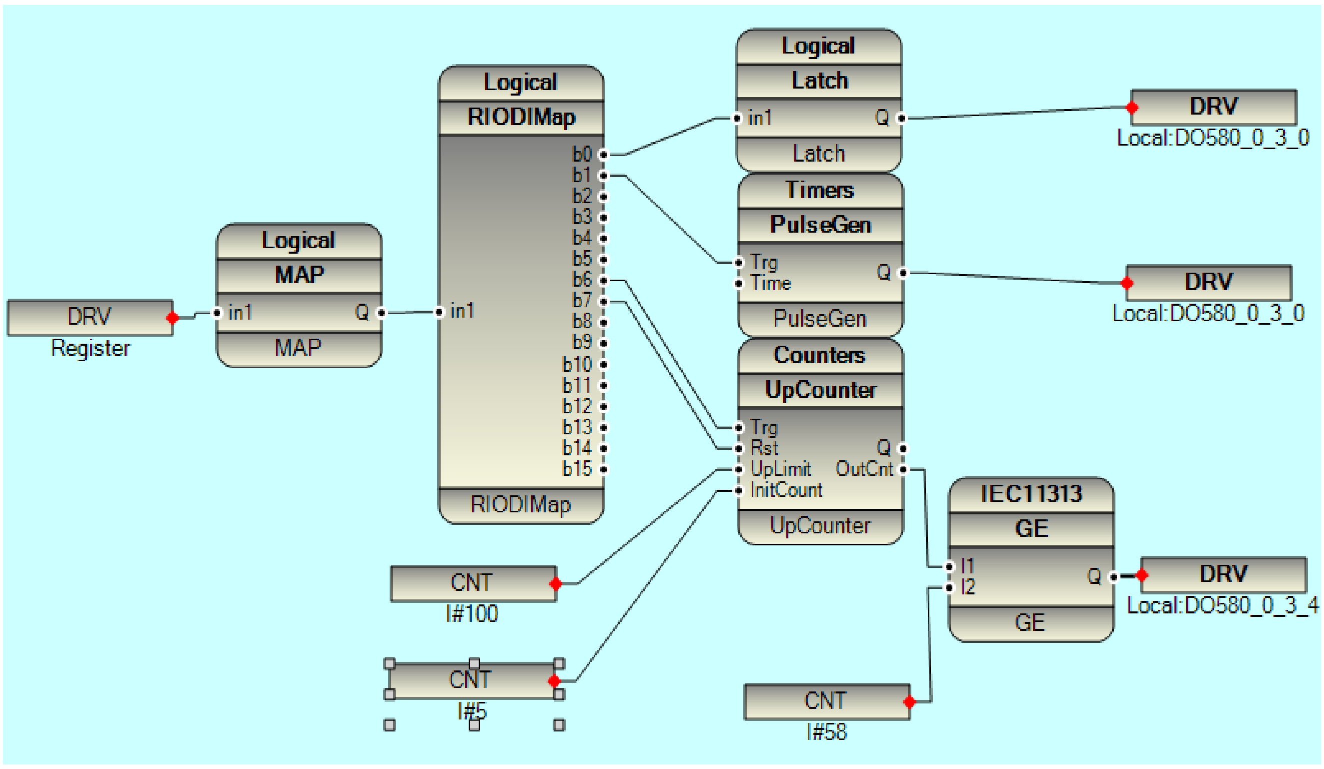
pbsSoftLogic Function Block User Manual



Version:2.0

Standard Function Blocks Definition

PBSSOFTLOGIC features a comprehensive library of **ready-to-use, pre-tested function blocks** for easy and rapid application development.

The function blocks are organized into the following logical groups:

- **Math:** Mathematical function groups.
- **Timers:** Timer function blocks and signal generators.
- **Counters:** Counter function blocks.
- **Logical:** Logical function block groups.
- **Process:** High-level process function blocks.
- **IEC 61131-3:** Standard function blocks based on the IEC 61131-3 programming standard.
- **Scheduling:** Daily, weekly, monthly, and yearly scheduling function blocks.

1 Math Group

1.1 ADD2 Function Block

The ADD2 function block is a basic arithmetic addition module designed for use in automation systems, control software, or programmable logic environments that support function block definitions (e.g., IEC 61499-compliant systems).

This block performs a simple summation operation: it adds the values of two input variables (in1 and in2) and outputs the result via Q. The block is inherently **active** (enabled by default), meaning it processes inputs immediately upon execution unless explicitly disabled in the hosting environment.

1.1.1 Inputs

The ADD2 block accepts two inputs, both configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
in1	First operand for addition.	Any (numeric recommended)	0.0	Connect a signal source here. Supports real-time updates.
in2	Second operand for addition.	Any (numeric recommended)	0.0	Connect a signal source here. Supports real-time updates.

- **Configuration**: Inputs can be set via direct assignment, linked to other blocks, or driven by sensors/variables in the system.
- **Validation**: Ensure inputs are numeric to prevent coercion errors. Non-numeric values may result in 0.0 or exceptions depending on the runtime.

1.1.2 Outputs

The ADD2 block produces one output, updated on each execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Result of the addition (in1 + in2).	Any (numeric)	0.0	Available for downstream connections. Updates synchronously with inputs.

1.1.3 Functional Behavior:

Add two numbers with different types.

1.1.4 Testing:

- Set in1 = 5.0 and in2 = 3.0; verify Q = 8.0.

- Handle edge cases: Negative numbers (e.g., in1 = -2.5, in2 = 1.0 → Q = -1.5), zeros (Q = 0.0), or large values (watch for overflow).

1.2 Multiply Function Block

The Multiply function block performs a basic arithmetic multiplication operation between two input values, producing their product as output. It is a stateless, combinatorial block suitable for mathematical computations in automation, signal processing, and control systems. The block supports generic data types ("Any") for flexibility across integer, floating-point, or other numeric representations, though the implementation assumes double-precision floating-point for precision.

1.2.1 Inputs

The Multiply block has two inputs, processed combinatorially on each execution cycle:

Input Name	Description	Data Type	Initial Value	Usage Notes
in1	First operand for multiplication.	Any	0.0	Accepts numeric types (e.g., int, real, double). Dynamic changes propagate immediately to output.
in2	Second operand for multiplication.	Any	0.0	Accepts numeric types (e.g., int, real, double). Dynamic changes propagate immediately to output.

1.2.2 Outputs

The Multiply block produces one output, updated based on the current inputs:

Output Name	Description	Data Type	Initial Value	Usage Notes
Q	Product of the two inputs (in1 * in2).	Any	0.0	Matches input types (e.g., double). Updates synchronously; no latching.

1.2.3 Functional behavior

The block executes a simple multiplication in a single scan cycle:

1. **Input Sampling**: Read current values of in1 and in2.
2. **Computation**: Calculate Q = in1 * in2.
3. **Output Update**: Assign the result to Q.

1.2.4 Testing

- **Basic Multiplication**: in1=2.5, in2=4.0 → Q=10.0.
- **Zero Handling**: in1=0, in2=100 → Q=0.0.
- **Dynamic Change**: in1 changes from 3 to 5 mid-cycle → Q immediately becomes 5 * in2.
- **Negative Values**: in1=-2, in2=3 → Q=-6.0.
- **Large Values**: in1=1e10, in2=1e10 → Q=1e20 (subject to double precision limits).

1.3 Divide Function Block

The **Divide** Function Block (FB) is a core arithmetic division. This FB calculates the quotient of two input values of any numeric type, producing their division result as output. It is essential for ratio computations, scaling factors, or normalization tasks in control applications. This FB supports diverse uses, from speed ratios in drives to concentration calculations in processes.

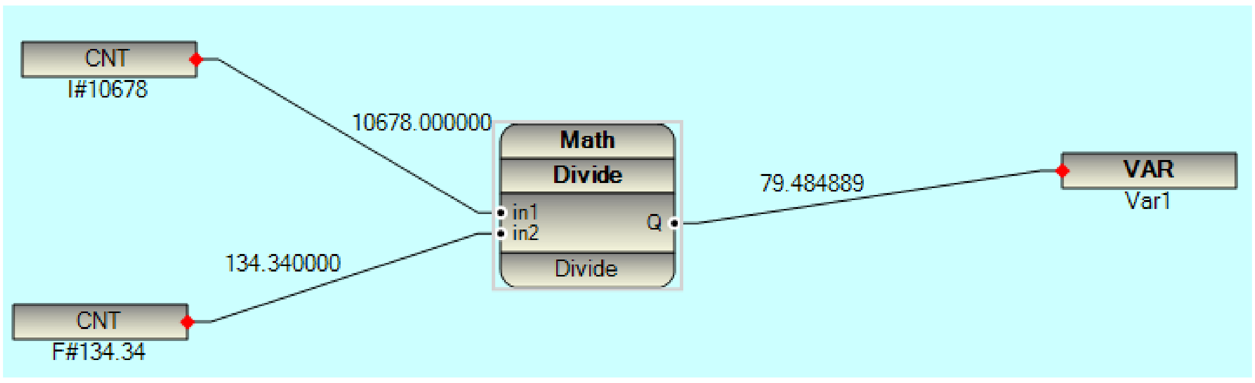


Figure 1-1 Math Group Function Block (Divide)

1.3.1 Inputs

The following table details all input pins, their types, initial values, and functional descriptions.

Pin Name	Type	Initial Value	Description
in1	Any	0.0	Dividend (numerator). Accepts any numeric type (integer or real).
in2	Any	1.0	Divisor (denominator). Accepts any numeric type; divides into in1 to produce Q. Defaults to 1.0 to avoid errors.

1.3.2 Outputs

The following table details all output pins, their types, initial values, and functional descriptions.

Pin Name	Type	Initial Value	Description
Q	Any	0.0	Result of the division: Q = in1 / in2. Output type matches or promotes from inputs.

1.3.3 Functional Behavior:

Divide two numbers with different types.

1.3.4 Example Scenarios

Scenario 1: Basic Integer Division

- **Conditions**: in1=20 (INT), in2=4 (INT).
- **Behavior**: Q=5 (INT). Truncates for whole-number results.

Scenario 2: Floating-Point Quotient

- **Conditions**: in1=15.5 (REAL), in2=3.0 (REAL).
- **Behavior**: Q=5.166... (REAL). Retains decimal precision.

Scenario 3: Mixed-Type Promotion

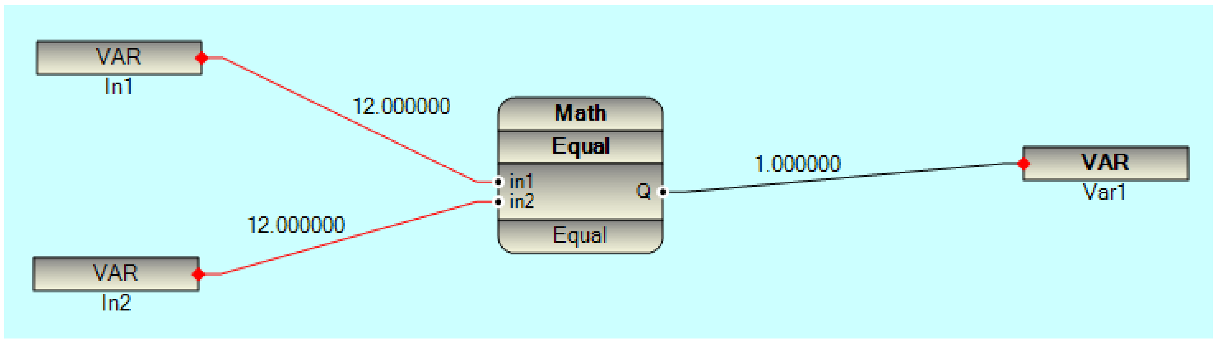
- **Conditions**: in1=100 (INT), in2=4.0 (REAL).
- **Behavior**: Q=25.0 (REAL). Ensures accurate scaling.

Scenario 4: Safe Default Operation

- **Conditions**: in1=0.0 (default), in2=1.0 (default).
- **Behavior**: Q=0.0. Harmless in uninitialized setups.

1.4 Equal Function Block

The **Equal** Function Block (FB) is a fundamental comparison module. This FB evaluates whether two input values of any numeric type are identical, outputting a digital true/false result. It is crucial for conditional logic, equality checks in sequences, or validation in control algorithms. This FB is ideal for applications like setpoint matching, fault detection, or data synchronization in SCADA systems.



1.4.1 Inputs

The following table details all input pins, their types, initial values, and functional descriptions.

Pin Name	Type	Initial Value	Description
in1	Any	0.0	First value for comparison. Accepts any numeric type (integer or real).
in2	Any	0.0	Second value for comparison. Accepts any numeric type; Q=True if in1 equals in2.

1.4.2 Outputs

The following table details all output pins, their types, initial values, and functional descriptions.

Pin Name	Type	Initial Value	Description
Q	Digital	False	Equality result: True if in1 == in2, False otherwise.

1.4.3 Functional Behavior:

If In1 is Equal to In2, Output is 1, otherwise Output is 0.

1.4.4 Example Scenarios

Scenario 1: Exact Integer Match

- **Conditions**: in1=42 (INT), in2=42 (INT).
- **Behavior**: Q=True. Triggers downstream logic like "match found."

Scenario 2: Floating-Point Equality

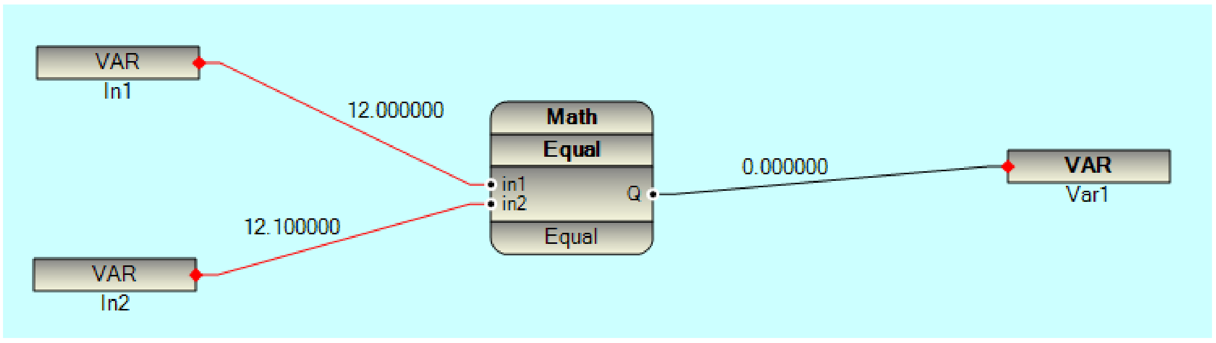
- **Conditions**: in1=3.14159 (REAL), in2=3.14159 (REAL).
- **Behavior**: Q=True. Validates precise calculations.

Scenario 3: Type-Promoted Comparison

- **Conditions**: in1=10 (INT), in2=10.0 (REAL).
- **Behavior**: Q=True. Handles heterogeneous data sources.

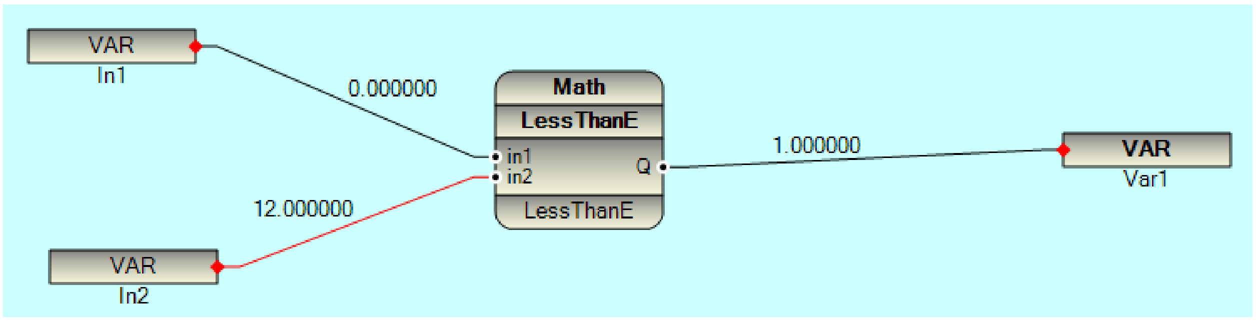
Scenario 4: Non-Match with Defaults

- **Conditions**: in1=0.0 (default), in2=1.0.
- **Behavior**: Q=False. Safe for initialization checks.



1.5 LessThanE Function Block

The **LessThanE** Function Block (FB) is a core comparison module. This FB determines if the first input value is less than or equal to the second, outputting a Boolean result. It is vital for threshold monitoring, sequencing decisions, or conditional branching in control logic. This FB excels in applications like alarm setpoints, flow limits, or safety interlocks where boundary-inclusive decisions are required.



1.5.1 Inputs

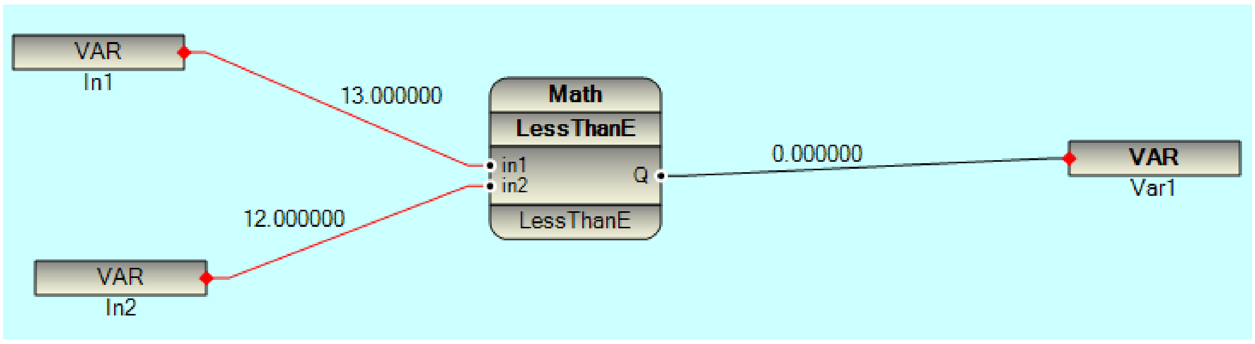
The following table details all input pins, their types, initial values, and functional descriptions.

Pin Name	Type	Initial Value	Description
in1	Any	0.0	First value (left operand). Accepts any numeric type (integer or real).
in2	Any	0.0	Second value (threshold). Accepts any numeric type; Q=True if in1 ≤ in2.

1.5.2 Outputs

The following table details all output pins, their types, initial values, and functional descriptions.

Pin Name	Type	Initial Value	Description
Q	bool	false	Inequality result: True if in1 ≤ in2, False otherwise.



1.5.3 Functional Behavior:

If In1 is Less Than or Equal to In2, Output is 1, otherwise Output is 0

1.5.4 Example Scenarios

Scenario 1: Equality Boundary

- **Conditions**: in1=10.0 (REAL), in2=10.0 (REAL).
- **Behavior**: Q=True. Activates "at setpoint" logic.

Scenario 2: Strict Less-Than

- **Conditions**: in1=7 (INT), in2=15 (INT).
- **Behavior**: Q=True. Triggers low-level actions like "below threshold."

Scenario 3: Greater-Than Failure

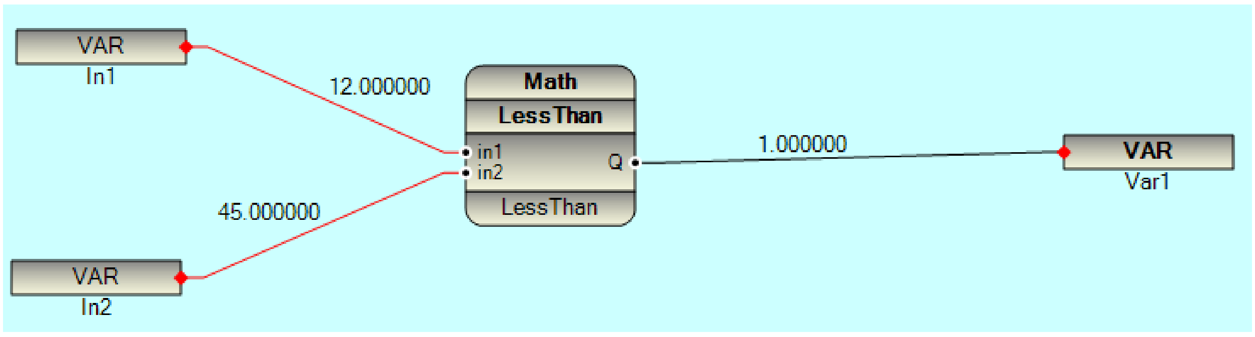
- **Conditions**: in1=20 (INT), in2=10 (INT).
- **Behavior**: Q=False. Inhibits operation, e.g., high-limit alarm.

Scenario 4: Default Initialization

- **Conditions**: in1=0.0 (default), in2=5.0.
- **Behavior**: Q=True. Ensures safe startup state.

1.6 LessThan Function Block

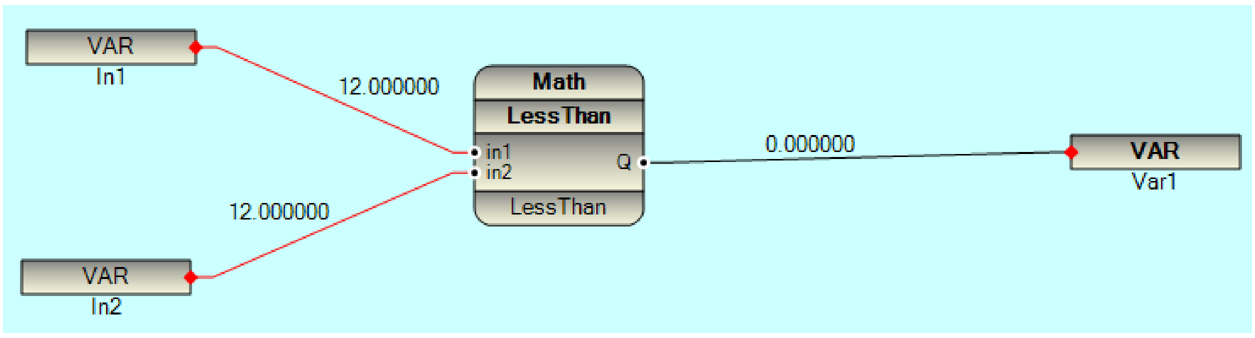
The **LessThan** Function Block (FB) is a essential comparison module. This FB checks if the first input value is strictly less than the second, outputting a Boolean result. It is key for strict threshold enforcement, ordering decisions, or conditional controls in automation sequences. This FB is perfect for applications such as under-speed detection, low-level alarms, or phase sequencing where equality does not qualify.



1.6.1 Inputs

The following table details all input pins, their types, initial values, and functional descriptions.

Pin Name	Type	Initial Value	Description
in1	Any	0.0	First value (left operand). Accepts any numeric type (integer or real).
in2	Any	0.0	Second value (threshold). Accepts any numeric type; Q=True if in1 < in2.



1.6.2 Outputs

The following table details all output pins, their types, initial values, and functional descriptions.

Pin Name	Type	Initial Value	Description
Q	bool	false	Strict inequality result: True if in1 < in2, False otherwise (including equality).

1.6.3 Functional Behavior:

If In1 is Less Than to In2, Output is 1, otherwise Output is 0

1.6.4 Example Scenarios

Scenario 1: Strict Under Threshold

- **Conditions**: in1=8 (INT), in2=10 (INT).
- **Behavior**: Q=True. Initiates "low value" response.

Scenario 2: Equality Rejection

- **Conditions**: in1=15.0 (REAL), in2=15.0 (REAL).
- **Behavior**: Q=False. Distinguishes from at-threshold states.

Scenario 3: Greater-Than Case

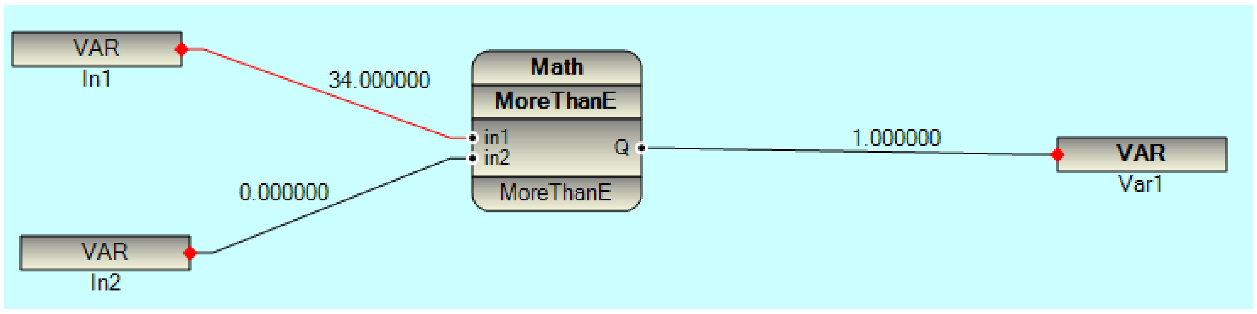
- **Conditions**: in1=12 (INT), in2=9 (INT).
- **Behavior**: Q=False. Blocks actions like "fill required."

Scenario 4: Default Non-Trigger

- **Conditions**: in1=0.0 (default), in2=0.0 (default).
- **Behavior**: Q=False. Avoids false positives on startup.

1.7 MoreThanE Function Block

The **MoreThanE** evaluates whether the first input value exceeds the second input value, producing a Boolean output. This block supports input type ("Any"), automatically handling integers, floats, or other compatible numerical values during each scan cycle.



1.7.1 Inputs

The MoreThanE block accepts the following inputs, which are compared directly.

Input Name	Type	Description	Initial Value	Notes
in1	Any	Primary input: The value to test against the in2.	0.0	Can be integer, float, or compatible type; treated numerically.
in2	Any	Threshold input: The reference value for comparison.	0.0	Can be integer, float, or compatible type; treated numerically.

1.7.2 Outputs

The MoreThanE block produces a single boolean output, updated at the end of each execution cycle.

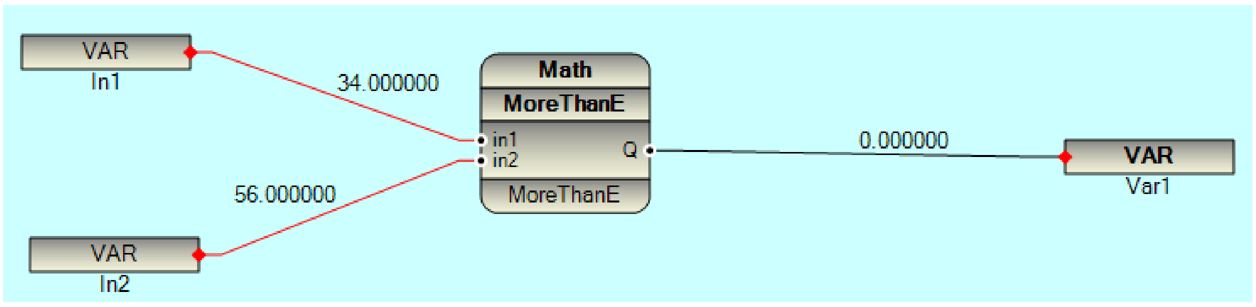
Output Name	Type	Description	Initial Value	Notes
Q	bool	Comparison result: True if in1 ≥ in2 ;False otherwise.	false	-

1.7.3 Functional Behavior:

If In1 is More Than or Equal to In2, Output is 1, otherwise Output is 0

1.7.4 Example Scenarios:

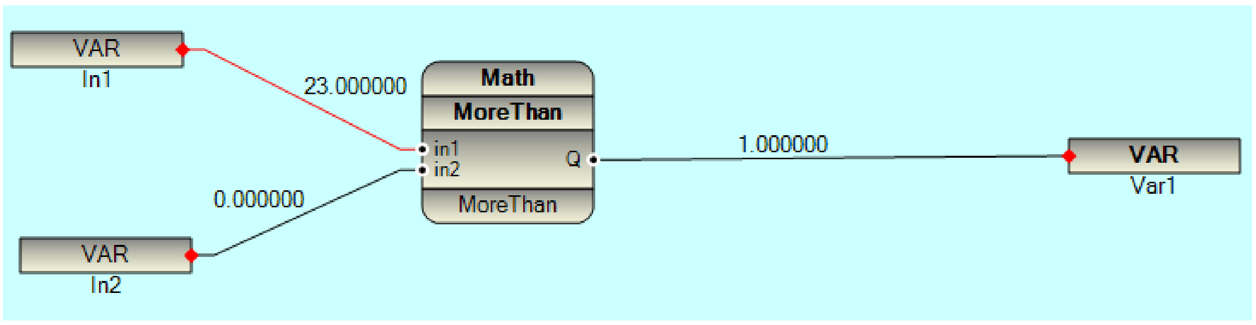
- **Integer Case**: in1 = 10, in2 = 5 → Q = True.
- **Float Case**: in1 = 5.000001, in2 = 5.0 → Q = True (epsilon allows slight excess).
- **Equal Case**: in1 = 5.0, in2 = 5.0 → Q = False.
- **Below Threshold**: in1 = 4.9, in2 = 5.0 → Q = False.



1.8 MoreThan Function Block

The **MoreThan** function block evaluates whether the value of the first input (in1) is strictly greater than the value of the second input (in2).

- **Behavior**:
  - If in1 > in2, the output Q is set to true.
  - Otherwise, Q is set to false.

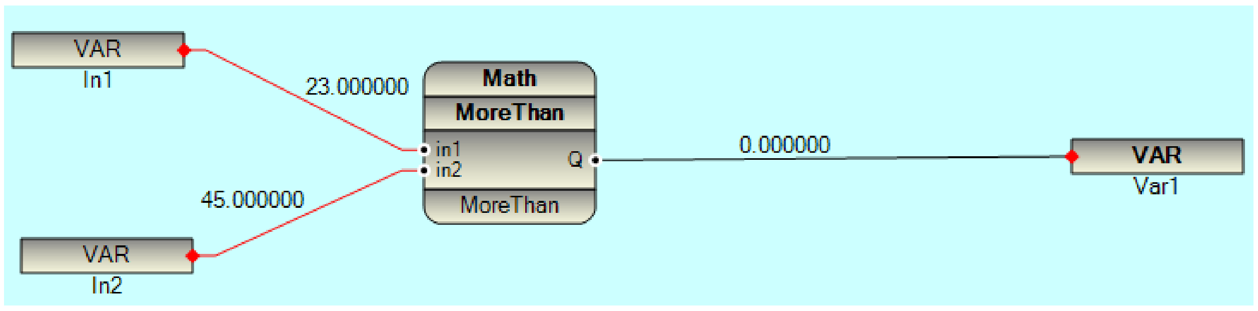


1.8.1 Inputs

The block accepts two inputs, both configurable as numeric types.

Input Name	Description	Data Type	Initial Value	Notes
in1	First operand for comparison (the value to check if greater).	Any (numeric recommended: INT, REAL, etc.)	0.0	Connect to a variable, constant, or sensor reading.
in2	Second operand for comparison (the reference value).	Any (numeric recommended: INT, REAL, etc.)	0.0	Connect to a setpoint, constant, or another variable.

- **Type Flexibility**: The "Any" type allows broad compatibility, but for optimal performance, use consistent numeric types (e.g., both REAL for floating-point precision).
- **Initialization**: Defaults to 0.0 ensure safe startup; override as needed in your program.



1.8.2 Outputs

The block produces a single Boolean output.

Output Name	Description	Data Type	Initial Value	Notes
Q	Result of the comparison (true if in1 > in2, else false).	BOOL	false	Use to drive logic, enable/disable other blocks, or set flags. Rising edge detection may be applied externally if needed.

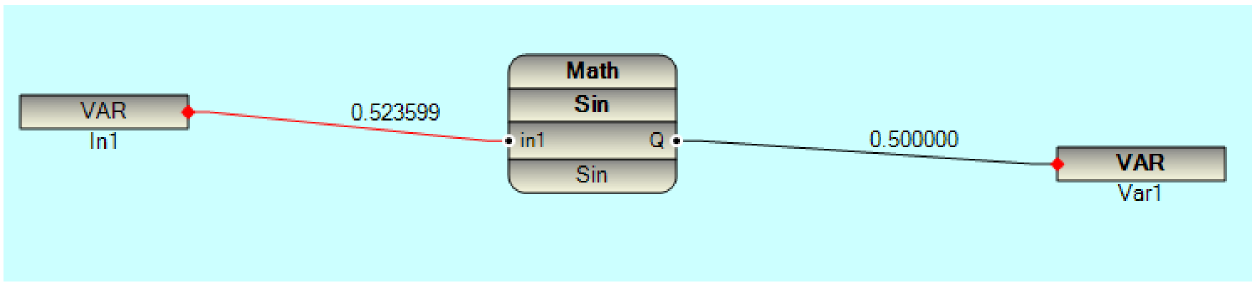
1.8.3 Functional Behavior:

If In1 is More Than In2, Output is 1, otherwise Output is 0.

1.9 Sin Function Block

The **Sin** function block calculates the sine of the input value (in1), assuming the angle is provided in radians (standard mathematical convention).

- **Behavior**:
  - The output Q is the sine value, ranging from -1.0 to 1.0.
  - For angles outside the typical [-π, π] range, it wraps periodically as per the sine function's properties.
- **Assumptions**: Input is a numeric value in radians. Degrees mode is not supported; convert degrees to radians externally if needed (e.g., multiply by π/180). Non-numeric or out-of-range inputs may produce NaN or undefined results—validate in your application.



1.9.1 Inputs

The block accepts a single input of double precision.

Input Name	Description	Data Type	Initial Value	Notes
in1	Angle value in radians for which to compute the sine.	DOUBLE	0	Connect to a variable, constant, or calculated angle. Range: Typically -∞ to ∞, but periodic every 2π.

- **Type Specificity**: DOUBLE ensures high precision for floating-point calculations.
- **Initialization**: Defaults to 0 (sin(0) = 0), providing a neutral startup value.

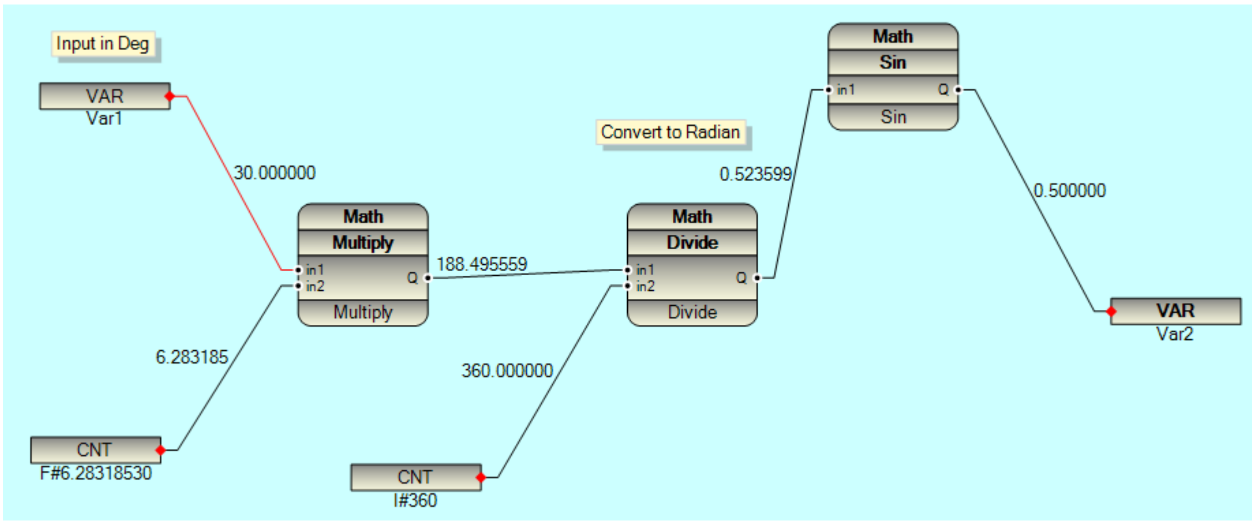
1.9.2 Outputs

The block produces a single double-precision output.

Output Name	Description	Data Type	Initial Value	Notes
Q	Sine of the input angle (sin(in1)).	DOUBLE	0	-

1.9.3 Functional Behavior:

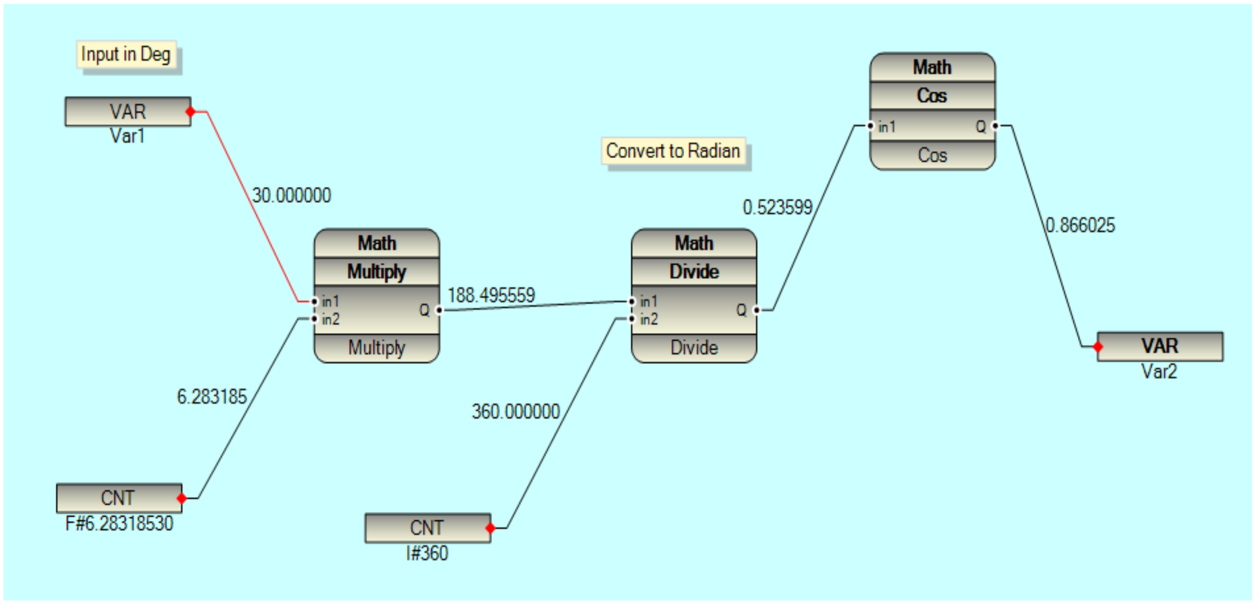
Sin Function returns sine of "in1", where "in1" is given in Radians.



1.10 Cos Function Block

The **Cos** function block calculates the Cos of the input value (in1), assuming the angle is provided in radians (standard mathematical convention).

- **Behavior**:
  - The output Q is the cosine value, ranging from -1.0 to 1.0.
  - For angles outside the typical [-π, π] range, it wraps periodically as per the cosine function's properties.
- **Assumptions**: Input is a numeric value in radians. Degrees mode is not supported; convert degrees to radians externally if needed (e.g., multiply by π/180). Non-numeric or out-of-range inputs may produce NaN or undefined results—validate in your application.



1.10.1 Inputs

The block accepts a single input of double precision.

Input Name	Description	Data Type	Initial Value	Notes
in1	Angle value in radians for which to compute the cosine.	DOUBLE	0	-

- **Type Specificity**: DOUBLE ensures high precision for floating-point calculations.
- **Initialization**: Defaults to 0 (cos (0) = 1.0), providing a neutral startup value.

1.10.2 Outputs

The block produces a single double-precision output.

Output Name	Description	Data Type	Initial Value	Notes
Q	Cosine of the input angle (cos(in1)).	DOUBLE	0	Use for further math operations, phase calculations, or driving servos. Precision matches input.

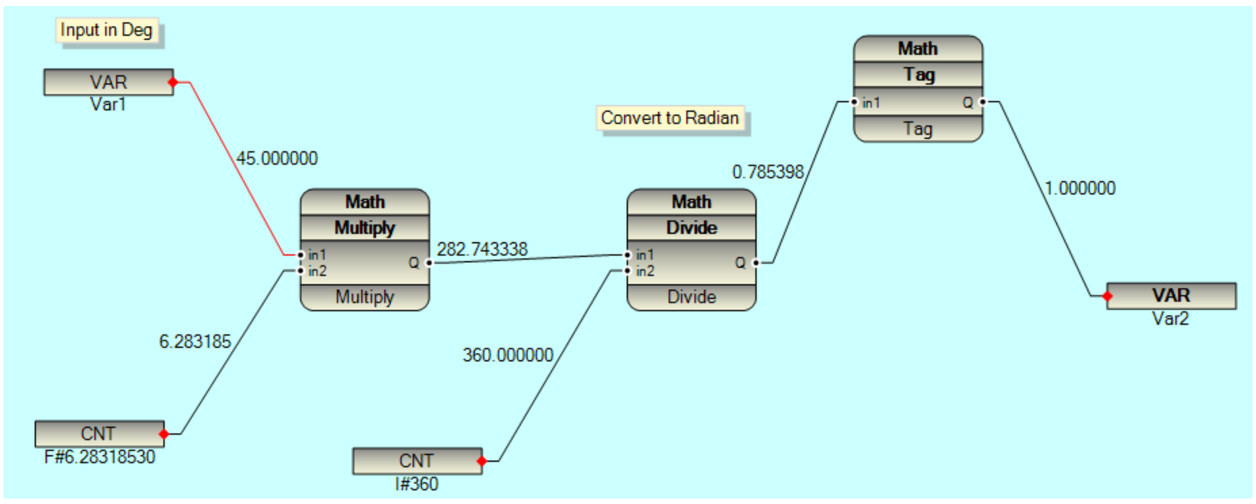
1.10.3 Functional Behavior:

Cos Function Returns Cosine of "in1", where "in1" is given in Radians.

1.11 Tag (Tangent) Function Block

The **Tag** function block computes the tangent of the input value (in1)

-
- **Behavior**:
  - The output Q is the tangent value, ranging from -∞ to +∞
  - Undefined at points like π/2 + kπ (k integer), where output may be NaN or infinity.
- **Assumptions**: Input is a double-precision numeric value in radians. Degrees are not supported—convert externally (e.g., multiply by π/180). Avoid inputs near asymptotes to prevent errors; validate in your application.



1.11.1 Inputs

The block accepts a single input of double precision.

Input Name	Description	Data Type	Initial Value	Notes
in1	Angle value in radians for tangent computation.	DOUBLE	0	Connect to angles from sensors, timers, or calculations. Safe range: Avoid ±π/2 + kπ.

- **Type Specificity**: DOUBLE for precise floating-point handling.
- **Initialization**: 0 (tan (0) = 0), ensuring stable startup.

1.11.2 Outputs

The block produces a single double-precision output.

Output Name	Description	Data Type	Initial Value	Notes
Q	Tangent of the input (tan(in1)).	DOUBLE	0	Monitor "Q" for infinities.

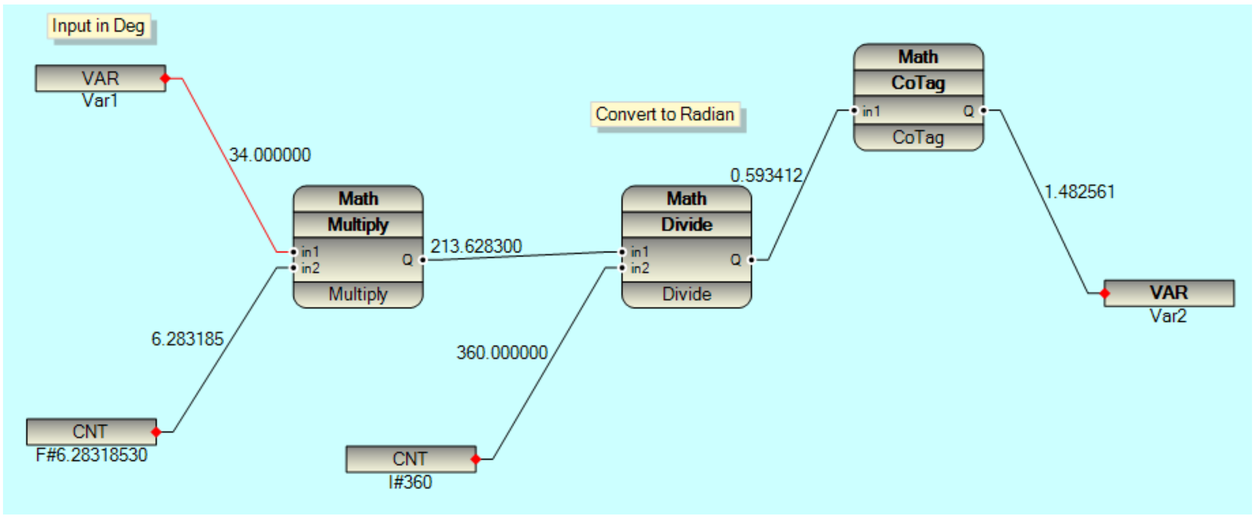
1.11.3 Functional Behavior:

Tag Function returns Tangent of "in1", where "in1" is given in Radians.

1.12 CoTag (Cotangent) Function Block

The **CoTag** function computes the cotangent of the input value (in1), assuming the angle is provided in radians (standard mathematical convention

- **Behavior**:
  - The output Q ranges from -∞ to +∞, with vertical asymptotes at integer multiples of π (kπ, where k is integer).
  - Undefined at points where sin(in1) = 0 (i.e., kπ), resulting in NaN or infinity.
- **Assumptions**: Input is a double-precision numeric value in radians. Degrees are not supported—convert externally (e.g., multiply by π/180). Avoid inputs at or near kπ to prevent errors; validate in your application.



1.12.1 Inputs

The block accepts a single input of double precision.

Input Name	Description	Data Type	Initial Value	Notes
in1	Angle value in radians for cotangent computation.	DOUBLE	0	Connect to angles from computations, timers, or sensors. Safe range: Avoid kπ.

- **Type Specificity**: DOUBLE for precise floating-point handling.
- **Initialization**: 0 (cot(0) is undefined, but implementation may default to a large value or NaN; monitor at startup).

1.12.2 Outputs

The block produces a single double-precision output.

Output Name	Description	Data Type	Initial Value	Notes
Q	Cotangent of the input (cot(in1)).	DOUBLE	0	Monitor "Q" for infinities.

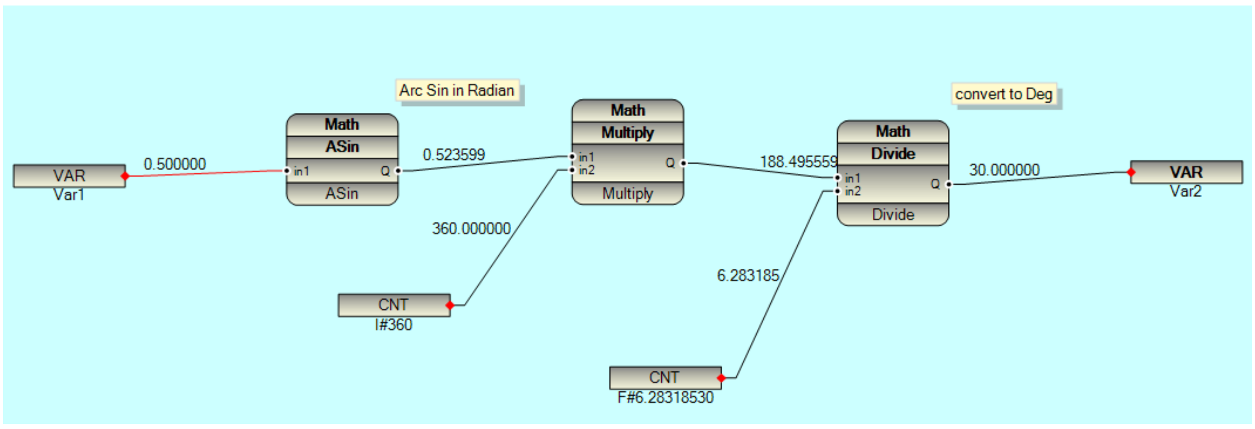
1.12.3 Functional Behavior:

CoTag Function returns reverse of Tangent of "in1", where "in1" is given in Radians.

1.13 ASin Function Block

The **ASin** function block computes the arcsine (inverse sine) of the input value (in1), returning the angle in radians whose sine is the input. The input must be in the range [-1, 1]; values outside this range typically result in NaN.

- **Behavior**:
  - Output Q ranges from -π/2 to π/2.
  - Domain restriction: in1 ∈ [-1, 1]; otherwise, undefined (NaN).
- **Assumptions**: Input is a double-precision value normalized to [-1, 1]. Outputs radians (not degrees)—convert to degrees externally if needed (e.g., multiply by 180/π). Validate inputs to avoid NaN in your application.



1.13.1 Inputs

The block accepts a single input of double precision.

Input Name	Description	Data Type	Initial Value	Notes
in1	Value for which to compute the arcsine (must be in [-1, 1]).	DOUBLE	0	Connect to normalized signals (e.g., from Sin output or sensor data scaled to [-1, 1]).

- **Type Specificity**: DOUBLE for precise floating-point handling.
- **Initialization**: 0 (asin(0) = 0), ensuring neutral startup.

1.13.2 Outputs

The block produces a single double-precision output.

Output Name	Description	Data Type	Initial Value	Notes
Q	Arcsine of the input (asin(in1)) in radians.	DOUBLE	0	-

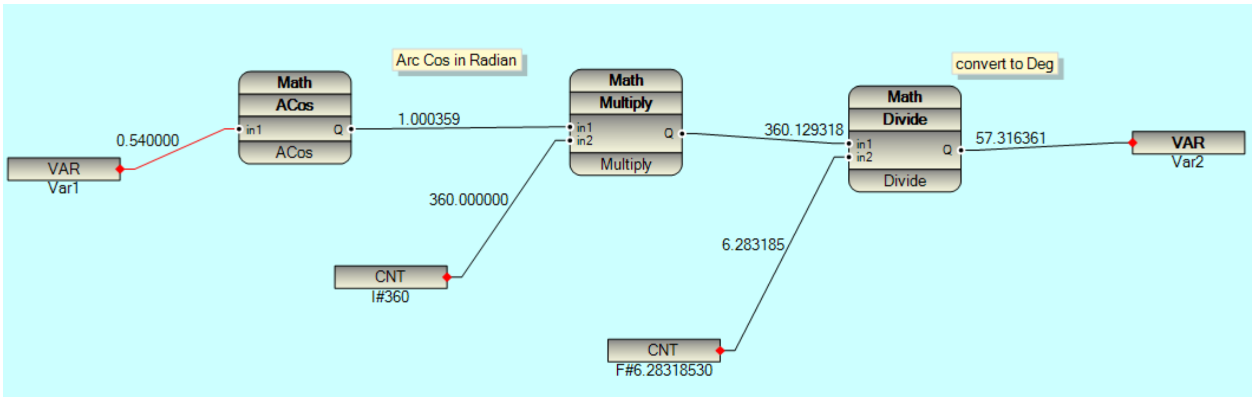
1.13.3 Functional Behavior:

ASin Function Returns Arc Sin of "in1", where "in1" is given in Radians.

1.14 ACos Function Block

The **ACos** function block computes the arccosine (inverse cosine) of the input value (in1), returning the angle in radians whose cosine is the input. The input must be in the range [-1, 1]; values outside this range typically result in NaN.

- **Behavior**:
  - Output Q ranges from 0 to π.
  - Domain restriction: in1 ∈ [-1, 1]; otherwise, undefined (NaN).
- **Assumptions**: Input is a double-precision value normalized to [-1, 1]. Outputs radians (not degrees)—convert to degrees externally if needed (e.g., multiply by 180/π). Validate inputs to avoid NaN in your application.



1.14.1 Inputs

The block accepts a single input of double precision.

Input Name	Description	Data Type	Initial Value	Notes
in1	Value for which to compute the arccosine (must be in [-1, 1]).	DOUBLE	0	Connect to normalized signals (e.g., from Cos output or sensor data scaled to [-1, 1]).

- **Type Specificity**: DOUBLE for precise floating-point handling.
- **Initialization**: 0 (acos(0) = π/2 ≈ 1.5708), providing a defined startup value.

1.14.2 Outputs

The block produces a single double-precision output.

Output Name	Description	Data Type	Initial Value	Notes

Q	Arccosine of the input (acos(in1)) in radians.	DOUBLE	0	-

1.14.3 Functional Behavior:

ACos Function returns Arc Cosine of "in1", where "in1" is given in Radians.

1.15 ATag (Arctangent) Function Block

The **ATag** function block computes the arctangent (inverse tangent) of the input value (in1), returning the angle in radians whose tangent is the input. The input represents the ratio (opposite over adjacent) and can range from -∞ to +∞; the output is always in the principal range [-π/2, π/2].

- **Behavior**:
  - Output Q ranges from -π/2 to π/2.
  - No domain restriction on input (handles full real line), but extreme values approach ±π/2 asymptotically.
- **Assumptions**: Input is a double-precision numeric value (the tangent ratio). Outputs radians (not degrees)—convert to degrees externally if needed (e.g., multiply by 180/π). The phrasing "In is given in Radians" in the description appears to be a misstatement; the input is a dimensionless ratio, and the output is the angle in radians.

1.15.1 Inputs

The block accepts a single input of double precision.

Input Name	Description	Data Type	Initial Value	Notes
in1	Ratio value (tangent) for which to compute the arctangent.	DOUBLE	0	-

- **Type Specificity**: DOUBLE for precise floating-point handling.
- **Initialization**: 0 (atan(0) = 0), ensuring neutral startup.

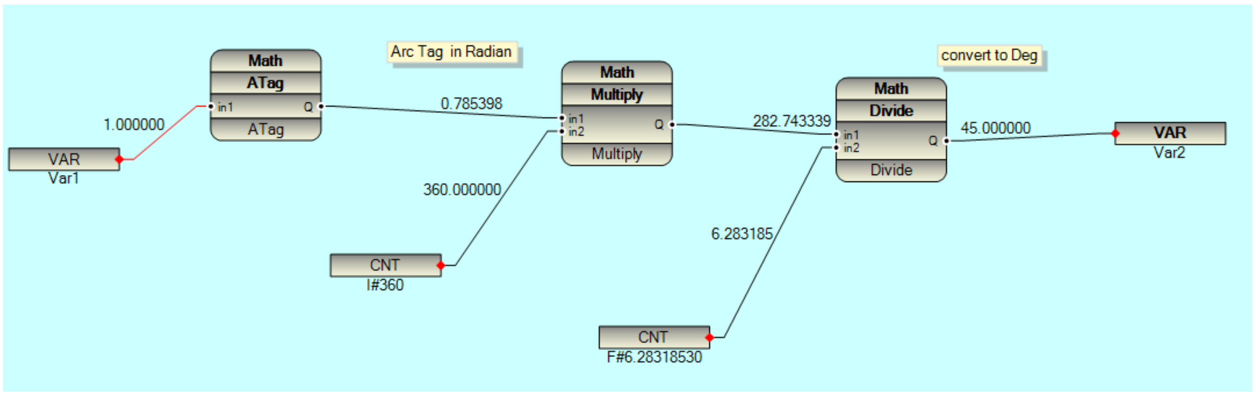
1.15.2 Outputs

The block produces a single double-precision output.

Output Name	Description	Data Type	Initial Value	Notes
Q	Arctangent of the input (atan(in1)) in radians.	DOUBLE	0	-

1.15.3 Functional Behavior:

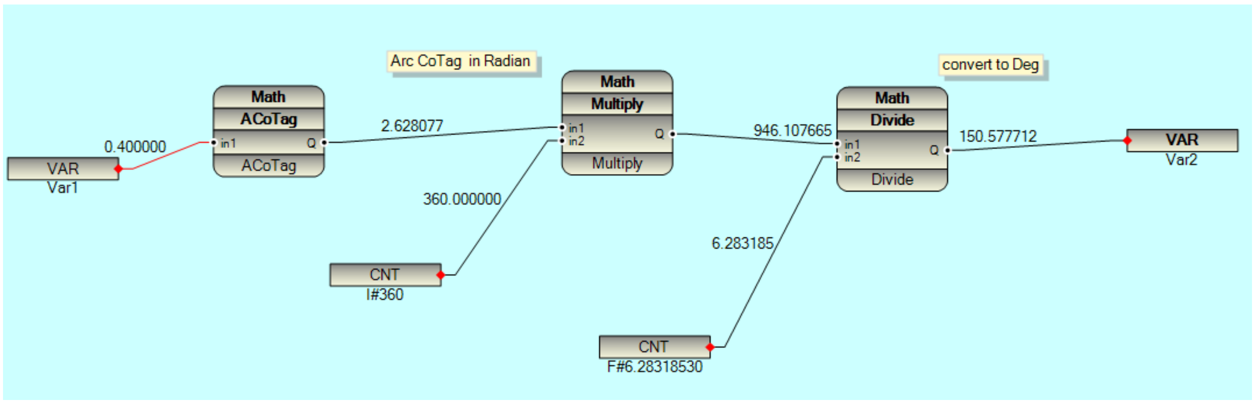
ATag Function Returns Arc Tangent of "in1", where "in1" is given in Radians.



1.16 ACoTag (Arccotangent) Function Block

The **ACoTag** function block computes the arc cotangent (inverse cotangent) of the input value (in1), returning the angle in radians whose cotangent is the input. The input represents the cotangent ratio and can range from -∞ to +∞; the output is in the principal range (0, π).

- **Behavior**:
  - Output Q ranges from 0 to π (exclusive).
  - Handles full real line input; at in1 = 0, output is π/2.
- **Assumptions**: Input is a double-precision numeric value (the cotangent ratio). Outputs radians (not degrees)—convert to degrees externally if needed (e.g., multiply by 180/π).



1.16.1 Inputs

The block accepts a single input of double precision.

Input Name	Description	Data Type	Initial Value	Notes
in1	Ratio value (cotangent) for which to compute the Arccotangent.	DOUBLE	0	-

- **Type Specificity**: DOUBLE for precise floating-point handling.
- **Initialization**: 0 (Acot (0) = π/2 ≈ 1.5708), providing a defined startup value.

1.16.2 Outputs

The block produces a single double-precision output.

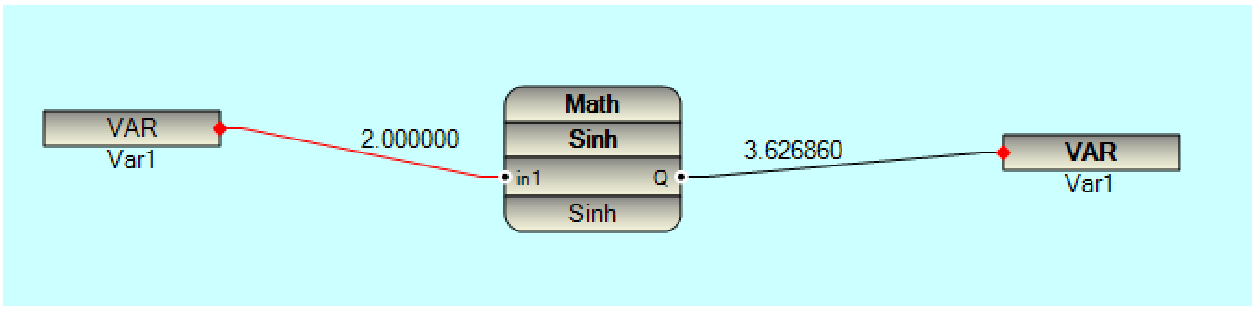
Output Name	Description	Data Type	Initial Value	Notes
Q	Arccotangent of the input (acot(in1)) in radians.	DOUBLE	0	-

1.16.3 Functional Behavior:

ACoTag Function returns reversed of Arc Tangent of In1, where in is given in Radians.

1.17 Sinh Function Block

The Sinh function block computes the hyperbolic sine of an input value, this block takes a single double-precision floating-point input (in1) and produces the result via output Q.



1.17.1 Inputs

The Sinh block accepts one input, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
in1	Input value x for hyperbolic sine computation.	double	0.0	-

- **Configuration**: Set via direct assignment, linkage to other blocks, or external variables/sensors.
- **Validation**: Input is coerced to double; non-numeric values may yield 0.0 or exceptions in the runtime.

1.17.2 Outputs

The Sinh block produces one output, updated on each execution cycle.

Output Name	Description	Type	Initial Value	Notes

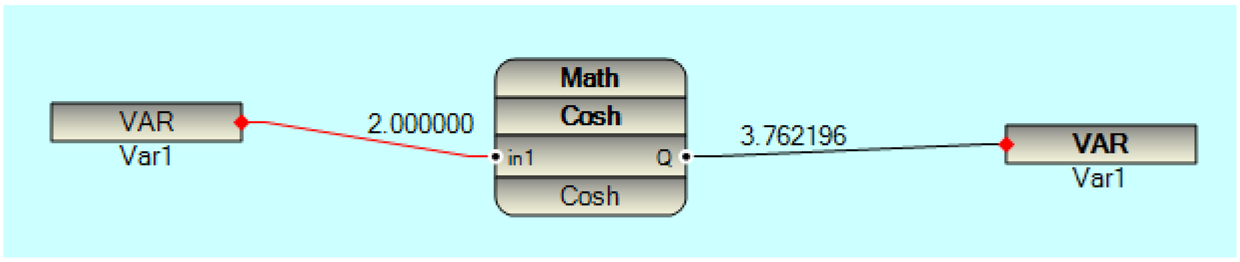
Q	Hyperbolic sine of in1 (sinh(in1)).	double	0.0	Available for downstream connections. Updates synchronously with input.

1.17.3 Testing:

- Set in1 = 0; verify Q = 0 (sinh(0) = 0).
- in1 = 1; Q ≈ 1.1752011936438014.
- in1 = -1; Q ≈ -1.1752011936438014.
- Test large values: in1 = 10; Q ≈ 11013.232874703393

1.18 Cosh Function Block

The Cosh function block computes the hyperbolic Cos of an input value, this block takes a single double-precision floating-point input (in1) and produces the result via output Q..



1.18.1 Inputs

The Cosh block accepts one input, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
in1	Input value x for hyperbolic cosine computation.	double	0.0	-

- **Configuration**: Assign directly, link to other blocks, or drive with sensors/variables.
- **Validation**: Coerces to double; invalid inputs may default to 0.0 or raise exceptions.

1.18.2 Outputs

The Cosh block produces one output, updated on each execution cycle.

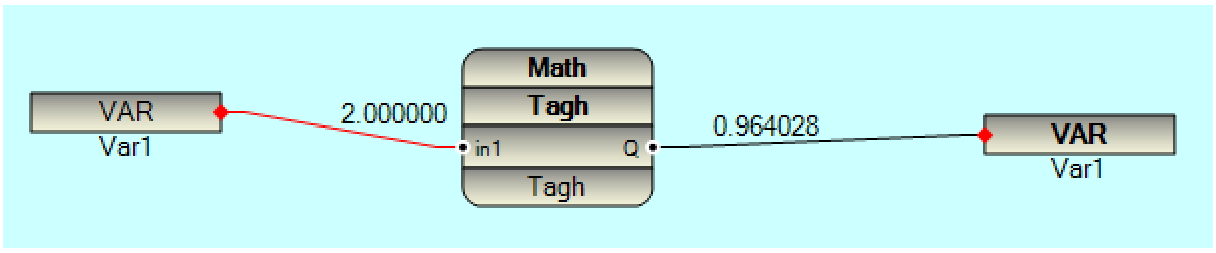
Output Name	Description	Type	Initial Value	Notes
Q	Hyperbolic cosine of in1 (cosh(in1)).	double	0.0	Available for downstream connections. Updates synchronously with input.

1.18.3 Testing:

- in1 = 0; Q = 1 (cosh(0) = 1).
- in1 = 1; Q ≈ 1.5430806348152437.
- in1 = -1; Q ≈ 1.5430806348152437 (symmetric).
- Large input: in1 = 5; Q ≈ 74.20994852478785 (grows exponentially).

1.19 Tagh Function Block

The Tagh function block computes the hyperbolic Tan of an input value, this block takes a single double-precision floating-point input (in1) and produces the result via output Q...



1.19.1 Inputs

The Tagh block accepts one input, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
in1	Input value x for hyperbolic tangent computation.	double	0.0	-

- **Configuration**: Set through direct values, block linkages, or sensor inputs.
- **Validation**: Automatically cast to double; non-numeric inputs may resolve to 0.0 or trigger errors.

1.19.2 Outputs

The Tagh block produces one output, refreshed per execution cycle.

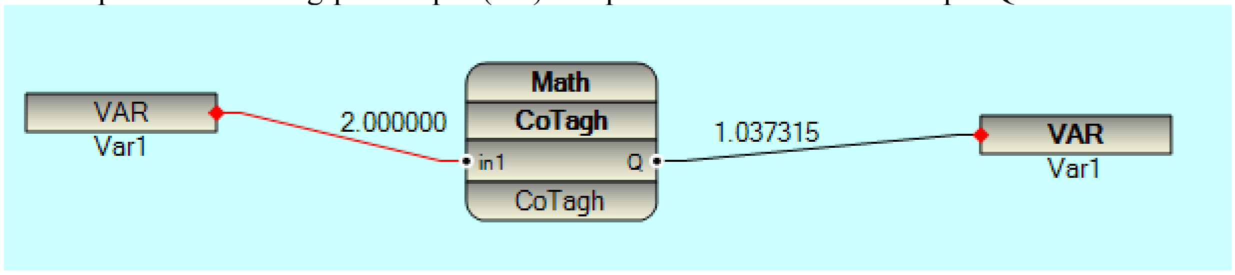
Output Name	Description	Type	Initial Value	Notes
Q	Hyperbolic tangent of in1 (tanh(in1)).	double	0.0	Available for downstream connections. Updates in sync with input.

1.19.3 Testing:

- in1 = 0; Q = 0 (tanh (0) = 0).
- in1 = 1; Q ≈ 0.7615941559557649.
- in1 = -1; Q ≈ -0.7615941559557649.
- High value: in1 = 5; Q ≈ 0.9999092042625951 (nears 1).

1.20 CoTagH Function Block

The CoTagH function block computes the hyperbolic Cot of an input value, this block takes a single double-precision floating-point input (in1) and produces the result via output Q...



1.20.1 Inputs

The CoTagH block accepts one input, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
in1	Input value x for hyperbolic cotangent computation.	double	0.0	-

- **Configuration**: Direct assignment, block connections, or sensor/variable feeds.
- **Validation**: Cast to double; x=0 triggers runtime handling (e.g., skip or error).

1.20.2 Outputs

The CoTagH block produces one output, updated per execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Hyperbolic cotangent of in1 (coth(in1)).	double	0.0	Available for downstream connections. Updates synchronously with input; may be NaN/infinite at x=0.

1.20.3 Testing:

- in1 = 1; Q ≈ 1.313035285499331 (coth(1) > 1).
- in1 = -1; Q ≈ -1.313035285499331.
- Small non-zero: in1 = 0.1; Q ≈ 10.0167084162019 (≈ 1/x for small x).
- Avoid in1 = 0 (undefined; expect NaN).

1.21 ASinH Function Block

The ASinH function block computes the inverse hyperbolic sine of an input value This block takes a single double-precision floating-point input (in1) and outputs the result via Q.

1.21.1 Inputs

The ASinH block accepts one input, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
in1	Input value x for inverse hyperbolic sine computation.	double	0.0	-

1.21.2 Outputs

The ASinH block produces one output, updated on each execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Inverse hyperbolic sine of in1 (asinh(in1)).	double	0.0	Available for downstream connections. Updates synchronously with input.

1.21.3 Testing:

- in1 = 0; Q = 0 (asinh(0) = 0).
- in1 = 1; Q ≈ 0.881373587019543.
- in1 = -1; Q ≈ -0.881373587019543.
- Large input: in1 = 10; Q ≈ 3.1414926535900345

1.22 ACosh Function Block

The ACosh function block computes the inverse hyperbolic Cos of an input value This block takes a single double-precision floating-point input (in1) and outputs the result via Q.

1.22.1 Inputs

The ACosh block accepts one input, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
in1	Input value x for inverse hyperbolic cosine computation.	double	0.0	Connect a signal source here. Supports real-time updates; restrict to >=1 to avoid NaN.

- **Configuration**: Assign directly, link to other blocks, or source from variables/sensors.
- **Validation**: Coerced to double; values <1 trigger domain error handling.

1.22.2 Outputs

The ACosh block produces one output, updated on each execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Inverse hyperbolic cosine of in1 (acosh(in1)).	double	0.0	Available for downstream connections. Updates synchronously with input; NaN for invalid inputs.

1.22.3 Testing**:**

- in1 = 1; Q = 0 (acosh(1) = 0).
- in1 = 2; Q ≈ 1.3169578969248166.
- Invalid: in1 = 0.5; Q = NaN.
- Large: in1 = 10; Q ≈ 2.993222846126381.

1.23 ATagh Function Block

The ATagh function block computes the inverse hyperbolic Tan of an input value This block takes a single double-precision floating-point input (in1) and outputs the result via Q.

1.23.1 Inputs

The ATagh block accepts one input, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
in1	Input value x for inverse hyperbolic tangent computation.	double	0.0	-

- **Configuration**: Set via direct assignment, block linkages, or sensor/variable inputs.
- **Validation**: Cast to double; |in1| >=1 trigger runtime domain handling.

1.23.2 Outputs

The ATagh block produces one output, updated on each execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Inverse hyperbolic tangent of in1 (atanh(in1)).	double	0.0	Available for downstream connections. Updates synchronously with input; NaN for invalid domain.

1.23.3 Testing:

- in1 = 0; Q = 0 (atanh(0) = 0).
- in1 = 0.5; Q ≈ 0.5493061443340548.
- in1 = -0.5; Q ≈ -0.5493061443340548.
- Invalid: in1 = 1; Q = NaN.

1.24 ACoTagh Function Block

The ACoTagh function block computes the inverse hyperbolic Cot of an input value This block takes a single double-precision floating-point input (in1) and outputs the result via Q.

1.24.1 Inputs

The ACoTagh block accepts one input, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
in1	Input value x for inverse hyperbolic cotangent computation.	double	0.0	-

- **Configuration**: Direct assignment, block connections, or sensor feeds.
- **Validation**: Coerced to double; |in1| <=1 leads to runtime domain handling.

1.24.2 Outputs

The ACoTagh block produces one output, updated per execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Inverse hyperbolic cotangent of in1 (Acoth(in1)).	double	0.0	Available for downstream connections. Updates synchronously with input; NaN for invalid inputs.

1.24.3 Testing:

- in1 = 2; Q ≈ 0.5493061443340548
- in1 = -2; Q ≈ -0.5493061443340548.
- Invalid: in1 = 0.5; Q = NaN
- Edge: in1 = 1.1; Q ≈ 3.1415926535897936

1.25 Exp Function Block

The Exp function block computes the exponential function of an input value. This function Defined as:

$$\boldsymbol{Exp(x) = e^{x}}$$

where "**e**" is the base of the natural logarithm (approximately 2.71828), this block takes a single double-precision floating-point input (in1) and produces the result via output Q.

1.25.1 Inputs

The Exp block accepts one input, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
in1	Input value x for exponential computation.	double	0.0	Connect a signal source here. Supports real-time updates; domain all reals, but large

- **Configuration**: Set via direct assignment, linkage to other blocks, or external variables/sensors.
- **Validation**: Input coerced to double; non-numeric may yield 0.0 or exceptions.

1.25.2 Outputs

The Exp block produces one output, updated on each execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Exponential of in1 (Exp(in1)).	double	0.0	Available for downstream connections. Updates synchronously with input.

1.25.3 Functional Behavior

The Exp function block performs a pointwise exponential computation on its input signal. For each time step or sample, it applies the mathematical exponential function to the scalar input value.

1.25.4 Testing:

- in1 = 0; Q = 1.0 (Exp (0) = 1).
- in1 = 1; Q ≈ 2.718281828459045.
- in1 = -1; Q ≈ 0.36787944117144233.
- Large: in1 = 5; Q ≈ 148.4131591025766.

1.26 Exp2 Function Block

The Exp2 function block computes the base-2 exponential of an input value This function Defined as:

$$\boldsymbol{Exp2(x) = 2^{x}}$$

this block takes a single double-precision floating-point input (in1) and produces the result via output Q.

1.26.1 Inputs

The Exp2 block accepts one input, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
in1	Input value x for base-2 exponential computation.	double	0.0	Connect a signal source here. Supports real-time updates; valid for all real x, but extreme values risk overflow/underflow.

- **Configuration**: Direct assignment, block linkages, or sensor/variable drivers.
- **Validation**: Coerced to double; non-numeric inputs may default to 0.0 or error.

1.26.2 Outputs

The Exp2 block produces one output, updated on each execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Base-2 exponential of in1 (exp2(in1)).	double	0.0	Available for downstream connections. Updates synchronously with input.

1.26.3 Testing:

- in1 = 0; Q = 1.0 (exp2(0) = 1).
- in1 = 1; Q = 2.0.
- in1 = 3; Q = 8.0.
- Negative: in1 = -1; Q = 0.5.

1.27 Pow10 Function Block

The Pow10 function block computes the base-10 exponential of an input value, this function Defined as:

$$\boldsymbol{Pow10(x) = 10^{x}}$$

this block accepts a single double-precision floating-point input (in1) and outputs the result via Q.

1.27.1 Inputs

The Pow10 block accepts one input, configurable at runtime.

Input	Description	Type	Initial	Notes

Name			Value	
in1	Input value x for base-10 exponential computation.	double	0.0	Connect a signal source here. Supports real-time updates; all real x valid, but extremes cause overflow/underflow.

- **Configuration**: Direct set, block links, or sensor/variable inputs.
- **Validation**: Coerced to double; non-numeric may result in 0.0 or errors.

1.27.2 Outputs

The Pow10 block produces one output, updated on each execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Base-10 exponential of in1 (Pow10(in1)).	double	0.0	Available for downstream connections. Updates synchronously with input.

1.27.3 Testing:

- in1 = 0; Q = 1.0 (pow10(0) = 1).
- in1 = 2; Q = 100.0.
- in1 = -1; Q = 0.1.
- Large: in1 = 3; Q = 1000.0.

1.28 Log Function Block

The Log function block computes the natural logarithm (base-e) of an input value, this function Defined as:

$$\boldsymbol{Log(x)} = \mathbf{Ln(x)}$$

where e is the base of the natural logarithm (approximately 2.71828), this block takes a single double-precision floating-point input (in1) and produces the result via output Q. The function is undefined for non-positive inputs, typically yielding NaN or -infinite

1.28.1 Inputs

The Log block accepts one input, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
in1	Input value x for natural logarithm computation.	double	0.0	Connect a signal source here. Supports real-time updates; must be >0 to avoid NaN.

- **Configuration**: Assign directly, link to other blocks, or drive with sensors/variables.
- **Validation**: Coerced to double; <=0 triggers domain error handling.

1.28.2 Outputs

The Log block produces one output, updated on each execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Natural logarithm of in1 (ln(in1)).	double	0.0	Available for downstream connections. Updates synchronously with input; NaN for invalid inputs.

1.28.3 Testing:

- in1 = 1; Q = 0 (Ln (1) = 0).
- in1 ≈ 2.71828 (e); Q ≈ 1.
- in1 = 0.5; Q ≈ -0.6931471805599453.
- Invalid: in1 = 0; Q = NaN.

1.29 Log2 Function Block

The Log2 function block computes the base-2 logarithm of an input value, this function Defined as:

$$\boldsymbol{Log2(x) = Log_2^{(x)}}$$

this block takes a single double-precision floating-point input (in1) and outputs the result via output Q. The function is undefined for non-positive inputs, typically resulting in NaN or -infinite.

1.29.1 Inputs

The Log2 block accepts one input, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
in1	Input value x for base-2 logarithm computation.	double	0.0	Connect a signal source here. Supports real-time updates; must be >0 to avoid NaN.

- **Configuration**: Set through direct values, block linkages, or sensor inputs.
- **Validation**: Automatically cast to double; <=0 triggers domain error handling.

1.29.2 Outputs

The Log2 block produces one output, refreshed per execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Base-2 logarithm of in1 (log2(in1)).	double	0.0	Available for downstream connections. Updates in sync with input; NaN for invalid domain.

1.29.3 Testing:

- in1 = 1; Q = 0 (Log2(1) = 0).
- in1 = 2; Q = 1.
- in1 = 8; Q = 3.
- Invalid: in1 = 0; Q = NaN.

1.30 Log10 Function Block

The Log10 function block computes the base-2 logarithm of an input value, this function Defined as:

$$\boldsymbol{Log10(x) = Log_{10}^{(x)}}$$

this block takes a single double-precision floating-point input (in1) and outputs the result via output Q. The function is undefined for non-positive inputs, typically resulting in NaN or -infinite.

1.30.1 Inputs

The Log10 block accepts one input, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
in1	Input value x for base-10 logarithm computation.	double	0.0	Connect a signal source here. Supports real-time updates; must be >0 to avoid NaN.

- **Configuration**: Direct assignment, linkages to other blocks, or sensor/variable feeds.
- **Validation**: Cast to double; <=0 triggers domain error handling.

1.30.2 Outputs

The Log10 block produces one output, updated per execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Base-10 logarithm of in1 (Log10(in1)).	double	0.0	Available for downstream connections. Updates synchronously with input; NaN for invalid inputs.

1.30.3 Testing:

- in1 = 1; Q = 0 (log10(1) = 0).
- in1 = 10; Q = 1.
- in1 = 0.1; Q = -1.
- Invalid: in1 = 0; Q = NaN.

1.31 Sqrt Function Block

The Sqrt function block computes the an input value, this block takes a single double-precision floating-point input (in1) and produces the result via output Q. For negative inputs, the result is typically NaN

1.31.1 Inputs

The Sqrt block accepts one input, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
in1	Input value x for square root computation.	double	0.0	Connect a signal source here. Supports real-time updates; restrict to >=0 to avoid NaN.

- **Configuration**: Direct assignment, block linkages, or sensor/variable inputs.
- **Validation**: Coerced to double; negatives trigger domain error handling.

1.31.2 Outputs

The Sqrt block produces one output, updated on each execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Square root of in1 (sqrt(in1)).	double	0.0	Available for downstream connections. Updates synchronously with input; NaN for invalid inputs.

1.31.3 Testing:

- in1 = 0; Q = 0 (sqrt(0) = 0).
- in1 = 4; Q = 2.0.
- in1 = 9; Q = 3.0.
- Invalid: in1 = -1; Q = NaN.

1.32 Cbrt Function Block

The Cbrt function block computes the real cube root of an input value, this block takes a single double-precision floating-point input (in1) and outputs the result via Q.

$$\mathbf{Cbrt}(\boldsymbol{X}) = \sqrt[3]{\boldsymbol{X}}$$

1.32.1 Inputs

The Cbrt block accepts one input, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
in1	Input value x for cube root computation.	double	0.0	Connect a signal source here. Supports real-time updates; valid for all real numbers, including negatives.

- **Configuration**: Assign directly, link to other blocks, or source from sensors/variables.
- **Validation**: Coerced to double; no domain errors for reals.

1.32.2 Outputs

The Cbrt block produces one output, updated on each execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Cube root of in1 (Cbrt(in1)).	double	0.0	Available for downstream connections. Updates synchronously with input.

1.32.3 Testing:

- in1 = 0; Q = 0 (Cbrt(0) = 0).
- in1 = 8; Q = 2.0.
- in1 = -27; Q = -3.0.
- Non-integer: in1 = 10; Q ≈ 2.154434690031884.

1.33 Hypot Function Block

The Hypot block calculates the length of the hypotenuse in a right-angled triangle.

This function is defined as:

$$\mathbf{Hypot}(\mathbf{X}, \mathbf{Y}) = \sqrt{\mathbf{X}^2 + \mathbf{Y}^2}$$

The block receives two double-precision floating-point inputs (X and Y) and produces a non-negative output through Q..

1.33.1 Inputs

The Hypot block accepts two inputs, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
x	X-coordinate or first leg of the right triangle.	double	0.0	Connect a signal source here. Supports real-time updates; any real value.
y	Y-coordinate or second leg of the right triangle.	double	0.0	Connect a signal source here. Supports real-time updates; any real value.

- **Configuration**: Set via direct assignment, linkages to other blocks, or external variables/sensors.
- **Validation**: Coerced to double; no domain restrictions.

1.33.2 Outputs

The Hypot block produces one output, updated on each execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Euclidean norm (hypotenuse) of (x, y) (hypot(x, y)).	double	0.0	Available for downstream connections. Updates synchronously with inputs.

1.33.3 Testing:

- x = 3, y = 4; Q = 5.0
- x = 0, y = 0; Q = 0.0.
- x = -5, y = 12; Q = 13.0

1.34 Expm1 Function Block

The "Expm1" block calculates the result of the following expression with high precision.

$$\mathbf{Expm1(X)}=e^{X}-1$$

This block is specifically designed for cases where the input value **"X"** is very small, because in such situations, direct computation may cause numerical errors in floating-point systems.

In other words, in normal computation, when the value of **"X"** is very close to zero, the result may, due to limited computational precision, be incorrectly calculated as zero or an inaccurate value.

The **"Expm1"** block performs this calculation using a more precise numerical method and returns the actual result.

The output **"Q"** provides the accurately computed value corresponding to the input **"in1"**.

1.34.1 Inputs

The Expm1 block accepts one input, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
in1	Input value x for expm1 computation.	double	0.0	Connect a signal source here. Supports real-time updates; all reals valid, but small

- **Configuration**: Direct assignment, block linkages, or sensor/variable inputs.
- **Validation**: Coerced to double; non-numeric may default to 0.0 or error.

1.34.2 Outputs

The Expm1 block produces one output, updated on each execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Expm1(X)=$e^X - 1$ precisely computed.	double	0.0	Available for downstream connections. Updates synchronously with input.

1.34.3 Testing:

- in1 = 0; Q = 0 (expm1(0) = 0).
- in1 = 1; Q ≈ 1.718281828459045.
- Small: in1 = 0.001; Q ≈ 0.0010005001667083838 (precise, vs. exp(0.001)-1 ≈ 0.0010005001667090256 with rounding).
- Large: in1 = 5; Q ≈ 147.4131591025766.

1.35 Log1p Function Block

The **Log1p** block computes the natural logarithm of **(1 + x)** with high precision, especially for values where **x** is close to zero, because the ordinary computation of **log(1 + x)** may cause rounding errors due to subtraction cancellation. This function is defined as:

Log1p=Ln(1+x)

This block receives a double-precision floating-point input (in1) and provides the output through Q**.**

1.35.1 Inputs

The Log1p block accepts one input, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
in1	Input value **"X"** for log1p computation.	double	0.0	Connect a signal source here. Supports real-time updates; restrict to > -1 to avoid NaN; precision shines for small x.

- **Configuration**: Set via direct assignment, block linkages, or sensor/variable inputs.
- **Validation**: Coerced to double; x <= -1 triggers domain error handling.

1.35.2 Outputs

The **Log1p** block produces one output, updated on each execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Natural log of (1 + in1) (log1p(in1)), precisely computed.	double	0.0	Available for downstream connections. Updates synchronously with input; NaN for invalid domain.

1.35.3 Testing:

- in1 = 0; Q = 0 (log1p(0) = 0).
- in1 = 1; Q ≈ 0.6931471805599453.
- Small: in1 = 0.001; Q ≈ 0.0009995003330835332 (precise, vs. log(1.001) ≈ 0.000999500333108301 with potential loss).
- Invalid: in1 = -1.1; Q = NaN.

1.36 Erf Function Block

The **Erf** block computes the value of the input's error function (**Erf**), which is a special integral function.

This function is defined as follows:

$$Erf(X) = \frac{2}{\pi} \int_0^X e^{-2t}\, dt$$

Its value approaches ±1 as x → ±∞, and it is an odd function (erf(-x) = -erf(x)).

This block receives a double-precision floating-point input (in1) and provides the output through Q

1.36.1 Inputs

The Erf block accepts one input, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
in1	Input value **"X"** for error function computation.	double	0.0	Connect a signal source here. Supports real-time updates; all real values valid.

- **Configuration**: Direct assignment, block linkages, or sensor/variable feeds.
- **Validation**: Coerced to double; no domain errors.

1.36.2 Outputs

The Erf block produces one output, updated on each execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Error function of in1 (erf(in1)).	double	0.0	Available for downstream connections. Updates synchronously with input; bounded in [-1, 1].

1.36.3 Testing:

- in1 = 0; Q = 0 (Erf (0) = 0).
- in1 = 1; Q ≈ 0.8427007929497149.
- in1 = -1; Q ≈ -0.8427007929497149.
- Large: in1 = 3; Q ≈ 0.9999779095030014 (nears 1).

1.37 Lgamma Function Block

The **Lgamma** function block computes the natural logarithm of the absolute value of the gamma function, calculated as follows:

For whole numbers:	$\Gamma(X)=(X-1)!$
For Fractional numbers:	$\Gamma(X)= \int_0^\infty t^{(X-1)} . e^{-t} dt$

where Γ(x) is the gamma function. This block takes a double-precision floating-point input (in1) and outputs the result via Q. By operating in the logarithmic domain, it prevents overflow issues with large values of Γ(x).

1.37.1 Inputs

The **Lgamma** block accepts one input, configurable at runtime.

Input	Description	Type	Initial	Notes

Name			Value	
in1	Input value "X" for Lgamma computation.	double	0.0	Connect a signal source here. Supports real-time updates; To prevent an infinite error, X must be within the range -1<X<0 OR 0<X.

- **Configuration**: Direct assignment, block linkages, or sensor/variable inputs.
- **Validation**: Coerced to double; Numbers outside the valid range may produce NaN or inf values.

1.37.2 Outputs

The Lgamma block produces one output, updated on each execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Natural log of	Γ(in1)	(lgamma(in1)).	double

1.37.3 Testing:

- in1 = 1; Q = 0 (Γ(1) = 1, log=0).
- in1 = 5; Q ≈ 2.0780860135762397 (log(24) ≈ log(4!)).
- in1 = 0.5; Q ≈ 0.5723649429247001 (Γ(0.5) = √π).
- Invalid: in1 = 0; Q = inf (pole at 0).

1.38 Abs Function Block

The Abs function block computes the absolute value of an input, defined as **"abs(x) = |x|"**. This block takes a double-precision floating-point input (in1) and produces a non-negative result through the output Q.

1.38.1 Inputs

The Abs block accepts one input, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
in1	Input value "X" for absolute value computation.	double	0.0	Connect a signal source here. Supports real-time updates; any real value, including negatives.

- **Configuration**: Direct assignment, block linkages, or sensor/variable inputs.
- **Validation**: Coerced to double; NaN propagates.

1.38.2 Outputs

The Abs block produces one output, updated on each execution cycle.

Output	Description	Type	Initial	Notes

Name			Value	
Q	Absolute value of in1 (abs(in1)).	double	0.0	Available for downstream connections. Updates synchronously with input

1.38.3 Testing:

- in1 = 5.0; Q = 5.0.
- in1 = -3.7; Q = 3.7.
- in1 = 0; Q = 0.0.
- Special: in1 = NaN; Q = NaN.

1.39 Ceil Function Block

The Ceil function block computes the ceiling of an input value and rounds it up to the nearest integer greater than or equal to the input. It is defined as ceil(x). It is preserved as a double-precision floating-point number (double). For exact integers, the input value is returned unchanged.

1.39.1 Inputs

The Ceil block accepts one input, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
in1	Input value "X" for ceiling computation.	double	0.0	Connect a signal source here. Supports real-time updates; any real value valid.

- **Configuration**: Direct assignment, block linkages, or sensor/variable inputs.
- **Validation**: Coerced to double; specials (NaN, inf) propagate.

1.39.2 Outputs

The Ceil block produces one output, updated on each execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Ceiling of in1 (ceil(in1)), as double.	double	0.0	Available for downstream connections. Updates synchronously with input.

1.39.3 Testing:

- in1 = 3.2; Q = 4.0.
- in1 = -3.7; Q = -3.0.

- in1 = 5.0; Q = 5.0.
- Special: in1 = NaN; Q = NaN.

1.40 Floor Function Block

The Floor function block computes the floor of an input value and rounds it down to the nearest integer less than or equal to the input. It is defined as floor(x). It is as a double-precision floating-point number (double). For exact integers, the input value is returned unchanged

1.40.1 Inputs

The Floor block accepts one input, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
in1	Input value "X" for floor computation.	double	0.0	Connect a signal source here. Supports real-time updates; any real value valid.

- **Configuration**: Direct assignment, block linkages, or sensor/variable inputs.
- **Validation**: Coerced to double; specials (NaN, inf) propagate.

1.40.2 Outputs

The Floor block produces one output, updated on each execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Floor of in1 (floor(in1)), as double.	double	0.0	Available for downstream connections. Updates synchronously with input.

1.40.3 Testing:

- in1 = 3.2; Q = 3.0.
- in1 = -3.7; Q = -4.0.
- in1 = 5.0; Q = 5.0.
- Special: in1 = NaN; Q = NaN.

1.41 Trunc Function Block

The Trunc function block performs truncation of the input value toward zero, removing the fractional part and returning the integer portion (the number closer to zero). This function is defined as trunc(x), which returns the integer part of x toward zero (for example, trunc(3.7) = 3.0 and trunc(-3.7) = -3.0). The output is a double-precision floating-point number (double).

1.41.1 Inputs

The **Trunc** block accepts one input, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
in1	Input value "X" for truncation computation.	double	0.0	Connect a signal source here. Supports real-time updates; any real value valid.

- **Configuration**: Direct assignment, block linkages, or sensor/variable inputs.
- **Validation**: Coerced to double; specials (NaN, inf) propagate.

1.41.2 Outputs

The Trunc block produces one output, updated on each execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Truncation of in1 toward zero (trunc(in1)), as double.	double	0.0	Available for downstream connections. Updates synchronously with input.

1.41.3 Testing:

- in1 = 3.7; Q = 3.0.
- in1 = -3.7; Q = -3.0 (Linux); Q = -4.0 (WinCE—note difference).
- in1 = 5.0; Q = 5.0.
- Special: in1 = NaN; Q = NaN.
-

1.42 Rint Function Block

The Rint function block rounds the input value to the nearest integer, following the round-to-even rule in the case of a tie. This function is defined as Rint(x), which returns the integer value nearest to x (for example, Rint(2.5) = 2.0 and Rint(3.5) = 4.0). The output is a double-precision floating-point number (double).

1.42.1 Inputs

The **Rint** block accepts one input, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
in1	Input value "X" for rounding	double	0.0	Connect a signal source here. Supports real-

	computation.			time updates; any real value valid.

- **Configuration**: Direct assignment, block linkages, or sensor/variable inputs.
- **Validation**: Coerced to double; specials (NaN, inf) propagate.

1.42.2 Outputs

The **Rint** block produces one output, updated on each execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Rounded value of in1 to nearest integer (rint(in1)), as double.	double	0.0	Available for downstream connections. Updates synchronously with input.

1.42.3 Testing:

- in1 = 3.2; Q = 3.0.
- in1 = 3.5; Q = 4.0 (ties away from even? Wait, standard rint(3.5)=4.0, but ties-to-even: 3.5 to 4 (even)).
- in1 = -2.5; Q = -2.0 (ties to even).
- Special: in1 = NaN; Q = NaN.

1.43 Round Function Block

The Round function block rounds the input value to the nearest integer; in halfway cases, it rounds values away from zero (for example, Round (2.5) = 3.0 and Round (-2.5) = -3.0).

1.43.1 Inputs

The Round block accepts one input, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
in1	Input value "X" for rounding computation.	double	0.0	Connect a signal source here. Supports real-time updates; any real value valid.

- **Configuration**: Direct assignment, block linkages, or sensor/variable inputs.
- **Validation**: Coerced to double; specials (NaN, inf) propagate.

1.43.2 Outputs

The **Round** block produces one output, updated on each execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Rounded value of in1 to nearest integer (round(in1)), as double.	double	0.0	Available for downstream connections. Updates synchronously with input.

1.43.3 Testing:

- in1 = 3.2; Q = 3.0.
- in1 = 3.5; Q = 4.0
- in1 = -2.5; Q = -3.0
- Special: in1 = NaN; Q = NaN.

1.44 Fmod Function Block

The Fmod function block computes the remainder of the floating-point division of x by y as below:

Fmod(X, Y) = X – (n.Y)

where n is the quotient of the division rounded toward zero. If y = 0, the result will be NaN.

1.44.1 Inputs

The **Fmod** block accepts two inputs, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
x	Dividend (numerator in division).	double	0.0	Connect a signal source here. Supports real-time updates; any real value.
y	Divisor (denominator in division).	double	0.0	Connect a signal source here. Supports real-time updates; must be ≠0 to avoid NaN.

- **Configuration**: Assign directly, link to other blocks, or drive with sensors/variables.
- **Validation**: Coerced to double; y=0 yields NaN.

1.44.2 Outputs

The **Fmod** block produces one output, updated on each execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Floating-point remainder of x / y (fmod(x, y)).	double	0.0	Available for downstream connections. Updates synchronously with inputs; NaN if y=0.

1.44.3 Testing:

- x = 10.5, y = 3.0; Q = 1.5 (10.5 - 3*3 = 1.5).
- x = -10.5, y = 3.0; Q = -1.5 (sign of x).
- x = 7.0, y = 0; Q = NaN.
- Edge: x = 0, y = 5.0; Q = 0.0.

1.45 Remainder Function Block

The Remainder function computes the remainder of the division of X by Y. This function rounds the quotient of the division to the nearest integer and then returns the difference between X and this rounded multiple of Y. This function is defined as:

Remainder (X, Y) = X – [Round (X / Y). Y]

The sign of the remainder is the same as the sign of "X".

1.45.1 Inputs

The Remainder block accepts two inputs, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
x	Dividend (numerator in division).	double	0.0	Connect a signal source here. Supports real-time updates; any real value.
y	Divisor (denominator in division).	double	0.0	Connect a signal source here. Supports real-time updates; must be ≠0 to avoid NaN.

- **Configuration**: Assign directly, link to other blocks, or drive with sensors/variables.
- **Validation**: Coerced to double; y=0 yields NaN.

1.45.2 Outputs

The Remainder block produces one output, updated on each execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	remainder of x / y (Remainder(x, y)).	double	0.0	Available for downstream connections. Updates synchronously with inputs; NaN if y=0.

1.46 SubTract Function Block

The SubTract function block performs the subtraction operation. This block subtracts the second input (in2) from the

first input (in1) and produces the result through the output Q
(i.e., Q = in1 - in2).

1.46.1 Inputs

The **SubTract** block accepts two inputs, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
in1	Minuend (first operand for subtraction)	Any (numeric recommended)	0.0	Connect a signal source here. Supports real-time updates.
in2	Subtrahend (value to subtract from in1)	Any (numeric recommended)	0.0	Connect a signal source here. Supports real-time updates.

- **Configuration**: Inputs can be assigned directly, linked to other blocks, or sourced from sensors/variables.
- **Validation**: Ensure numeric types; non-numeric may coerce to 0.0 or raise exceptions.

1.46.2 Outputs

The SubTract block produces one output, updated on each execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Result of subtraction (in1 - in2).	Any (numeric)	0.0	Available for downstream connections. Updates synchronously with inputs.

1.46.3 ting:

- Set in1 = 10.0 and in2 = 4.0; verify Q = 6.0.
- Edge cases: Negatives (in1 = 5.0, in2 = -3.0 → Q = 8.0), zeros (Q = 0.0), or large differences (monitor for overflow).

2 Timer Group

2.1 OnTimer Function Block

The OnTimer function block is an on-delay timer. When the Trg input is activated, the block starts an internal countdown based on the input Time. After the specified time has elapsed, the output Q becomes 1 and remains 1 as long as the Trg input is active. Additionally, ETime shows the elapsed time from the activation of Trg until the output turns on. As soon as the Trg input is deactivated, both Q and ETime reset to 0.

2.1.1 Inputs

The OnTimer block accepts two inputs, configurable at runtime.

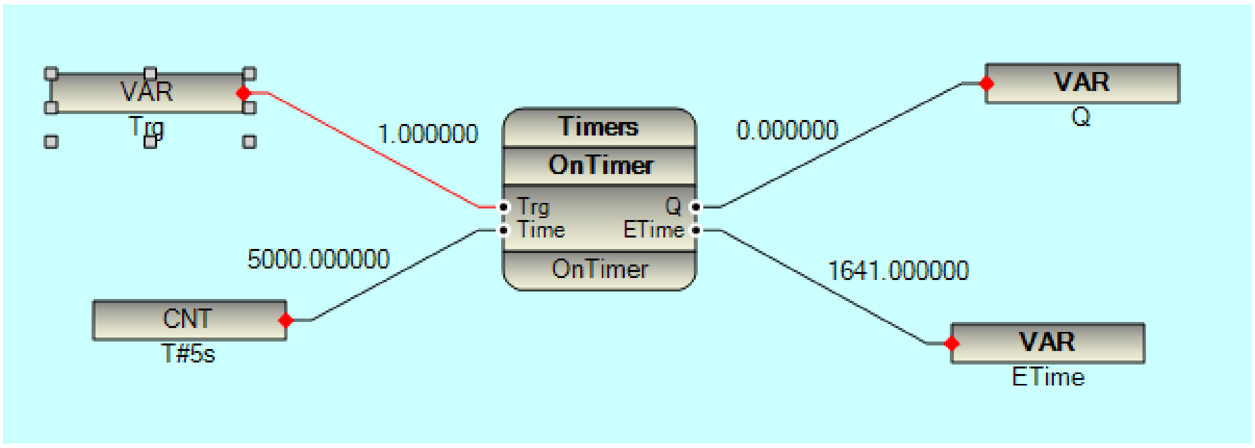
Input Name	Description	Type	Initial Value	Notes
Trg	Trigger signal to start the timer	Digital	False	Connect a Boolean source (e.g., a button or sensor); the input must remain active for the block to operate.
Time	Delay duration before Q activates (in ms; e.g., T#10s = 10000 ms).	Time	T#10s	Set via constants or variables; adjustable at runtime for flexibility.

- **Configuration**: Wire Trg to event sources; set Time statically or dynamically.
- **Validation**: Trg as bool; Time as positive duration (zero may cause immediate trigger).

2.1.2 Outputs

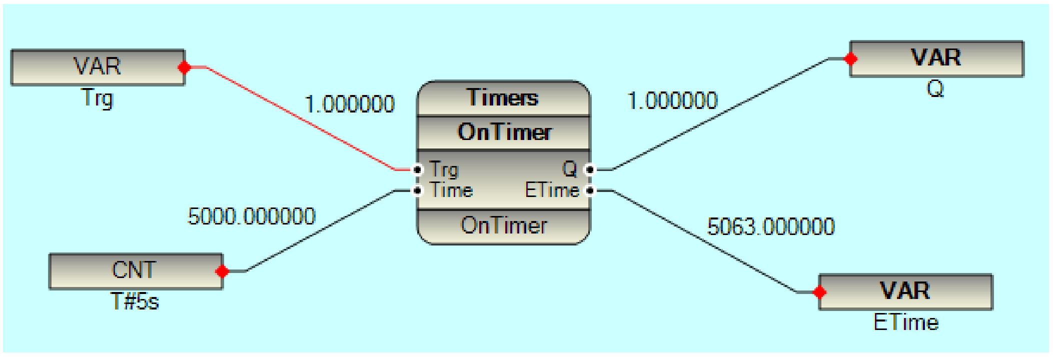
The OnTimer block produces two outputs, updated per execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Timed output (sets to 1 after delay elapses, resets on next trigger or timeout).	Digital	False	Connect to actuators or downstream logic
ETime	Elapsed time since last trigger (in ms, resets on new trigger).	Time	T#0s	Monitor for progress; max value typically matches Time input.



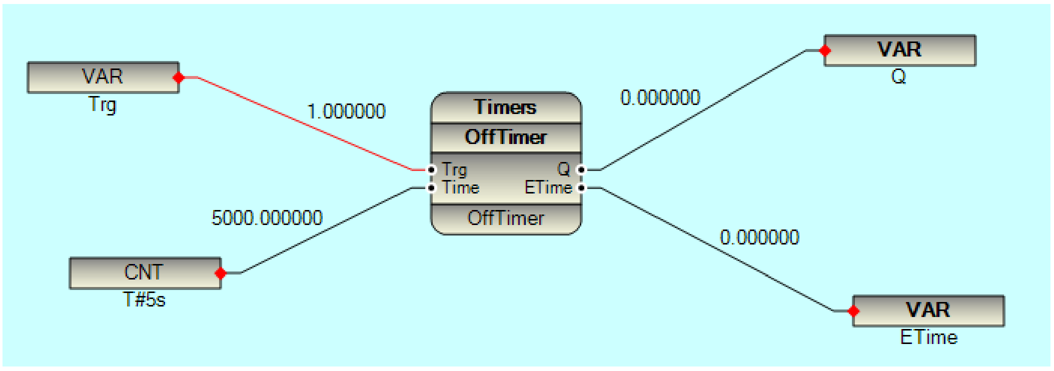
2.1.3 Testing:

• Set Time to T#5s (5000 milliseconds); change the Trg input from False to True; you will observe that Q becomes 1 after 5 seconds and ETime reaches T#5s.
• Restarting during the countdown: the timer restarts, and ETime resets to 0.
• Boundary condition: if Trg remains True continuously, no further action occurs until Trg is turned off and then on again.



2.2 OffTimer Function Block

The **OffTimer** function block is an off-delay timer. Its operation is such that when the **Trg** input is 1, the **Q** output also becomes 1. On the falling edge of **Trg**, the block starts an internal countdown based on the **Time** input. During this delay, the Q output remains 1, and after the specified time has elapsed, the output switches to 0. The **ETime** output shows the elapsed time from the falling edge in seconds.



2.2.1 Inputs

The OffTimer block accepts two inputs, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
Trg	Trigger signal; Q follows Trg when high, falling edge (1→0) initiates delay.	Digital	False	Connect a boolean source (e.g., switch, sensor).
Time	Delay duration after falling edge before Q deactivates (in ms; e.g., T#2s = 2000 ms).	Time	T#2s	Adjustable at runtime; zero causes immediate off.

- **Configuration**: Connect Trg to control signals; set Time via constants or variables.
- **Validation**: Trg as bool; Time as non-negative duration.

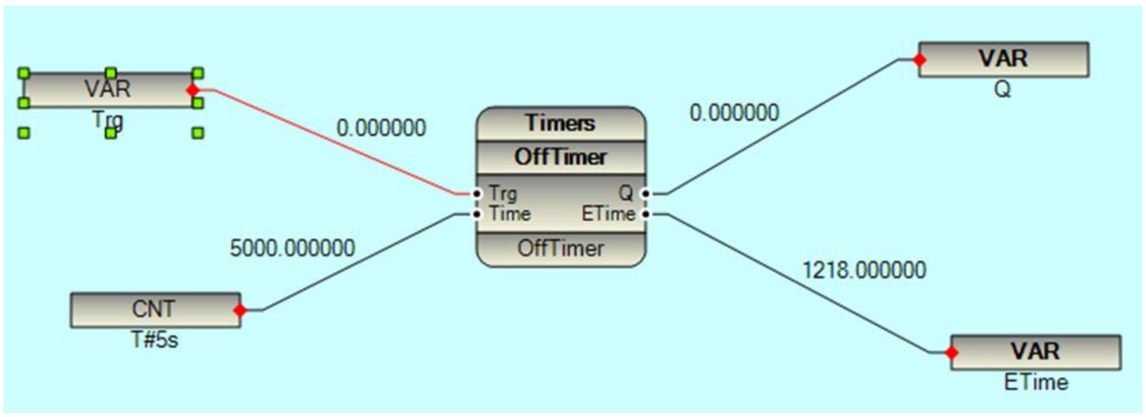
2.2.2 Outputs

The OffTimer block produces two outputs, updated per execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Delayed off output (1 while Trg high or during delay; 0 after timeout).	Digital	False	-
ETime	Elapsed time since last falling edge (in S, resets on rising edge).	Time	T#0s	For monitoring; caps at Time value during countdown.

2.2.3 Testing:

- Set Time = T#5s (5000 ms); assert Trg=True → Q=True immediate; Trg=False → Q stays True for ~5s, then False; ETime increments to ~T#5s.



2.3 PulseGen Function Block

The **PulseGen** function block is a pulse generator that is used to generate square pulses. By detecting the rising edge at the **Trg** input terminal, pulses with a frequency proportional to the time value set at the **Time** input are generated. An important point in this block is that the duration of each of the pulse's on (1) and off (0) states is equal to the time value defined at the **Time** input.

2.3.1 Inputs

The PulseGen block accepts two inputs, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
Trg	Trigger signal to start/stop pulsing	Digital	False	Connect a Boolean source (e.g., switch, event, logic…).
Time	Pulse period (toggle interval for Q).	Time	T#1s	Adjustable duration; e.g., T#500ms for 2 Hz. Zero may cause immediate toggle.

- **Configuration**: Wire Trg to control signals; set Time via constants or variables.
- **Validation**: Trg as bool; Time as positive duration.

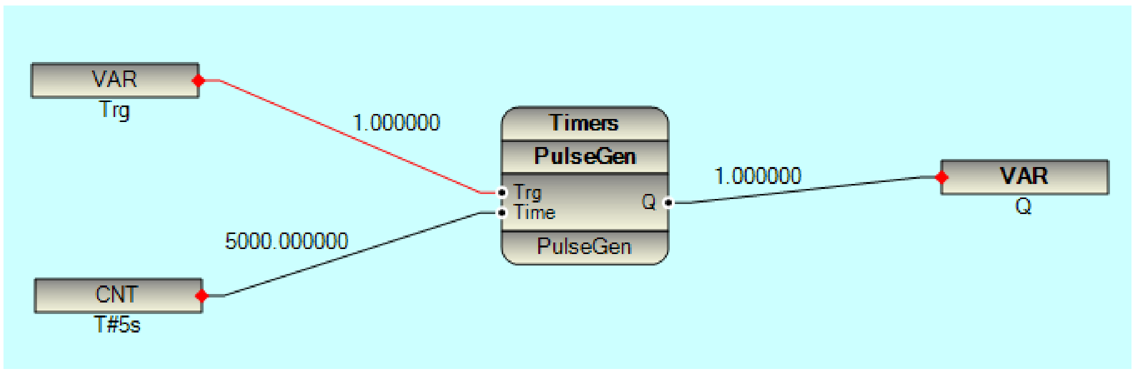
2.3.2 Outputs

The PulseGen block produces one output, updated per execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Pulsing digital output	Digital	False	-

2.3.3 Testing

- Set Time = T#5s; assert Trg=False → Q=False; toggle Trg True → Q toggles every ~5s; Trg False → Q=False.



2.4 PulseGen2 Function Block

The **PulseGen2** function block is an asymmetric pulse generator designed to produce a digital output (Q) with adjustable durations for the on (HTime) and off (LTime) states. Upon detecting a rising edge at the Trg input, pulse generation begins; Q stays at 1 for the duration of HTime and then at 0 for the duration of LTime. This cycle continues as long as Trg remains at 1. When a falling edge of Trg occurs, pulse generation stops, and the Q output returns to the low (0) state.

2.4.1 Inputs

The PulseGen2 block accepts three inputs, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
Trg	Trigger signal to start/stop pulsing (rising edge starts, falling edge stops).	Digital	False	Connect a Boolean source (e.g., switch, event). While 1, sustains pulsing.
HTime	High pulse duration (time Q stays 1 per cycle).	Time	T#1s	Adjustable; e.g., T#500ms for high phase. Zero may cause immediate low.
LTime	Low pulse duration (time Q stays 0 per cycle).	Time	T#2s	Adjustable; e.g., T#1000ms for low phase. Zero may cause immediate high.

- **Configuration**: Wire Trg to control signals; set HTime/LTime via constants or variables.
- **Validation**: Trg as bool; HTime/LTime as positive durations.

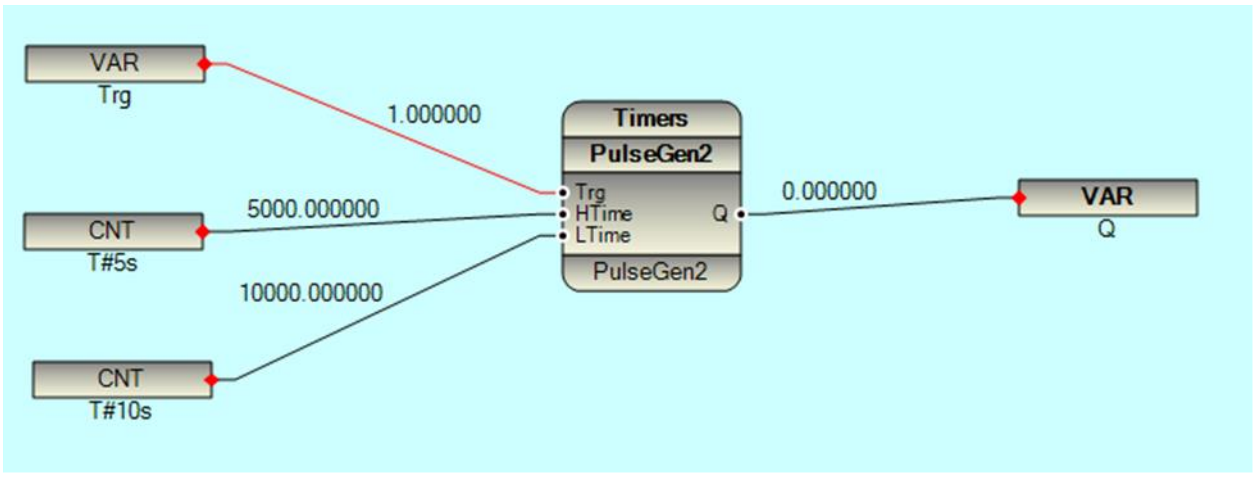
2.4.2 Outputs

The PulseGen2 block produces one output, updated per execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Asymmetric pulsing digital output (1 for HTime, 0 for LTime while Trg high; when Trg is off).	Digital	False	-

2.4.3 Testing

- Set HTime=T#5s, LTime=T#10s; Trg=False → Q=False; Trg True → Q high 5s, low 10s, repeat; Trg False → Q=False.
- Restart: Pulser resumes from file state.
- Edge: Equal HTime/LTime for symmetric; short times for fast cycles.



2.5 RampGen Function Block

The **RampGen** function block is a trigger-driven linear ramp generator designed to produce an output (Q). When the trigger input (Trg) is activated, the output increases from 0 to its maximum value over the time period specified by the Time input. Upon reaching the maximum value, it resets to 0. This cycle repeats as long as Trg remains at 1. The ramp slope is determined by the Ramp parameter, computed as:

$$\textbf{Ramp slope} = \textbf{Tan}\left(\frac{\boldsymbol{2\pi . Ramp}}{\boldsymbol{360}}\right)$$

Not that interpreting Ramp as degrees for angular scaling. On a rising edge of Trg (0 to 1), it starts ramping from 0; when the elapsed time reaches Time, it resets Q to 0 and restarts the ramp. On a falling edge (1 to 0), it stops ramping and sets Q to 0.

2.5.1 Inputs

The RampGen block accepts three inputs, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
Trg	Trigger signal to start/stop ramping (rising edge starts, falling edge stops/resets).	Digital	False	When in state 1, the ramp cycles continue.
Time	Ramp duration per cycle (ms; time to peak before reset to 0).	Time	T#1s	Zero may cause immediate reset.
Ramp	Slope factor	int	45	Limit 0-89° to avoid tan(inf).

- **Configuration**: Wire Trg to control signals; set Time/Ramp via constants or variables.
- **Validation**: Trg as bool; Time as positive duration; Ramp as int (0-360 recommended).

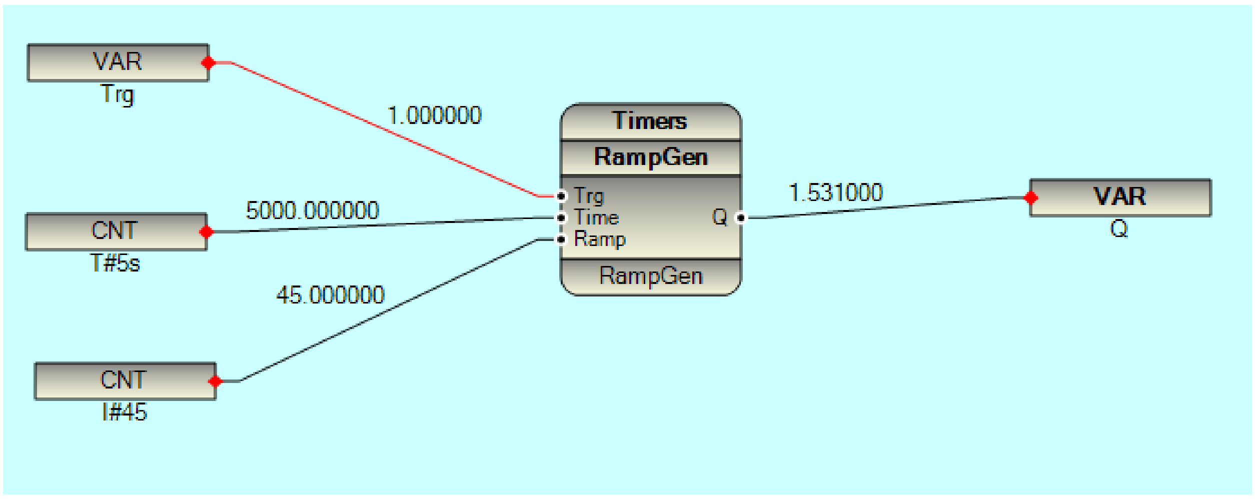
2.5.2 Outputs

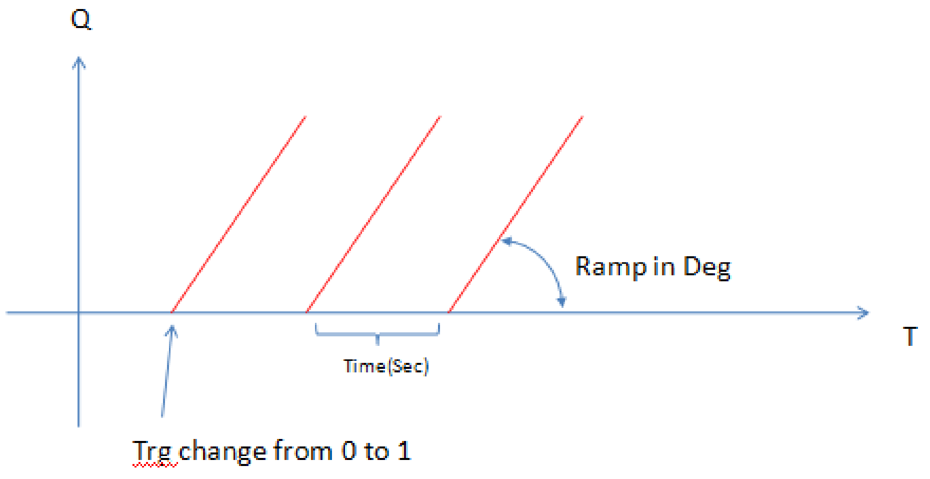
The RampGen block produces one output, updated per execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Sawtooth ramp output (linear from 0 to peak over Time, resets to 0; 0 when off).	float	0.0	-

2.5.3 Testing

- Set Time=T#5000ms, Ramp=45 (slope=1); Trg=False → Q=0; Trg True → Q ramps 0 to 5 over 5s, resets to 0, repeat; Trg False → Q=0.





2.6 RampGen2 Description

The RampGen2 function block is a triangular wave generator that produces an isosceles triangle at the output

(Q). When the trigger input (Trg) is activated, the output increases from 0 to its maximum value over the time interval defined by the Time input. Upon reaching the maximum value, it returns to 0 with the same slope in reverse. This cycle continues as long as Trg remains at 1.

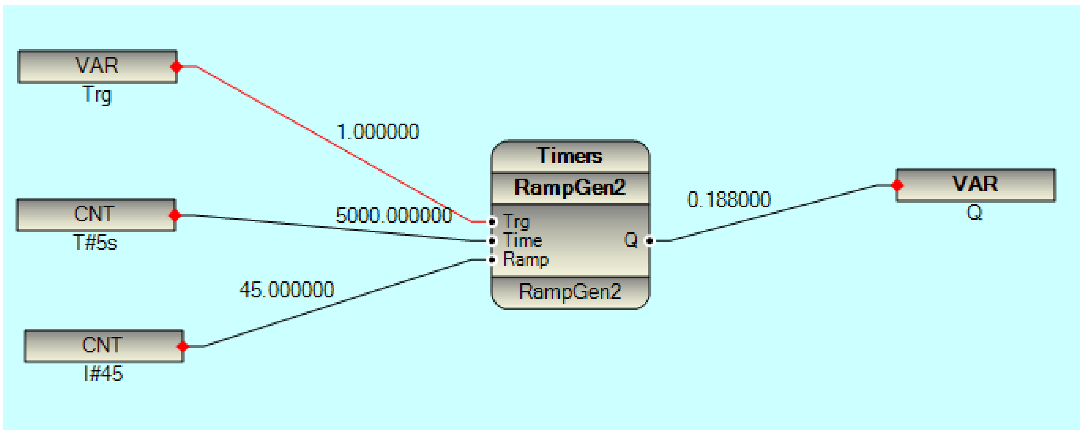
The ramp slopes (in the increasing or decreasing direction) are determined by the Ramp parameter and are calculated according to the following relation:

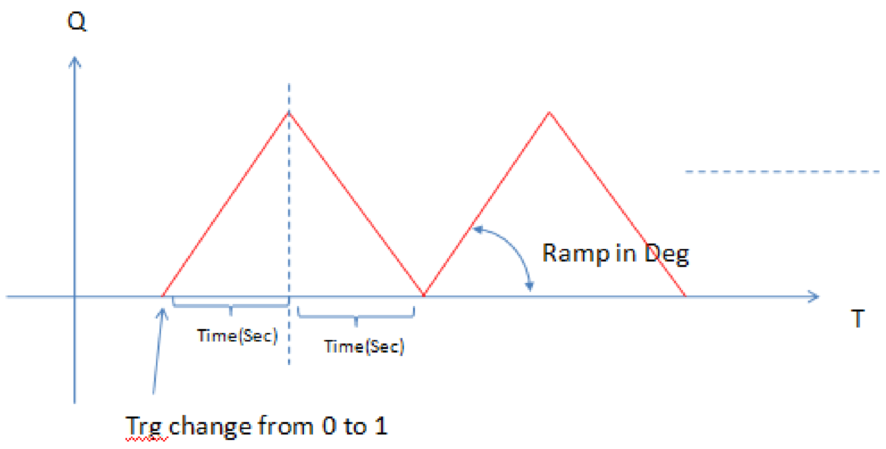
$$\textbf{Ramp slope1} = \textbf{Tan}\left(\frac{2\pi\,.\,Ramp}{360}\right)$$

$$\textbf{Ramp slope2} = -\textbf{Tan}\left(\frac{2\pi\,.\,Ramp}{360}\right)$$

Note that Ramp is interpreted in degrees to apply the angular scale.

On the rising edge of the Trg input (change from 0 to 1), the ramp starts from 0; when the elapsed time reaches the value of Time, the value of Q returns to 0 with the inverse of the same slope and the same Time duration, and the ramp starts again. On the falling edge (change from 1 to 0), the ramps stop and Q is set to





2.7 PulseTimer Function Block

The Pulse Timer function block is a single-pulse timer that is activated by a rising edge at the input (In) and keeps the output (Q) at 1 for the time duration specified by the Time parameter, then automatically returns to 0.

When a rising edge occurs at the **In** input, the timer starts, sets the Q output to 1, and maintains it until the specified time elapses; then, after the time ends, the Q output changes to 0.
If the **In** input returns to 0 before the time expires, the Q output remains at 1 until the end of the timing period (no reset occurs on the falling edge).

2.7.1 Inputs

The PulseTimer block accepts two inputs, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
In	Trigger input (rising edge starts timer and sets Q 1).	bool	0	Connect a digital source (e.g., switch, sensor). Falling edge does not reset timer.
Time	Timer duration (ms; time Q stays 1 before auto-reset).	Time	T#10s	Adjustable; e.g., T#5s for 5-second pulse. Zero may cause immediate timeout.

- **Configuration**: Wire In to event signals; set Time via constants or variables.
- **Validation**: In as bool (0/1); Time as positive duration.

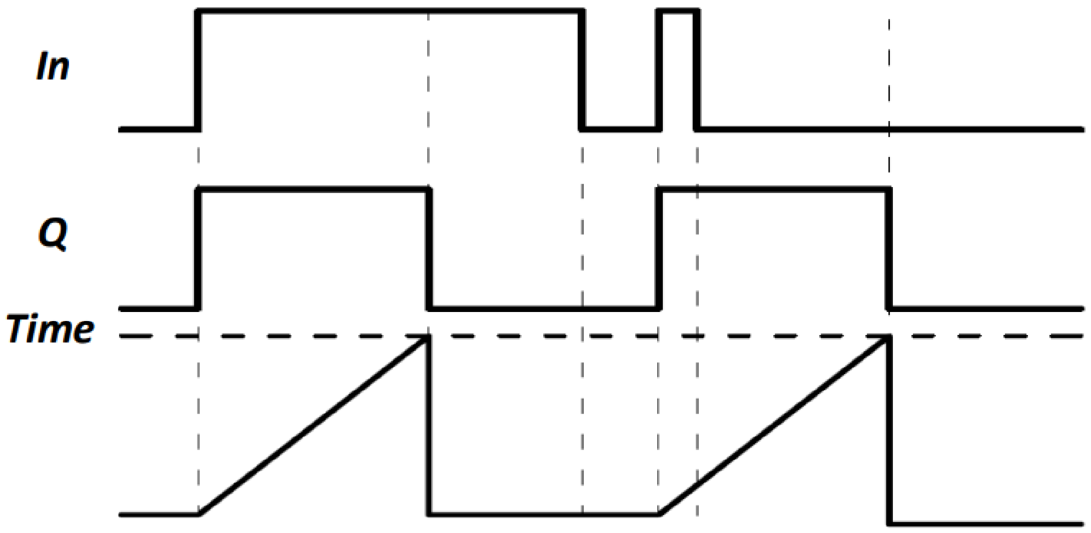
2.7.2 Outputs

The PulseTimer block produces one output, updated per execution cycle.

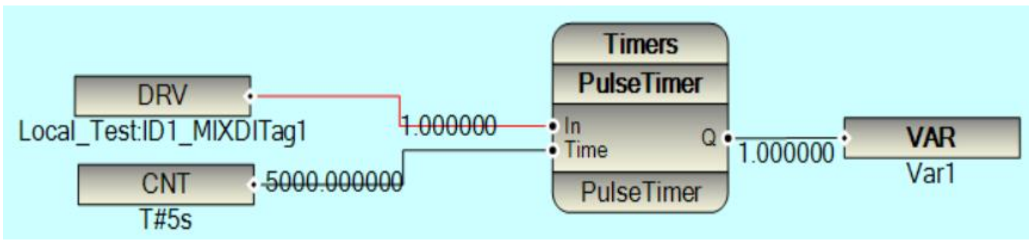
Output Name	Description	Type	Initial Value	Notes
Q	Timed pulse output (1 during timer run; 0 after timeout or off).	bool	False	Connect to actuators (e.g., relay); one-shot pulse per trigger.

2.7.3 Testing:

- Set Time=T#5s; In=0 → Q=0; toggle In=1 (rising) → Q=1 for ~5s, then Q=0; In=0 during run → Q stays 1 until end.
- Restart: Timer resumes from file state (continues if running).
- Edge: Short Time (T#1s) for quick pulse; multiple triggers ignored until timeout.



•



2.8 GetTime Function Block

The **GetTime** function block retrieves the system's current local time and provides it as separate components (year, month, day, hour, minute, and second). This block is updated periodically while active and serves as a time sampler for remote terminal units (RTUs).

When **Enable** is 1, the block fetches the time from the system at intervals set by the **Period** input (in Second). If **Enable** is 0, all outputs are set to 0.Upon startup, the block loads the previously stored state to maintain continuity across restarts.

2.8.1 Inputs

The GetTime block accepts two inputs, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
Enable	Enable time fetching (1=on, updates every Period; 0=off, outputs=0).	int	1	Connect a digital source (0/1); runtime changeable.
Period	Update interval in seconds (e.g., 60 for 1	int	60	Adjustable; min 1 to avoid excessive

	min).			updates. Zero treated as 1.

- **Configuration**: Set Enable via switches/variables; Period via constants.
- **Validation**: Enable 0/1; Period positive integer.

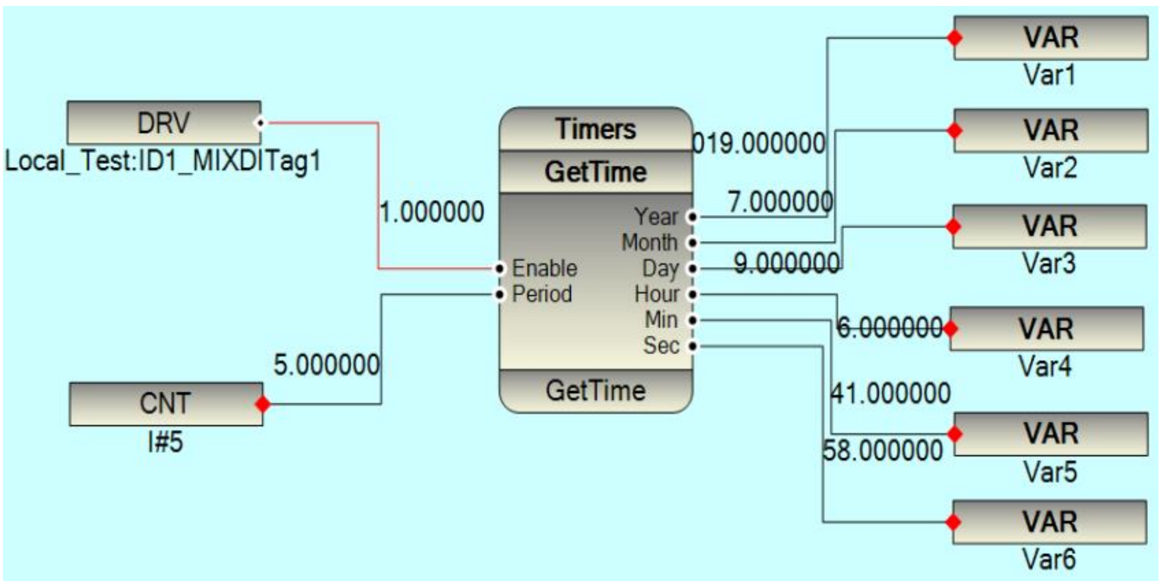
2.8.2 Outputs

The GetTime block produces six outputs, updated per execution cycle if enabled and due.

Output Name	Description	Type	Initial Value	Notes
Year	Current year (e.g., 2025).	int	0	0 when disabled.
Month	Current month (1-12).	int	0	0 when disabled.
Day	Current day of month (1-31).	int	0	0 when disabled.
Hour	Current hour (0-23).	int	0	0 when disabled.
Min	Current minute (0-59).	int	0	0 when disabled.
Sec	Current second (0-59).	int	0	0 when disabled.

2.8.3 Testing:

- Enable=1, Period=5; outputs current time; wait >5s → updates; Enable=0 → all 0.
- Edge: Period=1 for frequent; Enable toggle mid-period holds values.



2.9 SinGen Function Block

The SinGen function block is a sine wave generator that represents a sinusoidal signal with an adjustable frequency (Frq) in hertz and is activated when the trigger input (Trg) is enabled.Upon detecting a rising edge of Trg, the phase is reset to 0, and the output generation begins according to the formula:

Q=Sin (π . Frq . Elapsed time)

where elapsed_time is the time passed since the start in seconds. The signal continues as long as Trg remains at 1. When a falling edge of Trg occurs, signal generation stops, and the Q output is set to 0.The internal states of the block (such as the previous input, start time, and current phase/value) are stored in a configuration file to maintain operational stability across restarts. The storage path is derived from block parameters, such as the program ID or PID.The generated sine wave ranges from -1 to 1, and its period equals 2 / Frq seconds (due to the π factor; for the standard 2π period, double the frequency 1/Frq).

2.9.1 Inputs

The SinGen block accepts two inputs, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
Trg	Trigger signal to start/stop sine generation (rising edge starts/resets phase, falling edge stops).	Digital	False	Connect a boolean source (e.g., switch, event). While high, sustains sine output.
Frq	Frequency in Hz (controls sine wave rate).	int	10	Adjustable; e.g., 5 for slower wave. Zero yields constant 0. Positive integers recommended.

- **Configuration**: Wire Trg to control signals; set Frq via constants or variables.
- **Validation**: Trg as bool; Frq as positive int.

2.9.2 Outputs

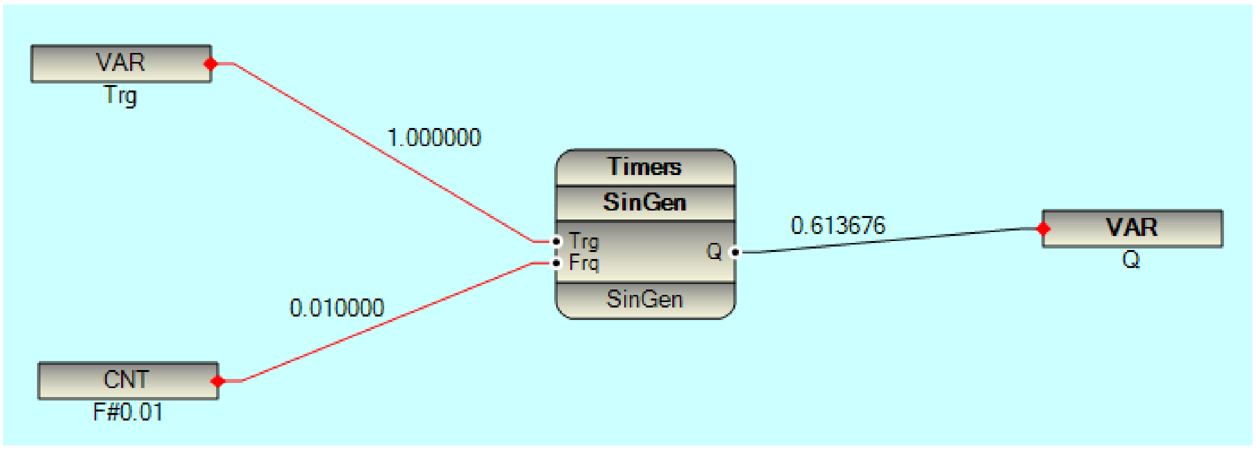
The SinGen block produces one output, updated per execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Sine wave output (sin(π * Frq * elapsed_sec); 0 when off).	float	0.0	Connect to analog outputs (e.g., DAC for waveform); ranges -1 to 1.

2.9.3 Functional Behavior

The SinGen function block (FB) generates a sine wave output. The behavior is as follows:

- **Trigger Activation**: When the trigger input (Trg) transitions from 0 to 1, the function block starts generating the sine output.
- **Frequency Parameter**: The Frq parameter specifies the frequency of the sine wave.



2.9.4 Testing:

- Set Frq=1; Trg=False → Q=0; Trg True → Q oscillates sin(π * 1 * t), period 2s (-1 to 1); Trg False → Q=0.
- Restart: Sine resumes from file state (phase continuous).
- Edge: Frq=0 → Q=0; high Frq (e.g., 50) for fast oscillation.

2.10 ChTimer Function Block

The ChTimer function block is a change-detection timer that activates whenever the input value (In), of type floating-point, changes. Upon detecting any variation in In, it sets the output Q to 1 for the duration specified by the Time parameter, after which it automatically resets Q to 0.This block operates based on value changes rather than signal edges. Whenever In changes (i.e., differs from its previous value), the timer starts, sets Q = 1, and keeps it high until the specified time elapses. After the time period ends, Q returns to 0.If the input changes again during this timing period, the timer restarts and begins counting the time from zero.

2.10.1 Inputs

The ChTimer block accepts two inputs, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
In	Input value (float; any change triggers timer).	float	0.0	Connect an analog source (e.g., sensor reading). Retriggers on new value.
Time	Timer duration (ms; time Q stays high before auto-reset).	Time	T#10s	Adjustable; e.g., T#5s for 5-second pulse. Zero may cause immediate timeout.

- **Configuration**: Wire In to analog signals; set Time via constants or variables.
- **Validation**: In as float; Time as positive duration.

2.10.2 Outputs

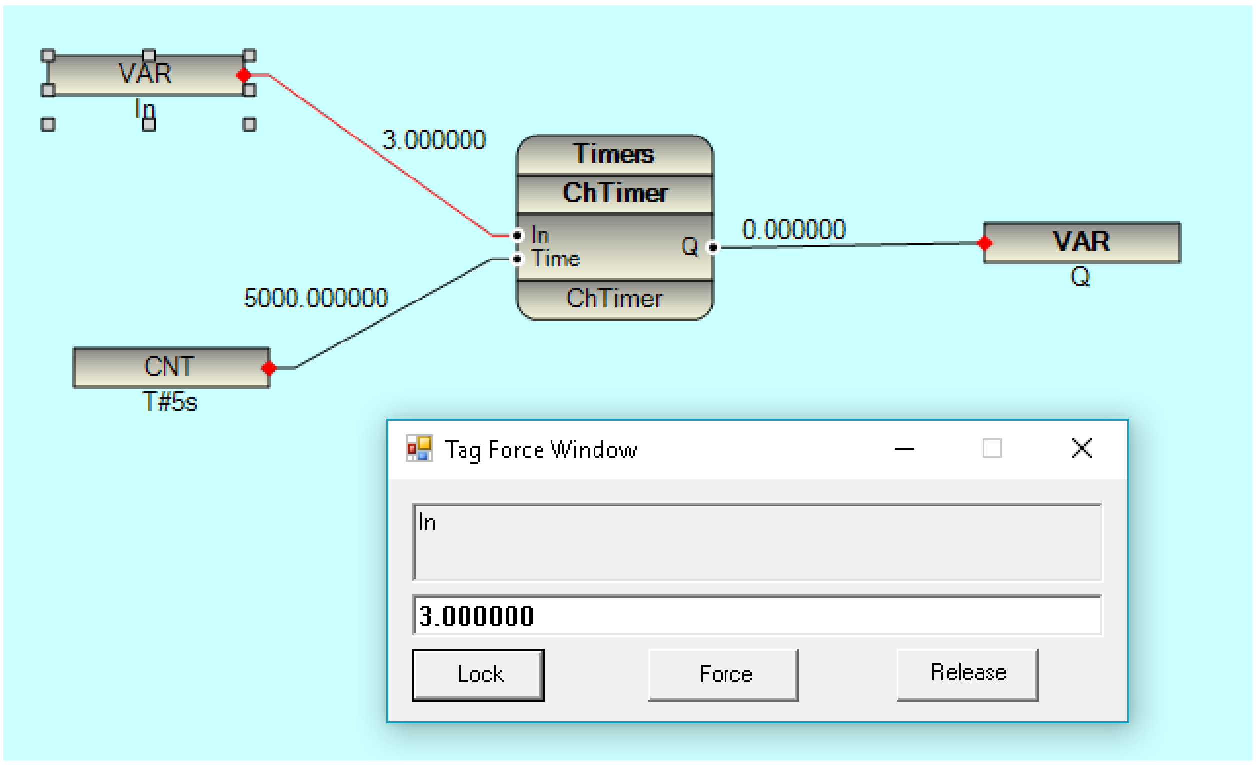
The ChTimer block produces one output, updated per execution cycle.

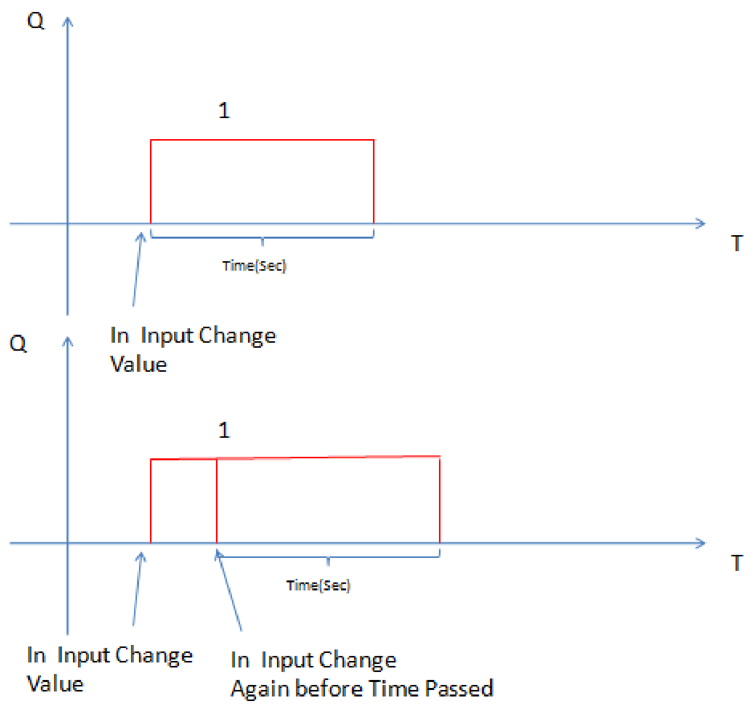
Output Name	Description	Type	Initial Value	Notes
Q	Timed pulse output (high during timer run; low after timeout).	bool	False	Connect to digital actuators (e.g., relay); pulse per input change.

2.10.3 Functional Behavior:

The ChTimer (Change Timer) function block operates as follows:

- **Input Change Detection**: When the input signal (In) changes (e.g., from 0 to 3), the output (Q) is set to 1 for the duration specified by the Time parameter.
- **Output Behavior**: The output (Q) remains set to 1 for the specified time (in seconds) and then resets to 0.
- **Example**: If the input signal changes from 0 to 3, the output (Q) is set to 1 for 5 seconds, after which it falls back to 0.





2.10.4 Testing:

- Set Time=T#10s; In=0 → Q=0; change In=1.0 → Q=1 for 10s, then Q=0; change In=2.0 during → restarts timer.

2.11 DelayTimer Function Block

The **DelayTimer** function block is a delay-pulse timer with a hold mode, designed to produce a digital output (**Q**) with an initial delay followed by a high-level pulse. Upon detecting a rising edge, after the time specified in the **Delay** input elapses, the output **Q** is set to **1** and remains high for the duration defined in the **HTime** input, after which it resets to **0**. If **Trg** changes from **1** to **0** during either the delay or pulse phase, the timer immediately resets and **Q** is set to **0**.

2.11.1 Inputs

The DelayTimer block accepts three inputs, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
Trg	Trigger signal (rising edge starts delay-pulse sequence; falling edge resets).	bool	0	Connect a digital source (e.g., switch, sensor). Resets during delay or pulse.
Delay	Initial delay duration (ms; time Q stays low before high pulse).	Time	T#5s	Adjustable; e.g., T#2s for shorter delay. Zero may skip delay.
HTime	High pulse duration (ms; time Q stays high after delay).	Time	T#10s	Adjustable; e.g., T#3s for shorter pulse. Zero may skip pulse.

- **Configuration**: Wire Trg to event signals; set Delay/HTime via constants or variables.
- **Validation**: Trg as bool (0/1); Delay/HTime as positive durations.

2.11.2 Outputs

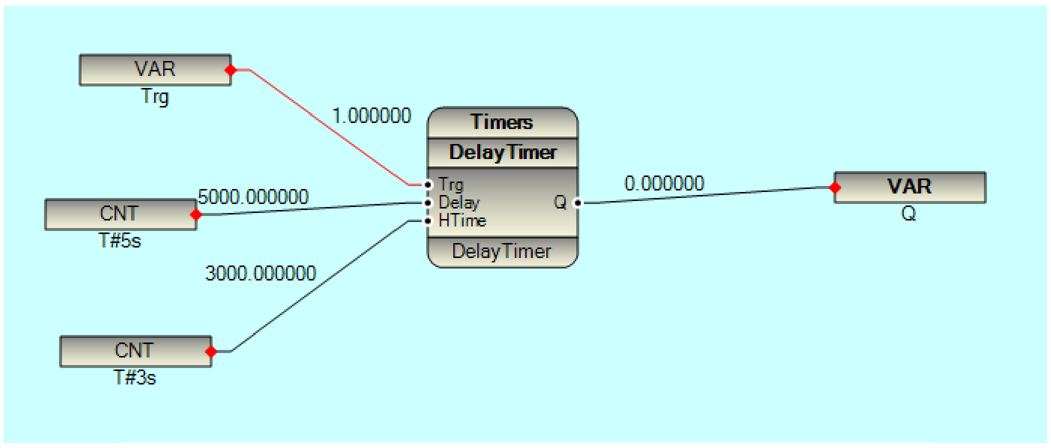
The DelayTimer block produces one output, updated per execution cycle.

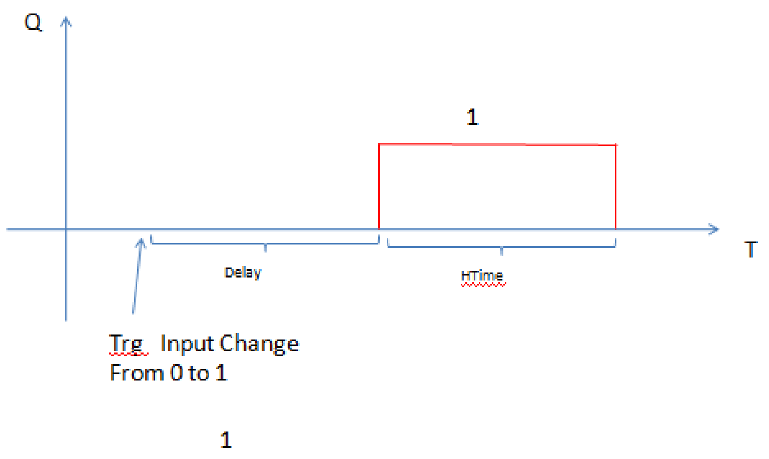
Output Name	Description	Type	Initial Value	Notes
Q	Delayed pulse output (0 during delay, 1 during HTime, low after; 0 on reset).	bool	False	-

2.11.3 Functional Behavior

The DelayTimer function block operates as follows:

- **Trigger Activation**: When the trigger input (Trg) transitions from 0 to 1, the timer starts.
- **Delay Phase**: After the specified Delay Time (in milliseconds) elapses, the output (Q) changes to 1.
- **Output Duration**: The output (Q) remains at 1 for the duration specified by HTime (in milliseconds), then resets to 0.





2.11.4 Testing:

- Set Delay=T#5s, HTime=T#10s; Trg=0 → Q=0; Trg=1 → Q=0 for 5s, Q=1 for 10s, Q=0; Trg=0 during → immediate Q=0.

3 Counter Group

3.1 UpCounter Function Block

The UpCounter function block is an up-counter designed to increment its count on each rising edge of the trigger input (Trg). This block outputs the current count value on OutCnt and has a Boolean output Q, which is set whenever the count exceeds the UpLimit.The block also supports an initial value (InitCount) and reset via the rising edge of the Rst input. Activating the reset returns the count to InitCount and clears the Q output.

3.1.1 Inputs

The UpCounter block accepts four inputs, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
Trg	Trigger input (rising edge increments count).	Digital	False	Connect a pulse source (e.g., sensor event). Debounce if noisy.
Rst	Reset input to InitCount	Digital	False	To reset block to initial state
UpLimit	Maximum count threshold	Int	10	Adjustable; e.g., 100 for higher limit. Non-negative.
InitCount	Initial/reset count value.	Int	0	Starting value; e.g., 5 to begin at 5. Non-negative.

- **Configuration**: Wire Trg/Rst to digital inputs; set UpLimit/InitCount via constants or variables.
- **Validation**: Trg/Rst as bool; UpLimit/InitCount as non-negative ints.

3.1.2 Outputs

The UpCounter block produces two outputs, updated per execution cycle.

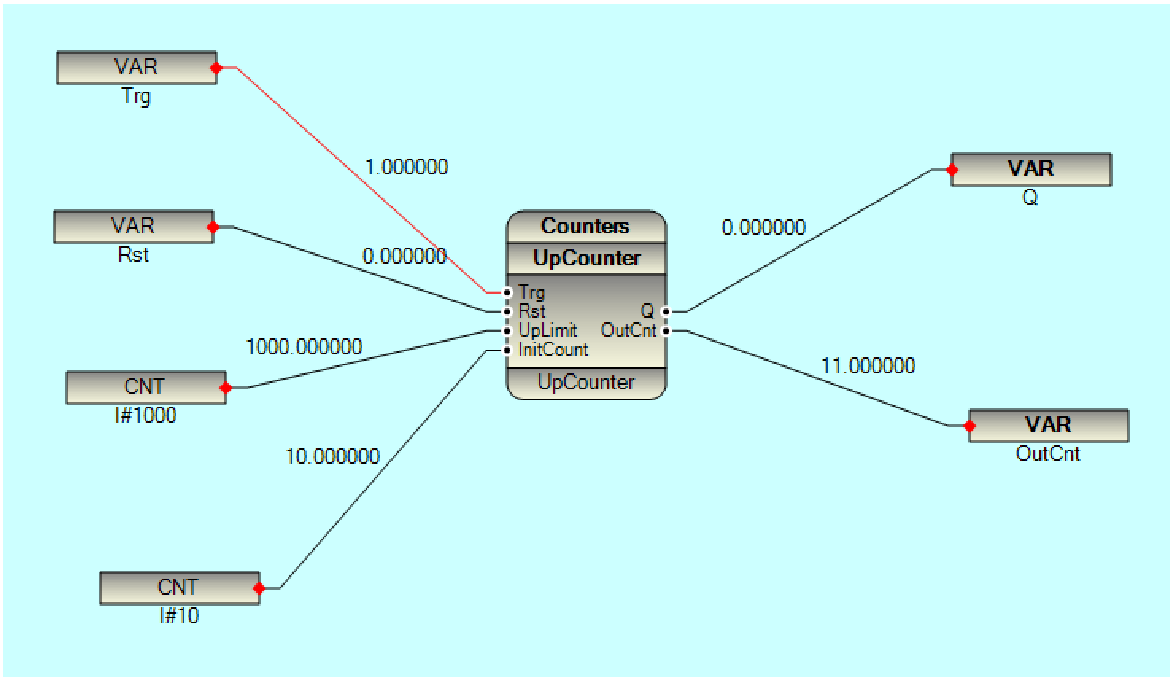
Output Name	Description	Type	Initial Value	Notes
Q	Overflow flag (high when count > UpLimit).	Digital	False	-
OutCnt	Current count value	Int	0	-

3.1.3 Functional Behavior

When Trg Input Change from 0 to 1, OutCnt will increase by One. When OutCnt reach to UpLimit then Q will set to

1 and OutCnt will not change any more, until Rst Signal is changed from 0 to 1.

When Rst Is changed from 0 to 1: OutCnt set to InitCount, Q set to 0. When Trg Changed from 0 to 1: If OutCnt is not reached to UpLimit, OutCnt increase by one. If OutCnt reach to UpLimit then Q will set to 1.

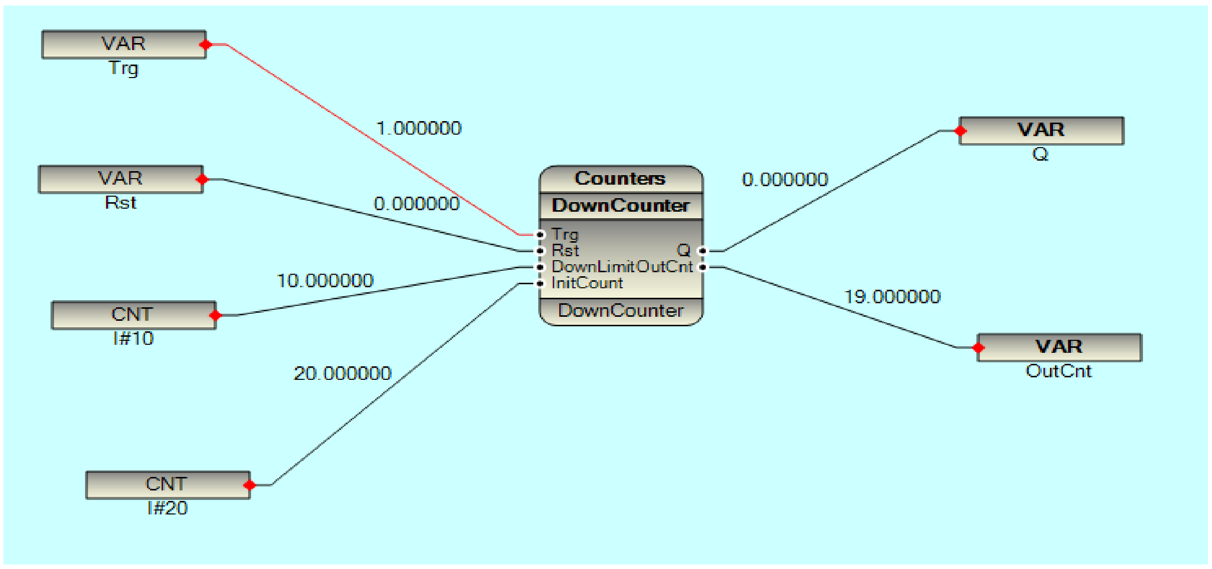


3.1.4 Testing:

- Set UpLimit=5, InitCount=0; Trg=0, Rst=0 → OutCnt=0, Q=0; pulse Trg high/low 6 times → OutCnt=5, Q=1; pulse Rst → OutCnt=0, Q=0.

3.2 DownCounter Function Block

The DownCounter function block is a down-counter designed to decrement its count on each rising edge of the trigger input (Trg). This block outputs the current count value on OutCnt and has a Boolean output Q, which is set whenever the count goes below the DownLimit.The block also supports an initial value (InitCount) and reset via the rising edge of the Rst input. Activating the reset returns the count to InitCount and clears the Q output.



3.2.1 Inputs

The DownCounter block accepts the following inputs. All inputs are processed during each scan cycle.

Input Name	Type	Description	Initial Value	Notes
Trg	Digital	Trigger: A rising edge (from False to True) decrements the counter by 1.	False	Use for pulse or event signals.
Rst	Digital	Reset: A rising edge resets the counter to the InitCount value and clears the Q output.	False	Ensures reliable restart functionality.
DownLimit	Int	Down Limit: The minimum value the counter can reach. If decremented below this, the counter stops at this value and Q activates.	0	Typically set to 0 or a non-negative integer.
InitCount	Int	Init Value: The starting count value or reset value for the counter.	10	Positive integer

3.2.2 Outputs

The DownCounter block produces the following outputs, updated at the end of each execution cycle.

Output Name	Type	Description	Initial Value	Notes
Q	Digital	Output Flag: True when the counter reaches or falls below DownLimit, indicating countdown complete. False otherwise.	False	Use to trigger alarms or end-of-process actions.
OutCnt	Int	Out Count Value: The current counter value.	0	Reflects real-time

				countdown progress.

3.2.3 Functional Behavior

When Function block is run for first time (There is no Static data for FB), OutCnt will set to InitCount. When Trg changed from 0 to 1, OutCnt will decrease by one. When OutCnt reached to DownLimit, then Q will set to 1 and OutCnt will not change until Rst Input changed from 0 to 1. When Rst Input changed from 0 to 1: OutCnt will set to InitCount, Q will set to 0.

4 Logical Group

4.1 Latch Function Block

The Latch function block implements a toggle latch (rising-edge triggered flip-flop), designed to toggle the digital output (Q) on each rising edge of the input (in1). It functions as a single-bit toggle, where Q switches between high and low states upon detecting a 0→1 transition on in1, holding the state until the next edge. This is useful for alternating actions, simple state machines, LED blink toggles, or bit-flip logic in control systems. State (previous input, current Q) is persisted to a configuration file for reliability across restarts, using paths derived from block parameters.

4.1.1 Inputs

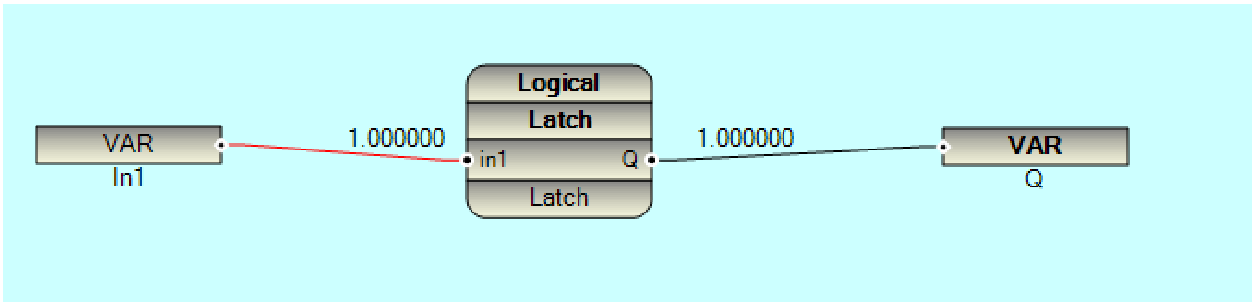
Name	Type	Description	Initial Value	Range/Notes
In	Digital	Toggle input: Rising edge (0→1) inverts Q.	0	0/1 (boolean/digital; int-cast internally)

4.1.2 Outputs

Name	Type	Description	Initial Value	Range/Notes
Q	Digital	Latched output: Toggles on rising In edge (0↔1); holds otherwise.	0	0/1 (boolean/digital)

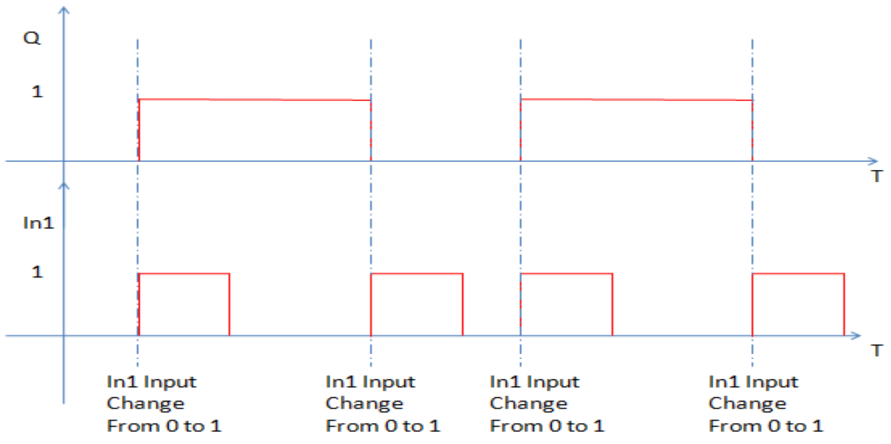
4.1.3 Functional Behavior:

Latch Function. When in1 changed from 0 to 1, Q will set to 1. Q is set to 1 until In1 is changed again from 0 to 1.



4.1.4 Scenario: Alternating Relay Control

- **Setup**: Wire In to a momentary pushbutton (pulse 0→1→0). Initial Q=0 (relay off).
- **Sequence**:
  - Press 1: Rising edge → Q=1 (relay on).
  - Hold/Release: No edge → Q holds 1.
  - Press 2: Rising edge → Q=0 (relay off).
  - Power cycle: Recovers Q=0 (persisted).



4.2 RS_FF Function Block

The RS_FF function block implements an RS latch (Set-Reset flip-flop), designed to hold a digital state based on rising edges of the Set (S) and Reset (R) inputs. On a rising edge of S, it sets the output Q high (1); on a rising edge of R, it resets Q low (0). If both S and R rise simultaneously, it prioritizes Reset (Q=0). The state holds until the next edge, making it useful for memory elements, priority logic, or bi-stable control in systems like safety interlocks or mode selection. State (previous R/S, current Q) is persisted to a configuration file for reliability across restarts, using paths derived from block parameters (e.g., program ID, PID).

4.2.1 Inputs

The RS_FF block accepts two inputs, configurable at runtime.

Inputs	Description	Type	Initial Value	Notes
R	Reset input (rising edge sets Q=0).	Digital	False	Connect a digital source (e.g., emergency stop). Prioritizes over S if simultaneous.
S	Set input (rising edge sets Q=1).	Digital	False	Connect a digital source (e.g., enable signal).

Connect the R/S to digital inputs. R/S is a Boolean input (0 or 1).

4.2.2 Outputs

The RS_FF block produces one output, updated per execution cycle.

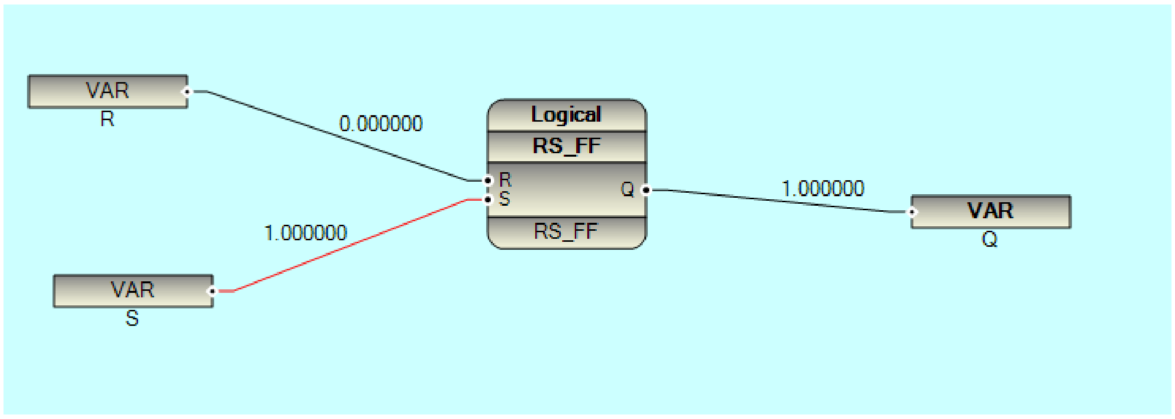
Output	Description	Type	Initial Value	Notes
Q	Latched state (1 after S, 0 after R; holds otherwise).	Digital	False	Connect to actuators (e.g., relay); latches until opposite edge.

4.2.3 Functional Behavior

When R Input changed from 0 to 1: Q will set to 0 When S Input Changed from 0 to 1: Q will set to 1 If R and S changed from 0 to 1 at same time, Q will set to 0.

4.2.4 Testing:

- R=S=0 → Q=0 (initial); pulse S high/low → Q=1; pulse R high/low → Q=0; simultaneous pulse R/S → Q=0.
- Restart: State resumes from file.
- Edge: Steady high R/S → no change after first edge.



4.2.5 State Transition Table

Previous State	R (Current)	S (Current)	Edge on R	Edge on S	New Q	Notes
Q=0, ROld=0, SOld=0	0	0	No	No	0	Hold
Q=0, ROld=0, SOld=0	1	0	Yes	No	0	Reset (no change)
Q=0, ROld=0, SOld=0	0	1	No	Yes	1	Set
Q=0, ROld=0, SOld=0	1	1	Yes	Yes	0	Reset priority
Q=1, ROld=0, SOld=1	0	0	No	No	1	Hold
Q=1, ROld=0, SOld=1	1	0	Yes	No	0	Reset
Q=1, ROld=0, SOld=1	0	1	No	No	1	Hold (no edge)
Q=1, ROld=0, SOld=1	1	1	Yes	No	0	Reset

4.3 SR_FF Function Block

The **SR_FF** function block (FB) implements a Set-Reset (SR) flip-flop with edge-triggered inputs and non-volatile state persistence. It maintains a binary output state (Q) that can be set to true (1) by the Set input or reset to false (0) by the Reset input. The FB detects rising edges on these inputs to trigger state changes, ensuring glitch-free operation in noisy environments.

4.3.1 Inputs

Name	Type	Description	Initial Value	Range/Notes
S	Digital	Set input: Rising edge (0→1) sets Q=1 (latches true state).	false	0/1 (boolean/digital); edge-triggered
R	Digital	Reset input: Rising edge (0→1) resets Q=0 (latches false state).	false	0/1 (boolean/digital); edge-triggered

4.3.2 Outputs

Name	Type	Description	Initial Value	Range/Notes
Q	Digital	Flip-flop output state (latched: 1 if set, 0 if reset or initial).	false	0/1 (boolean/digital)

4.3.3 Functional Behavior

When S Input Changed from 0 to 1: Q will set to 1 When R Input changed from 0 to 1: Q will set to 0 If R and S changed from 0 to 1 at same time, Q will set to 1.

4.3.4 Scenario: Latching a Motor Overload Alarm

- **Setup**: Connect S to overload sensor (pulse on fault), R to reset button (pulse on press). Q drives alarm horn.
- **Sequence**:
  - Initial: Q=false (no alarm).
  - Overload triggers (S rising edge): Q=1 (latched alarm, persists if system restarts).
  - Reset pressed (R rising edge): Q=0 (clears alarm).
- If both trigger simultaneously (rare): Alarm stays on (Set priority).

4.3.5 State Transition Table

Previous State	S (Current)	R (Current)	Edge on S?	Edge on R?	New Q	Notes
Q=0, SOld=0, ROld=0	0	0	No	No	0	Hold
Q=0, SOld=0, ROld=0	1	0	Yes	No	1	Set
Q=0, SOld=0, ROld=0	0	1	No	Yes	0	Reset (no change)
Q=0, SOld=0, ROld=0	1	1	Yes	Yes	1	Set priority
Q=1, SOld=1, ROld=0	0	0	No	No	1	Hold
Q=1, SOld=1, ROld=0	1	0	No	No	1	Hold (no edge)
Q=1, SOld=1, ROld=0	0	1	No	Yes	0	Reset
Q=1, SOld=1, ROld=0	1	1	No	Yes	0	Reset

4.4 JK_FF Function Block

The JK_FF function block implements a JK flip-flop, a versatile memory element that responds to rising edges on the J (set) and K (reset) inputs. On a rising edge of J, it sets the output Q to 1; on a rising edge of K, it resets Q to 0. If both J and K rise simultaneously, it toggles the current Q state. This behavior supports set, reset, hold (both low), and toggle modes. State (previous J/K, current Q) is persisted to a configuration file for reliability across restarts, using paths derived from block parameters.

4.4.1 Inputs

The JK_FF block accepts two inputs, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
J	Set input (rising edge sets Q=1; with K rising, toggles Q).	Digital	False	Connect a digital source (e.g., set signal).
K	Reset input (rising edge sets Q=0; with J rising, toggles Q).	Digital	False	Connect a digital source (e.g., reset signal).

4.4.2 Outputs

The JK_FF block produces one output, updated per execution cycle.

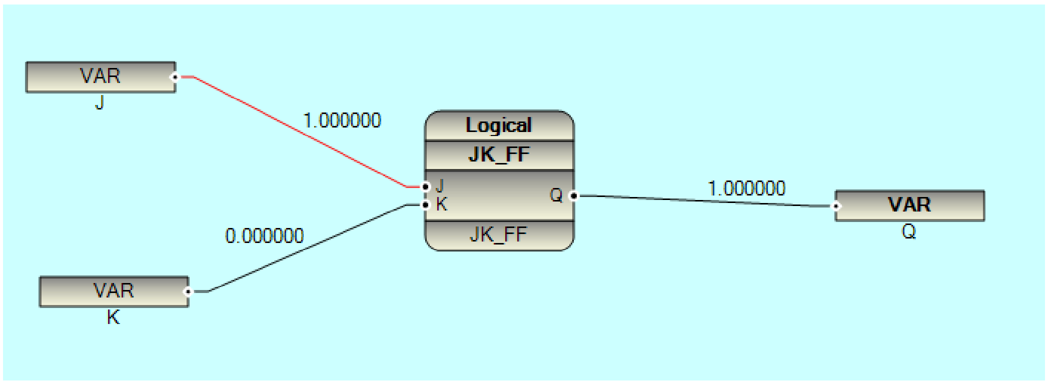
Output Name	Description	Type	Initial Value	Notes
Q	Latched state (set by J, reset by K, toggled on both; holds otherwise).	Digital	False	Connect to actuators (e.g., relay); latches until next edge.

4.4.3 Functional Behavior

JK Flip Flop. If J is changed from 0 to 1: Q will set to 1. If K is changed from 0 to 1: Q will set to 0. If J & K changed for 0 to 1 In the same time: - If Q is 1, Q will set to 0. - If Q is 0, Q will set to 1.

4.4.4 Testing:

- J=K=0 → Q=0 (initial); pulse J high/low → Q=1; pulse K high/low → Q=0; both pulses simultaneous → Q=1 (toggle from 0).
- Restart: State resumes from file.
- Edge: Steady high J/K → no change after first edge.



4.5 Pack2 Function Block

The Pack2 function block packs two individual digital boolean inputs (b0 and b1) into a single 2-bit integer output (Q), treating the inputs as bits of a nibble. The packing assumes b0 as the least significant bit (LSB) and b1 as the most significant bit (MSB), so Q = b1 * 2 + b0 * 1. All inputs are processed synchronously on each execution cycle, with no state or persistence required.

4.5.1 Inputs

The Pack2 block accepts two inputs, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
b0	Least significant bit (bit 0, 2^0 weight).	bool	False	Connect a digital source (e.g., flag 0).
b1	Most significant bit (bit 1, 2^1 weight).	bool	False	Connect a digital source (e.g., flag 1).

- **Configuration**: Wire b0-b1 to individual digital signals or flags.
- **Validation**: Each bX as bool (0/1).

4.5.2 Outputs

The Pack2 block produces one output, updated per execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Packed 2-bit integer (b1 b0 as binary).	int	0	Range 0-3; connect to integer registers or serializers.

4.5.3 Functional Behavior

The Pack2 function block packs two Boolean input signals into a single 2-bit unsigned integer output, treating each input as a bit and weighting them positionally: bit 0 (LSB, first input), bit 1 (second input). The output ranges from 0 (both false) to 3 (both true). Outputs update instantaneously with inputs.

4.5.4 Testing:

- both False → Q=0
- b0=True, b1=False → Q=1.
- b0=False, b1=True → Q=2.
- Both True → Q=3

4.6 UnPack2 Function Block

The UnPack2 function block unpacks a single 2-bit integer input (in1) into two individuals' digital Boolean outputs (b0 and b1), treating the input as a 2-bit value and extracting each bit. The unpacking assumes b0 as the least significant bit (LSB) and b1 as the most significant bit (MSB), so b0 = in1 & 1, b1 = (in1 >> 1) & 1. All outputs are processed synchronously on each execution cycle, with no state or persistence required. Values outside 0-3 are masked to 2 bits (mod 4).

4.6.1 Inputs

The UnPack2 block accepts one input, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
in1	Packed 2-bit integer to unpack.	int	0	Connect an integer source (e.g., register value). Range 0-3; larger values masked.

- **Configuration**: Wire in1 to an integer signal or register.
- **Validation**: in1 as int; masked to 2 bits.

4.6.2 Outputs

The UnPack2 block produces two outputs, updated per execution cycle.

Output	Description	Type	Initial Value	Notes

Name				
b0	Least significant bit (in1 bit 0).	Digital	False	Bit 0 (LSB, 2^0).
b1	Most significant bit (in1 bit 1).	Digital	False	Bit 1 (MSB, 2^1).

4.6.3 Functional Behavior

The UnPack2 function block extracts the two least significant bits from a single integer input (treated as a 2-bit value, 0-3) and outputs them as separate Boolean signals (0.0 or 1.0) on two ports. It performs a right-shift and bitwise AND operation in a loop: the first output gets bit 0 (LSB), the second output gets bit 1. This is a combinatorial demultiplexer for bit-level unpacking, useful in digital logic, flag extraction, or parallel Boolean processing in simulation environments. No state or timing involved; outputs update instantaneously with input.

4.6.4 Testing:

- in1=0 → b0=False, b1=False.
- in1=1 → b0=True, b1=False.
- in1=2 → b0=False, b1=True.
- in1=3 → b0=True, b1=True.

4.7 Pack8 Function Block

The Pack8 function block packs eight individual digital Boolean inputs (b0 to b7) into a single 8-bit integer output (Q), treating the inputs as bits of a byte. The packing assumes b0 as the least significant bit (LSB) and b7 as the most significant bit (MSB), so Q = b7 * 128 + b6 * 64 + ... + b0 * 1. All inputs are processed synchronously on each execution cycle, with no state or persistence required.

4.7.1 Inputs

The Pack8 block accepts eight inputs, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
b0	LSB (bit 0, 2^0 weight).	bool	False	Connect a digital source (e.g., flag 0).
b1	Bit 1 (2^1 weight).	bool	False	Connect a digital source (e.g., flag 1).
b2	Bit 2 (2^2 weight).	bool	False	Connect a digital source (e.g., flag 2).
b3	Bit 3 (2^3 weight).	bool	False	Connect a digital source (e.g., flag 3).

b4	Bit 4 (2^4 weight).	bool	False	Connect a digital source (e.g., flag 4).
b5	Bit 5 (2^5 weight).	bool	False	Connect a digital source (e.g., flag 5).
b6	Bit 6 (2^6 weight).	bool	False	Connect a digital source (e.g., flag 6).
b7	MSB (bit 7, 2^7 weight).	bool	False	Connect a digital source (e.g., flag 7).

- **Configuration**: Wire b0-b7 to individual digital signals or flags.
- **Validation**: Each bX as bool (0/1).

4.7.2 Outputs

The Pack8 block produces one output, updated per execution cycle.

Output Name	Description	Type	Initial Value	Notes
Q	Packed 8-bit integer (b7...b0 as binary).	int	0	Range 0-255; connect to integer registers or serializers.

4.7.3 Functional Behavior

The Pack8 function block packs eight Boolean input signals into a single 8-bit unsigned integer output, treating each input as a bit (1 if !=0, else 0) and weighting them positionally: bit 0 (LSB, first input) = 2^0=1, up to bit 7 (MSB, eighth input) = 2^7=128. The output ranges from 0 (all false) to 255 (all true). This is a combinatorial multiplexer for bit-packing, useful in digital logic, data serialization, or flag consolidation in simulation environments. No state or timing; outputs update instantaneously with inputs.

4.7.4 Testing:

- Set b0=True, others=False → Q=1.
- Set b7=True, others=False → Q=128.
- All True → Q=255; all False → Q=0.

4.8 UnPack8 Function Block

The UnPack8 function block unpacks a single 8-bit integer input (in1) into eight individuals digital boolean outputs (b0 to b7), treating the input as a byte and extracting each bit. The unpacking assumes b0 as the least significant bit (LSB) and b7 as the most significant bit (MSB), so b0 = in1 & 1, b1 = (in1 >> 1) & 1, ..., b7

= (in1 >> 7) & 1. All outputs are processed synchronously on each execution cycle, with no state or persistence required. Values outside 0-255 are masked to 8 bits (mod 256).

4.8.1 Inputs

The UnPack8 block accepts one input, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
in1	Packed 8-bit integer to unpack.	int	0	Connect an integer source (e.g., register value). Range 0-255; larger values masked.

- **Configuration**: Wire in1 to an integer signal or register.
- **Validation**: in1 as int; masked to 8 bits.

4.8.2 Outputs

The UnPack8 block produces eight outputs, updated per execution cycle.

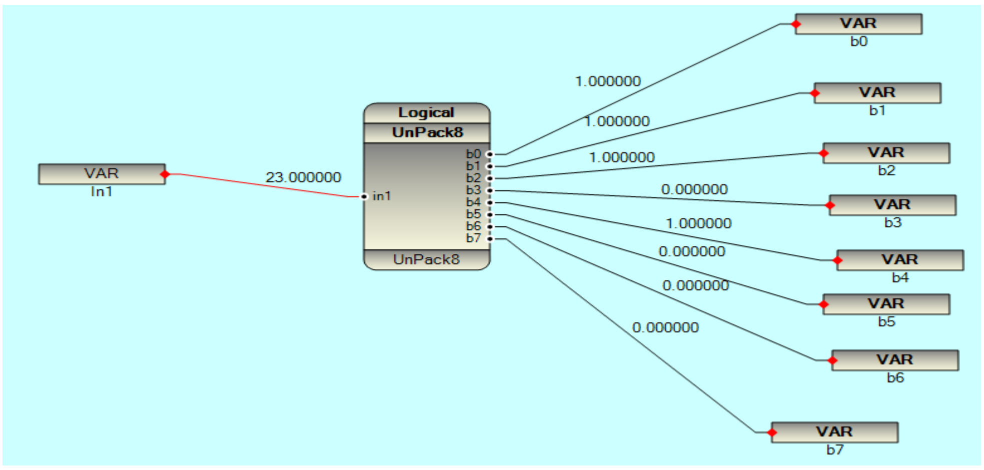
Output Name	Description	Type	Initial Value	Notes
b0	Least significant bit (in1 bit 0).	Digital	False	Bit 0 (LSB, 2^0).
b1	Bit 1 (in1 bit 1).	Digital	False	Bit 1 (2^1).
b2	Bit 2 (in1 bit 2).	Digital	False	Bit 2 (2^2).
b3	Bit 3 (in1 bit 3).	Digital	False	Bit 3 (2^3).
b4	Bit 4 (in1 bit 4).	Digital	False	Bit 4 (2^4).
b5	Bit 5 (in1 bit 5).	Digital	False	Bit 5 (2^5).
b6	Bit 6 (in1 bit 6).	Digital	False	Bit 6 (2^6).
b7	Most significant bit (in1 bit 7).	Digital	False	Bit 7 (MSB, 2^7).

4.8.3 Functional Behavior

The UnPack8 function block extracts the eight least significant bits from a single integer input (treated as an 8-bit value, 0-255) and outputs them as separate boolean signals (0.0 or 1.0) on eight ports. It performs a right-shift and bitwise AND operation in a loop: the first output gets bit 0 (LSB), up to the eighth output gets bit 7 (MSB). This is a combinatorial demultiplexer for bit-level unpacking, useful in digital logic, flag extraction, or parallel boolean processing in simulation environments. No state or timing involved; outputs update instantaneously with input.

4.8.4 Testing:

- in1=1 → b0=True, others=False.
- in1=128 → b7=True, others=False.
- in1=255 → all b0-b7=True; in1=0 → all False.



4.9 Pack16 Function Block

The Pack16 function block packs sixteen individual digital Boolean inputs (b0 to b15) into a single 16-bit integer output (Q), treating the inputs as bits of a word. The packing assumes b0 as the least significant bit (LSB) and b15 as the most significant bit (MSB), so Q = b15 * 32768 + b14 * 16384 + ... + b0 * 1. All inputs are processed synchronously on each execution cycle, with no state or persistence required.

4.9.1 Inputs

The Pack16 block accepts sixteen inputs, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
b0	Least significant bit (bit 0, 2^0 weight).	bool	False	Connect a digital source (e.g., flag 0).
b1	Bit 1 (2^1 weight).	bool	False	Connect a digital source (e.g., flag 1).
b2	Bit 2 (2^2 weight).	bool	False	Connect a digital source (e.g., flag 2).
b3	Bit 3 (2^3 weight).	bool	False	Connect a digital source (e.g., flag 3).
b4	Bit 4 (2^4 weight).	bool	False	Connect a digital source (e.g., flag 4).
b5	Bit 5 (2^5 weight).	bool	False	Connect a digital source (e.g., flag 5).
b6	Bit 6 (2^6 weight).	bool	False	Connect a digital source (e.g., flag 6).
b7	Bit 7 (2^7 weight).	bool	False	Connect a digital source (e.g., flag 7).

b8	Bit 8 (2^8 weight).	bool	False	Connect a digital source (e.g., flag 8).
b9	Bit 9 (2^9 weight).	bool	False	Connect a digital source (e.g., flag 9).
b10	Bit 10 (2^10 weight).	bool	False	Connect a digital source (e.g., flag 10).
b11	Bit 11 (2^11 weight).	bool	False	Connect a digital source (e.g., flag 11).
b12	Bit 12 (2^12 weight).	bool	False	Connect a digital source (e.g., flag 12).
b13	Bit 13 (2^13 weight).	bool	False	Connect a digital source (e.g., flag 13).
b14	Bit 14 (2^14 weight).	bool	False	Connect a digital source (e.g., flag 14).
b15	Most significant bit (bit 15, 2^15 weight).	bool	False	Connect a digital source (e.g., flag 15).

- **Configuration**: Wire b0-b15 to individual digital signals or flags.
- **Validation**: Each bX as bool (0/1).

4.9.2 Outputs

The Pack16 block produces one output, updated per execution cycle.

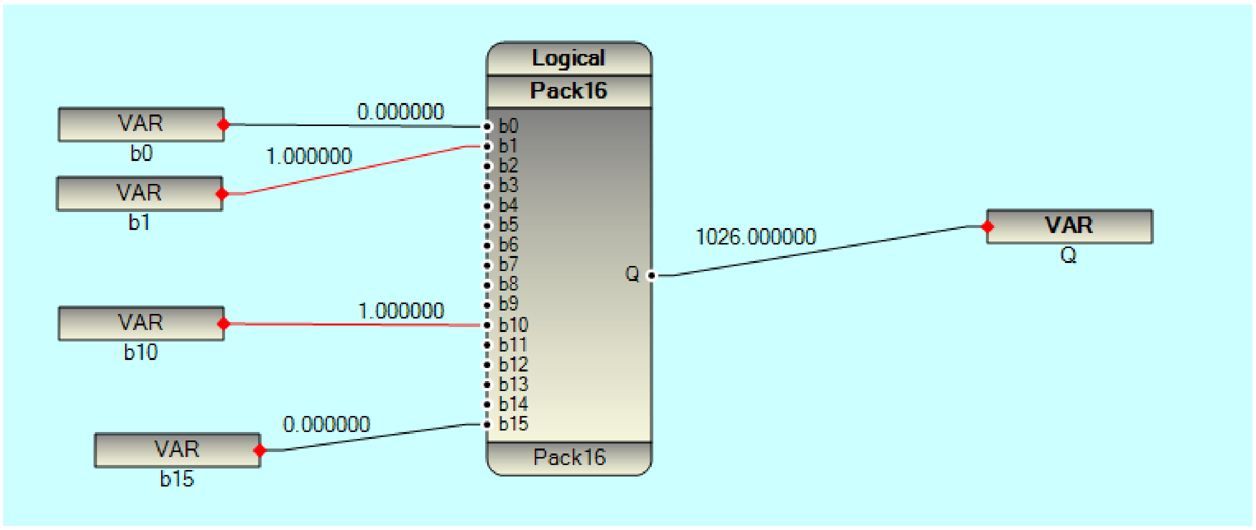
Output Name	Description	Type	Initial Value	Notes
Q	Packed 16-bit integer (b15...b0 as binary).	int	0	Range 0-65535 unsigned; connect to integer registers or serializers.

4.9.3 Functional Behavior

The Pack16 function block packs sixteen Boolean input signals into a single 16-bit unsigned integer output, treating each input as a bit (1 if !=0, else 0) and weighting them positionally: bit 0 (LSB, first input) = 2^0=1, up to bit 15 (MSB, sixteenth input) = 2^15=32768. The output ranges from 0 (all false) to 65535 (all true). No state or timing; outputs update instantaneously with inputs.

4.9.4 Testing:

- Set b0=True, others=False → Q=1.
- Set b15=True, others=False → Q=32768.
- All True → Q=65535; all False → Q=0.



4.10 UnPack16 Function Block

The UnPack16 function block unpacks a single 16-bit integer input (in1) into sixteen individual digital Boolean outputs (b0 to b15), treating the input as a word and extracting each bit. This is the reverse of Pack16. The unpacking assumes b0 as the least significant bit (LSB) and b15 as the most significant bit (MSB), so b0 = in1 & 1, b1 = (in1 >> 1) & 1, ..., b15 = (in1 >> 15) & 1. All outputs are processed synchronously on each execution cycle, with no state or persistence required. Values outside 0-65535 are masked to 16 bits (mod 65536).

4.10.1 Inputs

The UnPack16 block accepts one input, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
in1	Packed 16-bit integer to unpack.	int	0	Connect an integer source (e.g., register value). Range 0-65535; larger values masked.

- **Configuration**: Wire in1 to an integer signal or register.
- **Validation**: in1 as int; masked to 16 bits.

4.10.2 Outputs

The UnPack16 block produces sixteen outputs, updated per execution cycle.

Output Name	Description	Type	Initial Value	Notes
b0	Least significant bit (in1 bit 0).	Digital	False	Bit 0 (LSB, 2^0).
b1	Bit 1 (in1 bit 1).	Digital	False	Bit 1 (2^1).
b2	Bit 2 (in1 bit 2).	Digital	False	Bit 2 (2^2).
b3	Bit 3 (in1 bit 3).	Digital	False	Bit 3 (2^3).
b4	Bit 4 (in1 bit 4).	Digital	False	Bit 4 (2^4).
b5	Bit 5 (in1 bit 5).	Digital	False	Bit 5 (2^5).
b6	Bit 6 (in1 bit 6).	Digital	False	Bit 6 (2^6).
b7	Bit 7 (in1 bit 7).	Digital	False	Bit 7 (2^7).
b8	Bit 8 (in1 bit 8).	Digital	False	Bit 8 (2^8).
b9	Bit 9 (in1 bit 9).	Digital	False	Bit 9 (2^9).
b10	Bit 10 (in1 bit 10).	Digital	False	Bit 10 (2^10).
b11	Bit 11 (in1 bit 11).	Digital	False	Bit 11 (2^11).
b12	Bit 12 (in1 bit 12).	Digital	False	Bit 12 (2^12).
b13	Bit 13 (in1 bit 13).	Digital	False	Bit 13 (2^13).
b14	Bit 14 (in1 bit 14).	Digital	False	Bit 14 (2^14).
b15	Most significant bit (in1 bit 15).	Digital	False	Bit 15 (MSB, 2^15).

4.10.3 Functional Behavior

The UnPack16 function block extracts the sixteen least significant bits from a single integer input (treated as a 16-bit value, 0-65535) and outputs them as separate Boolean signals (0.0 or 1.0) on sixteen ports. It performs a right-shift and bitwise AND operation in a loop: the first output gets bit 0 (LSB), up to the sixteenth output gets bit 15 (MSB). No state or timing involved; outputs update instantaneously with input.

4.10.4 Testing:

- in1=1 → b0=True, others=False.
- in1=32768 → b15=True, others=False.
- in1=65535 → all b0-b15=True; in1=0 → all False.

4.11 MAP8 Function Block

The MAP8 function block maps eight independent input signals (in1 to in8) directly to eight corresponding output signals (Q1 to Q8), providing a simple pass-through or routing mechanism for any data type. Each output **QX** is a direct copy of input **inX** (e.g., Q1 = in1, Q2 = in2, etc.), processed synchronously on each execution cycle. Since the type is "Any", it supports flexible data types (e.g., bool, int, float) as long as inputs and outputs match. No computation, scaling, or state is involved—it's purely for signal distribution.

4.11.1 Inputs

The MAP8 block accepts eight inputs, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
in1	Signal to map to Q1.	Any	0	Connect any compatible source (e.g., sensor value).
in2	Signal to map to Q2.	Any	0	Connect any compatible source.
in3	Signal to map to Q3.	Any	0	Connect any compatible source.
in4	Signal to map to Q4.	Any	0	Connect any compatible source.
in5	Signal to map to Q5.	Any	0	Connect any compatible source.
in6	Signal to map to Q6.	Any	0	Connect any compatible source.
in7	Signal to map to Q7.	Any	0	Connect any compatible source.
in8	Signal to map to Q8.	Any	0	Connect any compatible source.

- **Configuration**: Wire in1-in8 to source signals; types must match outputs.
- **Validation**: Any type; ensure consistency to avoid coercion errors.

4.11.2 Outputs

The MAP8 block produces eight outputs, updated per execution cycle.

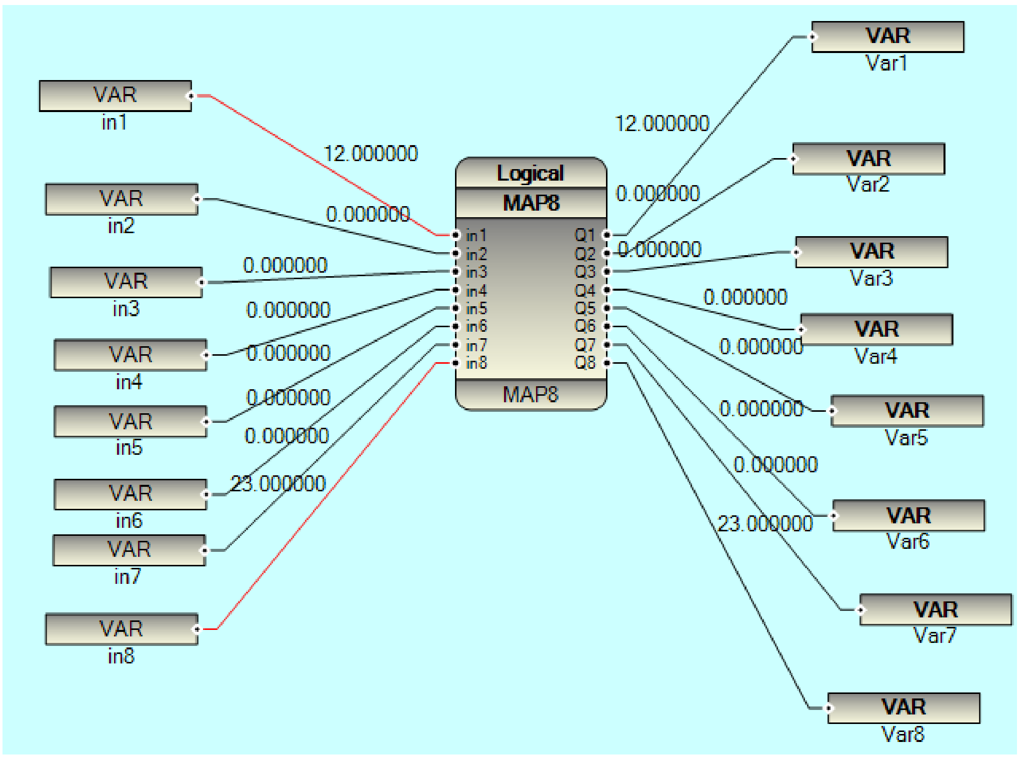
Output Name	Description	Type	Initial Value	Notes
Q1	Direct copy of in1.	Any	0	Mirrors in1 value.
Q2	Direct copy of in2.	Any	0	Mirrors in2 value.
Q3	Direct copy of in3.	Any	0	Mirrors in3 value.
Q4	Direct copy of in4.	Any	0	Mirrors in4 value.
Q5	Direct copy of in5.	Any	0	Mirrors in5 value.
Q6	Direct copy of in6.	Any	0	Mirrors in6 value.
Q7	Direct copy of in7.	Any	0	Mirrors in7 value.
Q8	Direct copy of in8.	Any	0	Mirrors in8 value.

4.11.3 Functional Behavior

The MAP8 function block performs a direct one-to-one mapping of eight input signals to eight output ports, copying double-precision values without modification or computation. This acts as a combinatorial signal router or passthrough buffer. No state, timing, or processing involved; outputs update instantaneously with inputs.

4.11.4 Testing:

- Set in1=5, others=0 → Q1=5, others=0.
- Change in3=True → Q3=True immediately.



4.12 MAP Function Block

The MAP function block provides a direct mapping (pass-through) of a single input signal (in1) to the output (Q), allowing for signal routing, aliasing, or reorganization without any transformation, scaling, or computation. The output Q is an exact copy of in1, supporting any compatible data type (e.g., bool, int, float), processed synchronously on each execution cycle. No state, persistence, or logic is involved—it's purely for signal distribution.

4.12.1 Inputs

The MAP block accepts one input, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
in1	Input signal to map to Q.	Any	0	Connect any compatible source (e.g., sensor value or variable).

- **Configuration**: Wire in1 to a source signal.
- **Validation**: Any type; ensure output compatibility.

4.12.2 Outputs

The MAP block produces one output, updated per execution cycle.

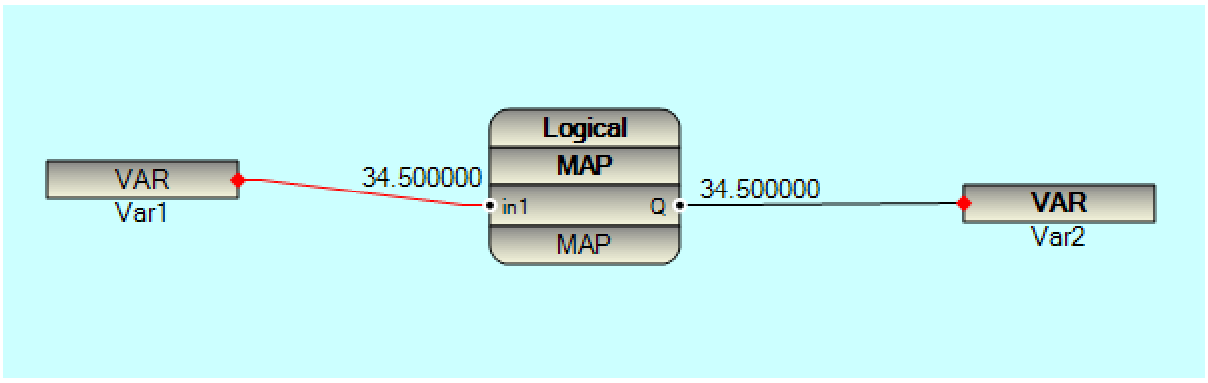
Output Name	Description	Type	Initial Value	Notes
Q	Direct copy of in1.	Any	0	Mirrors in1 value; connect to destination (e.g., actuator or other block).

4.12.3 Functional Behavior

The MAP8 function block performs a direct one-to-one mapping of eight input signals to eight output ports, copying double-precision values without modification or computation. This acts as a combinatorial signal router or passthrough buffer. No state, timing, or processing involved; outputs update instantaneously with inputs.

4.12.4 Testing:

- Set in1=5 → Q=5.
- Change in1=True → Q=True (if bool).



4.13 MAP_RE Function Block

The MAP_RE function block is an edge-triggered signal sampler and mapper, designed to capture the value of the input signal (Value) and hold it at the output (Q) only when the trigger input (Trg) transitions from 0 to 1 (rising edge). Changes to Value are ignored until the next rising edge on **Trg**, at which point Q is updated to the current Value and held steady. This provides sample-and-hold functionality.

4.13.1 Inputs

The MAP_RE block accepts two inputs, configurable at runtime.

Input Name	Description	Type	Initial Value	Notes
Trg	Trigger input (rising edge 0→1 samples Value to Q).	Digital	False	Connect a digital source (e.g., clock or event pulse). Steady high/low ignored after edge.
Value	Input value to sample and map to Q on trigger.	Any	0	Connect any compatible source (e.g., sensor, variable). Held at Q until next trigger.

- **Configuration**: Wire Trg to an edge source; Value to the signal to sample.
- **Validation**: Trg as bool; Value as Any (matches Q type).

4.13.2 Outputs

The MAP_RE block produces one output, updated per execution cycle.

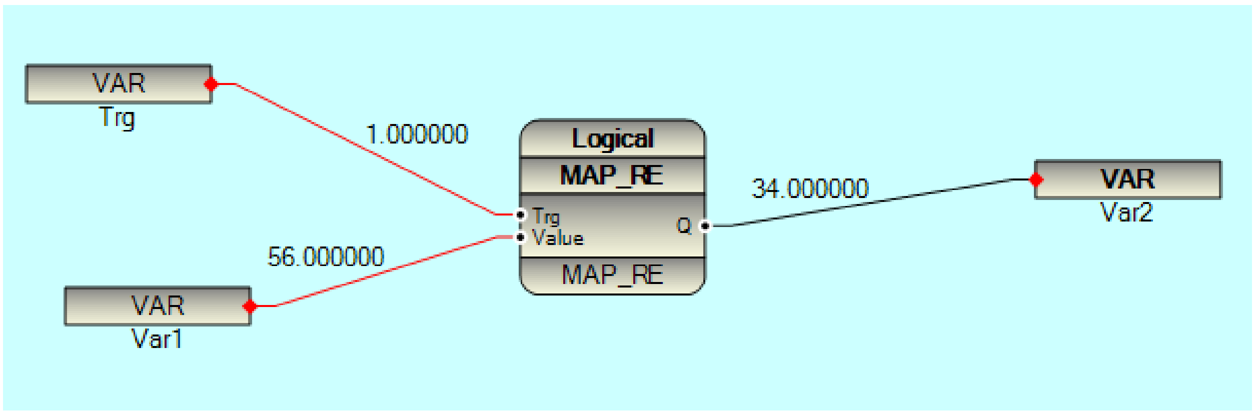
Output Name	Description	Type	Initial Value	Notes
Q	Sampled/held value (copies Value on rising Trg; holds otherwise).	Any	0	Connect to destinations (e.g., display or actuator); latches Value on trigger.

4.13.3 Functional Behavior

The MAP_RE function block implements a rising-edge triggered sample-and-hold latch that captures and holds an input value on the rising edge of a digital trigger signal, outputting the latched value continuously until the next trigger event. The held value remains unchanged on sustained high trigger or falling edge, providing edge-sensitive latching without retriggering. Latch state and trigger history are persisted across executions via a binary key-value file, enabling non-volatile retention. Unused path inputs allow for potential storage extensions. This suits event-driven data capture, setpoint latching, or snapshot buffering in control systems or simulations.

4.13.4 Testing:

- Trg=False, Value=5 → Q=0 (initial); Trg pulse high/low → Q=5 (holds); change Value=10 (Trg low) → Q=5 (no change); next Trg pulse → Q=10.



4.14 DIMap8 Function Block

The DIMap function block maps eight integer inputs (DI1 to DI8, typically treated as digital 0/1 values) to eight Value outputs and eight corresponding Status outputs, providing a simple signal distribution with chatter filtering. Each Value output is a direct copy of the corresponding DI input (as bool), while Status indicates whether the input is stable (chatter-free, e.g., no recent changes within a filter window). The mapping is 1:1 (Value1=DI1, Status1 for DI1 stability, etc.), processed synchronously on each execution cycle. No state or persistence is required, but chatter detection may use internal timing.

4.14.1 Inputs

The DIMap block accepts eight inputs, configurable at runtime.

Input	Description	Type	Initial	Notes

Name			Value	
DI1	First digital input register (maps to Value1/Status1).	int	0	Connect an integer source (e.g., DI register 0/1).
DI2	Second digital input register (maps to Value2/Status2).	int	0	Connect an integer source (e.g., DI register 0/1).
DI3	Third digital input register (maps to Value3/Status3).	int	0	Connect an integer source (e.g., DI register 0/1).
DI4	Fourth digital input register (maps to Value4/Status4).	int	0	Connect an integer source (e.g., DI register 0/1).
DI5	Fifth digital input register (maps to Value5/Status5).	int	0	Connect an integer source (e.g., DI register 0/1).
DI6	Sixth digital input register (maps to Value6/Status6).	int	0	Connect an integer source (e.g., DI register 0/1).
DI7	Seventh digital input register (maps to Value7/Status7).	int	0	Connect an integer source (e.g., DI register 0/1).
DI8	Eighth digital input register (maps to Value8/Status8).	int	0	Connect an integer source (e.g., DI register 0/1).

- **Configuration**: Wire DI1-DI8 to integer sources (e.g., DI registers or packed values).
- **Validation**: DI as int (treated as bool: 0=False, >0=True).

4.14.2 Outputs

The DIMap block produces sixteen outputs, updated per execution cycle.

Output Name	Description	Type	Initial Value	Notes
Value1	Mapped value from DI1 (as bool).	Digital	False	Direct copy of DI1 (0/1).
Status1	Stability status for DI1 (chatter-free).	Digital	False	True if DI1 stable (no recent change).
Value2	Mapped value from DI2 (as bool).	Digital	False	Direct copy of DI2 (0/1).
Status2	Stability status for DI2 (chatter-free).	Digital	False	True if DI2 stable.
Value3	Mapped value from DI3 (as bool).	Digital	False	Direct copy of DI3 (0/1).
Status3	Stability status for DI3 (chatter-free).	Digital	False	True if DI3 stable.
Value4	Mapped value from DI4 (as bool).	Digital	False	Direct copy of DI4 (0/1).
Status4	Stability status for DI4 (chatter-free).	Digital	False	True if DI4 stable.

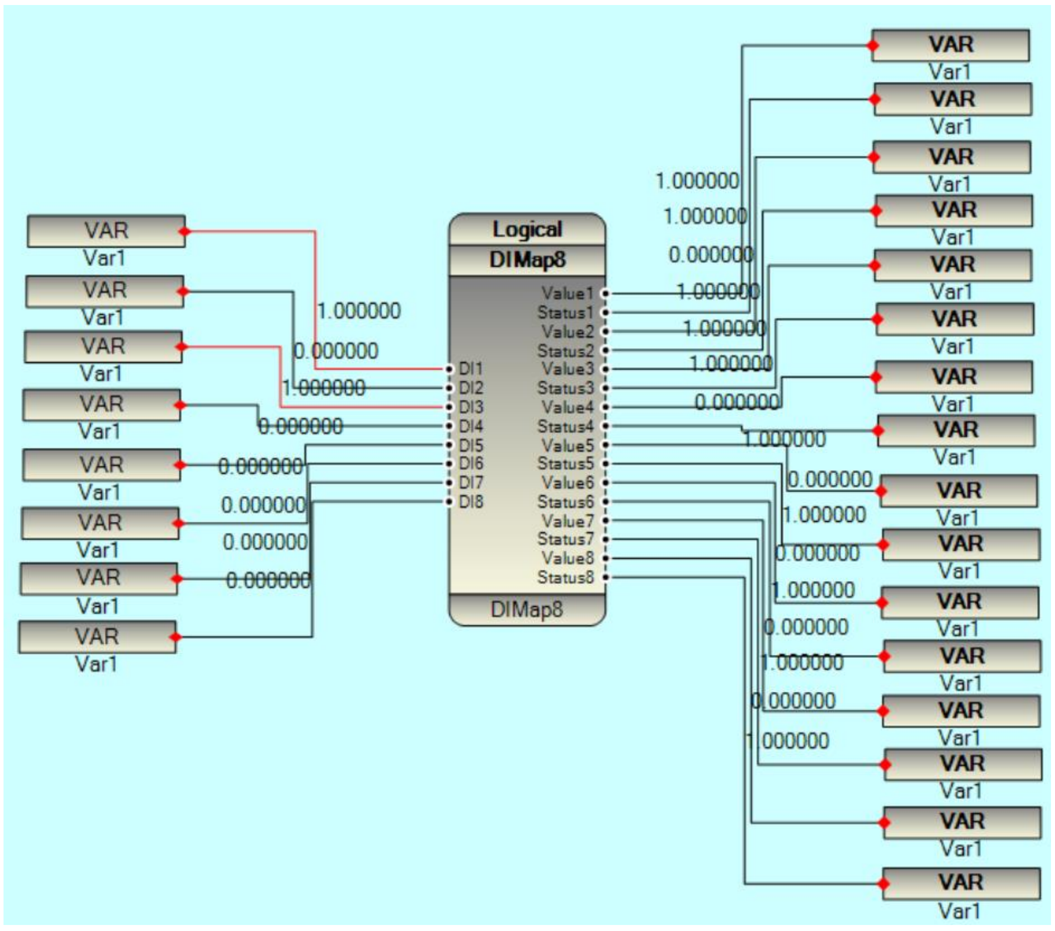
Value5	Mapped value from DI5 (as bool).	Digital	False	Direct copy of DI5 (0/1).
Status5	Stability status for DI5 (chatter-free).	Digital	False	True if DI5 stable.
Value6	Mapped value from DI6 (as bool).	Digital	False	Direct copy of DI6 (0/1).
Status6	Stability status for DI6 (chatter-free).	Digital	False	True if DI6 stable.
Value7	Mapped value from DI7 (as bool).	Digital	False	Direct copy of DI7 (0/1).
Status7	Stability status for DI7 (chatter-free).	Digital	False	True if DI7 stable.
Value8	Mapped value from DI8 (as bool).	Digital	False	Direct copy of DI8 (0/1).
Status8	Stability status for DI8 (chatter-free).	Digital	False	True if DI8 stable.

4.14.3 Functional Behavior:

Then DI is stable, Status is 1 but when DI is unstable, get in chattering filter and Status is 32. Value is equal of DI.

4.14.4 Testing:

- Set DI1=1, others=0 → Value1=True, Status1=True (if stable), others=False/False.
- Toggle DI2 rapidly → Value2 changes, Status2=False (chatter detected).



4.15 RIODIMap Function Block

The **RIODIMap** function block (FB) is a utility component designed for mapping Remote I/O (RIO) Digital Input (DI) status values in automation systems. It processes a 16-bit integer input representing packed digital input states and unpacks them into individual output signals. This allows for granular control and monitoring of discrete I/O points, such as sensor statuses, switches, or binary alarms.

4.15.1 Inputs

Name	Type	Description	Initial Value	Range
in1	INT	Packed 16-bit RIO DI status register (e.g., from a RIO module read). Bit 0 = LSB (first DI channel), Bit 15 = MSB (last DI channel).	0	0–65535

4.15.2 Outputs

Name	Type	Description	Initial Value	Range	Notes
b0	INT	Bit 0 state (direct).	0	0–1	Even index: 0 (off) or 1 (on).
b1	INT	Bit 1 state (scaled).	1	1–32	Odd index: 1 (off) or 32 (on).
b2	INT	Bit 2 state (direct).	0	0–1	Even index.
b3	INT	Bit 3 state (scaled).	1	1–32	Odd index.
b4	INT	Bit 4 state (direct).	0	0–1	Even index.
b5	INT	Bit 5 state (scaled).	1	1–32	Odd index.
b6	INT	Bit 6 state (direct).	0	0–1	Even index.
b7	INT	Bit 7 state (scaled).	1	1–32	Odd index.
b8	INT	Bit 8 state (direct).	0	0–1	Even index.
b9	INT	Bit 9 state (scaled).	1	1–32	Odd index.
b10	INT	Bit 10 state (direct).	0	0–1	Even index.
b11	INT	Bit 11 state (scaled).	1	1–32	Odd index.
b12	INT	Bit 12 state	0	0–1	Even index.

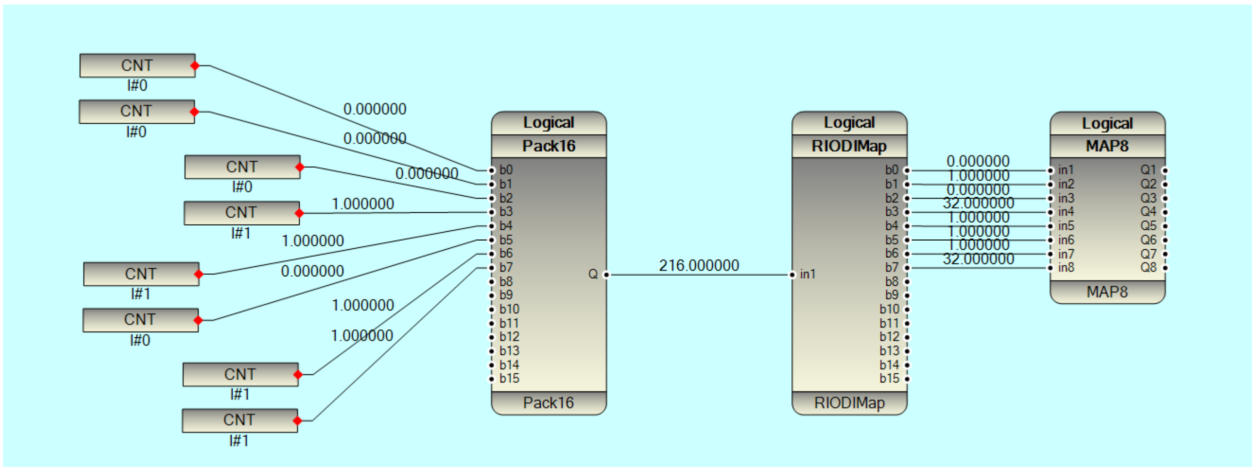
		(direct).			
b13	INT	Bit 13 state (scaled).	1	1–32	Odd index.
b14	INT	Bit 14 state (direct).	0	0–1	Even index.
b15	INT	Bit 15 state (scaled).	1	1–32	Odd index.

4.15.3 Functional Behavior:

When each bit of DI is stable the status of that will be set to one instead of that its status will be 32. Stability bits are on odd bit of input value.

4.15.4 Scenario: Monitoring 16 DI Channels from a RIO Module

- **Input**: in1 = 5 (binary: 0000 0000 0000 0101 → Bits 0 and 2 set).
- **Outputs**:
    - b0 = 1 (Bit 0 on)
    - b1 = 1 (Bit 1 off → 1)
    - b2 = 1 (Bit 2 on)
    - b3 = 1 (Bit 3 off → 1)
    - b4–b15 = 0 or 1 as per bits (all off → 0/1 alternating).



4.16 SELECTOR2 Function Block

The **SELECTOR2** function block (FB) is a versatile 2-to-1 multiplexer (selector) designed for signal routing in automation systems. It selects one of two input signals (of any compatible data type) and routes it to a single output based on a boolean selector input.

The FB supports generic "Any" type inputs/outputs, allowing seamless handling of integers, floats, booleans, or even strings in compatible systems. It operates deterministically with no latency, making it suitable for real-time applications.

4.16.1 Inputs

Name	Type	Description	Initial Value	Range/Notes
Sel	bool_HMI	Selector input: 0 (false) selects in0, 1 (true) selects in1. HMI-optimized for toggle switches or buttons.	false	0/1 (boolean)
in0	Any	First input signal (selected when Sel = 0/false).	0.0	Any compatible type/value
in1	Any	Second input signal (selected when Sel = 1/true).	0.0	Any compatible type/value

4.16.2 Outputs

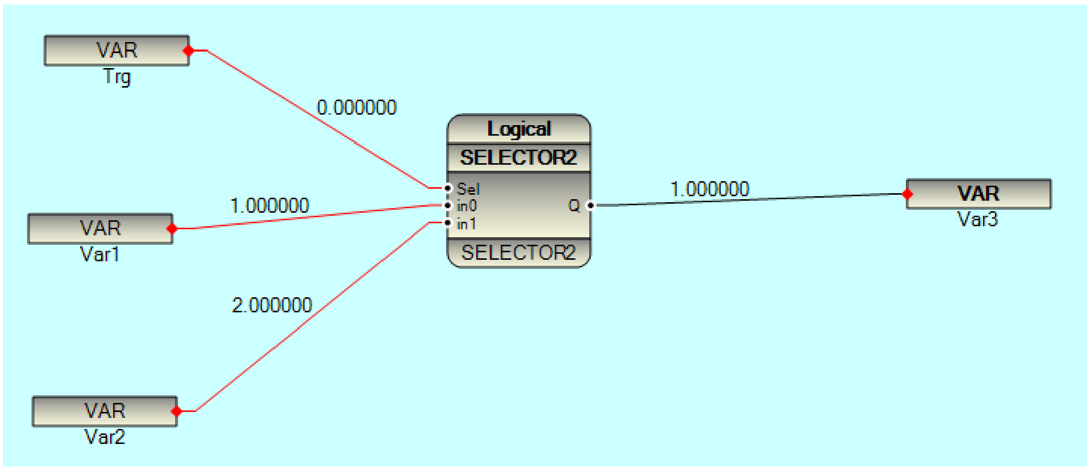
Name	Type	Description	Initial Value	Range/Notes
Q	Any	Selected output signal (either in0 or in1 based on Sel).	0.0	Matches selected input type/value

4.16.3 Functional Behavior:

When Sel Signal is 0, in0 is mapped to Q, When Sel is 1, in1 is mapped to Q.

4.16.4 Scenario: Switching Between Two Temperature Sensors

- **Setup**: Use Sel from a fault detector (1 if primary sensor fails, else 0). in0 = Primary Temp (°C), in1 = Backup Temp (°C).
- **Inputs**: Sel = 0, in0 = 25.3, in1 = 24.8.
- **Output**: Q = 25.3 (primary selected).
- If Sel = 1 (fault), Q = 24.8 (backup).



4.17 NOT Function Block

The **NOT** function block (FB) is a fundamental logical inverter designed for digital signal processing in automation systems. It performs a simple negation operation on a single boolean input, outputting the inverted state. This FB is essential for building complex control logic, such as creating interlocks, inverting sensor signals, or complementing gate outputs in safety circuits.

The FB treats the input as a digital signal (0/false → output 1/true; non-zero/true → output 0/false), ensuring compatibility with binary I/O. It operates instantaneously and deterministically, with no memory or timing dependencies, making it ideal for high-speed, real-time applications.

4.17.1 Inputs

Name	Type	Description	Initial Value	Range/Notes
in1	Digital	Input digital signal to invert (0/false inverts to true; non-zero/true inverts to false).	false	0/1 (boolean/digital)

4.17.2 Outputs

Name	Type	Description	Initial Value	Range/Notes
Q	Digital	Inverted output digital signal.	false	0/1 (boolean/digital)

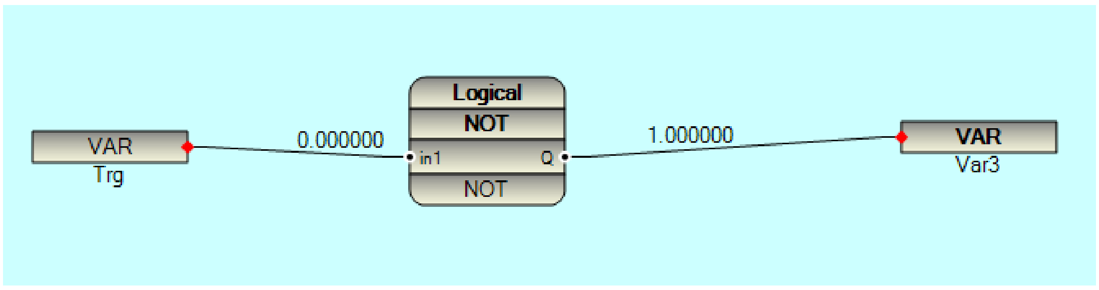
4.17.3 Functional Behavior:

Reverse Function – When In1 is 0, Q is 1 - When In1 is not 0, Q is 0

4.17.4 Scenario: Inverting a Limit Switch for Conveyor Control

- **Setup**: Use in1 from a limit switch (true when pressed). Invert to detect "not pressed" for starting the conveyor.
- **Inputs**: in1 = false (switch open).

- **Output**: Q = true (inverted: proceed to start conveyor).
- If in1 = true (switch closed), Q = false (stop condition).



4.18 ShiftL Function Block

The **ShiftL (FB)** function block performs a left bitwise shift on an integer input signal, moving its binary representation a specified number of positions to the left. This operation is carried out at the bit level, and the vacant bits on the right are filled with zeros.

4.18.1 Inputs

Name	Type	Description	Initial Value	Range/Notes
in1	INT	Input integer signal to shift (binary value to manipulate).	0	-2,147,483,648 to 2,147,483,647 (signed 32-bit)
Shift	INT	Number of positions to shift left (positive for left, 0 = no shift).	0	0–31 (shifts ≥32 typically result in 0)

4.18.2 Outputs

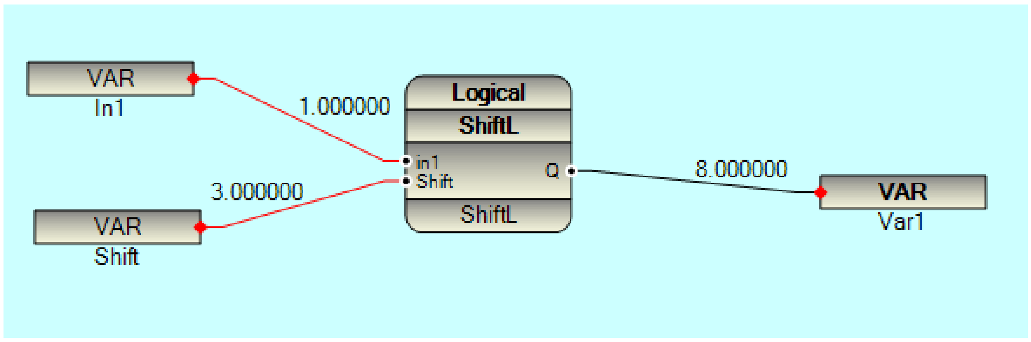
Name	Type	Description	Initial Value	Range/Notes
Q	INT	Shifted output integer (in1 << Shift).	0	-2,147,483,648 to 2,147,483,647 (may overflow)

4.18.3 Functional Behavior:

Shift Left. In1 Signals is shift to left side by Shift Number.

4.18.4 Scenario: Scaling a Counter Value

- **Setup**: Use in1 from a pulse counter (e.g., 5 pulses). Shift=2 to multiply by 4 for engineering units.
- **Inputs**: in1 = 5, Shift = 2.
- **Output**: Q = 20 (binary 101 shifted left by 2: 10100 = 20).
- If Shift = 0, Q = 5 (no change).



4.19 ShiftR Function Block

4.19.1 The ShiftR (FB) function block performs a right bitwise shift on an integer input signal, moving its binary representation a specified number of positions to the right. This operation is carried out at the bit level, and the vacant bits on the left are filled according to the sign bit for signed integers (arithmetic shift). This block supports 32-bit signed integers (INT), and shifts of more than 31 positions may be wrapped or truncated depending on the platform (usually modulo 32).

Note: For negative signed numbers, the shift is arithmetic, and the sign bit is preserved.

4.19.2 Inputs

Name	Type	Description	Initial Value	Range/Notes
in1	INT	Input integer signal to shift (binary value to manipulate).	0	-2,147,483,648 to 2,147,483,647 (signed 32-bit)
Shift	INT	Number of positions to shift right (positive for right, 0 = no shift).	0	0–31 (shifts ≥32 typically result in 0 or -1 for negatives)

4.19.3 Outputs

Name	Type	Description	Initial Value	Range/Notes
Q	INT	Shifted output integer (in1 >> Shift).	0	-2,147,483,648 to 2,147,483,647 (may underflow)

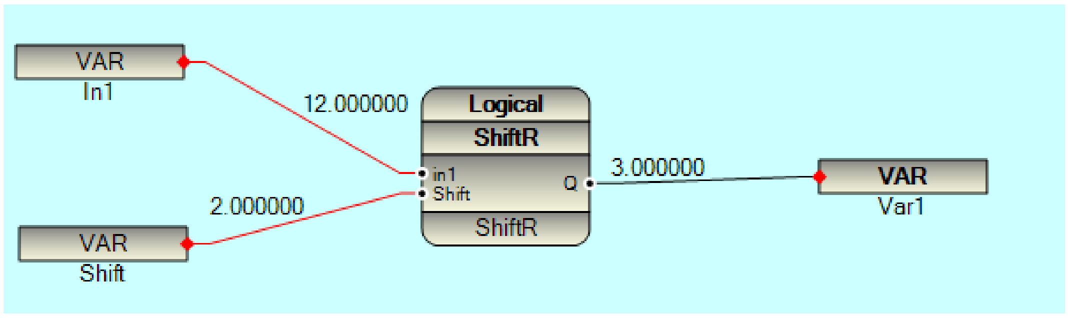
4.19.4 Functional Behavior:

Shift Right. In1 Signals is shift to right side by Shift Number.

4.19.5 Scenario: Scaling Down a High-Resolution Counter

- **Setup**: Use in1 from a high-speed encoder (e.g., 20 counts). Shift=2 to divide by 4 for display units.
- **Inputs**: in1 = 20, Shift = 2.
- **Output**: Q = 5 (binary 10100 shifted right by 2: 101 = 5).

- If Shift = 0, Q = 20 (no change).



4.20 OR[N] (OR2/OR3/OR4/OR5/OR6/OR7/OR8) Function Block

The **ORN** family of function blocks (where N = 2 to 8) implements a multi-input logical OR gate for digital signal processing in automation systems. Each variant performs a bitwise or logical OR operation across N input signals, outputting true (1) if **any** input is true, and false (0) otherwise. These FBs are scalable building blocks for aggregating conditions, such as combining multiple sensor alarms or enabling interlocks in control logic.

- **Variants**: OR2 (2 inputs), OR3 (3 inputs), ..., OR8 (8 inputs). Select the appropriate variant based on the number of signals to OR.
- The operation is stateless and instantaneous, ideal for real-time safety or monitoring applications.

4.20.1 Inputs (Common to All Variants)

Name	Type	Description	Initial Value	Range/Notes
in0	Digital	First input signal (contributes to OR).	false	0/1 (Boolean/digital)
in1	Digital	Second input signal (contributes to OR).	false	0/1 (Boolean/digital)
...	Digital	... (up to in{N-1} for N inputs).	false	0/1 (Boolean/digital)

Note: For OR2: in0, in1. For OR3: in0–in2. Up to OR8: in0–in7.

4.20.2 Outputs (Common to All Variants)

Name	Type	Description	Initial Value	Range/Notes
Q	Digital	Aggregated OR output (true if any input is true).	false	0/1 (Boolean/digital)

4.20.3 Functional Behavior:

Make OR all input signals and map to Output.

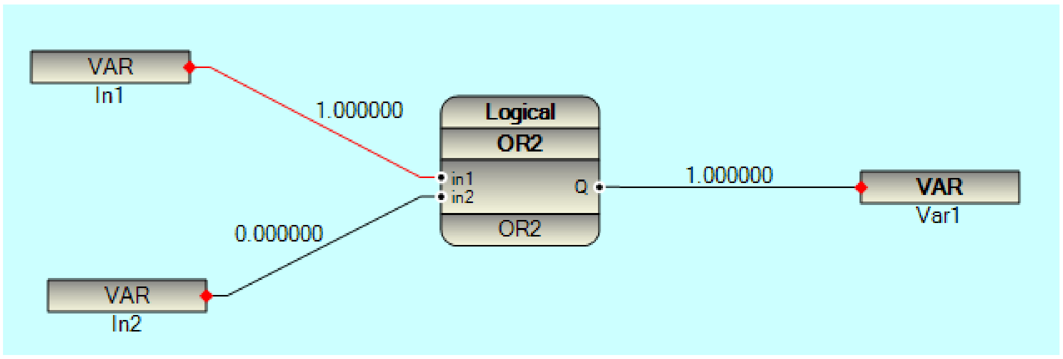
4.20.4 Aggregating 4 Sensor Alarms (Using OR4)

Setup: Wire in0–in3 to four binary sensors (e.g., high temp, low pressure, etc.). Q drives a shutdown relay.

Inputs: in0 = false, in1 = true, in2 = false, in3 = false.

Output: Q = true (since in1 is true → any alarm triggers shutdown).

If all false, Q = false (normal operation).



4.21 ANDN (AND2/AND3/AND4/AND5/AND6/AND7/AND8) Function Block

The **ANDN** family of function blocks (where N = 2 to 8) implements a multi-input logical AND gate for digital signal processing in automation systems. Each variant performs a bitwise or logical AND operation across N input signals, outputting true (1) only if **all** inputs are true, and false (0) otherwise. These FBs are scalable building blocks for combining conditions, such as requiring multiple sensors to confirm before enabling actions or creating permissive interlocks in control logic.

- **Variants**: AND2 (2 inputs), AND3 (3 inputs), ..., AND8 (8 inputs). Select the appropriate variant based on the number of signals to AND.
- The operation is stateless and instantaneous, ideal for real-time safety or permissive applications.

4.21.1 Inputs (Common to All Variants)

Name	Type	Description	Initial Value	Range/Notes
in0	Digital	First input signal (contributes to AND).	false	0/1 (Boolean/digital)
in1	Digital	Second input signal (contributes to AND).	false	0/1 (Boolean/digital)
...	Digital	... (up to in{N-1} for N inputs).	false	0/1 (Boolean/digital)

Note: For AND2: in0, in1. For AND3: in0–in2. Up to AND8: in0–in7.

4.21.2 Outputs (Common to All Variants)

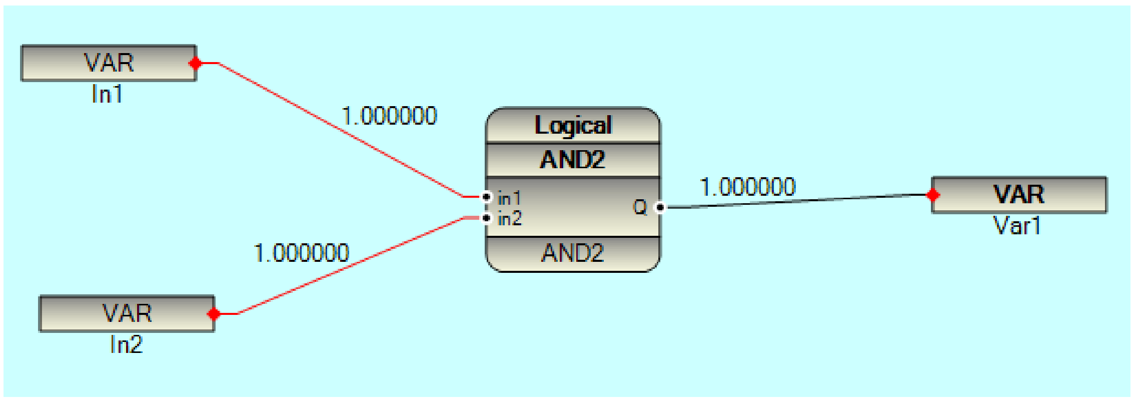
Name	Type	Description	Initial Value	Range/Notes
Q	Digital	Aggregated AND output (true only if all inputs are true).	false	0/1 (Boolean/digital)

4.21.3 Functional Behavior:

Make AND all input signals and map to Output.

4.21.4 Scenario: Safety Interlock for Machine Start (Using AND4)

- **Setup**: Wire in0–in3 to four safety signals (e.g., guard closed, e-stop off, etc.). Q enables the start button.
- **Inputs**: in0 = true, in1 = true, in2 = false, in3 = true.
- **Output**: Q = false (since in2 is false → not all conditions met; machine remains locked).
- If all true, Q = true (safe to start).



4.22 XORN (XOR2/XOR3/XOR4/XOR5/XOR6/XOR7/XOR8) Function Block

The **XORN** family of function blocks (where N = 2 to 8) implements a multi-input exclusive OR (XOR) gate for digital signal processing in automation systems. Each variant performs a logical XOR operation across N input signals, outputting true (1) only if an **odd number** of inputs are true, and false (0) otherwise. These FBs are scalable building blocks for parity checks, error detection, or toggling logic in control systems, such as validating odd/even counts of events or creating differential signals.

- **Variants**: XOR2 (2 inputs), XOR3 (3 inputs), ..., XOR8 (8 inputs). Select the appropriate variant based on the number of signals to XOR.
- The operation is stateless and instantaneous, ideal for real-time applications like communication protocols or fault parity monitoring.

4.22.1 Inputs (Common to All Variants)

Name	Type	Description	Initial Value	Range/Notes
in0	Digital	First input signal (contributes to XOR).	false	0/1 (Boolean/digital)
in1	Digital	Second input signal (contributes to XOR).	false	0/1 (Boolean/digital)
...	Digital	... (up to in{N-1} for N inputs).	false	0/1 (Boolean/digital)

Note: For XOR2: in0, in1. For XOR3: in0–in2. Up to XOR8: in0–in7.

4.22.2 Outputs (Common to All Variants)

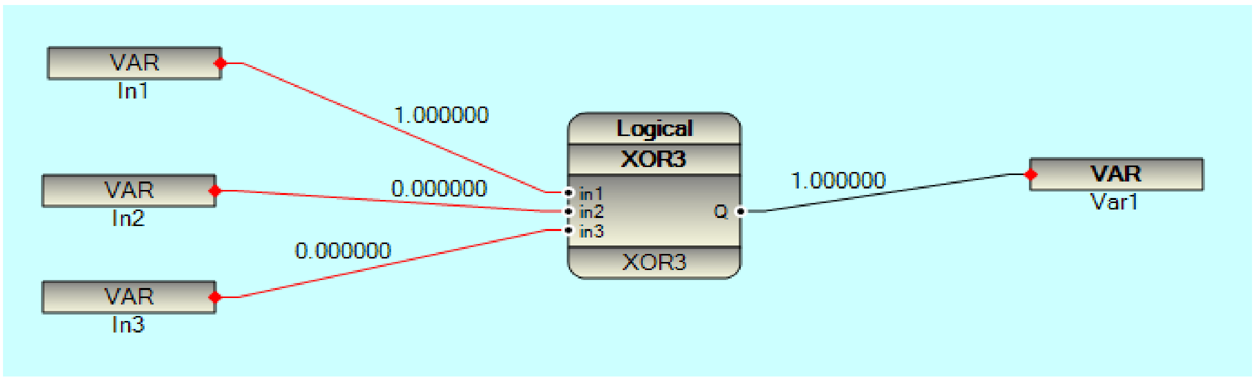
Name	Type	Description	Initial Value	Range/Notes
Q	Digital	Aggregated XOR output (true if odd number of inputs are true).	false	0/1 (Boolean/digital)

4.22.3 Functional Behavior:

Make XOR all input signals and map to Output. When all Input Signals or just one input is 1, Q will set to 1 Otherwise Q is set to 0.

4.22.4 Scenario: Parity Check on 3 Binary Sensors (Using XOR3)

- **Setup**: Wire in0–in2 to three toggle switches. Q indicates odd parity (e.g., for simple error flagging).
- **Inputs**: in0 = true, in1 = false, in2 = false (1 true → odd).
- **Output**: Q = true (odd number of trues → flag raised).
- If in0 = true, in1 = true, in2 = false (2 trues → even), Q = false.



5 Process Group

5.1 PID Function Block

The **PID** function block (FB) implements a standard Proportional-Integral-Derivative (PID) controller with feedforward initialization, output limiting, and non-volatile state persistence, designed for closed-loop process control in automation systems. It computes a control output (Qpid) based on the error between a setpoint (SP) and process variable (Signal), applying P, I, and D tuning parameters, plus an optional initial bias (Init). The FB includes a small deadband on error to reduce noise sensitivity and clamps the output within user-defined limits (pid_min to pid_max), flagging validity (v) accordingly.

5.1.1 Inputs

Name	Type	Description	Initial Value	Range/Notes
Enable	Digital	Enable PID computation (true=active, false=disable/output 0).	true	0/1 (boolean/digital; int-cast internally)
Signal	double	Process Variable (PV): Measured feedback signal.	0	Any (e.g., -1000 to 1000; engineering units)
SP	double	Setpoint (SP): Desired target value.	0	Any (matches Signal units)
P	double	Proportional gain (Kp): Amplifies current error.	0	≥0 (tuning: higher = faster response, more overshoot)
I	double	Integral gain (Ki): Accumulates error over time (1/Ti).	0	≥0 (tuning: higher = faster steady-state, windup risk)
D	double	Derivative gain (Kd): Responds to error rate (Td).	0	≥0 (tuning: higher = damping, noise sensitivity)
Init	double	Feedforward/Initial bias: Added constant offset to PID sum.	0	Any (e.g., manual bias for non-zero steady-state)
pid_min	double	Minimum output limit (saturation low).	0	Any (e.g., 0 for valves; ≤ pid_max)
pid_max	double	Maximum output limit (saturation high).	100	Any (e.g., 100% for actuators; ≥ pid_min)

5.1.2 Outputs

Name	Type	Description	Initial	Range/Notes

			Value	
Qpid	double	Computed PID control output (P*err + I*int + D*der + Init; clamped if saturated).	0	[pid_min, pid_max] when enabled
v	bool	Validity flag: true if Qpid unclamped (within limits), false if saturated or disabled.		

5.1.3 Functional Behavior:

Enable: When Enable is 1, Function Block is Enable, otherwise Outputs are 0.

Signal: Input signal which should be controlled by FB.

SP: Set Point

P: Proportional parameter

I: Integral Parameter

D: Derivative Parameter

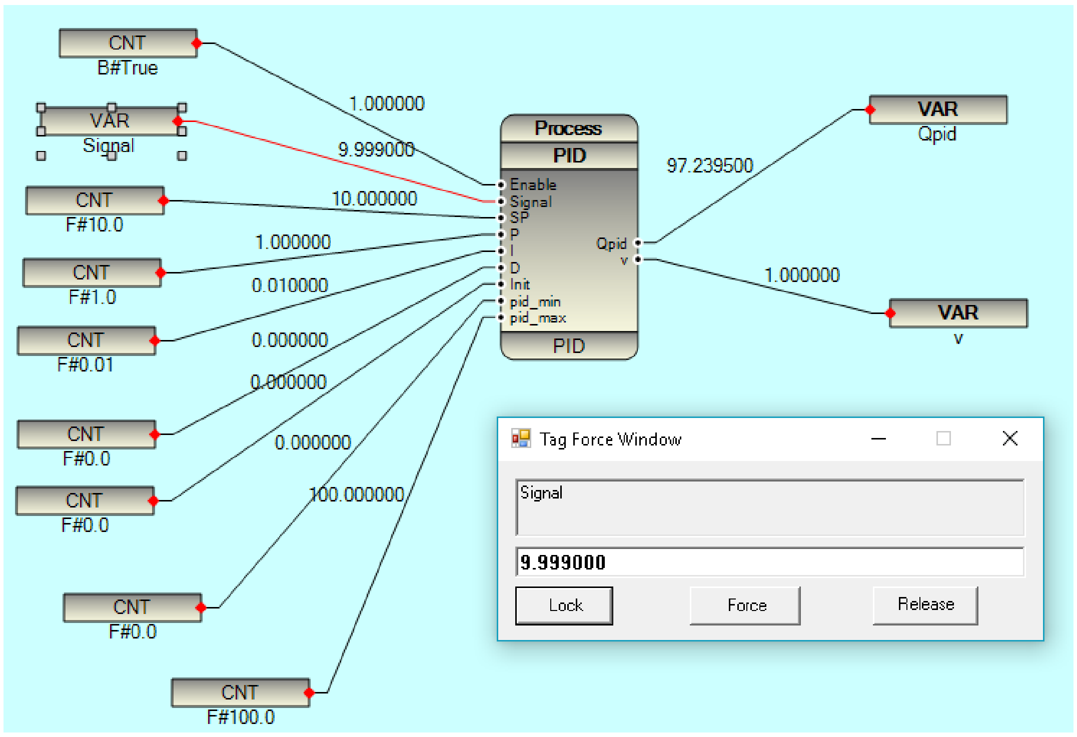
Init: Init (Bias) Value of Output

Pid_Min : Minimum Value of Output signal

Pid_Max : Maximum Value of Output Signal

Qpid : PID Output signal

V : Validity Signal. When QPid Signal is between pid_min and pid_Max , V is set to 1 (Output is Valid) otherwise it is set to 0.



5.2 Integral Function Block

The **Integral** function block (FB) implements a standard discrete-time integrator for accumulating a signal value over time, designed for signal processing and control applications in automation systems. It computes the running sum (Q) by adding the input Signal multiplied by the elapsed time interval (DeltaTime) to the previous integral value, but only when enabled. The Init input provides an initial or reset value for the accumulator.

5.2.1 Inputs

Name	Type	Description	Initial Value	Range/Notes
Enable	Digital	Enable/Trigger input: true to integrate Signal (level-gated); false to hold Q.	false	0/1 (boolean/digital; int-cast internally)
Signal	double	Input signal to integrate (e.g., rate or increment value).	0	Any (e.g., flow rate in L/s; units determine Q units * time)
Init	double	Initial or reset value for Q (applied on first execution or when re-enabled after disable).	0	Any (e.g., starting total; overrides previous Q on reset logic)

5.2.2 Outputs

Name	Type	Description	Initial Value	Range/Notes
Q	double	Integrated output: Running sum (Signal * time) when enabled.	0	Any (grows with integration; no auto-limits)

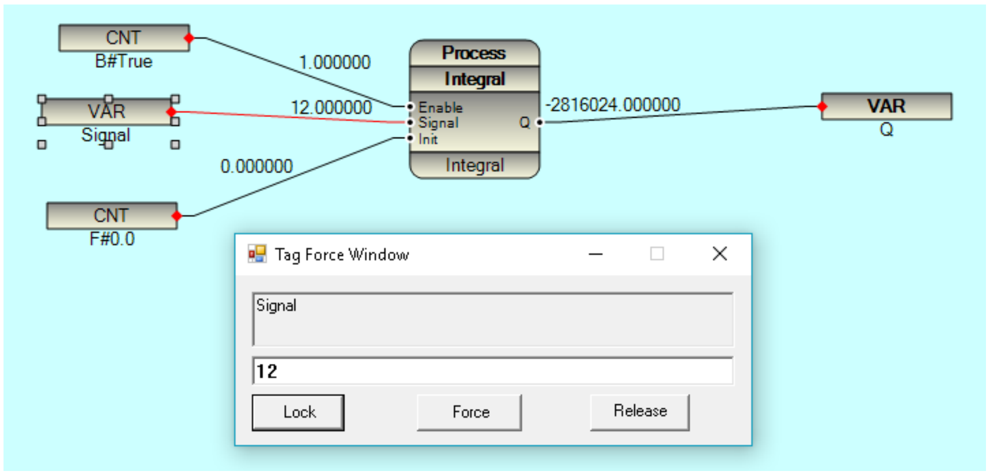
5.2.3 Functional Behavior:

Enable: When 1, FB is enabled, Otherwise FB Output is 0

Signal: Input Signal for getting integral

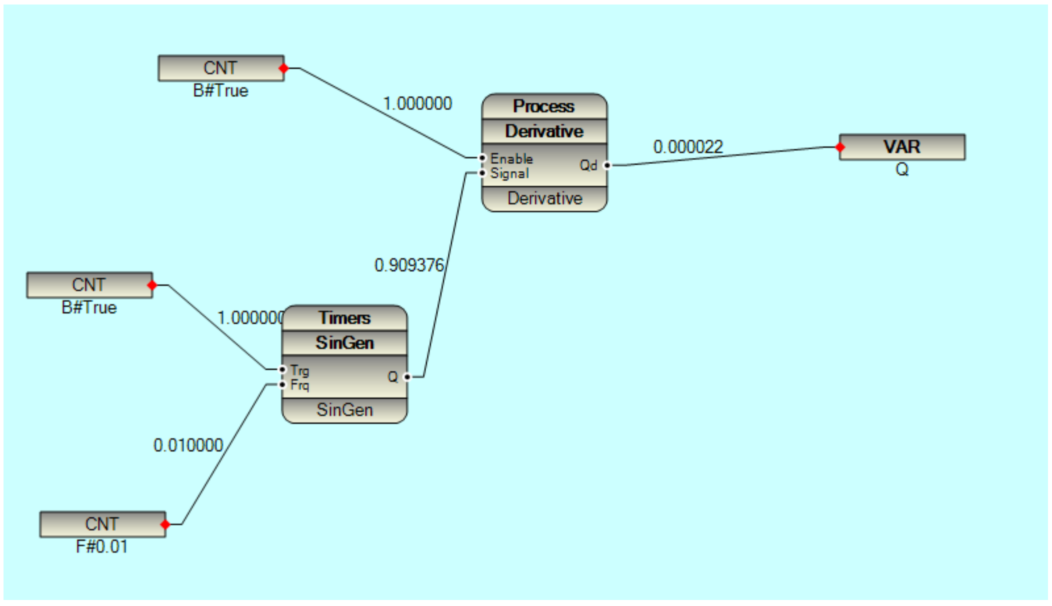
Init: Init (Bias) Value for integration

Q: Integrated Output signal



5.3 Derivative Function Block

This Function Block is Standard Derivative Function.



The Derivative function block operates as follows:

5.3.1 Enable Control:

When the Enable input is 1, the function block is active, and the derivative operation is performed. When Enable is 0, the output (Q) is set to 0.

5.3.2 Input Parameter:

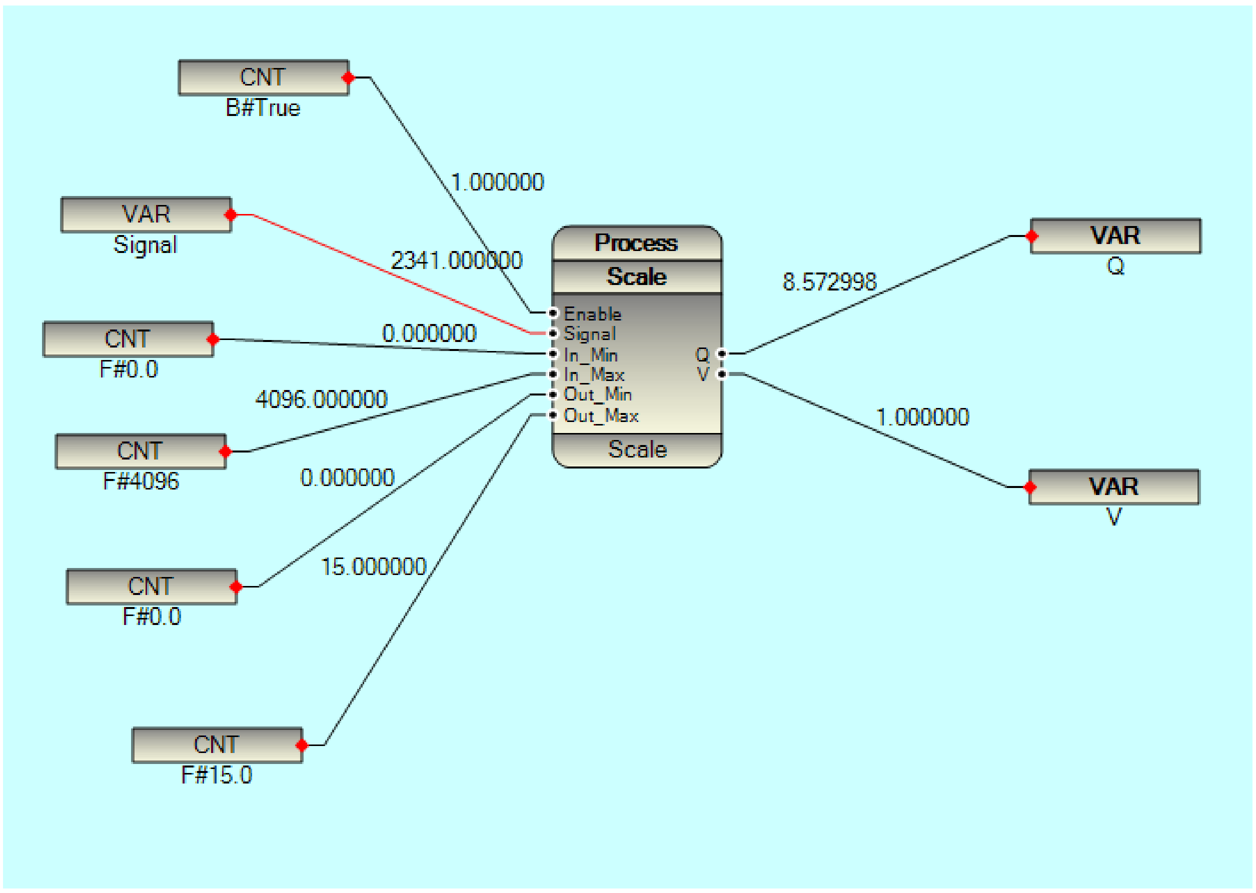
- Signal: The input signal used to calculate the derivative.

5.3.3 Output Signal:

- Q: The output signal representing the derivative of the input signal.

5.4 Scale Function Block

This Function Block is Scaling Input Signal.



The Scaler function block operates as follows:

5.4.1 Enable Control:

When the Enable input is 1, the function block is active, and scaling is performed. When Enable is 0, the output (Q) is set to 0.

5.4.2 Input Parameters:

- Signal: The raw input signal to be scaled.
- In_Min: The minimum value of the raw input signal.
- In_Max: The maximum value of the raw input signal.
- Out_Min: The minimum value of the scaled output.

- Out_Max: The maximum value of the scaled output.

5.4.3 Output Signals:

- Q: The scaled output signal, calculated by mapping the raw input signal from the range [In_Min, In_Max] to the range [Out_Min, Out_Max].
- V: The validity signal. Set to 1 when the output signal (Q) is within the range [Out_Min, Out_Max], indicating a valid output. Set to 0 otherwise.

5.5 NLScale Function Block

The NLScale function block performs nonlinear scaling of an input signal based on a user-defined graph modeled by 10 points. The operation is as follows:

5.5.1 Enable Control:

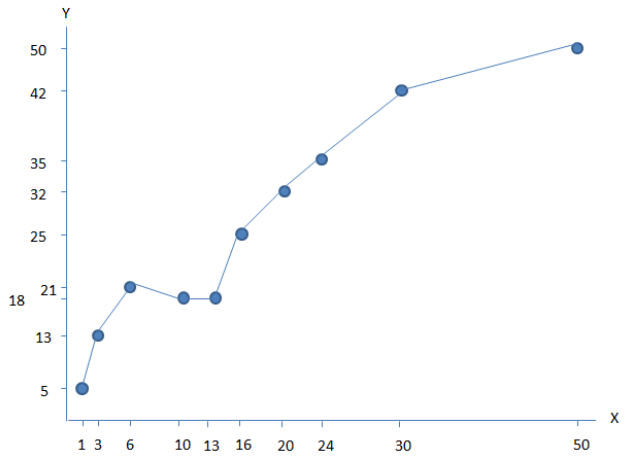
When the Enable input is 1, the function block is active, and scaling is performed. When Enable is 0, the output (Y) is set to 0.

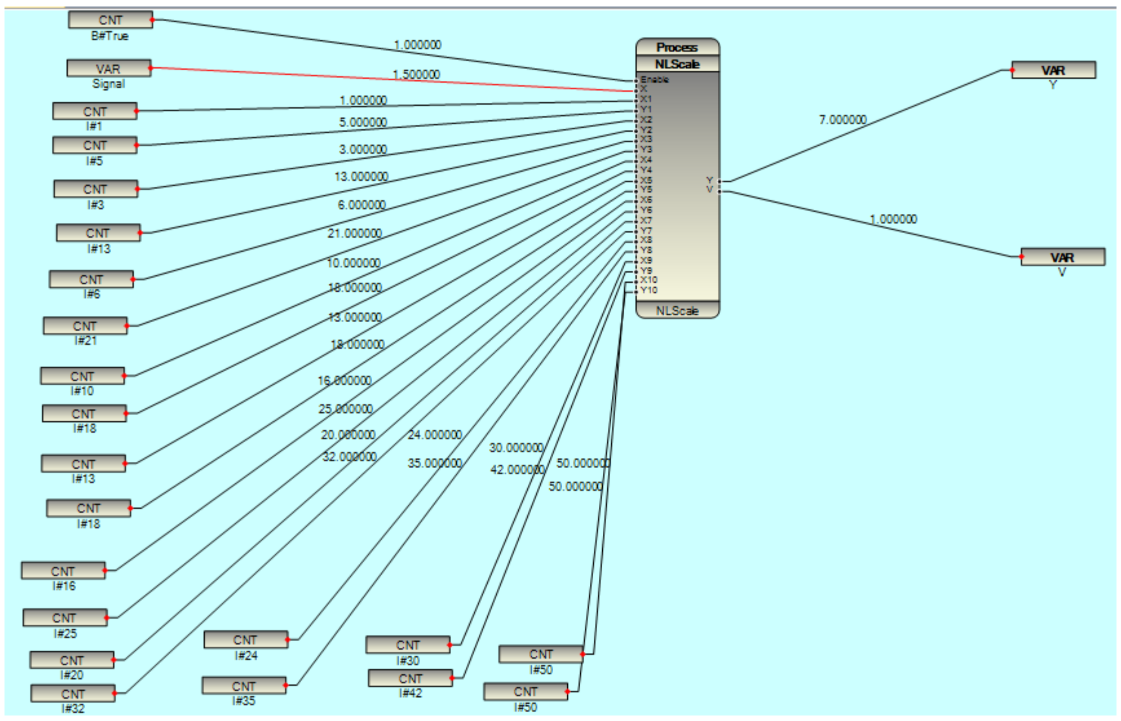
5.5.2 Input Parameters:

- X: The input signal to be scaled.
- Xn, Yn (n = 1 to 10): Pairs of coordinates defining the nonlinear scaling graph, where Xn represents the input points and Yn represents the corresponding output points.

5.5.3 Output Signals:

- Y: The scaled output signal, calculated based on the nonlinear interpolation between the defined (Xn, Yn) points.
- V: The validity signal. Set to 1 if the input X is within the range [X1, X10], indicating a valid output. Set to 0 if X is less than X1 or greater than X10.





5.6 Filter Function Block

The Filter function block implements a standard digital filter. The operation is as follows:

5.6.1 Enable Control:

When the Enable input is set to 1, the function block is operational. When Enable is 0, the output (Q) is not updated.

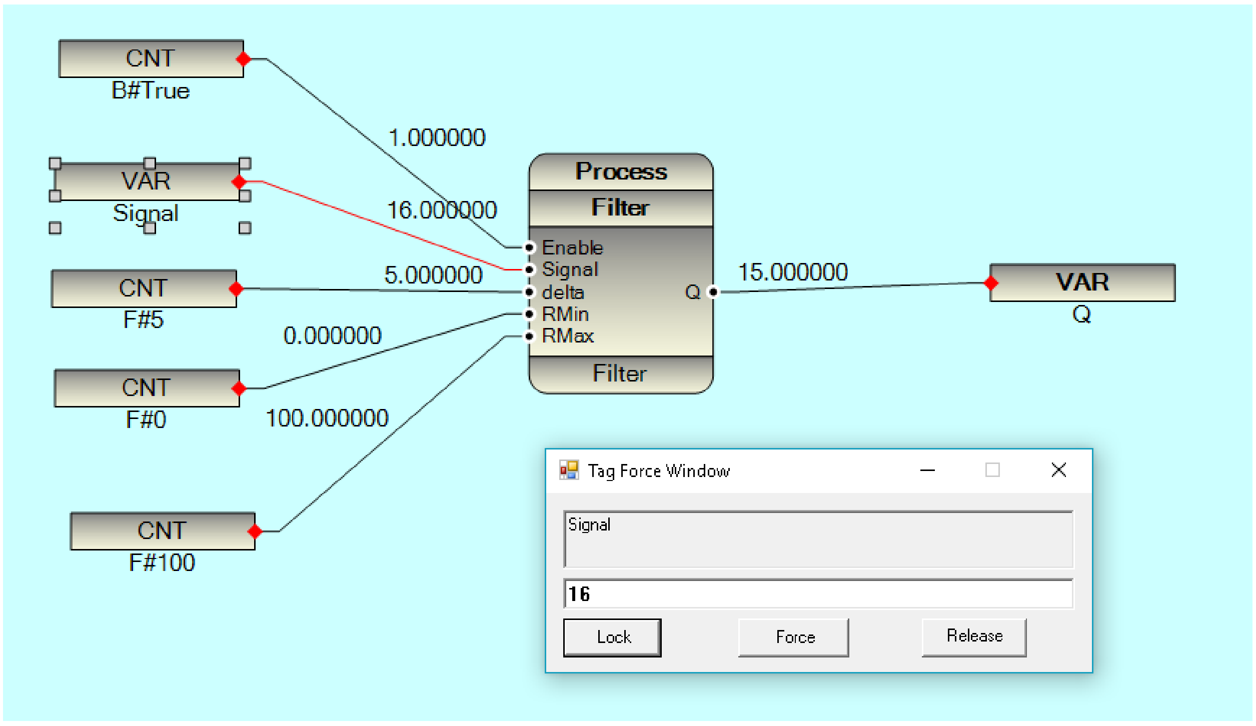
5.6.2 Input Parameters:

- Signal: The input signal to be filtered.

- Delta: The threshold for input signal change, expressed as a percentage. If the input signal changes by more than Delta, the current value of Signal is passed to the output (Q).
- RMin, RMax: The minimum and maximum allowable range for the input signal.

5.6.3 Output Behavior:

- If the change in the input Signal exceeds Delta (e.g., 5% as in the example), the current value of Signal is mapped to the output (Q).
- If the change in the input Signal is less than or equal to Delta, the previous value of Q is retained.



5.7 RawAFilter Function Block

The RawAFilter function block implements a standard digital filter that does not require specifying the minimum and maximum range of the input signal. The operation is as follows:

5.7.1 Enable Control:

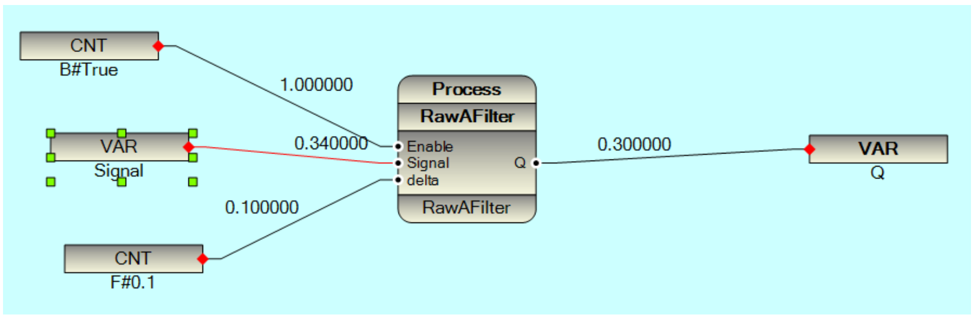
- When the Enable input is set to 1, the function block is operational. When Enable is 0, the output (Q) is not updated.

5.7.2 Input Parameters:

- Signal: The input signal to be filtered.
- Delta: The threshold for input signal change. If the absolute change in the input Signal exceeds Delta, the current value of Signal is mapped to the output (Q).

5.7.3 Output Behavior:

- If the change in the input Signal is greater than Delta (e.g., 0.1 in the example), the current value of Signal is passed to the output (Q).
- If the change in the input Signal is less than or equal to Delta, the previous value of Q is retained.



5.8 WDT Function Block

The **WDT** (Watch Dog Timer) function block is a signal activity monitor designed for detecting prolonged stability (lack of changes) in a digital or quantized analog input signal in automation systems. It tracks changes in the input Signal and starts a timer upon detecting stability (no change from previous value). If the signal remains unchanged for a duration exceeding the configured TimeOut period (in seconds), the output Q asserts to 1 (indicating inactivity or potential fault, e.g., "stuck" sensor). Any change during the timeout resets the timer and clears Q to 0.

5.8.1 Inputs

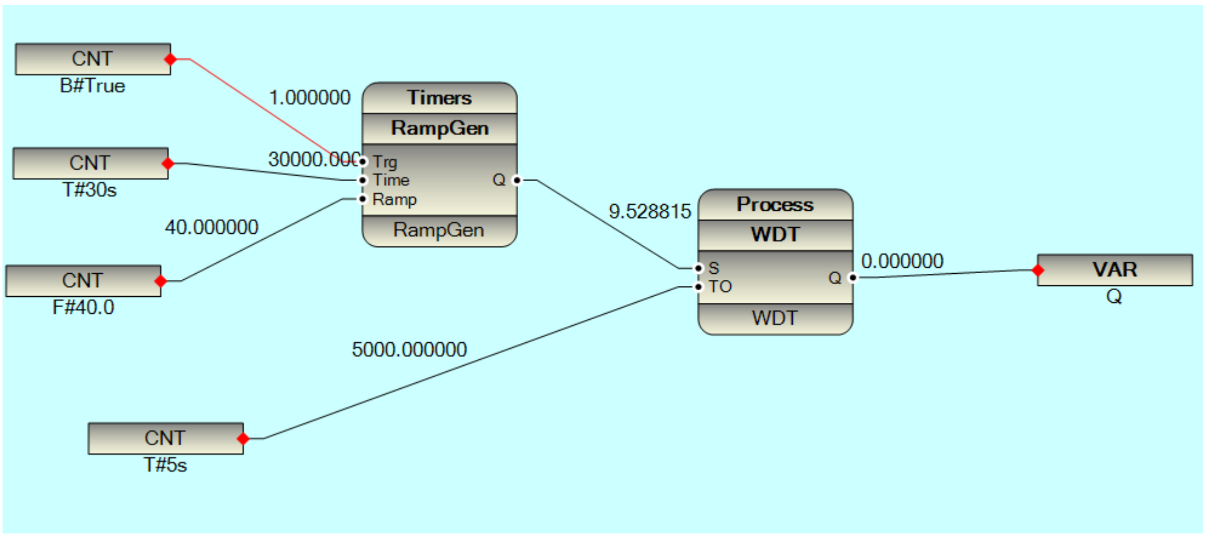
Name	Type	Description	Initial Value	Range/Notes
S	int	Input signal to monitor (digital or quantized analog; changes reset timer).	0	Any integer (e.g., 0/1 for digital; cast from double)
TO	double	Timeout period (milliseconds): Duration of stability before Q=1.	1000 (1s)	>0 (ms; e.g., 5000 for 5s; /1000 internally to sec)

5.8.2 Outputs

Name	Type	Description	Initial Value	Range/Notes
Q	int	Timeout indicator: 0 = signal changing/active, 1 = unchanged > TO (inactive/fault).		

5.8.3 Behavior

This Function is Watch Dog Function. If S input has any changes in less than TO, then Q Output is set to 0. If S Input is not changed for more than TO time then Q Output will set to 1.



5.9 FilterLP Function Block

The **FilterLP** function block (FB) implements a glitch suppression and change confirmation filter for analog signals in automation systems. It monitors the input Signal for changes relative to the full range (RMax - RMin), allowing small changes (≤ Delta % of range) to pass immediately while suppressing large transients (> Delta %) by holding the previous value for a configurable Delay period (in seconds). After the delay, it confirms the change: if still large, it accepts the new value; if reverted (small change), it holds the old value.

This FB is designed to reject noise spikes or brief faults in sensor data (e.g., electrical glitches) without introducing continuous lag, making it suitable for robust control loops like PID inputs or alarm thresholds. It includes non-volatile persistence for state retention across restarts and resets output to 0 when disabled. Note: Despite the name "FilterLP" (Low-Pass), it is not a traditional frequency-based LPF but a threshold-based transient suppressor.

5.9.1 Inputs

Name	Type	Description	Initial	Range/Notes

			Value	
Enable	bool	Enable processing: true to monitor/update Q; false to set Q=0 (disables, holds state).	true	true/false (1/0; int-cast internally)
Signal	double	Input signal to filter (e.g., raw sensor reading).	0	Any (e.g., 0–100%; range defines % scale)
delta	float	Change threshold (% of range): Changes ≤ delta pass immediately; > delta trigger delay.	5.0	>0 (e.g., 1.0–10.0 %; higher = more suppression)
RMin	float	Minimum range value (defines scale for % change; not a hard clamp on Q).	0.0	Any (e.g., 0; RMin < RMax)
RMax	float	Maximum range value (defines scale for % change; not a hard clamp on Q).	100.0	Any (e.g., 100; RMax > RMin)
Delay	int	Confirmation delay (seconds): Hold time for large changes before recheck.	100	≥0 (e.g., 1–300s; 0 = immediate accept, no suppression)

5.9.2 Outputs

Name	Type	Description	Initial Value	Range/Notes
Q	double	Filtered output: Passed small changes or confirmed large changes; holds old during suppression.	0	Matches Signal scale (no explicit limits)

5.9.3 Functional Behavior

The FilterLP function block implements a hysteresis-based low-pass filter with time-delay validation for noise rejection and change confirmation, outputting a filtered signal that passes small input variations immediately but holds the previous value for large changes until a specified delay elapses, then re-evaluating against a percentage threshold of the signal range. When enabled, it computes relative error as **|current - previous| / |range| * 100%**; if below threshold, updates output and previous; if above, delays and rechecks—if still above after delay, updates, else reverts to hold. Disabled mode zeros the output. State (previous value, output, timestamp, mode) is persisted via a binary key-value file for non-volatile resumption across restarts. Unused path inputs allow storage fallbacks (e.g., SRAM/SD). This suits robust signal conditioning in noisy environments, like sensor debouncing or control inputs, with real-time polling.

5.10 Drive1 Function Block

The **Drive1** function block (FB) implements a state machine for controlling a motor drive or actuator in remote/local, auto/manual modes, with interlock support, fault handling, and timeout-based confirmation, designed for industrial automation systems. It generates a binary command (CMD: 1=start/run, 0=stop) based on edge-triggered start/stop inputs, monitors drive feedback (Run), and asserts alarms (StartAlm, StopAlm) if the drive fails to respond within a configurable TimeOut period. Faults halt operation, and a rising-edge Reset clears alarms and resets the state.

It operates continuously when Enable=true, with non-volatile persistence for recovery across restarts, preventing spurious alarms on power-up.

5.10.1 Inputs

Name	Type	Description	Initial Value	Range/Notes
Enable	int	Enable FB operation: 1=active (process commands), 0=idle (CMD=0, alarms=0, state=0).	0	0/1 (digital; forces safe state when 0)
RL	int	Remote/Local mode: 1=Remote (use A/H/I inputs), 0=Local (ignore commands, CMD=0).	0	0/1 (1 enables remote control)
AM	int	Auto/Manual mode (if RL=1): 1=Auto (use AStart/AStop), 0=Manual (use HStart/HStop).	0	0/1 (mode selector)
AStart	int	Auto start command: Rising edge (0→1) starts drive if RL=1, AM=1, no fault.	0	0/1 (edge-triggered)
AStop	int	Auto stop command: Rising edge (0→1) stops drive if RL=1, AM=1, no fault.	0	0/1 (edge-triggered)
HStart	int	Manual start command: Rising edge (0→1) starts drive if RL=1, AM=0, no fault.	0	0/1 (edge-triggered)
HStop	int	Manual stop command: Rising edge (0→1) stops drive if RL=1, AM=0, no fault.	0	0/1 (edge-triggered)
IStart	int	Interlock start: Rising edge (0→1) starts drive (overrides modes/AM, if RL=1, no fault).	0	0/1 (edge-triggered; priority)
IStop	int	Interlock stop: Rising edge (0→1) stops drive (overrides modes/AM, if RL=1).	0	0/1 (edge-triggered; priority)
Fault	int	Drive fault input: 1= faulted (immediate stop,	0	0/1 (latched until reset)

		state=100).		
Run	int	Drive feedback: 1=running, 0=stopped (used for timeout confirmation).	0	0/1 (from drive status)
Reset	int	Reset command: Rising edge (0→1) clears alarms/faults, returns to state=0.	0	0/1 (edge-triggered; global clear)
TimeOut	long	Timeout period (ms): Delay to confirm Run after start/stop command.	5000	>0 (ms; e.g., 2000–10000; compared to pbsgetTime() ms)

5.10.2 Outputs

Name	Type	Description	Initial Value	Range/Notes
CMD	int	Drive command: 1=start/run, 0=stop.	0	0/1 (to drive input)
StartAlm	int	Start alarm: 1= start timeout (Run!=1 after TimeOut).	0	0/1 (alarm flag)
StopAlm	int	Stop alarm: 1= stop timeout (Run!=0 after TimeOut).	0	0/1 (alarm flag)
State	int	Current state code (0=idle, 1=starting, 2=running, etc.; for diagnostics).	0	0–100 (enum-like)

5.10.3 Functional Behavior

The Drive1 function block implements a state machine for controlling a drive or motor device, managing start/stop sequences in remote/local and auto/manual modes via edge-triggered commands from auto, hand, or interlock sources, while monitoring run feedback, faults, and timeouts to issue a command signal, alarms, and status. When enabled, it transitions through idle, starting, running, stopping, and fault states, enforcing delays for safe operation and resetting on dedicated input; local mode forces idle. State, timing, and edge histories are persisted via a binary key-value file for non-volatile resumption across restarts. Unused path inputs allow storage fallbacks (e.g., SRAM/SD). This suits industrial automation, PLC sequencing, or motor control with safety interlocks.

5.11 Drive1V2 Function Block

The **Drive1V2** function block (FB) implements a simplified state machine for controlling a motor drive or actuator in remote/local, auto/manual modes, with fault handling and timeout-based confirmation, designed for industrial automation systems. It generates a binary command (CMD: 1=start/run, 0=stop) and an enable signal (Enb: 1=remote enabled) based on level-triggered start/stop inputs (detected via edges), monitors drive feedback (Run), and asserts alarms (StartAlm, StopAlm) if the drive fails to respond within a configurable TimeOut period. Faults halt operation, and a rising-edge Reset clears alarms and resets the state.

This FB is a streamlined variant of Drive1, combining start/stop into single level inputs (AStSp for Auto: 1=start, 0=stop; similar for Manual HStSp), eliminating interlocks for simpler wiring. It operates continuously when Enable=true, with non-volatile persistence for recovery across restarts, preventing spurious alarms on power-up.

5.11.1 Inputs

Name	Type	Description	Initial Value	Range/Notes
Enable	int	Enable FB operation: 1=active (process commands), 0=idle (CMD=0, Enb=0, alarms=0, state=0).	0	0/1 (digital; forces safe state when 0)
RL	int	Remote/Local mode: 1=Remote (use A/H inputs, Enb=1), 0=Local (ignore commands, Enb=0, CMD=0).	0	0/1 (1 enables remote control)
AM	int	Auto/Manual mode (if RL=1): 1=Auto (use AStSp), 0=Manual (use HStSp).	0	0/1 (mode selector)
AStSp	int	Auto start/stop level: 1=start (rising 0→1 triggers), 0=stop (falling 1→0 triggers), if RL=1, AM=1, no fault.	0	0/1 (level; edges detected)
HStSp	int	Manual start/stop level: 1=start (rising 0→1), 0=stop (falling 1→0), if RL=1, AM=0, no fault.	0	0/1 (level; edges detected)
Fault	int	Drive fault input: 1=faulted (immediate stop, state=100).	0	0/1 (latched until reset)
Run	int	Drive feedback: 1=running, 0=stopped (used for timeout confirmation).	0	0/1 (from drive status)
Reset	int	Reset command: Rising edge (0→1) clears alarms/faults, disables Enb, returns to state=0.	0	0/1 (edge-triggered; global clear)
TimeOut	long	Timeout period (ms): Delay to confirm Run after start/stop command.	5000	>0 (ms; e.g., 2000–10000; compared to pbsgetTime() ms)

5.11.2 Outputs

Name	Type	Description	Initial Value	Range/Notes
Enb	int	Remote enable: 1=remote mode active (RL=1 and enabled), 0=local/disabled.	0	0/1 (gating signal)
CMD	int	Drive command: 1=start/run, 0=stop.	0	0/1 (to drive input)
StartAlm	int	Start alarm: 1=start timeout (Run!=1 after TimeOut).	0	0/1 (alarm flag)
StopAlm	int	Stop alarm: 1=stop timeout (Run!=0 after TimeOut).	0	0/1 (alarm flag)
State	int	Current state code (0=idle, 1=starting, 2=running, etc.; for diagnostics).	0	0–100 (enum-like)

5.11.3 Functional Behavior

The **Drive1V2** function block implements a simplified state machine for drive or motor control, handling sequenced start/stop in remote local/auto/manual modes via combined edge-triggered commands, monitoring run feedback and faults with timeout-based alarms, issuing an enable signal alongside command, alarms, and status. When enabled and remote, rising start or falling stop on auto/hand inputs initiates transitions through idle, starting, running, stopping, and fault states, enforcing delays for verification and resetting on dedicated input; local mode disables enable and idles. No interlock inputs; state, timing, and edge histories persisted via binary key-value file for non-volatile resumption. Unused path inputs allow storage fallbacks (e.g., SRAM/SD). This suit streamlined industrial sequencing or motor drives with reduced inputs.

5.12 Drive2 Function Block

The Drive2 function block is a state-machine-based controller for managing drive or motor operations in industrial automation, simulation, or PLC environments. It handles sequenced start/stop commands in remote/local and auto/manual modes via combined level-based signals (rising for start, falling for stop), with fault monitoring, dual run feedback verification, and timeout-based alarms. The block ensures safe transitions through timed delays, issues an enable signal plus dual commands (e.g., for redundant channels), and provides status/alarms for diagnostics. State persistence via file storage allows non-volatile operation across restarts. Compared to Drive1, it uses consolidated start/stop inputs and adds an explicit enable output.

5.12.1 Inputs

The block accepts 14 inputs (indices 5-18 in the object array), categorized as control, mode, command, feedback, and configuration. All are digital (0/1) except TimeOut (time value). Defaults are inferred from code (False/0 unless noted).

Input Name	Description	Type	Default	Index
Enable	Activates the block (1 = enabled, 0 = disabled; outputs zeroed on 0)	Digital	1	5
RL	Remote/Local mode (1 = Remote, 0 = Local; Local forces idle/disable)	Digital	0	6
AM	Auto/Manual mode (1 = Auto, 0 = Manual; affects command sources)	Digital	0	7
AStSp	Auto Start/Stop (rising edge to 1 = start, falling to 0 = stop in Auto mode)	Digital	0	8
HStSp	HMI/Manual Start/Stop (rising to 1 = start, falling to 0 = stop in Manual mode)	Digital	0	9
Fault	Fault feedback (1 = faulted; halts and transitions to fault state)	Digital	0	10
Run1	Run feedback channel 1 (1 = running; verified during start)	Digital	0	11
Run2	Run feedback channel 2 (1 = running for stop verification; 0 confirms stop)	Digital	0	12
Reset	Reset command (rising edge clears faults/alarms, returns to idle)	Digital	0	13
TimeOut	Timeout duration for start/stop verification (e.g., T#30s)	Time	T#30s	14

5.12.2 Outputs

The block produces 5 outputs (indices 0-4), all digital (0/1) except State (integer 0-100).

Output Name	Description	Type	Default
CMD1	Primary command (1 = run/start, active during running/starting)	Digital	0
CMD2	Secondary command (1 = stop, active during stopping)	Digital	0
StartAlm	Start alarm (1 = start timeout or failure)	Digital	0
StopAlm	Stop alarm (1 = stop timeout or failure)	Digital	0
State	Current state code (0=Idle, 1=Starting, 2=Running, 5=Start Alarm, 10=Stopping, 15=Stop Alarm, 100=Fault)	int	0

5.12.3 Functional Behavior

When enabled and in remote mode, the block monitors the appropriate start/stop signal based on auto/manual selection for level changes: a rising transition initiates a starting sequence, asserting the primary command and enabling the output while timing a verification period. If run feedback confirms operation within the timeout, it transitions to the running state, maintaining the primary command; otherwise, it alarms for start failure and desserts the command. A falling transition on the signal initiates a stopping sequence, asserting the secondary command and timing for stop confirmation via the second run feedback; success returns to idle, failure alarms for stop timeout. A fault input immediately halts commands and enters a fault condition. Local mode or disable forces idle with no enable or commands. A rising reset clears alarms and faults, returning to idle and resetting any active timing.

The block is retrigger able, restarting sequences on new level changes during verification. Persistence via file stores internal timing, levels, and conditions for resumption after restarts, ensuring continuity (e.g., ongoing verification completes on next cycle).

5.12.4 Example Scenario

1. **Auto Start/Stop:** Enabled, remote, auto. Start/stop signal 0→1 → starting (primary command on, enable on). Run1=1 within timeout → running. Signal 1→0 → stopping (secondary command on). Run2=0 within timeout → idle.
2. **Fault:** During running, fault=1 → fault (commands off). Reset rising → idle.
3. **Timeout:** Start, no Run1 after timeout → start alarm. Reset clears.

5.13 AAlarm Function Block

The **AAlarm** (Analog Alarm) function block (FB) is a multi-level threshold monitor for analog signals, designed for alarm generation in automation systems. It asserts binary alarms (HHAlm, HAlm, LAlm, LLAlm) when the input signal (In) exceeds high/high-high or falls below low/low-low setpoints, incorporating hysteresis (Hys) to prevent chattering near thresholds. Alarms are level-triggered (active when in the band) and cleared by a rising-edge reset (Reset) or disable (Enable=false). The Delay input is provided for potential timing but is not used in the current implementation (future-proofing).

5.13.1 Inputs

Name	Type	Description	Initial Value	Range/Notes

Enable	Digital_HMI	Enable alarms: True=monitor and assert, False=clears all alarms.	True	True/False (1/0; HMI toggle)
In	float	Analog input signal to monitor (e.g., sensor value).	0	Any real number (engineering units)
HHSP	float_HMI	High-High setpoint: Upper alarm threshold.	0	Any (>HSP typically; HMI adjustable)
HSP	float_HMI	High setpoint: Upper alarm threshold.	0	Any (HSP < HHSP, >LSP)
LSP	float_HMI	Low setpoint: Lower alarm threshold.	0	Any (LSP > LLSP)
LLSP	float_HMI	Low-Low setpoint: Lower alarm threshold.	0	Any (<LSP)
Hys	float_HMI	Hysteresis width: Deadband around setpoints to prevent chattering.	1	≥0 (e.g., 0.5–5% of range; HMI slider)
Delay	Time_HMI	Delay time (unused in current logic; reserved for future timing).	T#5s	Time (e.g., T#1s–T#30s; HMI selector)
Reset	Digital_HMI	Reset command: Rising edge (False→True) clears all alarms.	False	True/False (1/0; HMI button)

5.13.2 Outputs

Name	Type	Description	Initial Value	Range/Notes
HHAlm	Digital	High-High alarm: True if In > (HHSP + Hys).	False	True/False (1/0; latched until reset/disable)
HAlm	Digital	High alarm: True if (HSP + Hys) < In < (HHSP - Hys).	False	True/False (1/0)
LAlm	Digital	Low alarm: True if (LLSP + Hys) < In < (LSP - Hys).	False	True/False (1/0)
LLAlm	Digital	Low-Low alarm: True if In < (LLSP - Hys).	False	True/False (1/0)
State	int	Internal state (0=normal/cleared; set to 0 on reset/disable).	0	0 (debug; always 0 in current logic)

5.13.3 Functional Behavior

The “AAlarm” FB monitors the analog input signal (In) against four configurable setpoints (HHSP, HSP, LSP, LLSP) with hysteresis (Hys) to generate independent high/high-high and low/low-low alarms. The logic is level-triggered: alarms assert when “**In**” enters the respective hysteresis band and remain active (latched) until cleared by a rising-edge on Reset or by disabling the block (Enable=False). The Delay input is reserved for potential future timing (e.g., confirmation delay) but does not affect current behavior. The State output provides a diagnostic indicator (typically 0 for normal operation, updated on reset/disable).

1. **Enable Gating**: If Enable is False, all alarms (HHAlm, HAlm, LAlm, LLAlm) are forced to False, and State = 0 (inactive mode).
2. **Hysteresis Band Evaluation** (when Enable = True):
   - **High-High Alarm (HHAlm)**: Asserts True if In > (HHSP + Hys). Remains True until reset/disable.
   - **High Alarm (HAlm)**: Asserts True if (HSP + Hys) < In < (HHSP - Hys). Remains True until reset/disable.
   - **Low Alarm (LAlm)**: Asserts True if (LLSP + Hys) < In < (LSP - Hys). Remains True until reset/disable.
   - **Low-Low Alarm (LLAlm)**: Asserts True if In < (LLSP - Hys). Remains True until reset/disable.
   - Bands are non-overlapping if setpoints are spaced > 2*Hys; overlaps allow multiple alarms.
3. **Reset Handling**: On rising edge of Reset (False → True), clear all alarms to False and set State = 0.
4. **State Output**: Set to 0 on reset or disable; otherwise holds prior value (diagnostic for alarm latching).

- **Latching Behavior**: Alarms do not auto-de-assert when In exits the band—manual reset or disable required.
- **Hysteresis Role**: Prevents rapid on/off near setpoints (e.g., HH asserts above upper band, de-asserts below lower band).
- **Edge Cases**:
  - Hys = 0: No deadband (alarms toggle at exact setpoints).
  - Unsorted Setpoints: Bands may overlap or invert (e.g., if HSP > HHSP, HAlm undefined).
  - Delay Unused: No confirmation timing; alarms immediate.

5.14 DAlarm Function Block

The DAlarm (Digital Alarm) function block is a delayed digital alarm designed for fault detection and timing monitoring in automation systems. When the input (In) remains steadily at 1 (HIGH) for the specified delay time (Delay), the alarm output (Alm) is activated. If the input returns to 0 (LOW) or the Reset is activated, the alarm is immediately deactivated. Once triggered, the alarm remains active until it is explicitly cleared. This ensures that short pulses do not generate false alarms and only stable conditions (such as a stuck switch or a heartbeat signal issue) are recorded. The alarm status and timing history are stored on disk so that even after a system restart, genuine alarms are preserved, preventing false alarms.

5.14.1 Inputs

Name	Type	Description	Initial Value	Range/Notes
Enable	Digital_HMI	Enable monitoring: True=active (time and alarm), False=clears Alm.	True	True/False (1/0; HMI toggle)
In	Digital	Digital input signal to monitor (e.g., sensor status).	False	True/False (1/0; rising edge starts timer)
Delay	Time_HMI	Delay period: Time steady HIGH before Alm=true (in seconds).	T#5s	Time (e.g., T#1s–T#60s; cast to int seconds)
Reset	Digital_HMI	Reset command: Rising edge (False→True) clears latched Alm.	False	True/False (1/0; HMI button)

5.14.2 Outputs

Name	Type	Description	Initial Value	Range/Notes
Alm	Digital	Alarm output: True after In steady HIGH for Delay; latched until clear.	0	-

5.14.3 Functional Behavior

The DAlarm FB implements a delay-on alarm for digital signals, asserting the Alm output after the input In remains steadily True (HIGH) for the specified Delay period. The alarm latches once triggered and clears on a falling edge of In, rising edge of Reset, or when Enable is False. The logic uses edge detection for triggers and a timestamp comparison for timing, with internal states for latching and persistence.

5.14.4 State Machine Transitions

The block uses a simple three-state machine (monitoring, timing, alarm), with edge- and level-based triggers.

- **Monitoring**: Default; alarm low. Waits for rising input signal.
  - Rising input (low to high) → Timing
- **Timing**: Timer started on rising; alarm low.
  - Input falls (high to low) → Monitoring [clears timer]
  - Timer expires (signal stays high ≥ delay) → Alarm
- **Alarm**: Alarm high; latched.
  - Rising reset (low to high) → Monitoring [clears alarm/timer]

Additional Behaviors:

- **Sustained High During Timing:** If input stays high but timer not expired, remains in Timing.
- **Disable:** Forces Monitoring, alarm low, ignores input.
- **Reset:** Only effective in Alarm; clears to Monitoring regardless of input level.

5.15 AWatch Function Block

The **AWatch** (Analog Watch) function block (FB) is a range watchdog timer for analog signals, designed for detecting sustained stability or "stuck" values within defined bounds in automation systems. It asserts the alarm output (Alm) if the input signal (In) remains continuously within the [Min, Max] range for the full Duration period, latching the alarm until cleared by a rising-edge reset (Reset) or by disabling the block (Enable=False). This prevents false alarms from transient excursions while flagging prolonged normal-range conditions (e.g., frozen sensor).

This FB is ideal for analog fault detection (e.g., monitoring for stuck readings in pressure/temperature sensors) with HMI integration (Digital_HMI/Time_HMI types for visualization). It persists timing and alarm states to disk for retention across restarts, ensuring no spurious alarms on power-up.

5.15.1 Inputs

Name	Type	Description	Initial Value	Range/Notes
Enable	Digital_HMI	Enable monitoring: True=active (time and alarm), False=clears Alm.	True	True/False (1/0; HMI toggle)
In	double	Analog input signal to watch (e.g., sensor value).	0	Any real number (engineering units)
Min	double	Minimum bound: Lower limit of watched range.	0	Any (≤ Max; e.g., 0 for non-negative)
Max	double	Maximum bound: Upper limit of watched range.	100	Any (≥ Min; e.g., 100% scale)
Duration	Time_HMI	Stability duration: Time in-range before Alm=True (in ms).	T#20s	Time (e.g., T#1s–T#60s; cast to int ms)
Reset	Digital_HMI	Reset command: Rising edge (False→True) clears latched Alm.	False	True/False (1/0; HMI button)

5.15.2 Outputs

Name	Type	Description	Initial Value	Range/Notes
Alm	Digital	Alarm output: True if In steady in [Min, Max] for Duration; latched until clear.	0	-

5.15.3 Functional Behavior

The AWatch FB implements a range-based watchdog timer for analog signals, asserting the Alm output if the input In remains continuously within the [Min, Max] bounds for the full Duration period. The alarm latches once triggered and clears on exiting the range, a rising edge of Reset, or when Enable is False. The logic uses range entry detection and timestamp comparison for timing, with internal states for latching and persistence.

5.15.4 State Machine Transitions

The block uses a simple three-state machine (monitoring, timing, alarm), with range-based and time-based triggers.

- **Monitoring**: Default; alarm low. Waits for signal to enter range.
    - Signal enters range [min, max] → Timing
- **Timing**: Timer started on entry; alarm low.
    - Signal exits range (outside [min, max]) → Monitoring [clears timer]
    - Timer expires (signal stays in range ≥ duration) → Alarm
- **Alarm**: Alarm high; latched.
    - Rising reset (low to high) → Monitoring [clears alarm/timer]

Additional Behaviors:

- **Sustained In-Range During Timing:** If signal stays in range but timer not expired, remains in Timing.
- **Disable:** Forces Monitoring, alarm low, ignores signal.
- **Reset:** Only effective in Alarm; clears to Monitoring regardless of signal position.

5.16 DWatch Function Block

The **DWatch** (Digital Watch) function block (FB) is a versatile digital watchdog timer for monitoring signal stability or periodicity, designed for fault detection in automation systems. It asserts the alarm output (Alm) based on the selected Mode:

- **Mode 0 (Low)**: Alarms if in remains LOW (0) for Duration seconds.
- **Mode 1 (High)**: Alarms if in remains HIGH (1) for Duration seconds.
- **Mode 2 (Cyclic)**: Alarms if fewer than CyclNum rising edges (pulses) occur within each Duration window (resets timer periodically, counts cycles).

The alarm latches once triggered and clears on opposite edge, rising-edge Reset, or disable (Enable=False). This FB is ideal for detecting stuck bits, missed heartbeats, or irregular pulsing with HMI integration (Digital_HMI types for visualization). It persists timing, count, and alarm states to disk for retention across restarts.

5.16.1 Inputs

Name	Type	Description	Initial Value	Range/Notes
Enable	Digital_HMI	Enable monitoring: True=active (time/count and alarm), False=clears Alm.	True	True/False (1/0; HMI toggle)
In	Digital	Digital input signal to watch (e.g., status bit).	False	True/False (1/0; edges trigger logic)
Mode	int	Watch mode: 0=Low duration, 1=High duration, 2=Cyclic count.	0	0–2 (0=alarm on steady LOW, 1=steady HIGH, 2=missed pulses)
Duration	int	Duration window (seconds): Time for steady state (Mode 0/1) or per cycle (Mode 2).	20	≥0 (e.g., 5–60s; int seconds)
CyclNum	int	Required cycles/pulses: Number of rising edges needed per Duration (Mode 2 only).	10	≥1 (e.g., 5–20; ignored in Mode 0/1)
Reset	Digital_HMI	Reset command: Rising edge (False→True) clears latched Alm.	False	True/False (1/0; HMI button)

5.16.2 Outputs

Name	Type	Description	Initial Value	Range/Notes
Alm	Digital	Alarm output: True on sustained LOW/HIGH (Mode 0/1) for Duration or insufficient pulses (Mode 2); latched until clear.	0	-

5.16.3 Functional Behavior

The DWatch FB monitors the digital input In for sustained states or periodicity based on the Mode selection, asserting the Alm output after the condition is met for the Duration period. The alarm latches once triggered and clears on the opposite state change (edge), rising edge of Reset, or when Enable is False. For cyclic mode, it counts rising edges over rolling Duration windows. The logic operates on each enabled scan cycle.

5.16.4 State Machine Transitions

The block operates in mode-specific detection loops (monitoring, detecting, alarm), with edge-, level-, and time/count-based triggers.

Common to All Modes:

- **Alarm**: Alarm high; latched.
    - Rising reset (low to high) → Monitoring [clears alarm/timer/counter]
- **Disable:** Forces Monitoring, alarm low, ignores input.

Low Mode (sustained low detection):

- **Monitoring**: Alarm low. Waits for falling input (high to low).
    - Falling input → Detecting
- **Detecting**: Timer started on falling; alarm low.
    - Input rises (low to high) → Monitoring [clears timer]
    - Timer expires (signal stays low ≥ duration) → Alarm

High Mode (sustained high detection):

- **Monitoring**: Alarm low. Waits for rising input (low to high).
    - Rising input → Detecting
- **Detecting**: Timer started on rising; alarm low.
    - Input falls (high to low) → Monitoring [clears timer]
    - Timer expires (signal stays high ≥ duration) → Alarm

Cyclic Mode (rising edge counting):

- **Monitoring**: Alarm low; first-run initializes timer. Counts rising edges in window.
    - Rising input (low to high) → Increments count; if window expires (≥ duration), checks threshold → Alarm if met, else resets window
    - Window expires without threshold → Resets window/timer, continues monitoring

5.17 RunHours Function Block

The **RunHours** function block (FB) is a runtime accumulator for tracking operational hours of a device or process, designed for maintenance scheduling and logging in automation systems. It accumulates minutes and hours when the run signal (In1) is True, rolling over daily totals to yesterday's value on calendar day change, and maintains lifetime and maintenance totals. A rising-edge Reset clears the maintenance counter without affecting other totals. Initial values (Today0, Yes0, Main0, Total0) allow seeding on startup. All counters persist to disk for retention across restarts, ensuring accurate logging even after power cycles.

5.17.1 Inputs

Name	Type	Description	Initial Value	Range/Notes
In1	Bool	Run signal: True = accumulate time (device active).	False	True/False (1/0; edge-triggered start/stop)
Reset	Digital_HMI	Reset maintenance hours: Rising edge (False → True) clears maintenance counter.	False	True/False (1/0; HMI button; lifetime unaffected)
Today0	int	Initial today's minutes: Seed value for today's total on first run.	0	≥0 (minutes; e.g., 0–1440 for day)
Yes0	int	Initial yesterday's minutes: Seed value for yesterday's total on first run.	0	≥0 (minutes)
Main0	int	Initial maintenance hours: Seed value for maintenance counter on first run.	0	≥0 (hours)
Total0	int	Initial total hours: Seed value for lifetime total on first run.	0	≥0 (hours)

5.17.2 Outputs

Name	Type	Description	Initial Value	Range/Notes

Today	int	Today's accumulated minutes: Resets daily; increments when running.	0	≥0 (minutes; rolls to Yes on day change)
Yes	int	Yesterday's total minutes: Receives rollover from Today on day change.	0	≥0 (minutes; static until next rollover)
Main	int	Maintenance hours: Increments with total; cleared by Reset.	0	≥0 (hours; resettable)
Total	int	Lifetime total hours: Non-resettable accumulation of all run time.		

5.17.3 Functional Behavior

The **RunHours** FB accumulates operational time (in minutes and hours) when the run signal (In1) is True, providing daily, yesterday's, maintenance (resettable), and lifetime totals. It seeds initial values from the 0 inputs on first execution and rolls over the daily total to yesterday's on calendar day change (local time). The maintenance total increments alongside lifetime but clears on a rising edge of Reset. All totals update in real-time during run periods, with partial minutes added on stop if ≥30 seconds elapsed since start.

1. **Initialization**: On first execution, set Today's minutes = Today0, Yesterday's minutes = Yes0, Maintenance hours = Main0, Total hours = Total0 (seeds override defaults).
2. **Runtime Accumulation** (when In1 = True):
    - Start timing on transition to True.
    - Increment Today's minutes every 60 seconds of continuous run.
    - Increment minutes counter; every 60 minutes, increment Maintenance hours and Total hours.
    - On transition to False (stop): If elapsed since start ≥30 seconds, add 1 minute to Today's total (partial credit).
3. **Daily Rollover**: On local calendar day change, move Today's total minutes to Yesterday's total minutes, and reset Today's to 0.
4. **Reset Handling**: On rising edge of Reset (False → True), clear Maintenance hours to 0 (Today's, Yesterday's, and Total unaffected).
5. **Outputs**:
    - Today: Current day's accumulated minutes (rolls over daily).
    - Yes: Yesterday's total minutes (receives rollover from previous day).
    - Main: Maintenance hours (increments with Total; resettable via Reset).
    - Total: Lifetime total hours (non-resettable; cumulative run time).

- **Timing Resolution**: 60-second increments for minutes; 30-second threshold for partial stop credit.
- **Edge Cases**:
    - Continuous run across days: Totals carry over correctly with rollover.

- Reset during run: Clears only Maintenance; accumulation continues.
- Short runs (<30s): No credit on stop.
- Non-Binary In1: Treated as False (0) or True (non-zero).

5.18 RawAFilterLP Function Block

The **RawAFilterLP** function block provides low-pass filtering for analog signals, designed to suppress transient noise or glitches while preserving gradual or stable changes. It uses a threshold-based mechanism combined with a fixed time delay to validate signal changes, ensuring that only confirmed, small variations are passed to the output.

5.18.1 Inputs

Name	Type	Default	Description
Enable	bool	True	Activates the filtering logic. When false, the output is immediately set to 0, bypassing all processing.
Signal	double	0	The raw input signal value to be filtered (e.g., from a sensor).
delta	float	5.0	The sensitivity threshold for change detection, in the same units as the Signal. Values below this are considered noise-free and passed directly.

5.18.2 Outputs

Name	Type	Default	Description
Q	double	0	The filtered signal output, representing the smoothed version of the input Signal.

5.18.3 Functional Behavior

5.18.3.1 Initialization

- On first execution (no prior state available), the output Q is set to the current Signal value, establishing it as the baseline.

5.18.3.2 Enabled Operation (Enable = true)

The block continuously monitors changes in the Signal relative to its internally tracked baseline value (the last successfully validated signal).

1. **Small Change Detection**:
    - If the absolute difference between the current Signal and the baseline is less than delta, the change is deemed valid.
    - Q is updated to the current Signal.

    - The baseline is updated to this new Signal value.
2. **Large Change Detection (Potential Transient)**:
    - If the absolute difference is greater than or equal to delta, the change is treated as a potential glitch.
    - Q holds the current baseline value (no update).
    - A 5-second validation timer starts (or resets if already active).
3. **Validation After Hold Period**:
    - During the 5-second hold, Q remains at the baseline value, ignoring the Signal.
    - After 5 seconds elapse:
        - Recompute the absolute difference between the current Signal and the baseline.
        - If now less than delta, accept the change: update Q to the current Signal and set it as the new baseline.
        - If still greater than or equal to delta, continue holding: Q stays at the baseline, and the timer resets for another 5 seconds.

This creates a hysteresis effect: brief spikes are rejected outright, while sustained deviations are only accepted if they stabilize close to the original baseline within the validation window.

5.18.3.3 Disabled Operation (Enable = false)

- Q is forced to 0.
- The internal baseline and timer are preserved but not updated, allowing quick resumption when re-enabled.

5.18.3.4 Edge Cases

- **Zero Input**: If Signal is 0 and no baseline exists, Q initializes to 0.
- **Constant Signal**: Q tracks Signal exactly, as differences are 0 (< delta).
- **Gradual Changes**: Multiple small steps (each < delta) accumulate normally.
- **Permanent Step Change (> delta)**: Q holds the pre-change baseline indefinitely, as repeated validations will fail. Increase delta for such scenarios.
- **Frequent Spikes**: Each new spike resets the timer, ensuring robust rejection.
- **Timing**: The 5-second delay is fixed (based on system seconds) and not user-configurable.

5.19 ChatterFilter Function Block

The ChatterFilter function block is designed to suppress chattering in Boolean digital signals, such as those from switches, relays, or sensors prone to rapid, unintended toggling due to electrical noise, mechanical bounce, or interference. It monitors transitions (on/off changes) over a configurable time window and, if excessive changes are detected, temporarily freezes the output to a stable "last good" value for a recovery period. This prevents false triggers in control systems while allowing normal operation to resume automatically.

5.19.1 Inputs

Name	Type	Default	Description
Enable	bool	True	Activates the filtering logic. When false, the output Q is set to false, but monitoring continues in the background for quick resumption.
Signal	bool	False	The raw boolean input signal to be filtered (e.g., from a switch or sensor).
CBTime	long	1000	Base time window in milliseconds for evaluating signal changes. Defines the period over which transitions are counted.
CVCNum	int	10	Minimum number of signal changes required within CBTime to trigger chatter detection and freezing.
CFTime	long	10000	Freeze duration in milliseconds when chatter is detected. The output holds steady during this period before recovery.

5.19.2 Outputs

Name	Type	Default	Description
Q	bool	False	The filtered boolean output signal. Mirrors the input unless frozen due to detected chattering, in which case it holds the last stable value.
V	bool	False	Chatter validity flag: true when the output is frozen due to detected chattering, false otherwise. Useful for alarming or logging.
N	int	0	Current count of signal changes within the active monitoring window. Resets on evaluation; aids in tuning and diagnostics.

5.19.3 Functional Behavior

When enabled, the block monitors a digital input signal for chattering (excessive transitions) within a sliding time window, counting rising/falling edges. If the change count exceeds a threshold within the window, it detects chatter, freezes the output to the last stable (pre-chatter) value for a configurable freeze duration, and sets a validity flag low during freeze (indicating unreliable input). After freeze, it resumes normal pass-through, resetting the count and window. The block outputs the filtered signal, validity flag, and current change count for diagnostics. Disabled mode passes input directly (or zeros if initial). Retrigger able: New chatter during freeze extends or restarts detection post-freeze. Persistence via file stores signal history, timestamps, counts, last good value, freeze timer, and mode for resumption after restarts (e.g., ongoing freeze completes if time remains).

5.19.4 State Machine Transitions

The block uses a two-state machine (normal, frozen), with edge-count and time-based triggers.

- **Normal**: Outputs current input; counts changes in window; validity high.
  - Change count ≥ threshold within window → Frozen
  - Window expires without threshold → Resets count/window, stays Normal
- **Frozen**: Outputs last good value; validity low; freeze timer active.
  - Freeze timer expires (≥ duration) → Normal [resets count/window]

Additional Behaviors:

- **Every Change:** Increments count (rising/falling edges).
- **Window Reset:** On expiry without threshold, clears count.
- **Disable:** Bypasses to Normal (direct input), validity high, no counting.
- **First Run:** Initializes window/timer immediately.

5.20 Scheduler Function Block

The Scheduler function block provides time-based and event-driven scheduling for periodic activations in control systems, such as timed maintenance alerts, dosing cycles, or batch processes. It triggers an output pulse at configurable intervals within each hour, offset by a pre-schedule lead time, and sustains the activation for a post-schedule duration. Additionally, it supports immediate manual triggering via an event input. The block leverages the local system clock for precise timing and maintains internal state across executions for reliability in cyclic or restarted environments.

5.20.1 Inputs

Name	Type	Default	Description
En	bool	True	Activates the scheduling logic. When false, the output Q is set to false, but internal timers and state are preserved for immediate resumption.
SchT	int	5	Schedule interval in minutes. Defines the periodicity of triggers within each hour (e.g., 5 for every 5 minutes).
PreSchT	int	1	Pre-schedule lead time in minutes. Offsets the trigger earlier by this amount to provide advance notice (e.g., 1 minute before the nominal interval).

PosSchT	int	2	Post-schedule hold time in minutes. Extends the active duration after each trigger (added to PreSchT for total pulse width).
Event	bool	False	Manual trigger input. A rising edge (false to true) immediately activates the output, overriding the schedule.

5.20.2 Outputs

Name	Type	Default	Description
Q	bool	False	Scheduled activation output: true during active periods (trigger pulse or event override), false otherwise. Use to drive relays, alarms, or processes.
CurH	int	0	Current local hour (0-23) from the system clock. Updated on each execution for time-aware logic.
CurM	int	0	Current local minute (0-59) from the system clock. Updated on each execution for time-aware logic.
State	int	0	Internal state indicator: 0 (idle/waiting), 1 (active/holding). Useful for diagnostics or interlocking with other functions.

5.20.3 Functional Behavior

5.20.3.1 Initialization

- On first execution (no prior state), the block enters idle mode (State = 0). The schedule lookup (trigger minutes per hour) is computed based on SchT and PreSchT.
- Trigger minutes are calculated as offsets before multiples of SchT within the hour, wrapping around at 60 minutes (e.g., for SchT=5, PreSchT=1: triggers at minutes 4, 9, 14, ..., 59).
- Outputs CurH and CurM reflect the immediate system time; Q starts as false.

5.20.3.2 Enabled Operation (En = true)

The block monitors the system clock and input event continuously.

1. **Scheduled Triggering** (Idle State):
   - On each minute boundary (detected via CurM change):
     - Checks if the current minute matches a pre-computed trigger point in the hourly schedule.
     - If matched: Sets Q = true, advances to active state (State = 1), and starts a hold timer for (PreSchT + PosSchT) minutes (converted to seconds).
   - No action if no match; remains idle.
2. **Active Hold** (Active State):
   - Q remains true throughout the hold period.

   - On timer expiration: Resets Q = false, returns to idle state (State = 0), and updates the minute reference to prevent immediate re-trigger.
3. **Event Override**:
   - Detects rising edge on Event (false to true).
   - Immediately sets Q = true and enters active state, starting the hold timer regardless of current schedule or state.
   - Multiple events during hold are ignored until the hold completes.

CurH and CurM are refreshed from the local system time on every execution, supporting 24-hour operation with hourly reset of the schedule.

5.20.3.3 Disabled Operation (En = false)

- Q is forced to false.
- Schedule monitoring and event detection pause, but the current state, timers, and last-minute reference are retained for seamless re-enabling.

5.20.3.4 Edge Cases

- **SchT = 0 or 1**: May cause continuous or overly frequent triggers; not recommended—use values ≥5 for stability.
- **PreSchT ≥ SchT**: Overlaps trigger points; the block will still compute unique minutes but may lead to merged pulses.
- **Total Hold > SchT**: Pulses may overlap across cycles; adjust PosSchT to avoid this.
- **Event During Hold**: No effect—hold timer continues unchanged.
- **Minute Boundary Miss**: If execution lags across a minute change, the trigger check occurs on the next cycle; use fast scan rates (e.g., <1 second) for precision.
- **System Time Jump**: Assumes monotonic clock; large jumps (e.g., due to NTP sync) may skip or duplicate triggers—monitor in time-sensitive apps.
- **Hour Rollover**: Schedule resets automatically at minute 0, aligning to the new hour.

5.21 int2Float Function Block

The **int2Float** function block performs a direct type conversion by combining two 16-bit signed integer values into a single 32-bit floating-point number. This is useful in systems where data is transmitted or stored as separate integer components (e.g., over serial links or in legacy protocols) and needs to be reconstructed into a native float format for further processing, such as in SCADA, PLC, or embedded control applications.

The block supports optional swapping of the integer order to accommodate different byte or word endianness conventions, ensuring compatibility across heterogeneous hardware or networks.

5.21.1 Inputs

Name	Type	Default	Description
in1	int	0	First 16-bit signed integer component (typically the low-order word of the float).
in2	int	0	Second 16-bit signed integer component (typically the high-order word of the float).
Swap	bool	False	Enables swapping of in1 and in2 order. When true, in2 becomes the low-order word and in1 the high-order word; when false, no swap occurs. Use to match source endianness.

5.21.2 Outputs

Name	Type	Default	Description
Q	float	0	The reconstructed 32-bit floating-point value from the combined integers. Represents the exact bit pattern of the input words interpreted as a float.

5.21.3 Functional Behavior

5.21.3.1 Core Operation

- The block interprets the two input integers as the two 16-bit halves (words) of a 32-bit float.
- Without swapping (Swap = false): in1 provides the low-order 16 bits, in2 the high-order 16 bits.
- With swapping (Swap = true): in2 provides the low-order 16 bits, in1 the high-order 16 bits.
- The resulting bit pattern is directly cast to a float, preserving the exact IEEE 754 representation without scaling or arithmetic adjustment.
- Output Q updates immediately on input changes, with no hysteresis or validation.

5.21.3.2 Example Scenarios

- **No Swap, Standard Values**: in1 = 0 (0x0000), in2 = 16384 (0x4000) → Q ≈ 1.0 (binary: 0x3F800000).
- **With Swap**: in1 = 16384 (0x4000), in2 = 0 (0x0000), Swap = true → Q ≈ 1.0 (reverses to same as above).
- **Negative Float**: Inputs representing -2.5 (e.g., low word 0x0000, high word 0xC020 for little-endian) yield Q = -2.5.
- **Out-of-Range Integers**: Values outside -32768 to 32767 are truncated to 16-bit signed range before combination.

5.22 int2Long Function Block

The **int2Long** function block performs a direct type conversion by combining two 16-bit unsigned integer values into a single 32-bit unsigned long integer, which is then represented as a double-precision floating-point number for output. This is particularly useful in systems where 32-bit values are serialized or transmitted as separate 16-bit components and need to be reassembled for arithmetic operations, comparisons, or display in control applications like PLCs or SCADA systems. The block assumes a fixed word order (low-order first) without swapping, making it suitable for consistent little-endian data sources.

5.22.1 Inputs

Name	Type	Default	Description
in1	int	0	Low-order 16-bit unsigned integer component (treated as 0-65535; values outside this range wrap modulo 65536).
in2	int	0	High-order 16-bit unsigned integer component (treated as 0-65535; values outside this range wrap modulo 65536).

5.22.2 Outputs

Name	Type	Default	Description
Q	double	0	The reconstructed 32-bit unsigned long value as a double (e.g., in1 + (in2 << 16)). Exact integer representation preserved within double's mantissa.

5.22.3 Functional Behavior

5.22.3.1 Core Operation

- The block casts each input to an unsigned 16-bit integer, discarding any sign or overflow bits.
- Combines them into a 32-bit unsigned value: Q = (in2 * 65536) + in1 (high word shifted left by 16 bits, plus low word).
- The result is directly assigned to a double output, maintaining exact integer precision (doubles can represent integers up to ~2^53 losslessly).
- Updates occur instantly on input changes; no validation, scaling, or additional arithmetic.

5.22.3.2 Example Scenarios

- **Basic Assembly**: in1 = 1, in2 = 0 → Q = 1.0 (binary: 0x00000001).
- **Full Range**: in1 = 65535 (0xFFFF), in2 = 65535 (0xFFFF) → Q = 4294967295.0 (binary: 0xFFFFFFFF).
- **Negative Inputs**: in1 = -1 → treated as 65535 (0xFFFF due to unsigned cast); Q remains positive.

- **Overflow Handling**: in1 = 70000 → wraps to 4464 (70000 % 65536); use upstream modulo if exact clamping is needed.

5.23 Long2Int Function Block

The **Long2Int** function block performs a direct type conversion by splitting a 32-bit unsigned long integer into two separate 16-bit unsigned integer components. This is essential in systems where larger data types must be serialized or transmitted as smaller words for compatibility with hardware constraints or network packet sizes. It is commonly used in industrial automation, data logging, or embedded systems to prepare 32-bit values for downstream processing, such as splitting counters, timestamps, or IDs into register pairs.

5.23.1 Inputs

Name	Type	Default	Description
L1	long	0	The 32-bit unsigned long integer to be split (treated as unsigned; negative or signed inputs may yield unexpected results due to reinterpretation).

5.23.2 Outputs

Name	Type	Default	Description
int0	int	0	Low-order 16-bit unsigned integer component (bits 0-15 of the input, range 0-65535).
int1	int	0	High-order 16-bit unsigned integer component (bits 16-31 of the input, range 0-65535).

5.23.3 Functional Behavior

5.23.3.1 Core Operation

- The input value is cast to an unsigned 32-bit long, preserving its bit pattern.
- It is then divided into two 16-bit unsigned segments:
  - int0 receives the least significant 16 bits (low word).
  - int1 receives the most significant 16 bits (high word, shifted right by 16).
- Outputs update instantaneously on input changes; no additional scaling, signing, or validation is applied.
- The formula is: int0 = L1 % 65536, int1 = L1 / 65536 (integer division).

5.23.3.2 Example Scenarios

- **Basic Split**: L1 = 1 → int0 = 1, int1 = 0.
- **Full Range**: L1 = 4294967295 (0xFFFFFFFF) → int0 = 65535 (0xFFFF), int1 = 65535 (0xFFFF).

- **Mid-Range**: L1 = 16777216 (0x01000000) → int0 = 0, int1 = 65536 (wait, no: 65536 / 65536 = 1, but actually 0x01000000 is 1 << 24, so high=1<<8=256, low=0).
  - Correction: int0 = 0, int1 = 256.
- **Negative Input**: L1 = -1 → Treated as large unsigned (e.g., 4294967295 on 32-bit), yielding int0 = 65535, int1 = 65535.

This allows exact reconstruction downstream by recombining (e.g., L1 = (int1 * 65536) + int0).

5.24 int2Double Function Block

The **int2Double** function block performs a direct type conversion by combining four 16-bit unsigned integer values into a single 64-bit double-precision floating-point number. This is valuable in environments where double-precision data is split across multiple 16-bit registers or words for transmission, storage, or processing. It reconstructs the original IEEE 754 double bit pattern without alteration, enabling seamless integration in applications requiring high-precision arithmetic, such as scientific computations, position calculations, or high-resolution sensor fusion.

5.24.1 Inputs

Name	Type	Default	Description
in1	int	0	Least significant 16-bit unsigned integer component (bits 0-15 of the double).
in2	int	0	Second-lowest 16-bit unsigned integer component (bits 16-31 of the double).
in3	int	0	Second-highest 16-bit unsigned integer component (bits 32-47 of the double).
in4	int	0	Most significant 16-bit unsigned integer component (bits 48-63 of the double).

5.24.2 Outputs

Name	Type	Default	Description
Q	double	0	The reconstructed 64-bit double-precision floating-point value from the combined integers. Exact bit pattern of the input words interpreted as a double.

5.24.3 Functional Behavior

5.24.3.1 Core Operation

- Each input is cast to an unsigned 16-bit integer, ignoring sign and wrapping values modulo 65536.
- The four components are concatenated in order (in1 as lowest word, in4 as highest) to form a 64-bit pattern.
- This pattern is directly reinterpreted as a double-precision float per IEEE 754, without any arithmetic adjustment or normalization.
- Output Q reflects the immediate result of input changes, supporting subnormal, normal, infinite, and NaN representations.

5.24.3.2 Example Scenarios

- **Simple Value**: in1=0 (0x0000), in2=0 (0x0000), in3=0 (0x0000), in4=64 (0x4000) → Q = 1.0 (binary: 0x3FF0000000000000).
- **Large Number**: Inputs representing 3.14159 (π) split into words yield Q ≈ 3.14159.
- **Negative Value**: Inputs for -2.0 (e.g., high word with sign bit set) produce Q = -2.0.
- **Out-of-Range Input**: in1 = 70000 → wraps to 4464 (70000 % 65536); downstream recombination remains consistent.

This facilitates lossless recovery when inputs match the serialized form of the original double.

5.25 Double2Int Function Block

The **Double2Int** function block performs a direct type conversion by decomposing a 64-bit double-precision floating-point number into four separate 16-bit unsigned integer components. This is crucial for serializing high-precision data across constrained interfaces, such as splitting doubles into multiple registers for protocols like Modbus, Profibus, or OPC UA in industrial automation systems. It extracts the exact bit pattern of the IEEE 754 double without modification, facilitating transmission or storage in word-based formats while preserving full precision for reconstruction.

5.25.1 Inputs

Name	Type	Default	Description
ind	double	0	The 64-bit double-precision floating-point value to be split into components.

5.25.2 Outputs

Name	Type	Default	Description

in1	int	0	Least significant 16-bit unsigned component (bits 0-15 of the double).
in2	int	0	Second-lowest 16-bit unsigned component (bits 16-31 of the double).
in3	int	0	Second-highest 16-bit unsigned component (bits 32-47 of the double).
in4	int	0	Most significant 16-bit unsigned component (bits 48-63 of the double).

5.25.3 Functional Behavior

5.25.3.1 Core Operation

- The input double's bit pattern is directly extracted via union reinterpretation.
- It is divided into four 16-bit unsigned segments in little-endian order:
  - in1 gets the lowest 16 bits.
  - in2 gets the next 16 bits.
  - in3 gets the following 16 bits.
  - in4 gets the highest 16 bits.
- Each component is cast to a signed integer, but represents unsigned values (0-65535); no arithmetic or rounding occurs.
- Outputs update immediately upon input change, enabling exact bit-level serialization.

5.25.3.2 Example Scenarios

- **Simple Value**: ind = 1.0 → in1 = 0 (0x0000), in2 = 0 (0x0000), in3 = 0 (0x0000), in4 = 64 (0x4000) (binary: 0x3FF0000000000000).
- **Large Number**: ind ≈ 3.14159 (π) → Components matching its hex: e.g., in1=24352 (0x5F18), in2=56269 (0xDB6F), in3=14352 (0x3810), in4=64 (0x4000).
- **Negative Value**: ind = -2.0 → in1 = 0, in2 = 0, in3 = 0, in4 = -1073741824 (sign bit set in high word, appears negative as signed int).
- **Special Cases**: ind = NaN → Arbitrary but consistent bits across components; ind = Infinity → High word with exponent all 1s, others 0.

Reconstruction is achieved by recombining: (in4 << 48) | (in3 << 32) | (in2 << 16) | in1, then reinterpreting as double.

5.26 Float2Int Function Block

The **Float2Int** function block performs a direct type conversion by decomposing a 32-bit single-precision floating-point number into two separate 16-bit signed integer components. This is essential for serializing or transmitting float data across systems that handle smaller word sizes. It extracts the exact IEEE 754 float bit pattern, allowing optional swapping of the output order to support different endianness conventions, ensuring compatibility in mixed-hardware setups.

5.26.1 Inputs

Name	Type	Default	Description
inf	float	0	The 32-bit single-precision floating-point value to be split into components.
Swap	bool	False	Controls output word order. When true, reverses the low/high assignment for big-endian compatibility; when false, uses standard little-endian order (low word first).

5.26.2 Outputs

Name	Type	Default	Description
in1	int	0	First 16-bit signed integer component (low-order word without swap; high-order with swap). Range: -32768 to 32767.
in2	int	0	Second 16-bit signed integer component (high-order word without swap; low-order with swap). Range: -32768 to 32767.

5.26.3 Functional Behavior

5.26.3.1 Core Operation

- The input float's bit pattern is directly reinterpreted via union to access its 32-bit integer representation.
- It is then divided into two 16-bit signed segments:
  - Without swapping (Swap = false): in1 receives the least significant 16 bits (low word), in2 the most significant 16 bits (high word).
  - With swapping (Swap = true): in1 receives the high word, in2 the low word.
- Outputs reflect the exact bits, with no scaling or modification, enabling bit-identical reconstruction.
- Updates are immediate on input changes, supporting all float representations (normal, subnormal, zero, infinite, NaN).

5.26.3.2 Example Scenarios

- **Simple Value**: inf = 1.0 → Without swap: in1 = 0 (0x0000), in2 = 16384 (0x4000); with swap: in1 = 16384, in2 = 0.
- **Negative Float**: inf = -2.5 → Without swap: in1 = 0 (0x0000), in2 = -1074790400 (sign and exponent bits); with swap: reversed.
- **Full Range**: inf ≈ 3.4028235e38 (max finite) → Components matching its binary: e.g., without swap in1 = 0x7F7F, in2 = 0xFFFF.
- **Special Cases**: inf = NaN → Arbitrary consistent bits; inf = Infinity → High word with all-1s exponent.

Reassembly: Combine as (in2 << 16) | in1 (no swap) or (in1 << 16) | in2 (swap), then reinterpret as float.

5.27 byte2Long Function Block

The **byte2Long** function block performs a direct type conversion by combining four 8-bit unsigned integer values into a single 32-bit unsigned long integer, which is then represented as a double-precision floating-point number for output. This is particularly useful in low-level data protocols or hardware interfaces where 32-bit values are transmitted or stored as individual bytes. It reconstructs the original bit pattern without alteration, enabling efficient deserialization for counters, addresses, or IDs in control applications.

5.27.1 Inputs

Name	Type	Default	Description
in1	int	0	Least significant 8-bit unsigned integer component (byte 0, bits 0-7 of the long).
in2	int	0	Second-lowest 8-bit unsigned integer component (byte 1, bits 8-15 of the long).
in3	int	0	Second-highest 8-bit unsigned integer component (byte 2, bits 16-23 of the long).
in4	int	0	Most significant 8-bit unsigned integer component (byte 3, bits 24-31 of the long).

5.27.2 Outputs

Name	Type	Default	Description
Q	double	0	The reconstructed 32-bit unsigned long value as a double (e.g., in1 + (in2 << 8) + (in3 << 16) + (in4 << 24)). Exact integer precision maintained within double's capabilities.

5.27.3 Functional Behavior

5.27.3.1 Core Operation

- Each input is cast to an unsigned 8-bit integer (0-255), discarding higher bits via modulo 256.
- The four bytes are concatenated in little-endian order: Q = in1 + (in2 * 256) + (in3 * 65536) + (in4 * 16777216).
- The resulting 32-bit unsigned value is assigned directly to a double, preserving exact integer representation (doubles handle integers up to ~2^53 without loss).
- Output updates instantly on any input change; no signing, scaling, or validation applied.

5.27.3.2 Example Scenarios

- **Basic Assembly**: in1 = 1, in2 = 0, in3 = 0, in4 = 0 → Q = 1.0.
- **Full Range**: in1 = 255 (0xFF), in2 = 255 (0xFF), in3 = 255 (0xFF), in4 = 255 (0xFF) → Q = 4294967295.0.

- **Mid-Range**: in1 = 0, in2 = 1, in3 = 0, in4 = 0 → Q = 256.0.
- **Negative/Wrapped Inputs**: in1 = -1 → treated as 255 (0xFF); Q adjusts accordingly for positive unsigned result.

5.28 byte2Float Function Block

The **byte2Float** function block performs a direct type conversion by combining four 8-bit unsigned integer values (treated as bytes) into a single 32-bit single-precision floating-point number. This is designed for deserializing float data that has been transmitted or stored as individual bytes. It reconstructs the exact IEEE 754 float bit pattern, with a built-in safeguard: if all input bytes are 0xFF (255), the output is forced to 0.0 to indicate invalid or sentinel data, preventing propagation of error states.

5.28.1 Inputs

Name	Type	Default	Description
b1	int	0	Least significant 8-bit unsigned byte component (bits 0-7 of the float).
b2	int	0	Second-lowest 8-bit unsigned byte component (bits 8-15 of the float).
b3	int	0	Second-highest 8-bit unsigned byte component (bits 16-23 of the float).
b4	int	0	Most significant 8-bit unsigned byte component (bits 24-31 of the float).

5.28.2 Outputs

Name	Type	Default	Description
Q	float	0	The reconstructed 32-bit single-precision floating-point value from the byte pattern, or 0.0 if all inputs are 255 (0xFF). Exact IEEE 754 bit interpretation applied.

5.28.3 Functional Behavior

5.28.3.1 Core Operation

- Each input is cast to an unsigned 8-bit value (0-255), wrapping larger or negative inputs modulo 256.
- The four bytes are concatenated in little-endian order to form a 32-bit pattern: Q is the reinterpretation of b1 | (b2 << 8) | (b3 << 16) | (b4 << 24) as a float.
- **Sentinel Check**: If all four inputs equal 255 (0xFF), Q is explicitly set to 0.0, overriding the bit pattern to provide a neutral default for error detection (e.g., uninitialized or corrupted data).
- Otherwise, the bit pattern is directly cast to a float, supporting normals, subnormals, zeros, infinities, and NaNs.
- Output updates immediately on input changes, with no further processing.

5.28.3.2 Example Scenarios

- **Standard Reconstruction**: b1 = 0 (0x00), b2 = 0 (0x00), b3 = 128 (0x80), b4 = 63 (0x3F) → Q = 1.0 (binary: 0x3F800000).
- **All Sentinel**: b1 = 255, b2 = 255, b3 = 255, b4 = 255 → Q = 0.0 (forced default, ignoring 0xFFFFFFFF pattern which would be NaN).
- **Negative Value**: b1 = 0 (0x00), b2 = 0 (0x00), b3 = 0 (0x00), b4 = 192 (0xC0) → Q = -1.0 (sign bit set).
- **Wrapped Input**: b1 = 256 → treated as 0 (256 % 256); b2 = -1 → treated as 255.

5.29 bcd2dec Function Block

The **bcd2dec** function block converts a 12-digit Binary Coded Decimal (BCD) representation into a standard decimal number. BCD encoding packs two decimal digits into each 8-bit byte (one per 4-bit nibble). This block takes six BCD bytes—representing digits 11-10 (most significant) down to 1-0 (least significant)—and reconstructs the full 12-digit integer as a double-precision output, suitable for arithmetic, display, or logging in control systems.

5.29.1 Inputs

Name	Type	Default	Description
in1	int	0	Most significant BCD byte (holds digits 11 and 10; e.g., 0x12 for digits 1 and 2).
in2	int	0	Second-most significant BCD byte (holds digits 9 and 8).
in3	int	0	Third BCD byte (holds digits 7 and 6).
in4	int	0	Fourth BCD byte (holds digits 5 and 4).
in5	int	0	Fifth BCD byte (holds digits 3 and 2).
in6	int	0	Least significant BCD byte (holds digits 1 and 0).

5.29.2 Outputs

Name	Type	Default	Description
Q	double	0	The converted 12-digit decimal number (e.g., BCD bytes for 123456789012 yield 123456789012.0).

5.29.3 Functional Behavior

5.29.3.1 Core Operation

- Each input byte is interpreted as a packed BCD pair: high nibble (bits 7-4) as the tens digit (0-9), low nibble (bits 3-0) as the units digit (0-9).
- The two-digit decimal value from each byte is extracted and positioned:

  - in1: Placed at 10^10 (for digits 11-10).
  - in2: Placed at 10^8 (for digits 9-8).
  - in3: Placed at 10^6 (for digits 7-6).
  - in4: Placed at 10^4 (for digits 5-4).
  - in5: Placed at 10^2 (for digits 3-2).
  - in6: Placed at 10^0 (for digits 1-0).
- The sum forms the full 12-digit integer, output as a double.
- Conversion is exact for valid BCD; outputs update instantly on input changes.

5.29.3.2 Example Scenarios

- **Small Value**: in1=0 (00), in2=0 (00), in3=0 (00), in4=0 (00), in5=0 (00), in6=12 (0x0C for digits 1-2) → Q = 12.0.
- **Full 12 Digits**: in1=1 (0x01 for 00, but wait: for 123456789012, in1=0x01 (digits 0 1? No: digits 1 2 3... so in1=0x12 (1=1,2=2? Wait, high nibble=1 (digit11=1), low=2 (digit10=2), but for 123... in1=0x01 (digit11=0? Example: for 123456789012, digit11=1,10=2 → in1=0x12; digit9=3,8=4 → in2=0x34; ... in6=0x90 (9=9,0=1? Digits 1=1,0=2? Wait, LSB digits 1=1,0=2 → in6=0x12 (high=1 digit1, low=2 digit0). → Q = 123456789012.0.
- **Leading Zeros**: in1=0 (00), others for 000123 → Q = 123.0 (preserves value, not string).
- **Invalid BCD**: in6=10 (0x0A, low nibble A>9) → Extracts as 10 (but mathematically 0*10 + 10=10, though non-standard; output 10.0 if positioned accordingly—recommend validation upstream.

5.30 Float2Int Function Block

The **Float2Int** function block performs a direct type conversion by decomposing a 32-bit single-precision floating-point number into four separate 8-bit unsigned integer components (bytes). This is vital for serializing float data into byte streams for transmission over byte-limited protocols. It extracts the exact IEEE 754 float bit pattern without modification, facilitating compact storage or network efficiency while allowing bit-identical reconstruction downstream.

5.30.1 Inputs

Name	Type	Default	Description
inf	float	0	The 32-bit single-precision floating-point value to be decomposed into byte components.
Swap	Bool	0	The Float data type is a 32-bit type. After conversion, the first 16 bits are transferred to in1, and the second 16 bits to in2. When this input is active, in1 and in2 are swapped.

5.30.2 Outputs

Name	Typ e	Default	Description
In1	int	0	The 16 most significant bits of the Float data type are placed in this output.
In2	int	0	The 16 least significant bits of the Float data type are placed in this output.

5.30.2.1 Example Scenarios

- **Value**: inf = 1.1, Swap=0 → in1 = CCCD=52429, in2=3F8C=16268

  inf = 1.1, Swap=1 → in1 =3F8C=16268, in2= CCCD=52429

5.31 int2byte Function Block User

The int2byte function block performs a direct type conversion by decomposing a 16-bit signed integer into two separate 8-bit signed integer components (bytes). It extracts the exact bit pattern of the input without modification, enabling compact data packaging while preserving sign extension for negative values.

5.31.1 Inputs

Name	Type	Default	Description
in	int	0	The 16-bit signed integer value to be split into byte components.

5.31.2 Outputs

Name	Type	Default	Description
b1	int	0	Least significant 8-bit signed byte component (bits 0-7 of the input). Range: -128 to 127.
b2	int	0	Most significant 8-bit signed byte component (bits 8-15 of the input). Range: -128 to 127.

5.31.3 Functional Behavior

5.31.3.1 Core Operation

- The input value is cast to a 16-bit signed short, preserving its bit pattern.
- It is divided into two 8-bit signed segments in little-endian order:
  - b1 receives the least significant byte (bits 0-7).
  - b2 receives the most significant byte (bits 8-15).

- Sign extension is maintained: for negative inputs, the high byte will be negative (e.g., -1 yields b1=-1, b2=-1).
- Outputs update instantaneously on input changes; no arithmetic or validation is performed.

5.31.3.2 Example Scenarios

- **Positive Small**: in = 1 → b1 = 1, b2 = 0.
- **Full Positive**: in = 32767 → b1 = 255 (0xFF, but signed -1? Wait, 0x7FFF: b1=0xFF=255 unsigned but -1 signed? No: low=0xFF=-1 signed? 32767=0x7FFF: low=0xFF? 7FFF hex: low=FF (255/-1 signed), high=7F (127). But signed int output: b1=-1, b2=127? No, for 0x7FFF, low byte 0xFF signed is -1, high 0x7F=127.
  - Correction: b1 = -1 (0xFF), b2 = 127 (0x7F).
- **Negative**: in = -1 → b1 = -1 (0xFF), b2 = -1 (0xFF).
- **Zero**: in = 0 → b1 = 0, b2 = 0.
- **Edge**: in = -32768 (0x8000) → b1 = 0 (0x00), b2 = -128 (0x80).

Reassembly: (b2 << 8) | (b1 & 0xFF) with sign extension to 16-bit.

5.32 Hysteresis Function Block

The **Hysteresis** function block implements a three-level hysteresis controller for analog signals, preventing rapid toggling around a setpoint in applications. It uses asymmetric high and low hysteresis bands to create dead bands, ensuring stable output transitions: the main output activates above the setpoint and deactivates only after falling below the low band, while a secondary high flag indicates excursion into the upper band. The block is stateful, retaining its last known state across executions or restarts for continuity in cyclic or fault-tolerant environments.

5.32.1 Inputs

Name	Type	Default	Description
Enable	bool	True	Activates the hysteresis logic. When false, both outputs are set to false, but the internal state is preserved for quick resumption.
Signal	double	0	The analog input signal value (e.g., from a sensor or process variable) to evaluate against the setpoint and bands.
SetPoint	double	50	The central threshold value. The main output activates when the signal reaches or exceeds this level.
HysHigh	double	5	Upper hysteresis band width (added to SetPoint). Defines the range above the setpoint where the high flag can activate.
HysLow	doubl	5	Lower hysteresis band width (subtracted from SetPoint). Defines the range below

	e		the setpoint where deactivation occurs.

5.32.2 Outputs

Name	Type	Default	Description
Q	bool	False	Primary hysteresis output: true when the signal is at or above the setpoint (and within bands), false when below the low band. Use for on/off control.
High	bool	False	Secondary high-band indicator: true only when the signal exceeds the upper band (SetPoint + HysHigh), false otherwise. Useful for alarms or mode switching.

5.32.3 Functional Behavior

5.32.3.1 Initialization

- On first execution (no prior state), the block starts in the inactive state (below setpoint). Outputs are Q = false, High = false.
- The internal state tracks whether the signal has crossed into active regions, enabling hysteresis.

5.32.3.2 Enabled Operation (Enable = true)

The block uses a three-state machine to manage transitions based on the signal relative to the setpoint and bands:

1. **Inactive State** (Signal below SetPoint):
   - Q = false, High = false.
   - Remains inactive until Signal >= SetPoint.
   - Transition: To active state, setting Q = true.
2. **Active State** (Signal between SetPoint - HysLow and SetPoint + HysHigh):
   - Q = true, High = false.
   - Holds activation within the symmetric or asymmetric deadband.
   - Transitions:
     - If Signal > SetPoint + HysHigh: To high state, High = true.
     - If Signal < SetPoint - HysLow: To inactive state, Q = false.
3. **High State** (Signal above SetPoint + HysHigh):
   - Q = true, High = true.
   - Indicates over-excursion for enhanced monitoring.
   - Transition: If Signal <= SetPoint + HysHigh (and still >= SetPoint - HysLow): Back to active state, High = false.

This creates a non-symmetric hysteresis loop: activation at SetPoint, deactivation at SetPoint - HysLow, with High flagging above SetPoint + HysHigh.

5.32.3.3 Disabled Operation (Enable = false)

- Forces Q = false, High = false.
- Internal state and history are retained, allowing immediate hysteresis resumption upon re-enabling without re-crossing thresholds.

5.32.3.4 Edge Cases

- **Equal to SetPoint**: Activates Q = true from inactive; holds in active state.
- **Zero Bands (HysHigh = HysLow = 0)**: Behaves as a simple comparator (on >= SetPoint, off < SetPoint); High never activates.
- **Negative Bands**: May cause erratic behavior (e.g., immediate deactivation)—avoid; use absolute values.
- **Signal Jump Across Bands**: Direct transition to appropriate state (e.g., from below low to above high: Q=true, High=true).
- **Asymmetric Bands**: HysHigh > HysLow widens upper tolerance; ideal for processes with directional bias.
- **Constant Signal**: Stabilizes in corresponding state; no toggling.

5.33 MkDnpDOBS Function Block

The **MkDnpDOBS** function block unpacks a packed 64-bit status value into the constituent fields of a DNP3 (Distributed Network Protocol) Control Relay Output Block (CROB), specifically for binary output control operations in SCADA and utility automation systems. DNP3 CROB (Object Group 12, Variation 1) is used to issue precise control commands like pulse-on, latch-on, or trip/close actions on digital outputs, with parameters for queuing, timing, and status. This block extracts the control code, count, on-time, and off-time from the low-order bytes of the input status, assuming a little-endian byte layout in the packed value and big-endian interpretation for the 16-bit time fields.

5.33.1 Inputs

Name	Type	Default	Description
Status	int	0	The packed 64-bit unsigned integer containing the DNP3 CROB data in little-endian byte order. Low bytes hold: byte 0 (control code), byte 1 (count), bytes 2-3 (on-time), bytes 4-5 (off-time). Higher bytes are unused.

5.33.2 Outputs

Name	Type	Default	Description
CC	int	0	Control Code: 8-bit unsigned value (0-255) from the least significant byte. Encodes operation type (e.g., bits 0-1: select/operate; bit 5-4: trip/close; bit 3-2:

			pulse/latch). Refer to DNP3 spec for bit meanings.
Count	int	0	Count: 8-bit unsigned queue count (0-255), specifying number of repetitions for the control action.
OnTime	int	0	On-Time: 16-bit unsigned value (0-65535 ms) from bytes 2-3, interpreted big-endian (MSB first). Duration the output stays active per cycle.
OffTime	int	0	Off-Time: 16-bit unsigned value (0-65535 ms) from bytes 4-5, interpreted big-endian (MSB first). Duration the output stays inactive between cycles.

5.33.3 Functional Behavior

5.33.3.1 Core Operation

- The input Status is treated as an unsigned 64-bit value and unpacked byte-by-byte from the least significant end.
- **Control Code (CC)**: Directly the lowest byte (bits 0-7 of Status).
- **Count**: The next byte (bits 8-15 of Status).
- **OnTime**: Bits 16-31 of Status, with the higher 8 bits (byte 2) as MSB and lower 8 bits (byte 3) as LSB (big-endian assembly: (byte2 << 8) | byte3).
- **OffTime**: Bits 32-47 of Status, similarly big-endian: (byte4 << 8) | byte5.
- Bytes 6-7 (bits 48-63) are discarded.
- Outputs update immediately on input changes; no validation of DNP3 conformance (e.g., valid control code bits) is performed—handle upstream.

5.33.3.2 Example Scenarios

- **Simple Latch-On**: Status packed as 0x0000000000000100 (little-endian bytes: CC=1 (latch-on), Count=0, OnTime=0, OffTime=0) → CC=1, Count=0, OnTime=0, OffTime=0.
- **Pulse with Timing**: For CC=3 (pulse-on), Count=1, OnTime=500 ms (0x01F4 big-endian: byte2=0x01, byte3=0xF4), OffTime=1000 ms (0x03E8: byte4=0x03, byte5=0xE8), Status low 48 bits: bytes [3,1,1,244,3,232] in little-endian ull → OnTime = (1<<8)|244 = 500, OffTime=(3<<8)|232=1000.
- **Max Values**: Status with bytes 0xFF,0xFF,0xFF,0xFF,0xFF,0xFF → CC=255, Count=255, OnTime=65535, OffTime=65535.
- **Zero Input**: All outputs 0.

5.34 MkDnpDOBSR Function Block

The **MkDnpDOBSR** function block packs the components of a DNP3 (Distributed Network Protocol) Control Relay Output Block (CROB) or similar binary output status response into a single 48-bit integer value, suitable for transmission or internal buffering in SCADA, substation automation, or utility control systems. DNP3 CROB (Object Group 12) defines structured commands for precise relay operations, including trip/close actions with timing and queuing. This block assembles the trip/close code, operation type, count, on-time, and off-time into a compact packed format, assuming little-endian byte order for the overall structure and big-endian for 16-bit time fields to match DNP3 wire conventions.

5.34.1 Inputs

Name	Type	Default	Description
OPType	int	0	Operation Type: 6-bit unsigned value (0-63) encoding the lower bits of the DNP3 control code. Includes: bits 0-1 (select/operate: 00=NUL, 01=SELECT, 10=OPERATE), bits 2-3 (operation: 00=PULSE ON, 01=LATCH ON, 10=PULSE OFF, 11=LATCH OFF), bit 4 (queue: 0=do not queue, 1=queue), bit 5 (clear: 0=do not clear, 1=clear).
TCC	int	0	Trip/Close Code: 2-bit unsigned value (0-3) for the upper bits of the control code. 00=NUL, 01=TRIP, 10=CLOSE, 11=TRIP then CLOSE.
Count	int	0	Count: 8-bit unsigned queue repetition count (0-255), specifying how many times to execute the operation.
OnTime	int	0	On-Time: 16-bit unsigned value (0-65535 ms) for the active pulse duration per cycle. Assumed big-endian packing.
OffTime	int	0	Off-Time: 16-bit unsigned value (0-65535 ms) for the inactive interval between cycles. Assumed big-endian packing.

5.34.2 Outputs

Name	Type	Default	Description
DOBSV	int	0	Packed DNP3 CROB/Status Response Value: 48-bit unsigned integer combining all inputs. Structure: bits 0-7 (control code = (TCC << 6)

5.34.3 Functional Behavior

5.34.3.1 Core Operation

- Computes the 8-bit control code as (TCC << 6) | OPType, placing TCC in bits 7-6 and OPType in bits 0-5.
- Assembles the packed value:
  - Bits 0-7: Control code.
  - Bits 8-15: Count.
  - Bits 16-23: Low byte of OnTime.
  - Bits 24-31: High byte of OnTime.
  - Bits 32-39: Low byte of OffTime.

  - Bits 40-47: High byte of OffTime.
- Note: The packing shifts OnTime by 16 bits and OffTime by 32 bits effectively (via multiplication), ensuring non-overlapping placement for full 16-bit fields. The result is output as a signed integer but represents an unsigned value.
- Updates occur immediately on any input change; no range clipping or error handling.

5.34.3.2 Example Scenarios

- **Latch-On Trip**: OPType=1 (01=LATCH ON, others 0), TCC=1 (TRIP), Count=0, OnTime=0, OffTime=0 → Control code = (1<<6) | 1 = 65 (0x41), DOBSV = 65.
- **Pulse with Timing**: OPType=0 (PULSE ON, queue=0), TCC=0 (NUL), Count=1, OnTime=500 (0x01F4), OffTime=1000 (0x03E8) → Control code=0, DOBSV = 0 + (1<<8) + (500<<16) + (1000<<32) = 1<<8 + 500<<16 + 1000<<32 (large number ~4.29e9 + ...).
- **Max Queue**: OPType=16 (queue=1, others 0), TCC=3 (TRIP+CLOSE), Count=255, OnTime=65535, OffTime=65535 → Full-range packing yielding maximum value.
- **Zero Inputs**: DOBSV=0.

5.35 MkDnpDIE Function Block

The **MkDnpDIE** function block constructs a packed 64-bit Digital Input Event (DIE) value for DNP3 (Distributed Network Protocol) reporting by embedding the digital input signal state into the most significant byte of the provided timestamp. This compact format is designed for efficient transmission of binary input change events with time-of-occurrence in SCADA, substation automation, or utility monitoring systems, aligning with DNP3 Binary Input Event structures (e.g., Group 2, Variation 2). The signal value (typically a binary 0 or 1) overwrites the high byte of the timestamp, assuming the timestamp's high byte is unused or zero to avoid data loss. This enables a single 64-bit integer to represent both the event state and timing for streamlined protocol handling.

5.35.1 Inputs

Name	Type	Default	Description
DI	int	0	The digital input signal value (0 = off/false, 1 = on/true; range 0-255, but typically binary). Placed as the high byte of the output.
TL	ulong	0	The 64-bit timestamp or time label (e.g., milliseconds since epoch or system ticks), providing the time-of-occurrence for the event. Lower 56 bits preserved.

5.35.2 Outputs

Nam	Type	Default	Description

e			
DIE	ulong	0	The packed 64-bit Digital Input Event value: (DI << 56)

5.35.3 Functional Behavior

5.35.3.1 Core Operation

- The input timestamp TL is masked to retain only its lower 56 bits.
- The signal DI is shifted left by 56 bits (effectively placing it in the most significant byte).
- The output DIE is the bitwise OR of these, embedding the signal in the high byte while preserving the timestamp's lower portion.
- Results update instantly on input changes, with no validation (e.g., DI range or timestamp format).

This packing assumes little-endian byte order in the 64-bit value, where the high byte (byte 7) holds the signal, and bytes 0-6 hold the timestamp's low-to-high bytes.

5.35.3.2 Example Scenarios

- **Off Event**: DI = 0, TL = 0x123456789ABCDEF0 → DIE = 0x00123456789ABCDEF0 (high byte 0x00, low from TL).
- **On Event**: DI = 1, TL = 0x123456789ABCDEF0 → DIE = 0x01123456789ABCDEF0 (high byte 0x01, timestamp low 56 bits unchanged).
- **With Flags**: DI = 5 (binary 101, e.g., value=1 + flag bit 2), TL = 0x00000000FFFFFFFF → DIE = 0x0500000000FFFFFF (high byte 0x05).
- **Zero Inputs**: DIE = 0.
- **High DI**: DI = 255 (0xFF), TL = 0 → DIE = 0xFF00000000000000.

5.36 MkDnpTime Function Block

The **MkDnpTime** function block generates a DNP3-compatible 64-bit timestamp representing the current system time in milliseconds since the Unix epoch (January 1, 1970, 00:00:00 UTC). This is essential for time-tagging events, responses, or data in DNP3 (Distributed Network Protocol) communications within SCADA, substation automation, or industrial control systems, ensuring synchronized logging and sequencing across devices. The block supports optional adjustment for time-zone offsets to produce UTC-aligned timestamps, making it suitable for global deployments where local time variations must be normalized.

5.36.1 Inputs

Name	Type	Default	Description
UTC	int	0	Time mode flag: 0 = adjust local system time to UTC (subtract timezone offset), 1 = output raw system time without adjustment (assumes input system is already UTC-aligned).

5.36.2 Outputs

Name	Type	Default	Description
DNPT	ulong	0	The 64-bit DNP3 timestamp in milliseconds since Unix epoch (UTC). On Windows CE platforms, defaults to 0 (no time support).

5.36.3 Functional Behavior

5.36.3.1 Core Operation

- Acquires the current system time using high-resolution APIs (nanosecond precision on Linux, microsecond on x86 variants).
- If UTC mode is 0:
    - Computes the local-to-UTC offset (e.g., current timezone difference in seconds).
    - Adjusts the timestamp by subtracting the offset to yield UTC milliseconds.
- If UTC mode is 1:
    - Outputs the raw system time in milliseconds, assuming the system clock is UTC.
- Conversion formula: (seconds * 1000) + (sub-millisecond fraction), where sub-millisecond is from nanoseconds (divided by 1,000,000) or microseconds (divided by 1,000).
- Output updates on every execution, reflecting the instantaneous system time.
- On Windows CE: Always outputs 0, as no time acquisition is implemented (use external synchronization).

5.36.3.2 Example Scenarios

- **UTC Adjustment (UTC=0)**: On a system in EST (-5 hours from UTC), at 2025-10-29 12:00:00 local, outputs ~1,753,920,000,000 ms (adjusted to UTC 17:00:00).
- **Raw Time (UTC=1)**: Same time, outputs local-equivalent ms (~1,753,916,800,000 ms) if system clock is local—ensure system is UTC for accuracy.
- **Zero Input**: UTC=0 yields current UTC ms; UTC=1 yields raw ms.
- **Edge Case**: At epoch (1970-01-01 00:00:00 UTC), outputs 0.

5.37 ReadSysInfo Function Block

The function block monitors key system resource metrics on Linux-based embedded or industrial platforms, providing periodic readings of CPU usage, available RAM, free internal flash storage, and free external SD card space. This is valuable for health monitoring, alerting, or logging in control systems, IoT gateways, or RTU/PLC environments to detect overloads, memory leaks, or storage exhaustion proactively. Updates occur at user-defined intervals to minimize overhead, with values persisted across executions for continuity during restarts or cyclic scans.

5.37.1 Inputs

Name	Type	Default	Description
Enable	bool	True	Activates periodic sampling. When false, no new readings are taken, but last known outputs are retained and provided.
TTime	Time	T#10s	Update interval in time format (e.g., T#5s for 5 seconds). Controls how often metrics are refreshed when enabled. Shorter periods increase accuracy but system load.

5.37.2 Outputs

Name	Type	Default	Description
CPU	double	0.0	CPU usage percentage (0.0-100.0). Represents average utilization over the update period; initializes and persists at 0.0 if unsampled.
RAM	double	0.0	Free RAM in bytes (e.g., available heap or system memory). Updated periodically; negative readings clamped to 0.0.
Flash	double	0.0	Free internal flash storage in bytes (e.g., eMMC or onboard NAND). Persists last known value; initializes at 0.0.
SD	double	0.0	Free external SD card storage in bytes. Updated periodically; negative readings clamped to 0.0.

5.37.3 Functional Behavior

5.37.3.1 Initialization

- On first execution (no prior state), all outputs initialize to 0.0.
- Subsequent runs load persisted values from storage, providing immediate historical metrics.

5.37.3.2 Enabled Operation (Enable = true)

- Compares current system time against the last update timestamp.
- If the elapsed time exceeds TTime (converted to milliseconds):

  - Refreshes the timestamp.
  - Samples free RAM and free SD space using platform-specific queries.
  - Clamps any negative results to 0.0 for validity.
  - CPU usage and free internal flash remain at their last persisted values (no active sampling in current implementation—consider extensions for full monitoring).
- Outputs always reflect the current (or last) sampled values, enabling continuous reporting even between updates.

5.37.3.3 Disabled Operation (Enable = false)

- No sampling occurs; the timer does not advance.
- Outputs continue to provide the most recent persisted values, allowing passive monitoring or fallback during maintenance.

5.37.3.4 Edge Cases

- **TTime = T#0s**: Behaves as continuous sampling (updates every cycle), but may increase load—avoid for long-running systems.
- **Negative/Invalid Readings**: Automatically set to 0.0 for RAM and SD; CPU and Flash stay at 0.0 if unsampled.
- **Storage Full**: If persistence fails (e.g., no write access), falls back to 0.0 on next boot.
- **Platform Differences**: On Windows CE, all outputs fixed at 0.0 regardless of enable or period—integrate custom APIs for equivalent metrics.
- **Constant Enable**: Steady-state reporting with periodic refreshes for RAM/SD; CPU/Flash require manual or extended sampling.

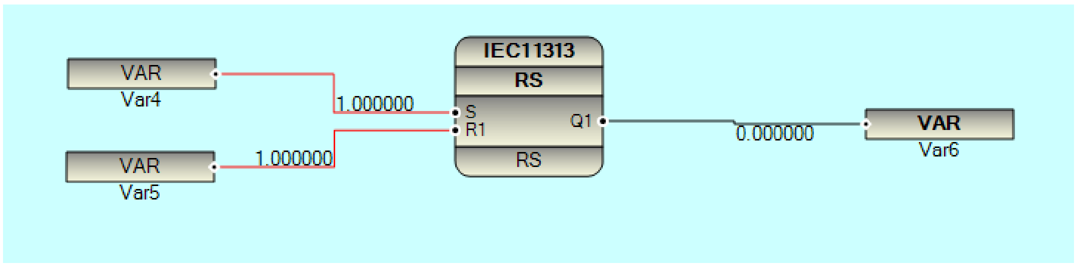
6 IEC1131-3 Group

All Function blocks in this group are based on IEC1131-3 Standard. You will find some of this Function Blocks in Other group with different names. Function Block Name, Input Output Pins and Function definition is completely based on IEC1131-3 Standard.

6.1 RS Description

The RS function block is a Reset-Set Flip Flop that conforms to the IEC 61131-3 standard. The Reset input (R1) has higher priority than the Set input (S) in determining the output value. The operation is as follows:

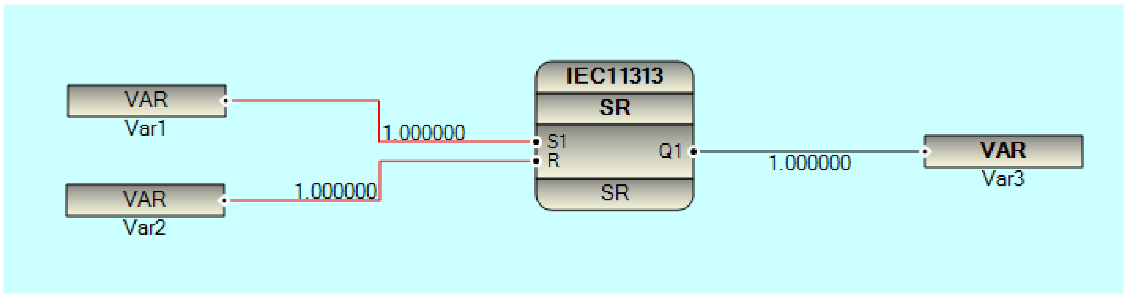
- **Output Logic**:
    - When the Reset input (R1) transitions from 0 to 1, the output (Q) is set to 0.
    - When the Set input (S) transitions from 0 to 1 and R1 is not active, the output (Q) is set to 1.
    - If both R1 and S transition from 0 to 1 simultaneously, R1 takes precedence, and Q is set to 0.



6.2 SR Description

The SR function block is a Set-Reset Flip Flop that conforms to the IEC 61131-3 standard. The Set input (S) has higher priority than the Reset input (R1) in determining the output value. The operation is as follows:

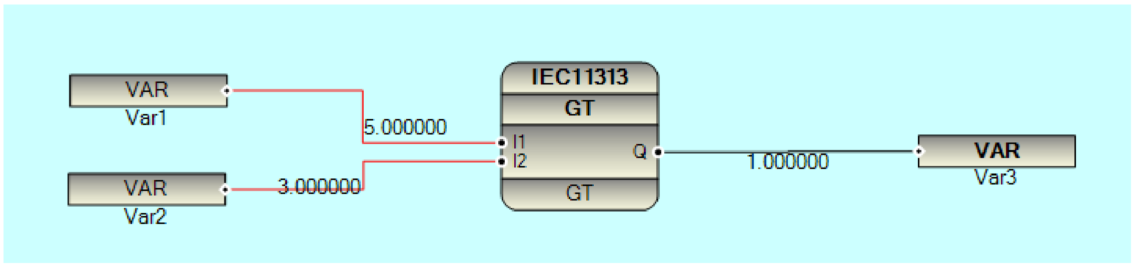
- **Output Logic**:
    - When the Set input (S) transitions from 0 to 1, the output (Q) is set to 1.
    - When the Reset input (R1) transitions from 0 to 1 and S is not active, the output (Q) is set to 0.
    - If both S and R1 transition from 0 to 1 simultaneously, S takes precedence, and Q is set to 1.



6.3 GT Description

The GT (Greater Than) function block operates as follows:

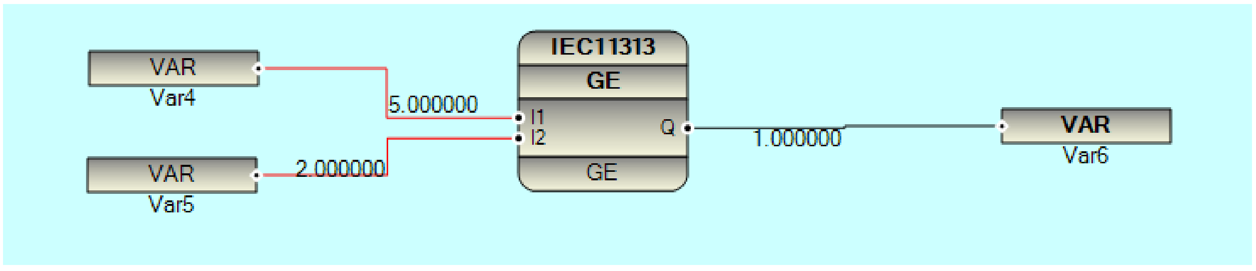
- **Output Logic**: The output (Q) is set to True when Input 1 is greater than all other inputs. Otherwise, the output (Q) is set to False.



6.4 GE Description

The GE (Greater Than or Equal) function block operates as follows:

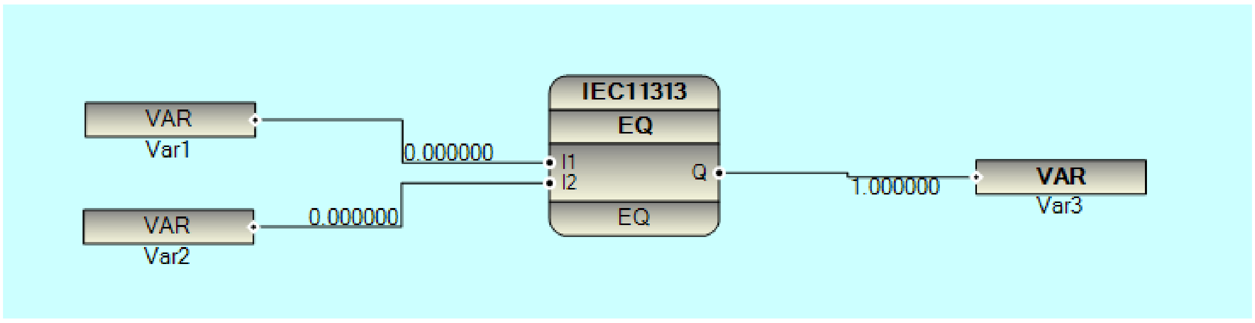
- **Output Logic**: The output (Q) is set to True when Input 1 is greater than or equal to all other inputs. Otherwise, the output (Q) is set to False.



6.5 EQ Description

The EQ (Equal) function block operates as follows:

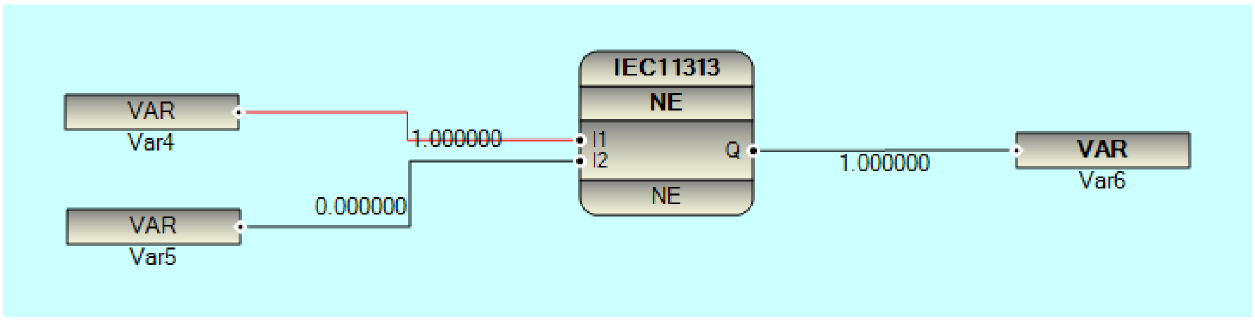
- **Output Logic**: The output (Q) is set to True when all inputs have equal values. Otherwise, the output (Q) is set to False.



6.6 NE Description

The NE (Not Equal) function block operates as follows:

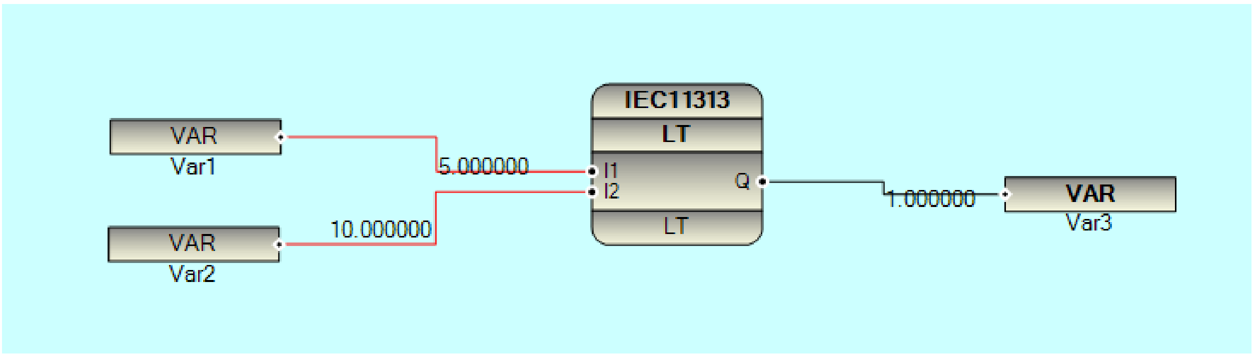
- **Output Logic**: The output (Q) is set to False when all inputs have equal values. Otherwise, the output (Q) is set to True.



6.7 LT Description

The LT (Less Than) function block operates as follows:

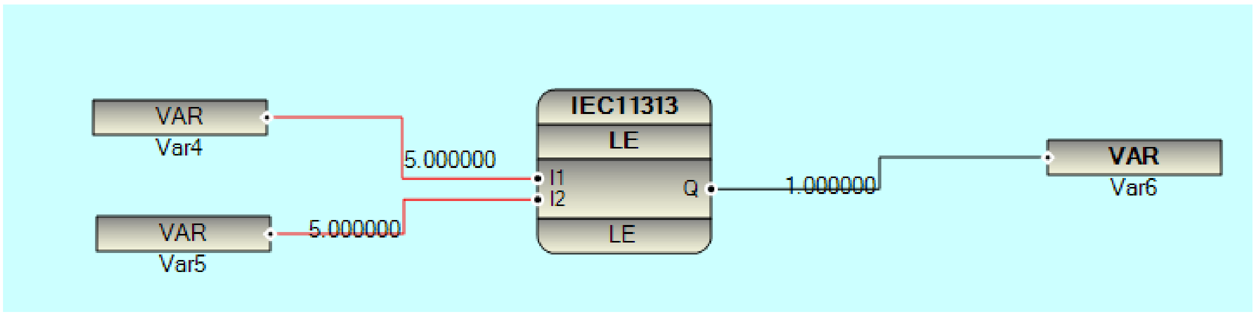
- **Output Logic**: The output (Q) is set to True when Input 1 is less than all other inputs. Otherwise, the output (Q) is set to False.



6.8 LE Description

The LE (Less Than or Equal) function block operates as follows:

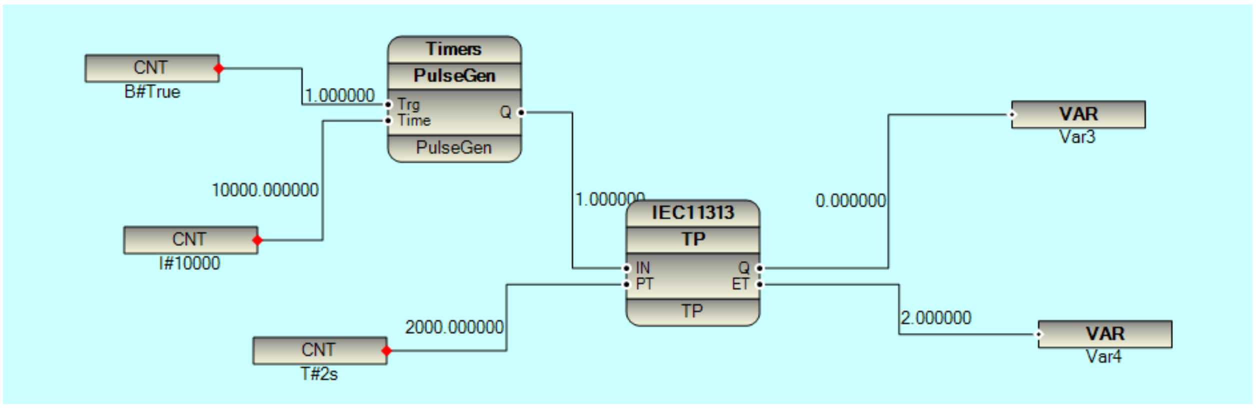
- **Output Logic**: The output (Q) is set to True when Input 1 is less than or equal to all other inputs. Otherwise, the output (Q) is set to False.



6.9 TP Description

The TP (Timer Pulse) function block is a pulse timer that operates as follows:

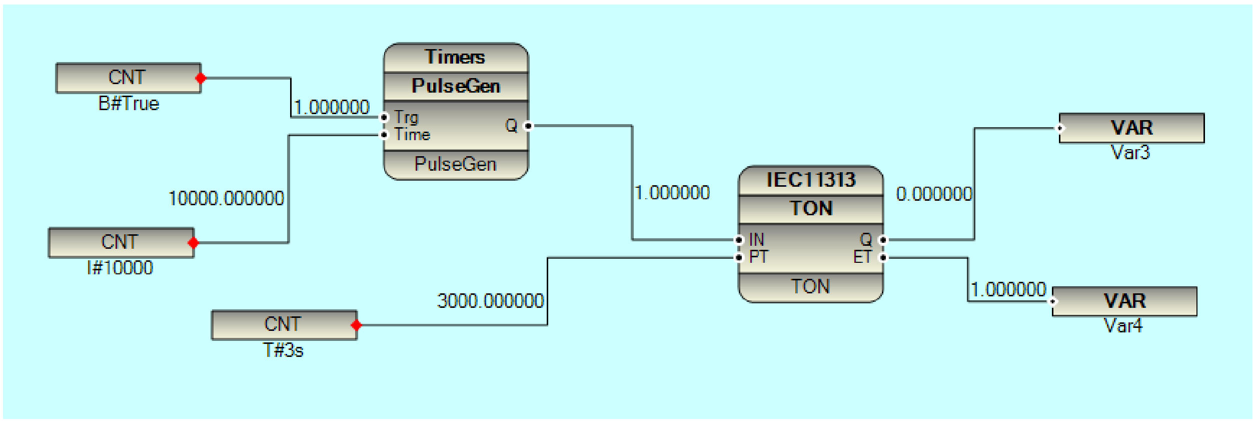
- **Operation**: When the trigger input (trig) is set to True, the output (q) is set to True, and the elapsed time counter (et) is reset to 0. The counter increments until the preset timer value is reached, after which the output (q) is set to False.



6.10 TON Description

The TON (Timer On Delay) function block is an ON delay timer that operates as follows:

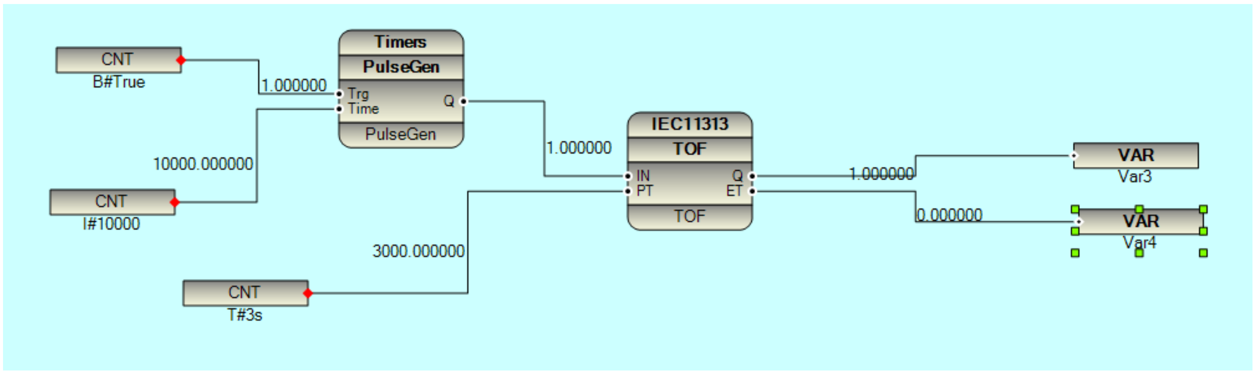
- **Operation**: When the trigger input (trig) is set to True, the output (q) is set to True after a specified delay (in milliseconds). The output remains True as long as trig is True.



6.11 TOF Description

The TOF (Timer Off Delay) function block is an OFF-delay timer that operates as follows:

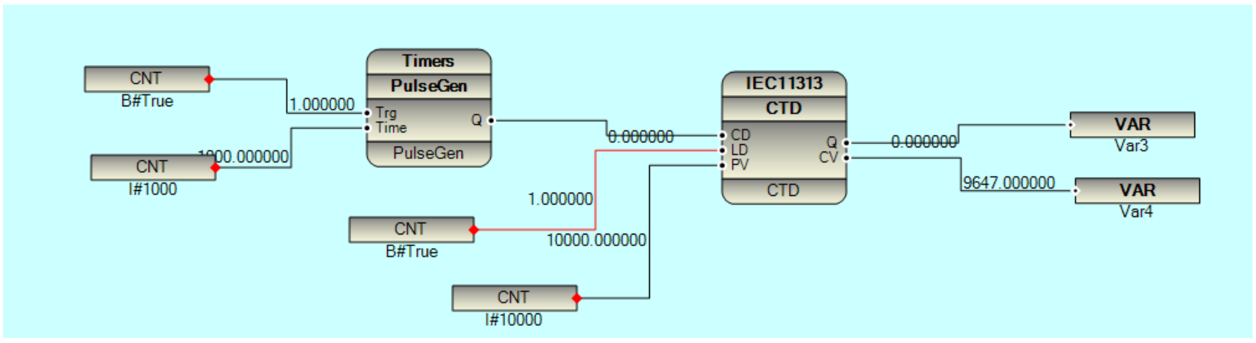
- **Operation**: When the trigger input (trig) is set to True, the elapsed time counter (et) is reset to 0, and the output (q) is set to True. When trig is set to False, the output (q) remains True for a specified delay (in milliseconds) before being set to False.



6.12 CTD Description

The CTD (Count Down) function block is a down counter that operates as follows:

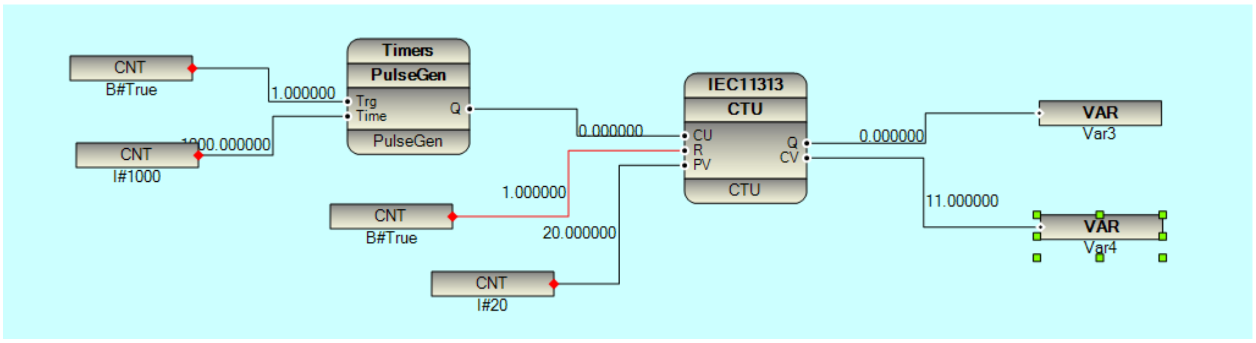
- **Operation**: The counter decreases by 1 when the countdown input (CD) transitions from False to True. The counter starts from a preset value (PV) and indicates when the count has decreased to a specified threshold.



6.13 CTU Description

The CTU (Count Up) function block is an up counter that operates as follows:

- **Operation**: The counter increments by 1 when the count up input (CU) transitions from False to True. It is typically used to indicate when the count reaches a specified maximum value (PV).

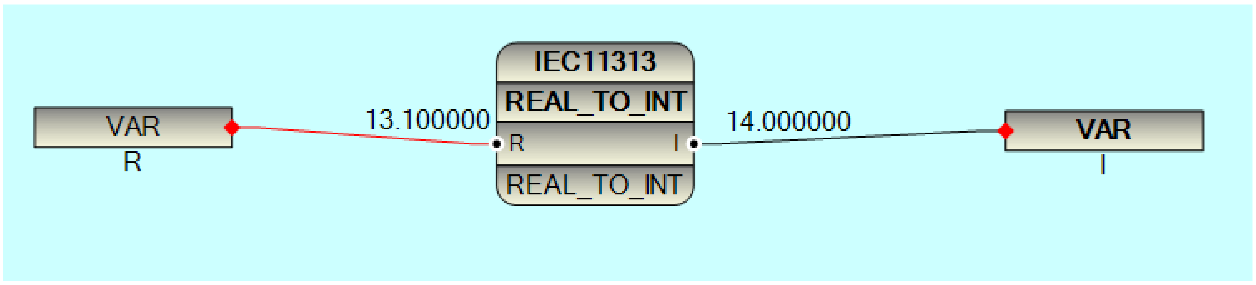


6.14 CTUD Description

The CTUD (Count Up and Down) function block is an up and down counter that operates as follows:

- **Operation**:
  - The counter increments by 1 when the count up input (CU) transitions from False to True.
  - The counter decrements by 1 when the countdown input (CD) transitions from False to True.
  - It is typically used to indicate when the count reaches a specified maximum value or decreases to 0.

REAL_TO_INT: This Function round input signal upwards to nearest integer value.



6.15 REAL_TO_INT Description

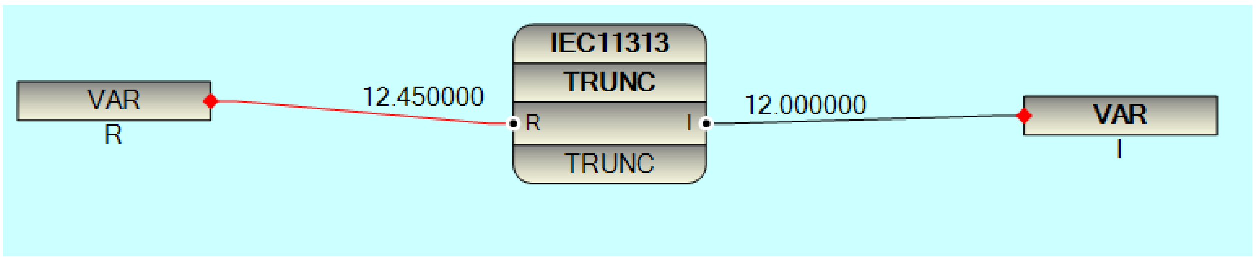
The REAL_TO_INT function block operates as follows:

- **Operation**: Rounds the input signal upward to the nearest integer value and maps it to the output.

6.16 TRUNC Description

The TRUNC function block operates as follows:

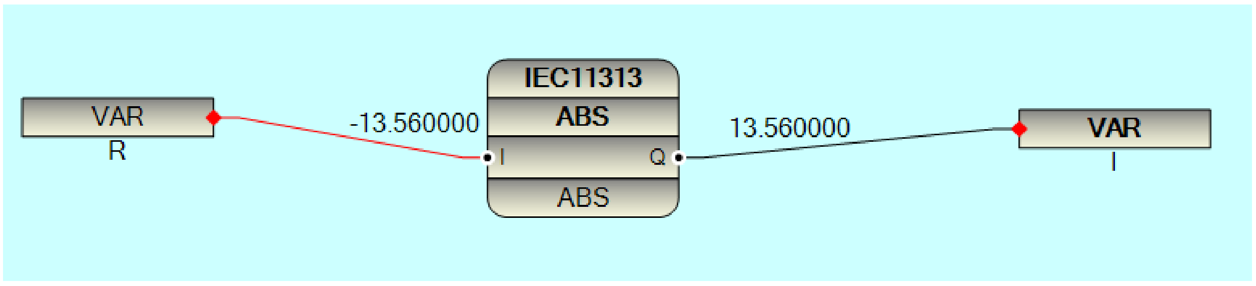
- **Operation**: Rounds the input signal downward to the nearest integer value and maps it to the output.



6.17 ABS Description

The ABS function block operates as follows:

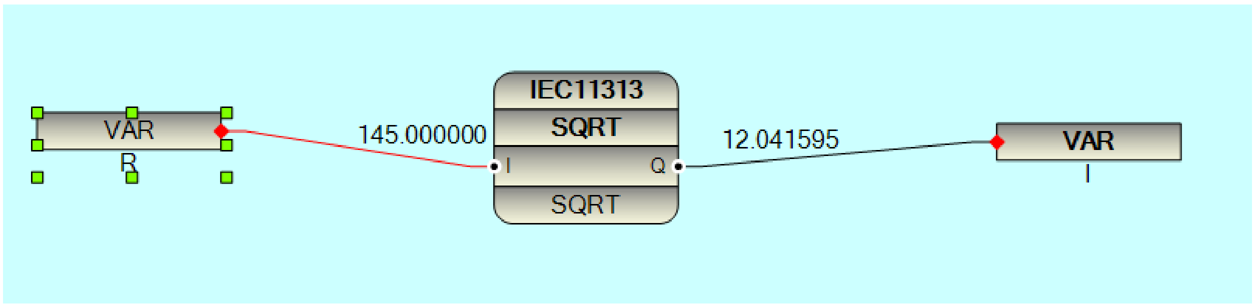
- **Operation**: Maps the absolute value of the input signal to the output.



6.18 SQRT Description

The SQRT function block operates as follows:

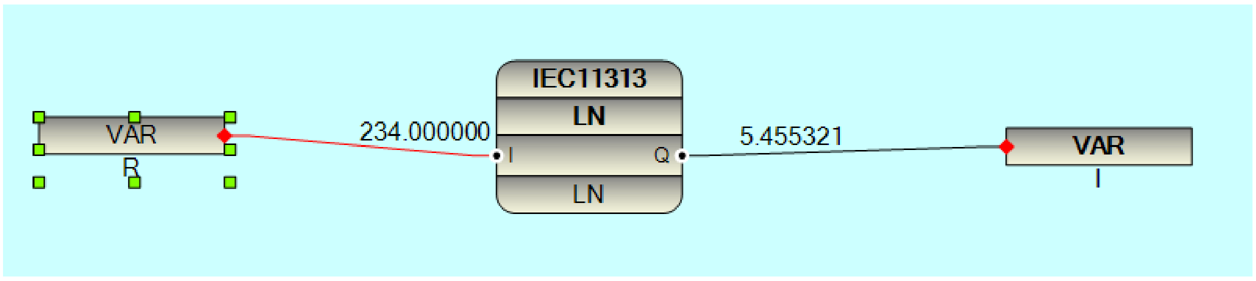
- **Operation**: Maps the square root of the input value to the output.



6.19 LN Description

The LN function block operates as follows:

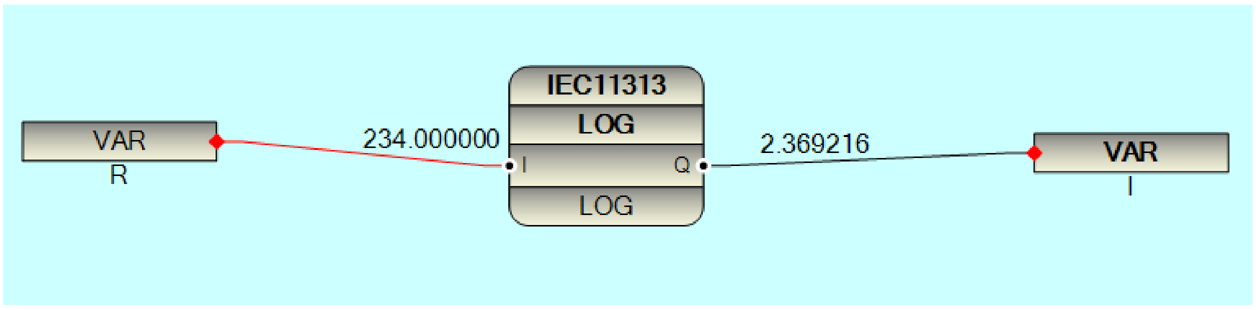
- **Operation**: Maps the natural logarithm (base e) of the input value to the output.



6.20 LOG Description

The LOG function block operates as follows:

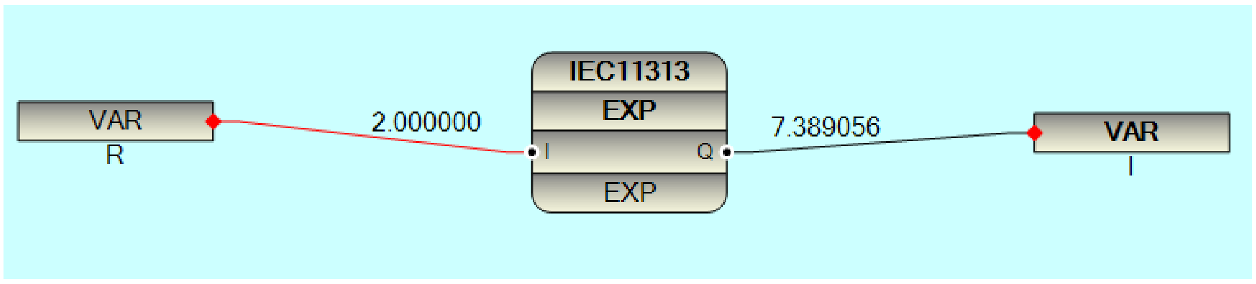
- **Operation**: Maps the logarithm (base 10) of the input value to the output.



6.21 EXP Description

The EXP function block operates as follows:

- **Operation**: Computes e (the base of natural logarithms) raised to the power of the input value and maps it to the output.

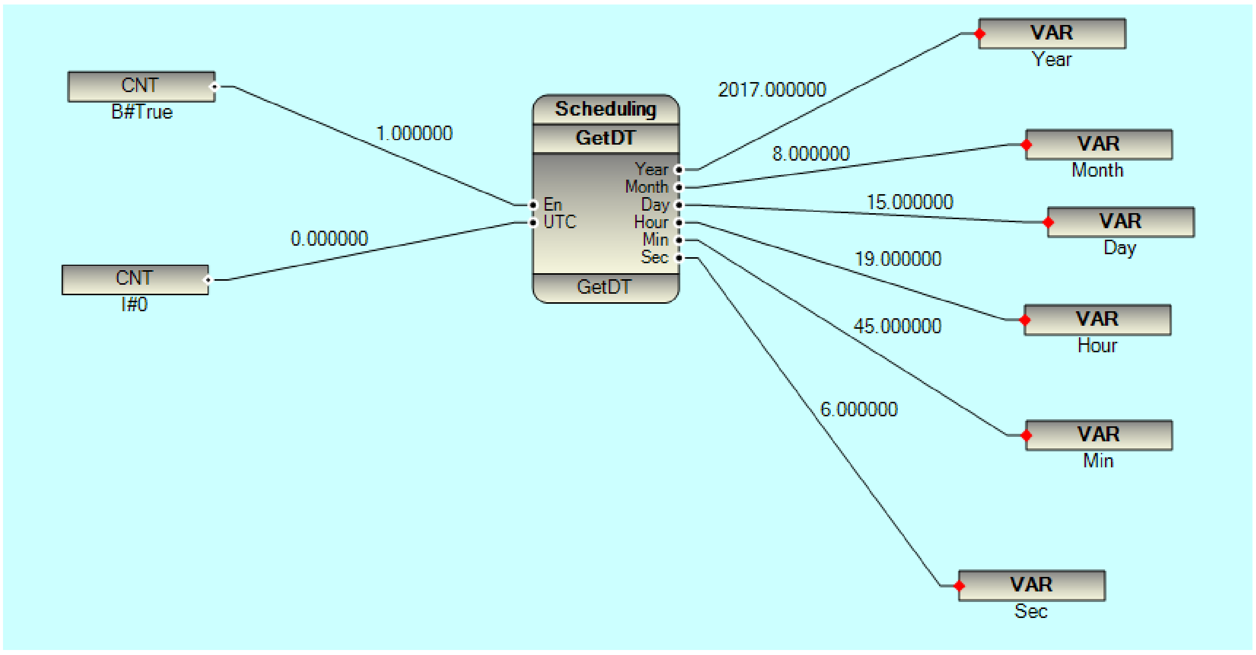


7 Scheduling

7.1.1 GetDT Description

The GetDT function block displays the current date and time of the Remote Terminal Unit (RTU). It operates as follows:

- **Enable Control**:
  - When the enable input (En) is False, all outputs are set to 0.
  - When En is True, the function block is active.
- **Time Selection**:
  - If the UTC input (UTC) is 0, the output (DT) provides the local time.
  - If UTC is 1, the output (DT) provides the UTC time.



7.1.2 DailySch Description

The DailySch function block is used for daily scheduling of an output signal, such as controlling an irrigation pump based on a predefined schedule. For example, to start a pump daily at the following times:

- 08:00 for 10 minutes
- 12:30 for 20 minutes
- 16:00 for 30 minutes

- 18:00 for 10 minutes

The function block has the following inputs and outputs:

7.1.2.1 Inputs

- **En**: Enables or disables the function block.
- **SchNum**: Specifies the number of schedules per day (maximum of 6 schedules).
- **Hourx**: The start hour for schedule number x (where x ranges from 1 to 6).
- **Minx**: The start minute for schedule number x (where x ranges from 1 to 6).
- **Durx**: The duration of schedule number x in seconds (where x ranges from 1 to 6).

7.1.2.2 Outputs

- **Q**: The main output signal, which can be connected to a pump management function block.
- **State**: An internal state signal used for monitoring or debugging purposes.

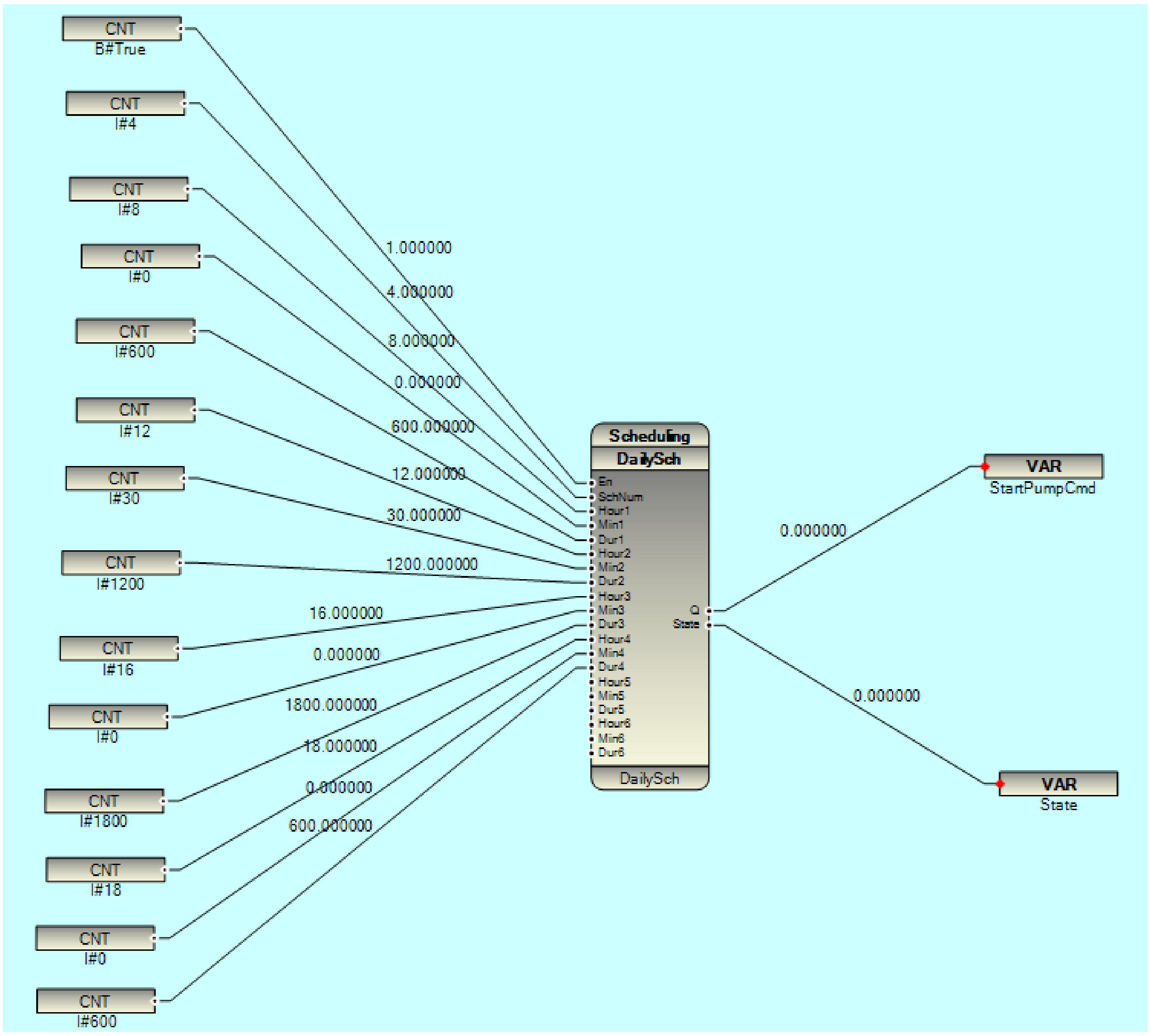
7.1.2.3 Operation

The DailySch function block checks the current RTU time and compares it to the defined schedule times. When the current time matches a scheduled start time (Hourx:Minx), the output (Q) is set to High for the specified duration (Durx) in seconds.

7.1.2.4 Example Configuration

For the irrigation pump example, the DailySch function block can be configured with the following parameters:

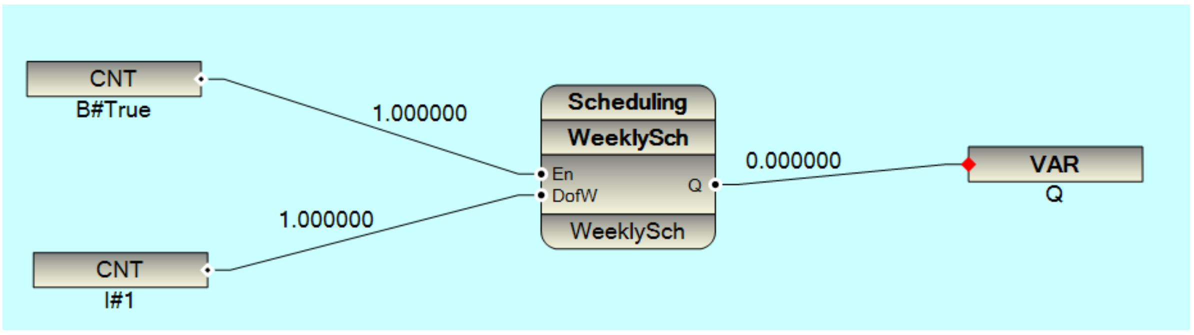
- SchNum: 4 (for four daily schedules)
- Hour1: 8, Min1: 0, Dur1: 600 (10 minutes)
- Hour2: 12, Min2: 30, Dur2: 1200 (20 minutes)
- Hour3: 16, Min3: 0, Dur3: 1800 (30 minutes)
- Hour4: 18, Min4: 0, Dur4: 600 (10 minutes)



7.1.3 WeeklySch Description

The WeeklySch function block checks the current day of the week and operates as follows:

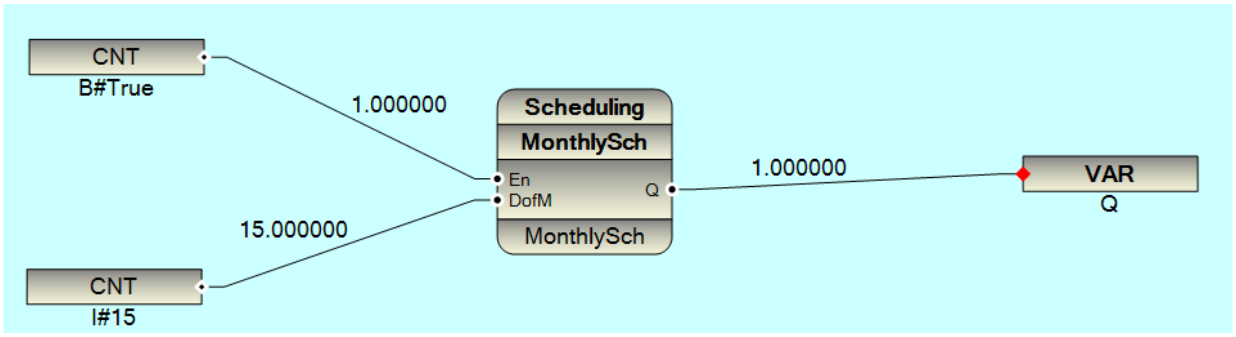
- **Operation**: The output (Q) is set to High if the current day of the week matches the specified DofW (Day of Week) input.
- **DofW Parameter**: An integer value from 0 (Sunday) to 6 (Saturday) representing the target day of the week.
- **Usage**: Can be combined with the DailySch function block to create different schedules for each day of the week.



7.1.4 MonthlySch Description

The MonthlySch function block checks the current day of the month and operates as follows:

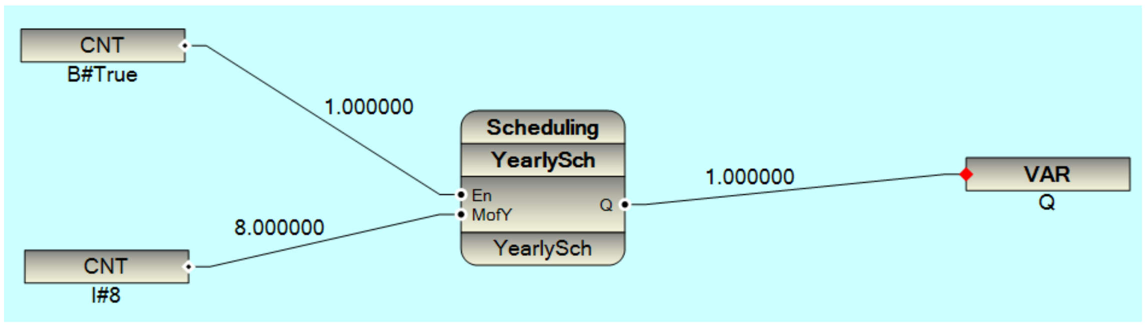
- **Operation**: The output (Q) is set to High if the current day of the month matches the specified DofM (Day of Month) input.
- **DofM Parameter**: An integer value from 1 to 31 representing the target day of the month.
- **Usage**: Can be combined with the DailySch function block to create schedules tailored to specific days of the month.



7.1.5 YearlySch Description

The YearlySch function block checks the current month and operates as follows:

- **Operation**: The output (Q) is set to High if the current month matches the specified MofY (Month of Year) input.
- **MofY Parameter**: An integer value representing the target month of the year (e.g., 1 for January, 12 for December).
- **Usage**: Can be combined with MonthlySch and DailySch function blocks to create schedules tailored to different seasons or months.


CNT
B#True
1.000000
Scheduling
YearlySch
En
MofY
Q
YearlySch
1.000000
VAR
Q
8.000000
CNT
I#8