Master Thesis

Computer Science

Thesis no: MCS-2004:23

Remote wireless control of building management systems automation

Marcus Gylling

Blekinge Institute of Technology
Department of Software Engineering and Computer Science
Examinator: Rune Gustavsson, Blekinge Institute of Technology

Preface

This thesis is submitted to the Department of Software Engineering and Computer Science at Blekinge Institute of Technology in partial fulfilment of the requirements for the degree of Master of Science in Computer Science. The thesis is equivalent to 20 weeks of full time studies.

The thesis is done in cooperation with Styrprojektering AB. Initiator to the project is Jan-Åke Gylling at Styrprojektering AB.

The target group for this thesis is companies that creates installations in the building systems automation business, the users that controls those installations and also the house-owners of the buildings where the installations are made.

Contact information
Author:
Marcus Gylling
Email:
marcus.gylling@bredband.net

External advisor:
Jan-Åke Gylling
Styrprojektering AB

University advisor:
Rune Gustravsson

Abstract

The controlling unit in building management system automation is a PLC. Every device in an installation is connected to the PLC. When a user wants to interact with a system an operator terminal, which is attached to a cabinet where the PLC is installed, is used. With this solution the user needs to be in front of the cabinet to interact with the system. Alarms do not get the user's attention until the user checks the operator terminal.

Using a solution where the communication with the PLC is done with a wireless interface would mean that the user interact with a system from a wider area. The solution should have the same functionality as today with the extension that the PLC should be able to contact the user if something is wrong in the installation.

A PDA is used as a replacement for the operator terminal. This PDA uses two different techniques to communicate with the PLC. Bluetooth is used when the user is in the building and interacts with the system in a similar way to the operator terminals. GSM is used when the PLC needs to get the users attention by sending out alarms. With this solution the PDA can be used for several installations and thereby decrease cost.

The project has turned out to be a success. The application that has been developed has improved a user's interaction with a PLC.

Keywords: Communication, Wireless, Alarms, PDA.

Table of contents

1. Introduction

1.1 Smart houses

Smart houses is a term that often is used when talking about making it easer for a person to handle different duties in the home. One example is having a refrigerator that reminds you when it is time to go shopping. This thesis is investigating smart houses on another level where the goal is to make it easer to control a whole real estate by using wireless techniques to control the different units in the building.

1.2 Styrprojektering AB

Styrprojektering AB works with building management systems automation. The company delivers automation equipment for controlling heating and ventilation installations. The controlling unit in these installations is a PLC[1]. In small plants Styrprojektering uses their own PLC's and in more complex plants PLC's from SAIA are used.
Every PLC is placed inside a cabinet.

Every device in an installation is either saved in flags or registers in the database of the PLC. The PLC is connected to all devices in the installation and frequently receives and saves their current status (Figure 1). The devises can be fans, pumps, valves, dampers, sensors etc. The status is either saved in flags (0 or 1) or in registers (integer) in the PLC.

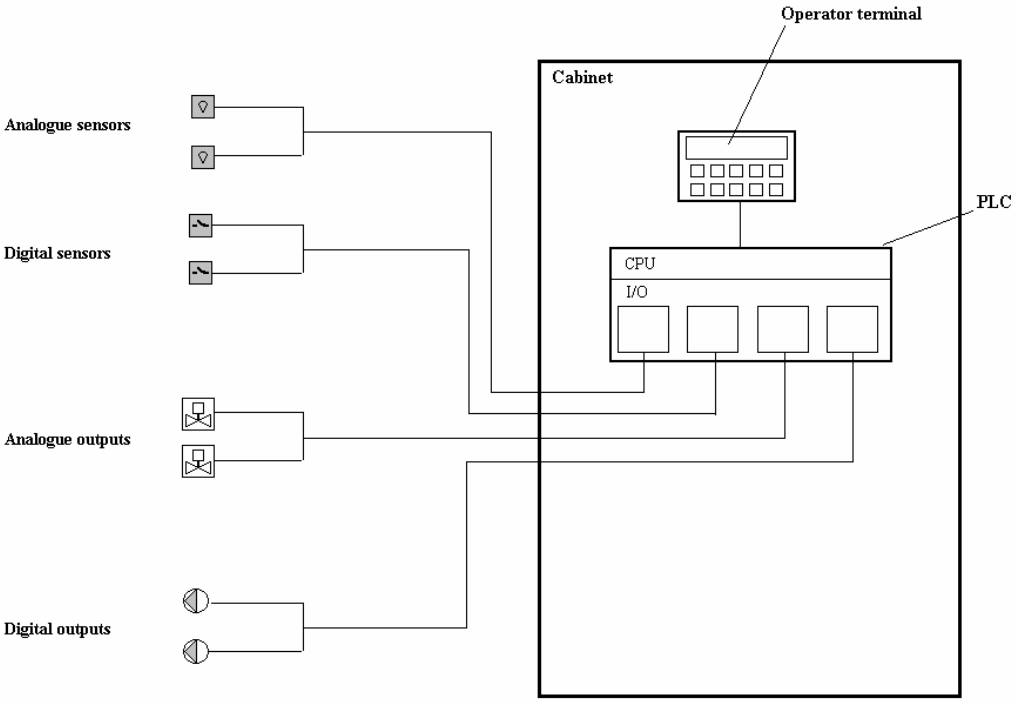


Figure 1: Example of an installation

[1] See glossary for details

1.3 Background

A user interacts with the system with an operator terminal. The operator terminal is attached on the outside of the cabinet and connected to the PLC. There is one operator terminal for each cabinet.
When the user changes a value in the PLC there is no possibility to see directly how a unit in the plant responds to this change unless the user gets help from a second person located by the unit.

If something goes wrong in the plant it is indicated as an alarm in the PLC. The active alarms are displayed in the operating terminal. These alarms may be active for a long time before they are noticed since the PLC has no way to contact the user.

1.4 Approach

The idea with this master thesis is to investigate the possibility to improve the way a user can interact with a PLC. The solution should replace the operator terminals used today.
The goal is to make it possible for a user to communicate with a PLC while moving around in a building, making it possible to check units in a plant and at the same time staying connected to the PLC.
The solution should also make it possible for the PLC to contact a user when something is indicated as erroneous in the plant.

Total cost of an installation is the next aspect to take into consideration. When choosing which techniques to use this must be taken into consideration. A solution were the total price of an installation increases will not be used.

1.5 Concept

The approach is to use a handheld device to communicate with the PLC(Figure 2). The handheld device should be able to communicate with every PLC in one or several installations.
The communication between the handheld and the PLC will be done with two different wireless techniques. One for when the user is walking around in the building and one technique when the PLC needs to contact the user at long range.
A PDA[2] will be used as handheld device. A SAIA PLC will be used in this project.

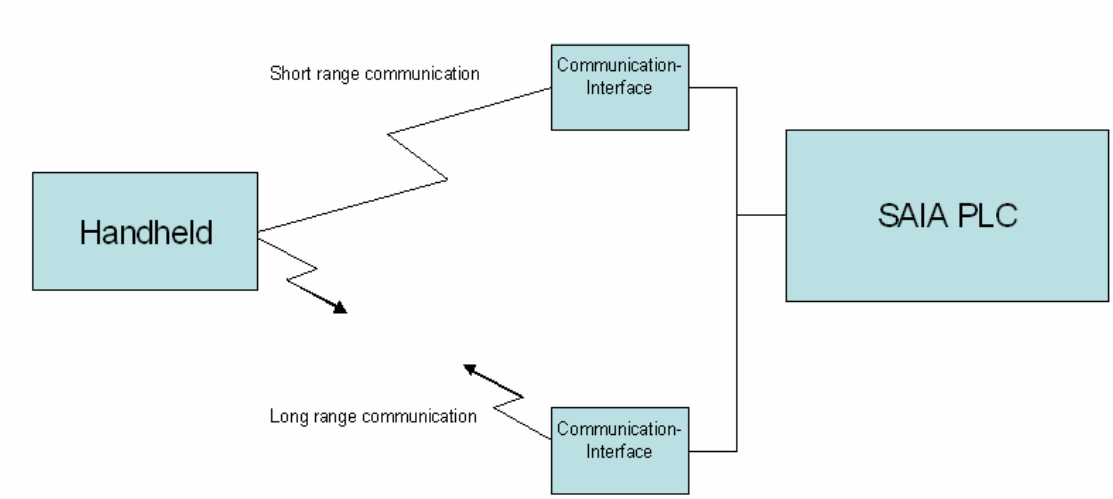


Figure 2: Conceptual model of the application

1.6 Requirements

- The PDA should be able to communicate with the PLC without having to change the communication protocol used in the PLC.
- The communication at short range should work as good as the wired communication that is used today.
- The application in the PDA should be easy to understand for users that are no experts on computers.
- The PLC should be able to contact a user at long range when an alarm is active

[2] See glossary for details

1.7 Problems

The following problems were identified when the project started.

- How does the protocol work that is used by the PLC.
- Which technologies will work best for the communication
- How should the signals from the PLC be transforming the into wireless signals
- What wireless techniques works for both the PLC and the PDA
- How to make the communication wireless without losing functionality
- Is it possible to improve the way a user communicates with a system without having to increase the cost of a system

1.8 Schedule

The main emphasis in this thesis is on developing the software programs.

Phases	Time
Feasibility study	3 Weeks
Construction	12 Weeks
Documentation	5 Weeks

Figure 3: Schedule

1.9 Procedure

The procedure of this thesis is as follows:

1. Inverstigate how to communicate with the PLC. This includes determining how to interpret the data that is sent from the PLC and how data should be sent to the PLC. It also includes which wireless technologies that are possible to use. This is done in chapter 2 and chapter 3.
2. Determining which hardware to use for the solution. This is based on the conclusions that are made in the previous step. This is done in chapter 4. Creating the applications. There are two applications created that are developed to fit the needs of the client. When doing this the different development tools are discussed and the applications created are described. This is done in chapter 5 and chapter 6.

2. Communication

There are two different scenarios when the PLC communicates with the PDA. One is the short range communication when PDA is used as a replacement for the operator terminals used today. The second is the long distance communication when PLC should be able to contact the PDA at any range.
The technologies that can be used in this application are limited to:

- What is available on the PDA market
- Possibility to be used with a PLC

2.1 SAIA S-Bus

S-Bus is the name of the communication protocol for the SAIA PLC. It can be used for both point-to-point communications and within a local master/slave network. When used in this application a serial interface (RS232 or RS485) is used.

When communicating with a PLC the application in the PDA needs to use the same protocol as the PLC. This communication is done with telegrams which are built up like byte-arrays. The telegrams used are:

- Read register
- Read flag
- Write a value to a register
- Write a value to a flag

These telegrams along with the respond telegrams from the PLC need to be known to be able to communicate.
When examining the structure of the telegrams Citect was used. Citect is a PC-program that can be used to connect a PC to a PLC. Citect is connected to the PLC via a serial port on the PC. The telegrams used for the communication could then be examined by reading the serial port. Using the results from reading the serial port together with the manual for SAIA S-Bus the structure of the diagrams could be decided.

The following telegrams are shown in hexadecimal to give a better overview. S-Bus uses CRC-16 error detection (see chapter 3).

2.1.1 Read register

Request (Figure 4)
Read the value of register on address 501 (Hex 01f5 = Dec 501).

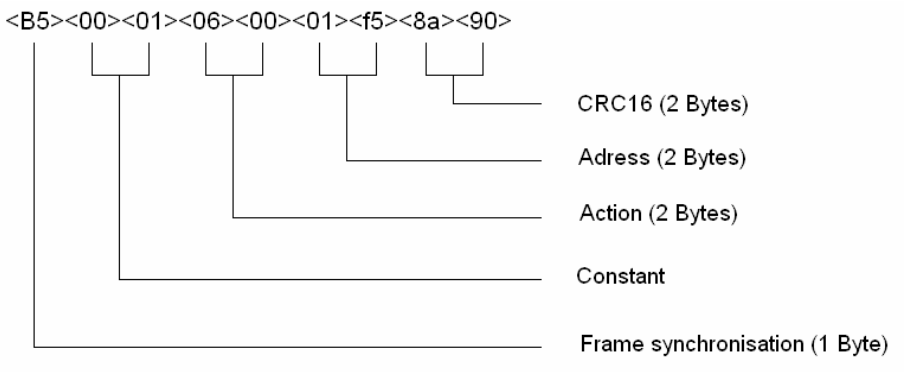


Figure 4: Read register telegram

Respond from PLC. (Figure 5)

The answer from the PLC indicates that the value of register 501 is 258.
(Hex 0102 = Dec 258)

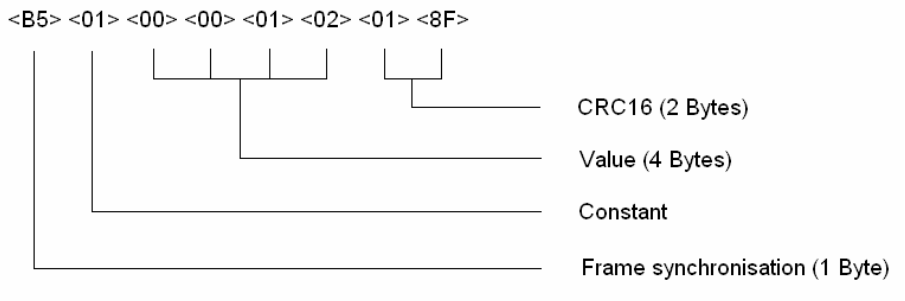


Figure 5: Respond from PLC on read register telgram

2.1.2 Read flag

Request (Figure 6)
Read the value on the flag with address 401. (Hex 0191 = Dec 401)

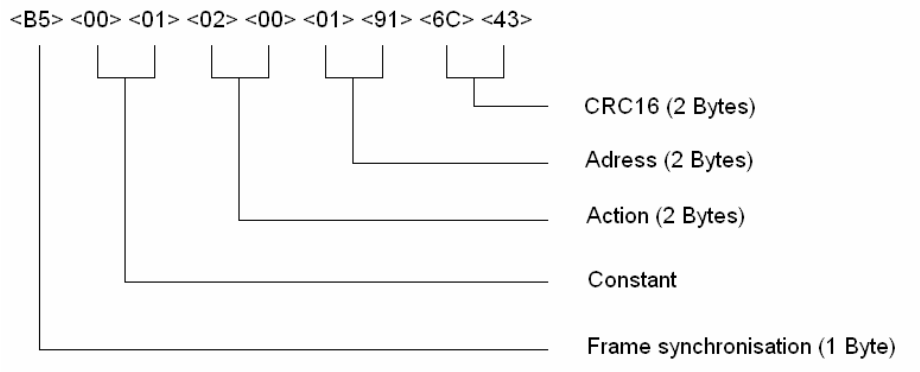


Figure 6: Read flag telegram

Respond from PLC (Figure 7)
The respond from the PLC indicates that the value is 1.

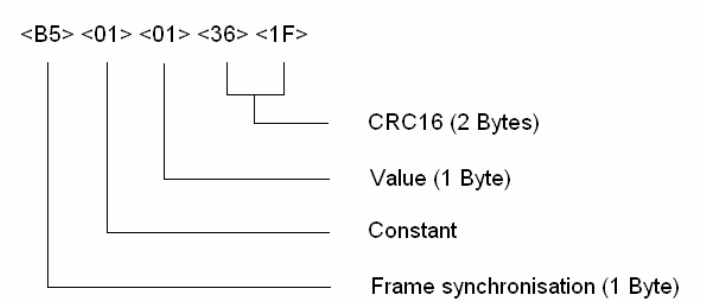


Figure 7: Respond from PLC on read flag telegram

2.1.3 Write to register

Request (Figure 8)
Write the value 2 to register 701.

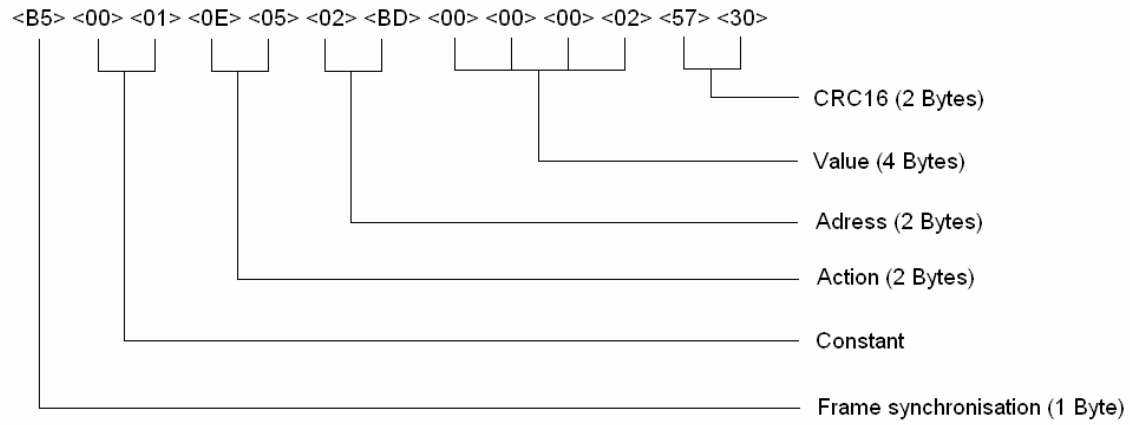


Figure8: Write to register telegram

Respond from PLC. (Figure 9)
The respond from the PLC does not include any useful data.

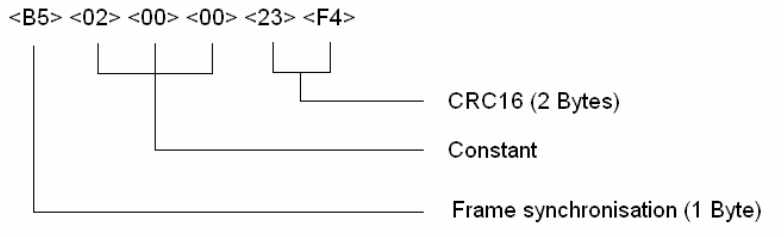


Figure 9: Respond from PLC on write to register telegram

2.1.4 Write to flag

Request (Figure 10)
Write the value 1 to flag 601.

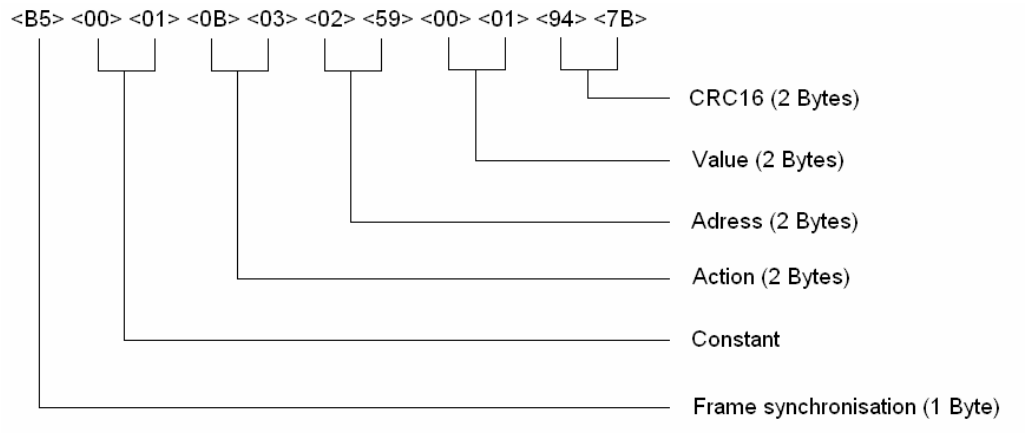


Figure 10: Write to flag telegram

Respond from PLC. (Figure 11)
The respond from the PLC does not include any useful data.

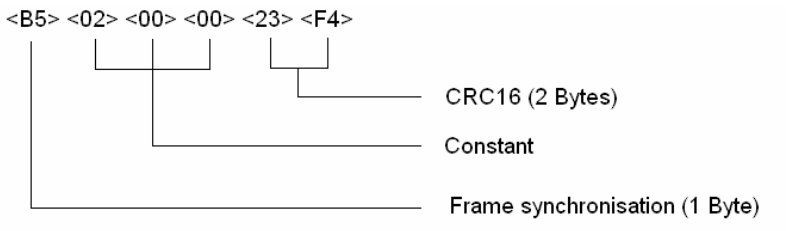


Figure 11: Respond from PLC on write to flag telegram

2.2 Short range communication

At short range the PDA is the initiator to communications with the PLC.

Scenario
The PDA sends a telegram to the PLC(Figure 12), for example read the value of a register. The PLC immediately responds with the value. The respond telegram from the PLC only includes the value of the requested register, it does not include any data indicating the address. This means that the PDA needs to keep track off the last requested value.

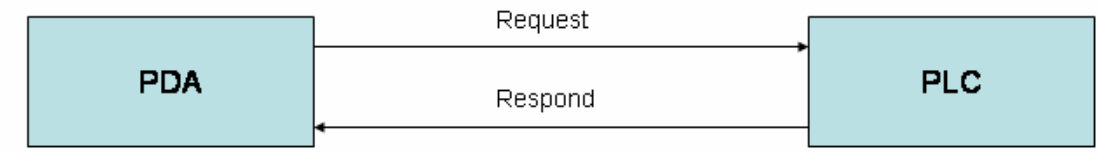


Figure 12: Short range communication

2.2.1 Technology

The technologies available on the PDA market today are:
- Wireless LAN
- IrDA
- Bluetooth

Wireless LAN

A wireless LAN (WLAN) is a flexible data communication system implemented as an extension to, or as an alternative for, a wired LAN within a building. Using electromagnetic waves, WLANs transmit and receive data over the air, minimizing the need for wired connections.

WLAN can not be used in this application since there are no WLAN devices today that can be connected to a PLC in a satisfactory way for this application.

IrDA

IrDA is a standard defined by the IrDA consortium. It specifies a way to wirelessly transfer data via infrared radiation. The IrDA specifications include standards for both the physical devices and the protocols they use to communicate with each other.

The problem with IrDA is that it needs a line-of-sight connection to work. In this type of application that is not always the case.

Bluetooth

Bluetooth allows compatible portable and stationary communications devices to communicate without using cables. The technology is based on a radio link that offers fast and reliable transmission of voice and data transmission. It doesn't require a line-of-sight connection in order to establish and maintain communication.

There are products on the market today that can be connected directly to a serial port and translate the serial data into Bluetooth signals.
Bluetooth will be used for the short range communication in this application.

The Bluetooth technology itself will not be discussed any further in this thesis, it has been done in several papers and books[3] before this one.

[3] Discovering Bluetooth – See references for details

2.3 Long range communication

The long range communication is used when the PLC needs to get the users attention when the PDA is out of range for the Bluetooth connection.
In the long range communication the PLC is the initiator.

2.3.1 Scenario

The PLC sends an alarm (Figure 13) to the user indicating that the temperature is to low in one part of a building. A problem like this could often be solved by changing a reference value. Changing the reference value to a higher temperature will make the heating system work harder and the temperature in that part of the building will increase.

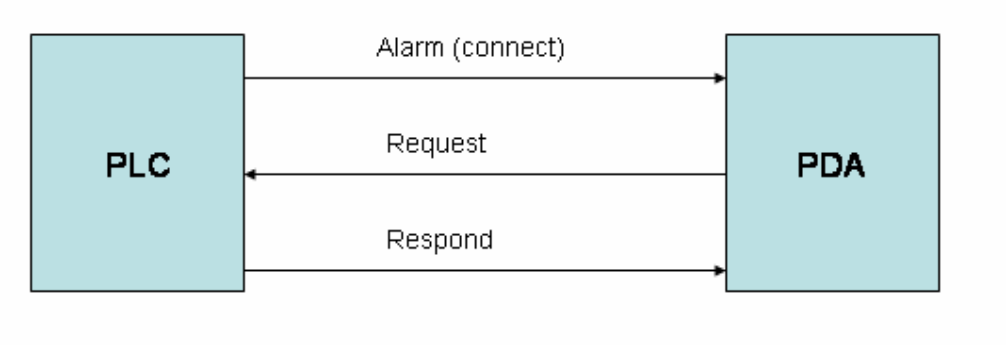


Figure 13: Long range communication

2.3.2 Technology

The idea is to use a modem that is connected to the PLC. When the PLC needs to contact the PDA, the modem calls the predefined telephone number (the PDA) and creates a connection. The modem also sends a predefined ASCII message that lets the PDA know which system that is calling. When the connection is up, the PDA can start to communicate with the PLC in a way similar to the short range communication.

3. CRC

Cyclic redundancy checking is a method of checking for errors in data that has been transmitted on a communications link. A sending device applies a polynomial[4] to a block of data that is to be transmitted and appends the resulting cyclic redundancy code (CRC) to the block. The receiving end applies the same polynomial to the data and compares its result with the result appended by the sender. If they agree, the data has been received successfully.

3.1 Example

The message 1110 0110 should be transmitted using CRC for error detection. A generator polynomial 11001 will be used.

When the polynomial is n bits long the CRC will be n-1 bits long. This means that in this example the CRC will be 4 bits long since the polynomial is 5 bits.

Sending

1. Add as many zero's to the message as the length of the CRC = 4 -> 1110 0110 0000
2. Calculate CRC using modulo-2 arithmetic
3. Add the CRC to the message and send

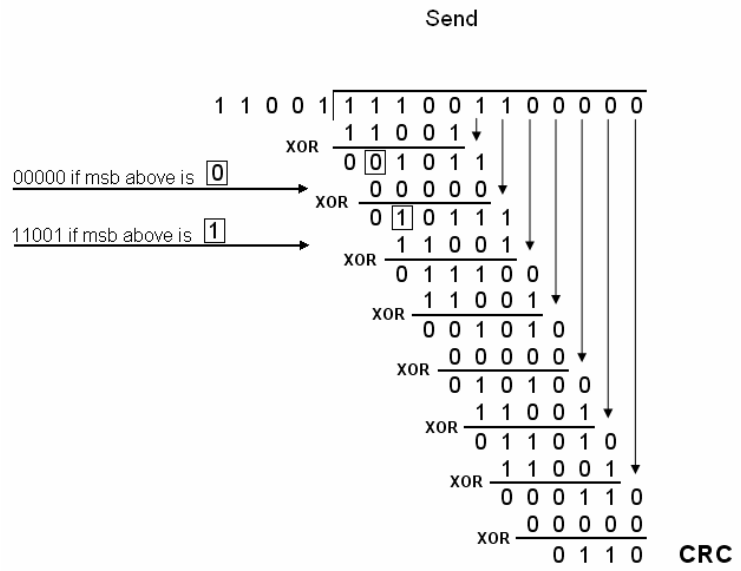


Figure 14: CRC

The result/CRC is 0110. The message including the CRC is then:
1110 0110 0110

[4] See glossary for details

Receiving
At the receiving end the same polynomial as on the sending side is used. The CRC is calculated using modulo-2 arithmetic. If the CRC is 0 the message has been received correctly.

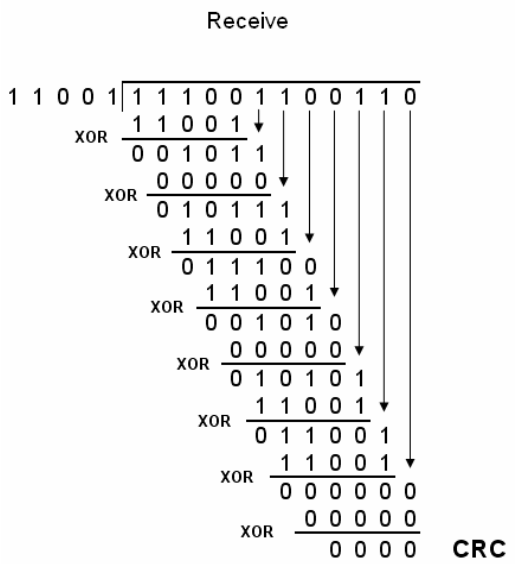


Figure 15: CRC

The CRC is 0 which means that the message has been received correctly.

3.2 CRC-16

CRC-16 works in the same way as the example above but with CRC-16 the polynomial is 1 1000 0000 0000 0101.
CRC-16 will detect all error bursts of less than 16 bits and most error bursts greater than or equal to 16 bits.

4. Hardware

4.1 Compaq IPAQ 3970

When deciding which PDA to use in this project there were a few features that were required from the unit. It should be equipped with Bluetooth and GSM to be able to communicate with the PLC. It should be easy to handle for windows-users. It should be able to present the data collected from the PLC in a satisfactory way.

Compaq IPAQ 3970 (Figure 16) uses the operating system Microsoft pocketPC which is a PDA version of windows. This means that a windows-user has no problem learning how to use the IPAQ.
It is equipped with Bluetooth but no GSM. A GSM cradle can be purchased separately.

The Bluetooth module on the PDA is addressed by using a virtual comport.

Figure 16: Compaq IPAQ 3970

There is one problem with using the IPAQ 3970 for an application like this.
The Bluetooth module on this PDA only has an output power of 0 dBm which gives a range 10 – 30 m which may be a bit short in some situations. If you use an output power of 20 dBm instead you get a range > 100 m.
At the time this project was started no PDA on the market had an output power of 20 dBm.

4.2 Connect Blue Serial Port Adapter

To make Bluetooth-communication possible between the PLC and the PDA, the PLC needs a communication interface.

Figure 17: Connect Blue Serial Port Adapter

The Connect Blue Serial Port Adapter (Figure 17) is a ready to use, out of the box, product designed for industrial use.
The PLC uses the RS232 standard which means that it sends and receives data in a serial way. The Serial Port Adapter from Connect blue transforms these serial signals to Bluetooth signals and Bluetooth signals into serial the other way.

The serial port adapter can be configured to fit your demands by using Serial Port Adapter Connection Wizard (Figure 18), which is delivered together with the product.

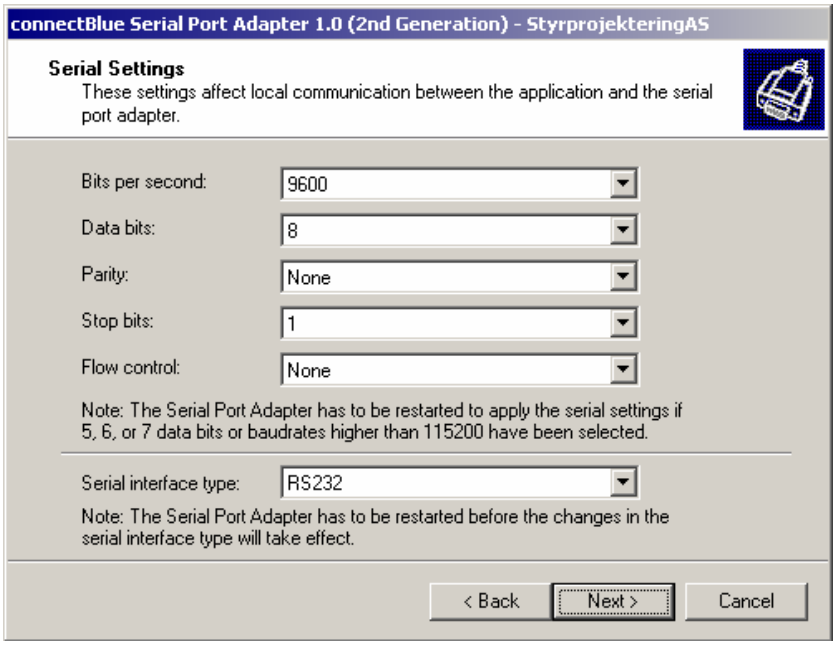


Figure 18: Serial Port Adapter Connection Wizard

When the serial port adapter and the Bluetooth in the PDA are configured in the right way the Bluetooth is invisible to the user. The user can send and receive data on the serial port as if there was an ordinary serial cable connected.

4.3 SIXNET PLC Self-dialing modem

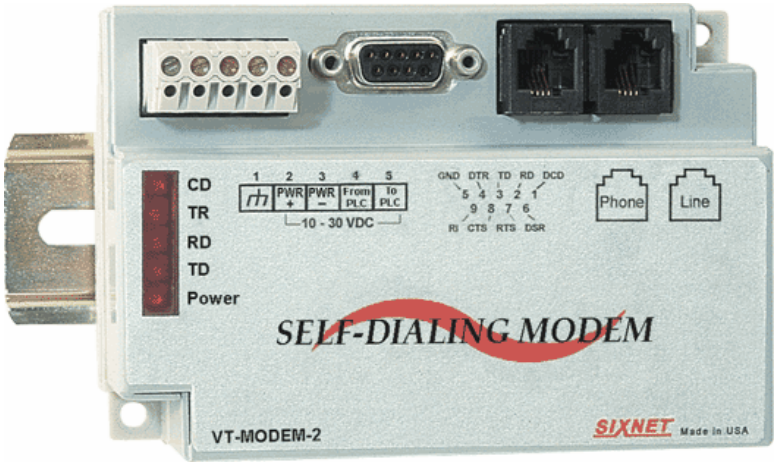


Figure 19: SIXNET PLC Self-dialing modem

The VT-modem from SIXNET (Figure 19) allows easy access to PLC's and other devices via telephone connections. The self-dialing modem is triggered by a switch closure or PLC output signal. It dials a pre-stored phone number and optionally identifies itself with a pre-stored ASCII message.

Once a connection has been established, the PLC's system port is connected to the computer (PDA) at the other end of the phone line and may be polled by that computer as if the computer had initiated the call.

The call will terminate when either:

- The computer completes its polling and hangs up
- The modem discrete input is turned off
- A telephone line problem disrupts the call

5. PDA development tools

5.1 Microsoft Embedded Visual Basic

Embedded Visual Basic version 3.0 is provided as a part of the Embedded Visual Tools 3.0 development software from Microsoft[5]. This is free to download from Microsoft. Embedded Visual Basic is a subset of the language used by the desktop version of Visual Basic. It is intended to provide a significant portion of robust power of Visual Basic with the portability and easy of use of VBScript.
The Embedded Visual Basic language has a full IDE and forms designer in the style of the full Visual Basic environment.

The structure of an Embedded Visual Basic application will be very familiar to any developer who has used the Desktop version of Visual Basic. However there are a number of significant limitations compared to Visual Basic version 6.0:
- Classes (CLS files) are not supported
- The With statement is not supported
- User-defined types (Type…..End Type) are not supported
- Embedded Visual Basic cannot be used to create dynamic link libraries (DLL's)

The run-time system of Embedded Visual Basic is a subset of the run-time system for Visual Basic 6.0. There is no true compilation. The development environment creates a pseudo-code that is interpreted at run-time.
Because it is interpreted, an Embedded Visual Basic application is CPU-independent.

5.1.1 Problems

The most important part of this application is the communication with the PLC. When the PDA communicates with the PLC via Bluetooth a virtual com port is used in the Visual Basic application. The communication is done with telegrams build up as byte-arrays.

When you try to send a byte-array in an Embedded Visual Basic application only the first two bytes gets transferred then the application locks the PDA. The only way to send the whole byte array is to send it one byte at a time. This makes the communication very slow. According to the help in Embedded Visual Basic the serial component should be able to handle byte-arrays but in real life it does not work.

When discussing this problem with other developers at several msdn-newsgroups[6] the conclusion was made that this problem could not be solved in Embedded Visual Basic.

[5] http://www.microsoft.com/windowsmobile/resources/downloads/developer
[6] http://msdn.microsoft.com/newsgroups/

5.2 Microsoft Visual Basic .NET with Compact Framework

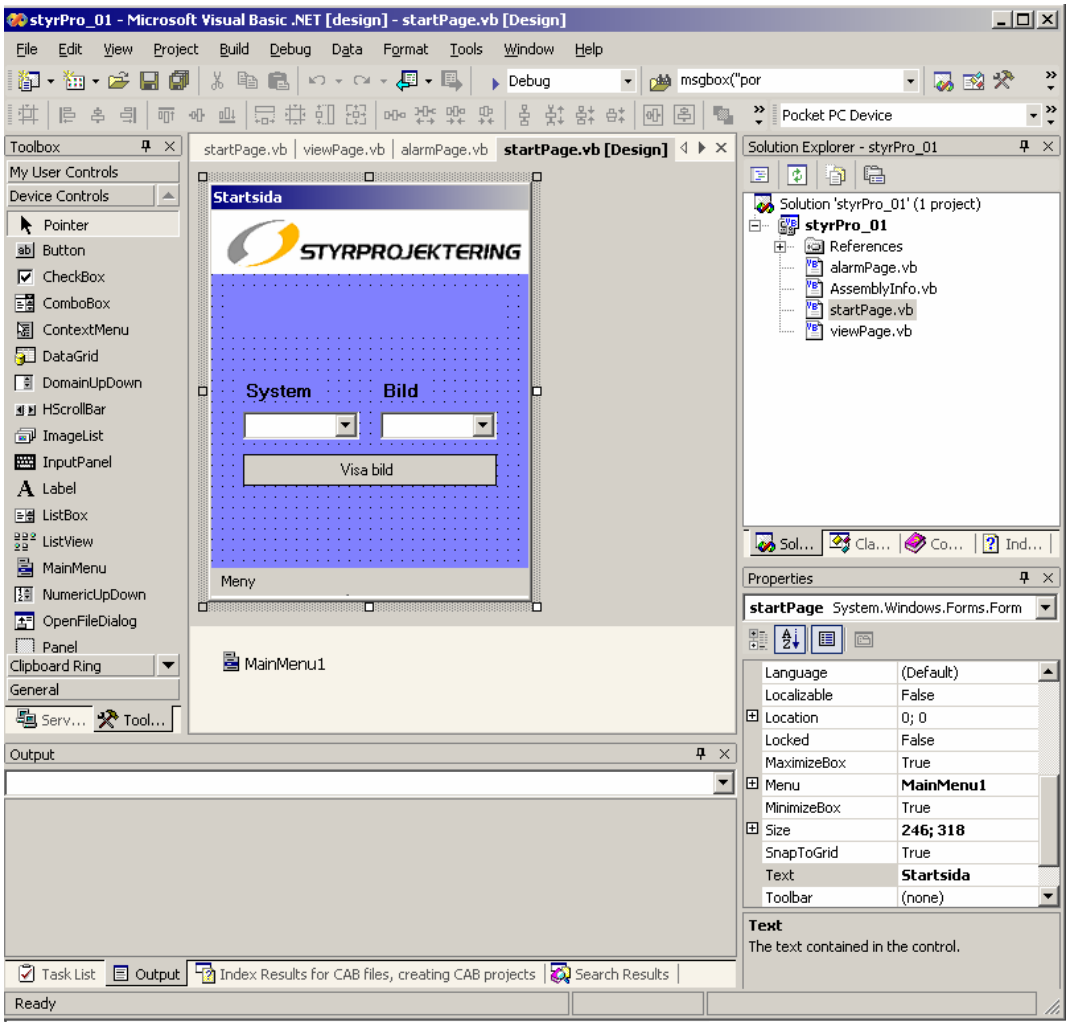


Figure 20: Microsoft Visual Studio .NET with Compact framework

The Compact Framework is included in Visual Studio .NET 2003 (Figure 20). It is a subset of the desktop .NET Framework and it includes many of the features key features fount in the desktop version.

Visual Studio .NET enables mobile-application developers to use the same unified development environment for developing mobile applications that they currently use to build desktop and server applications. Since the base templates for Pocket PC and Windows CE .NET – based applications are included in an additional "Smart device Application" project type, developers can begin writing applications using the same skills and practises they already use for desktop programming.

Applications running on the .NET Compact Framework will execute as native code, which is produced by the built-in just-in-time (JIT) compiler. The use of JIT technololy provides higher performance code execution than other device programming systems with code interpreters.

5.2.2 Problems

Visual Studio .NET 2003 does not include a control for serial communications. This problem is solved by downloading the NETComm.ocx component from hardandsoftware[7]. The component has the same functionality as the serial port in Visual Basic 6 which means that it can handle byte-arrays.

[7]http://www.hardandsoftware.net/

6. Construction

There are two separate applications created in this project:

- A programming tool that is used to program the PDA.
- The PDA-application

Both applications are developed in Visual Basic.

6.1 Programming tool

An installation can include a large number of units. This means that there are several registers and flags to be presented in the PDA. If there is no functional structure of this presentation a user can spend a lot of time finding a desired value.
The programming tool is used to create a database with flags and registers in an installation. It is also used to build a menu-structure that makes the presentation in the PDA more user-friendly.

For every installation a new database is created. The application in the PDA uses this database when interacting with that system.

6.1.1 Structure

The structure (Figure 21) is the way a user can navigate to find a certain value in the PDA. The different parts to consider here are:

- Installation
  The installation is the whole project
- PLC
  An installation can include several PLC's
- System
  Every PLC can have several systems
- View
  Every system has several views. The variables are added here. Since the display on the PDA can not contain an endless number of variables the view is split up into pages with a maximum of 8 variables on each page.

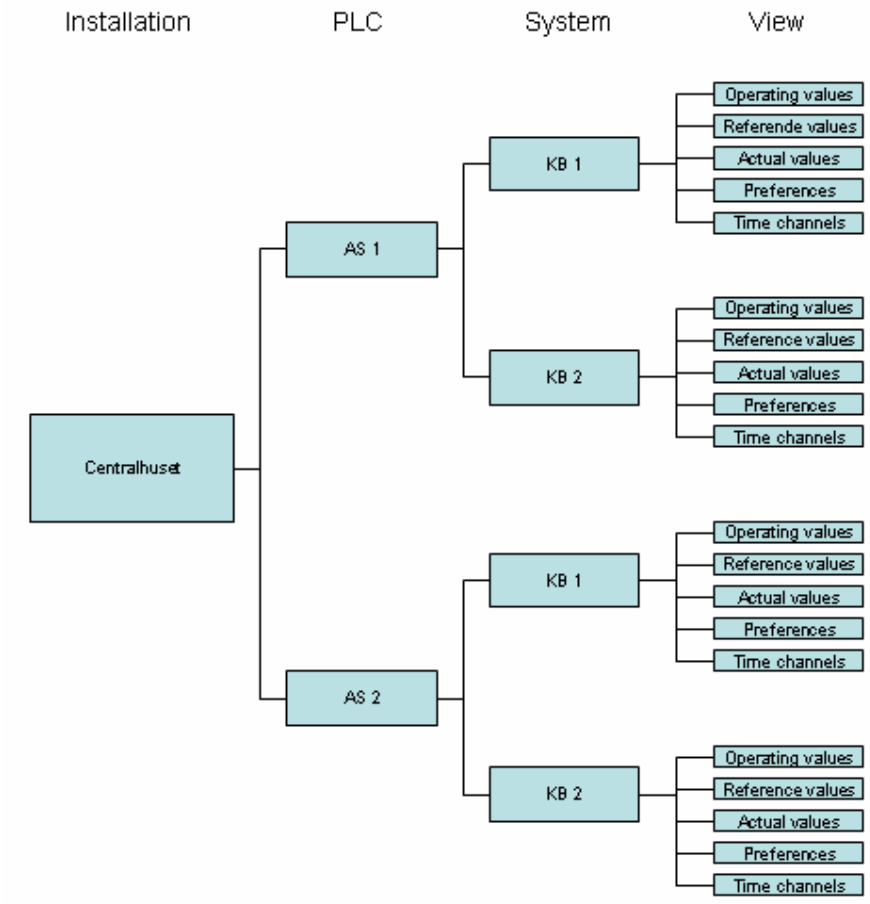


Figure 21: Installation layout

6.1.1 Register/flag type

Register and flag types are used to make it easier for a user to understand the data that is presented on the PDA screen. Instead of showing the variables value in a numeric form the value can be shown in a way that gives the user a little bit more information.
Example:
A register represents a switch in an installation. The switch can have three different states. The status of the switch is saved as 0, 1 or 2 in the PLC. When the user wants to check the status of the switch these numeric values doesn't say much to the user. Instead a register type is created that says:

- 0 = Manual
  The switch is always ON
- 1 = OFF
  The switch is always OFF
- 2 = Automatic
  The switch is controlled by the conditions in the PLC.

This makes the presentation a lot easier to understand for the user.

6.1.2 Variables

There are three types of variables. Registers and flags, with a value that is collected from the PLC, and text that is used for headings.
A variable contains the following:

- **Address**
  The variables address in the PLC.
  Text variables do not have an address.
- **Name**
  The variable name shown to the user of the PDA
- **Unit**
  The unit for the value of the variable (°C, Hz, % etc)
  Only used for registers that are displayed as decimal values.
- **Register, Flag or Text**
  The variables are saved in the PLC as either flags or registers. Text is used for headings in the PDA.
- **Register/Flag Type**
  Describes how the value of a variable is shown to the user.
  Not used for text.
- **Decimal form**
  The values in the PLC are saved as integers. The decimal form is used to display these integers in different ways. If the integer value is 1234 and the decimal form is ###,# the PDA will show 123,4. If the decimal form is ##:## the PDA will show 12:34
  Only used for registers shown in numeric form.
- **Controllable(Yes/No)**
  Some values should be able to be changed from the PDA. This attributes tells if the variable value can be changed.

6.1.4 Alarms

An alarm-file is created as a separate part of the database. The alarms are saved in the PLC as flags. These alarms are connected together into a register which may contain up to 25 alarms(Figure 22). This is done with the programming software to the SAIA PLC which is called PG5. Every alarm is represented as a bit in the register. This solution is used to save overhead in the communication. Instead of reading every alarm the PDA can read the register that contains several alarms. If the registers value is 0 then none of the alarms are active.

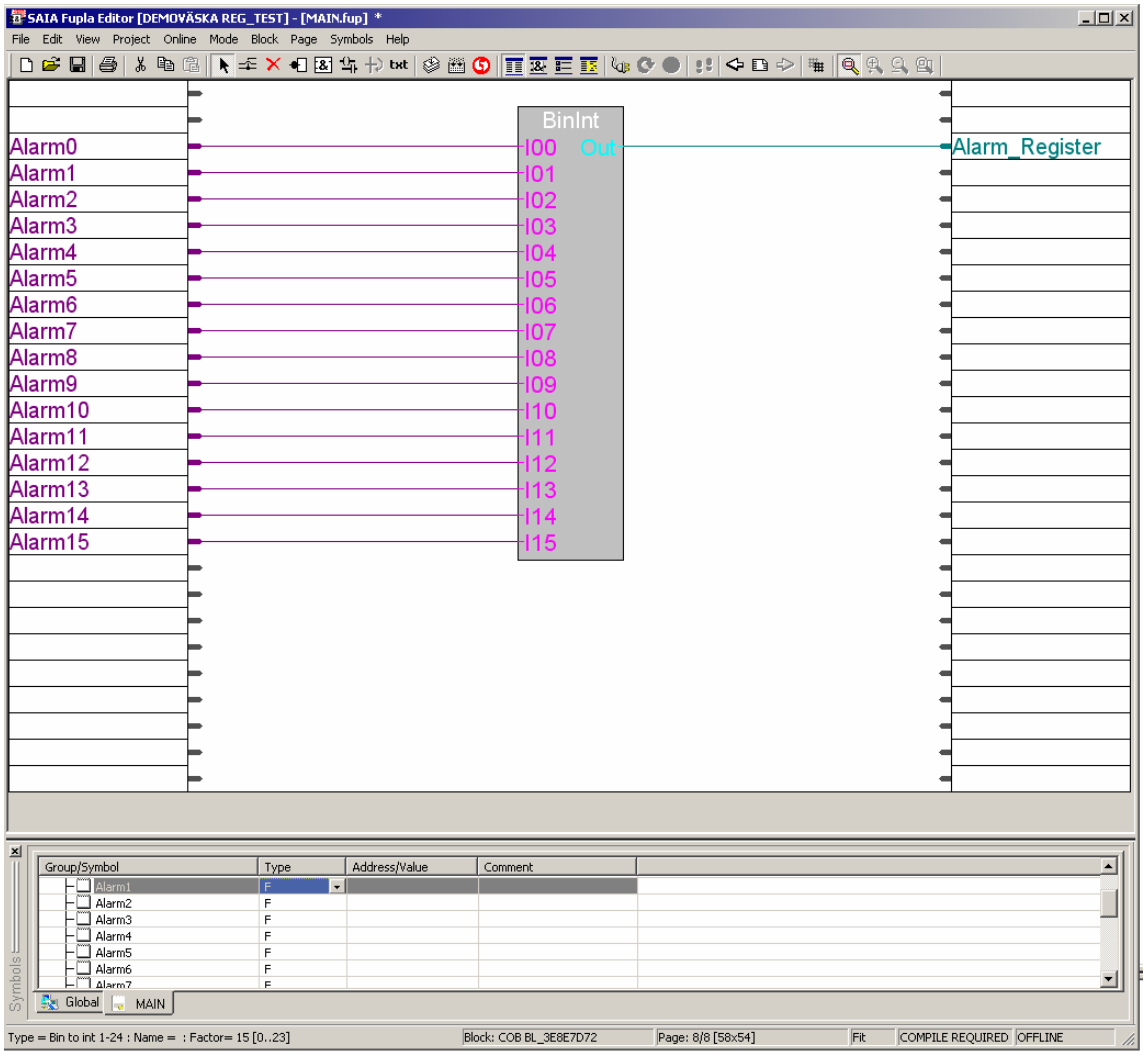


Figure 22: Binary to integer converter in PG5

A binary to integer converter is used. If an alarm is active the output to Alarm_Reg is the value of the bit for that alarm.

Example:

Alarm2 and Alarm6, which are connected to bits 2 and 6, are active. The output to Alarm_Reg will be $2^2 + 2^6$ which equals the decimal value 4 + 64 = 68. This value is interpreted by the application in the PDA and tells the user which alarms are active.

An alarm variable contains the following:

- **Address**
  Address of a collection of several alarms.
- **Bit**
  The bit of the specific alarm
- **Name**
  The alarm name shown to the user of the PDA
- **Priority (A, B, C, D, E)**
  How important the alarm is. A is the most important.
- **Text**
  Description about the alarm

6.1.5 Database

The database is build up in text files. This was chosen since there is no need for additional software to create text files and the PDA will have no problem understanding them. One file with the extension .proj is also created for each project. This is the only file that will be shown when opening a project from the PDA.
For every PLC the following files are created:

- One alarm file
- One register-type file
- One flag-type file
- One system-file that describes the systems and the views
- One file for each view
- One .proj file

6.1.6 The application

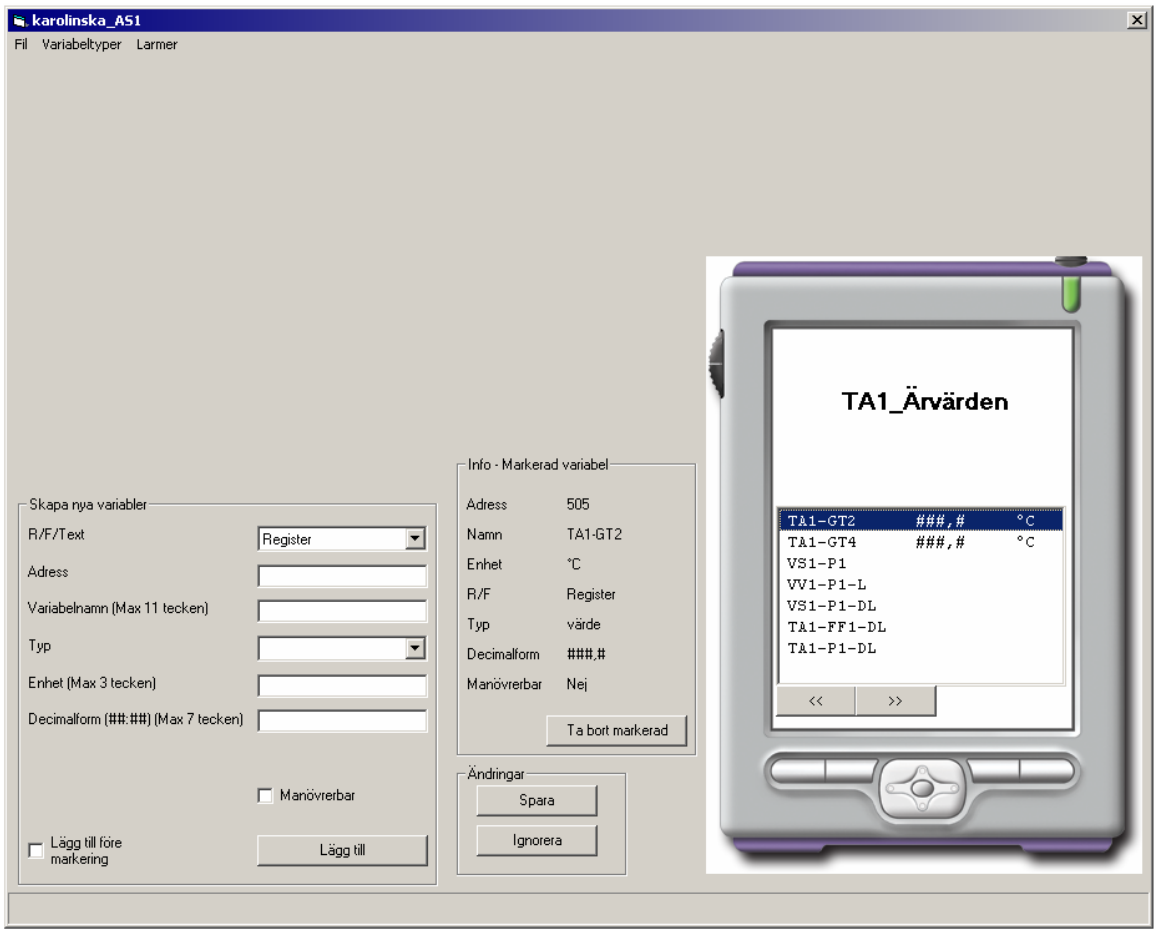


Figure 23: Programming tool

The tool(Figure 23) lets the programmer create the database for the PDA-application. It also provides a preview that shows how the values will be presented in the PDA. When the database is created it is downloaded to the PDA and the Variables will be presented in the same way as in the tool.

6.2 PDA application

In this thesis the PDA application will be limited to only use the short range communication. The long range communication can be implemented in the future.

The application is written in Visual Basic .NET 2003. The project type is "smart device application". When debugging a smart device application you can choose either to run the application on the actual device or in an emulator. When using the actual device the project is first downloaded to the PDA and then the debugging can start. During testing this can be a bit time consuming compared to debugging an ordinary desktop application. The emulator can not be used when the application uses communications like the com port in this application.

6.2.1 Test applications

Since the debugging and testing is a bit problematic when writing smart device applications the test-applications was first written as ordinary desktop applications and tested on a PC.

CRC-16
The S-Bus protocol used by the PLC communicates via telegrams using CRC-16 as error detection. The telegrams are built up as byte arrays. The first test application was created to calculate CRC-16.
A CRC-function[8] was created that takes a buffer, calculates and returns the CRC.

Communication
The next application was written to test the communication with the PLC. The PC application uses a serial cable to communicate instead of the Bluetooth communication that is used in the PDA application.
The communication is done using a virtual com port in the Visual Basic application.
This application was later extended to simulate the whole PDA application.

[8] See appendix

6.2.2 The application

After the desktop-applications were tested the writing of the PDA application(Figure 24) could start. The first version of the application was written in Embedded Visual Basic. After a few weeks of developing several problems occurred, the most important problem was the communication as mentioned earlier. When asking around how to solve this problem the use of smart device extensions with Visual Studio .NET was suggested by several developers.

Start page
When the user executes the application a start page is opened. The first thing to do here is to select which installation to open. This is done by selecting “open project” in the menu. When this is done the database is loaded into the PDA. The files in the database are opened and saved in buffers. When the project is opened the different systems are presented. Selecting a system will show the views of that system. The views are where the variables are presented.

Figure 24: PDA application first page

When system and view are selected the user presses the show view button and the next page is shown, the view page.

View page
When the view(Figure 25) is selected a buffer is filled with the variables of the view. The first eight variables are shown on the screen. A view can include several pages which are manoeuvred by using the arrow buttons.

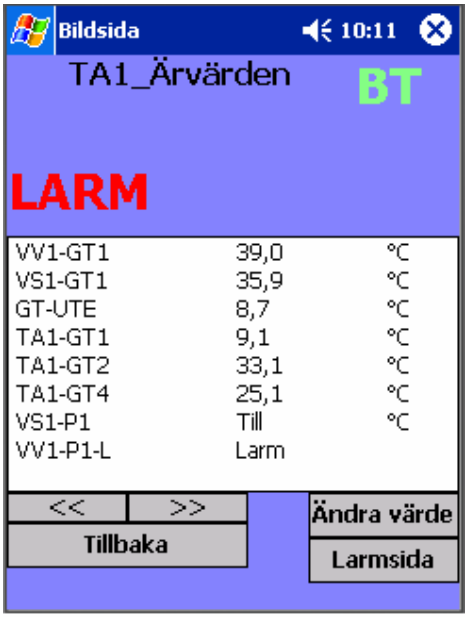


Figure 25: PDA application view page

When the variables are loaded the communication starts.

When sending and receiving data over Bluetooth a virtual comport in the PDA is used. The application needs a comport component to do this. Since Visual Basic .NET comes without any comport component the CFSerial component from hard and software is used. This component is free to download but comes without any help. The functionality is similar to the MSComm component in VB6.

The first approach to the communication was to send a telegram to the PLC and then wait for a OnComm[9] event on the serial port. The OnComm event indicates that there is something to read on the serial port. This approach turned out to be problematic in this application. If the program is in the middle of receiving a respond from the PLC and a user presses a button at the same time, the application hangs the PDA.
Another problem that came up when using this approach was that timers could not be used. Since no help was included with the downloaded serial port component this problem could not be solved without changing the communication approach.

[9] See Glossory

The next approach was to use a timer for controlling when to read the incoming data. When the timer is enabled it raises an event every 100 milliseconds. This controls when to read the com port.
When using the Oncomm approach the serial port tells the application that there is data on the serial port, using the timer approach means that the application has to check if there is data on the serial port.

The functionally of the timer was extended to control every action in the application. For this a variable called nextAction is used. The value of this variable controls what will be done when the timer raises an event.

1. Request the next value from the PLC[10]
2. Next page
3. Previous page
4. Read the com port[11]
5. Close the view page and go back to the startPage
6. Change the value of a selected variable
7. Read alarms[12]
8. Open alarm page

The reason for letting everything be controlled in this way was to make the application more stable. The different actions becomes more controlled than if for example pressing a button starts one routine while the application is in the middle of executing another.

The nextAction variable is set to different values depending on what is executed. If a request is sent to the PLC the nextAction variable is set to 4 (Read the com port). The next time the timer raises an event, the application knows that it should receive data from the PLC.

Alarms are checked at given intervals. The alarms are saved in alarm collections as mentioned earlier. If the value of an alarm collection is greater than zero this means that one or several of the alarms in that collection is active. If this is the case it is indicated on the screen that there are active alarms and a button is made visible that allows the user to open the alarm page.

If the user wants to change the value of a variable it is first checked if it is allowed to be changed. The next step is to make sure that the input from the user is correct.
The input possibilities depend on the type of the variable. If the variable is presented as either On or Off, the user can only choose between On and Off when changing the value of the variable. If the variable is shown in decimal form a field is shown were the user can write the new value.

[10,11,12] See appendix

Alarm page

The alarm page(Figure 26) is controlled by a timer in a similar way to the view page. The different actions here are:

1. Go back
2. Get the value of the next alarm collection
3. Receive alarm collection value
4. Sort alarms

As mentioned earlier the alarms are saved as collections of several alarms. The active alarms are calculated from the value of the alarm collection. This is done in sort alarms.

This routine takes the value of an alarm collection and perform a bit calculation that tells which bits are high. These bits are then compared to the alarms in the alarmbuffer which include every alarm in the project.
The info about the active alarms is then presented on the screen.

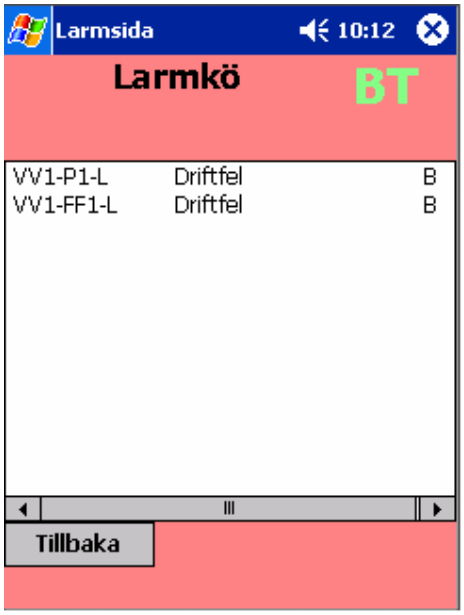


Figure 26: PDA application alarm page

Sending telegrams
The telegrams sent to the PLC are built up as byte arrays. These arrays are created in the sending routine. The different telegrams are described earlier in this paper.
The array is first built up with all bytes except the two CRC-16 bytes. This array is then sent to a CRC-16 routine that calculates the CRC and ads it to the array.

Before sending the telegram to the PLC, the PDA needs to make sure that it is not out of range for the Bluetooth connection. This is done by reading the Data set ready on the serial port. If it is high it means that the connection is up. If the connection is lost it is indicated to the user.

When sending the telegram the PDA needs to remember what was sent to be able to understand the answer from the PLC. The only useful information in the answer is a value. When the telegram is sent the nextAction is set to read the com port.

Receiving telegrams
Since the application knows what should be coming in on the serial port it also knows the length of the expected data. It is first checked if the incoming data matches this length. If the length is incorrect the application waits 100 milliseconds and checks again. After three unsuccessful tries the application moves on to sending the next request.

When the incoming data length is correct the application starts handling the telegram. First the CRC is checked. If the CRC is zero then the telegram has been received correct. The value of the incoming data is saved. Next it is controlled how the variable should be presented to the user. This is done by checking the register/flag type. When the value is formatted the right way it is presented on the screen.

After a telegram is received and presented correctly the nextAction variable is set to get next value.

7. Result and Conclusions

The PLC uses a protocol called S-Bus that uses telegrams to communicate. The structure of these telegrams could be examined by test where a PLC was connected to the serial port on a PC. The data on the serial port were then examined to figure out how the protocol works.

The PDA-application does not include the long range communication. This will be added if a decision is made to start using the product in future installations. According to the information about the SIXNET PLC Self-dialing modem this communication should not be a problem using a similar technique as is used for the Bluetooth communication.

The development of the PDA-application turned out to be more problematic than expected. These problems were mostly communication problems. This resulted in more developing time than what was expected at the beginning of the project. The programming tool also turned out to be a bit more complex than expected. These problems resulted in that the time-schedule for the project wasn't fulfilled.

The solution has been tested in an environment simulating a real installation. These tests included all scenarios that can com up in real life. The conclusion from the tests was that the product works without any errors.

The PDA is able to communicate with the PLC without having to change the communication protocol used by the PLC.
The short range communication works just as good as the wired communication used today.
The user interface in the PDA is easy to understand.

The solution will decrease cost since the same unit can be used to communicate with several installations.

8. Discussion

8.1 Bluetooth

The Bluetooth module in the PDA only has an output power of 0dBm which gives a range of 10-30 m. This might be a bit short in some situations. If a PDA is released on the market that has a more powerful Bluetooth module, it would be better for an application like this.
The serial port adapter is available with higher output power.

Another solution would be to use Bluetooth access points that are installed around in a building. This would take care of the problem with range-problem but it would also increase cost.

8.2 Practical use

As the product looks now it can be used as a replacement for the operator terminals used today. It has the same functionality as the operator terminals but it uses wireless communication to exchange data with the PLC. It also has the possibility to be used for several installations. If the long distance communication is added the more functionality will be added to the product since the alarm handling will improve.
Even without the long distance communication the product can be a good solution in many installations.

8.3 Security issues

Since the PDA is communicating with the PLC with a wireless interface the possibility for unauthorized persons to interact with the PLC increases.
This problem is solved by only letting the Bluetooth unit connected to the PLC allow connections from specific Bluetooth device addresses. This means that if the PLC side is set up to only accept connections from a certain PDA, no other unit can connect to the PLC.

Glossary

Reference value
The value of a variable wanted at a given instant, under specified contitions

Polynomial
A mathematical expression consisting of a sum of terms, each term including a variable or variables raised to a power and multiplied by a coefficient. The Polynomial for CRC-16 is $X^{16} + X^{15} + X^2 + 1 = 1\ 1000\ 0000\ 0000\ 0101$

PDA
Personal Digital Assistant. A small handheld computer like an IPAQ or a Palm.

PLC
Programmable Logic Controller

OnComm
The OnComm event is generated when either a communication event or an error has occurred. For example if there are data to receive on a serial port the OnComm event is generated.

References

Printed

Fred Halsall - Data communications, Computer Networks and Open systems
ISBN:0-201-42293-X

Michael Miller – Discovering Bluetooth
ISBN: 0-7821-2972-2

Internet

Connect Blue homepage
The home page of the distributor of the Bluetooth Serial Port Adapter
www.connectblue.se

SIXNET homepage
The home page of the distributor of the PLC self dialing modem
www.sixnetio.com

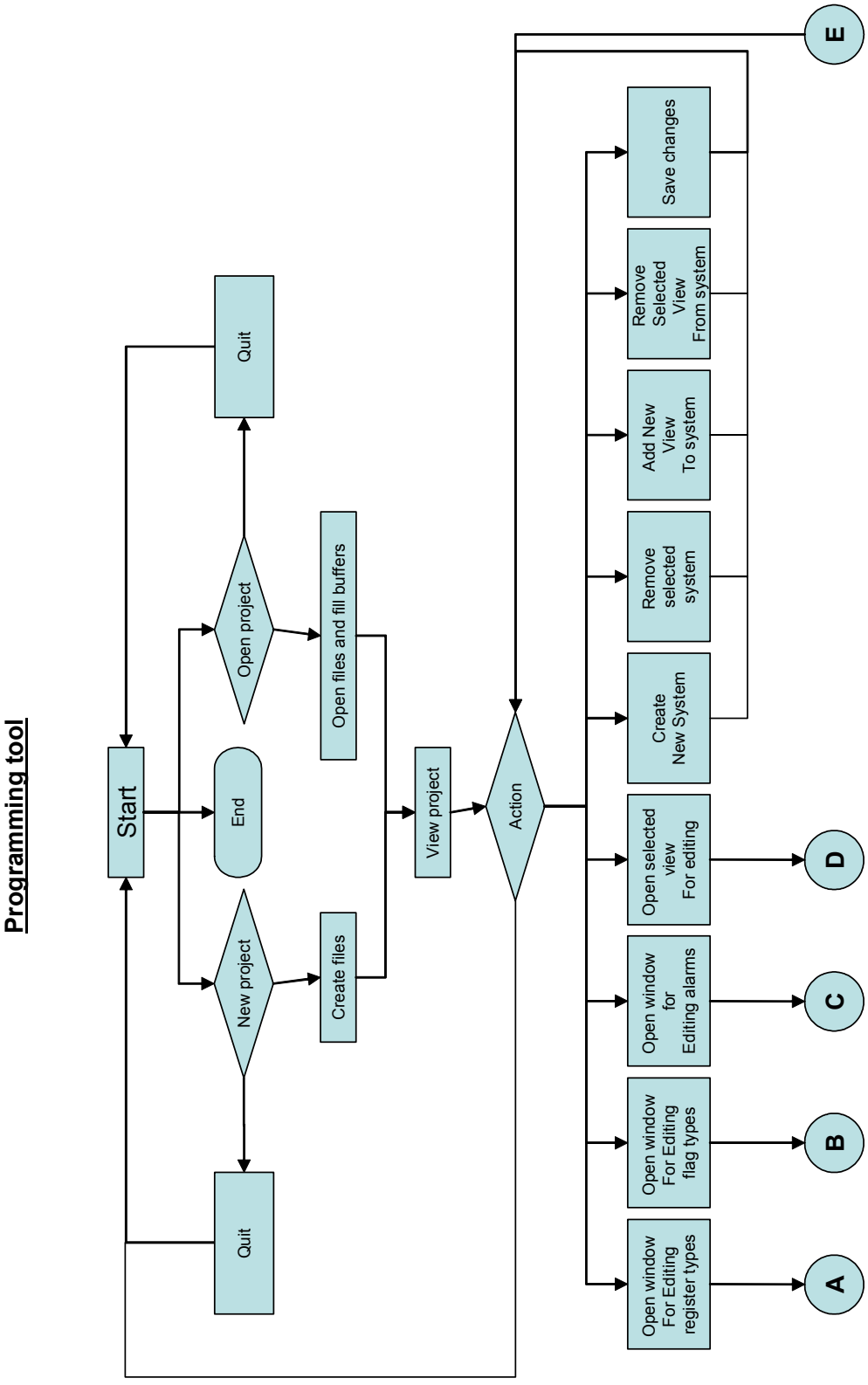
Malthe Winje
A distributor of the SAIA-PLC. The manual for SAIA-SBus can be downloaded here.
www.malthe-winje.se

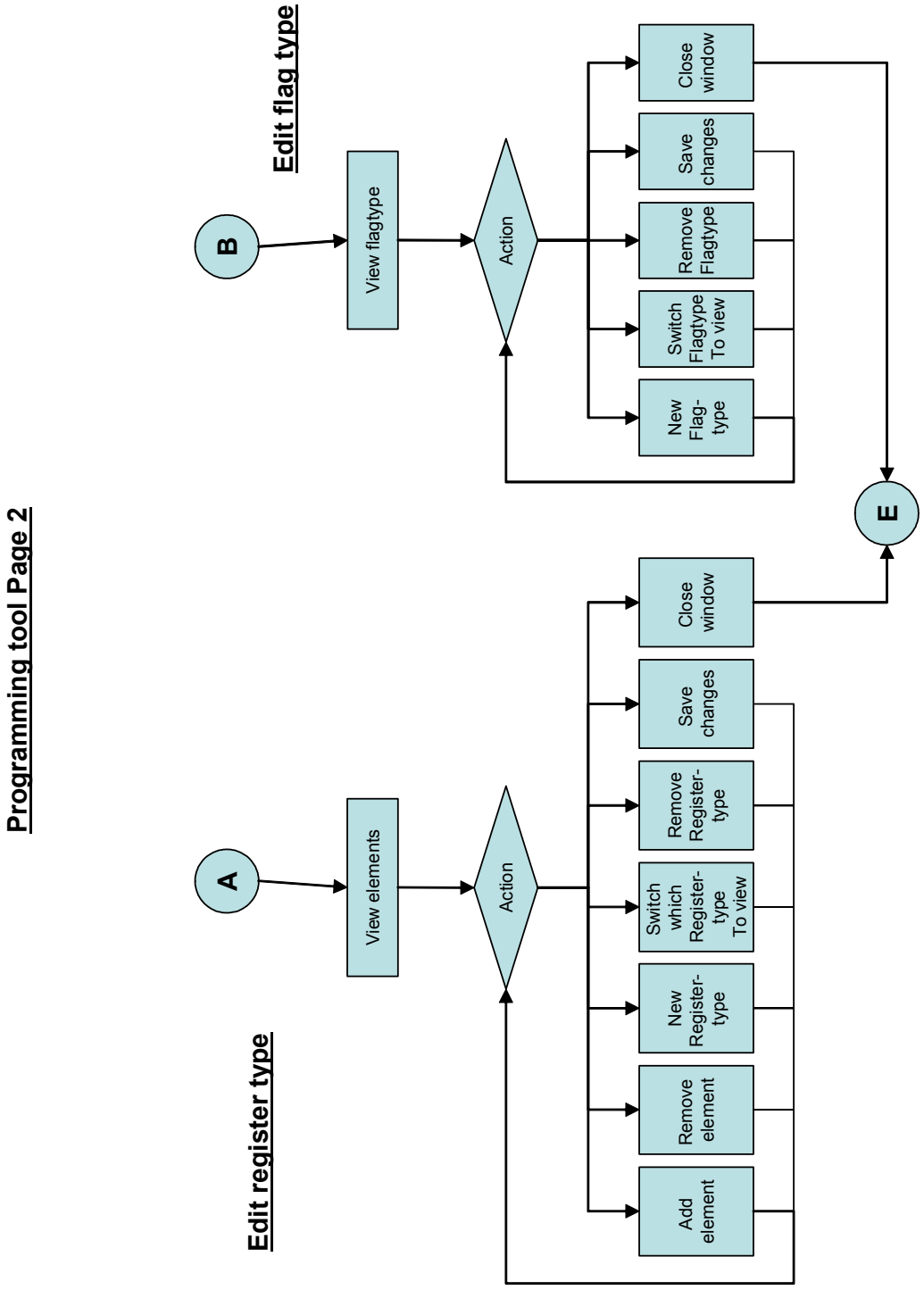
Hard and software
A home page were the serial port component for Visual Basic .NET Compact Framework can be downloaded
www.hardandsoftware.com

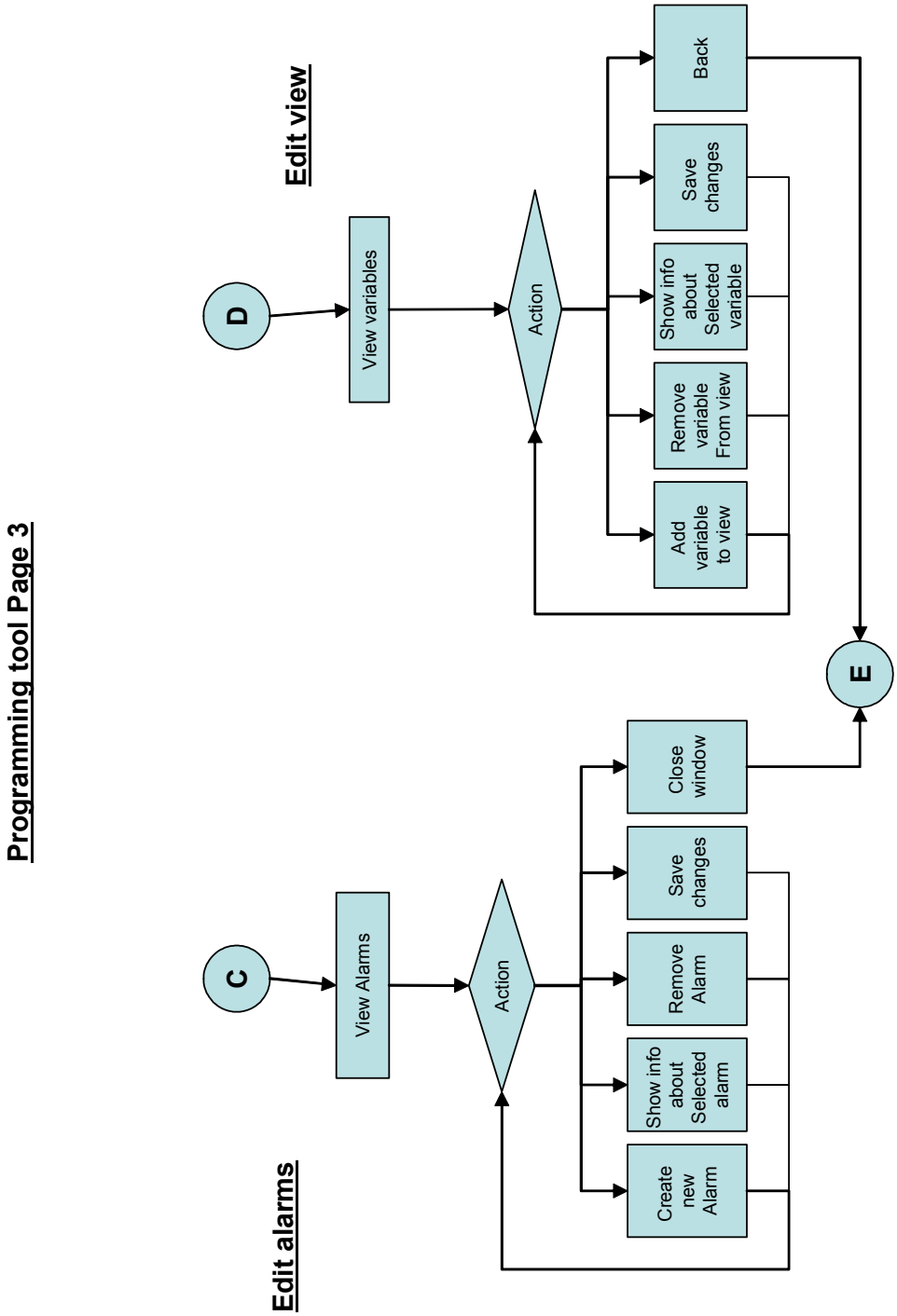
Devbuzz
A page for pocketPC developers with a lot of tips and code snippets.
www.devbuzz.com

Embedded visual tools download
A Microsoft download site where Embedded Visual Tools can be downloaded.
http://www.microsoft.com/windowsmobile/resources/downloads/developer

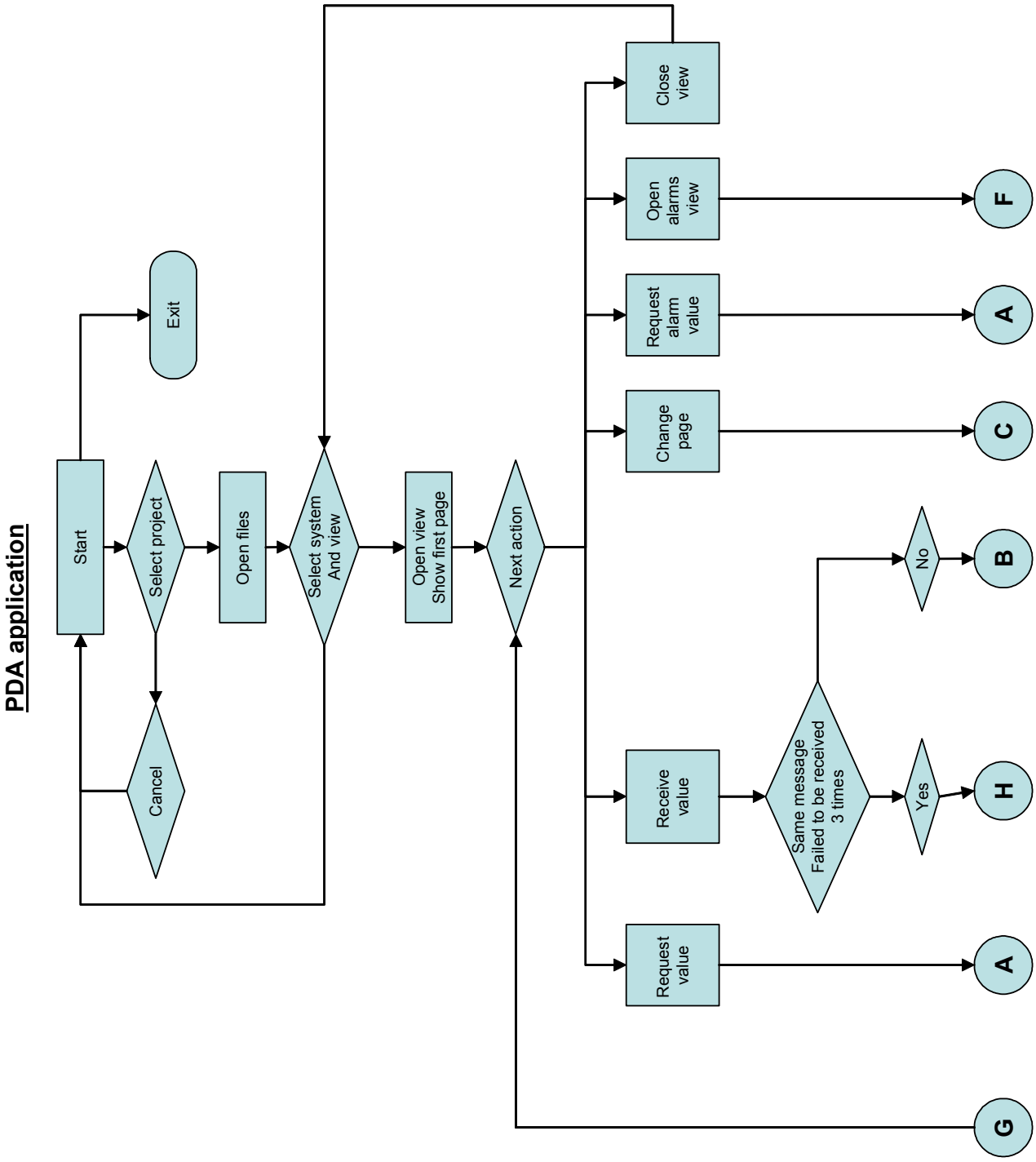
Appendix A: Flowchart Programming tool

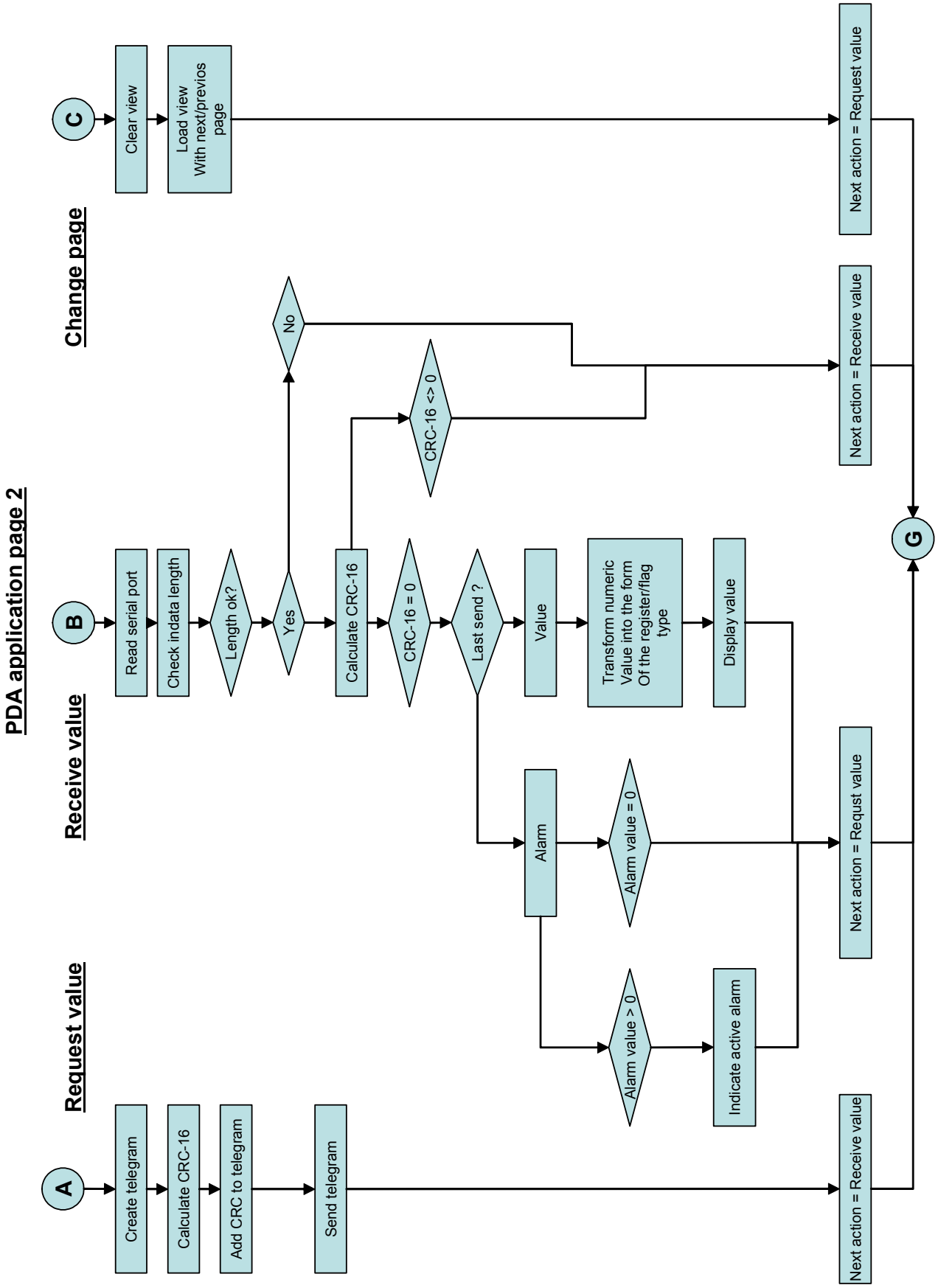


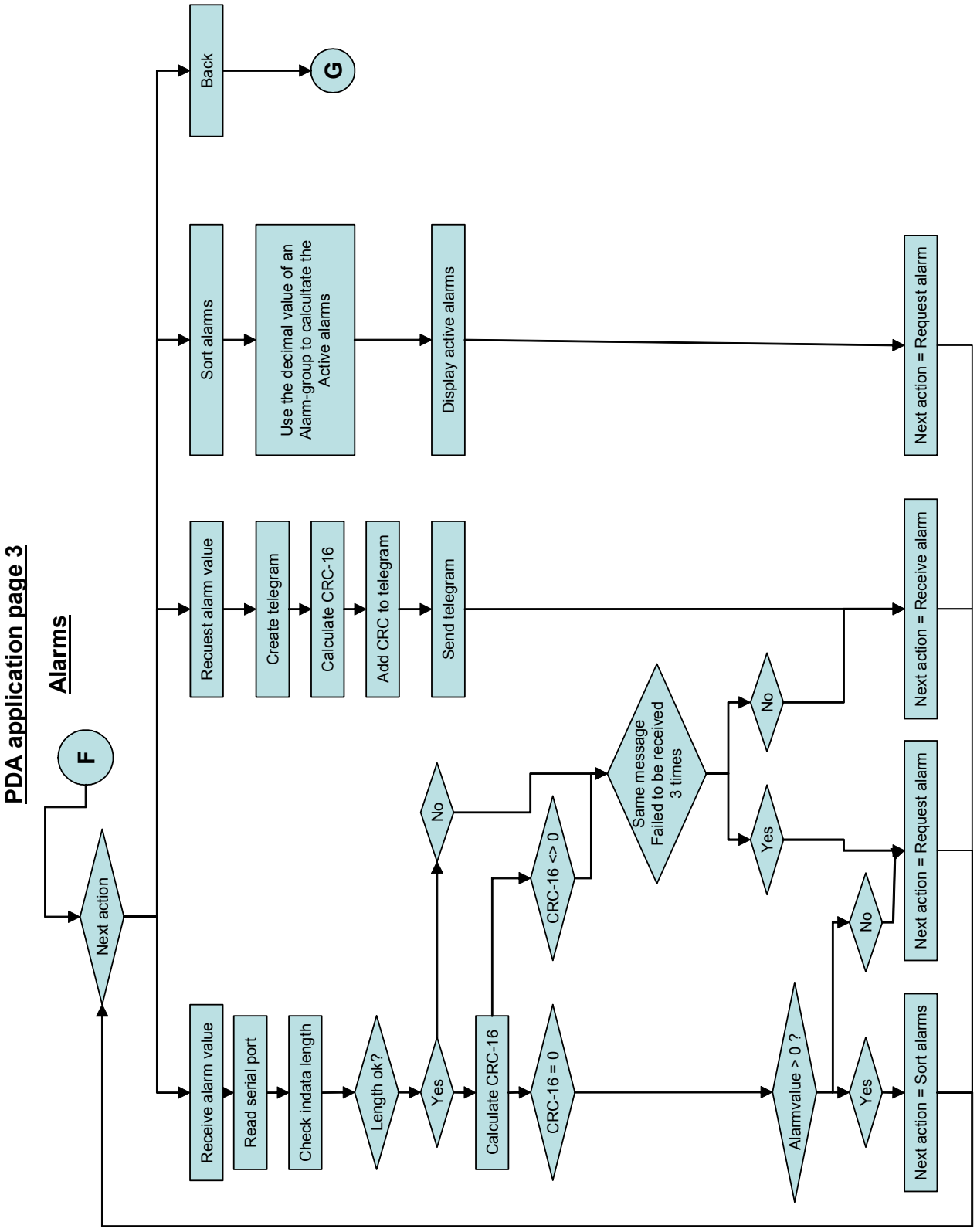

Programming tool Page 2
Edit register type
A
View elements
Action
Add element
Remove element
New Register-type
Switch which Register-type To view
Remove Register-type
Save changes
Close window
Edit flag type
B
View flagtype
Action
New Flag-type
Switch Flagtype To view
Remove Flagtype
Save changes
Close window
E


Programming tool Page 3
Edit alarms
C
View Alarms
Action
Create new Alarm
Show info about Selected alarm
Remove Alarm
Save changes
Close window
Edit view
D
View variables
Action
Add variable to view
Remove variable From view
Show info about Selected variable
Save changes
Back
E

Appendix B: Flowchart PDA application




PDA application page 2
Request value
Receive value
Change page
A
Create telegram
Calculate CRC-16
Add CRC to telegram
Send telegram
Next action = Receive value
B
Read serial port
Check indata length
Length ok?
No
Yes
Calculate CRC-16
CRC-16 <> 0
CRC-16 = 0
Last send ?
Value
Transform numeric Value into the form Of the register/flag type
Display value
Alarm
Alarm value = 0
Alarm value > 0
Indicate active alarm
Next action = Requst value
Next action = Receive value
C
Clear view
Load view With next/previos page
Next action = Request value
G


PDA application page 3
Alarms
F
Next action
Back
G
Sort alarms
Use the decimal value of an Alarm-group to calcultate the Active alarms
Display active alarms
Next action = Request alarm
Recuest alarm value
Create telegram
Calculate CRC-16
Add CRC to telegram
Send telegram
Next action = Receive alarm
Receive alarm value
Read serial port
Check indata length
Length ok?
No
Yes
Calculate CRC-16
CRC-16 = 0
CRC-16 <> 0
Same message Failed to be received 3 times
No
Yes
Next action = Request alarm
Alarmvalue > 0 ?
No
Yes
Next action = Sort alarms

Appendix C: Program code

CRC-16

Calculates CRC-16 on a buffer and returns the CRC-16 value.
bLen = how many elements of the buffer that should be considered in the calculation.

```
    Private Function CRC16A(ByVal buffer() As Byte, ByVal bLen As Integer)
As Long
        Dim i As Long
        Dim Temp As Long
        Dim CRC As Long
        Dim j As Integer
        Dim crcString As String
        Try
            crcString = ""
            For i = 0 To bLen
                Temp = buffer(i) * &H100&
                crcString = crcString & " " & buffer(i)

                CRC = CRC Xor Temp
                For j = 0 To 7
                    If (CRC And &H8000&) Then
                        CRC = ((CRC * 2) Xor &H1021&) And &HFFFF&
                    Else
                        CRC = (CRC * 2) And &HFFFF&
                    End If
                Next j
            Next i
            CRC16A = CRC And &HFFFF

        Catch ex As Exception
            'MsgBox("crc16a: " & ex.Message)
            CRC16A = -1
        End Try

    End Function
```

Calculate decimalform

Takes the value of an register and turnes it into the form described by the database. This function is only used for values presented as decimal values.

```
'*****************************************************************
    'formats the incomming data(register) in the way the user has decided
    '*****************************************************************
    Private Function calcToDecForm(ByVal decimalForm As String, ByVal
nummer As String) As String
        Dim numLen As Integer
        Dim decLen As Integer
        Dim numTxt As String
        Dim decTxt As String
        Dim returTxt As String
        Dim m As Integer
        Dim n As Integer
        Dim keepOn As Boolean

        returTxt = ""
        keepOn = True
        numLen = Len(nummer)
        decLen = Len(decimalForm)
        m = 0
        n = 0
        While (keepOn = True)
            decTxt = Mid(decimalForm, decLen - m, 1)

            If decTxt = "#" Then
                If n >= numLen Then
                Else
                    returTxt = Mid(nummer, numLen - n, 1) & returTxt
                    n = n + 1
                End If
                m = m + 1
            Else
                returTxt = decTxt & returTxt
                m = m + 1
            End If
            If m = decLen Then
                keepOn = False
            End If

        End While
        calcToDecForm = returTxt

    End Function
```

Read a file

Read the file “filen” and saved it in inBuffer. Filen is the filename including the full path to the file.

```
'************************************
    'Reads a file and saves it in inBuffer
    '************************************
    Private Sub readFile(ByVal filen As String)
        Dim sr As FileStream = File.OpenRead(filen)
        Dim str As String
        Dim inBuf() As Byte
        Dim nr As Integer
        Dim fileLen As Integer

        Try
            fileLen = sr.Length
            ReDim inBuf(fileLen)
            nr = sr.Read(inBuf, 0, fileLen)
            myText = ""
            inBufLen = 0
            For i = 0 To nr
                If (inBuf(i) = 10) Then

                ElseIf (inBuf(i) = 13) Then
                    inBuffer(inBufLen) = myText
                    inBufLen = inBufLen + 1
                    myText = ""
                ElseIf (inBuf(i) = 0) Then
                    inBuffer(inBufLen) = myText
                Else
                    myText = myText & Chr(inBuf(i))
                End If
            Next
        Catch ex As Exception
            MsgBox("fel i readFile: " & ex.Message)
        End Try

    End Sub
```

Sort alarms

Displays the active alarms according to the value of the alarm collection.

```
    Private Sub sortAlarms()
        Dim larmVal As Integer
        Dim bitBuffer(25) As String
        Dim antalBitar As Integer
        Dim tmpInt As Integer
        Dim larmAdr As String
        Dim larmQue(100) As String
        Dim queNr As Integer
        Dim k As Integer

        Try
            queNr = 0

            For i = 1 To ListView1.Items.Count
                ListView1.Items.RemoveAt(0)
            Next

            For i = 0 To alarmCount
                tempBuffer = Split(alarmRegBuffer(i), "¤")
                myText = tempBuffer(1)
                larmAdr = tempBuffer(0)
                larmVal = CInt(myText)

                For j = 25 To 0 Step -1
                    If larmVal > 2 ^ j - 1 Then
                        bitBuffer(j) = "1"
                        larmVal = larmVal - 2 ^ j
                        For k = 0 To UBound(alarmBuffer)
                            tempBuffer = Split(alarmBuffer(k), "¤")
                            If tempBuffer(0) = larmAdr Then
                                If tempBuffer(1) = j Then
                                    lItem = New ListViewItem(tempBuffer(2))
                                    ListView1.Items.Add(lItem)

ListView1.Items(queNr).SubItems.Add(tempBuffer(3))

ListView1.Items(queNr).SubItems.Add(tempBuffer(4))
                                    queNr += 1
                                End If
                            End If
                        Next
                    Else
                        bitBuffer(j) = "0"
                    End If
                Next

            Next
            nextAction = 1
            Timer1.Enabled = True
        Catch ex As Exception
        End Try

    End Sub
```

Get next

Sends a request to the PLC to return the value of the next variable.

```
Private Sub getNext()
        Try
            If SerialPort.PortOpen = False Then
                portOpen = openPort()
                If portOpen Then

                Else
                    nextAction = 0
                    GoTo line1
                End If
            End If
            If listNr = listLen Then
                listNr = 0
            Else
                listNr = listNr + 1
            End If
            tempBuffer = Split(varBuffer(listNr + currentElement), "¤")
            varAdress = tempBuffer(0)
            varRegFlg = tempBuffer(3)
            varType = tempBuffer(4)
            varDecForm = tempBuffer(5)

            If varAdress = "" Then ' "Text"
                GoTo line1
            End If
            currentAdress = CInt(varAdress)
            ReDim outBuffer(8)
            outBuffer(0) = 181
            outBuffer(1) = 0
            outBuffer(2) = 1
            If varRegFlg = "Flagga" Then
                outBuffer(3) = 2
                lastSend = 1
            Else
                outBuffer(3) = 6
                lastSend = 0
            End If
            outBuffer(4) = 0 '00
            outBuffer(5) = currentAdress \ 256
            outBuffer(6) = currentAdress Mod 256

            crc16 = CRC16A(outBuffer, 6)

            outBuffer(7) = (crc16 \ 256)
            outBuffer(8) = (crc16 Mod 256)

            dsrCheck = SerialPort.DSR
            If dsrCheck Then
                BTLabel.ForeColor = System.Drawing.Color.LimeGreen
                SerialPort.Output(outBuffer)
            Else
                BTLabel.ForeColor = System.Drawing.Color.Red
                nextAction = 9
            End If
```

```
line1:

        Catch ex As Exception
            MsgBox("getNext: " & ex.Message)
        End Try
    End Sub
```

Receive data

Receive a telegram from the PLC.

```
Private Sub receiveData()
        Dim tmpTxt As String
        Dim sluta As Boolean
        Dim recAntal As Integer
        Dim tmpNr As Integer
        Dim tmpInt As Integer
        Dim negReceive As Boolean
        Dim a, b, c, d As Integer
        sluta = False
        myText = ""

        Try

            Select Case (lastSend)
                Case 0
                    recAntal = 8
                Case 1
                    recAntal = 5
                Case 2
                    recAntal = 8
                    hideChangeValStuff()
                    items = SerialPort.InputArray
                    nextAction = 1
                    GoTo line2
                Case 3
                    recAntal = 8
                Case Else
                    MsgBox("fel i sel case oncomm")
                    GoTo line2
            End Select

            inbufferSize = SerialPort.InBufferCount
            inBuffer = SerialPort.InputArray

            crc16 = CRC16A(inBuffer, inbufferSize - 1)
            If crc16 = 0 Then
                If lastSend = 1 Then
                    currentValue = inBuffer(inbufferSize - 3)
                    For i = 0 To UBound(flgTypeBuffer)
                        tempBuffer = Split(flgTypeBuffer(i), "¤")
                        If tempBuffer(0) = varType Then
                            myText = tempBuffer(currentValue + 1)
                            GoTo line1
                        End If
                    Next
                Else
```

```
                    a = inBuffer(inbufferSize - 6)
                    b = inBuffer(inbufferSize - 5)
                    c = inBuffer(inbufferSize - 4)
                    d = inBuffer(inbufferSize - 3)
                    tmpInt = a Xor 127
                    If tmpInt > 127 Then 'Check if the incoming value is
negative
                        negReceive = True
                        a = a Xor 255
                        b = b Xor 255
                        c = c Xor 255
                        d = d Xor 255
                        currentValue = 65536 * a + 4096 * b + 256 * c + d +
1
                    Else
                        negReceive = False
                        currentValue = 65536 * a + 4096 * b + 256 * c + d
                    End If
                    tmpNr = 0
                    If lastSend = 3 Then
                        tempBuffer = Split(alarmRegBuffer(aktAlarm), "¤")
                        If currentValue <> 0 Then
                            tempBuffer(1) = "1"

                        Else
                            tempBuffer(1) = "0"

                        End If
                        myText = tempBuffer(0) & "¤" & tempBuffer(1)
                        alarmRegBuffer(aktAlarm) = myText
                        For i = 0 To alarmNr
                            tempBuffer = Split(alarmRegBuffer(i), "¤")
                            If tempBuffer(1) = "1" Then
                                tmpNr += 1
                            End If

                        Next
                        If tmpNr > 0 Then
                            alarmBtn.Show()
                            alarmLabel.Text = "Larm"
                        Else
                            alarmBtn.Hide()
                            alarmLabel.Text = ""
                        End If
                        GoTo line2
                    Else
                        For i = 0 To UBound(regTypeBuffer)
                            tempBuffer = Split(regTypeBuffer(i), "¤")
                            If tempBuffer(0) = varType Then
                                If UBound(tempBuffer) = 0 Then 'value
                                    If negReceive Then
                                        If currentValue < 10 Then
                                            myText = "-0" &
calcToDecForm(varDecForm, currentValue)
                                        Else
                                            myText = "-" &
calcToDecForm(varDecForm, currentValue)
                                        End If
```

```
                                    Else
                                        If currentValue < 10 Then
                                            myText = " 0" &
calcToDecForm(varDecForm, currentValue)
                                        Else
                                            myText = " " &
calcToDecForm(varDecForm, currentValue)
                                        End If

                                    End If
                                Else
                                    myText = tempBuffer(currentValue + 1)
                                End If
                                i = UBound(regTypeBuffer) + 1 'GoTo line1
                            End If
                        Next
                    End If

                End If
line1:
                comCheck = 1
                ListView1.Items(listNr).SubItems(1).Text = myText 'curVal
                myText = ""
            Else
                ListView1.Items(listNr).SubItems(1).Text = "Crc"
            End If

line2:
        Catch ex As Exception
            myText = ex.Message
        End Try
    End Sub
```

Check connection

Before sending telegrams to the PLC the connection is checked. This is done by checking if the data set ready line is high. If the data set ready line is high the Bluetooth connection indicator on the screen turnes green. If not the indicator turnes red.

```
            dsrCheck = SerialPort.DSR
             If dsrCheck Then
                 BTLabel.ForeColor = System.Drawing.Color.LimeGreen
                 SerialPort.Output(outBuffer)
             Else
                 BTLabel.ForeColor = System.Drawing.Color.Red
                 nextAction = 9
             End If
```

Read alarm

This sub is called in the viewPage when checking if the alarm collections value are greater than zero.

```
Private Sub readLarmer()
        Try
            If SerialPort.PortOpen = False Then
                portOpen = openPort()
                If portOpen Then

                Else
                    nextAction = 0
                    GoTo line1
                End If
            End If
            If aktAlarm = alarmNr Then
                aktAlarm = 0
            Else
                aktAlarm += 1
            End If
            tempBuffer = Split(alarmRegBuffer(aktAlarm), "¤")
            myText = tempBuffer(0)
            currentAdress = CInt(myText)

            ReDim outBuffer(8)
            outBuffer(0) = 181
            outBuffer(1) = 0
            outBuffer(2) = 1
            outBuffer(3) = 6
            outBuffer(4) = 0 '00
            outBuffer(5) = currentAdress \ 256
            outBuffer(6) = currentAdress Mod 256

            crc16 = CRC16A(outBuffer, 6)

            outBuffer(7) = (crc16 \ 256)
            outBuffer(8) = (crc16 Mod 256)
            lastSend = 3

            dsrCheck = SerialPort.DSR
            If dsrCheck Then
                BTLabel.ForeColor = System.Drawing.Color.LimeGreen
                SerialPort.Output(outBuffer)
            Else
                BTLabel.ForeColor = System.Drawing.Color.Red
                nextAction = 9
            End If

line1:
        Catch ex As Exception
            MsgBox("readlarmer: " & ex.Message)
        End Try

    End Sub
```