARP packet out through the socket, the pointer of PKT_ARP will be assigned as the **buf* argument of the **sendto()** command. In addition, the arguments of the *socket()* command needs to be changed to enable the *sendto()* command to send out the **PKT_ARP** structure. The encapsulated data will then be sent out through the Ethernet port and the packet can be recognized as ARP packet.

The AF_PACKET, SOCK_RAW and ETH_P_ARP Constants for socket() Command [25],[4] [27]

To send out the self defined PKT_ARP structure by the socket, the constants used to open the UDP sending socket do not work. Instead of using AF_INET, SOCK_DGRAM and IPPROTO_UDP as the arguments, AF_PACKET, SOCK_RAW and ETH_P_ARP constants are assigned when the **socket()** command is called. AF_PACKET specifies that the socket should provide low level packet interface; SOCK_RAW indicates that the socket will support raw network protocol access; ETH_P_ARP tells the socket that the protocol to be used is ARP. Now, this socket is be able to send out the **PKT_ARP** structure by calling **sendto()** command.

Give Root User Permission to the ARP Sending Code

Up to now, the ARP packet is created and the socket is configured for sending ARP packet as well. However, as mentioned in Section 2.2.2, Linux requires the root permission for using raw packet mode of the network interface. As a result, the root user permission should be given to the the ARP sending program. There are two ways to achieve that:

- Run the program with *"sudo"* command
- Make the program owned by root user group by running the following lines [28]:
  sudo chown root:root <filename>
  sudo chmod u+s <filename>
  where <filename> is the name of the program

The first iteration now is finished. The Wireshark capture of the ARP sending program is shown in Figure 4.4.

[4] `http://linux.die.net/man/2/socket`

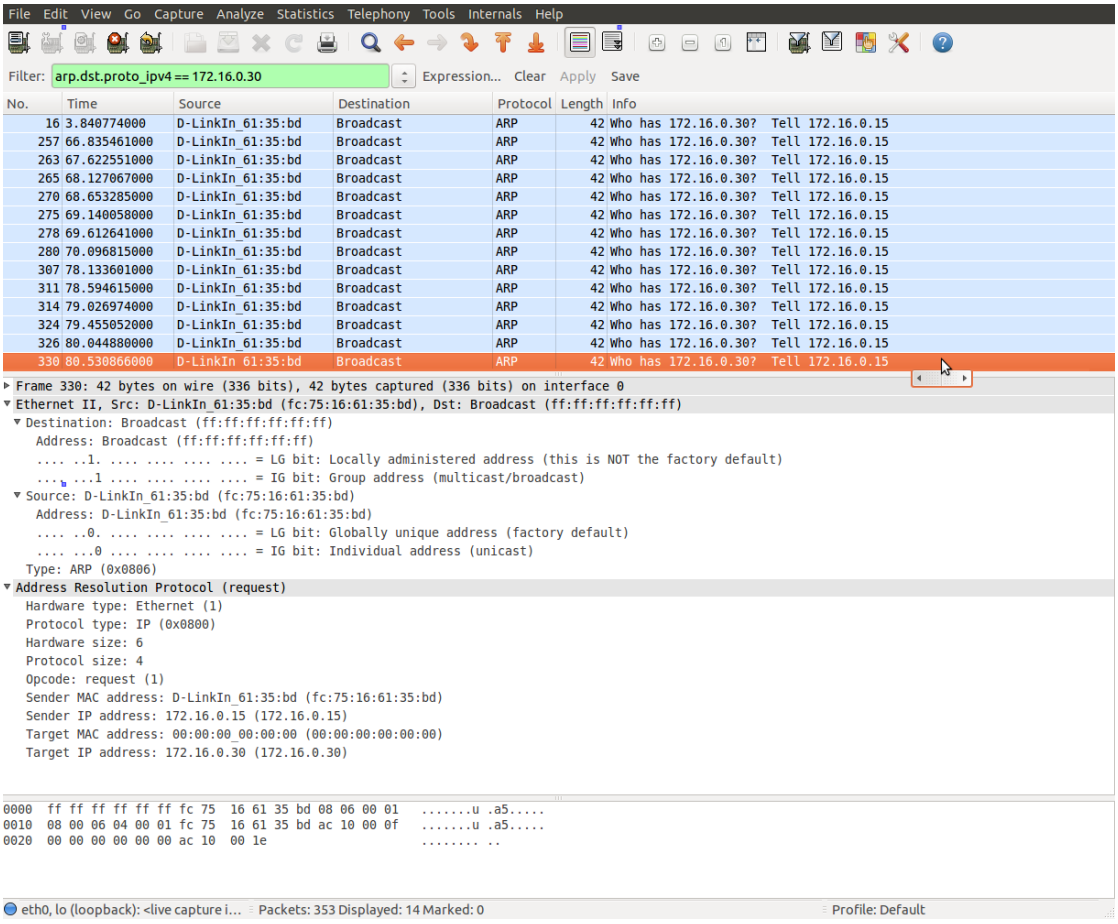


Figure 4.4: The Wireshark capture of ARP sending

4.2 2nd Iteration - 9-2 SV Protocol Implementation

The 2nd iteration is an extension of the the ARP sending program. Since SV is also a data link layer protocol, the arguments for **socket()** and **sendto()** commands can be kept the same as the ARP sending program. The part needs to be changed is the structure (PKT_ARP). It should be modified to the SV packet format according to the IEC 61850-9-2 standard. In this project, instead of the IEC 61850-9-2 standard [3], the *"Implementation Guideline for Digital Interface to Instrument Transformers Using IEC 61850-9-2"* [10] is used as the reference for defining the structure of SV packet. Besides the SV packet format task, the frequency of sending SV packet should be considered as well. The standard rate for SV samples is documented both in [10] and [3], which is 80 samples per second. The algorithm which controls the frequency of sending SV packet will be added into the SV sending program in this iteration as well.

4.2.1 The SV Packet Format

The [10] is a practical guidance which aims to help the industry introduce the IEC 61850-9-2 standard to the market in a fast manner. It is prepared by the persons

from different vendors (including ABB, Siemens, Areva). Since the IED used in this project is ABB RET 670, this guideline document will definitely save us a lot effort to implement the SV sending program.

According to [10], the SV packet format is defined as follow in Figure 4.5, Figure 4.6 and Figure 4.7. This definition is a simplified version of the one defined in [3].

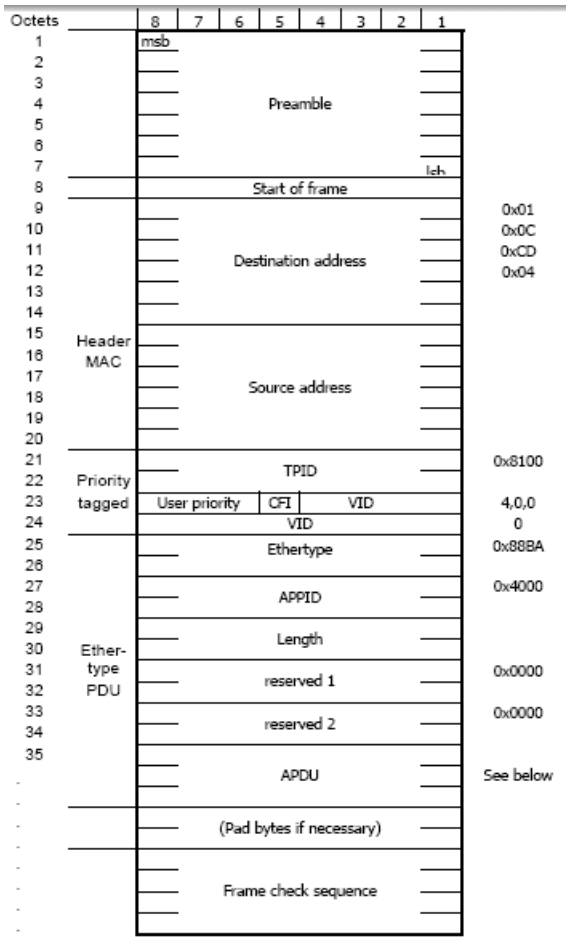


Figure 4.5: Definition of SV Ethernet frame[10]

As shown in Figure 4.5, the name and values of each fields are well defined. One thing to be noticed, in the SV Ethernet frame, is that the field Application Protocol Data Unit (APDU) is separately defined in detail. Figure 4.6 below shows the data structure of an APDU.

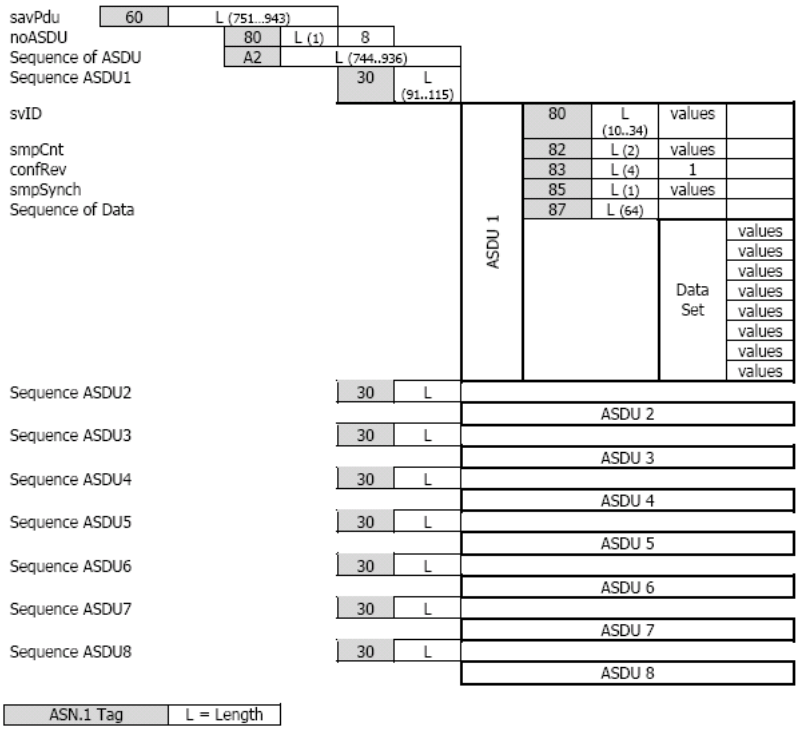


Figure 4.6: Definition of APDU in SV Ethernet frame [10]

From Figure 4.6 above it can be observed that different variables are encoded following the same pattern: tag+length+values. This pattern is according to the Abstract Syntax Notation One (ASN.1) Basic Encoding Rules. Normally, the ASN.1 tag and length take 1 byte (8 bits) each. The values can take multiple bytes depend on how many bytes are defined for the values. Also, all the fields are expressed in hexadecimal number. For instance, in Figure 4.6, the variable *"svID"* is started with the number 0x80, which means when the ASN.1 tag for *"svID"* is 80 **in hexadecimal** and it occupies 1 byte in the memory. Following the tag 0x80, there is a letter *"L"* which means length. There is a bracket under the *"L"* with numbers *"10..34"* in it. This indicates that the length of the *"svID"* could be in the range of 10 to 34 bytes. The *"length"* could be set to any numbers between 10 and 34. One important thing to pay attention on is that the range (10..34) for *"length"* is in decimal so it should be converted into hexadecimal during programming. The actual values of the *"svID"* is encoded after the *"length"* which should be also expressed in hexadecimal.

It can be seen in Figure 4.6, one APDU can include more than one Application Service Data Units (ASDUs). ASDU is the data unit in which the measured values and other important information are contained. The measured values are saved in the field *"Data Set"*. The standard also defines the detailed packet format for the *"Data Set"*. Figure 4.7 shows how [10] defines it.

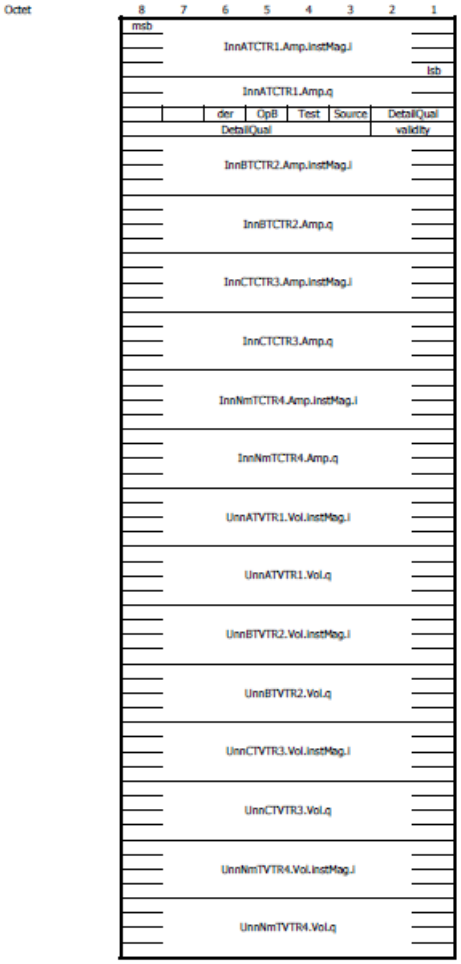


Figure 4.7: Definition of the Data Set PhsMeas1 [10]

The meaning of each variable can be easily recognized by observing the names. For example, *"InnATCTR1.Amp.instMag.i"* means Current$_{nn}$ of Phase A (InnA) from the logic node TCTR1, the unit is in Ampere (Amp) and it is an instant magnitude of the current (instMag.i). *"InnATCTR1.Amp.q"* means this variable shows the quality (.q) of the measured value from TCTR1. As shown in Figure 4.7, each variable takes 4 bytes (32 bits) and there are eight instant values and eight quality identifiers for each instant value. Normally, the eight values are assigned to represent three-phase current with neutral and three-phase voltage with neutral.

As shown in Figure 4.5,4.6 and 4.7, the SV Ethernet frame structure can be encoded into three layers: the 1st layer is the SV Ethernet frame layer which defines the Ethernet header and other fields which should be included in a SV packet; the APDU is the 2nd layer which is a part of the 1st layer; the 3rd layer is the PhsMeas1 Data Set. Figure 4.8 below shows the three-layer relation.

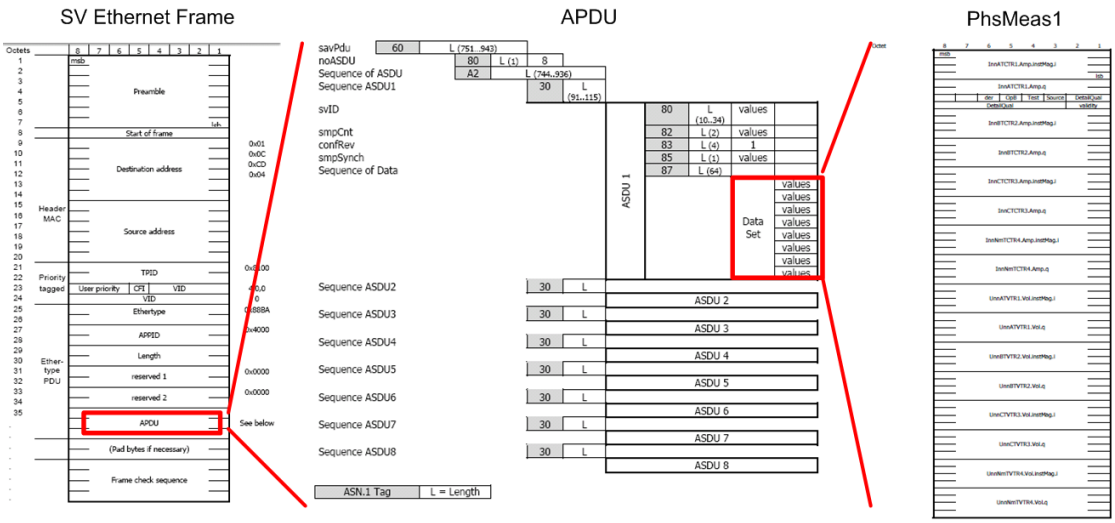


Figure 4.8: Three-layer Structure of SV Packet

4.2.2 Define the Structure for SV Packet

With the three-layer definition from [10], accordingly, the structure in the SV sending program can be defined in three layers accordingly. However, eventually, the structure in the code is defined into four layers:

- 1st layer - SV Ethernet frame
- 2nd layer - APDU
- 3rd layer - ASDU
- 4th layer - PhsMeas1

In Figure 4.6, it can be seen that one APDU could consist of several ASDUs. By defining the ASDU as a separate layer, the code could be easily expanded if an APDU contains more than one ASDUs. Although in this case there is only one ASDU included in the APDU, the ASDU structure is still separately defined in order to have possibility to include more ASDUs in the continious study.

The SV Ethernet frame layer structure is defined as follow:

```
1  struct FRM_8802{
2    //Header MAC
3    int8_t dest_mac[6];
4    int8_t src_mac[6];
5    //Priority tagged
6    //int16_t TPID;//0x8100 for 802.1Q Virtual LAN
7    //int16_t TCI;//0x4000 see 9-2 Implementation Guideline
8    //Ethertype PDU
9    int16_t Ethertype;//0x88BA for Sampled Value protocol
10   int16_t APPID;//0x4000 application identifier
11   int16_t lgth; //count start from APPID
12   int16_t rsrv1;
13   int16_t rsrv2;
14   //APDU application layer protocol data unit
```

```
15   struct APDU apdu;
16   //pad if necessary
17   //Frame check sequence if necessary
18 }__attribute__((__packed__));
```

The structure is named as FRM_8802 because the SV frame is a type of ISO/IEC 8802-3 data link layer frame. The destination Media Access Control (MAC) address should be in the range of "01-0C-CD-04-xx-xx". [10] Compared this FRM_8802 structure with Figure 4.5, the difference is the *"priority tagged"* part (Line 5, 6 and 7) where it is commented.

In the end of the **FRM_8802** structure, a new structure **APDU** is defined. That is the 2nd layer structure which is defined as follow:

```
1  struct APDU {
2    int8_t savPDU_tag;
3    int8_t savPDU_length;//sizeof APDU
4    int8_t noASDU_tag;
5    int8_t noASDU_length;//sizeof noASDU
6    int8_t noASDU;
7    int8_t SequenceofASDU_tag;
8    int8_t SequenceofASDU_length;//sizeof all ASDU
9    struct ASDU_LE ASDU1;
10 }__attribute__((__packed__));
```

As mentioned at the beginning of this section, the ASDU is taken out as a separated structure. The structure named **ASDU_LE** defines the ASDU layer. Since an APDU could contain more than one ASDUs, the ASDU_LE structure is named as ASDU1 in the **APDU** structure. In this project, the case that the APDU contains only one ASDU is studied.

```
1  struct ASDU_LE {
2    int8_t SequenceASDU_tag;//0x30
3    int8_t SequenceASDU_length;//0x01
4    int8_t svID_tag;//0x80
5    int8_t svID_length;//0x0A
6    int8_t svID[11];//naming rules is in IEC 61869-9
7    int8_t smpCnt_tag;//0x82
8    int8_t smpCnt_length;//0x02
9    int16_t smpCnt;//Counter specification see IEC 60044-8
10   int8_t confRev_tag;//0x83
11   int8_t confRev_length;//0x04
12   int32_t confRev;//Configuration revision number
13   int8_t smpSynch_tag;//0x85
14   int8_t smpSynch_length;//0x01
15   int8_t smpSynch;//Synchronization identifier
16   int8_t SequenceofData_tag;//0x87
17   int8_t SequenceofData_length;//0x40
18   struct PhsMeas1 DataSet;
19 }__attribute__((__packed__));
```

The *"svID"* is decided to be "KTH_ICS_SV1". Each character takes one bytes to represent so the *"svID"* will take 11 bytes in this project. American Standard Code for Information Interchange (ASCII) is used to encode the letters for the *"svID"*.

The 4th layer is the *"PhsMeas1"* data set.

```
struct PhsMeas1 {
  //Current Phase A, B, C and neutral instant magnitude in Ampere
  int32_t CurrentATCTR1_Amp_instMag_i;
  int32_t CurrentATCTR1_Amp_q;//quality
  int32_t CurrentBTCTR2_Amp_instMag_i;
  int32_t CurrentBTCTR2_Amp_q;
  int32_t CurrentCTCTR3_Amp_instMag_i;
  int32_t CurrentCTCTR3_Amp_q;
  int32_t CurrentNTCTR4_Amp_instMag_i;
  int32_t CurrentNTCTR4_Amp_q;
  //Voltage Phase A, B ,C and neutral instant magnitude in Volts
  int32_t VoltageATVTR1_Vol_instMag_i;
  int32_t VoltageATVTR1_Vol_q;//quality
  int32_t VoltageBTVTR2_Vol_instMag_i;
  int32_t VoltageBTVTR2_Vol_q;
  int32_t VoltageCTVTR3_Vol_instMag_i;
  int32_t VoltageCTVTR3_Vol_q;
  int32_t VoltageNTVTR4_Vol_instMag_i;
  int32_t VoltageNTVTR4_Vol_q;
}_attribute_((_packed_));
```

This structure will store all the measurement from Project A. According to [10] all the current and voltage values are scaled. The scaler is 0.001 for current and 0.01 for voltage. [10] For instance, the measured current is 1A at a instant for phase A. The value assigned to *"CurrentATCTR1_Amp_instMag_i"* should be 1/0.001=1000. The value should be converted in hexadecimal and 4 bytes i.e. 0x000003E8 should be assigned to the variable *"CurrentATCTR1_Amp_instMag_i"* so that the IED can understand the measured current is 1A. The meaning of each bit in the quality identifier is illustrated in Table 4.1 below.

Octet	8	7	6	5	4	3	2	1	
MSB	-	-	-	-	-	-	-	-	LSB
	-	-	-	-	-	-	-	-	
	-	-	Derived	Operator Blocked	Test	Source	Inaccurate	Inconsistent	
	Old Data	Failure	Oscillatory	Bad Reference	Out of range	Overflow	Validity		

Table 4.1: The definition of quality identifier for each bit

The quality identifier takes 4 bytes (32 bits) but the first 18 bits are not used. Only the last 14 bits indicate the quality and attributes of the data.

4.2.3 Sending 4000 samples/second Algorithm

With the structure defined in previous section, the SV protocol data can be encapsulated properly by the code. However, the SV traffic can not be recognized by the IED if it is sent out in the same way as the ARP sending program. In [10], the rate of sending SV packet is also specified. There are two types of SV rate: 80 and 256 samples per nominal period. In this project, 80 samples per cycle is considered and implemented. 80 samples/period is for general protection purpose. The nominal frequency of the power system is 50Hz in Europe i.e. 50 periods per second. By

simple calculation, the SV packet sending frequency is 80*50=4000 samples per second (4000Hz). Here, the word "sample" means one SV packet. Obviously, the command sendto() should be used 4000 times per second. The variable *"smpCnt"* shown in Figure 4.6 is to count the samples. It counts from 0 to 3999 for 4000 samples. Each time a SV packet is sent out, the variable *smpCnt* should be increased by 1. It takes one second for *"smpCnt"* to counts from 0 to 3999. In other words, the time interval between each SV packet should be 1s/4000=250μsecond.

To fulfil the SV packet sending rate requirement, command **usleep()** and **gettimeoftheday()** are called. The syntaxes for calling them are explained below:[25][5][6]

```
1 int usleep(useconds_t usec);
```

The **usleep()** function suspends the current thread for (at least) *usec* microseconds [25].[6]

```
1 int gettimeofday(struct timeval *tv, struct timezone *tz);
```

This command can get the time when it is called. The values of current time in the computer clock will be stored in the structure **timeval** and **timezone**. The **timeval** and **timezone** structures are specified in the header file <sys/time.h> as follow [25]: [7]

```
1 struct timeval {
2     time_t      tv_sec;     /* seconds */
3     suseconds_t tv_usec;    /* microseconds */
4 };
5 struct timezone {
6     int tz_minuteswest;     /* minutes west of Greenwich */
7     int tz_dsttime;         /* type of DST correction */
8 };
```

In the SV sending code, only **timeval** structure is interested. The seconds when the *gettimeofday()* command is called will be stored in the variable *tv_sec* and the microsecond is stored in *tv_usec*.

With all the commands introduced above and *sendto()* command, the 4000 samples/second algorithm is designed and programmed. The flow chart of this algorithm is shown in Figure 4.9

[5] http://linux.die.net/man/3/usleep
[6] http://linux.die.net/man/2/gettimeofday

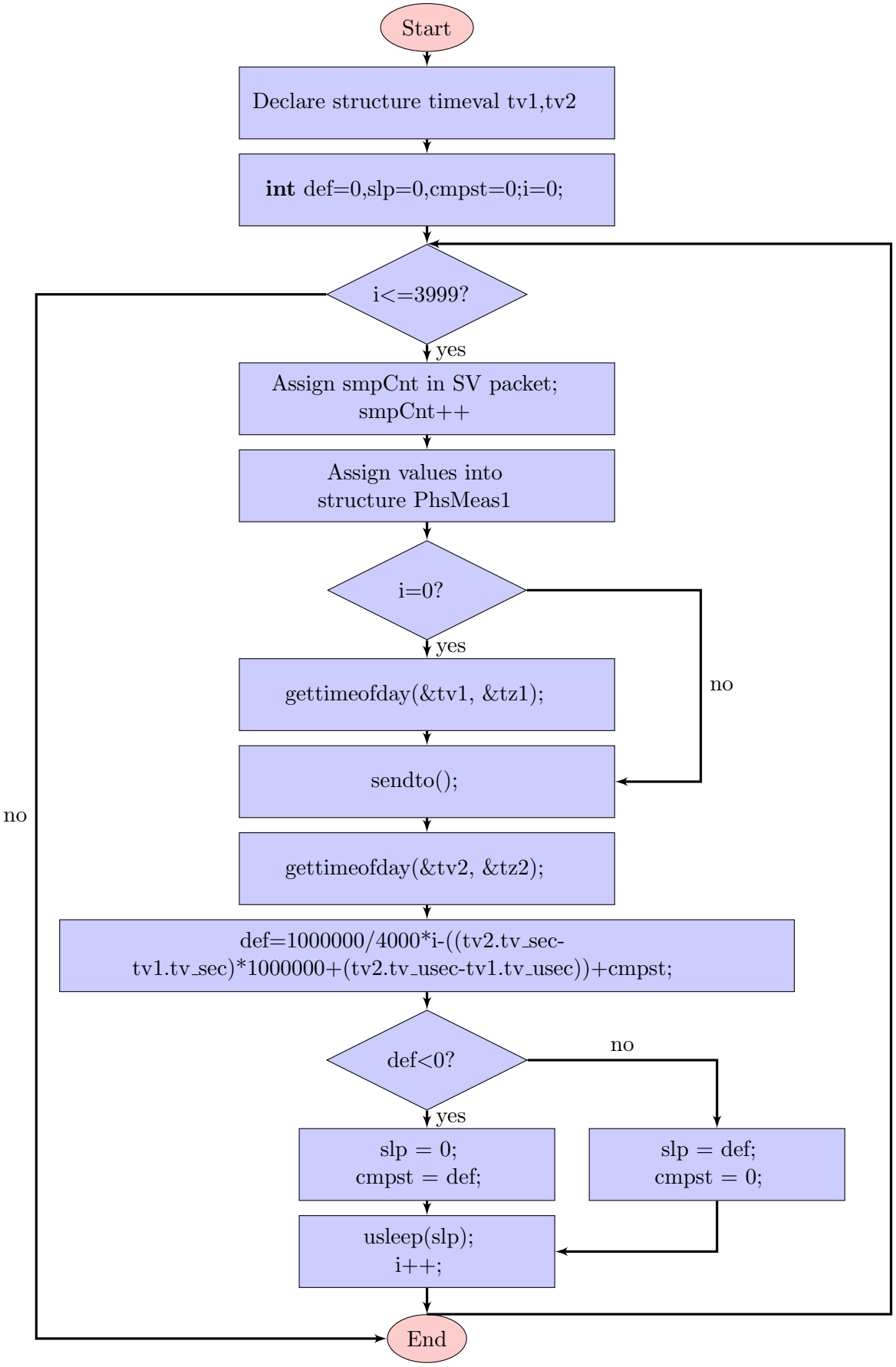


Figure 4.9: Flow chart of 4000 samples/s algorithm

As shown in Figure 4.9, every time a SV packet is sent the *"smpCnt"* variable increases by 1. When the *"smpCnt"* is 0, the *gettimeofday()* is called to get the time when the "0" SV packet is sent. The time is assigned to the structures *tv1* and *tz1*. Since only the time in second and microsecond are concerned, the values from time zone structure *tz1* are not used in this algorithm. After the call of *"sendto()"*, *gettimeofday()* is called again but the time get from this call is assigned to another structures *tv2* and *tz2*. According to the second and microsecond values in *tv1* and *tv2*, a value *"def"* can be calculated and depending on the sign of *"def"*, the argument value *"slp"* for "usleep()" can be decided. The following example explains the algorithm by giving an example.

At an instance, in the *"for"* loop the *"smpCnt"* is 0. Before the call of *"sendto()"*, *"gettimeofday()"* should be called and it is assumed that the time get from computer is 1000000 second and 0 microsecond. In structure *"timeval"*, the variable *"tv_sec"* stores the time in second (1000000) and *"tv_usec"* has the time in microsecond (0). After the "No.0" SV packet is sent, the *"smpCnt"* is increased to 1 and *"gettimeofday()"* is called again. It can be assumed that the *"sendto()"* and other functions take 10 microseconds so this time the call of *"gettimeofday()"* gets 1000000 seconds and 100010 microseconds back. Those values are stored in another structure *"timeval2"*. With values from structure *"timeval1"* and *timeval2*, the *"def"* can be calculated. The equation to calculate the *"def"* is presented in Figure 4.9 and it is rewritten here:

```
1 def=1000000/4000*(i+1)-((tv2.tv_sec-tv1.tv_sec)*1000000+(tv2.tv_usec-
      tv1.tv_usec))+cmpst;
```

There are 4000 SV packets in 1 second so the time interval between each SV is 100000/4000=250μs. The "smpCnt" of the first SV packet is "0". The time difference between "No.0" SV packet and "No.i" SV packet is 1000000/4000*(i-0)μs. For example, if i is equal to 0 which means the first packet in the 4000 samples is just sent out. The program should wait for 1000000/4000*(1-0)=250μs until the second SV packet can be sent out. Since the *"sendto()"* and other functions already take some time, the waiting time should be deducted by a certain time. By calculating the time difference of the structures *tv1* and *tv2*, the value to be deducted can be get. The deduction time calculation is done by the part *"((tv2.tv_sec-tv1.tv_sec)*1000000+(tv2.tv_usec-tv1.tv_usec))"* of the Equation 4.2.3. If *"def"* is positive sign, the value of *"def"* will be passed directly to *"slp"*. In case of the sending command and other commands' executing time is greater than 250μs, the program should not wait any more. In Figure 4.9, it can be seen that when *"def"* is smaller than 0 which means the executing time exceeds 250μs, the *"slp"* will be set to 0 which tells the *"usleep()"* function not "sleep" and the negative *"def"* value will be passed to *"cmpst"*. *"cmpst"* indicates how much time should be compensated in next "sleep" time calculation loop. The waiting time is compensated in this way so that the total time does not exceed 1s. The results of this algorithm will be presented in next section.

4.2.4 Results and Flaws

By replacing the PKT_ARP structure with the FRM_8802 and adding the 4000 samples algorithm into the code from the 1st iteration, the 2nd iteration is done. The flow chart of the code after 2nd iteration is shown in Figure 4.10.

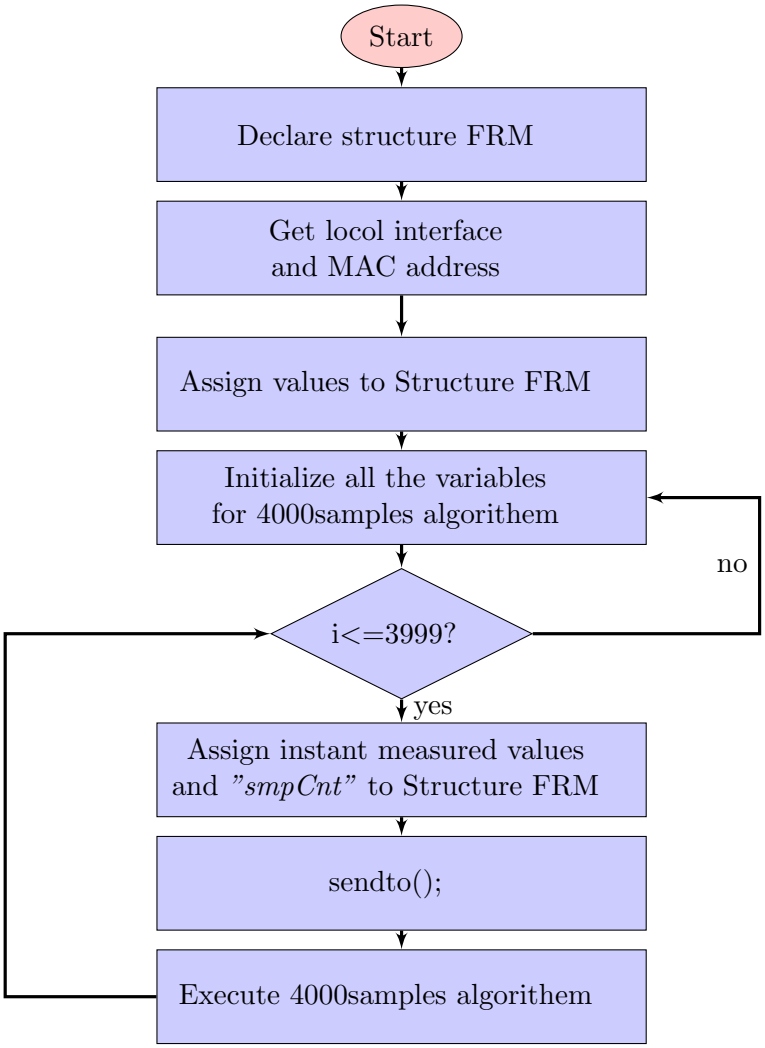


Figure 4.10: Flow chart of SV sending

As shown in Figure 4.10 above, it shows the flow chart of one MU case and the SV sending code will send SV from one MU continuously. If it is required to simulate SV stream from several MUs, the code should be modified. Several SV structures should be defined for each MU and the "sendto()" command will be called for several times to send out different SV packets for different MUs. The evaluation of sending out SV stream for two MUs will be presented in Section 5.5.

The values in *"PhsMeas1"* in the 2nd iteration are not the real instant measurement of a power system. They are "fake" values which are three-phase 50Hz sine wave with a fixed magnitude. Those values are also generated by C code. The Wireshark capture of SV stream is shown in Figure 4.11 below.

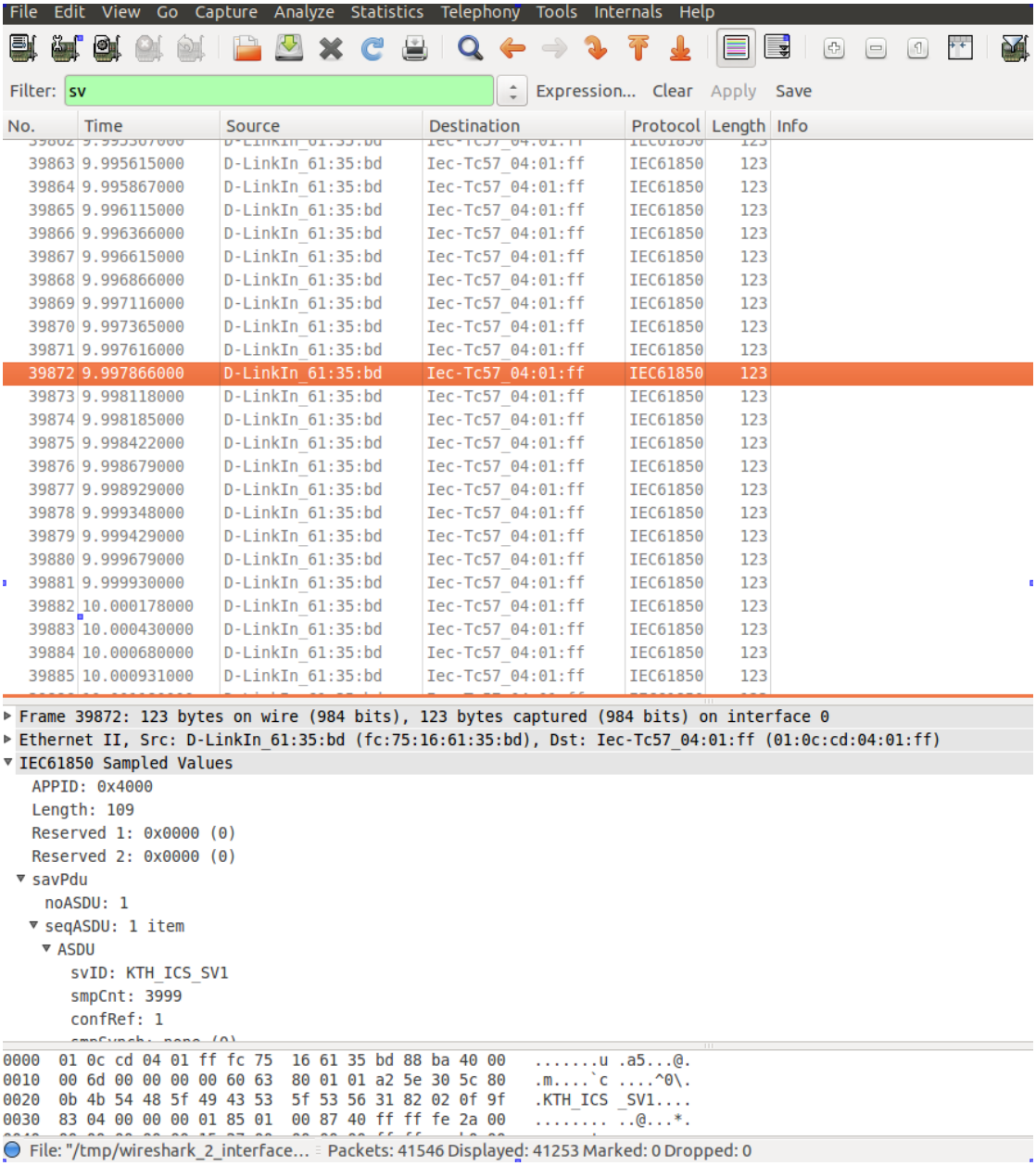


Figure 4.11: The Wireshark capture of SV

As shown in the Wireshark capture above, the time differences between each SV packets are not always 250μs. Also the total time of sending 4000 samples is always within 1 second which leads to that the errors are more and more. As shown in Figure 4.11, after 10 seconds, the time when the packet with "smpCnt=3999" is captured is about 9.997866s. Since the "smpCnt" starts from 0 to 3999 and loops back to 0 it means the errors already accumulate to more than 2 milliseconds. The inaccuracy is because the code is not run in a high priority and real-time thread so that there will be jitters. The accuracy of this algorithm is not comparable with the accuracy of One Pulse Per Second (1PPS) synchronization. But according to the evaluation results in Chapter 5, this algorithm is still accurate enough to make the IED recognize the SV stream and read the data correctly.

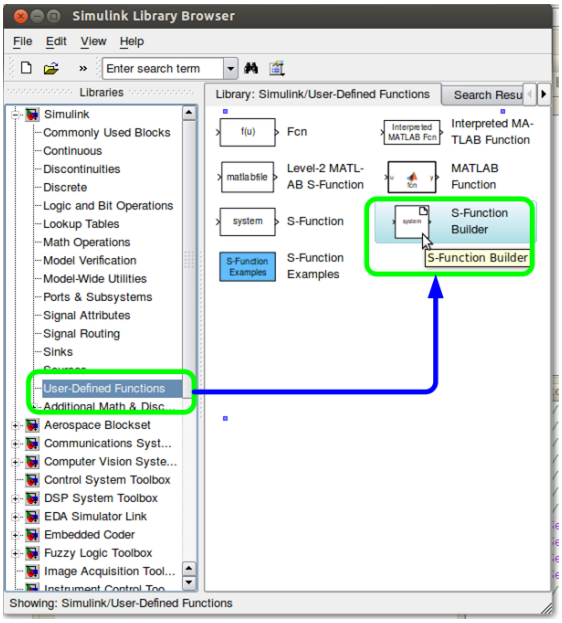
4.3 3rd Iteration - Interface with Matlab Environment

This project is Part B of the parent project. The report of Project A can be found in [5] and it can be found that Project A is developed in Matlab environment. Project A will give out the digitalized measurement values from Simulink model and the SV sending application from the is responsible for sending the measurement as SV traffic. As a consequence, the SV sending code developed in the 2nd iteration should be interfaced with Matlab environment to be able to read data from Simulink simulation results.

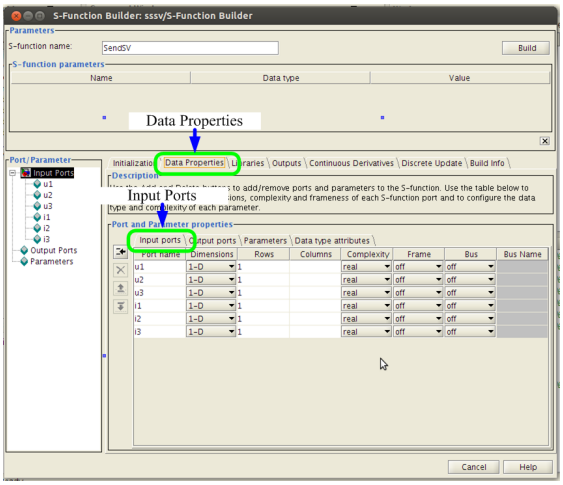
There are several ways to embed C code into Matlab environment and interface with Simulink simulation model. In this project, "S-function" and "mex command" solutions are proposed. Both ways have advantages and disadvantages. The following sections will discuss both of them.

4.3.1 S-function

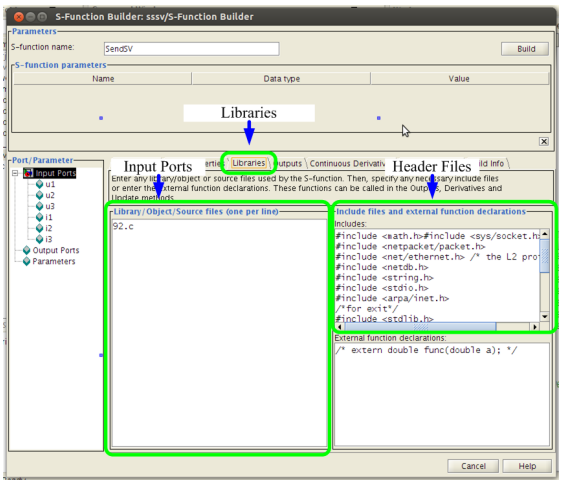
With the help of S-function, the users can run external code in Simulink model as a function block. The "S-function builder" block helps the users build an "S-function" easily.



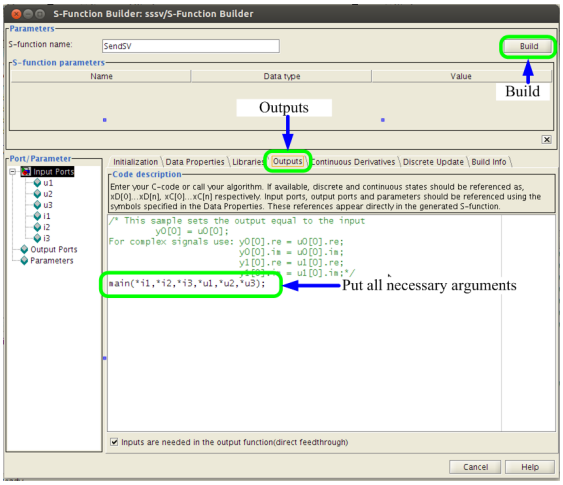
(a) Select the "S-Function Builder" block in the Simulink library



(b) Specify the input of the "S-Function"



(c) Specify the libraries of the "S-Function"



(d) Specify the output of the "S-Function"

Figure 4.12: The steps of building "S-Function"

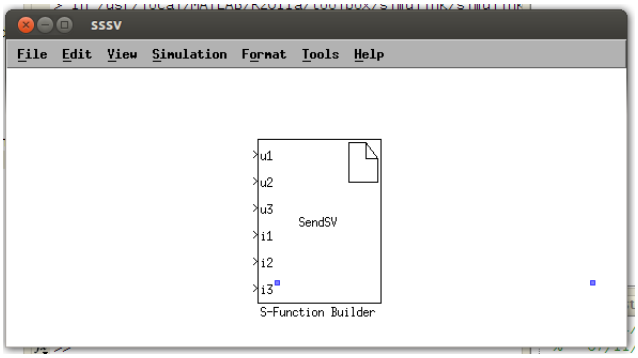


Figure 4.13: The SV sending S-Function block

One big issue rises when the s-function is running in Simulink. The s-function can slow down the whole simulation greatly. In each simulation step, the s-function is called once so the external code i.e. the SV sending program is also run once by s-function. A powerful computer is needed to run the s-function 4000 times per second in real time. Especially when the SV sending s-function is connected to a power system model to get measurement, an even more powerful computer is required since the power system should also be simulated 4000 steps per second. However, this project is aimed to have a **light weight** test environment which can run on a normal PC. Considering this point, the s-function can not be used as the interface solution between external code and Matlab environment.

4.3.2 "mex" Command

By calling the syntax "mex *filename*", the C source code can be compiled into a share library called a binary MEX-file from Matlab. The "mex" command can be run in command window in Matlab. It can build a executable for stand alone Matlab engine and MAT-file applications. [29] The "MAT-file application" feature of "mex" command will be used to solve the problem of reading values from Simulink model which will be reported in next section.

To compile C code in Matlab environment by "mex command", the source code needs to be slightly modified. The *main()* function syntax should be changed to *mexFunction()*. If there is the display command *printf()*, it should be replaced by *mexPrintf* in order to display message in Matlab instead of the terminal screen.

To compile code, type "mex <filename>" in the command window. It is the same way to run the code as it to run in the terminal. The code also runs as fast as it runs in terminal i.e. it can send out 4000 samples per second. However, the "mex" solution can not read values directly from Simulink. The ".mat" file then is proposed to enable the code to read data from Simulink model.

4.3.3 ".mat" File

During the simulation running, the Simulink model can write the values of all its parameters in files with the suffix ".mat". The output ".mat" file is shown in Figure 4.14.

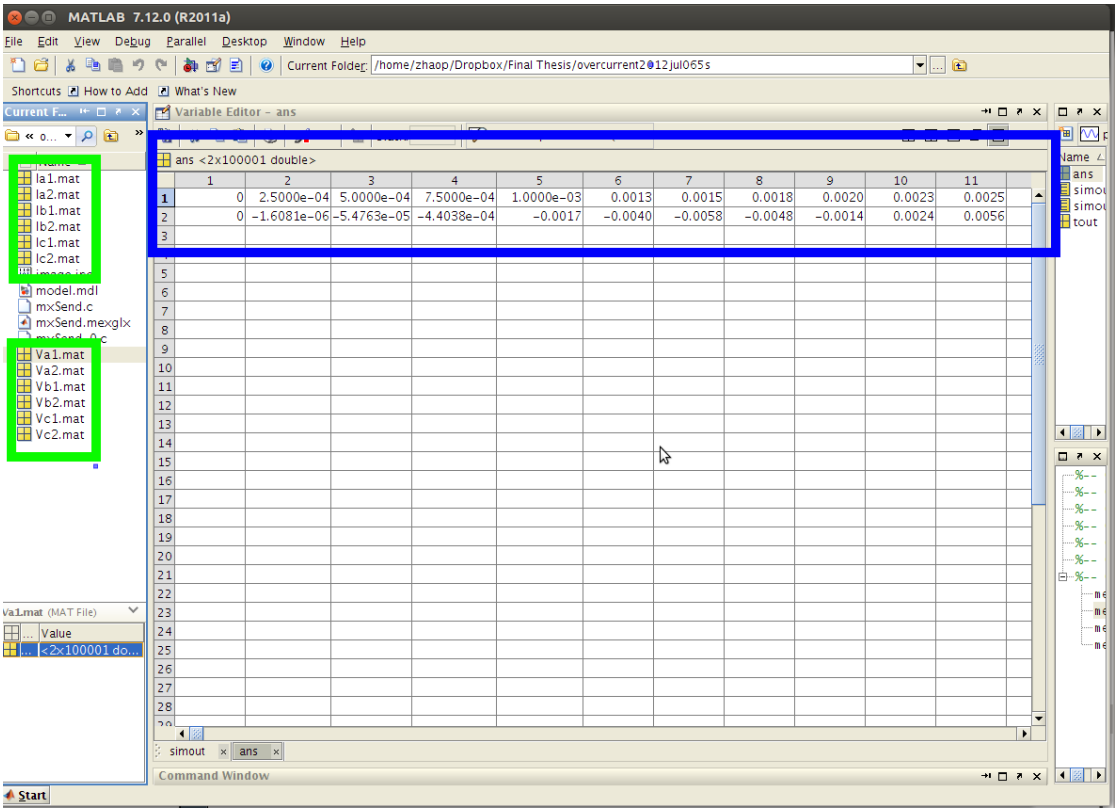


Figure 4.14: The ".mat" files

In Figure 4.14, the ".mat" files are marked by the rectangle with green color. The blue rectangle is the data in one of the ".mat" files. It can be seen that each simulation step gives one column of data which consists of the time stamp and the measurement. The first row is the time stamp from which it can be seen the step of simulation is 250μs. The second row is the measurement from a node. In this figure, the simulation is run for 25 seconds so there are 4000*25+1=100001 columns in one ".mat" file.

In "mex" environment, C code can read the ".mat" file by calling the command *"matOpen()"*. The command should be used in the following format:

```
1 MATFile *matOpen(const char *filename, const char *mode);
```

The argument *"filename"* is the name of the file to open. *"mode"* indicates the file opening mode. In this project, the mode *"r"* is used which means the file is opened for reading only. The call of *"matOpen()"* command will return the file handle if the file is opened successfully. Otherwise, *NULL* will be returned. [29]

With the ".mat" solution, the SV sending code can read data step by step from the Simulink model and encapsulate them into SV packet. The whole working procedure is shown in Figure 4.15 below.

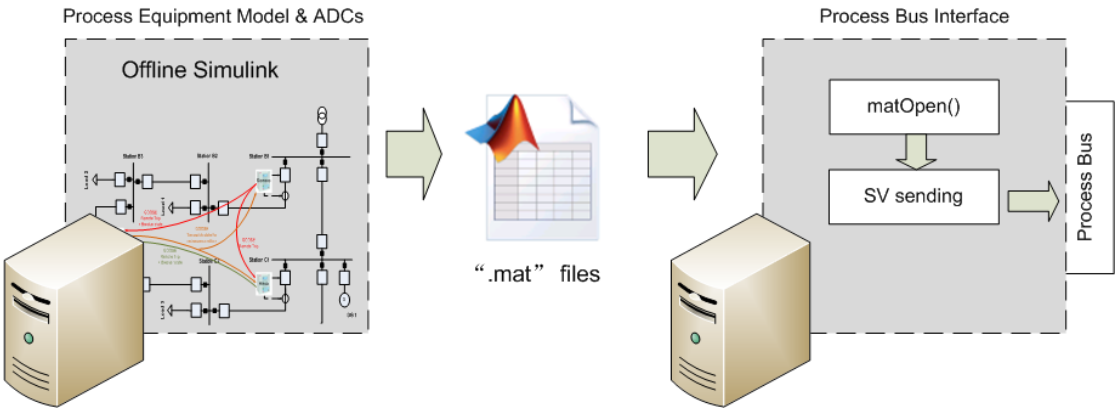


Figure 4.15: The ".mat" off-line simulation solution

By programming for the solution shown in Figure 4.15 above, the the 3rd iteration is finished and the process bus interface is done. According to the problems defined in Section 3.1, the next task is the evaluation work. But before the process bus interface can be evaluated, the process bus network needs to be set up.

4.4 Configure Process Bus Network

In order to prepare for evaluation work, a process bus network should be configured. Since the SV traffic is time critical, it is better to physically separate the process bus network from the station bus so that the network capacity can be guaranteed.

As shown in Figure 1.4 in Chapter 1, the process bus connects the IEDs and the MUs together. In this project, the IED is ABB RET 670 the MU is simulated by the MU testing environment. The SV stream sent by the MU testing environment is transferred to the RET 670 by a Ruggedcom switch. The connection between PC and switch is RJ45 cable. RET 670 is connected to the process bus network by fibre optic cable. The IED RET 670 has a separated fibre optical Ethernet port which is designed especially for 9-2 standard so that the process bus traffic is physically separated in the IED also. The Figure 4.16 below shows the topology of the process bus in the lab. The picture of the lab will be shown in Figure A.1 in the Appendix.

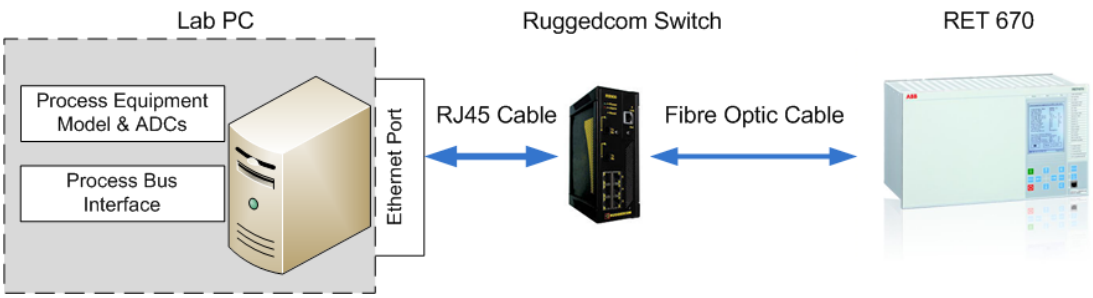


Figure 4.16: The topology of the process bus in the lab

Chapter 5

Evaluation

The evaluation method and results are presented in this chapter. The purpose of the evaluation work is to prove the process bus interface developed in Chapter 4 achieves the goal of this project defined in Section 1.6.2. In this chapter, the over current protection test and the case with two MUs are presented to demonstrate the evaluation procedure and results. The report of other evaluation work can be found in Appendix B.

5.1 Evaluation Method

The aim of the evaluation work is to prove that the process bus interface behaves like the real process bus communication equipment so that it can be used to test or illustrate the process bus concept. The evaluation idea is that if the IED operates normally with the measurement from the process bus interface, it is proved that the process bus interface is capable of simulating the real process bus communication device. The criteria of "normal operation" is that the protection function in the IED operates normally according to the configuration.

One IED is used in the evaluation work. The IED is configured with process bus communication so that it can subscribe the measurement from the process bus network. With the data from process bus network, the protection function is configured to perform the protection. The IED involved in this project is ABB RET 670. If the protection function passes the test, it can be concluded that the process bus interface can simulate the process bus SV service as the real process bus communication device such as MU.

The purpose of evaluation work is to prove the protection function can behave normally with the process bus testing environment. The performance of the protection function with process bus network is out of the scope since the accessible equipment in the lab is limited. Also there is no professional tool for measuring response time in the lab. Consequently, the response time is not recorded in this project. It is only interested that the protection function can or can not trip with the data from the process bus network.

5.2 Four Steps Phase Overcurrent Protection (OC4PTOC)

The application for this OC4PTOC is to perform short circuit protection for various equipment in power transmission system. Users can set up to four current pick up levels. The values of the current pick up levels is expressed as the percentage of the base current (I_{base}). [30] For instance, if the four current pick up levels are set to 150%, 250%, 500% and 1000% of I_{base} and I_{base}=200A, the OC4PTOC function will perform the correspond pre-set actions when the current reaches 300A (150%I_{base}), 500A (250%I_{base}), 1000A (500%I_{base}) and 2000A (1000%I_{base}). For different current pick up values, the waiting time for tripping can be set. If the current reaches the pick up value but does not last for more than the waiting time, the pre-set actions will still not be executed.

5.3 Evaluation Set-up

1. Set up the simulation model so that the measurement can be written into ".mat" files.

   As mentioned in Section 4.3, the process bus interface acquires data by the ".mat" file solution. To generate the ".mat" files, the simulation from Project

A is used. The model is shown in Figure 5.1 [5]

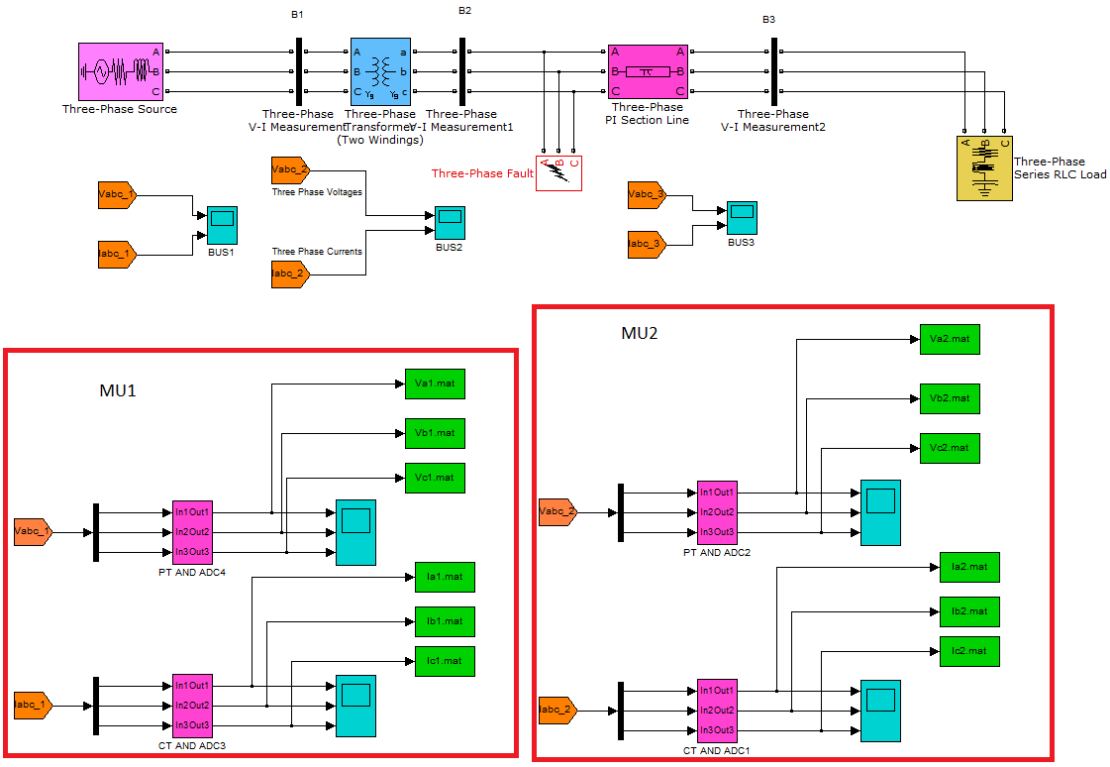


Figure 5.1: The Simulink model of three-phase fault [5]

As it is shown in Figure 5.1, the model simulates a simple three-bus power system. The three-phase to ground fault is applied at Bus 2. The measurement from Bus 2 is digitalized first and then written to the ".mat" files. The process bus interface will be run in the Matlab environment and read the values in ".mat" files. The simulation is run for 30 seconds and the waveform of the current at Bus 2 is shown in Figure 5.2.

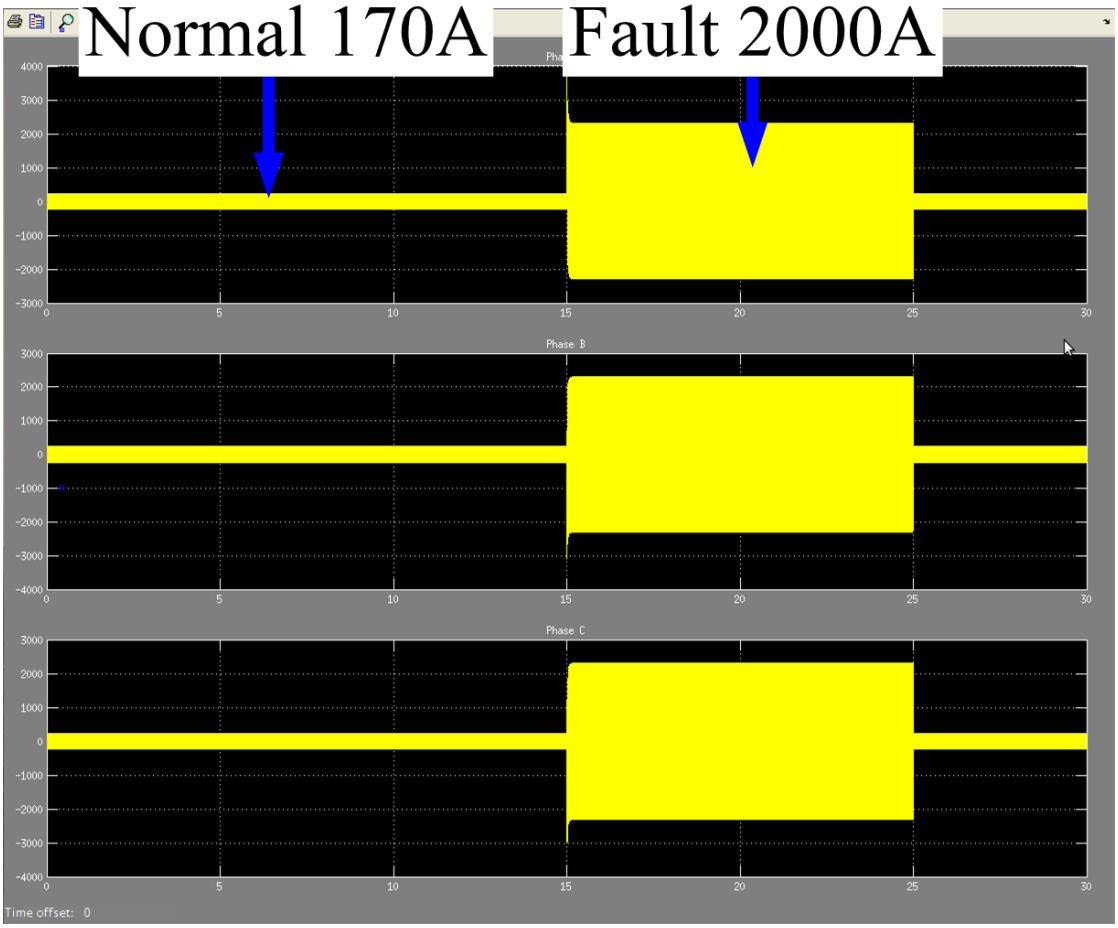


Figure 5.2: The three-phase waveform of Bus 2 current [5]

From the waveform it can be observed that the fault is introduced at 15 seconds and last for 10 seconds. The waveform shows the phase to neutral value but in the configuration of IED it should be calculated to Root Mean Square (RMS). In Figure 5.2, the normal operation current is 170A (RMS) and during fault the current can reach up to 2000A (peak value). The three-phase current and voltage measurement from Bus 2 will be taken in the overcurrent evaluation. Since all the measurement should be scaled down by the instrument transformer first and then digitalized, the saturation effect of the transformer should be considered. As a result, the maximum values the ADCs can give will not reach 2000A. The detailed explanation can be found in [5]. In this case, the maximum measurement value is around 850A (RMS).

2. Configure RET 670

   The RET 670 needs to be configured in advance to be able to take data from process bus network and run the OC4PTOC protection function. The RET 670 is configured by the tool PCM 600 which is shown in Figure 5.3.

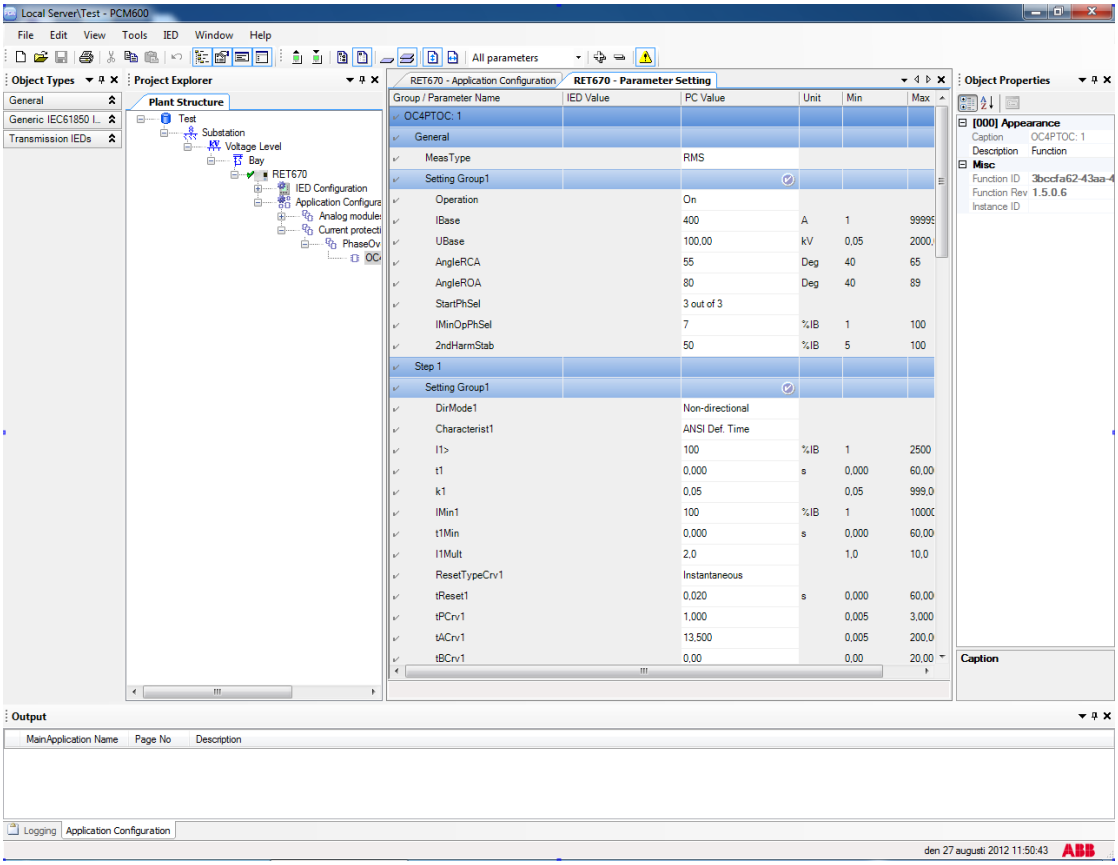


Figure 5.3: The screen shot of PCM 600

Firstly, the RET 670 should be configured with the process bus enabled mode. The IED needs to monitor the current values from MU instead of coper wires. The three-phase measurement values from MU are fed to OC4PTOC function. Figure 5.4 below shows the application configuration of the MU inputs and OC4PTOC function.

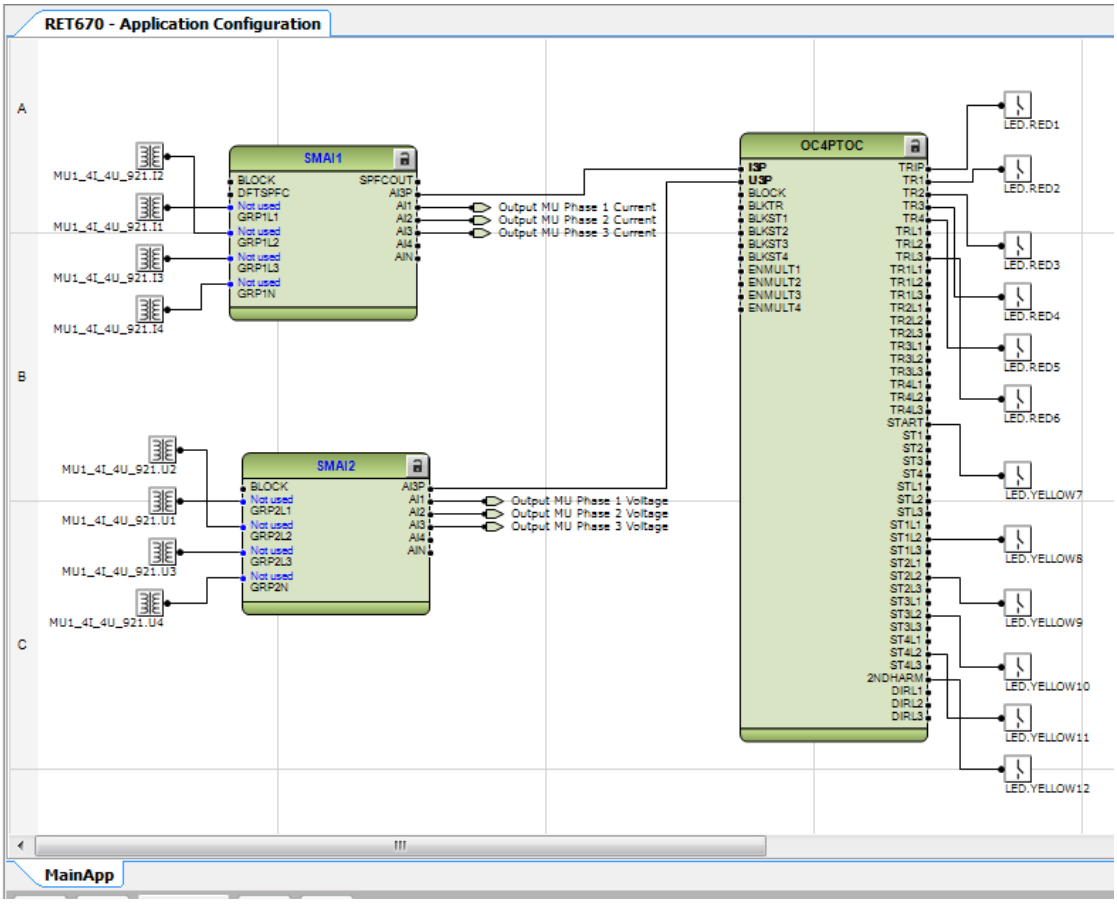


Figure 5.4: The application configuration in PCM 600

Secondly, the OC4PTOC function parameters mentioned in previous section should be set properly. Table 5.1 below lists all the values of the parameters configured for this evaluation work.

General			
Parameters		Values	
Ibase		170A(RMS)	
StartPhSel		3 out of 3	
Step 1		**Step 2**	
Parameters	Values	Parameters	Values
I1>	150%	I2>	250%
t1>	5s	t2>	2.5s
Step 3		**Step 4**	
I3>	500%	I4>	1000%
t3>	1s	t4>	0s

Table 5.1: OC4PTOC parameters

As listed in Table 5.1 above, the four current pick up levels are set to 150%, 250%, 500% and 1000% and Ibase=200A. Also the waiting time for each pick

up level is 5s, 2.5s, 1s and 0s respectively.

It is reported in the RET 670 application manual, [30], that the overcurrent protection function will give a trip signal when the current monitored by the IED is greater than the predefined level for each step. To indicate the trip signals, the Light-Emitting Diodes (LEDs) are configured. The LED in front of the RET 670 will be lighted when the trip signal is activated. The application configuration can be seen in Figure 5.4 above. The Table 5.2 below indicates which signal is assigned to which LED.

Signals	LEDs	Signals	LEDs
General Trip	RED1	General Start	YELLOW7
Trip for Step 1	RED2	Start signal from step1 phase L2	YELLOW8
Trip for Step 2	RED3	Start signal from step2 phase L2	YELLOW9
Trip for Step 3	RED4	Start signal from step3 phase L2	YELLOW10
Trip for Step 4	RED5	Start signal from step4 phase L2	YELLOW11
Trip signal from phase L3	RED6	Block from 2^{nd} harmonic detection	YELLOW12

Table 5.2: The map of LED

3. Set up the process bus hardware connections

   The process bus network set-up is already introduced in Section 4.4. According the topology shown in Figure 4.16, the process bus hardware connection can be set up. The picture of the hardware set-up in the lab is shown in Figure A.1 and Figure A.2 in the Appendix.

5.4 Results

As configured in the simulation of Project A, at the beginning, there is no fault in the power system. As shown in Figure 5.5 below, the IED recognizes the value sent by the process bus interface and there is no LED lighted. The values can be shown on the local Human Machine Interface (HMI) on RET 670 as follow in Figure 5.5.

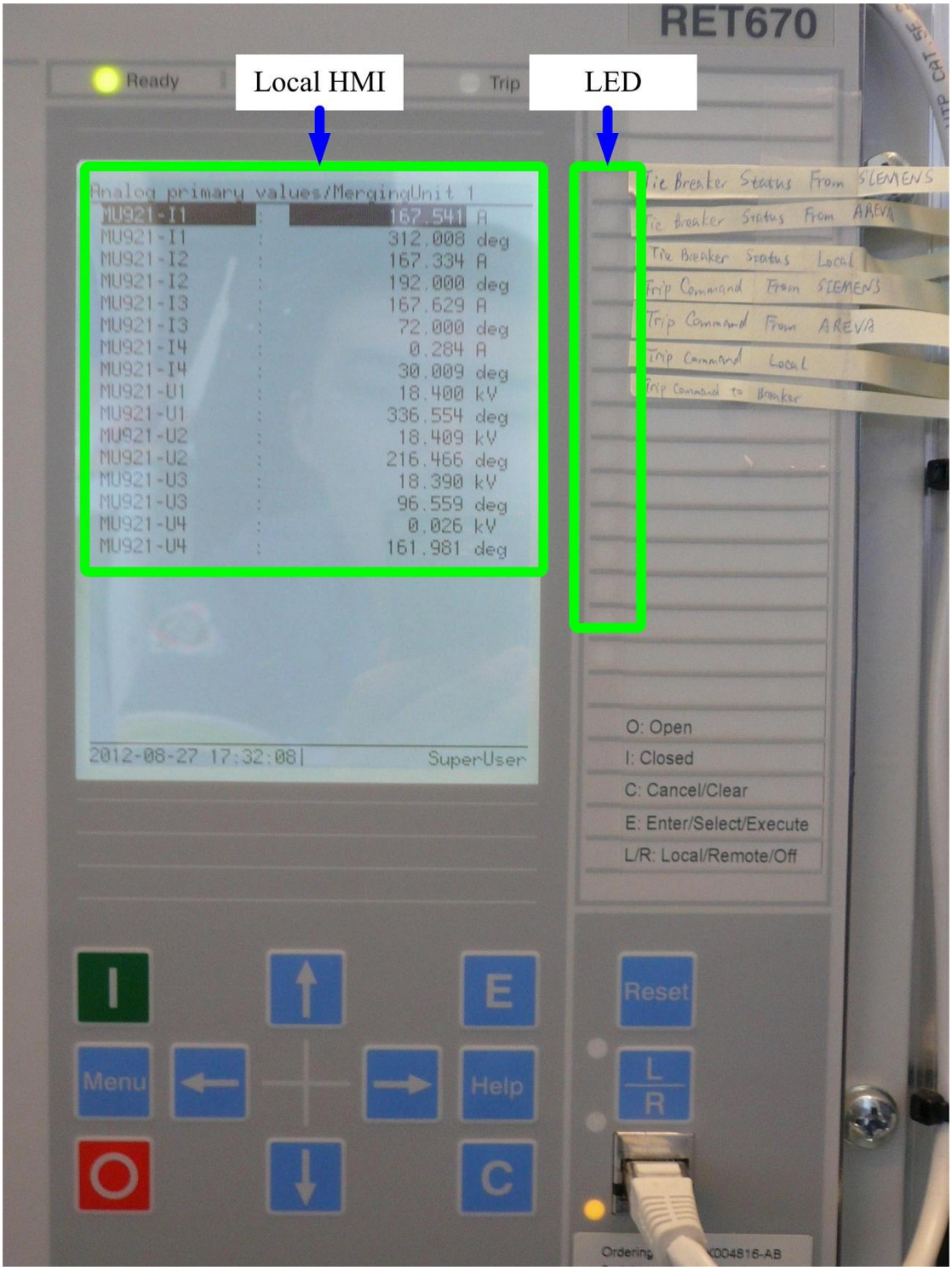
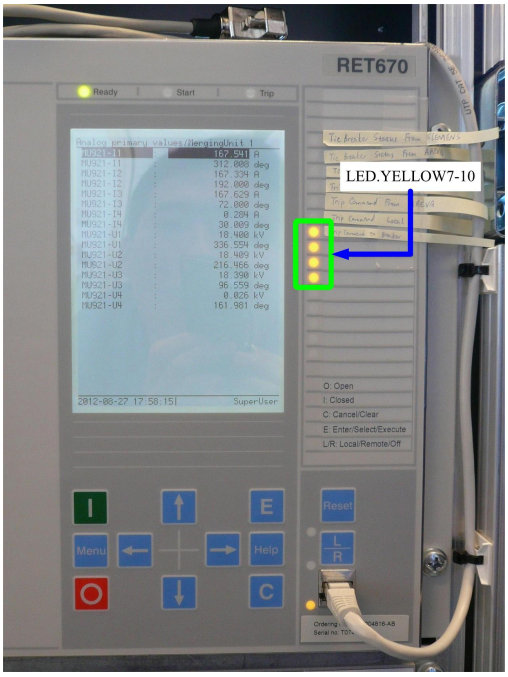


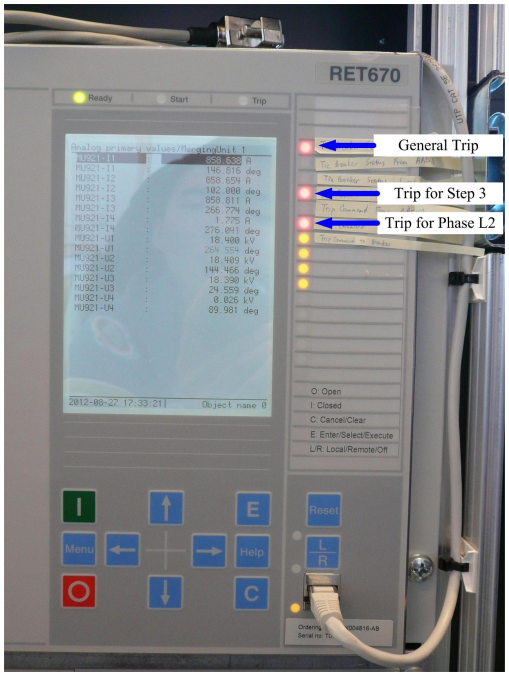
Figure 5.5: The reading values from HMI

It can been seen that the values shown on the screen is RMS values which is the same as it in the simulation waveform in Figure 5.2. Figure 5.5 proves that the process bus acquires the data from ".mat" file successfully and IED can recognize the SV stream correctly. The values are recognized as "MU92" which stands for 9-2 standard merging unit.

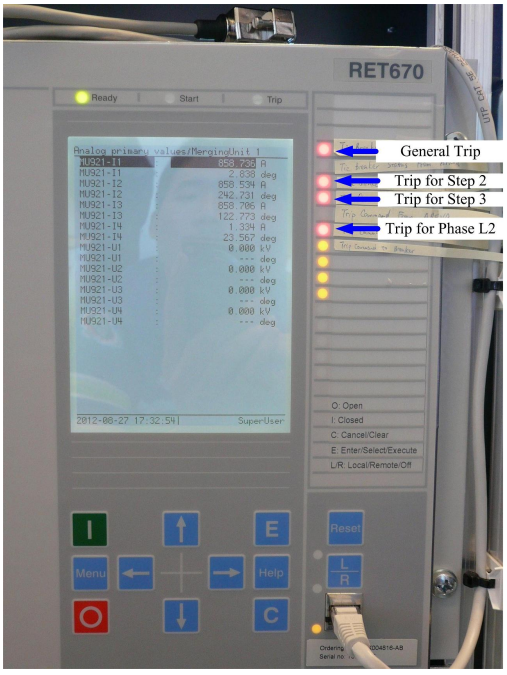
Since the fault lasts for 10 seconds, and the peak fault value is 850A (RMS) which is greater than the pick up current of step 1, 2 and 3 but not step 4, the step 1, 2 and 3 should be tripped while the step 4 should not be tripped. The first step (150%) should be tripped after the fault lasts for 5s. The second step is tripped after 2.5s and the third step waits for only 1s. Figure 5.6 shows the results.



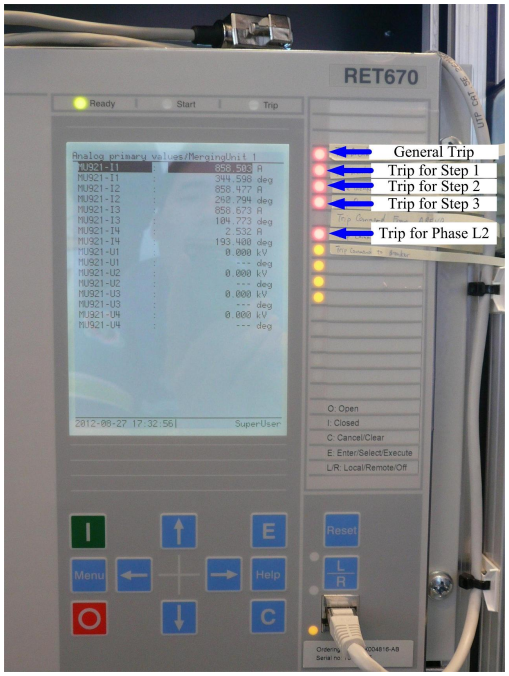
(a) The fault current is picked up



(b) Step 3 trips after 1s



(c) Step 2 trips after 2.5s



(d) Step 1 trips after 5s

Figure 5.6: The trips of OC4PTOC

From Figure 5.6 it is shown that the OC4PTOC function behaves exactly as it

is configured which in turn proves that the IED can operate correctly with SV stream sent by the process bus interface.

5.5 Sending SV Stream from Two MUs

One of the advantages of the light weight MU testing environment is that it is easy to set up the scenario with multiple MUs. It will be expensive and inflexible if physical MUs, such as the set-up in Figure 2.1, are utilized for multiple MUs testing. In order to evaluate the performance of sending out multiple MUs, the transformer differential protection function in RET 670 is configured and tested. The detailed evaluation of differential protection is in Appendix B. In this section, only the evaluation related to the process bus communication interface is reported.

The principle of differential protection is introduced in Appendix B.2. The differential protection need measurement from both primary and secondary sides of the transformer so two MUs are required to measure both sides. To send out different data sets for two MUs, two SV packet structures are defined in the code and the "sendto()" command is called twice to send two different data sets through the same Ethernet port. The Wireshark capture is shown in Figure 5.7 below.

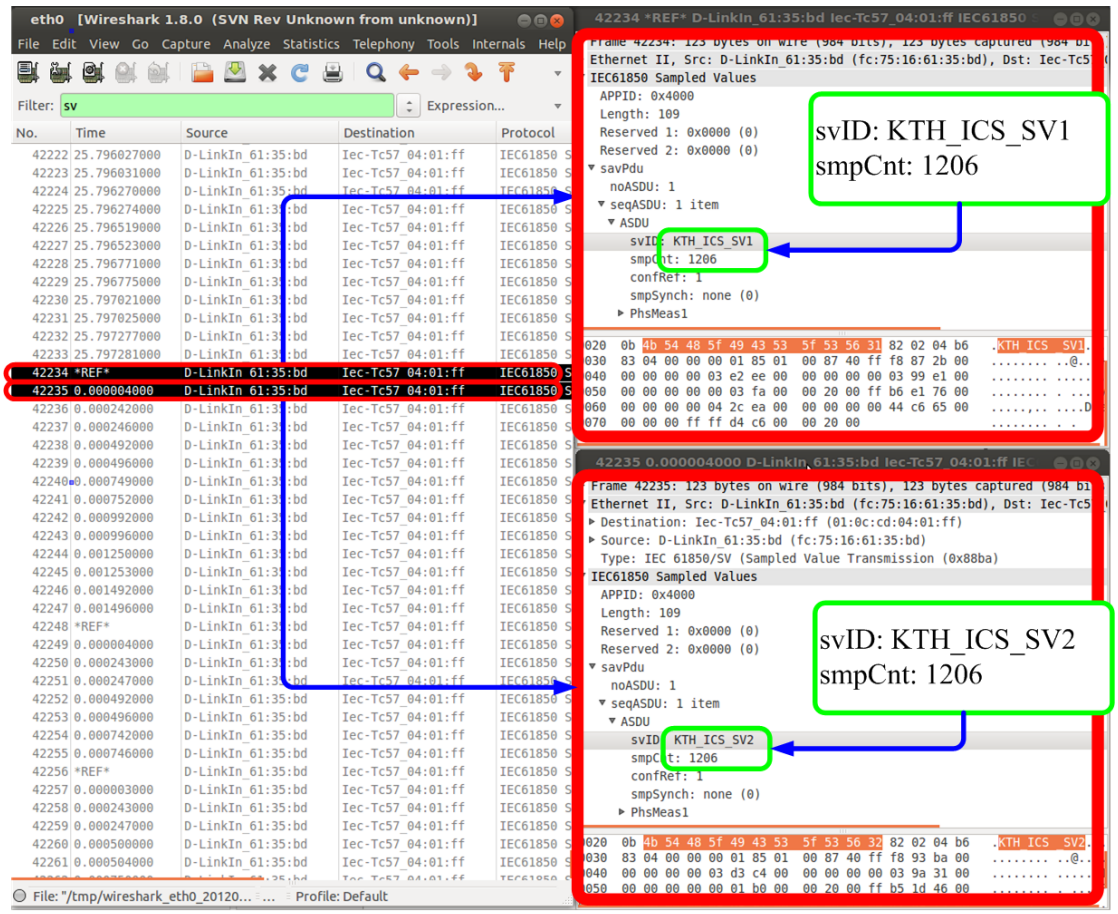
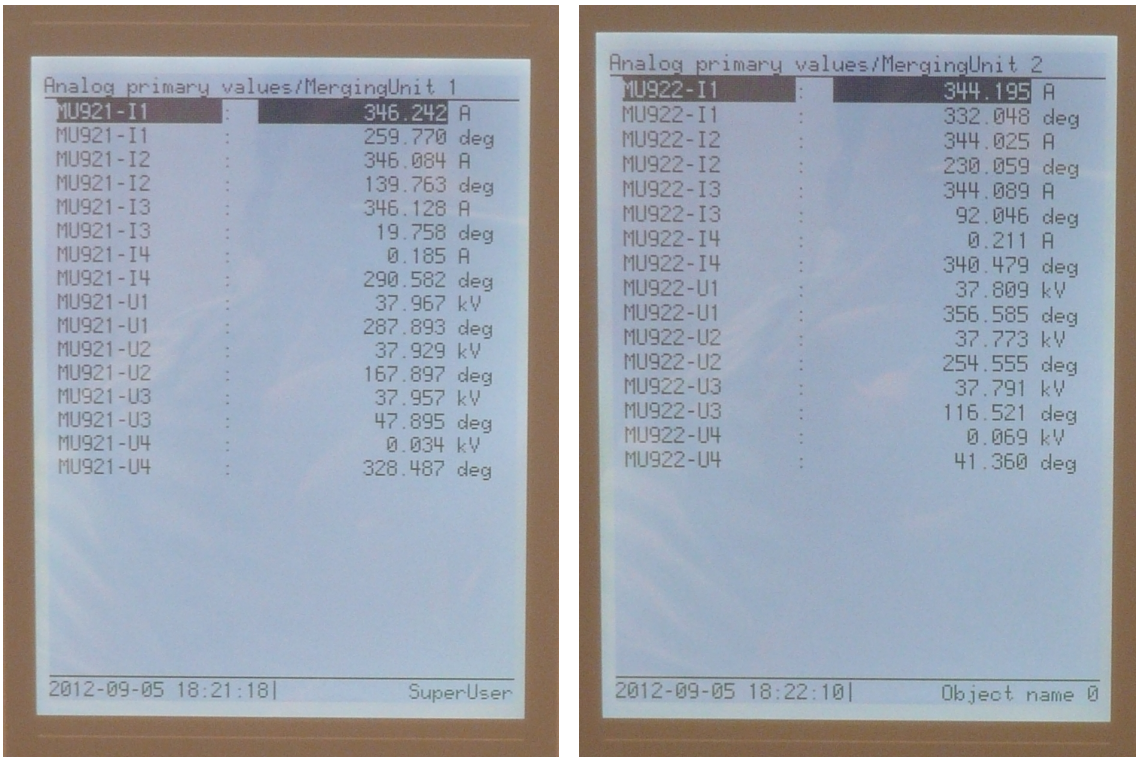


Figure 5.7: The Wireshark capture of sending SVs for two MUs

As shown in Figure 5.7 above, the time difference between two SV packets of different MUs is about 4μs. Due to the limited access to measuring tools, it can not be measured precisely how much those several microseconds difference will affect the protection function. According to the evaluation results shown in Appendix B.2, these several microseconds do not affect the normal operation of the differential function. RET 670 IED can recognize SVs from both MUs. Figure 5.8 shows the reading from RET 670.



(a) Reading of MU1 (b) Reading of MU2

Figure 5.8: Reading from RET 670 HMI

Chapter 6

Conclusions and Future Work

Based on the evaluation in Chapter 5, the conclusion is to be drawn in this chapter. During the execution of this project, new ideas keep coming up. Due to the time limitation, only some of the new ideas could be studied. Those interesting but unstudied ideas are proposed as the future work of the project in this chapter.

6.1 Conclusions

In this project, the process bus communication interface for the light weight MU testing environment is developed. The iterative research pattern is applied in order to finish the project in a manageable way. By executing three iterations, the goal of the project is successfully achieved. Protection function testing is then performed to evaluate the capability of this process bus interface. The evaluation results show that the IED can recognize the SV service from the communication interface and the protection functions operate correctly. Instead of purchasing the expensive physical MUs testing equipment, the electric utility integration engineers and researchers can just use this light weight MU testing environment (Project A and B) for basic testing and demonstrating the process bus technology concept. The whole environment can be easily run on any PC with normal processing power. The required software is just Matlab which is a commonly used and accessible research tool. The operating system (OS) should be Linux which is also a commonly used OS among researchers. The project is also extendible. Some interesting ideas could be implemented to make the testing environment be able to simulate the process bus more accurately and provide more functions.

6.2 Future Work

The process bus technology is just introduced recently and there is large space for developing testing tool. The process bus interface developed in this project is a part of the process bus testing environment. The present "soft" MU testing environment is simple and can not reach industry level testing accuracy so it can not act as a professional testing tool. But it has the potential to be improved to a professional testing tool. Here are some suggestions for the continuous work.

6.2.1 Time Synchronization

The SV traffic is time critical communication so the time synchronization between each process bus device is important. In this project, if there are different data sets from several MUs need to be sent out, the different data sets will be sent out by different commands in one source code. As a result, the time differences between each data set is at the level of several micro seconds (around 4μs). The evaluation of sending out two data sets from two MUs is presented in Appendix B.2. The results shows the protection function can operate normally which means the "4μs" is acceptable. Thus, in two MUs case, the time synchronization between each MU is not necessary. However, there is no synchronization between the IED, the MUs and the switch. The RET 670 supports IEEE 1588 synchronization protocol. The IEEE 1588 synchronization implementation could be the future work to improve this testing environment. The process bus interface could be further programmed to be able to receive 1PPS and send 4000 samples per second according to the synchronization instead of the algorithm presented in Section 4.2.3.

6.2.2 Close Loop Test

In this project, the RET 670 is configured in the way that the LED light shows the trip of the protection function. In the real world, the trip signal is to trip a certain kind of actions such as interlocking function and open/close the circuit breakers. In [31], it proved that the trip signal by Generic Object Oriented Substation Event (GOOSE) message has the same performance as the copper wired trip. It is feasible to use GOOSE message to send the trip. Based on this point, a GOOSE receiver could be programmed to receive the GOOSE trip message so that the testing environment has not only sending the SV packets to IEDs but also receiving the trip GOOSE from the IEDs. The testing loop then is closed.

6.2.3 Real-Time Solution

The interface between Project A and B is an "off-line" solution. The ADC model of Project A needs to be run first so all the data are recorded in the ".mat" files and then Project B reads all the data from those files and send them out to the process bus. It is not that convenient to run Project A and B separately. One solution is already proposed in Section 4.3, the "S-function". A powerful computer is required to run the "s-function" and the Simulink power system model 4000 times per second. This is not a practical solution since the light weight testing environment focuses on the "light weight" which means it should run on most of the computers in the lab.

Another solution could be that dividing the model of the whole project into several small parts and run them on several processors. The model should be divided in the way that the sub-models are "light" enough to be run 4000 times per seconds on the common computers. The way of splitting the model should be found.

Besides, the real-time problem might be solved by running the Simulink in C code. The core of Simulink model is C code. The reason why the Simulink model takes a lot of time may be that it compiles the C code of the model every step the Simulink is run. If the Simulink model could be transferred into raw C code and compiled outside Matlab environment, the process bus interface code could be embedded into the Simulink C code and the whole source code could be run as a normal C program other than a Simulink model. By this method, the simulation time could be reduced.

6.2.4 Interoperability

The interoperability is one big advantage of IEC 61850 which ensures the IEDs from different vendors can work together. Since the Siemens and Ariva IEDs in the lab have not updated to support process bus during the thesis project, the interoperability of process bus has not been evaluated in this project. One possible testing idea could be that all of the IEDs are connected to the process bus and configured with process bus protocol enabled. Each IED is configured with different protection functions and subscribes different measurement from the

MU testing environment. Even some interlock functions could be set up between different IEDs by GOOSE. The operation status of the "mixed" IEDs' protection system can be investigated and documented.

Bibliography

[1] V. Vyatkin, G. Zhabelova, N. Higgins, K. Schwarz, and N. Nair, "Towards intelligent Smart Grid devices with IEC 61850 Interoperability and IEC 61499 open control architecture," in *Transmission and Distribution Conference and Exposition, 2010 IEEE PES*, pp. 1–8, IEEE, 2010.

[2] "International electrotechnical commission." `http://www.iec.ch/`, 2012.

[3] International Electrotechnical Commission TC/SC 57, "IEC 61850 Communication networks and systems for power utility automation - Part 9-2: Specific communication service mapping (SCSM) - Sampled values over ISO/IEC 8802-3," September 2011.

[4] J. Liu, K. Li, and H. Yang, "The design of a merging unit of electronic transformer based on arm," in *Universities Power Engineering Conference, 2007. UPEC 2007. 42nd International*, pp. 712 –716, sept. 2007.

[5] Z. A. Khurram, "Interface Between Process Equipment And Process Bus for Light Weight Testing Of Protection Functions," Master's thesis, Royal Institute of Technology, August 2012.

[6] X. Sun and M. Redfern, "Process Bus Configurations for Protection Schemes in the Digital Substation: IEC 61850," *Universities' Power Engineering Conference (UPEC), Proceedings of 2011 46th International*, pp. 1 –6, sept. 2011.

[7] S. Kucuksari and G. Karady, "Development of test facility for compatibility and performance testing of all-digital protection systems connected to IEC 61850-9-2 standard," in *Power & Energy Society General Meeting, 2009. PES'09. IEEE*, pp. 1–8, IEEE, 2009.

[8] V. Theoharakis and D. Serpanos, *Enterprise networking: multilayer switching and applications*. IGI Global, 2002.

[9] K. Pratt, "Design patterns for research methods: Iterative field research," 2009.

[10] UCA International Users Group, "Implementation Guidance for Digital Interface to Instrument Transformers Using IEC 61850-9-2," July 2004.

[11] N. Higgins, V. Vyatkin, N. Nair, and K. Schwarz, "Distributed power system automation with IEC 61850, IEC 61499, and intelligent control," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 41, no. 1, pp. 81–92, 2011.

[12] S. Roostaee, R. Hooshmand, and M. Ataei, "Substation automation system using IEC 61850," in *Power Engineering and Optimization Conference (PEOCO), 2011 5th International*, pp. 393 –397, june 2011.

[13] M. Kanabar and T. Sidhu, "Performance of IEC 61850-9-2 process bus and corrective measure for digital relaying," *Power Delivery, IEEE Transactions on*, no. 99, pp. 1–1, 2011.

[14] S. D. Ramchurn, P. Vytelingum, A. Rogers, and N. R. Jennings, "Putting the 'smarts' into the smart grid: a grand challenge for artificial intelligence," *Commun. ACM*, vol. 55, pp. 86–97, Apr. 2012.

[15] U.S. Department of Energy, "Grid 2030: A national vision for electricitys second 100 years.," tech. rep., United States Department of Energy, Office of Electric Transmission and Distribution, July 2003.

[16] D. Von Dollen, "Report to NIST on the smart grid interoperability standards roadmap," *EPRI, Contract No. SB1341-09-CN-0031–Deliverable*, vol. 7, 2009.

[17] B. Pickett, T. Sidhu, S. Anderson, A. Apostolov, B. Bentert, K. Boers, O. Bolado, P. Carroll, S. Chano, A. Chaudhary, F. Cobelo, K. Cooley, M. Cooper, R. Cornelison, P. Elkin, A. Elneweihi, R. Garcia, K. Gardner, T. Kern, B. Kasztenny, G. Michel, J. Niemira, T. Nissen, S. Charles, D. Ware, and F. Plumptre, "Reducing outage durations through improved protection and autorestoration in distribution substations," *Power Delivery, IEEE Transactions on Power Delivery*, vol. 26, pp. 1554 –1562, july 2011.

[18] C. Brunner, "IEC 61850 Process bus–Challenges and benefits," 2009.

[19] Z. Djekic, L. Portillo, and M. Kezunovic, "Compatibility and interoperability evaluation of all-digital protection systems based on IEC 61850-9-2 communication standard," in *Power and Energy Society General Meeting-Conversion and Delivery of Electrical Energy in the 21st Century, 2008 IEEE*, pp. 1–5, IEEE, 2008.

[20] F. Engler, T. Kern, L. Andersson, B. Kruimer, G. Schimmel, and K. Schwarz, "IEC 61850 based digital communication as interface to the primary equipment," *Proc. CIGRE*, pp. B3–205, 2004.

[21] D. McGinn, V. Muthukrishnan, and M. Adamiak, "Designing copper wiring out of high voltage substations: A practical solution and actual installation," in *Protective Relay Engineers, 2009 62nd Annual Conference for*, pp. 63–82, IEEE, 2009.

[22] R. Petersen, *Linux: The Complete Reference, Sixth edition.* McGraw-Hill/Osborne, 2008.

[23] G. Gunnarsson, "UDP made simple." `http://www.abc.se/~m6695/udp.html`, October 2007.

[24] Chaitu, "Sending ARP request in LINUX using C." `http://linux-spawn.blogspot.se/2009/10/sending-arp-request-in-linux-using-c.html`, October 2009.

[25] "Linux man pages." http://linux.die.net/man/.

[26] D. C. Plummer, "An Ethernet Address Resolution Protocol–or–Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware." http://tools.ietf.org/html/rfc826, November 1982.

[27] Fred N. van Kempen, Donald Becker, Alan Cox, and Steve Whitehouse, "TOMOYO Linux Cross Reference." http://tomoyo.sourceforge.jp/cgi-bin/lxr/source/include/linux/if_ether.h#L63, August 1994.

[28] "Application need root permission to run." http://www.linuxquestions.org/questions/programming-9/application-need-root-permission-to-run-921772/, March 2012.

[29] "Matlab documentation." http://www.mathworks.se/help/techdoc/, 2012.

[30] ABB, *Transformer protection RET670 Application manual*, June 2010.

[31] T. Sidhu, M. Kanabar, and P. Parikh, "Configuration and performance testing of IEC 61850 GOOSE," in *Advanced Power System Automation and Protection (APAP), 2011 International Conference on*, vol. 2, pp. 1384 –1389, oct. 2011.

[32] F. Rekina and D. Ouahdi, "Analysis of the effects of magnetizing inrush current on power transformer differential protection," in *Proceedings of the 6th conference on Applications of electrical engineering*, pp. 65–70, World Scientific and Engineering Academy and Society (WSEAS), 2007.

[33] Z. Gajić, *Differential Protection for Arbitrary Three-Phase Power Transformers*. Department of Industrial Electrical Engineering and Automation, Lund University, 2008.

Appendix A

ICS IEC 61850 Lab Set-up

The process bus topology is already presented in Figure 4.16. The light weight MU testing environment is set up according to that topology. Figure A.1 shows the hardware configuration in the lab.

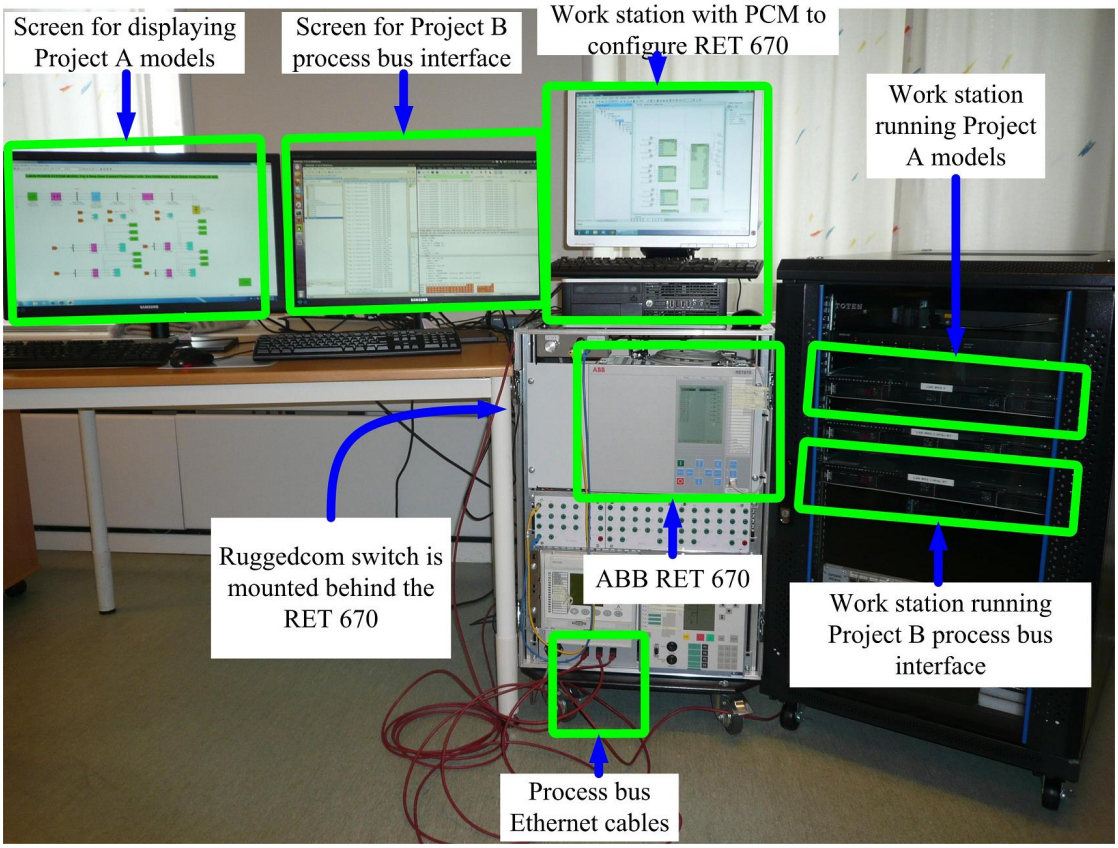


Figure A.1: The hardware configuration in the lab

As shown in Figure A.1, Project A and B are run on different computers. When the simulation from Project A is done, the ".mat" files will be copied to the work station with Project B running. Project B then can read the measurement and send out SV stream. The process bus uses a star configuration where the RET 670 and the work stations are connected through a Ruggedcom switch. The cables

between the switch and work stations are normal RJ45 cables. The connection between RET 670 and the switch is using fibre optical cable. Figure A.2 below shows the back side of the RET 670 rack where the Ruggedcom is mounted.

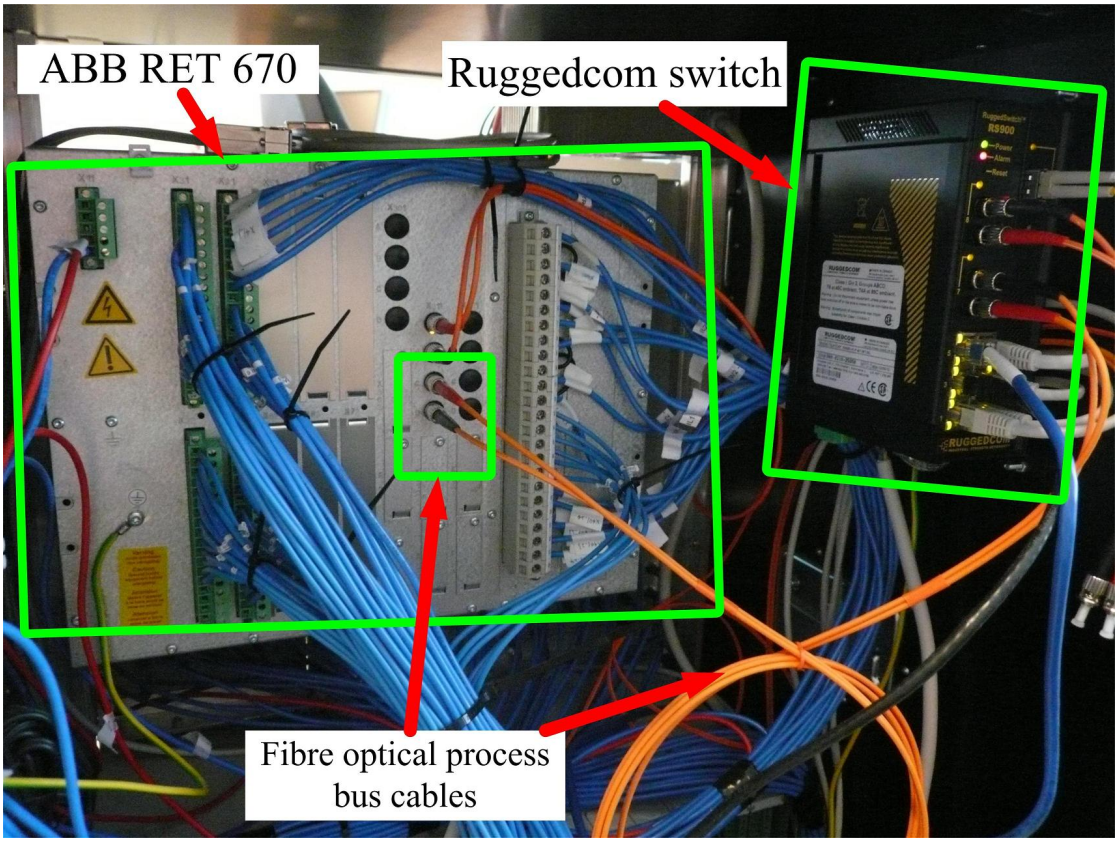


Figure A.2: Back side view of IED rack

Appendix B

Evaluation Report

B.1 Four Steps Phase Overcurrent Protection OC4PTOC with 2^{nd} Harmonics

The OC4PTOC without 2^{nd} Harmonics is already discussed in Chapter 5 so only the principle of harmonic restrain blocking function is introduced in next section.

Principle of the harmonic restrain blocking function

Over excitations of a transformer can cause unnecessary operation of transformer differential protection function. These over excitations can also be caused by the inrush currents or by sudden voltage rise. One condition might be the loss of a big load connected to the secondary of transformer and the primary is still energized. When the primary winding of a transformer is overexcited and driven into saturation as a result more power appears to be flowing in than flowing out of the secondary winding. As over excitation phenomena is developed due to some internal dynamics caused in the power system it also generates some of the other harmonics with the fundamental [32]. This situation causes the relay towards wrong trip condition which is not acceptable at any cost. Now it is necessary to have a algorithm in the relay to clearly discriminate between the over excitation and the faulty states. Therefore the relay we are using in our testing facility is RET 670 which is equipped with second harmonic restrain function. This function will not allow the trip due to inrush currents in the transformer. Power transformers can have a large inrush current, when being energized. This phenomenon is due to saturation of the transformer magnetic core during parts of the period. There is a risk that inrush current will reach levels above the pick-up current of the phase over current protection. The inrush current has a large 2nd harmonic content. This can be used to avoid unwanted operation of the protection. Therefore, OC4PTOC have a possibility of 2nd harmonic restrain if the level of this harmonic current reaches a value above a set percentage of the fundamental current.[30]

To avoid unwanted trip due to inrush current in the transformer, 2nd harmonic current is monitored. The 2nd harmonic current is compared to a pre-set restrain current level. If 2nd harmonic current exceeds the set level, the over current trip

from all steps will be blocked.

Testing setup

1. Simulink power system model which is used to generate testing data.

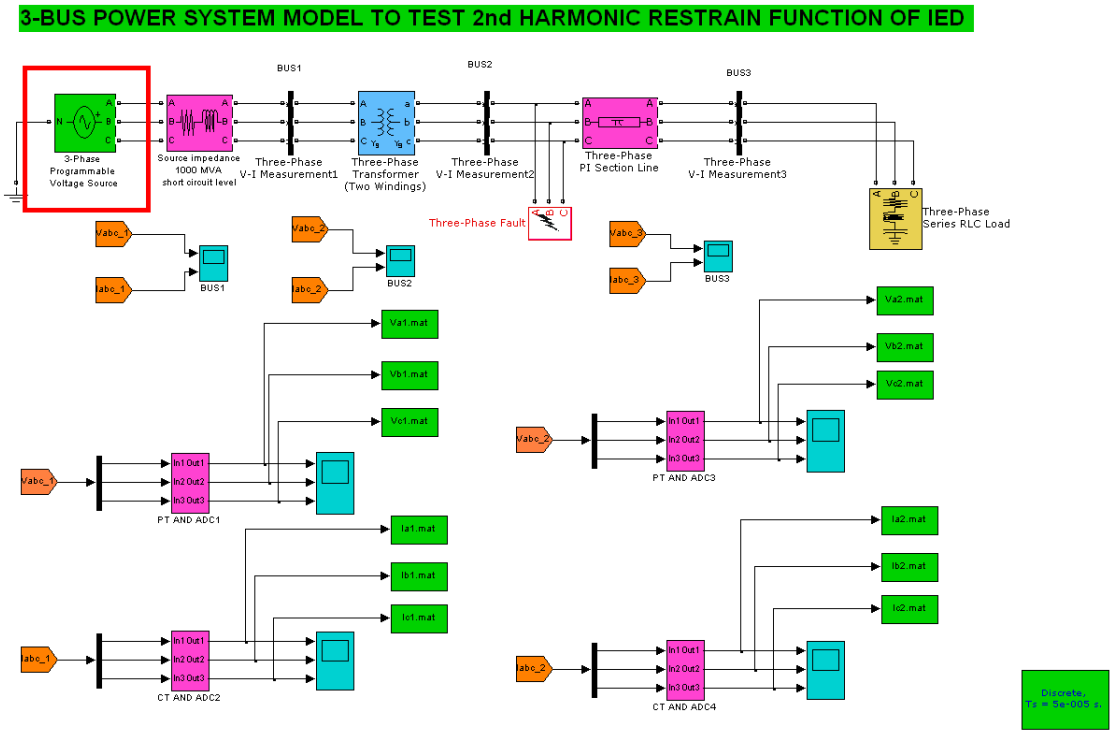


Figure B.1: The Simulink model of the testing

As shown in Figure B.1, the red rectangle marks the block which can inject 2^{nd} harmonic into the system. The fault is still the three-phase to ground fault. The time line of the simulation is shown in Figure B.2 below.

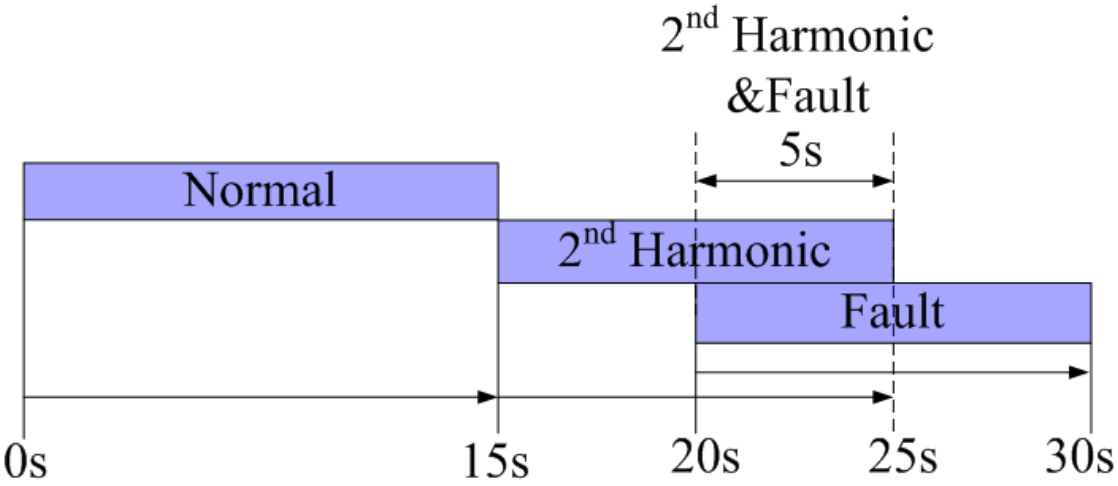


Figure B.2: The scheme of the testing

To test the 2^{nd} harmonic restrain in OC4PTOC function, the 2^{nd} harmonic is introduced with fault during 20-25s. The OC4PTOC function should not trip during that period. The OC4PTOC function will trip after 25s when the 2^{nd} harmonic is removed.

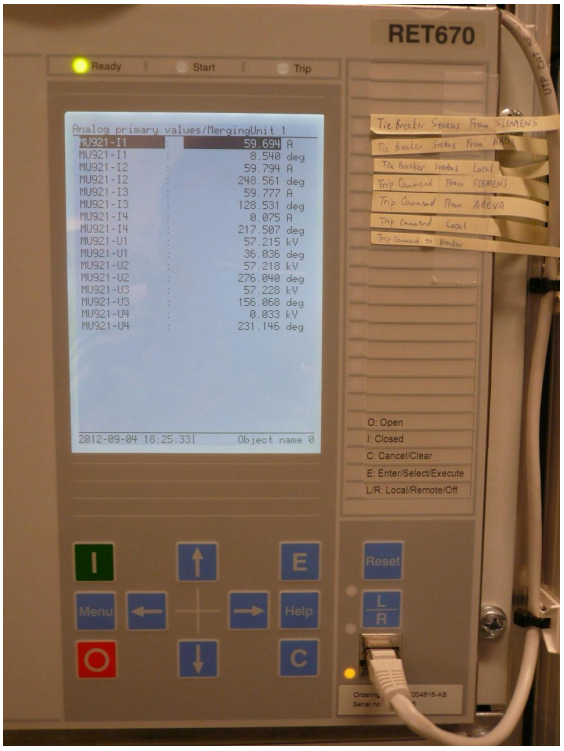
2. The application configuration is the same as the OC4PTOC in Figure 5.4, Table 5.1 and Table 5.2.

3. Parameter configuration of OC4PTOC is listed in Table B.1.

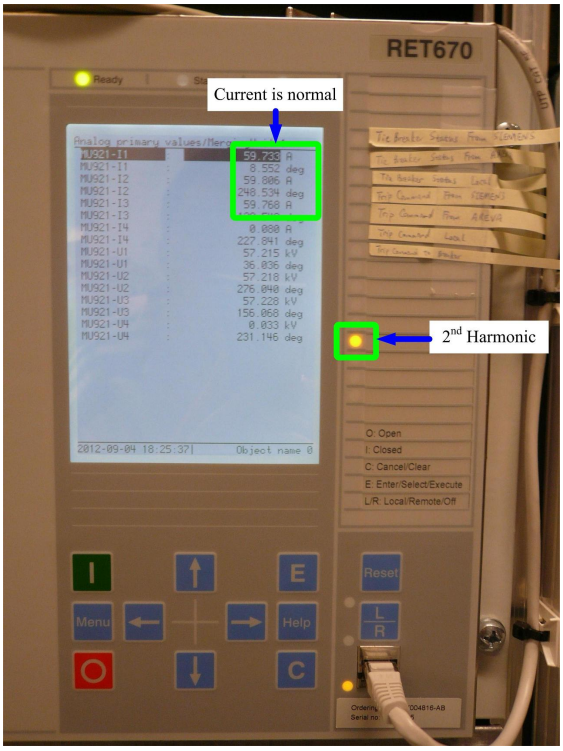
General			
Parameters		Values	
Ibase		170A(RMS)	
StartPhSel		3 out of 3	
2ndHarmStab		50%IB	
Step 1		**Step 2**	
Parameters	Values	Parameters	Values
I1>	150%IB	I2>	250%IB
t1>	5s	t2>	2.5s
HarmRestrain1	On	HarmRestrain2	On
Step 3		**Step 4**	
I3>	500%IB	I4>	1000%IB
t3>	1s	t4>	0s
HarmRestrain3	On	HarmRestrain4	On

Table B.1: OC4PTOC settings for 2^{nd} harmonic restrain

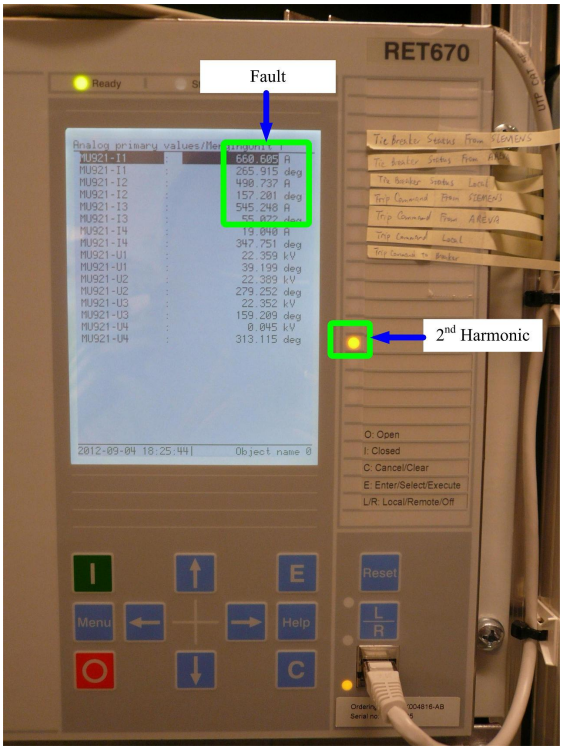
Results



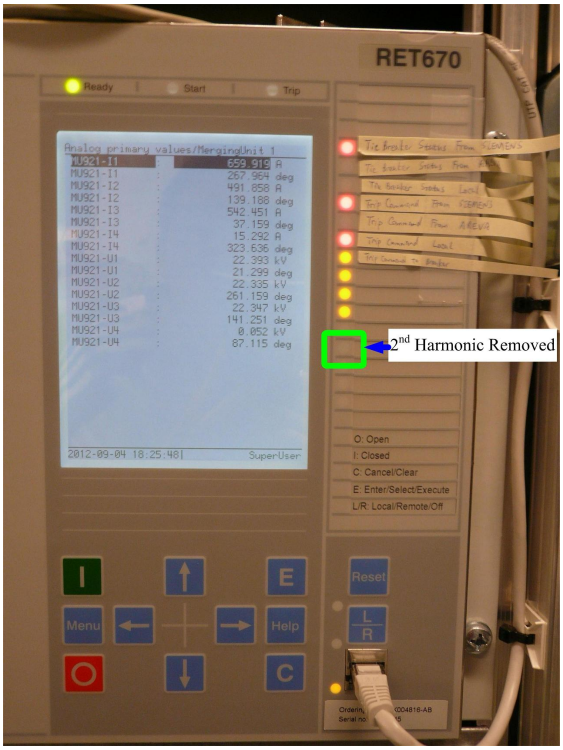
(a) Normal condition during 0-15s



(b) The 2^{nd} harmonic is introduced at 15s



(c) The fault is injected after 20s but no trip



(d) The 2^{nd} harmonic is removed and Step 3 is trip

Figure B.3: The results of OC4PTOC with 2^{nd} harmonic

B.2 Two Windings Transformer Differential Protection (T2WPDIF)

Principle of Transformer Differential Protection

Differential relays are often used as main protection for all important elements of the power system such as generators, transformers, buses, cables and overhead lines. The protected zone is clearly defined by the positioning of the main current transformers to which the differential relay is connected [33].

The principle of transformer differential operation is the comparison of value of current on both primary and secondary sides of the transformer. Under normal conditions I1 and I2 are equal and opposite such that the resultant current through the relay is zero. If the difference between the two currents is greater than the set threshold value the relay will detect the fault [33].

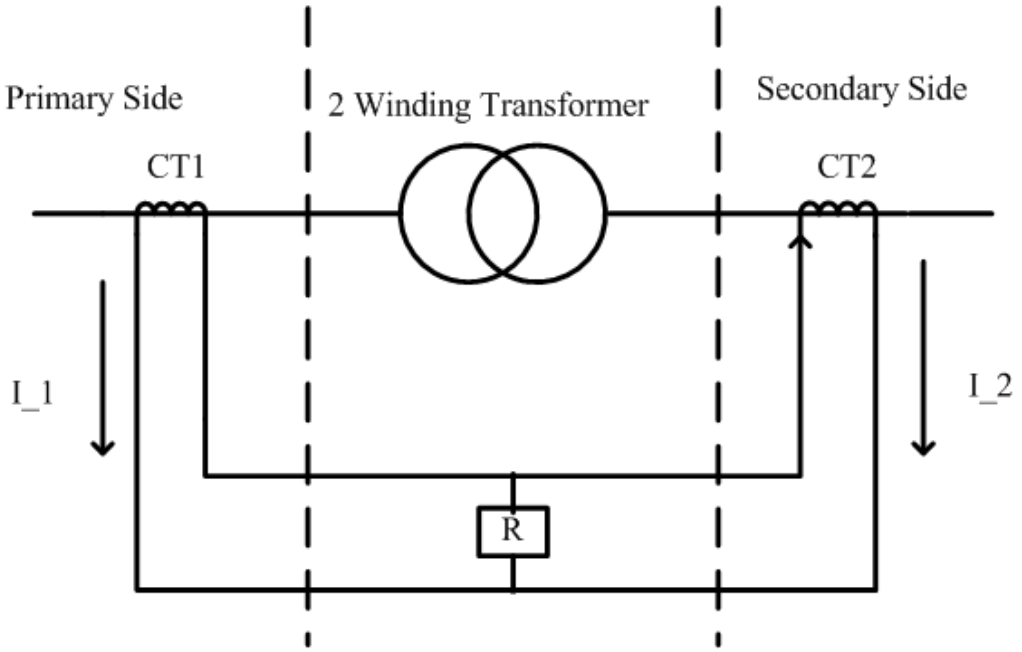


Figure B.4: Schematic of two windings transformer differential protection

Principle of T2WPDIF

As we are using ABB RET 670 differential protection relay for the transformer. RET 670 has both two winding and three winding differential protection functions which is shown in Figure B.5. In this evaluation work we used the two winding transformer protection block. The brief introduction about T2WPDIF is provided here for better understanding of the differential protection function of RET 670.

Function description	IEC 61850 identification	IEC 60617 identification	ANSI/IEEE C37.2 device number
Transformer differential protection, two-winding	T2WPDIF	3Id/I	87T
Transformer differential protection, three-winding	T3WPDIF	3Id/I	87T

Figure B.5: The transformer differential protection provided in RET 670

A transformer differential protection compares the current flowing into the transformer with the current leaving the transformer. A correct analysis of fault conditions by the differential protection must take into consideration changes due to voltages, currents and phase angle changes caused by protected transformer. Traditional transformer differential protection functions required auxiliary transformers for correction of the phase shift and ratio. The numerical microprocessor based differential algorithm as implemented in the IED compensates for both the turns-ratio and the phase shift internally in the software. No auxiliary current transformers are necessary. [30]

Testing Setup

1. Set up the offline testing model for differential protection

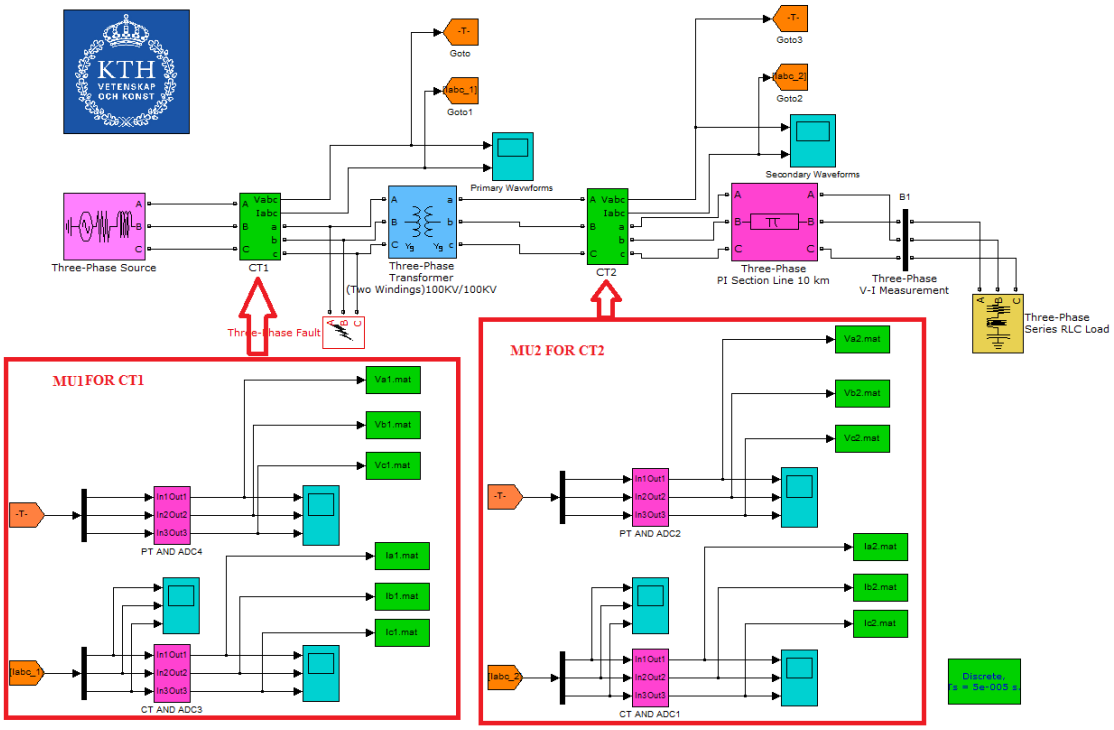


Figure B.6: The Simulink model for differential protection

As shown in Figure B.6 above, the power system comprises of a voltage source, two winding power transformer, a transmission line and a load. The transformer winding ratio is 1:1. The model is simulated in offline mode for 30 seconds to collect the measurements. The three phase short circuit fault is applied on the primary side of the transformer from 16 to 25 seconds. The Merging Unit 1 and Merging Unit 2 named "KTH_ICS_SV1" and "KTH_ICS_SV2" respectively are connected to the CT1 and CT2. After the simulation, the waveforms of the primary and secondary sides are shown in Figure B.7 below.

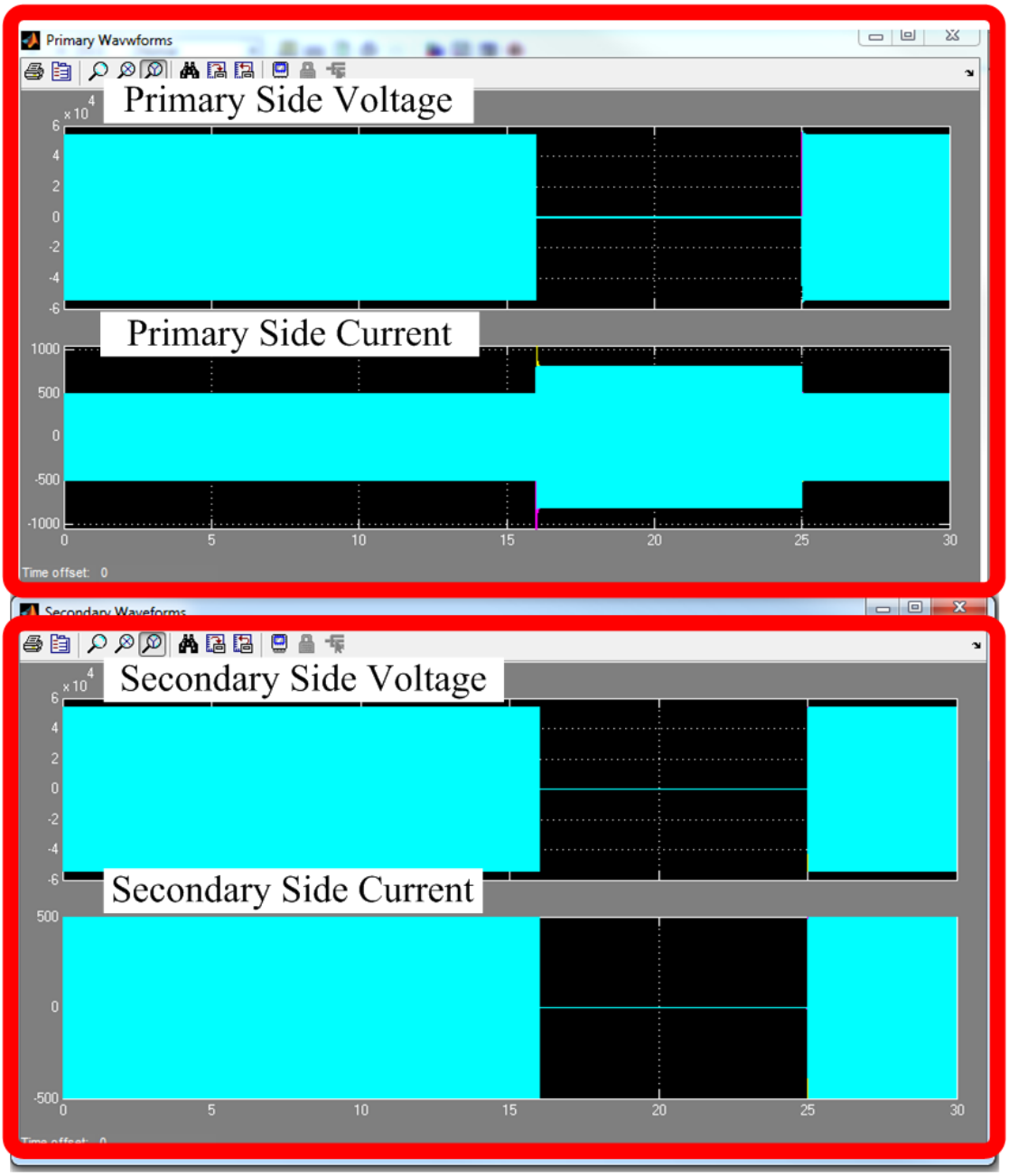


Figure B.7: The waveforms of primary and secondary sides

Druing the normal operation, the current on both sides is around 500A at peak. During the fault, the primary side current increases to approximately 700A at peak and the secondary side current goes to almost zero. As shown in Figure B.6, the three-phase to ground fault happens in the primary side of the transformer, all the current comes into the primary side will go to ground during the fault. There will be no current on the secondary side during the fault. During the fault, the current difference between primary side and secondary side is about 500A.

2. Set up the process bus communication interface. In this test, the process bus interface should read measurement from both primary and secondary sides

of transformer and send both of them out.

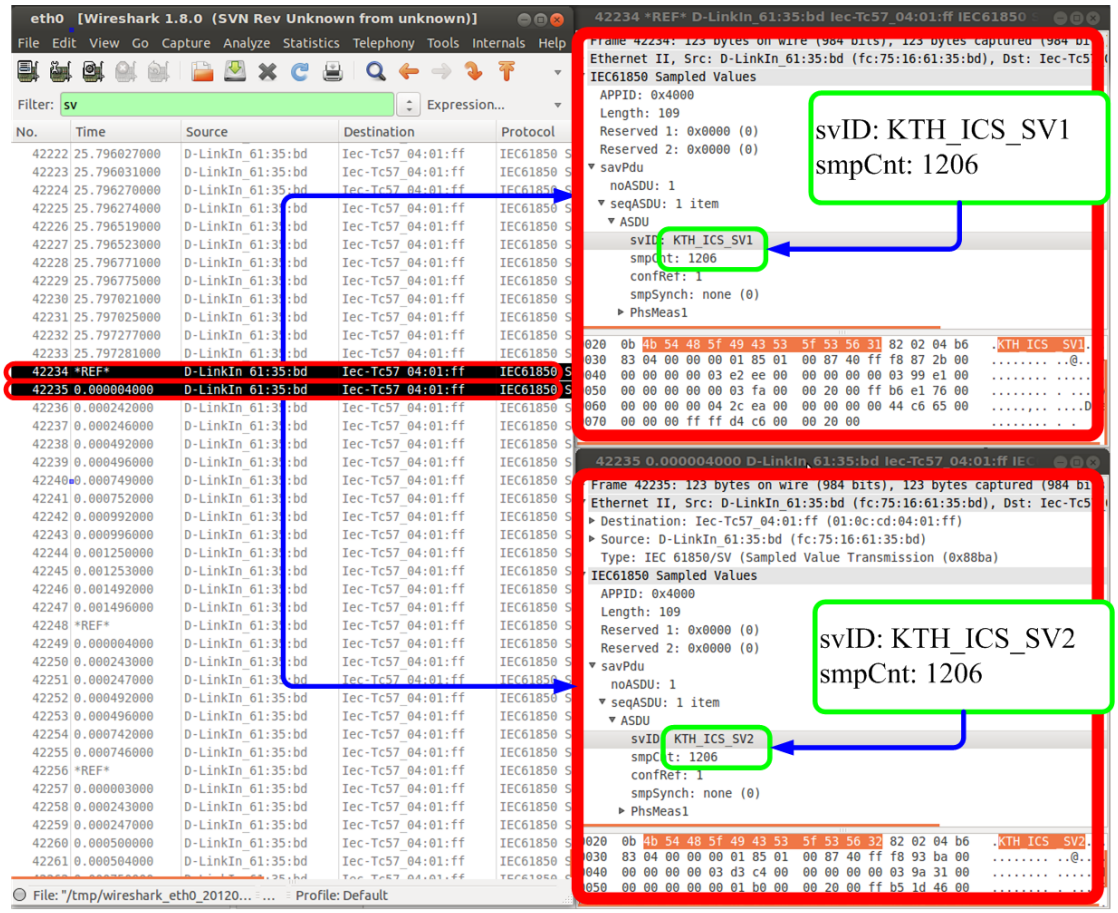


Figure B.8: The Wireshark capture of sending SVs for two MUs

3. To subscribe two measurement data sets, set up the "svID" for RET 670 to "KTH_ICS_SV1" and "KTH_ICS_SV2".

Parameter	Value
MU1_4I_4U_921	
SVId	KTH_ICS_SV1
MU2_4I_4U_921	
SVId	KTH_ICS_SV2

Table B.2: Parameter setting for MU

4. Connect the T2WPDIF function block in application configuration

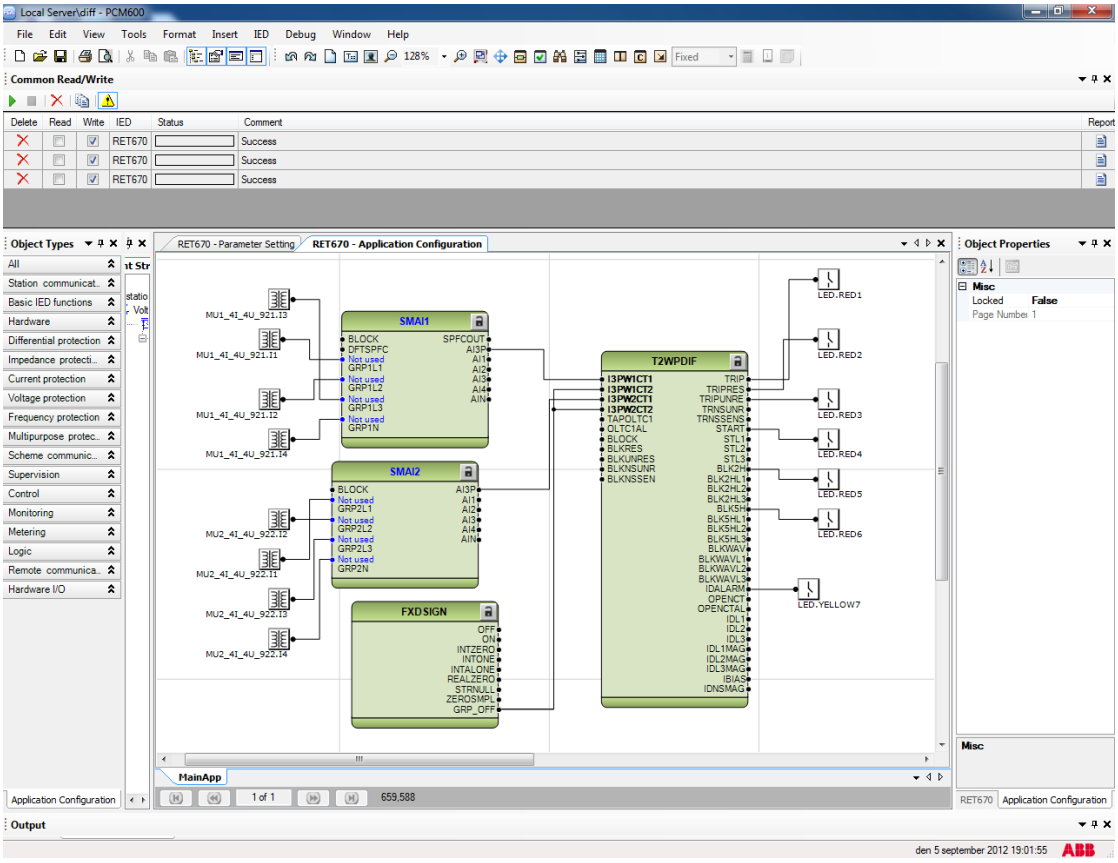


Figure B.9: Application configuration for T2WPDIF

The LED mapping for T2WPDIF is listed below in Table B.3.

Signals	LEDs	Signals	LEDs
General Trip	RED1	2^{nd} harmonic block	RED5
Trip from restrained function	RED2	5^{th} harmonic block	RED6
Trip from unrestrained function	RED3	Alarm from sustained diff current	YELLOW7
Common start	RED4		

Table B.3: The LED map for T2WPDIF

5. Parameter setting of the T2WPDIF

Parameters	Values	Parameters	Values
T2WPDIF:1			
RatedVoltageW1	100kV	ClockNumberW2	6[180deg]
RatedVoltageW2	100kV	ZSCurrSubtrW1	on
RatedCurrentW1	350A(RMS)	ZSCurrSubtrW2	on
RatedCurrentW2	350A(RMS)	TconfigForW1	no
ConnectionTypeW1	WYE(Y)	TconfigForW2	no
ConnectionTypeW2	WYE(Y)	LocationOLTC1	Not Used
Setting Group1			
Operation	On	tAlarmDelay	10s
SOFTMode	Off	IdMin	0.3 IB
IDiffAlarm	0.1 IB	IdUnre	10 IB

Table B.4: T2WPDIF settings

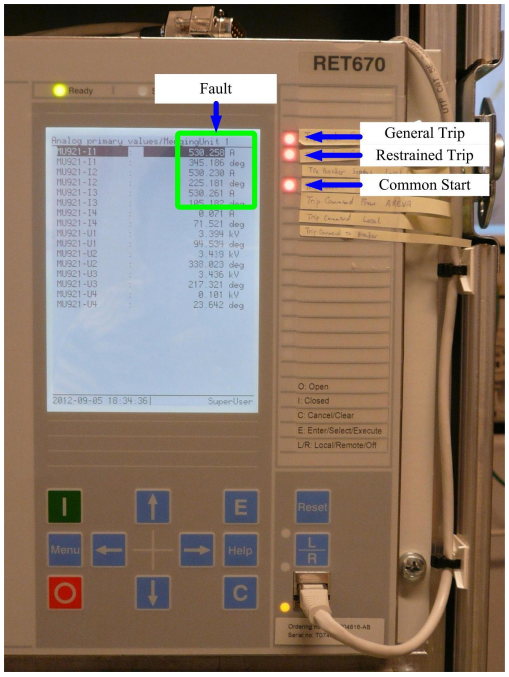
As shown in Table B.4, the rated current of winding 1 (W1) and winding 2 (W2) are set to the same value which indicates the ratio of the primary and secondary side is 1. The "IdMin" is the threshold current. When the current difference between primary and secondary sides exceeds "IdMin", the restrained protection will trip. In this case, the "IdMin" is 0.3*IB=0.3*350=105A(RMS). When the current difference exceeds "IdUnre" which is set to 10*IB, the unrestrained protection will trip.

Results

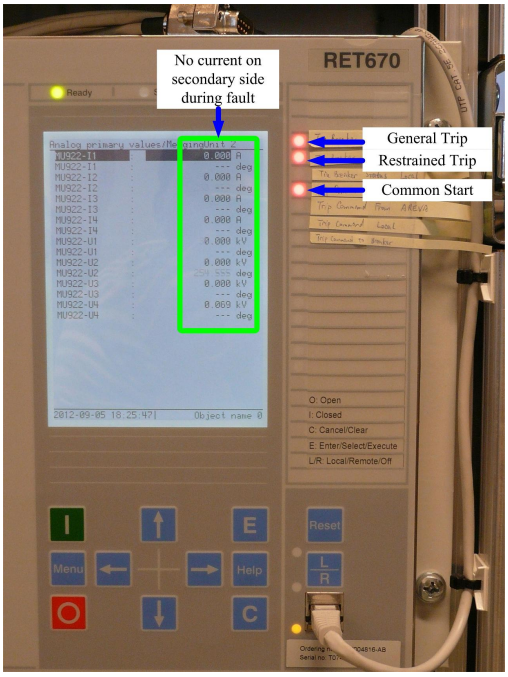
Normal Setting

Firstly, the setting as listed in Table B.4 is written to RET 670. Since the three-phase to ground fault happens in the primary side of the transformer, all the current comes into the primary side will go to ground during the fault. There will be no current on the secondary side during the fault. The current difference between primary and secondary sides is around 500A which already exceeds the threshold 105A. The RET 670 trips normally and the results are shown in Figure B.10 below.

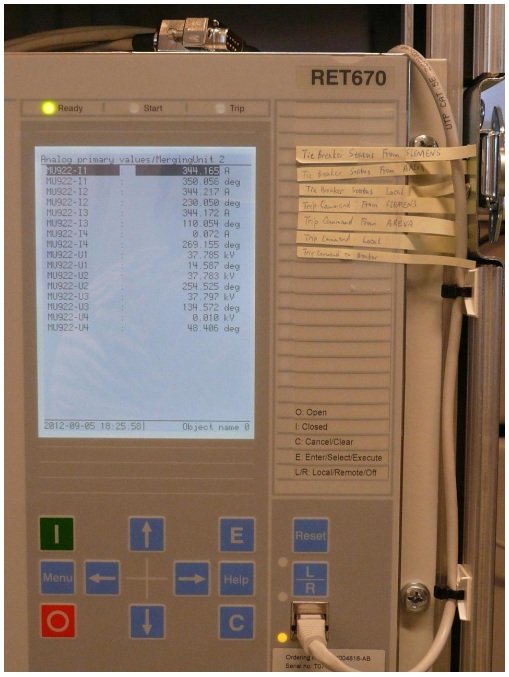
(a) The normal readings from MU1



(b) Readings from MU1 during trip



(c) Readings from MU2 during trip



(d) After fault readings from MU2

Figure B.10: The trips of T2WPDIF

High Threshold Setting

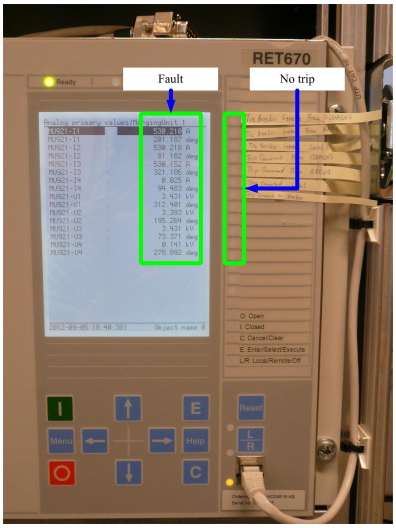
In this setting, most of the parameter settings are the same as that in Table B.4. The parameter changed are listed in Table B.5 below.

Parameters	Values
T2WPDIF:1	
RatedCurrentW1	1000A(RMS)
RatedCurrentW2	1000A(RMS)
Setting Group1	
IdMin	0.6 IB

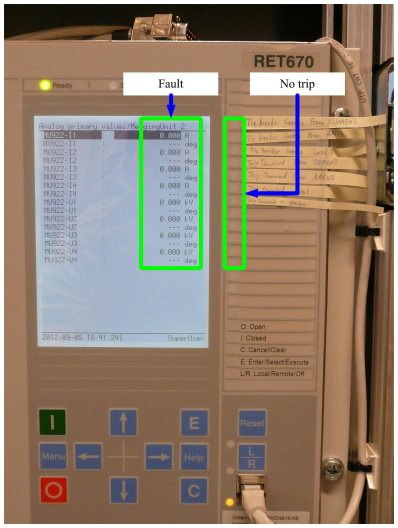
Table B.5: The changed parameters for T2WPDIF

It can be calculated, the threshold "IdMin" is 0.6*1000=600A in this case. Since the current difference during the fault is around 500A in the simulation, the T2WPDIF will not trip. The results are shown in Figure B.11.

(a) The normal readings from MU1



(b) Readings from MU1 during fault but no trip



(c) Readings from MU2 during fault but no trip

(d) After fault readings from MU2

Figure B.11: Results of a higher threshold current