# PowerMetering

**IEC 61131 Library for ACSELERATOR RTAC® Projects**

SEL Automation Controllers

# Table of Contents

RTAC LIBRARY

# PowerMetering

## Introduction

This library provides objects for performing common power metering functions. These functions provide event times for minimum and maximum thresholds, accumulated energy over time, demand over a configurable length of time, and KYZ/KY accumulators. Example applications include use of these function blocks with Axion analog input, digital input, and/or CT/PT (current transformer/potential transformer) modules.

## Supported Firmware Versions

You can use this library on any device configured using ACSELERATOR RTAC® SEL-5033 Software with firmware version R143 or higher.

Versions 3.5.1.0 and older can be used on RTAC firmware version R132 and higher.

## Function Blocks

### fb_Maximum

Compare input value to stored maximum value, update output if greater, and record the date/time of occurrence.

#### Initialization Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| settingsChange | BOOL | Flag to prevent setting outputs based on provided initial values; a value of TRUE results in setting *Maximum* to zero and the time stamp to the present time. |
| initialValue | REAL | Maximum value to use for initialization if *settingsChange* is FALSE. |
| initialTime | timestamp_t | Time stamp value to use for initialization if *settingsChange* is FALSE. |

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| EN | BOOL | Flag to enable or disable maximum comparison. |
| AnalogQuantity | REAL | Value to check against *Maximum*. |
| Reset | BOOL | Flag to reset *Maximum* and time stamp. |

### Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| Maximum | REAL | Maximum value. |
| EventTime | timestamp_t | The moment at which *Maximum* was last updated. |

### Processing

- ➤ If *settingsChange* is FALSE, the function block initializes to the values passed in.
- ➤ If *settingsChange* is TRUE or after reset is deasserted, *Maximum* is set to zero and will take on the next *AnalogQuantity* received.
- ➤ Compare the input *AnalogQuantity* to the stored *Maximum* value.
- ➤ If the input is greater than the stored value for two or more samples, update *Maximum* and record the date and time of occurrence.

## fb_Minimum

Compare input real value to stored minimum value, update output if greater, and record the date/time of occurrence.

### Initialization Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| settingsChange | BOOL | Flag to prevent setting outputs based on provided initial values; a value of TRUE results in setting *Minimum* to zero and the time stamp to the present time. |
| initialValue | REAL | Minimum value to use for initialization, if *settingsChange* is FALSE. |
| initialTime | timestamp_t | Time stamp value to use for initialization, if *settingsChange* is FALSE. |
| minimumThreshold | REAL | Lowest value this function block will record. |

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| EN | BOOL | Flag to enable or disable minimum comparison. |
| AnalogQuantity | REAL | Value to check against *Minimum*. |
| Reset | BOOL | Flag to reset *Minimum* and time stamp. |

### Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| Minimum | REAL | Minimum value. |
| EventTime | timestamp_t | The moment at which *Minimum* was last updated. |

### Processing

- ➤ If *settingsChange* is FALSE, the function block initializes to the values passed in.
- ➤ If *settingsChange* is TRUE or after reset is deasserted, *Minimum* is set to zero and will take on the next *AnalogQuantity* received.
- ➤ Compare the input *AnalogQuantity* to the stored *Minimum* value.
- ➤ If the input is less than the stored value and greater than *minimumThreshold* for two or more samples, update *Minimum* and record the date and time of occurrence.
- ➤ This function block will work with any values but is designed for use alongside fb_Maximum with positive numbers.

## fb_Energy

Collect energy input over time, accumulating positive and negative values in separate registers.

### Initialization Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| settingsChange | BOOL | Flag to prevent setting outputs based on provided initial values; a value of TRUE results in setting *EnergyIn* and *EnergyOut* to zero. |
| initialValueIn | REAL | *EnergyIn* value to use for initialization, if *settingsChange* is FALSE. |
| initialValueOut | REAL | *EnergyOut* value to use for initialization, if *settingsChange* is FALSE. |
| rolloverThreshold | REAL | Rollover value for this function block. |

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| EN | BOOL | Flag to enable or disable energy accumulation. |
| AnalogQuantity | REAL | Value to use for accumulating energy. |
| Reset | BOOL | Flag to reset *EnergyIn* and *EnergyOut*. |

### Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| EnergyIn | REAL | Accumulated energy in. |
| EnergyOut | REAL | Accumulated energy out. |

### Processing

- ➤ If *settingsChange* is FALSE, the function block initializes to the values passed in.
- ➤ If *settingsChange* is TRUE or after reset is deasserted, *EnergyIn* and *EnergyOut* are set to zero.
- ➤ Maintain the accumulated IN/OUT energy. A negative power value is considered IN energy while a positive power value is considered OUT energy. Receiving a positive power value (OUT) will not affect the accumulated IN value and vice versa.
- ➤ Update no more frequently than once a second regardless of the RTE cycle time.
- ➤ Restart either output to zero when it exceeds *rolloverThreshold*.

## fb_Demand

Calculates demand.

### Initialization Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| settingsChange | BOOL | Flag to prevent setting outputs based on provided initial values; a value of TRUE results in setting *Demand* to zero. |
| initialValue | REAL | Demand value to use for initialization, if *settingsChange* is FALSE. |
| demandType | Demand_Enum | The calculation method this function block will use; either ROLLING or THERMAL. |
| timeConstant | Time_Constant_Enum | The time constant this function block will use during demand calculations, MIN5, MIN10, MIN15, MIN20, MIN30, MIN60. |

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| EN | BOOL | Flag to enable or disable demand calculation. |
| AnalogQuantity | REAL | Value to use for calculating demand. |
| Reset | BOOL | Flag to reset *Demand*. |

### Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| Demand | REAL | Demand value. |

### Processing

- ➤ If *settingsChange* is FALSE, the function block initializes to the values passed in.
- ➤ If *settingsChange* is TRUE or after reset is deasserted, *Demand* is set to zero.
- ➤ Update no more frequently than once a second regardless of the RTE cycle time.
- ➤ Thermal demand output updates with each call to the function block.
- ➤ Thermal demand is a logarithmic average of the power used, with more-recent load weighted more heavily than less-recent load. For a steady state transition, this block outputs *Demand* of 90% of the change after *timeConstant* has passed.
- ➤ Rolling demand output only updates once every 5 minutes.
- ➤ Rolling demand averages input over periods of 5 minutes and outputs *Demand* as an average of enough 5 minute averages to equal *timeConstant*.

## fb_KYZ

Accumulate a count of transitions from only a Y of true to only a Z of true or back again.

### Initialization Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| settingsChange | BOOL | Flag to prevent setting outputs based on provided initial values; a value of TRUE results in setting *CV* and *ROV* to zero. |
| initialCV | BCR | Initial accumulator state, if *settingsChange* is FALSE. |
| initialROV | UDINT | Initial roll over value, if *settingsChange* is FALSE. |
| maxValue | UDINT | The value of the accumulator at which roll over occurs |

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| EN | BOOL | Flag to enable or disable the KYZ accumulator. |
| Y | SPS | Terminal Y. |
| Z | SPS | Terminal Z. |
| Reset | BOOL | Flag to reset the state of the KYZ block. |

### Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| CV | BCR | The number of transitions from Y to Z or Z to Y. |
| ROV | UDINT | The number of times *CV* has reset to zero. |

### Processing

- ➤ If *settingsChange* is FALSE, the function block initializes to the values passed in.
- ➤ If *settingsChange* is TRUE or after reset is deasserted, *CV* is set to the default state and *ROV* is set to zero.
- ➤ Monitor *Y* and *Z* when *EN* is TRUE and *Reset* is FALSE.
- ➤ Define a countable state as *Y* and *Z* being in opposite states with qualities of good.
- ➤ Define a countable transition as the present inputs being in a countable state and the present state of the inputs is opposite to the previous counted state of the inputs.
- ➤ Count only at times when both *Y* and *Z* report good quality (i.e., q.validity = good).
- ➤ Set the *CV* quality attribute based on the input with the least quality. (I.e., If input *Y.q.validity* is invalid and *Z.q.validity* is good, then *CV.q.validity* is invalid.)
- ➤ Override the quality of *CV* to invalid if the block is disabled or being reset.
- ➤ Increment *ROV* and the accumulator to zero when the accumulator equals *maxValue*. The practical implication is that *maxValue* declared at initialization is never reported, but instead the accumulator rolls over to zero allowing the total count to be calculated as *CV + ROV • maxValue*.

## fb_KY

Accumulate a count of transitions of a single variable Y.

### Initialization Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| settingsChange | BOOL | Flag to prevent setting outputs based on provided initial values; a value of TRUE results in setting *CV* and *ROV* to zero. |
| initialCV | BCR | Initial accumulator state, if *settingsChange* is FALSE. |
| initialROV | UDINT | Initial roll over value, if *settingsChange* is FALSE. |
| maxValue | UDINT | The value of the accumulator at which roll over occurs |

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| EN | BOOL | Flag to enable or disable the KY accumulator. |
| Y | SPS | Terminal Y. |
| Reset | BOOL | Flag to reset the state of the KY block. |

### Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| CV | BCR | The number of transitions of *Y*. |
| ROV | UDINT | The number of times *CV* has reset to zero. |

### Processing

- If *settingsChange* is FALSE, the function block initializes to the values passed in.
- If *settingsChange* is TRUE or after reset is deasserted, *CV* is set to the default state and *ROV* is set to zero.
- Monitor *Y* when *EN* is TRUE and *Reset* is FALSE.
- Define a countable transition as the present input being opposite its previous value.
- Count only at times when *Y* reports good quality (i.e., q.validity = good).
- Set the *CV* quality attribute as the quality of the input.
- Override the quality of *CV* to invalid if the block is disabled or being reset.
- Increment *ROV* and the accumulator to zero when the accumulator equals *maxValue*. The practical implication is that *maxValue* declared at initialization is never reported, but instead the accumulator rolls over to zero allowing the total count to be calculated as *CV* + *ROV* • *maxValue*.

# Benchmarks

## Benchmark Platforms

The benchmarking tests recorded for this library are performed on the following platforms.

- SEL-3505
  - R135-V1 firmware
- SEL-3530
  - R135-V2 firmware
- SEL-3555
  - Dual-core Intel i7-3555LE processor
  - 4 GB ECC RAM
  - R135-V0 firmware

## Benchmark Test Descriptions

### fb_Maximum

The posted time is the average execution time of 100 calls in which a new maximum value was observed. This constitutes the longest running time for this call.

### fb_Minimum

The posted time is the average execution time of 100 calls in which a new minimum value was observed. This constitutes the longest running time for this call.

### fb_Energy

The posted time is the average execution time of 100 calls at the top of the second. This constitutes the longest running time for this call.

### fb_Demand (Thermal)

The posted time is the average execution time of 100 calls at the top of the second. This constitutes the longest running time for this call.

### fb_Demand (Rolling)

The posted time is the average execution time of 100 calls at a 5-minute boundary. This constitutes the longest running time for this call.

### fb_KYZ

The posted time is the average execution time of 100 calls where the block incremented. This constitutes the longest running time for this call.

### fb_KY

The posted time is the average execution time of 100 calls where the block incremented. This constitutes the longest running time for this call.

## Benchmark Results

| Operation Tested | Platform (time in $\mu s$) | | |
|---|---|---|---|
| | SEL-3505 | SEL-3530 | SEL-3555 |
| fb_Maximum | 5 | 3 | 2 |
| fb_Minimum | 4 | 3 | 1 |
| fb_Energy | 4 | 3 | 1 |

| Operation Tested | Platform (time in $\mu s$) | | |
|---|---|---|---|
| | SEL-3505 | SEL-3530 | SEL-3555 |
| fb_Demand(Thermal) | 5 | 3 | 1 |
| fb_Demand(Rolling) | 5 | 3 | 1 |
| fb_KYZ | 1 | 1 | 1 |
| fb_KY | 1 | 1 | 1 |

# Examples

*These examples demonstrate the capabilities of this library. Do not mistake them as suggestions or recommendations from SEL.*

*Implement the best practices of your organization when using these libraries. As the user of this library, you are responsible for ensuring correct implementation and verifying that the project using these libraries performs as expected.*

## Maximum

### Objective

A user has an analog variable and wants to determine the greatest observed value of the analog.

### Assumptions

The following example provides code for two different situations. The first program assumes there is no requirement for non-volatile variables while the second program assumes there is a requirement that variables are retained through power loss.

### Solution

The user can create a program as shown in *Code Snippet 1*. Note that this example does not use retain variables.

**Code Snippet 1 prg_Maximum_Example**

```
PROGRAM prg_Maximum_Example
VAR
    En        : BOOL;
    Value     : REAL;
    InitValue : REAL;
    Reset     : BOOL;
    Maximum   : REAL;
    Max_Event : timestamp_t;

    Max1      : fb_Maximum(  settingsChange := TRUE,
                             initialValue := InitValue,
                             initialTime  := Max_Event);
END_VAR
```

```
//Call the Maximum function with the desired value
Max1(EN:=En, AnalogQuantity:=Value, Reset:=Reset);

// Assign the outputs
Maximum    := Max1.Maximum;
Max_Event  := Max1.EventTime;
```

## Solution With Retain Variables

Retain variables allow variable values to remain consistent through removal and restoration of power and program downloads. The user can create a program as shown in *Code Snippet 2*. RETAIN_UID must be initialized on the first run. See *Retain Variables on page 22* for more details on retain variables.

**Code Snippet 2 prg_Maximum_Retain_Example**

```
VAR_GLOBAL RETAIN
    // If Event Time is not desired to be retained a REAL value can be used
    Max1Retain      : MV;
    RETAIN_VERSION  : DWORD;
END_VAR
VAR_GLOBAL
    // Modify this when the retain value should not be used.
    VERSION         : DWORD:=1;
END_VAR
```

```
PROGRAM prg_Maximum_Example_Retain
VAR
    En      : BOOL;
    Value   : REAL;
    Reset   : BOOL;

    Max1    : fb_Maximum(   settingsChange := RETAIN_VERSION <> VERSION,
                            initialValue := Max1Retain.instMag,
                            initialTime := Max1Retain.t);
(*If VERSION has been modified, Maximum value will be reset to zero,
 *otherwise the retain values will be used. settingsChange should be
     evaluated
 *from constants or retain variables only. *)
END_VAR
```

**Code Snippet 2 prg_Maximum_Retain_Example (Continued)**

```
//Update the retain variable first thing
RETAIN_VERSION := VERSION;

//Call the Maximum function with the desired value
Max1(EN:=En, AnalogQuantity:=Value, Reset:=Reset);

// Assign the outputs to the retain variables
Max1Retain.instMag := Max1.Maximum;
Max1Retain.t       := Max1.EventTime;
```

## Minimum

### Objective

A user has an analog variable and wants to determine the smallest observed value of the analog.

### Assumptions

The following example provides code for two different situations. The first program assumes there is no requirement for nonvolatile variables, and the second program assumes there is a requirement that variables are retained through power loss.

### Solution

The user can create a program as shown in *Code Snippet 3*. Note that this example does not use retain variables.

**Code Snippet 3 prg_Minimum_Example**

```
PROGRAM prg_Minimum_Example
VAR
    En               : BOOL;
    InitValue        : REAL;
    Value            : REAL;
    Reset            : BOOL;
    Minimum          : REAL;
    MinThreshold     : REAL := 50000;
    Min_Event        : timestamp_t;

    Min1             : fb_Minimum(  settingsChange := TRUE,
                                    initialValue := InitValue,
                                    initialTime := Min_Event,
                                    minimumThreshold := MinThreshold);
END_VAR
```

**Code Snippet 3 prg_Minimum_Example (Continued)**

```
//Call the Minimum function with the desired value
Min1(EN := En, AnalogQuantity := Value, Reset := Reset);

// Assign the outputs
Minimum      := Min1.Minimum;
Min_Event    := Min1.EventTime;
```

## Solution With Retain Variables

Retain variables allow variable values to remain consistent through removal and restoration of power and program downloads. The user can create a program as shown in *Code Snippet 4*. RETAIN_UID must be initialized on the first run. See *Retain Variables on page 22* for more details on retain variables.

**Code Snippet 4 prg_Minimum_Retain_Example**

```
VAR_GLOBAL RETAIN
    // If Event Time is not desired to be retained a REAL value can be used
    Min1Retain     : MV;
    RETAIN_VERSION : DWORD;
END_VAR
VAR_GLOBAL
    // Modify this when the retain value should not be used.
    VERSION        : DWORD := 1;
END_VAR
```

```
PROGRAM prg_Minimum_Example_Retain
VAR
    En           : BOOL;
    Value        : REAL;
    Reset        : BOOL;
    MinThreshold : REAL := 50000;

    Min1         : fb_Minimum(  settingsChange := RETAIN_VERSION <>
         VERSION,
                                initialValue := Min1Retain.instMag,
                                initialTime := Min1Retain.t,
                                minimumThreshold := MinThreshold);
(*If VERSION has been modified, Minimum value will be reset to zero,
 *otherwise the retain values will be used. settingsChange should be
      evaluated
 *from constants or retain variables only. *)
END_VAR
```

```
//Update the retain variable first thing
RETAIN_VERSION := VERSION;

//Call the Minimum function with the desired value
Min1(EN:=En, AnalogQuantity:=Value, Reset:=Reset);

// Assign the outputs to the retain variables
Min1Retain.instMag := Min1.Minimum;
Min1Retain.t       := Min1.EventTime;
```

## Energy

### Objective

A user has a power quantity and wants to keep track of energy flow. Positive power value is considered to be "out" and negative power is considered to be "in."

### Assumptions

The following example provides code for two different situations. The first program assumes there is no requirement for nonvolatile variables while the second program assumes there is a requirement that variables are retained through power loss.

### Solution

The user can create a program as shown in *Code Snippet 5*. Note that this example does not use retain variables.

**Code Snippet 5 prg_Energy_Example**

```
PROGRAM prg_Energy_Example
VAR
    En            : BOOL;
    Value         : REAL;
    Reset         : BOOL;
    InitIn        : REAL;
    InitOut       : REAL;
    Threshold     : REAL;
    EnergyIn      : REAL;
    EnergyOut     : REAL;


    Energy1       : fb_Energy(  settingsChange := TRUE,
                                initialValueIn := InitIn,
                                initialValueOut := InitOut,
                                rollOverThreshold := Threshold);
END_VAR
```

```
//Call the Energy function with the desired value
Energy1(EN := En, AnalogQuantity := Value, Reset := Reset);

// Assign the outputs
EnergyIn        := Energy1.EnergyIn;
EnergyOut       := Energy1.EnergyOut;
```

### Solution With Retain Variables

Retain variables allow variable values to remain consistent through removal and restoration of power and program downloads. The user can create a program as shown in *Code Snippet 6*. RETAIN_UID must be initialized on the first run. See *Retain Variables on page 22* for more details on retain variables.

**Code Snippet 6 prg_Energy_Retain_Example**

```
VAR_GLOBAL RETAIN
    EnergyIn        : REAL;
    EnergyOut       : REAL;
    RETAIN_VERSION  : DWORD;
END_VAR
VAR_GLOBAL
    // Modify this when the retain value should not be used.
    VERSION         : DWORD := 1;
END_VAR
```

```
PROGRAM prg_Energy_Example_Retain
VAR
    En           : BOOL;
    Value        : REAL;
    Reset        : BOOL;
    Threshold    : REAL;

    Energy1      : fb_Energy(  settingsChange := (RETAIN_VERSION <> VERSION),
                               initialValueIn := EnergyIn,
                               initialValueOut := EnergyOut,
                               rollOverThreshold := Threshold);
(*If VERSION has been modified, the energy values will be reset to zero,
 *otherwise the retain values will be used. settingsChange should be
     evaluated
 *from constants or retain variables only. *)
END_VAR
```

```
//Update the retain variable first thing
RETAIN_VERSION := VERSION;

//Call the Energy function with the desired value
Energy1(EN := En, AnalogQuantity := Value, Reset := Reset);

// Assign the outputs to the retain variables
EnergyIn     := Energy1.EnergyIn;
EnergyOut    := Energy1.EnergyOut;
```

## Demand

### Objective

A user wants to calculate demand on an analog quantity.

### Assumptions

The following example provides code for two different situations. The first program assumes there is no requirement for nonvolatile variables while the second program assumes there is a requirement that variables are retained through power loss.

## Solution

The user can create a program as shown in *Code Snippet 7*. Note that this example does not use retain variables.

**Code Snippet 7 prg_Demand_Example**

```
PROGRAM prg_Demand_Example
VAR
    En              : BOOL;
    Value           : REAL;
    Reset           : BOOL;
    InitValue       : REAL;
    DemandTherm     : REAL;
    DemandRolling   : REAL;

    Demand1         : fb_Demand(  settingsChange := TRUE,
                                  initialValue := InitValue,
                                  DemandType := THERMAL,
                                  timeConstant := MIN10);

    Demand2         : fb_Demand(  settingsChange := TRUE,
                                  initialValue := InitValue,
                                  DemandType := ROLLING,
                                  timeConstant := MIN20);
END_VAR
```

```
//Call the Demand function with the desired values
Demand1(EN := En, AnalogQuantity := Value, Reset := Reset);
Demand2(EN := En, AnalogQuantity := Value, Reset := Reset);

// Assign the outputs
DemandTherm     := Demand1.Demand;
DemandRolling   := Demand2.Demand;
```

## Solution With Retain Variables

Retain variables allow variable values to remain consistent through removal and restoration of power and program downloads. The user can create a program as shown in *Code Snippet 8*. RETAIN_UID must be initialized on the first run. See *Retain Variables on page 22* for more details on retain variables.

**Code Snippet 8 prg_Demand_Retain_Example**

```
VAR_GLOBAL RETAIN
    DemandTherm      : REAL;
    DemandRolling    : REAL;
    RETAIN_VERSION   : DWORD;
END_VAR
VAR_GLOBAL
    // Modify this when the retain value should not be used.
    VERSION          : DWORD := 1;
END_VAR
```

**Code Snippet 8 prg_Demand_Retain_Example (Continued)**

```
PROGRAM prg_Demand_Example_Retain
VAR
    En       : BOOL;
    Value    : REAL;
    Reset    : BOOL;


    Demand1 : fb_Demand(  settingsChange := (RETAIN_VERSION <> VERSION),
                          initialValue := DemandTherm,
                          DemandType := THERMAL,
                          timeConstant := MIN10);

    Demand2 : fb_Demand(  settingsChange := (RETAIN_VERSION <> VERSION),
                          initialValue := DemandRolling,
                          DemandType := ROLLING,
                          timeConstant := MIN20);
(*If VERSION has been modified, the demand values will be reset to zero,
 *otherwise the retain values will be used. settingsChange should be
     evaluated
 *from constants or retain variables only. *)
END_VAR
```

```
//Update the retain variable first thing
RETAIN_VERSION := VERSION;

//Call the Demand function with the desired value
Demand1(EN := En, AnalogQuantity := Value, Reset := Reset);
Demand2(EN := En, AnalogQuantity := Value, Reset := Reset);

// Assign the outputs to the retain variables
DemandTherm    := Demand1.Demand;
DemandRolling  := Demand2.Demand;
```

## KYZ

### Objective

A user has two inputs connected to the Y and Z terminal of a meter and wants to count the number of transitions.

### Assumptions

The following example provides code for two different situations. The first program assumes there is no requirement for nonvolatile variables while the second program assumes there is a requirement that variables are retained through power loss.

### Solution

The user can create a program as shown in *Code Snippet 9*. Note that this example does not use retain variables.

**Code Snippet 9 prg_KYZ_Example**

```
PROGRAM prg_KYZ_Example
VAR
    //Terminal for Y and Z pulses
    DI_Y_Terminal   : SPS;
    DI_Z_Terminal   : SPS;
    //Enabling and Rest Conditions
    Enable          : BOOL;
    Reset           : BOOL;

    //Placeholder for DNP Tag
    DNP_Counter1    : BCR;
    DNP_Counter2    : BCR;
    //Additional Outputs
    RollOver1       : UDINT;
    RollOver2       : UDINT;

    //The KYZ blocks ignore their initial values because settingsChange is
        true.
    KYZ_12bit       : fb_KYZ(   settingsChange := TRUE, initialCV :=
        DNP_Counter1,
                                initialROV := 0, maxValue := 4095);

    KYZ_32bit       : fb_KYZ(   settingsChange := TRUE, initialCV :=
        DNP_Counter2,
                                initialROV := 0, maxValue := 4294967295);
END_VAR
```

```
//Call the function block every cycle and assign outputs
KYZ_12bit(  EN := Enable, Y := DI_Y_Terminal, Z := DI_Z_Terminal, Reset :=
    Reset,
            CV => DNP_Counter1, ROV => RollOver1);

KYZ_32bit(  EN := Enable, Y := DI_Y_Terminal, Z := DI_Z_Terminal, Reset :=
    Reset,
            CV => DNP_Counter2, ROV => RollOver2);
```

## Solution With Retain Variables

Retain variables allow variable values to remain consistent through removal and restoration of power and program downloads. The user can create a program as shown in *Code Snippet 10*. RETAIN_UID must be initialized on the first run. See *Retain Variables on page 22* for more details on retain variables.

**Code Snippet 10 prg_KYZ_Retain_Example**

```
VAR_GLOBAL RETAIN
    //Persistent storage for counter values
    Counter          : BCR;
    RollOver         : UDINT;
    RETAIN_VERSION   : DWORD;
END_VAR
VAR_GLOBAL
    // Modify this when the retain value should not be used.
    VERSION          : DWORD := 1;
END_VAR
```

**Code Snippet 10 prg_KYZ_Retain_Example (Continued)**

```
PROGRAM prg_KYZ_Retain_Example
VAR
    //Block inputs
    DI_Y_Terminal  : SPS;
    DI_Z_Terminal  : SPS;
    Enable         : BOOL;
    Reset          : BOOL;

    //Placeholder for DNP output tags
    DNP_Counter    : BCR;

    KYZ_12bit      : fb_KYZ(   settingsChange := (RETAIN_VERSION <> VERSION),
                               initialCV := Counter, initialROV := RollOver,
                               maxValue := 4095);
(*If VERSION has been modified, the counter values will be reset to zero,
 *otherwise the retain values will be used. settingsChange should be
     evaluated
 *from constants or retain variables only. *)
END_VAR
```

```
//Update the retain variable first thing
RETAIN_VERSION := VERSION;

//Call the function block every cycle and assign outputs
KYZ_12bit(  EN := Enable, Y := DI_Y_Terminal, Z := DI_Z_Terminal, Reset :=
    Reset,
            CV => Counter, ROV => RollOver);

//Copy output to outbound communication channels
DNP_Counter := Counter;
```

## KY

### Objective

A user has one input connected to the Y terminal of a meter and wants to count the number of transitions.

### Assumptions

The following example provides code for two different situations. The first program assumes there is no requirement for nonvolatile variables while the second program assumes there is a requirement that variables are retained through power loss.

### Solution

The user can create a program as shown in *Code Snippet 11*. Note that this example does not use retain variables.

**Code Snippet 11 prg_KY_Example**

```
PROGRAM prg_KY_Example
VAR
    //Terminal for Y pulses
    DI_Y_Terminal   : SPS;
    Enable          : BOOL;
    Reset           : BOOL;

    //Placeholder for DNP Tag
    DNP_Counter1    : BCR;
    DNP_Counter2    : BCR;
    //Additional Outputs
    RollOver1       : UDINT;
    RollOver2       : UDINT;

    //The KY blocks ignore their initial values because settingsChange is
        true.
    KY_12bit        : fb_KY(   settingsChange := TRUE, initialCV :=
        DNP_Counter1,
                               initialROV := 0, maxValue := 4095);

    KY_32bit        : fb_KY(   settingsChange := TRUE, initialCV :=
        DNP_Counter2,
                               initialROV := 0, maxValue := 4294967295);
END_VAR
```

```
//Call the function block every cycle and assign outputs
KY_12bit(   EN := Enable, Y := DI_Y_Terminal, Reset := Reset,
            CV => DNP_Counter1, ROV => RollOver1);

KY_32bit(   EN := Enable, Y := DI_Y_Terminal, Reset := Reset,
            CV => DNP_Counter2, ROV => RollOver2);
```

## Solution With Retain Variables

Retain variables allow variable values to remain consistent through removal and restoration of power and program downloads. The user can create a program as shown in *Code Snippet 12*. RETAIN_UID must be initialized on the first run. See *Retain Variables on page 22* for more details on retain variables.

**Code Snippet 12 prg_KY_Retain_Example**

```
VAR_GLOBAL RETAIN
    //Persistent storage for counter values
    Counter          : BCR;
    RollOver         : UDINT;
    RETAIN_VERSION   : DWORD;
END_VAR
VAR_GLOBAL
    // Modify this when the retain value should not be used.
    VERSION          : DWORD := 1;
END_VAR
```

**Code Snippet 12 prg_KY_Retain_Example (Continued)**

```
PROGRAM prg_KY_Retain_Example
VAR
    //Terminal for Y pulses
    DI_Y_Terminal  : SPS;
    //Enabling and Rest Conditions
    Enable         : BOOL;
    Reset          : BOOL;

    //Placeholder for DNP output tags
    DNP_Counter    : BCR;

    KY_12bit       : fb_KY(   settingsChange := (RETAIN_VERSION <> VERSION),
                              initialCV := Counter, initialROV := RollOver,
                              maxValue := 4095);
(*If VERSION has been modified, the counter values will be reset to zero,
 *otherwise the retain values will be used. settingsChange should be
     evaluated
 *from constants or retain variables only. *)
END_VAR
```

```
//Update the retain variable first thing
RETAIN_VERSION := VERSION;

//Call the function block every cycle and assign outputs
KY_12bit(   EN := Enable, Y := DI_Y_Terminal, Reset := Reset,
            CV => Counter, ROV => RollOver);

//Copy output to outbound communication channels
DNP_Counter := Counter;
```

# Retain Variables

## Note on Usage

Retain variables allow values to persist through removal and restoration of power, a reboot, and some program downloads. To accomplish this, retain variables point to a specific location in nonvolatile memory. This results in a situation where changing the definition of any retain variable (e.g., creating or deleting variables or changing variable order), can result in the variables pointing to a different location in memory, meaning an incorrect value would be used in logic. Therefore, you should initialize all retain variables when you change or add any retain variable declarations. Furthermore, it is best practice to keep all retain variables in the same global variable list to avoid the opportunity of the lists being reordered.

# Release Notes

| Version | Summary of Revisions | Date Code |
|---|---|---|
| 3.5.2.0 | ➤ Allows new versions of ACSELERATOR RTAC to compile projects for previous firmware versions without SEL IEC types "Cannot convert" messages.<br>➤ Must be used with R143 firmware or later. | 20180921 |
| 3.5.1.0 | ➤ Added fb_KYZ.<br>➤ Added fb_KY.<br>➤ Resolved an issue that could cause the fb_Demand block to discard the *initialValue* after the first five minutes, resulting in the *Demand* calculation being reset.<br>➤ Updated examples using retain values for clarity. | 20160415 |
| 3.5.0.5 | ➤ Initial release. | 20140714 |