Simulation of Standard Telecontrol Protocol
IEC 60870-5-101

Mudathir Mohammed Abdalla Sharif

Thesis submitted to University of Khartoum, the Faculty of Engineering and Architecture, Department of Electronics and Electrical, in partial fulfillment of the requirements for the degree of

Master of Science
In
Telecommunication and Information Systems

Supervisor: Dr. Iman Abuel Maaly Abdul Rahman

August 2004
ﺎﻴﺔ

.....artment أَلَّا أَنْ تَؤْلِفْ لَهَا غَيْرَ اللَّهِ وَلَا شَخْصٍ أَخَلَّاهَا !

..... رَفَّتْلَهَا إِلَيْهِ ّلَوْلاَ لِدَيْهِ مَا أَنْقَذَهَا !

..... لَكِنْ أَنْقُذْ إِلَيْهِ ّلَوْ أَوْحَى إِلَيْهِ ذِي الْكَيْمَةِ أَنْقُذَهَا !

..... أَدْتَفْ إِلَيْهِ ّلَوْ أَكَتْبَلْتُ لَهُ تَسْلِيمًا إِلَيْهِ أَنْقُذَهَا !

...... أَجَزَّى إِلَيْهِ ّلَوْ كَفَّرْتُ لَهُ رَحْمَةً إِلَيْهِ أَنْقُذَهَا !

..سمدح..
ACKNOWLEDGEMENT

Thanks to God for giving me the health and determination to complete this work.

I would like to express my sincere gratitude to my supervisor, Dr. Iman Abuel Maaly Abdel Rahman for her trust and help throughout the duration of this project. Her supervision and guidance were essential for the completion of this work.

Special thanks to the staff of the National Electricity Corporation for their help and guidance.

I appreciate the help of the staff of the Department of Electronics and Electrical Engineering at the University of Khartoum for their valuable forwarding and support.
لا يمكنني قراءة النص العربي المكتوب بالخط العربي.
ABSTRACT

The use of the telecontrol protocols helps developing of control systems. Technical and commercial competitions produce several protocols with different functions and specifications. This leads International Electrotechnical Commission to create standard protocol for telecontrol systems.

As a result of the development of communication systems, the International Electrotechnical Commission produced a series of telecontrol protocols IEC60870-5 that contains several protocols for different purposes. The IEC60870-5-101 is used to link two stations, primary and secondary stations, while the IEC60870-5-103 is used specially to transfer signals and data of protection equipments. The IEC60870-5-104 is used in telecontrol networks and can be easily connected to computer networks.

For this simulation, the IEC60870-5-101 was selected for its importance due to its widely spreading use among the telecontrol systems, and it is also considered as a base structure for modern telecontrol systems such as IEC60870-5-104.

The simulation was made by the Delphi programming language to link the primary and secondary stations using IEC60870-5-101 protocol. Measured values with floating point numbers have been transferred from a secondary station to a primary station. All the simulation levels for construction and extraction of telecontrol messages were implemented successfully.
## CONTENTS

Dedication ................................................................................................................................. i

Acknowledge ............................................................................................................................... ii

Abstract (Arabic) ......................................................................................................................... iii

Abstract ........................................................................................................................................ iv

Chapter One -- Introduction

1.1 Statement of the problem ....................................................................................................... 1

1.2 Subject significance ................................................................................................................ 1

1.3 Project objective ..................................................................................................................... 2

1.4 Approaches ............................................................................................................................ 2

1.5 Thesis layout .......................................................................................................................... 3

Chapter Two -- Theory of IEC60870-5-101 protocol

2.1 Introduction............................................................................................................................ 5

2.2 Reference model for communication networks ...................................................................... 7

2.3 Message structure .................................................................................................................. 9

2.3.1 Start frame ......................................................................................................................... 10

2.3.2 Data Unit Identifier .......................................................................................................... 16

2.3.3 Information Object ......................................................................................................... 20

2.4 Station initialization .............................................................................................................. 22

2.5 Application functions .......................................................................................................... 24

2.5.1 Station interrogation ......................................................................................................... 24

2.5.2 Cyclic data transmission ................................................................................................. 25

2.5.3 Background scan ............................................................................................................ 26

2.5.4 Acquisition of events ...................................................................................................... 26

2.5.5 Clock synchronization ..................................................................................................... 27
Chapter Three -- Simulation of a telecontrol outstation

3.1 Basis of the simulation
3.2 Main procedures
3.2.1 Transfer_click procedure
3.2.2 Primary_Request_Status proprocedure
3.2.4 Secondary_Receive procedure
3.2.4 Primary_Reset_Status procedure
3.2.5 Second_send_ASDU procedure
3.2.6 Prim_receive_ASDU procedure

Chapter Four -- Results

4.1 Program interfaces

Chapter Five – Conclusion and Comments

5.1 Conclusion
5.2 Comments

References

Appendix A – Program Code
Appendix B – Interoperability
Appendix C – Possibilities of the cause of transmission for the ASDUs
Appendix D – General structure of the ASDU message
Chapter One

Introduction to Standard Telecontrol Protocols
1.1 ) Statement of the problem
The increasing complexity of power networks, and its very high interconnection grade, causes a growing need of communication between the different Energy Management Systems (EMS). However, such equipments are commonly of different period, technology and vendor. In this situation, the communication among them becomes a serious problem. Many efforts have been accomplished to define a set of standardized protocols, for both the center-remote communication and for the link between centers of the same different levels.

The standard permits the user of the telecontrol installation to specify/choose his own system strategy for using the protocol provisions in ways that solve his system problems and adhere to his economic, practical and technical constraints.

1.2 ) Subject significance
Without standards it is almost impossible for utilities to choose several suppliers within one IT infrastructure. When using a proprietary protocol, a lot of custom-made, time-consuming and expensive solutions – gateways or interface modules - are usually necessary to build up a multi-vendor communication environment. Therefore, the major reasons for setting up protocol communication standards are:

- Optimisation of cost per unit.
- Risk management
- Flexibility.
The users are the ones who benefit most from standardization, but suppliers also see that standardization is beneficial to them. For example if we consider a GSM mobile phone and assume that every country within one continent decided to use its own GSM standard, then the suppliers have to manufacture a lot of GSM phones when exporting. On the other side, the user will need another phone or configuration for every country, and he will bear the costs of all extra efforts. The same situation, just as every standardization issue, applies to communication protocols.

1.3 ) Project objective
The aim of this project is to study the standard telecontrol protocol IEC60870-5-101, its structure, functions and applications. This study will be used to simulate easy helpful system that operates with the standard IEC60870-5-101 protocol.

In this simulation, a primary station and a secondary station will be connected by using IEC60870-5-101 protocol. Measured values with floating point number will be transferred from secondary station to primary station. The simulation will contain several levels for construction and extraction of the telecontrol messages.

1.4 ) Approaches
For implementation, the simulation was made using Borland Delphi programming language version 7.0, which run under windows. This
language was created from Pascal programming language that was one of the high level languages applied for engineering and mathematical applications.

Delphi programming language has many tools for debugging, errors detection and manipulation, which made the follow of the program processes very simple. Evaluate/modify, watches and breakpoints represent most important tools that deal with program variables, steps and procedures. The breakpoint pause the program in a certain line to allow programmer applies debugging tools. Watches displayed values of selected variables when the program had been paused by one of the breakpoints. Using evaluate/modify tool permits programmer to set any new value for variables to be processed after the program has been running again.

1.5 ) Thesis layout

Through this project a lot of work done to introduce and simulate the operation of standard telecontrol protocol IEC60870-5-101.

This thesis was structured in several parts, each part discuss and demonstrate many details associate with the standard telecontrol protocol IEC60870-5-101.

Firstly, chapter one states the problem and the project purpose which answered many questions such as why using the telecontrol protocol.
In chapter two, an introduction to the telecontrol protocols gives more detail and theory behind the protocol. Also it discusses the standard of the protocol and its development.

In the third chapter, an attempt to develop software of telecontrol protocol was executed. Using the efficient programming language called Delphi programming language to program this software. The objective of this software is to simulate the operation of the telecontrol protocol. The simulation was done by taking one line feeder as an example for the experiment. For this feeder, reading of currents, voltages, active and reactive powers are applied to the software. A result was produced and a corrected action was made.

The software consist of many procedures, each procedure was responsible from a certain task. The success of all procedures gave the proper result and lead to the right decision. Chapter four shows the program interface menus and the produced results.

Lastly in chapter five, strong evaluation and interpretation was done to get out with many ideas. Also details conclusion was made to determine the main advantages of these experimental efforts. And much recommendation was made to make useful utilization of this software in the real application.
Chapter Two

Theory of IEC60870-5-101 protocol
2.1 ) Introduction
The communication protocol is a small but important part of ‘utilities’ system that controls and monitors its core activities. It is uses for information exchange, monitoring and controlling functions.

The IEC Technical Committee 57 (TC57) have developed a protocol standard for telecontrol, teleprotection, and associated telecommunications for electric power systems. The result of this work is IEC 870-5 series, which is one of the standard telecontrol protocols that used in the 90s and still being updated regularly according to new technologies and industry requirements. There are many version of the IEC 60870-5 standard. The serial versions telecontrol scheme TCS101 and teleprotection scheme PCS103 are mostly used all over the world, while the telecontrol scheme TCS104, which was published by the end of 2000, is based on TCS101 and uses broadband (TCP/IP) technology. A lot of utilities have started already or will start to migrate from proprietary protocols to the serial TCS101/PCS103 or the TCS104. An applicable example of IEC TC57 protocol standard is shown in Figure (2.1). There are different protocols control communications between different substations equipments.
The 60870-5 protocol series is based on technology with limited performance and capacity. The costs of the infrastructure and hardware, e.g. processors and memory, have decreased tremendously and pose no restriction at the moment in the consideration to add intelligent features and functionality. The cost reduction has given the equipment more functionality and the opportunity to distribute the intelligence of the telemetry system. Substation automation systems using IEC 60870-5 series with built-in intelligence are the result. Therefore, these devices provide for a lot of utilities a reasonable and
sufficient solution with proven technology and optimized price/quality ratio.

2.2) Reference Model for communication networks

The open system interconnected (OSI) reference model is a layered set of protocols to facilitate open communications between computer networks. It is the basic reference model that divides a protocol into seven layers (Application, Presentation, Session, Transport, Network, Data link and physical layers). The top three layers are directly concerned with the actual application messages being sent between stations. The bottom four layers are concerned with the method used to transport those messages between stations.

The simplified reference model used in the IEC 60870–5–101 standard (and several other protocol standards) has fewer layers, because some of the facilities supported by the full seven layer model are not required and enhanced working of the remaining facilities is desired hence the model is often called the enhanced performance architecture (EPA) Model.

Two stations, shown in Figure (2.2), are communicating together by using the EPA model. Each station has a “stack” of protocol layers providing communication services to the station application processes at top and accessing the communications medium at the bottom.
Application data is accepted at the top of the protocol stack in one station and passes down through the stack, acquiring in each layer any necessary extra data needed to control the working of the protocol until it emerges in serial form at the bottom. It is then transmitted to the other station where it enters at the bottom of protocol stack. The data passes up this stack having the control data stripped off layer by layer until the original application data emerges at the top and is passed to the application processes in destination station.

This is called “peer to peer“ communication because all data originating in a particular layer is transported to the same layer in the remote station.
The layer interfaces shown in the protocol stack do not have to be presented in a real station. However the physical interface is usually present because the physical layer is often implemented using a separate Modem. The link Interface needs to be present because it is desired to implement the application layer software and the link layer software separately.¹

2.3) Message structure
Serial messages have a nested structure, which derives from the layered structure of the protocol. The ASDU (Application service Data unit) is a block of data being sent from the application processes in another station. ASDU is specified as frames with variable length.

Frame with variable length used in IEC 870-5-101 start with
- One octet Start Character.
- Two octets Frame Length.
- One octet Start Character.
- One octet Control Field.
- One octet Link Address.

And stop with
- One octet Checksum.
- One octet Stop Character.

¹ GBC Report 011-2
The ASDU is composed of a Data Unit Identifier and one or more Information Objects. The Data Unit Identifier has always the same structure for all ASDUs. The Information Objects of an ASDU are always of the same structure and type, which are defined in the type identification field.

The structure of the data unit identifier is:

- One octet Type Identification.
- One octet Variable Structure Qualifier.
- One/two octets Cause of Transmission.
- One/two octets Common Address of ASDU.

The structure of the information object is:

- Information object identifier.
- Set of information elements.
- Time tag of information object (optional).

General structure of the ASDU message is shown in appendix d.

2.3.1) start frame

Start frame is the first frame in the message and consists of the following fields.
a) Start / stop character
Start and stop characters have always the same structure for all ASDU messages. Each character has a fixed defined bit pattern, which holds one octet as shown in Figure (2.3).

<table>
<thead>
<tr>
<th>Bit</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$2^7$</td>
<td>$2^6$</td>
<td>$2^5$</td>
<td>$2^4$</td>
<td>$2^3$</td>
<td>$2^2$</td>
<td>$2^1$</td>
<td>$2^0$</td>
</tr>
</tbody>
</table>

Figure (2.3): Start /stop character format.

b) Length character
Length character specifies the number and subsequent user data octets including the control and address fields. It has a range up to 255 octets, which must be a parameter in the controlled station –see Figure (2.4) –. The two octets contain the same value of the length number.

<table>
<thead>
<tr>
<th>Bit</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$2^7$</td>
<td>$2^6$</td>
<td>$2^5$</td>
<td>$2^4$</td>
<td>$2^3$</td>
<td>$2^2$</td>
<td>$2^1$</td>
<td>$2^0$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$2^7$</td>
<td>$2^6$</td>
<td>$2^5$</td>
<td>$2^4$</td>
<td>$2^3$</td>
<td>$2^2$</td>
<td>$2^1$</td>
<td>$2^0$</td>
</tr>
</tbody>
</table>

Figure (2.4): Length character field, Range: 0 → 255
c) Control field
The control field contains information that characterizes the direction of the message, the type of service provided and supports control functions for suppressing losses or duplications of messages.

The control field is different for balanced and unbalanced mode. The unbalanced transmission procedures are used in supervisory control and data acquisition systems (SCADA systems) in which a master station controls the data traffic by polling outstation sequentially. In this case the master station is the primary station that initiates all message transfers while the outstations are secondary stations that may transmit only when they are polled. The control field character for unbalanced mode is shown in Figure (2.5).

\[
\begin{array}{cccccccc}
\text{Bit} & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \\
\text{RES} & \text{PRM} & \text{FCB} & \text{FCV} & 2^3 & 2^2 & 2^1 & 2^0 \\
\text{ACD} & \text{DFC} & \text{Function} \\
\end{array}
\]

Figure (2.5): Control field for unbalanced mode.

Function codes of the control field in unbalanced mode are illustrated in Table (2.1).
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>PRM</td>
<td>Primary message</td>
<td>0 = message from secondary (responding) station, 1 = message from primary (initiating) station.</td>
</tr>
<tr>
<td>FCB</td>
<td>Frame count bit. 0-1 = alternating bit for successive send/confirm or request/respond service per station. The frame count bit is used to delete losses and duplications of information transfers.</td>
<td></td>
</tr>
<tr>
<td>FCV</td>
<td>Frame count valid</td>
<td>0 = alternating function FCB bit is invalid, 1 = alternating function of FCB bit is valid. Some messages and services that ignore the deletion of duplication or less of information output do not alternate the FCB bit and indicates this by a cleared FCV bit.</td>
</tr>
<tr>
<td>DFC</td>
<td>Data flow control</td>
<td>0 = further message are acceptable, 1 = further message may cause data overflow.</td>
</tr>
<tr>
<td>ACD</td>
<td>Access demand</td>
<td>0 = no access demand for class 1 data transmission (used for events or for high priority messages), 1 = access demand for class 1 data transmission.</td>
</tr>
</tbody>
</table>

Table (2.1): Function codes of the control field for unbalanced mode.
When balanced transmission procedures are used each station may initiate message transfers. In this mode all stations are called combined station because they may act simultaneously as primary and secondary stations. The control field character for balanced mode is shown in Figure (2.6).

<table>
<thead>
<tr>
<th>Bit</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIR</td>
<td>PRM</td>
<td>FCB</td>
<td>FCV</td>
<td>$2^3$</td>
<td>$2^2$</td>
<td>$2^1$</td>
<td>$2^0$</td>
<td></td>
</tr>
<tr>
<td>RES</td>
<td>DFC</td>
<td>Function</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure (2.6): Control field for balanced mode.

Function codes of the control field in unbalanced mode are illustrated in Table (2.2).
| DIR  | Physical transmission direction
      | DIR=1 Data from controlling to controlled station
      | DIR=0 Data from controlled to controlling station |
|------|---------------------------------------------------------------------------------------------|
| PRM  | Primary message
      | 0 = message from secondary (responding) station
      | 1 = message from primary (initiating) station. |
| FCB  | Frame count bit. 0-1 = alternating bit for successive send/confirm or request/respond service per station
      | The frame count bit is used to delete losses and duplications of information transfers. |
| FCV  | Frame count valid
      | 0 = alternating function FCB bit is invalid.
      | 1 = alternating function of FCB bit is valid.
      | Some messages and services that ignore the deletion of duplication or less of information output do not alternate the FCB bit and indicates this by a cleared FCV bit. |
| DFC  | Data flow control
      | 0 = further message are acceptable.
      | 1 = further message may cause data overflow. |
| RES  | Reserved |

Table (2.2): Function codes of the control field for balanced mode.
The following transmission services, initiated by the primary station, are supported by the link in unbalanced mode.

- Send / reply: mainly used for global messages and for cyclic set points in control loops.
- Send / confirm: mainly used for control commands and set points commands
- Request / respond: used for polling sequences and may be used for cyclic updating functions.

d) Link address
Link transmission procedures use control field and the optional address field, and could be either one or two octets, or none, producing an address range up to 65536 addresses.

2.3.2 ) Data Unit Identifier
It is a part of the ASDU that consists of two octets of data unit type, one or two octets represented cause of transmission and other one or two octets represented common address of ASDU. The data unit type is composed of one octet as type identification and the other represents variable structure qualifier.

a) Type identification
The type identification defines structure type and format of the information objects. This means information objects with or without time tags are distinguished with different numbers of the
identification. Undefined values of the type identification for the ASDUs are acknowledged negatively and discarded by both controlling and controlled stations.

Table (2.3) illustrates function codes of the type identification.

<table>
<thead>
<tr>
<th>TYPE IDENTIFICATION</th>
<th>Range &lt;1..255&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;1..127&gt;</td>
<td>Used for standard definitions from IEC 870-5-101</td>
</tr>
<tr>
<td>&lt;128..135&gt;</td>
<td>Reserved for routing of messages (private range)</td>
</tr>
<tr>
<td>&lt;136..255&gt;</td>
<td>Used for special use (private range)</td>
</tr>
</tbody>
</table>

Table (2.3): Function codes of the type identification.

b) Variable structure qualifier.

The variable structure qualifier is represented by eight bits as shown in Figure (2.7). The most significant bit (SQ bit) is used to define single object or sequence of elements as illustrated in Table (2.4). The remaining seven bits defines the number of the information objects in the ASDU.

<table>
<thead>
<tr>
<th>Bit</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SQ</td>
<td>$2^6$</td>
<td>$2^5$</td>
<td>$2^4$</td>
<td>$2^3$</td>
<td>$2^2$</td>
<td>$2^1$</td>
<td>$2^0$</td>
</tr>
</tbody>
</table>

Figure (2.7): Variable Structure Qualifier field.
Table (2.4): Function codes of the Variable Structure Qualifier.

c) Cause of transmission

The cause of transmission directs the ASDU to a specific application task for processing. It is constructed from one or two octets as shown in Figure (2.8).

<table>
<thead>
<tr>
<th>Bit</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>P/N</td>
<td>Cause</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Originator Address (optional)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure (2.8): Cause of transmission field.
The P/N bit indicates the positive or negative confirmation of activation requested by the primary application function. The test bit defines ASDUs, which were generated during test condition and it is used to test transmission and equipment without controlling the process. Table (2.5) describes the function codes of the cause of transmission field.

<table>
<thead>
<tr>
<th>Cause of transmission</th>
<th>{SQ, P/N, Cause, Originator Address}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cause</td>
<td>Cause = 0 ; not defined.</td>
</tr>
<tr>
<td>Range &lt;0 – 63&gt;</td>
<td>Cause = &lt;1 .. 63&gt; ; number of cause.</td>
</tr>
<tr>
<td>P/N bit</td>
<td>P/N = 0 ; positive confirm.</td>
</tr>
<tr>
<td></td>
<td>P/N = 1 ; negative confirm.</td>
</tr>
<tr>
<td>T bit</td>
<td>T = 0 ; no test.</td>
</tr>
<tr>
<td></td>
<td>T = 1 ; test.</td>
</tr>
<tr>
<td>Originator Address</td>
<td>Originator Add. = 0 ; default.</td>
</tr>
<tr>
<td>Range &lt;0 – 225&gt;</td>
<td>Originator Add. = &lt;1..255&gt; ; number of</td>
</tr>
<tr>
<td></td>
<td>originator address.</td>
</tr>
</tbody>
</table>

Table (2.5): Function codes of the Cause of Transmission.

Possibilities of the cause of transmission for the ASDUs are illustrated in Appendix C.
d) **Common Address of ASDUs.**

The common address of the ASDU defines the station address and has a length, which is fixed per system (one or two octets). Function codes of the common address for both one and two octets modes are described in Table (2.6).

<table>
<thead>
<tr>
<th>Function</th>
<th>Common Address</th>
<th>Common Address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>One octet</td>
<td>Two octets</td>
</tr>
<tr>
<td></td>
<td>Range &lt;0 – 255&gt;</td>
<td>Range &lt;0 – 65535&gt;</td>
</tr>
<tr>
<td>Not used</td>
<td>Common Address = 0</td>
<td>Common Address = 0</td>
</tr>
<tr>
<td>Station address</td>
<td>Common Address = &lt;1..254&gt;</td>
<td>Common Address = &lt;1..65534&gt;</td>
</tr>
<tr>
<td>Global address</td>
<td>Common Address = &lt;255&gt;</td>
<td>Common Address = &lt;65535&gt;</td>
</tr>
</tbody>
</table>

Table (2.6): Function codes of the Common Address.

2.3.3 ) **Information Object**

The information object consists of information object identifier, set of information elements and time tag of object. The information object identifier consists only of (one, two or three octets) of information object address, which used as destination address in control direction and as source address in monitor direction. The zero number in all types (one, two and three octets), reflects that the information object address is irrelevant
The second field of information object is a set of information elements, which represents the transmitted data. There are different types of data depend on the required application and protocol used. For example command signals (tests and reset process), measured values (current, voltage, power, and frequency) and communication signals (initiation and end of initiation).

Structure for the information element of measured value type with short floating point number is shown in Figure (2.9). A Quality descriptor is added to fraction and exponent to construct the information element.

<table>
<thead>
<tr>
<th>Bit</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st octet</td>
<td>Fraction</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd octet</td>
<td>Fraction</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3rd octet</td>
<td>Fraction</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4th octet</td>
<td>Exponent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IV</td>
<td>NT</td>
<td>SB</td>
<td>BL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>OV</td>
<td></td>
</tr>
</tbody>
</table>

Figure (2.9): Structure of the measured value with short floating point number.

Function codes of the measured value with short floating point number are illustrated in Table (2.7).
### Chapter 2  Theory of IEC60870-5-101 protocol

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Bit = 0</th>
<th>Bit = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Sign</td>
<td>Positive</td>
<td>Negative</td>
</tr>
<tr>
<td>IV</td>
<td>Invalid / Valid</td>
<td>Valid</td>
<td>Invalid</td>
</tr>
<tr>
<td>NT</td>
<td>Not Topical / Topical</td>
<td>Topical</td>
<td>Not Topical</td>
</tr>
<tr>
<td>SB</td>
<td>Substituted / not Substituted</td>
<td>Not substituted</td>
<td>Substituted</td>
</tr>
<tr>
<td>BL</td>
<td>Blocked / Not Blocked</td>
<td>Not Blocked</td>
<td>Blocked</td>
</tr>
<tr>
<td>OV</td>
<td>Overflow / Overflow</td>
<td>No Overflow</td>
<td>Overflow</td>
</tr>
<tr>
<td>E</td>
<td>Reserved</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>

Table (2.7): Function codes of the measured value with short floating point number.

### 2.4) Station initialization

Next initiation procedure represents local initialization of the controlling station in unbalanced transmission systems. The initialization of the controlling station starts e.g. with power off/on. The controlling station couldn’t receive any data requested right before the initialization, because it is no longer available.

The link of the controlling station then establishes connection with the link of the controlled station by transmitting a "Request status of link" that is answered by a "Status of link" response from the controlled station. The controlling station then transmits a "Reset of remote link" that is answered by an "ACK", which confirms the start condition of the link layer of the controlled station.
After the initialization the controlling station is updated by issuing a general interrogation command to the controlled station. If appropriate, the time of the two stations is then synchronized by a clock synchronization command. The sequential procedure for local initialization of the controlling station is shown in Figure (2.10).

<table>
<thead>
<tr>
<th>Application function of Controlling Station.</th>
<th>Communication services.</th>
<th>Application function of Controlled Station.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start of local initiation.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power on.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Link is available.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Request status of the link.</td>
<td>PRM = 1, FC = 9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Status of link.</td>
<td></td>
</tr>
<tr>
<td>Reset of remote link.</td>
<td>PRM = 0, FC = 11</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Acknowledge, link is rest.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PRM = 1, FC = 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PRM = 0, FC = 0</td>
<td></td>
</tr>
<tr>
<td>Link connection established.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Controlling station initialized.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure (2.10): Local initialization of the controlling station.
The link telegrams that are used are fixed frame telegrams with Primary Message bit (PRM) and Function Code (FC) as indicated.\textsuperscript{1}

2.5 ) Application functions

The following Application Functions and associated ASDUs are supported by the telecontrol protocol IEC 870/101 services.

- Station interrogation.
- Cyclic data transmission.
- Background scan.
- Acquisition of Events.
- Clock Synchronization.
- Command transmission.
- Reverse direction.

The following sections give description of these functions.

2.5.1 ) Station interrogation.

After the central controlling station and the various controlled stations (outstations) have been initialized (made active). It is necessary for the Controlling station to obtain an image of the present states of all those digital (on/off) inputs, which are normally reported spontaneously and all the values of the analogue (process variable) inputs, which are similarly reported.

\textsuperscript{1} Norwegian IEC 870-5-1 User Convention, chapter six ‘communication procedure’.
To do this, a Station Interrogation activation command may be sent to all outstations, requesting that they return all their designated station interrogation data as soon as possible. When these data have been acquired, the Controlling station may construct an image of all the inputs to the installation, which are normally reported spontaneously. This image is then used as the starting point for future operations.

If, at a later time, communication with a particular outstation is lost and then restored (after whatever period), a station Interrogation may be used to obtain a static update of the part of the network image belonging to just that outstation.

2.5.2 ) Cyclic data transmission
This function may be used to continuously update another central image with information data obtained from measurements taken at regular intervals of time. It is often used for monitoring the less vital inputs to an installation; that is those, which are slow moving or do not require fast action at the Controlling station. For example the temperature of a piece of process equipment (such as a transformer) might be monitored using cyclic data transmission.
2.5.3) **Background scan**

This function (which is a slow cyclic scan) may be used to ensure that the network image values are up to date and have not been falsified by the undetected loss of spontaneously reported events since the last scan cycle.

2.5.4) **Acquisition of events.**

Once the network image has been constructed, using the static data obtained from Station Interrogation, the image must be kept up to date with dynamic data obtained by the Acquisition of Events function when changes take place.

This dynamic data is usually vital input information, which is likely to require quick action at the Controlling station. For example if an important switch, within the process equipment of the electric power system, changes state: then the outstation Application layer presents a Request ASDU to its Link layer.

In due course the Link layer of the central Controlling station will generate an Indication to its Application layer, presenting the received ASDU. The ASDU will be marked with Cause of Transmission = Spontaneous, which prompts the Application layer immediately to update the appropriate point in the network image. As a result other urgent operations may take place, for example an Alert may be signaled.
Optional double transmission of spontaneous events is permitted when desired. The first transmission is as described above. A second lower priority transmission of the same events, with an added Time Tag, may be used to construct a central event record. This may be analyzed to determine the exact sequence in which events occurred.

2.5.5 ) Clock synchronization.

Accurate Clock Synchronization in an outstation depends on knowing the time taken to transmit a telecontrol message to it from the central Controlling station containing the master clock. This permits an allowance to be made for the transmission time during synchronization. When using the Internet, there is a variable delay before a transmitted message arrives at an outstation. This causes a time uncertainty that depends on the maximum transmission time offered by the Internet service provider.

The resulting clock synchronization setting is unlikely to be more accurate than within one second. Thus individual radio clocks, or other accurate clocks, will be required at each outstation when time tags having a better accuracy than this are required.

2.5.6 ) Command transmission.

ASDUs containing Commands are sent from the Controlling station to the outstations when required. There are two main kinds of Commands: those for process information and those for system information.
Process information commands include those for changing the state of single digital (on/off) outputs, step (raise/lower) outputs and set point (analogue) outputs. Two modes of operation are provided: Direct (immediate) execution and Select/Execute, where the selection ASDU is confirmed back to the Controlling station before the actual execution ASDU is sent.

System information commands include Station Interrogation command, Counter Interrogation command, Clock Synchronization command, all of which are confirmed back to the Controlling station.

2.5.7 ) Reverse direction

In standard operation, Commands are sent from the Controlling station to a Controlled station and Events/Measurements are sent from a Controlled station to the Controlling station.

In some installations it may be advantageous to have dual mode operation between two equal stations, so that both Commands and Events/Measurements may be sent in both directions. This may be done using “Standard operation” and “Reverse operation” via a common Link layer. Individual Application functions and associated ASDUs may be chosen for Standard operational use, Reverse operational use or for both uses.
2.6) **Example of the use of IEC 60870-5-101 protocol.**
The IEC 60870-5-101 standard defines a communication protocol for telecontrol installations. It deliberately does not standardize other aspects of such installations. Figure (2.11) shows an example of a telecontrol outstation being used to control and monitor an electric power substation. The substation may or may not have some substation automation equipment to provide a degree of local autonomous control and internal intercommunication between items of plant equipment.

The protocol standard offers appropriate application functions and ASDUs to enable the communication of all the input/output data needed for this example.

![Diagram](image.png)

Figure (2.11): Example of a telecontrol outstation.
Chapter Three

Simulation of a Telecontrol Outstation
3.1 ) Basis of the simulation
The program simulates transferring of data from secondary station to primary station using the concepts of IEC870/101 protocol. The primary station represents a load dispatch center, which monitors and controls power flow into several substations. The secondary station represents one of the substations. The substation contains several line feeders, power transformers, bus bar feeder, control devices and protection equipments.

The primary station sends controlling commands (open, close, step up, step down, reset) to the secondary station, while the later sends status signals (indications, alarms, measurements) to the former for monitoring.

Only one line feeder (Line 1) was selected by the program to apply the use of telecontrol protocol. But the program reflects the ability to increase number of feeders and other equipment in the substation.

With Line 1 measured values of current, voltage, active power and reactive power were selected to be information objects of the application service data unit (ASDU). In addition, type of measured value with floating point number was used to permit fractions.

Appendix B shows a set of parameters and alternatives that are used by the simulation program.
3.2 ) Main procedures
In the following section represent the main procedures are presented.

3.2.1 ) Transfer_click procedure
This procedure represents main program, which contains initiation procedures in addition to application service data unit (ASDU) generation, sending and receiving procedures.

The initiation was made by using the following procedures:-
- Primary_Request_Status.
- Secondary_Receive.
- Primary_Respond_Status.

The “Generate_Reading” procedure generates the information elements for a certain feeder (Line 1). Values of current and voltage were set to certain starting values. Variation was applied for these starting values after a small period of time and values of active and reactive power have been calculated for each period.

“Second_send_ASDU” and “Prim_receive_ASDU” procedures performed sending and receiving respectively the frame of variable length containing ASDU. The flowchart of the “Transfer_click” procedure is shown in Figure (3.1).
START

Primary_Request_Status procedure

Secondary_Receive procedure

Primary_Request_Status procedure

Secondary_Receive procedure

Display ‘Initiation protocol transmission’

Generates started values of current & voltage.

t = 0

Generate_Reading procedure of Line 1

Second_send_ASDU procedure

Prim_receive_ASDU procedure

Increase t

Delay

Is t = 5 ?

No

Yes

END

Figure (3.1): Flow chart of Transfer click procedure.
3.2.2 ) Primary_Request_Status procedure
In this procedure the primary station generated a frame of fixed length, which represents a message known as request status of the link. Start, link address, control field, checksum and end characters were generated to construct the frame.

The control field character of this message takes binary byte (0.1.0.0.1.0.0.1), which was represented by decimal number “73”. The flowchart of this procedure is shown in Figure (3.2).
Chapter 3

Simulation of a telecontrol outstation

Figure (3.2): Flow chart of Primary_Request_Status procedure.
3.2.3) Secondary_Receive procedure
The secondary station checked a received frame in the input/output buffer and replied with relative message. For minimization, two messages have been defined in this procedure, the "request status of the link" and the "reset of the remote link" messages. The request status of the link message (with control field of decimal 73) will be replied by the respond status of the link message (with control field of decimal 11). The reset of the remote link message (with control field of decimal 64) will be replied by the acknowledge message (with control field of decimal 0).

Figures (3.3 part a & b) show flowchart of Secondary_Receive procedure.
START

Check one byte.

Start character found.

Check next byte.
- Control field -

Is control field = 73

Check next byte.
- Link Address -

Correct Link address.

Check next byte.
- Checksum-

(2)

(1)

Error procedure.

End main program.

100 bytes checked.

Error procedure.

End main program.

Is control field = 64

Correct Link address.

Check next byte.
- Link Address -

End main program.

Is control field = 73

End main program.

Figure (3.3.a): Flowchart of Secondary_Receive procedure, part a.
Compare number in Checksum field with the sum number of ones in the received message.

Error in received frame

Yes

Display ‘Secondary acknowledge’.

Construct acknowledge message format.

Filling input/output buffer with the message.

No

End procedure.

End main program.

Compare number in Checksum field with the sum number of ones in the received message.

Error in received frame

Yes

Display ‘Secondary respond status of the link’

Construct respond status of the link message

No

Filling input/output buffer with the message.

End procedure.

Figure (3.3.b): Flowchart of Secondary_Receive procedure, part b.
3.2.4) Primary_Reset_Status procedure
In this procedure when respond status of the link had been received
the primary station generated a frame of fixed length that represent a
message known as reset of the remote link.

Start, link address, control field, checksum and end characters were
generated to construct the frame.

The control field character of this message takes binary byte
(0.1.0.0.0.0.0.0), which was represented by decimal number “64”.

Flowchart of this procedure is shown in Figure (3.4).
Chapter 3  Simulation of a telecontrol outstation

START

Check one byte.

Check next byte.
- Control field -

Check next byte.
- Link Address -

Check next byte.
- Checksum -

Error procedure.

End main program.

Figure (3.4.a): Flow chart of Primary_Reset_Status procedure.
Compare number in Checksum field with the sum number of ones in the received frame.

Yes

Error in received frame

No

Display ‘Primary reset of the remote link’.

Construct ‘Reset of remote link’ message format.

Filling input/output buffer with the message.

End procedure.

Error procedure.

End main program.

Figure (3.4.b): Flow chart of Primary_Reset_Status procedure.
3.2.5) **Second_send_ASDU procedure**

In this procedure, the secondary station generated a variable length frame. This frame was built from start and stop frames in addition to the application service data unit (ASDU) of the feeder (Line 1). Flowchart for the above procedure is shown in Figure (3.5).

![Flowchart](image)

**Figure (3.5): Flow chart of Secondary_Send_ASDU procedure.**
Length character in the start frame was set to 30, which specified the number and subsequent user data octets including the control and address fields.

Control field was set to decimal “8” that represent a message sent from secondary contained user data. Link address was set to “4” from the used specifications.

The ASDU frame started with type identification, which was set to “13” to refer to a message of measured value, short floating number type.

The cause of transmission character was set to “2” that referred to background scan, which used in monitor direction to synchronize the process information of the controlling and controlled stations in law priority continuous basis.

Variable structure qualifier was set to “4” as a total number of information elements in the ASDU frame (current, voltage, active and reactive power).

Common ASDU address and all addresses of information elements were set according to the used specifications.
For simplicity, fraction of measured value was represented by one character. This estimation had no effect since fraction will be rounded to two digits only. In addition, all functions of quality descriptor were selected in order to set bits to zero.

By calculating the total number of ones, the checksum character will be generated. This character was added to the stop character to construct the stop frame. Finally the generated variable length frame would fill the input/output buffer.

3.2.6 Prim_receive_ASDU procedure
In this procedure, the primary station checked all bytes – one by one – of the received message. If there were no errors, compared with the used specifications, the primary station determined and displayed the measured values. Flowchart is shown in Figures (3.6.a & 3.6.b).
Figure (3.6.a): Flow chart of Primary_Receive_ASDU procedure. (part a)
Get measured values from information objects.

Check next byte / checksum

Error in received frame

Error procedure.

End main program.

No

Display measured values.

End procedure.

Figure (3.6.b): Flow chart of Primary_Receive_ASDU procedure. (part b)
Chapter Four

Results of the program simulation
4.1) Program interfaces

When the program was run a menu of name specifications appeared, which contains some of the most important parameters (addresses, size of fields) as shown in Figure (4.1). All the addresses were selected randomly because of the large range produced from using only one feeder from one secondary station. This also permits the selection size of the optional fields to be as small as possible (one byte for each).

![Figure (4.1): User Interface of the simulation program.](image)
When the user moves to the second menu of the program the main menu appears as shown in Figure (4.2). The user can select one of the buttons (Start, Stop, Specifications). The Specification button permits the program user to display the specifications menu.

![Simulation of Standard Telecontrol Protocol IEC60870-5-101](image)

**Figure (4.2): Main program menu**
By pressing the Start button, the simulation of data transfer between the primary and secondary stations will start. When the initiation procedure succeeds, the secondary station constructs the IEC870/101 message frame. The measurement values (Current, Voltage, Active and Reactive powers) represent the information elements of the IEC870/101 message. This message will be transmitted to the primary station that detects it to extract the measurement values. Figure (4.3) shows measurement values that have been generated by the secondary station and extracted by the primary station.

Figure (4.3): Displaying of Measurements values after transmission.
The Stop button disconnects the communication link between the two stations and terminates the running program. Figure (4.4) shows program termination by using the Stop button.

Figure (4.4): Program termination.
Chapter Five

Conclusion and Comments
5.1) Conclusion
The telecontrol protocol IEC60870/101 is very important protocol for communication in power system networks. It is one of the standard protocols that widely used in power control and automation systems. The main usage of this protocol is to carry control signals (commands, measurements, status and alarms) between devices. This protocol represents a base for the standard telecontrol protocol IEC60870/104, which is established on TCP/IP technology and very helpful for communications in complex power system networks and can be easily interfaced with the computer networks.

5.2) Comments
The main objective of this research is to study this protocol in details to get knowledge about the operation, configuration and implementation of the protocol framework. The second objective is to simulate part of the protocol functions, such as transferring measurement values with floating point number.

This research covered all the theoretical aspects of the IEC60870/101 protocol. The simulation program is implemented and covered part of the protocol functions, such as initiation procedure and transferring measurement values with floating point numbers.
The program was made in such a way that any addition of different signals (measured values or alarms) are possible, because of the idea of IEC870/101 protocol in arranging different types of signals. Function bits in the control field, type identification and cause of transmission can be altered to represent different signals.

The program needs some addition to apply controlling the secondary station by the primary station. This comes from the use of one direction of data transmission after initiation procedure (from secondary to primary), while controlling procedure needs to permit secondary station to deal with controlled signals (open, close, step up, step down, reset) that received from the primary station.

For future work, the simulation program needs future development to include all the remaining protocol functions that cover all types of massages.
References


Program Code

unit Initiation7;

interface

{ Determine all Delphi units that used in this program. }
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Math;

{ Definitions of different types of variables. }
type
  TForm1 = class(TForm)
    Label1: TLabel;
    Label3: TLabel;
    Transfer: TButton;
    Stop: TButton;
    Label2: TLabel;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Button5: TButton;
    Button6: TButton;
    Button7: TButton;
    Button8: TButton;
    Button9: TButton;
    Button10: TButton;
    Button11: TButton;
    Label4: TLabel;
    Label5: TLabel;
    Button12: TButton;
  procedure TransferClick(Sender: TObject);
  procedure StopClick(Sender: TObject);
  procedure Button12Click(Sender: TObject);
end.

 Appendix A

 A - 1
private
{ Private declarations }

public
{ Public declarations }
end;

type
binary_byte = array [ 1 .. 8 ] of integer;
character = array [ 1 .. 8 ] of boolean;
buffer = array [1..1000] of boolean;
Frame_Fixed_Length = record
  StartChr     : character;
  ControlField : character;
  LinkAdr      : character;
  CheckSum     : character;
  EndChr       : character;
end;
end;
Frame_Variable_Length = record
  StartChr     : character;
  Length       : character;
  CopyLength   : character;
  CopyStartChr : character;
  ControlField : character;
  LinkAdr      : character;
  CheckSum     : character;
  EndChr       : character;
end;
end;
Information_Object = record
  Inform_Obj_Addr : character ;
  Fraction1 : character ;
  Fraction2 : character ;
  Fraction3 : character ;
  Exponent : character ;
  Quality_Desciptor : character ;
end;
end;
SetOfInf_Elements = array [ 1 .. 4 ] of Information_Object ;
ASDU = record
  Type_Identification : character ;
  Var_Str_Qaul :character ;
  CauseOfTransm :character ;
  CommonASDU_Addr : character ;
  InformationObjects : SetOfInf_Elements ;
Feeder = record
  Current : real;
  Voltage : real;
  ActivePower : double;
  ReactivePower : double;
end;

{ The following variables can be used in any procedure or sub procedure in the program. }
{ Values of these variables can be maintained and processed. }
var
  Form1: TForm1;
  label1 : TLabel;
  output_frame, input_frame, Test_Frame : Frame_Fixed_Length;
  Test_binary : binary_byte ;
  io_buffer : buffer ;
  Line1 : feeder;
  VarFrmTst : Frame_Variable_Length;

implementation

uses Initiation7_2;

{ For a given integer number (number), Binary_format procedure generates the }
{ corresponding binary byte (binary_chr) in addition to the Boolean byte (true/false) as a }
{ simulation to voltage levels ( 0 & 5 ) that used for data transmission in the physical link. }
{ For example if the procedure called for a number = 19 , then the binary_chr = }
{ [0.0.0.1.0.0.1.1] & the Boolean_chr = [false, false, false, true, false, false, true, true]. }
{ Note that the right bit is the least significant bit. }

procedure Binary_format ( var boolean_chr : character ; var binary_chr : }
binary_byte ; number :integer);
var
  x, y, i, n : integer;
  k : boolean;
begin
  i := 1; k := false; x := 0 ;
  if number = 0 then
    binary_chr[i] := 0
  else if number = 1 then
  begin
    binary_chr[i] := 1;
    i := i + 1;
end
else
repeat
  if x = 1 then k := true;
  x := number div 2;
  y := number mod 2;
  binary_chr[i] := y;
  i := i + 1;
  number := x;
until k;
for n := i to 8 do
  binary_chr[n] := 0;
for n := 1 to 8 do
  begin
    if binary_chr[n] = 1 then boolean_chr[n] := true
    else boolean_chr[n] := false;
  end;
end;

{ For a given integer number (number), Binary_format procedure generates the
  corresponding binary byte (binary_chr) in addition to the Boolean byte (true/false) as a
  simulation to voltage levels ( 0 & 5 ) that used for data transmission in the physical link.
  For example if the procedure called for a number = 19 , then the binary_chr =
  [0.0.0.1.0.0.1.1] & the Boolean_chr = [false, false, false, true, false, false, true, true].
  Note that the right bit is the least significant bit.}

Procedure Bin_digit_format ( var boolean_chr : character ; var binary_chr :
  binary_byte ; number :real);
var
  i : integer;
begin
  for i := 1 to 8 do
    begin
      if number < power(2,-i) then
        begin
          boolean_chr[9-i] := false;
          binary_chr[9-i] := 0;
        end
      else
        begin
          boolean_chr[9-i] := true;
          binary_chr[9-i] := 1;
          number := number - power(2,-i);
        end
    end
end

Appendix A
{ The checksum procedure calculates number of ones in a given frame of fixed-length type. }

procedure checksum (Total_Frame : Frame_Fixed_Length ; var csum :integer);
var
  i : integer;
begin
  csum := 0;
  for i:= 1 to 8 do
    begin
      if Total_Frame.StartChr[i] = true then csum := csum + 1;
      if Total_Frame.ControlField[i] = true then csum := csum + 1;
      if Total_Frame.LinkAdr[i] = true then csum := csum + 1;
      if Total_Frame.EndChr[i] = true then csum := csum + 1;
    end;
end;

{The checksum_VarL procedure calculates number of ones in a given frame of variable-length type.}

procedure checksum_VarL (Total_Frame:Frame_Variable_Length ; Total_ASDU : ASDU ; var csum :integer);
var
  i : integer;
begin
  csum := 0;
  for i:= 1 to 8 do
    begin
      if Total_Frame.StartChr[i] = true then csum := csum + 1;
      if Total_Frame.Length[i] = true then csum := csum + 1;
      if Total_Frame.CopyLength[i] = true then csum := csum + 1;
      if Total_Frame.CopyStartChr[i] = true then csum := csum + 1;
      if Total_Frame.ControlField[i] = true then csum := csum + 1;
      if Total_Frame.LinkAdr[i] = true then csum := csum + 1;
      if Total_ASDU.Type_Identification[i] = true then csum := csum + 1;
      if Total_ASDU.Var_Str_Qaul[i] = true then csum := csum + 1;
      if Total_ASDU.CauseOfTransm[i] = true then csum := csum + 1;
      if Total_ASDU.CommonASDU_Addr[i] = true then csum := csum + 1;
      if Total_ASDU.InformationObjects[1].Inform_Obj_Addr[i]=true then csum:=csum+1;
      if Total_ASDU.InformationObjects[1].Fraction1[i] = true then csum := csum + 1;
    end;
end;
if Total_ASDU.InformationObjects[1].Fraction2[i] = true then csum := csum + 1;
if Total_ASDU.InformationObjects[1].Fraction3[i] = true then csum := csum + 1;
if Total_ASDU.InformationObjects[1].Exponent[i] = true then csum := csum + 1;
if Total_ASDU.InformationObjects[1].Quality_Desciptor[i] = true then csum := csum + 1;
if Total_ASDU.InformationObjects[2].Inform_Obj_Addr[i] = true then csum := csum + 1;
if Total_ASDU.InformationObjects[2].Fraction1[i] = true then csum := csum + 1;
if Total_ASDU.InformationObjects[2].Fraction2[i] = true then csum := csum + 1;
if Total_ASDU.InformationObjects[2].Fraction3[i] = true then csum := csum + 1;
if Total_ASDU.InformationObjects[2].Exponent[i] = true then csum := csum + 1;
if Total_ASDU.InformationObjects[2].Quality_Desciptor[i] = true then csum := csum + 1;
if Total_Frame.EndChr[i] = true then csum := csum + 1;
end;
end;

{ For a given Boolean character, the InvBinary_format procedure returns the binary byte in addition to the decimal number. }

procedure InvBinary_format ( var boolean_chr : character ; var binary_chr : binary_byte ; var number :integer);
var
  n : integer;
begin
  number := 0 ;
  for n := 1 to 8 do
  begin
    if boolean_chr[n] = true then
    begin
      binary_chr[n] := 1 ;
      number := number + trunc(power(2,n-1)) ;
    end
    else binary_chr[n] := 0 ;
  end;
end;
{ For a given Boolean character, the InvBinary_format procedure returns the fraction binary byte in addition to the fraction number. }

procedure InvBinary_format_digit(var boolean_chr : character; var binary_chr : binary_byte; var number :real);
var
    n : integer;
begin
    number := 0;
    for n := 1 to 8 do
    begin
        if boolean_chr[9-n] = true then
            begin
                binary_chr[9-n] := 1;
                number := number + power(2, -n);
            end
        else binary_chr[9-n] := 0;
    end;
end;

{ The error procedure terminates the program and displays a message that explains case of termination. }

procedure error(msg : string);
begin
    showmessage(msg);
    halt(1);
end;

{$R *.dfm}

{ When the stop button have been clicked the TForm1.StopClick procedure terminates the program and explains a message. }

procedure TForm1.StopClick(Sender: TObject);
begin
    showmessage('Data Transmition stoped');
    halt(1);
end;

procedure TForm1.Button12Click(Sender: TObject);
begin
    form1.Visible := false;
    form4.Visible := true;
end;
{ When the transfer button have been clicked the TForm1. TransferClick procedure simulates data transferring between primary and secondary stations. }

procedure TForm1.TransferClick(Sender: TObject);
{
    The procedures Primary_Request_Status, Secondary_Receive and Primary_Reset_Status are used for initiation purposes.
    In the Primary_Request_Status procedure, the primary station sends – by the input/output buffer – a request message for status of the link. Control field of this message takes binary byte (0.1.0.0.1.0.0.1) which represented by decimal number (73).}

procedure Primary_Request_Status;
var
    Request : Frame_Fixed_Length;
    Byte_1, Byte_2, Byte_3, Byte_4, Byte_5 : binary_byte;
    sum : integer;
begin
    Button11.caption := 'Primary request status of the link' ;
    Binary_format ( Request.StartChr , Byte_1 , 16);
    Binary_format ( Request.ControlField , Byte_2 , 73);
    Binary_format ( Request.LinkAdr , Byte_3 , 4);
    Binary_format ( Request.EndChr , Byte_5 , 22);
    checksum (Request , sum);
    Binary_format ( Request.CheckSum , Byte_4 , sum );
    for sum := 1 to 8 do
        begin
            io_buffer[ sum ] := Request.StartChr[sum];
            io_buffer[sum + 8] := Request-ControlField[sum];
            io_buffer[sum + 16] := Request.LinkAdr[sum];
            io_buffer[sum + 24] := Request.CheckSum[sum];
            io_buffer[sum + 32] := Request.EndChr[sum];
        end;
end;
{
    By the Secondary_Receive procedure, the secondary station check a received frame in the input/output buffer and replies with the relative message. For minimization, tow messages have been defined in this procedure the request status of the link and the reset of the remote link messages. The request status of the link -with control field of decimal (73)- will be replied by a respond status of the link message -with control field of decimal (11)-. The reset of the remote link message -with control field of decimal (64)- will be replied by an acknowledge message -with control field of decimal (0)-.}
procedure Secondary_Receive ;

var
  RespondStatus, Acknowledge : Frame_Fixed_Length;
  Byte_1, Byte_2, Byte_3, Byte_4, Byte_5 : binary_byte;
  sum, k, k1, j, start_no : integer;

begin
  start_no := 1;
  k := 0;
  repeat
    for j := start_no to start_no + 7 do
      Test_Frame.StartChr[j - start_no + 1] := io_buffer[j];
      InvBinary_format( Test_Frame.StartChr, Test_binary, k);
      start_no := start_no + 8;
    if start_no = 100 then error('Initiation fail');
  until k = 16;
  for j := start_no to start_no + 7 do
    Test_Frame.ControlField[j - start_no + 1] := io_buffer[j];
    InvBinary_format( Test_Frame.ControlField, Test_binary, k1);
    start_no := start_no + 8;
    case k1 of
    73: begin
      for j := start_no to start_no + 7 do
        Test_Frame.LinkAdr[j - start_no + 1] := io_buffer[j];
        InvBinary_format( Test_Frame.LinkAdr, Test_binary, k);
      start_no := start_no + 8;
      if k <> 4 then error('Initiation fail');
    end;
  for j := start_no to start_no + 7 do
    Test_Frame.CheckSum[j - start_no + 1] := io_buffer[j];
    InvBinary_format( Test_Frame.CheckSum, Test_binary, k);
    start_no := start_no + 8;
    sum := k;
  for j := start_no to start_no + 7 do
    Test_Frame.EndChr[j - start_no + 1] := io_buffer[j];
    InvBinary_format( Test_Frame.EndChr, Test_binary, k);
  if k <> 22 then error('Initiation fail');
  checksum( Test_Frame, k);
  if k <> sum then error('Error in received Data');
  Button11.caption := 'Secondary respond status of the link';
  Binary_format( RespondStatus.StartChr, Byte_1, 16);
  Binary_format( RespondStatus.ControlField, Byte_2, 11);
  Binary_format( RespondStatus.LinkAdr, Byte_3, 4);
  Binary_format( RespondStatus.EndChr, Byte_5, 22);
  checksum(RespondStatus, sum);

Appendix A
Binary_format (RespondStatus.CheckSum, Byte_4, sum);
for sum := 1 to 8 do
begin
  io_buffer[sum] := RespondStatus.StartChr[sum];
  io_buffer[sum + 8] := RespondStatus.ControlField[sum];
  io_buffer[sum + 16] := RespondStatus.LinkAdr[sum];
  io_buffer[sum + 24] := RespondStatus.CheckSum[sum];
  io_buffer[sum + 32] := RespondStatus.EndChr[sum];
end;
end;

64: begin
  for j := start_no to start_no + 7 do
    Test_Frame.LinkAdr[j - start_no + 1] := io_buffer[j];
  InvBinary_format( Test_Frame.LinkAdr, Test_binary, k );
  start_no := start_no + 8;
  if k <> 4 then error('NotAcknowladge');
  for j := start_no to start_no + 7 do
    Test_Frame.CheckSum[j - start_no + 1] := io_buffer[j];
  InvBinary_format( Test_Frame.CheckSum, Test_binary, k );
  start_no := start_no + 8;
  sum := k;
  for j := start_no to start_no + 7 do
    Test_Frame.EndChr[j - start_no + 1] := io_buffer[j];
  InvBinary_format( Test_Frame.EndChr, Test_binary, k );
  if k <> 22 then error('NotAcknowladge');
  checksum( Test_Frame, k);
  if k <> sum then error('Error in received Data');
  Button11.caption := 'Secondary acknowladge';
  Binary_format( Acknowladge.StartChr, Byte_1, 16);
  Binary_format( Acknowladge.ControlField, Byte_2, 32);
  Binary_format( Acknowladge.LinkAdr, Byte_3, 4);
  Binary_format( Acknowladge.EndChr, Byte_5, 22);
  checksum( Acknowladge, sum);
  Binary_format( Acknowladge.CheckSum, Byte_4, sum);
  for sum := 1 to 8 do
begin
  io_buffer[sum] := Acknowladge.StartChr[sum];
  io_buffer[sum + 8] := Acknowladge.ControlField[sum];
  io_buffer[sum + 16] := Acknowladge.LinkAdr[sum];
  io_buffer[sum + 24] := Acknowladge.CheckSum[sum];
  io_buffer[sum + 32] := Acknowladge.EndChr[sum];
end;
end;end;
end;

{ By the Primary_Reset_Status procedure, the primary station check a received frame in the input/output buffer and replies with the relative message. When a resond status of the link message with control field of decimal (11)- was received the primary will reply by a reset of the remote link message -with control field of decimal (64)-. }

procedure Primary_Reset_Status ;
var
  ResetLink : Frame_Fixed_Length;
  Byte_1, Byte_2, Byte_3, Byte_4, Byte_5 : binary_byte ;
  sum, k, k1, j, start_no : integer;
begin
  start_no := 1 ;
  k:=0;
  repeat
    for j := start_no to start_no + 7  do
      Test_Frame.StartChr[j - start_no + 1] := io_buffer[j] ;
      InvBinary_format( Test_Frame.StartChr, Test_binary, k );
    start_no := start_no + 8;
    if start_no = 100 then error('NotAcknowladge');
  until k = 16 ;
  for j := start_no to start_no + 7 do
    Test_Frame.ControlField[j - start_no + 1] := io_buffer[j] ;
    InvBinary_format( Test_Frame.ControlField, Test_binary, k1 );
  start_no := start_no + 8;
  case k1 of
  11:begin
    Button11.caption := 'Primary reset remote link' ;
    for j := start_no to start_no + 7 do
      InvBinary_format( Test_Frame.LinkAdr, Test_binary, k );
    start_no := start_no + 8;
    if k <> 4 then error('NotAcknowladge');
  end
  for j := start_no to start_no + 7 do
    Test_Frame.CheckSum[j - start_no + 1] := io_buffer[j] ;
    InvBinary_format( Test_Frame.CheckSum, Test_binary, k );
  start_no := start_no + 8;
  sum := k ;
  for j := start_no to start_no + 7 do
    Test_Frame.EndChr[j - start_no + 1] := io_buffer[j] ;
    InvBinary_format( Test_Frame.EndChr, Test_binary, k );
  if k <> 22 then error('NotAcknowladge');
checksum (Test_Frame, k);
if k <> sum then error('Error in received Data');
Binary_format (ResetLink.StartChr, Byte_1, 16);
Binary_format (ResetLink.ControlField, Byte_2, 64);
Binary_format (ResetLink.LinkAdr, Byte_3, 4);
Binary_format (ResetLink.EndChr, Byte_5, 22);
checksum (ResetLink, sum);
Binary_format (ResetLink.CheckSum, Byte_4, sum);
for sum := 1 to 8 do
begin
  io_buffer[sum] := ResetLink.StartChr[sum];
  io_buffer[sum + 8] := ResetLink.ControlField[sum];
  io_buffer[sum + 16] := ResetLink.LinkAdr[sum];
  io_buffer[sum + 24] := ResetLink.CheckSum[sum];
  io_buffer[sum + 32] := ResetLink.EndChr[sum];
end;
end;
end;

{For a generated Data (Data_Unit), the second_send procedure constructs the
structure format of a variable length message that contains the ASDU of Line1.}

procedure second_send_ASDU(var Data_Unit : ASDU);
var
  Byte_1 : binary_byte;
  sum : integer;
beginn
  Button9.caption := 'Secondary starting transmit data';
  Button3.Caption:= floattostr(Line1.Voltage) + 'kV';
  Button5.Caption:= floattostr(Line1.ActivePower) + 'MWat';
  Button7.Caption:= floattostr(Line1.ReactivePower) + 'MVar';
sleep(1000);
Binary_format (VarFrmTst.StartChr, Byte_1, 16);
Binary_format (VarFrmTst.Length, Byte_1, 30);
Binary_format (VarFrmTst.CopyLength, Byte_1, 30);
Binary_format (VarFrmTst.CopyStartChr, Byte_1, 16);
Binary_format (VarFrmTst.ControlField, Byte_1, 8);
Binary_format (VarFrmTst.LinkAdr, Byte_1, 4);
Binary_format (Data_Unit.type_Identification, Byte_1, 13);
Binary_format (Data_Unit.Var_Str_Qaul, Byte_1, 4);
Binary_format (Data_Unit.CauseOfTransm, Byte_1, 2);
Binary_format (Data_Unit.CommonASDU_Addr, Byte_1, 11);
Binary_format (Data_Unit.InformationObjects[1].Inform_Obj_Addr, Byte_1, 1);
Binary_format (Data_Unit.InformationObjects[1].Fraction1, Byte_1, 0);
Binary_format (Data_Unit.InformationObjects[1].Fraction2, Byte_1, 0);
Bin_digit_format (Data_Unit.InformationObjects[1].Fraction3, Byte_1, frac(Line1.Current));
Binary_format (Data_Unit-informationObjects[1].Exponent, Byte_1, trunc(int(Line1.Current)));
Binary_format (Data_Unit.InformationObjects[2].Inform_Obj_Addr, Byte_1, 2);
Binary_format (Data_Unit.InformationObjects[2].Fraction1, Byte_1, 0);
Binary_format (Data_Unit.InformationObjects[2].Fraction2, Byte_1, 0);
Bin_digit_format (Data_Unit.InformationObjects[2].Fraction3, Byte_1, frac(Line1.Voltage));
Binary_format (Data_Unit.InformationObjects[2].Exponent, Byte_1, trunc(int(Line1.Voltage)));
Binary_format (Data_Unit.InformationObjects[3].Inform_Obj_Addr, Byte_1, 3);
Binary_format (Data_Unit.InformationObjects[3].Fraction1, Byte_1, 0);
Binary_format (Data_Unit.InformationObjects[3].Fraction2, Byte_1, 0);
Bin_digit_format (Data_Unit.InformationObjects[3].Fraction3, Byte_1, frac(Line1.ActivePower));
Binary_format (Data_Unit.InformationObjects[3].Exponent, Byte_1, trunc(int(Line1.ActivePower)));
Binary_format (Data_Unit.InformationObjects[4].Inform_Obj_Addr, Byte_1, 4);
Binary_format (Data_Unit.InformationObjects[4].Fraction1, Byte_1, 0);
Binary_format (Data_Unit.InformationObjects[4].Fraction2, Byte_1, 0);
Bin_digit_format (Data_Unit.InformationObjects[4].Fraction3, Byte_1, frac(Line1.ReactivePower));
Binary_format (Data_Unit.InformationObjects[4].Exponent, Byte_1, trunc(int(Line1.ReactivePower)));
Binary_format (Data_Unit.InformationObjects[5].Inform_Obj_Addr, Byte_1, 5);
Binary_format (Data_Unit.InformationObjects[5].Fraction1, Byte_1, 0);
Binary_format (Data_Unit.InformationObjects[5].Fraction2, Byte_1, 0);
Bin_digit_format (Data_Unit.InformationObjects[5].Fraction3, Byte_1, frac(Line2.Current));
Binary_format (Data_Unit.InformationObjects[5].Exponent, Byte_1, trunc(int(Line2.Current)));
Binary_format (Data_Unit.InformationObjects[5].Quality_Descriptor, Byte_1, 0);
Binary_format (Data_Unit.InformationObjects[6].Inform_Obj_Addr, Byte_1, 6);
Binary_format (Data_Unit.InformationObjects[6].Fraction1, Byte_1, 0);
Binary_format (Data_Unit.InformationObjects[6].Fraction2, Byte_1, 0);
Bin_digit_format (Data_Unit.InformationObjects[6].Fraction3, Byte_1, frac(Line2.Voltage));
Binary_format (Data_Unit.InformationObjects[6].Exponent, Byte_1, trunc(int(Line2.Voltage)));
Binary_format (Data_Unit.InformationObjects[6].Quality_Descriptor, Byte_1, 0);
Binary_format (VarFrmTst.EndChr, Byte_1, 22);
checksum_VarL (VarFrmTst, Data_Unit, sum);
Binary_format (VarFrmTst.CheckSum, Byte_1, sum);
for sum := 1 to 8 do
begin
io_buffer[sum] := VarFrmTst.StartChr[sum];
io_buffer[sum + 8] := VarFrmTst.Length[sum];
io_buffer[sum + 16] := VarFrmTst.CopyLength[sum];
io_buffer[sum + 24] := VarFrmTst.CopyStartChr[sum];
io_buffer[sum + 32] := VarFrmTst.ControlField[sum];
io_buffer[sum + 40] := VarFrmTst.LinkAdr[sum];
io_buffer[sum + 48] := Data_Unit.type_Identification[sum];
io_buffer[sum + 56] := Data_Unit.Var_Str_Qaul[sum];
io_buffer[sum + 64] := Data_Unit.CauseOfTransm[sum];
io_buffer[sum + 72] := Data_Unit.CommonASDU_Addr[sum];
io_buffer[sum + 80] := Data_Unit.InformationObjects[1].Inform_Obj_Addr[sum];
io_buffer[sum + 88] := Data_Unit.InformationObjects[1].Fraction1[sum];
io_buffer[sum + 96] := Data_Unit.InformationObjects[1].Fraction2[sum];
io_buffer[sum + 104] := Data_Unit.InformationObjects[1].Fraction3[sum];
io_buffer[sum + 112] := Data_Unit.InformationObjects[1].Exponent[sum];
io_buffer[sum + 120] := Data_Unit.InformationObjects[1].Quality_Desciptor[sum];
io_buffer[sum + 128] := Data_Unit.InformationObjects[2].Inform_Obj_Addr[sum];
io_buffer[sum + 136] := Data_Unit.InformationObjects[2].Fraction1[sum];
io_buffer[sum + 144] := Data_Unit.InformationObjects[2].Fraction2[sum];
io_buffer[sum + 152] := Data_Unit.InformationObjects[2].Fraction3[sum];
io_buffer[sum + 160] := Data_Unit.InformationObjects[2].Exponent[sum];
io_buffer[sum + 168] := Data_Unit.InformationObjects[2].Quality_Desciptor[sum];
io_buffer[sum + 176] := Data_Unit.InformationObjects[3].Inform_Obj_Addr[sum];
io_buffer[sum + 184] := Data_Unit.InformationObjects[3].Fraction1[sum];
io_buffer[sum + 192] := Data_Unit.InformationObjects[3].Fraction2[sum];
io_buffer[sum + 200] := Data_Unit.InformationObjects[3].Fraction3[sum];
io_buffer[sum + 208] := Data_Unit.InformationObjects[3].Exponent[sum];
io_buffer[sum + 216] := Data_Unit.InformationObjects[3].Quality_Desciptor[sum];
io_buffer[sum + 224] := Data_Unit.InformationObjects[4].Inform_Obj_Addr[sum];
io_buffer[sum + 232] := Data_Unit.InformationObjects[4].Fraction1[sum];
io_buffer[sum + 240] := Data_Unit.InformationObjects[4].Fraction2[sum];
io_buffer[sum + 248] := Data_Unit.InformationObjects[4].Fraction3[sum];
io_buffer[sum + 256] := Data_Unit.InformationObjects[4].Exponent[sum];
io_buffer[sum + 264] := Data_Unit.InformationObjects[4].Quality_Desciptor[sum];
io_buffer[sum + 272] := VarFrmTst.CheckSum[sum];
io_buffer[sum + 280] := VarFrmTst.EndChr[sum];
end;
Button9.caption := '';
end;

procedure prim_receive_ASDU(var Data_unit:ASDU);

var
    value1:val1;
    value2:val2;

end;
begin
Button10.caption := 'Primary receiving data' ;
sleep(1000);
start_no := 1 ;
k:=0;
repeat
  for j := start_no to start_no + 7 do
    VarFrmTst.StartChr[j - start_no + 1] := io_buffer[j] ;
    InvBinary_format( VarFrmTst.StartChr, Test_binary, k ) ;
    start_no := start_no + 8 ;
    if start_no = 1000 then error('Starting character not found');
  until k = 16 ;
  for j := start_no to start_no + 7 do
    InvBinary_format( VarFrmTst.Length , Test_binary, k ) ;
    start_no := start_no + 8 ;
    length1:=k;
  for j := start_no to start_no + 7 do
    InvBinary_format( VarFrmTst.CopyLength , Test_binary, k ) ;
    start_no := start_no + 8 ;
    length2:=k;
if length1 <> length2 then error('Incorrect massage format');
  repeat
    for j := start_no to start_no + 7 do
      InvBinary_format( VarFrmTst.CopyStartChr, Test_binary, k ) ;
      start_no := start_no + 8 ;
      if start_no = 1000 then error('Starting character not found');
  until k = 16 ;
  for j := start_no to start_no + 7 do
    InvBinary_format( VarFrmTst.ControlField, Test_binary, k1 ) ;
    start_no := start_no + 8 ;
    if k1 <> 8 then error('undefine event');
case k1 of
  8 : begin
    for j := start_no to start_no + 7 do
      InvBinary_format( VarFrmTst.LinkAdr, Test_binary, k ) ;
      start_no := start_no + 8 ;
      if k < 4 then error('uncorrect link address');
  for j := start_no to start_no + 7 do

Data_unit.type_Identification[j - start_no + 1] := io_buffer[j] ;
InvBinary_format( Data_unit.type_Identification, Test_binary, k2 );
start_no := start_no + 8;
if k2 <> 13 then error('Error in ASDU');
case k2 of
13: begin
  for j := start_no to start_no + 7 do
    Data_unit.Var_Str_Qaul[j - start_no + 1] := io_buffer[j] ;
    InvBinary_format( Data_unit.Var_Str_Qaul, Test_binary, k );
  start_no := start_no + 8;
  Length_no1 := k;
for j := start_no to start_no + 7 do
InvBinary_format( Data_unit.CauseOfTransm, Test_binary, k3 );
start_no := start_no + 8;
case k3 of
  2: begin
    for j := start_no to start_no + 7 do
    InvBinary_format( Data_unit.CommonASDU_Addr, Test_binary, k );
    start_no := start_no + 8;
    if k <> 11 then error('Error in ASDU address');
    Length_test := start_no;
    for i:= 1 to Length_no1 do
      begin
        for j := start_no to start_no + 7 do
          Data_unit.InformationObjects[i].Inform_Obj_Addr[j - start_no + 1]
            := io_buffer[j];
        InvBinary_format( Data_unit.InformationObjects[i].Inform_Obj_Addr,
            Test_binary, k );
        start_no := start_no + 8;
        if k <> i then error('Error in ASDU');
        for j := start_no to start_no + 7 do
        InvBinary_format( Data_unit.InformationObjects[i].Fraction1, Test_binary, k );
        start_no := start_no + 8;
        if k <> 0 then error('Error in ASDU');
        for j := start_no to start_no + 7 do
        InvBinary_format( Data_unit.InformationObjects[i].Fraction2, Test_binary, k );
        start_no := start_no + 8;
        for j := start_no to start_no + 7 do
InvBinary_format_digit( Data_unit.InformationObjects[i].Fraction3, Test_binary, value1[i] );
start_no := start_no + 8;
for j := start_no to start_no + 7 do
InvBinary_format( Data_unit.InformationObjects[i].Exponent, Test_binary, value2[i] );
start_no := start_no + 8;
for j := start_no to start_no + 7 do
InvBinary_format( Data_unit.InformationObjects[i].Quality_Desciptor, Test_binary, k );
start_no := start_no + 8;
if k <> 0 then error('Error in ASDU');
end;
Length_test := start_no - Length_test;
if (Length_test div 48) <> Length_no1 then error('Error in ASDU');
end;end;
end;end;
end;end;

length_no := start_no - 33;
for j := start_no to start_no + 7 do
InvBinary_format( VarFrmTst.CheckSum, Test_binary, k );
start_no := start_no + 8;
sum := k ;
for j := start_no to start_no + 7 do
    VarFrmTst.EndChr [j - start_no + 1] := io_buffer[j] ;
InvBinary_format( VarFrmTst.EndChr, Test_binary, k );
if (k <> 22) or (Length_no div 8 <> length1) then error('Error in message format');
checksum_VarL ( VarFrmTst, Data_unit, k ) ;
if k <> sum then error('Error in received Data');
Button10.caption := '' ;
value1[1]:= SimpleRoundTo(value1[1],-2);
value1[2]:= SimpleRoundTo(value1[2],-2);
value1[3]:= SimpleRoundTo(value1[3],-2);
value1[4]:= SimpleRoundTo(value1[4],-2);
Button2.Caption:= floattostr(value1[1]+value2[1]) +'  Amps';
end;
{ For a given values for current and voltage, the Generate_Readings procedure simulates variations by adding a random real number that depends on the given values. The active and reactive powers will be calculated for three-phase system with 0.8 power factor. All real values are rounded to tow digits for simplicity. }

procedure Generate_Readings(var Line1: feeder);
begin
randomize;
with Line1 do
begin
  current := current + Random(10)/(Random(9)+0.2);
current := SimpleRoundTo(current, -2);
voltage := voltage + Random(5)/(Random(4)+0.2);
voltage:= SimpleRoundTo(voltage, -2);
ActivePower := sqrt(3) * voltage * current * 0.8/1000;
ActivePower := SimpleRoundTo(ActivePower, -2);
ReactivePower := sqrt(3) * voltage * current * 0.6/1000;
ReactivePower := SimpleRoundTo(ReactivePower, -2);
end;
end;
{ Main program which started with the initiation procedures Primary_Request_Status, Secondary_Receive, Primary_Reset_Status and Secondary_Receive. In addition, data for Line1 (current, voltage, active and reactive powers) have been generated. The secondary and primary stations started sending and receiving data respectively using second_send_ASDU and prim_receive_ASDU procedures.}

```pascal
var
  t : integer;
  Data : ASDU;
begin
  Primary_Request_Status; sleep(1000);
  Secondary_Receive; sleep(1000);
  Primary_Reset_Status; sleep(1000);
  Secondary_Receive; sleep(1000);
  Button11.caption := 'Initiation Prportocol Transmission';
  t := 0;
  Line1.Current := 150.532;
  Line1.Voltage := 110.456;
  repeat
    t := t + 1;
    Generate_Readings(Line1);
    second_send_ASDU(Data);
    prim_receive_ASDU(Data);
    sleep(5000);
  until t = 5;
end;

End.
```
Interoperability

This standard protocol presents sets of parameters and alternatives from which subsets have to be selected to implement particular telecontrol systems. Certain parameter values, such as the number of octets in the COMMON ADDRESS of ASDUs represent mutually exclusive alternatives. This means that only one value of the defined parameters is admitted per system. Other parameters, such as the listed set of different process information in command and in monitor direction allow the specification of the complete set or subsets, as appropriate for given applications. This clause summarizes the parameters of the previous clauses to facilitate a suitable selection for a specific application. The selected parameters were crossed in the white boxes (“✓”).

1) **Network configuration**
   (network-specific parameter)

   - Point-to-point
   - Multiple point-to-point
   - Multipoint-party line
   - Multipoint-star

2) **Physical layer**
   (network-specific parameter)

   **Transmission speed (control direction)**
   
<table>
<thead>
<tr>
<th>Unbalanced interchange</th>
<th>Unbalanced interchange</th>
<th>Balanced interchange</th>
</tr>
</thead>
<tbody>
<tr>
<td>circuit V.24/V.28</td>
<td>circuit V.24/V.28</td>
<td>circuit X.24/X.27</td>
</tr>
<tr>
<td>Standard</td>
<td>Recommended if &gt;1 200 bit/s</td>
<td></td>
</tr>
<tr>
<td>100 bit/s</td>
<td>2400 bit/s</td>
<td>2400 bit/s</td>
</tr>
<tr>
<td>200 bit/s</td>
<td>4800 bit/s</td>
<td>4800 bit/s</td>
</tr>
<tr>
<td>300 bit/s</td>
<td>9600 bit/s</td>
<td>9600 bit/s</td>
</tr>
<tr>
<td>600 bit/s</td>
<td></td>
<td>19200 bit/s</td>
</tr>
<tr>
<td>1200 bit/s</td>
<td></td>
<td>38400 bit/s</td>
</tr>
</tbody>
</table>

   **Transmission speed (monitor direction)**
3) **Link layer**

(network-specific parameter)

**Link transmission procedure**

- Balanced transmission
- Unbalanced transmission

**Address field of link**

- Not present (balanced transmission only)
- One octet
- Two octets

**Frame length**

- Maximum length L (number of octets) = 36

4) **Application Layer**

Transmission mode for application data

Mode 1 (Least significant octet first) is used exclusively in this companion standard.

**Common address of ASDU**

(system-specific parameter)

- One octet
- Two octets

**Information object address**

(system-specific parameter)

- One octet
- Two octets
- Three octets
- structured
- unstructured

**Cause of transmission**

(system-specific parameter)

- One octet
- Two octets (with originator address)
Selection of standard ASDUs

Process information in monitor direction
(station-specific parameter)

- <1> := Single-point information M_SP_NA_1
- <2> := Single-point information with time tag M_SP_TA_1
- <3> := Double-point information M_DP_NA_1
- <4> := Double-point information with time tag M_DP_TA_1
- <5> := Step position information M_ST_NA_1
- <6> := Step position information with time tag M_ST_TA_1
- <7> := Bitstring of 32 bit M_BO_NA_1
- <8> := Bitstring of 32 bit with time tag M_BO_TA_1
- <9> := Measured value, normalized value M_ME_NA_1
- <10> := Measured value, normalized value with time tag M_ME_TA_1
- <11> := Measured value, scaled value M_ME_NB_1
- <12> := Measured value, scaled value with time tag M_ME_TB_1
- <13> := Measured value, short floating point value M_ME_NC_1
- <14> := Measured value, short floating point value with time tag M_ME_TC_1
- <15> := Integrated totals M_IT_NA_1
- <16> := Integrated totals with time tag M_IT_TA_1
- <17> := Event of protection equipment with time tag M_EP_TA_1
- <18> := Packed start events of protection equipment with time tag M_EP_TB_1
- <19> := Packed output circuit information of protection equipment with time tag M_EP_TC_1
- <20> := Packed single-point information with status change detection M_PS_NA_1
- <21> := Measured value, normalized value without quality descriptor M_ME_ND_1
- <30> := Single-point information with time tag CP56Time2a M_SP_TB_1
- <31> := Double-point information with time tag CP56Time2A M_DP_TB_1
- <32> := Step position information with time tag CP56Time2A M_ST_TB_1
- <33> := Bitstring of 32 bit with time tag CP56Time2A M_BO_TB_1
- <34> := Measured value, normalized value with time tag CP56Time2A M_ME_TD_1
- <35> := Measured value, scaled value with time tag CP56Time2A M_ME_TE_1
- <36> := Measured value, short floating point value with time tag CP56Time2A M_ME_TF_1
- <37> := Integrated totals with time tag CP56Time2A M_IT_TB_1
- <38> := Event of protection equipment with time tag CP56Time2A M_EP_TD_1
- <39> := Packed start events of protection equipment with time tag CP56Time2A M_EP_TE_1
- <40> := Packed output circuit information of protection equipment with time tag CP56Time2a M_EP_TF_1
Process information in control direction
(station-specific parameter)

- <45> := Single command C_SC_NA_1
- <46> := Double command C_DC_NA_1
- <47> := Regulating step command C_RC_NA_1
- <48> := Set point command, normalized value C_SE_NA_1
- <49> := Set point command, scaled value C_SE_NB_1
- <50> := Set point command, short floating point value C_SE_NC_1
- <51> := Bitstring of 32 bit C_BO_NA_1

System information in monitor direction
(station-specific parameter)

- <70> := End of initialization M_EI_NA_1

System information in control direction
(station-specific parameter)

- <100> := Interrogation command C_IC_NA_1
- <101> := Counter interrogation command C_CI_NA_1
- <102> := Read command C_RD_NA_1
- <103> := Clock synchronization command C_CS_NA_1
- <104> := Test command C_TS_NB_1
- <105> := Reset process command C_RP_NC_1
- <106> := Delay acquisition command C_CD_NA_1

Parameter in control direction
(station-specific parameter)

- <110> := Parameter of measured value, normalized value P_ME_NA_1
- <111> := Parameter of measured value, scaled value P_ME_NB_1
- <112> := Parameter of measured value, short floating point value P_ME_NC_1
- <113> := Parameter activation P_AC_NA_1

File transfer
(station-specific parameter)

- <120> := File ready F_FR_NA_1
- <121> := Section ready F_SR_NA_1
- <122> := Call directory, select file, call file, call section F_SC_NA_1
- <123> := Last section, last segment F_LS_NA_1
- <124> := Ack file, ack section F_AF_NA_1
- <125> := Segment F_SG_NA_1
- <126> := Directory F_DR_TA_1
5) **Basic application functions**

Station initialization
(station-specific parameter)

☒ Remote initialization

**General Interrogation**
(system- or station-specific parameter)

☐ global

☐ group 1 ☐ group 7 ☐ group 13

☐ group 2 ☐ group 8 ☐ group 14

☐ group 3 ☐ group 9 ☐ group 15

☐ group 4 ☐ group 10 ☐ group 16

☐ group 5 ☐ group 11

☐ group 6 ☐ group 12

Addresses per group have to be defined

Clock synchronization
(station-specific parameter)

☐ Clock synchronization

Command transmission
(object-specific parameter)

☐ Direct command transmission ☐ Select and execute command

☐ Direct set point command transmission ☐ Select and execute set point command

☐ C_SE_ACTTERM used

☐ No additional definition

☐ Short pulse duration (duration determined by a system parameter in the outstation)

☐ Long pulse duration (duration determined by a system parameter in the outstation)

☐ Persistent output

Transmission of Integrated totals
(station- or object-specific parameter)

☐ Counter request ☐ General request counter

☐ Counter freeze without reset ☐ Request counter group 1

☐ Counter freeze with reset ☐ Request counter group 2

☐ Counter reset ☐ Request counter group 3

Addresses per group have to be defined

Represented in plain text.
**Parameter loading**  
(object-specific parameter)  
- Threshold value  
- Smoothing factor  
- Low limit for transmission of measured value  
- High limit for transmission of measured value  

**Parameter activation**  
(object-specific parameter)  
- Act/deact of persistent cyclic or periodic transmission of the addressed object  

**File transfer**  
(station-specific parameter)  
- File transfer in monitor direction  
- File transfer in control direction
Cause of transmission possibilities for each ASDU

Cause := UI6[1..6]<0..63>
<0> := not used
<1> := periodic, cyclic
<2> := background scan
<3> := spontaneous
<4> := initialized
<5> := request or requested
<6> := activation
<7> := activation confirmation
<8> := deactivation
<9> := deactivation confirmation
<10> := activation termination
<11> := return information caused by a remote command
<12> := return information caused by a local command
<13> := file transfer
<14..19> := reserved for further compatible definitions
<20> := interrogated by general interrogation
<21> := interrogated by group 1 interrogation
<22> := interrogated by group 2 interrogation
<23> := interrogated by group 3 interrogation
<24> := interrogated by group 4 interrogation
<25> := interrogated by group 5 interrogation
<26> := interrogated by group 6 interrogation
<27> := interrogated by group 7 interrogation
<28> := interrogated by group 8 interrogation
<29> := interrogated by group 9 interrogation
<30> := interrogated by group 10 interrogation
<31> := interrogated by group 11 interrogation
<32> := interrogated by group 12 interrogation
<33> := interrogated by group 13 interrogation
<34> := interrogated by group 14 interrogation
<35> := interrogated by group 15 interrogation
<36> := interrogated by group 16 interrogation
<37> := requested by general counter request
<38> := requested by group 1 counter request
<39> := requested by group 2 counter request
<40> := requested by group 3 counter request
<41> := requested by group 4 counter request
<42..47> := reserved for further compatible definitions
General structure of the ASDU message.

START FRAME

START CHARACTER

LENGTH

LENGTH

START CHARACTER

CONTROL FIELD

LINK ADDRESS

DATA UNIT TYPE

DATA UNIT IDENTIFIER

TYPE IDENTIFICATION

VARIABLE STRUCTURE QUALIFIER

CAUSE OF TRANSMISSION

COMMON ADDRESS OF ASDU

COMMON ADDRESS OF ASDU

INFORMATION OBJECT ADDRESS

INFORMATION OBJECT ADDRESS

INFORMATION OBJECT ADDRESS

SET OF INFORMATION ELEMENTS

TIME TAG ms

TIME TAG ms

IV Res TIME TAG min

APPLICATION SERVICE DATA UNIT

INFORMATION OBJECT

INFORMATION OBJECT n

CHECK SUM

STOP CHARACTER

STOP FRAME

Optional per system

Variable per ASDU