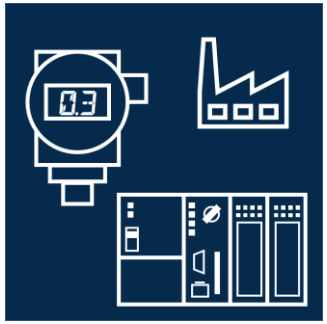
Function Block Diagram

Industrial control

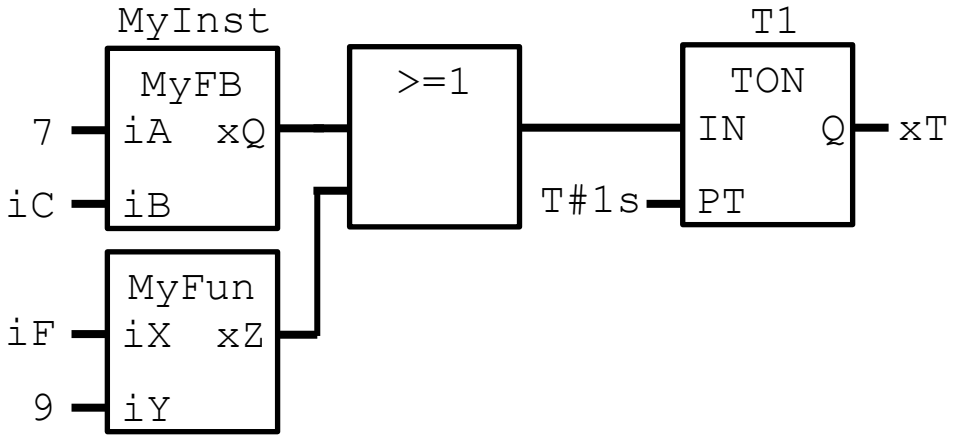
KOVÁCS Gábor

gkovacs@iit.bme.hu



Function Block Diagram (FBD)

- Mid-level graphical language
- Common default programming language of programmable relays and nanoPLCs
- Frequently used in process control (DCS systems)
- Suitable for both Boolean and non-Boolean operations
- Defines data flow between function and FB calls
- Origin: data flow graph
  - blocks: call of functions and FB instances
  - connections: ≈ variables

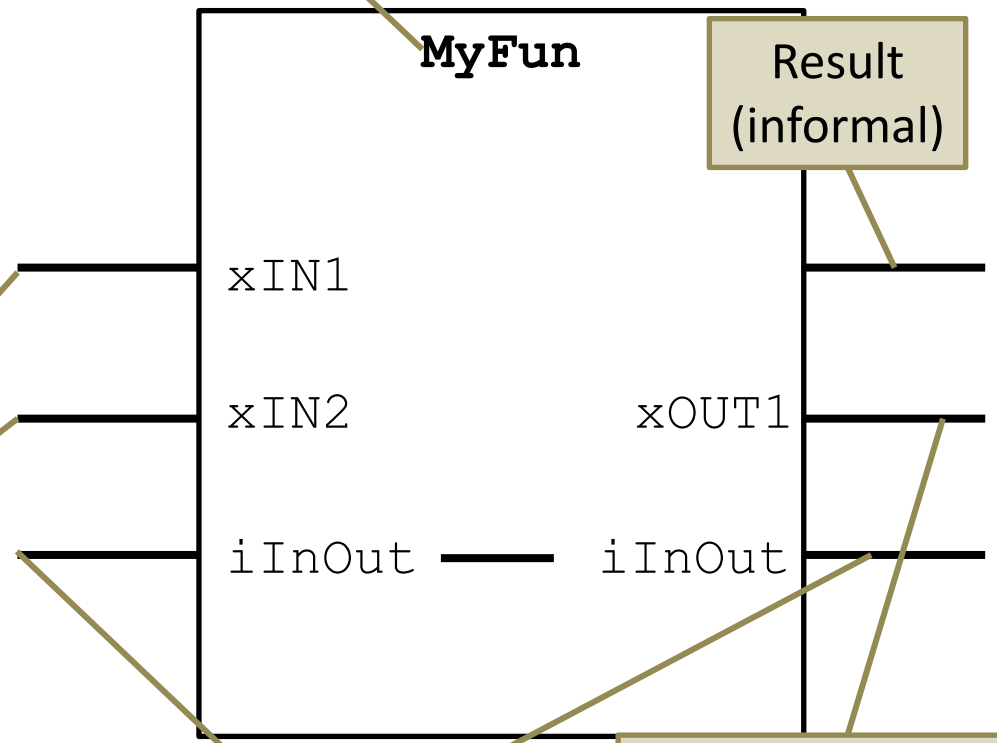


FBD networks

- Organizing unit of FBD language codes
- Elements of networks
  - function and FB calls (blocks)
  - connections (inside a network)
  - execution control elements
  - connectors (connection between networks)
- Networks are evaluated from the top to the bottom
- Each network might have a label assigned
- Labels can be jumped to
- Networks can be connected by connectors

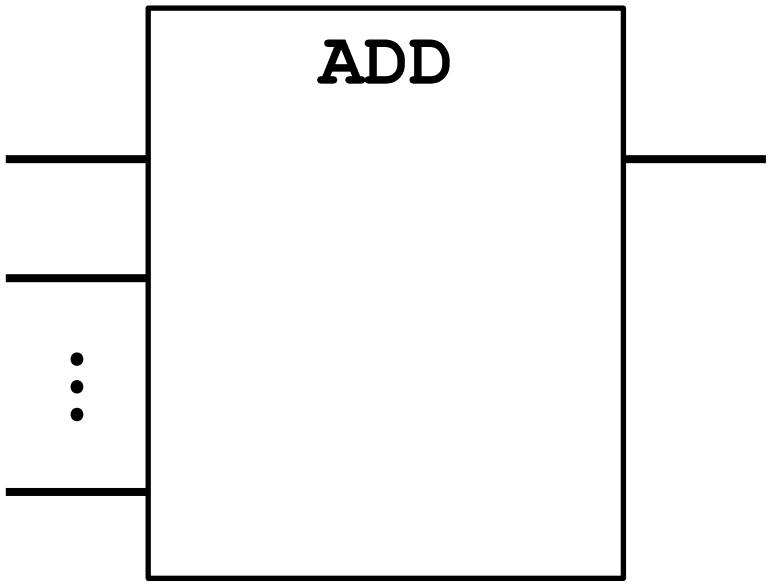
Function call

```
FUNCTION MyFun : INT
VAR_INPUT
    xIn1 : BOOL;
    xIn2 : BOOL;
END_VAR
VAR_OUTPUT
    xOut1 : BOOL;
END_VAR
VAR_IN_OUT
    iInOut : INT;
END_VAR
```



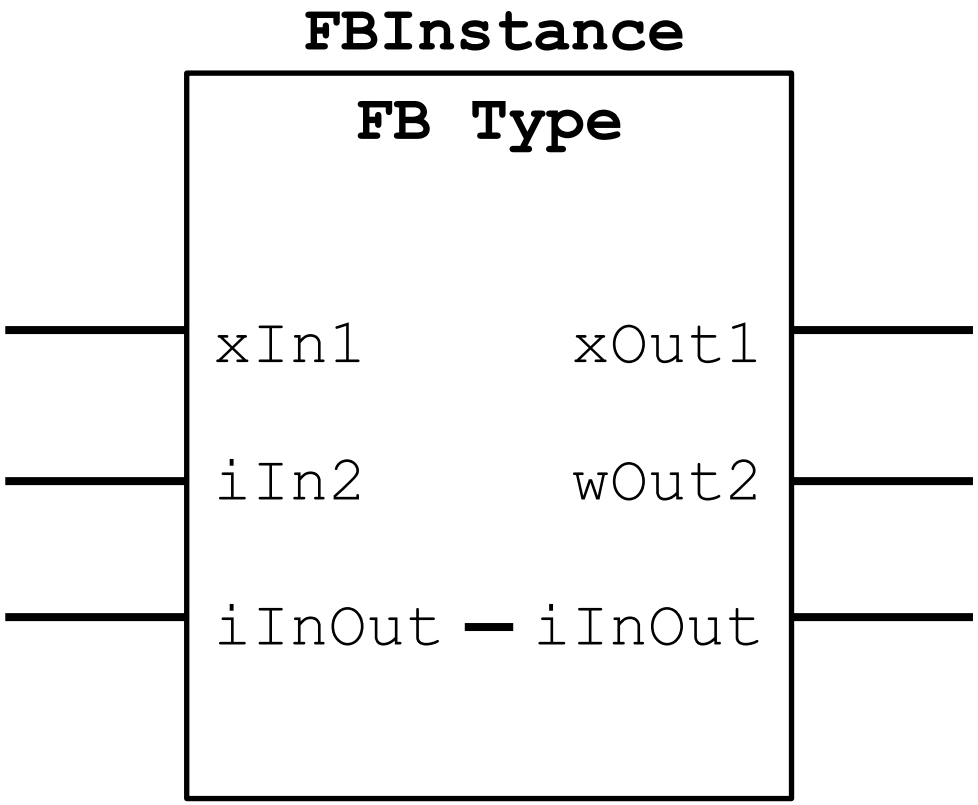
Function call

- Overloaded and extensible functions can be called without formal parameters
  - `ADD` (+), `MUL` (*)
  - `AND` (&), `OR` (>=1) , `XOR` (=2k+1)



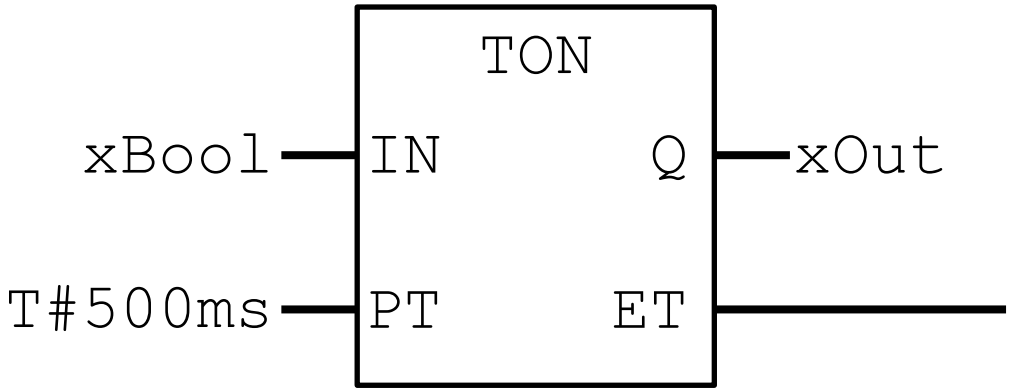
Function block call

All parameters are formal



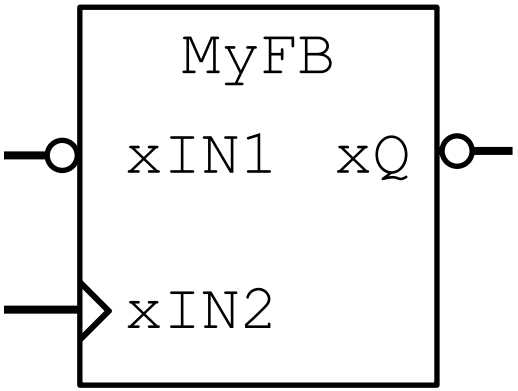
Inputs and outputs of blocks

- Inputs on the left, outputs on the right-hand side
- Inputs
  - variables
  - literals
  - connectors
- Outputs
  - variables
  - connectors
- Not connected inputs and outputs
  - input: default value of data type
  - output: no assignment made



Boolean inputs and outputs

- Negation of input/output : ○
- Edge-sensing inputs:
  - rising edge: >
  - falling edge: <

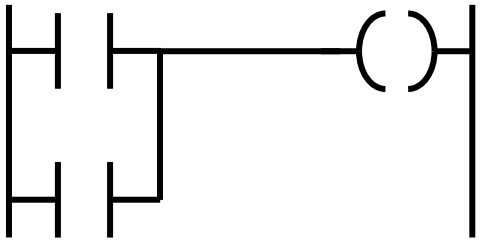


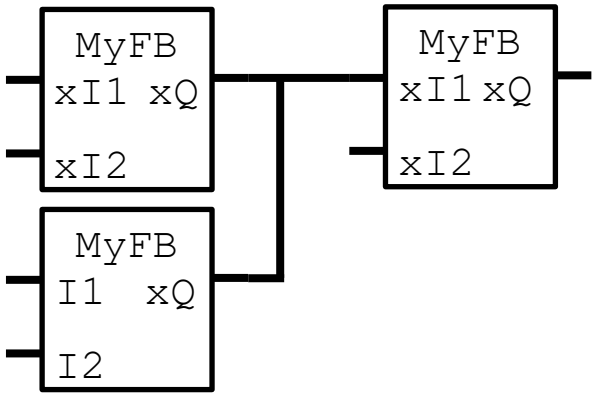
Connections

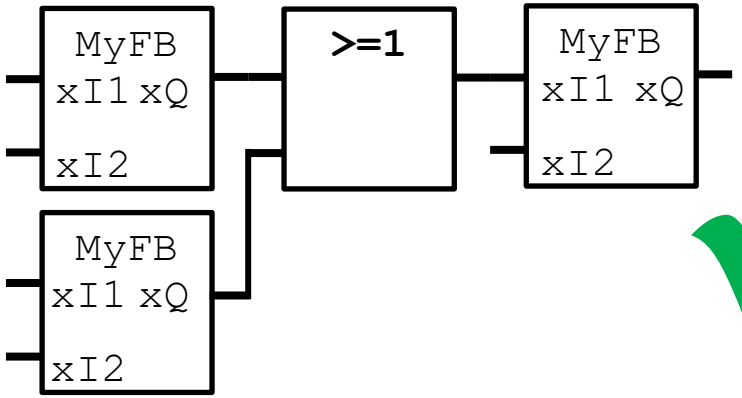
- Horizontal or vertical lines connecting blocks
- **Might represent any data type**
- Only inputs and outputs with compatible data types can be connected
- An output can be connected to multiple inputs
- An input can be connected to one single output only

Connections

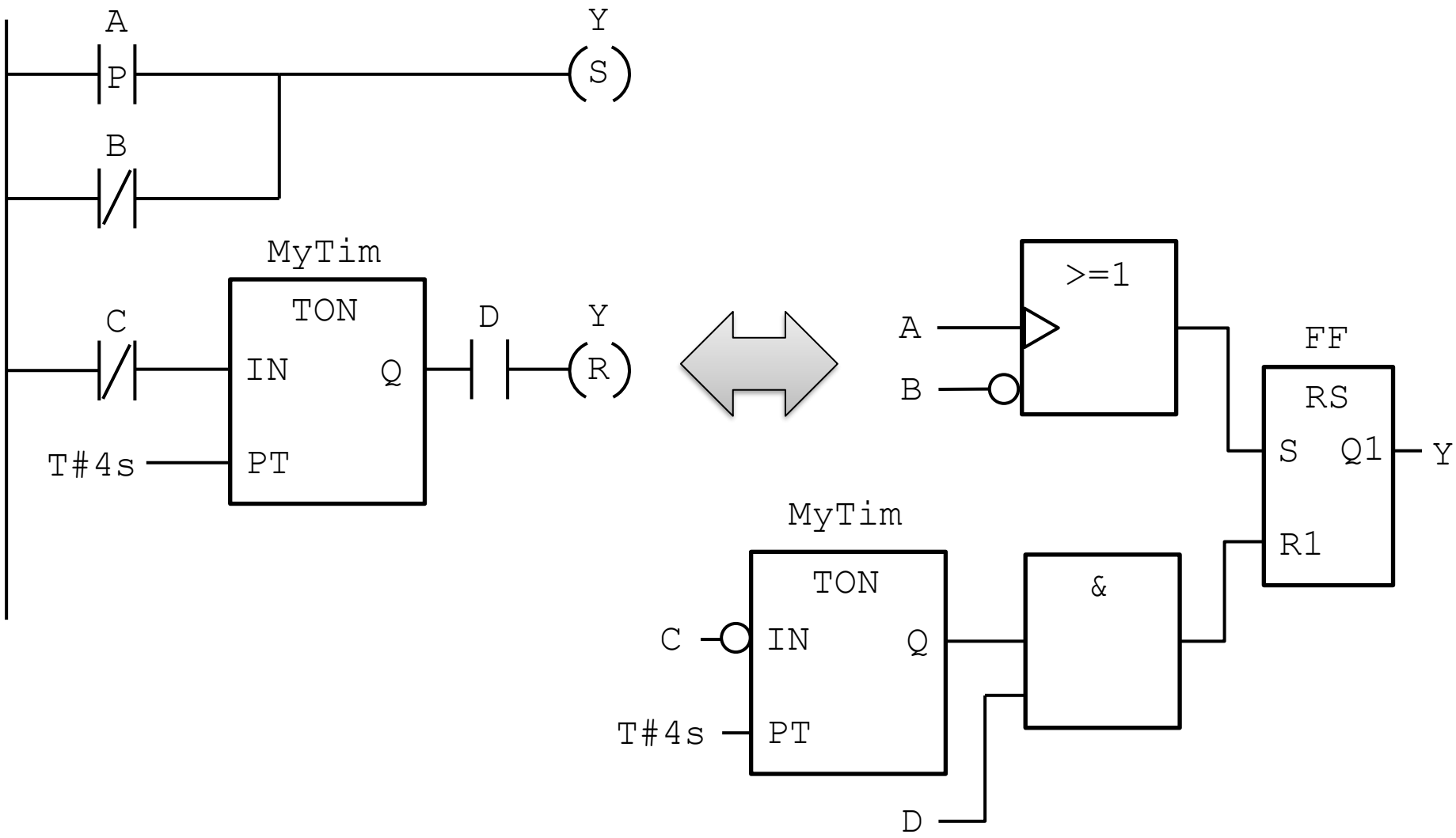
Wired OR is not permitted in FBD







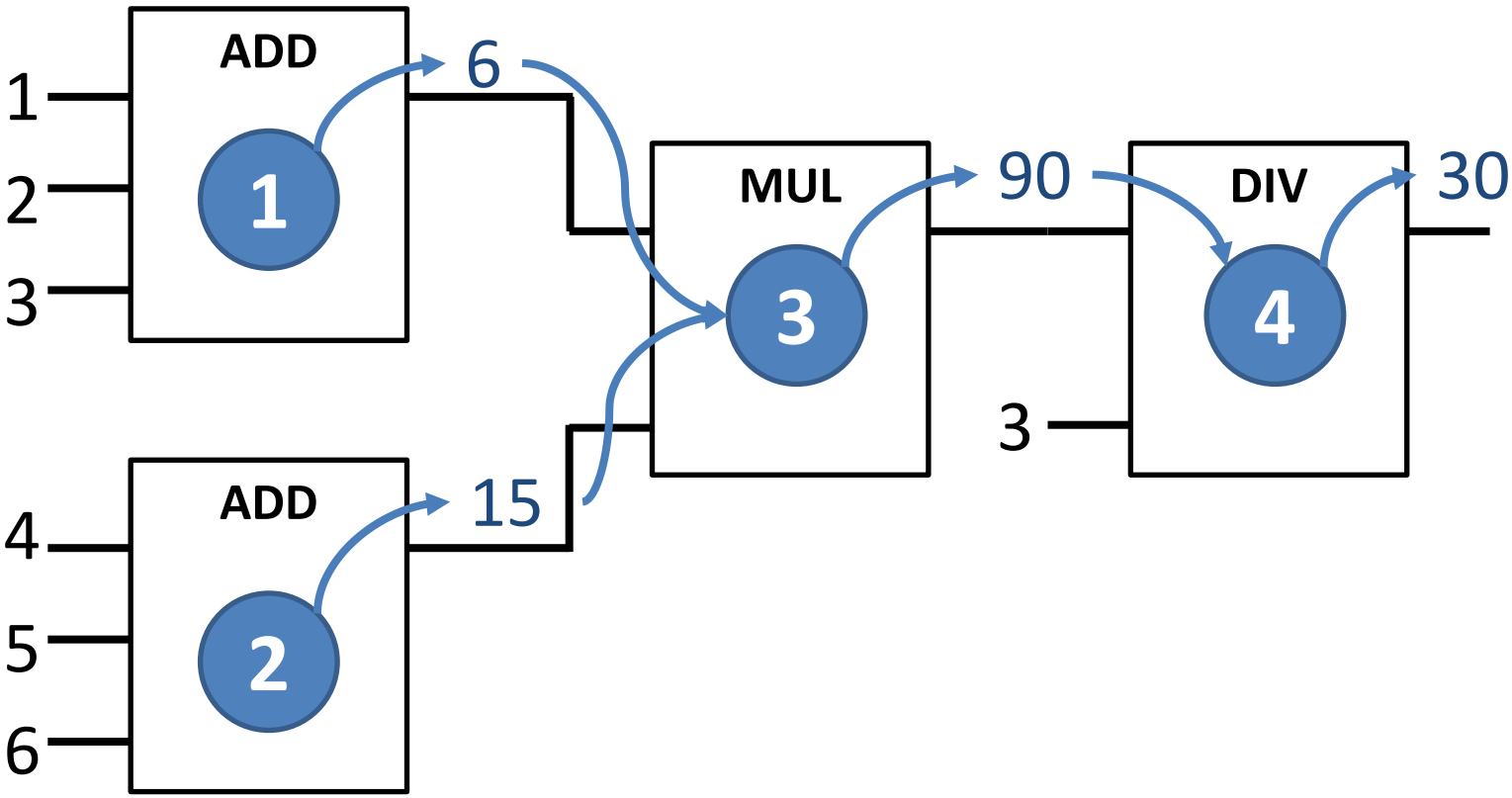
Ladder Diagram vs FBD - example



Evaluation of networks

- No element of a network shall be evaluated until the values of each its inputs have been evaluated
- The evaluation of a network element shall not be complete until the values of each its outputs have been evaluated
- The evaluation of a network is not complete until the outputs of each of its elements have been evaluated (even if it contains execution control elements)

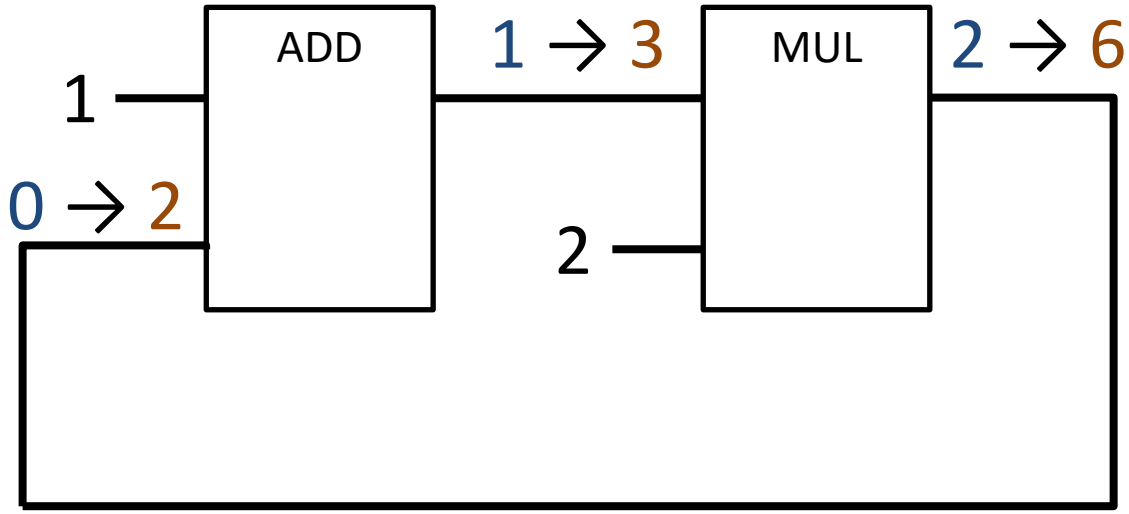
Evaluation of networks - Example



Evaluation order of the two ADD blocks is ambiguous, most development environments use top-to-down (left-to-right) evaluation order

Feedback path

- A feedback path can be defined between different blocks
- Executed once during a cycle (call)
- Output of the last output during the previous call is fed back to the first input
- First input evaluates to the initial value of the data type during the first execution



First execution
Second execution
Literal

Execution control elements

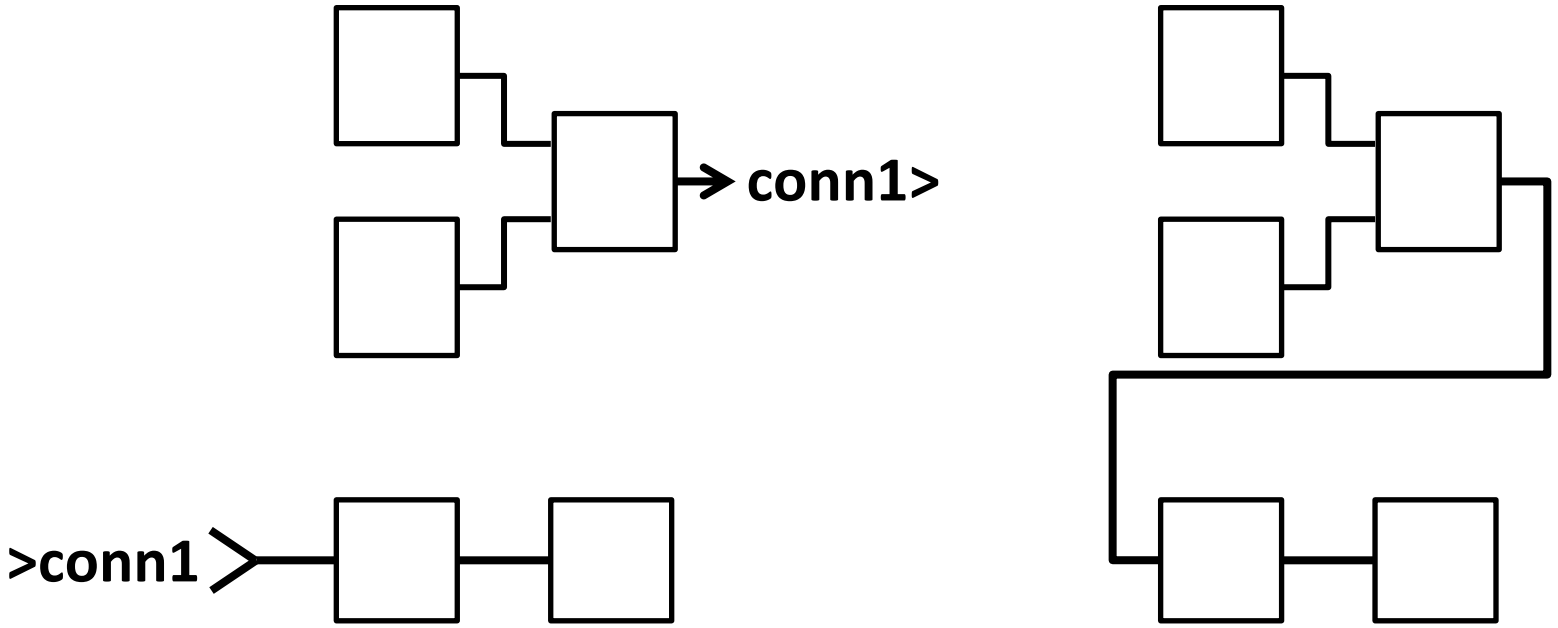
- Jump
  - conditional jump only 

  - Jumps to the given label if the value of its input evaluates to TRUE
- Return 

  - conditional return only
  - returns to calling POU if the value of its input evaluates to true

Connectors

- Connectors can be used to continue the network in a new line
- Useful if maximal width of the diagram is limited by the development environment



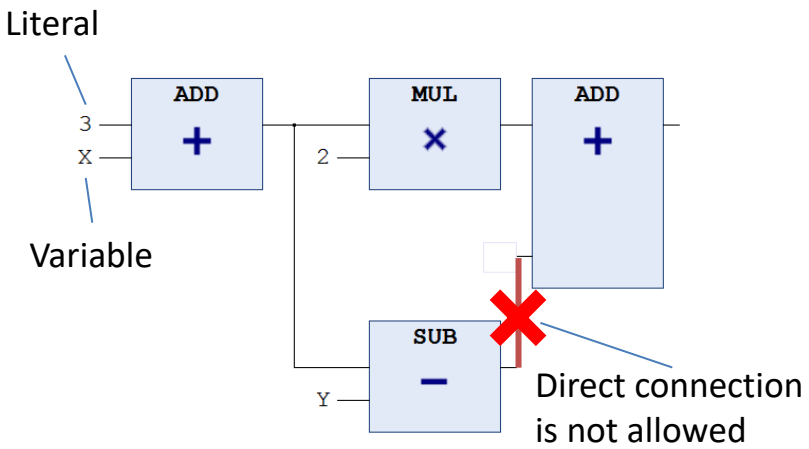
Defining FBD networks

- Lots of blocks, interconnections and feedback paths make the diagram hard to understand
- It is crucial to implement the application in a modular way
  - declare and call functions
  - declare and use function-block types, call the instances

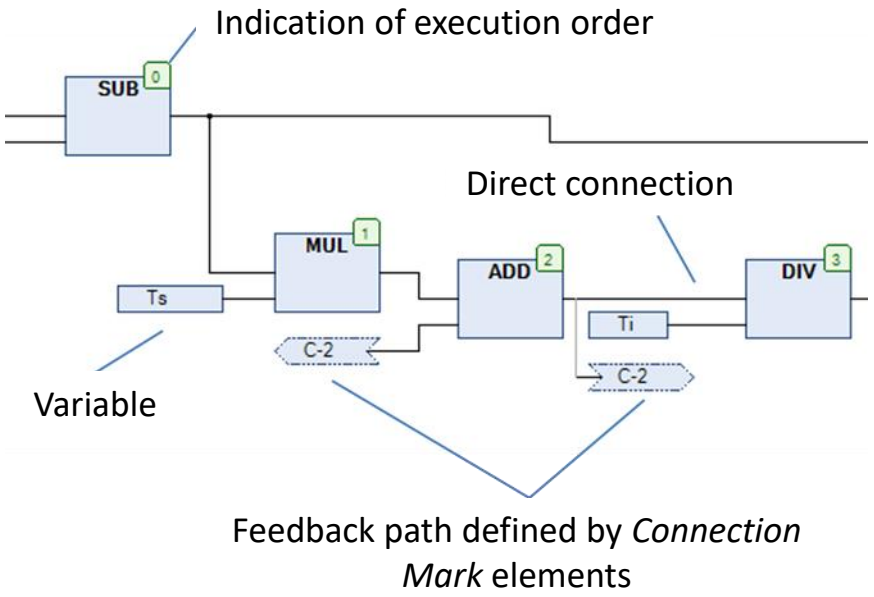
FBD in CODESYS 3 environment

- CODESYS 3 supports implementation of function block diagrams in two forms
- Neither of the two is fully standard-compliant but both provide useful features

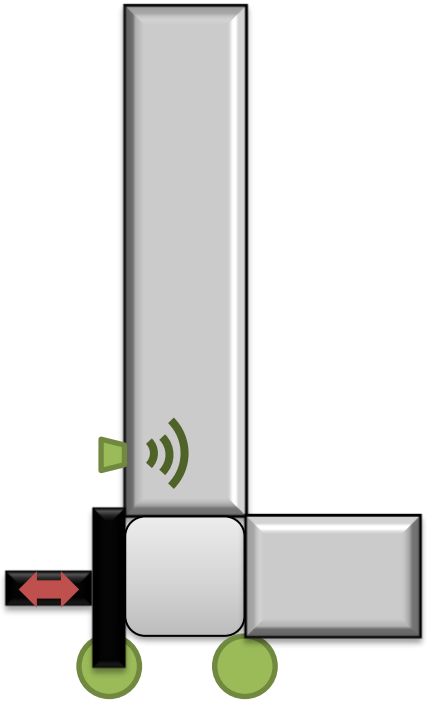
Function Block Diagram (FBD)



Continuous Function Chart (CFC)

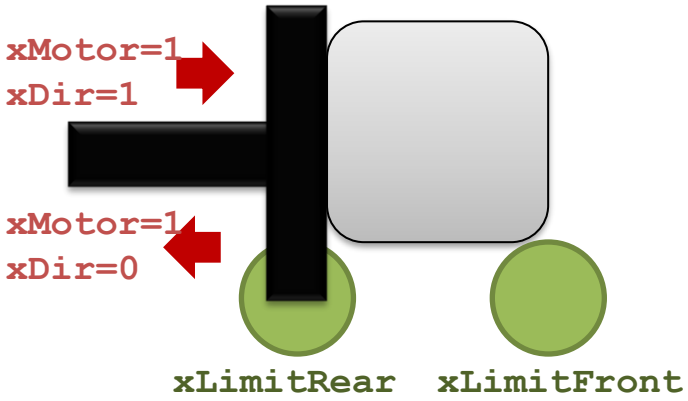


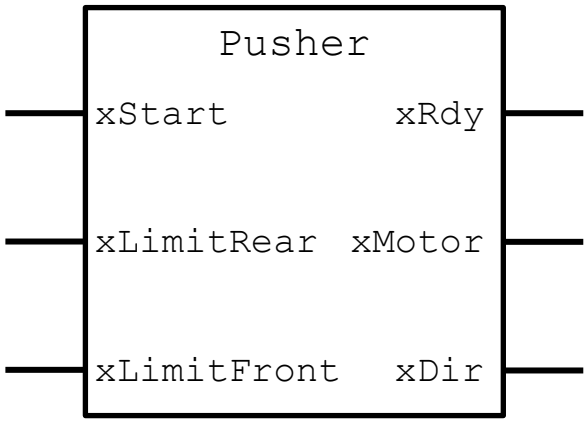
Problem – Pusher and Conveyor



- Packages arrive on a conveyor
- Packages shall be forwarded to a perpendicular chute by a pusher
- While the pusher is moving, the conveyor shall be stopped

Problem – The pusher





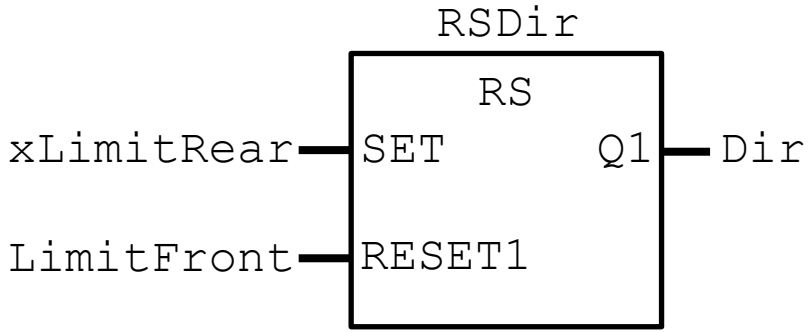
- The pusher waits at its rear position, keeping the `xReady` output active
- The pusher is started from its rear position upon a rising edge of the `xStart` input, then the `xReady` output shall be deactivated
- The pusher shall move forward (`xMotor=1`, `xDir=1`) until the signal of the front limit switch becomes active
- Then the direction shall be changed, and the pusher shall be retracted (`xMotor=1`, `xDir=0`) until it reaches the rear limit switch
- Upon reaching the rear limit switch, the pusher is stopped (`xMotor=0`) and the output `xReady` is activated

Pusher – Declaration part

```
FUNCTION_BLOCK Pusher
VAR_INPUT
  xLimitRear      : BOOL;
  xLimitFront     : BOOL;
  xStart          : BOOL;
END_VAR
VAR_OUTPUT
  xRdy            : BOOL;
  xMotor          : BOOL;
  xDir            : BOOL;
END_VAR
```

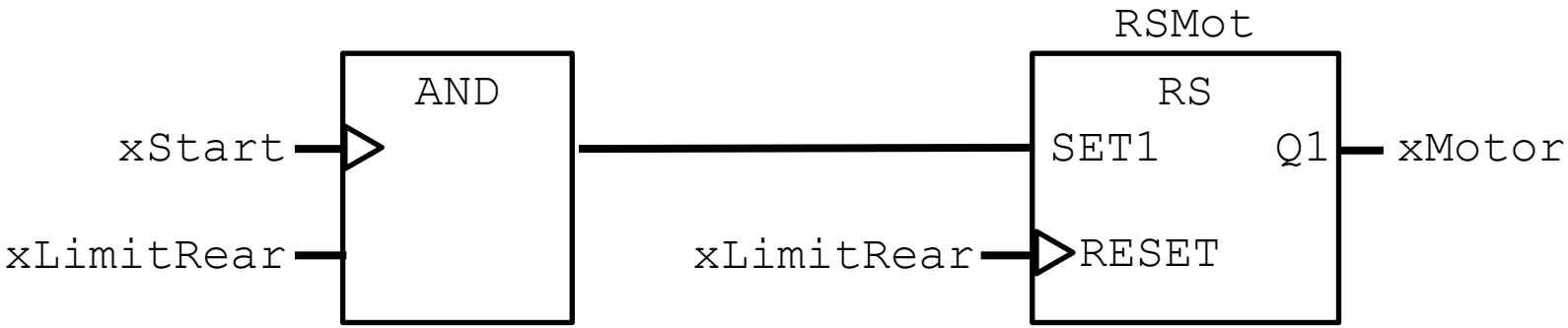
```
VAR
  RSDir: RS;
  SRMot: SR;
END_VAR
```

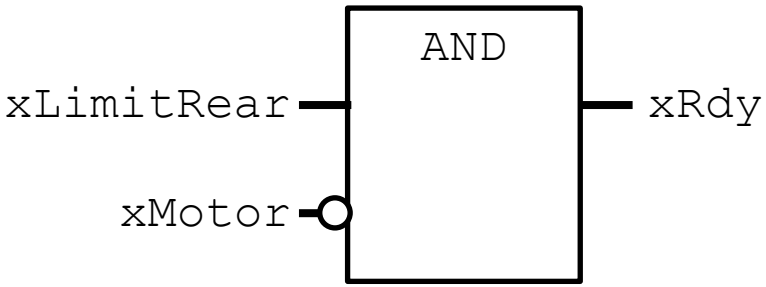
Pusher - Body



Direction is set to 1 (extension) if the rear limit switch and reset upon reaching the front limit switch

The motor is started by the rising edge of `xStart` if the pusher is in its rear position. The motor is turned off when returning to the rear position (rising edge of the limit switch).

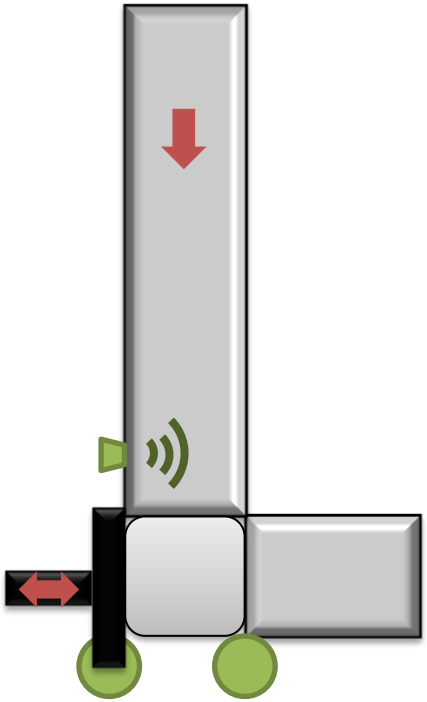




The `Ready` output is active if the pusher is steady in its rear position(`xMotor` input is negated)

Body of the FB consists of three FBD networks

Problem - Conveyor

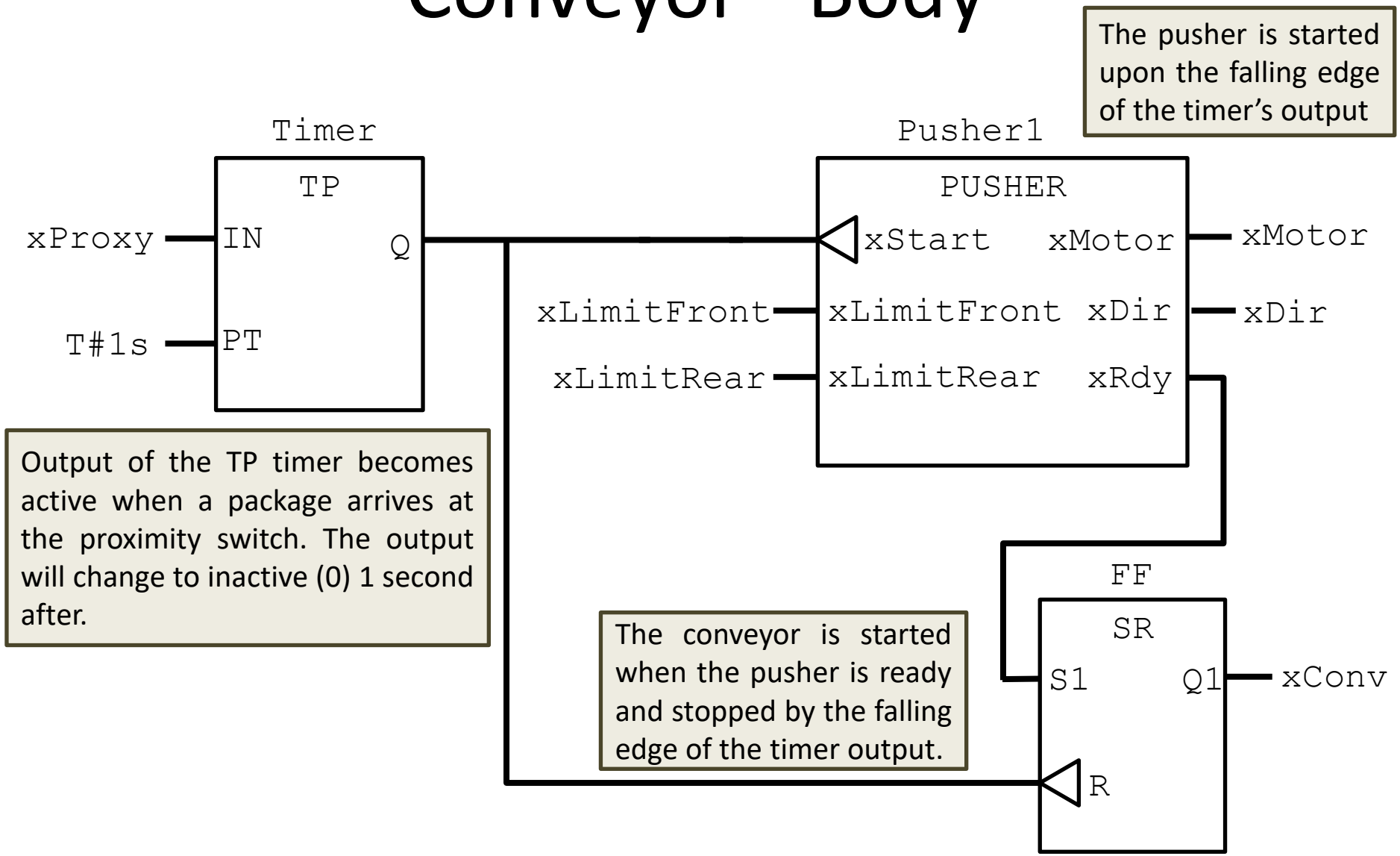


- The conveyor shall be turned on when starting the application
- If the signal of the proximity switch becomes active, then the conveyor shall be kept running for 1 second, then it shall be stopped, and the pusher shall be started
- Upon the pusher returns to its rear position, the conveyor shall be restarted

Conveyor – Declaration part

```
Program PusherConveyor
VAR_INPUT
  xLimitRear : BOOL AT %I0.0; // Rear limit switch of the pusher
  xLimitFront: BOOL AT %I0.1; // Front limit switch of the pusher
  xProxy     : BOOL AT %I0.2; // Proximity switch near the conveyor
END_VAR
VAR_OUTPUT
  xMotor     : BOOL AT %Q0.0; // Pusher motor on/off
  xDir       : BOOL AT %Q0.1; // Direction of pusher
  xConv      : BOOL AT %Q0.2; // Conveyor on/off
END_VAR
VAR
  Timer      : TP;
  Pusher1    : Pusher;
  FF         : SR;
END_VAR
```

Conveyor - Body



Implementation in the template project - FBD

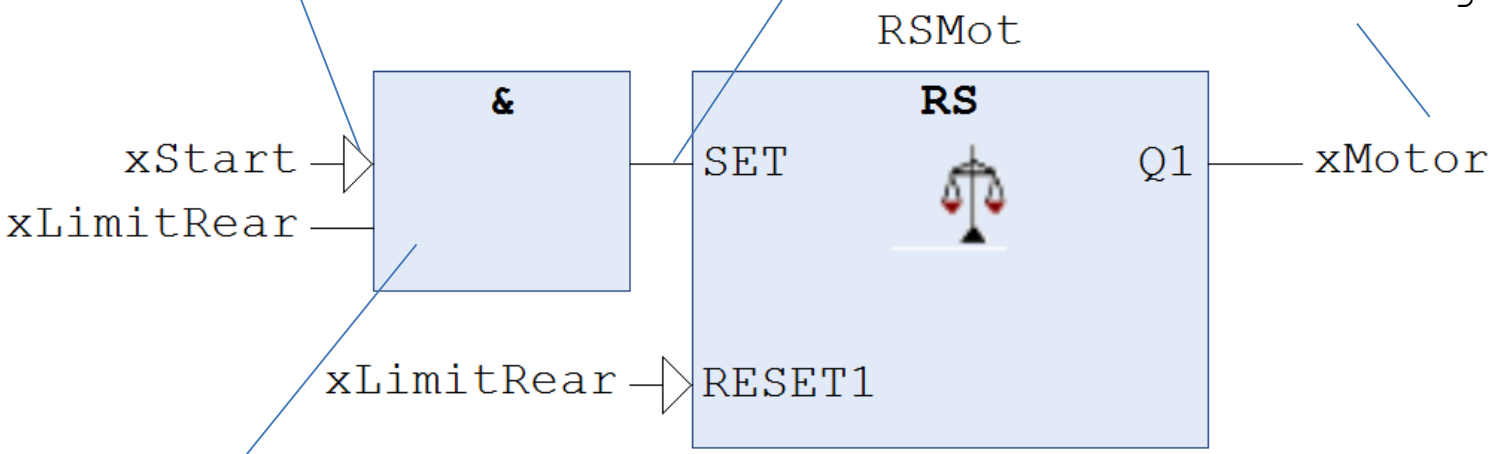
- Diagrams with mostly Boolean elements can be easily implemented using Function Block Diagram (FBD) language
- The diagram is network-based
- Networks are evaluated from top to bottom, elements in networks are evaluated from left to right
- Location of blocks is fixed; branches can only be opened but not closed
- Outputs of blocks following a branch can not be connected to inputs in other branches (data can be passed using auxiliary variables)
- Direct feedback paths can not be implemented (feedback is possible using auxiliary variables)
- The FBD editor is the same as the ladder diagram editor, however, ladder elements (contacts, coils) can not be used

Implementation in the template project - FBD

Negation or setting edge-sensing: right-click on the input and select `Assignment` or `Edge Detection` in the context menu

Connect another block to the output: drag a `Box` to the output

Assign the output of a block to a variable: `Assignment`



Function / FB call: `Box`
identifier of the function shall be specified inside the box,
Identifier of an FB shall be specified above the box

CODESYS

Implementation in the template project - FBD

- Choose the Function Block Diagram (FBD) language when adding a new POU
- Don’t forget the Freewheeling task scheduling the execution of the program
- Testing
  - packages arrive automatically on the conveyor
  - sensors (proximity switch of the conveyor, limit switches of the pusher) are displayed in green when active and in gray when inactive
  - active state of the conveyor belt is reported by its green color

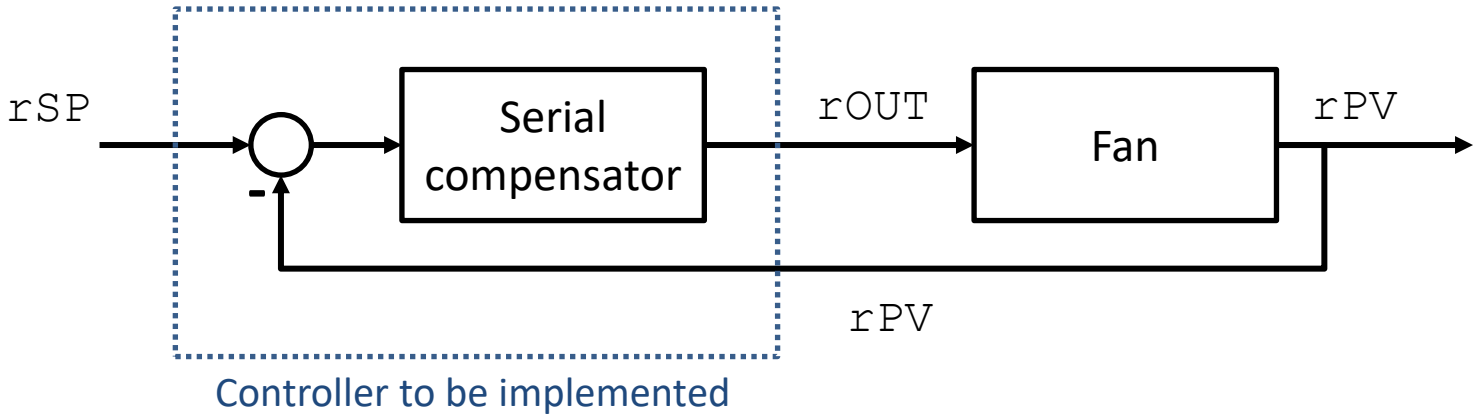
Address	Signal
%IX0.0	Rear limit switch of the pusher
%IX0.1	Front limit switch of the pusher
%IX0.2	Proximity switch of the conveyor belt

Address	Signal
%QX0.0	Pusher motor on/off
%QX0.1	Pusher direction (0: backward, 1: forward)
%QX0.2	Conveyor on/off

CODESYS

Problem – Fan speed control

- Analog control loop
  - Process Value (*PV*, y) – speed of the fan: `rPV`
  - Setpoint (*SP*, r) – required speed: `rSP`
  - Control signal (u) – frequency: `rOUT`
  - Range of signals is $[-100 \ldots 100]$ (%), their type is floating point (`REAL`)



Scaling

- Process variable and setpoint are available at analogue input channels equipped with 12 bit ADCs (`WORD` data type)
- The control signal is set by an analogue output channel equipped with a 12 bites DAC (`WORD` data type)
- Signals need to be scaled
  - Scaling ADC values:

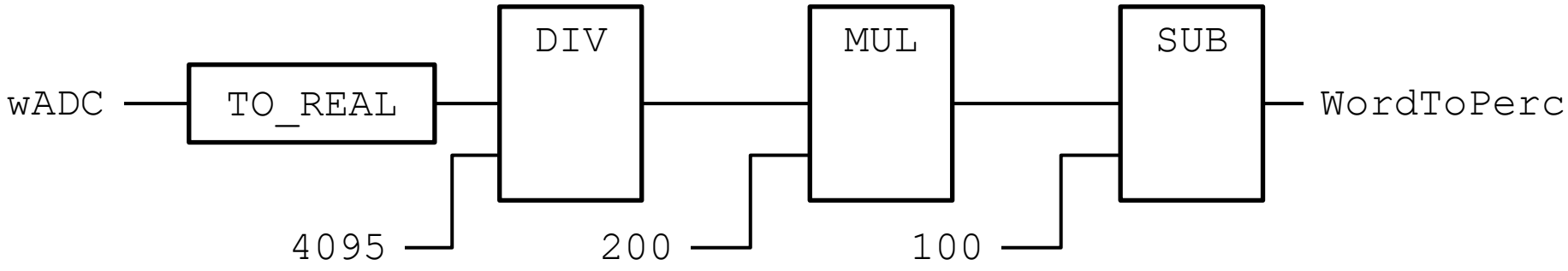
$$y[\%] = \frac{x_{ADC}}{4095} \cdot 200 - 100$$

  - Scaling the DAC value:

$$x_{DAC} = \frac{y[\%] + 100}{200} \cdot 4095$$

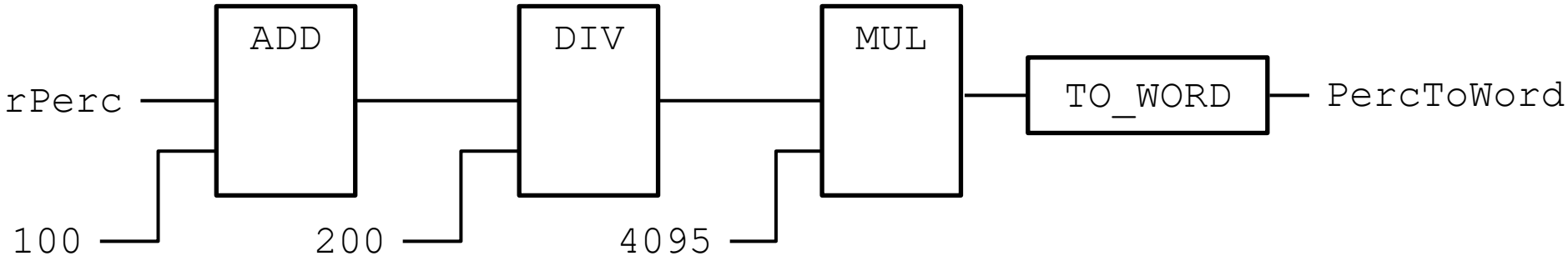
Scaling

```
FUNCTION WordToPerc : REAL
VAR_INPUT
    wADC : WORD;
END_VAR
```



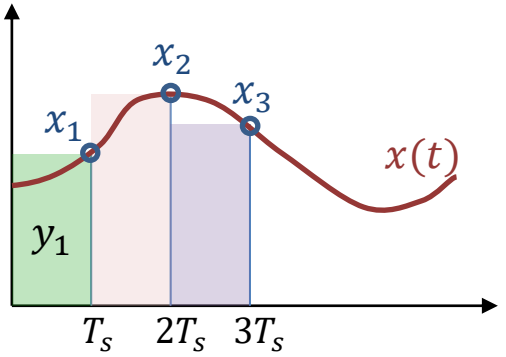
Scaling

```
FUNCTION PercToWord : WORD
VAR_INPUT
    rPerc : REAL;
END_VAR
```



Control algorithm (PI)

- $u(t) = A_p e(t) + \frac{A_p}{T_i} \int e(t)dt \Leftrightarrow u(t) = K_p e(t) + K_I \int e(t)dt$
- Numeric integration
  - sampling with T_s sampling time: $x_k = x(k \cdot T_s)$
  - $y(t) = \int_0^t x(\tau)d\tau \Leftrightarrow y_k \approx y_{k-1} + x_k \cdot T_s$
- Control algorithm
  - $e_k = r_k - y_k$
  - $I_k = I_{k-1} + T_s \cdot e_k$
  - $u_k = K_p e_k + K_i I_k$



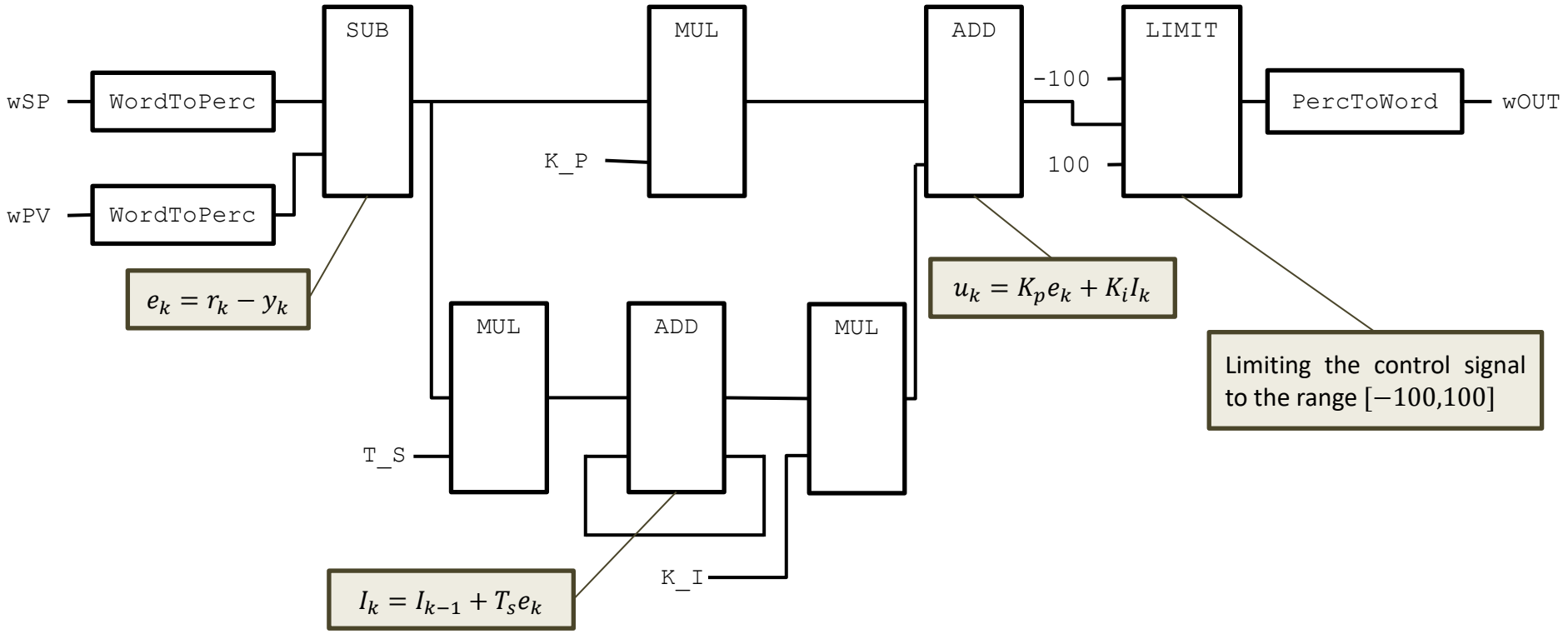
$y_1 = T_s x_1$
$y_2 = y_1 + T_s x_2$
$y_3 = y_2 + T_s x_3$

Control algorithm

```
PROGRAM FanControl
VAR_INPUT
  wSP AT %IW0 : WORD;
  wPV AT %IW1 : WORD;
END_VAR
```

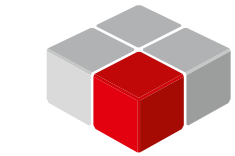
```
VAR_OUTPUT
  wOUT AT %QW0 : WORD;
END_VAR
```

```
VAR CONSTANT
  K_P : REAL := 4;
  K_I : REAL := 9;
  T_S : REAL := 0.05;
END_VAR
```



Implementation in the template project - CFC

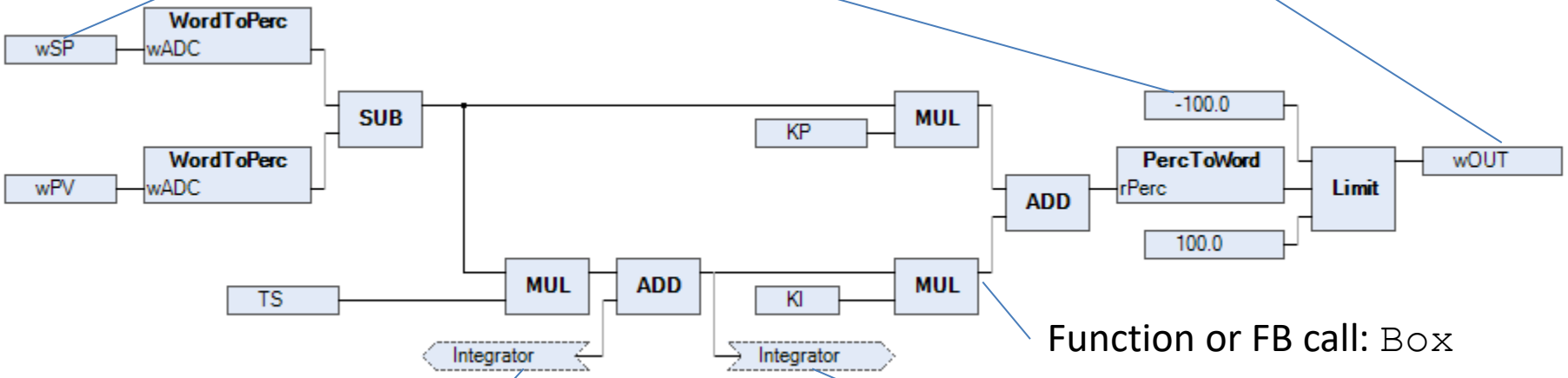
- The Continuous Function Chart (CFC) language is suitable for algorithms working on numeric variables
- The diagram is not organized to networks, blocks can be freely placed and connected
- Evaluation order of blocks follows the data flow principle by default but can be set individually
- Feedback paths can be defined even in forms of connectors



Implementation in the template project - CFC

Assign a literal or variable to an input of a block: `Input`

Assign the output of a block to a variable: `Output`



Function or FB call: `Box`

End point of connector: `Connection Sink`

Az összeköttetés itt visszacsatolást valósít meg

Start of the connector: `Connection Source`

Wires between blocks can be defined using the `Pointer` tool, by keeping the mouse button pressed

Implementation in the template project - CFC

- Select Continuous Function Chart (CFC) as implementation language for the POUs
- The template project already contains an empty program with declaration of the IO variables, a task scheduling the program according to the sample time and a conversion function
- Testing
  - the setpoint can be set using the slider
  - values of the setpoint, the process value and the control signal are displayed in a common chart
  - it is recommended to test the performance of the control loop with different values of parameters K_p and K_i

CODESYS