Smart Meter and Smart Energy Management

TISANALUK MAKEE

KAMONNAT SIRINALIRAT

BACHELOR OF ENGINEERING PROGRAM IN SOFTWARE ENGINEER

INTERNATIONAL COLLEGE

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2013

Thesis – Academic Year 2013

B.Eng. in Software Engineering

International College

King Mongkut's Institute of Technology Ladkrabang

Title: Smart Meter and Smart Energy Management

Authors:

1. Mr. Tisanaluk Makee Student ID : 53090014
2. Ms. Kamonnat Sirinalinrat Student ID : 52090001

Approved for submission

.....

(Asst.Prof.Dr.Visit Hirankitti)

Advisor

INTERNATIONAL COLLEGE KMITL

Date August 28, 2014

Smart Meter and Smart Energy Management

Mr. Tisanaluk Makee 53090014

Ms. Kamonnat Sirinalinrat 52090001

Asst.Prof.Dr. Visit Hirankitti Advisor

Academic Year 2013

ABSTRACT

Smart meters are electronic meters employed in AMI (Advanced Metering Infrastructure). AMI is a part of smart grid, which is the future of power generation, distribution, and management. A smart meter periodically monitors household energy consumption and reports this information to a utility company. It can record historical data of interval energy consumption and provides two-way communication for the company to remotely control and manage the meter.

This project develops supporting software and an in-home display software to work with a smart meter. The supporting software is a software for setting a smart meter, reading and displaying meter data, as well as upgrading its firmware. The supporting software is developed using Python and PySide library running on a PC connects to a smart meter via an optical probe or a ZigBee module in order to communicate with a smart meter.

The in-home display is smart phone or tablet running an application for displaying measurement values read from a smart meter and graph presentation of historical measurement values of a smart meter. In-home display software is developed in Objective-C and runs on an iOS device.

As communication with a smart meter requires the DLMS/COSEM protocol, we have designed and developed a DLMS/COSEM message decoder/encoder to use by our supporting software to communicate with a smart meter.

Acknowledgements

We most sincerely thank our thesis adviser Asst.Prof.Dr. Visit Hirankitti. During the final year, his willingness to give us support, understanding, and encouragement is the most thankful. We also would like to express our gratitude to others professors and lecturers in the International College for their help with useful suggestions and information that guide us to do the project in the right direction.

Contents

List of Tables

Table **Page**

List of Figures

Chapter 1

Introduction

1.1 Introduction

Smart meters are electronic meters employed in AMI (Advanced Metering Infrastructure), which periodically monitor household energy consumption and report this information to a utility company. It can record historical data of interval energy consumption and provide two-way communication for the company to remotely control and manage the meter.

AMI is the metering infrastructure for the smart grid. It is an integrated system for distributing, metering and billing. A smart meter is a part of this system along with communication network and the meter data management system. It enables two way communication between the utility company and the meter. It also includes many functions for consumer side (demand side).

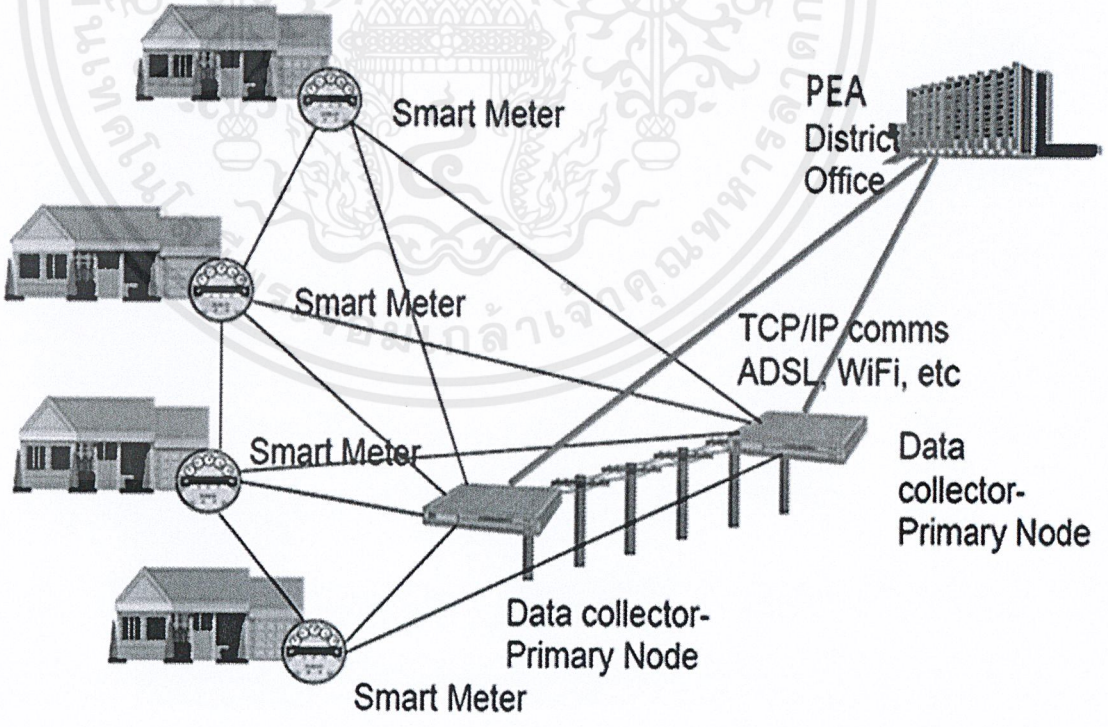


Figure 1.1 : Smart meters of an AMI system

In this system, smart meters are installed for monitoring customer's energy usage and send this information back to Data Concentrator Unit (DCU). Each DCU communicates and gathers data from approximately 100-500 meters in its coverage area using a communication protocol called "DLMS/COSEM". The DCU also communicates to the utility company to send back the gathered information.

The DCU can also control each smart meter using a command issued by the utility company such as remotely connect/disconnect a meter and upgrade a firmware of a meter.

Our project is a part of "Research and Development Smart Meter for AMI", a research project funded by PEA (Provincial Electricity Authority). The work we get involved is to develop a supporting software and an in-home display for smart meters.

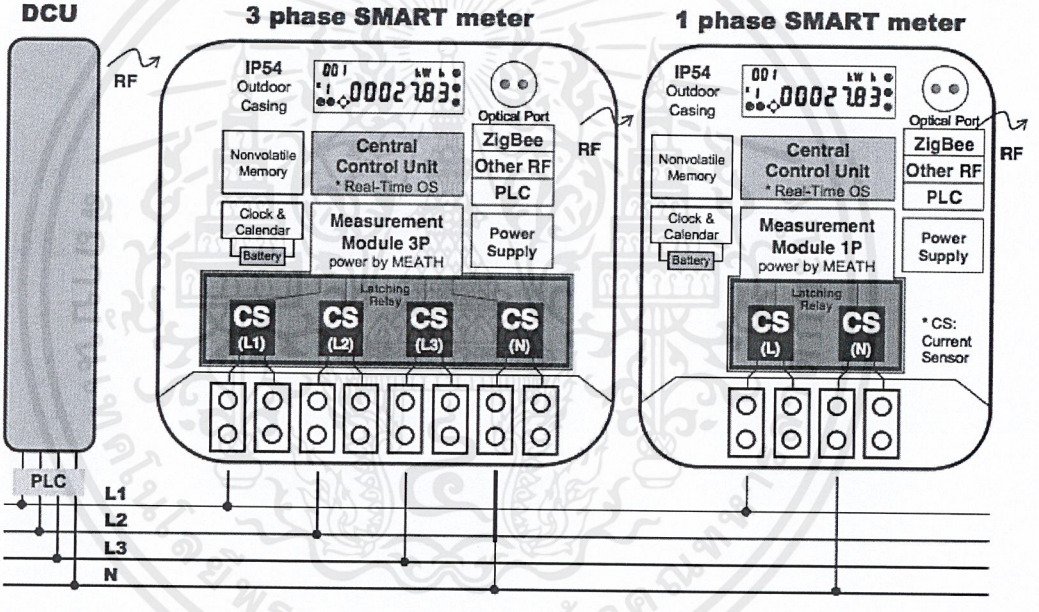


Figure 1.2 : Smart Meter of PEA

The supporting software is a software for setting a smart meter, reading and displaying meter data, as well as upgrading its firmware. The supporting software in this thesis is developed using Python and PySide library running on a PC connects to a smart meter via an optical probe or a ZigBee module in order to communicate with a smart meter. This software will be used by a PEA staff to install and manage a smart meter on the site.

In Home Display is smart phone or tablet running an application used by resident for smart energy management in order to reduce household energy consumption. The

software is running on a mobile device that will serve a user as a display of meter data in the house.

The DLMS/COSEM is a communication protocol used to communicate between smart meters and other devices of AMI. In order to communicate with a smart meter, our software need to use the DLMS/COSEM protocol too. The DLMS/COSEM is an ASN.1 base protocol designed for communication in metering equipment network. The protocol has a pattern and structure of the massages that need to be interpreted. So we need to develop an encoder/decoder to understand a DLMS/COSEM message.

For the in-home display, the software cannot communicate directly with a smart meter. So we use a Hi-Jack adapter [2] which uses an audio port to communicate with a PC running the supporting software which in turn connects to a smart meter. Since Hi-Jack uses sound to communicate on the audio port, we need to develop a sound modulation/demodulation module in order to retrieve and sending data with the Hi-Jack.

1.2 Motivation

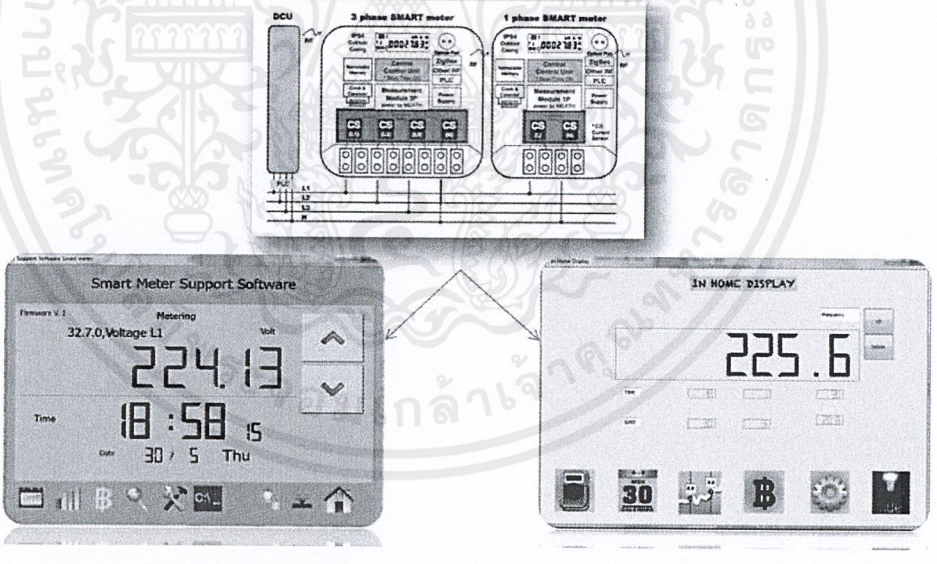


Figure 1.3 : Interface of Supporting Software and In Home Display

The supporting software is an essential tool for a PEA staff to install, set up and manage a smart meter. And also an in-home display it also essential for a resident to be able to observe his/her household electrical consumption.

1.3 Objectives

This thesis aims to develop a supporting software and in-home display for a smart meter. So our objectives are:

1. To develop a supporting software as an application on a PC with simple GUI interface.
2. To develop DLMS/COSEM encoder and decoder modules to use by the supporting software in order to communicate with a smart meter.
3. To develop an iOS in-home display application for an iPhone and iPad.
4. To develop a sound modulation/demodulation software for an in-home display application.

1.4 Scope of Project

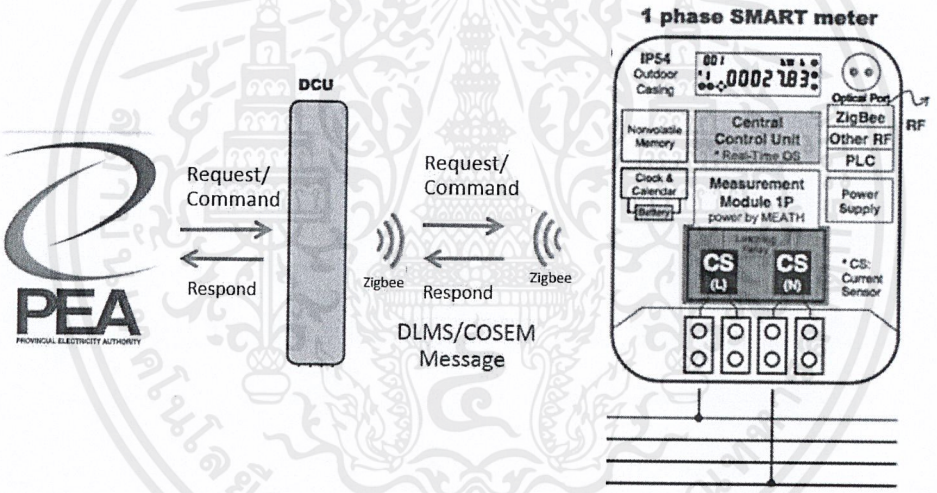


Figure 1.4 : Smart Meter Network

1.4.1 To develop supporting software to be able to:

1. Adjust and configuration on measurement value.
2. Setting and configuration on display and display value.
3. Remote connect/disconnect meter.
4. Communicate and retrieve data with meter via Zigbee, PLC, Optical Interface or RS-485
5. Real time display meter's display value.
6. Manual/Automatic data retrieval to generate reports.
7. Adjust or calibrate measurement part of meter

8. Upgrade firmware.
9. Generate reports for billing.

1.4.2 To develop encoder/decoder library for a DLMS/COSEM message that will be used by supporting software to communicate with smart meter using an interpreter technique. This library will be useful for another application that need to communicate with devices in smart meter network later on.

1.4.3 To develop the in-home display for electrical user for smart energy management in order to reduce household's energy consumption. The In Home Display develop in this project communicates with smart meters to analyze real-time energy consumption data and intelligently control electrical appliance to reduce household's energy consumption. This application will be develop using objective-C for using on IOS devices.

1.4.4 To develop a sound modulation/demodulation software module for audio signal that will be used by in-home display application to communicate with Hi-jack which using audio port and audio signal to transmit data.

1.5 Structure of project

Chapter 2 describes the comparison of existing software, related work and problem description.

Chapter 3 describes the technical background.

Chapter 4 describes the software requirement, and UML use case Diagram.

Chapter 5 describes the structure of software system that represent by system diagram.

Chapter 6 describes system development, which describes techniques, algorithms and tools used throughout the project.

Chapter 7 describes the result of the experiment.

Chapter 8 describes the evaluation and discussion of the results.

Chapter 9 shows the conclusion of the project and future works.

Chapter 2

Related Work

2.1 AMR

The software developed in this project is a part of an AMI system. Which is an integrated system for distributing, metering and billing. AMI is not the first automated meter reading system. The first automate meter reading system is AMR (Automated Meter Reading).

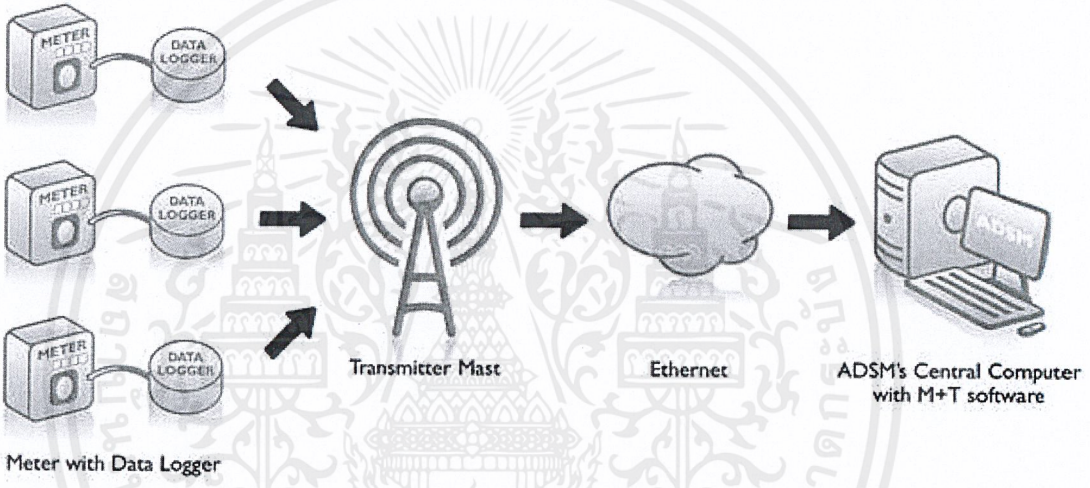


Figure 2.1 : AMR network

Before these automated systems have been invented, meter reading is done by human who observes each meter in a timely manner. This process is costly and error prone. Also this process cannot provide real-time meter reading which is important for demand-side management where customers want to know their current energy usage to be able to use the energy more efficiently.

AMR or Automated Meter Reading is an automated system for gathering energy consumption, diagnostic and status data from metering devices and send it to a utility company for billing, troubleshooting and analyzing of electricity distribution. In AMR the information gathered from metering devices, which is needed by the distributing network, needs will be kept periodically in a data logger and this information will then be send back to the utility company.

There are many ways to transmit the metering information back to the utility company such as, using short messages via a build-in mobile phone, via radio frequency network, or via PLC (Power Line Communication) from a substation to substation until the messages reach the utility company.

The AMR system is not invented for the utility company to control meter remotely. In order to change meter settings, the utility company needs to send its staffs to adjust a meter on the site. However, an AMI system has been invented not to only to collect meter information for a utility company, but the system also enable 2-way communication for remotely controlling each meter from the head quarter of the company. In addition, the AMI system also include demand-side management system which aims to reduce household energy consumption.

Next we look at an existing in-home display available in the market.

2.2 Wiser In-Home Display

In this project, one of our tasks is developing in-home display application for an iOS device, which can monitor household energy consumption for smart energy management.

At present, a number of in-home display solutions have been developed by a microchip electronic company, an electric appliance company, a metering equipment company, etc. Most products can display household consumption in real-time, control household electric appliances and store periodic data of energy consumption. The in-home display is connected to HAN (Home Area Network) which is created using ZigBee, Wi-Fi or PLC. Via HAN the in-home display can communicate household electricity appliances. One example of such an in-home display is Wiser In-Home Display Model EER20100.

Wiser In-Home Display Model EER20100 is a product of Schneider Electric company. This in-home display device is the central component of the wiser home energy management solution. It allows a home user to monitor and control household energy consumption though the informative displays. The display also shows other useful information, such as time and temperature. It is designed for residential and small business.

This device provides the following features for a home user:

1. Displaying energy consumption information, time and temperature.
2. Monitoring the temperature and thermostat

3. Adjusting the thermostat settings
4. Controlling thermostat's demand response
5. Controlling and monitoring load control devices and smart plugs
6. Setting the optional features of load control devices
7. Programming time schedule of turning on/off an electric appliance
8. Adjusting demand response
9. Setting TOU (Time of Use)
10. Setting the load voltage
11. Adding and setting electric appliances in home network

Figure 2.2 : Wiser In-Home Display Model EER20100

This device reads energy consumption from each electric appliance directly and from a smart plug but the company does not offer a smart meter in its solution.

2.3 XML to DLMS/COSEM Translator

In this project, we need to develop software to communicate with a smart meter using the DLMS/COSEM protocol. Since we develop our supporting software using Python and an in-home display software using Objective-C. There is not an encoder/decoder library in Python and Objective-C, so we need to develop our own DLMS/COSEM encoder/decoder library. Although there is not any encoder/decoder library available, we have found an XML translator for DLMS/COSEM protocol that is similar to our DLMS/COSEM encoder/decoder.

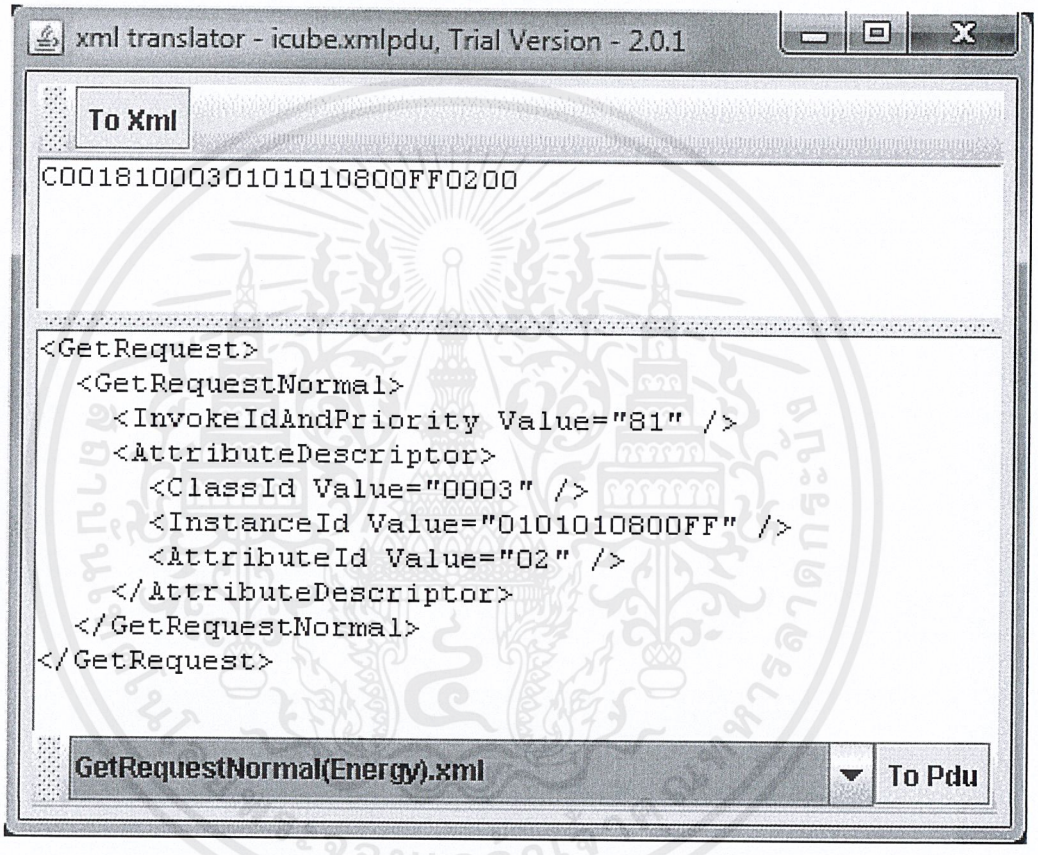


Figure 2.3 : XML to DLMS/COSEM Translator

From figure 2.3, it shows the xml translator of DLMS/COSEM protocol. This program is used to translate between DLMS/COSEM massages and XML messages. This freeware is developed by icube software (www.cyamon.com) using Java.

To use this translator is by typing a DLMS/COSEM massage into the top box and click "To Xml", the tool will translate the message to its XML form. On the other hand, if user types an XML form of a DLMS/COSEM message in the bottom box and click "To Pdu" the tool will translate the XML message to its corresponding DLMS/COSEM

message. This tool also provides basic examples of DLMS/COSEM massages for the user to modify and play with the tool.

Chapter 3

Background Knowledge

3.1 Smart Meter

Smart meters are electronic meters which can record historical data of interval energy consumption and provide two-way communication for the company to remotely control and manage the meter. The smart meter used for this thesis has the following features, it can:

- Measure electrical energy in order to display and record
- Display a real time metering data and meter status
- Detect anti-tampering
- Remote disconnect/reconnect the meter
- Automatically enroll itself into the network
- Have clock synchronization/ real time clock and calendar
- Retrieve data from and send data to a supporting software according to a schedule
- Provide secure data communication
- Upgrade firmware
- Diagnose internal problems
- Communicate with supporting software via an optical probe and ZigBee

The smart meter has 7 main components:

1. Central control unit is a microcontroller, which is a main processor inside the meter. Its function is to read measurement data from the measurement module, process the data, store the data into a memory, and control other components.
2. Measurement module is responsible for monitoring and measuring electricity usage and send this information to the central control unit.
3. An LCD Display is used by the meter to display the measurement values and other values that has been setting to be shown.
4. Non-volatile memory is used for keeping the meter firmware, data log and the other important information for smart meter.

5. Clock and calendar are used for keeping the current time and providing references of historical data inside the meter. They will be used for billing according to TOU. This part has a battery to backup to make sure the clock and calendar are still working even the meter is shut down.

6. Communication module is a part to communicate to the supporting software via an optical port and ZigBee.

7. Power supply is used to supply power to keep the meter operating.

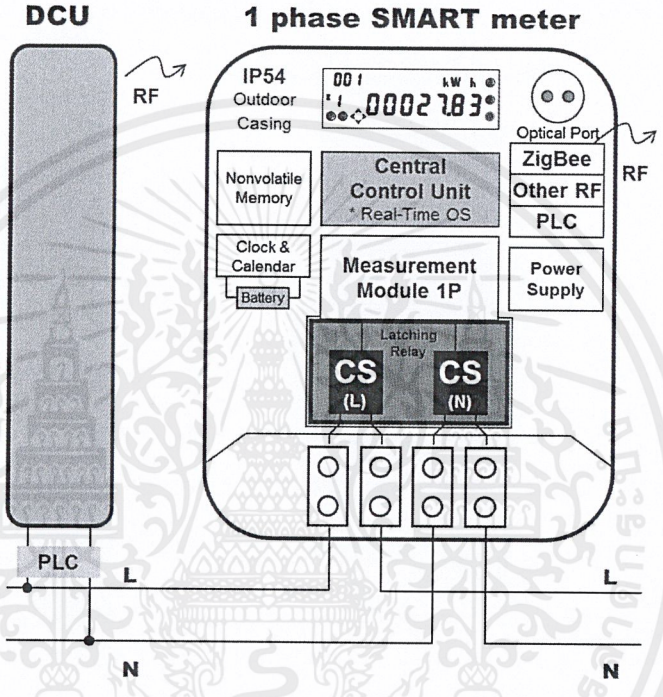


Figure 3.1 : Smart meter main components

3.2 Smart Meter Communication

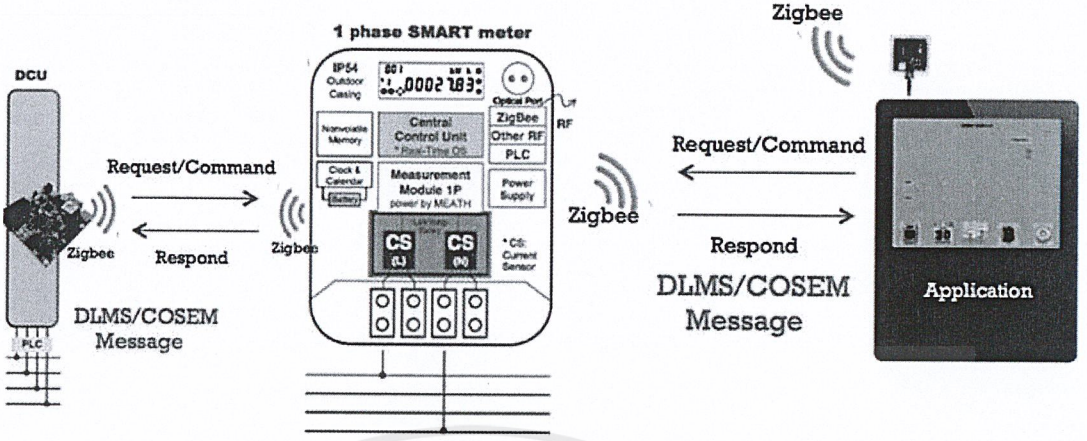


Figure 3.2 : Smart Meter communication

In the AMI system, the DCU gathers electricity usage information from smart meters and communicates with smart meters by sending a request massage in form of DLMS/COSEM massage via ZigBee to the smart meter. When the smart meter receives that DLMS/COSEM massage, the CCU inside smart meter interprets the message, execute a command inside a message, and send back a respond message to the DCU. So any application which communicates with the smart meter needs to do so using DLMS/COSEM messages.

3.3 DLMS/COSEM

DLMS/COSEM come from 2 words:

- DLMS: "Device Language Message Specification" – a generalized concept for abstract modelling of communication entities
- COSEM: "Companion Specification for Energy Metering" – sets the rules, based on existing standards, for data exchange with electronic meters

The DLMS/COSEM is an application layer protocol for communicating between a smart meter and a DCU/supporting software. It is a client-server type of communication, where the communication profiles assume a smart meters as a server and a DCU/supporting software as a client. The DLMS/COSEM protocol is designed based on the ASN.1 standard. The DLMS/COSEM protocol also includes:

1. An object model as the interface message model for viewing the functionality of the meter. For this part DLMS/COSEM provides only an abstract model, so a

developer need to develop an application to serve as the concrete object model of the DLMS/COSEM protocol.

2. An identification system as a data object represented by an OBIS code, which is a code to identify every metering data such as volt, current, watt, etc.
3. A message method for communicating between the meter and other devices. It provides a syntax pattern in a form of a pseudo-code parse tree to use for translating an object model into a series of bytes.
4. A transporting method, since the DLMS/COSEM protocol is an application layer protocol that cannot communicate without the support of the lower layer protocols, it needs a communication profile which specifies suitable setting of lower supporting layer protocols.

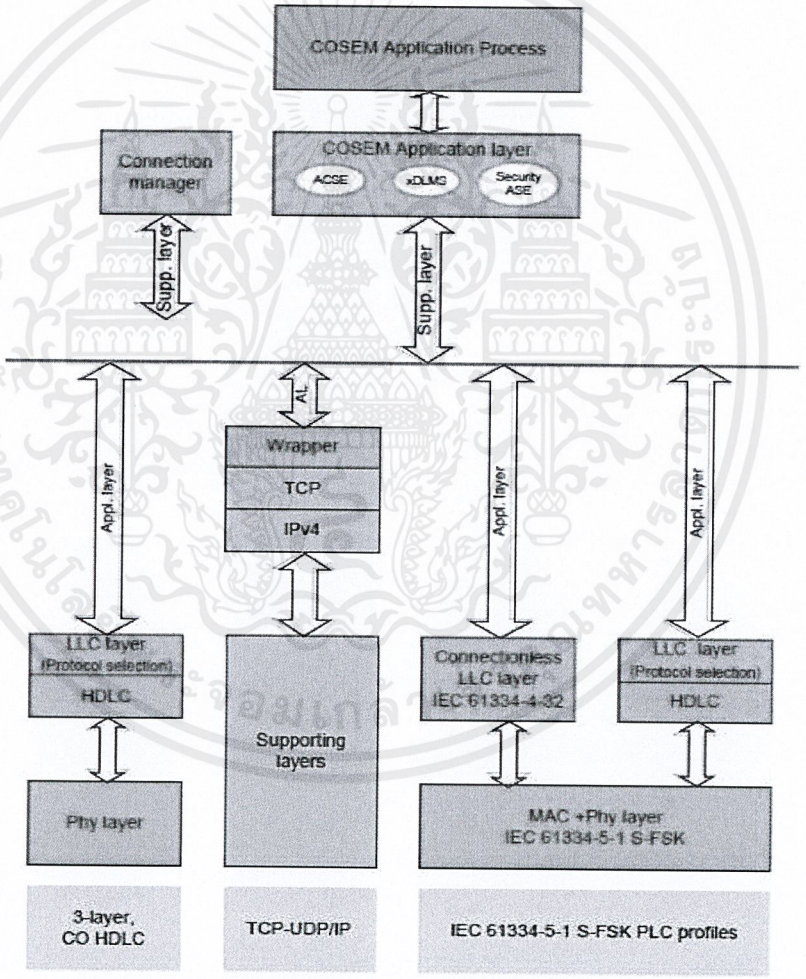


Figure 3.3 : DLMS/COSEM communication profiles

Since PEA has decided to use the DLMS/COSEM protocol as the communication protocol for the smart meter. Our developer team of the smart meter project needs to

develop a DLMS/COSEM encoder/decoder to use in a firmware and software of the smart meter so that the firmware and software can understand a DLMS/COSEM message.

3.4 ZigBee

ZigBee is a wireless communication technology which consumes low-energy with low-cost. Being a low-cost technology, ZigBee is popular to use for wireless control and a telemetry application. As a low-power energy consumption technology, a ZigBee module requires a small battery to operate for a long duration. A ZigBee device can transmit data in a long distant by passing data thought a mesh network of other ZigBee devices.

Figure 3.4 : ZigBee module

The ZigBee communication protocol is define as IEEE 802.15.4 standard. It operates in several radio bands, such as 868 MHz in Europe, 915 MHz in USA and Australia, but the most widely used radio band is 2.4 GHz. ZigBee network layer can support a mesh, star and tree network.

ZigBee devices are categorized into three types:

- A coordinator, which is the most capable one. It forms the root of a network tree and sometime it can bridge to another network. There is

exactly one coordinator in each network and it stores the network information.

- A router can be a destination node where the data is sent to or an immediate node of the network which passes data from one node to another node. It can also send its data to near-by node by itself.
- An end device is a destination node where the data is sent to from its parent (Coordinator or Router). It consumes much lower energy than the other two types.

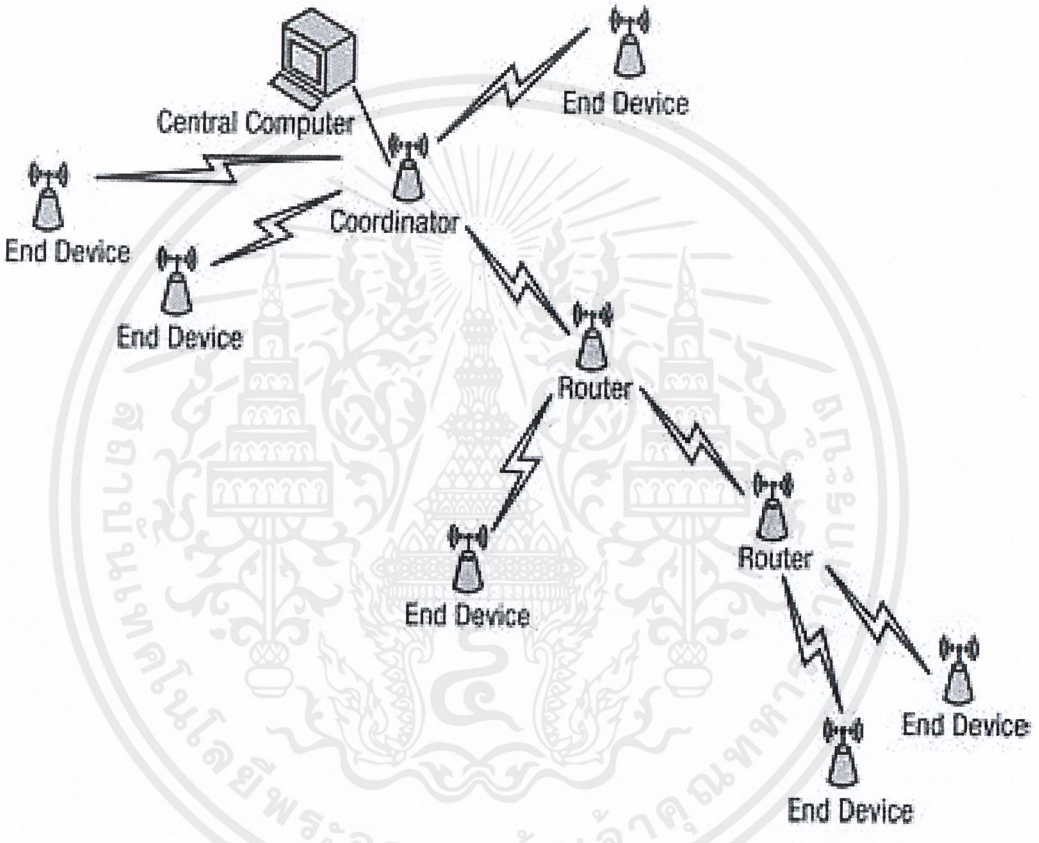


Figure 3.5 : Mesh network

In an AMI network, ZigBee is used as a low-cost wireless mesh network for communication between the smart meter and a DCU.

3.5 HDLC

High-Level Data Link Control (HDLC) is a communication protocol on the link layer. The protocol provides either best effort or reliable communication path between nodes depend on the HDLC mode being used. HDLC is a bit-oriented protocol, which is suitable for carrying a DLMS/COSEM message which consists of a series of bytes. So the

HDLC protocol is one of protocol chosen to be used as a link layer protocol of the AMI network which connects between smart meters and a DCU.

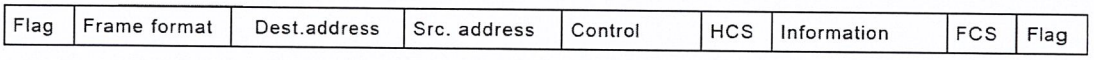


Figure 3.6: HDLC frame format type 3

In an AMI network, the HDLC frame format type 3 is chosen to be used for the MAC sub-layer as defined in Annex H.4 of ISO/IEC 13239. The elements of an HDLC frame are:

1. Flag field has a byte long with the value of 7Eh. For sending several frames continuously, each frame will start and close with this field.

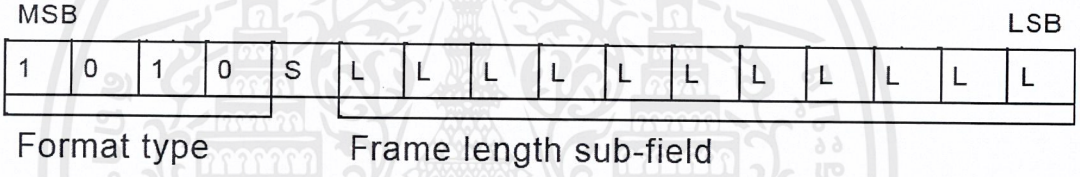


Figure 3.7 : The flame format field

2. Frame format field has 2 bytes long and contains three sub-fields which are:
   - Format type subfield (4 bits) used for showing a type of HDLC frame format
   - Segmentation (1 bit) indicating whether this flame is the last flame of this chuck of data or not
   - Frame length (11 bits) indicating the size of this frame
3. Destination and source address fields, as its name suggests, these fields are used for storing the addresses of the destination and the source. These fields size can be varied from 1 byte to 4 bytes. At the last digit of the last byte of each field will contains 1.

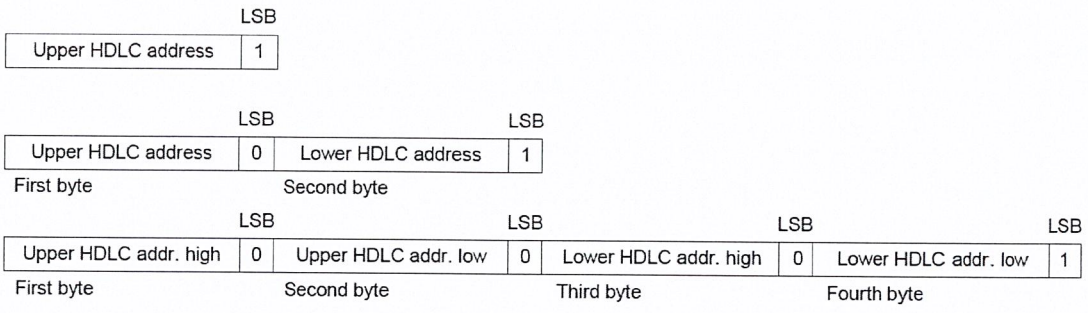


Figure 3.8 : The valid address structures

4. Control field has one byte long, it indicates the type of command or response; it also contains binary codes which indicates the type of command or response on the left. The type of command and response can be:
   - I : Information frame
   - RR : Receive ready frame
   - RNR : Receive not ready frame
   - SNRM : Set normal response mode frame
   - DISC : Disconnect frame
   - UA : Unnumbered acknowledge frame
   - DM : Disconnected mode frame
   - FRMR : Frame reject frame
   - UI : Unnumbered information frame

		MSB LSB
Command	Response	
I	I	R R R P/F S S S 0
RR	RR	R R R P/F 0 0 0 1
RNR	RNR	R R R P/F 0 1 0 1
SNRM		1 0 0 P 0 0 1 1
DISC		0 1 0 P 0 0 1 1
	UA	0 1 1 F 0 0 1 1
	DM	0 0 0 F 1 1 1 1
	FRMR	1 0 0 F 0 1 1 1
UI	UI	0 0 0 P/F 0 0 1 1

Figure 3.9 : Control field bits assignments

5. Header check sequence (HCS) field is two bytes long, it is generated from the header field, which consists of frame format, destination address, source address and control field of HDLC frame, in order to check the header for its correctness.
6. Information field is where carried data from upper layer will be placed here. If this frame is empty that will make the frame check sequence empty as well.
7. Frame check sequence (FSC) field, like HCS field, is two bytes long, but it is a sequence of codes for checking for the correctness of the whole data frame.

3.6 Sound Modulation

To develop an in-home display application, which can communicate with the Hi-jack adapter, we need to develop a program to encoder/decoder data using sound.

The technique we use to transmit data between our in-home display application and the Hi-jack adapter is FSK sound modulation technique. Next we would like to explain in detail of this technique.

3.6.1 Understanding the FSK Modulation techniques.

FSK or *Frequency shift keying* is one of many techniques that is used for sound modulation in order to transmit digital signal on an analogue transmission medium. The frequency of a Sine wave carrier is shifted up or down to represent either a single binary

value or a specific bit pattern. The FSK modulation involves alternating the value of a delta frequency from a carrier frequency according to the value of the bit to be represented.

According to the Hi-jack adapter the two frequencies to use for sound modulation are:

7,350 Hz representing a bit value of 1 and

4,900 Hz representing a bit value of 0.

The bandwidth of the wave for modulation is 1,225 Bits per second and the sample rate of the wave modulation is 29,400 samples per second, although the normal sample rate of the sound wave is 44,100 samples per second. Due to the standard of the hardware used by the Hi-jack adapter, the sample rate of the wave modulation we use is 29,400 samples per second.

According to the standard of frequency, bandwidth and the sample rate of the wave for modulation. 1 bit can represent 29,400/1,225 = 24 samples.

3.6.2 Applying BFSK modulation and demodulation

As we use a binary data to send between the in-home display and the Hi-jack adapter, the FSK technique we use is BFSK (binary frequency shift keying), which is one of the simplest FSK modulation techniques. The binary value "1" it represented by the high frequency and the binary "0" is represented by the low frequency. Therefore, the input of the modulation is a bit string of 0s and 1s and the output is the waveform shown below. On the contrary, the input of demodulation is a waveform and its output is a bit string of 0s and 1s.

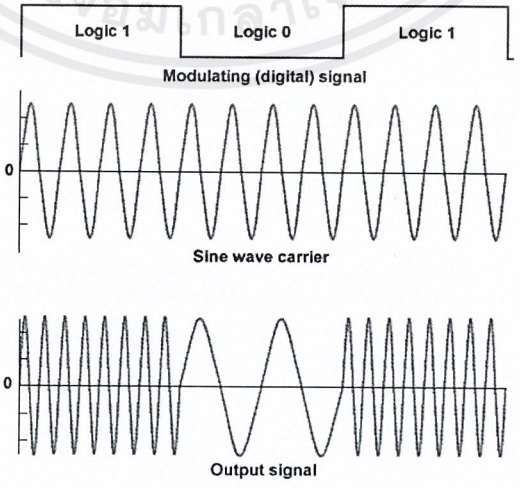


Figure 3.10 : Typical FSK Modulation output

Chapter 4

System requirements

4.1 Problem description

This project intends to develop supporting software and in-home display software to communicate with Smart Meter. Since our smart meter uses DLMS/COSEM protocol as its communication protocol so the supporting software need to support this protocol as well.

The supporting software is used by an engineer of a utility company for installing, maintaining and setting a smart meter. In this thesis our supporting software is developed using Python and Pyside GUI library running on PC and the software communicates to a smart meter via ZigBee communication and an optical probe interface.

In-home display is used by a home user for monitoring household energy consumption. This application is developed using Objective-C and it is run on an iOS device and communicates with a smart meter via ZigBee using the Hi-Jack adapter.

4.2 Requirement

4.2.1 Supporting Software requirements

Functional requirement:

1. The software can display measurement values read from a smart meter
2. The software has graph presentation for real-time values and historical data
3. The software can adjust and set meter parameters and configuration
4. The software can communicate with a smart meter via ZigBee communication and an optical probe interface
5. The software can manually and automatically collect meter data from a smart meter to create a report
6. The software can create a report for billing
7. The software can export meter data in an excel format file to use by a database software
8. The software can deliver firmware upgrade service on a smart meter
9. The software can remotely connect/disconnect a smart meter

10. The software has the database for keeping the information in the data log part.

Non-functional requirement:

1. The software is implemented using Python 2.7 and Pyside GUI library
2. The software has a user-friendly graphic user interface.
3. The software can run on windows XP/7/8 OS.
4. The software has secure communication with a smart meter
5. The software has 3 different levels of secured access to a smart meter for reading, reading & writing, and reading writing & modifying accesses
6. The software can support the DLMS/COSEM protocol.

4.2.2 In-home display requirements

Functional requirement:

1. The software can display measurement values read from a smart meter
2. The software has graph presentation for real-time values and historical data
3. The software can store historical data for presentation later
4. The software supports sound modulation/demodulation interface for the Hi-jack adapter
5. The software can communicate with a smart meter via ZigBee using the Hi-jack adapter

Non-functional requirement:

1. The software has user-friendly Graphic user interface.
2. The software is implemented using Objective-C and can run on an iOS device.
3. The software communicate with the Hi-jack adapter via the audio port with 2400 baud rate

4.3 Use Case Diagram

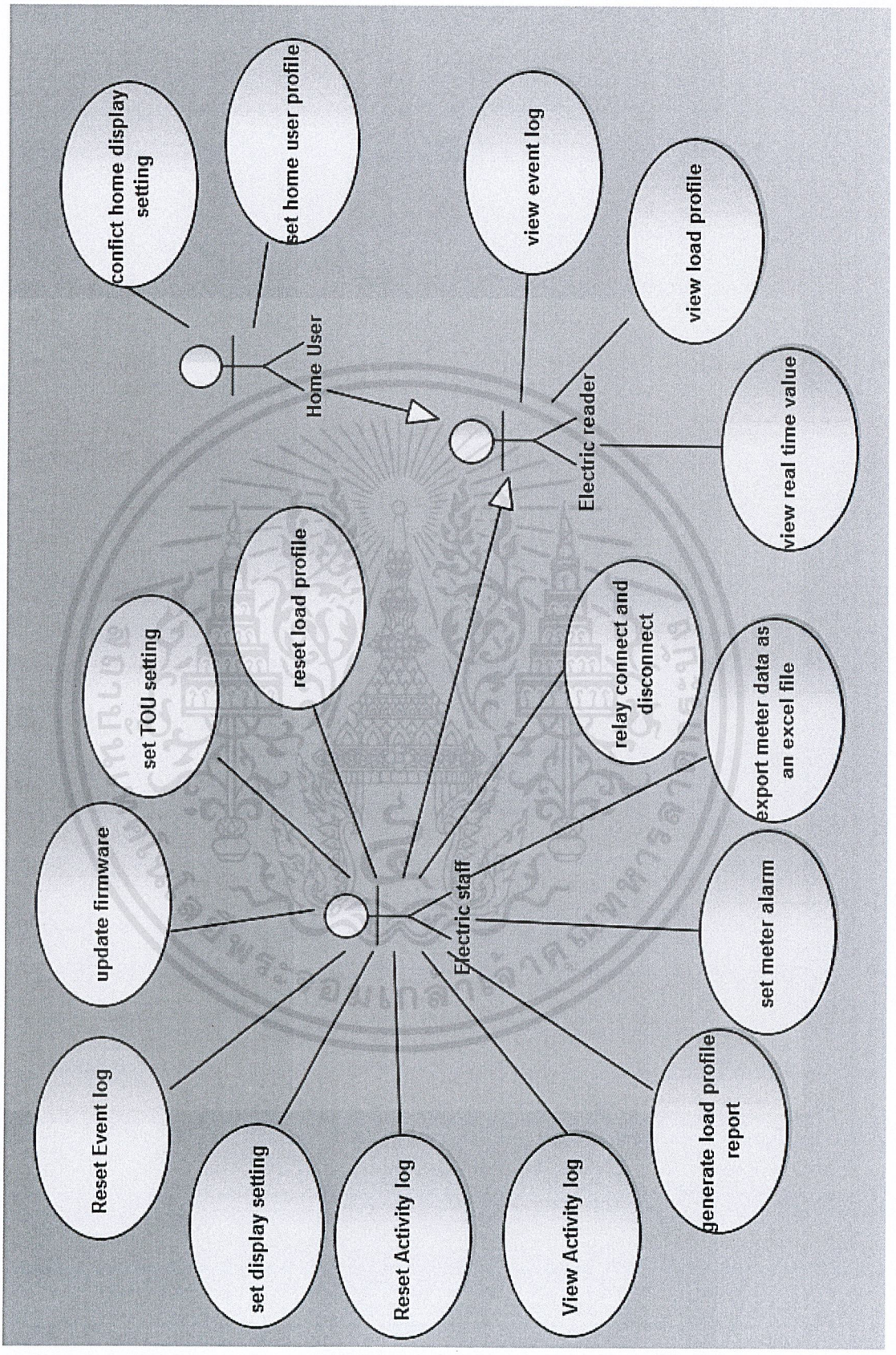


Figure 4.1 : Use Case Diagram of entire Smart Meter Project

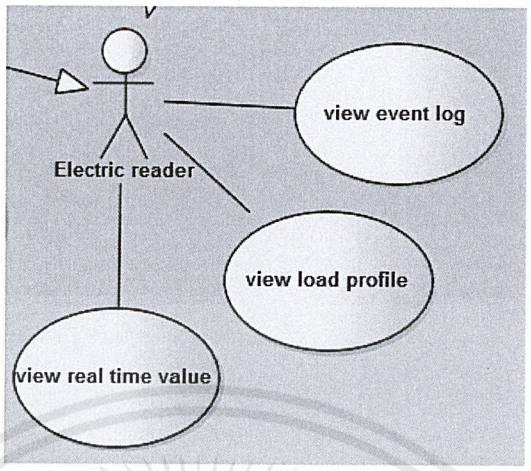


Figure 4.2 : Use Case of Electric Reader

Use case	View event log
Actor	Electric reader
Purpose	For view the event log information
Pre-condition	Log In
Post-condition	View event log
Process	1. Actor click to event log page 2. Actor view the event log information

Table 4-1 : Use Case of view event log

Use case	View billing load profile
Actor	Electric reader
Purpose	For view the load profile information
Pre-condition	Log In
Post-condition	View event log
Process	1. Actor click to event log page 2. Actor view the event log information

Table 4-2 : Use Case of view billing load profile

Use case	View real time value
Actor	Electric reader
Purpose	For view the electrical usage value
Pre-condition	Log In
Post-condition	View electrical usage value
Process	1. Actor click to view real time value page 2. Actor view the electrical usage value

Table 4-3 : Use Case of view real time value

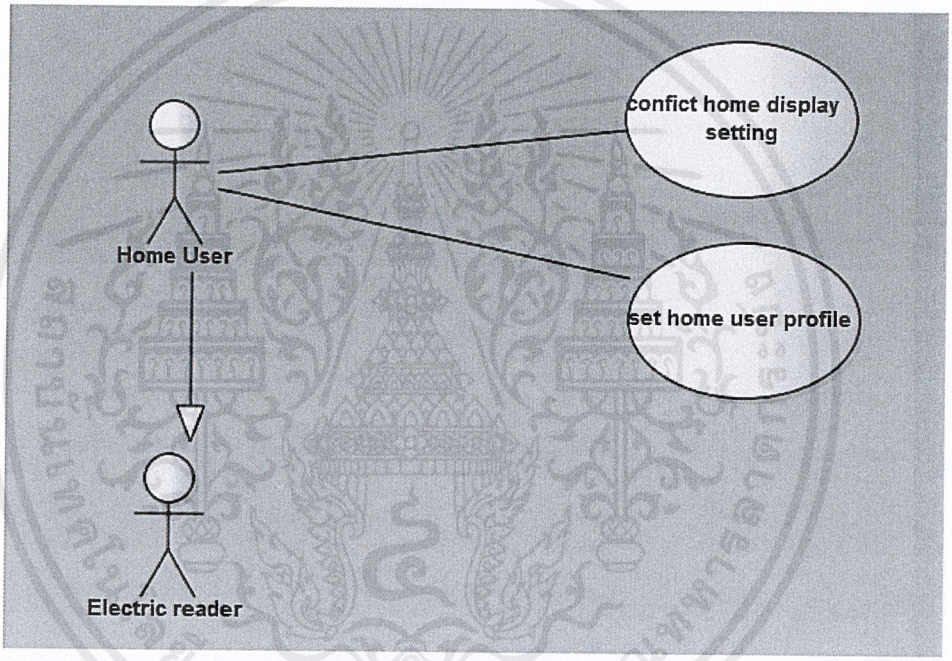


Figure 4.3 : Use Case of Home User

Use case	Configure home display setting
Actor	Home user
Purpose	For configure the setting at home display part
Pre-condition	Log In
Post-condition	Configure setting
Process	1. Actor click to configure home display setting button 2. Actor configures home display setting.

Table 4-4 : Use Case of Configure home display setting

Use case	Set home user profile
Actor	Home user
Purpose	For set Home User's Profile Information
Pre-condition	Log In
Post-condition	Set Home User's Profile Information
Process	1. Actor click to set Home User's Profile button 2. Actor set Home User's Profile Information

Table 4-5 : Use Case of Set home user profile

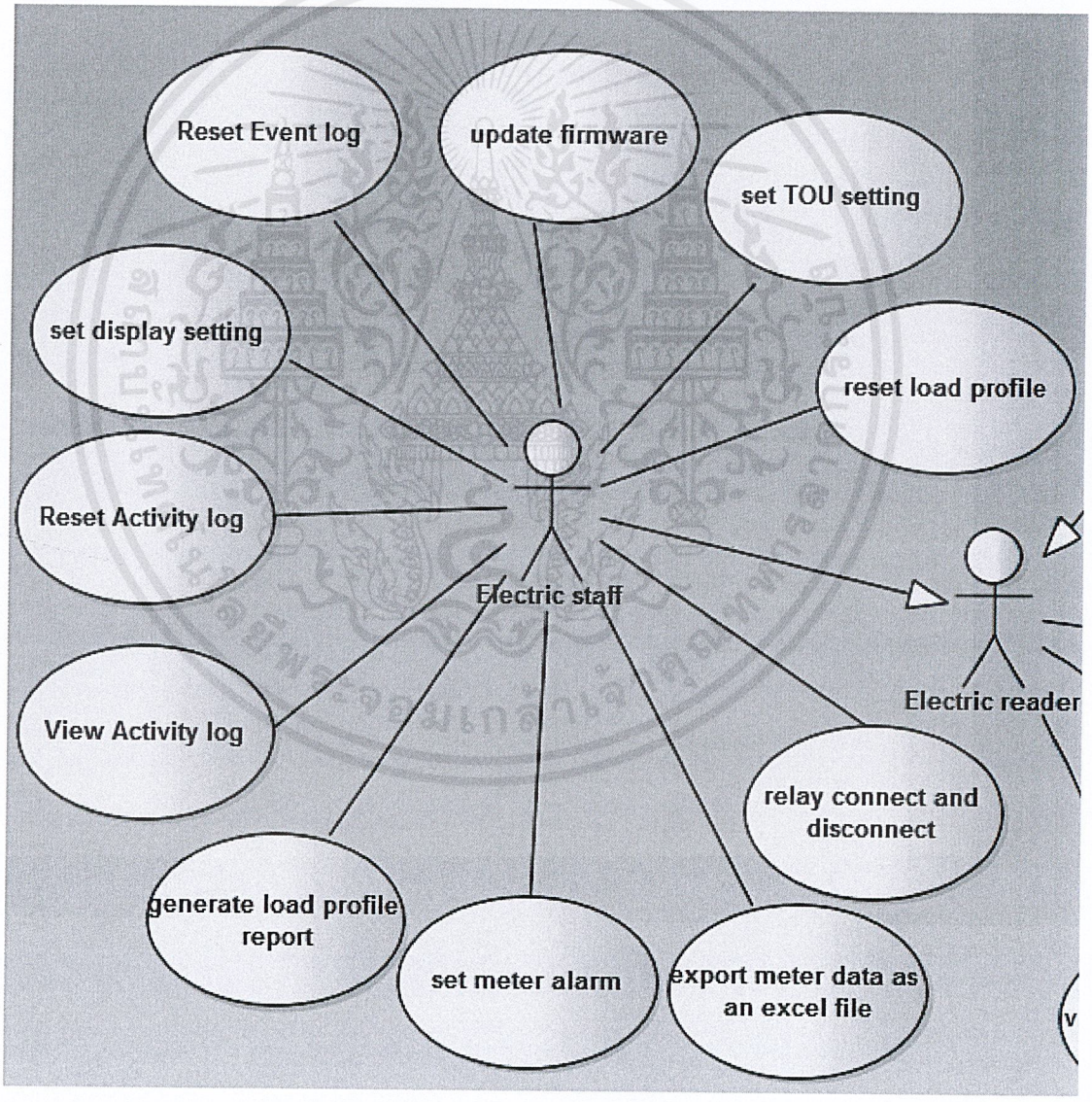


Figure 4.4 : Use Case of Electric Staff

Use case	Reset Load profile
Actor	Electric Staff
Purpose	For reset the Load profile
Pre-condition	Log In
Post-condition	Reset the data Load profile.
Process	1. Actor click to For reset the data at Load profile page 2. Actor reset the data at Load profile.

Table 4-6 : Use Case of reset load profile

Use case	Set TOU setting
Actor	Electric Staff
Purpose	For set the TOU setting
Pre-condition	Log In
Post-condition	Set the TOU setting
Process	1. Actor click to set the TOU setting page 2. Actor set the TOU setting

Table 4-7 : Use Case of set TOU setting

Use case	Update Firmware
Actor	Electric Staff
Purpose	For update the firmware
Pre-condition	Log In
Post-condition	Update the firmware
Process	1. Actor click to update firmware page 2. Actor Update the firmware

Table 4-8 : Use Case of Update Firmware

Use case	Set Display setting
Actor	Electric Staff
Purpose	For set the display setting
Pre-condition	Log In
Post-condition	Set the display setting
Process	1. Actor click to set the display setting page 2. Actor set the display setting.

Table 4-9 : Use Case of Set Display setting

Use case	Reset Event Log
Actor	Electric Staff
Purpose	For reset the data at event log
Pre-condition	Log In
Post-condition	Reset the data at event log.
Process	1. Actor click to For reset the data at event log page 2. Actor reset the data at event log.

Table 4-10 : Use Case of Reset Event Log

Use case	View Activity log
Actor	Electric Staff
Purpose	For view the Activity log information
Pre-condition	Log In
Post-condition	View Activity log
Process	1. Actor click to Activity log page 2. Actor view the Activity log information

Table 4-11 : Use Case of View Activity Log

Use case	Reset Activity Log
Actor	Electric Staff
Purpose	For reset the data at Activity log
Pre-condition	Log In
Post-condition	Reset the data at Activity log.
Process	1. Actor click to For reset the data at Activity log page 2. Actor reset the data at Activity log.

Table 4-12 : Use Case of Reset Activity Log

Use case	Generate load profile report
Actor	Electric Staff
Purpose	For generate load profile report from meter data for billing
Pre-condition	Log In
Post-condition	Load profile report
Process	1. Actor click to report page 2. Actor select date start and end of report 3. Actor click generate report 4. Actor select location to save report and save

Table 4-13 : Use Case of Generate load profile report

Use case	Export meter data as an excel file
Actor	Electric Staff
Purpose	For excel file from meter data for export to another database
Pre-condition	Log In
Post-condition	Excel file
Process	1. Actor click to report page 2. Actor select date start and end of report 3. Actor click generate excel 4. Actor select location to save file and save

Table 4-14 : Use Case of Export meter data as an excel file

Use case	Set meter alarm
Actor	Electric Staff
Purpose	For setting meter alarm notation
Pre-condition	Log In
Post-condition	Alarm notation
Process	1. Actor click to setting page 2. Actor select alarm tab 3. Actor setting alarm and save

Table 4-15 : Use Case of Set meter Alarm

Use case	Relay connect and disconnect
Actor	Electric Staff
Purpose	For setting Relay connection
Pre-condition	Log In
Post-condition	Relay connection
Process	1. Actor click to setting page 2. Actor select Relay tab 3. Actor click to connect or disconnect meter relay

Table 4-16 : Use Case of Relay connect and disconnect

Chapter 5

System Design

5.1 System Architecture

According to the system architecture of our supporting software and in-home display, the main components of a smart meter, which are related to our supporting software and in-home display, are:

- Communication module, which receives and sends messages between the meter and our supporting software/in-home display
- DLMS/COSEM encoder/decoder, which translates (decodes) a DLMS/COSEM message obtained from the communication module into an understandable format to be process by the central control unit. Also it translates (encodes) the message from the central control unit into a DLMS/COSEM message to communicate with an outside world.
- The central control unit, which preforms data processing and controlling the meter.

5.1.1 Supporting Software Architecture

Since supporting software is developed for installing, maintaining and setting a smart meter. So its architecture require these components:

- Control Unit, which is the main component for controlling and commanding another component. It also handles every event and interrupt from other components.
- GUI, which is the component to show the output of software and interact with the user.
- Database is used to store the current and historical information read from a meter such as a set of current meter data, a data log, an event log, etc.
- Upgrade firmware manager, as its name suggests, takes responsibility to upgrade new firmware into a meter
- DLMS/COSEM encoder/decoder. For the encoder, it takes a message from the control unit or the upgrade firmware manager and encodes it to a corresponding DLMS/COSEM message and sends the encoded message to the communication module. For the decoder, it decodes a DLMS/COSEM

message receive from the communication module and translates it into corresponding Python codes ready to execute.

- Communication Module, which is a serial communication module implemented using Pyserial library to exchange information between supporting software and a communication board. The module can communicate with a ZigBee board and an optical serial interface.

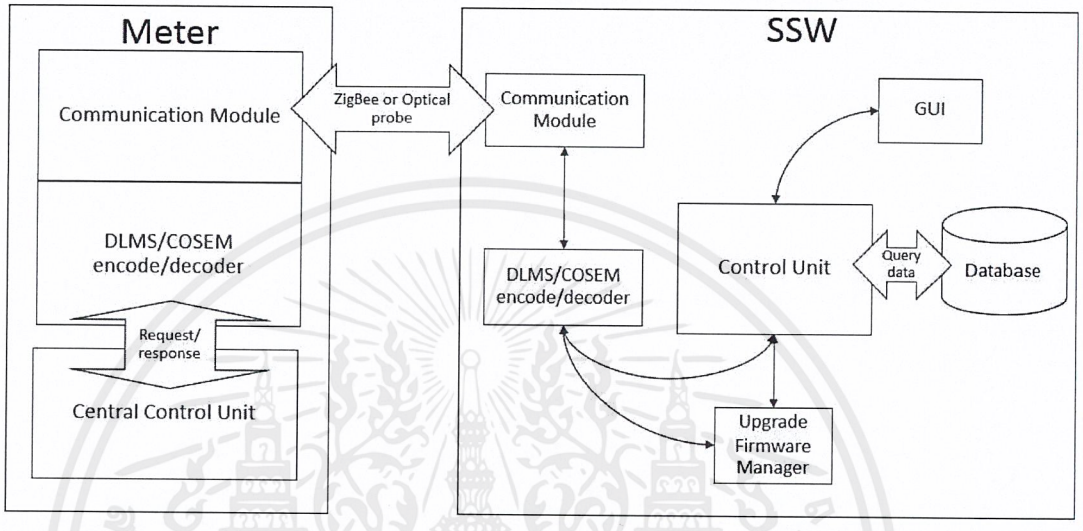


Figure 5.1 : Support Software Architecture

Supporting software has a control unit as a main component and main thread of software is to control all processes and collaborate with other components. It also invokes other components to process a task or to stop a processing task. GUI component is another thread that is running all the time to display the output of the software and interact with the user. For the other component, it will function when being invoked.

5.1.2 In-home display Architecture

In-home display software has a similar display functionality compared with supporting software. Its function is to monitor household energy consumption. So we design this software to have these components:

1. Control Unit, the main component for control and command another component. It also handling every event and interrupt from another components.
2. Display, the component to show the current status of software and interact with the user.
3. Database to collect to information from meter such as Data log, Event log, etc. This data will be keep for billing or displaying.

4. Reading data, this component is in charge of create reading message for request an information from the meter and send the receive data to central unit for displaying or keeping in database
5. DLMS/COSEM encoder/decoder, this component will take any message from command component (Reading data, Setting meter parameter and Update Firmware) and encode to COSEM message send to communication module and decode COSEM massage into a message then send back to correspond component
6. Sound Modulation is component for encode/decode between data and sound code and send to hi-jack via audio port

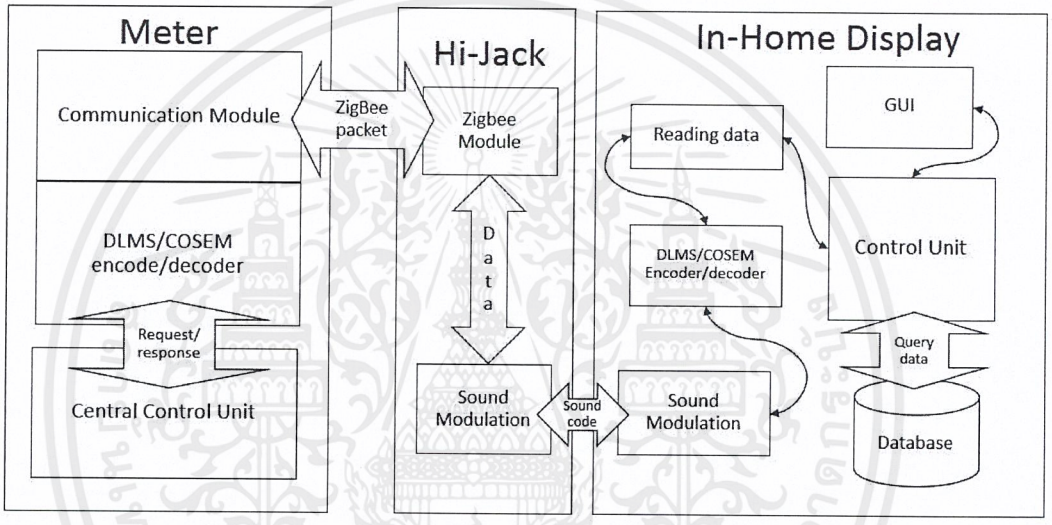


Figure 5.2 : In-home display Architecture

Most of components inside in-home display are similar to supporting software has control unit to control all activity of component inside application, display to show the status of application, and Encoder/Decoder for interpret between COSEM massage and massage of application. But it has only reading data as request component and instant of communication module, it has sound modulation for modulate between massage and sound code that will be send to Hi-Jack.

In Hi-Jack sound code will be demodulate to massage and pack that massage into ZigBee packet to send to meter. In another hand, ZigBee packet that receive for meter will be unpack to get massage and modulate into sound code to send back to In-home display.

5.2 Class Diagram

5.2.1 Supporting Software

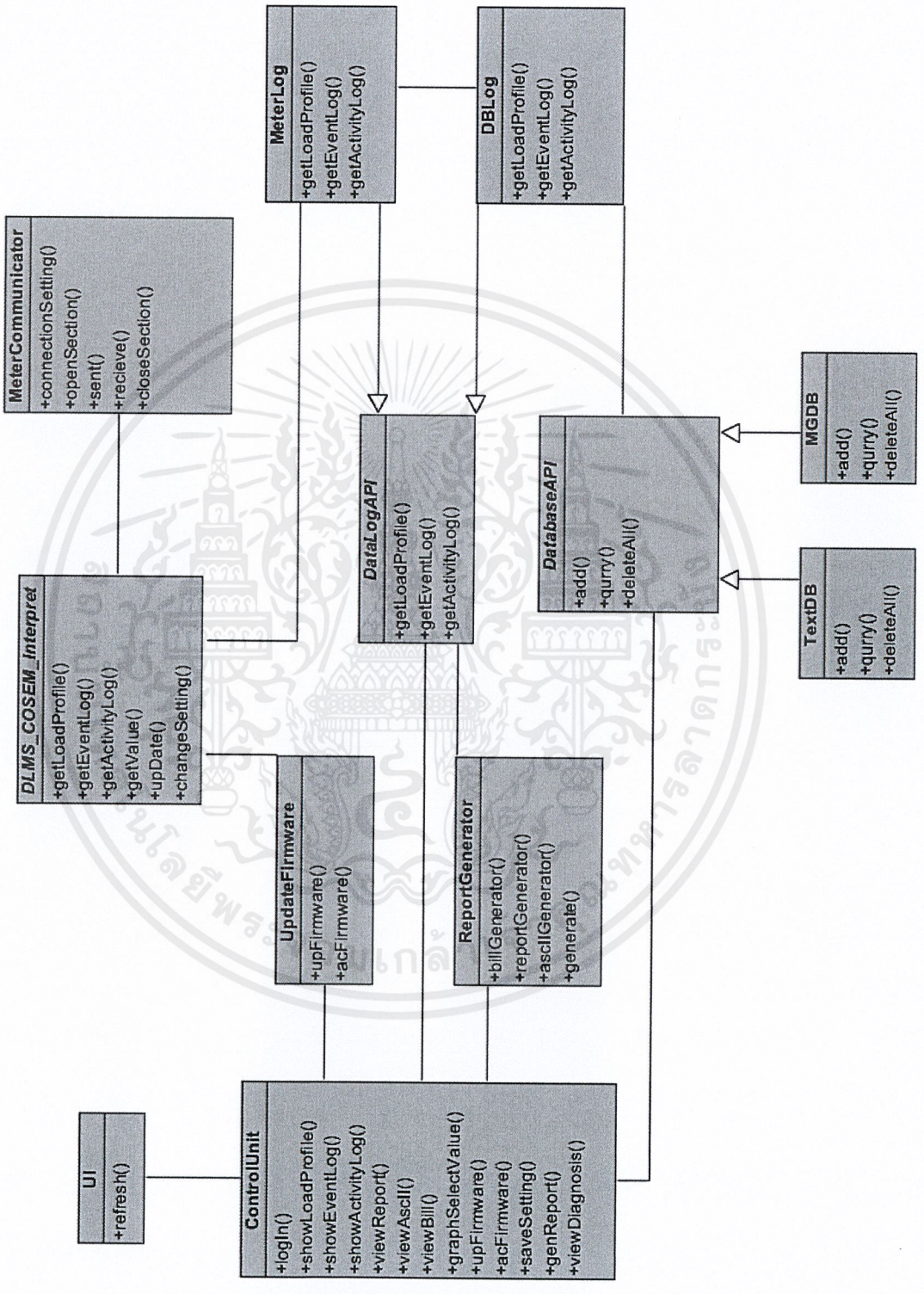


Figure 5.3 : Supporting Software Class Diagram

1. ControlUnit class, it is a main class of the supporting software. It is a main thread, which handling all of interrupt of the software.
2. UI class, it is an interface class to display the output of the software and interact with the user.
3. UpgradeFirmware class, it is a class, which is used for upload a new firmware to the smart meter.
4. ReportGenerator class, it is a class, which is used for generate the excel file report and ascii text file report.
5. DLMS_COSEM_Interpret class, it is an interface class of DLMS/COSEM encoder/decoder module.
6. MeterCommunicator class, it is a communication interface class of serial port communication module to the other device. This class can work with any kind of serial port communication.
7. DatabaseAPI class, it is an abstract class of database module inside the supporting software. Every database module class needs to inherit from this class.
8. TextDB class, it is a database class inherits from DatabaseAPI class for text database module.
9. MGDB class, it is a database class inherits from DatabaseAPI class for MongoDB database module.
10. DatalogAPI class, it is an abstract class of the metering datalog.
11. DBLog class, it is a proxy class of the metering datalog, which is called by the other class for the metering datalog, if that datalog is available in database it will read from database, if not it will call a MeterLog class to read from the meter.
12. MeterLog class, it is a class, which is called by a DBLog for reading the metering detalog from the smart meter.

5.2.2 In-home display

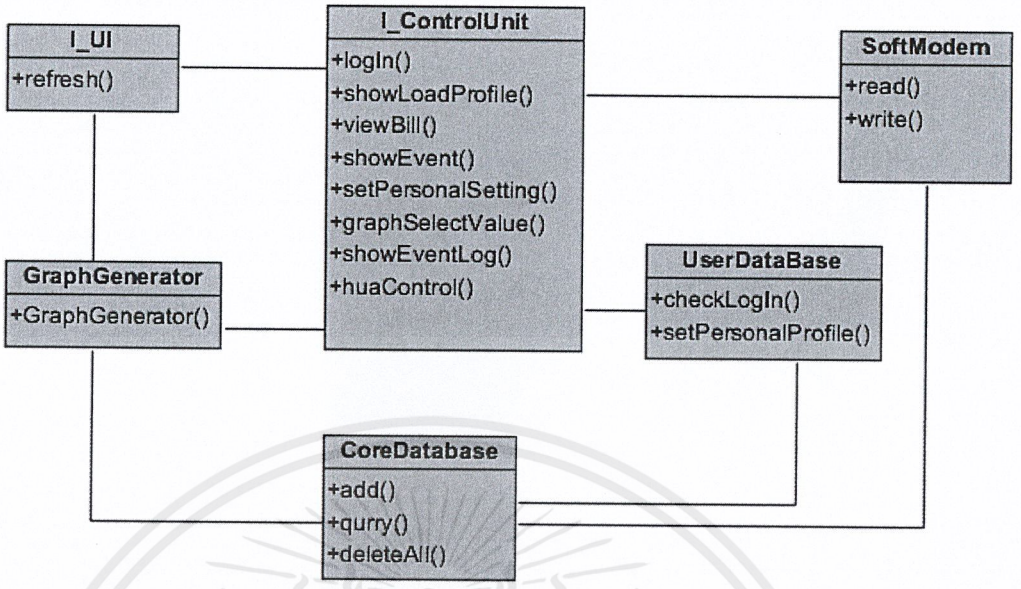


Figure 5.4 : In-home Display Class Diagram

1. I_ControlUnit class, it is a main class of the in-home display software. It is a main thread, which handling all of interrupt of the software.
2. I_UI class, it is an interface class to display the output of the software and interact with the user.
3. UserDataBase, it is a class, which controls the user information.
4. GraphGenerator class, it is a class, which is used for generate the graph that represent on the display.
5. CoreDatabase class, it is a database class of iOS database module.
6. SoftModem class, it is a communication interface class of audio port communication module to the other device.

5.3 DLMS/COSEM Encoder and Decoder

In order to develop the encoder and decoder of DLMS/COSEM messages, we need to study how to develop a concrete syntax tree, what the DLMS/COSEM syntax will be looked like and how to encode and decode the DLMS/COSEM messages.

A concrete syntax tree or a parse tree is an ordered tree that represents a syntactic structure of strings according to some formal grammar. Parse trees are usually constructed according to one of two competing relations, either in terms of the constituency relation of constituency grammars (= phrase structure grammars) or the dependency relation of dependency grammars. A parse tree is distinct from an abstract

syntax tree (also known simply as syntax trees), in that, its structure and elements more concretely reflect the syntax of the input language. Parse trees may be generated for sentences in natural languages as well as generated during processing of computer languages.

Since DLMS/COSEM is a protocol under ASN.1 standard, it already has a syntax tree available. The syntax tree we use for parsing DLMS/COSEM messages is created by a company named Icube Software and it is available at http://www.cyamon.com/Syntax/pdu_syntax.html. We have included a complete list of this syntax tree in Appendix A of this thesis. So we can develop our encoder and decoder using this syntax tree.

According to the DLMS/COSEM syntax tree, the tree has different types of nodes. These types of nodes are:

1. A choice node, it is a node, when the parser traverses to this node, it will take 1 or more token to decide which one of its branch nodes the parser should traverse into.
2. A sequence node, it is a node, when the parser traverses to this node, the parser needs to traverse into every of its branch node sequentially, except for some optional branch nodes in which the parser need not traverse into if the next token it receives is 00.
3. A leaf node, it is a node that does not have any branch node. The node of this type usually takes 1 or more token to process. If the input is correct and the parser does not have more sequence to traverse. The parser will stop at this node.
4. An enumerated node, this kind of node is similar to a leaf node in a sense that it does not have any branch node. This type of node takes one token, which is a member of a predefined set of constants.
5. A sequence-of node, it is a node that takes a token which tell how many number of times that the parser needs to repeat traversing into its branch node. This type of node has only one branch node which is a sequence node, a leaf node, or an enumerated node.
6. A sequence-of-choice node, it is similar to a sequence-of node, but instead of it having a sequence node, a leaf node, or an enumerated node as its branch node, this node has a choice node as its branch node. Every time that the

parser traverses into that choice node, the choice that the parser makes at that choice node may be different from the previous visit.

In order to create a parse tree to decode a DLMS/COSEM message, we need to handle different types of nodes, we just described above, differently, including its branch nodes, branching methods to apply, and a number of tokens to be assign to a node.

Because a DLMS/COSEM message is represented as a sequence of hexadecimal numbers. So it is unnecessary to use a lexical analysis technique to turn this DLMS/COSEM message into a meaningful sequence. We can however split this sequence of hexadecimal numbers into an array of one-byte long tokens. Then we can use this array as an input for the parser to create a parse tree. After the parser completes this process the DLMS/COSEM message will be interpreted into a Python command which has been stored in a dictionary, as seen in figure 5.4.

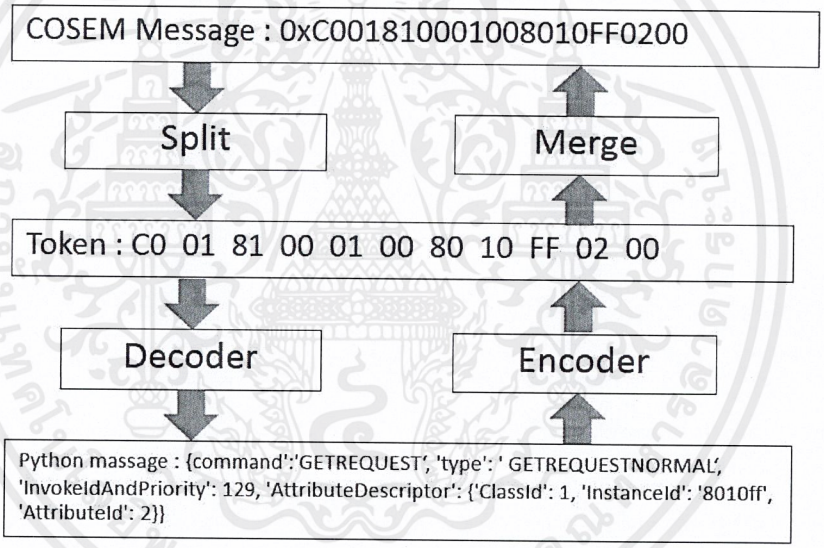


Figure 5.5 : Encoding & Decoding process

In the decoding process, the DLMS/COSEM message is split into an array of one-byte long tokens. Then the array of tokens will be given to the decoder to create a parse tree, if the process is successful, the decoder will return a corresponding Python command of the DLMS/COSEM message.

In the opposite, the encoder will take a Python command and encode it into a DLMS/COSEM message, if the command is correct, the encoder will return a corresponding array of tokens of the DLMS/COSEM message.

Chapter 6

System Development

In this chapter we will explain in detail how we can develop our DLMS/COSEM encoder and decoder. In addition we also describe how we implement the HDLC protocol.

6.1 DLMS/COSEM Encoder/Decoder

In order for the supporting software to understand the DLMS/COSEM protocol, the DLMS/COSEM encoder and decoder are needed. Since we cannot find a DLMS/COSEM encoder and decoder module in Python, so we need to develop the encoder and decoder by ourselves.

6.1.1 DLMS/COSEM decoder

We define a function **COSEMpdu** to work as the DLMS/COSEM decoder. Its definition is:

```
def COSEMpdu(x):
    global data,dataOri
    data = splitByte(x)
    dataOri = data
    temp = data.pop(0)
    return COSEMpduDic[temp]()
```

This function takes a large integer, which is a DLMS/COSEM message as its only argument. Initially the function defines two global variables **data** and **dataOri**. We split a DLMS/COSEM message into a list of one-byte long tokens using a function **splitByte(x)** and assign it to the two global variables. Later **data** will be used to parse into the parse tree and **dataOri** will keep the original split tokens.

For example, giving a DLMS/COSEM message

0xC001810001000080000FF0200

The function **splitByte(x)** will return

[0xC0, 0x01, 0x81, 0x00, 0x01, 0x00, 0x00, 0x80, 0x00, 0x00, 0xFF, 0x02, 0x00].

This function is defined as follow

```
def splitByte(data):
    temp = []
    while data > 0:
        temp.insert(0,int(data % 0x100))
        data = data / 0x100
    return temp
```

Then the function will get the first token and start parse into the parse tree, which is implemented differently according to a type of node.

1. Choice node, this kind of node has a list of branch node with a number indicate which branch will be chosen when specific indicate token has been brought.

```
_GetRequest ::= CHOICE
{
    GetRequestNormal [1] _GetRequestNormal,
    GetRequestForNextDataBlock [2] _GetRequestForNextDatablock,
    GetRequestWithList [3] _GetRequestWithList
}
```

A function that represent this kind of node need to have a matching dictionary that have a key is an indicating number and branch function as a value.

```
GetRequestDic = {
    1 : GetRequestNormal,
    2 : GetRequestForNextDataBlock,
    3 : GetRequestWithList
    }
def GetRequest():
    global data
    x = data.pop(0)
    temp = {'command':'GETREQUEST'}
    return GetRequestDic[x](temp)
```

When passing into this function, it took 1 token that decide which function in dictionary will be choose.

2. Sequence node, this kind of node has a list of branch node, which parser need to pass all item in this list except option branch. It usually not take a token except some special case.

```
_GetRequestForNextDatablock ::= SEQUENCE
{
  InvokeIdAndPriority _InvokeIdAndPriority ,
  BlockNumber _Unsigned32
}
```

A function that represent this kind of node is design for calling another function sequentially and keep in the dictionary.

```
def GetRequestForNextDataBlock(temp):
    temp['type'] = 'GETREQUESTFORNEXTDATABLOCK'
    temp["InvokeIdAndPriority"] = InvokeIdAndPriority()
    temp["BlockNumber"] = BlockNumber()
    return temp
```

3. Leaf node, this kind of node always represent variable or data in difference data types. It usually take constant number of token. But some node can vary the number of token took by it too.

```
def INTEGER16():
    global data
    x = data.pop(0)*0x100 + data.pop(0)
    if(x >= _v16b/2):
        x -= _v16b
    return x
```

This is example of leaf node, it represent 16 bits integer type, which takes exactly 2 tokens.

4. Enumerated node, this kind of node has a list of items to make a choice like choice node. But those items are not the function. Also it is similar with Leaf node in the sense that it not has any branch.

```
_DataAccessResult ::= ENUMERATED
{
   Success (0),
   HardwareFault (1),
   TemporaryFailure (2),
   ReadWriteDenied (3),
   ObjectUndefined (4),
   ObjectClassInconsistent (9),
   ObjectUnavailable (11),
   TypeUnmatched (12),
   ScopeOfAccessViolated (13),
   DataBlockUnavailable (14),
   LongGetOrReadAborted (15),
   NoLongGetOrReadInProgress (16),
   LongSetOrWriteAborted (17),
   NoLongSetOrWriteInProgress (18),
   DataBlockNumberInvalid (19),
   OtherReason (250)
}
```

Function of this kind of node developed similar to choice node. It takes a token to desire a result. But it has not branch to any further branch.

```
DataAccessResultDic = {
    0 : "Success",
    1 : "HardwareFault",
    2 : "TemporaryFailure",
    3 : "ReadWriteDenied",
    4 : "ObjectUndefined",
    9 : "ObjectClassInconsistent",
    11 : "ObjectUnavailable",
    12 : "TypeUnmatched",
    13 : "ScopeOfAccessViolated",
    14 : "DataBlockUnavailable",
    15 : "LongGetOrReadAborted",
    16 : "NoLongGetOrReadInProgress",
    17 : "LongSetOrWriteAborted",
    18 : "NoLongSetOrWriteInProgress",
    19 : "DataBlockNumberInvalid",
    250 : "OtherReason"
}
def DataAccessResult():
    global data
    x = data.pop(0)
    return DataAccessResultDic[x]
```

5. SequenceOf node, this kind of node always refer to another node, which is Sequence, Leaf or Enumerated node.

_ListOfDataAccessResult ::= **SEQUENCEOF** _DataAccessResult

The function that represent this kind of node is design to take a token and repeatedly call the same function and keep the result into a list to return.

```
def ListOfDataAccessResult(n):
    temp = []
    for i in range(n):
        temp.append(DataAccessResult())
    return temp
```

6. SequenceOfChoice node, this kind of node is work and look the same way as SequenceOf node but it refer to choice node only.

_ListOfData ::= **SEQUENCEOFCHOICE** _Data

So the function is look the same with the function of SequenceOf node too.

```
def ListOfData(n):
    temp = []
    for i in range(n):
        temp.append(Data())
    return temp
```

Then we link all of these functions together and create a tree for a list of bytes (tokens) to passing through and keep it in form of dictionary and after that this list of tokens is given to the parser to create a parse tree like

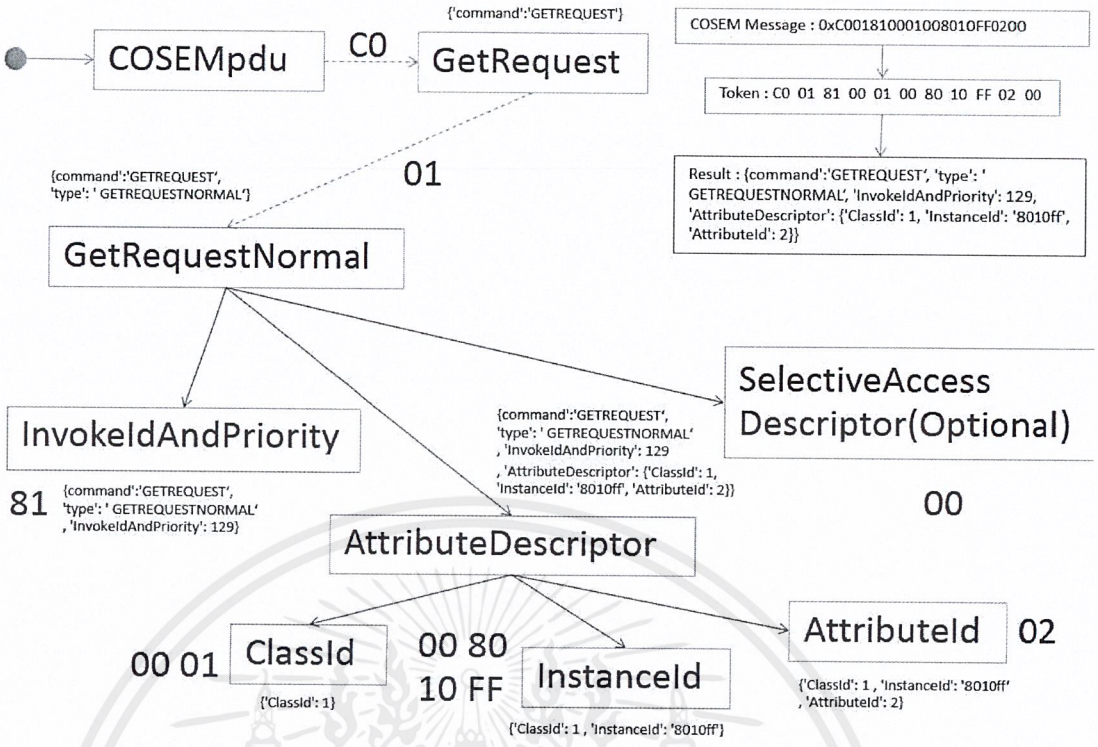


Figure 6.1 : Example of COSEM massage pass thought parsing tree

After finish create a parse tree of a DLMS/COSEM message, the decode function will return the correspond Python command like

{'AttributeDescriptor': {'ClassId': 1, 'InstanceId': '800000ff', 'AttributeId': 2}, 'type': 'GETREQUESTNORMAL', 'command': 'GETREQUEST', 'InvokeIdAndPriority': 129}

6.1.2 DLMS/COSEM encoder

We define a function **toCOSEM** to work as the DLMS/COSEM encoder. Its definition is:

```
def toCOSEM(x):
    global data
    data = x
    array = COSEMpduDic[data['command']]()
    temp = mixback(array)
    return temp
```

This function takes a dictionary, which is represented a Python command as its only argument. Initially the function defines one global variable **data**. We assign an input dictionary to that global variable. Later **data** will be used to parse into the parse tree. For example, a dictionary is.

{'AttributeDescriptor': {'ClassId': 1, 'InstanceId': '800000ff', 'AttributeId': 2}, 'type': 'GETREQUESTNORMAL', 'command': 'GETREQUEST', 'InvokeIdAndPriority': 129}

As same as a decoder the function will parse into a parse tree like figure 6.2

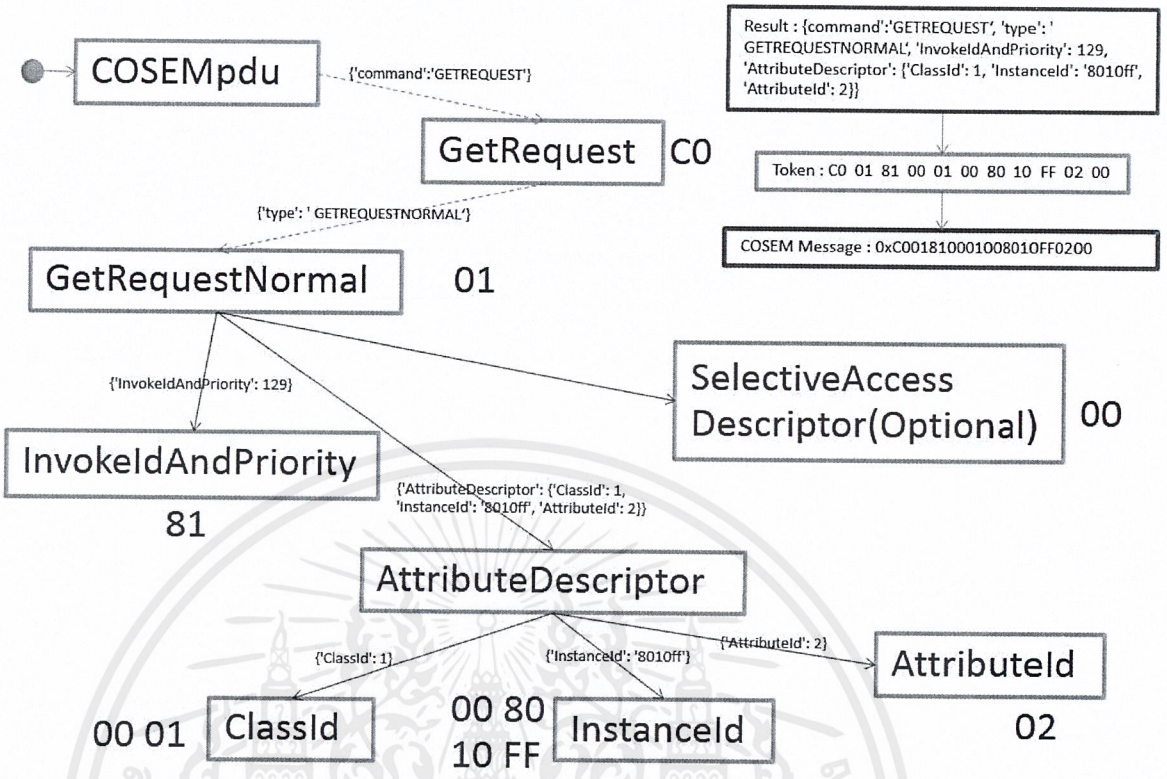


Figure 6.2 : Example of Python command pass thought parsing tree

The functions will gather the tokens along the path of parsing process and add it in to the end of the list and after finish this parsing process the list of tokens, which is assign to **`array`** variable, will be look like:

[0xC0, 0x01, 0x81, 0x00, 0x01, 0x00, 0x00, 0x80, 0x00, 0x00, 0xFF, 0x02, 0x00].

Then we use the function `mixback(array)` to merge a list of token into a DLMS/COSEM message like:

0xC0018100010000800000FF0200

This function is defined as follow

```
def mixback(data):
    temp = 0
    for i in data:
        temp *= 0x100
        temp += i
    return temp
```

Next, we will explain about how to send and receive the DLMS/COSEM message using HDLC protocol.

6.2 HDLC communication module

Since our smart meter uses HDLC protocol at the link layer we have to develop this protocol to accommodate transition of DLMS/COSEM messages. Next we define **sendHDLC()** for sending a DLMS/COSEM message from the supporting software to the smart meter, and we define **readHDLCTimeout()** for the supporting software to receive a DLMS/COSEM message from the smart meter.

6.2.1 HDLC Sender

This function **sendHDLC()** takes a string representing a HDLC frame type, and a long integer which is a DLMS/COSEM message, as its arguments. Its definition is:

```
def sendHDLC(com = '',d = 0):

    com = com.lower()
    if com not in comTable:
        print 'command wrong'
        return

    x,temp = comTable[com](d)

    sendIt(x)
```

Initially the function will check the argument string whether it is an HDLC frame type or not by checking it against the dictionary **comTable**:

```
comTable = {
            'dm' : createHDLC_DM,
            'ua' : createHDLC_UA,
            'snrm' : createHDLC_SNRM,
            'rr' : createHDLC_RR,
            'i' : createHDLC_I
            }
```

If it not in the dictionary, the function will terminate.

However, if the argument string is an HDLC frame type the function **sendHDLC()** will handle the argument string by the corresponding function picked up from the dictionary **comTable**. The function will be invoked by parsing the DLMS/COSEM message as its argument.

The result of the function call are a generated HDLC frame and its frame size. Then the generated HDLC frame will be passed to a function **sendIt()** to send to the smart meter via the serial port of a computer running the supporting software.

6.2.2 HDLC Receiver

We define a function **readHDLCTimeout()** which takes an integer representing a number of times of reading before timeout, as its only argument. The function reads a HDLC frame from the serial port. Its definition is:

```
def readHDLCTimeout(out = 5):
    x = -2
    while x < 0:
        if x == -1:
            out -= 1
            if(out < 0):
                return -1

        time.sleep(0.01)
        x = readHDLC()
        #print hex(x)
    return x
```

This function employs a function **readHDLC()**, which receives the HDLC frame from serial communication port and extract a DLMS/COSEM message from the HDLC frame and returns the DLMS/COSEM message. The function **readHDLCTimeout()** will repeat calling **readHDLC()** until it exceeds the limit of timeout or receive a valid HDLC frame.

6.3 SoftModemTerminal Application

Since our in-home display needs to communicate with Hi-jack using sound via an audio port. So we need a library translate between sound code and text characters using FSK modulation technique.

After we search for a modulation library, we found an application called "SoftModemTerminal", which is a simple terminal application developed for an iOS device by Switch-Science company (http://www.switch-science.com/catalog/364/). The application takes text characters and modulates them into a sound wave using the FSK technique.

The SoftModemTerminal application is written in Objective-C and can be downloaded from the website and we applied some of this application libraries to our in-home display application for it to communicate with the Hi-jack adapter.

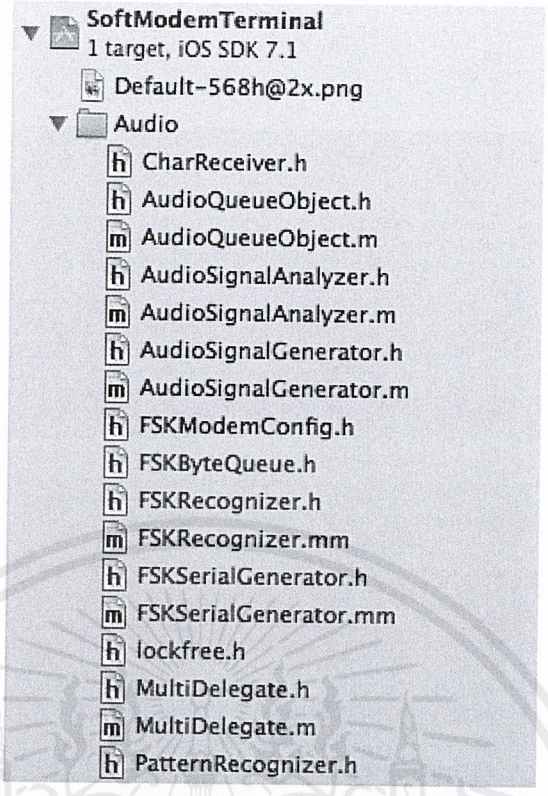


Figure 6.3 : The source files of the SoftModemTerminal.

6.3.1 Receiver

The SoftModemTerminal application uses AudioSignalAnalyzer module and FSKRecognizer module to initialize the data demodulation. AudioSignalAnalyzer module receives the input waveform from the microphone in order to normalize the input waveform to have a constant amplitude. The output waveform is passed to FSKRecognizer module to demodulate the waveform to include the data bits as seen below:

```
recognizer = [[FSKRecognizer alloc] init];
analyzer = [[AudioSignalAnalyzer alloc] init];
[Analyzer addRecognizer: recognizer]; // set recognizer to analyzer
[analyzer record]; // start analyzing
```

Then, the application employs the CharReceiver module to receive the data bits from FSKRecognizer module and convert them to an 8-bit character and pass the character to "**receivedChar:**" method. We can define this function to handle this character by ourselves.

```
@ Interface MainViewController: UIViewController <CharReceiver>

MainViewController * mainViewController;
[Recognizer addReceiver: mainViewController];
```

```
- (Void) receivedChar: (char) input
{
    // Receive handling
}
```

6.3.2 Sender

The application employs FSKSerialGenerator module, which is used for the data modulation, to create a module sound output from the data bit input.

```
generator = [[FSKSerialGenerator alloc] init];
[Generator play]; // audio output starts
```

When the application wants to send the data, it uses a method writeByte in FSKSerialGenerator module to modulate the data and play out the modulated sound.

```
[Generator writeByte: 0xff];
```

6.4 Testing the SoftModemTerminal with Hijack

In order to check whether the SoftModemTerminal can work with the Hijack or not, we need to test it by pluging the hi-jack to the audio port of the iPhone, which is already installed with the SoftModemTerminal application, and then run the application by sending the data from the hi-jack, if the communication of the iPhone is successfully, the data will be displayed in the textField.

```
- (BOOL)textFieldShouldReturn:(UITextField *)textField
{
    [[SoftModemTerminalAppDelegate getInstance].generator writeByte:0xff];
    [[SoftModemTerminalAppDelegate getInstance].generator writeBytes:[textField.text UTF8String]
                                                              length:[textField.text
                                                                lengthOfBytesUsingEncoding:
                                                                NSUTF8StringEncoding]];

    textField.text = @"";
    return YES;
}
```

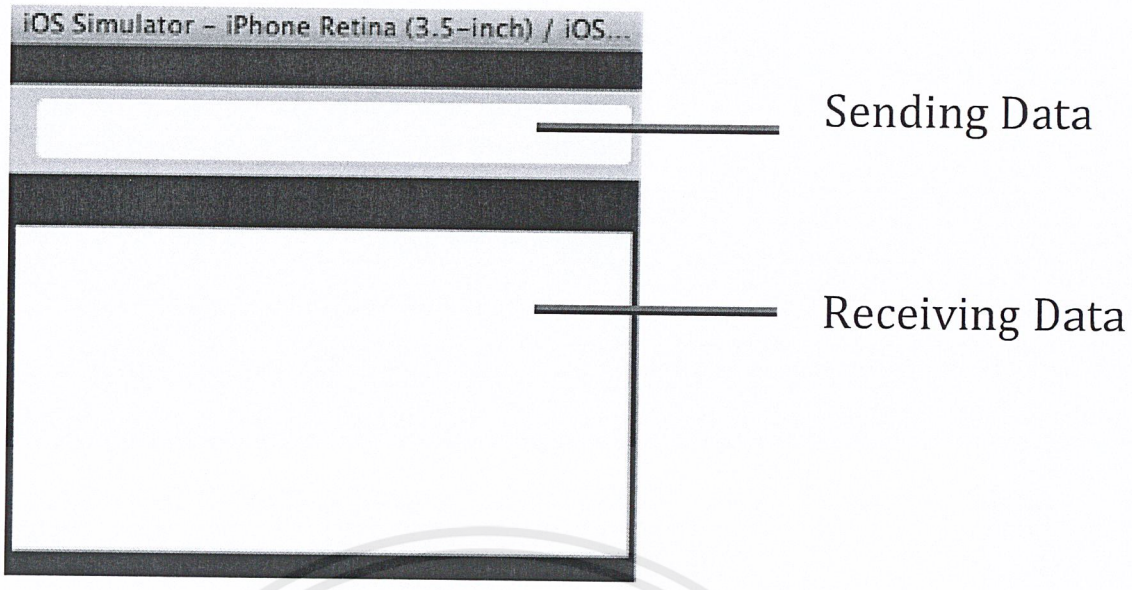


Figure 6.4 : SoftModemTerminal on iOS Simulator.

6.5 Integrating the SoftModemTerminal with the audio module

After we study and test the hi-jack and the modules inside the SoftModemTerminal application, we need to integrate these modules, which provide the FSK techniques as for sound modulation and demodulation, into our in-home display application to be used as the background module for communication.

```
#import "MeterViewController.h"
#import "DemoCalendarMonth.h"
#import "AudioSignalAnalyzer.h"
#import "FSKSerialGenerator.h"
#import "FSKRecognizer.h"
#import "CharReceiver.h"

#import <AVFoundation/AVFoundation.h>

@interface MeterViewController ()
<AVAudioSessionDelegate, CharReceiver>

@property(nonatomic, strong) NSTimer* timer;
@property(nonatomic, strong) NSTimer* timer2;
@property(nonatomic, strong) NSArray* data;
@property(nonatomic, assign) NSInteger index;

@property (nonatomic, strong) FSKRecognizer* recognizer;
@property (nonatomic, strong) FSKSerialGenerator* generator;
@property (nonatomic, strong) AudioSignalAnalyzer* analyzer;

@property (nonatomic, strong) NSMutableString* bufferStr;
@property (nonatomic) NSUInteger counter;

@end
```

We also need to check whether the integration between the background module for communication and the in-home display main application is successfully or not, by checking the module "receivedChar", whether it can receive and count the input, and then show the output in the application.

```
-(void) receivedChar:(char)input
{
    [self.bufferStr appendFormat:@"%c", input];

    if(self.bufferStr.length % 5 == 0){
        [self.bufferStr appendString:@"\n"];
    }
    self.contentTv.text = self.bufferStr;
    self.counter = self.counter + 1;
    self.counterLabel.text = [NSString stringWithFormat:@"Recv: %d ch",
                              self.counter];
}
```

Chapter 7

Experimentation

In this chapter we will explain how to use our supporting software and the in-home display. We will explain briefly each feature of each software.

7.1 Supporting software

Supporting software interface is designed to reduce a complexity for helping the utility company staff to manage a smart meter easily. The interface has separated screens for each kind of functionality of supporting software.

7.1.1 Home screen

The home screen is the main screen of the software. This screen shows meter data read from a smart meter. This page consist of:

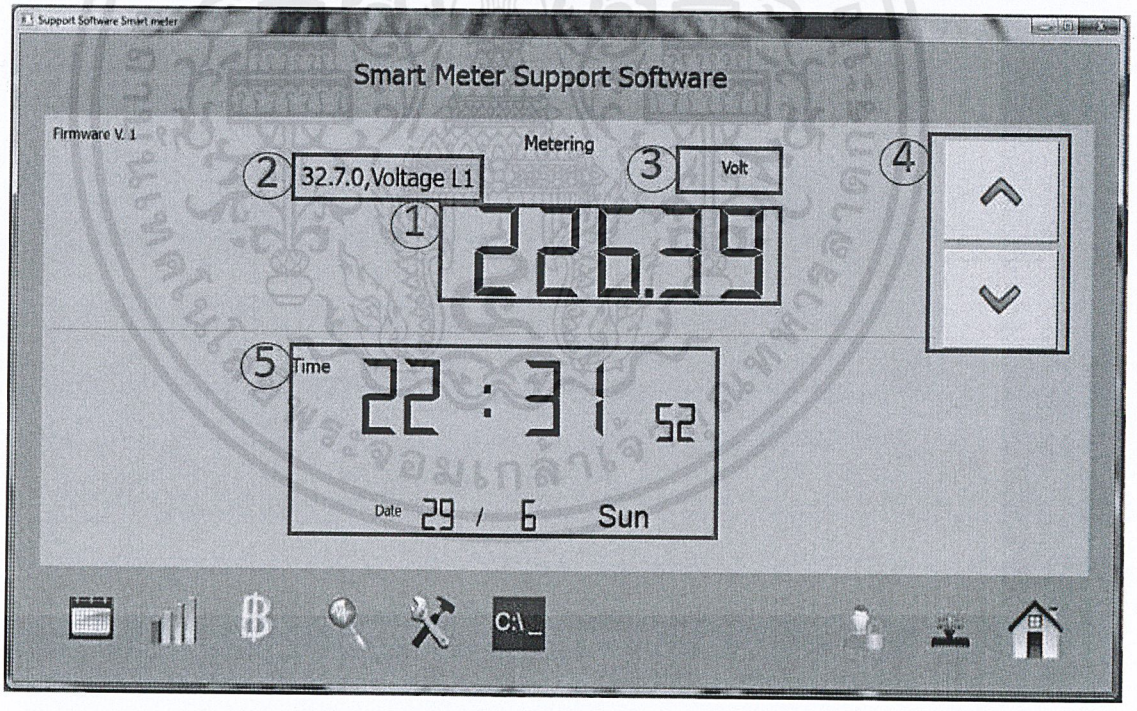


Figure 7.1 : Support software home screen

1. LCD display for meter value
2. Name of presenting meter value
3. Unit of presenting meter value
4. Scroll buttons to scroll to change presenting meter value

5. LCD display for time and date

This interface has the picture buttons in the below section, which used for redirect to difference screen that represent each functionality.

1.  Calendar button is to invoke the calendar screen
2.  Graph button is to invoke the real-time value graph represent screen
3.  Billing button is to invoke the bill and report generating screen
4.  Setting button is to invoke the setting screen
5.  Download button is to invoke the upgrade firmware screen
6.  Home button is to return to the home screen

7.1.2 Calendar screen

This screen presents a calendar interface for viewing historical meter data read from the smart meter according to a certain time and date. The user can choose a date he/she wants to look at the historical meter data of that date.

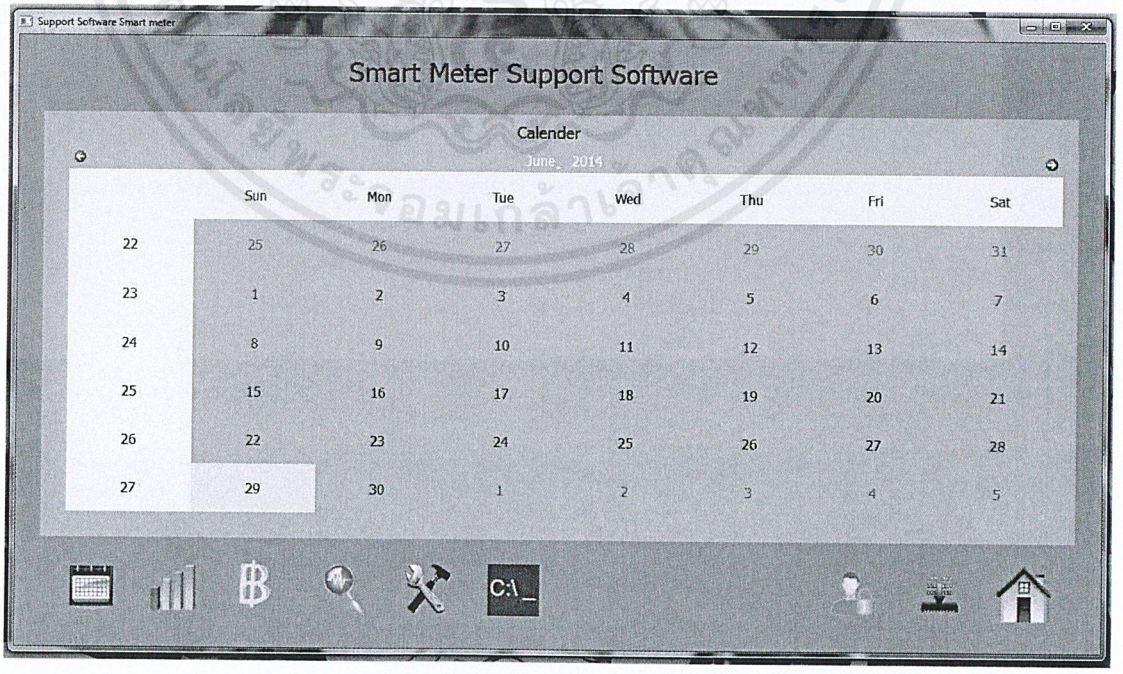
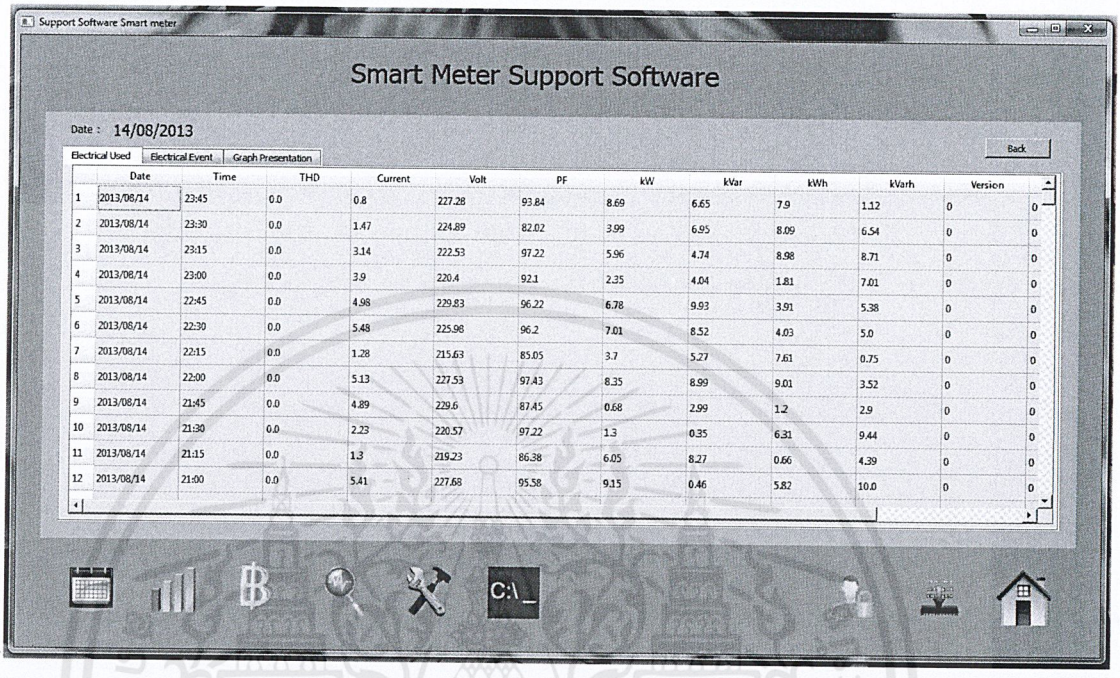


Figure 7.2 : Support software calendar screen

When user chooses a date from the calendar. The screen will invoke the historical data screen for that date, which has 2 forms of representation:

1. Tabular representation presents all of the historical data for the selected date in a table.



	Date	Time	THD	Current	Volt	PF	kW	kVar	kWh	kVarh	Version	
1	2013/08/14	23:45	0.0	0.8	227.28	93.84	8.69	6.65	7.9	1.12	0	0
2	2013/08/14	23:30	0.0	1.47	224.89	82.02	3.99	6.95	8.09	6.54	0	0
3	2013/08/14	23:15	0.0	3.14	222.53	97.22	5.96	4.74	8.98	8.71	0	0
4	2013/08/14	23:00	0.0	3.9	220.4	92.1	2.35	4.04	1.81	7.01	0	0
5	2013/08/14	22:45	0.0	4.98	229.83	96.22	6.78	9.93	3.91	5.38	0	0
6	2013/08/14	22:30	0.0	5.48	225.98	96.2	7.01	8.52	4.03	5.0	0	0
7	2013/08/14	22:15	0.0	1.28	215.63	85.05	3.7	5.27	7.61	0.75	0	0
8	2013/08/14	22:00	0.0	5.13	227.53	97.43	8.35	8.99	9.01	3.52	0	0
9	2013/08/14	21:45	0.0	4.89	229.6	87.45	0.68	2.99	1.2	2.9	0	0
10	2013/08/14	21:30	0.0	2.23	220.57	97.22	1.3	0.35	6.31	9.44	0	0
11	2013/08/14	21:15	0.0	1.3	219.23	86.38	6.05	8.27	0.66	4.39	0	0
12	2013/08/14	21:00	0.0	5.41	227.68	95.58	9.15	0.46	5.82	10.0	0	0

Figure 7.3 : Support software log table page

2. Graph representation presents a graph of changing value of one type of meter data. User can choose to view only one type of data at a time.

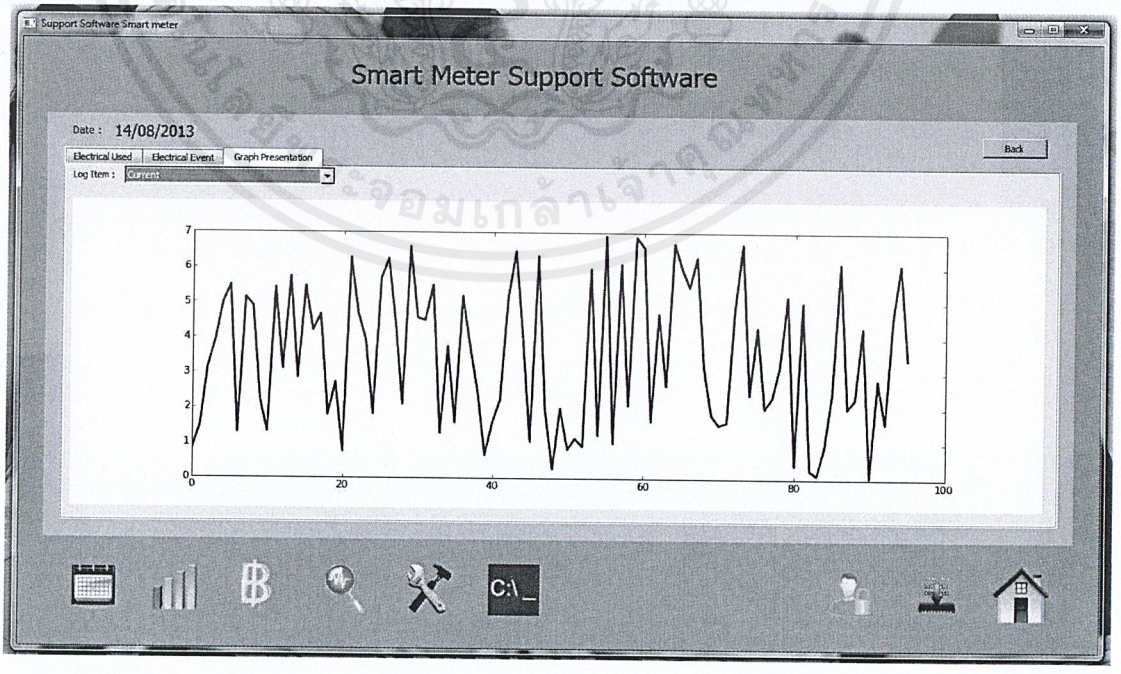


Figure 7.4 : Support software log graph screen

7.1.3 Graph screen

This screen presents the real-time meter data as the graph. User can choose to view only one type of meter data at a time, such as current, voltage, Watt, Var, VA, etc.

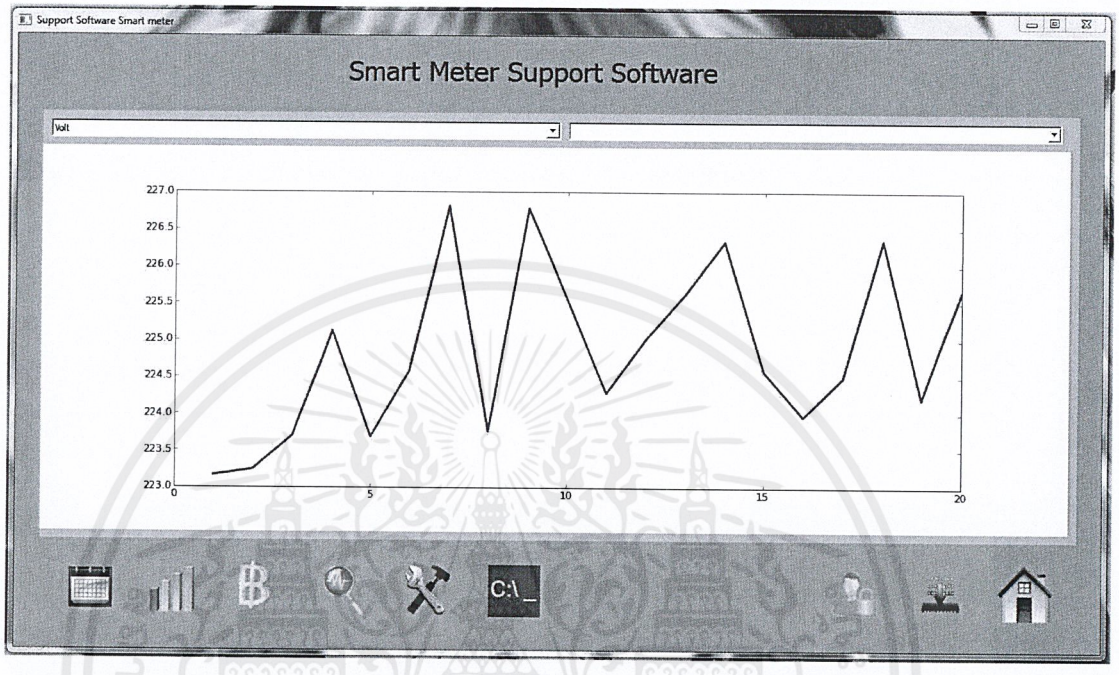


Figure 7.5 : Support software real-time graph screen

7.1.4 Billing and report generating screen

This screen is used by the user for generating an excel file and an ascii text file report by selecting a start and end dates of the electrical usage for the report. Then user clicks on "Create Excel File" to create an excel file report or "Create Ascii Text File" to create an acsii text file report.

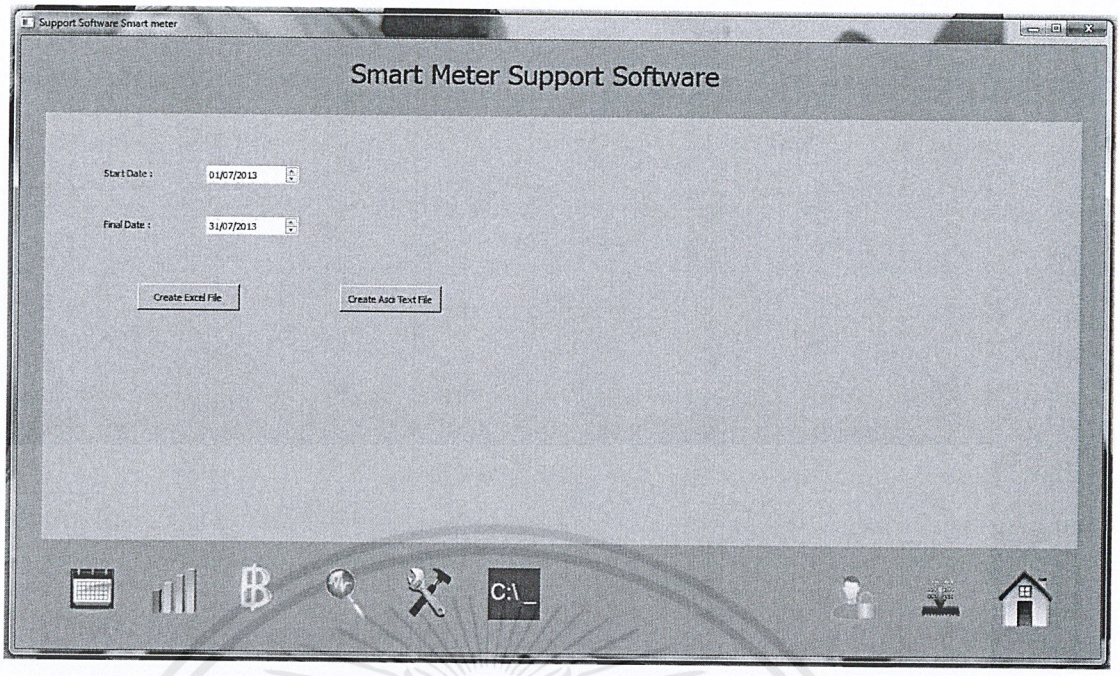


Figure 7.6 : Support software report generate screen

7.1.5 Setting screen

This screen is used for setting the parameters used by the meter, which includes:

1. Display values, which are the electrical values to show on the LCD display of the smart meter,
2. TOU, which is the tariff rate according to the Time-of-Use of the smart meter.

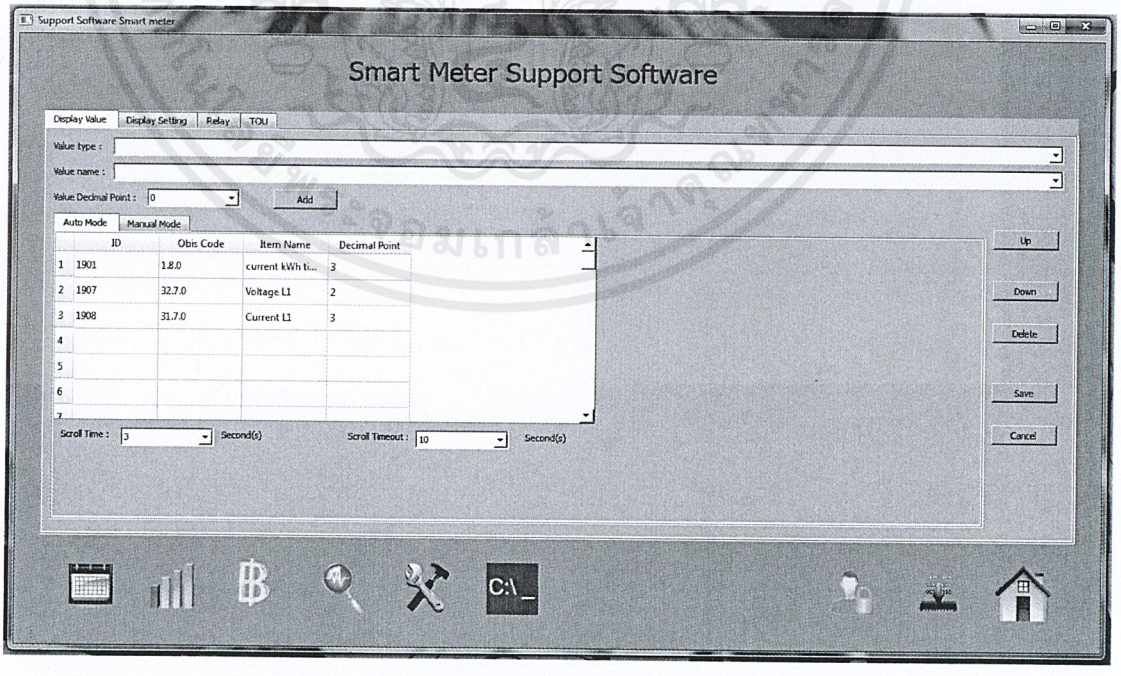


Figure 7.7 : Support software setting screen

7.1.6 Upgrade firmware screen

This screen is used for upload the new firmware to the smart meter. The user can upload a new firmware by choosing a new firmware, next click "Upgrade Firmware" and wait until the uploading process is finish and then click "Active Firmware" to make the smart meter restarts and runs the new firmware.

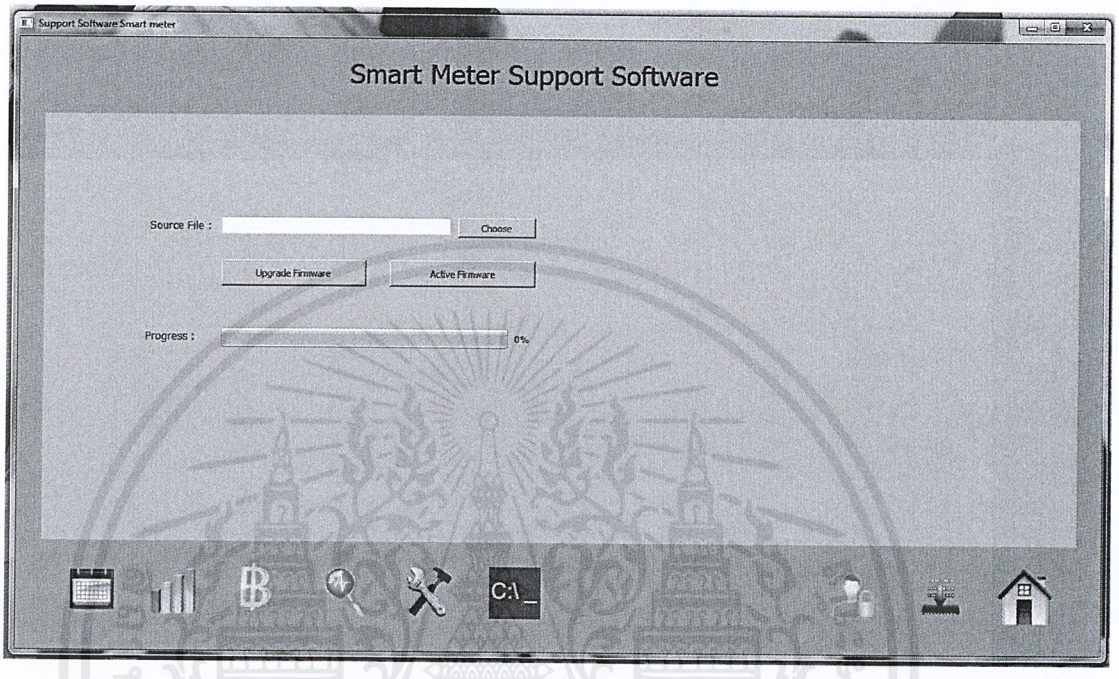
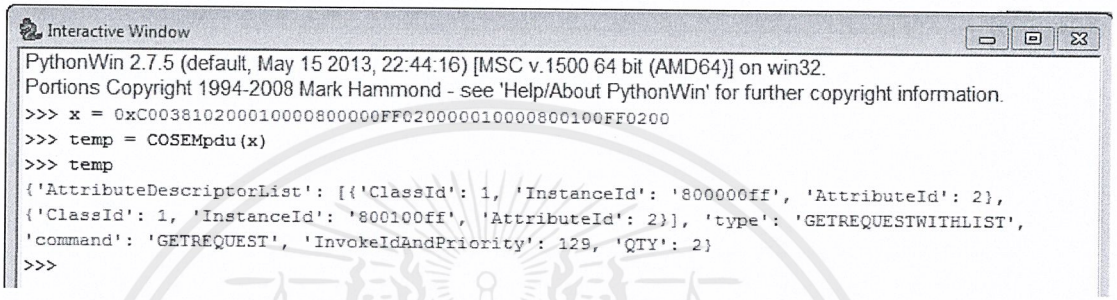


Figure 7.8 : Support software upgrade firmware screen

7.2 DLMS/COSEM Encoder/Decoder

7.2.1 Encoder and decoder cross checking

In order to confirm the accuracy of encoder and decoder and make sure they can be work together, we need to make a cross checking between these two modules by choose some example of DLMS/COSEM message then decode the message with our decoder, so we get Python message. Then we encode Python message with the encoder and check back it still be the same COSEM massage or not.



x = 0xC003810200010000800000FF020000010000800100FF0200

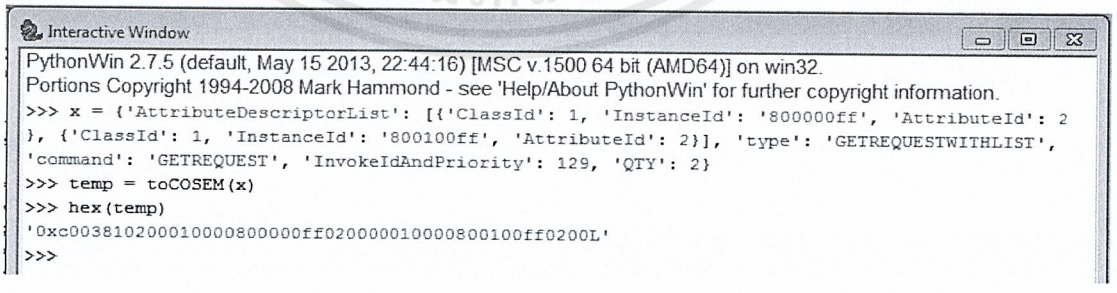
is an example of COSEM message

COSEMpdu(x)

is the function to translate COSEM message into python massage

{'AttributeDescriptorList': [{'ClassId': 1, 'InstanceId': '800000ff', 'AttributeId': 2}, {'ClassId': 1, 'InstanceId': '800100ff', 'AttributeId': 2}], 'type': 'GETREQUESTWITHLIST', 'command': 'GETREQUEST', 'InvokeIdAndPriority': 129, 'QTY': 2}

is python massage output of this interpret. Then we use is massage to test our encoder.



Output of encoder is

'0xc003810200010000800000ff020000010000800100ff0200L',

When compare to the input from the first time, which is

0xC003810200010000800000FF020000010000800100FF0200

They are the same massage (L letter in the end of result massage is the way Python indicate that the number is in Long format).

7.2.2 Experimenting with smart meter

After cross checking between encode and decoder, in this experiment we test our encoder and decoder with DLMS/COSEM message that send between smart meter and supporting software.

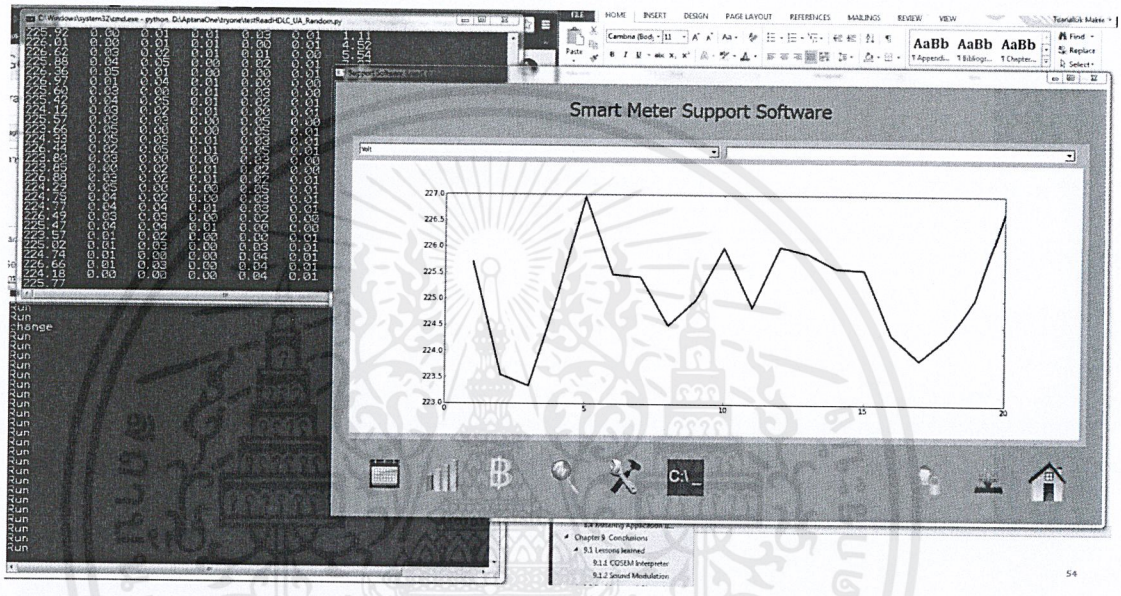


Figure 7.9 : Experiment encoder and decode by communicate with smart meter

The result of this experiment show that our encoder and decoder can work with COSEM massage, which sending between smart meter and support software properly.

7.3 In-home display

The in-home display software is designed for a home user to observe the household electrical consumption more easily. The interface has separated screen for each kind of functionality of in-home display.

7.3.1 Home screen

The home screen is the main screen of the software. This screen shows meter data read from a smart meter, time, and date. The user can scrolling the present data by using up/down button below.

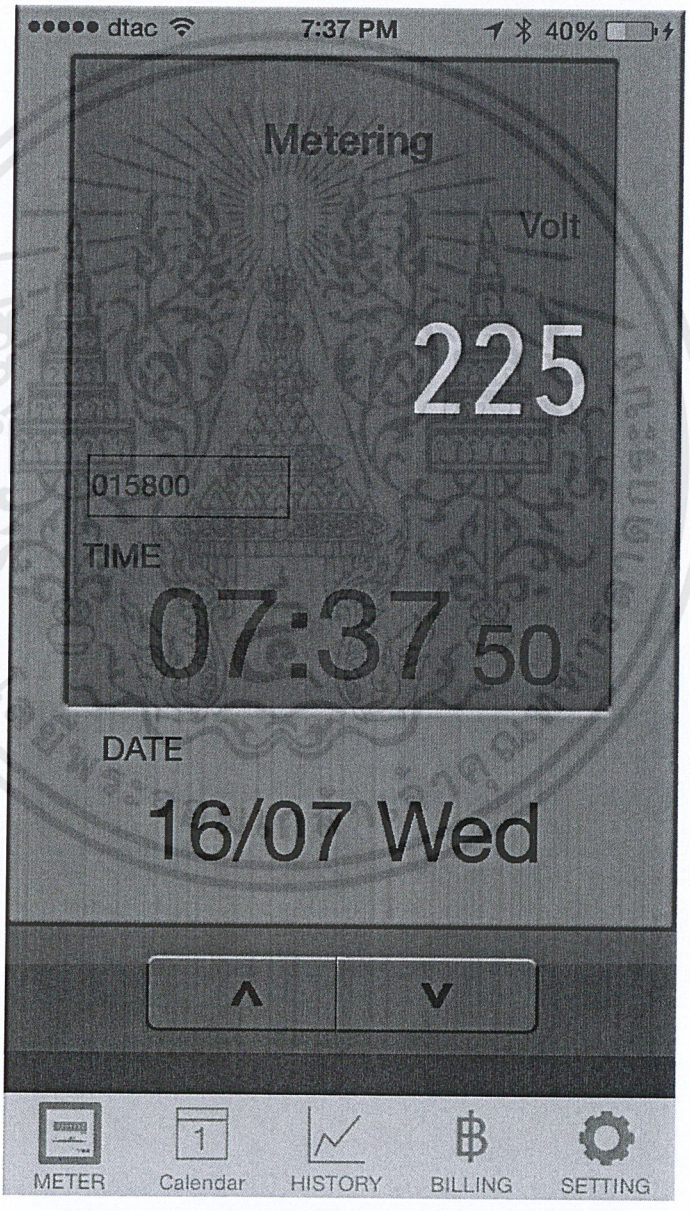


Figure 7.10 : In-home display home screen

7.3.2 Calendar screen

This screen presents a calendar interface for viewing historical meter data read from the smart meter according to a certain time and date. The user can choose a date he/she wants to look at the historical meter data of that date. The data will be presented in the below section of the screen.

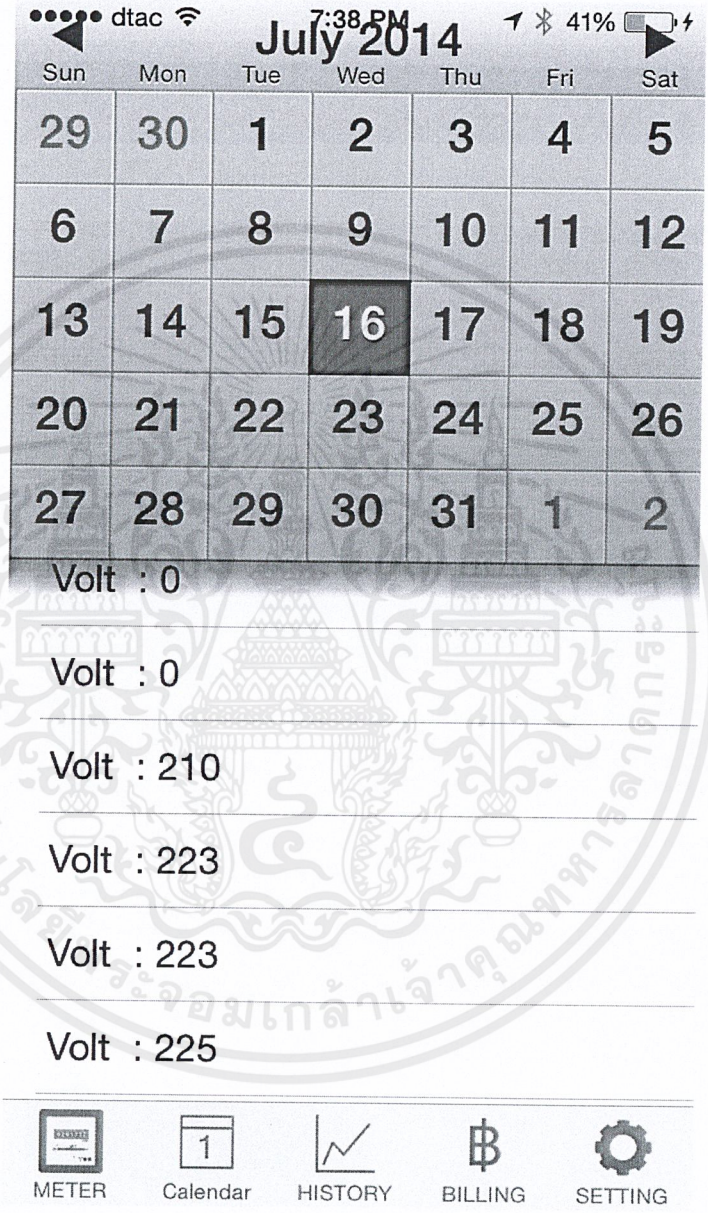


Figure 7.11 : In-home display calendar screen

7.3.3 Graph screen

This screen presents the historical meter data as the graph. The data presented in graph are voltage, current, Watt and Var. The graph can show up to 12 hours of recorded historical data.

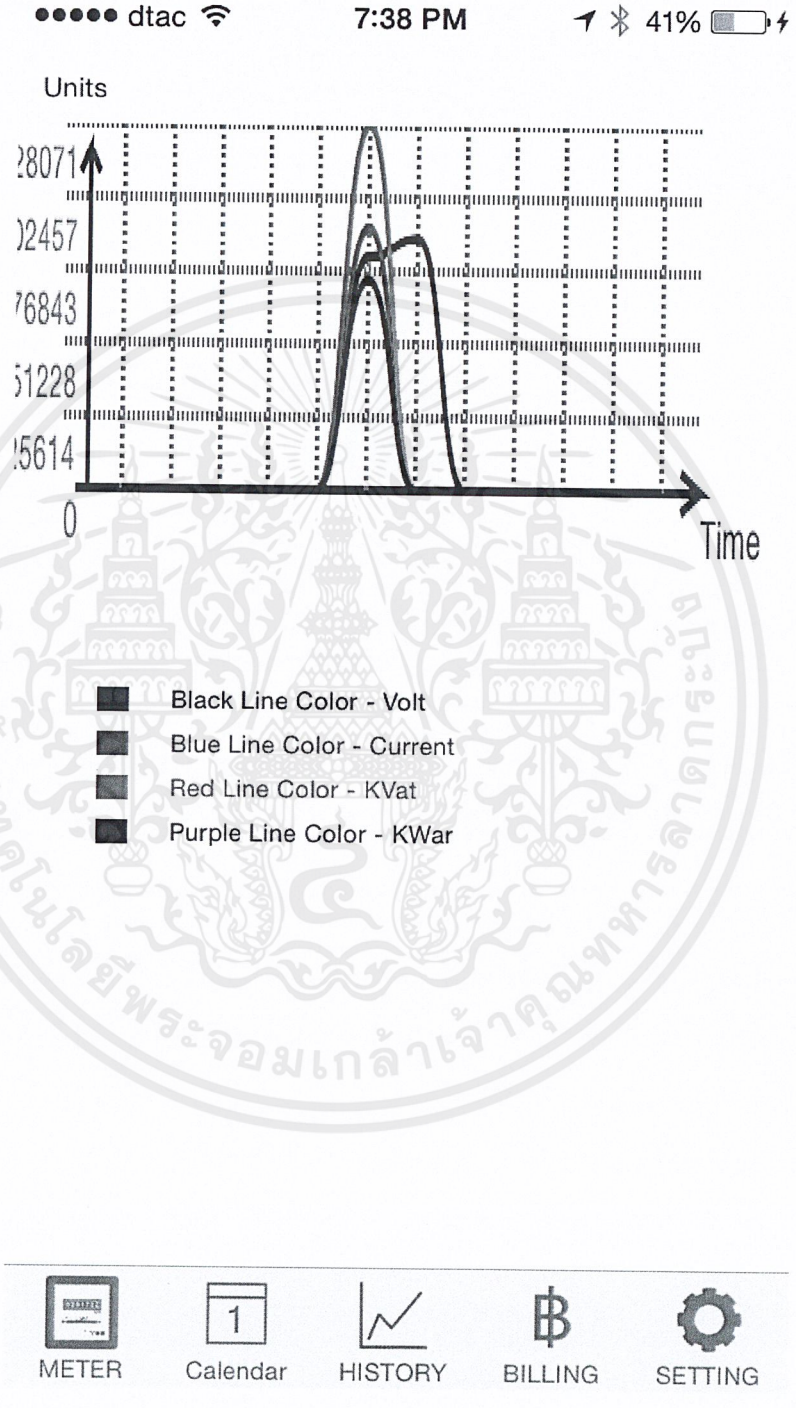


Figure 7.12 : In-home display graph screen

7.3.4 Billing screen

This screen presents the billing information. The data presented in this screen is including the general information of meter and user, and the bill of electrical on the recent month.

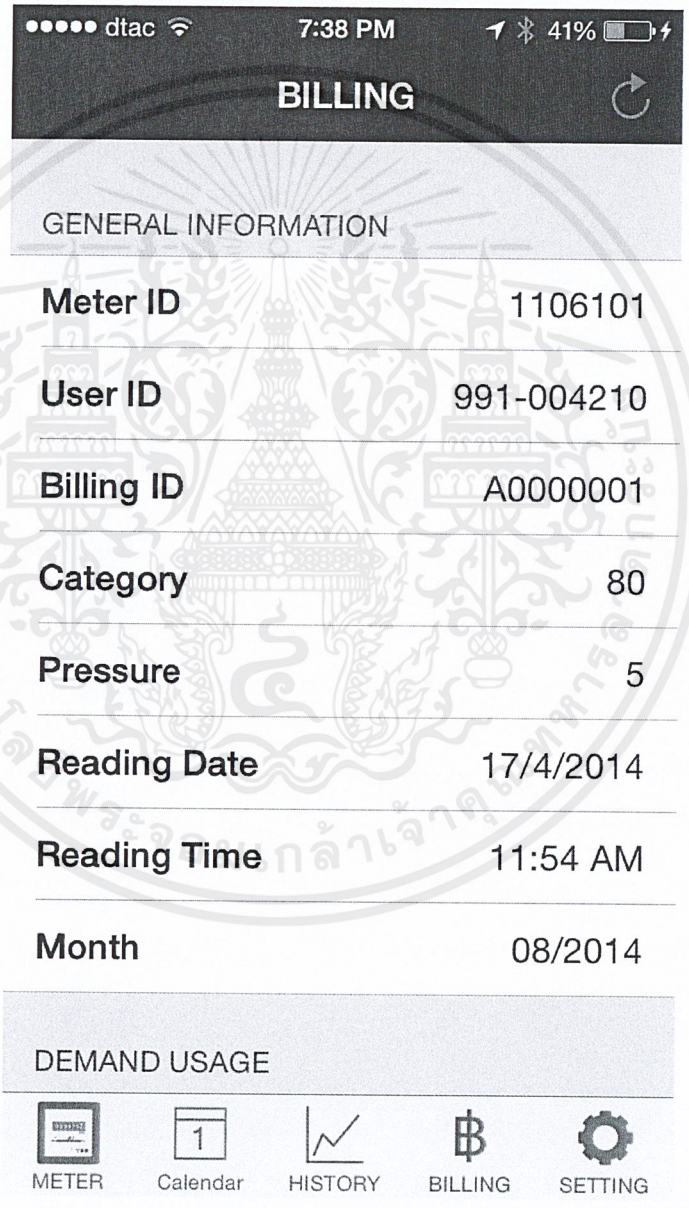


Figure 7.13 : In-home display billing screen

7.3.5 User setting screen

This screen presents the user information. The user can edit this information by tab on EDIT.

Figure 7.14 : In-home display user setting screen

Chapter 8

Evaluation and Discussion

In this chapter, we will evaluate and discuss the result of our developed software, which is supporting software and in-home display, including the DLMS/CODEM encoder/decoder and sound modulation module.

8.1 Supporting Software

The supporting software is developed by composing the varied modules and components together. We also use an interface class for de-clumping the components and can develop each component separately and easily to replace the new component without any effect on the whole software.

We need to work with the smart meter software, which are outside our project, and not yet finish. It also makes our software cannot finish completely. So we can finish and test only the part the smart meter software already finished and the part that not relate to the smart meter, which are:

1. GUI of the supporting software
2. Database
3. DLMS/COSEM encoder/decoder
4. Real-time meter measurement value reading
5. Serial port communication module

8.2 DLMS/COSEM Encoder/Decoder

We developed our DLMS/COSEM encoder/decoder using the function to parse the parse tree, which is the wrong way to making the parser, so it makes the task become the most difficult and time consuming task in the part of supporting software.

However our encoder and decoder have been tested in the most of needed testing case and it is a good decision to use Dictionary data type as the other end of the parse tree, instead of create a whole new class, because it easy to understand by human and the supporting software can process the message easily.

8.3 Sound Modulation

In this task, we spend so much time in studying the FSK modulation technique and handle the error from modulation, but we still face a problem of the module.

8.3.1 Modulation (Encode)

We face a lot of error of modulation sound, when we modulate and send a long message to the Hi-jack. After we investigate we can assume that the problem come from the voltage different in the Hi-jack hardware.

8.3.2 Demodulation (Decode)

We face a better result on the demodulation process than modulation. The error of demodulating the message receive from the Hi-jack is very slim. However, when we use this module in the application we already have error checking process to handle it.

8.4 Metering Application under FSK techniques

After the integration between the background project and the main metering application is successfully, the application can receive the DLMS/COSEM message from the Smart Meter as the real-time message in the white box, and also has the Label part as the counter for counting the receiving message.

Chapter 9

Conclusions

9.1 Conclusions

The task of developing supporting software needs to work parallel with the smart meter development process so it cannot be tested and finished completely, if the development of the smart meter hardware and software is not finished. So our supporting software will have the further development after this project.

The DLMS/COSEM encoder and decoder are completely finish in the part of interpret between the DLMS/COSEM message and the Python message (Dictionary object). The further development on this part of project can be study of the encryption DLMS/COSEM message and algorithm, the alarm message handling and the communication to multiple meters at the same time.

The in-home display software and FSK soft-modem for Hi-jack are not the easy tasks because the application could not directly receive data from the smart meter. The advantage of the Hi-jack and the FSK sound modulation and demodulation is the most significant techniques that help the implemented application works. In addition, in application which is running on the user smart phone device also the hard work part of the project.

9.2 Lessons learned

9.2.1 DLMS/COSEM encoder/decoder

1. Python can handle very a large integer by automatically convert into a long integer that have unlimited. An integer, which becomes a long integer will presented with 'L' as its postfix like, `125487L`, `0x4578abbd574L` or `0b1010011110000111L`.

2. There is only 4 numeric types in python, which are integer, long integer, floating point, and complex number. It is difficult to use when the program needed to use or show on hexadecimal or binary format. It's provide a build-in function called **`hex()`** and **`bin()`** but it just return the strings of value on that format.

3. It is better to use ply (Python-Lex-Yacc) module for the encoder and decoder than use the plain recursive function as we have done.

9.2.2 Sound Modulation

1. Modulation

Generally the nature of existing wave, there are 2 kinds of modulation techniques which are FSK (frequency shift keyed) and PSK (performance of phase shift keyed). The performance of each techniques are different depends on relative possibility, the operating range and the bandwidth of the system.

2. Port Audio

Port Audio is one of the available open source library, the cross platform library are provide in Port Audio as well. Therefore, if any program or computer required to work with the audio data type, the advantage of port audio that supporting many OS (operating system) platform is one of the best selection.

9.3 Problems and Obstacles

9.3.1 COSEM Interpreter

1. It has very little example of the COSEM massages to work and test.
2. Encoder and decoder developing without ply (Python-Lex-Yacc) module is so difficult.
3. The smart meter hardware and software are not complete, which is make our testing so difficult.
4. It does not has floating point type in ASN.1 and COSEM needed to create them by itself from HEXSTR and makes it as difficult point for parsing.

9.3.2 Sound Modulation

There are the obstacles with the hardware issue that directly affected with the encoding and decoding process of the implemented software. Therefore, the receiver side of the communication in Smart Meter Network could not receive the accuracy of data 100%.

Bibliography

[1] PEA - Provincial Electricity Authority: Smart Meter TOU Document; Smart Meter.

[2] Ye-Sheng Kuo, SonalVerma, Thomas Schmid, and PrabalDutta (2010), "Hijacking Power and Bandwidth from the Mobile Phone's Audio Interface"Available from URL: http://web.eecs.umich.edu/~prabal/pubs/papers/kuo10hijack.pdf

[3] Parse Tree: Available from URL: http://en.wikipedia.org/wiki/Parse_tree

[4] D.K.Sharma, A. Misha& Rajiv Saxena (2010). "Analog and Digital Modulation Techniques". Available from URL: http://www.techniajournal.com/attachments/article/128/ANALOG%20&%20DIGITAL%20MODULATION.pdf

[4] Audio Jack modem for Iphone and Android: Available from URL: http://trac.switch-science.com/wiki/ARMS22-SOFTMODEM-HOWTO

[5] AurioTouch: Available from URL: https://developer.apple.com/library/prerelease/ios/samplecode/aurioTouch/Introduction/Intro.html

[6] COSEM PDU syntax: Available from URL: http://www.cyamon.com/Syntax/pdu_syntax.html

[7] DLMS User Association (2013) Technical Report White Book– COSEM – Glossary of Terms

[8] DLMS User Association (2013) Technical Report Green Book – COSEM – Architecture and Protocols

[9] DLMS User Association (2013) Technical Report Blue Book – COSEM – Identification System and Interface Classes

[10] Mark Lutz, O'reilly Media.Inc : Learning Python, Fifth Edition