310/1779-8

Fourth Workshop on Distributed Laboratory Instrumentation Systems
(30 October - 24 November 2006)

Industrial Networking Systems

Anthony J. WETHERILT
UNIDO-ICHET
Sabri Ulker sok, 38/4
Cevizilbag, Zeytinburnu
34015 Istanbul
TURKEY

Industrial Networking Systems

James Wetherilt[*]

UNIDO-ICHET
Sabri Ulker Sok, 38/4,
Cevizlibag, Zeytinburnu
34015 Istanbul
Turkey

Supporting material for the
lectures given at:
The Fourth Workshop on
Distributed Laboratory Instrumentation Systems
Trieste, 30 October – 24 November 2006

LNS

[*]jwetherilt@unido-ichet.org

Abstract

Until recently, industrial networking relied heavily on serial based systems for communication. The physical layers of several such systems are discussed and communication protocols presented for Modbus, Profibus, Fieldbus and CANOpen. Industrial Ethernet and bus systems making use of them are presented. The use of OPC servers as universal interfaces for SCADA systems is introduced with reference to the OPC Data Access Specification.

Contents

1 Introduction

Throughout most of this workshop we will have been talking about the various methods of networking using the full power of TCP/IP running over Ethernet or similar physical systems. With modern techniques and hardware what can be done with such systems is limited basically only by one's imagination and experience. Nowadays everyone appears to need TCP/IP and Ethernet in one form or another and it should thus be found wherever intelligent sensors provide data for analysis systems. The hard facts of life, however, do not entirely bear this rosy picture out: there are large data processing wastelands where neither TCP/IP nor Ethernet can be found and other lower performance networking systems are routinely employed for control and monitoring purposes. Industrial plants, in Europe at least, until recently, traditionally have rarely had Ethernet installed: it is almost entirely confined to office use and seldom ventures onto the factory floor. Yet monitoring and control are vitally important fields for the process engineer. No industrial plant of even moderate scale could perform without the so called SCADA (Supervisory Control And Data Acquisition) and DCS (Distributed Control Systems) software monitoring systems showing the status of critical process components.

This chapter reviews the basics of industrial networking solutions and discusses some of the protocols used for industrial process control. We will begin by discussing the physical bases of serial communications used by common protocols. Next we will discuss the systems most commonly used in practice and show how their protocols work. Industrial Ethernet and the recent extensions of the common protocols to Ethernet will also be discussed. Finally, we will discuss briefly attempts to provide a common interface between SCADA systems via the OPC (OLE[1] for Process Control) standard.

2 Serial communications

2.1 RS 232

Since the early days of computing and more recently, the advent of the PC, the serial communications port has been virtually obligatory on every machine. The RS 232 standard of the Electronics Industry Association (EIA) is probably the most well-known and most implemented standard in the history of the computer. It is also probably the most non-standard 'standard' as every implementation seems to differ from the next. The standard specifies a pair of data transmission lines (RxD, TxD) together with a common return 'ground'. Additionally, there are several control lines intended primarily for modem handshaking control. In its simplest form therefore, RS 232 communications can be achieved with just the two data transmission lines and common return. There are however additional complications that make this simple case difficult to implement. Firstly, equip-

[1]Object Linking and Embedding

ment using RS 232 comes in two varieties: Data Communications Equipment (DCE) such as modems and Data Terminal Equipment (DTE) such as PCs, printers etc. If a DCE talks to a DTE then all is fine and dandy. Just connect the two pieces of equipment pin to pin (i.e. RxD – RxD, TxD – TxD, and common – common) and off you go. If, however, you wish to connect a DCE to a DCE or a DTE to a DTE, a 'Null modem' cable that reverses the connections between pairs of cables (i.e. TxD – RxD in each case) has to be used[1].

Even when you have figured out what type of equipment it is, the three-line connection may not work because of the logic levels required on the handshaking lines by serial port adaptors themselves. Some of these can be disabled in software but it is common practice in three wire connections to 'loop back' certain handshaking lines on the connector to provide the necessary logic levels. A fully working implementation of the standard between two DTE devices requires some two extra wire pairs over the simple approach. Data Terminal Ready (DTR) connected to Data Set Ready (DSR), and Request To Send (RTS) connected to Clear To Send (CTS), with the Data Carrier Detect (DCD) pins tied to DSR in each connector.

Voltage levels are bi-polar with +(3 to 12 V) indicating ON (or SPACE in the terminology) and -(3 to 12 V) OFF (or MARK). The dead area in between these two states is intended to absorb line ground noise. A typical serial driver/receiver chip or UART (Universal Asynchronous Receiver Transmitter) can transmit and receive at different internal clock frequencies. The only requirement being that the clocks at both ends of the line agree on the same frequency. The RS 232 standard specifies a maximum transmission rate of about 20 Kbits/s at a cable length of 15 m. At lower frequencies, longer cable lengths are possible. Use of high quality, screened and twisted pairs for the various line pairs can improve both the maximum cable length and the upper frequency but in general as a result of the single ended cable with a common ground return, noise is always a problem. Also connecting two remote ground systems often results in the flow of potentially large ground currents further affecting the noise susceptibility.

The RS 232 standard is intended primarily for asynchronous communications where each device has its own (un-synchronised) clock. Provision is made, however for clocked synchronous communications in the full 25-pin connector, which is described in table 1 on page 3.

2.2 RS 422/449

To overcome the problems inherent with EIA RS 232 communication techniques, the EIA RS 422 electrical standard was developed. RS 422 uses balanced differential drivers (labelled A or - and B or +) for transmission of data over twisted pair cables. Any externally developed noise is induced equally onto both wires of the twisted pair and hence cancels out. Data rates of up to 100 Kbits/s at

[1]The problem is knowing, in advance, what type of device a given piece of equipment thinks it is. A simple method involves measuring the voltages between the signal ground and TxD or RxD. If $V_{\mathrm{TxD}} < -3$ V then the device is a DTE; conversely if $V_{\mathrm{RxD}} < -3$ V then the device is a DCE.

Pin no	Function	
1	Frame ground	
2	Transmitted data	TxD
3	Received data	RxD
4	Request To Send	RTS
5	Clear To Send	CTS
6	Data Set Ready	DSR
7	Signal ground	
8	Received line signal detect	
9	Secondary Clear To Send	
6	Data Set Ready	DSR
7	Signal ground	
8	Received line signal detect	
9	Secondary Clear To Send	
10	Received line signal detect	
11	Undefined	
12	Secondary Received line signal detect	
13	Secondary Transmitted data	
14	Secondary Clear To Send	
15	Secondary Transmitted data	
16	Secondary Received data	
17	Receiver Signal element timing	
18	Undefined	
19	Secondary Request To Send	
20	Data Terminal Ready	DTR
21	Signal Quality Detector	
22	Ring Indicator	RI
23	Data Signal Rate Selector	
24	Transmitter Signal Element Timing	
25	Undefined	

Table 1: RS 232 — 25-pin connections

distances of up to 1200 m are specified in the standard. RS 422 relies on the RS 449 mechanical standard for connector specifications. A 37-pin connector is described, but allows 9-pin connectors on slave units. In practice however, the only connections needed are the data lines and a shield.

Multi-drop applications where one driver is connected to up to ten receivers are accommodated by the standard but true multi-point applications, where multiple drivers and receivers exist tied to the same bus, are not. Quasi multi-point networks on the other hand can be constructed using two pairs of data lines (the so called 4-wire connection – see figure 1 on page 6). These networks are often used in what is called half-duplex mode where a single master sends commands to multiple slaves each listening in on the master's transmission line pair. When a slave recognises that a command is intended for itself, it responds by sending data back on the second line pair attached to each slave's transmission output. It is important that data collisions are avoided by ensuring that only one device responds at a time as typically all handshaking signals are ignored for simplicity and cost.

In contrast to RS 232, RS 422 uses the voltage levels 0 and +5 V with the differential voltage between the A and B lines being important. $\mathrm{V_{AB}} < -0.2\ V$ is defined as OFF and $\mathrm{V_{AB}} > 0.2\ V$ as ON, while a value between these limits is undefined.

2.3 RS 485

The next instance in our discussion of bus types is the backbone of modern industrial communications. Like the RS 422 standard it uses balanced differential drivers and receivers to achieve low noise over long distances at high data rates. Its main difference is that whereas RS 422 specifies that the output driver is always active, RS 485 allows the output driver to be activated only during actual data transmission and disabled at all other times, by tri-stating (employing a third, high impedance output state in addition to the usual low and high states) the output. This allows transmission and reception between multiple devices over a two wire twisted pair bus (plus an additional signal ground). As before bus contention must be avoided by ensuring that only a single device is active at a given time. This is generally achieved in software by the use of a designated bus master, which decides which device is to talk.

The voltage states on the differential lines can be one of the three states: ON; OFF; and High Impedance. When all drivers are in the high impedance state, the voltage on the data line can drift to values in excess of the safe common mode limits, and bias resistors should be used to bring the idle voltages to safe levels without seriously loading the system at other times. A pull up resistor to 5 V and a pull down resistor to ground should be attached to the B and A terminals respectively. The exact value of the resistors depends on many factors including the cable length, number of attached devices and so on.

At high data transfer rates or long cable lengths problems with reflections can occur if the lines are not terminated correctly. We can determine the maximum data rate an unterminated line can accept for a given set of conditions[1]. Correct termination can be achieved by placing a resistance greater than or equal to (over damping is generally preferred to under damping) the characteristic impedance of the line between the two members of the twisted pair at each end of the cable. It is not necessary to terminate the individual tapped nodes. Together with the biasing resistors this technique is known as power termination.

The input impedance of each receiver is specified as being at least 12 kΩ. At this impedance a minimum of 32 devices can be attached to the network segment. If more than 32 devices are required, line repeaters must be employed at suitable intervals. Alternatively, multiport serial adaptors can be used in the host PC to set up multiple segments with up to 31 attached devices in each segment. In this manner literally hundreds of devices can be attached to the network and monitored from a single machine.

A common addition to RS 485 bus systems is a layer of optical isolation between the transceivers and the device. Industrial equipment can produce large electrical pulses from the rapid switching of high currents and despite all other precautions these pulses often find their way on to the twisted pairs where they manifest themselves as large voltage spikes of very short duration. Such spikes are often fatal to microprocessor based instrumentation causing resets, corrupted data and other problems. Optical isolation can reduce the effects of these pulses to manageable proportions allowing devices to function under electrically aggressive environments.

2.4 Communicating over a RS 485 network

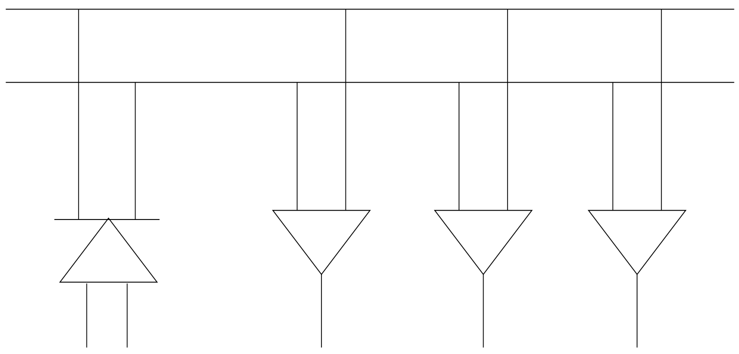
Having described the physical medium used in industrial networks, we will now examine a simple protocol for data transmission over a multipoint line. The protocol illustrates some of the principles needed when performing serial communications. In subsequent sections we examine some of the commoner 'standard' protocols available over the RS 485 physical layer. Non-standard protocols are popular with device manufacturers in that they are generally simple and both easy and cheap to implement and can be tailored to fit the needs of their device.

2.4.1 Common elements of address based communication protocols

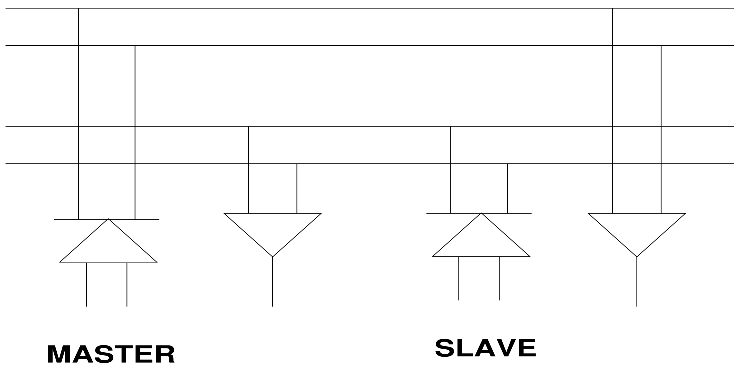
Many protocols are forced by the nature of the problem to contain similar elements, such as a byte to indicate the start of a command, the address of the

[1]We assume that any reflection will damp down after a few round trips and that as the voltage will be sampled in the middle of the pulse it must be stable at that point. We therefore look at the rise time caused by the propagation delay. This is calculated as 2*L*n/v, where L is the total length of cable, n the number of round trips, and v the propagation speed (obtainable from the cable manufacturer and typically about 66% of the speed of light). If this value is comparable to slightly less than half the pulse width we need termination. For example, at 19200 Baud, one bit is 52 μs wide and the signal should be stable appreciably before half this value, say 16 μs. If we assume that after three round trips the reflections are completely damped, the above formula gives the maximum cable length without terminations as about 530 m.

a. Two wire multidrop (1 transmitter-many receivers)



b.Four wire quasi multipoint



c.Two wire multi-point with RS 485

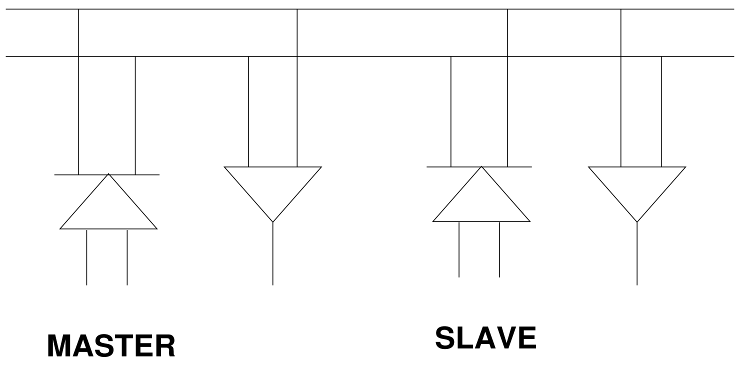


Figure 1: 2 and 4 wire-balanced connections

device to receive the command (as well as the address of the sender in a true multipoint network), the command itself, any data needed by the command and finally some form of checksum for ensuring that the command packet does not get scrambled en route to the receiver. In some cases the number of bytes in the command is fixed, in others the size will depend on both the nature of the command and any data associated with it. Many protocols start with the ASCII character "Start of Text" (STX) that has the value 2. Depending on the total number of devices that need to be addressed, the address field will require 1 or two bytes. As a total of 256 devices is often sufficient, we will use one byte for this purpose. We will also assume an arrangement in which one device (typically the host PC) is designated as the master. In this case we do not need to include the sender's address, which we can set to a reserved value (typically 0 or 255). For simple devices, the number of commands will rarely exceed 256 and so one byte can be reserved for this purpose. We can also assume that the device can work out how much data will be sent from the command and it is therefore not necessary explicitly to reserve a field for this value. Finally, we must indicate a checksum value that can be constructed from the other bytes in the packet and reconstructed again by the receiver to check the validity of the packet. There are many ways of constructing this checksum but we will use the simple method of forming the "exclusive OR" of the previous bytes in the packet and using this as the checksum. The receiver will take the packet and again form the "exclusive OR" of all its bytes. If this value is not zero, the packet has been corrupted and should be sent again. A typical command packet structure is shown in figure 2.

STX	ADDRESS	COMMAND	DADA1	DATA2	. . .	DATAn	CHECKSUM

Figure 2: Command packet structure

A good protocol should also specify the response of the receiver to each command. A satisfactory way of doing this is to echo the first three fields (STX, ADDRESS, COMMAND). In this case the address field is the address of the responding device, and COMMAND is the sent command if it was received satisfactorily or a separate command indicating a receiver error. Again, data may be sent with the packet and the checksum added as the final byte.

Protocols such as this are simple to implement and are sufficient for remarkably sophisticated devices. The actual protocol described here has been used for multidrop master-slave communications in several instruments.

2.5 Controller Area Network (CAN)

The Controller Area Network or CAN is a networking system originally introduced by the Bosch company in 1986 for communications between individual electronics components of automotive systems. Almost every car produced now contains at least one CAN network and it is estimated that annual sales of CAN devices were more than 100 million dollars in the year 2000. CAN interfaces are now common on many devices as second or third serial ports and its use extends

Function	
CAN_L (CAN-)	Dominant Low bus line
CAN_H (CAN+)	Dominant High bus line
CAN_GND	Common ground return
(CAN_SHLD)	Optional shield
(GND)	Optional power ground
(CAN_V+)	Optional power supply

Table 2: CAN bus line definitions

well beyond the original automotive applications. It is managed by the CAN in Automation (CiA) [1] organisation who promote and manage the standard as well as providing considerable information to potential vendors and developers.

In contrast to the RS 232 and RS 485 based protocols discussed previously which are either point to point or require a separate address for each device on the bus, a CAN device sends messages each with a unique identifier. Each unique message identifier is intended for a specific device or group of devices. CAN defines a minimum of 3 data lines for communications used as a differential pair and a common return (see table 2). Other optional lines are used for screening and powering of opto isolators. The CAN_L and CAN_H lines are connected to the inverting and non-inverting terminals of the differential inputs respectively resulting in a negative level when $\mathrm{CAN\_L} > \mathrm{CAN\_H}$ and a positive level otherwise. The differential driver outputs are arranged so that the logical result of two or more values is the logical AND of the values. Since the logical AND of 0 with any other value is always 0, the 0 state is said to be *dominant* whereas the 1 state is *recessive*. Cable lengths are specified up to 1000 m at a maximum baudrate of 50 Kbits/s. At higher baudrates, the maximum cable length is reduced considerably.

3 Common industrial bus systems

There exist numerous bus systems in use throughout the industrial world. Some of the properties of these systems are summarised in table 6, on page 23. The choice is bewildering, but in practice, only a few systems are worth describing: Modbus [2], Profibus [3], Fieldbus [4] and CANOpen [1]. The first of these, Modbus has an open specification and is commonly used in many systems especially motor control devices such as inverters. Since it has an open architecture, it can be used without royalties or other fees and PC port adaptors are widely available from various manufacturers. Profibus, on the other hand was developed primarily by Siemens for use in its own products and then released to the community at large. It is now governed by a European standard and its use regulated by the Profibus Foundation, which exists to further the use of Profibus, to provide information concerning the availability of products containing Profibus interfaces and to provide hardware for product manufactures. The Fieldbus Foundation performs a similar role for the Fieldbus. Whereas Profibus is almost exclusively European, Fieldbus is the emerging standard in North America and any company seeking to sell its products in that region had better seriously consider its implementation. All three systems (Modbus, Profibus and Fieldbus) have organisations active in embedding their technologies in TCP/IP over Industrial Ethernet.

The final modern fieldbus to be discussed is CANOpen and is based on the CAN system described earlier.

Since each of these systems is a topic in its own right, in this chapter we will give only an overview of the important points in each case.

3.1 Modbus

Modbus uses a twisted pair connection for multidrop communications with one master and up to 247 slaves. All communications are initiated by the master and devices talk only in response to a master command. The Modbus protocol establishes a format for all commands and responses and each packet contains the desired device address, the command code, any data needed by the command and an error-checking field. If a device receives a command that either contains an error or in executing that command the device is in error, the device can construct an error response containing relevant information.

Modbus uses two distinct modes of communication: ASCII mode or Remote Terminal Unit (RTU) mode. The mode is selected along with the other communication parameters (Baud rate, parity, data bits, stop bits, etc) during initial configuration. It is essential that all devices are configured in the same way as otherwise the entire network will probably fail.

3.1.1 ASCII mode

In this mode, each eight-bit byte in a message packet is sent as two ASCII characters. The main advantage of this mode is that it allows delays between individual characters within the message packet of up to one second without error. The message data is sent as the hexadecimal digits 0-9, and A-F with each digit sent as one ASCII character. One start bit followed by 7 data bits, an optional parity bit and finally one or two stop bits depending on whether parity was selected, make up each transmitted character. The format of each packet is shown in figure 3.

Start	Address	Command	Data 1	...	Data n	LRC	End
:	2 chars	2 chars	2 chars	2 chars	2 chars	2 chars	CRLF 2 chars

Figure 3: ASCII mode packet

The start of each packet is always signified by the ‘:’ character and the end with a carriage return-line feed pair. The longitudinal redundancy check (LRC) is calculated as follows. The values of each byte of the message (excluding both the starting colon and the terminating CRLF pair) are added together without carry and then two's complemented to form the result.

3.1.2 RTU mode

In RTU (Remote Terminal Unit) mode each message frame must be preceded by a period equal to at least 3.5 characters at the selected baud rate. Similarly, if more than this period of time elapses between successive bytes transmitted during a message, the packet is assumed to be finished. On the other hand, if a period less than this value occurs between successive packets, the two packets are considered as one. This gives the packet structure that is shown in figure 4.

Start	Address	Command	Data 1	...	Data n	CRC	End
4 silent chars	8 bits	8 bits	8 bits	8 bits	16 bits	4 silent chars	

Figure 4: RTU mode packet

The transmission of each byte consists of one start bit, eight data bits, an optional parity bit and one or two stop bits (depending on whether parity was selected). The characters are transmitted as two digit hexadecimal values within the eight bits of each field.

Calculation of the Cyclical Redundancy Check (CRC) used in RTU mode is a much more complicated affair than for ASCII mode. First a sixteen-bit register is loaded with ones. Then each data byte is first exclusively OR'ed with the register and the contents of the register shifted right towards the least significant bit (LSB), zeroes being padded in the most significant bit. If the LSB is one, the register is again EOR'ed with a preset value. The process continues until eight such right shifts have occurred. Successive data bytes are then subjected to the same process to obtain the final CRC.

3.1.2.1 Command response Following receipt of a legal command, a slave will respond with its address and will echo the command in the packet. Any required data will also be included. Allowable commands are values in the range 1-127. If an error occurs as the result of an illegal or improperly executed command the slave will set the most significant bit of the command field. In this case the data in the data field will convey extra information concerning the type of error that occurred.

3.1.2.2 Broadcasting The master can broadcast to all slaves by setting the address field to zero. Any slave receiving such a broadcast will perform the required action but will not respond for obvious reasons. In particular this means that error responses will not occur with broadcasts.

3.1.2.3 Data and control functions Modbus defines a number of data and control functions intended to permit a variety of simple instruments and devices to be controlled by a uniform command set. These are summarised in table 3.

Function No	**Function name**	**Description**
01	Read coil status	Reads the ON/OFF status of discrete outputs such as relays or other logic devices
02	Read input status	Read ON/OFF status of discrete inputs
03	Read holding register	Read the binary contents of the output registers (16 bit location) between specified offsets
04	Read inputs registers	Reads the binary contents of the input (4x) registers between specified offsets
05	Force signal coil	Force the specified single discrete output device to give ON/OFF state
06		Load the specified (4x) register with a present value
07	Read exception status	Reads a set of eight status bits. Certain bits are predefined but some are device dependent
08	Diagnostics	Get diagnostic information
11	Fetch communication event counter	Fetch a status word and the count of communications events. This allows a master to determine whether a command was handled correctly
12	Fetch common event log	Fetch a status word, a message count and a array of messages for diagnostics
15	Force multiple coins	Set the logic states of a series of discrete outputs
16	Presets multiple registers	Loads a sequence of registers with preset values
17	Report slave ID	Return device dependent information from a slave
20	Read general reference	Read from an extended set of register
21	Write general reference	Read from an extended set of register
22	Mask write 4x register	Modify a general register using logical operations
23	Read/write 4x register	Read and write a specified register in a single operation
24	Read FIFO queue	Read from a set of queued FIFO registers

Table 3: Predefined Modbus commands

3.2 Profibus

Profibus is a considerably more complex system than Modbus, and is a more modern attempt to address the problems of device interoperability. It was designed using the Open System Interconnection (OSI) model shown in Figure 5 and utilises layers 1, 2 and 7 of that model.

Layer 7	Application layer (FMS, PA)
Layer 6	Presentation layer
Layer 5	Session layer
Layer 4	Transport layer
Layer 3	Network layer
Layer 2	Data link layer
Layer 1	Physical layer

Figure 5: The ISO/OSI Reference Model

At the bottom of the model, the Physical layer describes the physical characteristics of the transmission medium. Most frequently this is the RS 485 technology discussed earlier, but also can be fibre optic technology. When used with RS 485 cabling Profibus uses four wires: a single twisted pair for data transmission and one wire for ground return and a second wire at +5 V. All are shielded. Power termination is used at both ends of each segment, which can have a maximum of 32 devices attached to it. Profibus RS 485 cable definitions and terminations are shown in figure 6. Repeaters are used to connect multiple segments together up to a total maximum of 127.

Communication rates between 9.6 KBaud and 12 Mbaud make data transfer fast and efficient.

Three varieties of Profibus exist: Profibus DP, Profibus FMS and Profibus PA, respectively for high speed data communication in automation equipment, object oriented general purpose communications, and process application areas where intrinsic safety may be an issue. Approximately 90% of Profibus applications at this time are Profibus DP which uses only layers one and two of the OSI model (figure 5).

Both single and multiple masters can exist on a Profibus network. At a single instant, only one master can be active and all others are temporarily reduced to slave status. However, within a predefined time, the current master passes control to the next master by issuing a special command called a token. On issue of this command the current master relinquishes bus control to the new master, which in turn takes control. This process repeats itself on a cyclic basis with the token periodically returning to a particular master device. Most devices, however, are configured as slaves only and do not take part in the token passing. A given master can address all the devices on the bus or a group of devices can be specified. It is important that the time that a master owns the token is long enough for it to communicate with all slave devices attached to it. This parameter must be configured before bus usage.

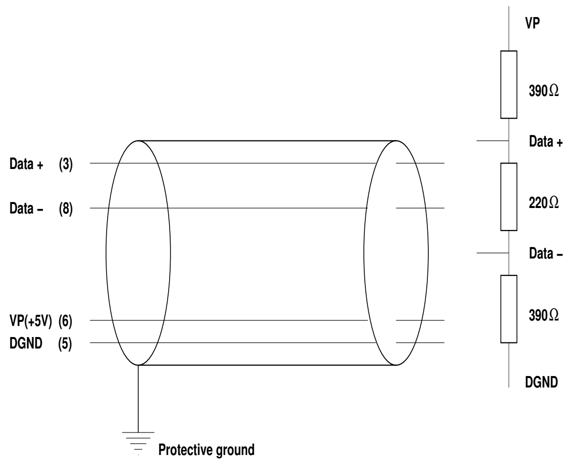


Figure 6: Profibus RS 485 cable definitions and terminations

The second layer of the OSI model describes the general format of data packets, which are known as telegrams (see figure 7 on page 14), and can each contain up to 244 bytes of data. Transmission services defined by this standard are:

SDA — Send Data with Acknowledge (for FMS only). Data are sent to the master or slave and a short acknowledgement sent in response;

SRD — Send and Request Data with acknowledgement (for DP and FMS). Data are sent and received in one telegram cycle;

SDN — Send Data with No-acknowledgement (for DP and FMS). Broadcast or Multicast (to a selected group telegrams);

CRSD — Cyclic Request and Send Data (FMS only).

Preceding each telegram must be a silent or idle period of at least 33 sync bits (roughly 3.3 bytes). The first byte in any telegram is the Start Delimiter (SD) used to distinguish the type of packet.

3.2.1 GSD (General Station Description) Files

An important concept in Profibus and one which must strictly be adhered to for a device to be approved as Profibus compatible, is the concept of a GSD file. An instrument's GSD file, which is an electronically readable ASCII device data file, provides a clear and comprehensive description of the properties and behaviour of that device. It contains both general and device specific parameters for communication and network configuration. During configuration, the GSD file can be loaded and using it a controller can both communicate with the device and understand the meaning of data being transferred. With this view, gone are the days when an engineer has to read the fine print of the device manual in order to get the device talking to the network. All that is needed is to load the GSD file and all translations are automatically performed.

Telegram 1: Request FDL Status: Used to determine any new active stations on the bus

Start Delimiter SD	Destination Address DA	Source Address SA	Function Code FC	Frame Checking Sequence, FCS	End Delimiter ED
0x10					0x16

Telegram 2: Data telegram with variable length data

SD	LE	LEr	SD	DA	SA	FC	DSAP	SSAP	DU	FCS	ED
0x68			0x68								0x16

Telegram 3: Data telegram with fixed length

SD	DA	SA	FC	DU	FCS	ED
0xA2				8 bytes		0x16

Telegram 4: Token telegram: Telegram transferring access rights between two bus masters

SD	DA	SA	ED
0xDC			0x16

Figure 7: Profibus Telegrams

Note: Keys are shown in table 4 on page 15.

Key:	SD	Start Delimiter	Distinguishes between telegram types
	DA	Destination Address	Address of receiving device
	SA	Source Address	Address of sending device
	FC	Function Code	Function code to identify whether telegram is a primary request, acknowledgment, or response
	FCS	Frame Checking Sequence	Addition of the bytes within the indicated length
	ED	End Delimiter	End of the telegram
	LE	Length	Length of data within the telegram. Includes DA, SA, DSAP, SSAP
	LEr	Length	Repetition of the length for error checking
	DSAP	Destination Service Access Point	The destination station uses this to determine which service is to be executed
	SSAP	Source Service Access Point	
	DU	Data Unit	Data bytes from 1 to 244 bytes

Table 4: Keys of Profibus Telegrams

3.3 Fieldbus

Fieldbus is a very advanced system that defines and deals with all aspects of industrial process control. Like Profibus it is based on layers 1, 2 and 7 of the OSI model but achieves its implementation in a very different manner. Again its goals are interoperability of devices, open but regulated definitions that encourage independent manufactures to produce compatible devices and intrinsically safe operation.

The only communication medium yet available is a cable consisting of a single twisted pair. Rather than the asynchronous RS 485 based systems discussed earlier, Fieldbus uses synchronous communications based around what is known as the Manchester Biphase-L coding technique (figure 8). In this system a clock and a data signal are combined to give an output signal in which the edges of the transitions rather than the amplitude of the signal are important. A rising edge corresponds to a logical 0 and a falling edge to 1. The output voltage is superimposed on a constant DC level and is designed to give a current variation of between 15-20 mA when dropped over a pair of terminating resistors at either end of the bus or trunk as it is known.

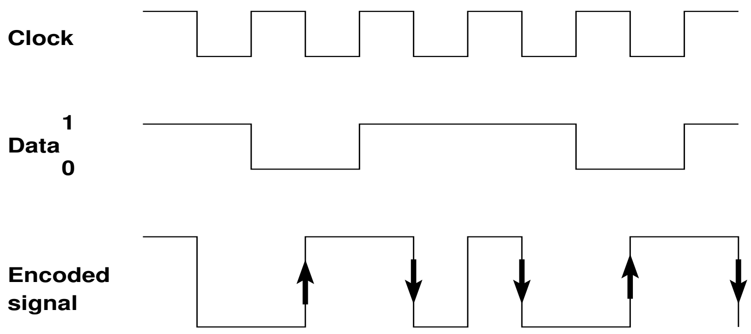


Figure 8: The Manchester Biphase -L encoding scheme

Each packet starts with a preamble consisting of a sequence of four 1,0 values. The data start and end with delimiters which are not Manchester encoded (allowing transitions on the rising edge of the clock pulse).

It is possible to obtain power from the DC levels used to transmit the signal although the number of devices that make use of this feature is limited by the total current drawn.

From the Fieldbus point of view, any device attached to the bus is seen as a node containing sets of parameters which collectively form what is known as the Virtual Field Device (VFD). The design of Fieldbus uses an Object Oriented approach to model the complex structures of typical devices in such a way that they can properly be controlled. The VFD is responsible for communicating the various objects that describe the device to other devices wishing to interact in some way. Various classes of objects are defined such as *Input classes, Control classes, Calculation classes* and *Output classes*, and each device must be described in terms of its class. Other properties of a device may be specified in terms of standard objects such as alarm objects or trending objects and once it is known that a device supports a type of object it can be manipulated accordingly.

Fieldbus goes well beyond any other system in its process control capabilities, but its introduction has been slow. At present, the Fieldbus Foundation's official website boasts of 100 registered devices, in comparison with over 300 for Profibus in roughly the same time span. It is extremely versatile at the expense of complexity, and only time will probably tell whether what is advertised as the "21st century fieldbus" will really succeed.

3.4 CANOpen

CANOpen and DeviceNet are derivatives of the CAN system outlined earlier (section 2.5). At any one time only a single device can be allowed to talk and effectively become the master. The mechanism whereby a device recognises whether or not it is to respond to a message is interesting and worth outlining.

Figure 9 illustrates how a CAN bus handles contention when three devices A, B and C contend for the bus at the same time. All are in step until position 5 of the Identifier when device A emits a high (recessive) level and drops into the listening state. Similarly device B drops out at position 1, leaving device C as bus master.

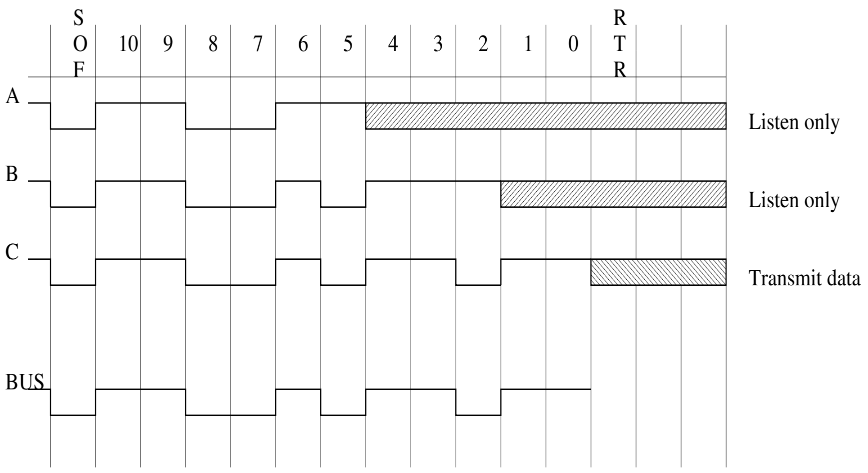


Figure 9: How a CAN bus handles contention

IDLE	SOF	IDEN	RTR	IDE	DLC	DATA	CRC	ACK	EOF	IFS	IDLE

Figure 10: The CAN message frame

When the bus is idle, the lines are held in the recessive state. At this stage a device that wishes to communicate, pulls the bus to a dominant state and immediately releases it to indicate the start of a new frame (SOF). Figure 10 shows the CAN message frame. Each listening device must then follow the sequence of bits defined in the message identifier and put the correct value onto the bus. The message identifier is a sequence of 11 or 29 bits (depending on whether the frame is extended or not as determined by the IDE bit) starting with the most significant bit. As the bits are output, each device checks to see whether it can match the identifier output so far. If it cannot it goes into a passive listening condition and outputs a recessive state. Otherwise it continues trying to match bits. In this way as the identifier bits are output, devices pass from the active to the passive listening states one by one until the message identifier is complete. If a device is still active at the end of the identifier it starts to processes the message. Following the message identifier, the RTR bit distinguishes the message as either a request for a device to send data or actual data. In the later case the data are contained in up to 8 data bytes in the DATA field with the actual number of bytes set in the DLC byte. Error checking can be performed by a cyclic redundancy checksum contained in the CRC byte. An indication of whether or not the message has been received correctly is given in the ACK bit. This bit is sent as recessive and is overwritten with dominant values by each device that correctly received the message. Finally an EOF bit is sent. Following the end of transmission, a time equivalent to set minimum number of bits (IFS) must elapse prior to the next transmission and the bus remains in the idle state until a device is ready to transmit.

Note that there is no need for separate acknowledgement messages as in many other protocols as a device can immediately indicate any error by setting the acknowledgement bit to the dominant state. Bus conflicts where two or more devices compete for the right to talk are also handled by the bus in a simple and natural manner. Following the start SOF bit, the bits will be placed onto the bus by the competing devices. At each step any device that puts out a recessive bit when another puts out a dominant bit goes into the listening state until only a single device is left. Since the priority of a message is defined by the inverse of the message identifier value (i.e. the higher the priority, the lower the message identifier value), this mechanism determines that the device with the highest priority message always wins in case of conflict (the highest priority will consist of dominant zeroes).

4 Industrial Ethernet

As industrial automation and control gets more sophisticated and ubiquitous, a natural step is to use the information gathered directly in planning, stock control, and management systems. Previous attempts at Process Application Management (PAM) and Enterprise Resource Planning (ERP) have utilised separate networks such as fieldbus systems for the factory level, and the office level Local Area Networks running TCP/IP, with PC based gateways acting to cross the divide. As noted in the introduction, traditionally very few devices have had Ethernet connectivity. This situation is, however, changing with the introduction of Industrial Ethernet and other networking systems. Industrial Ethernet is basically a more robust version of standard Ethernet, better suited to the more extreme conditions met in industry, with greater operating temperature range, higher noise immunity hardware, and a wider range of connectors. All equipment is specified to operate using the industrial standard 24 V DC and is guaranteed to be backward compatible for at least 10 years, the typical lifetime of industrial components. TCP/IP is used as the main protocol with the existing fieldbus protocols embedded within the upper layers of the TCP/IP stack. Several protocols have taken advantage of the many benefits offered by Ethernet (high bandwidth and transmission rates, fault tolerance, and cost, to name a few) made this transition within the last few years including Modbus (Modbus TCP/IP), Profibus (ProfiNet) and Fieldbus (Fieldbus HSE). Widespread usage of any of these has yet to be achieved, but as instruments equipped with the appropriate hardware become available, this situation can be expected to change significantly.

5 OPC: A universal interface

One of the problems encountered by any systems integrator faced with the task of connecting a device to a SCADA monitoring system is to identify the various functions a device performs and how to extract the information needed by the monitoring software for alarm indication and trending. As we have seen many protocols define sets of registers in which the various items of data can be found. The problem is that no two devices ever appear to have a common set of register definitions and that adding a new device to an existing fieldbus network has often required extensive modifications to the SCADA software. A workable solution to the problem is the provision of drivers for each device on the network, that the software loads dynamically at run time. Such a driver will take care of the peculiarities of a specific device and provide an interface through which the software can access its data in a device independent manner. To be really useful, however, the device driver should not be specific to a single SCADA system but be available to all such systems.

OPC (OLE for Process Control) [5] has been developed to fulfil these needs and in its present form comprises a wide range of specifications and guidelines covering many aspects of process control. At its heart is the concept of transmission of objects (in the full object oriented sense) between different processes and threads as originally specified by the Open Software Foundation (OSF) in the 1990s. Early efforts at implementing these ideas resulted in the Object Linking and Embedding (OLE) technology developed by Microsoft which were later refined into ActiveX controls working on top of Component Object Model (COM) components. A COM server can either be *In process* or *Out process* meaning that it is a Dynamically Linked Library (DLL) or a separate process respectively. For communication with objects on other machines on a network, Distributed COM (DCOM) is used. In any case, each COM object specifies a number of *interfaces* (or methods in OOPS) that once published are guaranteed not to change.

The creators of OPC [6] have modelled the behaviour of generic devices and defined a large number of specifications for all areas of functionality. Libraries of base objects exist and COM controls for new devices can be created inheriting all the interfaces of the parent objects, but tuned to the characteristics of a specific device. All a monitoring software needs to do is to load the particular COM object at run time and use its published interfaces in the correct manner, thus achieving both device and software vendor independence.

Although developed for the MSWindows^{TM} environment, COM and DCOM technologies have been ported to other operating systems including Linux and other versions of Unix as well as several operating systems for embedded applications (Wind River in particular have versions for a number of processors). Thus the technology is not confined to a single platform and OPC has gained wide spread usage over the entire industry. Microsoft's much vaunted ".NET" platform is the successor to COM, and OPC is currently being ported to that environment. This will make it much more portable to other environments as at least one project to produce an Open Source port of the entire .NET platform is in progress.

The various specifications currently covered by OPC are shown in table 5.

	SD	Start	Distinguishes between telegram types
	OPC	Overview	General description
	OPC	Common Definitions	Common definitions and interfaces
	OPC	Data Access	Reading and writing real time data
Key:	OPC	alarms and events	Event monitoring
	OPC	historical data	Data trending
	OPC	Batch	Extensions to Data Access
	OPC	and XML	Integration of OPC and XML for web applications

Table 5: OPC specifications

In the following section, the Data Access specification is examined as an example.

5.1 The Data Access Specification

The Data Access Specification is the oldest of the OPC specifications and defines an interface between client and server processes. One or more *data access client* can access a *data access server* which can provide what are known as *data sources* (such as temperature sensors, vibration sensors, etc) and *data sinks* (controllers of various types). Together these data sinks and sources constitute a model of the device to be connected. A data access client could be a very simple application displaying data in a table or be a single component of something more complicated such as a SCADA system. Similarly, a data access server can be a simple process such as one that communicates with a device and returns selected values from its registers, or a complicated one performing multiple control functions.

A data access server has two components that a client can use, the *namespace*, and the *OPC object hierarchy* an example of which is shown in figure 11 on page 21. The namespace contains lists of all the sources and sinks maintained by the server as a tree structure. Each node in the tree also can contain information *properties* detailing how the node is to be used and *access paths* describing the communication settings and pathways. The object hierarchy consists of three levels of objects consisting the *OPCServer object*, the *OPCGroup objects* and lastly the *OPCItem objects*. A single OPCServer object can manage multiple OPCGroup objects which in turn are used to group or structure multiple OPCItem objects. OPCItem objects represent the items within the namespace. Both the OPCServer and OPCGroup objects are implementation independent with all implementation details being carried by the OPCItem objects.

As the system boots, during the configuration stage, the namespace is scanned to extract its information and OPCGroup and OPCItem objects are created accordingly. For example, consider the following situation where the temperature

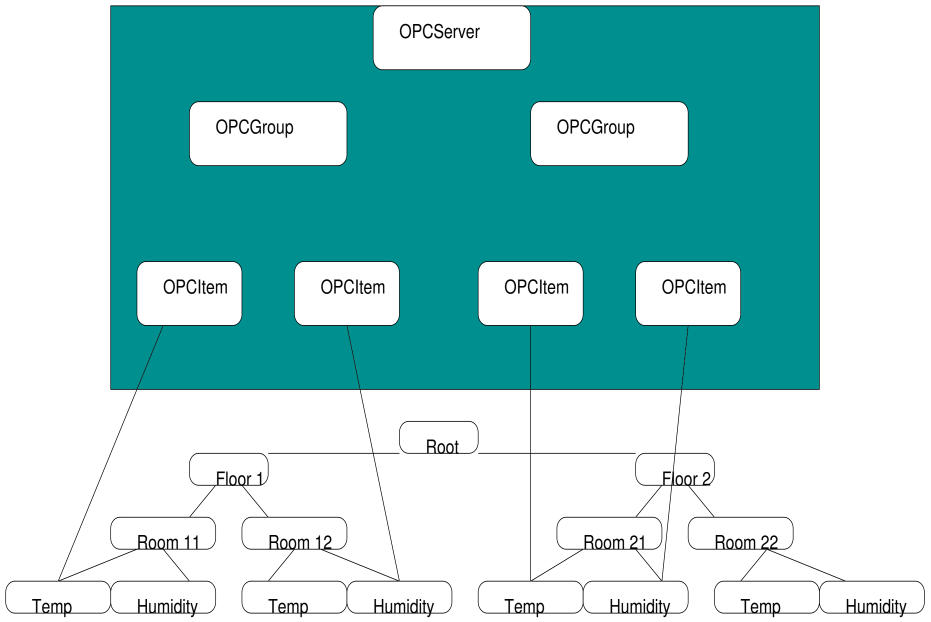


Figure 11: An example of namespace and object hierarchies. Each item within the namespace can have a corresponding OPCItem.

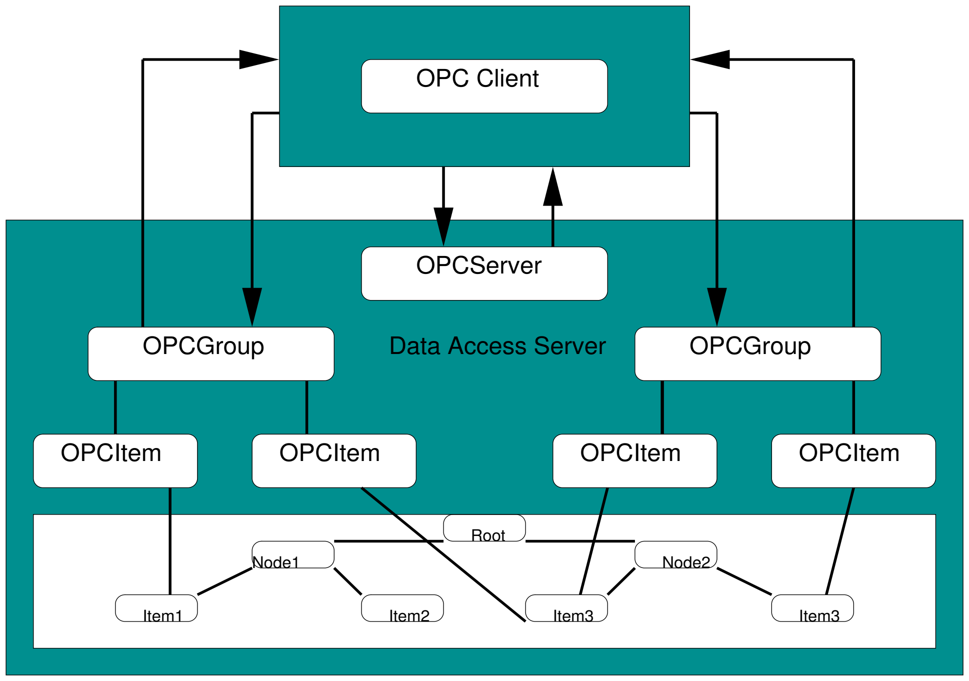


Figure 12: A schematic representation of a Data Access Server. The OPC Client connects to the OPCServer and OPCGroup objects. The namespace, represented here as a tree starting at Root is scanned at startup and has OPCItem objects created for the various items within the namespace.

and humidity of rooms in a multistory building are being monitored and controlled. Each floor of the building has several rooms (in total N) in which measurements are to be made giving rise to the tree shown in figure 12 on page 21. If all the temperature and humidity measurements are grouped together to form two OPCGroups, each has N OPCItems representing measurements from each room. The client can access the data by using the methods of the exposed OPC object interfaces.

6 Future trends

As field buses get more complicated, faster and cheaper hardware solutions are required. One of the great selling points of Profibus for example was the reduction in cost possible over the old 4-20 mA current loop technology, mostly achieved as a result of using linear cable topologies rather than bringing all cables back to a central star point. It is evident that the main remaining expense (apart from software) is the cost of the interfaces themselves. On the other hand Ethernet technology is cheap and widely available. It can be expected that Ethernet will become an important player in industrial networking.

Whether the 'Fieldbus Wars' as one commentator has described the current position will be resolved in favour of a single standard is doubtful. Probably a few of the older, less open standards will slowly disappear, as the advantages of the modern systems become more apparent, but as in any democracy, there will always be a small but vocal minority of devotees to a particular system who will stoutly refuse to allow its demise.

Fieldbus System	Developer	Year	Standart	Degree of openness	Media	Max # of nodes	Max distance
Profibus	Siemens	1995	EN50170/DIN19245 IEC 1158-2	Asics from Siemens Products from > 300 manufactures	Twisted pair Optic fiber	127	100m between segments @12MBaud
INTERBUS-S	Phoenix	1984	DIN 19258	Products from > 400 manufactures	Twisted pair Optic fiber	256	400m between segments 12.8km total
DeviceNet	Allen-Bradley	1994	ISO11898 ISO11519	17 asic manufactures Products from > 300 manufactures	Twisted pair	64	6km with repeaters
ARCNET	Datapoint	1977	ANSI/ATA 878.1	Asics, boards, ANSI documents	Coax Twisted pair Optic fiber	255	Coax 60m T/P 130m O/F 2000m
Foundation Fieldbus	Fieldbus Foundation	1995	ISA SP50/IEC61158	Chips from multiple vendors	Twisted pair Optic fiber	240/segment	1900m@31.2kBaud
Foundation Fieldbus HSE	Fieldbus Foundation	Under test	IEEE 802.3u	Ethernet, many suppliers	Twisted pair Optic fiber	∞	T/P 100m @ 100MkBaud O/F 2km @ 100MBaud
CANopen	CAN In Automation	1995	CiA	17 chip vendors, Open specification	Twisted pair	127	25-1000m (depends on Baud rate)
Modbus	Modicon		EN 1434 IEC 870-5	Open specification No special hardware needed	Twisted pair	250 Nodes /seg	350m

Table 6: Comparison of some common industrial bus systems (source: Synergetic)

References

[1] `http://www.can-cia.org`. 2.5, 3

[2] `http://www.modbus.org`. 3

[3] `http://www.profibus.org`. 3

[4] `http://www.fieldbus.org`. 3

[5] Iwanitz F and Lange J, (Eds), ***OPC Fundamentals, Implementation and Application***, Hüthig Fachverlag, 2002. 5

[6] `http://www.opcfoundation.org`. 5