



**DEPARTMENT No. 12
ELECTRICAL ENGINEERING
AND COMPUTER SCIENCE**

Institute for Power Electronics
and Electrical Drives
Prof. Dr.-Ing. G. Schröder

Automation and industrial communication

Chapter 2 : Interfaces to the process and
between automation devices

Revision date: 08.2002

Contents

1.	Interfaces of modern automation devices . . .	- 1 -
1.1	Interface to the process	- 1 -
1.1.1	Bit-/Byte-/Word-oriented I/O . . .	- 1 -
1.1.2	Counter inputs	- 14 -
1.1.3	Analog I/O	- 15 -
1.1.3.1	Isolation amplifiers	- 15 -
1.1.3.2	Conversion of analog values to digital	- 17 -
1.1.3.3	Conversion of digital into analog values	- 32 -
1.2	Interfaces to other automation devices .	- 36 -
1.2.1	-- Chapter is left blank -- . . .	- 36 -
1.2.2	Serial Point-to-Point-Link (RS232-C and TTY)	- 36 -
1.2.3	Networks	- 41 -
1.2.3.1	Topologies	- 41 -
1.2.3.2	Transmission methods	- 44 -
1.2.3.2.2	RS 485	- 44 -
1.2.3.4	Selected communication systems	- 49 -
1.2.3.4.1	PROFIBUS	- 49 -

- 1. Interfaces of modern automation devices**
- 1.1 Interface to the process**
- 1.1.1 Bit-/Byte-/Word-oriented I/O**

When talking about bit-oriented (binary) inputs and outputs it is assumed, that each bit represents a complete information entity of its own, which describes the state of a contact or the presence or absence of a voltage in a process electric circuit.

If the program for example sets a bit to logically "1", the closing of a contact (or switching of a transistor) in the output interface is initiated. This can be used for example to start an external device or to stop it.

The input and output interfaces are often manufactured as modules, which can handle several inputs and outputs. These modules have either a single external supply or different supplies with electric isolation against each other. These modules are used in so called modular devices in contrast to compact devices, in which type and number of inputs and outputs is fixed and can not be changed significantly.

a) Digital input modules

Digital input modules convert the external process signals into internal signals of the automation device. Input signals can be produced e.g. by manual pushbuttons, manual switches, proximity switches, relay contacts, et cetera. The process signals are either DC or AC signals. Besides the conversion into internal signals (mostly 5-Volt-TTL level) the galvanic isolation between the process voltages and the internal voltage must be assured. Protection against overvoltage, reverse voltage and short circuit is to be provided.

Generally circuits are utilized, that are adapted to the following process voltage levels:

- DC 24V
- DC 110V
- AC 120V
- AC 230V

The following parameters characterize the process signal input modules:

- number of inputs
- isolation process signals/ internal signals
- isolation between process signals
- input current
- delay
- admissible cable length
- input voltage
- max. admissible input voltage
- switching thresholds

Fig. 1.1.1.1 shows a typical schematic diagram for the connection of a input module of a PLC.

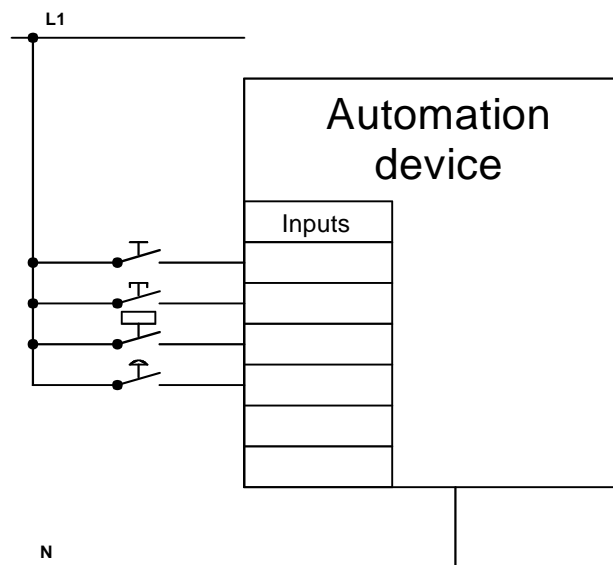


Fig 1.1.1.1: Connection of binary signal generators to an automation device

A typical AC input circuit is displayed in Fig. 1.1.1.2.

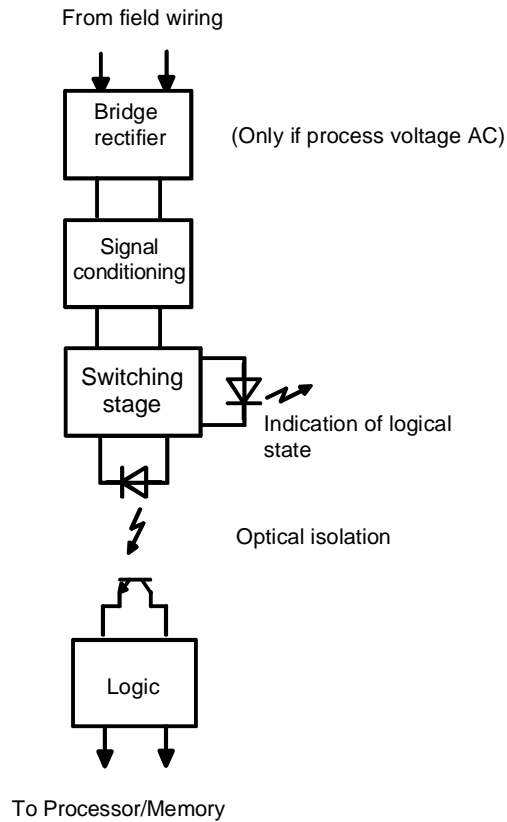


Fig. 1.1.1.2: Input circuit of an automation device for binary AC input signals

There is a bridge rectifier for the AC voltage. The bridge is normally followed by a signal conditioning circuit, which smoothens the signal to suppress disturbances and contact bouncing.

Then a circuit follows, which determines, if the input voltage is above or below a certain threshold. The logical state is derived from the voltage level and the time as well.

After this block has recognized the signal safely, it normally passes an isolation stage, which prevents external circuit or device faults from disturbing the internal circuitry. This is usually obtained with an optocoupler.

The last stage of a input module contains a logic, that assigns

the logic "0"- and "1"- states to the correct levels. Normally the momentary state of the input is displayed by providing a LED, which shines in the case of the logic "1"-state.

The recognition of the exceeding of a voltage level in the input module and the damping of the signal generates a delay of the recognition of a state change. This delay depends of the structure of the particular input module and is between 5 and 30 ms, with 10 ms being a usual value. Often the behaviour is asymmetrical, i.e. changes from ON to OFF are recognized faster than changes from OFF to ON.

b) Digital output modules

The bit oriented discrete interface for output looks similar to the input interface. Fig. 1.1.1.3 displays the basic concept.

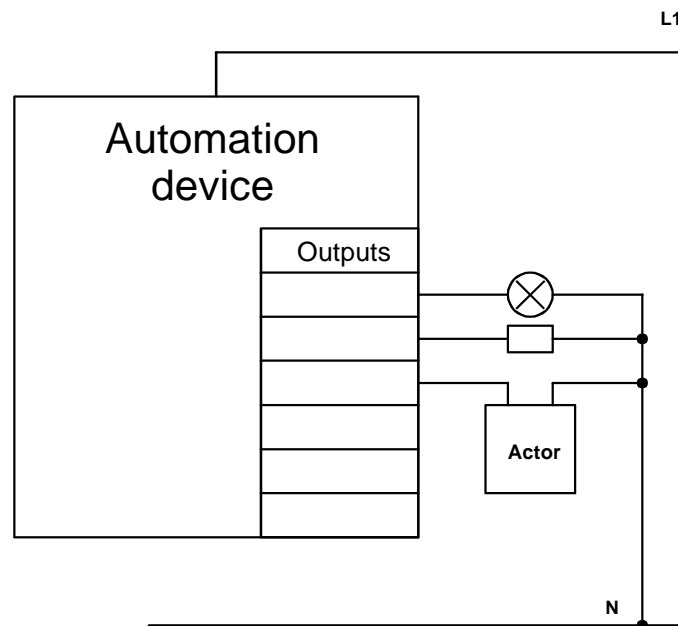


Fig. 1.1.1.3: Connection of binary actors to an automation device

A typical block diagram of an output module is shown in Fig. 1.1.1.4.

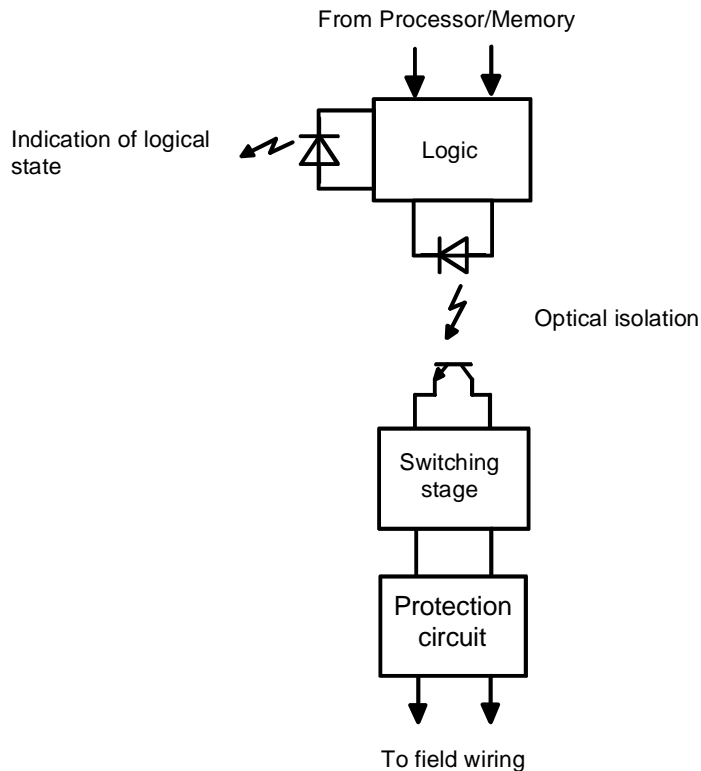


Fig. 1.1.1.4: Block diagram of a binary output circuit

The logical signal produced by the PLC passes an isolation stage, followed by a switching stage. This circuit works with a supply voltage fed from external and is normally equipped with a protection against reverse voltage and overload.

For output modules the following parameters are important:

- number of outputs on a module
- the minimum/maximum supply voltage
- the maximum load current
- the isolation of the outputs against each other
- the protection against short circuit
- the maximum switching frequency

Output modules for AC are normally slower than DC modules.

c) Variables with N values

The most commonly spread type of absolute encoder uses an optical method, in which a source of light spotlights a disk with transparent and non-transparent regions. On the other side of the disk a row of detectors is situated, one for each track. The output signal of each detector is one binary information unit.

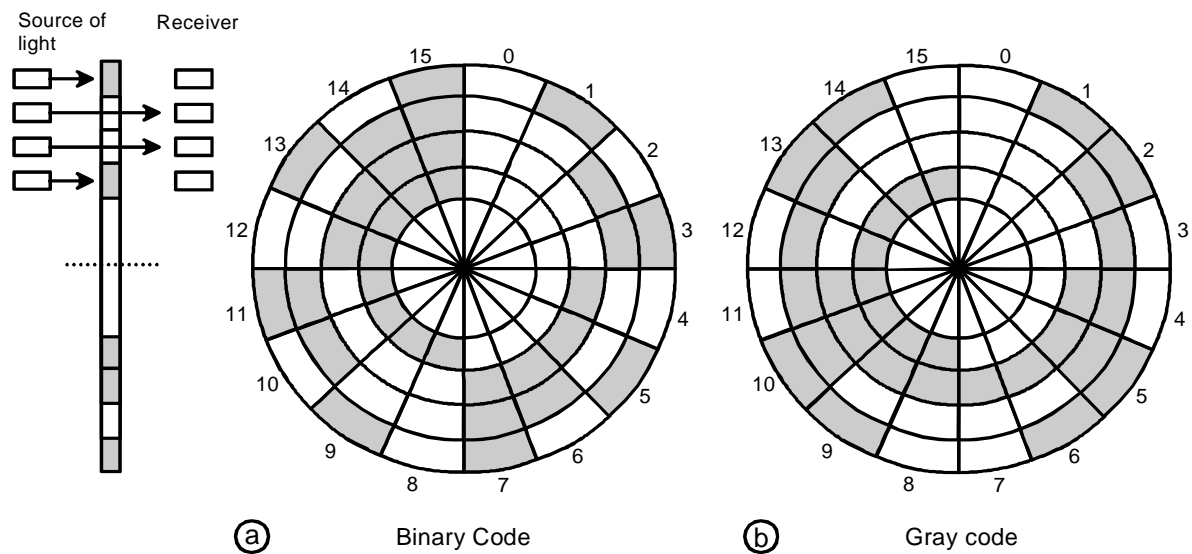


Fig. 1.1.1.5: Absolute angle encoder: a) Binary code; b) Gray code

In Fig. 1.1.1.5 two disks with different codes are to be seen. On the one hand we have the well known binary code, on the other hand the Gray code. The reason for the fact, that in most cases the binary code is not used in position and angle transducers is, that very small tolerances in the manufacturing of the disk and the adjustment of the detectors are necessary to avoid intermediate states, when several bits change their state at the same time. This disadvantage does not exist when using the Gray code, because there never two tracks change their state at the same time.

The absolutely coded angle transducer has the advantage, that at any time within the measurement accuracy the exact position of

the shaft is known, even after a power failure. Going to a reference position, which is sometimes necessary with incremental encoders is not applicable here.

Other examples of variables with N values (at the human machine interface):

- input values from a decade switch
- output values to number displays

Both of these variables are normally of the type BCD.

Serial Transmission of angle informations: Synchronous serial interface (SSI)

The parallel, absolute angle information inside an angle encoder is converted into a serial information by an internal parallel-serial-transformer (shift register) and is transmitted synchronously with a clock to a receiver electronics.

This angle information is generated inside the angle encoder in Gray code and is also transmitted in Gray code. To reduce the expense of components the information is not converted to binary code until it reaches the receiver electronics.

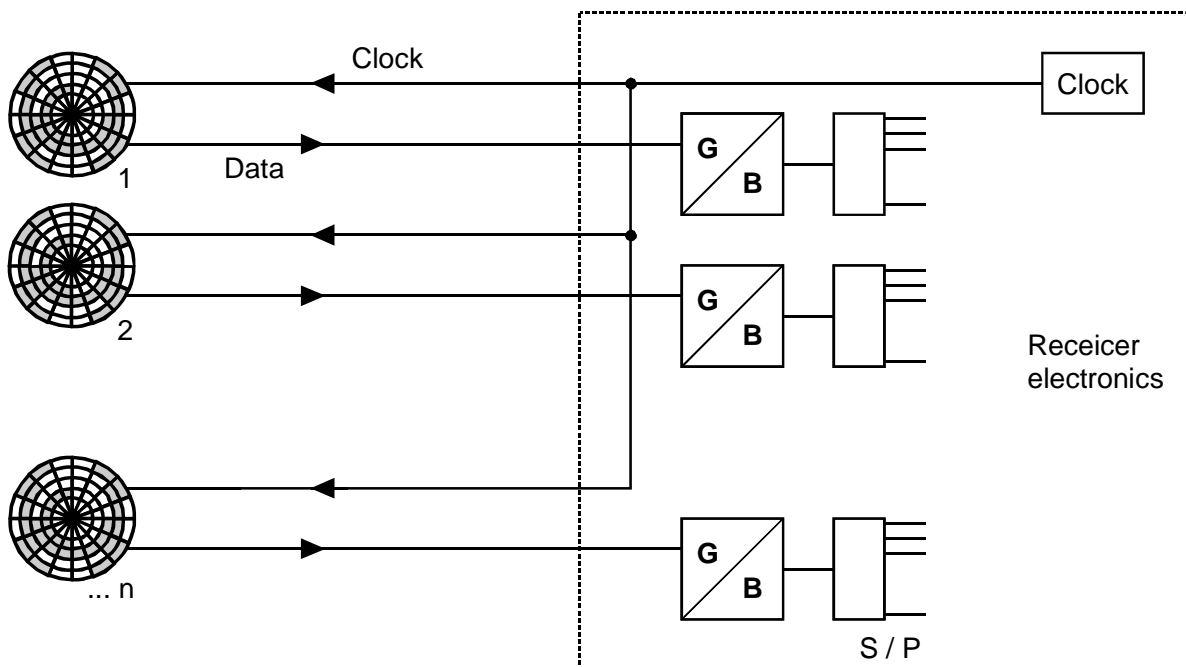


Fig. 1.1.1.6: Synchronous serial data transmission with several angle encoders: Serial Gray-binary conversion (G/B) and serial-parallel conversion (S/P) in the receiver electronics

The synchronous transmission of the data word is initiated and controlled by the receiver electronics by a clock signal. The length of the clock sequence determines the length of the data word, so that with this SSI system arbitrarily long data words can be transmitted. For the transmission of an n bit long data word a clock sequence of n + 1 clock pulses is necessary. The

speed of the data transmission is determined by the clock frequency. The maximum value depends on the cable length and is in practice max. 200 kBit/s at 200 m transmission length.

Example for the saving of strands with serial transmission

Regarding a multitour angle encoder with parallel output e.g. 1024 Steps/revolution and 1024 revolutions (20 bit) for the data transmission 20 strands are necessary. The SSI only needs a twisted pair of strands for the data and a twisted pair of strands for the clock. For power supply and additional functions (e.g. direction of code; enable) the expense is identical in both cases. The minimum number of strands in a cable is 6.

The SSI transmission protocol for multitour angle encoders

The logical levels mentioned later are related to the Clock+ and data+ signals respectively. In the standby or idle state of the SSI clock and data wire are logically "1". The receiver electronics initiates the data transmission by a change of the clock signal from logically "1" to logically "0". With this change in the angle encoder a retriggerable monoflop is set, the output of which switches over a shift register from parallel to serial, whereby the data available in gray code are stored.

With the next change of the clock from logically "0" to "1" the most significant bit of the angle information is shifted to the data output of the angle encoder. Each further rising edge of the clock delivers the next bit until the least significant bit is at the output. At the same time with every falling edge of the clock the monoflop is retriggered.

The monoflop time of e.g. 20 μ s determines the pause between two transmissions and the minimum clock frequency.

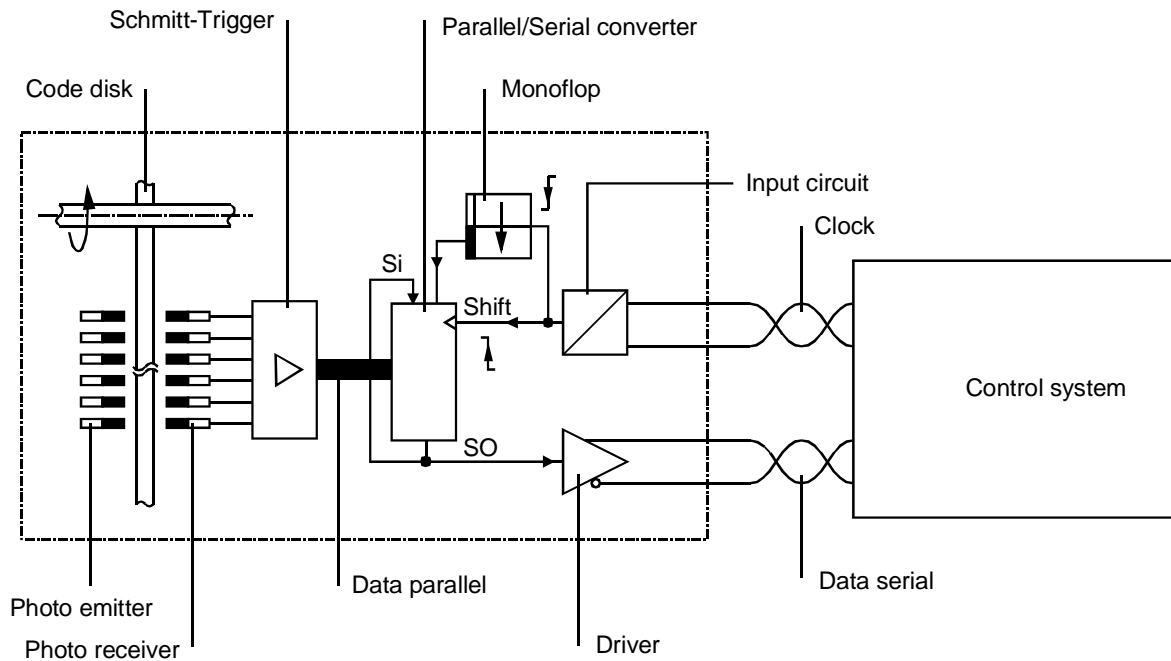


Fig. 1.1.1.7: Basic schematic diagram of an absolute angle encoder with serial output

With the last rising edge of the clock the data wire is set to logically "0". Then the transmission of the complete data word is finished. When the clock signal remains logically "1" (End of the clock sequence), the monoflop is not retriggeder any more causing the data signal to switch to logically "1" after the monoflop time has expired.

With this state the readiness for transmission of a new data word is indicated.

Transmission example for an angle encoder with 18 bits

An angle encoder with 1024 Steps/revolution (10 bits singletour) and 256 revolutions (8 bits multitour) is assumed.

The transmission protocol is designed for a data word of 25 bit in the standard version. 12 of these bits are provided for the revolutions and 13 for the steps/revolution.

Because the transmission always starts with the multitour bit M12, but the multitour part in our example is designed only for 8 bits, in the beginning 4 empty digits with a logical "0" are transmitted and then the existing 8 bits of the multitour part. Afterwards the bits of the singletour part follow, beginning with S10 to S1. The 3 bit, which are not existing are also transmitted as a logical "0".

		Data word with n = 25																										
		MSB												Bit Nr. in Data word											LSB			
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25		
4096	12	M12	M11	M10	M9	M8	M7	M6	M5	M4	M3	M2	M1	S13	S12	S11	S10	S9	S8	S7	S6	S5	S4	S3	S2	S1	8192	13
2048	11	0	M11	M10	M9	M8	M7	M6	M5	M4	M3	M2	M1	S12	S11	S10	S9	S8	S7	S6	S5	S4	S3	S2	S1	0	4096	12
1024	10	0	0	M10	M9	M8	M7	M6	M5	M4	M3	M2	M1	S11	S10	S9	S8	S7	S6	S5	S4	S3	S2	S1	0	0	2048	11
512	9	0	0	0	M9	M8	M7	M6	M5	M4	M3	M2	M1	S10	S9	S8	S7	S6	S5	S4	S3	S2	S1	0	0	0	1024	10
256	8	0	0	0	0	M8	M7	M6	M5	M4	M3	M2	M1	S9	S8	S7	S6	S5	S4	S3	S2	S1	0	0	0	0	512	9
128	7	0	0	0	0	0	M7	M6	M5	M4	M3	M2	M1	S8	S7	S6	S5	S4	S3	S2	S1	0	0	0	0	0	256	8
64	6	0	0	0	0	0	0	M6	M5	M4	M3	M2	M1	S7	S6	S5	S4	S3	S2	S1	0	0	0	0	0	0	128	7
32	5	0	0	0	0	0	0	0	M5	M4	M3	M2	M1	S6	S5	S4	S3	S2	S1	0	0	0	0	0	0	0	64	6
16	4	0	0	0	0	0	0	0	0	M4	M3	M2	M1	S5	S4	S3	S2	S1	0	0	0	0	0	0	0	0	32	5
8	3	0	0	0	0	0	0	0	0	0	M3	M2	M1	S4	S3	S2	S1	0	0	0	0	0	0	0	0	0	16	4
4	2	0	0	0	0	0	0	0	0	0	0	M2	M1	S3	S2	S1	0	0	0	0	0	0	0	0	0	0	8	3
2	1	0	0	0	0	0	0	0	0	0	0	0	M1	S2	S1	0	0	0	0	0	0	0	0	0	0	0	4	2
Revol. count	Bit / rev..	Multitour - Bit											Singletour - Bit											Steps/ rev.	Bit/ rev.			

Fig. 1.1.1.8: Transmission of angle information of different width at a agreed data word length of 25 bit

The control electronics of the SSI

The control electronics generates the clock sequence the transmission of the data word from the angle encoder is controlled by. In this electronics the serial data can be converted to parallel and where applicable the gray code can be recoded into binary code.

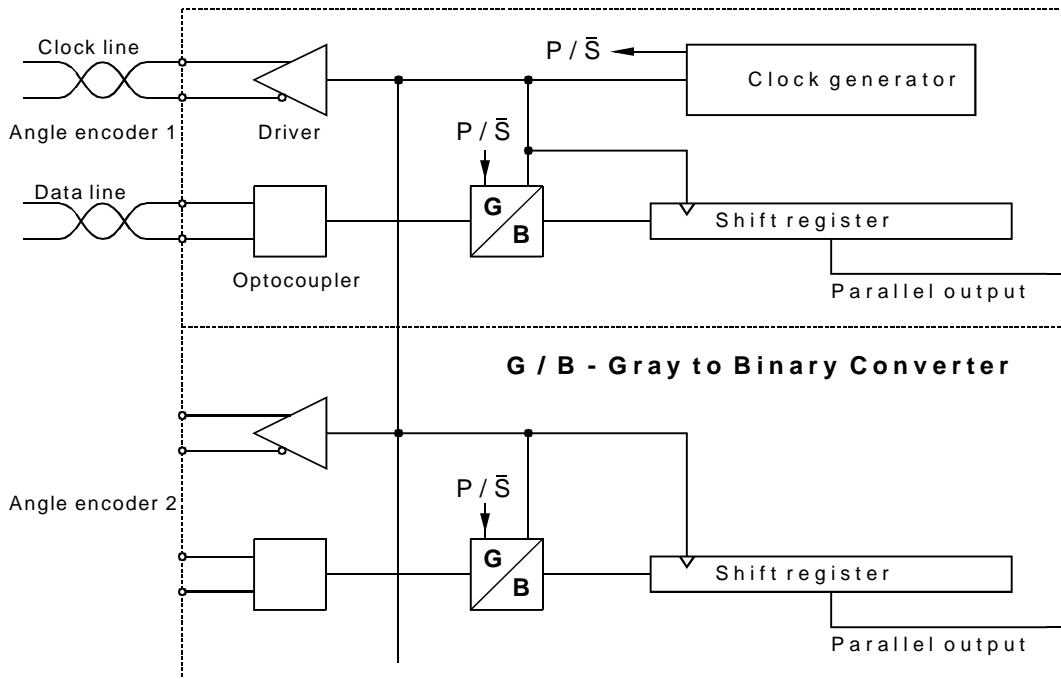
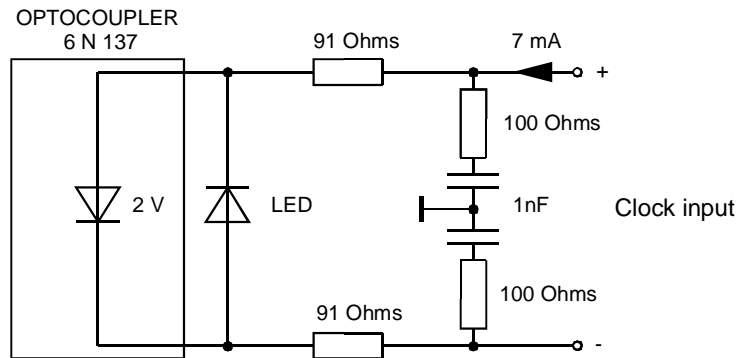


Fig 1.1.1.9: Block diagram of the control electronics of the SSI

Input and output circuits of the SSI

The input (optocoupler) and output circuits (Linedriver) can be identical in the angle encoder and the control electronics.

Input circuit



Output circuit

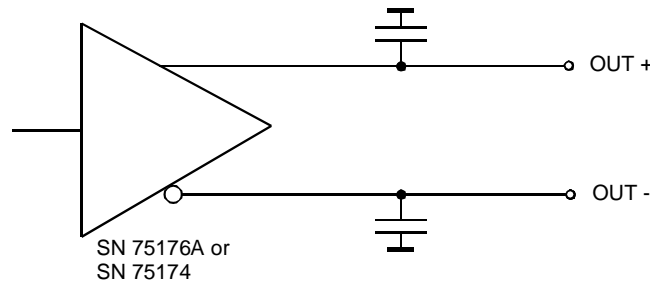


Fig. 1.1.1.10: Input and output circuits of the SSI

In the case of the output circuit we have a differential line driver, which complies to the electrical data of the RS 422 standard. By the differential, symmetrical design a high amount of interference immunity is provided. By using optocoupler inputs ground loops are avoided, increasing further the noise immunity. This is especially important when several angle encoders are connected to the control electronics.

1.1.2 Counter inputs

Counter modules provide the possibility to count fast pulse trains, which are usual when using incremental displacement or angle encoders. No pulses are allowed to get lost, because in that case the information about the position of the encoder would be lost. A typical task is to count two pulse trains which are shifted in phase by 90 degrees against each other or counting of forward and backward pulses (siehe Fig. 1.1.2.1).

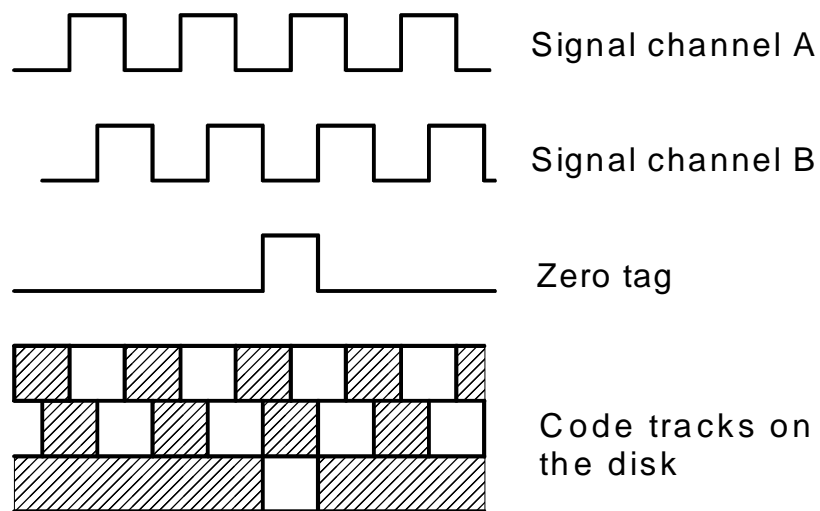


Fig. 1.1.2.1: Output signal of an incremental displacement or angle encoders and the coded tracks on the encoder disk

Typical parameters and properties of counter cards are:

- max. counting frequency
- Input voltage level
- Cascadeability of several counters to increase the maximum counting value
- Input symmetrical/unsymmetrical

1.1.3 Analog I/O

Currents and voltages can be acquired by A/D converters and be converted to digital values. Digital output values of the application software can be converted into voltages and currents with D/A converters. Before the A/D converter or after the D/A converter we find components like instrumentation amplifiers, multiplexers, sample- and hold blocks, isolation stages.

1.1.3.1 Isolation amplifiers

One possibility to isolate the internal and external parts of an automation device is the utilization of isolation amplifiers.

The most commonly used methods to achieve isolation between input and output are optical and transformer coupling.

a) Optical coupling

Optical coupling is a very interference safe isolation, because it cannot be disturbed by electromagnetic interference. The schematic symbol of an optical isolation amplifier is displayed in Fig. 1.1.3.1.

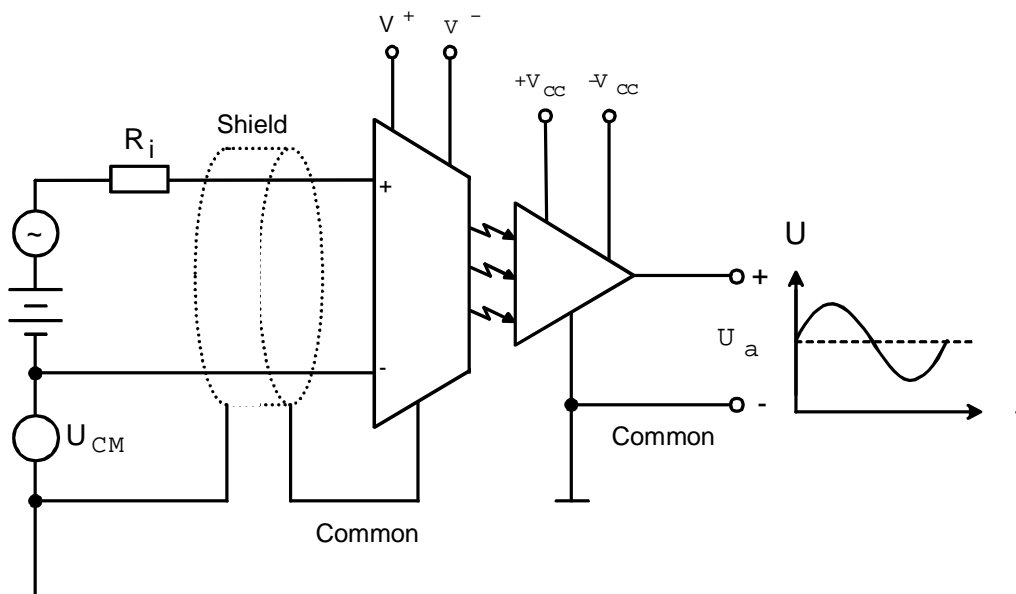


Fig. 1.1.3.1: Optically coupled isolation amplifier

b) Transformer coupling

Typical isolation amplifiers based on transformer coupling provide absolute separation, a small capacity $< 10\text{pF}$ between input and output common and a common mode rejection ratio of 110 dB at 50 Hz as well as common mode voltages up to 5 kV. Because they are capable of transmitting millivolt signals they are applied e.g. in the medical technology for measuring ECG signals.

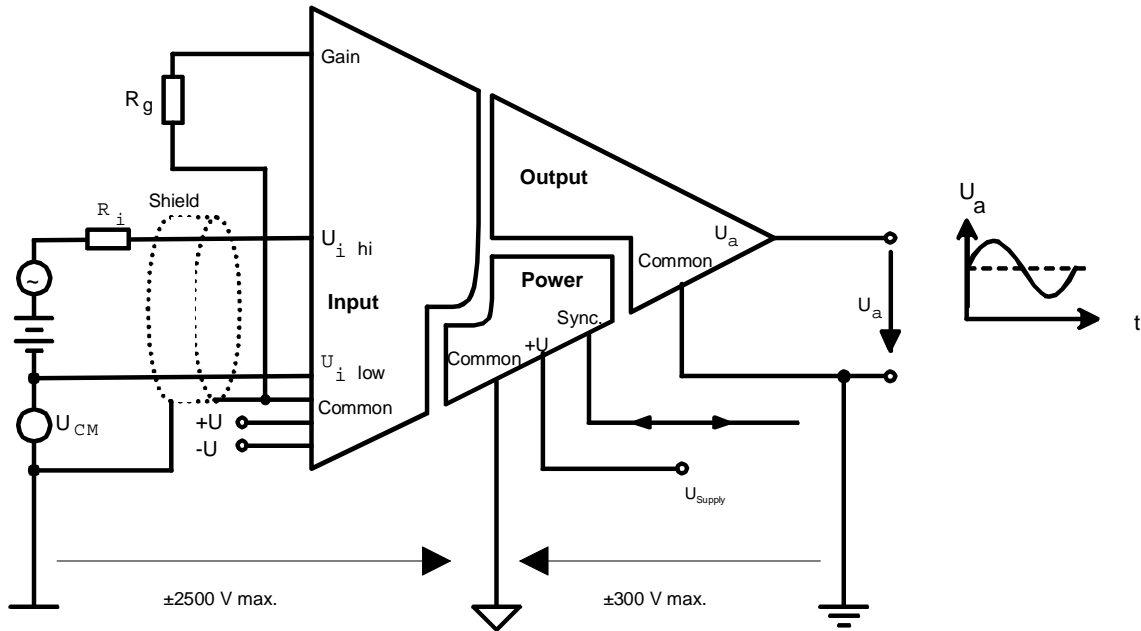


Fig. 1.1.3.2: Transformer-coupled isolation amplifier schematic symbol

1.1.3.2 Conversion of analog values to digital

Behind the (isolating) amplifier in a measurement system usually a low-pass filter is utilized. The filtering is applied for two reasons:

- to limit the bandwidth of the signal to less than half of the sampling frequency and to eliminate hereby the aliasing effect
- to filter out high frequency interference in the input signal.

If more than one analog channel shall be A/D converted an analog multiplexing is necessary to connect the analog inputs with the single A/D converter (cost-effective). A digital multiplexing is necessary to connect the outputs of all the A/D converters to the following digital processing when using one converter per channel (fast).

a) Analog multiplexing

Analog multiplexers allow time-sharing of A/D converters between several analog information channels. An analog multiplexer consists of a group of MOSFET switches being connected to the individual inputs on the one side and with one another on the other side.

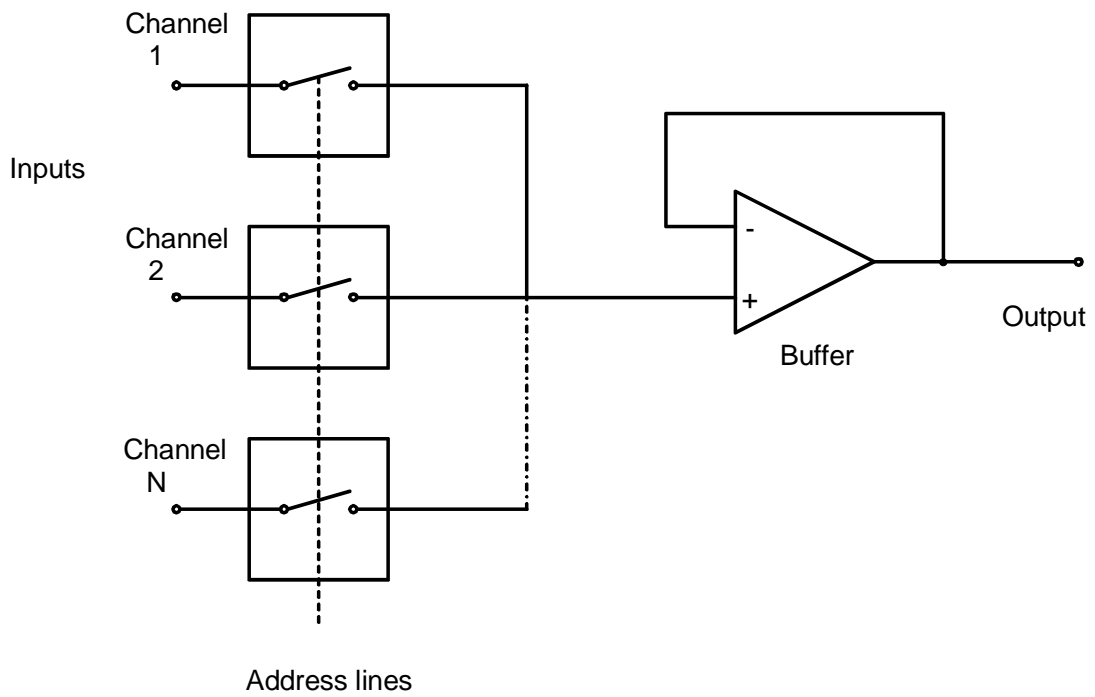


Fig. 1.1.3.3: Analog multiplexing

The switches are addressed with a digital code.

b) Digital multiplexing

If high throughput at a small number of measuring channels is demanded the method of **digital multiplexing** is applied. The digital conversion results of several A/D converters are switched onto the data bus of an automation device in time multiplex.

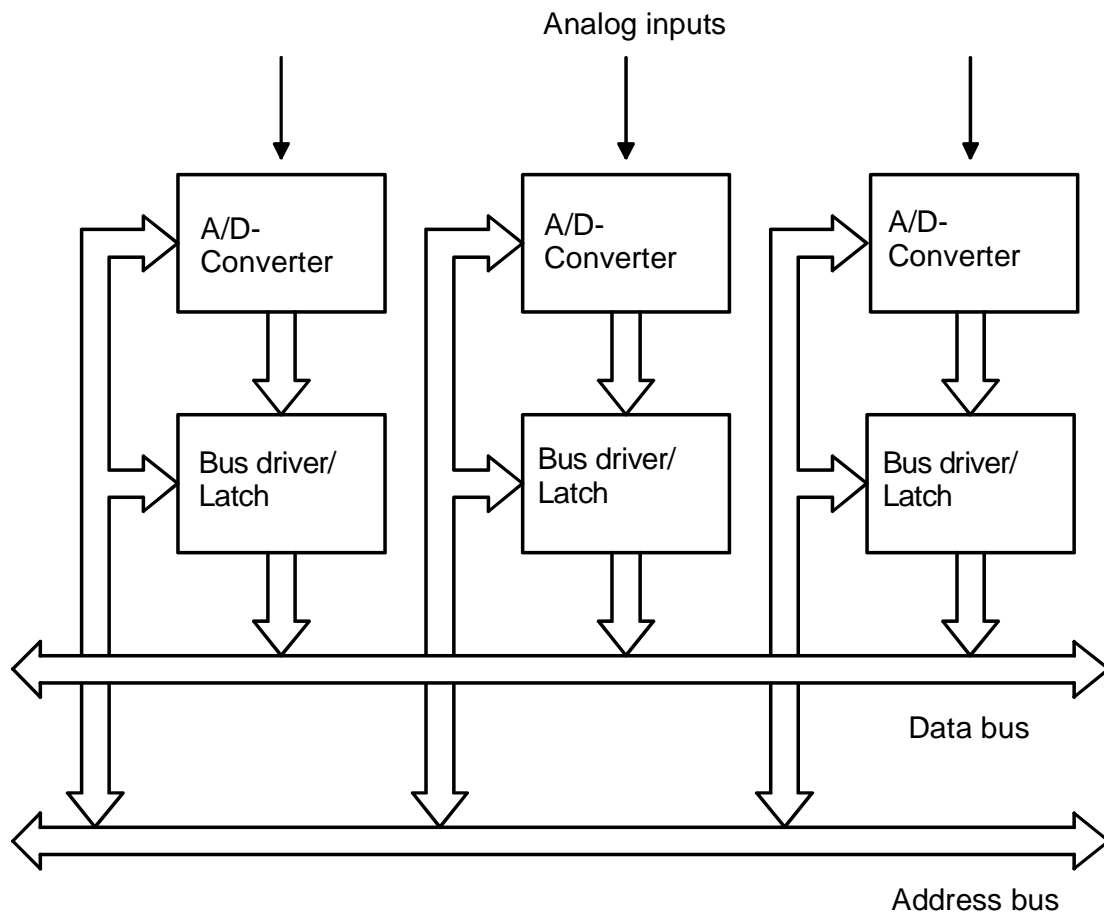


Fig. 1.1.3.4: Digital multiplexing into a microcomputer bus system

If the input signals vary with different speed, an access scheme is appropriate, where the channel with the highest dynamics is accessed more often than the others.

Sample&Hold

Before the analog signal arrives at certain A/D converters it is usually fed into a sample&hold block. The sample&hold block has a signal input, an output and a control input. It knows two modes of operation: hold or follow. In the follow mode the output follows the input, until the hold command comes. Then the sample&hold block holds the voltage of that point of time, when the hold command came. Sample&hold blocks usually have unity gain and are non-inverting. The control input is TTL compatible.

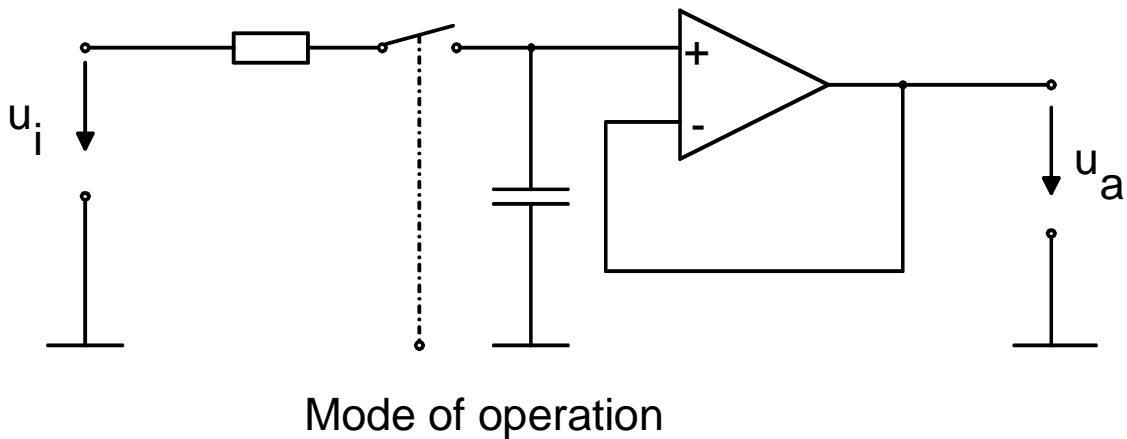


Fig. 1.1.3.5: Basic structure of a S&H block

A sample&hold block in its basic layout consists of a switch and a capacitor. If the switch is closed, the device is in the follow mode, whereas the output follows the input. If the switch is opened the capacitor holds the last input voltage for a certain time, which depends on the capacity and the leakage current. Practical sample&hold blocks have input and output buffers as well as optimized switching technologies to avoid leakage current.

The task of the S&H block is, to keep the input signal of the following A/D converter constant during the conversion interval, while the preceding multiplexer possibly already addresses the next analog channel. After completion of the conversion the S&H block follows again and the cycle can start again.

Often S&H blocks are intergrated in the A/D converter.

The Sample-and-Hold-Characteristics

An ideal Sample&Hold block, also called a "zero order hold element", takes a sample of the input signal in zero time and holds this value for an infinite time with highest precision.

In practice the sample is taken in a time being short compared to the hold time. During the hold period the output signal varies marginally and reduces in that way the accuracy of the entire system.

Sample&Hold specifications:

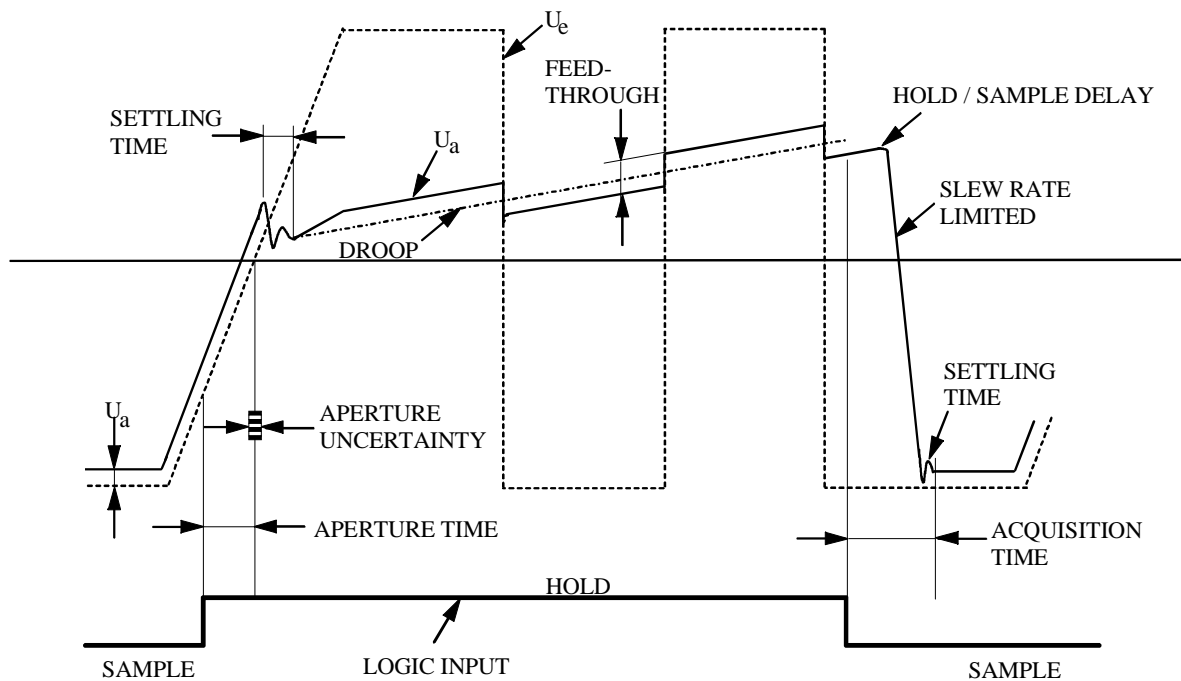


Fig. 1.1.3.6: Parameters of a sample&hold block

Real devices are specified with the deviation from the ideal behaviour:

1. Acquisition Time

The time span from the point of time, when the follow command

was given to the point of time, when the output becomes equal to the input signal and stays inside a specified tolerance band. At the end of the acquisition time the output follows the input.

2. Aperture Time

The time span between the hold command and the point of time, when the switch is completely open.

3. Aperture uncertainty time

The uncertainty in the aperture time, i.e. the difference between the maximum and minimum aperture time.

4. Decay rate or Droop

The variation of the output voltage in time in hold mode.

5. Feedthrough

The amount of the input signal that appears at the output in hold mode. Feedthrough depends on the signal frequency and is sometimes specified as an attenuation in dB.

Besides in A/D conversion like described above S&H blocks are applied for other purposes:

For instance a so called peak follower is synthesized from a S&H and a comparator.

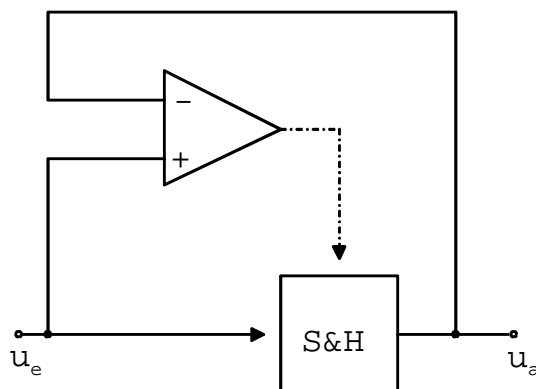


Fig. 1.1.3.7: Peak follower with sample&hold

If the input voltage is higher than the S&H output voltage, the positive output voltage of the comparator switches the S&H into follow mode. If the input voltage becomes smaller than the output of the S&H, the comparator switches the S&H to hold mode until the input voltage again becomes higher than the output voltage.

A/D converters

The analog-to-digital converter is at the core of any data-acquisition system designed to transform data in the form of continuous analog variables into a discrete binary code suitable for digital processing.

The ideal transfer curve for a 3-bit A/D converter is shown in Fig. 1.1.3.8 with the analog levels on the horizontal axis and the digital outputs which correspond to those input values on the vertical axis.

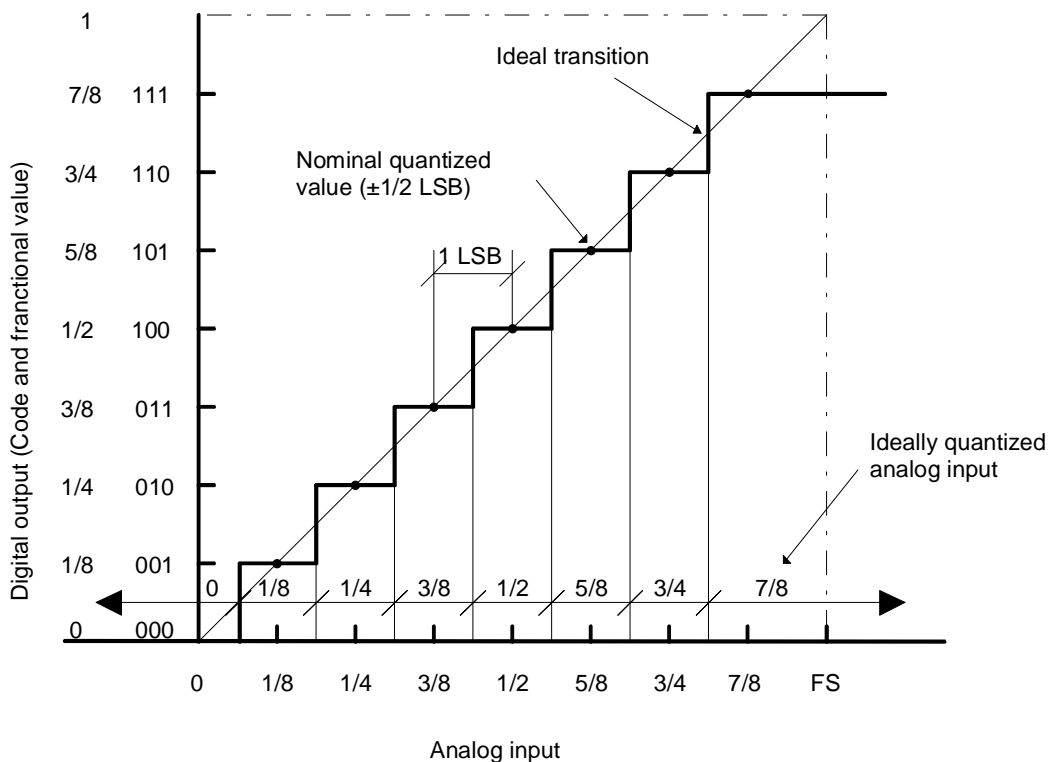


Fig. 1.1.3.8: Conversion relationship for an ideal A/D converter

In order to select the best ADC for a given application, the application engineer should be familiar with the various types of converters available.

The checklist in table 1.1.3.1 outlines the proper priorities for selecting an ADC. First and foremost, the required resolution of the converter must be determined. This will determine the number of recognizable quantization intervals in the ADC transfer function. Since all ADCs exhibit a fundamental quantization uncertainty of $\pm\frac{1}{2}\text{LSB}$, the application engineer must choose an ADC with sufficient resolution to reduce this „digitizing noise“ to an acceptable low value. A useful rule of thumb is that each bit of resolution reduces quantization noise an additional 6 dB. Thus a 10-bit converter exhibits „best case“ quantization noise which is approximately 60 dB below full scale.

Resolution however does not imply accuracy! Many A/D converters have been built with far higher resolution than accuracy (Reason e.g. temperature drift).

Conversion speed is another important selection factor, since it defines the upper limit on system bandwidth. Generally speaking, the conversion speed requirements will dictate the type of converter selected.

- Resolution
- Accuracy
 - Initial (25° C)
 - Drift
- Speed
- Power requirements
- Reference
 - intern/extern
- Interfacing (ser./parallel output)
- Cost
- Size

Table 1.1.3.1: A/D selection features

The supply voltages required have to be available in the automation device.

Applications that use a single ADC can be best satisfied by choosing an ADC with built-in reference. In systems where the reference must be variable, or if a master system reference is available, an ADC requiring an external reference can be used to advantage.

Since most ADCs are interfaced to microprocessors, it logically follows, that interface logic should be included on-chip. Most newer ADCs include at least a three-state buffer for bus interface. However, high-speed data buses with significant activity during the conversion period may inject noise into the ADC and cause erratic or unstable output data. The solution to this problem (generally encountered only on 12-bit and higher-resolution ADCs) is to use an external three-state buffer.

Converter types (Sample)

Type	Features
Integrating (Single slope, dual slope, multi slope)	High accuracy Low speed Low cost
Successive approximation	High speed Accuracy at cost
Tracking (Counter/Comparator)	High speed in track Susceptible to noise
Multicomparator "Flash"	Highest speed High resolution expensive
V/f Converter	Fast responding Continuous serial output

Table 1.1.3.2: A/D converter types

The list in table 1.1.3.2 contains the most popular types, especially those currently produced in integrated-circuit form.

The integrating A/D design, one of the oldest implementations of this function, suffers from speed limitation but can achieve high accuracy at relatively low cost.

The successive-approximation A/D is the most popular for data-acquisition systems, because of its moderately high speed and resolution.

The tracking A/D and the voltage-to-frequency converter can be looked upon as variations of the successive-approximation and the integrator design techniques; in these types the digital data are available on a virtually continuous basis.

The multicomparator ladder (or parallel) design provides the highest speed, at the expense of limited resolution. Most monolithic parallel converters are limited to 6- or 8-bit resolution. The parallel converter consists of a tapped resistor network and an array of $2^N - 1$ comparators (see Fig. 1.1.3.9).

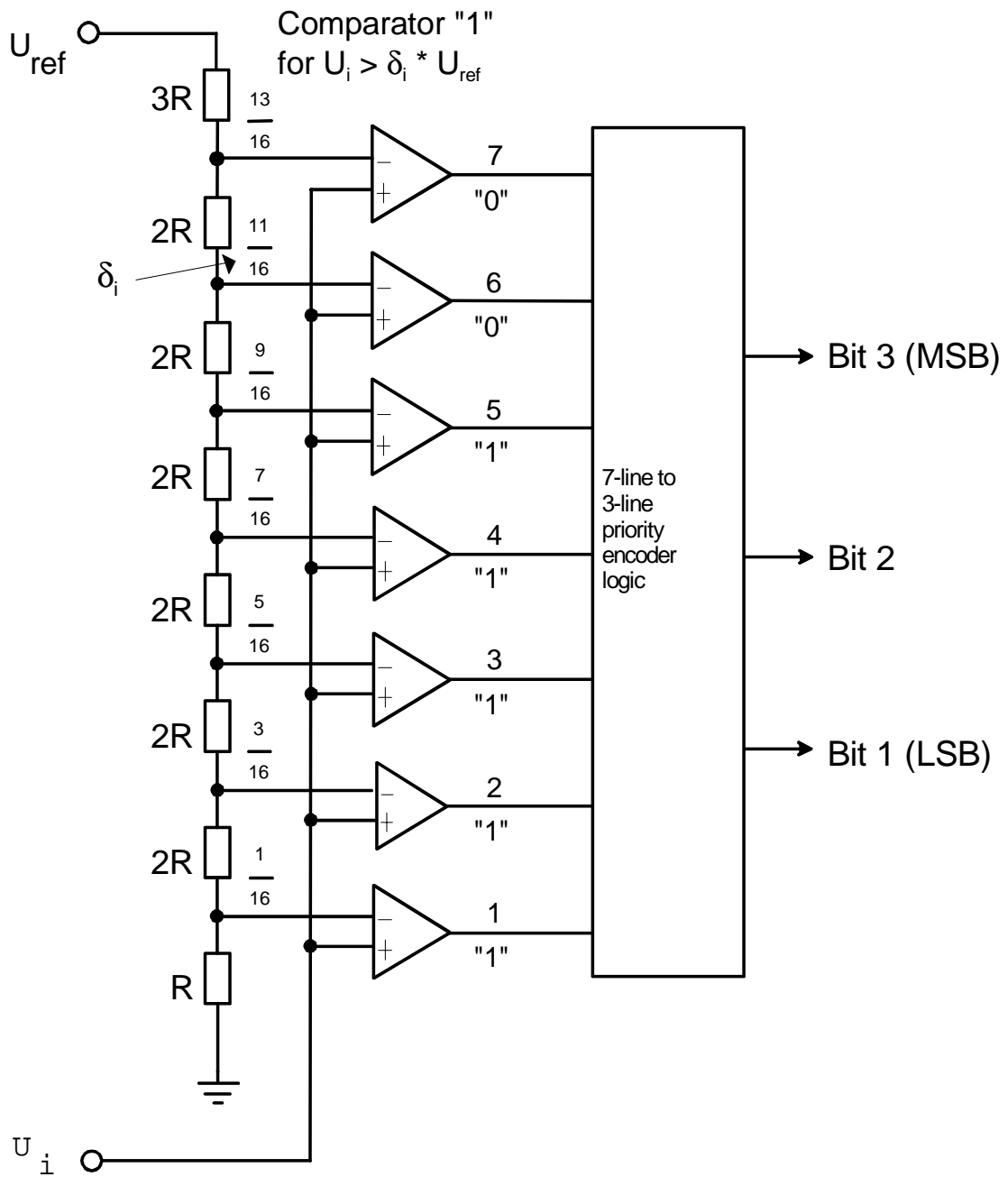
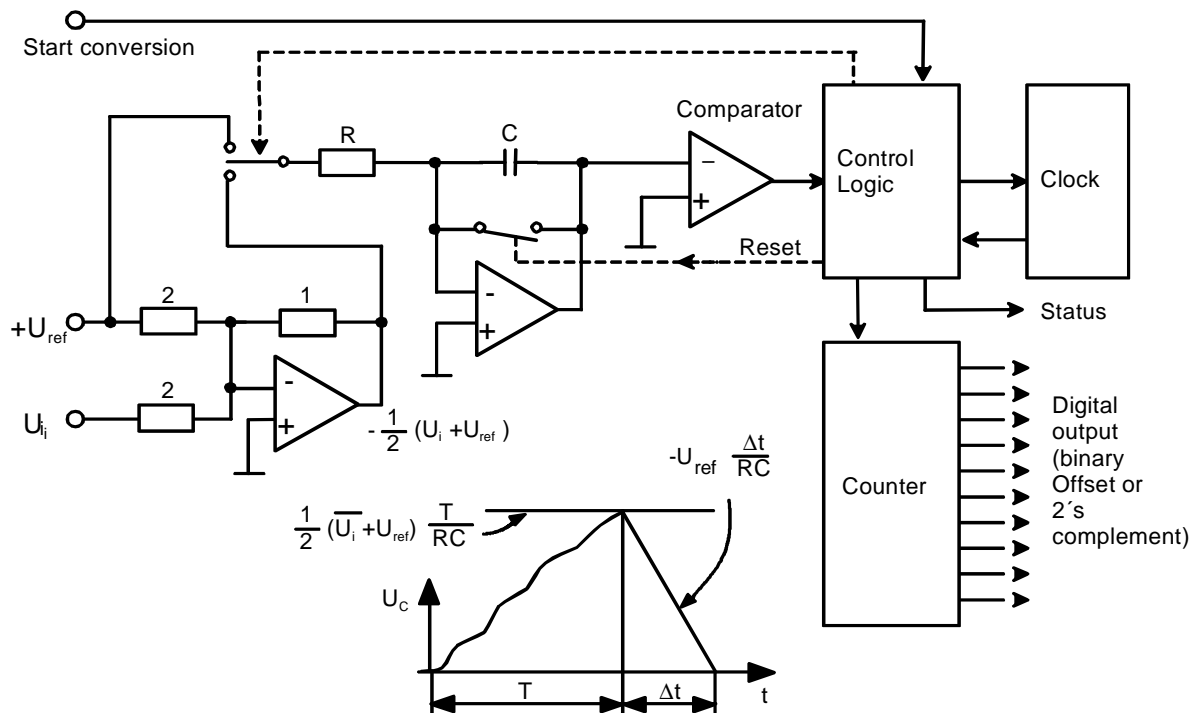


Fig. 1.1.3.9: Multicomparator, "Flash", or parallel converter

Each comparator provides a logic output indicating whether the analog input is above or below a particular fraction of the reference voltage. These logic outputs are then decoded to a more useful code (binary or Gray code, for example). The total conversion time for this type of converter is equal to the sum of the propagation delays of the comparators and the decoding logic. This may be as short as a few tens of nanoseconds, translating into conversion rates of tens of megasamples per second.

The disadvantage of the parallel converter is the limited resolution available. A 10-bit converter would, for example, require 1024 matched precision resistors, 1023 high speed comparators, and an extensive logic array. Integrating this complex a device remains currently beyond the practical state of the art.



Comparator is tripped when:

$$\frac{1}{2} (\overline{U_i} + U_{ref}) \frac{T}{RC} = U_{ref} \frac{\Delta t}{RC}$$

$$\frac{\Delta t}{T} = \frac{1}{2} \left(\frac{\overline{U_i}}{U_{ref}} + 1 \right) \quad \text{(Discharging time is proportional to the mean value of the input voltage)}$$

Fig. 1.1.3.10: Basic diagram of a dual-slope integrating A/D converter

The concept of the „dual slope“ integrating A/D is simple. A current proportional to the input signal charges a capacitor for a fixed length of time - the capacitor is then discharged by a current proportional to the reference until the starting point is crossed. The discharge time is then directly proportional to the average value of the input signal.

The integration provides rejection of high-frequency noise and averaging of changes that occur during the sampling period. The fixed average period also makes it possible to obtain „infinite“ normal-mode rejection at frequencies that are integral multiples of $1/T$.

One of the most important uses of D/A converters is in the core of certain types of A/D converters. In these A/D converters, the output of the internal DAC is compared to the input level; the logic section of the A/D uses the information from the comparator (internal level above or below input) to modify the code to the DAC until the two levels are brought as close together as possible (within the resolution limit of the DAC).

In the simple tracking A/D shown in Fig. 1.1.3.11, an up/down counter drives the DAC in the proper direction to match the input in response to commands from the comparator.

This converter can follow small changes quite rapidly (it will follow 1 LSB changes at the clock rate), but it will require the full count to acquire full-scale step changes.

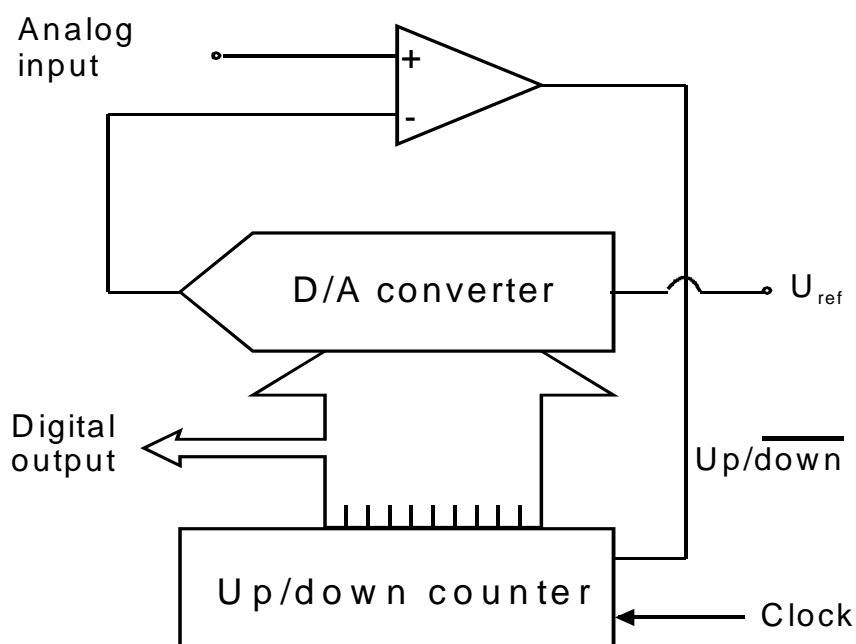


Fig. 1.1.3.11: Tracking ADC

The successive-approximation converter is more popular for high speed data acquisition systems since it is always able to complete a conversion with 8, 10, or 12 „counts“ or comparisons (for 8, 10, or 12-bit resolution).

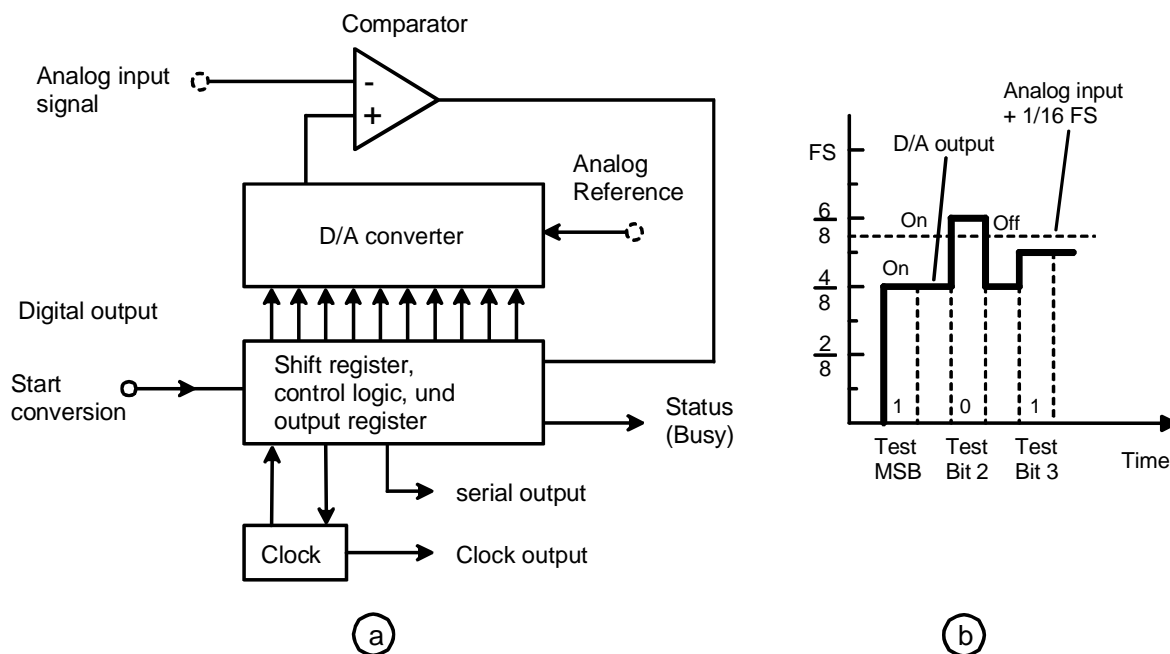


Fig. 1.1.3.12: Successive approximation A/D converter: (a) block diagram; (b) three-bit weighting

If a weight equal to one-half the smallest weight is added to the unknown side, it can be shown that the measurement is always accurate to one-half the smallest weight.



Fig. 1.1.3.13: Usual configuration of a data acquisition channel

1.1.3.3 Conversion of digital into analog values

A digital-to-analog converter converts digital code to an analog parameter by switching successive-ratio analog bit-weights to a common summing point. The concept of the binary weighting of switched analog levels is illustrated in Fig. 1.1.3.14:

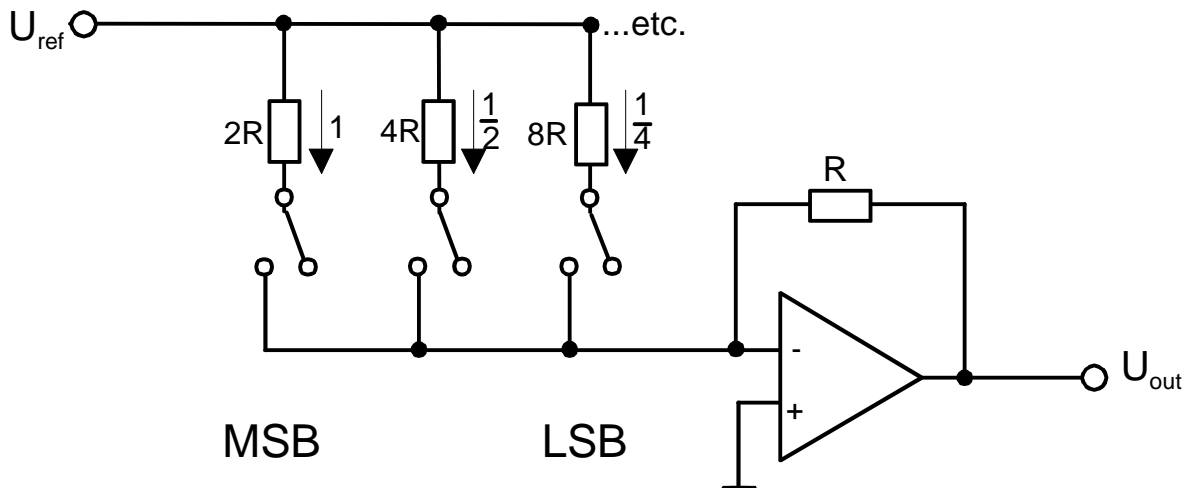


Fig. 1.1.3.14: Basic D/A conversion technique, continuous binary-weighted

The D/A converter can be thought of as a digitally controlled potentiometer that produces an analog output that is a normalized fraction of its „full-scale“ setting. The output voltage or current depends on the reference value chosen to determine „full-scale“ output. If the reference may vary in response to an analog signal, the output is proportional to the product of the digital number and the analog input.

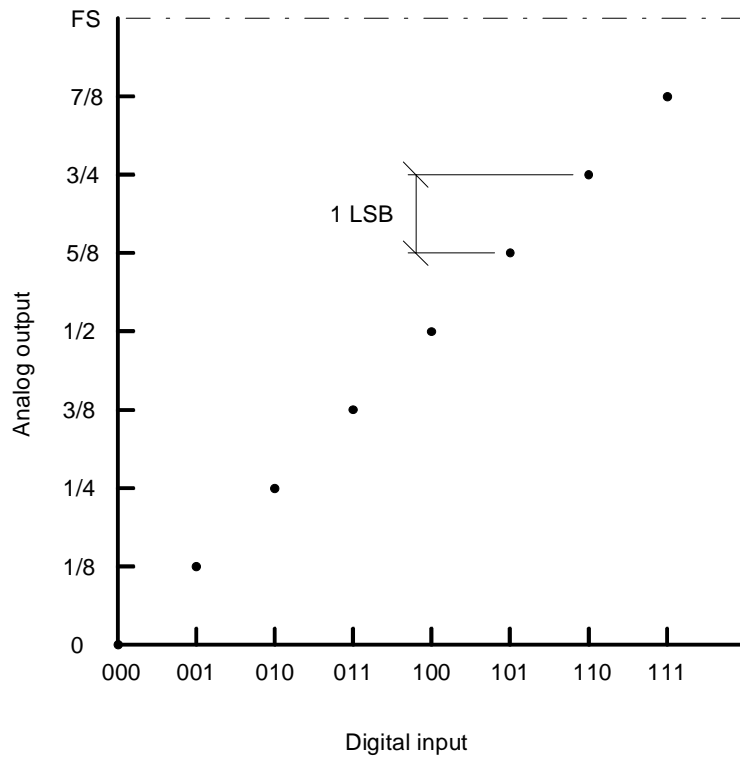


Fig. 1.1.3.15: Conversion relationship for an ideal 3.bit D/A converter

10 V Full scale

Resolution:	Weight of the least significant bit (LSB)
8 bit	1 LSB = 39,1 mV = 0,391 % FS = 3906 ppm 1/2 LSB = 19,5 mV = 0,2 % FS = 1953 ppm
10 Bit	1 LSB = 9,77 mV = 0,1 % FS = 977 ppm 1/2 LSB = 4,88 mV = 0,05 % FS = 488 ppm
12 bit	1 LSB = 2,44 mV = 0,024 % FS = 244 ppm 1/2 LSB = 1,22 mV = 0,012 % FS = 122 ppm
14 bit	1 LSB = 0,61 mV = 0,006 % FS = 61 ppm 1/2 LSB = 0,305 mV = 0,0031 % FS = 31 ppm
16 bit	1 LSB = 0,153 mV = 0,0015 % FS = 15 ppm 1/2 LSB = 0,076 mV = 0,00076 % FS = 8 ppm

Fig. 1.1.3.16: Data converter error terminology

Besides the unavoidable quantization error according to Fig. 1.1.3.16 possibly also the following errors occur in the static accuracy:

- offset errors
- gain errors
- linearity errors (relative accuracy)
- differential Nonlinearity (step width)

If the differential nonlinearity exceeds certain limits, the converter becomes nonmonotonic and the A/D converter using it may miss one or more codes.

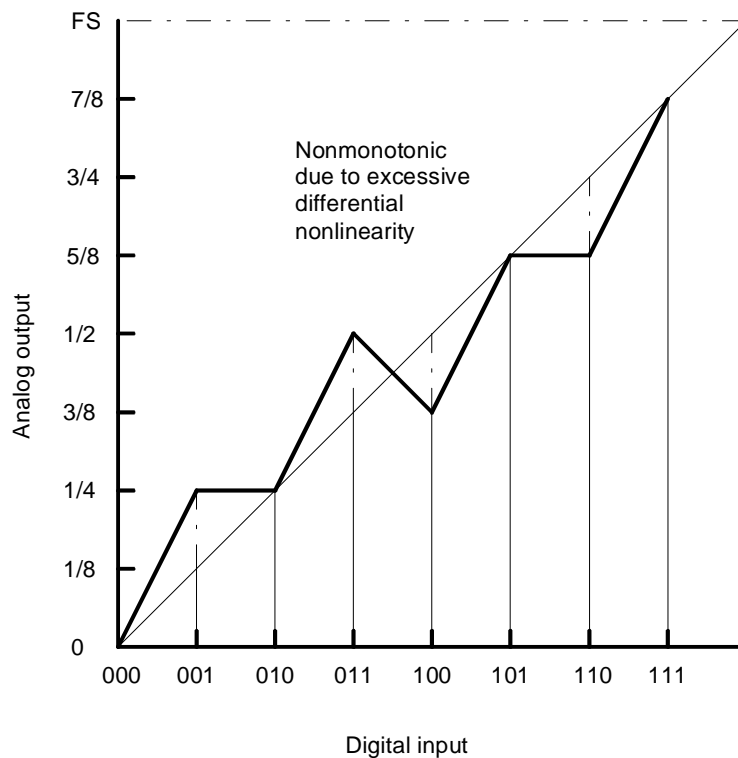


Fig. 1.1.3.17: Nonmonotonicity

The temperature performance of a D/A can be completely described by its offset, gain, and differential-nonlinearity temperature coefficients.

Monotonic behaviour is achieved if the differential nonlinearity is less than 1 LSB at any temperature in the range of interest.

In most new IC designs not the circuit in Fig 1.1.3.14 (continuously binary weighted) but the inverted R-2R or current switching ladder is being used:

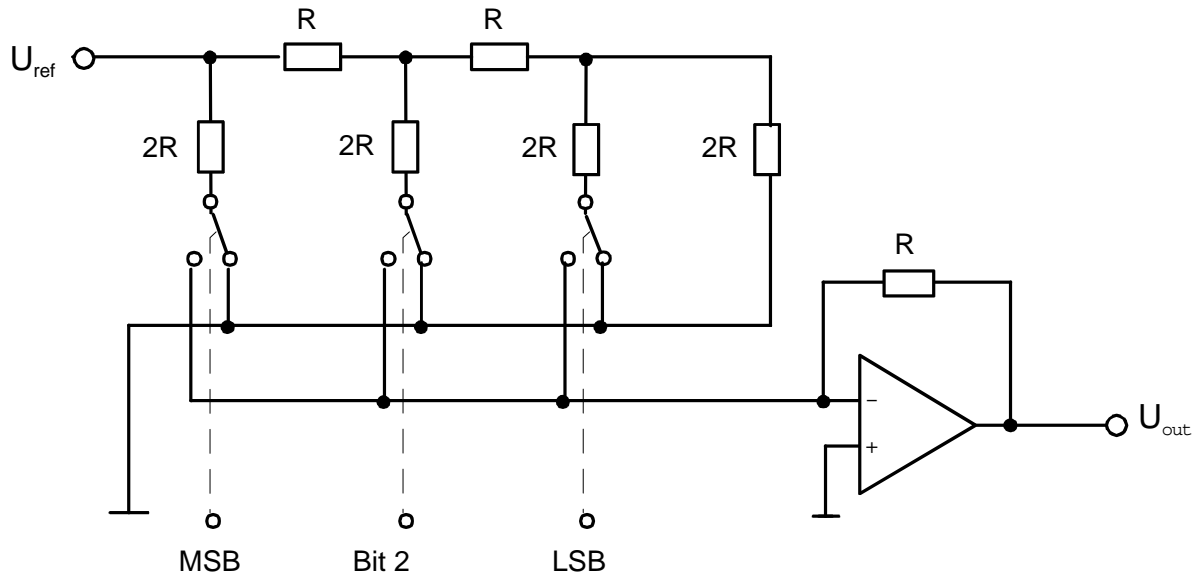


Bild 1.1.3.18: R-2R D/A converter

Important parameters of D/A converters:

- resolution
- voltage and/or current output
- Reference internal/external
- Latches integrated or not

1.2 Interfaces to other automation devices

1.2.1 -- Chapter is left blank --

1.2.2 Serial Point-to-Point-Link (RS232-C and TTY)

RS232-C is the most commonly used communication protocol. It uses the logical voltage levels according to Fig. 1.2.2.1. The RS232-C-Signals are usually assumed to be nominal +12V or -12V. Thereby the +12V represent the logical "0" and the -12V the logical "1".

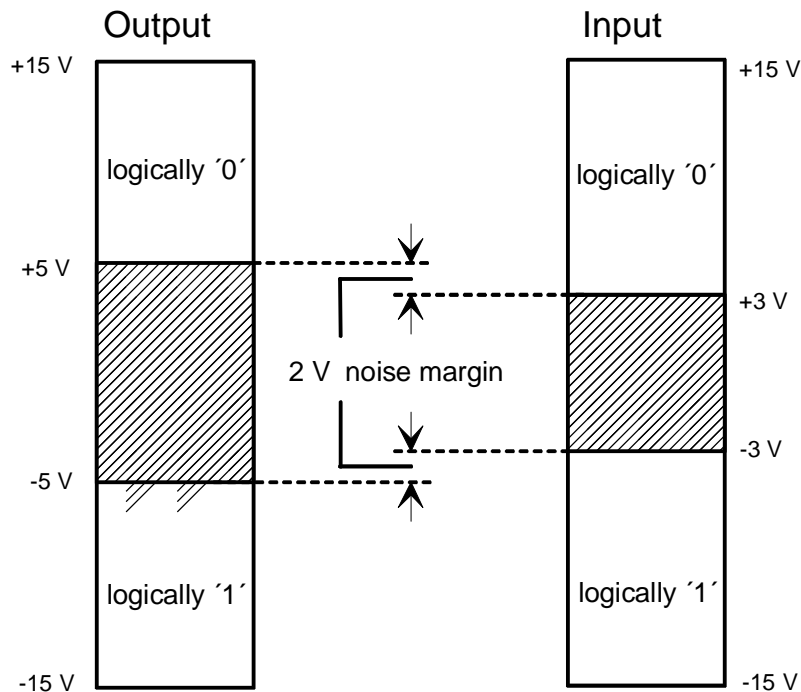


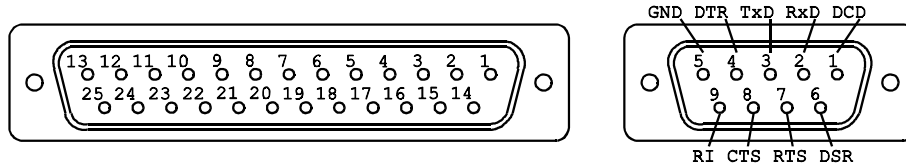
Fig. 1.2.2.1: RS232-C-voltage levels

The wide bands for the two logic levels and the wide band in between provide a substantially higher noise immunity than e.g. the TTL-level.

Further electrical specifications are e.g.:

- The voltage at open outputs may not exceed 25V
- A driver circuit must be able to survive a short circuit on each terminal without damage for itself or other connected circuits

The mechanical arrangement of the RS232-C-interface is also defined, so that the pin assingment of the 25-lead Cannon plug is like shown in Fig. 1.2.2.2:



1	Protective 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	Data carrier detect (for Modem)
9/10	reserved
11	not used
12	Secondary received line signal detector
13	Secondary clear to send
14	Secondary transmitted data
15	Transmit signal element timing
16	Secondary received data
17	Receive signal element timing
18	not used
19	Secondary request to send
20	Data terminal ready (DTR)
21	Signal quality detector
22	Ring indicator (RI)
23	Data signal rate detect
24	Transmit signal element timing
25	not used

Fig. 1.2.2.2: Pin asignment of the 25-lead RS232-C interface (9-lead variant not standardized, therefore manufacturer specific)

Most of the computer interfaces utilize only very few of these pins. This is possible, because RS232-C allows variability in its definition. Fig. 1.2.2.3 shows the complete arrangement,

that allows a full duplex connection together with a 3-wire-connection, used for the same purpose and also called RS232-C.

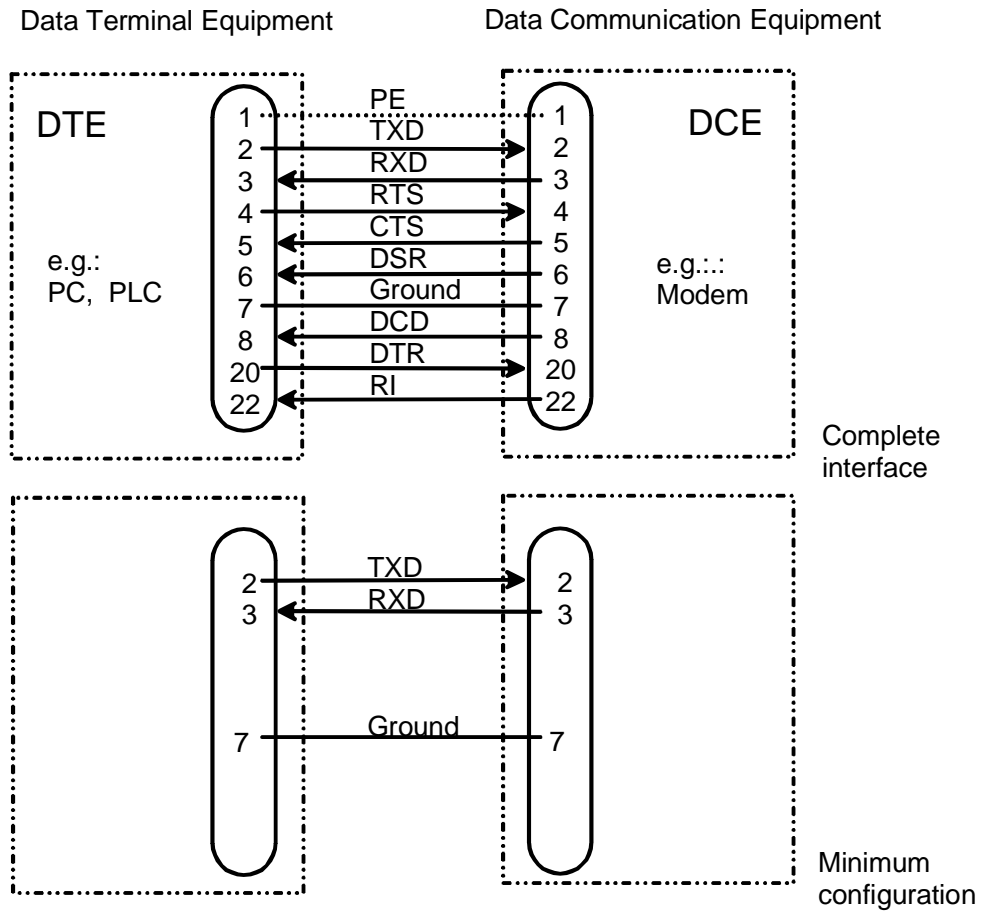


Fig. 1.2.2.3: Full duplex signal traffic on the RS232-C-interface

The standard RS232-C defines the data transmission between two equipments of different type, namely between a "Data Communication Equipment (DCE)" and a Data Terminal Equipment (DTE)". The send data (TxD) are an output of the DTE and the receive data (RxD) are an input to the DTE. Unfortunately it is also possible, to connect devices of the same type. Then in the connection cable the assignment of pins 2 and 3 has to be twisted! The same applies analogously to the signal wires at the terminals 4 and 5 as well as 6 and 20.

Between the two extrema in Fig. 1.2.2.3 there are a number of further possibilities, which represent all an improvement against the 3-wire solution. Full-Duplex data transmission according to the RS232-C protocol takes place under utilization of the signals Request To Send RTS und Clear To Send CTS (see timing diagram in Fig. 1.2.2.4)

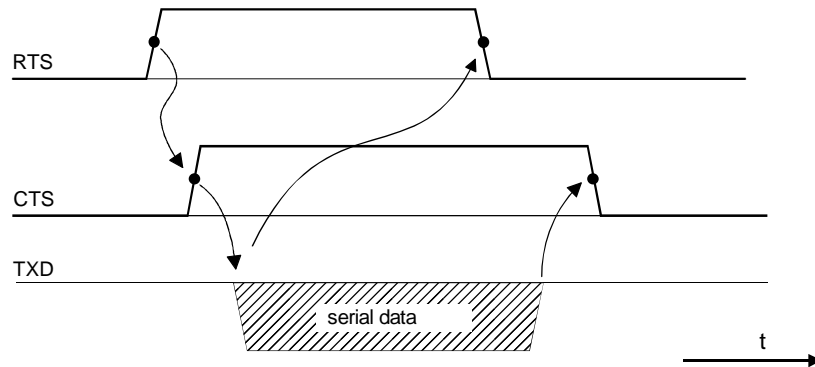


Fig. 1.2.2.4: Timing diagram of the serial data transmission

There are a number of restrictions with the RS232-C, especially the maximum transmission distance of approx. 20 m. For this reason often the so called current interface is utilized, which is able of serving up to a transmission distance of approx. 300m.

The so called 20-mA interface uses 20 mA to represent a logical "1" and 0 mA for logical "0". The necessary arrangement is displayed in Fig. 1.2.2.5. The minimum number of strands for full-duplex operation is 4.

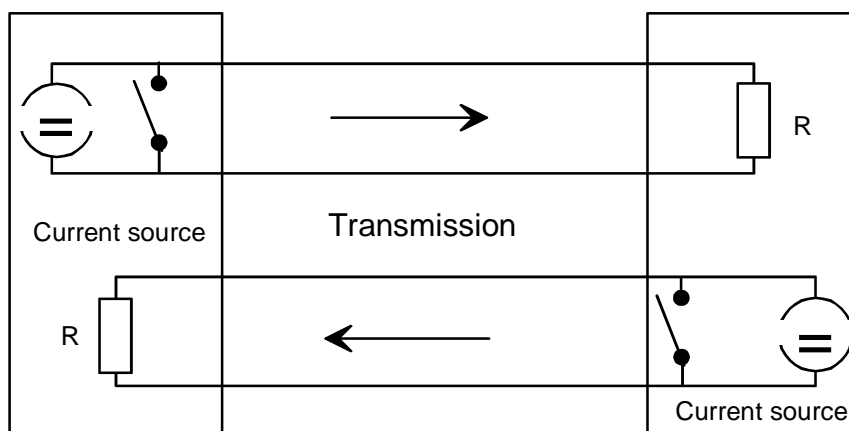


Fig. 1.2.2.5: Full duplex current interface

On the RS232-C interface a Byte-oriented asynchronous data transfer takes place. "Byte-oriented" means, that the smallest effective data unit is not a single bit but a combination of 5 - 8 bits. Such a character is stored in the memory of each communication partner in a byte. The word "asynchronous" means, that the sender does not send a clock signal together with the data. The receiver has to synchronize itself to the data stream received again and again. This happens on the basis of a transmission rate negotiated between sender and receiver, the Baud rate in bits/s.

The transmission of a character starts with the so called start bit. The following bits are data bits, 5 to 8 in number. The data bits are followed by an optional parity bit and one, one and a half or 2 stop bits. The Baud rates are standardized with the following values: 50,75,110, 134, 150, 300, 600, 1200, 1800, 2400, 3600, 4800, 7200, 9600 and 19200.

Digital asynchronous data transmission

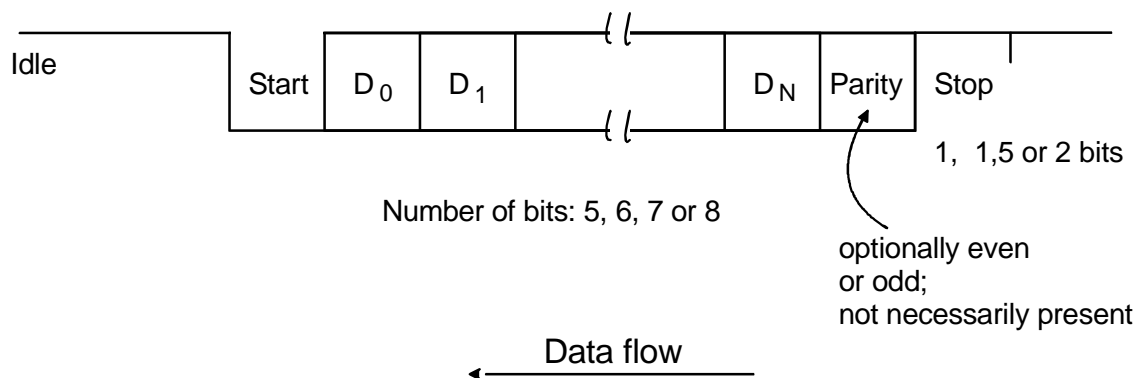


Fig. 1.2.2.6: General format of a character on the asynchronous interface

1.2.3 Networks

1.2.3.1 Topologies

For economic and operating efficiency and operational reliability the logical arrangement and the kind of connection of the communication partners in the network - thus the topology of the network - is a decisive factor.

There are three logically possible basic topologies: Star, ring and bus (Fig. 1.2.3.1)

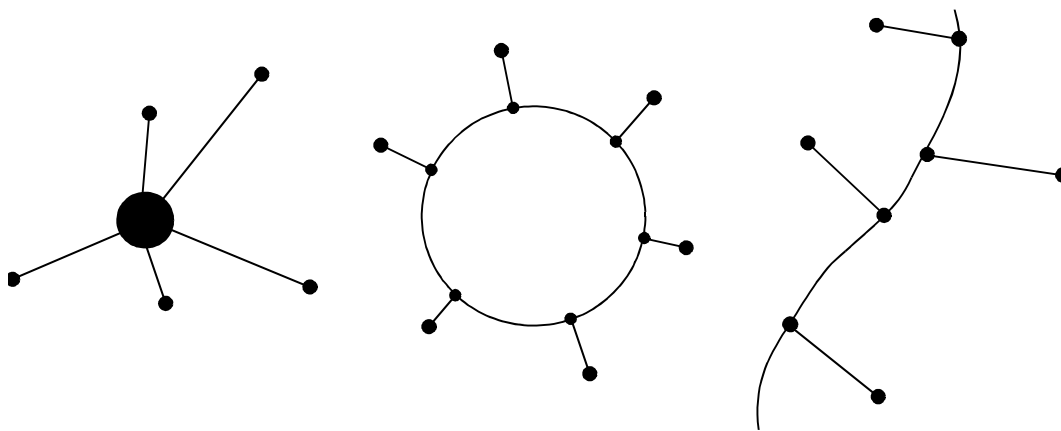


Fig.1.2.3.1: Star, ring and bus topology

Star networks

In a star network the routing is taken in hand by a single node, to which all participants are connected. In popular multi-user-systems the star node is formed by the central computer, which is accessed by a number of terminals. Though in such systems it is possible to deposit messages for other participants; a direct communication between the single connected participants does not take place in this implementation of the star topology. In real star shaped networks nodes are used, which allow data transmission from one participant to the other and perform if necessary data preprocessing or protocol conversion ("Intelligent node").

The common disadvantage of the star topology is the fact that in case of a failure in the central node all the communication paths are disrupted. Furthermore the node becomes bulky in hardware and software, if it shall be able to hold up multiple connections.

Ring network

The ring network avoids the disadvantage of a central node and connects each participant via an own node with its right and left partner until the circle is closed. Because the intelligence of a central node is now relocated into a number of nodes, the structure of these nodes is fairly simple.

In a ring network the information is transmitted in **one** direction from node to node. There is no way to be found ; each information is simply passed on, until it reaches the recipient. This requires from the single nodes only the ability to evaluate addresses and to recognize them as the the own one. From this fact results low cost for the termination.

A disadvantage of this topology is, that the failure of one station paralyzes the whole ring, or costly measures have to be taken to avoid this.

An example for a practically implemented ring network is IBM's Ring network (IBM-Token-Ring).

Bus network

The most modern network topology is the "Bus". Like the ring the bus network avoids a central node, connects instead a number of nodes with their neighbours however without forming a ring. Messages spread from a sending node to both sides and are received bei the destination node.

The disadvantages of other topologies are avoided; instead the following advantages arise:

- A central station is not necessary and the network control

is distributed to the nodes/participants connected

- easy reconfiguration; nodes/participants can be added or switched off without any influence on the operation of the net
- failure of a node/participant does affect neither the other participants nor the net. Merely a damage of the transport medium (Cable break, short circuit) can bring the net out of service.

As transport medium for bus networks besides coaxial cable twisted pair cables, optical fiber, and radio communication are applicable. A typical example for a network with bus topology is PROFIBUS.

1.2.3.2 Transmission methods

1.2.3.2.2 RS 485

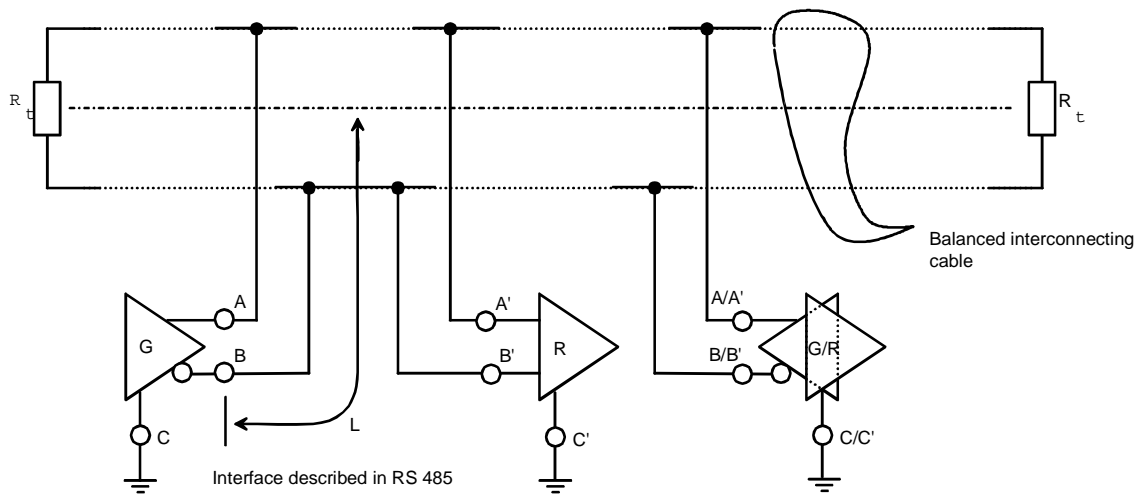
This standard specifies the electrical characteristics of senders ("Generators") and receivers for the interchange of binary signals in multipoint interconnections of digital equipment.

The circuits whose characteristics are specified herein will be utilized normally in data, timing or control interconnections where the data signalling rate is up to 10 megabit/s.

Electrical characteristics

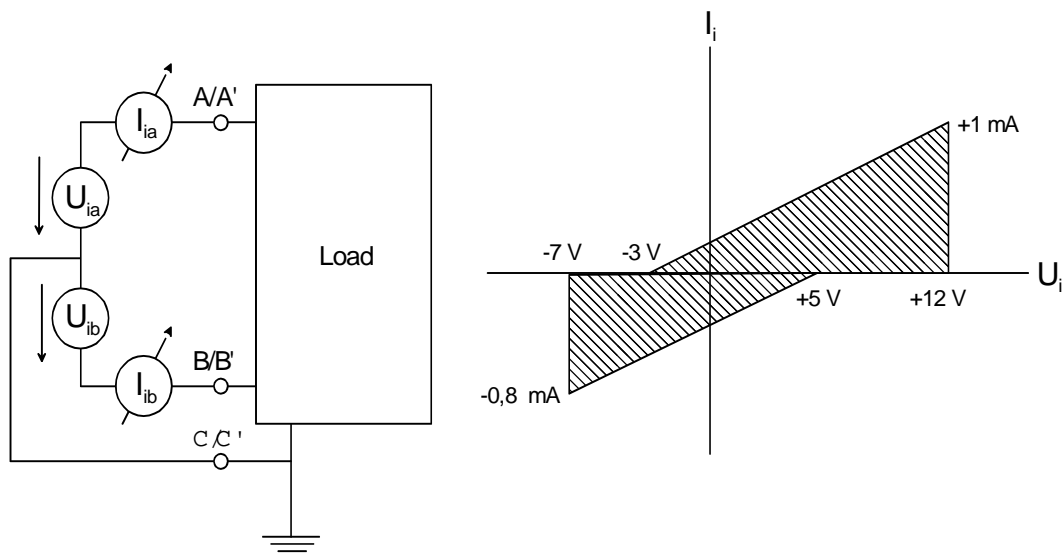
Fig. 1.2.3.5 shows an interconnection application of generators and receivers according to RS 485. The elements in the application are: generators, receivers, transmission cables, and termination resistances. The load on an active generator shall be defined in terms of unit loads. The load on the system caused by each receiver and passive generator shall be specified by the number of unit loads each presents. The electrical parameters specified in RS 485 are selected so that a generator can drive a total load having the value of 32 unit loads and an effective total termination resistance as low as 60 ohms.

Load characteristics



- G - Generator
- R - Receiver
- G/R - Combination Generator/Receiver
- L - Length of stub, the guidelines assume length of stub to be effectively zero
- R_t - Termination resistance; location and value are not specified in this standard, but a generator can driver 32 unit loads plus two termination resistances of 120 ohms each.

Fig. 1.2.3.5: RS485 as multipoint interconnect application



A load is a passive generator, a receiver or a combination generator/receiver

Fig. 1.2.3.6: Unit load input current-voltage measurement

In order that the current required from a generator in the active state be limited to a practical value, the loading effect of generators in the passive state and receivers is specified in terms of the loading produced by a hypothetical unit load. With

the voltage U_{ia} (or U_{ib}) ranging between -7 V and +12 V volts, while U_{ib} (or U_{ia}) is held at 0 volts (grounded), the resultant input current I_{ia} (or I_{ib}) shall remain within the shaded region shown in the graph in figure 1.2.3.6 for one unit load.

Generator characteristics

A generator is either in the passive or in the active state. While in the passive state, the generator load characteristics shall be specified in terms of the number of equivalent unit loads. While in the active state, the generator electrical characteristics are specified in accordance with the measurements illustrated in fig. 1.2.3.8. A generator circuit meeting these requirements results in a low impedance balanced voltage source, that will produce a differential voltage applied to the interconnecting cable in the range of 1.5 volts to 6.0 volts. The signalling sense of the voltages appearing across the interconnection cable are defined as follows:

- a) The A terminal of the generator shall be negative with respect to the B terminal for a binary 1 (MARK or OFF) state.
- b) The A terminal of the generator shall be positive with respect to the B terminal for a binary 0 (SPACE or ON) state.

Open circuit measurement

For either binary state, the magnitude of the differential voltage (U_o) measured between the two generator output terminals shall be not less than 1.5 V and not more than 6.0 V; and the magnitude of U_{oa} and U_{ob} measured independently between each generator output terminal and generator circuit ground shall be not more than 6.0 V.

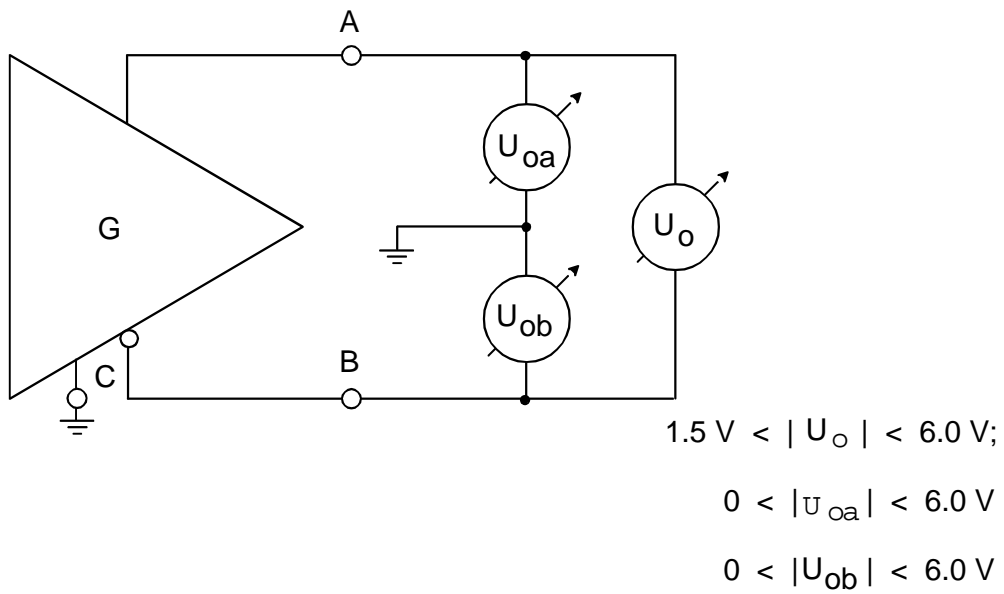


Fig. 1.2.3.8: Generator open circuit measurement

Simultaneous activity of multiple generators on the cable must be possible without destruction of the generators. Therefore the maximum current in current source mode and in current drain mode as well is limited to a peak value of 250 mA.

Receiver characteristics

The receiver electrical characteristics are specified in accordance to the load specifications above (Unit load). A circuit meeting these requirements results in a differential receiver having a high input impedance and a threshold that lies in the region between -0.2 und 0.2 volts.

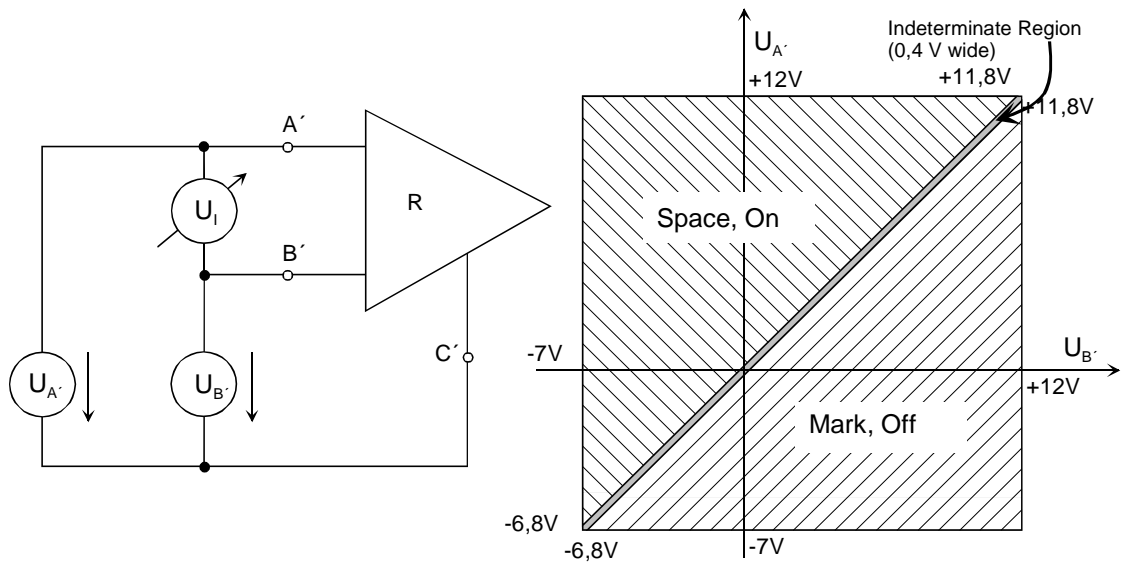
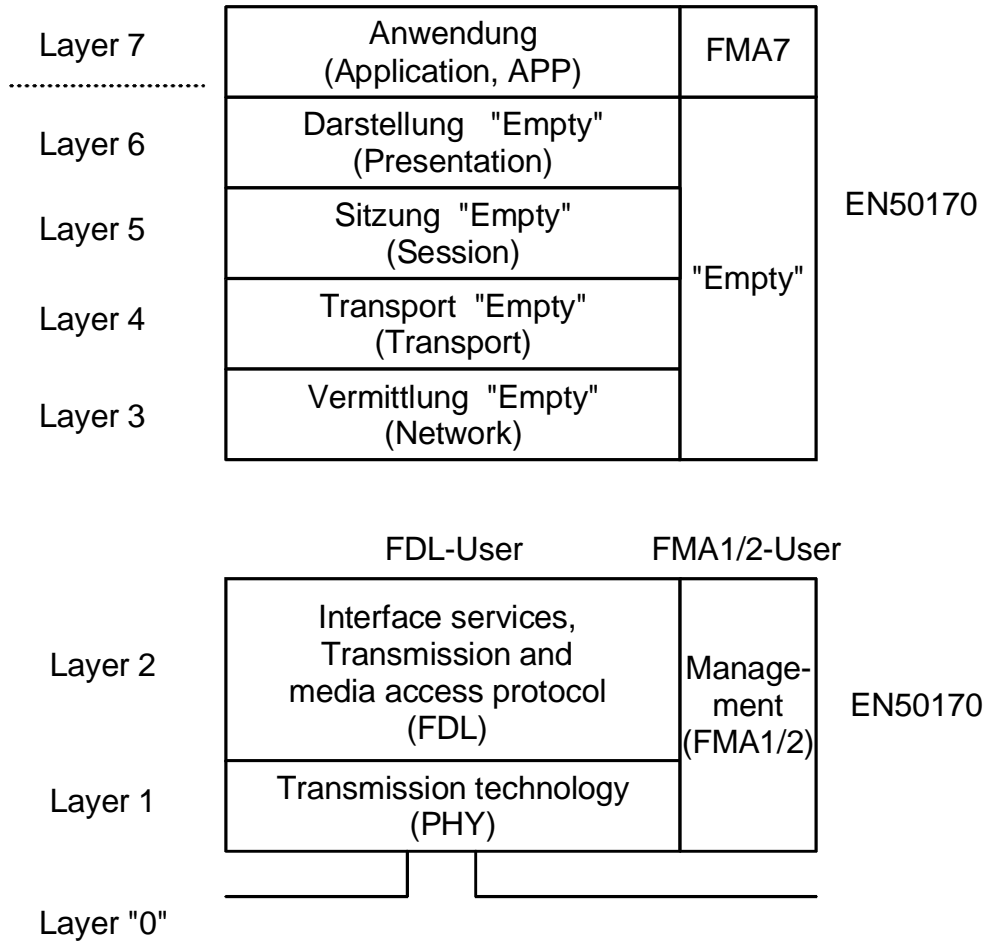


Fig. 1.2.3.9: Receiver input voltage range

1.2.3.4 Selected communication systems

1.2.3.4.1 PROFIBUS

1.2.3.4.1.1 The layer model of Profibus



FDL = Fieldbus Data Link; FMA = Fieldbus Management

Fig. 1.2.3.33: ISO layer model of PROFIBUS

As transmission technology Profibus utilizes RS 485 or fiber optics. Transmission rates are standardized up to 12 MBaud. Data are transmitted asynchronously in UART format.

1.2.3.4.1.2 Bus access and transmission protocol (Data Link Layer, FDL)

The PROFIBUS utilizes a bus access with a **hybrid bus access method**, namely following the principle of **Token Passing** and additionally following the **Master Slave** principle. The access control is located in every **active participant**. The **passive participants** are neutral with regard to bus access, i.e. they don't exercise any autonomous sending activity, but only send on request.

The initiative of communication always emanates from that active participant, who gets the bus access right, the token. The Token is passed in a logical ring from an active participant to another and determines the point of time, when an active participant is allowed to access the medium.

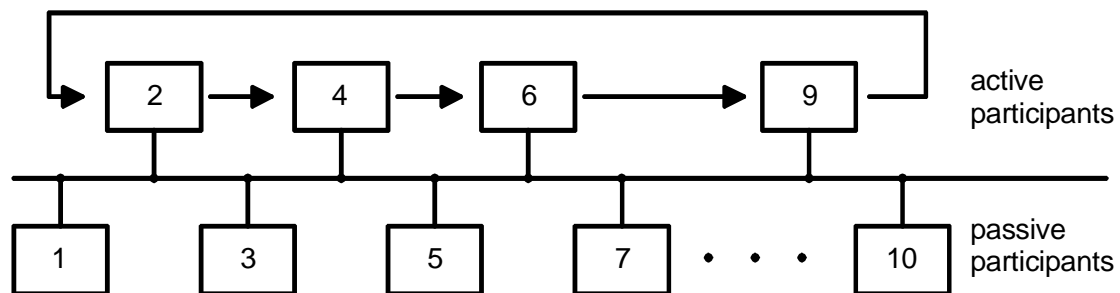


Fig. 1.2.3.34: Logical Token Ring of active participants with direction of Token passing

If the logical ring consists of only one active participant and the network contains multiple passive participants, then this corresponds to a pure Master-Slave-System.

The following faults and special operational states in the system are managed:

- 1) Multiple tokens
- 2) Lost token
- 3) Failure in token passing
- 4) Multiple usage of identical participant addresses
- 5) participants with faulty sender/receiver
- 6) Adding and removing of participants during operation
- 7) Arbitrary combinations of active and passive participants

1. Transmission modes of operation and FDL control

The message exchange takes place in **cycles**. A message cycle consists of a request telegram (request- or send/request frame) of an active participant and the respective acknowledgement or response frame of an active or passive participant. Hereby **user data** may be transmitted in the request (Send) telegram and in the response telegram as well. The acknowledgement telegram does not contain user data.

The complete message cycle is only suspended in the token passing and in sending data without notification of receipt - (e.g. necessary for broadcast messages). In both modes of operation an acknowledgement is not applied.

In case of broadcast messages all other participants are addressed by one active participant (Initiator) with a global address (max. participant address, all address bits binary "1") at the same time (Application e.g. time synchronisation).

1.1 Token rotation time

When an active participant has got the token, the measurement of the token rotation time begins. At the next taking over of the token this time measurement ends for the expired cycle with the real rotation time T_{RR} as the result. At the same time a new measurement of the next rotation time begins. The T_{RR} is important for the handling of low priority message cycles.

For keeping of a certain system reaction time implied by the application a target rotation time T_{TR} of the token in the logical ring must be given.

As **system reaction time** the maximum difference of time (worst case) between two successive, high priority message cycles of an active participant, related to the FDL interface at full load of the bus capacity is defined.

Every active participant can independently of the real token rotation time basically always perform one high priority (High) message sequence per token receipt.

For the handling of the low priority (low) message cycles at the time of the handling the T_{RR} has to be less than T_{TR} , otherwise the participant has to keep back queued low priority message cycles and transmit them at the next token receipt or at the following ones.

1.2 Acyclical request- or send/request operation

In the acyclical request- or send/request operation sporadically single message cycles are handled. The FDL control of the active participant initiates this operation at the token receipt as a result of a request by the user. If multiple request are pending this mode of operation can be performed until the max. admissible token rotation time has expired.

1.3 Cyclical send-request operation

In polling mode the active participant addresses other participants cyclically according to a predetermined order, the poll list, with the request "Send and Request Data low". The poll list is handed over by the user of the active participant to the FDL control. In this list all active and passive participants to be polled are marked. Those participants, who do not respond during the polling procedure in spite of request repetitions, are marked as "not operative". In later request cycles these participants are addressed on a trial basis without repetition. If thereby participants reply, they are marked as "operative".

2. Telegram structure

2.1 Telegram characters (UART characters)

Each telegram is made up by a number of telegram characters, the UART characters. The UART character (UC) is a Start-Stop character for asynchronous transmission and has the following structure **when used in PROFIBUS**:

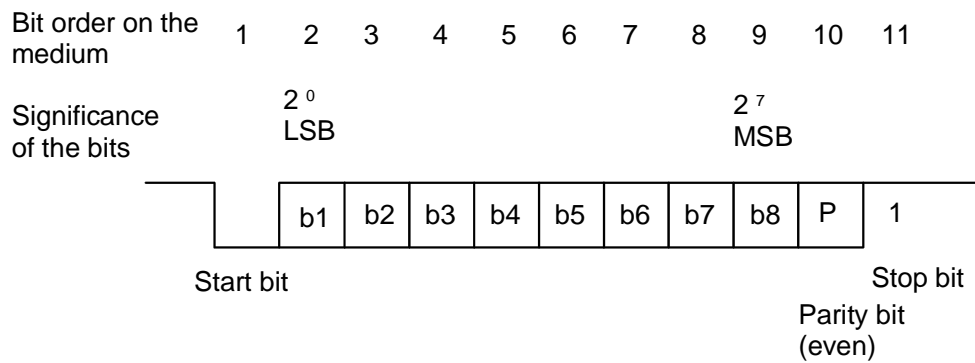


Fig. 1.2.3.35: Character transmission

Transmission rule

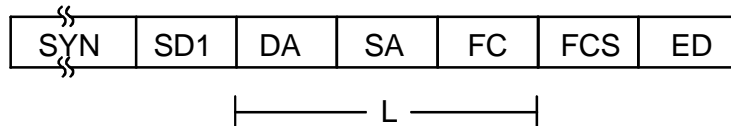
Each UART character has 11 bits, namely a start bit (ST) with binary "0" signal, 8 Information bits (I) with binary "0" or "1" signal, a even parity bit (P) with binary "0" or "1" signal and a stop bit (SP) with binary "1" signal.

3. Telegram formats

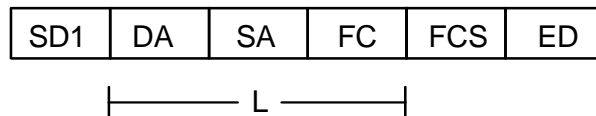
The figures contained in the following text don't show sequences (Request ---> Acknowledge or response), but telegram formats of equal category (Hamming distance HD=4, fixed length with/without Data field and variable length), i.e. different acknowledge or response telegrams can follow the request telegrams (see Chapter 5).

3.1 Formats with fixed length of the information field without data field

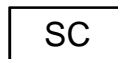
A) Format of the request telegram (Request-Frame):



B) Format of the acknowledgement frame:



C) Format of the short acknowledgement



SYN = Synchronisation bits, min. 33 bit idle state

SD1 = Start byte 1 (Start Delimiter), Code: 10H

DA = Destination address

SA = Source address

FC = Frame Control

FCS = Frame Check Sequence

ED = End Delimiter, Code: 16H

L = Length of information field, fixed number of Bytes L = 3

SC = Single character, Code: E5H

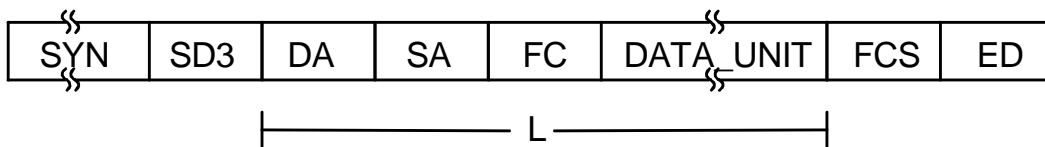
Fig. 1.2.3.36: Frame format with fixed length without data field

Transmission rules

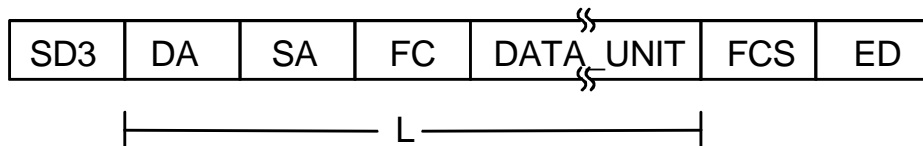
1. Idle state of the wire corresponds to a binary "1"-signal.
2. Before each **request** telegram a minimum length of 33 bits (Syn time) idle time is required.
3. Between the UART characters of a telegram no idle states are allowed.

3.2 Formats with fixed length of information field with data field

A) Format of the request telegram (Send/Request-Frame):



B) Format of the response telegram (Response-Frame):



SYN = Synchronisation bits, min. 33 bits idle state (Idle)

SD3 = Start byte 3 (Start delimiter), Code: A2H

DA = Destination address

SA = Source address

FC = Frame Control

DATA_UNIT = Data field, fixed length (L-3) = 8 bytes

FCS = Frame Check Sequence

ED = End Delimiter, Code: 16H

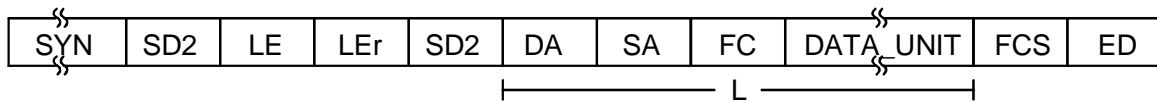
L = Length of Information field, fixed number of Bytes L = 11

Fig. 1.2.3.37: Frame format of fixed length with data field

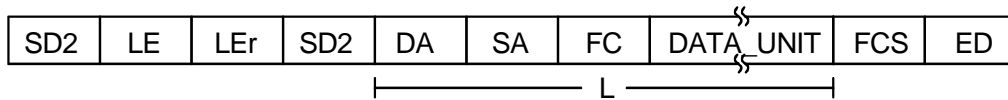
3.3 Formats with variable information field length

For a variable number of data bytes the actual length has to be transmitted in the telegram header. This length indication is located twice (because of transmission security) at the telegram beginning in a fixed telegram header.

A) Format of the request telegram (Send/request frame):



B) Format of the response telegram (Response frame):



SYN = Synchronisation bits, min. 33 bits idle state (Idle)

SD2 = Start byte 2 (Start delimiter), Code: 68H

LE = Length, Value: 4 to 249

LEr = Length byte repeated (repeat)

DA = Destination address

SA = Source address

FC = Frame Control

DATA_UNIT = Data field, variable Length (L-3), max 246 bytes

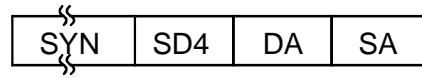
FCS = Frame Check Sequence

ED = End Delimiter, Code: 16H

L = Length of information field, variable number of bytes L = 4 bis 249

Fig. 1.2.3.38: Frame format with variable length

4. Token Telegram



SYN = Synchronisation bits, min. 33 bit idle state (Idle)
SD4 = Start byte 4 (Start Delimiter), Code: DCH
DA = Destination address
SA = Source address

Fig. 1.2.3.39: Token telegram

5. Transmission rules

In the following text possible telegram sequences (message cycles) are displayed. Failure sequences are not included, also Broadcast/Multicast and messages not to be acknowledged.

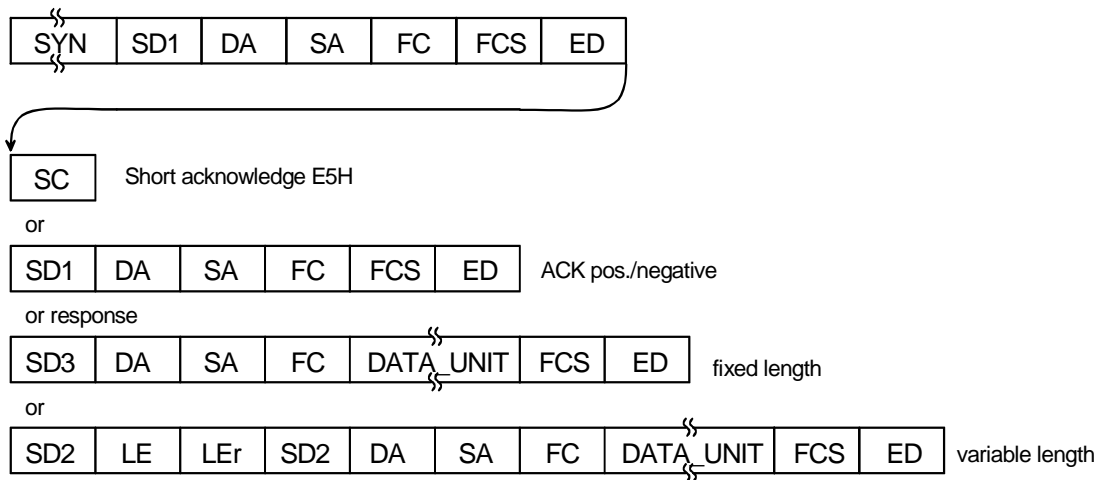


Fig. 1.2.3.40: Request of fixed length without data (Send/Request-Frame)

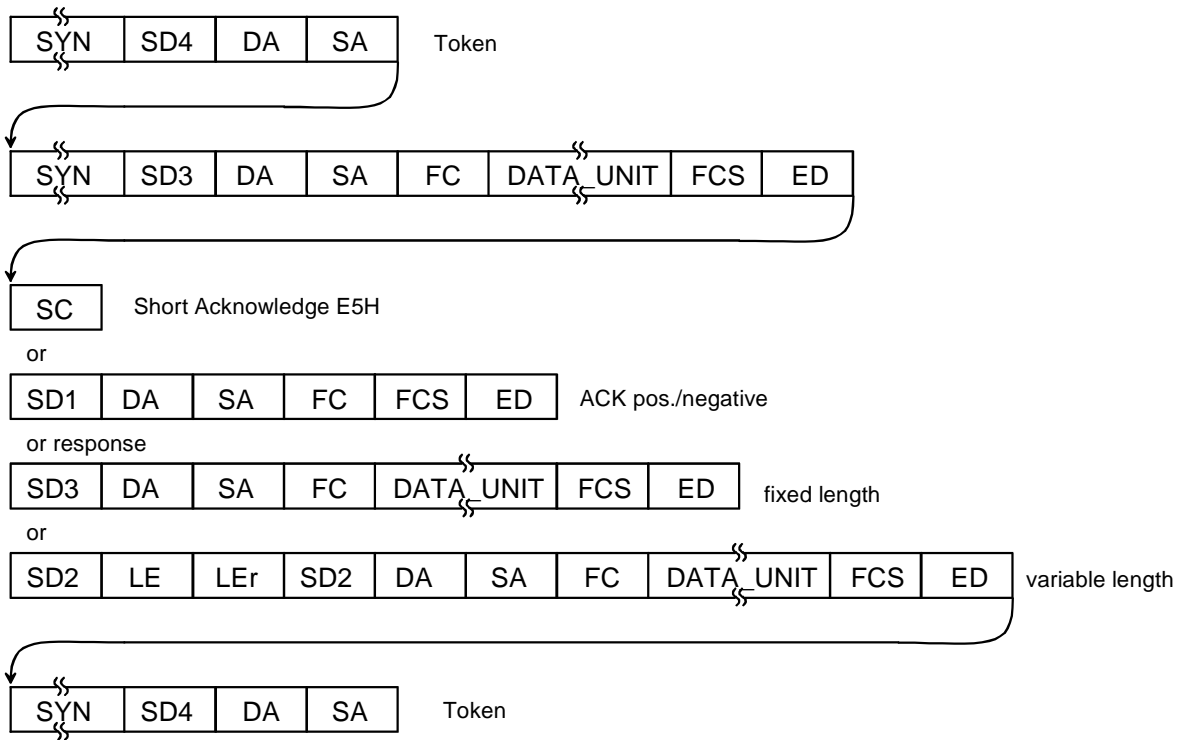


Fig. 1.2.3.41: Token telegram and request of fixed length with data (Send/Request-Frame)

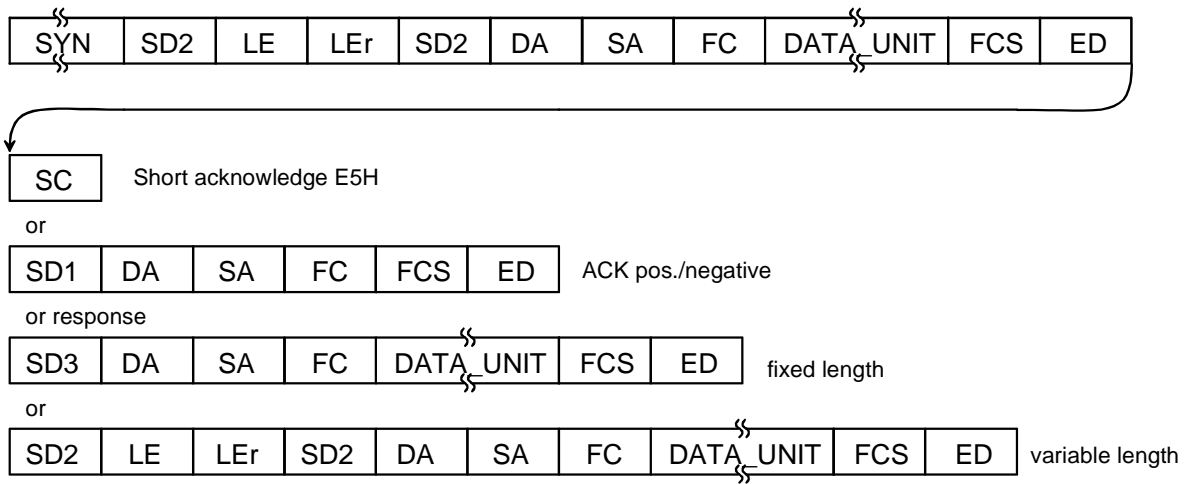


Fig. 1.2.3.42: Request with variable length (Send/Request-Frame)

1.2.3.4.1.3 Interface to Profibus layer 2

1. FDL-User-FDL-interface

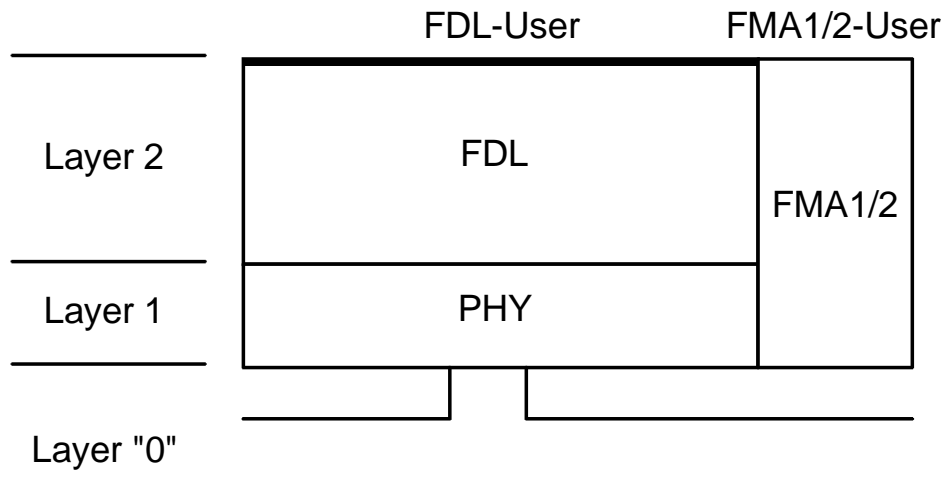


Fig. 1.2.3.43: Interface between FDL-User and FDL in the layer model

At the FDL-User-FDL-interface a (or more) data transmission service is handled via a service access point (Link Service Access Point, LSAP). At the active and passive participants multiple LSAPs at the same time are admissible. For this case in case of a transmission of a message the respective LSAP has also to be transmitted. For the SSAP (Source Service Access Point) a value of 0 to 62, for the DSAP (Destination Service Access Point) a value of 0 to 63 can be chosen. The DSAP value 63 represents the global access address. This DSAP is only admissible in case of the Send-Data services "SDA" and "SDN".

If transmission of LSAPs is abandoned for telegram efficiency reasons, the data transmission services are to be handled using the **Default-LSAP**.

1.1 Overview of the services

The following services are provided to the user of layer 2:

- **Send Data with Acknowledge (SDA)**
- **Send Data with No Acknowledge (SDN)**
- **Send and Request Data with Reply (SRD)**
- **Cyclic Send and Request Data with Reply (CSRSD)**

Send Data with Acknowledge (SDA)

This service allows the user of the FDL (Layer 2) of an **active** participant (called **Local-User** in future), to send user data (Link_Service_data_unit, L_sdu) to a single remote participant. At the remote participant the L_sdu, in case of being received without failure, is transferred to the user (called **Remote-User** in future). The local user receives a confirmation about the receipt or non-receipt of the user data. If a transmission failure occurs, the FDL of the local user repeats the data transmission.

Send Data with no Acknowledge (SDN)

This service allows the local user, to send data (L_sdu) to a single remote participant, to multiple (Multicast) or to all (Broadcast) remote participants at the same time. The local user receives a confirmation about the termination of the transmission, but not about the proper receipt of the data. At the remote participants this L_sdu, in case of being received without failure, is transferred to the Remote-User. However there is no confirmation, that such a transmission has taken place.

Send and Request Data with Reply (SRD)

This service allows a local user, to send data (L_sdu) to a single remote participant and to request data (L_sdu) at the

same time, which have been provided by the remote user previously. At the remote participant the L_sdu received, in case of being without failure, is transferred to the remote user. The service also allows a local user to request data from the remote user, without sending data (L_sdu = Null) to him.

The local user receives either the data requested or an indication, that the data have not been available or a confirmation about not having received the the data sent. With the first two reactions also the receipt of the data sent is confirmed.

If a failure in the transmission occurs, the FDL of the local user repeats the data transmission with request.

Cyclic Send and Request Data with Reply (CSR)

This service allows the local user to send data (L_sdu) cyclically to remote users and to request data from them at the same time. At the remote participant the data received without failure are transferred cyclically to the remote user. The service allows a local user also to request data from the remote user without sending data to him.

The local user receives cyclically either the requested data or an indication, that the data have not been available or a confirmation about not having received the data sent. With the first two reactions also the receipt of data sent is confirmed. In case of failures in the transmission the FDL repeats the data transmission with request.

The desired remote participants and the number and the order of data transmissions with request for the cyclical operation is to be provided by the local user with the **Poll list**.

1.2 Overview of the interactions

The services are provided by using a set of service primitives (marked by FDL_...). To request a service, a request primitive is used by the user. A confirmation primitive comes back to the user after completion of the service, in case of services with cyclical repetition after every send/request cycle. If an

unexpected event occurs at the remote participant, an indication primitive is transferred to the remote user. For the services mentioned above the following primitives are transferred:

Possible for the following participants

Send Data with Acknowledge (SDA)

FDL_DATA_ACK.request	active
FDL_DATA_ACK.indication	active and passive
FDL_DATA_ACK.confirm	active

Send Data with No Acknowledge (SDN)

FDL_DATA.request	active
FDL_DATA.indication	active and passive
FDL_DATA.confirm	active

Send and Request Data with Reply (SRD)

FDL_DATA_REPLY.request	active
FDL_DATA_REPLY.indication	active and passive
FDL_DATA_REPLY.confirm	active
FDL_REPLY_UPDATE.request	active and passive
FDL_REPLY_UPDATE.confirm	active and passive

Cyclic Send and Request Data with Reply (CSRD)

FDL_SEND_UPDATE.request	active
FDL_SEND_UPDATE.confirm	active
FDL_CYC_DATA_REPLY.request	active
FDL_CYC_DATA_REPLY.confirm	active
FDL_CYC_ENTRY.request	active
FDL_CYC_ENTRY.confirm	active

FDL_CYC_DEACT.request	active
FDL_CYC_DEACT.confirm	active
FDL_DATA_REPLY.indication	active and passive
FDL_REPLY_UPDATE.request	active and passive
FDL_REPLY_UPDATE.confirm	active and passive

Timing relations of the primitives for the services:

In the following figures means:

.req= .request .ind = .indication .con = .confirmation

L_sdu = Link_service_data_unit

L_pci = Link_protocol_control_information

L_pdu = Link_protocol_data_unit

$L_pdu = L_pci + L_sdu$

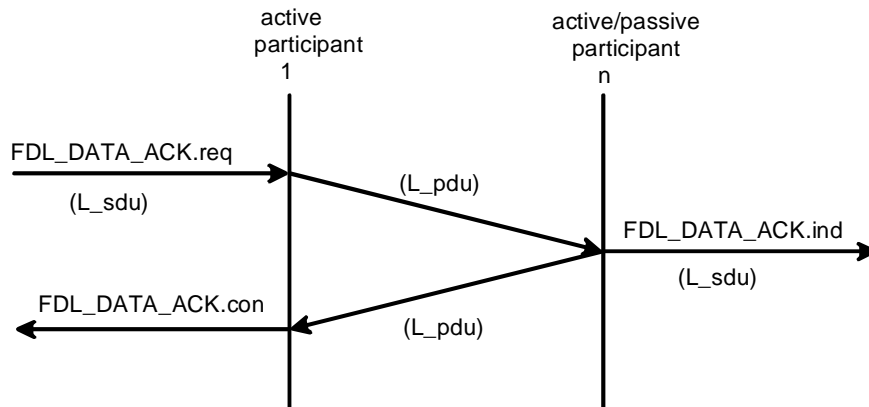


Fig. 1.2.3.44: SDA service

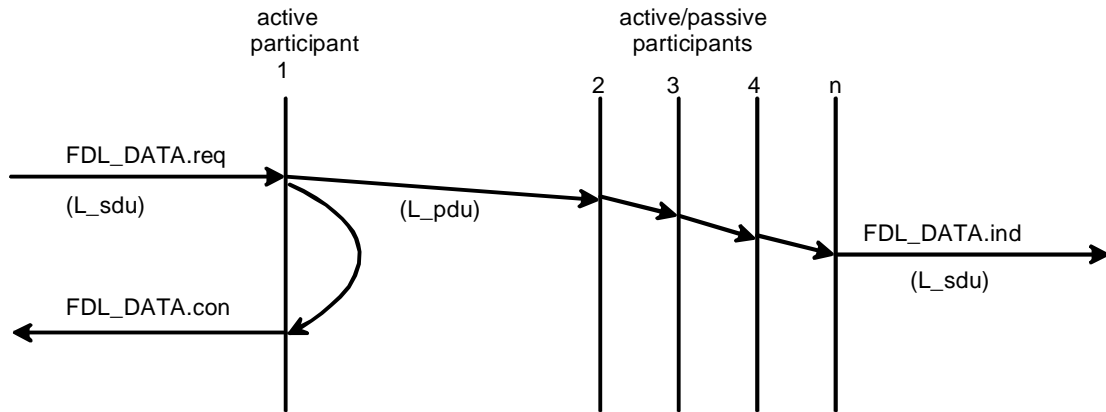


Fig. 1.2.3.45: SDN service

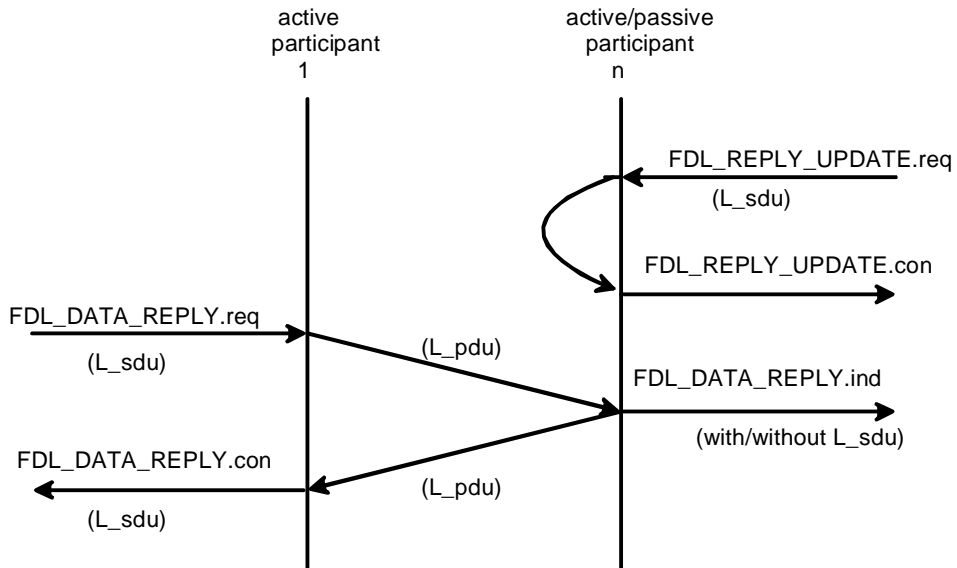


Fig. 1.2.3.46: SRD service

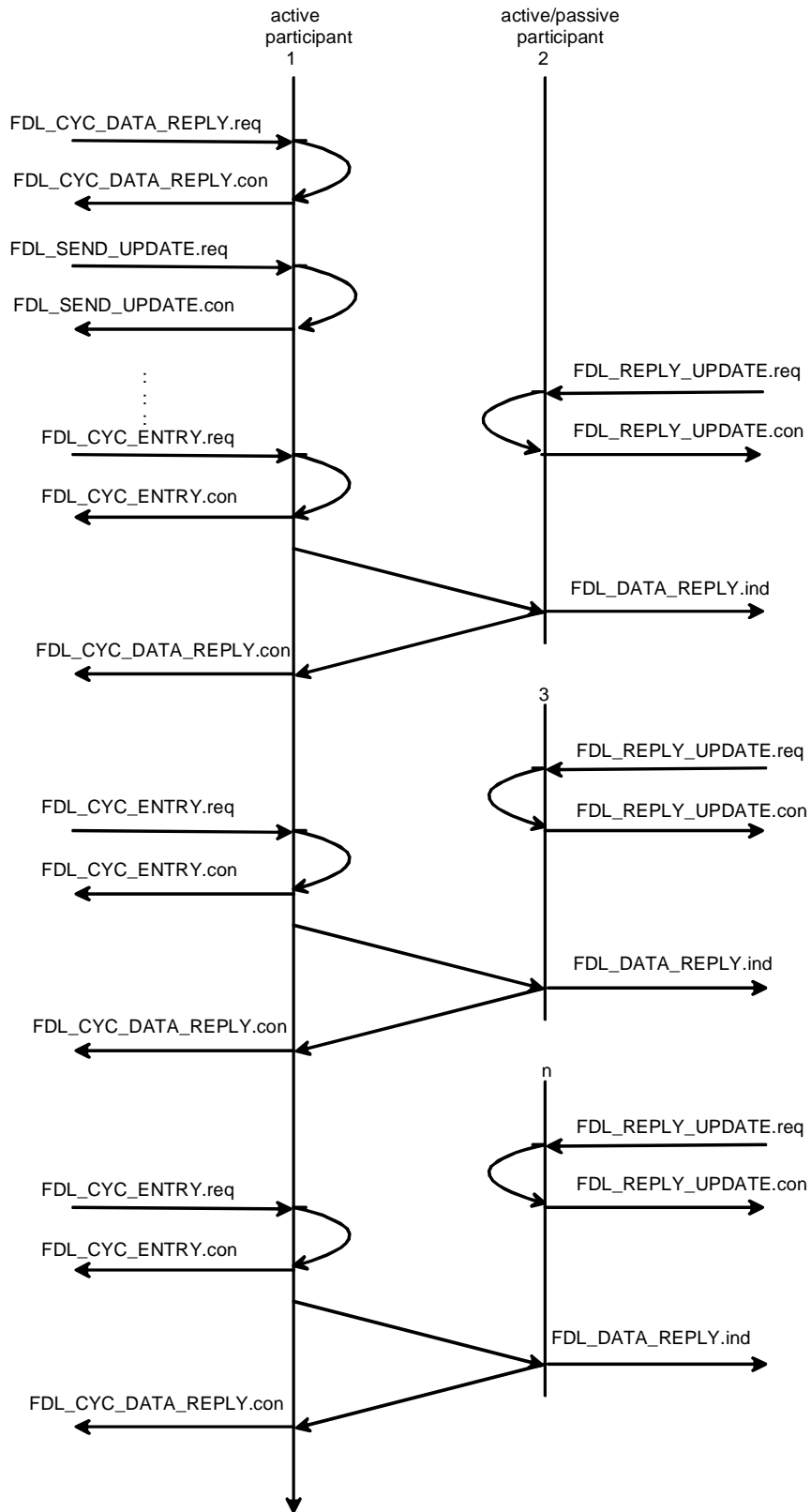


Fig. 1.2.3.47a: CSRD service start

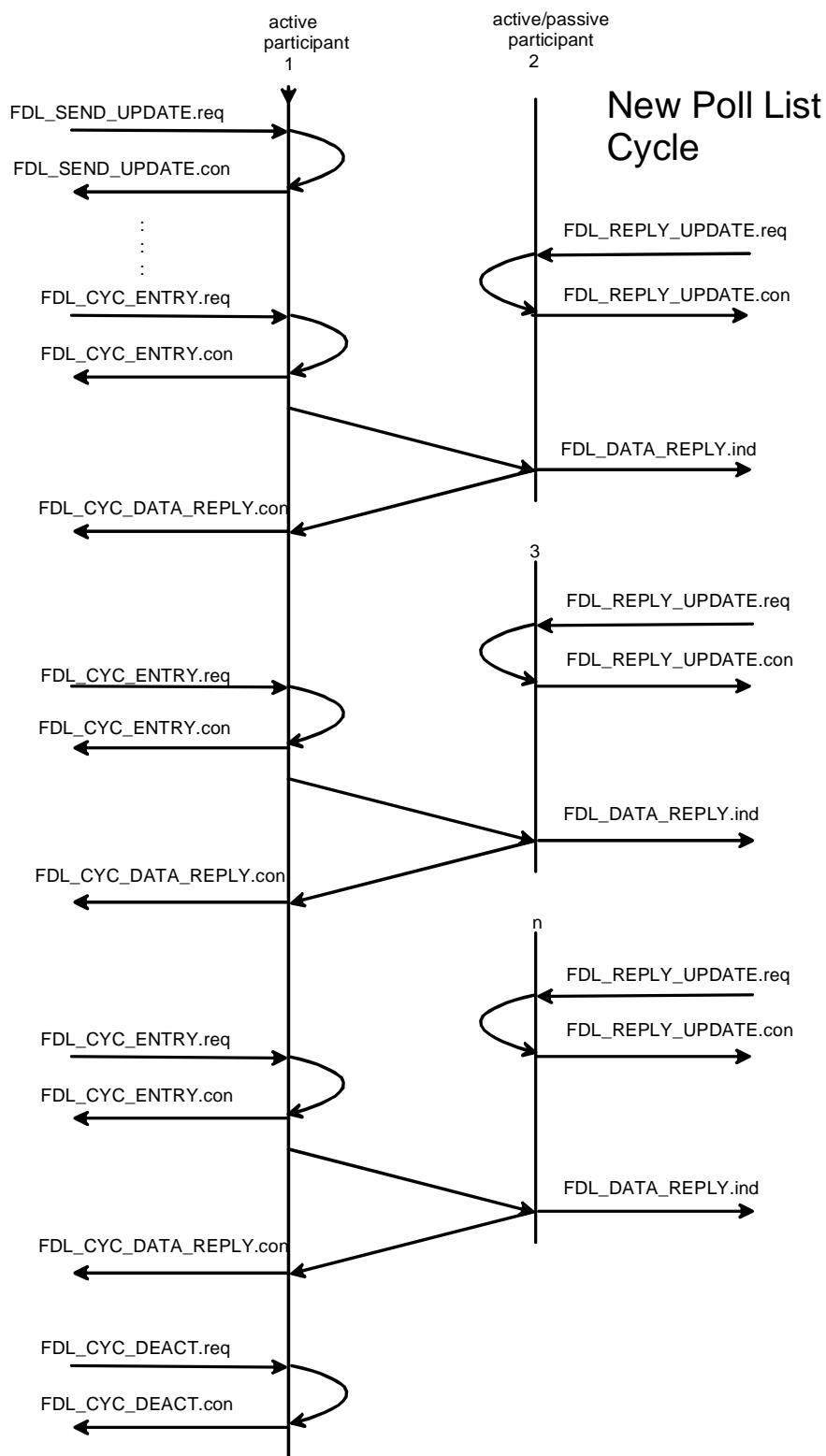


Fig. 1.2.3.47b: CSR service end

1.2.3.4.1.6 Application Layer, FMS

1. General

1.1 Architecture and subsumption into the ISO/OSI layer model

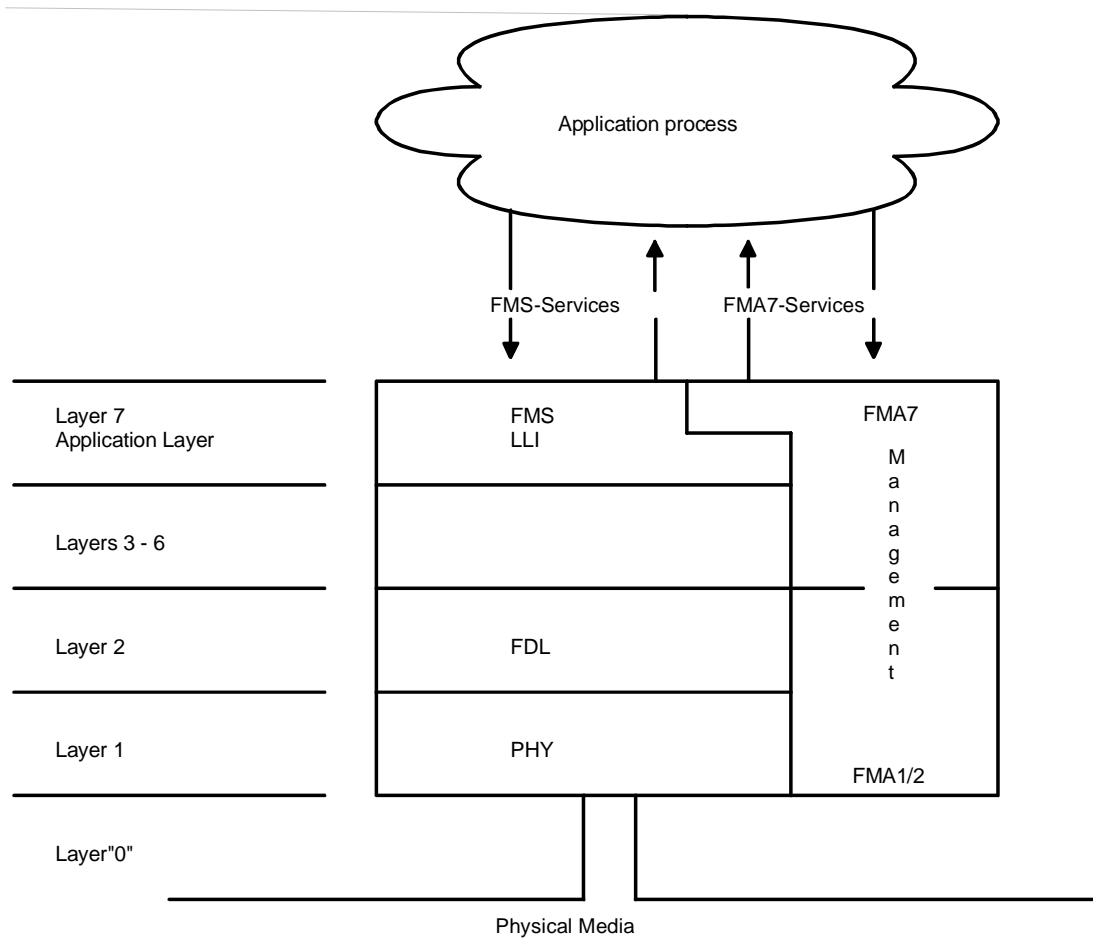


Fig. 1.2.3.57: Subsumption into the ISO/OSI layer model

2. Fieldbus Message Specification (FMS)

The communication system PROFIBUS provides a variety of services for usage of open communication to the user system.

This variety of services is not necessary in all devices and in all application sectors. A restriction takes place by application specific profiles.

Short description of services at the interface between application and FMS:

Initiate (establish a connection)

With this service a connection between two communication partners is established. At this mutual connection specific agreements about the manner of communication usage (Context) are made. All the other communication services are not allowed until a successful establishment of connection has taken place in a connection oriented mode of operation.

Abort (Abort a connection)

An existing connection between two communication partners is aborted. Resuming a communication is in a connection oriented mode of operation not possible until a new connection establishment (Initiate).

Reject (reject an inadmissible service)

With the reject service the FMS protokoll rejects a PDU (protocol data unit). Reasons for the rejection are:

- Violation of the transmission protocol
- Failure in an call of a service
- Failure in the processing of a service
- Service not executable
- Service violates the context agreement

Status (Read device or user status)

Unsolicited-Status (report status unrequestedly)

With this service the device/ user status is sent spontaneously.

Identify (Read manufacturer, Type and Version)

With this service vendor name, model name and revision of a device are read.

Get-OV (Read Object description)

With this service one or more object descriptions are read from the object directory (OV). For this purpose in the get-OV-Service the index or the name of the object description wanted must be specified or the option to read all object descriptions can be chosen.

Initiate-Put-OV (Initiation of reading the OV)

With this service a client announces to a server, that it wants to provide object descriptions.

Put-OV (Load an object description)

With this service the client provides object descriptions to the server for the object directory of the server.

Terminate-Put-OV (Termination of loading the OV)

With this service the client announced to the server, that it terminates the transfer of object descriptions.

Initiate-Download-Sequence

With this service a transfer of data (Parameters, Program code....) on a block by block basis from the "Client" into a Domain of the "Server" is announced.

Download-Segment (Transmit download data block)

With this service the "Server" fetches a block of data at the "Client". The "Client" sends together with the data a message indicating whether or not more data are ready for transmission.

Terminate-Download-Sequence

With this service the "Server" informs, that the data transfer on a block by block basis in its domain is terminated. Additionally the "Server" reports if the transmission was successfully terminated.

Request-Domain-Download

With this service the "Server" requests a download from its "Client". In the request the "Server" indicates optionally, from which file of the "Client" the data are to be read.

Note: The response (Acknowledgement) to this request takes place not before the download is handled completely.

Initiate-Upload-Sequence

With this service a transfer of data (Parameters, Program code...) on a block by block basis from a Domain of the server is announced by the client.

Upload-Segment (Transmit upload data block)

With this service the client fetches an block of data from a domain of the server. The server sends together with the data a message indicating whether or not more data are ready for transmission.

Terminate-Upload-Sequence

With this service the client informs, that the data transmission block by block from the server to the client is terminated.

Request-Domain-Upload

With this service the "Server" requests an upload from its "Client". In the request the "Server" indicates optionally into which file of the "Client" the Upload data are to be written.

Note: The response (Acknowledgement) to this request takes place not before the complete handling of the upload.

Create-Program-Invocation (Compose a program)

With this service domains (e.g. code, data domains), described in the object directory are online composed to a program invocation object. This object is afterwards addressed with a logical address.

Delete-Program-Invocation

With this service a program invocation object is deleted.

Start (Start a program - after reset)

A program is started and is executed from its very beginning.

Stop (Stop program)

A running program is stopped, but not reset to the beginning.

Resume (Resume Program - after stop)

A stopped program is brought to the state RUNNING, but not reset.

Reset (Reset program)

A stopped program is set to its beginning.

Kill (Shut down a program)

A program invocation is independently of its momentary state brought to the state UNRUNNABLE.

Read (Read a variable)

The value of a variable object described in the object directory is read.

Write (Write a variable)

A value is assigned to a variable object described in the object directory.

Read-With-Type (Read a variable)

The value and the data type description of a variable object described in the object directory is read.

Write-With-Type (Write a variable)

A value is assigned to a variable object described in the object directory. The data type description is added to the value.

Phys-Read (read of a physical access object)

With this service the value of a physical access object is read.

Phys-Write (Write a physical access object)

With this service a value is assigned to a physical access object.

Information-Report (Send data - without acknowledgement)

The value of a variable object described in the object directory is transmitted to all the other participants e.g. for synchronisation.

Information-Report-With-Type (Send data - without acknowledgement)

The value and the data type description of a variable object described in the object directory is transmitted to all the other participants e.g. for synchronisation.

Define-Variable-List (Compose a variable list)

With this service variable objects described in the static object directory are composed online to a new variable object called variable list. This object is afterwards addressed with a logical address.

Delete-Variable-List

With this service a object variable list is deleted.

Event-Notification

With this service a predefined event message is transferred to another communication partner.

Event-Notification-With Type

With this service a predefined event message is transferred to another communication partner. The data type description is added to the event data.

Acknowledge-Event-Notification

With this service the receipt of a event is acknowledged.

Alter-Event-Condition-Monitoring (enable/ disable event)

With this service the transmission of an event is enabled or disabled.

1.2.3.4.1.7 PROFIBUS-DP

PROFIBUS-DP is designed for fast data exchange on the sensor/actor level. Here central control devices (e.g. PLCs) communicate via a fast serial link with peripheral input and output devices. The data exchange with these peripheral devices predominantly takes place cyclically. The central controller (Master) reads the input informations from the slaves and writes the output informations to the slaves. Here the bus cycle time has to be shorter than the program cycle time of the central controller, which amounts to approx. 10 ms in many applications.

1.2.3.4.1.7.1 Basic properties

Protocol architecture:

In PROFIBUS-DP the layers 3-6 are not filled. Also the application layer (7) is omitted to obtain the necessary speed. The Direct Data Link Mapper (DDLMM) provides a comfortable access to layer 2 to the user interface. The application functions usable by the user as well as the system behaviour and the device behaviour of the different PROFIBUS-DP device types are defined in the user interface.

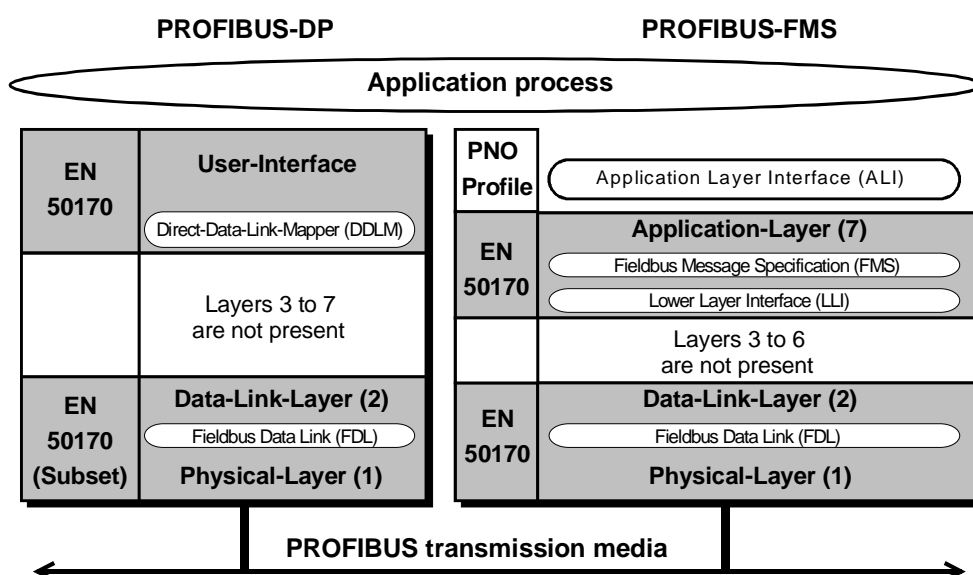


Fig. 1.2.3.67: Protocol architecture of PROFIBUS-FMS and -DP

Speed:

For transmission of 512 bits of input and 512 bits of output data distributed to 32 participants PROFIBUS-DP needs at a transmission rate of 1,5 Mbit/s approx. 6 ms and at 12 Mbit/s less than 2 ms. The demand for a short system reaction time is fulfilled therewith. Fig. 1.2.3.68 shows the transmission time of PROFIBUS-DP depending on the number of participants and on the transmission speed.

The substantial increase of speed compared to PROFIBUS-FMS is especially to be traced back to the fact, that the transmission of the input and output data is realized in one message cycle by using the SRD service (Send and Request Data Service) of layer 2. Additionally minimum requirements for the protocol implementation have been defined.

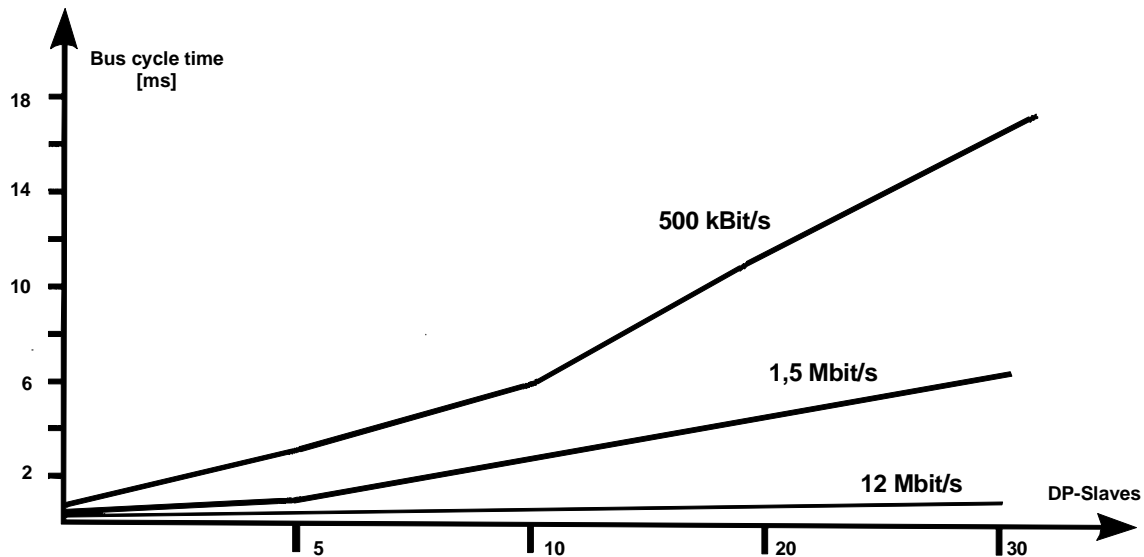
Diagnosis functions:

The extensive diagnosis functions of PROFIBUS-DP provide a fast localization of failures. The diagnosis messages are transferred via the bus and summed up by the master. They are divided into three levels:

- **Station related diagnosis**
Messages about the general readiness to operate of a participant like e.g. overtemperature or supply voltage being low.
- **Module related diagnosis**
These messages indicate, that in a certain I/O partial area (e.g. 8 Bit output module) of a participant a diagnosis is pending.
- **Channel related diagnosis**
Here the cause of a failure related to a single input/output bit (channel) is indicated, e.g. short circuit on output No. 7.

Handling and installation:

The RS-485 transmission technique is very easy to handle. For the installation of the twisted pair cable no expert knowlegde is necessary. The bus structure allows the connection and disconnection of stations without disturbing the other participants and the stepwise commisioning of the system. Expansions at a later date have no influence on the stations already being in operation.



Boundary condition: Each slave has 2 bytes of input data and 2 bytes of output data.
The minimum Slave-Intervall-Time amounts to 200 microseconds
 $T_{sdi} = 37$ Bit times, $T_{sdr} = 11$ Bit times

Fig. 1.2.3.68: Bus cycle time of a PROFIBUS-DP Mono-Master system

1.2.3.4.1.7.2 System configuration and device types

With PROFIBUS-DP **Mono or Multi-Master** systems as well can be implemented. So a high degree of flexibility in the system configuration is provided. Max. 126 devices (Master oder Slaves) can be connected to one bus. The definitions for the system configuration include the number of stations, the assignment of the station addresses to the I/O addresses, data consistency of the I/O data, format of the diagnosis messages and the bus parameters used.

Each PROFIBUS-DP system consists of different device types. According to the respective task the device types are

distinguished:

DP-Master Class 1 (DPM1)

Here we are talking about a central controller, which exchanges informations in a fixed message cycle with peripheral stations (DP-Slaves). Typical devices are e.g. Programmable Logic Controllers (PLC), Numerical Controls (CNC) or Robot Controls (RC).

DP-Master Class 2

Devices of this type are programming, configuration, and diagnosis devices. They are utilized during commissioning to create the configuration of the DP system.

DP slave

A DP-Slave is a peripheral device (Sensor/Actor), which reads input informations and writes output informations to the periphery. Also devices are possible, which provide only input or only output informations. Typical DP slaves are devices with binary inputs/outputs for 24 V oder 230 V, analog inputs, analog outputs, counters etc..

The amount of input and output informations is depending on the device and is allowed to be maximum 246 bytes of input and 246 bytes of output data. For expense and implementation reasons many of the devices available today work with a max. effective data length of 32 bytes.

In **Mono-Master-Systems** in the operation phase of the bus system only one master is active at the bus. In Fig. 1.2.3.69 the system configuration of a mono master system is displayed.

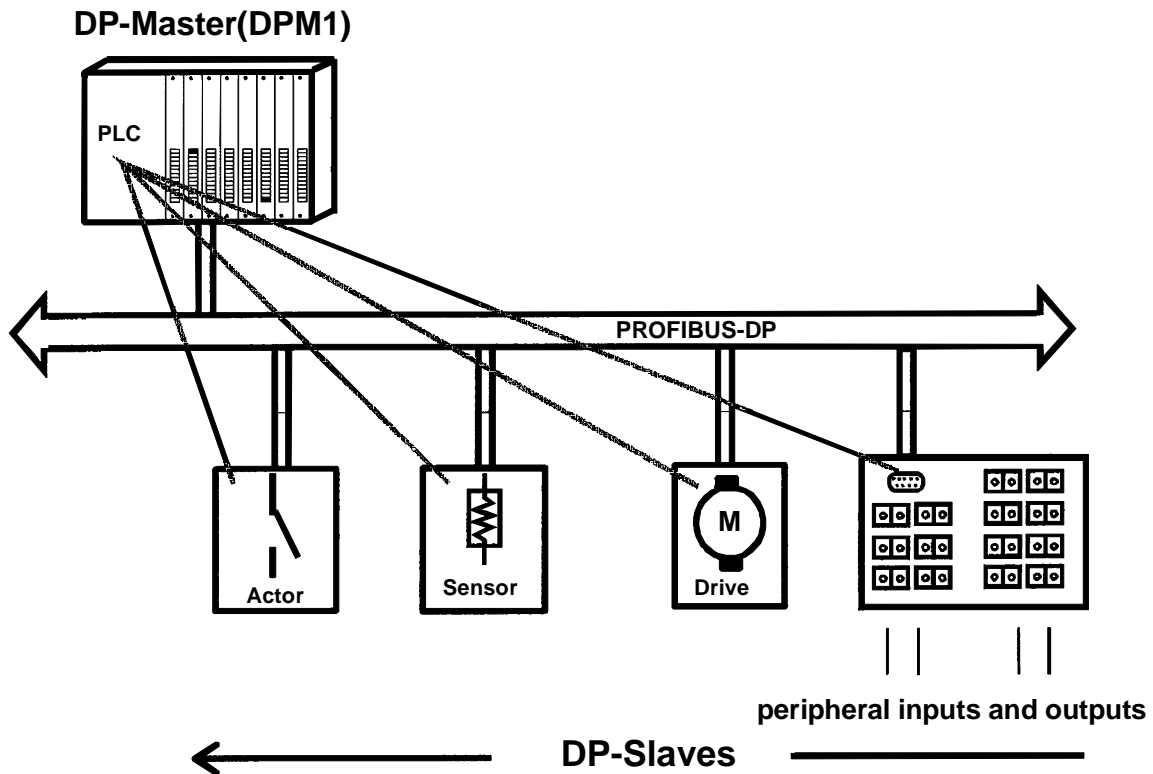


Fig. 1.2.3.69: PROFIBUS-DP Mono-Master system

The PLC is the central control component. The DP-Slaves are coupled to the PLC by means of the transmission medium. With this configuration the shortest bus cycle time is achieved.

In **Multi-Master-Operation** multiple masters are situated at one bus. They either establish subsystems independent of each other, each consisting of a DPM1 and the respective DP-Slaves or additional configuration and diagnosis devices (see fig. 1.2.3.70). The input and output images of the DP-slaves can be read by all DP-Masters. Writing of outputs is only possible for one DP-Master (The DPM1 assigned in the configuration). Multi-Master systems achieve a medium bus cycle time.

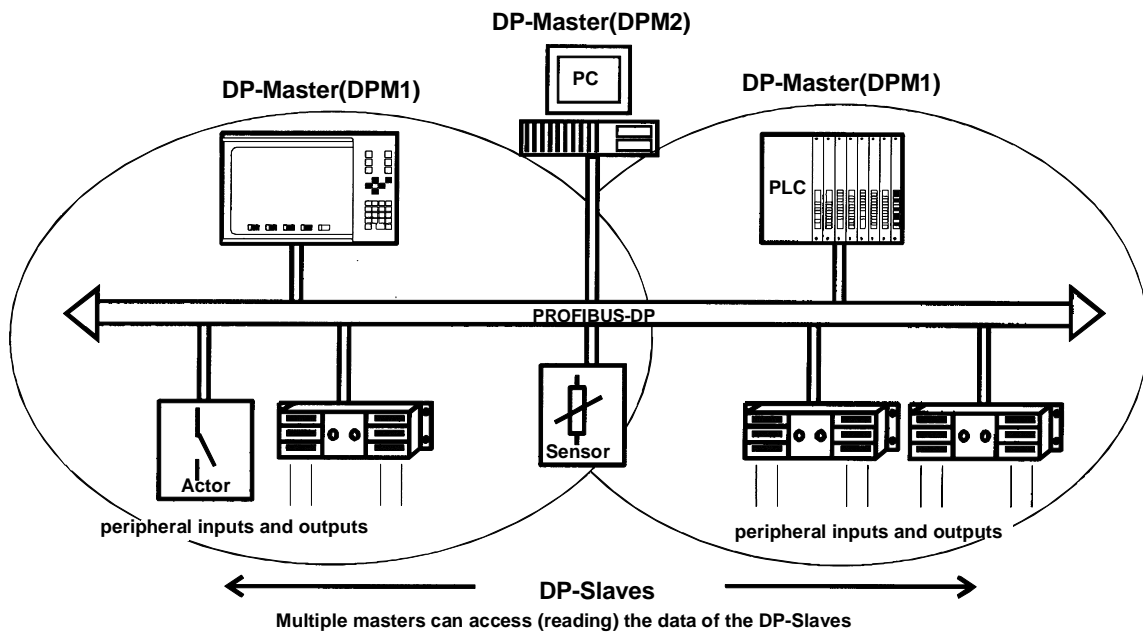


Fig. 1.2.3.70: PROFIBUS-DP Multi-Master System

Literature

Dunning, Gary: Introduction to programmable logic controllers,
Delmar / Thomson learning, 2002, ISBN 0-7668-
1768-7



Automation and industrial communication

Chapter 1: PLC Programming

Content

1 PLC Programming

1.1	Introduction to the basics of industrial automation	- 1 -
1.1.1	Terms	- 1 -
1.1.1.1	Differentiation regarding the program implementation	- 2 -
1.1.1.2	Differentiation regarding the hierarchical allocation	- 4 -
1.2	Software of PLC devices	- 10 -
1.2.1	Organisation of the operational software in PLC devices	- 10 -
1.2.1.1	Cyclical processing	- 10 -
1.2.1.1.1	The image table	- 15 -
1.2.1.2	Alarm processing	- 16 -
1.2.1.3	Time controlled processing	- 16 -
1.2.2	Language elements of the application software	- 20 -
1.2.2.1	Binary operations	- 20 -
1.2.2.1.1	Operand types	- 20 -
1.2.2.1.2	Binary (Boolean) operations	- 21 -
1.2.2.1.3	R/S-flip flops	- 22 -
1.2.2.1.4	Binary Timers	- 23 -
1.2.2.2	Algebraic operations	- 23 -
1.2.2.2.1	Data types	- 23 -
1.2.2.2.2	Operand types	- 25 -
1.2.2.2.3	Computing operations	- 25 -
1.2.2.2.4	Load and assignment operations	- 26 -
1.2.2.2.5	Organizational functions	- 27 -
1.2.2.2.6	Shift functions	- 27 -
1.2.2.2.7	Type conversions	- 27 -
1.2.3	Kinds of representation of the application software	- 30 -
1.2.3.1	Statement list (STL)	- 30 -
1.2.3.1.1	Control statements	- 30 -
1.2.3.1.2	Operations, Modifiers and Operands	- 31 -

1.2.3.2	Ladder diagram (LAD)	- 34 -
1.2.3.3	Function block diagram (FBD)	- 40 -
1.2.4	Sequential function chart (SFC)	- 50 -
1.2.4.1	Steps	- 50 -
1.2.4.2	Transitions	- 51 -
1.2.4.3	Actions	- 55 -
1.2.4.3.1	Action block	- 59 -
1.2.4.4	Sequence rules	- 60 -

1 PLC Programming

1.1 Introduction to the basics of industrial automation

1.1.1 Terms

Logic control

A logic control assigns certain logic states of the output signals to certain states of the input signals with the help of boolean operations. Also control systems with logic operations and single memory and timing functions are called the same.

Sequence control

A sequence control is a control with an inevitably stepwise sequence, in which the transition from one step to the following step(s) according to the program takes place depending of transition conditions.

Program of a control

The program of a control is the entirety of all control statements and declarations for signal processing, by which a plant to be controlled (process) is affected according to the task.

Remark: This definition is valid independently of the signal processing being connection oriented or memory programmed.

**1.1.1.1 Differentiation regarding the program
 implementation**

Connection oriented control

A connection oriented control is a control, the program of which is defined by the type of the functional units and by their connections.

Control with fixed program

A control with a fixed program is a connection oriented control, where program modifications are not planned.

Reprogrammable control

A reprogrammable control is a connection oriented control, where program modifications are provided for and are easy to carry out.

Programmable logic control

A programmable logic control is a control, the program of which is stored in a program memory.

Freely programmable control

A freely programmable control is a programmable logic control with read/write memory (RAM) as program memory, whose entire content can be modified even in arbitrarily small extent without mechanical intervention into the control device.

Control with replaceable program

A control with replaceable program is a programmable logic control with read-only-memory (ROM) as program memory, whose content can be modified after programming being once completed only by a mechanical intervention into the control device.

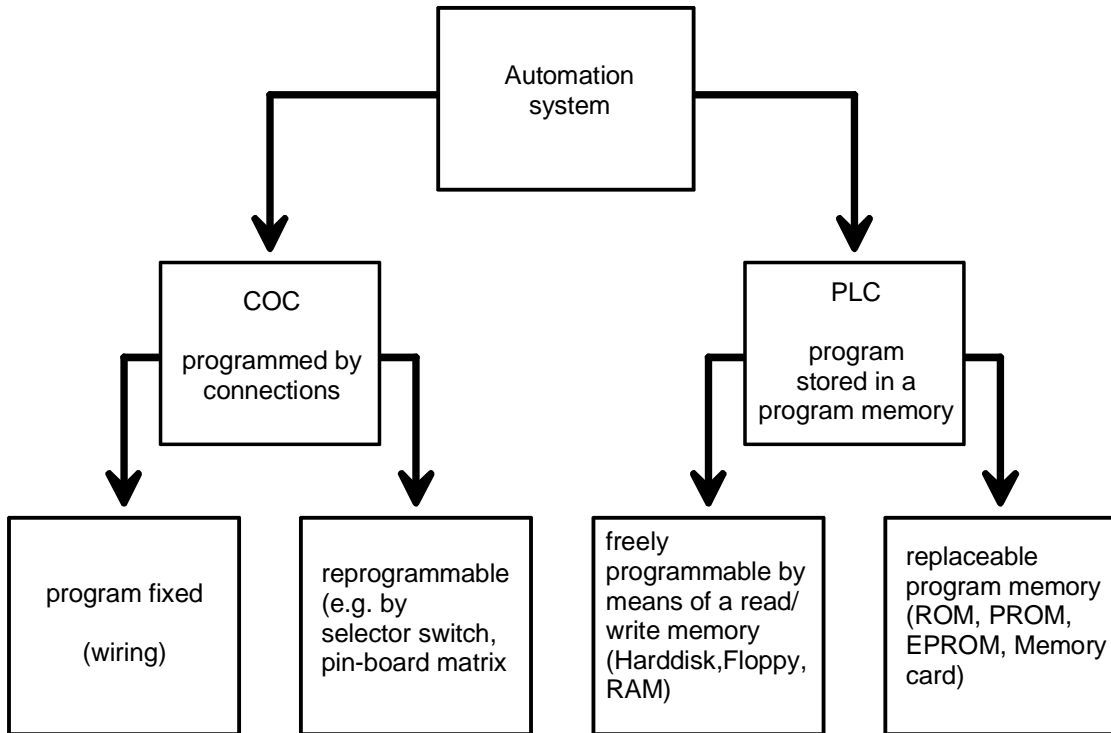


Figure 1.1.1.1: Different possibilities of program implementation

1.1.1.2 Differentiation regarding the hierarchical allocation

Individual control, drive control

A individual control or drive control is a functional unit for the control of a single final control element or a drive.

Individual control level, drive control level

The individual or drive control level summarizes all parts of a control system that act directly on the process by means of final control elements or drives.

Group control

A group control is a functional unit for the control of a coherent process part, which is superordinated to the belonging individual or drive controls.

Group control level

The group control level summarizes all parts of the control unit that act on certain partitions of the individual control level.

Remark: The group control level can be split into several hierarchically arranged levels.

Process control

The process control is the functional unit superordinated to the group controls for the control of the entire process.

Process control level

The process control level summarizes all parts of the control unit that act on the group control level.

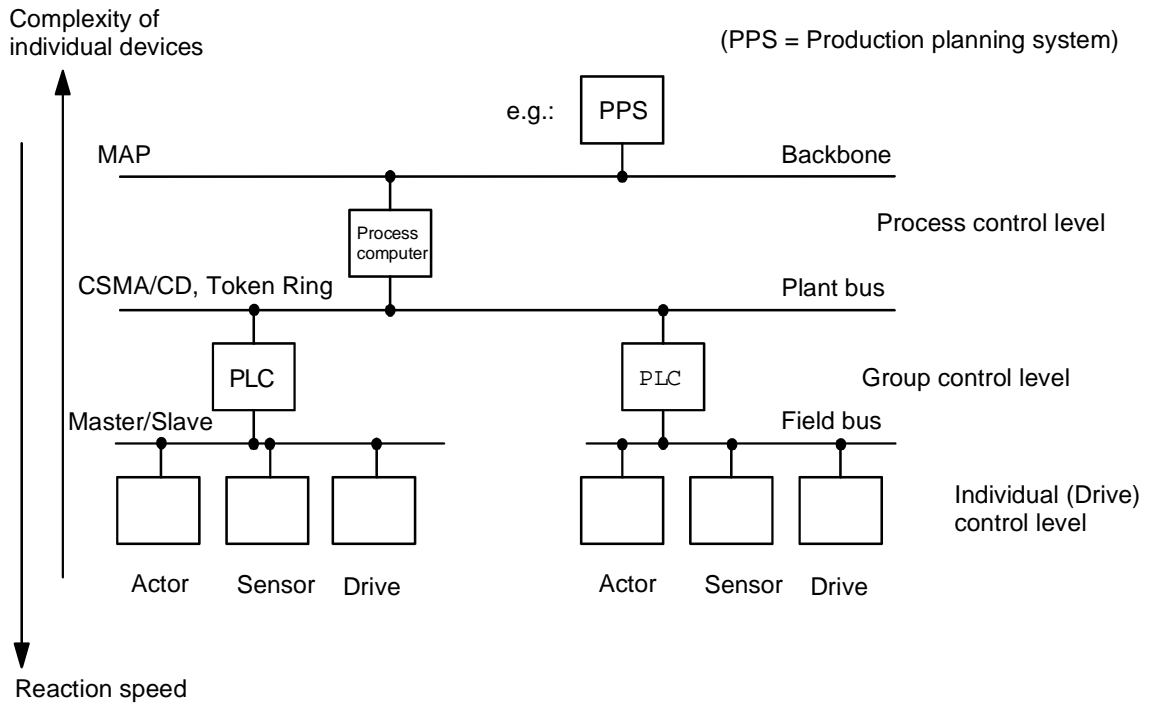


Figure 1.1.1.2: Hierarchical arrangement of an automation system

Pages 6-9 intentionally left blank

1.2 Software of PLC devices

1.2.1 Organisation of the operational software in PLC devices

For structuring of the software in PLC devices so called blocks or program organization entities (POEs) are employed. The block types are separated into

- Organizational blocks (OB)
- Function blocks (FB) and functions (FC)
- Data blocks (DB)
- System functions (SFC) and system function blocks (SFB)

A block is a part of the application program that is delimited by function, structure or purpose. There is a differentiation between code blocks, containing statements for data processing (OBs, FCs, FBs) and blocks containing data (DBs).

Certain organizational blocks establish the interface between application software and system program. They are called by the system program and contain executable code deposited by the user. By deposition of the code in very certain organizational blocks the user decides whether the code is executed cyclically, time controlled or alarm controlled.

1.2.1.1 Cyclical processing

The cyclical processing is the normal processing mode in programmable logic controllers (Figure 1.2.1.1). After a singular processing of a start up block the processor starts automatically the cyclical processing at the top of the application program. It processes the statements sequentially up to the end of the application program, passes through some system routines and then starts again the processing at the top of the program.

Certain blocks (e.g. SIEMENS: OB1) are the interface between the system program and the cyclical processing of the application program. The first statement in such a block is at the same time the first statement of the application program, so being the same as the program start.

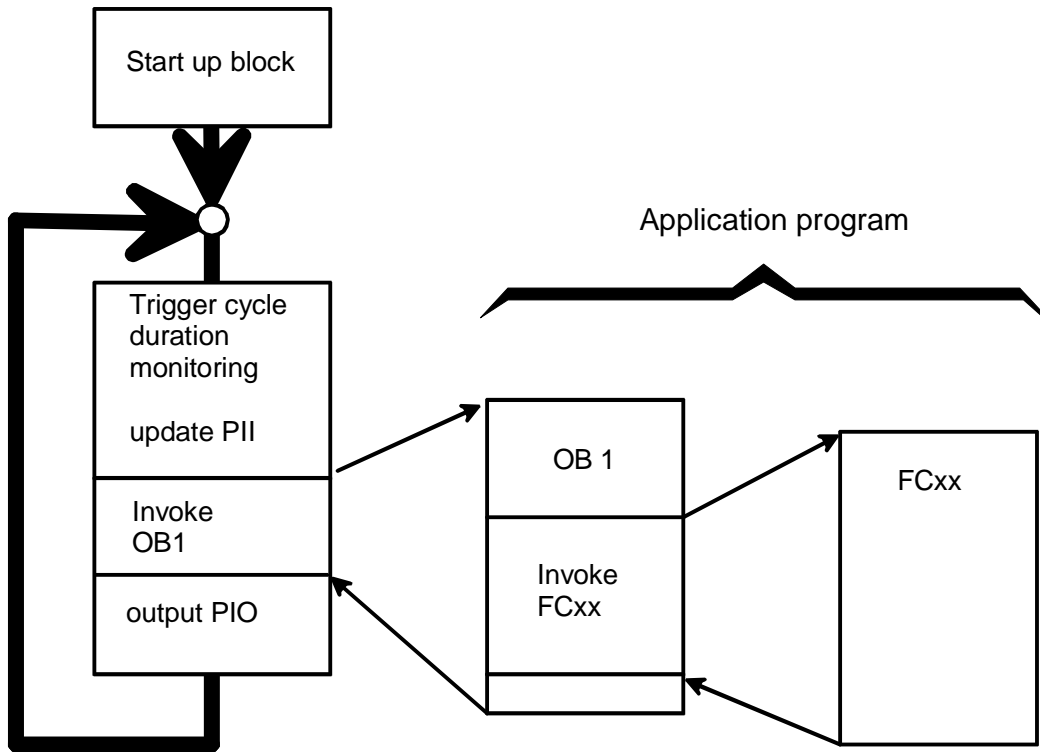


Figure 1.2.1.1: Cyclical processing

In these key blocks the functions and function blocks of the cyclical program are invoked. In these blocks being invoked again program invocations are possible, up to a certain nesting depth, which is depending on the device or its manufacturer.

The processing time of the application program results from the sum of all blocks being invoked. If a block is invoked n times its processing time has to be considered n times in the summation.

● **Raw structuring of the application program**

In the block being invoked first the raw structuring of the application program is located. The documentation of this block shall show the fundamental program structures at the first glance (Figure 1.1.2.2) or highlight interrelated plant parts (Figure 1.2.1.3).

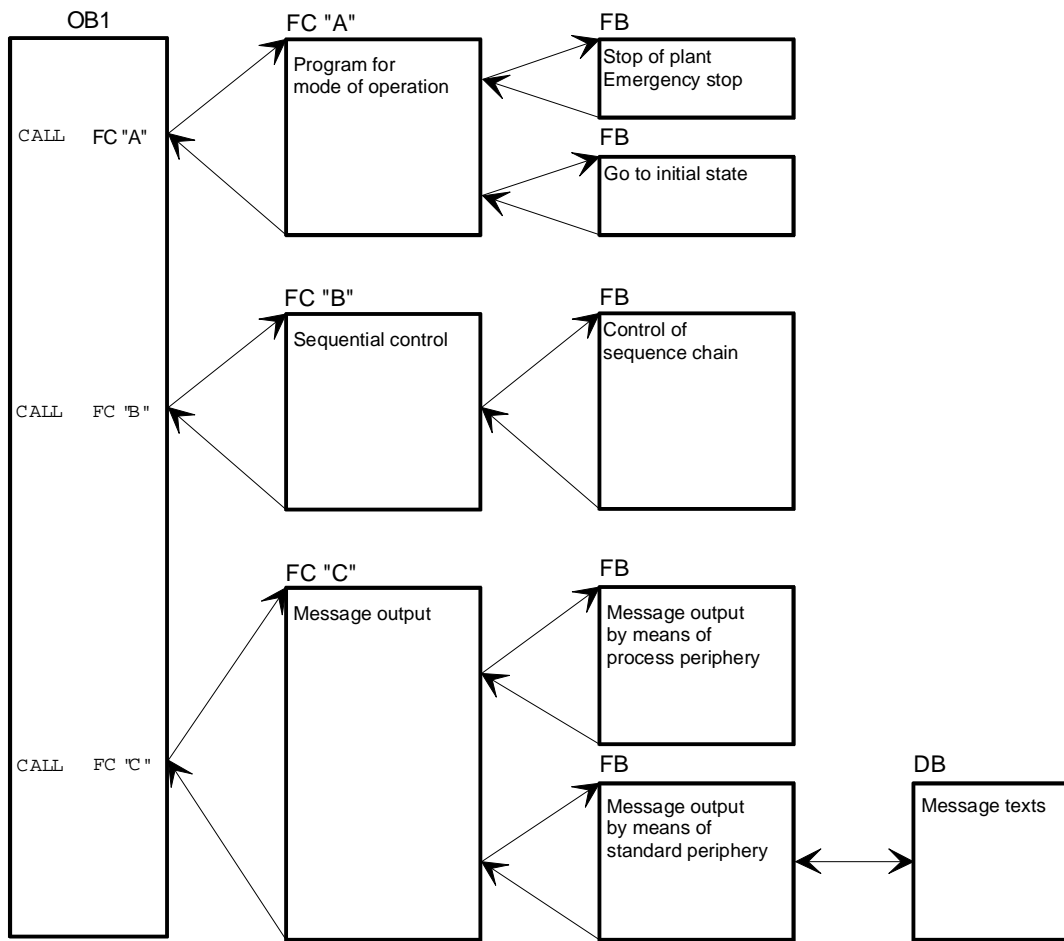


Figure 1.2.1.2: Raw structuring of the application program following the program structure

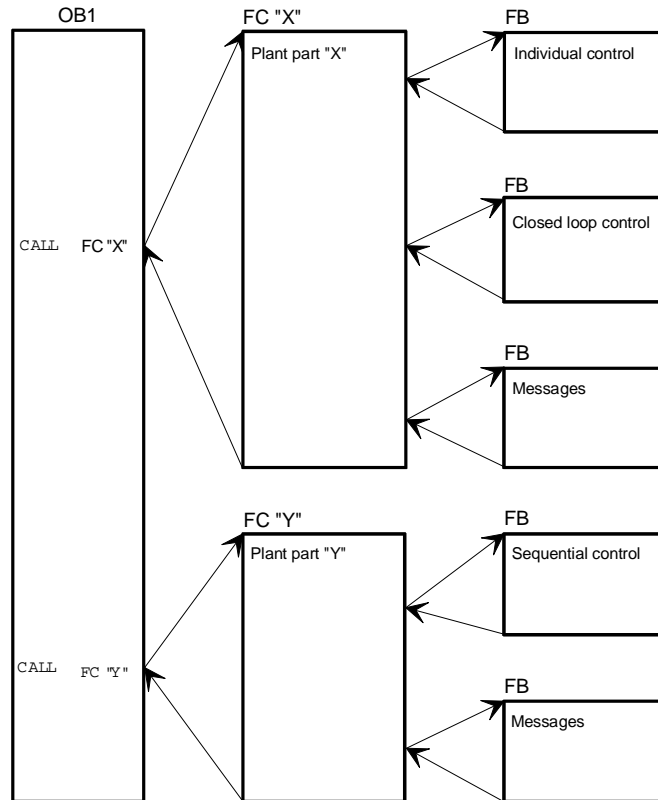


Figure 1.2.1.3: Raw structuring of the application program following the plant structure

● **Explanation to the block types:**

1) Organizational blocks

The organizational blocks are the interface between the system program and the application program. They are called by the operating system and control the cyclical and alarm controlled program processing, the start up behaviour of the automation system and the error treatment. They can be separated into two groups:

- The first group is being called by the system program. The members of this group control the program processing, the start up behaviour of the processor and the behaviour in case of an error. These OBs are to be programmed by the user.

- The second group are the integrated special functions. They are inherent parts of the system program and are called in the application program if required.

2) Functions

A function is a code block without memory. FCs are parameterizable and provide a return value (output parameter) to the invoking block.

3) Function blocks

A function block is a code block with memory. FBs serve for programming of frequently recurring or also complex functions (e.g. digital functions, sequence control, closed loop control, message functions). A function block can be invoked by superimposed blocks several times and can be provided with new operands (parameterized) at each call.

4) Data blocks

DBs contain (fixed or alterable) data the application program works with. This type of blocks does not contain executable code and differs basically from the other blocks in its function.

The data of a DB can be:

- arbitrary bit patterns, e.g. for plant states
- Numbers (hexadecimal, dual, decimal) for time values, computing results
- alphanumerical characters, e.g. for message text

5) System functions (SFCs) and system function blocks (SFBs)

Not every function is to be self-programmed. In many cases it is possible to revert to prefabricated blocks being available in the operating system of a CPU card, e.g. for programming of communication functions. In detail these are the following blocks:

- a) System functions (SFC), with properties like functions (FC)
- b) System function blocks (SFB), with properties like function blocks (FB)

● **Interrupt of the cyclical processing**

The cyclical program processing can be interrupted by

- alarm processing
- time controlled processing
- Programming failures and device failures
- Signals from the hardware
- Change of mode of operation (caused by the switch at the device or by the programming tool)

1.2.1.1.1 The image table

● **Process image table of the inputs (PII)**

At the beginning of each cycle the states of all binary inputs are scanned and stored in the process image of the inputs. During program processing the central processing unit accesses the signal states in the process image table but not the signal states of the inputs. During one cycle of the application program the states of the inputs therewith are consistent.

● **Process image table of the outputs (PIA)**

If an output is assigned a value during the processing of the application program or the system program the central processing unit stores this information in the process image table of the outputs. At the end of a cycle the central processing unit copies the process image table of the outputs to the outputs. Afterwards a new cycle starts (see figure 1.2.1.1).

1.2.1.2 Alarm processing

In this mode of operation the cyclical operation is interrupted by a signal on a interrupt line. A certain block is then invoked by the system program (Interrupt service routine), by means of which the user can initiate a wanted reaction. After the processing of this program the processor returns to the interrupt location and resumes the cyclical processing.

The process alarm processing so provides to the user the **immediate reaction** to process signals.

1.2.1.3 Time controlled processing

A time controlled processing is present if a signal coming from an "internal clock" causes the processor to interrupt the normal cyclical processing and to call a certain block, by means of which the user can let a program be run controlled by time.

After processing of this program the processor returns to the interrupt location in the cyclical program and resumes its processing there.

Time controlled processing is in the majority of cases necessary for closed loop control tasks.

1.2.2 Language elements of the application software

1.2.2.1 Binary operations

1.2.2.1.1 Operand types

The operand types listed below are applied:

Type:	Operand indicator:
- Inputs	E, %I or %IX
- Outputs	A, %Q or %QX
- Memory bits	M, %M or %MX
- Local data	L
- Timers	T
- Counter	C
- Data bits	DBX, DIX
- System memory bits	SM

1.2.2.1.2 Binary (Boolean) operations

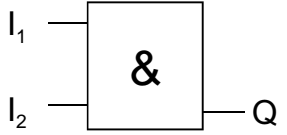
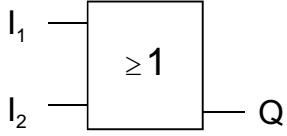
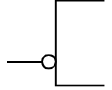
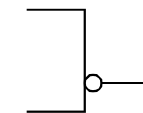
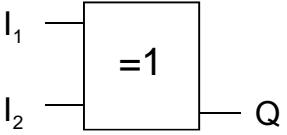
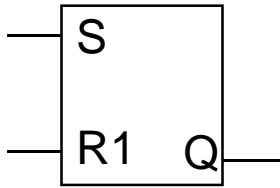
Name	Math. Symbol	Operation	Switching table	Graphical symbol in FBD language															
AND Conjunction AND function	\wedge	$I_1 \wedge I_2 = Q$	<table border="1"> <thead> <tr> <th>I_1</th> <th>I_2</th> <th>Q</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	I_1	I_2	Q	0	0	0	0	1	0	1	0	0	1	1	1	
I_1	I_2	Q																	
0	0	0																	
0	1	0																	
1	0	0																	
1	1	1																	
OR Disjunction OR function	\vee	$I_1 \vee I_2 = Q$	<table border="1"> <thead> <tr> <th>I_1</th> <th>I_2</th> <th>Q</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	I_1	I_2	Q	0	0	0	0	1	1	1	0	1	1	1	1	<p style="text-align: right;">*)</p> 
I_1	I_2	Q																	
0	0	0																	
0	1	1																	
1	0	1																	
1	1	1																	
NOT Negation	\neg	$\neg I = \bar{I}$	<table border="1"> <thead> <tr> <th>I</th> <th>\bar{I}</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> </tr> </tbody> </table>	I	\bar{I}	0	1	1	0	 <p>at an input</p>									
I	\bar{I}																		
0	1																		
1	0																		
		$\neg Q = \bar{Q}$	<table border="1"> <thead> <tr> <th>Q</th> <th>\bar{Q}</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> </tr> </tbody> </table>	Q	\bar{Q}	0	1	1	0	 <p>at an output</p>									
Q	\bar{Q}																		
0	1																		
1	0																		
ANTIVALENCE Exclusive OR XOR function	\oplus	$I_1 \oplus I_2 = Q$ $(I_1 \wedge \neg I_2) \vee (\neg I_1 \wedge I_2) = Q$	<table border="1"> <thead> <tr> <th>I_1</th> <th>I_2</th> <th>Q</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	I_1	I_2	Q	0	0	0	0	1	1	1	0	1	1	1	0	
I_1	I_2	Q																	
0	0	0																	
0	1	1																	
1	0	1																	
1	1	0																	

Table 1.2.2.1: Boolean operations for binary operands with switching table and graphical symbol (excerpt);

*)Representation not according to standard but de facto standard

1.2.2.1.3 R/S-flip flops

A R/S flip flop is a storing function with two input variables R and S and an output variable Q, where the value $Q = 1$ belongs to the input states $R = 0$ and $S = 1$ and $Q = 0$ belongs to the input states $R = 1$ and $S = 0$. Here is to be declared which value is taken by Q in the case of $R = 1$ und $S = 1$, whereas Q keeps ist previous value in case of $R = 0$ und $S = 0$.



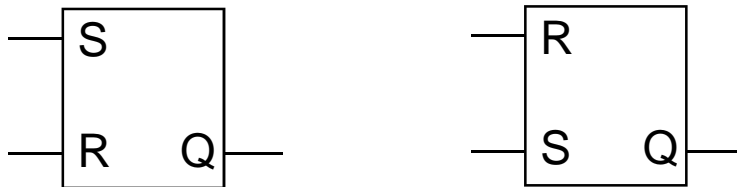
R	S	Q_{k+1}
0	0	Q_k
0	1	1
1	0	0
1	1	0

a) Graphical symbol

b) Switching table

Figure 1.2.2.1: Definition of a RS-flip flop with overriding R input

It proves workable in the software implementation to define the priority of the inputs by their order (the statement processed last determines the result of the logic operation RLO).



a) overrriding R input

b) overriding S input

Figure 1.2.2.2: RS flip flop, dominance is determined by the order of processing

1.2.2.1.4 Binary Timers

Syntax	Meaning
SI T	Start a timer as a pulse timer
SE T	Start a timer as an on delay
SA T	Start a timer as an off delay

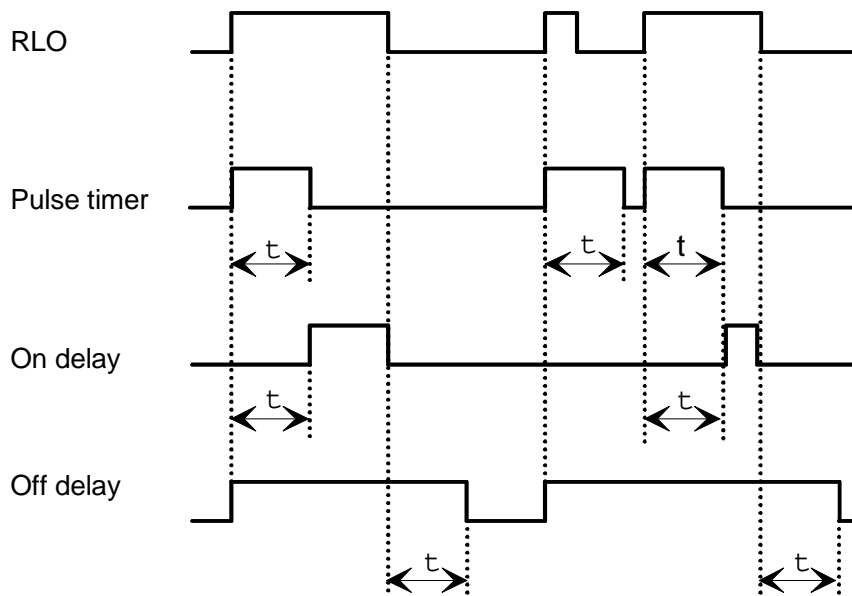


Figure 1.2.2.2: Behaviour of timers

1.2.2.2 Algebraic operations

1.2.2.2.1 Data types

Beneath the data type BOOL there are e.g. data types as follows (here: Example SIEMENS):

Data type	Description	(Width)	Example for constant notation
BYTE	Byte 8-bit hexadecimal number	8 Bits	B#16#01
CHAR	One character (ASCII)	8 Bits	,A'

WORD	Word 16-bit hexadecimal Number 16-bit binary number Count value, 3 decades BCD 2x8-bit unsigned decimal numbers	16 Bits	W#16#F100 2#1100_0011_1001_0110 C#250 B#(20,103)
DWORD	Double word 32-bit hexadecimal number 32-bit binary number 4x8-bit unsigned decimal numbers	32 Bits	DW#16#E800_0054 2#00111100_11000011_10010110_11110000 B#(1,55,3,0)
INT	Fixed-point number	16 Bits	-32768 to +32767
DINT	Fixed-point number	32 Bits	-2 147 483 648 to +2 147 483 647
REAL	Floating-point number	32 Bits	Decimal number with dot (1.0) or exponential representation (+1.234567E-01)
S5TIME	Time value in S5-format	16 Bits	S5T#0ms or S5TIME#2h46m30s
TIME	Time value in IEC format	32 Bits	T#0ms or TIME#24d20h31m23s647ms
DATE	Date	16 Bits	D#1990-01-01 or DATE#2168-12-31
TIME_OF_DAY	Time of day	32 Bits	TOD#00:00:00 or TIME_OF_DAY#23:59:59.999

Table 1.2.2.2: Overview of elementary data types

Data type	Description		Example
DATE_AND_TIME	Date and time	64 Bits	DT#1990-01-01-00:00.000 DATE_AND_TIME#2168-12-31:23:59:59.999
STRING	String of characters	variable	String of ASCII characters, e.g. ,String 1'
ARRAY	Field, array	variable	Array of components with same data type
STRUCT	Structure	variable	Structure of components with arbitrary data types

Table 1.2.2.3: Overview of extended data types

1.2.2.2.2 Operand types

The following operand types are applied:

Type:	Indicator:
- constants	- without indicator -
- inputs	E, %I *)
- outputs	A, %Q *)
- memory data	M, %M *)
- local data	L *)
- periphery inputs	PE *)
- periphery outputs	PA *)
- timers	T
- counters	C
- data	D *)
- system data	SM *)

*) : Here the indicator is to be supplemented by a second character (e.g. W for WORD or D for DOUBLE WORD) to determine the type exactly.

1.2.2.2.3 Computing operations

● Compare operations

e.g: <, >, <=, >=, ==, <>

The result of a compare operation is a boolean variable, which can be further processed by binary operations, but which can also be used to make the program flow depend on

it.

- Basic arithmetic operations

Syntax e.g. ADD, SUB, MUL, DIV

(SIEMENS: +, -, *, / each with specification of the data type: I for INT, D for DINT, R for REAL)

Tip: On Multiplication and Division the number representation is to be regarded!

The result of a **Fixed-point**-Multiplication of two numbers of a certain width (e.g. Type WORD with 16 bit) is a number which requires a representation in twice the width (e.g. Type DOUBLE WORD with 32 bits).

Because: $2^{15} \times 2^{15} = 2^{30}$

On division the result (Quotient) gets the same type as the Dividend.

Because: $2^{15} : 1 = 2^{15}$

The representation in Floating Point is like the Fixpoint representation defined only for a certain (bigger) number range. Overflow in the exponent is to be treated by the programmer!

On division always the division by 0 is to be taken care of!

This can either take place by making sure that the divisor can never be zero (by programming) or by evaluating the status informations of the processor and by exception processing.

1.2.2.2.4 Load and assignment operations

The load operation serves to provide a input variable for an operation.

Operator indicator: L or LD

Some automation devices also know the transfer function. With the aid of it a value is assigned to a variable (e.g. the content of the accumulator). Basically this function can also be implemented by the = character.

Operator indicator: T or ST

1.2.2.2.5 Organizational functions

Branch functions, conditionally or unconditionally.

Conditions: RLO, internal status variables

Examples to the syntax:

SPA	[label]	Branch unconditionally
SPB	[label]	Branch conditionally (depending on RLO)
SPO	[label]	Branch on overflow

1.2.2.2.6 Shift functions

Shift and rotate (to the left or to the right) on accumulator or memory variables.

Examples to the syntax:

SLW	3	Shift left by three positions in a word
SRW	5	Shift right by five positions in a word
RLD	6	Rotate left by six positions in a double word

1.2.2.2.7 Type conversions

Examples to the syntax:

BTI	Fixed point conversion from BCD to INT
ITB	Fixed point conversion from INT to

BCD

DTR Conversion of a fixed point number to
a floating point number

RND Conversion of a floating point number
to a fixed point number

Page 29 intentionally left blank

1.2.3 Kinds of representation of the application software

1.2.3.1 Statement list (STL)

1.2.3.1.1 Control statements

As shown in table 1.2.3.1 a **statement list** consists of a number of **control statements**. Each control statement has to start in a new line and has to contain an **operator** (eventually with a so called **Modifier**). If necessary for the particular operation the control statement contains one or more **Operands**, separated from one another by a comma.

The control statement can contain a preceding **Label**, followed by a colon (:). If a **comment** is present it has to be the last element in a line. Between the control statements blank lines can be inserted.

Label	Operator	Operand	Comment
START:	L	I0.1	Pushbutton
	AN	M0.5	Not locked
	S	Q0.2	Fan on

Table 1.2.3.1: Examples for the fields of a statement

In case of binary operands the pre-occupation of the RLO can also be done by the first statement of a AND or OR operation.

1.2.3.1.2 Operations, Modifiers and Operands

Some operators with their modifiers and operands are shown in Table 1.2.3.2 below.

If the definition in Table 1.2.3.2 is not different, the semantics as follows is to be applied:

RLO = RLO OP Operand

(RLO : Result of the preceding logic operation)

This means that the new value of the RLO emerges from its old value by processing it with the operator and the new operand; e.g. the statement

A „Value“

is to be understood as

RLO = RLO AND „Value“.

The **Modifier** "N" causes the boolean negation of the operand; e.g. the statement

AN „INPUT_1“

is to be understood as

RLO = RLO AND NOT(„INPUT_1“)

The Modifier left parenthesis "(" indicates, that the processing of the operator is suspended, until an operator right parenthesis ")" occurs; e.g. the series of statements

A(
O „INPUT_2“
O „INPUT_3“
)

is to be interpreted as

RLO = RLO AND („INPUT_2“ OR „INPUT_3“)

The Modifier "C" (conditionally) indicates, that the respective statement is only processed, if the previously determined RLO is a boolean "1" (or a boolean "0", if the operator is combined with the "N"-modifier).

Operator	Modifier	Operand	Semantics
A	N, (Boole	boolean AND
O	N, (Boole	boolean OR
N	(Boole	boolean NOT
XOR	N, (Boole	boolean EXCLUSIVE OR
=, ST	N, see Rem. 3	all variable types	Assignment
S	see Rem. 1	Boole	sets boolean operand to „1"
R	siehe Bem. 1	Boole	sets boolean operand to „0"
)			Evaluation of the suspended operation
CF		Counter value	Count forward
CB		Counter value	Count backward
ADD, +		Integer, Real	Addition
SUB, -		Integer, Real	Subtraction
Mul, *		Integer, Real	Multiplication
DIV, :		Integer, Real	Division
GT, >	see Rem. 2	Integer, Real	Comparison: >
GTE, >=	see Rem. 2	Integer, Real	Comparison: >=
EQ, =	see Rem. 2	Integer, Real	Comparison: =
LT, <	see Rem. 2	Integer, Real	Comparison: <
LTE, <=	see Rem. 2	Integer, Real	Comparison: <=
BTI		Integer	Code converter Decimal/Binary
ITB		Integer	Code converter Binary/Decimal
NOP			Null operation

Operator	Modifier	Operand	Semantics
L		all variable types	Load (provides the operands for subsequent operations)
SP	C	Label	Jump to label
BA	C	Name	Invocation of a block
CALL	C	Name	Invocation of a block
BE	C		Block end

Rem. 1: These operations are only processed, when the result of logic operation RLO is of the value "1".

Rem. 2: result is of Type "Bool"

Rem. 3: Only with binary Operand

Table 1.2.3.3

1.2.3.2 Ladder diagram (LAD)

A LAD program provides processing, test and modification of data by means of standardised graphical symbols. These symbols are displayed in networks in a manner similar to the current paths (rungs) in a relay logic diagram. LAD networks are limited to the left and to the right by **conductor rails**.

● Conductor rails


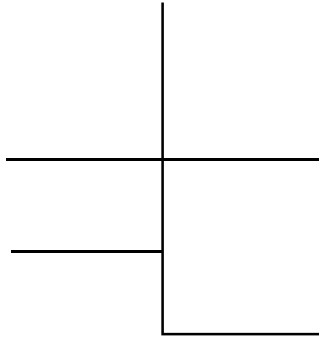
A network is limited to the left by a vertical line, known as the "left conductor rail", and to the right by a vertical line, known as "right conductor rail". The right conductor rail may be omitted.

● Connection elements and states

Connection elements are horizontal or vertical lines. The state of a connection element is ON or OFF, corresponding to the boolean values 1 or 0 resp.

The state of the left conductor rail must be regarded as „ON“; unless it is connected to an inactive step (Definition of a step see later in chapter 1.2.4 about sequence control). For the right conductor rail no state is defined.

A connection element is displayed by a line. A horizontal connection element transfers the state of an element to its left to the element on its right side.

Symbol	Description
	Horizontal connection
	Vertical connection in combination with horizontal connections

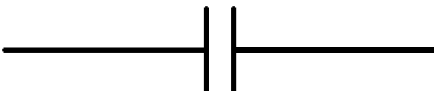
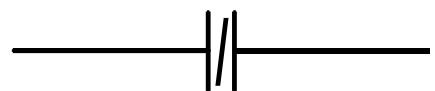
The state of the vertical connection is the OR function of the "ON" states of its left hand horizontal connections; this means the state of the vertical connection is as follows:

- OFF, if the states of all connected horizontal lines on its left side are OFF;
- ON, if the state of one or more connected lines on its left side is ON.

The state of the vertical line determines the state of all horizontal lines connected to the right side. The state is not copied to one of the connections on the left side.

● Scan operations (Contacts)


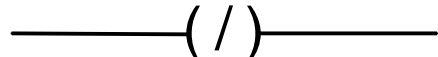
A contact is an element, that passes the state of the horizontal connection on its left side to the horizontal connection on its right side, if the state of the corresponding boolean input, output or memory variable is ON. A contact does not change the state of the corresponding boolean variable.

Symbol	Beschreibung
<p style="text-align: center;">***</p> 	<p>"NO Contact" (normally open)</p> <p>The state of the left connection is passed to the right connection, if the state of the corresponding boolean variable (marked with ***) is ON. Otherwise the state of the connection to the right is OFF.</p>
<p style="text-align: center;">***</p> 	<p>"NC Contact" (normally closed)</p> <p>The state of the left connection is passed to the right connection, if the state of the corresponding boolean variable (marked with ***) is OFF.</p>

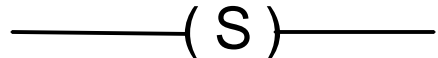
● **Assignment operations ("Coils")**

A coil copies the state of its left hand connection to the state of its right hand connection without any change and takes over the state of its left hand connection into the associated boolean variable.

Assignment operations ("Coils")

Symbol	Beschreibung
<p style="text-align: center;">***</p> 	<p>"Coil"</p> <p>The state of the left hand connection is copied to the boolean variable and to the right hand connection.</p>
<p style="text-align: center;">***</p> 	<p>"Negative Coil"</p> <p>The state of the left hand connection is copied to the right hand connection. The inverse state of the left hand connection is copied to the associated boolean variable. i.e. if the state of the left hand connection is OFF the state of the associated variable is set to ON and vice versa.</p>

Memory functions ("Set and Reset Coils")

Symbol	Description
<p style="text-align: center;">***</p> 	<p>"SET coil"</p> <p>The associated boolean variable is SET to the ON state, if the left hand connection is in the ON state and remains set until the coil is RESET.</p>

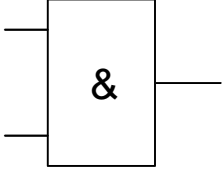
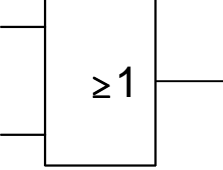

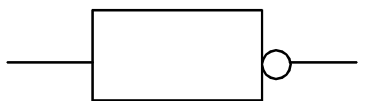
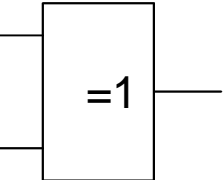
<p style="text-align: center;">***</p> <p style="text-align: center;">— (R) —</p>	<p>"RESET coil"</p> <p>The associated boolean variable is RESET, if the left hand connection is in the ON state and remains reset until the coil is SET.</p>
---	--

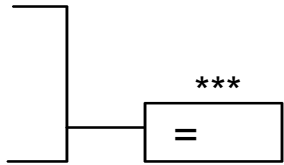
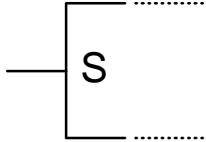
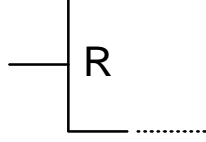
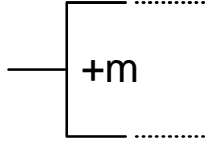
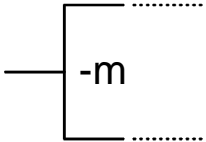
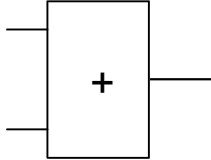
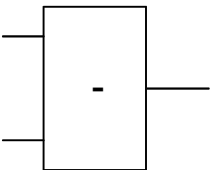
Pages 38-39 intentionally left blank

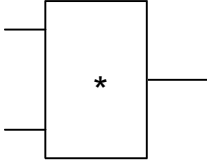
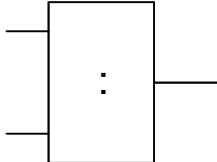
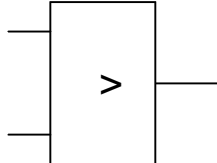
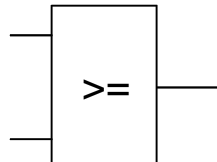
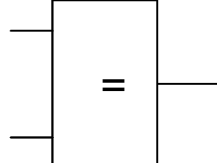
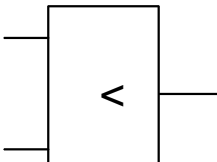
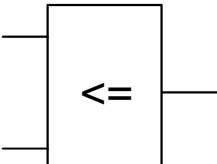
1.2.3.3 Function block diagram (FBD)

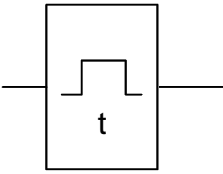
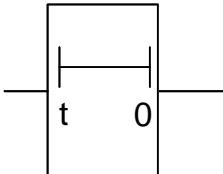
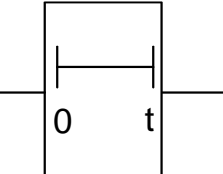


With function block diagrams arbitrary combinatory and sequential controls including associated arithmetic and closed loop control functions can be implemented.

Like in ladder diagram directly connected functional elements can be denoted as **Networks**. The single networks are separated from one another explicitly.

Name	Character in STL	Symbol in FBD
AND	A	
OR	O	
NOT	N	at the input 
		at the output 
Exclusive-OR	XOR	

Name	Character in STL	Symbol in FBD
Assignment	=, ST	
Set	S	
Reset	R	
Count forward	CF	 <p data-bbox="1011 1122 1382 1240">Count (+1) when signal changes from "0" to "1"</p>
Count backward	CB	 <p data-bbox="1011 1449 1382 1568">Count (-1) when signal changes from "0" to "1"</p>
Add	ADD, +	
Subtract	SUB, -	

Name	Character in STL	Symbol in FBD
Multiply	MUL, *	
Divide	DIV, :	
Greater than	GT	
Greater than or equal	GTE	
Equal	EQ	
Less than	LT	
Less than or equal	LTE	

Name	Character in STL	Symbol in FBD
Pulse timer	SI Tx	<p style="text-align: center;">Tx</p> 
On delay timer	SE Tx	<p style="text-align: center;">Tx</p> 
Off delay timer	SA Tx	<p style="text-align: center;">Tx</p> 
Code converter Decimal/Binary	BTI	
Code comnverter Binary/Decimal	ITB	

pages 44-49 intentionally left blank

1.2.4 Sequential function chart (SFC)

The elements of the sequential control language provide means to subdivide a PLC program into a set of **Steps** and **Transitions**, which are connected with one another by **directional connections**. To each step belongs a set of **Actions** and to each transition belongs a **transition condition**.

1.2.4.1 Steps

A step corresponds to a situation, in which the program behaves with regard to its inputs and outputs according to a set of certain rules. These rules are defined by the **actions** belonging to the step. A step is either **active** or **inactive**; at each given point of time the state of the system is defined by the set of steps being active and the values of the internal and external variables.

As indicated in table 1.2.4.1 a step is represented by a symbol containing a **Step name**. The directional connection coming to the step is represented by a vertical line connected at the top. The directional connection(s) leaving the step are represented by vertical lines connected to the bottom of the step.

The **Step flag** (active or inactive state of the step) is represented by a boolean variable. This boolean variable is "1", if the corresponding step is active and "0", if it is inactive.

The initial state of the program is determined by the initial values of its internal and external variables and by the set of initial steps, i.e. by the steps being active at the beginning.

A initial step kann be marked by twin lines at the margin.

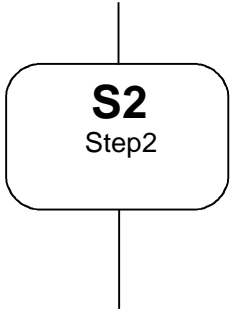
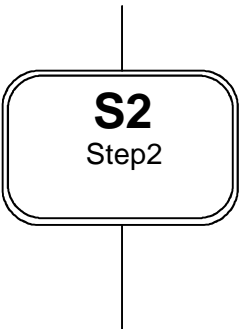
Symbol	Description
	<p>Step - Graphical representation</p> <ul style="list-style-type: none"> - with directional connections - "S2 / Step 2" = Step name (example)
	<p>Initial step</p> <ul style="list-style-type: none"> - Graphical representation - with directional connections - "S2 / Step2" = Name of initial step (example) <p>Note: The directional connection on the top is not necessary if the initial step has no predecessor</p>

Table 1.2.4.1 Step and initial step

1.2.4.2 Transitions

A transition specifies the condition which causes the control to move from one or more steps preceding the transition to one or more steps following the transition along the directional connection. The transition is represented by a horizontal line crossing the vertical directional connection. A boolean variable or expression, which specifies the transition condition, is associated with it(see Table 1.2.4.2).

The direction of control movement is from the bottom side of a predecessor step (or more than one) to the top side of a following one (ore more than one). It moves along the directonal connections.

Each transition has an associated **Transition condition**, which is the result of the analysis of a single boolean expression.

A transition condition, which is always true, is to be represented by the symbol "1".

A transition condition can be connected to a transition by one of the following means (see Table 1.2.4.2):

- (a) by a ladder diagram network in the language LAD, whose output is linked to the vertical directional connection instead of a conductor rail.
- (b) by a network in the FBD language, whose output is linked to the vertical directional connection.
- (c) by a LAD or FBD network, whose output is linked to the vertical directional connection by a connector.
- (d) by application of a transition label in the form of an identifier on the right side of the directional connection. This label refers to a construction defined by one of the following entities: (The analysis of the entity ends with the assignment of a boolean value to the transition label)
 - a network in the LAD or FBD language;
 - a list of control statements in the STL language.

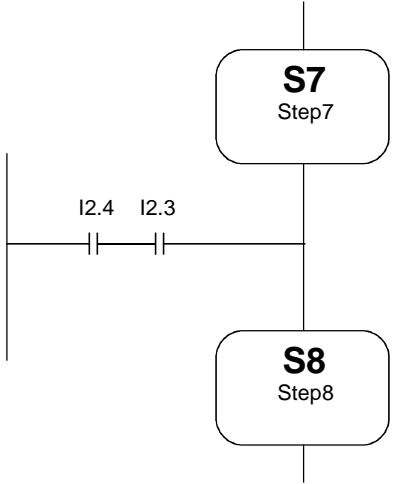
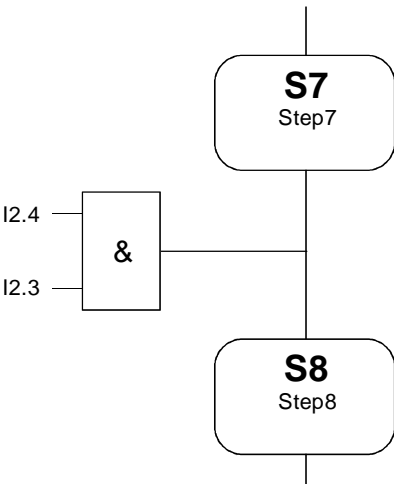
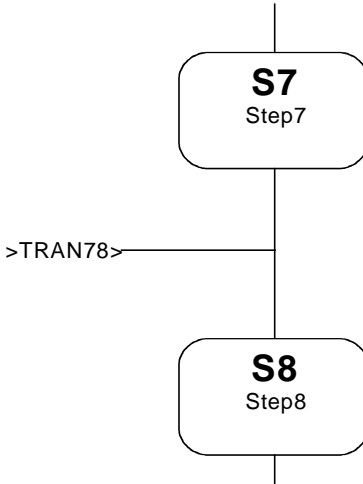
Nr.	Example	Description
(a)	 <p>The diagram shows a vertical line representing the step sequence. At the top is a rounded rectangle labeled 'S7 Step7'. Below it is a horizontal line representing the transition. This transition contains two normally closed contacts in series, labeled 'I2.4' and 'I2.3'. Below the transition is another rounded rectangle labeled 'S8 Step8'.</p>	<p>Preceding step</p> <p>Transition condition</p> <p>Following step</p>
(b)	 <p>The diagram shows a vertical line representing the step sequence. At the top is a rounded rectangle labeled 'S7 Step7'. Below it is a horizontal line representing the transition. This transition is connected to an AND gate symbol (a rectangle with an ampersand '&'). The AND gate has two inputs on its left side, labeled 'I2.4' and 'I2.3'. Below the transition is another rounded rectangle labeled 'S8 Step8'.</p>	<p>Preceding step</p> <p>Transition condition</p> <p>Following step</p>
(c)	 <p>The diagram shows a vertical line representing the step sequence. At the top is a rounded rectangle labeled 'S7 Step7'. Below it is a horizontal line representing the transition. This transition is controlled by a timer symbol labeled '>TRAN78>'. Below the transition is another rounded rectangle labeled 'S8 Step8'.</p>	<p>Preceding step</p> <p>Transition condition</p> <p>following step</p>

Table 1.2.4.2 Part 1

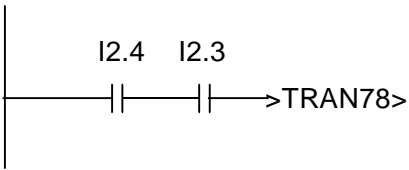
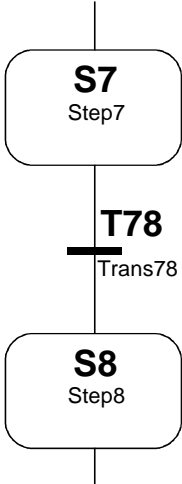
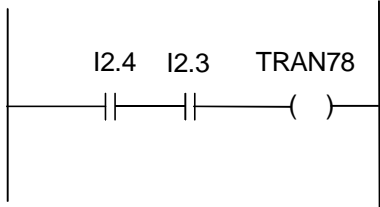
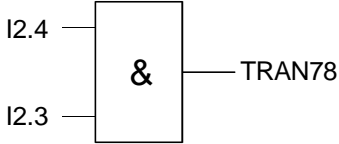
Nr.	Example	Description
		<p>Transition condition LAD language</p>
(d)		<p>Preceding step Transition mark Following step</p>
	<p>TRANSITION TRAN78</p>  <p>END_TRANSITION</p>	<p>Transition condition LAD language</p>
	<p>TRANSITION TRAN78</p>  <p>END_TRANSITION</p>	<p>Transition condition FBD language</p>
	<p>TRANSITION TRAN78</p> <pre>L I 2.4 A I 2.3 = TRAN78</pre> <p>END_TRANSITION</p>	<p>Transition condition STL language</p>

Table 1.2.4.2 Part 2

1.2.4.3 **Actions**

To each step one or more actions can be associated. The actions define the operations that are always to be performed when the respective step is active. A step, that has no associated actions is to be considered to have a **Waiting function** (i.e. to be waiting for a following transition to become true).

As indicated in table 1.2.4.3, actions can be concatenated with a step by a direct connection of the step symbol, namely from the right side of the step block to:

- (a) the left conductor rail of a ladder diagram in LAD language;
- (b) one or more networks in the FBD language;
- (c) an action block according to chapter 1.2.4.3.1
- (d) a block of control statements in the STL language

Connectors can be applied to extend the direct connections above.

Actions are to be considered to be performed continuously **as long as the associated step is active**. If a step is reset, the associated actions must be performed at least one more time with a step flag having now the value "0".

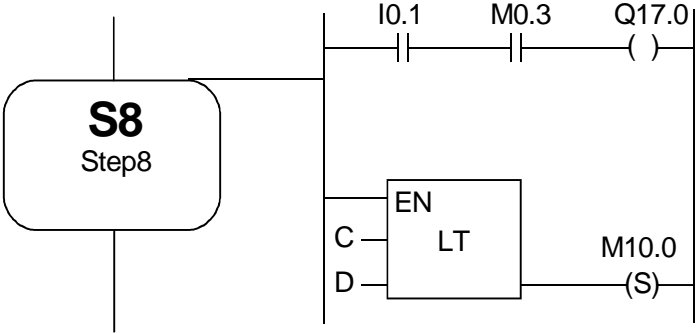
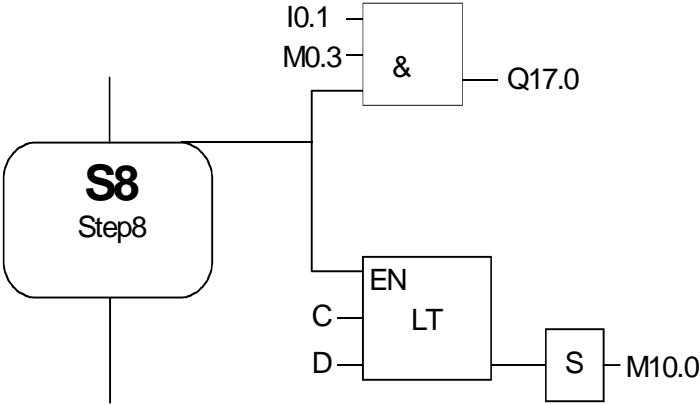
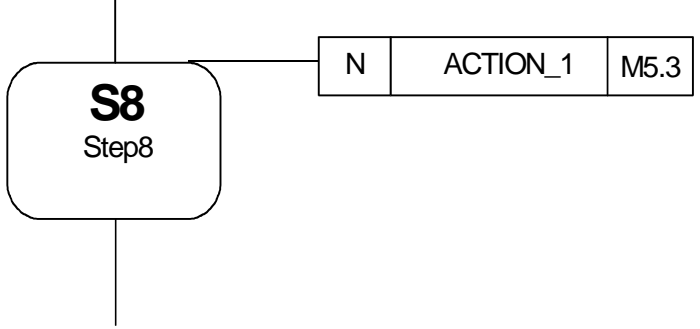
Nr.	Example	Description
(a)		LAD language
(b)		FBD language
(c)		Action block

Table 1.2.4.3 Part 1

Note: If S8 is inactive, the impact is fed to the logic connected; e.g. Q 17.0:= 0 und M 10.0 keeps its previous state.

Note: The block "Comparator with Enable input" applied above is a user defined block (so called Function block) and is not part of the basic operation pool of a PLC.

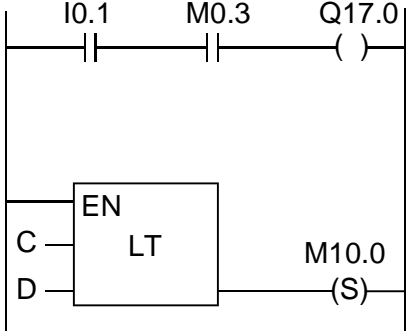
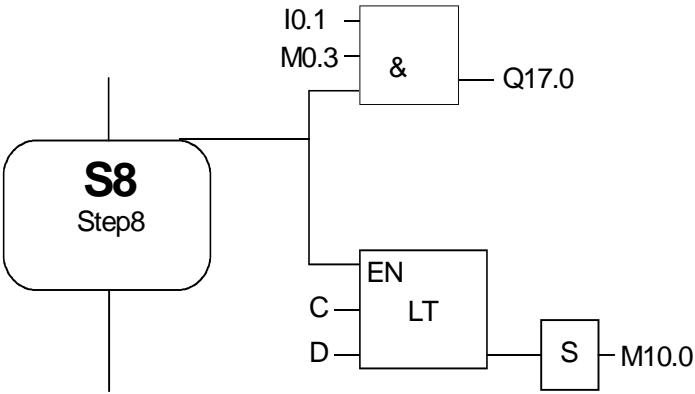
Nr.	Example	Description
	<p>STEP S8 :</p>  <p>END_STEP</p>	LAD language
	<p>STEP S8 :</p> 	FBD language
	<p>STEP S8 :</p> <pre> L S8 A I0.1 A M0.3 = Q17.0 L C LT D A S8 S M10.0 </pre> <p>END_STEP</p>	STL language

Table 1.2.4.3 Part 2

1.2.4.3.1 Action block

An action block permits the concise representation of complex actions. It is a means to connect a "Control structure" to a step.

The action block also provides a means to determine a boolean feedback variable, which can be used by the associated action to show its termination. When this field ("c" in Table 1.2.4.4) is not present, the boolean variable in field "b" is to be regarded as feedback variable if necessary.

Action blocks can, as displayed in table 1.2.4.4, be arranged in a row graphically. Such groups of action blocks may have several feedback variables but they must have only one boolean input variable, which acts on all blocks simultaneously.

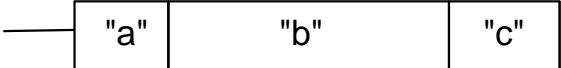
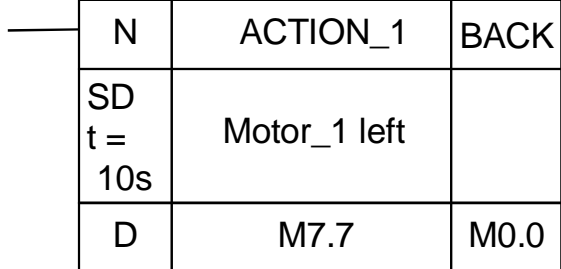
Symbol	Erklärung
	<p>General symbol</p> <p>"a": Qualification of the command acc. to Table 1.2.4.5</p> <p>"b": Textual description of the command</p> <p>"c": Reference label of the corresponding check back variable (optional)</p>
	<p>Example for action blocks arranged in a row</p>

Table 1.2.4.4

In Table 1.2.4.5 the valid values for the timing characterization of the action ("a" in table 1.2.4.4) are indicated, in combination with the associated control structure in the FBD language.


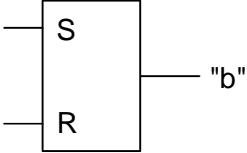
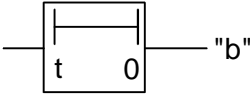
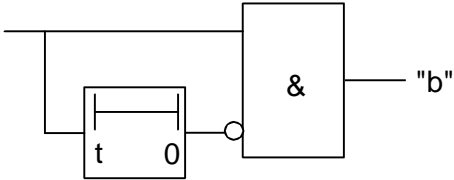
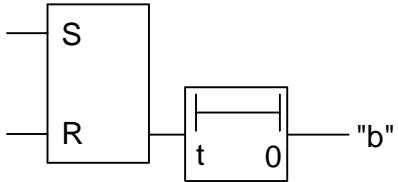
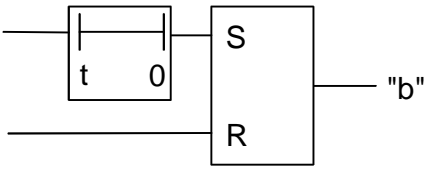
Timing	Description	Control structure/ remarks
N	not stored	
S R	set/ stored reset	
D	not stored, but delayed	
L	Not stored, but time limited	
SD R	stored and delayed reset	
DS R	delayed and stored reset	

Table 1.2.4.5

1.2.4.4 Sequence rules

The **Initial situation** of a sequence chain is characterized by the **Initial steps**, which are in an active state during the initialization of the program.

The **sequence** of the active step states takes place along the directional connections, triggered by the switching of one or more transitions.

A transition is **enabled**, if all preceding steps connected to the respective transition symbol are active. Switching of a transition takes place if the transition is enabled and at the same time the transition condition is fulfilled.

Switching of a transition leads to the simultaneous **activation** of all directly following steps being connected to the respective transition symbol and to the **deactivation** (or "reset") of all directly preceding steps.

The change step/transition and transition/step is always to be assured in the sequence plan, i.e.:

- Two steps must never be connected directly; they have always to be separated by a transition.
- Two transitions must never be connected directly; they have always to be separated by a step.

If the switching of a transition leads to the activation of multiple steps at the same time, then the sequence chains these steps belong to are called **Simultaneous chains**. After their activation the sequence of each of these chains is independent. To emphasize the special kind of these transitions, the branching and joining of simultaneous chains is displayed by a twin horizontal line.

Table 1.2.4.6 defines the syntax and semantics of possible combinations of steps and transitions in the sequence language.

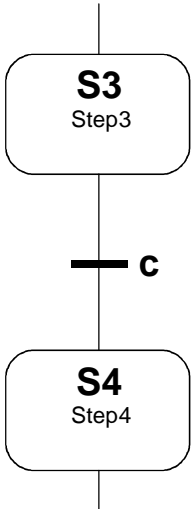
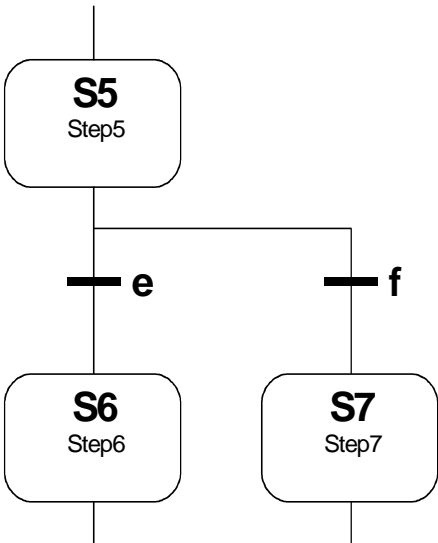
 <p>The diagram shows a vertical line starting from the top, leading to a rounded rectangular box labeled 'S3 Step3'. Below this box is a horizontal bar labeled 'c', representing a transition. A vertical line continues from the bottom of the 'c' bar to another rounded rectangular box labeled 'S4 Step4'. The line ends at the bottom of the box.</p>	<p>Simple chain:</p> <p>The change step-transition is always repeated.</p> <p>Example:</p> <p>A sequence from step S3 to S4 only takes place if step S3 is in an active state and the transition condition c is fulfilled.</p>
 <p>The diagram shows a vertical line starting from the top, leading to a rounded rectangular box labeled 'S5 Step5'. Below this box, the line splits into two horizontal branches. The left branch leads to a horizontal bar labeled 'e', which then leads to a rounded rectangular box labeled 'S6 Step6'. The right branch leads to a horizontal bar labeled 'f', which then leads to a rounded rectangular box labeled 'S7 Step7'. Both boxes have lines extending downwards from their bottom centers.</p>	<p>Branching in case of chain selection:</p> <p>A selection between several chains is displayed by as many transition symbols under the horizontal line, as different possible sequences are present.</p> <p>Example:</p> <p>A sequence from S5 to S6 may take place only, if S5 is active and the transition condition e is fulfilled, or from S5 to S7 only if S5 is active and f is fulfilled.</p>

Table 1.2.4.6, Part 1

Remark to "Branching with chain selection":

For selecting only one chain it is necessary, that the transition conditions connected to the chains are mutually exclusive, so that they are never true at the same time. It is always possible to include a order of priority into the formulation of the transition conditions.

	<p>Joining with chain selection:</p> <p>The end of a chain selection is displayed by as many transition symbols above the horizontal line, as selection paths to be finished exist.</p> <p>Example:</p> <p>A sequence from S7 to S10 may take place only, if S7 is active and the transition condition h is fulfilled or from S9 to S10 only if S9 is active and j is fulfilled.</p>
	<p>Branching of simultaneous chains:</p> <p>Only one common transition symbol directly above the twin horizontal synchronization line is possible.</p> <p>Example:</p> <p>A sequence from S11 to S12, S14, .. may only take place if S11 is active and the transition condition b, which is belonging to the common transition, is fulfilled. After the synchronous activation of S12, S14, and so on the chains sequences are independent of one another.</p>

Table 1.2.4.6, Part 2

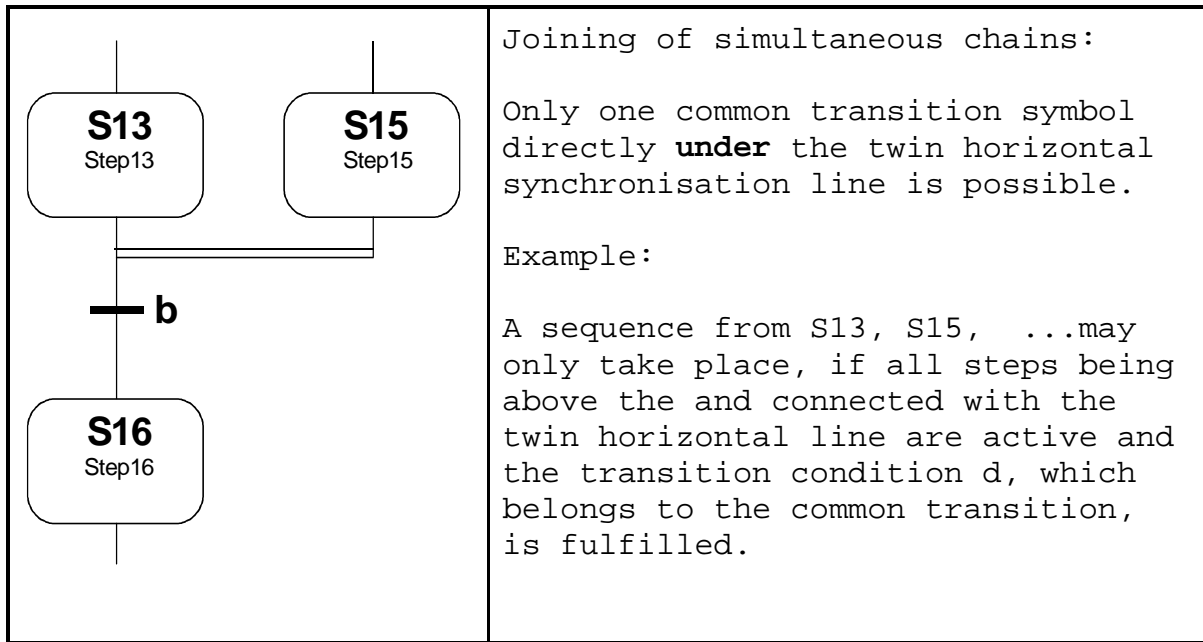
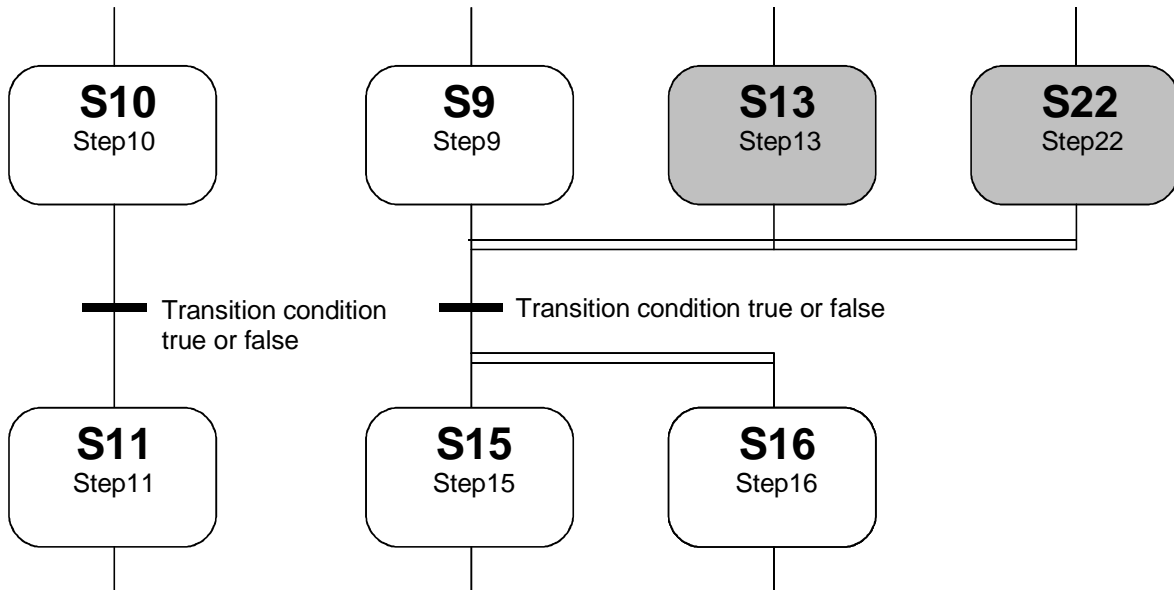


Table 1.2.4.6, Part 3

Figure 1.2.4.1 demonstrates the application of the rules mentioned above. In this figure the active state of a step is illustrated by an asterisk in the respective block; this representation is not part of the sequence language but is only for demonstration.

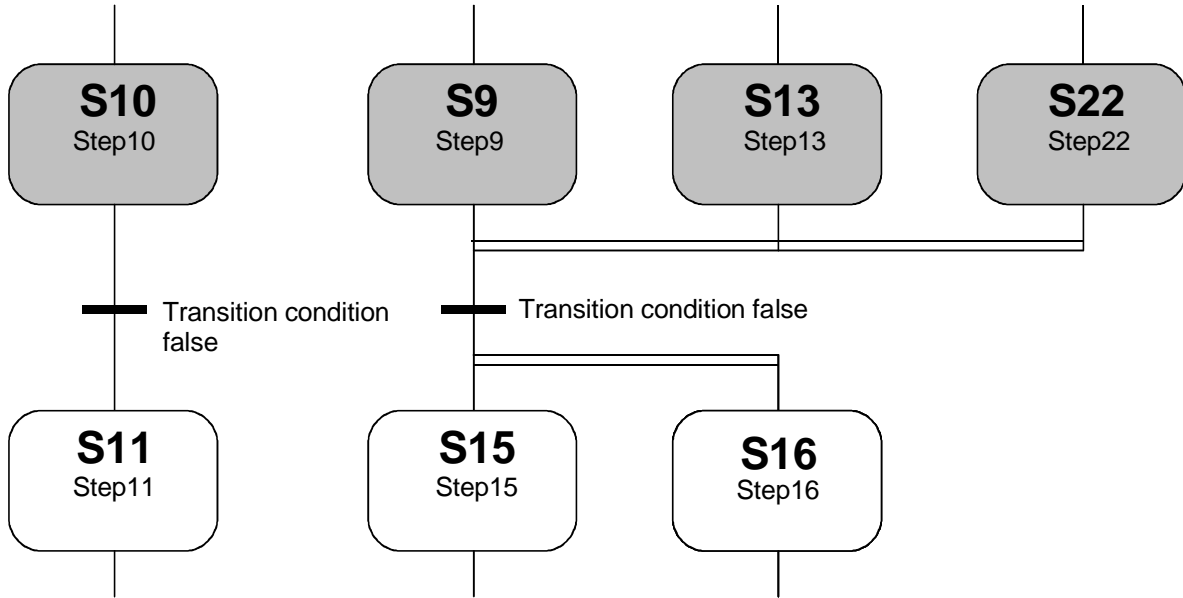


Transition not enabled

The transitions are not enabled, because the steps 9 or 10 resp. are not active. The corresponding transition conditions may be true or false.

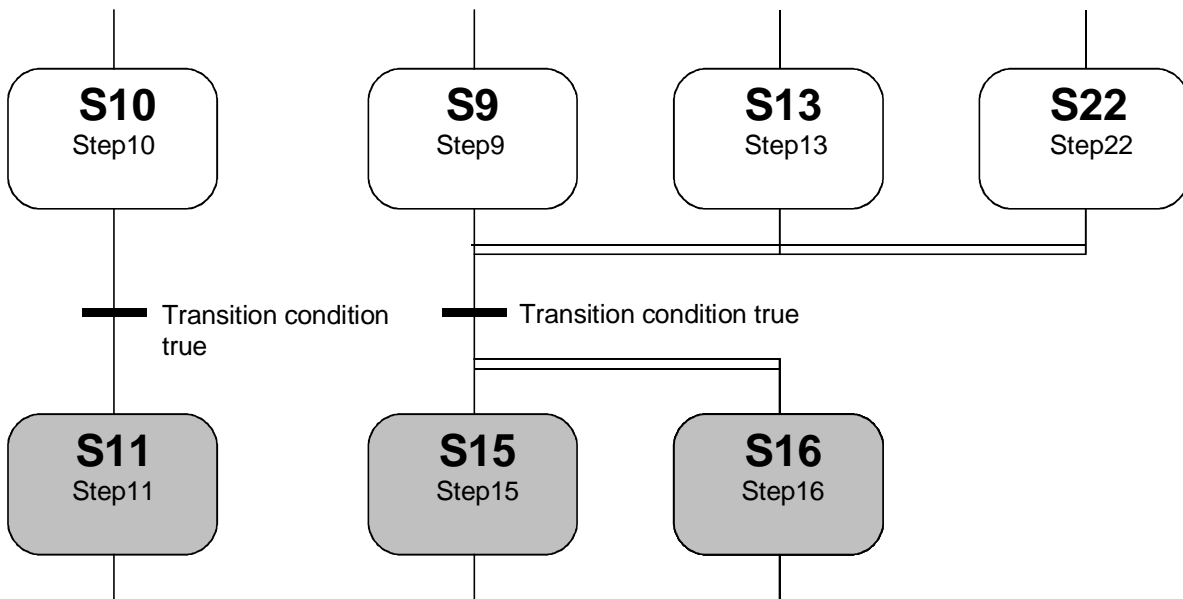
Fig. 1.2.4.1: Transitions not enabled, enabled or switching resulting from some preceding steps, first part of figure.

Remark: The shaded step symbol shows the active state of a step.



Transition enabled

The transitions are enabled, but can not switch, because the corresponding transition conditions are false.



Transition has switched

The transitions have switched now, because the corresponding transition conditions are true.

Figure 1.2.4.1: Transitions not enabled, enabled and switched resulting from some preceding steps, second part of figure

Literature

- Berger, H.: Automating with STEP7 in LAD and FBD, 2nd Edition 2001, Publicis MCD Corporate Publishing, ISBN 3-89578-170-3
- Dunning. G.: Introduction to Programmable Logic Controllers, 2nd Edition 2002, Delmar Thomson Learning