Programmable Logic Controllers

2.3 Automates Programmables
Speicherprogrammierbare Steuerungen

Prof. Dr. H. Kirrmann
ABB Research Center, Baden, Switzerland
2.3.1 PLCs: Definition and Market

2.1 Instrumentation
2.2 Control
2.3 Programmable Logic Controllers
   2.3.1 PLCs: Definition and Market
   2.3.2 PLCs: Kinds
   2.3.3 PLCs: Functions and construction
   2.3.4 Continuous and Discrete Control
   2.3.5 PLC Programming Languages
      2.3.5.1 IEC 61131 Languages
      2.3.5.2 Function blocks
      2.3.5.3 Program Execution
      2.3.5.4 Input / Output
      2.3.5.5 Structured Text
      2.3.5.6 Sequential Function Charts
      2.3.5.7 Ladder Logic
      2.3.5.8 Instruction Lists
      2.3.5.9 Programming environment
**PLC = Programmable Logic Controller: Definition**

**Definition:** “small computers, dedicated to automation tasks in an industrial environment"

**Formerly:** cabled relay control (hence 'logic'), analog (pneumatic, hydraulic) governors

**Today:** real-time (embedded) computer with extensive input/output

**Function:** Measure, Control, Protect

**Distinguish** **Instrumentation**

flow meter, temperature, position,… but also actors (pump, …)

**Control**

programmable logic controllers with digital peripherals & field bus

**Visualization**

HMI* in PLCs (when it exists) is limited to control of operator displays

*Human Machine Interface
PLC is a cabinet

CPU1

redundant field bus connection

CPU2

serial connections

inputs/outputs
example: turbine control (in the test lab)
PLC: functions

(Messen, Schützen, Regeln = MSR)

PLC = PMC: Protection, Measurement and Control

- Measure
- Control (Command and Regulation)
- Protection
  - Event Logging
  - Communication
  - Human interface
PLC: Characteristics

• large number of peripherals: 20..100 I/O per CPU, high density of wiring, easy assembly.
• binary and analog Input/Output with standard levels
• located near the plant (field level), require robust construction, protection against dirt, water and mechanical threats, electro-magnetic noise, vibration, extreme temperature range (-30C..85C)
• programming: either very primitive with hand-help terminals on the target machine itself, or with a lap-top able to down-load programs.
• network connection is becoming common, allowing programming on workstations.
• field bus connection for remote I/Os
• primitive Man-Machine interface, either through LCD-display or connection of a laptop over serial lines (RS232).
• economical - €1000.- .. €15'000.- for a full crate.
• the value is in the application software (licenses €20'000 ..€50'000)
PLC: Location in the control architecture

Control Bus (e.g. Ethernet)

Engineer station | Operator station | Supervisor Station

Enterprise Network

gateway

large PLCs

small PLC

very simple PLCs

Control Station with Field Bus

Field Bus

Field Stations

FB gateway

Field Devices

Sensor Bus (e.g. ASI)
PLC: manufacturers

Switzerland
  SAIA, Weidmüller

Europe:
  Siemens (60% market share) [Simatic],
  ABB (includes Hartmann&Braun, Elsag-Bailey, SattControl,…) [IndustrialIT],
  Groupe Schneider [Télémécanique],
  WAGO,
  Phoenix Contact ...

World Market:
  GE-Fanuc,
  Honeywell,
  Invensys (Foxboro)
  Rockwell, (Allen-Bradley,…)
  Emerson (Fisher Control, Rosemount, Westinghouse)
  Hitachi, Toshiba, Fujitsu, Yokogawa
  ...

large number of bidders, fusions and acquisitions in the last years.
Distinguish PLCs for the open market (OEM) and proprietary PLCs
2.3.3 PLCs: Kinds

2.1 Instrumentation
2.2 Control
2.3 Programmable Logic Controllers
   2.3.1 PLCs: Definition and Market
   2.3.2 PLCs: Kinds
   2.3.3 PLCs: Functions and construction
   2.3.4 Continuous and Discrete Control
   2.3.5 PLC Programming Languages
      2.3.5.1 IEC 61131 Languages
      2.3.5.2 Function blocks
      2.3.5.3 Program Execution
      2.3.5.4 Input / Output
      2.3.5.5 Structured Text
      2.3.5.6 Sequential Function Charts
      2.3.5.7 Ladder Logic
      2.3.5.8 Instruction Lists
      2.3.5.9 Programming environment
Kinds of PLC

(1) Compact
Monolithic construction
Monoprocessor
Fieldbus connection
   Fixed casing
   Fixed number of I/O (most of them binary)
   No process computer capabilities (no MMC)
   Typical product: Mitsubishi MELSEC F, ABB AC31, SIMATIC S7

(2) Modular PLC
Modular construction (backplane)
One- or multiprocessor system
Fieldbus and LAN connection
   3U or 6U rack, sometimes DIN-rail
   Large variety of input/output boards
   Connection to serial bus
   Small MMC function possible
   Typical products: SIMATIC S5-115, Hitachi H-Serie, ABB AC110

(3) Soft-PLC
Windows NT or CE-based automation products
Direct use of CPU or co-processors
Modular PLC

- tailored to the needs of an application
- housed in a 19" (42 cm) rack (height 6U (= 233 mm) or 3U (=100mm)
- high processing power (several CPU)
- large choice of I/O boards
- concentration of a large number of I/O
- interface boards to field busses
- requires marshalling of signals
- primitive or no HMI
- cost effective if the rack can be filled
- supply 115-230V~, 24V= or 48V= (redundant)
- cost ~ €10’000 for a filled crate
Small modular PLC

mounted on DIN-rail, 24V supply cheaper (€5000)
not water-proof,
no ventilator
extensible by a parallel bus (flat cable or rail)

courtesy ABB
courtesy Backmann
Specific controller (railways)

three PLCs networked by a data bus.

special construction: no fans, large temperature range, vibrations
Compact or modular?

- **compact PLC** (fixed number of I/Os)
- **modular PLC** (variable number of I/Os)

*field bus extension*

Limit of local I/O

# I/O modules
Industry- PC

Wintel architecture
   (but also: Motorola, PowerPC),
MMI offered (LCD..)
Limited modularity through mezzanine boards
   (PC104, PC-Cards, IndustryPack)
Backplane-mounted versions with PCI or Compact-PCI

Competes with modular PLC
no local I/O,
fieldbus connection instead,
costs: € 2000.-
Soft-PLC (PC as PLC)

- PC as engineering workstation
- PC as human interface (Visual Basic, Intellution, Wonderware)
- PC as real-time processor (Soft-PLC)
- PC assisted by a Co-Processor (ISA- or PC104 board)
- PC as field bus gateway to a distributed I/O system
Compact PLC

Monolithic (one-piece) construction
Fixed casing
Fixed number of I/O (most of them binary)
No process computer capabilities (no MMC)
Can be extended and networked by an extension (field) bus
Sometimes LAN connection (Ethernet, Arcnet)
Monoprocessor

Typical product: Mitsubishi MELSEC F, ABB AC31, SIMATIC S7

costs: € 2000
Specific Controller (example: Turbine)

tailored for a specific application, produced in large series

cost: € 1000.-

courtesy Turbec
Protection devices are highly specialized PLCs that measure the current and voltages in an electrical substation, along with other statuses (position of the switches,…) to detect situations that could endanger the equipment (over-current, short circuit, overheat) and triggers the circuit breaker ("trip") to protect the substation.

In addition, it records disturbances and sends the reports to the substation’s SCADA.

Sampling: 4.8 kHz, reaction time: < 5 ms.

Costs: € 5000
2.3 Programmable Logic Controllers

Industrial Automation

Market share

<table>
<thead>
<tr>
<th>Type</th>
<th>% Installed PLCs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Micro: 15 to 128 I/O points</td>
<td>32%</td>
</tr>
<tr>
<td>Medium: 128 - 512 I/O points</td>
<td>29%</td>
</tr>
<tr>
<td>Large: &gt; 512 I/O points</td>
<td>20%</td>
</tr>
<tr>
<td>Nano: &lt; 15 I/O points</td>
<td>7%</td>
</tr>
<tr>
<td>PC-based</td>
<td>6%</td>
</tr>
<tr>
<td>Software PLC</td>
<td>4%</td>
</tr>
<tr>
<td>Embedded control</td>
<td>2%</td>
</tr>
</tbody>
</table>

Source: Control Engineering, Reed Research, 2002-09
Global players

Total sales in 2004: 7'000 Mio €

Source: ARC Research, 2005-10
## Comparison Criteria – Example

<table>
<thead>
<tr>
<th>Brand</th>
<th>Siemens</th>
<th>Hitachi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Points</td>
<td>1024</td>
<td>640</td>
</tr>
<tr>
<td>Memory</td>
<td>10 KB</td>
<td>16 KB</td>
</tr>
</tbody>
</table>
| Programming Language | • Ladder logic  
• Instructions  
• Logic symbols  
• Hand-terminal | • Ladder Logic  
• Instructions  
• Logic symbols  
• Basic  
• Hand-terminal |
| Programming Tools | • Graphic on PC  
no | • Graphic on PC  
yes |
| Download | | |
| Real estate per 250 I/O | 2678 cm² | 1000 cm² |
| Label surface per line/point | 5.3 mm²  
7 characters | 6 mm²  
6 characters |
| Network | 10 Mbit/s | 19.2 kbit/s |
| Mounting | DIN rail | cabinet |
2.3.3 PLCs: Function and construction

2.1 Instrumentation
2.2 Control
2.3 Programmable Logic Controllers
  2.3.1 PLCs: Definition and Market
  2.3.2 PLCs: Kinds
  2.3.3 PLCs: Functions and construction
  2.3.4 Continuous and Discrete Control
  2.3.5 PLC Programming Languages
    2.3.5.1 IEC 61131 Languages
    2.3.5.2 Function blocks
    2.3.5.3 Program Execution
    2.3.5.4 Input / Output
    2.3.5.5 Structured Text
    2.3.5.6 Sequential Function Charts
    2.3.5.7 Ladder Logic
    2.3.5.8 Instruction Lists
    2.3.5.9 Programming environment
2.3 Programmable Logic Controllers

General PLC architecture

- CPU
- Real-Time Clock
- ROM
- flash EPROM
- serial port controller
- ethernet controller

Buffers

Parallel bus

fieldbus controller
- analog-digital converters
- signal conditioning

digital-analog converters
- power amplifiers
- relays
- signal conditioning

Digital Output

Digital Input

RS 232
Ethernet

extension bus

external I/Os

field bus

direct Inputs and Outputs
The signal chain within a PLC

- Analog variable (e.g. 4..20mA)
  - Filtering & scaling
  - Sampling
  - Analog-digital converter
  - Processing
  - Digital-analog converter
  - Amplifier
  - Analog variable (e.g. -10V..10V)

- Binary variable (e.g. 0..24V)
  - Filtering
  - Sampling
  - Counter
  - Non-volatile memory
  - Transistor or relay
  - Binary variable
Signal chain in a protection device

- Input transformer
- Anti-aliasing filter
- Sample and hold A/D conversion
- Digital filter
- Protection algorithm
- Output driver

- U/I
- A/D
- CPU
- Trip

-f = 1 MHz
-f = 200 kHz
-f = 100 kHz
-f = 300 - 1200 Hz
-reaction < 10 ms
2.3.4 Continuous and discrete control

2.1 Instrumentation
2.2 Control
2.3 Programmable Logic Controllers
   2.3.1 PLCs: Definition and Market
   2.3.2 PLCs: Kinds
   2.3.3 PLCs: Functions and construction
   2.3.4 Continuous and Discrete Control
   2.3.5 PLC Programming Languages
      2.3.5.1 IEC 61131 Languages
      2.3.5.2 Function blocks
      2.3.5.3 Program Execution
      2.3.5.4 Input / Output
      2.3.5.5 Structured Text
      2.3.5.6 Sequential Function Charts
      2.3.5.7 Ladder Logic
      2.3.5.8 Instruction Lists
      2.3.5.9 Programming environment
Matching the analog and binary world

discrete control

analog regulation
PLC evolution

Binary World
- relay controls, Relay control
- pneumatic sequencer

Analog World
- Pneumatic and electromechanical controllers

Discrete processes
- combinatorial
- sequential

Continuous processes

Programmable Logic Controllers
(Speicherprogrammierbare Steuerungen, Automates Programmables)
Continuous Plant (reminder)

Example: traction motors, ovens, pressure vessel,...

The state of continuous plants is described by continuous (analog) state variables like temperature, voltage, speed, etc.

There exist a fixed relationship between input and output, described by a continuous model in form of a transfer function F.

This transfer function can be expressed by a set of differential equations.

If equations are linear, the transfer function may expressed as Laplace or Z-transform.

\[ F(s) = \frac{(1+Ts)}{(1+T_1s + T_2s^2)} \]

Continuous plants are normally reversible and monotone.

This is the condition to allow their regulation.

The time constant of the control system must be at least one order of magnitude smaller than the smallest time constant of the plant.

The principal task of the control system for a continuous plant is its regulation.
The plant is described by variables which take well-defined, non-overlapping values. The transition from one state to another is abrupt, it is caused by an external event. Discrete plants are normally reversible, but not monotone, i.e. negating the event which caused a transition will not revert the plant to the previous state.

Example: an elevator doesn't return to the previous floor when the button is released.

Discrete plants are described e.g. by finite state machines or Petri nets.

The main task of a control system with discrete plants is its sequential control.
Continuous and Discrete Control (comparison)

"combinatorial"\(^1\)

<table>
<thead>
<tr>
<th>e.g. ladder logic, CMOS logic</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Ladder Logic Circuit" /></td>
</tr>
<tr>
<td><img src="image2" alt="CMOS Logic Circuit" /></td>
</tr>
</tbody>
</table>

- Out = A · B
- Out = (A + B) · C

"sequential"

<table>
<thead>
<tr>
<th>e.g. GRAFCET, Petri Nets</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image3" alt="Sequential Logic Circuit" /></td>
</tr>
</tbody>
</table>

1) not really combinatorial: blocs may have memory
2.3.5 Programming languages

2.1 Instrumentation
2.2 Control
2.3 Programmable Logic Controllers
  2.3.1 PLCs: Definition and Market
  2.3.2 PLCs: Kinds
  2.3.3 PLCs: Functions and construction
  2.3.4 Continuous and Discrete Control
  2.3.5 Programming languages
    2.3.5.1 IEC 61131 Languages
    2.3.5.2 Function blocks
    2.3.5.3 Program Execution
    2.3.5.4 Input / Output
    2.3.5.5 Structured Text
    2.3.5.6 Sequential Function Charts
    2.3.5.7 Ladder Logic
    2.3.5.8 Instruction Lists
    2.3.5.9 Programming environment
Extend procedural languages to express time
("introduce programming constructs to influence scheduling and control flow")

- ADA
- Real-Time Java
- MARS (TU Wien)
- Forth
- “C” with real-time features
- etc...

"Real-Time" languages

languages developed for cyclic execution and real-time
("application-oriented languages")

- ladder logic
- function block language
- instruction lists
- GRAFCET
- SDL
- etc...

could not impose themselves

wide-spread in the control industry.
Now standardized as IEC 61131
The long march to IEC 61131

NEMA Programmable Controllers Committee formed (USA)

GRAFCET (France)

DIN 40719, Function Charts (Germany)

NEMA ICS-3-304, Programmable Controllers (USA)

IEC SC65A/WG6 formed

DIN 19 239, Programmable Controller (Germany)

IEC 65A(Sec)38, Programmable Controllers

MIL-STD-1815 Ada (USA)

IEC SC65A(Sec)49, PC Languages

IEC 64A(Sec)90

IEC 848, Function Charts

IEC 1131-3

Type 3 report recommendation

IEC 61131-3 name change

Source: Dr. J. Christensen

it took 20 years to make that standard…
The five IEC 61131-3 Programming languages

**Function Block Diagram (FBD)**

- **Ladder Diagram (LD)**

- **Instruction List (IL)**
  
  ```
  A: LD %IX1 (* PUSH BUTTON *)
  ANDN %MX5 (* NOT INHIBITED *)
  ST %QX2 (* FAN ON *)
  ```

- **Structured Text (ST)**
  
  ```
  VAR CONSTANT X : REAL := 53.8 ;
  Z : REAL; END_VAR
  VAR aFB, bFB : FB_type; END_VAR
  bFB(A:=1, B:="OK");
  Z := X - INT_TO_REAL (bFB.OUT1);
  IF Z>57.0 THEN aFB(A:=0, B:="ERR");
  ELSE  aFB(A:=1, B:="Z is OK");
  END_IF
  ```

- **Sequential Flow Chart (SFC)**

  ![SFC Diagram](http://www.isagraf.com)
Importance of IEC 61131

IEC 61131-3 is the most important automation language in industry.

80% of all PLCs support it, all new developments base on it.

Depending on the country, some languages are more popular.
2.4.2.1 Function Blocks Language

2.1 Instrumentation
2.2 Control
2.3 Programmable Logic Controllers
   2.3.1 PLCs: Definition and Market
   2.3.2 PLCs: Kinds
   2.3.3 PLCs: Functions and construction
   2.3.4 Continuous and Discrete Control
   2.3.5 PLC Programming Languages
      2.3.5.1 IEC 61131 Languages
      2.3.5.2 Function blocks language
      2.3.5.3 Program Execution
      2.3.5.4 Input / Output
      2.3.5.5 Structured Text
      2.3.5.6 Sequential Function Charts
      2.3.5.7 Ladder Logic
      2.3.5.8 Instruction Lists
      2.3.5.9 Programming environment
Function Block Languages

(Funktionsblocksprache, langage de blocs de fonctions)
(Also called "Function Chart" or "Function Plan" - FuPla)

The function block languages express "combinatorial" programs in a way similar to electronic circuits. They draw on a large variety of predefined and custom functions.

This language is similar to the Matlab / Simulink language used in simulations.
Function blocks is a graphical programming language, which is akin to the electrical and block diagrams of the analog and digital technique.

It mostly expresses combinatorial logic, but its blocks may have a memory (e.g. flip-flops).
The block is defined by its:
- Data flow interface (number and type of input/output signals)
- Black-Box-Behavior (functional semantic, e.g. in textual form).

**Signals**
Connections that carry a pseudo-continuous data flow.
Connects the function blocks.
Function Block Rules

There exist exactly two rules for connecting function blocks by signals (this is the actual programming):

- Each signal is connected to exactly one source. This source can be the output of a function block or a plant signal.
- The type of the output pin, the type of the input pin and the signal type must be identical.

For convenience, the function plan should be drawn so the signals flow from left to right and from top to bottom. Some editors impose additional rules.

Retroactions are exception to this rule. In this case, the signal direction is identified by an arrow. (Some editors forbid retroactions - use duplicates instead).
Types of Programming Organisation Units (POUs)

1) “Functions”  
   - are part of the base library.  
   - have **no memory**.  
   Example are: adder, multiplier, selector,....

2) “Elementary Function Blocks” (EFB)  
   - are part of the base library  
   - have a **memory** ("static" data).  
   - may access global variables (side-effects !)  
   Examples: counter, filter, integrator,.....

3) “Programs” (Compound blocks)  
   - user-defined or application-specific blocks  
   - may **have a memory**  
   - may be configurable (control flow not visible in the FBD  
   Examples: PID controller, Overcurrent protection, Motor sequence  
   (a library of compound blocks may be found in IEC 61804-1)
Function Block library

The programmer chooses the blocks in a block library, similarly to the hardware engineer who chooses integrated circuits out of the catalogue.

This library indicates the pinning of each block, its semantics and the execution time.

The programmer may extend the library by defining function block macros out of library elements.

If some blocks are often used, they will be programmed in an external language (e.g. “C”, micro-code) following strict rules.
The number of inputs or outputs and their type is restricted. The execution time of each block depends on the block type, the number of inputs and on the processor.
Exercise: Tooth saw generator

exercise: build a tooth-saw (asymmetric) generator with the IEC 61131 elements of the preceding page
Library functions for discrete plants

Basic blocks
- logical combinations (AND, OR, NOT, EXOR)
- Flip-flop
- Selector m-out-of-n
- Multiplexer m-to-n
- Timer
- Counter
- Memory
- Sequencing

Compound blocks
- Display
- Manual input, touch-screen
- Safety blocks (interlocking)
- Alarm signaling
- Logging
Analog function blocks for continuous control

Basic blocks
- Summator / Subtractor
- Multiplier / Divider
- Integrator / Differentiator
- Filter
- Minimal value, Maximum value
- Radix
- Function generator

Regulation Functions
- P, PI, PID, PDT2 controller
- Fixed set-point
- Ratio and multi-component regulation
- Parameter variation / setting
- 2-point regulation
- 3-point regulation
- Output value limitation
- Ramp generator
- Adaptive regulation
- Drive Control
Function Block library for specialized applications

MoveAbsolute

<table>
<thead>
<tr>
<th>Axis</th>
<th>Execute</th>
<th>Position</th>
<th>Velocity</th>
<th>Acceleration</th>
<th>Deceleration</th>
<th>Jerk</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>CommandAborted</td>
<td>Error</td>
<td>ErrorID</td>
<td>WORD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MC_Direction</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example: FB for motion control
Specifying the behaviour of Function Block

Time Diagram:

Truth Table:

Mathematical Formula:

Textual Description: Calculates the root mean square of the input with a filtering constant 

Equal
Function Block specification in Structured Text

```plaintext
FUNCTION_BLOCK HYSTERESIS
  VAR_INPUT
    XIN1, XIN2 : REAL;
    EPS : REAL; (* Hysteresis:
  END_VAR
  VAR_OUTPUT
    Q : BOOL := 0
  END_VAR
  IF Q THEN
    IF XIN1 < (XIN2-EPS) THEN
      Q := 0 (* XIN1 decreases:
    END_IF;
    ELSIF XIN1 > (XIN2 + EPS) THEN
      Q := 1; (* XIN1 increases:
    END_IF;
  END_FUNCTION_BLOCK
```
Function Block decomposition

A function block describes a *data flow interface*. Its *body* can be implemented differently:

**Elementary block**

The body is implemented in an *external language* (micro-code, assembler, java, IEC 61131 ST):

```plaintext
procedure xy(a,b:BOOLEAN; VAR b,c: BOOLEAN);
begin
    ..... 
    end xy;
```

**Compound block**

The body is realized as a *function block program*

Each input (output) pin of the interface is implemented as exactly one input (output) of the function block.

All signals must appear at the interface to guarantee freedom from *side effects.*
Function Block segmentation

An application program is decomposed into segments ("Programs") for easier reading, each segment being represented on one (A4) printed page.

- Within a segment, the connections are represented *graphically*.
- Between the segments, the connections are expressed by *signal names*.

![Segment A](image)

- X1
- M2
- M1
- Y1

![Segment B](image)

- X2
- M1
- X3
- M2
- Y2
2.3.5.3 Program execution

2.1 Instrumentation
2.2 Control
2.3 Programmable Logic Controllers
   2.3.1 PLCs: Definition and Market
   2.3.2 PLCs: Kinds
   2.3.3 PLCs: Functions and construction
   2.3.4 Continuous and Discrete Control
   2.3.5 PLC Programming Languages
      2.3.5.1 IEC 61131 Languages
      2.3.5.2 Function blocks
      2.3.5.3 Program Execution
      2.3.5.4 Input / Output
      2.3.5.5 Structured Text
      2.3.5.6 Sequential Function Charts
      2.3.5.7 Ladder Logic
      2.3.5.8 Instruction Lists
      2.3.5.9 Programming environment
Segment or POU (program organization unit)

Machine Code:

The function blocks are translated to machine language (intermediate code, IL), that is either interpreted or compiled to assembly language.

Blocks are executed in sequence, normally from upper left to lower right.

The sequence is repeated every x ms.
The function blocks are executed cyclically.
• all inputs are read from memory or from the plant (possibly cached)
• the segment is executed
• the results are written into memory or to the plant (possibly to a cache)

The order of execution of the blocks generally does not matter.
To speed up algorithms and avoid cascading, it is helpful to impose an execution order to the blocks.

The different segments may be assigned a different individual period.
Function blocks are particularly well suited for true multiprocessing (parallel processors).

The performance limit is given by the needed exchange of signals by means of a shared memories.

Semaphores are not used since they could block an execution and make the concerned processes non-deterministic.
Program configuration

The programmer divides the program into tasks (sometimes called pages or segments), which may be executed each with a different period.

The programmer assigns each task (each page) an execution period.

Since the execution time of each block in a task is fixed, the execution time is fixed.

Event-driven operations are encapsulated into blocks, e.g. for transmitting messages.

If the execution time of these tasks cannot be bound, they are executed in background.

The periodic execution always has the highest priority.
IEC 61131 - Execution engine

- configuration
- resource
  - task
  - program
    - FB
- task
  - program
    - FB
- resource
  - task
  - program
    - FB

- access paths
- global and directly represented variables
- communication function

legend:
- execution control path
- variable access paths
- function block
- variable
2.3.5.4 Input and Output
Connecting to Input/Output, Method 1: dedicated I/O blocks

The Inputs and Outputs of the PLC must be connected to (typed) variables

The I/O blocks are configured to be attached to the corresponding I/O groups.
Connecting to Input / Output, Method 2: Variables configuration

All program variables must be declared with name and type, initial value and volatility. A variable may be connected to an input or an output, giving it an I/O address. Several properties can be set: default value, fall-back value, store at power fail,… These variables may not be connected as input, resp. output to a function block.

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Attributes</th>
<th>Initial Value</th>
<th>I/O Address</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>HK_Float</td>
<td>real</td>
<td>retain</td>
<td>0.0</td>
<td>Controller_1.0.11.3.1</td>
<td></td>
</tr>
<tr>
<td>HK_Min</td>
<td>real</td>
<td>retain</td>
<td>-100.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HK_Max</td>
<td>real</td>
<td>retain</td>
<td>+100.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HK_OnOff</td>
<td>bool</td>
<td>retain</td>
<td></td>
<td>Controller_1.0.11.1</td>
<td></td>
</tr>
<tr>
<td>DZ_Input</td>
<td>bool</td>
<td>retain</td>
<td></td>
<td>Controller_1.0.11.2</td>
<td></td>
</tr>
<tr>
<td>HK_Dinput1</td>
<td>bool</td>
<td>retain</td>
<td></td>
<td>Controller_1.0.11.2.1</td>
<td></td>
</tr>
<tr>
<td>HK_Dinput2</td>
<td>bool</td>
<td>retain</td>
<td></td>
<td>Controller_1.0.11.2.2</td>
<td></td>
</tr>
<tr>
<td>HK_Dinput3</td>
<td>bool</td>
<td>retain</td>
<td></td>
<td>Controller_1.0.11.2.3</td>
<td></td>
</tr>
<tr>
<td>HK_Dinput4</td>
<td>bool</td>
<td>retain</td>
<td></td>
<td>Controller_1.0.11.2.4</td>
<td></td>
</tr>
<tr>
<td>HK_Dinput5</td>
<td>bool</td>
<td>retain</td>
<td></td>
<td>Controller_1.0.11.2.5</td>
<td></td>
</tr>
<tr>
<td>HK_Dinput6</td>
<td>bool</td>
<td>retain</td>
<td></td>
<td>Controller_1.0.11.2.6</td>
<td></td>
</tr>
<tr>
<td>HK_Dinput7</td>
<td>bool</td>
<td>retain</td>
<td></td>
<td>Controller_1.0.11.2.7</td>
<td></td>
</tr>
<tr>
<td>HK_Dinput8</td>
<td>bool</td>
<td>retain</td>
<td></td>
<td>Controller_1.0.11.2.8</td>
<td></td>
</tr>
<tr>
<td>Right2LeftCnt</td>
<td>int</td>
<td>retain</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HK_MemTime</td>
<td>time</td>
<td>retain</td>
<td>5s*23ms</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

predefined addresses
2.3.5.5 Structured Text

2.1 Instrumentation
2.2 Control
2.3 Programmable Logic Controllers
   2.3.1 PLCs: Definition and Market
   2.3.2 PLCs: Kinds
   2.3.3 PLCs: Functions and construction
   2.3.4 Continuous and Discrete Control
   2.3.5 PLC Programming Languages
      2.3.5.1 IEC 61131 Languages
      2.3.5.2 Function blocks
      2.3.5.3 Program Execution
      2.3.5.4 Input / Output
      2.3.5.5 Structured Text
      2.3.5.6 Sequential Function Charts
      2.3.5.7 Ladder Logic
      2.3.5.8 Programming environment
Structured Text

*(Strukturierte Textsprache, langage littéral structuré)*

Structured Text is a language similar to Pascal (If, While, etc.).

The variables defined in ST can be used in other languages.

It is used to do complex data manipulation and write blocs.

Caution: writing programs in structured text can breach the real-time rules!
Data Types

Since Function Blocks are typed, the types of connection, input and output must match.

• Elementary Types are defined either in Structured Text or in the FB configuration.

<table>
<thead>
<tr>
<th>Binary Types</th>
<th>Analog Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>REAL</td>
</tr>
<tr>
<td>BYTE</td>
<td>(Real32)</td>
</tr>
<tr>
<td>WORD</td>
<td>LREAL</td>
</tr>
<tr>
<td>DWORD</td>
<td>(Real64)</td>
</tr>
</tbody>
</table>

• Derived Types are user-defined and must be declared in **Structured Text**
  subrange,
  enumerated,
  arrays,
  structured types
  (e.g. AntivalentBoolean2)

variable can receive initial values and be declared as non-volatile (RETAIN)
### 61131 Elementary Data Types

<table>
<thead>
<tr>
<th>No.</th>
<th>Keyword</th>
<th>Data Type</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BOOL</td>
<td>Boolean</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>SINT</td>
<td>Short integer</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>INT</td>
<td>Integer</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>DINT</td>
<td>Double integer</td>
<td>32</td>
</tr>
<tr>
<td>5</td>
<td>LINT</td>
<td>Long integer</td>
<td>64</td>
</tr>
<tr>
<td>6</td>
<td>USINT</td>
<td>Unsigned short integer</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>UINT</td>
<td>Unsigned integer</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td>UDINT</td>
<td>Unsigned double integer</td>
<td>32</td>
</tr>
<tr>
<td>9</td>
<td>UUINT</td>
<td>Unsigned long integer</td>
<td>64</td>
</tr>
<tr>
<td>10</td>
<td>REAL</td>
<td>Real numbers</td>
<td>32</td>
</tr>
<tr>
<td>11</td>
<td>LREAL</td>
<td>Long reals</td>
<td>64</td>
</tr>
<tr>
<td>12</td>
<td>TIME</td>
<td>Duration</td>
<td>depends</td>
</tr>
<tr>
<td>13</td>
<td>DATE</td>
<td>Date (only)</td>
<td>depends</td>
</tr>
<tr>
<td>14</td>
<td>TIME_OF_DAY or TOD</td>
<td>Time of day (only)</td>
<td>depends</td>
</tr>
<tr>
<td>15</td>
<td>DATE_AND_TIME or DT</td>
<td>Date and time of day</td>
<td>depends</td>
</tr>
<tr>
<td>16</td>
<td>STRING</td>
<td>Character string</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>BYTE</td>
<td>Bit string of length 8</td>
<td>8</td>
</tr>
<tr>
<td>18</td>
<td>WORD</td>
<td>Bit string of length 16</td>
<td>16</td>
</tr>
<tr>
<td>19</td>
<td>DWORD</td>
<td>Bit string of length 32</td>
<td>32</td>
</tr>
<tr>
<td>20</td>
<td>LWORD</td>
<td>Bit string of length 64</td>
<td>64</td>
</tr>
<tr>
<td>21</td>
<td>lword</td>
<td>variable length double-byte string</td>
<td></td>
</tr>
</tbody>
</table>
Example of Derived Types

```
TYPE
  ANALOG_CHANNEL_CONFIGURATION
  STRUCT
    RANGE: ANALOG_SIGNAL_RANGE;
    MIN_SCALE : ANALOG_DATA ;
    MAX_SCALE : ANALOG_DATA ;
  END_STRUCT;
  ANALOG_16_INPUT_CONFIGURATION :
  STRUCT
    SIGNAL_TYPE : ANALOG_SIGNAL_TYPE;
    FILTER_CHARACTERISTIC : SINT (0.99)
    CHANNEL: ARRAY [1..16] OF ANALOG_CHANNEL_CONFIGURATION;
  END_STRUCT ;
END_TYPE
```
2.3.5.6 Sequential Function Charts
SFC (Sequential Flow Chart)

(Ablaufdiagramme, diagrammes de flux en séquence - grafcet)

SFC describes sequences of operations and interactions between parallel processes. It is derived from the languages Grafcet and SDL (used for communication protocols), its mathematical foundation lies in Petri Nets.
The sequential program consists of states connected by transitions. A state is activated by the presence of a token (the corresponding variable becomes TRUE). The token leaves the state when the transition condition (event) on the state output is true. Only one transition takes place at a time.

The execution period is a configuration parameter (task to which this program is attached).

**example:** Sc is true, S0, Sa, Sb are false

\[ Ec = ((\text{varX} \& \text{varY}) \mid \text{varZ}) \]

The rule is:

- There is always a transition between two states.
- There is always a state between two transitions.
SFC: Initial state

State which come into existence with a token are called *initial states*.

All initial states receive exactly one token, the other states receive none.

Initialization takes place explicitly at start-up.

In some systems, initialization may be triggered in a user program (initialization pin in a function block).
token switch: the token crosses the first active transition (at random if both Ea and Eb are true)
Note: transitions are after the alternance

token forking: when the transition Ee is true, the token is replicated to all connected states
Note: transition is before the fork

token join: when all connected states have tokens and transition Eg is true, one single token is forwarded.
Note: transition is after the join
P1 (pulse raise) action is executed once when the state is entered.
P0 (pulse fall) action is executed once when the state is left.
N (non-stored) action is executed continuously while the token is in the state.

P1 and P0 actions could be replaced by additional states.

The actions are described by a code block written e.g. in Structured Text.
Special action: the timer

rather than define a P0 action “reset timer…”, there is an implicit variable defined as State.t that express the time spent in that state.

\[ S.t > t\#5s \]
The input and output flow of a state are always in the same vertical line (simplifies structure)

Alternative paths are drawn such that no path is placed in the vertical flow (otherwise would mean this is a preferential path)

Priority:
- The alternative path most to the left has the highest priority, priority decreases towards the right.
- Loop: exit has a higher priority than loopback.
SFC: Exercise

Variables:
Input: In0, In1, In2, In3;
Output:
   Trap = {0: closed; 1: open}
   Speed = {+20: +1 m/s; +1: +5 cm/s}
   Register = {0: closed; 1: open}
negative values: opposite direction

Initially: let vehicle until it touches I0 at reduced speed and open the trap for 5s (empty the vehicle).

1 - Let the vehicle move from I0 to I3
2 - Stop the vehicle when it reaches I3.
3 - Open the tank during 5s.
4 - Go back to I0
5 - Open the trap and wait 5s.
   repeat above steps indefinitely

\[\text{Generates "1" as long as the tag of the vehicle (1cm) is over the sensor.}\]
SFC: Building subprograms

T-element

transition  T-sequence  alternative paths

S-element

state  S-sequence  parallel paths  loop

The meta-symbols T and S define structures - they may not appear as elements in the flow chart.

A flow chart may only contain the terminal symbols: state and transition.
SFC: Structuring

Every flow chart without a token generator may be redrawn as a structured flow chart (by possibly duplicating program parts)

Not structured

structured
SFC: Complex structures

These general rules serve to build networks, termed by DIN and IEC as *flow charts*.

Problems with general networks:
- deadlocks
- uncontrolled token multiplication

Solution:
assistance through the flow chart editor.
Many PLC applications mix continuous and discrete control.
A PLC may execute alternatively function blocks and flow charts.
A communication between these program parts must be possible.

Principle:

The flow chart taken as a whole can be considered a function block with binary inputs (transitions) and binary outputs (states).
Executing Flow Charts As blocks

A function block may be implemented in three different ways:

procedure
xy(...);
begin
...
end xy;

extern (ST)

diagram

function blocks

flow chart

Function blocks and flow chart communicate over binary signals.
Flow Charts or Function Blocs?

A task can sometimes be written indifferently as function blocs or as flow chart. The application may decide which representation is more appropriate:
Flow Charts Or Blocks? (2)

In this example, flow chart seems to be more appropriate:
2.3.5.7 Ladder Logic

2.1 Instrumentation
2.2 Control
2.3 Programmable Logic Controllers
  2.3.1 PLCs: Definition and Market
  2.3.2 PLCs: Kinds
  2.3.3 PLCs: Functions and construction
  2.3.4 Continuous and Discrete Control
  2.3.5 PLC Programming Languages
    2.3.5.1 IEC 61131 Languages
    2.3.5.2 Function blocks
    2.3.5.3 Program Execution
    2.3.5.4 Input / Output
    2.3.5.5 Structured Text
    2.3.5.6 Sequential Function Charts
    2.3.5.7 Ladder Logic
    2.3.5.8 Programming environment
Ladder logic (1)

*(Kontaktplansprache, langage à contacts)*

The ladder logic is the oldest programming language for PLC it bases directly on the relay intuition of the electricians. it is widely in use outside Europe. It is described here but not recommended for new projects.
Ladder Logic (2)

origin: electrical circuit

corresponding ladder diagram

"coil" 50 is used to move other contact(s)
Ladder logic (3)

The contact plan or "ladder logic" language allows an easy transition from the traditional relay logic diagrams to the programming of binary functions.

It is well suited to express combinational logic.

It is not suited for process control programming (there are no analog elements).

The main ladder logic symbols represent the elements:

- **make contact**
- **break contact**
- **relay coil**

contact travail  Arbeitskontakt
contact repos  Ruhekontakt
bobine  Spule
Ladder logic (4)

Binary combinations are expressed by series and parallel relay contact:

Series

Coil 50 is active (current flows) when 01 is active and 02 is not.

Parallel

Coil 40 is active (current flows) when 01 is active or 02 is not.
Ladder logic (5)

The ladder logic is more intuitive for complex binary expressions than literal languages.

```
!N 1 & 2 STR 3 & N 4 STR N 5 & 6 / STR & STR = 50
```

```
!0 & 1 STR 2 & 3 / STR STR 4 & 5 STR N 6 & 7 / STR & STR STR 10 & 11 / STR & 12 = 50
```
Ladder logic (6)

Ladder logic stems from the time of the relay technology. As PLCs replaced relays, their new possibilities could not be expressed any more in relay terms.

The contact plan language was extended to express functions:

![Diagram of ladder logic]

\[
\text{literal expression: } \neg 00 \land 01 \text{ FUN 02 } = 200
\]

The intuition of contacts and coil gets lost.

The introduction of «functions» that influence the control flow itself, is problematic.

The contact plan is - mathematically - a functional representation.

The introduction of a more or less hidden control of the flow destroys the freedom of side effects and makes programs difficult to read.
Ladder logic (7)

Ladder logic provides neither:
• sub-programs (blocks), nor
• data encapsulation nor
• structured data types.

It is not suited to make reusable modules.

IEC 61131 does not prescribe the minimum requirements for a compiler / interpreter such as number of rungs per page nor does it specifies the minimum subset to be implemented.

Therefore, it should not be used for large programs made by different persons.

It is very limited when considering analog values (it has only counters)

→ used in manufacturing, not process control
2.3.6 Instruction Lists

2.1 Instrumentation
2.2 Control
2.3 Programmable Logic Controllers
   2.3.1 PLCs: Definition and Market
   2.3.2 PLCs: Kinds
   2.3.3 PLCs: Functions and construction
   2.3.4 Continuous and Discrete Control
   2.3.5 PLC Programming Languages
      2.3.5.1 IEC 61131 Languages
      2.3.5.2 Function blocks
      2.3.5.3 Program Execution
      2.3.5.4 Input / Output
      2.3.5.5 Structured Text
      2.3.5.6 Sequential Function Charts
      2.3.5.7 Ladder Logic
      2.3.5.8 Instructions Lists
      2.3.5.9 Programming environment
Instruction lists is the machine language of PLC programming. It has 21 instructions (see table).

Three modifiers are defined: "N" negates the result, "C" makes it conditional and "(" delays it.

All operations relate to one result register (RR) or accumulator.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Modifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>N</td>
<td>Loads operand in RR</td>
</tr>
<tr>
<td>ST</td>
<td>N</td>
<td>Stores current result from RR</td>
</tr>
<tr>
<td>S</td>
<td></td>
<td>Sets the operand</td>
</tr>
<tr>
<td>R</td>
<td></td>
<td>Resets the operand</td>
</tr>
<tr>
<td>AND</td>
<td>N, (</td>
<td>Boolean AND</td>
</tr>
<tr>
<td>OR</td>
<td>N, (</td>
<td>Boolean OR</td>
</tr>
<tr>
<td>XOR</td>
<td>N, (</td>
<td>Exclusive OR</td>
</tr>
<tr>
<td>ADD</td>
<td>(</td>
<td>Arithmetic addition</td>
</tr>
<tr>
<td>SUB</td>
<td>(</td>
<td>Arithmetic subtraction</td>
</tr>
<tr>
<td>MUL</td>
<td>(</td>
<td>Arithmetic multiplication</td>
</tr>
<tr>
<td>DIV</td>
<td>(</td>
<td>Arithmetic division</td>
</tr>
<tr>
<td>GT</td>
<td>(</td>
<td>Comparison greater than</td>
</tr>
<tr>
<td>GE</td>
<td>(</td>
<td>Comparison greater or equal</td>
</tr>
<tr>
<td>EQ</td>
<td>(</td>
<td>Comparison equal</td>
</tr>
<tr>
<td>LE</td>
<td>(</td>
<td>Comparison less than</td>
</tr>
<tr>
<td>LT</td>
<td>(</td>
<td>Comparison less or equal</td>
</tr>
<tr>
<td>NE</td>
<td>(</td>
<td>Comparison not equal</td>
</tr>
<tr>
<td>)</td>
<td></td>
<td>Executes delayed operation</td>
</tr>
<tr>
<td>CAL</td>
<td>C, N</td>
<td>Calls a function block</td>
</tr>
<tr>
<td>JMP</td>
<td>C, N</td>
<td>Jumps to label</td>
</tr>
<tr>
<td>RET</td>
<td>C, N</td>
<td>Returns from called function</td>
</tr>
</tbody>
</table>
### Instruction Lists Example (2)

<table>
<thead>
<tr>
<th>Label</th>
<th>Operator</th>
<th>Operand</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>temp1</td>
<td></td>
<td>(<em>Load temp1 and</em>)</td>
</tr>
<tr>
<td>GT</td>
<td>temp2</td>
<td></td>
<td>(<em>Test if temp1 &gt; temp2</em>)</td>
</tr>
<tr>
<td>JMP CN</td>
<td>Greater</td>
<td></td>
<td>(<em>Jump if not true to Greater</em>)</td>
</tr>
<tr>
<td>LD</td>
<td>speed1</td>
<td></td>
<td>(<em>Load speed1</em>)</td>
</tr>
<tr>
<td>ADD</td>
<td>200</td>
<td></td>
<td>(<em>Add constant 200</em>)</td>
</tr>
<tr>
<td>JMP</td>
<td>End</td>
<td></td>
<td>(<em>Jump unconditional to End</em>)</td>
</tr>
<tr>
<td>Greater:</td>
<td>LD</td>
<td>speed2</td>
<td>(<em>Load speed2</em>)</td>
</tr>
<tr>
<td>End:</td>
<td>ST</td>
<td>temp3</td>
<td>(* result *)</td>
</tr>
</tbody>
</table>

Instructions Lists is the most efficient way to write code, but only for specialists.

Otherwise, IL should not be used, because this language:
- provides no code structuring
- has weak semantics
- is machine dependent
2.3.5.9 Programming environment

2.1 Instrumentation
2.2 Control
2.3 Programmable Logic Controllers
  2.3.1 PLCs: Definition and Market
  2.3.2 PLCs: Kinds
  2.3.3 PLCs: Functions and construction
  2.3.4 Continuous and Discrete Control
  2.3.5 PLC Programming Languages
    2.3.5.1 IEC 61131 Languages
    2.3.5.2 Function blocks
    2.3.5.3 Program Execution
    2.3.5.4 Input / Output
    2.3.5.5 Structured Text
    2.3.5.6 Sequential Function Charts
    2.3.5.7 Ladder Logic
    2.3.5.8 Instructions Lists
  2.3.5.9 Programming environment
Programming environment capabilities

A PLC programming environment (ABB, Siemens, CoDeSys,...) allows:

- programming of the PLC in one of the IEC 61131 languages
- defining the variables (name and type)
- binding of the variables to the input/output (binary, analog)
- simulating
- downloading to the PLC of programs and firmware
- uploading of the PLC (seldom provided)
- monitoring of the PLC
- documenting and printing.
61131 Programming environment

configuration, editor, compiler, library

symbols

laptop

code

variable monitoring and forcing for debugging

firmware

download

network

PLC

configuration, editor, compiler, library

symbols

laptop

code

variable monitoring and forcing for debugging

firmware

download

network

PLC
The source of the PLC program is generally on the laptop of the technician.

This copy is frequently modified, it is difficult to track the original in a process database, especially if several persons work on the same machine.

Therefore, it would be convenient to be able to reconstruct the source programs out of the PLC's memory (called back-tracking, *Rückdokumentation*, reconstitution).

This supposes that the instruction lists in the PLC can be mapped directly to graphic representations -> set of rules how to display the information.

Names of variables, blocks and comments must be kept in clear text, otherwise the code, although correct, would not be readable.

For cost reasons, this is seldom implemented.
Is IEC 61131 FB an object-oriented language?

Not really: it does not support inheritance.

Blocks are not recursive.

But it supports interface definition (typed signals), instantiation, encapsulation, some form of polymorphism.

Some programming environments offer “control modules” for better object-orientation.
Limitations of IEC 61131

- it is not foreseen to distribute execution of programs over several devices

- event-driven execution is not foreseen. Blocks may be triggered by a Boolean variable, (but this is good so).

- if structured text increases in importance, better constructs are required (object-oriented)
IEC 61499 – Extension to Event-triggered operation

- Function Blocks
- Data and Event Flows
- Distributable among Multiple Devices

A research topic without industry applications until now….
Assessment

Which are programming languages defined in IEC 61131 and for what are they used?
In a function block language, which are the two elements of programming?
How is a PLC program executed and why is it that way?
Draw a ladder diagram and the corresponding function chart.
Draw a sequential chart implementing a 2-bit counter
Program a saw tooth waveform generator with function blocks
How are inputs and outputs to the process treated in a function chart language?
Program a sequencer for a simple chewing-gum coin machine
Program a ramp generator for a ventilator speed control (soft start and stop in 5s)
open V1 until tank’s L1 indicates upper level
open V2 during 25 seconds
open V3 until the tank’s L1 indicate it is void while stirring.
heat mixture during 50 minutes while stirring
empty the reactor while the drying bed in moving