AnalogCond

IEC 61131 Library for ACSELERATOR RTAC® Projects

SEL Automation Controllers

Table of Contents

AnalogCond

Introduction

This library contains classes that allow for simplified processing of analog quantities within applications. Generally, measured analog quantities require filtering and checks before being used. This library provides this filtering via encapsulated classes.

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

Global Parameters

The library applies the following values as maximums; they can be modified when the library is included in a project.

Name	IEC 61131 Type	Value	Description
g_p_MaxFilterOrder	UINT	4	The maximum order of the filter for class_ArmaFilter. This determines the maximum number of coefficients and the maximum delay, in samples, for the filter.

Interface Definitions

This section outlines the various interfaces defined within this library.

I_Filter

Classes implementing this interface provide a filter for analog values.

ConditionValue (Method)

This method takes *inputValue* as the next input for the filter and provides an output for the new conditioned value.

Inputs

Name	IEC 61131 Type	Description
inputValue	REAL	The new raw input to the filter.

Outputs

Name	IEC 61131 Type	Description
conditionedValue	REAL	The filtered output.

Return Value

IEC 61131 Type	Description
BOOL	True when filter windup is complete and *conditionedValue* output is fully filtered.

Reset (Method)

This method resets the filter and clears any internal state.

I_LimitedSplpf

This interface extends the I_Filter interface described in *I_Filter on page 2*, meaning that classes implementing this interface also implement the I_Filter methods. This interface is implemented by classes that condition analog values through a limited, single-pole, low-pass filter (SPLPF).

Classes implementing this interface provide the following features:

➤ Conditioning of the raw input through a low-pass filter controlled by a time constant defined in the object.

➤ Controlled output if the class is in Alarm. In the event that the conditioning class is in alarm, the class provides a result which approaches a predefined default value.

➤ Output bounded by the limits defined in the object.

➤ An out-of-bounds alarm which asserts if the input exceeds the high limit or falls below the low limit.

Properties

Name	IEC 61131 Type	Access	Description
Alarm	BOOL	R/W	Sets an alarm when true. Clears the alarm state if false.

Properties are internal values made visible through Get and Set accessors. Access is defined as R (read), W (write), or R/W (read/write).

OutOfBoundsAlarm (Method)

Provides the out-of-bounds state of the `ConditionValue()` input.

Return Value

IEC 61131 Type	Description
BOOL	Returns true if the input value is out of the boundaries specified in the object's constructor.

Public Class Definitions

This section contains the basic definitions, descriptions, and public methods for the public classes that can be instantiated by the user.

class_PassThroughFilter

This class implements a simple pass-through, where *conditionedValue* is set directly to *inputValue*. It is meant to be used in place of a filter during testing phases of development where it may be desirable to bypass a filtering stage.

Implemented Interfaces

An interface defines a required set of functionality as methods and properties. As an implementer of any interface all methods and properties declared in that interface must exist as members of this class. This allows multiple generally unrelated classes to be used interchangeably for a specific feature set.

➤ I_Filter

class_ArmaFilter

This class implements an AutoRegressive Moving Average (ARMA) filter, generally used to filter oscillating signals. This implementation provides either Infinite Impulse Response (IIR) or Finite Impulse Response (FIR) behavior, depending on the coefficients provided. The filter implements the form:

$$H(z) = \frac{B(z)}{A(z)}$$

Where:

$$B(z) = b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_N z^{-N}$$

$$A(z) = 1 - a_1 z^{-1} - a_2 z^{-2} - \ldots - a_M z^{-M}$$

Obtaining the coefficients for low-pass, high-pass, band-pass, or band-stop filters is made relatively simple using tools like Matlab or OCTAVE, but the mathematical methods for obtaining these values are outside the scope of this document; the user of this library should be aware that many filter designs used to obtain coefficients for this filter can produce numerical instability. See the example in Section 1 for a brief discussion on how to determine if the filter is numerically stable or not.

Once the coefficients b_0 to b_N and a_1 to a_M are determined, they are loaded as initialization inputs to the class. These coefficients must be normalized, as the leading 1 in *A(z)* is assumed in this class; in other words, a_0 is always assumed to be exactly 1.

Figure 1 shows how the filter works when three (3) coefficients for *A(z)* and five (5) coefficients for *B(z)* are provided. Note how the depth of the filter is normalized so that there are as many *A(z)* branches as there are *B(z)* branches. Because there is one less coefficient in the *A(z)* array (when including the assumed a_0 = 1) than in the *B(z)* coefficient array, the coefficient of the last branch a_4 is set to zero (0). In this particular example, there are five *B(z)* coefficients, and because each *z* value is shifted in time, four previous intermediate values must be stored in the filter. This means that the filter is not sufficiently primed until the 5th input value is provided. For this set of coefficients, the first four calls to `ConditionValue()` will yield a partially filtered output value, and the method will return false. The 5th call of the method, and all subsequent calls, will return true.

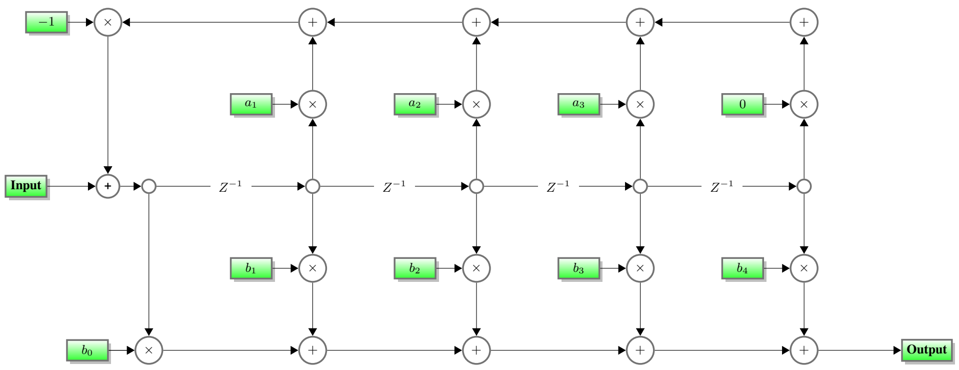

Figure 1 A Digital Filter Using Three (3) Coefficients for A(z) and Five (5) for B(z)

Implemented Interfaces

An interface defines a required set of functionality as methods and properties. As an implementer of any interface all methods and properties declared in that interface must exist as members of this class. This allows multiple generally unrelated classes to be used interchangeably for a specific feature set.

➤ I_Filter

Initialization Inputs

Name	IEC 61131 Type	Description
aCoefficients	ARRAY [1..g_p_MaxFilterOrder] OF REAL	Coefficients for *A(z)*. The coefficients must be normalized, as the leading 1 is assumed and should not be entered in this array.
bCoefficients	ARRAY [0..g_p_MaxFilterOrder] OF REAL	Coefficients for *B(z)*.
numACoefficients	UINT(1..g_p_MaxFilterOrder)	The number of coefficients within the *aCoefficients* array.
numBCoefficients	UINT(1..g_p_MaxFilterOrder + 1)	The number of coefficients within the *bCoefficients* array.

class_ArmaFilter_LREAL

This class implements an AutoRegressive Moving Average (ARMA) filter, generally used to filter oscillating signals. This implementation provides either Infinite Impulse Response (IIR) or Finite Impulse Response (FIR) behavior, depending on the coefficients provided. The filter implements the form:

$$H(z) = \frac{B(z)}{A(z)}$$

Where:

$$B(z) = b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_N z^{-N}$$

$$A(z) = 1 - a_1 z^{-1} - a_2 z^{-2} - \ldots - a_M z^{-M}$$

The implementation of this filter is the same as the class_ArmaFilter, *class_ArmaFilter on page 4*, but it retains greater precision internally. This allows for greater stability.

Implemented Interfaces

An interface defines a required set of functionality as methods and properties. As an implementer of any interface all methods and properties declared in that interface must exist as members of this class. This allows multiple generally unrelated classes to be used interchangeably for a specific feature set.

➤ I_Filter

Initialization Inputs

Name	IEC 61131 Type	Description
aCoefficients	ARRAY [1..g_p_MaxFilterOrder] OF LREAL	Coefficients for *A(z)*. The coefficients must be normalized, as the leading 1 is assumed and should not be entered in this array.
bCoefficients	ARRAY [0..g_p_MaxFilterOrder] OF LREAL	Coefficients for *B(z)*.
numACoefficients	UINT(1..g_p_MaxFilterOrder)	The number of coefficients within the *aCoefficients* array.
numBCoefficients	UINT(1..g_p_MaxFilterOrder + 1)	The number of coefficients within the *bCoefficients* array.

class_LimitedSplpfStepToDefault

Instantiate this class when a single-pole low-pass filter that has an imposed range of acceptable values is desired. When in alarm, this class will cause the output to step, in a single time step, to the *defaultOutput* set in the constructor method for the class.

Implemented Interfaces

An interface defines a required set of functionality as methods and properties. As an implementer of any interface all methods and properties declared in that interface must exist as members of this class. This allows multiple generally unrelated classes to be used interchangeably for a specific feature set.

➤ I_LimitedSplpf

LimitedSplpfStepToDefault (Method)

This method acts as the constructor and must be called before the class can operate. It initializes the characteristics of the filter.

Inputs

Name	IEC 61131 Type	Description
highLimit	REAL	The largest valid value for the input variable.
lowLimit	REAL	The smallest valid value for the input variable.
defaultOutput	REAL	The conditioned output defaults to this value if the input is out of range or the alarm is high.
timeConstant	UINT	Range: 100–60000 ms. The time constant to use for the low-pass filter within this method.

Return Value

IEC 61131 Type	Description
POINTER TO STRING	Return a pointer to an error message if an error occurred. Return zero if no errors exist.

Processing

This method:

➤ Sets *defaultOutput* as the initial output and input to the filter in order to eliminate "wind-up" during the first few scans.

➤ Returns a pointer to an error message if *lowLimit* exceeds *highLimit*.

➤ Returns a pointer to an error message if *defaultOutput* is less than *lowLimit* or greater than *highLimit*.

bootstrap_SetInitialValue (Method)

This method may be called at startup if the user desires a value different than the *default-Output* (set previously in the constructor method call) as the initial output.

Inputs

Name	IEC 61131 Type	Description
initialValue	REAL	Range: *lowLimit* $\leq$ *initialValue* $\leq$ *highLimit*. Sets the initial value to be used by the filter at startup.

Return Value

IEC 61131 Type	Description
POINTER TO STRING	Return a pointer to an error message if an error occurred. Return zero if no errors exist.

Processing

This method:

➤ Bypasses all internal filtering and changes both the input and conditioned output to be equal to *initialValue*.

➤ Returns a pointer to an error message if the constructor has not been called.

➤ Returns a pointer to an error message if *initialValue* is less than *lowLimit* or greater than *highLimit* set in the constructor.

Processing of Interface Methods

This section provides specifics regarding the implementation of the methods required by the implemented interface(s).

l_LimitedSplpf—ConditionValue

This describes the behavior of this class when the `ConditionValue()` method is called.

- ➤ When the constructor has not yet been called, then this method returns false and sets the method output *conditionedValue* to zero (0).
- ➤ The time between calls is limited to a minimum of 1 ms and a maximum of 60000 ms. This section references the limited value as *timeElapsedLimited*.
- ➤ The time constant used in calculating the output value, *timeConstantUsed*, is limited such that it must exceed or equal five times the elapsed time between calls of this method, *timeElapsed*.
- ➤ When the *inputValue* is less than *lowLimit*, set in the constructor, then the input is limited to *lowLimit* and the out-of-bounds internal flag is set.
- ➤ When *inputValue* exceeds *highLimit*, set in the constructor, then the input is limited to *highLimit* and the out-of-bounds internal flag is set.
- ➤ When *inputValue* is within the limits outlined in the constructor, the input is filtered through a low-pass filter in order to provide the output and the out-of-bounds internal flag is reset.
- ➤ When all of the following conditions are met:
  - ➢ *inputValue* is less than *highLimit*
  - ➢ *inputValue* is greater than the *lowLimit*
  - ➢ *Alarm* property is false

  this method computes the *conditionedValue* output equivalent to:
  $((inputValue - lastConditionedValue) \bullet 0.632 \bullet \frac{1}{timeConstantUsed} \bullet timeElapsedLimited) + lastConditionedValue$
  where:
  - ➢ *inputValue* is the current input to `ConditionValue()`
  - ➢ *lastConditionedValue* is the input to `ConditionValue()` from the previous scan
  - ➢ *timeConstantUsed* is the range limited time constant
  - ➢ *timeElapsedLimited* is the range limited elapsed time since the last scan
- ➤ When the input is out of range, it is limited to the corresponding range value, and on a subsequent call where the input is within the specified range, the conditioned output ramps to that value from the limit where it was being held.
- ➤ When the input is in alarm, the output, input, and any internal filtering values are set to *defaultOutput*. Once the alarm is removed, the input is no longer overridden and the output value ramps to the input value through the filter, i.e., steps to *defaultOutput* when in alarm and ramps from *defaultOutput* back to *inputValue* after the alarm is removed.

class_LimitedSplpfRampToDefault

Instantiate this class when single-pole low-pass filter that has an imposed range of acceptable values is desired. When in alarm, this class ramps the conditioned value to *defaultOutput*, set in the constructor method, at the same rate it would any other input.

Implemented Interfaces

An interface defines a required set of functionality as methods and properties. As an implementer of any interface all methods and properties declared in that interface must exist as members of this class. This allows multiple generally unrelated classes to be used interchangeably for a specific feature set.

➤ I_LimitedSplpf

LimitedSplpfRampToDefault (Method)

This method acts as the constructor and must be called before the class can operate. It initializes the characteristics of the filter.

Inputs

Name	IEC 61131 Type	Description
highLimit	REAL	The largest valid value for the input variable.
lowLimit	REAL	The smallest valid value for the input variable.
defaultOutput	REAL	The conditioned output defaults to this value if the input is out of range or the alarm is high.
timeConstant	UINT	Range: 100–60000 ms. The time constant to use for the low-pass filter within this method.

Return Value

IEC 61131 Type	Description
POINTER TO STRING	Return a pointer to an error message if an error occurred. Return zero if no errors exist.

Processing

This method:

➤ Sets *defaultOutput* as the initial output and input to the filter in order to eliminate “wind-up” during the first few scans.

➤ Returns a pointer to an error message if *lowLimit* exceeds *highLimit*.

➤ Returns a pointer to an error message if *defaultOutput* is less than *lowLimit* or greater than *highLimit*.

bootstrap_SetInitialValue (Method)

This method may be called at startup if something other than the *defaultOutput* (set previously in the constructor method call) is desired as the initial value by the user.

Inputs

Name	IEC 61131 Type	Description
initialValue	REAL	Range: $lowLimit \leq initialValue \leq highLimit$. Sets the initial value to be used by the filter at startup.

Return Value

IEC 61131 Type	Description
POINTER TO STRING	Return a pointer to an error message if an error occurred. Return zero if no errors exist.

Processing

This method:

- ➤ Bypasses all internal filtering and changes both the input and conditioned output to be equal to *initialValue*.
- ➤ Returns a pointer to an error message if the constructor has not been called.
- ➤ Returns a pointer to an error message if *initialValue* is less than *lowLimit* or greater than *highLimit* set in the constructor.

Processing of Interface Methods

This section provides specifics regarding the implementation of the methods required by the implemented interface(s).

I_LimitedSplpf—ConditionValue

This describes the behavior of this class when the `ConditionValue()` method is called.

- ➤ When the constructor has not yet been called, then this method returns false and sets the method output *conditionedValue* to zero (0).
- ➤ The time between calls is limited to a minimum of 1 ms and a maximum of 60000 ms. This section references the limited value as *timeElapsedLimited*.
- ➤ The time constant used in calculating the output value, *timeConstantUsed*, is limited such that it must exceed or equal five times the elapsed time between calls of this method, *timeElapsed*.
- ➤ When the *inputValue* is less than *lowLimit*, set in the constructor, then the input is limited to *lowLimit* and the out-of-bounds internal flag is set.
- ➤ When *inputValue* exceeds *highLimit*, set in the constructor, then the input is limited to *highLimit* and the out-of-bounds internal flag is set.

- When *inputValue* is within the limits outlined in the constructor, the input is filtered through a low-pass filter in order to provide the output and the out-of-bounds internal flag is reset.
- When all of the following conditions are met:
  - *inputValue* is less than *highLimit*
  - *inputValue* is greater than the *lowLimit*
  - *Alarm* property is false

  this method computes the *conditionedValue* output equivalent to:
  $((inputValue - lastConditionedValue) \bullet 0.632 \bullet \frac{1}{timeConstantUsed} \bullet timeElapsedLimited) + lastConditionedValue$
  where:
  - *inputValue* is the current input to `ConditionValue()`
  - *lastConditionedValue* is the input to `ConditionValue()` from the previous scan
  - *timeConstantUsed* is the range limited time constant
  - *timeElapsedLimited* is the range limited elapsed time since the last scan
- When the input is out of range, it is limited to the corresponding range value, and on a subsequent call where the input is within the specified range, the conditioned output ramps to that value from the limit where it was being held.
- When the input is in alarm, the input is overridden and set to *defaultOutput*. This allows the output value to ramp to *defaultOutput* through the filter. Once the alarm is removed, the input is no longer overridden and the output value ramps to the input value through the filter, i.e. ramps to *defaultOutput* through the filter when in alarm and ramps from *defaultOutput* back to *inputValue* after the alarm is removed.

Benchmarks

Benchmark Platforms

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

- SEL-3530
  - R134 firmware
- SEL-3555
  - Dual-core Intel i7-3555LE processor
  - 4 GB ECC RAM
  - R134-V1 firmware
- SEL-3505
  - R134 firmware

Benchmark Test Descriptions

Any time less than one microsecond was rounded up to one microsecond for this report.

class_LimitedSplpfStepToDefault—ConditionValue

The posted time is the average execution time of 100 consecutive calls.

class_LimitedSplpfRampToDefault—ConditionValue

The posted time is the average execution time of 100 consecutive calls.

class_ArmaFilter—ConditionValue

The posted time is the average execution time of 100 consecutive calls.

class_ArmaFilter_LREAL—ConditionValue

The posted time is the average execution time of 100 consecutive calls.

class_PassThroughFilter—ConditionValue

The posted time is the average execution time of 100 consecutive calls.

Benchmark Results

ConditionValue Timing Results

Operation Tested	**Platform (time in μs)**		
	SEL-3505	**SEL-3530**	**SEL-3555**
class_LimitedSplpfStepToDefault	15	11	2
class_LimitedSplpfRampToDefault	17	5	1
class_ArmaFilter	1	1	1
class_ArmaFilter_LREAL	1	1	1
class_PassThroughFilter	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.

Filtering with class_LimitedSplpfStepToDefault

The example code shown in *Code Snippet 1* demonstrates a very simple use of this library. This code instantiates a simple filter with the following attributes:

1. The filter output has a range of 0–100 inclusive.
2. If the filter input goes out of range or the *Alarm* property is set, the output steps to the default value of 50.
3. The filter time constant is 1000 ms.
4. The filter has an initial value of zero.

This code filters *rawValue* normally for 100 time steps. After 100 time steps, *Alarm* is set to true.

Code Snippet 1 prg_FilterStepToDefault

```
PROGRAM prg_FilterStepToDefault
VAR
    initialized : BOOL := FALSE;
    filter : class_LimitedSplpfStepToDefault;
    rawValue : REAL := 75;
    filteredValue : REAL;
    step : UINT;
END_VAR
```

```
IF NOT initialized THEN // Only initialize the filter once.
    // Initialize the filter with a range of 0-100, a default output
    // of 50, and a time constant of 1000 ms.
    filter.LimitedSplpfStepToDefault(highLimit := 100, lowLimit := 0,
           defaultOutput := 50, timeConstant := 1000);
    // Start the filter with an initial value of zero.
    filter.bootstrap_SetInitialValue(0);
    initialized := TRUE;
END_IF

IF step < 100 THEN
    // Filter normally for 100 time steps.
    filter.ConditionValue(rawValue, conditionedValue => filteredValue);
    step := step + 1;
ELSE
    // After 100 time steps, set the Alarm property.
    filter.Alarm := TRUE;
    filter.ConditionValue(rawValue, conditionedValue => filteredValue);
END_IF
```

When this code is executed, the *filteredValue* will start at zero and move towards the input value of 75 with each time step, according to the filter and time constant. After 100 time steps, the output steps directly to the default value of 50 because the *Alarm* property was set. *Figure 2* shows an example of the filtered output plotted against time, where each step is 100 ms.

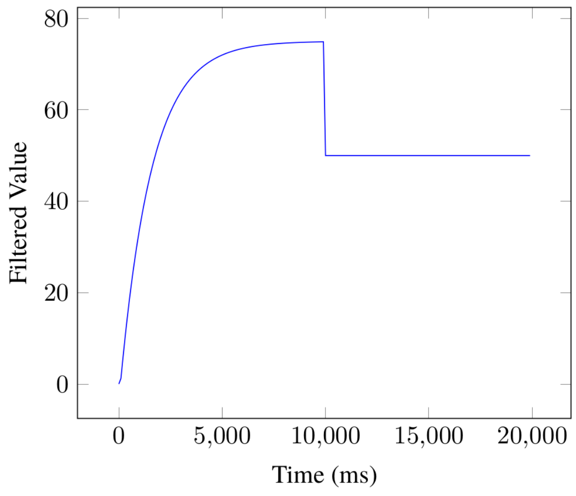

Figure 2 class_LimitedSplpfStepToDefault With a Time Constant of 1000 ms

Filtering with class_LimitedSplpfRampToDefault

The example code shown in *Code Snippet 2* demonstrates a simple use of this library. This code instantiates a simple filter with the following attributes:

1. The filter output has a range of 0–100 inclusive.
2. If the filter input goes out of range or the *Alarm* property is set, the output will ramp to the default value of 50.
3. The filter time constant is 1000 ms.
4. The filter has an initial value of zero.

This code filters *rawValue* normally for 100 time steps. After 100 time steps, *Alarm* is set to true.

Code Snippet 2 prg_FilterRampToDefault

```
PROGRAM prg_FilterRampToDefault
VAR
    initialized : BOOL := FALSE;
    filter : class_LimitedSplpfRampToDefault;
    rawValue : REAL := 75;
    filteredValue : REAL;
    step : UINT;
END_VAR
```

Code Snippet 2 prg_FilterRampToDefault (Continued)

```
IF NOT initialized THEN // Only initialize the filter once.
    // Initialize the filter with a range of 0-100, a default
        output
    // of 50, and a time constant of 1000 ms.
    filter.LimitedSplpfRampToDefault(highLimit := 100,
        lowLimit := 0,
             defaultOutput := 50, timeConstant := 1000);
    // Start the filter with an initial value of zero.
    filter.bootstrap_SetInitialValue(0);
    initialized := TRUE;
END_IF

IF step < 100 THEN
    // Filter normally for 100 time steps.
    filter.ConditionValue(rawValue, conditionedValue =>
        filteredValue);
    step := step + 1;
ELSE
    // After 100 time steps, set the Alarm property.
    filter.Alarm := TRUE;
    filter.ConditionValue(rawValue, conditionedValue =>
        filteredValue);
END_IF
```

When this code is executed, the *filteredValue* will start at zero and move towards the input value of 75 with each time step, according to the filter and time constant. After 100 time steps, the output ramps to the default value of 50 because the *Alarm* property was set. *Figure 3* shows an example of the filtered output plotted against time, where each step is 100 ms.

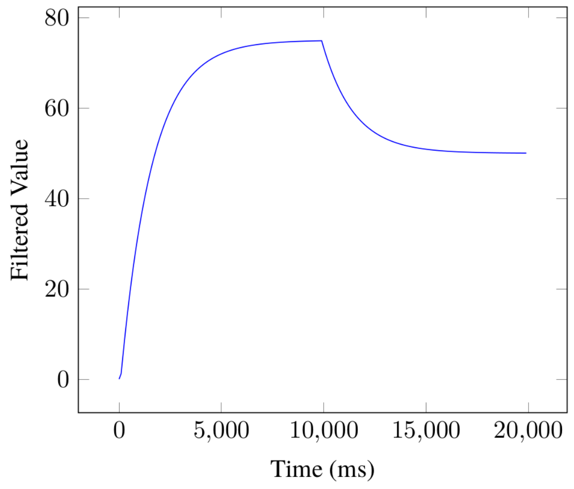

Figure 3 class_LimitedSplpfRampToDefault With a Time Constant of 1000 ms

Filtering with class_ArmaFilter

The ARMA filter is best suited to filtering oscillating signals, and depending on the coefficients used, provides a high-pass, low-pass, band-pass, or band-stop filter. This example shows a specific implementation of the ARMA filter, but the basic approach described is general, and can be used to provide whatever filtering the user of this library requires.

Objective

An oscillating signal can be run through a low-pass filter to remove high-frequency noise, leaving only the lower frequency signals of interest.

Assumptions

The signal provided to the filter is generated in IEC 61131 code by adding a high-frequency component, a low-frequency component, and a mid-frequency component. This signal is then sent through an ARMA filter, with coefficients set using the Butterworth method to remove the high-frequency component, leaving the mid- and low-range frequencies.

The signal sent to the filter is comprised using the following equations:

$$Sample1_n = 0.25 \times COS((1/10) \times 2\pi n)$$

$$Sample2_n = 2 \times SIN((1/100) \times 2\pi n)$$

$$Sample3_n = SIN((1/1000) \times 2\pi n)$$

where n is the sample number.

These equations will provide one sample each time n is increased, and these discrete samples are described as follows:

$Sample1_n$ will have a period of 10 samples (high-frequency), a $\frac{period}{sample}$ ratio of 0.1, be offset by 90 degrees from the other two waves and a magnitude of $1/8$ of the primary frequency.

$Sample2_n$ is the primary frequency. It will have a period of 100 samples (mid-frequency) and a $\frac{period}{sample}$ ratio of 0.01.

$Sample3_n$ will have a period of 1000 samples (low-frequency), a $\frac{period}{sample}$ ratio of 0.001 and a magnitude that is half of the primary frequency.

The sum of these three sine waves is fed into the low-pass filter.

$$FilterInput_n = Sample1_n + Sample2_n + Sample3_n$$

Solution

Coefficients for a Butterworth low-pass filter(http://octave.sourceforge.net/ signal/function/butter.html), which will filter out the high-frequency noise with a filter depth of three (3), are determined using OCTAVE (http://octave-online.net/), by entering the equation defined in *Code Snippet 3*.

Code Snippet 3 OCTAVE Code to Design a Butterworth Filter

```
octave:1>[B,A] = butter(3, 0.05)
    B =
    4.1655e-04   1.2496e-03   1.2496e-03   4.1655e-04

    A =
    1.00000   -2.68616   2.41966   -0.73017
```

The number of coefficients required for a low-pass filter with depth three (3) is four (4), so the global parameter g_p_MaxFilterOrder must be set to three (3) or greater, allowing coefficients 0–3 to be provided.

This Butterworth filter is shown to be stable by checking that the roots of the A coefficients have an absolute value less than 1. This can be done using the OCTAVE code shown in *Code Snippet 4*.

NOTE: An unstable filter will not cause the filter to crash, but it will cause the filtered output to become Infinity. Because of the way the ARMA model is constructed, the next output after infinity is reached will be NaN (Not a Number). Once this happens, all future outputs of the filter will be NaN until the filter is reset.

Code Snippet 4 OCTAVE Code to Check Stability of Filter

```
octave:1> abs(roots([1.00000  -2.68616   2.41966   -0.73017]))
    ans =
        0.92455
        0.92455
        0.85420
```

The program shown in *Code Snippet 5* generates the signals, passes the sum of these signals into the low-pass filter, and provides the outputs into an array.

Code Snippet 5 prg_LowpassFilter

```
PROGRAM prg_LowPassFilterDemo
VAR CONSTANT
    c_Steps : UDINT := 1000;
    c_Acoeff : ARRAY[1..g_p_MaxFilterOrder] OF REAL := [
           -2.68616, 2.41966, -0.73017];
    c_Bcoeff : ARRAY[0..g_p_MaxFilterOrder] OF REAL := [
           4.1655e-04, 1.2496e-03, 1.2496e-03, 4.1655e-04];
END_VAR
VAR
    Filter : class_ArmaFilter(c_Acoeff, c_Bcoeff, 3, 4);
    Signals : ARRAY[1..3] OF ARRAY[1..c_Steps] OF REAL;
    DesiredSignals       : ARRAY[1..c_Steps] OF REAL;
    TotalSignal          : ARRAY[1..c_Steps] OF REAL;
    FilterOutput         : ARRAY[1..c_Steps] OF REAL;
    Stage                : UDINT := 0;
END_VAR
VAR_TEMP
    i : UDINT;
END_VAR
```

Code Snippet 5 prg_LowpassFilter (Continued)

```
CASE Stage OF
0:
    ;// Do nothing on the first scan
1:
    FOR i := 1 TO c_Steps DO
        (* Calculate the samples used to create a compound signal *)
        Signals[1][i] := 0.25 * COS(0.1*UDINT_TO_REAL(i)*2*PI);
        Signals[2][i] := 2 * SIN(0.01*UDINT_TO_REAL(i)*2*PI);
        Signals[3][i] := SIN(0.001*UDINT_TO_REAL(i)*2*PI);

        (* Add the low and mid frequency components together to compare
        against filtered output for accuracy. *)
        DesiredSignals[i] := Signals[2][i] + Signals[3][i];

        (* Add the high-frequency component to this signal to observe that
        the Butterworth low-pass filter removes it, leaving just the
        desired signal. *)
        TotalSignal[i] := DesiredSignals[i] + Signals[1][i];

        (* Pass the entire signal into the filter. *)
        Filter.ConditionValue(TotalSignal[i],
                          conditionedValue => FilterOutput[i]);
    END_FOR
2:
    ;// Add a way to print out the results to a file here if desired
ELSE
    ;// Done
END_CASE
Stage := Stage + 1;
```

When this code is executed, the low-pass filter removes the high-frequency components imposed on the samples, leaving the desired $Sample2_n$ and $Sample3_n$ low-frequency components alone.

The plot in *Figure 4* shows the three waveforms added together and the result of the filter. The resulting wave is shifted in time; this time delay is expected from a filter.

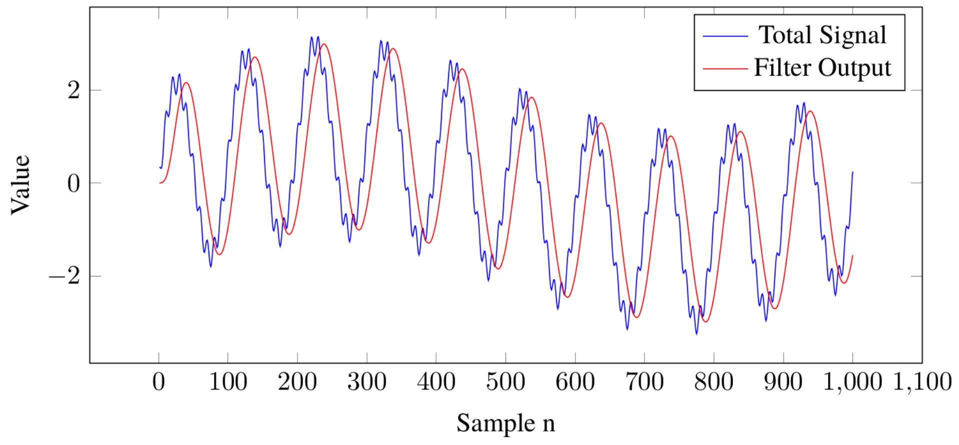

Figure 4 Plot of Total Signal and Filtered Output of the Low-Pass Filter

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 type “Cannot convert” messages. ➤ Must be used with R143 firmware or later.	20180921
3.5.1.1	➤ Added class_ArmaFilter_LREAL.	20150722
3.5.1.0	➤ Changed I_LimitedSplpf to inherit from I_Filter. ➤ Modified class_LimitedSplpfStepToDefault to implement updated I_LimitedSplpf. ➤ Modified class_LimitedSplpfRampToDefault to implement updated I_LimitedSplpf. ➤ Added advanced ARMA filter. ➤ Added dummy class_PassThroughFilter.	20141107
3.5.0.1	➤ Initial release.	20140701