MASTER

Design of an intelligent actuator-controller using Profibus-DP

David, C.F.L.

Award date:
1997

Link to publication
Master's Thesis:

Design of an intelligent actuator-controller using Profibus-DP

C.F.L. David

Location: Ellips B.V., Eindhoven, The Netherlands
Coach(es): ir. E.P.M. Bakker en ir. J.P.C.F.H. Smeets
Supervisor: Prof.ir. M.P.J. Stevens
Period: October 1996 - June 1997

The Faculty of Electrical Engineering of Eindhoven University of Technology does not accept any responsibility regarding the contents of Master's Theses.
Summary

At the firm Ellips B.V. in Eindhoven one is developing several vision-systems for a range of industrial applications. One of those systems is a fruit-grading system. A fruit-grading system has the basic function of sorting several sorts of fruit like apples, pears, peaches and kiwis. Fruit can be sorted (or better: graded) on size, weight and flush. In the near future it should be possible to grade on quality also. At the moment of writing this document, a lot of those machines are sold to several firms in Europe. The main problem of the current machines is that they are physically very large, because of a lot of grading-exits and more transporting lines driven parallel. The length of a line can be up to 100 meters long and with more machines to control a total length of more than 200 meters can be achieved. To control all relays, sensors and weight-devices hundreds of meters cable is needed. It’s not just that a lot of cable is needed (cost-reasons), but also the search for errors in a defect system is difficult because of a lack of a well-designed structure in the system. Also the development of new industrial fieldbuses makes it necessary to introduce a new platform for the fruit-grading-systems. At Ellips B.V. one decided to use Profibus as the industrial fieldbus. Profibus stands for PROcess FIeldBUS and is now widely used in the industry. With Profibus, intelligence is distributed over the system and the processor-load of the main system therefore will be reduced. Profibus is based on the seven layer OSI-model and the transmission-protocol is based on the RS-485 standard. This makes the Profibus very suitable for high speed transmissions up to 12 Mbit/s in noisy environments.

The main assignment was to implement an intelligent actuator-controller into a Profibus-DP-system. DP means Decentralized Periphery and is one of the three possible Profibus-protocols. Profibus-DP is designed for very high speed transmission in less complex applications as I/O-controllers like actuator/sensor-controllers. In these applications the number of data-exchange- and diagnostics-bytes per access is less then more complex applications where diagnostics are more important and therefore introducing a bigger overhead and for that reason a lower effective transmission-rate. The actuator-controller is designed around a microcontroller 80C32 based on the well-known family 8051-microcontrollers. In order to match with the high transmission-speed of 12 Mbit/s an asic (Application Specific Integrated Circuit) SPC3 from Siemens is used to perform almost the complete communication-protocol to relief the processor-time of the microcontroller. In order to communicate with the SPC3, communication-software had to be bought from Siemens. This firmware should be implemented on the microcontroller and be expanded with self-defined routines to perform the functions of the actuator-controller. The high-level-flow for the software was given. The functions of the actuator-controller are to control 16 different relays and perform several diagnostics like overvoltage-detection and short-circuit-detection. At last recommendations are give about the sensor/encoder-controller and the weight-machine.
# Table of Contents

## CHAPTER 1

1.1 INTRODUCTION .................................................................................................................. 5
1.2 PROBLEM-DEFINITIONS ..................................................................................................... 7

## CHAPTER 2

2.1 DESCRIPTION OF THE FRUIT-GRADING-SYSTEM IN THE PRESENT STATE .................. 8
   2.1.1 Functional description of the machine ......................................................................... 8
   2.1.2 Relay-control and sensor-control in the current system ............................................. 9
2.2 DESCRIPTION OF THE FRUIT-GRADING-SYSTEM IN THE NEAR FUTURE .................... 10
   2.2.1 Use of the Profibus-DP-protocol ............................................................................. 10
   2.2.2 Functional description of the new fruit-grading-system ......................................... 10
   2.2.3 Relay-control and sensor-control in the new system .............................................. 12

## CHAPTER 3

3.1 INTRODUCTION TO INDUSTRIAL FIELD-BUSES ......................................................... 14
   3.1.1 Brief description of possible busstructures .............................................................. 14
   3.1.2 ASI .......................................................................................................................... 15
   3.1.3 BITBUS ................................................................................................................ 16
   3.1.4 CAN ...................................................................................................................... 16
   3.1.5 Interbus-S .............................................................................................................. 16
   3.1.6 P-NET ................................................................................................................... 17
   3.1.7 WorldFIP .............................................................................................................. 18
   3.1.8 Conclusions of the choice of an appropriate fieldbus ............................................. 19
3.2 SPECIFICATION AND EXPLANATION OF PROFIBUS-DP ........................................... 20
   3.2.1 General description of Profibus-DP ...................................................................... 20
   3.2.2 The frame-structure in Profibus-DP ..................................................................... 24
   3.2.3 Possible frame-formats in Profibus-DP ................................................................. 24
   3.2.4 Parameterization, configuration, diagnostic and data-exchange ............................. 27
3.3 PROFIBUS-DP MASTER ............................................................................................... 38
3.4 PROFIBUS-DP SLAVE .................................................................................................. 40
3.5 TIMING-ASPECTS IN PROFIBUS-DP .......................................................................... 42

## CHAPTER 4

4.1 DESIGN OF THE HARDWARE OF THE ACTUATOR-CONTROLLER ........................... 45
4.2 THE TOTAL PICTURE OF THE ACTUATOR-CONTROLLER-HARDWARE .................. 47
4.3 MODULAR DESCRIPTION OF HARDWARE .................................................................. 48
   4.3.1 Design of the Profibus-interface ............................................................................ 48
   4.3.2 Design of a power-supply ................................................................................... 49
   4.3.3 The "glue"-logic: in- and outputs needed ............................................................... 30
   4.3.4 Design of the memory-module ............................................................................ 31
   4.3.5 Design of the interface to the actuators ............................................................... 33
   4.3.6 Design of the core of the actuator-controller ...................................................... 35
4.4 MEMORY-MODEL OF THE ACTUATOR-CONTROLLER ............................................. 56
4.5 DESIGN AND USE OF SOFTWARE OF ACTUATOR-CONTROLLER ......................... 57
   4.5.1 General software-functions needed by the actuator-controller ........................... 57
   4.5.2 Use of SIEMENS firmware for control of the asic SPC3 .................................... 38
   4.5.3 Design of software for control and diagnostics of actuators ............................... 60
Chapter 1

1.1 Introduction

At the firm Ellips B.V. in Eindhoven one is specialized in developing hardware and software for vision-systems. With one of those systems a fruit-grading-system was build for fruit-sorting-purposes. This machine is capable to sort fruit (like apples, pears and peaches) on diameter, weight and flush. This sorting (or grading) is done by measuring the fruit with camera's and weight-machines. Frame-grabbers are used to load the measured pictures from the camera into a computer. With these pictures, diameter, size and flush are calculated by the computer. Together with the weight-information, fruit can be grading and can be sent to the right exits of the machine. The fruit is transported in cups, which are directly connected to a conveyor. This conveyor is actuated by a motor with an almost fixed speed. Sensors are used to measure the position of the cups and relays are used to set a cup in the right position. Two positions per cup are possible. A normal mode: the cup is positioned at horizontal level and therefore the fruit stays in the cup. The other mode is at diagonal/vertical level with the result that a cup will released from the fruit. The fruit will fall onto the right exit, if controlled well and at the right time. An average machine has at least 3 lines driven parallel with at least 6 exits. With more machines coupled together the total physical length of the total system can be more than 100 meters. With one main control-system and several relays and sensors to control a lot of electrical cable is needed. This introduces expensive costs and errors because of a lack of a structured overview. Therefore at Ellips was decided to introduce a new platform with a fieldbus called Profibus. This fieldbus makes it possible to decentralize the control-periphery where it is needed. A Profibus is a special token-passing twisted pair cable network and is widely used in industrial applications.

In this report the design of a actuator-controller is discussed. First of all a summary of the fruit-grading-system and an analysis of the Profibus-DP-protocol will be given. The design of the hardware and software for the actuator-controller are also discussed separately. Also the choices for the specific components that are needed are given. Finally some test-results are described. In the following chapter the specific assignment for the graduation work will be given.

Below, all needs for the new fruit-grading-system are stated:

Four hardware-boards have to be developed:

1. A PC-master-board
2. A relay-controller (slave)
3. A sensor/encoder-controller (master/slave)
4. A weight-board (master/slave)
Each board will now be specified:

1. A PC-master-board has the main function to communicate with other master-boards and to control some dedicated slave-boards. The board must be 16-bit AT-bus-compatible and I/O-mapped. The master should have two separate galvanic isolated channels for the Profibus. One for communication with other masters and one for data-exchange with slaves. One interrupt is needed to give the PC a message about an event. Also a 256kB RAM has to be on board together with a boot-EPROM. It should be possible to download software from the host. A very fast CPU is needed to perform all master-tasks in time. At last the board has to be EMC-approved.

2. The actuator-controller has to drive 16 relays with a continue current from minimal 0.75 Ampere and a voltage from 40 Volts. The relays should be quickly replaceable and have to be galvanic isolated. Of course the connection with the Profibus has to be galvanic isolated from the rest of the controller. There has to be a possibility to perform diagnostics on the relays like overvoltage-detection, eventually overtemperature-detection, short-circuit and open load-detection. Software running on the board has to be updated in the field, without de-installing the board from the machine. This introduces the need for a flash-memory to download software from the host over the Profibus. A microcontroller has to be used to control the relays and to communicate with the Profibus. Also the board should be EMC-approved.

3. The sensor/encoder-controller has to measure the velocity of the conveyor and has to calculate the position of the fruit-cups. This is done by a rotation-encoder and by a null-sensor. With the encoder it's possible to determine the direction of the conveyor. Of course the board must be EMC-approved and galvanic isolated from the Profibus. Just like the actuator-controller there has to be a microcontroller with flash-memory on board for update-purposes.

4. The weight-board is needed to measure the weight of the fruit as it arrives at the weight-machine. A 16-bit DSP (Digital Signal Processor) is needed to perform real-time calculations. An EEPROM is needed to fetch specific filter-coefficients for the weight-calculation. The board will probably be a busmaster because of the number of calculations. Eight pre-amplified differential channels with a sensitivity of at least 20mV to 5V, with a drift <250 nV/°C, noise <1μV in a range to 4 kHz are needed. With a variable sample-frequency the results are transmitted over the Profibus to the PC's.
1.2 Problem-definitions

In this paragraph the problem-definitions for the graduation-work will be described. The assignment can be divided into sub-parts. The main part is the design of a actuator-controller, which should be able to drive 16 relays. The controller has to be compatible with the Profibus-DP-protocol and several relay-diagnostics have to be made. For a proper design analyses have to be made of the Profibus-DP-protocol and the fruit-grading-machine. These analyses form the sub-parts of the assignment. Below the problem-definitions are stated.

- Main: Design an actuator-controller which can be used with the Profibus-DP-standard

- Sub: Relay-diagnostics must be made via the Profibus and diagnostics about several relay-failures like overload, short circuits and open loads should be possible
- Sub: Software that is installed on the controller should be upgraded in the field via the Profibus, without the need to de-install the controller from the machine
- Sub: Describe the Profibus-DP-protocol, especially the protocol concerning the master-slave communications
- Sub: Describe the fruit-grading-machine concerning the control of the relays and describe shortly the control of the sensors in the current machine and the machine that will be developed in the near future
Chapter 2

2.1 Description of the fruit-grading-system in the present state

To understand the aspects of the design of the actuator-controller, a short description of the fruit-grading-system is needed. However, the actuator-controller will not be limited to one system but is has at least to meet with the needs of the fruit-grading-machine. In the next paragraph the main functions of the machine are discussed in relation to the relays and sensors. From paragraph 2.2 and further, the fieldbus Profibus-DP will be introduced into the fruit-grading-system as it will be in the near future.

2.1.1 Functional description of the machine

The main function of the fruit-grading-system is to sort out fruit with respect to size, weight or flush. In the future grading on quality should also be possible. The kinds of fruit that are sorted in the current machines are apples, pears, peaches and kiwi’s for example. If we look at picture 2.1 below, we can see how the machine looks like in functional terms. At the left, fruit is inserted, cleaned and placed in a row. From here fruit is lighted by light-bulbs to assure a good image-contrast. Then binary camera’s are used to measure the diameter of the fruit. Also full color camera’s are used to measure the flush of the fruit. In the future these camera’s can also be used to measure the quality of fruit. Several lines can be driven parallel at one time. These lines are indicated from 1 to M in the picture. The exits are indicated from 1 to N. The exits are the lines where the fruits finally fall into, after they are graded. At the beginning of each transport-line, a weight-machine will measure the weight of the fruit. However at this time, the fruit should be weighted faster and more accurate. The sensor S0 indicates the position of the first cup on the conveyor. Sensors S1 and S2 measure the displacement of the cups. Each cup can carry one piece fruit. Since the sensor S0 only registers the position of the first cup on the conveyor with sensors S1 and S2 the position of all other cups can be calculated. At every exit and line a relay is connected to drive the mechanism of the fruit-cup. The timing of switching the relays should be accurate, since the conveyor is driven at a fairly constant speed. If a relay is switched too late or too early severe damage of the machine can result. Another result can be a wrong sorting-exit for the fruit which makes the whole sorting-procedure senseless.
2.1.2 Relay-control and sensor-control in the current system

In the current fruit-grading-machine, the relays and sensors are all controlled by the main system. This means that the more relays and sensors (weight-machines included) needed to be controlled, the bigger the overhead for the main system will be. All relays, sensors, camera’s and weight-machines have a separate connection through I/O-cards with the main system. This introduces a lot of cables with lengths up to 100 meters. If malfunctions of the system occur, every line should be checked, since there are only very simple diagnostics-facilities (i.e.: LED’s are used to indicate the status of a relay, ON or OFF). However the reason for malfunction can’t be determined in the current system. In the new system, malfunctions will be classified and reported with diagnostics-messages.
2.2 Description of the fruit-grading-system in the near future

In the near future (i.e.: autumn 1997 - spring 1998) a new platform will be introduced into the fruit-grading-system. This platform will be based on a new protocol: the Profibus-DP-protocol. In the next chapter Profibus will be discussed in detail and it will be compared with other busstructures. In these last paragraphs of this chapter it will only shortly be mentioned with respect to the fruit-grading-system.

2.2.1 Use of the Profibus-DP-protocol

As told in the introduction, a new protocol will be used in the next version of the fruit-grading-system. At the firm Ellips B.V. one decided to choose the Profibus-DP-protocol. Some reasons to choose this protocols, were the possibility of high-speed transmissions and the world-wide use of the protocol in industrial applications. The Profibus-protocol is also well designed for input- and outputcontrol. The Profibus is a token-passing network with at least one master and one slave. Only masters can exchange a token with each other. If a master owns a token, it can poll one or more dedicated slaves. The time a master can hold a token is limited and therefore the time of polling is also limited. However the use of asics (Application Specific Integrated Circuits) in the Profibus-protocol will assure the high transmission-speed. With the new protocol, relays and sensors are assumed as slaves and the main control-system is the master. Other masters in the system are for instance the weight-system and the real-time operating systems. These systems need to take control over the Profibus at any time (if they own the token), since they are dealing with time-dependent data.

2.2.2 Functional description of the new fruit-grading-system

Since DP stands for Decentralized Periphery in the Profibus-DP-protocol, local periphery will take care of the relays and the sensors. This will unload the main system and reducing the data-exchange to input- or outputdata of the relays and sensors. Diagnostics are done in the local periphery and only short diagnostics-bytes will be exchanged with the main system. In figure 2.2 we can see the total functional description of the fruit-grading-system with the Profibus-DP-protocol. Two separate Profibus-DP-buses are used and they are defined as bus A and bus B. Bus A is a "low-rate"-polling-bus, meaning that every device at bus A is only
Design of an intelligent actuator-controller using Profibus-DP

polled once per fruitcup, i.e.: 10-20 times per second. On the other side, bus B is a “high-rate”-polling-bus, where every device is polled at a rate of 100-1000 times per second.

One master-PC controls both separate Profibuses and exchanges data and diagnostics from- and to every connected Profibus-device. The same master-PC is connected along a TCP/IP-protocol based bus with Windows-based PC’s. With these PC’s, the user can control the total fruit-grading-system. A modem-line is provided to exchange data and diagnostics with the rest of the world for service- and maintenance purposes. One of these services is the downloading of update-software for the several dedicated Profibus-DP-devices through this modem-connection.

Fig. 2.2: The total fruit-grading-system with the Profibus-DP-protocol implemented
On bus A camera's with the appropriate hardware are connected to the Profibus. Real-time systems calculate diameter, flush (color0) and quality (color1) of the fruit. The weight of the fruit is calculated with a dsp-controlled-board (dsp = digital signal processor). Every calculation is performed once per fruitcup. The result of a calculation is sent over the Profibus to the master-PC. The next section discusses about the relay- and sensor-controllers on bus B.

2.2.3 Relay-control and sensor-control in the new system

In figure 2.3 the configuration from the actuator- and sensor/encoder-controller is stated. Two other I/O-controllers are also stated to show how additional hardware can be added to the system. These other I/O-controllers can also be actuator-controllers. The maximum number of slave-devices in a one-master Profibus-DP-system is 126. This number is the absolute maximum, but in the fruit-grading-system this limit will not be reached at all. In fact the maximum number of slave-devices in the fruit-grading-system will be about 5 to 6 on bus B. First of all the functions of the actuator-controller will be described.

One actuator-controller should be able to drive 16 different relays at one time and it should be able to diagnose the 16 relays on three kinds of errors.

These possible errors are:

1. Shorted load, Overload
2. Shorted load to ground
3. Open load

These errors can be reported to the main system with diagnostics-messages. Only data-bytes are exchanged between the actuator-controller and the main system.
These data-bytes are the data (ON or OFF) for the 16 relays or the data with diagnostic information about the relays. The switching and diagnosing of the relays is completely handled by the actuator-controllers and therefore relieves the main-system. The same is valid for the sensor-controller. The sensor-controller handles the “zero”-cup-sensor S0 together with an rotation-encoder. This rotation-encoder replaces the position-sensors S1 and S2 from the previous paragraph 2.1.1 and measures the rotational displacement of the conveyor-axis. The encoder emits a certain amount of pulses per displacement. Together with the information of sensor S0 the exact position of a fruitcup can be calculated, since the fruitcups are mechanically attached to the conveyor. By counting the pulses (for example 10 pulses per cup), the exact time for switching the relays is known. The encoder also indicates the direction of the rotational displacement. This makes it possible to correct the calculation of the position of the cups in case of ceasing the conveyor at certain moments. The sensor/encoder-controller also should have multiple inputs for additional features and/or sensors.

The slave-devices will be able to receive new program-code from the master-PC. This code is downloaded into flash-memory which is located on the decentralized devices. This makes updating the software of the devices very easy, with the advantage of downloading new code into operational devices from every possible place. Also the ease of adding and installing new modules (i.e.: other I/O-devices for example) is a big advantage, because with Profibus-DP a new device is directly seen by the master. This master interrogates the new device and will configure and send parameters to the new device. After that (if everything went well) the new device will go in data-exchange mode. In the next chapter the protocol of Profibus-DP will be discussed more extensive in detail concerning a one-master, many slaves configuration.
Chapter 3

3.1 Introduction to industrial field-buses

Nowadays about 200 fieldbuses exist all over the world. Most of them are vendor-specific or limited to a small area of applications. In this chapter a few fieldbuses are discussed shortly, while the fieldbus Profibus will be described in detail. Profibus is now accepted in Europe by the European Norm EN 50170 and is now the fastest industrial fieldbus with a transmission-rate of 12 Mbit/s. One more advantage of this fieldbus, is the independence of vendors which makes this bus one of the most favorites. The next paragraphs will discuss some of the other fieldbuses, followed by a detailed description of the Profibus-DP-protocol.

3.1.1 Brief description of possible busstructures

In order to implement an appropriate busstructure for the fruit-grading-system several existent busstructures must be analyzed and compared with each other. Several criteria to choose the right one are:

- Transmission-rate
- Cost
- Easiness of installation
- Range of applications
- Maximum length of line
- Availability of additional products on the market
- Demand and use of the busstructure by other vendors or users
- Acceptances by other European countries (European Norm)
- Features and expandability for future-applications

At the beginning of my work for graduation, the choice was already made by Ellips B.V. The protocol of Profibus-DP was chosen. Profibus stands for PROcess FIeldBUS. The suffix DP means Decentralized Periphery. However for completeness, several bus-protocols will be compared with each other and shortly be discussed. Other used fieldbuses in industrial applications are:
• ASI
• BITBUS
• CAN
• DeviceNet
• DIN-Measurement Bus
• FOUNDATION Fieldbus
• HART
• INTERBUS-S
• P-NET
• Profibus (-FMS, -DP and -PA)
• SDS Smart Distributed System
• WorldFIP
• Filbus
• Arcnet
• IEC/ISA sp50
• Seriplex
• Lonworks

Only the most important and most accepted fieldbuses will be discussed now. The list stated on the previous page is given for completeness.

3.1.2 ASI

To begin at the top of the previous list, ASI is an Actuator Sensor Interface and is more an addition to already implemented busstructures and thus no replacement. This interface is especially designed for simple I/O-devices like actuator-controllers and sensor-controllers and reduces the use of cable significantly. However, the Profibus-DP-protocol is already well-suited for distributed simple I/O-devices and also reduces the use of cable. However if more than 32 actuators or relays are connected from one place, it could be an option to reduce the costs of cable. Furthermore, the use of more protocols in one system while both protocols do almost the same is not advised in system-design if it’s not necessary. Also in the new fruit-grading-system I/O-devices are already distributed in the field reducing the cost of cable fairly enough.
3.1.3 BITBUS

BITBUS is a fieldbus that exist already for almost 14 years. BITBUS was invented by Intel. The BITBUS is a serial communication system for industrial use, usually called fieldbus. BITBUS is based on a RS-485 party-line (several communication stations on the same twisted pair of wires) and is optimized for the transmission of short real-time messages. In modern installations more and more fiber optic links are used for BITBUS too. A BITBUS communication network always has a master and one or more slaves. Each slave has its own network address which makes it uniquely identifiable in the network. The master operates the network by polling the slaves. The slaves may only respond when polled by the master.

3.1.4 CAN

The principle of CAN (Controller Area Network) is as follows: data-messages transmitted from any node on a CAN bus do not contain addresses of either the transmitting node, or of any intended receiving node. Instead, the content of the message (e.g. Revolutions Per Minute, Hopper Full, X-ray Dosage, etc.) is labeled by an identifier that is unique throughout the network. All other nodes on the network receive the message and each performs an acceptance test on the identifier to determine if the message, and thus its content, is relevant to that particular node. If the message is relevant, it will be processed, otherwise it is ignored. This mode of operation is known as multi-cast. The communication is based on CSMA/CD (Carrier Sense Multiple Access with Collision Detect).

3.1.5 Interbus-S

In 1987 Phoenix Contact invented the fieldbus Interbus-S. This fieldbus was introduced as a simple Actuator/Sensor-Bus. The protocol is described in the international IEC-norm DIN 19258. The protocol for the INTERBUS-S (abbreviated to IBS) system is a hardware dependent structure that has been developed around shifting data. I/O modules are connected together in a manner similar to a series of shift registers. Figure 3.1 shows the serial data flow in the IBS network. The major components of the network are the two protocol chips. The INTERBUS-S Protocol Master (IPMS) chip operates the network from the IBS controller board. The Serial Microprocessor Interface (SuPI) chip links I/O devices to the network.
During each scan cycle, process data words are constantly shifting through the network. Process data containing output information enters the SuPI chip where it is fed to the proper output actuator device. The same process data containing input information leaving the SuPI chip is fed to the IBS controller board. Note that the process data words contain both input and output information. An additional feature of the protocol is that all input and output data is updated at the same time.

The IBS controller board eliminates address settings by performing an identification (ID) cycle to initialize the network. This ID cycle tells the IBS controller the type and position of I/O modules on the bus. The ID cycle is initiated by the host controller prior to the bus going active.

3.1.6 P-NET

The Danish P-NET Fieldbus is designed to connect distributed process components like process computers, intelligent sensors, actuators, I/O modules, field and central controllers, PLC's etc., via a common two wire cable, as shown in fig. 3.2. This replaces traditional wiring, where a great many cables are involved. Process data (e.g. measurement values, valve signals) are transmitted digitally. P-NET is also used for data collection, for configuration of nodes/sensors, and for down-loading of programs.

Apart from the usual measurement values and status data, the bus provides a bi-directional exchange of additional information concerning limit values, actuator positioning and feedback signals, fault signals and internal system data. P-NET can be used to download parameters and programs to modules, which then control the process. The use of intelligent P-
NET sensors and actuators also offers much better diagnostic features than with traditional wiring.

Fig. 3.2: The serial transmission-protocol of P-NET

The specification of P-NET is based on the RS485 standard using a shielded twisted pair cable. This allows a cable length of up to 1200 m without repeaters. Data is sent as an asynchronous transmission in NRZ code. P-NET interfaces are galvanically isolated, and up to 125 devices per bus segment can be connected, due to a special clamp circuit, and again without the use of repeaters. P-NET is a very efficient Fieldbus protocol, in that it can handle up to 300 confirmed data transactions per second, from 300 independent addresses. Data can be transferred in the form of fully processed values (floating point), such as temperature, pressure, current, voltage etc., or as blocks of 32 independent binary signals, indicating valve states, switch positions etc. This results in a performance of up to 9,600 binary signals per second being accessed from anywhere within the complete system. This high rate of fully acknowledged data transmissions can be achieved, because P-NET slaves handle the processing of data and the reception or transmission of frames, in parallel. The processing of a request by the slave is initiated as soon as the first data bytes arrive. This is in contrast to dedicated chip solutions, where the entire frame arrives before processing begins. In this way, the standard P-NET data rate of 76,800 bit/s, is not a limiting factor in performance.

3.1.7 WorldFIP

WorldFIP (World Factory Instrumentation Protocol) was developed by French and Italian firms and was introduced in 1988. In France it became a national fieldbus norm described in IEC 1158-2.

FIP provides a deterministic and reliable scheme for communicating process variables (generated by sensors and executed by actuators) and messages (events, configuration commands, ...) at up to 2.5 Mbit per second on inexpensive twisted pairs cables. FIP uses an original mechanism where the bus arbitrator broadcasts a variable identifier to all nodes on
the network, triggering the node producing that variable to place its value on the network. Once on the network, all modules who need that information "consume" it simultaneously. This concept results in a decentralized data base of variables in the nodes and remarkable real-time characteristics. This feature eliminates the notion of node address and makes it possible to design truly distributed process control systems.

3.1.8 Conclusions of the choice of an appropriate fieldbus

In the European Fieldbus norm EN 50170 Profibus, P-NET and WorldFIP are the only three standards that are accepted as an international fieldbus-standard in the near future. According to European Standardization rules, EN 50 170 will be accepted in all European countries, even those who voted negative. The existing national standards (DIN 19 245 - PROFIBUS, UTE 4660 - World FIP and DSF 21906-P-NET) will be withdrawn by June 30 1997, to be replaced by EN 50170.

The ratification of EN 50170 will cause significant change in the overall fieldbus market. To this date a plethora of mainly vendor-specific Fieldbus solutions has been available. EN 50 170 reduces this bewildering choice to just three. The EN 50 170 standard will thus protect its users' investments and provide an enhanced position for vendors when penetrating global markets. From now on, throughout Europe, all public procurements requiring a fieldbus will be based on EN 50 170. All other standards will be omitted from that moment and are to be implemented at own risk. For that reason, it should be considered very seriously to choose one of those three standards to avoid the risk that a developed product will not be accepted or supported in the near future. In this case Ellips B.V. has made the right choice of choosing the German protocol Profibus-DP.

Also, Profibus is an open standard which allows manufacturers to build applications with a maximum of freedom in designing.
3.2 Specification and explanation of Profibus-DP

As already mentioned, Ellips made her choice for the German fieldbus Profibus-DP. Although easy readable literature of Profibus-DP is just coming up, some efforts were made to give a sufficient understandable description of this protocol. In the next paragraphs of this chapter we will try to explain the main parts and purposes of the protocol mostly taken from the Standard DIN 19245 part 1 and part 3.

3.2.1 General description of Profibus-DP

To begin with: Profibus means PROcess FieldBUS. Three types of Profibus are possible: Profibus-DP, Profibus-FMS and Profibus-PA. Profibus-FMS is the most complex protocol and mainly used for interconnections between programmers such as computers. Profibus-FMS is more complex than Profibus-DP since it can set up connections for the time needed and it works with objects. Profibus-DP works connectionless and straight on, therefore there's overhead of setting up connections. Profibus-PA is a protocol especially designed for industrial applications in hazardous neighborhoods. In the PA-protocol very low voltages are used to prevent electrical discharges causing possible explosions. Profibus-DP is a redesign of Profibus-FMS where several functions are omitted or rewritten to gain more speed. Profibus-DP is based on the OSI-7-layer-model (OSI: Open Systems Interconnection) from ISO (International Organization for Standardization). In figure 3.3 these two layer models are compared with each other. It is shown in this figure that in Profibus-DP layer 3 to 7 are omitted in comparison with the OSI-model. This is done to maintain a high transmission-rate. In fact layer 7 is not fully omitted because the application of the user has to be in layer 7 conform the OSI-standard. A part of layer 7 is prescribed by the protocol as the application-interface. It is better to say that the layer 7 is partly the DDLM, the Direct Data Link Mapper.
As in figure 3.3 stated, the layer above layer 7 is called DDLM. DDLM means Direct-Data-Link-Mapper and makes the connection between the User-interface and layer 2. From the user-interface services are delivered to and from the user. The FDL is the Fieldbus Data Link. This is the name in Profibus for the layer 2 as in the OSI-model. PHY is the physical layer 1.

The physical layer is based on the RS-485-standard and an unshielded- or shielded twisted pair cable is needed. For high transmission rates (1.5 Mbit/s and more) shielded twisted pair cables are needed to prevent interferences from outside. Up to 126 stations (active and passive) can be connected on one bus. However repeaters have to be used (repeaters are counted as passive stations) because RS-485-drivers allow a maximum of 32 participants on one line-segment. In the Profibus-DP/FMS-protocol data is exchanged in NRZ-code (NON RETURN to ZERO). During the duration of a bit the signal-form stays constant. The startbit is always a logical ‘0’ and the stopbit is always a logical ‘1’. The parity-bit is set or reset on even parity. (See also paragraph 3.2.2)
A typical Profibus-DP-system consist of at least one master and one or more slaves. A token-passing-protocol is used to allow a master to enter the bus (see figure 3.4). Only masters are able to hold a token. For this reason, masters are called active stations and slaves are called passive stations. The active stations form a virtual token-ring.

![Virtual token-network](image)

**Fig. 3.4 : An example of a realistic multimaster Profibus-based-system**

If an active station (a master) holds the token it will be able to poll it’s dedicated slaves. In power-up-mode the master parameterizes and configures the slaves that exist in the list of the specific master. Only the slaves that are parameterized and configured by a specific master can receive output-data from that master. However diagnostics and input-data from slaves can be checked by every active station. If all parameterization and configurations are done, the master can go in operate-mode and exchange data with it’s slaves. As already said, a master can only poll it’s slaves when it holds the token. If the token-hold-time expires the master shall pass the token to the next active station in the virtual token-ring. More aspects that are master-specific or slave-specific are explained in paragraph 3.3 and 3.4.

Two types of communication are possible in Profibus. These types are:

1. one to one communication
2. one to many communication (Multicast)

The first one is the most used. Both types of communication are connectionless. The second one is to submit a message to a range of dedicated slaves (this is called broadcast or multicast). In the initialization-phase, the master gives a groupnumber to a set of slaves which
belong together, for example a group of actuator-controllers. Such a message can be a global command like sync or freeze to assure that slaves are synchronized with each other. In the next paragraphs the frame-structures and frame-formats are explained. Also the protocol of Profibus-DP will be discussed considering the role of masters and slaves in one system. All standard details about parameterization, configuration and diagnostics are explained. Also some extensions that are user-specific are described. At last some considerations about the timing-aspects are examined. First of all the pre-defined SAPs in Profibus-DP are explained. (SAP = Service Access Point). With these SAPs a station knows what service is required. All communication is done via these SAPs. The SAPs are transmitted with the request/respond-telegrams in the DSAP (destination SAP) and SSAP (source SAP, where the service was initiated).

The reserved SAPs for Profibus-DP are stated below:

Default SAP: Data-exchange (Write_Read_Data)
SAP54: Master-Master SAP (Master-Master-communication)
SAP55: Change station address (Set_Slave_Add)
SAP56: Read inputs (Rd_Inp)
SAP57: Read outputs (Rd_Outp)
SAP58: Control commands to a DP-slave (Global_Control)
SAP59: Read configuration (GetCfg)
SAP60: Read diagnostic information (Slave_Diagnosis)
SAP61: Transmit parameters (Set_Prm)
SAP62: Check configuration (Chk_Cfg)

All these functions are also stated and explained in paragraph 3.2.4. These functions are requested by the master (class 1 or class 2) in specified telegram-formats. In the next paragraphs this will be explained. Only a few telegram-formats do exist: telegrams without data-fields and telegrams with data-fields with fixed or variable data-field-lengths. In every telegram an start-delimiter (always the first octet) indicates what kind of telegram is sent and an octet containing a function-code indicates what exactly is requested/indicated/acknowledged/confirmed in a particular function-call (SAP number also indicated in a telegram, except in data-exchange-mode).
3.2.2 The frame-structure in Profibus-DP

In the Profibus-protocol messages are always sent in frames. Several formats in frames are possible dependent of the message. In the next paragraph they will be described, but first will be explained how a frame is build up.

A frame consist of a number of octets. One octet contains together with 8 bits information one start-bit, a stop-bit and an even parity-bit. This implies a character of 11 bits. This character is called an UART-Character (Universal Asynchronous Receiver/Transmitter - Character) and is always of the same length of 11 bits. The start-bit should be always a logical ‘0’, the stop-bit always a logical ‘1’. The 8 bits following after the start-bit can be either a logical ‘0’ or a logical ‘1’ and are called the information-bits or octet. The even parity-bit is calculated from the 8 information-bits.

Bit-synchronizing starts always at the falling edge of the start-bit. This is from a binary ‘1’ to a binary ‘0’. If no binary ‘0’ is encountered in the middle of the bit-time, an error is reported and synchronization fails. If a failure occurs, the next leading edge of the start-bit is waited for. Line-idle states correspond with logical “1”-levels. Every action-frame should be preceded by 33 line idle bits (for synchronization). No idle states are permitted between the octets in a frame. Every frame in the Profibus-protocol consist of these elementary UART-Characters and thus determines the minimum required time to send one byte of information. The time to send one bit depends on the chosen transmission-rate but is always called 1 Tbit (bit-time). To ease calculations in timing, one calculates with bit-times which are independent of the transmission-rate. At a transmission-rate of 12 Mbit/s this implicates that the time to send one byte of information is: 11*\( \frac{1}{(12 \cdot 10^6)} \) = 917 ns = 11 Tbit. \( \Rightarrow \) 1 Tbit = 83 ns at a transmissionrate of 12 Mbit/s.

3.2.3 Possible frame-formats in Profibus-DP

In this paragraph the five possible frame-formats in the Profibus-protocol are described. This paragraph only describes the format-structure and doesn’t tell anything about the sequences from frames (request \( \Rightarrow \) acknowledge/response). Two transmission-services exist in Profibus-DP: SRD and SDN. SRD means Send and Receive Data with acknowledge. If a slave is just an output-device it will response with a short acknowledge-telegram “E5H’. The other service SDN means Send Data without acknowledge. This service can be used to send global-control commands like freeze and sync to one or groups of slaves. Several frame-formats do exist: with or without data-field and fixed length or variable length. In figure 3.4 the possible formats are shown.
Fig. 3.5: The different possible frame-formats in the Profibus-protocol

The abbreviations used in figure 3.4 are explained now.

Every frame also called telegram begins with a start-delimiter called SD. Four different start-delimiters are possible:

- **SD1** (hexcode: 10H) is the start-delimiter to indicate that a telegram follows without data-field and a thus a fixed length. This start-delimiter is used in the Profibus-DP-protocol by active stations to look for new active stations (Request_FDL_Status).
- **SD2** (hexcode: 68H) is the start-delimiter to indicate that a telegram follows with data-field and with variable length.
- **SD3** (hexcode: A2H) is the start-delimiter to indicate that a telegram follows with data-field but with fixed length (i.e.: 11 octets). This SD is not used in Profibus-DP.
- **SD4** (hexcode: DCH) is the start-delimiter to indicate that a token-frame follows.

SC is the shortest telegram possible: it consist of one octet (hexcode: E5H) and is meant to give a short acknowledge.
DA is the Destination Address. This is the address of the station that gets a request/response. If the MSB (most significant bit) of this octet is a logical “1” then the address DSAP (Destination SAP) will follow immediately after the FC-octet (in fact the first octet in the DU-field, but not defined as is).

SA is the Source Address. This is the address of the station that requests/responses. If the MSB of this octet is a logical “1” then the address SSAP (Source SAP) will follow immediately after the FC-octet (this is in fact the second octet in the DU-field, but not defined as is).

In the Profibus-DP-protocol the MSB’s of the SA and DA will always be the same (i.e.: DSAP and SSAP occur together or not at all).

LE is the length of the data-field. The data-field is the data included with DA, SA and FC.

LEr is the length of the data-field one more time repeated. This is done for data-security and because of the international accepted Hamming Distance of HD=4.

FC is the Function Code. This is to indicate what kind of telegram the action-frame is (send/request or acknowledge/response). Also used as function and control information to prevent multiplication or loss of messages.

DU is the Data-Unit. This field contains several sorts of data, depending on the sort of requested service. For example diagnostic data about the device or channel-related data of the device. This field can also contain data for exchange-purposes (in- and output-data).

FCS is the Frame Check Sequence. This octet is the result of an arithmetic sum-calculation of DA, SA, FC and DU. This is a way to control the consistency of the submitted data.

Finally ED is the End-delimiter (hexcode always 16H). This octet closes the telegram and thus indicates the end of the frame.
3.2.4 Parameterization, configuration, diagnostic and data-exchange

In the picture showed on this page, we can see all possible interactions between masters and slaves and between masters of class 1 and masters of class 2.

![Diagram showing interactions between masters and slaves]

Fig. 3.6: The interactions between masters (class 1 and 2) and slaves

In this chapter especially the interactions between masters and slaves are discussed. The most important functions are explained here and how they are defined in the Profibus-DP-protocol. The functions that are requested via SAPs in the datalink-layer and that are most used are listed here:

- Change of the station-address (SAP55)
- Diagnostics (SAP60)
- Parameterization (SAP61)
- Configuration (SAP59 and SAP62)
- Data-exchange (SAP nil, and for images SAP56 and SAP57)
The change of an station address can be done by a master of class 2 in power-on mode. Of course this is only possible if a device has the possibility to change the address (i.e.: availability of EEPROM).

<table>
<thead>
<tr>
<th>SD</th>
<th>LE</th>
<th>LED</th>
<th>SD</th>
<th>DA</th>
<th>SA</th>
<th>FC</th>
<th>DSAP</th>
<th>SSAP</th>
<th>DU</th>
<th>FCS</th>
<th>ED</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD2 09H</td>
<td>09H</td>
<td>SD2 8xH</td>
<td>8xH</td>
<td>x</td>
<td>37H</td>
<td>3EH</td>
<td>Octet 1..4</td>
<td>x</td>
<td>16H</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This telegram is send to the station with the address indicated in the DA-octet.

The meanings of octet 1..4 are as follows:

- octet 1: The new address-number
- octet 2: Identification number, high byte (the Ident_Number is a unique device-dependent number provided by the PNO)
- octet 3: Ident_Number, low byte
- octet 4: OOR: Changes are always allowed
  - 01H: No changes allowed again after initial reset

If the length given is bigger than 9 then some user-defined data can follow after the fourth octet. The contents of this user-defined data could be anything and the user is free in his application to define this data to something useful.

Diagnostics of a station is done at the initialization-phase or after a request in data-exchange-mode. Diagnostics are always performed before parameterization and data-exchange-mode to prevent malfunction. If an error occurs for some reason (short circuit on a channel for instance), a high-priority message is sent in the response-telegram. After the master has received the telegram it will go in diagnosis-mode the next time it polls the station. The diagnostic information consist of 6 octets of standard information and any user-related information if preferred. The description of the diagnostic information is stated below:

This telegram is sent in the request from the master to the station specified in the DA-octet:

<table>
<thead>
<tr>
<th>SD</th>
<th>LE</th>
<th>LED</th>
<th>SD</th>
<th>DA</th>
<th>SA</th>
<th>FC</th>
<th>DSAP</th>
<th>SSAP</th>
<th>FCS</th>
<th>ED</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD2 x</td>
<td>x</td>
<td>SD2 8xH</td>
<td>8xH</td>
<td>x</td>
<td>3CH</td>
<td>3EH</td>
<td>x</td>
<td>16H</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The addressed station will respond with the following telegram:

<table>
<thead>
<tr>
<th>SD</th>
<th>LE</th>
<th>LED</th>
<th>SD</th>
<th>DA</th>
<th>SA</th>
<th>FC</th>
<th>DSAP</th>
<th>SSAP</th>
<th>DU</th>
<th>FCS</th>
<th>ED</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD2 x</td>
<td>x</td>
<td>SD2 8xH</td>
<td>8xH</td>
<td>x</td>
<td>3CH</td>
<td>3EH</td>
<td>Octet 1..6 (+ user-data)</td>
<td>x</td>
<td>16H</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Some octets contain information about more aspects, they use one or more bits in the octets.

```
  7  6  5  4  3  2  1  0
```

This picture shows how the bits are organized in an octet: the MSB is bit 7 and the LSB is bit 0.

We'll now explain the response-diagnostic-data:

Octet 1:
- bit 7: Master_Lock, is set by the master and indicates that the slave is parameterized by another master
- bit 6: Prm_Fault, Wrong parameterization (ident_number etc.)
- bit 5: Invalid_slave_response, this bit is fixed and set to 0
- bit 4: Not_supported, the requested service is not supported by the slave
- bit 3: Ext_Diag, the slave has external diagnosis if this bit is set, if not set (i.e.: "0") then the user-related diagnostic area contains status-information and this has low priority
- bit 2: Cfg_Fault, the configuration data do not match
- bit 1: This bit indicates whether a station is ready for data-exchange or not
- bit 0: This bit indicates whether a station exist or not (this is set by the master)

Octet 2:
- bit 7: Station deactivated (set by the master)
- bit 6: Reserved
- bit 5: Sync_Mode, Sync command is received
- bit 4: Freeze_Mode, Freeze command is received
- bit 3: WD_ON, the watchdog is active
- bit 2: Fixed and set to 1
- bit 1: Stat_Diag, static diagnosis, if set then the master will request diagnostic data until reset by slave
- bit 0: Prm_Req, the slave has to be parameterized (set by application)

Octet 3:
- bit 7: External overflow, if more external diagnostic data is available than the send- or receive-buffer can hold
- bit 6..0: Reserved

Octet 4: Master_add, holds the address of the master that did the parameterization (will be FFH if no parameterization was done at all)

Octet 5: Ident_Number, high byte
Design of an intelligent actuator-controller using Profibus-DP

Octet 6: Ident_Number, low byte

Octet 7: Length of external diagnosis

Octet 8 and further: External diagnosis

Three sorts of diagnostic information exist:

1. Device-related diagnostics (for instance: power-failure caused by a line-break)
2. Identifier-related diagnostics (for instance: short-circuit at actuator 4)
3. Channel-related diagnostics (errors in modular systems)

Octet 7 consist always of a header-byte. The sort of diagnostics as mentioned above, are indicated by bit 6 and bit 7 in this header-byte:

Device-related diagnostics:

<table>
<thead>
<tr>
<th>Header Byte</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5..0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>Block length incl. header byte</td>
</tr>
</tbody>
</table>

In the diagnostic field bytes can be defined as needed by the user.

Identifier-related diagnostics:

<table>
<thead>
<tr>
<th>Header Byte</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5..0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>Block length in bytes incl. header</td>
</tr>
</tbody>
</table>

In the bit-structure every bit stands for an identifier diagnostic byte (if a bit is set to “1” then there is diagnostic information)

Channel-related diagnostics:
table 3.3

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4..0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Header Byte</strong></td>
<td><strong>Coding Input/Output</strong></td>
<td><strong>Identifier number</strong></td>
<td><strong>Channel number</strong></td>
</tr>
<tr>
<td>Channel number</td>
<td>Type of diagnosis</td>
<td>Coding type of channel</td>
<td>Coding type of error</td>
</tr>
</tbody>
</table>

The coding of the error-type (bit 4..0) are user-related. The other codings are standard.

If the ext_diag_overflow bit (bit 7) is set in octet 3 then there’s more diagnostic data then the buffer of a master can hold.

Parameterization is done in the power-up-phase and eventually in operate-mode if requested. The master uses this parameterization of a slave to identify itself with it. Also in this parameterization the mode is set of the slave how it will operate. Parameterization consist of the following aspects (this is conform the standard):

- The slave operates with or without watch-dog. If a master does not a request to a slave cyclically, the slave will get an alarm from that watch-dog. This is useful to check whether a master is still there.

- The definition of the TSDR. This is the time a slave has to wait before it can react on a request. (Idle time has expired). A minimum and maximum time is defined mostly, however conform the standard only minimum is required.

- Freeze- or sync-mode is defined. This useful for devices that require synchronization with in- and outputs.

- The slave is enabled or not enabled for other masters than the master that parameterizes.

- The allocation for a group-assignment is done. This is for global-control-commands given to a set of slaves. Each bit defines a separate group.

- The master-address of the master that gives the parameters is given to the slave to prevent other masters to give parameters to this slave. The slaves checks the master-address when it receives a parameter-telegram.

- User-related data can be given to a slave. This is useful to define what the slave should to in the Clear-mode of a master for instance.

The parameterization-telegram looks as follows:

<table>
<thead>
<tr>
<th>SD</th>
<th>LE</th>
<th>LEr</th>
<th>SD</th>
<th>DA</th>
<th>SA</th>
<th>FC</th>
<th>DSAP</th>
<th>SSAP</th>
<th>DU</th>
<th>FCS</th>
<th>ED</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD2</td>
<td>x</td>
<td>x</td>
<td>SD2</td>
<td>8xH</td>
<td>8xH</td>
<td>x</td>
<td>3DH</td>
<td>3EH</td>
<td>Octet 1..7 (+ user-data)</td>
<td>x</td>
<td>16H</td>
</tr>
</tbody>
</table>

-31-
Octet 1:
- bit 7: Lock-request (see table below)
- bit 6: Unlock-request (see table below)
- bit 5: Sync-request: If this bit is set, the slave will go in sync-mode as soon as the global-control command is received. If the slave doesn’t support sync-mode it will set the diagnostics-bit “Not supported” (see diagnostics)
- bit 4: Freeze-request: If this bit is set the slave will freeze its inputs as soon as the global-control command is received. If the slave doesn’t support freeze-mode, it will set the diagnostics-bit “Not supported” just like the description of bit 5
- bit 3: If this bit is set then the watch-dog is enabled, if not set the watch-dog is disabled
- bit 2..0: Reserved

Table 3.4: Definition of bit 6 and bit 7 in octet 1

<table>
<thead>
<tr>
<th>bit 7</th>
<th>bit 6</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>The parameter min TSDR can be changed. All other parameters stay unchanged</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>The slave will be unlocked for other masters</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>The slave is locked for other masters</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>The slave is unlocked for other masters</td>
</tr>
</tbody>
</table>

Octet 2: This octet contains the watch-dog-factor number 1, a value between 1 and 255 is possible

Octet 3: This octet contains the watch-dog-factor number 2, a value between 1 and 255 is allowed.

Octet 2 and octet 3 together define the watch-dog time:
TWD in seconds = 10 ms* WD1(octet2)*WD2(octet3).
For a 12 Mbit/s-network this time is set to 1ms.

Octet 4: This octet defines the minimum time a slave has to wait before it can send a response to the requesting master. This time is called the min TSDR

Octet 5: Ident_Number, high byte

Octet 6: Ident_Number, low byte

Octet 7: The group-identification-number is given in this octet (for global-control-commands)

Octet 8 and further: Defined by the user.
The slave will respond to this telegram-request by a short "E5H". If errors occurred while the parameters were given, then the slave will confirm this in the diagnostics-response in a later stadium before it enters the data-exchange mode.

After this parameterization a configuration-phase will start. In this phase, all aspects about the length, consistency and sort of data will be given. The slave will compare the configuration-data with the configuration-data it already contains (real_cfg). If these data are the same, then the slave can enter the data-exchange mode.

The configuration-telegram looks as follows:

```
SD LE LErr SD DA SA FC DSAP SSAP DU FCS ED
SD2 x x SD2 8xH 8xH x 3EH 3EH Octet 1..and more x 16H
```

In case of a modular device one identifier-byte per module is used (not the case with the controllers in the fruit-grading-project at Ellips B.V.).

The input- and output-areas can be grouped together and all groups are identified by one identifier-byte.

```
7 6 5 4 3 2 1 0
```

Format of the identifier-byte

Bit 3..0 are used to indicate the length of data:
- 00H means 1 byte or 1 word (depends on format-structure)
- ... 0FH means 16 bytes or 16 words (depends on format-structure)

Bit 5..4 are used to identify the sort of data:
- 00 indicates a special identifier-format
- 01 indicates input-data
- 10 indicates output-data
- 11 indicates input and output combined

Bit 6 indicates the format-structure:
- 0 byte, byte-structure
- 1 word, word-structure (if words are sent then the protocol will first send the high byte of the word followed by the low byte).
Bit 7 is to indicate whether consistency is needed or not
0 means consistency over byte or word (depends on format-structure)
1 means consistency over the whole length of data

If the bits 5 and 4 are 00 then the special identifier-format is used. This format replaces the standard identifier. The meaning of this special identifier-format is now explained:

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

Special identifier-byte

To begin with:

Bit 5 and bit 4 are fixed to “0” to indicate the special identifier.

Bit 3..0 indicate the length of manufacturer-specific data

This length has two different meanings depending on the request:

If a Chk_Cfg-request is done by a master of class 1 then these 4 bits mean:

00H: There are no manufacturer-specific data
01H-0EH: There are manufacturer-specific data of the specified length
0FH: There are no manufacturer-specific data, the verification of it is not necessary

If a Get_Cfg-request is done by a master of class 2 then these 4 bits mean:

00H: There are no manufacturer-specific data
01H-0EH: There are manufacturer-specific data of the specified length
0FH: Not allowed

Bit 7 and bit 6 indicate:

00: A free place will follow after this identifier-byte
01: one length-byte for inputs will follow after this identifier-byte
10: one length-byte for outputs will follow after this identifier-byte
11: one length-byte for outputs and one length-byte for inputs will follow after this identifier-byte

Finally the length-bytes are defined as follows:

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

Structure of a length-byte
Bit 5..0 indicate the length of inputs/outputs:
   00H: 1 byte or one word (depends on format-structure)
   ...
   3FH: 64 bytes or 64 words

Bit 6 indicates the length-format:
   0: byte
   1: word

Bit 7 indicates about the consistency:
   0: consistency over byte/word
   1: consistency over the whole length

Table 3.5: An example to clarify the configuration-setup

| Octet 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| Octet 6 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |

Manufacturer
Specific
Data

The table stated above is an example of how the configuration of a slave can be.

The first octet indicates that a special identifier-byte is used. It also indicates that 3 bytes of specific manufacturer-data will follow after the length-bytes. Finally by bit 7 and bit 6 it is indicated that one length-byte for outputs and one length-byte for inputs will follow after this identifier-byte.

The second octet indicates that 16 words of output will follow in data-exchange and that there’s consistency over the whole length.

The third octet indicates that 8 words of input will follow in data-exchange and that there’s consistency over the whole length.

The last three octets are the bytes that contain specific manufacturer-data.

At last: The acknowledge of this will be a short "E5H" if everything is correct. If the configuration is not correct, then the slave will report that in the diagnostics-phase. If the configuration is not correct, then the slave will not be able to enter the data-exchange-mode.
In data-exchange-mode the telegrams for output-exchange and input-exchange look as follows:

<table>
<thead>
<tr>
<th>SD</th>
<th>LE</th>
<th>LEr</th>
<th>SD</th>
<th>DA</th>
<th>SA</th>
<th>FC</th>
<th>DU</th>
<th>FCS</th>
<th>ED</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD2</td>
<td>x</td>
<td>x</td>
<td>SD2</td>
<td>xxH</td>
<td>xxH</td>
<td>x</td>
<td>Octet 1..x user-data</td>
<td>x</td>
<td>16H</td>
</tr>
</tbody>
</table>

If a slave is just an output-device it will respond with a “E5H”, a short acknowledge. If there was an error it responds with a high-priority-telegram.

Every master can read the input-image of a slave (only if slave is in data-exchange-mode). This is done by the following telegram-request:

<table>
<thead>
<tr>
<th>SD</th>
<th>LE</th>
<th>LEr</th>
<th>SD</th>
<th>DA</th>
<th>SA</th>
<th>FC</th>
<th>DSAP</th>
<th>SSAP</th>
<th>FCS</th>
<th>ED</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD2</td>
<td>05H</td>
<td>05H</td>
<td>SD2</td>
<td>8xH</td>
<td>8xH</td>
<td>x</td>
<td>38H</td>
<td>3EH</td>
<td>x</td>
<td>16H</td>
</tr>
</tbody>
</table>

The slave will respond with a telegram as defined in the data-exchange-mode. Only the DSAP/SSAP are mirrored. Note that these two bytes are the first two bytes in the DU-area.

Also every master can read the output-image of a slave (only if slave is in data-exchange-mode). The following telegram is sent by the master:

<table>
<thead>
<tr>
<th>SD</th>
<th>LE</th>
<th>LEr</th>
<th>SD</th>
<th>DA</th>
<th>SA</th>
<th>FC</th>
<th>DSAP</th>
<th>SSAP</th>
<th>FCS</th>
<th>ED</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD2</td>
<td>05H</td>
<td>05H</td>
<td>SD2</td>
<td>8xH</td>
<td>8xH</td>
<td>x</td>
<td>39H</td>
<td>3EH</td>
<td>x</td>
<td>16H</td>
</tr>
</tbody>
</table>

And again: the slave will respond with a telegram as defined in the data-exchange-mode. Only the DSAP/SSAP are mirrored. Note that these two bytes are the first two bytes in the DU-area.

With a SDN-service, global control-commands can be sent to a slave or a group of slaves. Global commands are freeze, sync, clear_data, unfreeze and unsync. The telegram to be sent for this purpose:

<table>
<thead>
<tr>
<th>SD</th>
<th>LE</th>
<th>LEr</th>
<th>SD</th>
<th>DA</th>
<th>SA</th>
<th>FC</th>
<th>DSAP</th>
<th>SSAP</th>
<th>DU</th>
<th>FCS</th>
<th>ED</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD2</td>
<td>x</td>
<td>x</td>
<td>SD2</td>
<td>8xH</td>
<td>8xH</td>
<td>x</td>
<td>3EH</td>
<td>3AH</td>
<td>Octet 1..2</td>
<td>x</td>
<td>16H</td>
</tr>
</tbody>
</table>
Where:
  Octet 1:
    bit 7: Reserved
    bit 6: Sync
    bit 5: Unsync
    bit 4: Freeze
    bit 3: Unfreeze
    bit 2: Reserved
    bit 1: Clear_Data
    bit 0: Reserved
  Octet 2: The group-number is indicated here
3.3 Profibus-DP Master

Two kinds of masters are possible in Profibus-DP. The first one is a master of class 1 and the other one a master of class 2. A master of class 1 polls the associated slaves cyclically to submit data to its local users. A master of class 2 requests or sends acyclically messages with masters and slaves. A master of class 2 is used as a programming- and/or management- system. In a realistic system it's possible that one master acts as a master class 1 as well as a master class 2.

A master of class 1 has the following functions in a system:

- Initialization of the DP-system
- Parameter-assignment and configuration of the dedicated DP-slaves, if parameterization and configuration was okay then the data-exchange-mode can be entered, otherwise a slave will be made inactive
- Cyclic data-exchange with the DP-slaves, this is done by polling all slaves sequentially
- Monitoring of the DP-slaves
- Fetching of diagnostic information from the DP-slaves if errors occur

A master of class 2 has the following functions in a system:

- Commissioning of new active stations
- Down- and upload of parameters from and to other masters
- Checking of the configuration of slaves
- Downloading of in- and output-images of DP-slaves

A master communicates in four modes with the slaves:

- Off-line: No communication at all between a master and his slaves. This is the starting phase when the system is booted up.
- Stop: There's no communication between the master and his slaves, however a master of class 2 can read out diagnostic information of a class 1 master
- Clear: This mode is the first real communication with the slaves. The slaves are parameterized and configured. If all went well the data-exchange-mode can be entered. In this state "00H" is sent as output indicating that the slaves are still to be inactive in processing outputs
- Operate: Now the master can cyclically exchange data with the dedicated DP-slaves. Every slave is polled in a sequence. Each slave can receive output-data and send input-data to the master in one poll-cycle. If an error occurs at the slave then this will indicated
to the master with a high-priority telegram. A poll-cycle looks as showed in the picture below:

![Diagram of telegram sequence in Profibus-DP system]

Fig. 3.7: The telegram-sequence in the Profibus-DP-system

The master will always start in off-line-mode (power-up). After this the following sequence has to be made to enter the productive (i.e.: operate) mode:

Off-line ➞ Stop ➞ Clear ➞ Operate

Furthermore to assure the correct timing in the DP-system, the master (de-)activates the watch-dog-timer of the slave. If the master doesn’t communicate with the slave within the specified watchdog-time (defined by the watchdog-factor-parameters in the parameter-telegram), the slave will assume that an serious fatal error occurred (for example a power-failure or line-break). If this happens then the master has to parameterize and configure the slave again to give the slave the possibility to enter the data-exchange mode again.
3.4 Profibus-DP Slave

A slave in the Profibus-DP-protocol can be interrogated by a master from class 1 as well by a master class 2. To understand how a slave must behave if it uses the Profibus-DP-protocol to communicate, we’ll show a figure that presents the state-machine of the slave. Every ellips in the picture is a state and every arrow is an event that directs the slave into a possible other state.

![State-machine from a Profibus-DP-slave](image)

We’ll start from the upper edge with the first phase: Power-on-mode. In this state only, an address-change-request is permitted only from a class 2 master. If the slave supports this
changing of the station-address, it shall have non-volatile memory on board. After this state the slave will go in wait-parameter-mode.

In the state wait-parameter the slave will await the arrival of parameter-telegrams or get-configuration-telegrams. All other kinds of telegrams are denied in this phase and still data-exchange is not possible. Diagnostic information is asked for before the data-exchange-mode is entered.

After the wait-parameter-state, the Wait_Cfg-state is entered. The slave expects configuration-data from the master to check whether the master knows the length, sort and consistency of the exchange-data the slave expects. A master of class 2 can ask for the configuration of a slave by a Get_Cfg-request. After the master gets the configuration of the slave, it will send it back to the slave to check for correctness.

After these two last states are approved, then the data-exchange-mode will follow. In the data-exchange-mode the real data is exchanged (i.e.: the out- and/or input-data the slave deals with). During this state diagnostic information can be requested by the master if the slave indicates an error to the master with a high-priority telegram (this is indicated in the FC-octet together with eventually input-data from the slave).

The images of the inputs and outputs of a slave can be requested by every master. However exchanging of data from master to slave such that the slave has to process it, is only possible by the master which parameterized and configured the slave before. This is done for security-reasons.

In the next and last paragraph of this chapter we'll discuss some timing-aspects in a straightforward system (mostly used: one master and several slaves dedicated to it).
3.5 Timing-aspects in Profibus-DP

Important timing-parameters are the token-rotation-time and the maximum time the master may poll a associated slave. The optimum in timing depends of the configuration of the system. For high-speed transmissions the number of used masters should be set to a minimum. Also the system-reaction-time is very important. This system-reaction-time depends on the number of total used stations, the number of used bytes per slave and the number and type of masters in the system. In a one master-many slaves system, the calculation of the system-reaction-time is straightforward. In this case the token-rotation-time is determined by the number of slaves that are dedicated to the master and the time that the master needs to look for new active stations. The system-reaction-time is the time that a slave has to wait before it is polled again by the master (in a one master-system and if every slave is polled cyclically). The system-reaction-time is also affected by the TSDR (the reaction time of a slave after which it may and must respond). The minimum slave interval has also to be taken into the calculations. This time is defined as the time that the slave needs to have between two polling cycles. With the use of nowadays asics this time is approximately 100 µs.

We’ll now make a calculation based on data-exchange-mode and all slaves working correctly (i.e.: no errors and thus no diagnostics needed). As we already know the time of transmitting one bit is defined as Tbit. At a transmission-rate of 12 Mbit/s, 1 Tbit will take 83 ns. If we consider that 11 bits are needed to transmit one byte of information, the time to transmit this byte will be: $11 \times 1 \text{Tbit} = 0.917 \mu \text{s}$. To simplify the calculations we’ll assume that this time is approximately 1 µs. This is a good approximation because we’re interested in an average time with sometimes an diagnostic telegram.

Let’s look at the picture below:

![Diagram of poll-cycle timing for one slave](image_url)

Fig. 3.9: The poll-cycle timing for one slave
The poll-cycle can be calculated easy: The time of one poll-cycle in normal data-exchange-mode is the sum of the time needed by the request telegram plus the time the slave has to wait before it may respond plus the time needed by the response-telegram.

If we assume the sync-time $T_{sync} = 33$ Tbit and the $T_{id} = 75$ Tbit and with use of the SPC3 the $T_{SDR} = 30$ Tbit (realistic system configuration, see also the minimum and maximum $T_{SDR}$ in the table showed on this page) then we can calculate the time of one poll-cycle without the data-bytes (the time to send one byte is almost 1 $\mu$s). In a normal request- and response-telegram the telegram consists of 9 bytes (the header).

Thus the poll-cycle time $T_{mc} = (2*\text{length of header in bytes}) * 11 \text{ bits} + T_{SDR} + T_{syn} + T_{id} \ (\text{Tbits}) \Rightarrow T_{mc} = (198 + 30 + 33 + 75) \ \text{Tbits} = 28 \ \mu\text{s}$

Therefore for data-exchange we can assume the poll-cycle as the sum of the basis-time ($28 \ \mu\text{s}$) plus the $1 \ \mu\text{s}$/data-byte. For the actuator-controller with two output-bytes this implicates a poll-cycle of $30 \ \mu\text{s}$/actuator-controller-slave.

If we now look at the fruit-grading-system as needed by Ellips B.V. with one sensor/encoder-slave that has to be polled approximately 1000 times per second this yields that the system reaction-time of the system has to be 1 ms. In this raw calculation this implicates that about 30 actuator-controller-slaves could be connected on one bus while the system reaction time still is 1 ms. However, this is a calculation without the aspects of diagnostic data-fetches. If diagnostic information is needed by a master (because a slave indicated an error) then the need for compact diagnostic data will be obvious (to assure that the system-reaction time will not be more than 1 ms).

Table 3.6: The $T_{SDR}$-parameters minimum and maximum depending on rate.

<table>
<thead>
<tr>
<th>Transmission-rate in kBit/s</th>
<th>9.6</th>
<th>19.2</th>
<th>93.75</th>
<th>187.5</th>
<th>500</th>
<th>1500</th>
<th>12000</th>
</tr>
</thead>
<tbody>
<tr>
<td>min $T_{SDR}$</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>max $T_{SDR}$</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>100</td>
<td>150</td>
<td>800</td>
</tr>
</tbody>
</table>

We can conclude now that the system-reaction time is defined by the slaves that need to be polled at the highest-frequency. This implicates also that slaves that do not need to be polled at a high frequency will receive output-data while not necessary (i.e.: more times the same value). Also input-data will then be requested more than needed. In this case the global-control commands $\text{sync}$ and $\text{freeze}$ will be useful.
In the figure on this page finally, we can see the performance of a one-master-system and more dedicated slaves. The slaves are all exchanging 2 bytes of output data and 2 bytes of input data. We're assuming that all slaves work correctly and that the system is completely in data-exchange-mode. From the figure we can see that if we use a transmission-rate of 12 Mbit/s, that the reaction-time stays under 1 ms if 30 slaves are used. This implicates that every slave can be polled 1000 times in a second.

![Graph showing reaction-times with transmission rates up to 12 Mbit/s in data-exchange-mode.](image)

Fig. 3.10: Reaction-times with transmission rates up to 12 Mbit/s in data-exchange-mode
4.1 Design of the hardware of the actuator-controller

To design a dedicated actuator-controller in a Profibus-DP-system, accurate specifications have to be made. In this specific application the actuator is a relay, so we could speak of a relay-controller in this case. The specifications are meant for the hardware as well for the software. At first the specifications should be made for one hundred percent for the Profibus-DP-protocol, otherwise the controller will not be a approved Profibus-DP-module. A special user-organization in Germany certifies approved modules as Profibus-product. This user-organization is called the PNO (Profibus Nutzer Organisation).

Secondly specifications have to be made for the application of the slave itself. For high-speed performances the number of data-exchanges with the dedicated master has to be set at a minimum. Only at error-events full diagnostics should be done to prevent unnecessary overhead. This is also what Profibus-DP describes in the standard. In the next paragraphs the actual design-process will be described for the hardware as well as the software. Some high-level descriptions will be given to explain the design-process. This will be done by splitting the hardware and software in modules, and from that point the modules themselves will be described.

The specifications for the actuator-controller are:

- 16 actuators need to be controlled per slave-device, in this particular case of controlling relays maximum ratings of possible currents and voltages are defined as follow:
  - a maximum peak-current of 1-2 A should be possible with a continuous current of 0.75 A,
  - a maximum peak-voltage of 40 V should also be possible without damaging the device

- diagnostics need to be fed back to the master:
  - diagnostics about the actuators:
    - overload (active mode)
    - shorted load (active mode)
    - shorted output to ground (inactive mode)
    - open load (inactive mode)
  - diagnostics about the slave-device itself:
Design of an intelligent actuator-controller using Profibus-DP

- power-failure caused by a line-break, before all power-supply has gone the device should indicate this state by communication with the master or quick storage about this status in the eeprom
- faulty download of new software-code
- wrong parameterization
- wrong configuration
- watchdog alarm

- a flash-memory should be used to allow software-updates
- a small eeprom for quick storage of for example the station-address and power-failure-status
- software-updates must be possible by downloading new code over the Profibus
- the slave-device has to be at least fully compatible with the standard of Profibus-DP to certificate this device at the PNO. Only with a approved certificate, the slave-device will own an unique identification-number (Ident_Number). With these specifications hardware and software can be designed. In the next paragraphs the design of the hardware is discussed in phases and after that the design of software is described.
4.2 The total picture of the actuator-controller-hardware

First of all the total picture of needed modules is stated in figure 4.1. We can see how all modules are connected together. In this chapter we will only describe the hardware in high-level. For the details of the hardware-design, the schemes at Ellips B.V. show the design made with the hardware-design-program Capture from the company Orcad. Let’s take a look at the picture:

At the most left side we can see the fieldbus itself. This fieldbus is conform the Profibus-DP-protocol a shielded twisted pair cable. Because of the high transmission rate of 12 Mbit/s, special requirements are needed. The cable has to be a type A cable with a minimum capacitance. In the middle we see all control-logic and a cpu with memory connected to it. As we also can see an oscillator is needed to provide a correct clock-signal for the asic and cpu. At the most right the powerdriver-module and actuators are situated. All the modules will be described in more detail in the following paragraphs. The actuators (relays, other solenoid products and lights for instance) are not directly driven by the cpu, but instead so-called powerdrivers are used to provide the correct high current and voltages. These powerdrivers also have a special heatsink to prevent overtemperature of these devices.
4.3 Modular description of hardware

As already said in the introduction, the hardware and software will be divided into several modular parts. In this paragraph the hardware-modules will be explained. As we will see, some modules are very general and can be used again in other hardware-designs for the fruit-grading-system needed by Ellips B.V. In this case we can think of the weight-controller and the sensor/encoder-controller belonging into the Profibus-fruit-grading-project also.

4.3.1 Design of the Profibus-interface

If we study the standard of Profibus-DP, we can see that an interface is needed between the hardware of the application and the bus itself (Profibus in this case). We now have the first module found in the hardware and we call it the Profibus-interface. If we look at figure 4.1 then we can see the Profibus-interface at the left side of the hardware. This module is called the PB-driv (Profibus-driver). In figure 4.2 we see a more detailed description of this module.

Fig. 4.2: Functional hardware-description Profibus-interface

The UART in the asic SPC3 is responsible for the receive- and transmit-signals that come from and go to the Profibus. These signals are converted in serial signals by the UART from a parallel buffer-structure. The opto-couplers are used for safety-reasons. They prevent disturbances from the hardware to occur at the Profibus and therefore disturbing the whole
Design of an intelligent actuator-controller using Profibus-DP

The opto-couplers are from Hewlett Packard the HCPL 7100, capable to work at high transmission rates. The opto-coupler for the RTS-signal is also from Hewlett Packard the HCPL 0601. This interface gets its own power from the powersupply as described in the next paragraph. The RTS-signal (request to send) has two purposes: first if the asic SPC3 wants to send then the RTS-signal enables the RS-485-driver to send the TxD (transmit data). On the other side it indicates the direction that has to be taken for the repeater (if any) in the Profibus-system. The RxD (receive data) is always enabled, so that the asic SPC3 can always listen and react on telegrams addressed to it. The RS-485 is from Analog Devices the ADM1485, capable to drive at high transmission rates with a very good slew-rate.

4.3.2 Design of a power-supply

Together with the Profibus-twisted-pair-cable, a power-line with a DC-voltage of 24 Volts is distributed over the system. With this power-line all slaves and relays are supplied with the appropriate power. However, only the relays use a voltage of 24 Volts. The control-logic itself uses a voltage of 5 Volts. This introduces the need for a DC-DC-converter. With a standard DC-DC-converter the power-dissipation in the converter-component is too high introducing to much heat. A search for a switched voltage regulator was done to eliminate the heat as much as possible. A switched voltage regulator switches the output-voltage ‘off’ and ‘on’ at a high speed of approximately 100 kHz. This high speed switching gives a mean value as needed. A low-pass filter filters the high frequency spikes out. The WD-device (watchdog) measures continuously the +24 V line to indicate a occasionally power-failure caused by a line-break for instance.

![Diagram](image)

Fig. 4.3: Functional description of the powersupply needed for the actuator-controller
4.3.3 The "glue"-logic: in- and outputs needed

We'll describe every in- and output-signal shortly.

The combined address/data-lines from the microcontroller are 8 inputs for the glue-logic. With the ALE (Address-latch-enable) the lowest 8 address-lines are generated. Address-lines 13 to 15 are used for the internal registers in the glue-logic. These internal registers are the latches needed to store the actuator-information for the powerdrivers and the memory-management-units (MMUs). The MMUs are used for bankswitching, which makes it possible to address flash-memory and sram in every possible way (two modes for the actuator-controller: flash-update-mode and normal-mode). The A14_OUT-A16_OUT are the resulting addressing lines.

The WRN and RDN and PSEN come from the microcontroller and are meant to indicate the direction of data: read or write, where the PSEN is to read the program-memory. The RD_OUTN is the logical OR-function of PSEN and RDN. These signal can be combined since only one on them can occur in a time.
The RST_OUTN-signal is the inverted signal RST_80C32 since the powerdrivers need a logical ‘0’ signal to reset. The chip-select-signals SPC3_CSN, FLASH_CSN and RAM_CSN are used to select the correct chip (i.e.: the SPC3, Flash-memory or SRAM). Finally the 16 lines that go to the powerdrivers contain the internally latched information for the actuator. A logical ‘0’ will switch the actuator while a logical ‘1’ means inactive mode for the powerdrivers (i.e.: the actuators are not switched).

4.3.4 Design of the memory-module

The memory-module can be divided into three parts:

1. A RAM-module
2. A small EEPROM (128 bytes data)
3. Flash-memory

Fig. 4.5: Functional description of the memory-module
The RAM is addressed via the glue-logic and microcontroller to store variable-data and new program-code for the flash-memory. Also the stack from the microcontroller is located here if wished.

The flash-memory is used to hold the complete actuator-controller-software. Also specific parameter-data can be stored here. The Ident_Number used to identify the actuator-controller is stored here.

To assure that it is always possible to download new firmware, a boot-sector-program is always present in the lowest 16K-part of the flash-memory. This program has only the most elementary communications-code capable to download new code over the Profibus with the asic and Profibus-DP-protocol.

The EEPROM is used to store the actual station-address of the device. The EEPROM is connected to the microcontroller over two lines. One line provides a clock-signal, while the other line uses the PC-protocol to transmit data and address. In this way the EEPROM is quickly addressable and is less complex to program with new data than the flash-memory.
4.3.5 Design of the interface to the actuators

The powerdrivers are used to conduct the connected actuators (relays in this case). The powerdrivers are from Siemens and are called TLE5216G. Every powerdriver can drive 4 relays and diagnose also 4 relays. The diagnoses that can be made, are: overload and short circuit in active mode (the relays are switched) and open load and short circuit in inactive mode (the relays are in rest). Several powerdrivers can be cascaded with the diagnostics-lines. These lines provide the diagnostics conform the serial peripheral interface (SPI). The serial diagnostic lines are cascaded: Serial information from a previous powerdriver goes into the input-serial-line from the next powerdriver. Finally the output-serial-line from the last powerdriver is driven to port 1.1 from the microcontroller. This port is an input-port for timer 2 of the microcontroller. On an event (serial line goes low) an
interrupt is set in the microcontroller. The microcontroller can handle this interrupt and start a
diagnosis-phase to fetch the diagnosis-data from the powerdrivers. If an error occurred, an
error-flag is set and diagnostic data is prepared before a high-priority-telegram is sent to the
master to indicate that there’s diagnostic data from the actuator-controller. After the errors are
processed, the powerdriver-channel can be reset.
4.3.6 Design of the core of the actuator-controller

Figure 4.7 shows the core of the actuator-controller. In fact theasic SPC3 handles all communication with the Profibus. All telegrams are processed here and by a 1,5 Kbytes RAM all data is given to the microcontroller for further processing. The SPC3 indicates the kind of telegram with an interrupt-register. With new telegrams arriving an interrupt is sent to the microcontroller. This is done with the INT_SPC3-signal, which has highest priority at the microcontroller (INT0). The SPC3 is addressed just like normal memory. A clock-signal of 48 MHz is provided to the SPC3 and this signal is divided by 4 and sent to the microcontroller (i.e.: The microcontroller works with a clock-rate of 12 MHz). The SPC3 can be reset by the microcontroller trough the RST_SPC3-line. A switch connected to the microcontroller is used to reset the station-address to default (126) if the device is removed from the system. The switch can also be defined for debug-purposes. To assure everything will work well, the microcontroller has to toggle the watch-dog-circuit, indicating that there’s still activity. If the watch-dog-timer goes off, then the microcontroller will be reset and the whole device has to initialize again. The watch-dog also checks the power. If the power is going down, there’s still some time (1-10ms) to indicate the power-failure to the master. Finally, a debug-option is introduced by the microcontroller trough the UART of it.
4.4 Memory-model of the actuator-controller

As figure 4.8 shows, the memory-model is defined in a special way. First it's useful to say that the microcontroller-memory is divided into two parts. One part is for data and the other part contains the program-code. Because the flash-memory normally contains the program-code, but also has the possibility to be updated, it is necessary to run code from SRAM also. Therefore, with a bank-switching-technique it's possible to define SRAM or Flash-memory in program-area or data-area (F0-F2 and R0-R2 indicate 16Kbytes block each of flash or SRAM). Further the 2K-part from data-space from address E000H is designated to the SPC3-RAM. The latches for the powerdrivers and the Memory-Management-Units (MMU for program- and data-space separately) are memory-mapped from address C000H and higher, as

<table>
<thead>
<tr>
<th>Program-memory</th>
<th>Data-memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2K SPC3</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>16K F2 or R2</td>
<td>F0 or R0</td>
</tr>
<tr>
<td>16K F1 or R1</td>
<td>F1 or R1</td>
</tr>
<tr>
<td>16K F0 or R0</td>
<td></td>
</tr>
</tbody>
</table>

**Fig. 4.8: Description of the memory-architecture**
indicated by the picture. Although 128K of Flash-memory and SRAM is provided, the space above CO00H isn’t designated to them. However, if needed, the possibility can be built in. Finally, an EEPROM can be used to store the assigned station-address. It’s possible to store more data in the EEPROM (128 bytes available), for instance the software-version-number.

4.5 Design and use of software of actuator-controller

In this paragraph the software for the actuator-controller will be discussed. The design of the software can be divided into two parts. The first part is the communication-software for the Profibus. The second part is the application-software in combination with the communication-software. In the next two paragraphs these two parts will be discussed. The communication-software was bought from Siemens because it is written especially for the Profibus-slave-asic SPC 3. Integrated with this software, the user can now write his own application-specific software. The application-software consists of the controlling of the relays and the diagnosing of the relays in combination with the asic SPC 3 and the communication-software. Also a part of the application-software is the possibility of downloading new software into the flash-memory that is on board of the actuator-controller. In the last paragraph of this chapter the design of the glue-logic in a FPGA (Field Programmable Gate Array) will be explained. For the design a software-package is used called VHDL (Very high-speed Hardware Description Language). VHDL is a description-language for the design of complex digital hardware. With this software it is possible to simulate the hardware before it is made.

4.5.3 General software-functions needed by the actuator-controller

In general some software-functions are needed to boot up and initialize the actuator-controller. After these first functions are called, more functions need to be called:

The first things the actuator-controller does after a reset in boot-up:
• run boot-software with very elementary functions:
  • Check if valid software-code is available, if so then run this code (in boot-sequence)
  • If code is not available: activate initializing function and flash_update_function
    after updating and no errors this new code is run
• Initialize the SPC3 (in off-line mode)
• Enter the wait_parameter_state
After that, enter the Chk_Config_State
If everything is still okay then enter the data-exchange-mode
In this data-exchange-mode drive inputs of powerdrivers every data-exchange-cycle and set an error-flag if something went wrong in the powerdriver.
If the error-flag is set then prepare diagnose-data
If a request for flash_update arrives then perform a flash_update, code must be run from sram in this phase and the flash-memory gets new program-code. However the elementary boot-up software may never be overwritten, otherwise the actuator-controller might never start again...
The data-exchange-loop is always done until a new_parameter-request or new_configuration-request arrives. If this happens then the initializing is done again

### 4.5.2 Use of SIEMENS firmware for control of the asic SPC3

When Ellips B.V. decided to use an asic of Siemens, to be specific the asic SPC3, it was a logical next step to buy it together with the firmware especially designed for the SPC3 and the microcontroller 80C32 (based on 8051). This source-code was compiled by Siemens to specific assembly-code depending on the processor-controller. In the hardware for the actuator-controller a 8051-based microcontroller is used. Siemens has used the 8051-compiler software from Keil Software V5.0. Therefore this package was also bought by Ellips B.V. to avoid problems while compiling source-code designed for the SPC3 (for instance symbol-tables).

Because the SPC3 has the complete state-machine built in from the Profibus-DP-protocol, the software has all indications about events from the SPC3 available to the user with macro-definitions. The complete software-package from Siemens for this SPC3 is called DPS2. This package contains the interrupt-module, which handles parameter-assignment and configuration, help functions to calculate the buffer-organization from the desired configuration and macro's and definitions to access the register-structure in an easy way. This software can be integrated with the application. In figure 4.9 all sequences and function-calls from and to the DPS2 or user are shown.
Fig. 4.9: The interface between the user and the SPC3 (DPS2-software)
4.5.3 Design of software for control and diagnostics of actuators

As we’ve mentioned in the introduction, the specifications of the actuator-controller have to be realized for a part in hardware and for the other part in software.

For a correct control of the actuators, some functions need to be defined:

- Send output to the right actuator (this is done by addressing the right latch in the glue-logic)
- A diagnosis-routine is needed to perform diagnoses at the powerdrivers
- Synchronization-commands have to be supported from the Profibus-DP-protocol, to assure that the actuators are driven at the right time.
- If errors are detected, a correct error-handling routine has to be defined to solve the problems.

4.5.4 Use of VHDL for programming of the FPGA

Instead of separate components like NAND’s, OR’s, LATCHES and other kinds of logic, one component is used. This component is called a FPGA. FPGA is the abbreviation for Field Programmable Gate Array. With this FPGA it is possible to define and program the “glue-logic”. Glue-logic is the name for all interfacing logic between memories, cpu, asics, watchdogs etc. The programming of a FPGA is done with a specific programming device. This device needs specific code for the FPGA to fulfill the needed functions. The code is generated by a compiler, that compiles a high-level description language into FPGA-specific code. The high-level language that mostly is used for this purpose is VHDL. VHDL is the Very high-speed Hardware Description Language. The specific VHDL-code for the desired GLUE-LOGIC is given in appendix A.
Chapter 5

5.1 Choice of power-drivers

Table 5.1: Comparison of powerdriver-IC's

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Type</th>
<th>input</th>
<th>output</th>
<th>diagnostic</th>
<th>max. output-voltage</th>
<th>max. output-current</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motorola</td>
<td>MC33298</td>
<td>8 input</td>
<td>8 channel</td>
<td>yes, spi, short circuit open load overload, over-temp.</td>
<td>65 V clamp</td>
<td>1 A/ ch.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>serial</td>
<td>parallel</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Siemens</td>
<td>TLE5216G</td>
<td>4 channel</td>
<td>4 channel</td>
<td>yes, spi, short circuit overload open load</td>
<td>75 V clamp</td>
<td>2 A/ ch.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>parallel</td>
<td>parallel</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Siemens</td>
<td>TLE5226G</td>
<td>4 channel</td>
<td>4 channel</td>
<td>yes, per channel short circuit overload open load</td>
<td>60 V clamp</td>
<td>2<em>3A 2</em>5A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>parallel</td>
<td>parallel</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Because the powerdrivers need to drive at least a continuous current of 0.75 A at a voltage of a maximum of 60 V, the MC33298 is not appropriate. Because the TLE5226G is not available yet, the TLE5216G was chosen. Because high currents are involved, a special heatsink is required on the PCB. The package of the TLE5216G is well suited for this. Also a minimum of diagnostics lines is an advantage. The price of these powerdrivers is approximately HFL 8,- /piece/with 300 pieces/year.
5.2 Choice of the microcontroller

Choice of the appropriate microcontroller (actuator-controller):

selections have to be made considering:

- 8 or 16 bit µC (8 bit preferred because of the simple function of the actuator-controller)
- speed (not critical, but enough to control 16 actuators at once)
- watchdog (in case of malfunction of the relays)
- interfacing with an ASIC (the ASIC has to be designed for use with Profibus-DP, i.e. layer 1 and 2 of the OSI-model will be replaced by the ASIC, i.e. software replaced by hardware)
- price (as low as possible, 8 bit is cheaper than 16 bit)
- number of ports (how many control-lines are needed)
- serial communications-protocol (UART, I2C,...)
- software-compatibility with a standard 8051 (experience with 8051-software at Ellips)
- number of pins (design of pcb easier)
- Flash-ROM preferable instead of EPROM/OTP-ROM/mask-ROM, because of possibility to upload code via Profibus.

A lot of microcontrollers do exist nowadays. Presenting all types would be very time-consuming, while the standard-functions are only needed. Many manufacturers produce microcontrollers based on the 8-bit 8051 from Intel. Because some experience is available at Ellips B.V. with the 8051-family, the 80C32 was chosen. Also this processor is recommended in the bus-interface with the asic SPC3 from Siemens. The 80C32 is delivered without internal program-rom. Philips and Siemens have both 80C32's. Finally the SAB-C501-LN20 from Siemens was chosen, because of the very low price of HFL 1,75 /piece. However the Philips P80C32 is fully compatible with this processor from Siemens. This means that a second source is always available.

5.3 Choice of flash-memory

Several flash-memories are available, but not all of them are programmable in a trivial way. The most convenient way is to program the memory with only software and not with additional hardware. A lot of flash-memories need a higher voltage for programming the memory, only a few of flash-memories are able to be programmed by the same 5-volts voltage that is used to supply the power for the device. The use of the same voltage for either programming or reading the device is preferable, because it eliminates the need for additional
Design of an intelligent actuator-controller using Profibus-DP

hardware. Manufacturer AMD has introduced some flash-memories that can be programmed with a voltage of 5V. The minimum size of these flash-memories is 128Kbytes. These flash-memories have the possibility to erase whole blocks (sectors) of memory at once. The chosen product from AMD is the flash-memory AM29F010. This device has 128Kbytes onboard and programming-software is presented with it and very straight-forward. Second sources for this device are from ATMEL with the 29C010 or eventually Toshiba with the TMS29F040. The price for this memory is about HFL 8,-/piece.

5.4 Choice of SRAM

The choice of static ram (sram) will depend on the following:

- price
- speed
- second sources
- compatibility
- availability
- size of memory (64K or more needed)

Several manufacturers produce standard sram’s.
Some manufacturers are: Samsung, Hitachi, UMC, Toshiba.

A sram from Samsung was chosen because of the availability and the price. The type is the KM681024-70LL. This type has a size of 128Kbytes and is pincompatible with the 64Kbytes-version KM68512. Also second sources are available from UMC and Hitachi. The price of the sram differs continuously and is about HFL 6,-/piece.

5.5 Choice of opto-couplers

To make the right choice in opto-couplers for the Profibus-interface we’ll have to look at the maximum possible transmission-rate. Since Profibus-DP is also defined at speeds of 12 Mbit/s, the opto-couplers should be able to deal with that speed. Recommended opto-couplers are opto-couplers from Hewlett-Packard (recommended in Profibus-literature). The chosen Hewlett-Packard opto-couplers are the HCPL7100 for ultra-high-speed purposes and the HCPL0601 for the RTS-signal (lower speed then 12 Mbit/s). These opto-couplers are available in a surface-mounted-device-package (SMD).
5.6 Choice of serial RS-485-drivers

As told in the previous paragraph Profibus-DP is defined for transmission rates up to 12 Mbit/s. Therefore a recommended driver from Analog Devices is used. The type is the ADM1485. This driver is also available in SMD-package.
5.7 Choice of an ASIC for use with the Profibus-DP-protocol

Choice of proper ASIC (actuator-control for instance):
selections to be made:

- is it compatible with Profibus, and more specific with Profibus-DP?
- price
- easy to interface with a μC?
- how many providers of Profibus-asics? (search on Internet)
- is there a full hardware-solution or combined hardware/software-solution to the layer 1 and layer 2 from the OSI-model?
- availability
- support with software

Siemens is the main producer of suitable asics. Also the most support is delivered from Siemens. Table 5.2 shows the Profibus-asics from Siemens. The SPC3 was chosen.

Table 5.2: Comparison of Profibus-asics from Siemens

<table>
<thead>
<tr>
<th>Profibus-ASIC’s</th>
<th>LSPM2</th>
<th>SPM2</th>
<th>SPC3</th>
<th>SPC4</th>
<th>SIM1</th>
<th>ASPC2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>simple Slave solution</td>
<td>simple Slave solution</td>
<td>intelligent Slave solution</td>
<td>intelligent Slave solution</td>
<td>Medium Access Unit</td>
<td>Master devices</td>
</tr>
<tr>
<td>Baudrate detection</td>
<td>automatically</td>
<td>automatically</td>
<td>automatically</td>
<td>in Software</td>
<td>-</td>
<td>in Software</td>
</tr>
<tr>
<td>Bus access</td>
<td>in ASIC</td>
<td>in ASIC</td>
<td>in ASIC</td>
<td>-</td>
<td>partly in ASIC</td>
<td>partly in ASIC</td>
</tr>
<tr>
<td>Microprocessor</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>-</td>
<td>yes</td>
</tr>
<tr>
<td>Firmware code size</td>
<td>-</td>
<td>-</td>
<td>6 KBytes</td>
<td>30 KBytes</td>
<td>-</td>
<td>80 KBytes</td>
</tr>
<tr>
<td>RAM</td>
<td>-</td>
<td>-</td>
<td>1,5 KBytes</td>
<td>1,5 KBytes</td>
<td>-</td>
<td>1 Mbyte (extern)</td>
</tr>
<tr>
<td>Voltage</td>
<td>DC 5V</td>
<td>DC 5V</td>
<td>DC 5V</td>
<td>DC 5V, 3.3V</td>
<td>via bus</td>
<td>DC 5V</td>
</tr>
<tr>
<td>Package</td>
<td>MQFP, 80 pin</td>
<td>PQFP, 120 pin</td>
<td>PQFP, 44 pin</td>
<td>TQFP, 44 pin</td>
<td>TQFP, 44 pin</td>
<td>PQ-MQFP, 100 pin</td>
</tr>
</tbody>
</table>

The price of the SPC3 is approximately HFL 35,-/piece with 300 pieces/year.
Chapter 6

6.1 Description of the microcontroller 80C32

The 80C32 is fully compatible with the well-known micro-controller 8051. The 80C32 that is used in the actuator-controller is the Siemens SAB-C501-LN20. The C501 contains a volatile 256 \^{} 8 read/write data memory, four ports, three 16-bit timers counters, a seven source, two priority level interrupt structure and a serial port.

In the next three paragraphs a global description will be given of this microcontroller. The memory-organization is described and the interrupt-model will be explained. For further detailed information, the databooks will describe all details. In this chapter only a few items are discussed.

Fig. 6.1: The 80C32-microcontroller from Siemens, SAB501-LN20
6.1.1 Functional description

![Diagram of microcontroller](image)

Fig. 6.2: Functional description of the microcontroller

Port 0 is used as a multiplexed address/data-bus. With asserting ALE high internally, an address is sent to a extern latch. As ALE goes low, the address is latched and data can be sent. Port 2 is used as the upper 8 bit address-bus. The 80C32 from Siemens has no program-memory on board (Figure 6.2 indicates it, but this figure is a general one for a complete series of compatible microcontrollers). Read- and write-control-lines can be used to fetch data from and to for example external memory. The PSEN is to fetch program-code from for example an eeprom or flash-memory. An USART can perform serial communication with other devices. Also two timers are available to the user for counter/timer-purposes.
6.1.2 The memory-organization of the microcontroller

This microcontroller doesn’t have any program-space internal, but instead all program-space is accessed outside. The memory-model of this controller is showed below:

![Memory Organization Diagram](image)

Fig. 6.3: The memory-organization of the microcontroller

Because the 80C32 doesn’t have any internal code-space, the (not EA)-signal has to be set at a logical ‘0’. As we can see from figure 6.3, the microcontroller has internally some space reserved for SFR’s (Special Function Registers). The registers, except the program counter and the four general purpose register banks, reside in the special function register area. The 27 special function registers (SFRs) include pointers and registers that provide an interface between the CPU and the other on-chip peripherals. All SFRs with addresses where address bits 0-2 are 0 (e.g. 80 H, 88 H, 90 H, 98 H, ..., F8 H, FF H) are bitaddressable.
6.1.3 The interrupt-system of the microcontroller

Fig. 6.4: The interrupt-model of the microcontroller

From figure 6.4, it should be clear that a lot of user-defined interrupt-settings can be set. In total six interrupt-sources are possible and they’re maskable if needed. Also the priority can be set. A high priority is useful for time-critical applications. The interrupts can be level-
sensitive or event-sensitive if wished. All these settings can be prepared by using the special function registers.

6.1.4 Conclusions about the microcontroller

The 8-bit microcontroller 80C32, which is fully compatible with the well-known 8051 from Intel, is well useable for many applications. Almost everything is configurable and the user can always find a perfect solution for the application.
Six interrupt-sources make this microcontroller very usable for controlling-applications, such as actuator-controllers etc.
6.2 Description of the asic SPC3 from SIEMENS

The Siemens asic SPC 3 is capable to handle almost the complete Profibus-DP-protocol. It has the complete slave-state-machine integrated and can work up to a transmission rate of 12Mbit/s. In the following paragraphs the most important features are briefly described to have an overview. For more details, the SPC3-documentation from Siemens is recommended.

6.2.1 General functions of the SPC3

The following processors are supported by the SPC3-interface:

- Intel 8051-based microcontrollers: 80C31, 80C32, 80C52
- Intel microprocessors: 80x86
- Siemens microcontrollers: SAB-C501, 80C165/166/167
- Motorola microcontrollers: HC11-, HC16-, HC916-types

The following is integrated in the SPC3:
- transfer engineering (Layer1) except for analog functions (RS485 driver)
- the FDL transfer protocol (Fieldbus Data Link) for slave stations (Layer2a)
- support of interface services (Layer2b)
- some Layer2 FMA services
- the entire DP slave protocol (USIF: user interface which makes access to Layer2 possible for the user).

The remaining functions of Layer2 (interface services, management) have to be processed with software.

The integrated 1.5k Dual Port RAM is the interface between the SPC3 and the software application. The entire memory is divided into 192 segments of 8 bytes each. Addressing by the user is performed directly, and by the internal micro-sequencer (MS) with the base pointer which can be positioned on a segment in the memory as required. For that reason, all buffers have to be always positioned on the start of a segment.

If the SPC3 is to operate with DP-communication, it will set up all DP-SAPs autonomously. The various message information is made available to the user in separate data buffers (for example, parameter assignment data, configuring data etc.). For data communication, three exchange buffers are made available for the output as well as for the input data. Thus, an exchange buffer is always available to the communication, and there will be no resource problem. To optimally support diagnostics, the SPC3 has two diagnostic exchange buffers where the user enters the current diagnostic data. One diagnostic buffer is always assigned to the SPC3.
The ASIC supports other PROFIBUS Layer 2 functions (distribution data base DDB, free Layer2 FDL), which, however, won't be discussed here. The **Bus Interface** is a parameterizable synchronous/asynchronous 8-Bit interface for various Intel and Motorola micro-controllers/processors. Via the 11Bit address bus, the user can directly access the internal 1.5k RAM or the parameter latches. In the **Parameter Register File** and in the **Mode Registers**, procedure-specific parameters (station address, control bits etc.) are to be transferred by the processor after switch-on. In the **Status Register**, the *Mac state* can be scanned any time. In the **Interrupt Controller**, various events are entered (such as various indications, error events etc.). Via a mask register, these events can be enabled individually. The acknowledgment register provides the confirmation. The SPC3 has a joint interrupt output. The integrated **Watchdog Timer** is operated in three different modes: 'Baud Search', 'Baud_Control' and 'DP_Control'. The **Microsequencer** (MS) controls the entire run. Procedure-specific parameters (buffer pointers, buffer lengths, station address etc.) and the data buffers are contained in the integrated 1.5 Kbytes RAM which a Controller operates as Dual-Port-RAM. In the **UART**, the parallel data current is converted into a serial data current and vice versa. The SPC3 is capable of recognizing the baudrates (9.6 kBd to 12 MBd) automatically. The **Idle Timer** controls the bus timing directly on the serial bus line.
6.2.2 The memory-architecture of the SPC3

<table>
<thead>
<tr>
<th>Address</th>
<th>Function</th>
<th>Internal work cells</th>
</tr>
</thead>
<tbody>
<tr>
<td>000H</td>
<td>Control Unit Parameters</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Latches/Registers</td>
<td>(21 Bytes)</td>
</tr>
<tr>
<td>016H</td>
<td>Organizational Parameters</td>
<td>(42 Bytes)</td>
</tr>
<tr>
<td>040H</td>
<td>DP Buffer:</td>
<td>Data In (3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data Out (3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Diagnosis (2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Parameter Assignment Data (1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Configuring Data (1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auxiliary (2)</td>
</tr>
<tr>
<td>5FFH</td>
<td></td>
<td>SSA Buffer (1)</td>
</tr>
</tbody>
</table>

Fig. 6.5: The allocation of memory in the SPC3

In figure 6.5, we see how the memory is allocated in the SPC3. The first 63 bytes of the memory consists of processor-specific parameters, protocol-specific-parameters and several latches and registers with information about the status of the SPC3 and in which mode it is running. Also the interrupt-registers are located here.

From address 40H and further, the DP-SAP-buffer structure is located. The size of the buffers for data-exchange, diagnostics, configuration and parameters depend on the user-application. However, the maximum sizes (244 bytes each) can be used, because there's enough memory left over (1536 bytes in total).
6.2.3 The processor parameters of the SPC3

The processor parameters consists of the following:

- Interrupt-controller-registers (indications about events)
- Status-registers (for example: about the mode in which the SPC3 is)
- Buffer-parameters (acknowledges of configuration, parameterization and data-fetching)

6.2.4 The organizational parameters of the SPC3

The user can store all organizational parameters in the internal memory of the SPC3. The organizational parameters are:

- Length of the output-data-buffers
- Length of the input-data-buffers
- Length of the diagnostics-buffer
- Length of the parameters_set-buffer
- Length of the set_station_address-buffer

- Also all segment-base-addresses are stored here for the DP-SAP-buffer-structure.
6.2.5 The interrupt-controller of the SPC3

As figure 6.6 shows:

Via the interrupt controller, the processor is notified of indication messages and different error events. Overall, up to 16 events are stored in the interrupt controller which are applied to an interrupt output. The controller doesn't have a prioritization level and doesn't supply an interrupt vector (not compatible with 8259A!).

Every interrupt event which the processor processed has to be cleared via the IAR (except for New_Prm_Data, New_DDB_Prm_Data, New_Cfg_Data). A log '1' is to be written into the corresponding bit position. If a new event and an acknowledge from the previous event are pending on the IRR at the same time, the event remains stored. If the processor subsequently enables a mask, it has to be ensured that there is no entry from the past in the IRR. To make sure, the position is to be cleared in the IRR before the mask is enabled.

Before leaving the interrupt routine, the processor has to set the "End of Interrupt signal (EOI) = 1".
With this edge change, the interrupt line is switched inactive. If an event should still be stored, the interrupt output won't be activated until after an interrupt inactive time of at least 1 usec or 1-2 ms.

This interrupt inactive time can be set via 'EOI_Timebase'. This makes it possible to return to the interrupt routine when using an edge-triggered interrupt input.

The polarity of the interrupt output can be parameterized via the mode-bit INT_POL. After hardware reset, the output is low-active.
6.2.6 The DP-buffer structure

Fig. 6.7: The DP-SAP-buffer structure of the SPC3
Figure 6.7 shows the DP-SAP-buffer structure of the SPC3. All relevant Profibus-DP-SAPs are supported through this structure. The user can directly access all relevant data, configuration, parameters etc.

The SAPs are presented here:

Default-SAP: Data exchange (Write_Read_Data)
SAP53: optional for DDB-parameter assignment message (Set_DDB_Param)
SAP55: Changing the station address (Set_Slave_Address)
SAP56: Reading the inputs (Read_Inputs)
SAP57: Reading the outputs (Read_Outputs)
SAP58: Control commands to the DP-Slave (Global_Control)
SAP59: read configuring data (Get_Config)
SAP60: read diagnostic information data (Slave_Diagnostics)
SAP61: send parameter assignment data (Set_Param)
SAP62: check configuring data (Check_Config)

The DP slave protocol is completely integrated in the SPC3 and is processed autonomously. The user has to parameterize the ASIC correspondingly, and to process and acknowledge transferred messages. All SAPs, except for the default SAP, SAP56, SAP57 and SAP58 are always enabled.

The remaining four SAPs aren't enabled until the DP slave machine (DP_SM) enters the 'DATA_EX'-mode. The user can disable the SAP55. For this, the corresponding buffer pointer R_SSA_Buf_Ptr is to be set to '00H'. The DDB service is to be disabled by initializing the RAM cells already described.

The user configures all buffers (length and buffer start) in the 'off-line mode'. The buffer configuration is not to be changed during operation except for the length of the Dout-/Din buffers.

The user is permitted to adjust them in the 'Wait_Cfg' mode according to the configuring message (Check_Config).

**In the 'DATA_EX' mode, only the same configuration is to be accepted.**

The buffer structure is divided into data-, diagnostic- and control buffer.

For the output- and input data, three buffers respectively of the same length are available; they process as exchange buffers. One buffer respectively is assigned to the data transfer 'D' and the user 'U'. The third buffer is either in a next 'N' mode or a free 'F' mode; one of the two modes is always unassigned.

For diagnosis, two diagnostic buffers are available. They may have different lengths. One
diagnostic buffer is always assigned to the SPC3 for transmitting 'D', and the other belongs to the user for preparing new diagnostic data 'U'.

The different parameter assignment messages (Set_Slave_Address, Set_Param) and the configuration message (Check_Config) are moved to the Aux-Buffer1 or Aux-Buffer2 by the SPC3. After faultless reception or after verifying the SPC3-specific data, it is exchanged with the corresponding destination buffer (SSA-, Prm-, Cfg buffer).

In that case, the buffers to be exchanged have to be the same length. In the parameter cell 'R_Aux_Buf_Sel', the user parameterizes which Aux_buffers are used for the messages mentioned above, whereby the Aux-buffer1 always has to be available. Aux-buffer2 is optional. If the data profiles of the DP messages are very different -for example, the data volume in the Set_Param message is considerably larger than for the other messages- it is recommended making an Aux-buffer2 available for this message (Aux_Sel_Set_Param = 1). The other messages are then handled via Aux-buffer1 (Aux_Sel... = 0). If the buffer is too small, the SPC3 responds with "no resources"!

For reading the configuration data (Get_Config): it is made available by the user in the Read_Cfg buffer. The Read_Cfg buffer has to have the same length as the Cfg buffer.

The Read_Input_Data message is serviced from the Din buffer in the 'D mode', and the Read_Output_Data message is serviced from the Dout buffer in the 'U mode'.

All buffer pointers are 8 bit segment addresses, because the SPC3 internally has only 8 bit address registers. When accessing the RAM, the SPC3 adds an 8 bit offset address to the segment address which has been shifted by 3 bits (Result: 11 bit physical address). This input results in an 8 byte granularity regarding buffer start addresses.
6.2.7 The bus-interfacing with the microcontroller 80C32

The bus-interface depends on the micro-processor used. If the 80C32 is used, then the interface as discussed in chapter 4.3.6 is valid. The lowest 11 address-bits are used to address the internal memory of the SPC3. The MSB of the address-lines is used to select the SPC3, when needed. Because the microcontroller receives the clock-signal of the SPC3, all timing is synchronized.

6.2.8 Conclusions about the SPC3

The SPC3 from Siemens is well suited for sophisticated DP-applications. Almost everything is changeable and can be made user-specific. The supported transmission rate of up to 12 Mbit/s makes the SPC3 useable for all nowadays Profibus-DP-slaves-applications. Because the SPC3 has almost the complete Profibus-DP-protocol integrated, it relieves microcontrollers from communications-overhead and the microcontroller can be dedicated to run the application-specific at the speed possible.
6.3 Description of the power-driver TLE-5216G from SIEMENS

These drivers are able to drive 4 actuators in parallel mode and to diagnose 4 actuators in serial mode. Actuators are solenoid devices like relays, valves and lamps. In the next paragraphs some brief explanations are done about this device.

6.3.1 Functional block of the powerdriver

Fig. 6.8: Functional description of the powerdriver TLE5216G
If we look at figure 6.8, we can see the logical block of the powerdriver TLE5216G. As shown, 4 input-lines can be driven parallel to drive 4 outputs in parallel. The device can perform some diagnostics like overload, open load and short circuits. The device is a low-side switch, which means that the actuator is pulled to ground when switched. A actuator will be activated by a logical ‘0’ on the referring input. The clamp diode will limit the output voltage when the actuator is shut off. The reset-line is active LOW. When low, then it shuts down all outputs and resets the error flags (when going from low to high again).

6.3.2 The coding of the diagnostic signals

![Diagram]

Fig. 6.9: The coding of the signals and events in the TLE5216G

As we can see from figure 6.9, the serout-line will go low if an error occurs at the output-stage. The serout will go down while the chip-select (CS) is still high. This indicates an error. It must be noted that a new error on the same output-stage will overwrite the old error-report. As soon as the CS is asserted low by the user, the error-report will be sent over the serout-line. All diagnostic data is transferred sequentially per output-stage and is controlled by the
serial clock-input. If another powerdriver is cascaded via the serin-line, then the data of that powerdriver will follow after the current powerdriver in the same serial way. The error-codes can be read from figure 6.10.

Fig. 6.10: The definition of the error-code in the diagnostic mode

Because the user knows how many powerdrivers are cascaded with each other, he will know when the CS can be asserted high again. By asserting the CS high, all error-reports will be reset. If an overload persists too long, then the output will be switched off. The output can only switched on again by driving the corresponding input-line on and off again.

6.3.3 Conclusions about the powerdriver

The powerdriver TLE5216G is ideal for driving relays and other solenoid actuators. By embedded protection-mechanisms and diagnoses-options, this device is very useful for printboards that are used in the field (i.e.: no need to disconnect or repair a device, when errors occur). The protocol that is used for diagnostics is also known as the SPI-protocol (Serial Peripheral Interface), which makes it easy installable.
6.4 Description of the flash-memory AM29F010 from AMD

This memory can be reprogrammed with the same voltage-level as the supply-voltage. This is a great advantage, because no additional hardware is needed to reprogram the memory. The reprogramming can be done in software by entering a so-called command-mode in the flash-memory. This command-mode can be entered by writing the correct command-bytes to the flash-memory. With this command-mode embedded algorithms are started. These embedded algorithms do the actual programming of a byte at the right address. Embedded algorithms are also used to verify whether a byte was written correct or not. In the next paragraphs we'll discuss some aspects of this particular flash-memory from AMD.

Fig. 6.11: Logical block of the flash-memory

First we'll take a look at picture 6.11. We can see the common address- and data-lines. Also the chip-select (not CE), write-enable (not WE) and read-enable (not OE) are available just like a standard memory. Reading a byte from a specific address goes straight-on, but writing bytes to the device is done by first writing some special bytes. These special bytes activate build-in algorithms. These special bytes are called a command-sequence.
6.4.1 Sector-architecture of the flash-memory

The flash-memory is divided into 8 equal parts, called sectors. Each sector is 16Kbytes big as the figure shows below.

<table>
<thead>
<tr>
<th>Sector (SA)</th>
<th>Memory Size</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA0</td>
<td>16 KByte</td>
<td>00000h</td>
</tr>
<tr>
<td>SA1</td>
<td>16 KByte</td>
<td>03FFFh</td>
</tr>
<tr>
<td>SA2</td>
<td>16 KByte</td>
<td>07FFFFh</td>
</tr>
<tr>
<td>SA3</td>
<td>16 KByte</td>
<td>0BFFFFh</td>
</tr>
<tr>
<td>SA4</td>
<td>16 KByte</td>
<td>13FFFFh</td>
</tr>
<tr>
<td>SA5</td>
<td>16 KByte</td>
<td>17FFFFh</td>
</tr>
<tr>
<td>SA6</td>
<td>16 KByte</td>
<td>1BFFFFh</td>
</tr>
<tr>
<td>SA7</td>
<td>16 KByte</td>
<td>1FFFFFh</td>
</tr>
</tbody>
</table>

Fig. 6.12: Sector-organization of the flash-memory

All sectors can be erased together in one sequence, or they can be erased separately. If a sector is protected, then it can not be erased.
### 6.4.2 Command-definitions

<table>
<thead>
<tr>
<th>Command Sequence</th>
<th>Bus Write Cycles Req'd</th>
<th>First Bus Write Cycle</th>
<th>Second Bus Write Cycle</th>
<th>Third Bus Write Cycle</th>
<th>Fourth Bus Read/Write Cycle</th>
<th>Fifth Bus Write Cycle</th>
<th>Sixth Bus Write Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Addr</td>
<td>Data</td>
<td>Addr</td>
<td>Data</td>
<td>Addr</td>
<td>Data</td>
</tr>
<tr>
<td>Reset/Read</td>
<td>3</td>
<td>5555H</td>
<td>AAH</td>
<td>2AAAH</td>
<td>55H</td>
<td>5555H</td>
<td>F0H</td>
</tr>
<tr>
<td>Autoslect</td>
<td>3</td>
<td>5555H</td>
<td>AAH</td>
<td>2AAAH</td>
<td>55H</td>
<td>5555H</td>
<td>90H</td>
</tr>
<tr>
<td>Byte Program</td>
<td>4</td>
<td>5555H</td>
<td>AAH</td>
<td>2AAAH</td>
<td>55H</td>
<td>5555H</td>
<td>A0H</td>
</tr>
<tr>
<td>Chip Erase</td>
<td>6</td>
<td>5555H</td>
<td>AAH</td>
<td>2AAAH</td>
<td>55H</td>
<td>5555H</td>
<td>80H</td>
</tr>
<tr>
<td>Sector Erase</td>
<td>6</td>
<td>5555H</td>
<td>AAH</td>
<td>2AAAH</td>
<td>55H</td>
<td>5555H</td>
<td>80H</td>
</tr>
</tbody>
</table>

Fig. 6.13: The command-definitions and sequences for the flash-memory
In figure 6.13, we can see how many bytes are needed for each command-sequence before a specific algorithm is started. The most important sequences are the sector-erase- and byte-programming-sequences. The algorithms needed for these two features are discussed in the next paragraphs.
6.4.3 Embedded programming algorithm

The algorithm as shown in figure 6.14, must be followed by programming software to achieve successful flash-programming. In figure 6.15, the command-sequence is shown. After the byte is programmed by the embedded algorithm, a verify must be done. This done by the data-polling algorithm. If all bytes are programmed, the update is done. It should be noted that only logical ‘0’s can be written. If a logical ‘1’ is needed, then the specific sector has to be erased first.

---

**Fig. 6.14: The programming-algorithm of the flash-memory**

**Fig. 6.15: The needed command-sequence for byte-programming**
6.4.4 Embedded erasing algorithm

Two erase-modes exist: a single/multiple sector-erase (stated at the right of figure 6.16) and a complete chip-erase (at the left of the same picture). The data-polling is used here also.

Fig. 6.16: The erase-algorithms together with the command-sequences
6.4.5 Embedded data-verify/bit-toggle algorithm

![Flowchart of the verify-data algorithm]

The data-verify-algorithm is used after the programming of a byte or a sector-erase. If in the end the most significant bit DQ7 from the data-byte that was programmed is not equal to the data that was given, then the byte-programming has failed. Another try can be done by starting the command-sequence again. If, however, the failure persist, then the flash-memory has become useless.
Fig. 6.18: The toggle-bit verifying algorithm

The toggle-bit-algorithm is another way to verify whether the data-byte or sector was written or resp. erased correct. The same story can be told here; if failure persist, then the flash-memory has become useless.

6.4.6 Conclusions about the Flash-memory

The flash-memory AM29F010 has been discussed and the most important command-sequences are explained (i.e.: program byte and erase a sector)
Chapter 7

7.1 Tests performed with a PC-master (Siemens CP 5412-A2) and a multiple input-output-slave (Siemens ET200L)

The Siemens ET200L is an I/O-module capable of handling 16 inputs and 16 outputs. This module has the asic LSPM2 onboard. This asic is a device that has almost everything in it to perform simple data-exchange and to perform standard diagnostics.

Together with the PC-master CP5412-A2 configuration- and parameterization-software is delivered to work under Windows NT. Also a demo-program is delivered to perform simple data-exchange and diagnostics. The system is set up with the SINEC software-package from Siemens. This package consist of the three following modules:

1. DBASE-setup, cp-installing-tool for the master CP5412-A2 for Windows NT, V1.02
2. COMIL-DP, Configuration-software, V1.01
3. DP-CP5412A2, Demo-program for data-exchange, V2.0

Together with this software and the hardware simple data-exchange and diagnosis was done successfully. It was easy to drive the 16 outputs separately and to read the 16 inputs.
Design of an intelligent actuator-controller using Profibus-DP

separately. With this software, it should be possible to test the actuator-controller by sending two output-bytes (two bytes contain the information for 16 actuators).

### 7.2 Connecting the designed actuator-controller-board in a test-system

For prototyping purposes the actuator-controller has to be tested in a simple test-system. This test-system consists of one master and at least one slave. Because Ellips B.V. already has a slave-product from Siemens (i.e.: the ET 200L), at least two slaves can be used in the test-system. This is useful to learn about the aspects of addressing several slaves in one system. Because Ellips B.V. has no already designed PC-master, the also bought PC-master CP 5412A2 from Siemens is used as class 1 and class 2 master. Some steps are given now that will lead to a successful product.

![Diagram of test-system](image)

Fig. 7.2: Connection of four stations (one master and three slaves) in a test-system

As we can see in figure 7.2, one master is used, the CP5412A2. This master has both master-classes in one. For debugging-purposes the actuator-controller-board has a target-debug-connector connected to the UART of the 80C32. This makes it possible to debug the software while it’s running. Also adapters are used for the flash-memory (i.e.: the flash-memory is not soldered on the board, but can be removed for re-programming to remove software-bugs).
Also a separate LED is connected to port 3.5 of the 80C32 to test whether the software is working. If the software has grown to an end-product, indications can be given by LEDs in front of the powerdrivers. This is done to test whether the powerdrivers are driven well or not. To assure that the communications-software with the ASIC SPC3 is working correctly, it’s recommended to build a very stripped version of the actuator-controller-software first (i.e.: no possibility to do a flash-update, only one configuration and just driving one LED at the test-port). From that point the total software can be build up. By testing the software step by step, the total end-product will be stable.
7.3 Connecting the designed actuator-controller-board in a real system

As soon as the designed actuator-controller is ready to be connected in a real system, some considerations have to be made to achieve success in a short time. First of all, all slaves start with a default-address of 126. For this reason, slaves have to be commissioned to the system one by one. This commissioning is done by a master of class 2. If a slave is approved by the master it will get a unique, till then, unused station-address. Also if a slave has to be disconnected from the bus, for maintenance or removement, the station-address has to be reset to the default-address 126 to avoid problems with double addresses in one bus at reconnection.

In the real fruit-grading-system, a pc-master designed by Ellips B.V. will be used.

Fig. 7.3: The connections between stations in the fruit-grading-system

If more then 31 devices are needed on one bus, repeaters should be used. The direction must be indicated by the RTS-signals from the devices separately.
Chapter 8

8.1 Conclusions

The fruit-grading-machine is described as it is now and how it will look like in the near future. The Profibus-DP-protocol is implemented in the actuator-controller by the asic SPC3 and source-code. Therefore the Profibus-DP-protocol was analyzed and a description was made of it to understand the overall performances. Hardware-design of the actuator-controller is made and released. The physical layout and hardware will be released in the middle of June 1997. All components were chosen depending on price, availability and second sources. The software-functions that are needed by the actuator-controller are described and a beginning is made with the software-programming of this controller. First of all the SPC3_DPS2-software from Siemens is studied. This firmware handles all communication with the Profibus-asic SPC3 and the user. Also macros are defined to ease the calculation of the buffer-structure of the SPC3. For first test-purposes a PC-master-station from Siemens was used, the CP 5412 A2. This product is a PC-controlled master-board capable to perform master class 1 and 2 functions. In a simple Profibus-DP-system this card was used together with a simple slave-device from Siemens to learn about the Profibus-aspects.

8.2 Recommendations

The design of the sensor/encoder-controller is almost fully similar to the design of the actuator-controller. This board should also have flash-memory as non-volatile memory for updating-purposes. In fact all slave-devices to be designed have the same design concerning the Profibus-interface. This interface is represented by the asic SPC3 from Siemens together with a 80C32-microcontroller. To assure that the products will be compatible with Profibus-DP, special care has to be taken by writing the software for the master and slaves. Therefore the Profibus-DP standard definitions should always be used.
Design of an intelligent actuator-controller using Profibus-DP

Chapter 9

9.1 List of Literature


[3] 80C51 based 8-bit microcontrollers, Philips datahandbook IC20, March 1995, Philips Electronics, USA

[4] SPC3 Siemens PROFIBUS Controller, Siemens SINEC L2, userguide, version 1.4, October 1996, Siemens AG Order no.: 6ES7-195-0BD00-8AA0


9.2 List of Internet-addresses

AMD: http://www.amd.com/ info on flash-memories
DigiKey: http://www.digikey.com/ info on semiconductors
Linear Technology: http://www.linear-tech.com/ info on powerproducts
Marshall: http://www.marshall.com/ info on semiconductors
Maxim: http://www.maxim-ic.com/ info on watch-dog-circuits
Motorola: http://www.mot.com/ info on general semiconductors
Profibus homepage: http://www.profibus.com info on Profibus
Samsung: http://www.sec.samsung.com/index.html info on sram’s
Siemens automation: http://www.aut.siemens.de info on Profibus-products
Siemens semi-conductors: http://www.sci.siemens.com/ info on semiconductors
UMC: http://www.umc.com.tw/ info on sram’s
### 9.3 Phone-List of Electronic Components Distributors

<table>
<thead>
<tr>
<th>Distributor</th>
<th>Phone No.</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBV</td>
<td>++31 346-583010</td>
<td>The Netherlands</td>
</tr>
<tr>
<td>Eurodis/Texim</td>
<td>++31 53-5733333</td>
<td>The Netherlands</td>
</tr>
<tr>
<td>Farnell</td>
<td>++31 30-2412323</td>
<td>The Netherlands</td>
</tr>
<tr>
<td>Nijkerk</td>
<td>++31 20-5041424</td>
<td>The Netherlands</td>
</tr>
<tr>
<td>Sonetech</td>
<td>++31 76-5722333</td>
<td>The Netherlands</td>
</tr>
<tr>
<td>Spoerle</td>
<td>++31 30-6091217</td>
<td>The Netherlands</td>
</tr>
<tr>
<td>Tritec Benelux B.V.</td>
<td>++31 78-6816133</td>
<td>The Netherlands</td>
</tr>
</tbody>
</table>
## Appendices

### A List of abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASIC</td>
<td>Application Specific Integrated Circuit</td>
</tr>
<tr>
<td>FDL</td>
<td>Fieldbus Data Link</td>
</tr>
<tr>
<td>FMA</td>
<td>Fieldbus Management</td>
</tr>
<tr>
<td>ISO</td>
<td>International Standards Organization</td>
</tr>
<tr>
<td>LSPM2</td>
<td>Lean Siemens Profibus Multiplexer version 2</td>
</tr>
<tr>
<td>μC</td>
<td>Microcontroller</td>
</tr>
<tr>
<td>Osc.</td>
<td>Oscillator</td>
</tr>
<tr>
<td>OSI</td>
<td>Open Systems Interconnection</td>
</tr>
<tr>
<td>PNO</td>
<td>Profibus Nutzer Organisation</td>
</tr>
<tr>
<td>Profibus</td>
<td>Process Field-BUS</td>
</tr>
<tr>
<td>Profibus-DP</td>
<td>Profibus-Decentralized Periphery</td>
</tr>
<tr>
<td>Profibus-FMS</td>
<td>Profibus-Fieldbus Message Specification</td>
</tr>
<tr>
<td>Profibus-PA</td>
<td>Profibus-Process Automation</td>
</tr>
<tr>
<td>SPC3</td>
<td>Siemens Profibus Controller generation 3</td>
</tr>
<tr>
<td>SPI</td>
<td>Serial Peripheral Interface</td>
</tr>
<tr>
<td>SAP</td>
<td>Service Access Point</td>
</tr>
<tr>
<td>WD</td>
<td>Watch-dog</td>
</tr>
</tbody>
</table>
### B VHDL-code of the glue-logic

--- Profibus solenoid card glue logic
---
--- Revision history
--- 2-4-'97 first version based on addressdecoder and controldecoder

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity memdec is
    port(
        rst_in : in std_logic;
        ale: in std_logic;
        psen, rdn, wrn : in std_logic;
        addr_in_15_13 : in std_logic_vector(15 downto 13);
        ad_3_0 : in std_logic_vector(3 downto 0);
        rd_outn: out std_logic;
        addr_out_16_14: out std_logic_vector(16 downto 14);
        flash_csn: out std_logic;
        ram_csn: out std_logic;
        spc3_csn: out std_logic;
        relay0 : out std_logic_vector(3 downto 0);
        relay1 : out std_logic_vector(3 downto 0);
        relay2 : out std_logic_vector(3 downto 0);
        relay3 : out std_logic_vector(3 downto 0)
    );
end memdec;

architecture memdec_arch of memdec is
```

-100-

type mem4_type is array (natural range <> of std_logic_vector (3 downto 0));

subtype mmu_width_type is integer range 3 downto 0;
subtype mmu_data_type is std_logic_vector(mmu_width_type);
type mmu_type is array (natural range <> of mmu_data_type);

constant spc3_base: std_logic_vector (15 downto 0):= X"E000";
constant glue_base: std_logic_vector (15 downto 0) := X"C000";

-- local signals
signal addr_3_0 : std_logic_vector(3 downto 0);
signal lrelay : mem4_type(3 downto 0);

signal pmmu : mmu_type(2 downto 0);
signal dmmu : mmu_type(2 downto 0);

begin

-- addr latch
addr_latch_proc: process (ale, ad_3_0)
begin
  if ale = '1' then
    addr_3_0 <= ad_3_0;
  end if;
end process;

-- rst_out inverted version of rst_in
rst_outn <= not rst_in;

-- assert rd_out as either psen or rd_in is asserted
rd_outn <= not (not psen or not rdn);

-- relay registers
process (wrn, rst_in)
begin
  if rst_in = '1' then
    lrelay <= (others => (others => 'I'));
  elsif (wrn'event and wrn = '1') then
    if (addr_in_15_13 = glue_base(15 downto 13)) and (addr_3_0(3) = '0') then

-101-
Design of an intelligent actuator-controller using Profibus-DP

```
process (psen, addr_in_15_13, pmmu, dmmu)
variable mmu_out: mmu_data_type;
begin
if addr_in_15_13(15 downto 14) = "11" then -- top 16K no MMU
addr_out_16_14 <= '0' & addr_in_15_13(15 downto 14);
end if;
end process;
```

```
lrelay(conv_integer(addr_3_0(1 downto 0))) <= ad_3_0;
end if;
end if;
end process;
```

```
relay0 <= lrelay(0);
relay1 <= lrelay(1);
relay2 <= lrelay(2);
relay3 <= lrelay(3);
```

```
-- pmmu registers
process (wm, rst_in)
begin
if rst_in = '1' then
pmmu <= (others => (others => '0'));
elsif (wm'event and wm = 'I') then
if (addr_in_15_13 = glue_base(15 downto 13)) and (addr_3_0(3 downto 2) = "10") then
pmmu(conv_integer(addr_3_0(1 downto 0))) <= ad_3_0(mmu_width_type'high downto 0);
end if;
end if;
end process;
```

```
-- dmmu registers
process (wm, rst_in)
begin
if rst_in = '1' then
dmmu <= (others => (others => '0'));
elsif (wm'event and wm = 'I') then
if (addr_in_15_13 = glue_base(15 downto 13)) and (addr_3_0(3 downto 2) = "11") then
dmmu(conv_integer(addr_3_0(1 downto 0))) <= ad_3_0(mmu_width_type'high downto 0);
end if;
end if;
end process;
```
Design of an intelligent actuator-controller using Profibus-DP

```vhdl
flash_csn <= '1';
ram_csn <= '1';
else
  if psen = '0' then
    mmu_out := pmmu(conv_integer(addr_in_15_13(15 downto 14)));
  else -- default to data memory
    mmu_out := dmmu(conv_integer(addr_in_15_13(15 downto 14)));
  end if;
  addr_out_16_14 <= mmu_out(mmu_width_type'high - 1 downto 0);
  flash_csn <= mmu_out(mmu_width_type'high);
  ram_csn <= not mmu_out(mmu_width_type'high);
end if;
end process;

spc3_csn <= '0' when addr_in_15_13 = spc3_base(15 downto 13) else '1';

end memdec_arch;
```