Verslag van het afstudeerwerk
uitgevoerd van maart tot en met december 1984
bij het Natuurkundig Laboratorium van Philips.
Afstudeerhoogleraar: prof. ir. J. van der Plaats
Begeleiders: ir. A.H. Dieleman (Philips Nat.Lab.)
Dr. ir. W. van Etten

De afdeling der electrotechniek van de Technische Hogeschool Eindhoven aanvaardt geen verantwoordelijkheid voor de inhoud van stage- en afstudeerverslagen.
"A promising area of application for glass fibres seems to be the area of LAN's, local area networks: data communication over relatively short distances between computers, word processors, peripheral devices, measuring apparatus etc.

In these data communication networks glass fibres could be extremely well suited for transmission over distances longer than a few metres, while on the other hand connections over smaller distances can still be made with copper wires for the time being.

This means that the use of glass fibres should be fitted in existing interconnection systems like RS 232C, IEC-625, Ethernet and many others".

(quotation of a part of the graduation task description)
## CONTENTS

<table>
<thead>
<tr>
<th>Outline</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contents</td>
<td>1</td>
</tr>
<tr>
<td>Introduction</td>
<td>2</td>
</tr>
<tr>
<td>Chapter 1 General</td>
<td>4</td>
</tr>
<tr>
<td>1.1 Possible consequences of extending interconnection systems</td>
<td>5</td>
</tr>
<tr>
<td>Chapter 2 IEC-bus Interface</td>
<td>8</td>
</tr>
<tr>
<td>2.1 General description of the IEC-bus interface</td>
<td>8</td>
</tr>
<tr>
<td>2.2 Function description of an IEC-bus fode</td>
<td>12</td>
</tr>
<tr>
<td>Chapter 3 Lines of sight for an IEC-bus fode with respect to byte transport and the uniline message ATN</td>
<td>19</td>
</tr>
<tr>
<td>3.1 Data mode</td>
<td>19</td>
</tr>
<tr>
<td>3.2 Transition of the data mode to the command mode, and the command mode</td>
<td>22</td>
</tr>
<tr>
<td>Chapter 4 Effects of the extension on the general management unilines and on the polling procedures</td>
<td>29</td>
</tr>
<tr>
<td>4.1 Influence of the extension on the uniline messages</td>
<td>29</td>
</tr>
<tr>
<td>4.2 Influence of the extension on serial and parallel polling procedures</td>
<td>31</td>
</tr>
<tr>
<td>Chapter 5 Design of an IEC-bus fode</td>
<td>32</td>
</tr>
<tr>
<td>5.1 Datapath from the bus to the glass fibre and from the receiving part of the fode to the bus</td>
<td>35</td>
</tr>
<tr>
<td>5.2 Determination of the talker side and the controller side, and managing the transceivers</td>
<td>37</td>
</tr>
<tr>
<td>5.3 The acceptor handshake part</td>
<td>40</td>
</tr>
<tr>
<td>5.4 Frame counter of the transmitting part of the fode</td>
<td>44</td>
</tr>
<tr>
<td>5.5 Determination of the time order of EOI, IFC and ATN in case of a fode connected with the controller by bus</td>
<td>48</td>
</tr>
<tr>
<td>5.6 Emptying the FIFO-memory using the parallel polling method</td>
<td>52</td>
</tr>
<tr>
<td>5.7 Frame counter of the receiving part of the fode</td>
<td>54</td>
</tr>
<tr>
<td>5.8 Frame receiving part of the fode</td>
<td>56</td>
</tr>
<tr>
<td>5.9 Determination of the time order of EOI, IFC and ATN in case of a fode connected to the controller by the extension</td>
<td>58</td>
</tr>
<tr>
<td>5.10 Writing data bytes in the FIFO-memory and reading them out</td>
<td>61</td>
</tr>
<tr>
<td>Source handshake part of the fode</td>
<td>66</td>
</tr>
<tr>
<td>5.11 FIFO-memory counter</td>
<td>66</td>
</tr>
</tbody>
</table>
INTRODUCTION

In the beginning of this graduation work several interconnection systems between computers and peripheral devices have been investigated. The intention was to find out which of the interconnection systems would be suited to be implemented on, or extended with glass fibres. Attention was also paid to the question whether implementation or extension would prove worthwhile.

Study of the interconnection systems revealed that two systems are very often used. These two systems are the instrumental bus interface standard IEC-625 or IEEE 488 and the interface standard V-24, which is (practically) identical to the RS 232C interface. A third system, the bisync protocol standard, was studied as well, because it was regarded as a possible interesting item for the Philips Research Laboratories. The items V-24 and bisync protocol will be dealt with in the appendix 1.

The instrumental bus interface IEC-625, further on referred to as IEC-bus, proved to be very interesting and it has therefore been studied profoundly. (See chapters 1 to 4)

The study of the IEC-bus has resulted in the realisation on paper of a device capable of extending the limited length of 20 metres of a normal IEC-bus to 5 kilometres. (See chapter 5) This extending device will be called a FODE, which is the acronym for Fibre Optical Data Link Extension. The problems, encountered while creating this device and which spring from the very strict defined IEC-bus protocol rules, together with their solutions will be discussed in full in this report. It will be explained that the maximum throughput rate of 1 Mbyte/s is feasible and that there are very few restrictions to normal IEC-procedures.
Chapter 1 General

After having studied several interconnection systems between computer and peripheral devices to find out if any of them could be implemented on or extended with glass fibres, two systems turned out to be sufficiently interesting. These two systems are the interface standard V-24 [2],[3],[8],[9], which is (practically) identical to the RS 232C interface, and the IEC-bus interface standard IEC-625, which we will refer to as IEC-bus [1],[4],[5].

RS 232C is often used to connect a computer to a peripheral device in a direct way, and the IEC bus is a so-called instrumental bus. Apart from these two obvious systems a third system has been examined as well. This third system is the bisync protocol, which is a half duplex protocol on coaxial cable [2],[3],[6]. This protocol is used to maintain the communication between the IBM computer and its terminals at the Philips Research Laboratories.

Note: It seems only appropriate to remark that problems can arise when we want to compare the three mentioned systems. The IEC-bus interface and the V-24 interface can only be compared when they are looked at as being interconnection systems, as is done in this report. The bisync protocol, however, doesn't fit in this description because it cannot be considered as a concrete connection system. On the other hand when we consider the connection medium (coaxial cable) and the protocol as a unity, we can see this combination as a real interconnection system and hence comparable with the other two systems.

Before we will focus entirely on the three systems mentioned earlier, it seems a good idea to roughly examine the possible consequences of extending a connection.

1.1 Possible consequences of extending interconnection systems figure 1.1

Consider two devices A and B interconnected in a for them traditional way. See figure 1.1a. Data exchange between devices is often done with the help of a so-called handshake. This handshake consists of signals (messages) which have to ensure a correct data transport. Another often encountered matter is that in some systems the connected devices have to react on certain signals or commands within a limited time interval.
Let us assume that the connection is opened and that it is extended with glass fibre as is shown in figure 1.1b.

Of course we need a device capable of extending the connection using glass fibres. In this report we will call such an extending device a FODE, which is the acronym for Fibre Optical Data-link Extension.

If we try to extend the connection without taking any precautions, there will be a fair chance that the data transport will be influenced, especially when we are dealing with relatively long extension lengths.

Let us look at an example.

Suppose we are dealing with an extended connection with a total length of 5 kilometres.

We will assume parallel data transport and furthermore that a data byte is accompanied by a simple handshake. The procedure will be as follows: upon reception of a data byte a data receiving device will report this to a data transmitting device.

The speed of light in the glass fibre is approximately 200000 km/s, so a data byte will need 25 microseconds to go from A to B.

The message that confirms the reception of the data byte by the data receiving device will also need 25 microseconds.

This brings the total time needed to transport one single data byte to 50 microseconds, which makes it obvious that no higher throughput rates can be expected than 20 kbytes/s in this case.

Likewise it is understandable that because of this transmission delay time it often is not possible for extended devices to respond to certain signals or commands in the required time intervals.

The conclusion we can draw from this example is, that we cannot extend most of the connections without taking necessary precautions.

To find the correct precautions it is necessary to study the protocol of the original, not extended connection.

This is essential as the protocol describes the exact process of data transport.

It tells us what really happens between devices in a certain system.
Figure 1.1 General Situation
\[ F = \text{FODE} = \]

fibre optical data link extension

\[ gf = \text{glassfibre} \]
Chapter 2 IEC-bus interface

In this chapter and the next ones we will direct our entire attention to the IEC-bus interface.

2.1 General description of the IEC-bus interface
figures 2.1 , 2.2

This section is meant to refresh the memory of the reader and it gives no detailed description of the IEC-bus interface. For more detailed information it is best to see [1], [4], [5]. This interface is known as IEC-625, IEEE 488, GPIB and HP-IB. The complete IEC-bus system consists of a controller and a collection of devices capable to transmit data and devices capable to receive data. In the IEC-bus parlance the data transmitting devices are called talkers and the data receiving devices are called listeners. See figure 2.1.

When the controller is in an active state (the so-called command mode) all the other devices can do is listen to the commands given by that controller.

When the controller is not active (the data mode) only one talker can send its data to several listeners. The talk and listen functions of the devices are activated by the controller commands.

Note: A talker (only one at a time) can send its data to one or more listeners.

It is not essential how many listeners are connected to the talker.

Therefore we will use the word listener both in the singular and in the plural form indifferently.

Normally the devices are interconnected by an IEC-bus, and there is a maximum of 15 devices that can be connected to the bus.

The IEC-bus has a total of 16 parallel lines for data transmission, handshake signals and general interface management signals. The data bus is formed by 8 lines: DIO 1-8 (Data Input Output). For the "data byte transfer control" three handshake signals are available: DAV (Data Valid), NRFD (Not Ready For Data) and NDAC (Not Data Accepted).

The other 5 lines are used for general management of the interface. These lines are: ATN (Attention), IFC (Interface Clear), SRQ (Service Request), REN (Remote Enable) and EOI (End Or Identify).

The messages sent over the 'handshake' lines and over the lines of 'general management of the interface' will be called uniline messages, because each of the messages is sent over a single line. This is in contrast with the so-called multiline messages that are sent via the 8 data lines by the controller only.
The meaning of the last mentioned interface management signals is the following.
When ATN is true the controller makes clear to the other devices that it wants to practice its function.
SRQ true means that a device has something to report to the controller. The contents of this message is asked later during a polling.
The functions REN and IFC speak for themselves.
The last function is that of EOI.
EOI = true can have two different meanings. When EOI = true in combination with ATN = true this means that the controller makes it clear to all connected devices that it is about to start a parallel polling.
On the other hand EOI = true and ATN = false is the sign given by the current talker to indicate that it transmits its last data byte.
To complete this little inventory it must be mentioned that the IEC-bus system uses a complete hardware handshake. Figure 2.2 shows this handshake.
Fig 2.1

Controller

Listener

Talker

Listener

Talker / Listener

DIO 1
DIO 2
DIO 3
DIO 4
DIO 5
DIO 6
DIO 7
DIO 8
DAV
NRFD
NDAC
ATN
IFC
SRQ
REN
EOI
Figure 2.1 General IEC-configuration
Figure 2.2 IEC-handshake procedure
FIG 22

HANDSHAKE PROCEDURE IEC BUS
2.2 Functional description of an IEC-bus fode
figures 2.3, 2.4, 2.5

As mentioned before, it is often desirable to have a total connection length of approximately 5 kilometres to our disposal instead of the normal 20 metres in case of not extended IEC-bus systems. The IEC-bus protocol, however, doesn't take into account transmission delay times longer than those of 20 metre connections. When we want to extend the IEC-bus, we can think of two ways of doing so.
The first way of extending restricts us to a device extension as shown in figure 2.3.
The second way is extending the bus. Figure 2.4 shows this extension method.
Note: In the figures 2.3 and 2.4 two glass fibres are shown. One glass fibre will be used for sending information to the other fode, the other glass fibre's function is to enable a fode to receive the information sent by that other fode. (See appendix A4 for possible other solutions)

After having studied the IEC-bus protocol it was soon clear that a bus extension would be possible. Happily the IEC-bus protocol is very well known and surrounded by well defined rules.
Incompatibility of two devices although they both meet the standard, like in the case of the V-24 interface standard (Appendix 1), is not possible in case of the IEC bus.
Violating the standard rules by redefining functions is also not possible unless the user is willing to abandon the entire IEC-bus concept.
Using the bus extension as a starting point, let us look at some criteria for the design of an IEC-bus fode.
Note: this section is only meant to be a general survey of the way a fode must function.
A lot of aspects which will be mentioned here, will be explained and discussed in following sections and chapters.
We will refer to these explaining sections in this text.

From the beginning it has been the intention to design an IEC-bus fode as transparant as possible with respect to the IEC-bus protocol. This means that an existing IEC-bus system should not notice the presence of a fode; the system should not have to reserve dedicated lines or addresses for the fode.
Besides meeting the demands of the IEC-bus protocol it was decided upon to meet as many of the not dictated specifications as possible.
One of those not dictated specifications is the maximum throughput rate. The IEC-bus protocol allows a maximum throughput rate of 1 Mbyte/s. See appendix 3 for the relation throughput rate - data rate. The whole concept of the design will be based on making this high rate possible when the system is in the so-called data mode.
Another very important limiting condition is that the fode will work in a synchronous way. This means a continuous stream of bits, grouped in frames, from one fode to the other, whether these bits contain relevant information or not. Had we used the asynchronous way of sending information only relevant information would have been sent from one fode to the other. The reason for choosing synchronous transmission, is that one of the group members works on a synchronous transmitter-receiver because it is easier to extract the clock signal in this case, provided the received signals contain enough clock information (fig 2.5: AMI-encoder and AMI-decoder), and because the synchronisation of a frame is easier since there is no need to critically detect steep (start) bit edges indicating the beginning of a frame.

As to the actual design of the fode itself, it must be clear that the fode must be able to either pick information from the bus and send it to the other fode, or else that it can receive the information sent by the other fode and put it on the bus. The fode must have a (data-) receive part and a (data-) transmit part.

Since we are dealing with a bus interface and with only two glass fibres, it will be obvious that the parallel information from the bus must be serialized before it can be transmitted over a fibre. The reverse procedure, converting the serial information into signals suited for the parallel IEC-bus, must be done by the receiving part of the fode of course. For these serial to parallel conversions (and vice versa of course) shift registers will be used.

Let us take a closer look at the transmitting part of the fode first. Later on we will pay more attention to the receiving part. (See figure 2.5) The information concerned, a data byte or a command byte, supplied to the fode, will have to be loaded into the shift registers (shift reg) in order to serialize the information. Of course the information must be stable at the moment it is loaded as otherwise it would be impossible to 'know' whether the correct information, a talker or controller wanted to send, has been transmitted to the other fode yet. This information stability will be provided for by buffers, which will sample the offered information at correct moments. In a way these buffers also take care of the transition from the asynchronously working IEC-bus to the synchronous information transport between the two fodes.
But not only data or command bytes have to be serialized and transmitted to the other side of the extension.
As we have seen before the bus also consists of a number of 'handshake' and 'general interface management' uniline messages.
Because we will be dealing with relatively long extension lengths, it will prove impossible to just serialize and transmit these signals (chapter 3 and chapter 4).

Most of the afore mentioned uniline messages will have to be processed in one way or another, and this will be done in the block that is called the control logic in figure 2.5.
When the system is in the command mode, the handshake signals are sent over the glass fibre connection so the handshake procedure will be a normal one.
In the data mode, however, the handshake signals will not be sent over the glass fibre connection. This is necessary when we want to allow throughput rates up to 1 Mbyte/s.
That is the reason why the handshake procedure will be dealt with locally in that case (§3.1). This means that the node will act as some kind of substitution for the extended listeners, and that it will give the normal listeners reactions on behalf of those extended devices.
The block of fig 2.5 called logic control will also take care of the general management uniline messages.
Because the extended devices often cannot respond to controller commands in the limited reaction times the IEC-bus protocol demands, the logic control block of the node will provide these very essential reactions for them.
In a way the node will represent the extended devices in this mode.
It will also see to the fact that correct uniline messages are sent over the glass fibre, since the influence of small time delays of the bus will be removed and corrected by the transmitting node.

In the receiving part of the node the reverse procedures take place.
Let us first look at what happens to a data or a command byte.
Because the handshake signals will be sent over the glass fibre extension in the command mode, the handshake procedure will be normal in that case.
A command byte will be offered to the bus after the serial signal has been converted into a parallel signal.
In the data mode, however, the handshake signals will not be sent over the glass fibre connection. This implies that a talker doesn't 'know' whether the data byte it sends can be accepted by the extended listeners or not. Those listeners can decide for themselves (but not separately) when they can or want to receive a data byte.
If a listener is not ready to receive a data byte it will keep the handshake signal NRFD = true just like a listener would do in a normal situation.
But since there is no actual contact between a talker and the extended listeners, the receiving part of the node must have a storage possibility, in order to prevent data bytes to get lost.
The FIFO memory of figure 2.5 will be used as storage possibility. A data byte that arrives in a frame will be loaded in this FIFO-memory. The extended listeners start handshake procedures with the fode, and as a result of those procedures the data byte in the FIFO-memory is put on the bus and offered to those listeners. If for one reason or another the data bytes are loaded in the FIFO-memory faster than they are read out of this memory, they are stored in the correct order of arrival. When the FIFO-memory is getting full a signal will be sent to the other fode in order to stop the data transport using the handshake procedure.

The function of the block logic control in the receiving part of the fode is the correct liquidation of procedures. Because there is no actual contact between devices separated by the extension, the danger could exist that procedures get mixed up. The block logic control has to prevent this; it makes sure that successive procedures are dealt with in the correct order.
controller

device

FODE fibre

device

FODE

IEC_bus

device

device_extension

IEC_bus

fig 2.3
Figure 2.3 IEC-device extension
Figure 2.4 IEC-bus extension
fig 24

controller

device

device

FODE

device

IEC_bus

IEC_bus

bus_extension
Figure 2.5 Functional design of an IEC fode
Chapter 3 Lines of sight for an IEC-bus FODE with respect to byte transport and the uniline message ATN

Roughly speaking, we can distinguish two modes in case of the IEC-bus. These modes are the data mode (ATN = false) and the command mode (ATN = true). First we will look at the data mode to find out what possibly needs to be done in this mode. Secondly we will look at the consequences of designing a fode when the system changes from the data mode to the command mode.

Note: before we will look in more detail at matters it is necessary to state the following definitions.

Fode1 := the fode to which the controller is connected by bus.
Fode2 := the fode connected to the controller via the glass fibre connection and so via the fode1.

3.1 Data mode

In the data mode ordinary data bytes are sent by an addressed talker to one or more addressed listeners. Data transport takes place as outlined before in a parallel way by sending an 8 bits byte wrapped up in a handshake so to speak, as shown in figure 3.1.a. The maximum throughput rate according to the official IEC-standard is 1 Mbyte/s.

If the data transport would be handled in the traditional way after extending the connection, the talker and the listener(s) would have to handle the handshake procedures between each other. The maximum allowable throughput rate would certainly not be possible (See figure 3.1.b). When we are dealing with a total length of 5 kilometres, the maximum attainable throughput rate will be 10 kbyte/s.

In order to avoid this problem we now introduce a so-called local handshake. This means, as shown in figure 3.1c, that a fode not only will have to convert the electric signals into optical signals and vice versa, but that the fode will have to start playing an active role as well.
In case of this local handshake a talker will start handshake procedures with the fode and possible listeners with which the talker is connected by bus after a transition from the command mode to the data mode. On the other side of the extension the listener will start its handshake procedures with the fode to which it is connected, as well.

Data transmitted by the talker will be send to fode1. After that, fode1 will send the received data to fode2 and this fode will send the data to the bus connected listener(s).

However, since talker and listener no longer work together in a direct way, the data received by fode2 cannot be transmitted to the listener(s) directly upon arrival.

A talker doesn't know if the listener is ready to receive the offered data byte. So that a buffer has to be used. The data received by fode2 will be stored in a buffer so that the listener can take this data out (using the handshake) at a moment the listener can decide for itself. This way the co-operation between talker and listener(s) is restored in a sense.

As an elastic buffer we will use a FIFO-memory. The reasons why we have chosen a FIFO-memory are rather obvious. Most FIFO memories allow simultaneous read and write procedures. This means that we can read a data byte from and write a data byte in the FIFO-memory simultaneously.

Another advantage is the fact that a FIFO-memory takes care of the necessary addressing itself. Every data byte is stored in the order of arrival, which is very fortunate.

When for one reason or another a listener is not capable to receive data for a short period or when the listener cannot receive the data bytes as fast as the talker transmits them, these not yet accepted data bytes are stored in the FIFO-memory in the correct order of arrival. Storing the data bytes in the correct order automatically implies the possibility of reading out the data bytes in the correct order.

Of course we have to take into account that the FIFO-memory only has a limited storage capacity. Therefore fode2 will send a signal to fode1 when the FIFO-memory is getting filled.

Upon reception of this signal fode1 will block the data transport with the handshake procedure (NRFD will not go false, but is kept true). Of course the FIFO-memory must have extra capacity because there can be 25 data bytes on the glass fibre connection on their way and furthermore it will take 25 microseconds for the signal "memory full" to reach fode 1. These 25 microseconds can represent 25 data bytes. Upon transmission of the signal "memory full" the FIFO-memory must be able to store 50 extra data bytes (this number is based upon a total connection length of 5 kilometres).

In short we can say that because of the local way of handling the handshake and by using a FIFO-memory the possibility is created to realise the maximum throughput rate of 1 Mbyte/s even for connection lengths of 5 kilometres.
Figure 3.1 Local handshake
fig 3.1

L -> T

L -> F2 -> F1 -> T

L -> # F2 -> # F1 -> T

F: FODE
handshake
fifo_memory:
influence fifo_memory on handshake
data_transport from talker T to listener L
in data_mode
### 3.2 Transition of data mode to command mode, and command mode
figures 3.2, 3.3

The second possible mode is the command mode. The system is in this mode when the controller has made ATN = true. In this state a talker cannot exchange data with a listener because the only thing that both a talker and a listener can do is receiving the controller's commands.

In this state the so-called serial and parallel polling is possible. During those pollings the connected devices are asked whether they have something to report or not.

But before we will focus on the command mode, it is essential to look at the transition from data mode to command mode. A lot of problems can occur when ATN changes from false to true.

First of all we must note that a local action will be required from fode1, when ATN changes from false to true. According to the IEC-bus protocol the controller can demand a reaction from the connected devices within 500 nanoseconds after the transition. If we were to use the fodes only as electrical optical "translators", it would be impossible for the extended devices to respond to the transition in the demanded time. Again that is the reason why the fode will have to give the appropriate reaction. It is self-evident that the fode also has to take care for the transportation of this message to the other fode as fast as possible.

The transition of ATN from false to true also has an impact on the data exchange in the data mode. Assume for instance that both controller and talker are connected to fode1 and that one or more listeners are connected to fode2 as shown in figure 3.2.a. Data bytes from the talker will be sent via fode1 and fode2 to the listeners. Another assumption will be that there are some data bytes stored in the FIFO-memory. At this moment the controller makes ATN = true in order to allow the control of the system. Fode1 will send this uniline message to fode2, but if this message was to be put on the bus immediately after reception, it would deprive the listener its listening capability. If the first command of the controller would be an unlisten command, it would be impossible for the listener to read out the data bytes still stored.
Of course this is an intolerable situation. We have to see to it that the uniline message $\text{ATH} = \text{true}$ is passed through after the FIFO-memory has been emptied.

A counter is connected to the FIFO-memory to find out if the FIFO-memory is empty or not. This counter keeps the number of times that a byte is stored in the FIFO-memory and a byte is read out. During the period that the FIFO-memory is not yet empty, fodel must see to the fact that the controller doesn't put command bytes on the bus.

As soon as the FIFO-memory is empty, the uniline message $\text{ATH} = \text{true}$ will no longer be blocked and will be put on the bus on the extension side. Fode2 will also inform fodel of this event and after reception of this message fodel will allow the controller to transmit command bytes.

On the extension side the command bytes will not be put on the bus via the FIFO-memory but directly. In this situation the handshake will not be handled locally.

The reason for not using the FIFO-memory in case of command bytes is that we would need another counter to count the number of incoming command bytes in order to be able to make a distinction between data bytes and command bytes.

Another reason for not using the FIFO-memory is that in this way we can be sure that the FIFO-memory is empty when the system is switched back from the command mode to the data mode.

Since command bytes cannot be stored in the FIFO-memory it also means that the complete system has accepted the commands after the command mode has been left and that the transport of data bytes in the data mode can start at once.

Of course we have to admit, that the because of this the throughput rate of command bytes is much lower in comparison with the throughput rate of data bytes, but it must be realized that the number of command bytes will be much lower than the number of data bytes. The extra delay is allowable, especially when we also take into account that by not using the FIFO-memory, there will be no fall through time.

That is the reason why we have chosen for this solution. Again, the advantage of this way of handling is that when the system switches back from the command mode to the data mode we can be sure of two things: first we can be certain that all commands have been accepted by the system, and secondly that the FIFO-memory is ready to be used.
If the position of talker and listener in figure 3.2.a is interchanged a new problem can arise.

Figure 3.2.b shows the new configuration of the system.

In this figure the controller and a listener are connected with fodel and the talker is connected to fode2.

As soon as the controller makes the uniline message ATN = true, it will be impossible for the listener to read the data bytes stored in the FIFO-memory of fode1.

Indeed we can see to it that the command bytes are not put on the bus by the controller by keeping NRFD true, but we cannot force the uniline message ATN = true to go false for the time needed to empty the FIFO-memory.

The FIFO-memory, however, can get fuller and fuller because on the one hand no bytes can be read out and on the other hand time is needed for the uniline message to reach the talker and silence it.

From the moment that the uniline message ATN = true is put on the bus by the controller (and deactivates the read capability of the listener) until the moment that it has been reported back to fode1 that the entire system is in the command mode, it can be expected that a maximum of 50 bytes more will have to be stored in the FIFO-memory (5 kilometres).

In the previous part of this section we have encountered no real problems when trying to find a solution to read out all the data bytes from the FIFO-memory of fode2.

In this case, however, matters are not that easy because there is a certain restriction, which could require giving up transparency with respect to the IEC-bus protocol. If we want to take care of the stored bytes in the FIFO-memory of fode1 we have to change the software of the controller.

With the help of the software changes we can see to it that the stored bytes are read out.

The reason for the software changes is obvious. Suppose that we would allow the controller to transmit arbitrary command bytes after fode1 has given permission to the controller to start transmitting command bytes.

This could mean that the listener(s) could receive an unlisten command, which would imply that the destination of the data bytes stored in the FIFO-memory would be lost. Of course this is unacceptable.

There are two solutions to overcome this problem.

The first solution is a general one. As soon as fode1 has received a message from fode2 that the entire system is in the command mode, the controller is allowed only to transmit an untalk command.

Once this untalk command has been accepted by the system, the controller must see to the re-entry of the data mode.

Because it is impossible to report directly to the controller that the FIFO-memory is empty, the controller must be made capable of finding out whether the handshake line DAV is made true within a certain time interval after a true -> false transition of the handshake line NRFD or not.
If that is not the case it is a sign for the controller that the FIFO-memory of fodel is empty, and thus the sign to enter the command mode and to start the possible issue of commands. In this way we can make sure that the FIFO memories of both fodes are emptied before actual commands can be given. This solution has the advantage that it can function without the use of addresses, dedicated lines etc. We can provide the controller with this software change and from that moment on the controller doesn't need information about whether the connection is extended or not. The total system stays transparent with respect to the IEC-bus protocol. This means among other things that the fode doesn't need to be fitted in the system explicitly, it doesn't need addresses or dedicated lines.

The second solution uses the parallel polling method. By making both uniline messages ATN and EOI true, a controller can start a parallel polling procedure. The polled devices can then report to the controller if they want some kind of service or not. If we want to use this solution a parallel polling respond line must be appointed to fodel. Of course we must tell the controller what to do when the fode requests service. The last item is the least important, because it is an ordinary software change. The first item, the appointing of a respond line, means a radical change in our whole concept.

This time we will have to reserve a polling line for the fode or we will have to appoint a line to the fode. Until now the fode acted as an anonymous extending and data processing device, but when a respond line is appointed to the fode this function has changed. Now the system has lost its transparency. Still it is a solution and therefore we will explain the procedure. After the false -> true transition of the uniline message ATN the controller must send an untalk command to the other devices. After confirmation of the reception of this command the controller has to start parallel polling procedures. When the fode reports that the FIFO-memory is not empty, yet the controller must return to the data mode to enable listeners to read out the data from the memory. After a "while" the controller can start the parallel polling procedures again in order to check if the memory is empty or not. If it still contains data bytes the data mode is re-entered, otherwise the "normal" command mode is entered and the controller can start giving commands.
Serial polling cannot be used as a means to empty the FIFO-memory of fodel. In case of serial polling a device is asked if it needs some kind of service by addressing each device as a talker. But if we would allow a fode to be polled in this serial way, this would also mean that the other devices will be addressed as talkers so that they would automatically loose their listen function. This is of course a very bad remedy because the destination of the stored data bytes might get lost. Figure 3.3 shows the general solution for emptying the FIFO memories in a flow chart.
Figure 3.2 Important configurations with respect to emptying the FIFO memory of fode1 and fode2
fig 3.2

a

C

T

F₁

D

L

F₂

b

C controller
T talker
L listener
D device

F fode
B IEC_bus
# fifo_memory
--- data
FIG 3.3

CONTROLLER  FOIDE1  FOIDE2

CONTR: SEND ATN = 1
FOIDE1: RECEIVE ATN = 1 → BLOCK COMMAND BYTE EXCHANGE
WITH CONTROLLER
SEND ATN=1 TO FOIDE 2

FOIDE2 RECEIVE ATN = 1

FOIDE2 EMPTY?

CONTINUE
DATA
EXCHANGE

PUT ATN=1 ON BUS AND REPORT TO FOIDE 1

FOIDE1: DEBLOCK COMMAND BYTE EXCHANGE
WITH CONTROLLER
CONTR: SEND UNTALK COMMAND

HAS ENTIRE SYSTEM CONFIRMED UNTALK COM.?

WAIT

GO TO DATAMODE

FOIDE1: FOIDE1 EMPTY?

CONTINUE DATA EXCHANGE

GO TO COMMAND MODE & GIVE COMMANDS

CONTR:
Figure 3.3 Flow chart: general solution for emptying the FIFO-memories of fode1 and fode2
Chapter 4 Effects of extension on the general management unilines and on the polling procedures

Of course the extension length not only has an impact on the transportation of data bytes, command bytes and on the uniline message ATN as we have seen in the previous chapter. Its influence will also be noticeable with respect to other messages or situations. Another important factor is the way these messages from the bus are handled by the node, for instance, as has been mentioned before the node will correct mistakes due to small time delays. In this chapter we will take a closer look at these things.

4.1 Influence of the extension on the uniline messages

Because we are dealing with a synchronous frame transport system between the nodes a frame will not always contain relevant information, but it will also contain irrelevant information once and a while. Furthermore, as the shiftregisters are loaded every microsecond in order to serialize the offered signals, it could very well happen that one frame contains several relevant signal changes, that have happened one after another. If nothing is done, it may even happen that incorrect messages are sent to the other side of the extension, which is intolerable of course. We will look at the situations involved one by one.

EOI - IFC
It is a normal procedure for a controller to activate the uniline message IFC after the current talker has made EOI = true to inform the listeners that it has put its last data byte on the bus. It will be one of the node's tasks to see to it that the time order information is passed on correctly. If both EOI = true and IFC = true would occur in one and the same frame and if this information would be passed on to the bus by the receiving node without processing, the situation on the bus on the extended side would be a bad representation of the original situation. That is the reason that the IFC uniline message will be delayed for one frame period by the transmitting node when EOI = true. In that case EOI = true arrives at the receiving node one frame period earlier then the uniline message IFC = true. This enables the receiving node to make sure that the correct time order is maintained. First EOI = true will be put on the bus, and after this message line has changed to false again, the uniline message IFC = true will be put on the bus.
EOI - ATN

A similar situation arises in case of the uniline messages EOI and ATN. The combination EOI = true and ATN = false means that a current talker informs the connected listeners that it has put its last data byte on the bus.

However, the combination EOI = true and ATN = true indicates that the controller has started a parallel polling procedure.

Now suppose a current talker makes the uniline message EOI = true because it sends its last data byte. After EOI is made false again, the controller makes ATN = true. What has happened thus far is that the talker has informed the bus of its last data byte, after which the controller has changed the data mode into the command mode.

If we would allow EOI = true and ATN = true together in one frame and if the receiver would place both on the bus without taking any precautions, the original meaning of the combined uniline messages would have been changed into a parallel polling command. Of course this cannot be allowed, and therefore the fode will have to be designed in such a way that the problems as outlined before cannot arise.

Because of small time delays between the two uniline messages the reverse could happen as well. It is possible that a 'parallel polling' message is sent as a 'last data byte' message in one frame before the correct message is sent in the next frame.

We cannot allow these things and that is the reason why the fode will have to correct these mistakes due to small time delays. This brings us to a related problem: polling methods.
4.2 Influence of the extension on serial and parallel polling procedures.

In the case of serial polling procedures there are no time limits specified by the IEC-bus standard. Transitions from one state to another do not have to take place within a limited time, nor are there any time restrictions to possible reactions. Therefore serial polling can even be used to check the extended devices. The only restriction is that we cannot use this polling method to find out if the FIFO memory of fode1 is empty or not.

Parallel polling procedures cannot be used for checking the devices at the other side of the extension. Here too the controller can demand reaction times of 500 nanoseconds according to the IEC-bus standard. Again it is all together obvious that these demands cannot be met in case of the extended devices. A logical question is: can't fode1 handle the parallel polling locally. The answer unfortunately is no. This is impossible because this would mean that the fode has to start a parallel polling procedure by itself and has to do this even before the real polling has started. One could say that the fode has to be clairvoyant in this case. Since responses from the extended devices cannot be expected in time and can even arrive on a most inconvenient moment, it has been decided on to let the fode block the parallel polling command (ATN = true and EOI = true). It will be impossible for the controller to parallel poll the extended devices.
Chapter 5 Design of a FODE

Figure 5.0

In the previous chapters we have examined the criteria a fode will have to meet.
We have seen that the IEC-bus connection cannot be extended just like that; very often we will have to process the signals offered from the bus to the fode.
In this chapter we will look at matters from another angle.
We will examine a concrete fode design with the help of several figures.
Because we are dealing with a rather complex design we will only look at one part of the design at a time.
Figure 5.0 is a schematical overall diagram of the design.
In this diagram the specific parts we are going to look at in more detail, are indicated with Roman figures.
When a block contains a (part of a) block, this will be indicated by the continuation of the bordering lines of that block in the other block as dotted lines.
For instance block II contains a part of block I.
Signals which cross the borders of the blocks are indicated as well.

Before we will continue, the meaning of some terms used in the texts accompanying the figures will be explained.
Flag a: signal sent by fode2 to report that the devices connected to that fode are in the command-mode and that those devices have reported themselves ready to receive a command-byte from the controller, it is a combined \text{NRFD} = false message from the extended devices.
Flag b: signal sent by fode2 to report that all devices connected to that fode have accepted the command-byte from the controller, it is a combined \text{NDAC} = false message from the extended devices.
Flag c: the fode which sends flag c = true is connected to the current talker directly by bus. The fode which receives flag c = true has to leave the acceptor-state and has to enter the source-state because it is not connected to the current talker directly by bus, but it is connected to the talker by glass fibre.
Flag d: flag d is used to inform the fode on the other side of the extension that it has to interrupt the data transport because the FIFO-memory of the fode sending flag d is getting full.
Flag e: with this flag a fode that transmits a data or command byte makes it clear to the receiving fode whether the frame contains relevant byte information or not.
When flag e = true the information is relevant, otherwise it is worthless in which case the receiving fode doesn't have to process the data.

N.B. Flags a and b will be used in the command mode only.
Flags c and d will be used in the data mode only.
Flag e will be used in both the command mode and the data mode.
Some of the symbols used in the figures are:

'1' = logical one = true
( ≤ 0.8 Volt for TTL)
( ≥ 1.5 Volt for ECL)

'0' = logical zero = false
( ≥ 2.0 Volt for TTL)
( ≤ 1.1 Volt for ECL)

X = don't care

init= initialisation pulse

Sometimes underscored figures are added to flipflops, gates, counters etc. This is done to make it easy to find a specific device in the drawing that is discussed in the accompanying text.

A last remark concerns the use of indices. In some cases we will indexate the signals to prevent confusion. Here is a list of the indices used together with their meaning.

<table>
<thead>
<tr>
<th>Index</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>signal from receiver, or received signal</td>
</tr>
<tr>
<td>t</td>
<td>signal from transmitter, or signal to be transmitted</td>
</tr>
<tr>
<td>in</td>
<td>signal entering a circuit</td>
</tr>
<tr>
<td>out</td>
<td>signal leaving circuit</td>
</tr>
<tr>
<td>bus</td>
<td>signal picked up from the bus</td>
</tr>
<tr>
<td>free</td>
<td>signal that possibly has been blocked, but is available now</td>
</tr>
<tr>
<td>buffer</td>
<td>signal that comes directly from receiving buffer</td>
</tr>
</tbody>
</table>
fig 5.0 General Scheme
5.1 Datapath from the bus to the glassfibre and 
from the receiving part of the fode to the bus 
part I, figure 5.1

Figure 5.1 shows that part of the fode the receiving and the 
transmitting parts have in common. 
This common part consists of transceivers and of ECL <-> TTL 
translators. 
With the transceivers we are able to pick up the signals from the 
IEC-bus (receive) or put them on the bus (drive or send). 
The TTL -> ECL translator converts the TTL-levels of the IEC-bus signals 
into ECL-levels; the ECL -> TTL translator works the other way around. 
This has to be done because the major part of the fode has been made 
with ECL integrated circuits. 
These ECL circuits have been chosen because of their speed and because 
of the fact that they are not very sensitive for electro-magnetic 
interference. 
This concludes the common part.

Now let us look at what happens to a data or command byte received from 
the bus. 
After having been received by the transceivers and after the signal 
levels have been converted, the byte will be offered to a buffer. 
After a load pulse from the 'Acceptor Handshake' part the data or 
command byte will be loaded in the buffers. We have to be aware that 
the data or command byte lines are sampled by this operation, and that 
from now on we will be dealing with samples of the 8 lines which 
together form a data or command byte. 
After the data or command byte has been loaded, the data or command byte 
samples will be offered to the parallel-serial shift-registers (we will 
not use the term samples again; for instance, we will not speak of 
command byte samples but instead we will use the term command byte). 
When the contents of the 'Frame-Counter of the transmitting part' is 
0 (zero) the shiftregisters will load the offered data. 
Now together with (the samples of) some other signals the data or 
command byte is loaded into the registers. 
The shift frequency is 24 MHz. 

This shift procedure will result in a serial signal that will be offered 
to the AMI-encoder (part IV;fig 5.4)
fig 5.1 Datapath
5.2 Determination of the talker-side and the controller-side  
and controlling the transceivers  
part II, figure 5.2

If we would open an IEC-bus on an arbitrary place, we would not be able to know to which side of the extension the controller is connected.  
This knowledge is important, for instance if we want to prevent retransmission of a received signal by the fode.  
It is also important because the IEC-bus has two wired-or handshake signals. It is essential that the fode doesn't affect this wired-or capability.

Both arguments also hold for the data mode. Here it is necessary that we know to which side of the extension the active talker is connected.

Determination of the controller-side
During initialization the flipflops 1 and 2 will be reset.  
Next the fode will try to determine the controller-side.  
This is done by checking if the specific controller uniline messages ATN, REN and IFC reach the fode via the bus or via the glass fibre connection.

Flipflop 1 will be set by the reception of one or more of the uniline messages ATN = true, IFC = true and REN = true.  
On the other hand flipflop 2 will be set when one or more of the three signals reach the fode via the glass fibre connection.  
Of course the signals concerned are the signals on the output of the receiving buffers.
We will not use the delayed signals for this purpose.
If both flipflops are set accidentally, this error will be rectified automatically by resetting both flipflops.

When flipflop 1 has been set, the drivers of the transceivers for the uniline messages ATN, IFC and REN will be deactivated.
The TTL->ECL translators marked 5 will be activated.

If flipflop 2 had been set, the drivers of the mentioned signals would have been activated.
In this case the signals would have been put on the bus.
The enable input of the TTL-ECL translators would see to it that retransmission of the signal would not be possible.
The reception of the uniline message ATN = true via the bus or via the glass fibre extension also determines the direction of the command byte transport.
When it has been established that a fode is connected to the controller via the bus (and can be considered as fode1 from then on) a command byte on the data lines DIO 1 to 8 and the handshake signal DAV will be picked up from the bus. The other two handshake signals NRFD and NDAC will be put on the bus as wired-or signals.

Should ATN = true have been received via the glass fibre connections things would have been arranged the other way around: the data lines as well as the handshake signal DAV would have been put on the bus, and the handshake signals NRFD and NDAC would have been picked up from the bus.

The above is only valid when the uniline message EOI = false.
In the command mode EOI = true means that the controller wants to parallel poll the devices.
In that case the direction of the data lines DIO 1 to 8 should be towards the controller. The information on the data lines should not be transmitted, it has to be received by the controller.

**Determination of the talker-side**

When the system is in the data mode we have to determine to which side of the extension the talker is connected.

We can do this in the following way.

After the transition to the data mode both fodes will react as listeners.

They will both report NRFD = false.

Because only one talker can be active at a time, only one fode will receive the handshake signal DAV = true.

As soon as this handshake message arrives at one of the fodes it is known to which of the fodes the talker is connected.

Then flipflop 3 will be set and flag c = true will be transmitted to the other fode.

When the other fode receives flag c = true, it 'knows' that the talker is on the other side of the connection. It will leave the acceptor state and it will enter the source state.

This will be done by setting flipflop 4.

If both flipflops are accidentally set, this will automatically lead to a reset signal for both.

Once the talker side has been established the direction of the data byte transport is known too.

When the system leaves the data mode and enters the command mode the flipflops 3 and 4 will be reset.

During the command mode the flipflops 3 and 4 cannot be set.
When it has been established that a fode is connected to the controller via the bus (and can be considered as fodel from then on) a command byte on the data lines DIO 1 to 8 and the handshake signal DAV will be picked up from the bus. The other two handshake signals NRFD and NDAC will be put on the bus as wired-or signals.

Should ATN = true have been received via the glass fibre connections things would have been arranged the other way around: the data lines as well as the handshake signal DAV would have been put on the bus, and the handshake signals NRFD and NDAC would have been picked up from the bus.

The above is only valid when the uniline message EOI = false.

In the command mode EOI = true means that the controller wants to parallel poll the devices.

In that case the direction of the data lines DIO 1 to 8 should be towards the controller. The information on the data lines should not be transmitted, it has to be received by the controller.

**Determination of the talker-side**

When the system is in the data mode we have to determine to which side of the extension the talker is connected.

We can do this in the following way.

After the transition to the data mode both fodes will react as listeners.

They will both report NRFD = false.

Because only one talker can be active at a time, only one fode will receive the handshake signal DAV = true.

As soon as this handshake message arrives at one of the fodes it is known to which of the fodes the talker is connected.

Then flipflop 3 will be set and flag c = true will be transmitted to the other fode.

When the other fode receives flag c = true, it 'knows' that the talker is on the other side of the connection. It will leave the acceptor state and it will enter the source state.

This will be done by setting flipflop 4.

If both flipflops are accidentally set, this will automatically lead to a reset signal for both.

Once the talker side has been established the direction of the data byte transport is known too.

When the system leaves the data mode and enters the command mode the flipflops 3 and 4 will be reset.

During the command mode the flipflops 3 and 4 cannot be set.
fig 5.2 Determination Controller & Talkerside
5.3 The acceptor handshake part

The acceptor handshake part consists of two parts. The first part is the acceptor handshake in the data mode (ATN = false) and the second part is the acceptor handshake in the command mode (ATN = true).

Roughly speaking, the acceptor handshake has the following structure. A listener reports that it is ready to receive a data byte by making NRFD = false. A talker will react to this transition of NRFD by making the handshake signal DAV = true as soon as the data byte it wants to send is valid. After that the listener will make NRFD = true. When the listener has accepted the offered data byte it will make the handshake signal NDAC = false. Knowing that its data byte has been accepted the listener will make DAV = false again. The listener will react to this transition by making NDAC = true. Now the handshake procedure can start again.

NRFD   |       |
       |
DAV    |       |       |
       |
NDAC   |       |
       |

Acceptor handshake in the data mode

First we will pay attention to the acceptor handshake procedure in the data mode. In this mode the handshake will be dealt with locally. This means, that devices interconnected by the glass fibre extension will not deal with each other directly, so the handshake will not be transmitted along the glass fibre. The devices will start handshake procedures with the node to which they are connected by bus. Let us take a closer look at how the acceptor handshake of the node in the data mode works.

During initialization flipflop 1 will be set. Now the node reports to the bus that it is ready to accept a data byte: NRFD = false. A talker can respond to this by making DAV = true as a sign that the data byte it offers can be considered as valid. After reception of the handshake signal DAV = true by the acceptor handshake of the node, flipflop 1 will be reset so that NRFD = true.
Now a number of conditions have to be fulfilled. When DAV = true and the frame-counting circuit doesn't send a blocking signal (see the frame counter transmitter part of the fode $5.4$) and when flag $e$ is not true yet, the output of OR-gate 6 will go false and the connected multivibrator will give a pulse.

(When flag $e$ = true the output of OR-gate 6 cannot go false, because there is still a byte waiting to be put in a frame.)

We have to remark that the first data byte of a new data mode cyclus will be delayed at least a frame. This delay gives the fode on the other end of the extension the time to "prepare" the data transport.

The flipflops 4 and 5 will take care of this delay, that is initialized by the true $\rightarrow$ false transition of the uniline message ATN.

The pulse generated by the mentioned multivibrator has several functions.

This pulse will see to the fact that the data byte will be sampled by the buffers and that these samples are offered to the shiftregisters.

The second function of this pulse is to set flipflop 3.

When flipflop 3 is set this means that flag $e$ will be offered to the frame as being true. Flag $e$ is a sign for the other fode; it informs that fode if the received data byte is relevant information.

If flag $e$ = false the receiving fode doesn't have to store the data byte in the FIFO-memory. If on the other hand this fode receives flag $e$ = true, it will automatically store that data byte in the FIFO-memory because it represents relevant information.

The third function of the pulse is to set flipflop 2.

In this way NDAC is made false, so that the talker knows that its offered data have been accepted.

Once the talker is informed of the fact that its data byte has been accepted by the listeners, it will make DAV = false.

This true $\rightarrow$ false transition of the handshake signal DAV will reset flipflop 2. Thus the handshake signal NDAC is made true again.

Provided flag $d$ is not true the true $\rightarrow$ false transition of NDAC will reset flipflop 1 (NRFD = false). Now the fode reports itself ready to receive a new data byte.

Flag $d$ is the sign from the fode on the other side of the extension with which it makes clear that it wants to have the data transport stopped, because its FIFO-memory is getting full.

Upon reception of flag $d$ = true the transmitting fode will block the data transport by not allowing NRFD to go false. The data transport will be resumed as soon as the transmitting fode receives flag $d$ = false.
The acceptor handshake in the command mode
In the command mode the handshake will not be handled locally, but is
transmitted via the glass fibre.
This means that the controller and all the connected devices handle the
handshake procedures mutually.
This also holds for the extended devices.
The node will take no part in the handshake procedures.
As the handshake will not be handled in a local way, a few automatic
reactions of the acceptor handshake part in the data mode of the node
will have to be blocked. The gates 7, 8, and 9 have to take care of this.

The handshake procedure in the command mode proceeds as follows.
As soon as the node receives the uniline message ATN-true flipflop 1
will be reset. This means that NRFD will stay true or changes to true.
Only after reception of flag a = true flipflop 1 will be reset and NRFD
will be made false. Flag a is the sign of node 2 to report to node 1 that
ATN = true has been placed on the bus and that all extended devices have
reported themselves ready to receive a data byte.

As soon as the controller has received NRFD = false and as soon as the
command byte on the bus may be considered valid, the controller will
make DAV = true.
When the conditions as mentioned earlier are fulfilled (flag e = false,
no blocking signal from the frame counting circuit and DAV = true)
flipflop 3 will be set. This means that flag e = true
(data byte in frame is relevant).
The pulse used to set flipflop 3 is also used to load the valid data
byte into the buffers, so that now the samples will be offered to the
shift registers.
The third function of the pulse, setting flipflop 2 in order to make
NDAC = false, is suspended in this case.
This function will be taken over by flag b. This flag informs node 1 that
the transmitted command byte has been accepted by the extended devices.
After the handshake signal DAV has been made false by the controller
flipflop 2 will be reset.
The reception of flag a = true by node 1 will start the handshake pro-
cedure for an additional time.
fig 5.3 Acceptor Handshake
FIG53 / III Acceptor handshake

buffer pulse

I

flag

lock signal

count 14

IV

count 15

FIG53
5.4 Frame-counter transmitting part of fode
part IV, fig 5.4

In figure 5.4 besides the framecounter of the transmitting part of the fode other parts have been drawn as well. The relevant parts are: the framecounter itself, the parallel-serial shiftregisters and the part that has to take care of the AMI-encoding. The counting part of the figure shows a framecounter, the task of which being to inform the rest of the fode which bit of the frame is transmitted. This framecounter functions in an autonomously way, which means that it can not be influenced by other parts of the fode.

This is in contrast with the framecounter of the receiving part of the fode (§5.7) where the counter is influenced by the frame contents in order to synchronise the counter with the incoming frame.

The framecounter of the transmitting part of the fode counts from 0 (zero) up to 23 because one frame consists of 24 bits. The framecounter sees to correct management of the parallel-serial shiftregisters and of the AMI-encoding circuit.

The parallel-serial shiftregister's function is to change the offered parallel data into a serial signal. The offered data will be loaded into the shiftregisters on the count of 0 (zero) and from then on (1 to 23) the loaded data are shifted. After this, the serial signal is offered to the AMI-encoder. This part converts every frame bit into two bits. AMI is the acronym for Alternate Mark Inversion and this kind of encoding must see to the fact that the receiver can easily extract the clock signal from the incoming signal. This kind of AMI-encoding provides a transition (logical one to logical zero or vice versa) at the beginning of every offered frame-bit. However when the offered frame-bit is a logical zero, there will be an addition transition in the middle of the frame-bit.

So a logical one is converted into two logical ones or two logical zeros (this is dependent of what happened before) while a logical zero will be encoded as a logical one and a logical zero or as a logical zero and a logical one (also dependent of what happened before).

\[
\begin{align*}
\text{true} & \quad \ldots \quad \rightarrow \quad \ldots \quad \text{OR} \quad \ldots \\
'1' : \text{false} & \quad \ldots \quad \rightarrow \quad \ldots \\
\text{true} & \quad \ldots \quad \rightarrow \quad \ldots \\
'0' : \text{false} & \quad \ldots \quad \rightarrow \quad \ldots \\
\end{align*}
\]

To enable the framecounter of the receiving fode to synchronise with the arriving bits the transmitting fode will see to it that the first two bits (0 and 1) and the last two bits (22 and 23) will not be AMI-encoded.
The last two frame-bits will be sent as four logical zeros and this violation of the AMI-code is used to synchronise the frame counter of the receiving node.

The first two frame-bits will be encoded as four logical ones, so that the equilibrium between logical ones and logical zeros is maintained.

It is obvious that it is of utmost importance that the frame counter sends its control signals at the right moments to the parts of the node which need those signals.

With respect to timing diagram 2A.a of the AMI-encoding part in appendix 2 we can remark that this diagram shows us at which moments the two control signals are released.

The signals shown in the diagram are the output signals of the gates, flipflops etc. The time diagram shows worst case situations, with which we mean that in case of dealing with limited reaction times, the maximum propagation delay times and the maximum rise or fall times etc. are used. When on the other hand a signal is not allowed to change before a certain moment, the minimum propagation delay times and the minimum rise or fall times etc. are used.

The control signal count-zero-or-one (bit number zero or bit number one) is represented by the output of NOR-gate 5, and the control signal count-22-or-23 (bit number 22 or bit number 23) will be represented by the output of AND-gate 6.

The timing diagram 2A.b in appendix 2 shows what happens to the signals so that the correct encoding demands including the encoding violation demands, are met.

Another matter concerns the loading of the parallel data in the shift-registers (12).

Of course this has to be done at the correct moment: framecounter contents equals 0 (zero) and the data on the input-gates of the shiftregister are stable. Flipflop 1 will see to the loading of the data in the shiftregisters at the right moment.

This set-reset flipflop will be set by the framecounter when its contents is 23. It will be reset by the frame counter when its contents is 0 (zero).

The set flipflop in combination with a positive clock pulse will see to it that the offered data are loaded at the correct moment and that the counter restarts at 0 (zero).

The stability of the data is another matter of interest. When we want to be sure that the offered data are indeed loaded with the appropriate clock-signal, we have to take care that the data are offered to the shiftregister at least 2.5 nanoseconds before a positive clock-edge.

When this demand is not met we can't be sure of the data's stability. To prevent the loading of the unstable data in the shiftregisters the set possibility of flag e is blocked as well as the buffer load pulse. This blocking signal of the framecounter is shown in figure IV and it is represented by SET 9 in the timing-diagram of appendix 2a.
Furthermore it proved necessary (although this cannot be learned from the time diagram in the appendix) to release the blocking-signal at least 40 nanoseconds before the data are to be loaded in the shift-registers. It is also necessary that the block signal will last at least 20 nanoseconds.
To be on the safe side we have chosen for an earlier release of the blocking signal and it will last until the framecounter contents is 0 (zero) again.
The blockade will last 75 nanoseconds at the most.

The last control signals of the frame counter of the transmitting part of the fode are the signals count-14 (framecounter contents is 14) and count-15 (framecounter contents is 15). These signals are used in different parts of the transmitting and receiving part of the fode.
These count signals are used to make sure that some signals are blocked for at least one frame period, while on the other hand they are also used to see to it that signals are offered to a transmitting frame for at least one frame period.
In the latter case we can be sure that the offered information is indeed put in a frame and sent to the other fode.
In all cases a flipflop is set with "count-15" and it is reset with "count-14".
This way the flipflop will be set for one frame period.
fig 5.4 Framecounter & AMI-Encoder Transmitter
FIGS. 4/IV Frame counter & AML encoder transmitter

CLOCK FREQ

1/2 CLOCK FREQ
5.5  Determination of the time order of EOI, IFC and ATN
in case of a fode connected with the controller by bus.
part V, fig 5.5

IFC-part
The IFC flipflop 1 will be set by the incoming IFC uniline message
(IFC-in) on condition that EOI = false.
This flipflop will be reset again when the IFC uniline message goes
false.
According to the IEC-bus standard the IFC-line will stay true for at
least 100 microseconds.
The IFC part of the circuit can only be influenced by the EOI part.
As long as the incoming EOI uniline message (EOI-in) is true, the IFC
flipflop 1 cannot be set. This is also the case as long as the EOI
message (EOI-out) offered to the frame has not been accepted yet
(EOI-out is not transmitted yet) even though the incoming EOI uniline
message has gone false again.

EOI-part (End) (see § 3.2)
As explained before, the uniline message EOI = true can have two
possible meanings.
The combination EOI = true with ATN = false means that the current
talker sends its last data byte, while EOI = true combined with
ATN = false means that the controller has started a parallel polling
procedure.
The start of a parallel polling procedure is characterized by the change
of both uniline messages EOI and ATN from false to true.

EOI         > parallel polling procedure
            __________|
         ________|________
       ATN

It is understandable that because of the possible differences in
distances both signals have to travel and since they have to pass
through different logical circuits, very small differences may occur
in the arrival times of the two mentioned signals.
Assume, for instance, that because of the very small delay time dif-
fers the uniline message EOI will go from false to true first,
followed a little later by the uniline message ATN.

EOI         > delayed parallel
            __________|
         ________|________
       ATN

Although it is the intention to make clear that the system should be
in the parallel polling state, it seems to the logical circuits as
if a current talker gives the sign for the last data byte.
To prevent misinterpretation a test has been included.
After the uniline messages EOI = true and ATN = false have arrived at the determination circuitry, flipflop 4 will be set. Should in the time needed by the signal to go from the AND-gate to the output gate of the flipflop, the uniline message ATN have changed from false to true or, in other words, are we dealing with a parallel polling command, the decisive flipflop 3 will not be set. The wrong 'last-data byte' message will not be sent in that case. When on the other hand we are dealing with a correct 'last-data byte' message flipflop 3 will be set of course. The output-gate of the decisive flipflop 3 will stay true for the duration of at least one frame. This is done with the help of flipflop 2. This way we can be sure that the EOI message (EOI-out) is put in a frame and transmitted. When this has happened, the relevant flipflops (2, 3, 4) will be reset.

**Parallel polling blockade** (see § 4.3) When the controller starts parallel polling procedures the polled devices have to react within a certain time interval determined by the system. The shortest possible time interval is 500 nanoseconds. The extended devices are not able to react within this limited time. Therefore we have decided not to send parallel polling commands to those extended devices. As a consequence fodel will have to block these parallel polling commands. This blockade implies that both ATN-out = true and EOI-out = true cannot be sent in one and the same frame. As soon as this polling state ceases to be either EOI-out = true OR ATN = true can be transmitted. Flipflops 5 and 6 take care of the detection of the parallel polling state and see to the blocking of the command.
ATN-part
The ATN part is made as follows.
As long as the uniline message EOI = false as well as EOI-out = false (output-gate of flipflop 3 = false), the ATN flipflop 7 can be set.
To be sure we are not dealing with a delayed parallel polling command (in which case we are not allowed to send ATN = 1) a test has been built in, just as we have seen in the EOI-part.

When there is no parallel polling command the decisive flipflop 8 will be set.

Here too care has been taken to see to the fact that the output-gate of the decisive flipflop remains true for the duration of at least one frame. This is done with the help of flipflop 10. After this period the ATN-out line will be kept true as long as the incoming uniline message stays true.

Suppose, however, that after a true → false transition of the incoming uniline message EOI and during the period that the EOI-out line has not been put into the frame yet, the uniline message line ATN (ATN-in) changes from false to true.
Flipflop 7 cannot be set so the information might get lost.
A solution was found for this possible problem.
The incoming uniline message ATN = true will be stored in flipflop 9 but will not be released before the EOI-out message is put in a frame and sent to the other node.
After EOI-out has been sent to the other node flipflop 11 will no longer block the output of flipflop 9.
In this case too the ATN-out message will stay true as long as the incoming uniline message ATN is true, but never for a shorter period than the frame duration.
fig 5.5 Determination Timeorder Controller-connected Fode
FIG55/V Determination time order
controller connected mode
5.6 Emptying the FIFO-memory using the parallel polling method
part VI, fig 5.6 (see § 4.2)

In the previous chapter we have seen that special precautions have to be taken to see to it that the FIFO-memory of the node can be emptied when the controller wants to change the state of the system from the data mode to the command mode.

One of the possibilities is to change the software of the controller in such a way that the data mode is re-entered after an untalk command. Once back in the data mode the controller has to check whether the handshake line DAV is made true within a certain time after the true <-> false transition of the handshake line NRFD.

The second solution is the following. After the data mode to command mode transition the controller gives an untalk command. When the entire system has confirmed the reception of this command, the controller will start a parallel polling procedure. When the controller receives a true state of an earlier appointed line during this parallel polling procedure, this means that the controller is informed that the FIFO-memory of the node is not empty yet. Now the controller has to start a subroutine in which the systems state returns to the data mode.

After an arbitrary time interval, say 50 microseconds, a new parallel polling procedure has to be started to see if the memory is empty yet. If indeed the memory is empty, the line will be sent logical false and this implies that the controller can start transmitting commands. When on the other hand the line is still received as being logical true, the subroutine has to be initialized again.

Of course there is the question of which of the 8 data lines will represent the FIFO-memory of the node during parallel polling procedures. A switch will be incorporated so we can establish our choice by hand. Line X (X = 1, ,8) will represent the following.

Line X = (NONDIOX+ATNbus) + (MEMORYEMPTY+NONSWITCH+NONATNbus)
Line X = (DIOX.NONATNbus) + (SWITCHX.ATNbus.NONMEMORYEMPTY)
so in case ATN = 0 : Line X = DIOX
and in case ATN = 1 : Line X = (SWITCH.NONMEMORYEMPTY)

Suppose line 8 is appointed to represent the node as a polling-line for the FIFO-memory, then

in case ATN = 1 : Line 1 to 7 are false and Line 8 = NONMEMORYEMPTY

+ = logical or
. = logical and
NON = = invert logical signal
fig 5.6 Empty # With Parallel Poll
5.7 **Frame counter receiver part of the fode**

part VII, fig 5.7

The frame counter in the receiving part of the fode practically has the same function as the frame counter of the transmitting part of the fode.

The counter has to count from 0 (zero) to 23, because a frame consists of 24 bits.

In this case too the counter has to inform the rest of the fode what signal the current bit represents.

So it is essential that control signals from the frame counter are sent at the correct moment to the parts concerned.

In figure 5.7 not only the frame counter has been drawn, but also the AMI-decoder, the frame counter synchronisation part and (schematically) the buffers and the serial-parallel shift registers.

The AMI-encoded data arrive in shiftregister 1 after which they are decoded with an exclusive OR.

As a result of this decoding, the 48 bits passing through shiftregister 1 will be reduced to 24 frame bits. Then these 24 frame bits are shifted in shiftregister 12. The data will be loaded in the buffers 13 using a pulse that is sent by OR-gate 10. From now on the serially offered data can be handled in a parallel way.

One of the possibilities is to store the data-bits DIO 1-8 in a FIFO-memory.

In order to be able to do this a write pulse is given to the FIFO-memory by the frame counter of the receiving part when the contents of this counter equals 0 (zero).

It will be clear that it is absolutely essential that the control signals are released at the correct moments.

Therefore it is equally clear that it is very important that the frame counter is synchronised with the frame.

To guarantee this synchronisation the frame counter not only has an autonomous part, but it can also be influenced by the contents of the frame.

The autonomous part of the counter counts automatically and repeatingly from 0 (zero) to 23. However, with the use of OR-gate 2, shiftregister 3, multivibrator 3a and OR-gate 4 the counter can be reset by the contents of the frame. This is done as follows.

The last two frame bits (22 and 23) will be shifted in shiftregister 1 as 4 logical zeros.

This AMI-encoding violation will be detected by OR-gate 2.

This OR-gate will pass on the message. At the moment bit 0 (zero) of the next frame is offered to shiftregister 12 the frame counter will be reset.

In case of a possible disturbance of the AMI-encoding violation the frame counter will not be reset by the frame contents but by itself (autonomous part).

In the time diagram A2.c of appendix 2 several discussed items concerning timing are lined up.
fig 5.7 Framecounter & AMI-Decoder Receiver
5.8 Frame receiving part of the fode

After the 48 bits of the incoming frame have been decoded, the frame will consist of 24 bits again. These 24 bits will be shifted in the serial-parallel shift registers with a shift frequency of 24 MHz. When a complete frame has been shifted into the shift registers the 'frame counter of the receiving part' of the fode will give a load pulse to the buffers connected to the shift registers. The contents of the frame-counter will be 23. After that the data on the output gates of the shift registers will be loaded into the buffers and then the contents of the frame, minus the data or command byte lines DIO 1-8, will be available for the rest of the receiving part of the fode.

A data or command byte will be handled in a different way. One of the components of the frame is flag e. This flag informs a receiving fode whether the received byte contains relevant information or not. In case we are dealing with a relevant data or command byte, flag e = true will be sent and the information is loaded in a second set of buffers. After the received byte has been stored in a buffer for the second time it can be stored in a FIFO-memory when it is a data byte. When the received byte is a command byte it will be lead around this FIFO-memory and is sent 'directly' to the bus. The ATN-free message will determine the way a byte will be handled. When the received byte does not contain any relevant information, the fode will receive flag e = false. In this case the received byte will not be stored in the secondary buffers, so it cannot be stored in the FIFO-memory or be sent to the bus 'directly'.
fig 5.8 Frame Receiver
FIG 58/VII Framereceiver

- EOI buffer
- ATN buffer
- REN buffer
- IFC buffer
- SRQ

- 3 ECL → TTL
- 2 TTL → ECL

- Flag E
- Flag D
- Flag A
- Flag B

- 1

- DIOx (data mode)
- ATN_Free
- DIOx (command mode)
5.9 Determination of the time order of EOI, IFC and ATN
in case of a fode connected to the controller by the extension
part IX, fig 5.9

The EOI- and IFC-part
(see § 3.2 and § 4.3)
As we have seen at the transmitting side (§ 5.5), it is not
possible that both uniline messages EOI and ATN are true in one frame.
It is also impossible that the uniline messages EOI and IFC are true
in one frame.

At the transmitting side care was taken to guard the correct time
information.
Sometimes the time order was even accented.
Of course we have to see to it that this very relevant information is
kept intact at the receiving side of the extension.

After the reception of $EOI = true$ flipflop 1 will be set. The output
gate of this flipflop must prevent that the output gate of flipflop 2,
which can be set upon arrival of the next frame by the $IFC = true$
message, is put on the bus.
The $EOI = true$ message is released when the last data byte in the
FIFO-memory is put on the bus.

This will happen at the moment the handshake line $DAV = true$ will be
put on the bus to inform listeners that the data on the bus are valid.
After the next true $\rightarrow$ false transition of the handshake signal $DAV$,
the EOI-part (flipflops 1 and 3) is reset under the strict condition
that $EOI = true$ is on the bus.
Once the EOI-part has been reset, the $IFC = true$ message can be put on
the bus. This message is kept true for 128 microseconds (the IEC-bus
protocol demands that the IFC-message is kept true for at least 100
microseconds).

A possible problem concerning the IFC message could be that this message
is blocked for the maximum time or, in other words, that at the moment
of the arrival of the IFC-message the FIFO-memory is completely full.
This could mean that the uniline message IFC on the transmitting side
of the extension has changed back from true to false again before on
the receiving side of the extension this message has been put on the
bus. This has no serious consequences.
After an IFC-message every device that has to listen or talk in the next
data exchange session has to be addressed first.
This means that the system has to stay in or has to enter the command
mode.
Should the controller on the transmitting side of the extension decide
to activate the command mode, this 'wish' will be received by fode2.
However, as long as IFC = true the node will not return a confirmation message (flag a) to the other node.

As long as IFC remains true the handshake signal NRFD is kept true, and this means that flipflop 5 (part X, fig 5.10) and thus flag a cannot be set.

**ATN-part**

Should the uniline message ATN = true arrive in a frame following a frame with EOI = true, the ATN message will be blocked by the set EOI-flipflop 1.

After this flipflop has been reset the ATN = true message is offered to the rest of the system.
FIG 5-9 / IX Determination time order
not.controller.connected mode

[Diagram of a circuit with various components and connections, including labels such as 'ATN', 'EOL', 'EOL', '1 byte in', 'IFC', 'IFC', and 'not.controller.connected mode'.]
fig 5.9 Determination Time-order Not-controller-connected Fode
5.10 Writing data bytes in the FIFO-memory and reading them out

Source hand-shake part of a fode
part X, fig 5.10 (see § 4.2)

When the system is in the data mode (ATN = false), a data byte that has been received in a frame will be stored in a FIFO-memory. The read and write parts of this memory work independently and can even work in an asynchronous way. This has the advantage that although the bytes are stored in the FIFO-memory at regular times, the listeners can determine themselves (but not independently!) when they want to read out the data bytes from the memory.

The storage of data bytes in the FIFO-memory proceeds as follows. When a complete frame consisting of 24 bits (0 - 23) has been received, this information is transferred from the shiftregisters to the receiving buffers. This transfer will be carried out when the contents of the framecounter is 23. The received information will be considered relevant when flag e = true. When the frame counter contents is 0 (zero) a pulse will be given by OR-gate 9. This pulse will set flipflop 1 and this will make the input-gate PL (parallel load) of the FIFO-memory true. The false → true transition of the output of flipflop 1 will also be used to give a count-up pulse to the FIFO-memory counter. After the FIFO-memory has taken the data byte in its inputregister this data byte will be put on top of the stack. The transfer of the data byte will make the IRF output-gate of the FIFO-memory (output register full) true. This signal will be used to reset flipflop 1 and the procedure can start all over again.
Source handshake in data mode in combination with reading out the data bytes from the FIFO-memory.

The source handshake in the data mode will be handled in a local way. In the command mode the source handshake will be dealt with traditionally, which means that the handshake will be sent via the glass fibre connection.

When a listener is ready to receive data it will make NRFD = false. This will set flipflop 2 and will make the FIFO-memory input signals TOP and EO true (TOP=top of stack, and EO=enable output).

When the FIFO-memory has a data byte in its output register it will make ORE true (output register empty) as soon as this data byte may be regarded as valid. Obviously we will use this ORE signal as the IEC handshake message DAV.

When the data byte has been accepted by the listener this will be shown by NDAC = false. This signal will reset flipflop 2, which will show to it that ORE will be made false, and consequently DAV -> false.

This terminates the source handshake in the data mode. The handshake procedure can start again in order to get a new data byte out of the FIFO-memory.

Source handshake, transition data mode to command mode (see § 4.2)

Before we will look at matters concerning the command mode, we will examine the situation that is created at the moment the uniline message ATN = true is received by the fode via the glass fibre connection.

It is clear that the system on the other side of the extension has changed from data mode to command mode when the uniline message ATN = true is received by the extended fode. Unfortunately fode2 cannot pass this message on just like that.

We will illustrate this by looking at some possible states. Because we have to buffer and sample the incoming lines in the transmitting fode, it may happen that both the last data byte in the data mode and the uniline message ATN = true are in the same frame.

This can only happen when the last data byte is not accompanied by the uniline message EOI = true. If the last data byte had been accompanied by EOI = true, the uniline message ATN = true would have been delayed one frame.

Obviously this last data byte has to be stored in the FIFO-memory as well.

That is why after the PL input gate of the FIFO-memory has changed to true, the output of AND-gate 10 will become true which will set flipflop 3.

This flipflop will block the set possibility of flipflop 1. No new bytes (command bytes) can be stored in the FIFO-memory from now on.

Flipflop 3 will be reset and thus the storage possibility of the FIFO-memory restored, after a true -> false transition of the uniline message ATN.
Now care has been taken of the last data byte. The last data byte can be stored in the FIFO-memory, but that doesn't mean that we can put ATN = true on the bus. We will have to block the uniline message as long as the FIFO-memory is not empty yet. The FIFO-memory counter will inform the concerning parts when the FIFO-memory is empty. When this is the case, ATN = true will be put on the bus, so the entire system is in the command mode now. In this figure and in some others the released ATN message will be called ATN-free.

However, we have to take care not to fall in a trap. It could very well happen that the FIFO-memory is empty at the moment a frame arrives with ATN = true and the last data byte. If we would not take precautions the uniline message ATN = true would not be blocked because the contents of the FIFO-memory counter is 0 (zero). Already we have seen to the fact that this last data byte is stored in the FIFO-memory, but now it will be necessary to see to it that this last data byte can be written out again. We have to prevent the release of the uniline message ATN = true because this message would block the read possibility. Therefore we will do the following.

A valid data byte is accompanied by flag e = true. Flipflop 3 has not been set yet (this can be done only by ATN-free and by PL&ATN). ( & means logical AND) The output of AND-gate 11, in combination with flag e = true and the inverting output-gate of flipflop 3 = true will see to it that ATN = true is blocked at OR-gate 12. During this blockade ATN = true cannot be put on the bus as ATN-free. This blockade will last one frame, which is more than enough time to let the FIFO-memory counter contents change from 0 (zero) to 1. This way the command mode is delayed. This allows the last data byte to be written in the FIFO-memory and to be read out of this memory.
The source handshake in the command mode
In the command mode we will not use the storage possibilities of the FIFO-memory.
The handshake will be handled in a traditional way, which means that the handshake signals will be sent via the glassfibre connection.
Devices connected to the bus will make the handshake signal NRFD = false to indicate to the controller that they are ready to receive a command byte.
This true -> false transition of the handshake line NRFD will set flipflop 5, which will set flipflop 6.
The output of flipflop 6 will be sent as flag a to the other side of the extension. Flag a can be seen as a combined NRFD = false message of the extended devices.
Flipflop 6 will be reset after flag a has been offered to the transmitting frame for more than a frame period.
In this way we can be sure that flag a is transmitted.
Because ATN-free = true, OR-gate 12 is no longer blocked.

When the controller has sent a command byte so that flipflop 4 will be set, the handshake signal DAV = true will be put on the bus.
The connected devices will respond to this signal by making NRFD = true and they will send NDAC = false as soon as they have accepted the offered data byte.
Then flipflop 4 will be reset and the handshake signal DAV will change from true to false.
Because the handshake signal NDAC = false has been received by the node, the flipflops 7 and 8 will be set. The output of flipflop 8 is called flag b, and this flag can be considered as a combined NDAC = false message from the extended devices to the controller.
Now the data byte has been accepted and the controller can start sending another command byte.
Flag b will be reset after it has been offered to the frame for more than a frame period, so we can be sure that it will be sent to the other side.
fig 5.10 Source Handshake
5.11 FIFO-memory counter  
part XI, fig 5.11  (see § 4.2)

This counter counts up when a databyte is written in the FIFO-memory, and of course it counts down when a databyte is read out of this memory.

This circuit reports to the rest of the receiving part of the fode when there is only one databyte left in the FIFO-memory and when the memory is empty.

Another important task of this counter is to see to it that the fode on the other end of the extension is warned when the FIFO-memory is getting full. In that case the data-transport has to be stopped by that fode.

As soon as a count-up or count-down signal is received, flipflop 1 or flipflop 2 will be set respectively.

To prevent the count-up and the count-down signal from being passed on to the actual counter simultaneously, a regulating circuit has been included.

Remark: The reason for this is that read and write functions of the FIFO-memory can be executed at the same time, so that the count-up and count-down signal for the counter can arrive simultaneously.

With the use of the regulating circuit we can see to it that either the write signal or the read signal influences the counter. Once a signal has been dealt with, the representing flipflop is reset and a next signal can be offered to the counting circuit.

As mentioned earlier there are some logical circuits around the actual counter, with the function to indicate the 'state' of the FIFO-memory. The first signal: FIFO-#-empty is important for the release of the uniline message ATN = true.

This release will be effectuated not before the FIFO-memory is empty, so no databytes can be 'trapped' in it.

The second signal 'One-byte-in FIFO-' is important in case this byte was accompanied by the uniline message EOI = true. With this message a talker can indicate that it transmits its last databyte.

Of course the last databyte and the uniline message EOI = true must be rejoined at the right moment.

The information 'last-byte' and 'FIFO-memory empty' will not be released to the system at random moments.

The information cannot be released before the flipflops 4 and 5, with which they are coupled, have been set. These flipflops can only be set when the handshake lines NRFD and DAV are both false.

This means that the information will become available for the rest of the fode system just before a databyte becomes valid.
A last remark concerning the logic that sees to flag d.
Flag d is the signature a node can send to the other node and with which it reports that its memory is getting full. After reception of flag d the node on the other side of the extension will interrupt the data transport by not letting the handshake signal NRFD go false.
Flag d will be sent true (flipflop 3 set) when the contents of the counter is 64, so there is still room for 64 data bytes extra.
Flag d will be sent false again (flipflop 3 reset) when the counter contents has diminished to 32. A kind of hysteresis has been build in.
fig 5.11 Fifo-memory Counter
Conclusions and Recommendations

This report has made it clear, that it is possible to extend a normal IEC-bus.
It was shown that many of the problems encountered can quite easily be solved.
Summarizing we can say, that extending a standard IEC-bus connection to a total length of some 5 kilometres has the following implications:

A) Normal throughput rates in the data mode with a maximum of 1 Mbyte/s.
B) Slower throughput rates in the command mode, which is acceptable when the number of bytes in the command mode is much smaller.
C) Software changes in the controller are necessary when it is desirable to empty the FIFO-memory of fodel. When these software changes are not made, it is possible that data bytes, stored in the FIFO-memory during the data mode, can not get out of this memory and are lost.
D) No parallel poll possibilities for the extended devices.

With respect to the concrete fode design in ECL it is only appropriate to say, that this is only meant as an starting point. It hardly seems possible to produce, sell or use this ECL fode. However the apparent interest justifies a further reseach program, in order to come to a more manageable concept.
To come to this more manageable concept one could think of using fast microprocessors, the new CMOS generation, programmable logic arrays, a custom designed IC etc.

Another remark has to be made with respect to the V-24 interface. Because this interface standard is so frequently used, it is highly recommandable to implement a fode with a V-24 interface possibility. This too is a matter which requires more study: given an IEC-bus fode how can this fode be made suited for both the IEC-bus connections and the V-24 connections.
ACKNOWLEDGMENT

After having stayed in their midst for nine months, I would like to thank all the members of the group Swanenburg of the Natuurkundig Laboratorium of Philips for their time, help, their good-fellowship and their willingly ears.
I like to express a special thank you to ir. Dieleman, who had the difficult task of being my coach during all those months.

At the end of my studies I would also like to thank my wife Ans and my parents for their support during all those years.
REFERENCES


Digital Instrument Course
part 4 : IEC-625

N.V. Philips' Gloeilampenfabrieken
Test and Measuring Department
Eindhoven, The Netherlands

[2] John E. McNamara

Technical Aspects of Data Communication

Chapters 5 and 17
U.S.A., Digital Equipment Corporation, 1977


Datacommunicatie

Chapters 6 and 7, and the appendix
Deventer, Kluwer Technische Boeken, 1982


ANSI MC 1.1-1975|IEEE Std 488-1975
IEEE Standard Digital Interface for
Programmable Instrumentation

The Institute of Electrical and Electronics Engineers INC,
U.S.A., 1975


The Input/Output Primer, Part 3
The Parallel and HPIB (IEEE-488) Interfaces

Byte, april 1982, pp 186 - 208
[6] International Business Machines Corporation

Binary Synchronous Communications
General Information

GA27-3004 File TP-09 IBM, Armonk, New York 10504

[7] International Business Machines Corporation

IBM 3274, 3276 Control unit to device


[8] CCITT, VIIth Plenary Assembly

Data Communication over the Telephone Network
Recommendation of the V series

Yellow Book, Recommendation V 24
Volume VIII - Fascicle VIII,7
Geneva, 1981

[9] S. Leibson

The Input/Output Primer, Part 4
The BCD and Serial Interfaces

Byte, May 1982, pp 202 - 220

[10] California Computer Systems

CCS Model 7710
APPLE II Asynchronous Serial Interface
Owner's manual

U.S.A., California Computer Systems, Appendix C


APPLE II Super Serial Card
Installation and Operating Manual

U.S.A., California Computer Systems, page 6
APPENDIX 1

Biasync protocol and V-24 interface

As mentioned in the introduction the IEC-bus interface standard, the V-24 interface standard and the biasync protocol have been examined. The IEC-bus has been dealt with thoroughly in the previous chapters. The two remaining systems will be looked at in this appendix.

A1.1 Biasync protocol

Let us consider the Biasync protocol first [2],[3],[6],[7]. The biasync protocol is, as mentioned before, a description of what happens between an IBM computer and its terminals with respect to data transport. The computer is linked to its terminals via coaxial cables and the protocol is a half duplex one. Despite many efforts it was impossible to get the exact information about the protocol used for the datatransport by the system installed at Philips Research Laboratories (Nat.lab). Neither at the Nat.lab. nor at IBM Nederland the necessary information could be given.

At the Nat.lab., in the group of dr. Severin, the same problem was faced by a student of the local CAT in 1983. This student was able to convert the electrical signal in a light signal and vice versa, but it was impossible for him to exceed a total length of 1.5 kilometres. This length of 1.5 kilometres is approximately the maximum length for which IBM guarantees correct data transport.

Due to the lack of information we were obliged to drop this item.

A1.2 V-24 or RS 232C interface

The second system we are going to discuss is the V-24 interface which is practically identical to the RS 232C interface [2],[3],[8],[9]. In this case we are dealing with serial data transport. Figure A1.1 shows the connector that is used for this kind of interface. It also shows the functions of the connection pins.

The problem that arises here is that we cannot strictly speak of the V-24 interface standard when this interface is used to connect a computer directly with a peripheral device. The standardised interface is meant to be used as an interface between a so-called DTE = Data Terminal Equipment and a DCE = Data Circuit terminating Equipment. See figure A1.2a
This means that the interface according to the standard should be used to connect a computer or peripheral device (DTE) on one the side with a modem (DCE) on the other side. Of course the originally meant function of the V-24 interface is still used, but very often the V-24 interface is misused, so to speak, to connect a computer directly to a peripheral device. Using the parlance of this standard, the V-24 interface is frequently used to interconnect DTE's. See figure A1.2b.

The logical consequence of this is, that the connections can no longer be easily made by connecting the pins with each other [10],[11]. Some connection pins will have to be cross-connected, for instance, the pin "transmitted data" of a device has to be connected with "received data" from another device. In normal situations those cross-connections will not be necessary because they are made in a modem.

Another possible source of difficulties is that not every connection pin has to be used. It is often sufficient to use a certain subset, but which subset has to be used depends on the devices to be connected. See figure A1.3. It is also quite common that different devices have different interpretations of the same functions of the connector pins.

We will illustrate the reason for this with the following example [9]. The official standard V-24 only uses half a hardware handshake. A data transmitting device reports to the modem with which it is connected directly that it wants to transmit data. After that a data receiving device will be asked by its modem whether or not it is ready to receive data. Once a data receiving device has answered affirmatively, data transport can start. From that moment on, however, according to the official standard, the receiving device is no longer able to influence the data transport using hardware signals. This means that a device cannot stop the stream of data using hardware signals, whether it can handle this stream or not. In practice, however, several methods are used to implement a full hardware handshake in case two DTE's are interconnected. The only way that this can be done is by redefining a certain function or certain functions, even though this is in conflict with the official standard rules.
The conclusion we must draw from this example is, that the V-24 interface is used in a rather messy way. Because of that, it was decided upon to extend every pin, without processing them. Comparing with the IEC fode this means, that the V-24 fode will not handle matters in a local way or use (elastic) buffers. A transmitting V-24 fode will sample the pins and serialize these samples. A receiving V-24 fode will deserialize the offered signal and it will see to it that the (extended) pins get the right sample values. We can say, that every pin connection (with the exception of ground connections) will be extended one to one. Furthermore the fode will make no cross connections, so it can not be considered to be a so-called null modem.

Because of the limited time there was no possibility to go deeper into details concerning the V-24 interface. Still it seems only appropriate to remark that it is highly recommendable to implement this interface as one of the possibilities in a general fode
figure A1.1 Connector of V-24 interface
### FIG A1.1

<table>
<thead>
<tr>
<th>Direction</th>
<th>Name</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>To DCE</td>
<td>Secondary Transmitted Data</td>
<td>To DCE</td>
</tr>
<tr>
<td>To DTE</td>
<td>Transmit Clock</td>
<td>To DTE</td>
</tr>
<tr>
<td>To DTE</td>
<td>Secondary Received Data</td>
<td>To DCE</td>
</tr>
<tr>
<td>To DTE</td>
<td>Receiver Clock</td>
<td>To DCE</td>
</tr>
<tr>
<td></td>
<td>Unassigned</td>
<td></td>
</tr>
<tr>
<td>To DCE</td>
<td>Secondary Request to Send</td>
<td></td>
</tr>
<tr>
<td>To DCE</td>
<td>Data Terminal Ready</td>
<td></td>
</tr>
<tr>
<td>To DTE</td>
<td>Signal Quality Detect</td>
<td></td>
</tr>
<tr>
<td>To DTE</td>
<td>Ring Detect</td>
<td></td>
</tr>
<tr>
<td>To DCE</td>
<td>Data Rate Select</td>
<td></td>
</tr>
<tr>
<td>To DCE</td>
<td>Transmit Clock</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Unassigned</td>
<td></td>
</tr>
</tbody>
</table>

- 0: Earth Ground
- 10: Transmitted Data
- 15: Received Data
- 16: Request to Send
- 17: Clear to Send
- 18: Data Set Ready
- 19: Logic Ground
- 20: Carrier Detect
- 21: Reserved
- 22: Reserved
- 23: Unassigned
- 24: Secondary Carrier Detect
- 25: Secondary Clear to Send

V-24 or RS 232C Connector
figure A1.2 DTE-DCE and DTE-DTE connection possibilities
Explanation of figure A1.3

Figure A1.3 shows a scheme in which several printers (1 to 15) are connected to an APPLE II serial card, or super serial card [10],[11]. The connections which have to be made, are regarded from the side of the RS 232-C connector of the printer.

In this scheme this means that

- pin 3 of the connector of the printer should be connected to pin 3 of the APPLE card,
- pin 7 of the connector of the printer should be connected to pin 7 of the APPLE card, and
- pin 20 of the connector of the printer should be connected to pin 4 of the APPLE card.

The jumpers in the scheme indicate the interconnections of pins of the RS 232 C connector.

Above the dotted line the jumper refers to the interconnection of the pins of the APPLE card. For instance printer number 7: the pins 6 and 20 of the RS 232 C connector of the APPLE card are interconnected.

Under the dotted line the jumpers interconnect the pins of the connector of the printer. For instance printer number 12: the jumper interconnects the pins 6, 8 and 20 of the RS 232 C connector of the printer.

In case of printer 6 there is an option connection of pin 4 of the APPLE card. If the operation at 300 baud is acceptable, pin 4 of the connector of the printer can be connected to pin 4 of the APPLE card; when this low baud rate is not acceptable pin 11 of the printer should be connected to pin 4 of the APPLE card.

The printers looked at are:

1 = IDS 560 Paper Tiger
2 = NEC 5510 Spinwriter
3 = NEC 5510 Spinwriter
4 = Anadex Printers
5 = Anderson Jacobson Printers
6 = Centronics Serial Interface Printers
7 = C-ITOH Printers
8 = DECwriter
9 = Diablo 630
10 = EPSON MX Series
11 = IDS Paper Tiger
12 = NEC Spinwriter
13 = Qume Printers
14 = Talley 1612
15 = Texas Instruments TI800 Series
figure A1.3 APPLE II - Printer connections RS 232 C
### FIGA13  APPLE II PRINTER CONNECTIONS  RS.232.C

<table>
<thead>
<tr>
<th>Pin Number</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS 232C Connections</td>
<td>( \Rightarrow )</td>
<td>sign 0</td>
<td>( \Rightarrow )</td>
<td>20</td>
<td>( \Rightarrow )</td>
<td>( \Rightarrow )</td>
<td>20</td>
<td>9</td>
<td>20</td>
<td>( \Rightarrow )</td>
<td>( \Rightarrow )</td>
<td>( \Rightarrow )</td>
<td>( \Rightarrow )</td>
<td>( \Rightarrow )</td>
<td>( \Rightarrow )</td>
<td>( \Rightarrow )</td>
<td>( \Rightarrow )</td>
<td>( \Rightarrow )</td>
<td>( \Rightarrow )</td>
<td>( \Rightarrow )</td>
<td>( \Rightarrow )</td>
<td>( \Rightarrow )</td>
<td>( \Rightarrow )</td>
<td>( \Rightarrow )</td>
</tr>
</tbody>
</table>

- \( \Rightarrow \) depends on baud rate
- example: number 7 pin 3 printer connected to pin 8 apple card
- pin 20 jumper above dotted line interconnects RS 232C pins apple card
- under --- printer
APPENDIX 2

Time diagrams belonging to the frame counters
of the IEC-bus fode

The figures of this appendix are the time diagrams of some
parts of the IEC-bus fode.

Figure A2.1 : Time diagram of 'Frame counter transmitting part of
the fode' with respect to the AMI encoding.

Figure A2.2 : Time diagram of 'Frame counter transmitting part of
the fode' with respect to the handling of data or
command bytes.

Figure A2.3 : Time diagram of 'Frame counter receiving part of fode' .

List of used abbreviations

OR or/nor gate
AND and/nand gate
XNOR exclusive nor gate
C counter
SR shiftregister
BUF buffer
SET set-reset flipflop
JK JK flipflop
Figure A2.1 Frame counter transmitter, AMI encoding
latest possible time instant that an output of a device can change.

earliest possible time instant that an output of a device can change.

SR1
AND2
SR3
Q3
SR3
Q3
SR3
Q3
XNOR11
SR12
OR10
BUF13

AND7
DR4

SET5
AND7
DR4

SET5

SET5

SR1

52 ns
Figure A2.2  Frame counter transmitter, byte loading
Figure A2.3 Frame counter receiver
latest possible time instant that
an output of a device can change.

earliest possible time instant that
an output of a device can change.
APPENDIX 3

Throughput rate - data rate

In this report we have used the term 'throughput rate' instead of the term 'data rate'. This has been done to avoid misinterpretation. With throughput rate we mean the rate with which a talker or controller transmits its data. One could say that we pay attention to one point, for instance point A, and look how many bytes pass this point within a certain time; the quotient (number divided by time) stands for the throughput rate. One the other hand the definition of the data rate is different. When we assume perfect devices (receiving devices accept the offered data immediately) we can define the data rate as follows: The data rate is time starting from the moment the talker transmits its first data byte up until the moment the listener receives the last data byte, divided by the total number of bytes.

\[
R = \frac{N}{T + \tau + N \cdot t_b}
\]

In formula: \( R \) = data rate
\( T \) = total time starting from the moment the talker or controller transmits its first byte, until the moment the listener receives the last byte.
\( N \) = total number of bytes
\( \tau \) = total delay time due to the extension (including the fall through time of the FIFO-memory)
\( t_b \) = byte time
\( \frac{1}{t_b} \) = throughput rate

This means that the data rate approaches the throughput rate for large numbers of bytes. For small numbers of bytes the influence of the delay time cannot be neglected, and the data rate will be much lower than the throughput rate.

See figure A3.
Figure A3 Data rates as a function of number of bytes in one block.
APPENDIX 4

Possible fode interconnections

Throughout this report we have assumed that the two fodes are interconnected by two glass fibres. One glass fibre is used by a fode to transmit information, the other is used to receive information. Of course there are other ways to interconnect the two fode, for instance using only one glass fibre. Using only one glass fibre leaves us with a number of possibilities. The first possibility is based on the idea of shuttling frames, the second is based on the use of an optical hybrid and light of only one wavelength, and the third possibility also uses an optical hybrid but now colour multiplexing (also called wavelength multiplexing) is applied. The method of shuttling frames between the two fodes is not suited of our application. The idea is that a fode transmits a frame to the other fode, but is not allowed to transmit a new frame before the frame of the other fode has been received. This reduces the maximum allowable throughput rate to an unacceptable low level. For instance in case of a link of five kilometres, the delay due to the glass fibre extension is approximately 25 microseconds. This means that a fode can transmit its information once every 50 microseconds (assuming that the fodes are infinitely fast), which comes down to a maximum throughput rate of 20 kbytes/s.

The second method uses an optical hybrid and light of only one wavelength. See figure A4.a

Here problems can arise with respect to cross talk effects due to reflections of of the transmitted signal. The third possibility (optical hybrid and colour multiplexing) is a good alternative for the method used in this report. See figure A4.b. The choice between the two possibilities will have to be based on other grounds than the technical grounds used in this report. In some cases it will be more efficient (and cheaper) to use two fibres, in other cases the other possibility may prove advantageous.
Figure A4 Optical hybrid with and without colour multiplexing
FIG A3.

\[ R = \frac{1}{\tau/N + t_b} \]

**Variables:**
- \( R \): Data rate (s\(^{-1}\))
- \( N \): Number of bytes
- \( \tau \): Delay time (s)
- \( t_b \): Byte time (s)
- \( 1/t_b \): Throughput rate (s\(^{-1}\))

**Graph:**
- The graph shows various curves labeled a, b, c, d, e, f, g, h, i, each corresponding to different values of \( (x \times 10^{-6}) \tau \) and \( t_b \).

<table>
<thead>
<tr>
<th>( \tau \times 10^6 )</th>
<th>10</th>
<th>25</th>
<th>50</th>
<th>10</th>
<th>25</th>
<th>50</th>
<th>10</th>
<th>25</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_b \times 10^{-6} )</td>
<td>10^{-4}</td>
<td>10^{-4}</td>
<td>10^{-5}</td>
<td>10^{-4}</td>
<td>10^{-5}</td>
<td>10^{-4}</td>
<td>10^{-4}</td>
<td>10^{-4}</td>
<td>10^{-4}</td>
</tr>
</tbody>
</table>
optical hybrid using light of one wavelength

optical hybrid using light of two wavelengths

R receiver
T transmitter
WDM wavelength division (de-)multiplexer