RL78 Family

RL78 Digital Signal Controller Library - Filter

Introduction

This document presents the specifications for a Digital Signal Controller (DSC) Library function library for the Renesas RL78 which includes generic specifications, detailed specifications for filter algorithm kernels and guidelines for the DSC Library API. In this document, the word kernel refers to a common DSCL function such as FIR filter. In the DSC library, several different C-language function calls may be associated with a single DSP kernel. To avoid confusion, the word kernel refers to the DSP algorithm, including the collection of functions that implement the DSP algorithm in the DSC Library. Specific DSC Library function names are used to refer to individual functions.

Target Device

RL78/G14, RL78/G23, RL78/G15, RL78/G24

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.

Contents

1. DSC Library Kernels

Filter kernels defined in this document include:
1. Generic FIR
2. IIR Biquad
3. Single-pole IIR

1.1 List of Abbreviations and Acronyms

Table 1-1 List of Abbreviations and Acronyms

Abbreviation	Full Form
DSC	Digital Signal Controller
DSP	Digital Signal Processor
FIR	Finite Impulse Response
GPIO	General Purpose I/O
I/O	Input/Output
LSB	Least Significant Bit
MSB	Most Significant Bit

1.2 DSC Library Build information

1.2.1 Tool Chain Information

The DSC Library was built and tested using the following:

Table 1-2 DSC library for RL78/G14 and RL78/G23

C Compiler Version	Integrated Development Environment
CC-RL V1.10.00	CS+ for CC V8.06.00
	e²studio Version: 2021-07 (21.7.0)
IAR C/C++ Compiler for Renesas RL78 version 4.21.1.2409 (4.21.1.2409)	IAR Embedded Workbench for Renesas RL78 version 4.21.1
LLVM V10.0.0.202203	e²studio Version: 2022-04 (22.4.0)

Table 1-3 DSC library for RL78/G15

C Compiler Version	Integrated Development Environment
CC-RL V1.11.00	CS+ for CC V8.08.00
	e²studio Version: 2022-10 (22.10.0)
IAR C/C++ Compiler for Renesas RL78 version 4.21.1.2409 (4.21.1.2409)	IAR Embedded Workbench for Renesas RL78 version 4.21.1
LLVM V10.0.0.202207	e²studio Version: 2022-10 (22.10.0)

Table 1-4 DSC library for RL78/G24 for FAA

C Compiler Version	Integrated Development Environment
CC-RL V1.12.00	CS+ for CC V8.09.00
	e²studio Version: 2023-01 (23.01.0)
IAR C/C++ Compiler for Renesas RL78 5.10.3.2716 (5.10.3.2716)	IAR Embedded Workbench for Renesas RL78 V5.10.3

2. DSC Library API

In this document we outline aspects of the Renesas DSC Library API design that are common to all functions within the DSC Library.

2.1 Terminology

In this document we use the word "kernel" to refer to a DSP algorithm (or a variant of a DSP algorithm) implemented in the DSC Library. We use the word "function" to refer to a single, specific function call in the DSC Library API. Note that a kernel's implementation may require multiple functions. For example, a filter kernel typically requires one or more functions for initialization and/or other housekeeping tasks, and a main function for the filter processing.

2.2 Data structures

The library will define the following categories of data structures:
- Vectors
- Algorithm kernel handles

2.2.1 Vectors

Vector data structures contain vector dimensions, and a pointer to the actual array of data:

```
typedef struct
{
    uint32_t n;
    void *data;
} vector_t;
```

Note: the user is responsible for allocating buffer memory for the vector data. In addition, because the "data" member of the vector structure is declared as (void *), there is no need for separate vector structures for the different data types supported by the library.

2.2.2 Algorithm kernel handles

For kernel functions that require state information, constant data, and/or various run-time parameters, all such data is aggregated in a "handle" data structure specific to the kernel function (or class of functions, e.g. transforms). For example, an FIR filter handle may be defined as follows:

```
typdef struct
{
        uint16_t taps;    // number of filter taps
        void *coefs;      // pointer to filter coefficients
        void *state;      // pointer to filter state data, including
                          // the filter's delay line and any other
                          // implementation-dependent state
        uint16_t options; // option flags that may specify rounding,
                          // saturation, or other behaviors
} r_dscl_firfilter_t;
```

Note: the handle data structure contains only members that need to be visible to the user. Some kernels may need to maintain additional implementation-specific states.

All members of kernel "handle" data structures must be initialized by the user. This includes pointers to coefficient and/or state memory. Coefficient and/or state memory must be allocated by the user. Some DSP kernels have implementation-dependent memory requirements for the state and/or coefficients. In such cases, API functions are provided that return the amount of memory to be allocated, given the desired parameters for the kernel.

Also note that many functions will have to perform run-time checks on handle structure members such as '*options*,' in order to branch to the appropriate implementation of the kernel for the given parameters. To ensure that the most common implementation selections accrue the smallest overhead from these run-time checks, a default value of NULL is defined wherever possible. The default value provides the most common desired behavior (often the fastest possible implementation of the kernel).

Kernel parameters supplied in a handle structure must not be changed by the user without re-initializing the kernel. For example, changing the rounding mode or number of taps of an FIR filter "on the fly" is prohibited. When changing such kernel parameters, users must ensure that sufficient memory is allocated for the kernel's internal state, given the new parameters, and must re-initialize the kernel. Note that this restriction does not apply to changes to filter coefficient values. Filter coefficients can be always changed "on the fly."

2.3 Function arguments

All functions accept arguments in the following order:

<handle>: a pointer to a kernel handle data structure containing kernel-specific state, coefficients, parameters, and options.

<input1>…<inputN>: one or more input arguments passed as pointers for most data types, except scalar data. Scalar data values may be passed directly.

<output1>…<outputN>: one or more output pointers

<additional options>: any kernel parameters or options that are not included in the kernel handle data structure.

Note: a function call may not include all of the above elements. For example, a FIR filter initialization function does not have inputs or outputs.

Most functions return to a 16-bit integer result. The integer result may contain an error code or other information required for application housekeeping tasks. For example, the return value may indicate the amount of memory that the user must allocate for a kernel's internal state or may indicate the occurrence of kernel-specific special conditions. Exceptions to this rule occur where functions compute a single real-valued scalar result, and no error conditions can occur. In such cases, the result may be returned by the function instead of a status code.

Functions whose return value indicate the amount of memory that the user must allocate for some tasks may also return a negative value to indicate an error condition (see section 2.4 Error handling). Since C99, the *malloc*() function expects *size_t*, which is an unsigned data type. The actual bit-width of *size_t* is platform dependent. Therefore, care must be taken to ensure that a valid (non-error) result has been returned by the DSC Library function before passing the result to *malloc*().

In summary, most function calls have the following format:

```
int16_t <status/size> = function(<handle>, <input1>,…, <inputN>, <output1>,…,
<outputN>, <additional options>);
```


Where most functions will include only a subset of the above argument classes.

2.4 Error handling

All functions perform checks on input arguments and kernel parameters to the greatest extent possible. Most functions return to a 16-bit integer status code. Functions that return the size of a kernel's internal state (for memory allocation purposes) are a noteworthy exception. For example, the R_DSCL_FIR_stateSize_i16i16 functions fall in this latter category.

All functions return a negative integer value to indicate an error condition. Specific error conditions are assigned unique negative integer values specified on a per-function basis. Functions may return zero for success, or may return a positive integer value to indicate non-error result or special condition. For example, the R_DSCL_FIR_stateSize_i16i16 functions return the memory size requirement for the FIR filter's state. Other functions may return positive integer values indicating special non-error conditions such as the occurrence of overflow.

Functions that return a memory size result for memory allocation purposes may return zero if no memory is required for the given kernel parameters.

Note the distinction between error and status conditions: error conditions (declared with R_DSCL_ERR_<description>) always have a negative integer value and indicate conditions that prevent operation of a kernel, e.g. a NULL input pointer. In contrast, status conditions (declared with R_DSCL_STATUS_<description>) have positive integer values (or zero for R_DSCL_STATUS_OK) and indicate conditions that may impact the kernel's output, but do not prevent the arithmetic operations of the kernel from proceeding. For example, arithmetic overflow is indicated with a status condition. Thus, status conditions can be ignored in some applications, while error conditions always require attention. The assignment of negative values to error codes and positive (or zero) values to status codes allows the user's code to easily distinguish between these two types of conditions.

Error and status codes will be defined by an *enum* declaration in the header file *r_dscl_types.h*.

The following common formats will be used for error codes of all functions:

R_DSCL_STATUS_OK: No issues encountered. This code has a value of zero.

R_DSCL_ERR_<pointer>_NULL: A null pointer was encountered by the function. <pointer> names the offending pointer, e.g. R_DSCL_ERR_INPUT_NULL means that the pointer to an input argument is null. If *input* is a vector or matrix, this code would also be used if the data pointer within the vector/matrix struct is null.

R_DSCL_ERR_INVALID_<x>: an option or parameter passed to the function (via a handle or directly) is not supported by the implementation. <x> identifies the offending function argument, kernel parameter, or struct member. For example, if a filter's handle structure has an 'options' member that specified rounding modes, and this member has an unsupported value, then the R_DSCL_ERR_INVALID_OPTIONS code would be used.

In addition, some function-specific errors and status codes are defined. The error and status codes defined in Phase 1 of the DSC Library specification include:

R_DSCL_STATUS_OK = Status Ok, no issues encountered.

R_DSCL_ERR_HANDLE_NULL = If the pointer to the handle is NULL.

R_DSCL_ERR_INPUT_NULL = If the pointer to the input vector or the data therein is NULL.

R_DSCL_ERR_OUTPUT_NULL = If the pointer to the output vector or the data therein is NULL.

R_DSCL_ERR_STATE_NULL = If the pointer to the FIR or IIR filter internal state is NULL.

R_DSCL_ERR_COEFF_NULL = If the pointer to the coefficient array is NULL.

R_DSCL_ERR_INVALID_TAPS = The number of filter taps is zero or is not supported by the implementation.

R_DSCL_ERR_INVALID_STAGES = The number of filter stages is zero or is not supported by the implementation.

R_DSCL_ERR_INVALID_OPTIONS = *options* value in *handle* specified a mode not currently supported.

2.5   Rounding Support

Some kernels in the DSC Library provide support for multiple rounding modes. These modes apply to the fixed-point data types.

Rounding modes are supported via an options element in the kernel's handle structure. The following bit-fields in options are reserved for rounding and saturation modes:

- Bits 0-2: rounding mode
  - R_DSCL_ROUNDING_DEFAULT = 0
  - R_DSCL_ROUNDING_TRUNC = 1
  - R_DSCL_ROUNDING_NEAREST = 2
  - reserved = 3-7

Note: R_DSCL_ROUNDING_DEFAULT is the default behaviors of the kernals. For all the filter types in the library, the default behavior is the truncation.

3. Filter Function API

This section covers the filter functions implemented in the RL78 DSC Library.

3.1 FIR Data Structure Definition

The FIR kernel uses a handle to the filter of type r_dscl_firfilter_t. This handle is passed as part of the call to the filter. The data structure for the handle type is as follows:

```
typedef struct
{
    uint16_t  taps;  // number of filter taps
    void *    coefs; // pointer to filter coefficients
    void *    state; // pointer to filter state data, including the filter's
delay line
                     // and any other implementation-dependent state
    uint16_t  options;// options that specify rounding, saturation, or other
behaviors
} r_dscl_firfilter_t;
```

Each member of the data structure is explained below:

  taps = Number of filter taps

  coefs = Pointer to the coefficient vector (must be the same data type as the input vector). The content of this array is maintained by the user.

  state = Pointer to the internal state of the filter, including the delay line and any other implementation-dependent state. The memory for the internal state is allocated by the user and the content of the internal state is maintained by the kernel.

  options = A bit-mapped parameter controlling options. See "Rounding Support" in software overview section, for the definition of available modes.

3.2 FIR Initialize API

This is a function used to initialize the filter state (including zeroing the delay line and other parameters), given the options specified in the handle. It must be called once prior to invoking the run-time calling function.

Format

```
int16_t R_DSCL_FIR_Init_i16i16 (r_dscl_firfilter_t * handle)
```

Parameters

handle   Pointer to an instance of the r_dscl_firfilter_t data structure.

handle →state   Pointer to the starting address of delay line, which shall be on the same array as the input data.

Return Values

R_DSCL_STATUS_OK = Status OK, no issues encountered.
R_DSCL_ERR_HANDLE_NULL = Pointers to the handle is NULL.
R_DSCL_ERR_STATE_NULL = Pointer to delay-line, is NULL.
R_DSCL_ERR_INVALID_TAPS = Number of taps is 0.
R_DSCL_ERR_INVALID_OPTIONS = options value in handle specified a mode not currently supported.
Others = Reserved.
Note: This function initializes only the contents of the FIR state pointed to by the state element of the handle structure. It does not initialize the filter coefficients or any other contents of the handle structure, which must be initialized separately.

Example

Since this function is not used by itself, see FIR filter example for the use of this in Context.

Restrictions

Handle must have already been instantiated. See FIR example for more details.

3.3 FIR Filter API

The block finite impulse response (FIR) filter kernel operates on a user selectable number of input samples and produces the same number of output samples each time it is invoked.

Format

```
int16_t R_DSCL_FIR_i16i16 (const r_dscl_firfilter_t * handle, const vector_t *
input, vector_t * output)
```

Parameters

handle	Pointer to an instance of the r_dscl_firfilter_t data structure.
input	Pointer to an instance of vector_t data structure for the input data. Neither the instance nor the actual input data will be altered by the function.
input→n	Number of input samples to be processed by the function. This value must be set before the function call.
input→data	Pointer to starting address of the input data. This pointer must be set before the function call.
output	Pointer to an instance of vector_t data structure for the output data. Both the instance and the actual output data will be altered by the function.
output→n	Number of output samples produced by the function. This value will be filled by the function.
output→data	Pointer to the output data buffer. This pointer must be set before the function call. The output data buffer will be filled by the function.

Return Values

R_DSCL_STATUS_OK = Status OK, no issues encountered.
R_DSCL_ERR_HANDLE_NULL = If the pointer to the handle is NULL.
R_DSCL_ERR_INPUT_NULL = If the pointer to the input vector or the data therein is NULL.
R_DSCL_ERR_OUTPUT_NULL = If the pointer to the output vector or the data therein is NULL.
R_DSCL_ERR_STATE_NULL = If the pointer to the filter internal state is NULL.
R_DSCL_ERR_COEFF_NULL = If the pointer to the coefficient array is NULL.
R_DSCL_ERR_INVALID_TAPS = The number of filter taps is zero.
R_DSCL_ERR_INVALID_OPTIONS = options value in handle specified a mode not currently supported.
Others = Reserved.

Description

The Block FIR filter kernels implement a finite impulse response filter on each input sample. The following equation shows the general structure of a T-tap FIR filter where h represents the coefficients, x represents the input data, and y represents the output data.

$$y(n) = \sum_{i=0}^{T-1} h(i) * x(n - i)$$

Each output sample is the result of performing FIR filter of n taps. This is shown graphically in Figure 3-1.
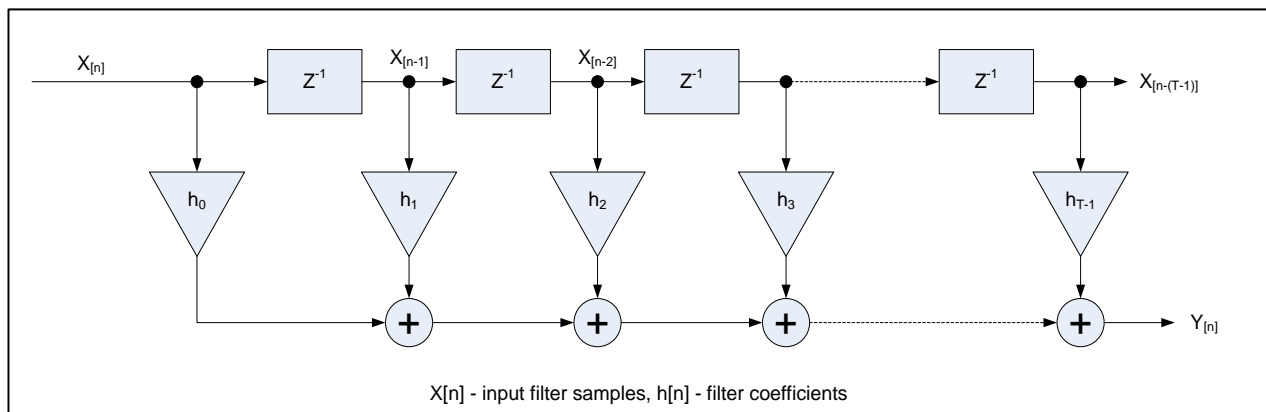
Figure 3-1 FIR Filter

Fixed-point Behavior

Since the function is implemented by fixed point, the behavior of fixed point must be taken care. The following issues must be considered:

- Scaling
- Overflow

Scaling: The scaling factor "*FIR_SCALE_A*" for the output data is defined in "*r_dscl_filter_asm.inc*". Results are right-shifted by scale prior to writing the output to memory.

The scale must be equal to the number of fraction bits of coefficient. For example:

If the filter coefficients are in Q4.12 format and the filter's input is in Q2.14 format, then the accumulated result for each output sample is in Q6.26 format. The scale value of 12 should be set, such that it will perform the required conversion by discarding 12 LSBs of the accumulated result, leaving 14 fractional bits in the final output word.

The default value of this scaling factor is 15. If the value is changed, the library needs to be compiled again.

Overflow: The function is optimized for speed at the cost of precision and overflow protection. It's implemented by using a sequence of multiply-accumulate operations. The accumulator is just 32 bits, overflows may happen. After accumulations, the final result is converted to 16 bits, precision is also lost. To avoid overflows completely, the input data must be scaled down by $\log2(\text{taps})$ bits (Maximum is 15 bits).

Example

The following is an example showing initialization and run-time usage for the FIR Filter.

```
#define NUM_TAPS        (64)
#define NUM_SAMPLES     (200)

r_dscl_firfilter_t  myFilterHandle;    // instantiate a handle for this filter
vector_t            myInput;           // See introduction section describing
the API document
vector_t            myOutput;          // for a definition of the "vector_t"
data type.

// Coefficients should be stored in time-reversed order
int16_t     myCoeffs[NUM_TAPS] = {…};

// The input data buffer should contain previous (T-1) input samples (i.e.
delay line)
// contiguous with the present (N) input samples
int16_t   inputData[NUM_TAPS - 1 + NUM_SAMPLES];
int16_t   outputData[NUM_SAMPLES];
int16_t   myFIRFlags;

/*-------------  Set up the FIR filter  ------------*/
myFilterHandle.taps  = NUM_TAPS;
myFilterHandle.options = 0;   // default

/* No need to call StateSize API for FIR */
myFilterHandle.state = (void *)&inputData[0];      // starting address of
delayline

/*----- Initialize the coefficients and internal state ------*/
myFilterHandle.coefs = (void *)myCoeffs;
myFIRFlags = R_DSCL_FIR_Init_i16i16(&myFilterHandle);

/*---------------  Set up the input/output  ---------------*/
myInput.n = NUM_SAMPLES;
myInput.data  = (void *)&inputData[NUM_TAPS - 1]; // starting address of
current input block
myOutput.data = (void *)outputData;
/*-----------------  Wait for input data  -----------------*/
/*--------------  Main library function call  --------------*/
myFIRFlags = R_DSCL_FIR_i16i16 (&myFilterHandle, &myInput, &myOutput);

/*-------------  Output data are now ready  --------------
 * Note: At this point myOutput.n holds the number of output samples generated
by
 * the library, where the data are written to the array pointed to by
myOutput.data.
 *----------------------------------------------------------*/
```

Processing Flow

The example above just gives an example to do all the filter sample once, in this case, the input buffer and output buffer should be sufficient to hold all the data, if not, we need to filter the input sample into several times. The processing flow is shown in Figure 3-2, including the setting of scaling factor.

```
              ┌──────────────────────┐
              │  Set "FIR_SCALE_A"   │
              └──────────────────────┘
                         │
                         ▼
              ┌──────────────────────┐
              │   Compile Library    │
              └──────────────────────┘
                         │
                         ▼
              ┌──────────────────────┐
              │ Initialization and call │
              │ R_DSCL_FIR_Init_i16i16 () │
              └──────────────────────┘
                         │
                         ▼
              ┌──────────────────────┐
              │ Assign sample number │ ◄───────────┐
              │ to be processed,     │             │
              │ input pointer,       │             │
              │ output pointer       │             │
              └──────────────────────┘             │
                         │                         │
                         ▼                         │
              ┌──────────────────────┐             │
              │ Call R_DSCL_FIR_i16i16 () │         │
              └──────────────────────┘             │
                         │                         │
                         ▼                   No    │
              ┌──────────────────────┐─────────────┘
              │ Check sample finished? │
              └──────────────────────┘
                         │
                        Yes
                         ▼
                  ┌────────────┐
                  │    End     │
                  └────────────┘
```
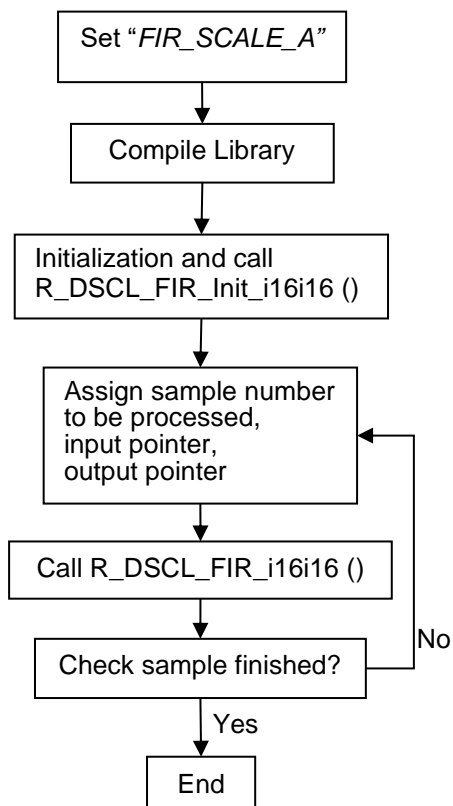
Figure 3-2 Processing flow

Restrictions

The number of taps in the coefficient vector must match that specified in the filter handle.

3.4 IIR Biquad Data Structure Definition

The following is the definition of the filter handle r_dscl_iirbiquad_t.

```
typedef struct
{
    uint16_t        stages;         // number of biquad stages
    void *          coefs;          // pointer to filter coefficients
    void *          state;          // pointer to filter's internal state (delay
line)
    uint16_t        options;        // options that specify rounding,
saturation, or other behaviors
} r_dscl_iirbiquad_t;
```

Each member of the data structure is explained below:

stages = Number of biquad stages

c*oefs* = Pointer to the coefficient vector (must be the same data type as the input vector). The content of this array is maintained by the user.

state = Pointer to the internal state of the filter, including the delay line and any other implementation-dependent state. The memory for the internal state is allocated by the user and the content of the internal state is maintained by the kernel.

options = A bit-mapped parameter controlling options. See "Rounding Support" section, for the definition of available modes.

3.5   IIR Biquad State Size API

This is a "maintenance" function for the IIR filters. This function returns the size (in bytes) that must be allocated by the user to maintain the internal state (including the delay line) of the filter.

Format

```
int16_t R_DSCL_IIRBiquad_StateSize_i16i16 (const r_dscl_iirbiquad_t * handle)
```

Parameters

handle Pointer to an instance of the r_dscl_iirbiquad_t data structure.

Return Values

Buffer Size in Bytes (type int16_t) required by the filter.
Note: The size returned should allow the implementer to use this buffer to maintain any private information associated with the filter including items such as pointers, a record of input and output data types, etc. Also, the size returned by this function does not include the filter handle, nor does it include the coefficient array.

Description

The function can be used as part of the filter initialization to decide the buffer size a user must allocate. Alternatively, the user can use this function to determine the required memory size during development and allocate a static array of that size for the internal state (for example, in the faster on-chip RAM).
Note: Since C99, the *malloc*() function expects *size_t*, which is an unsigned data type. The actual bit-width of *size_t* is platform dependent. If malloc(R_DSCL_IIRBiquad_StateSize_i16i16()) is used to allocate the memory for the internal state, and R_DSCL_IIRBiquad_StateSize_i16i16() returns a negative value, unexpected behavior could happen.

Example

Since this function is not used by itself, see IIR filter example for the use of this in Context.

Restrictions

IIRHandle must have already been instantiated. See IIR example for more details.

3.6   IIR Biquad Initialize API

This is a function used to initialize the filter state (including zeroing the delay line and other parameters), given the options specified in the handle. It must be called once prior to invoking the run-time calling function.

Format

```
int16_t R_DSCL_IIRBiquad_Init_i16i16 (r_dscl_iirbiquad_t * handle)
```

Parameters

handle Pointer to an instance of the r_dscl_iirbiquad_t data structure.

Return Values

R_DSCL_STATUS_OK = Status OK, no issues encountered.
R_DSCL_ERR_HANDLE_NULL = Pointers to the handle is NULL.
R_DSCL_ERR_STATE_NULL = Pointer to delay-line, is NULL.
R_DSCL_ERR_INVALID_STAGES= Number of biquad stages is 0.
R_DSCL_ERR_INVALID_OPTIONS = options value in handle specified a mode not currently supported.
Others = Reserved.
Note: This function initializes only the contents of the IIR state pointed to by the state element of the handle
      structure. It does not initialize the filter coefficients or any other contents of the handle structure,
      which must be initialized separately.

Description

This is a function used to initialize the filter state (including zeroing the delay line and possibly other implementation-dependent parameters). It must be called once prior to invoking the run-time calling function.

Example

Since this function is not used by itself, see IIR filter example for the use of this in Context.

Restrictions

IIRHandle must have already been instantiated. See IIR example for more details.

3.7   IIR Biquad Filter API

This kernel implements the IIR (Infinite Impulse Response) filter in the form of cascaded biquads. A biquad is a section of the 2nd-order IIR filter. For higher order IIR filters, cascaded biquads often generate smaller numerical errors than a direct-form implementation.

A biquad can have many forms, such as direct-forms I and II, transposed direct-forms I and II. Each has its advantages and disadvantages. The IIR biquad API is designed by using direct-forms I.

This kernel operates on a user selectable number of inputs samples and produces the same number of output samples each time it is invoked. The number of cascaded biquads is also selectable by the user.

Fomat

```
int16_t R_DSCL_IIRBiquad_i16i16 (const r_dscl_iirbiquad_t * handle, const
vector_t * input, vector_t * output)
```

Parameters

handle	Pointer to an instance of the r_dscl_iirbiquad_t data structure.
input	Pointer to an instance of vector_t data structure for the input data. Neither the instance nor the actual input data will be altered by the function.
input→n	Number of input samples to be processed by the function. This value must be set before the function call.
input→data	Pointer to the input data buffer. This pointer must be set before the function call.
output	Pointer to an instance of vector_t data structure for the output data. Both the instance and the actual output data will be altered by the function.
output→n	Number of output samples produced by the function. This value will be filled by the function.
output→data	Pointer to the output data buffer. This pointer must be set before the function call. The output data buffer will be filled by the function.

Return Values

R_DSCL_STATUS_OK = Status OK, no issues encountered.
R_DSCL_ERR_HANDLE_NULL = If the pointer to the handle is NULL.
R_DSCL_ERR_INPUT_NULL = If the pointer to the input vector or the data therein is NULL.
R_DSCL_ERR_OUTPUT_NULL = If the pointer to the output vector or the data therein is NULL.
R_DSCL_ERR_STATE_NULL = If the pointer to the filter internal state is NULL.
R_DSCL_ERR_COEFF_NULL = If the pointer to the coefficient array is NULL.
R_DSCL_ERR_INVALID_STAGES= Number of biquad stages is 0.
R_DSCL_ERR_INVALID_OPTIONS = options value in handle specified a mode not currently supported.
Others = Reserved.

Description

The IIR biquad filter is in the form of cascaded biquads. Each biquad is a section of the 2nd-order IIR filter with the following equation between its input and output:

$$y(n) = b0 * x(n) + b1 * x(n-1) + b2 * x(n-2) - a1 * y(n-1) - a2 * y(n-2)$$

where y(n) is the output sample, x(n) is the input sample, y(n-1) and x(n-1) are output and input samples delayed by one sampling period, respectively, y(n-2) and x(n-2) are output and input samples delayed by two sampling periods, respectively, b0, b1, and b2 are feed forward coefficients, and a1 and a2 are feedback coefficients.

The overall transfer function is as below:

$$H(z) = \prod_{0}^{N-1} \frac{b0 + b1z^{-1} + b2z^{-2}}{1 + a1z^{-1} + a2z^{-2}}$$

where N is the number of cascaded biquad stages. Note that each stage has a different set of coefficients b0, b1, b2, a1, and a2.

Figure 3-3 shows the diagram of IIR biquad Direct Form I.

$X^k[n]$ – Input Samples, $Y^k[n]$ Output Samples, $b^k[n]/a^k[n]$ Filter Coefficients, Z^{-1} – Delay Line
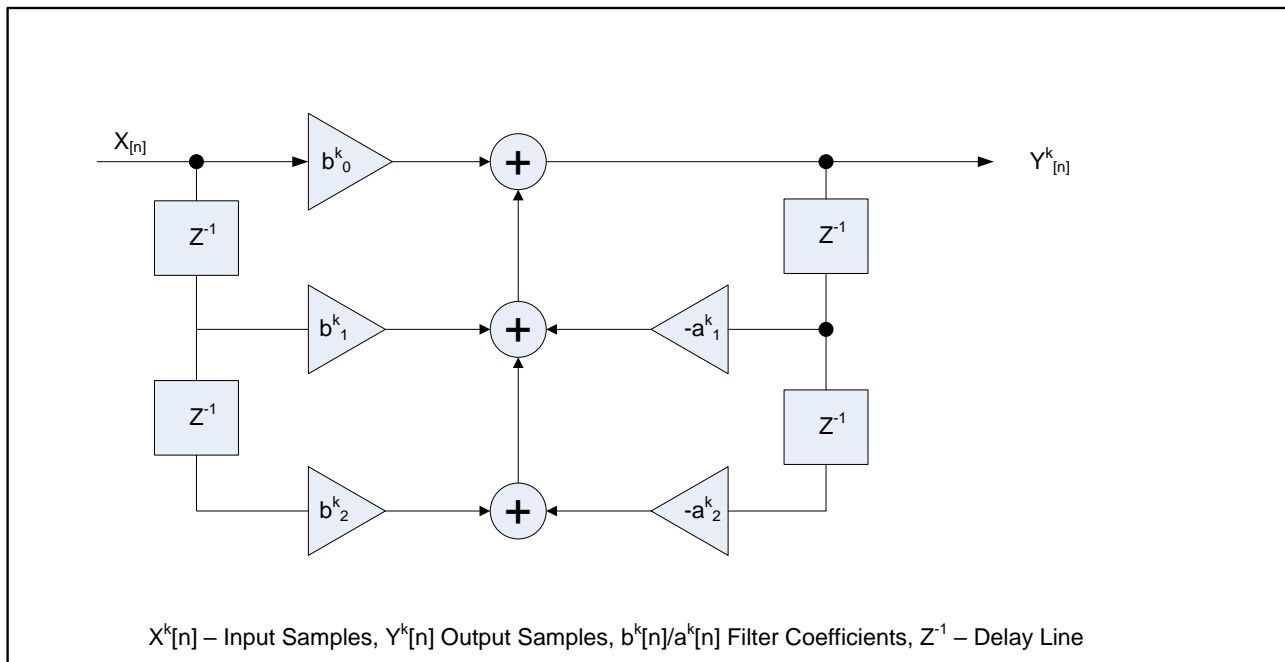
Figure 3-3 IIR Bi-Quad, Direct Form I

Fixed-point Behavior

Since the function is implemented by fixed point, the behavior of fixed point must be taken care. The following issues must be considered:

- Scaling
- Overflow

Scaling: The scaling factor "*IIR_BQ_SCALE_A*" for the output data is defined in "*r_dscl_filter_asm.inc*".
Results are right-shifted by scale prior to writing the output to memory.
The scale must be equal to the number of fraction bits of coefficient. For example:
If the filter coefficients are in Q4.12 format and the filter's input is in Q2.14 format, then the accumulated result for each output sample is in Q6.26 format. The scale value of 12 should be set, such that it will perform the required conversion by discarding 12 LSBs of the accumulated result, leaving 14 fractional bits in the final output word.
The default value of this scaling factor is 14. This means the coefficients can represent the value in the range of [-2, 2]. If all the coefficients values are in the range of [-1, 1], the scaling factor can be changed to 15. To do this, the library needs to be compiled again.

Overflow: The function is optimized for speed at the cost of precision and overflow protection. It's implemented by using a sequence of multiply-accumulate operations. The accumulator is just 32 bits, overflows may happen. After accumulations, the final result is converted to 16 bits, precision is also lost. To avoid overflows completely, the input data must be scaled down by 3 bits (Maximum is 15 bits).

Example

The following is an example showing how to use IIR biquad function.

```
#define NUM_TAPS_PER_BIQUAD       (5)
#define NUM_BIQUAD_STAGES         (3)
r_dscl_iirbiquad_t   myFilterHandle; // instantiate a handle for my use
vector_t                myInput;    // See introduction section API section
vector_t    myOutput;               // for a definition of the "vector_t"
data type
int16_t    myCoeffs[NUM_TAPS_PER_BIQUAD * NUM_BIQUAD_STAGES]
            = {b0, b1, b2, a1, a2,…};
int16_t    myDLine[NUM_TAPS_PER_BIQUAD * NUM_BIQUAD_STAGES];
int16_t    inputData[NUM_SAMPLES];
int16_t    outputData[NUM_SAMPLES];
int16_t    myIIRFlags;
int16_t    dynMemSize, staMemSize;


/*-------------  Set up the IIR filter biquads  ------------*/
myFilterHandle.stages = NUM_BIQUAD_STAGES;

/* Setup data format and options */
myFilterHandle.options  = 0;   // default

/* !!! It is important to setup the stages and the form before */
/* !!! calling function R_DSCL_IIRBiquad_StateSize_i16i16 () */
staMemSize = NUM_TAPS_PER_BIQUAD * NUM_BIQUAD_STAGES * sizeof(int16_t);
dynMemSize = R_DSCL_IIRBiquad_StateSize_i16i16(&myFilterHandle);
if (staMemSize >= dynMemSize)
{
  myFilterHandle.state = (void *)myDLine;  // probably more common
}
else
{
  myFilterHandle.state = malloc((size_t) dynMemSize);  //malloc expects size_t
}
/* Initialize the coefficients and internal state */
myFilterHandle.coefs = (void *)myCoeffs;
myIIRFlags = R_DSCL_IIRBiquad_Init_i16i16(&myFilterHandle);
/*---------------  Set up the input/output  ---------------*/
myInput.n = NUM_SAMPLES;
myInput.data  = (void *)inputData;
myOutput.data = (void *)outputData;
/*----------------  Wait for input data  -----------------*/
/*-------------  Main library function call  -------------*/
myIIRFlags = R_DSCL_IIRBiquad_i16i16(&myFilterHandle, &myInput, &myOutput);
/*-------------  Output data are now ready  --------------*/
/* Note: At this point myOutput.n holds the number of output samples generated
by
* the library, where the data are written to the array pointed to by
myOutput.data.
*-----------------------------------------------------------*/
```

Processing Flow

The example above just gives an example to do all the filter sample once, in this case, the input buffer and output buffer should be sufficient to hold all the data, if not, we need to filter the input sample into several times. The processing flow is shown in Figure 3-4, including the setting of scaling factor.

```
                    ┌─────────────────────────┐
                    │ Set "IIR_BQ _SCALE_A"    │
                    └─────────────────────────┘
                                │
                                ▼
                    ┌─────────────────────────┐
                    │     Compile Library      │
                    └─────────────────────────┘
                                │
                                ▼
            ┌───────────────────────────────────┐
            │ Call                               │
            │ R_DSCL_IIRBiquad_statesize_i16i16  │
            └───────────────────────────────────┘
                                │
                                ▼
            ┌───────────────────────────────────┐
            │ Call                               │
            │ R_DSCL_ IIRBiquad _Init_i16i16     │
            └───────────────────────────────────┘
                                │
                                ▼
            ┌───────────────────────────────────┐
            │ Assign sample number               │◄──────────┐
            │ to be processed,                   │           │
            │ input pointer,                     │           │
            │ output pointer                     │           │
            └───────────────────────────────────┘           │
                                │                            │
                                ▼                            │
            ┌───────────────────────────────────┐           │
            │ Call R_DSCL_ IIRBiquad             │           │
            └───────────────────────────────────┘           │
                                │                     No     │
                                ▼                            │
            ┌───────────────────────────────────┐           │
            │ Check sample                       │───────────┘
            │ finished?                          │
            └───────────────────────────────────┘
                                │ Yes
                                ▼
                    ┌─────────────────────────┐
                    │           End            │
                    └─────────────────────────┘
```
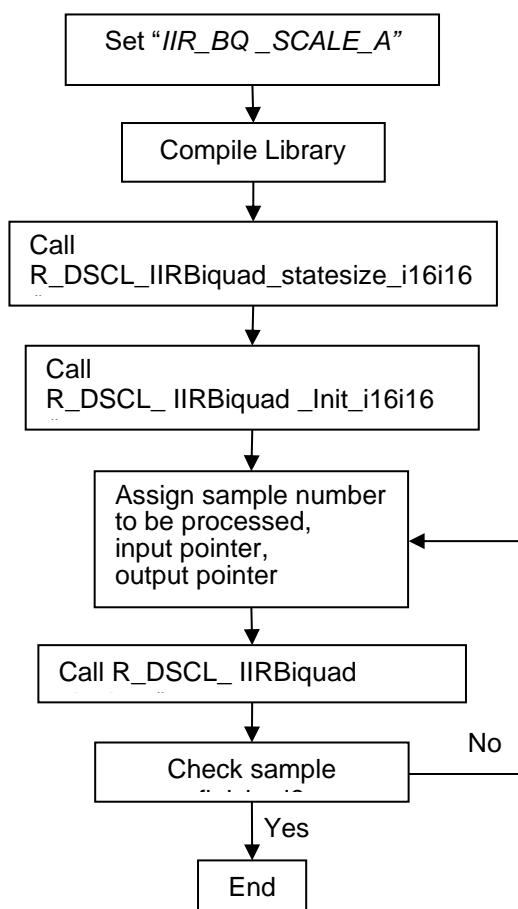
Figure 3-4 Processing flow

Restrictions

The length of the delay line depends on the number of cascaded stages. Therefore, this parameter must be set before the function call to R_DSCL_IIRBiquad_StateSize_i16i16 ().

3.8   Single Pole IIR Data Structure Definition

The following is the definition of the filter handle r_dscl_iirsinglepole_t, which is used for all variants of the single-pole filter kernel.

```
typdef struct
{
    void *        coefs;        // pointer to filter coefficient
    void *        state;        // pointer to filter's internal state (delay
line)
    uin16_t       options;      // options that specify rounding,
saturation, or other behaviors
} r_dscl_iirsinglepole_t;
```

Each member of the data structure is explained below:

coefs = Pointer to the coefficient of the feedback tap (must be the same data type as the input). The coefficient is maintained by the user.

state = Pointer to the state of the feedback tap. The state is maintained by the kernel.

options = A bit-mapped parameter controlling options. See "Rounding Support" section, for the definition of available modes.

3.9   Single-Pole IIR Filter API

This kernel implements the single-pole filter, which is an IIR (Infinite Impulse Response) filter with one feedback tap. The maximum gain is unity.

Format

```
int16_t R_DSCL_IIRSinglePole_i16i16 ( const r_dscl_iirsinglepole_t * handle,
const vector_t * input, vector_t * output)
```

Parameters

handle	Pointer to an instance of the r_dscl_iirsinglepole_t data structure.
input	Pointer to an instance of vector_t data structure for the input data. Neither the instance nor the actual input data will be altered by the function.
input→n	Number of input samples to be processed by the function. This value must be set before the function call.
input→data	Pointer to the input data buffer. This pointer must be set before the function call.
output	Pointer to an instance of vector_t data structure for the output data. Both the instance and the actual output data will be altered by the function.
output→n	Number of output samples produced by the function. This value will be filled by the function.
output→data	Pointer to the output data buffer. This pointer must be set before the function call. The output data buffer will be filled by the function.

Return Values

R_DSCL_STATUS_OK = Status OK, no issues encountered.
R_DSCL_ERR_HANDLE_NULL = If the pointer to the handle is NULL.
R_DSCL_ERR_INPUT_NULL = If the pointer to the input vector or the data therein is NULL.
R_DSCL_ERR_OUTPUT_NULL = If the pointer to the output vector or the data therein is NULL.
R_DSCL_ERR_INVALID_OPTIONS = options value in handle specified a mode not currently supported.
Others = Reserved.

Note: this kernel does not provide an init function. It is the user's responsibility to initialize the internal state to 0.

Description

The single-pole IIR filter is shown graphically in Figure 3-5 and Figure 3-6, low-pass and high pass respectively.
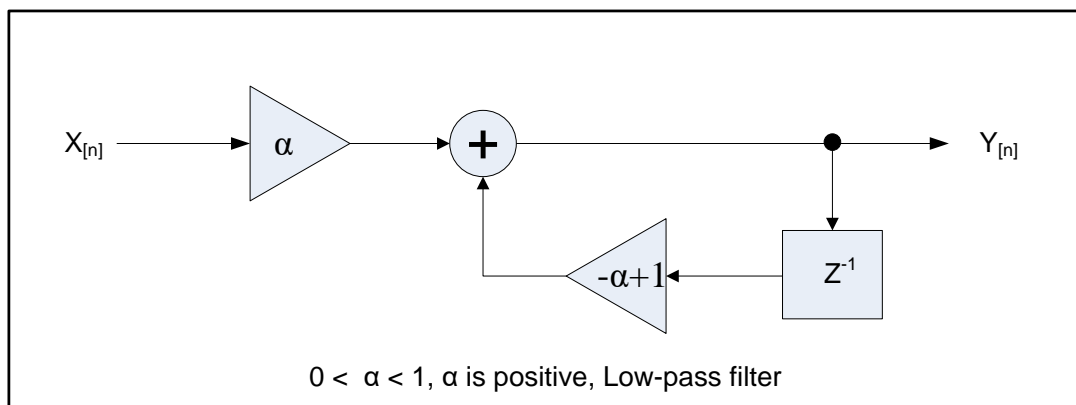
Figure 3-5 Low-pass Single Pole IIR
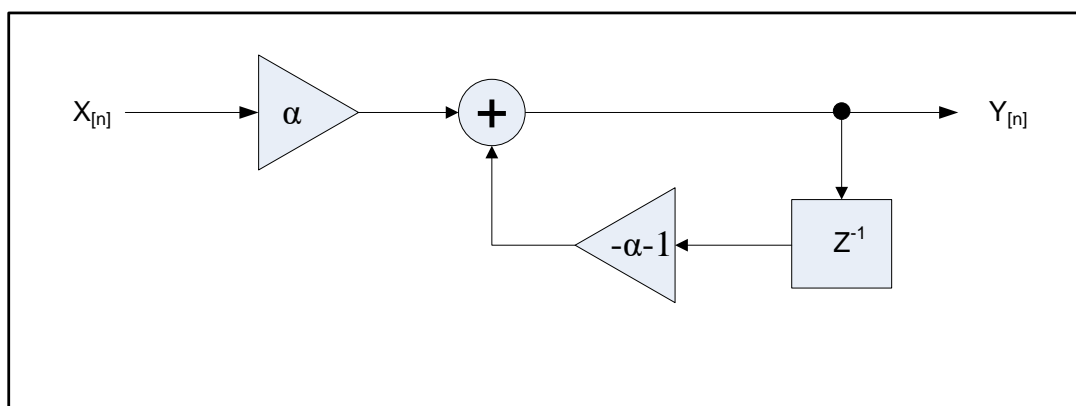
Figure 3-6 High-Pass Single Pole IIR

The single-pole low-pass IIR filter has the following transfer function:

$$H(z) = \frac{a}{1-(1-a)z^{-1}}$$

where the coefficient α is always positive and determines the filter characteristic. When α is 1.0, the filter passes the input signal unchanged. As α decreases toward zero, greater attenuation of high frequencies results. The output of the single-pole low-pass filter can be computed as follows:

$$y_n = y_{n-1}(1-a) + x_n a$$

or

$$y_n = y_{n-1} + (x_n - y_{n-1})a$$

where x_n is the input signal and y_n is the filter output.
A single-pole high-pass filter can be implemented with the transfer function:

$$H(z) = \frac{a}{1+(a+1)z^{-1}}$$

However, this high-pass filter tends to oscillate at the Nyquist frequency as α approaches zero. Because this property of the filter is undesirable in many applications, a single-pole high-pass filter is often implemented by subtracting the output of a single-pole low-pass filter from the input signal. Thus, the high-pass output is the simple difference:

$$y'_n = x_n - y_n$$

Where x_n is the input signal, y_n is the low-pass filter output computed as describe above, and y'_n is the high-pass filter output.

Fixed-point Behavior

Since the function is implemented by fixed point, the behavior of fixed point must be taken. The following issues must be considered:

- Scaling
- Overflow

Scaling: The scaling factor "*IIR_SP_SCALE_A*" for the output data is defined in "*r_dscl_filter_asm.inc*".
Results are right-shifted by scale prior to writing the output to memory.
The scale must be equal to the number of fraction bits of coefficient. For example:
If the filter coefficients are in Q4.12 format and the filter's input is in Q2.14 format, then the accumulated result for each output sample is in Q6.26 format. The scale value of 12 should be set, such that it will perform the required conversion by discarding 12 LSBs of the accumulated result, leaving 14 fractional bits in the final output word.

The default value of this scaling factor is 15. If the value is changed, the library needs to be compiled again.
Overflow: The function is optimized for speed at the cost of precision and overflow protection. It's implemented by using multiply-accumulate operations. The accumulator is just 32 bits, overflows may happen. After accumulations, the final result is converted to 16 bits, precision is also lost. To avoid overflows completely, the input data must be scaled down by 1 bit (Maximum is 15 bits).

Example

The following is an example showing how to use the single-pole IIR function with real 16-bit fixed-point input and output data.

```
r_dscl_iirsinglepole_t  myFilterHandle;
vector_t    myInput;  // See introduction section describing the API document
vector_t    myOutput; // for a definition of the "vector_t" data type.
int16_t     inputData[NUM_SAMPLES];
int16_t     outputData[NUM_SAMPLES];
int16_t     myIIRFlags;
int16_t     mystate;
int16_t     mycoeff;


/*-------------   Set up the single-pole IIR filter ------------*/
mystate = 0; // initialize state
mycoeff = (int16_t) (-0.15 * 0x7FFF);
myFilterHandle.coefs = &mycoeff;
myFilterHandle.state = &mystate;
myFilterHandle.options  = R_DSCL_ROUNDING_TRUNC;


/*---------------   Set up the input/output  ---------------*/
myInput.n = NUM_SAMPLES;
myInput.data  = (void *)inputData;
myOutput.data = (void *)outputData;


/*-----------------  Wait for input data  -----------------*/

/*--------------   Main library function call  -------------*/
myIIRFlags = R_DSCL_IIRSinglePole_i16i16(&myFilterHandle, &myInput,
&myOutput);


/*--------------  Output data are now ready  --------------*/
/* Note: At this point myOutput.n holds the number of output samples generated
by
* the library, where the data are written to the array pointed to by
myOutput.data.
*----------------------------------------------------------*/
```

Processing Flow

The example above just gives an example to do all the filter sample once, in this case, the input buffer and output buffer should be sufficient to hold all the data, if not, we need to filter the input sample into several times. The processing flow is shown in Figure 3-7, including the setting of scaling factor.


```
          ┌─────────────────────────┐
          │  Set "IIR_SP_SCALE_A"   │
          └───────────┬─────────────┘
                      ▼
          ┌─────────────────────────┐
          │     Compile Library     │
          └───────────┬─────────────┘
                      ▼
          ┌─────────────────────────┐
          │      Initialization     │
          └───────────┬─────────────┘
                      ▼
          ┌─────────────────────────┐
          │ Assign sample number    │◄──────┐
          │ to be processed,        │       │
          │ input pointer,          │       │
          │ output pointer          │       │
          └───────────┬─────────────┘       │
                      ▼                      │
          ┌─────────────────────────┐       │
          │ Call R_DSCL_ IIRSinglePole _i16i16 │   │
          └───────────┬─────────────┘       │
                      ▼               No     │
          ┌─────────────────────────┐───────┘
          │      Check sample       │
          └───────────┬─────────────┘
                      │ Yes
                      ▼
          ┌─────────────────────────┐
          │          End            │
          └─────────────────────────┘
```
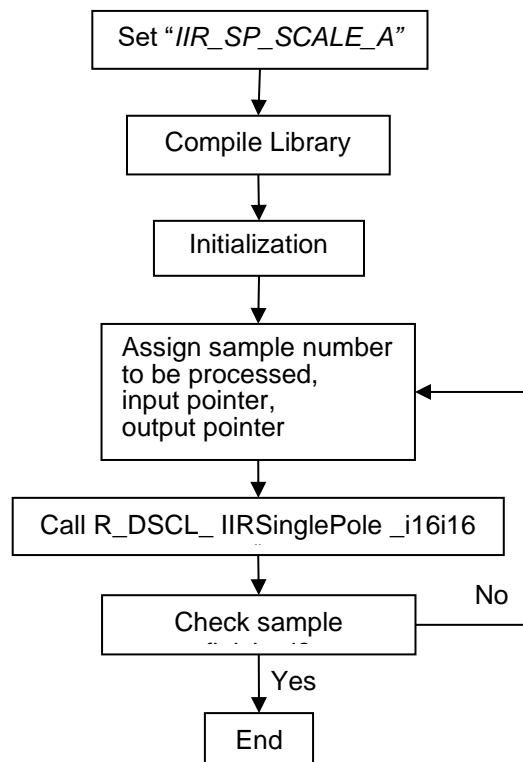
Figure 3-7 Processing flow

Restrictions

The magnitude of the coefficient must be less than 1.0.

4. Sample Workspace on CS+, e²studio

4.1 DSC Library

The following include files and library files are provided.
When using this library alone, include the file indicated in Table 4-1, and link the library file (corresponding to the compiler option) indicated in Table 4-2.

Table 4-1 Include File for the DSC Library

Library	Function	Include File Name
DSC Library	Implements digital filters	"r_dscl_filters.h"

Table 4-2 DSC Library

| Library name | Compiler Option |
	Cpu
R_dscl_filter_rl78.lib	RL78/G14, RL78/G23, RL78/G24 for IIR Single Pole Filter
R_dscl_filter_rl78_S2_NOMDA.lib	RL78/G15

Before using it, copy these files into your local include or library directories.

```
include directory          r_dscl_filters.h, r_dscl_types.h, r_stdint.h

library                    R_dscl_filter_rl78.lib (RL78/G14, RL78/G23,
                           RL78/G24 FAA for Single Pole Filter)
                           R_dscl_filter_rl78_S2_NOMDA.lib (RL78/G15)
```

Example of Usage
The following example shows a program using IIR Single Pole and how to specify the library under CS+, e²studio.

[Source Program]

```c
#include <stdlib.h>
#include "sample_dscl_iirsinglepole.h"

/*******************************************************************************
*
Macro definitions
********************************************************************************
/
#define   INPUT_N      (10)



/*******************************************************************************
*
Typedef definitions
********************************************************************************
/
static int16_t sp_buff_out16[INPUT_N];



/*******************************************************************************
*
Exported global variables (to be accessed by other files)
********************************************************************************
/

/*******************************************************************************
*
Private global variables and functions
********************************************************************************
/
static const int16_t sp_buff_in[INPUT_N] =
{(int16_t)(1.0000000000000000 *0x7FFF)
,(int16_t)(0.0710197609601031 *0x7FFF)
,(int16_t)(0.5590169943749470 *0x7FFF)
,(int16_t)(0.4484011233337100 *0x7FFF)
,(int16_t)((-0.2500000000000000)*0x7FFF)
,(int16_t)(0.5000000000000000 *0x7FFF)
,(int16_t)((-0.5590169943749470)*0x7FFF)
,(int16_t)((-0.1393841289587630)*0x7FFF)
,(int16_t)((-0.2500000000000000)*0x7FFF)
,(int16_t)((-0.8800367553350520)*0x7FFF)
};

/*******************************************************************************
*
* Function Name: sample_dscl_iirsinglepole
* Description  : Sample code to demonstrate single-pole IIR filter
* Arguments    : none
* Return Value : r_dsp_status_t   Function status code
********************************************************************************
/

int16_t sample_dscl_iirsinglepole (void)
{

  int16_t    result;
  vector_t   input;
  vector_t * input_ptr;
```

```
    vector_t    output;
    vector_t * output_ptr;

    int16_t state;
    int16_t coeff;

    /*--------------------------*/
    /* Single-pole IIR filter    */
    /*--------------------------*/
    r_dscl_iirsinglepole_t    sp_handle;
    r_dscl_iirsinglepole_t * sp_handle_ptr;

    /*--------------------------*/
    /* Single-pole IIR filter    */
    /*--------------------------*/
    state              = 0;
     coeff             = (int16_t) ((-0.15) * 0x7FFF);
    sp_handle.options     = R_DSCL_ROUNDING_TRUNC;
    sp_handle.coefs       = &coeff;
    sp_handle.state       = &state;
    sp_handle_ptr        = &sp_handle;

    input.n   = INPUT_N;
    input.data = (void*)(&sp_buff_in[0]);
    input_ptr = &input;

    output_ptr = &output;
     output.data = (void *)sp_buff_out16;
     result      = R_DSCL_IIRSinglePole_i16i16
(sp_handle_ptr,input_ptr,output_ptr);

    return (result);

}
```

[How to specify the library under CS+]

Select [Property] of [CC-RL] in project tree menu. In the dialog box [Property], select tab [Frequently Used Options (for Link)], and specify the library in "Using libraries" and the library path in "Additional library paths".

Figure 4-1 Specifying library (CS+)
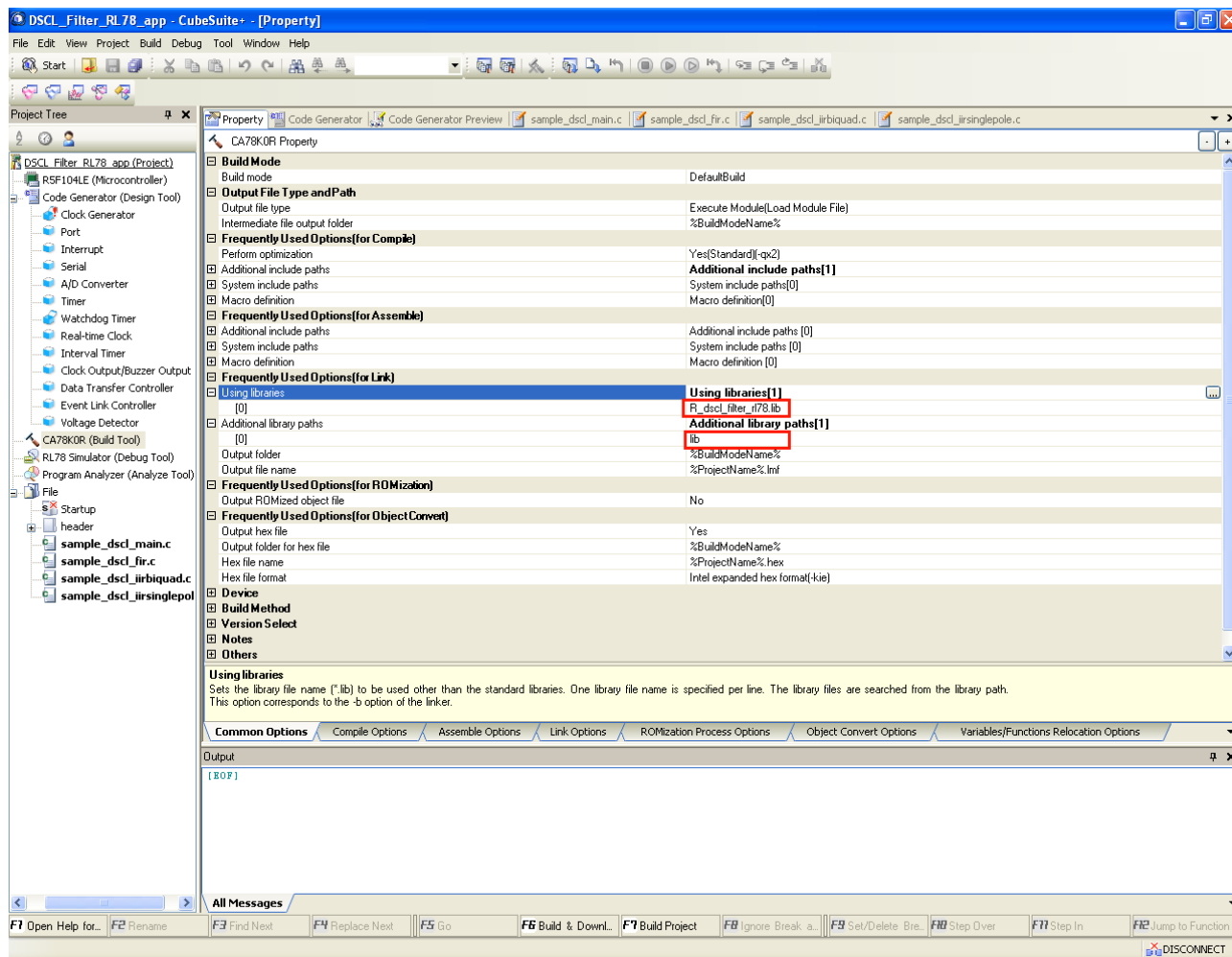
[How to specify the library under e²studio]
Select [Property] from the [Project] menu. Opens the properties window. In the Properties window, select
C/C++ Build> Settings. Select [Linker] → [Input] on the [Tool Settings] tab and specify the library path in
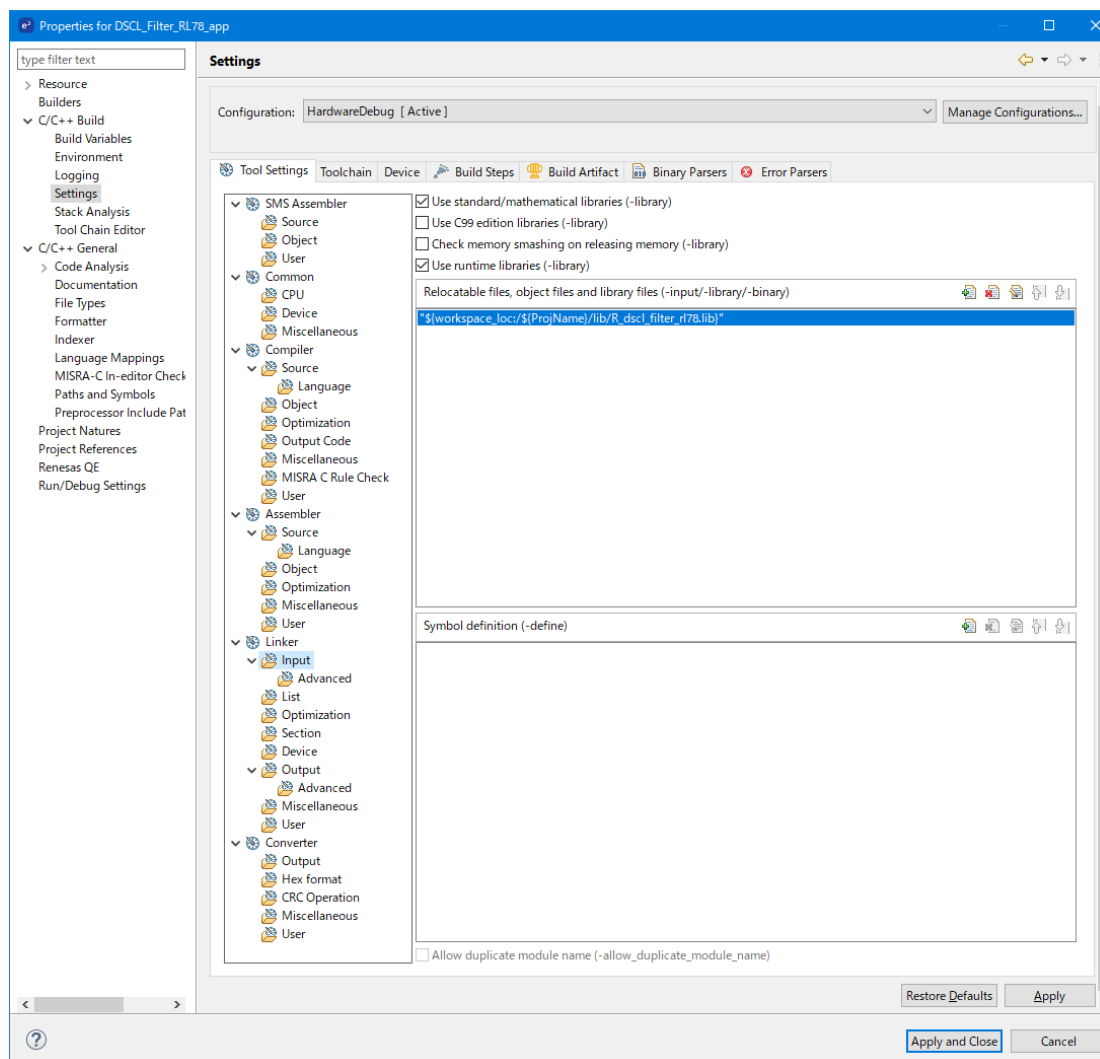"Linkable Relocatable File, Object File, and Library File (-input / -library / -binary)".

Figure 4-2 Specifying library (e²studio)

4.2   Resource Requirements

4.2.1   Code Size and Stack Size

Table 4-3 DSC library for RL78/G14, RL78/G23, RL78/G24 for IIR Single Pole Filter

No.	Kernel Category	Kernel Type	In/Out Format	Function	Options	Code size (DEC)	Total code size (DEC)	Stack size (DEC)	Overall stack size (DEC)
1	Filter	Generic FIR	i16i16	R_DSCL_FIR_StateSize_i16i16	-	13	13	4	4
				R_DSCL_FIR_Init_i16i16	-	111	111	8	8
				R_DSCL_FIR_i16i16	c interface	189	477	4	26
				R_DSCL_FIR_i16i16	nr	137		20	
				R_DSCL_FIR_i16i16	r	151		22	
		IIR-Biquad	i16i16	R_DSCL_IIRBiquad_StateSize_i16i16	-	8	8	2	4
				R_DSCL_IIRBiquad_Init_i16i16	-	109	109	12	4
				R_DSCL_IIRBiquad_i16i16	c interface	174	635	4	34
				R_DSCL_IIRBiquad_i16i16	nr	222		28	
				R_DSCL_IIRBiquad_i16i16	r	239		30	
		Single-pole-IIR	i16i16	R_DSCL_IIRSinglePole_i16i16	c interface	173	488	6	32
				R_DSCL_IIRSinglePole_i16i16	nr	143		22	
				R_DSCL_IIRSinglePole_i16i16	r	172		26	

Notes:

nr = R_DSCL_ROUNDING_TRUNC (or no option)

r = R_DSCL_ROUNDING_NEAREST

Table 4-4 DSC library for RL78/G15

No.	Kernel Category	Kernel Type	In/Out Format	Function	Options	Code size (DEC)	Total code size (DEC)	Stack size (DEC)	Overall stack size (DEC)
1	Filter	Generic FIR	i16i16	R_DSCL_FIR_StateSize_i16i16	-	2	2	4	4
				R_DSCL_FIR_Init_i16i16	-	34	34	6	6
				R_DSCL_FIR_i16i16	c interface	27	351	4	48
				R_DSCL_FIR_i16i16	nr	154		42	
				R_DSCL_FIR_i16i16	r	170		44	
		IIR-Biquad	i16i16	R_DSCL_IIRBiquad_StateSize_i16i16	-	5	5	4	4
				R_DSCL_IIRBiquad_Init_i16i16	-	65	65	10	10
				R_DSCL_IIRBiquad_i16i16	c interface	46	664	8	62
				R_DSCL_IIRBiquad_i16i16	nr	293		50	
				R_DSCL_IIRBiquad_i16i16	r	325		54	
		Single-pole-IIR	i16i16	R_DSCL_IIRSinglePole_i16i16	c interface	110	445	10	54
				R_DSCL_IIRSinglePole_i16i16	nr	153		40	
				R_DSCL_IIRSinglePole_i16i16	r	182		44	

Notes:

nr = R_DSCL_ROUNDING_TRUNC (or no option)

r = R_DSCL_ROUNDING_NEAREST

4.2.2 Cycles and Accuracy

Table 4-5 DSC library for RL78/G14, RL78/G23, RL78/G24 for IIR Single Pole Filter

No.	Filter		Samples	Taps	Options	Cycles	Max Err	Ave Err
1	Generic FIR		200	64	nr	354,215	3.03E-05	1.58E-05
2			200	64	r	354,503	1.53E-05	8.43E-06
3	IIR Single pole	low pass	200	1	nr	8,191	3.02E-04	2.20E-04
4			200	1	r	9,915	4.44E-05	1.99E-05
5		high pass	200	1	nr	8,482	4.20E-05	1.86E-05
6			200	1	r	9,789	4.24E-05	1.48E-05
7	IIR Biquad		200	4	nr	81,235	5.32E-04	4.00E-04
8			200	4	r	82,131	1.66E-04	4.82E-05

Notes:
> nr = R_DSCL_ROUNDING_TRUNC (or no option)
> r = R_DSCL_ROUNDING_NEAREST

Table 4-6 DSC library for RL78/G15

No.	Filter		Samples	Taps	Options	Cycles	Max Err	Ave Err
1	Generic FIR		168	64	nr	18,808,000	3.03E-05	1.58E-05
2			168	64	r	18,812,000	1.53E-05	8.43E-06
3	IIR Single pole	low pass	192	1	nr	547,200	3.02E-04	2.20E-04
4			192	1	r	572,800	4.44E-05	1.99E-05
5		high pass	192	1	nr	558,400	4.20E-05	1.86E-05
6			192	1	r	585,600	4.24E-05	1.48E-05
7	IIR Biquad		132	4	nr	1,025,600	5.32E-04	4.00E-04
8			132	4	r	1,049,600	1.66E-04	4.82E-05

Notes:
> nr = R_DSCL_ROUNDING_TRUNC (or no option)
> r = R_DSCL_ROUNDING_NEAREST

5. Sample Workspace on IAR Embedded Workbench

5.1   DSC Library

The following include files and library files are provided.

If you want to use this library alone, include the files shown in Table 5-1 and link the library files
(corresponding to the compiler options) in Table 5-2.

Table 5-1 Include File for the DSC Library

Library	Function	Include File Name
DSC Library	Implements digital filters	r_dscl_filters.h

Table 5-2 DSC Library

Library name	Compiler option
	Cpu
R_dscl_filter_rl78.a	RL78/G14, RL78/G23, RL78/G24 for IIR Single Pole Filter
R_dscl_filter_rl78_S2_NOMDA.a	RL78/G15

Before using these files, copy them to your local include or library directory.

```
include directory          r_dscl_filters.h, r_dscl_types.h, r_stdint.h

library                    R_dscl_filter_rl78.a (RL78/G14, RL78/G23,
                                     RL78/G24 FAA for Single Pole Filter)
                           R_dscl_filter_rl78_S2_NOMDA.a (RL78/G15)
```

Example of use
An example program using an IIR unipolar filter shows how to specify a library in IAR Embedded Workbench.

[Source Program]

```c
#include <stdlib.h>
#include "sample_dscl_iirsinglepole.h"


/***********************************************************************
*
Macro definitions
************************************************************************
/
#define   INPUT_N      (10)



/***********************************************************************
*
Typedef definitions
************************************************************************
/
static int16_t sp_buff_out16[INPUT_N];



/***********************************************************************
*
Exported global variables (to be accessed by other files)
************************************************************************
/


/***********************************************************************
*
Private global variables and functions
************************************************************************
/
static const int16_t sp_buff_in[INPUT_N] =
{(int16_t)(1.0000000000000000 *0x7FFF)
,(int16_t)(0.0710197609601031 *0x7FFF)
,(int16_t)(0.5590169943749470 *0x7FFF)
,(int16_t)(0.4484011233337100 *0x7FFF)
,(int16_t)((-0.2500000000000000)*0x7FFF)
,(int16_t)(0.5000000000000000 *0x7FFF)
,(int16_t)((-0.5590169943749470)*0x7FFF)
,(int16_t)((-0.1393841289587630)*0x7FFF)
,(int16_t)((-0.2500000000000000)*0x7FFF)
,(int16_t)((-0.8800367553350520)*0x7FFF)
};


/***********************************************************************
*
* Function Name: sample_dscl_iirsinglepole
* Description  : Sample code to demonstrate single-pole IIR filter
* Arguments    : none
* Return Value : r_dsp_status_t   Function status code
************************************************************************
/

int16_t sample_dscl_iirsinglepole (void)
{

  int16_t    result;
  vector_t   input;
  vector_t * input_ptr;
```

```c
    vector_t   output;
    vector_t * output_ptr;

    int16_t state;
    int16_t coeff;

    /*---------------------------*/
    /* Single-pole IIR filter    */
    /*---------------------------*/
    r_dscl_iirsinglepole_t   sp_handle;
    r_dscl_iirsinglepole_t * sp_handle_ptr;

    /*---------------------------*/
    /* Single-pole IIR filter    */
    /*---------------------------*/
    state            = 0;
    coeff            = (int16_t) ((-0.15) * 0x7FFF);
    sp_handle.options      = R_DSCL_ROUNDING_TRUNC;
    sp_handle.coefs        = &coeff;
    sp_handle.state        = &state;
    sp_handle_ptr      = &sp_handle;

    input.n   = INPUT_N;
    input.data = (void*)(&sp_buff_in[0]);
    input_ptr = &input;

    output_ptr = &output;
      output.data = (void *)sp_buff_out16;
     result      = R_DSCL_IIRSinglePole_i16i16
(sp_handle_ptr,input_ptr,output_ptr);

    return (result);
}
```

[How to specify the library under IAR Embedded Workbench]

Select a project option in Workspace to open the Options for Node "Project Name" window. In the Options for Node "Project Name" window, select Static Analysis> Linker and select the "Library" tab. Specify the library in [Additional Library].
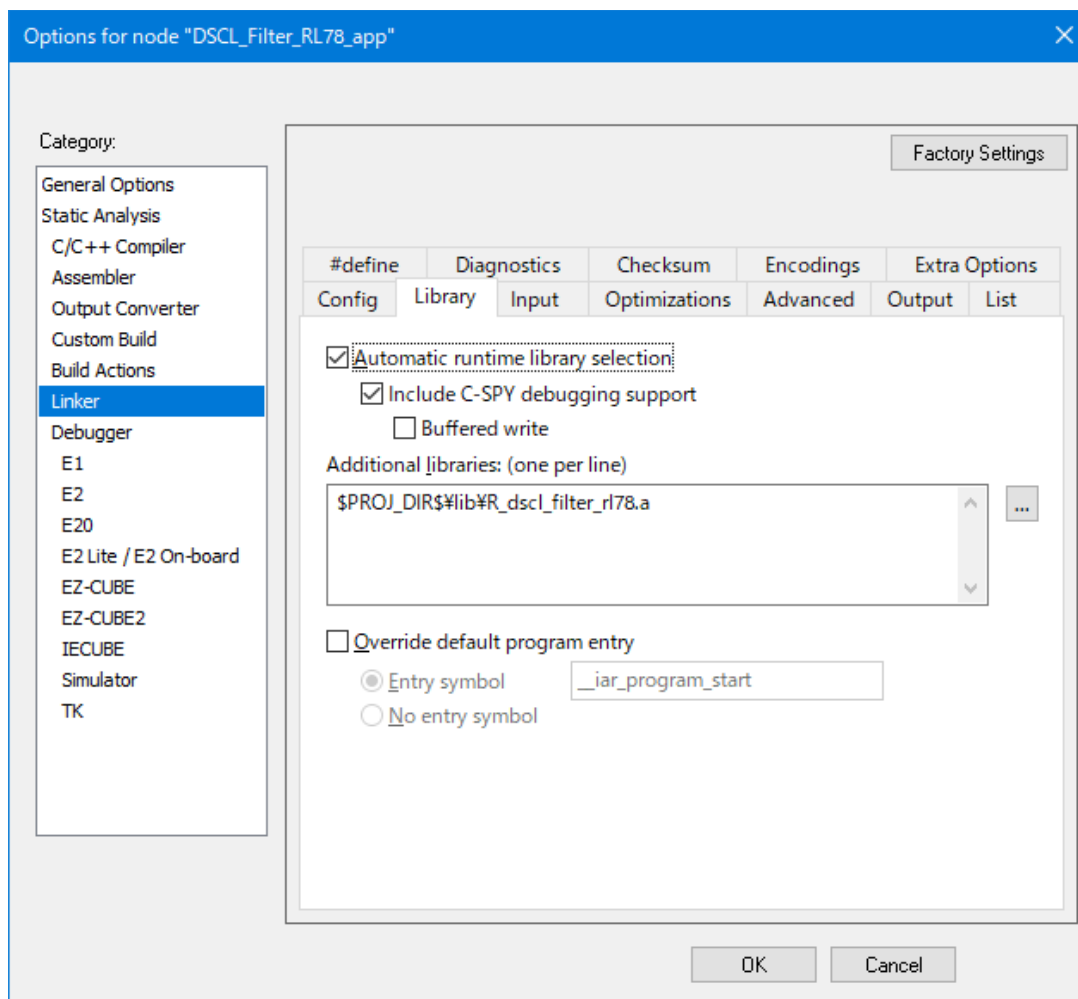

Figure 5-1 Specifying library (IAR Embedded Workbench)

5.2 Resource Requirements

5.2.1 Code Size and Stack Size

Table 5-3 DSC library for RL78/G14, RL78/G23 and RL78/G24 for IIR Single Pole Filter

No.	Kernel Category	Kernel Type	In/Out Format	Function	Options	Code size (DEC)	Total code size (DEC)	Stack size (DEC)	Overall stack size (DEC)
1	Filter	Generic FIR	i16i16	R_DSCL_FIR_StateSize_i16i16	-	2	2	4	4
				R_DSCL_FIR_Init_i16i16	-	92	92	16	16
				R_DSCL_FIR_i16i16	c interface	159	449	8	36
				R_DSCL_FIR_i16i16	nr	138		26	
				R_DSCL_FIR_i16i16	r	152		28	
		IIR-Biquad	i16i16	R_DSCL_IIRBiquad_StateSize_i16i16	-	5	5	4	4
				R_DSCL_IIRBiquad_Init_i16i16	-	126	126	12	12
				R_DSCL_IIRBiquad_i16i16	c interface	155	618	8	46
				R_DSCL_IIRBiquad_i16i16	nr	223		34	
				R_DSCL_IIRBiquad_i16i16	r	240		38	
		Single-pole-IIR	i16i16	R_DSCL_IIRSinglePole_i16i16	c interface	143	466	8	40
				R_DSCL_IIRSinglePole_i16i16	nr	147		28	
				R_DSCL_IIRSinglePole_i16i16	r	176		32	

Notes:

nr = R_DSCL_ROUNDING_TRUNC (or no option)

r = R_DSCL_ROUNDING_NEAREST

Table 5-4 DSC library for RL78/G15

No.	Kernel Category	Kernel Type	In/Out Format	Function	Options	Code size (DEC)	Total code size (DEC)	Stack size (DEC)	Overall stack size (DEC)
1	Filter	Generic FIR	i16i16	R_DSCL_FIR_StateSize_i16i16	-	2	2	2	2
				R_DSCL_FIR_Init_i16i16	-	45	45	8	8
				R_DSCL_FIR_i16i16	c interface	44	367	4	48
				R_DSCL_FIR_i16i16	nr	154		42	
				R_DSCL_FIR_i16i16	r	169		44	
		IIR-Biquad	i16i16	R_DSCL_IIRBiquad_StateSize_i16i16	-	5	5	4	4
				R_DSCL_IIRBiquad_Init_i16i16	-	126	126	12	12
				R_DSCL_IIRBiquad_i16i16	c interface	64	672	4	58
				R_DSCL_IIRBiquad_i16i16	nr	288		50	
				R_DSCL_IIRBiquad_i16i16	r	320		54	
		Single-pole-IIR	i16i16	R_DSCL_IIRSinglePole_i16i16	c interface	143	474	4	48
				R_DSCL_IIRSinglePole_i16i16	nr	151		40	
				R_DSCL_IIRSinglePole_i16i16	r	180		44	

Notes:

nr = R_DSCL_ROUNDING_TRUNC (or no option)

r = R_DSCL_ROUNDING_NEAREST

5.2.2 Cycles and Accuracy

Table 5-5 DSC library for RL78/G14, RL78/G23 and RL78/G24 for IIR Single Pole Filter

No.	Filter		Samples	Taps	Options	Cycles	Max Err	Ave Err
1	Generic FIR		200	64	nr	315,941	3.03E-05	1.58E-05
2			200	64	r	316,253	1.53E-05	8.43E-06
3	IIR Single pole	low pass	200	1	nr	7,945	3.02E-04	2.20E-04
4			200	1	r	9,259	4.44E-05	1.99E-05
5		high pass	200	1	nr	7,947	4.20E-05	1.86E-05
6			200	1	r	9,261	4.24E-05	1.48E-05
7	IIR Biquad		200	4	nr	370,532	5.32E-04	4.00E-04
8			200	4	r	374,952	1.66E-04	4.82E-05

Notes:

     nr = R_DSCL_ROUNDING_TRUNC (or no option)

     r = R_DSCL_ROUNDING_NEAREST

Table 5-6 DSC library for RL78/G15

No.	Filter		Samples	Taps	Options	Cycles	Max Err	Ave Err
1	Generic FIR		131	64	nr	1,458,811	3.03E-05	1.58E-05
2			131	64	r	1,460,198	1.53E-05	8.43E-06
3	IIR Single pole	low pass	128	1	nr	43,818	3.02E-04	2.20E-04
4			128	1	r	45,277	4.44E-05	1.99E-05
5		high pass	128	1	nr	64,035	4.20E-05	1.86E-05
6			128	1	r	65,445	4.24E-05	1.48E-05
7	IIR Biquad		84	4	nr	289,736	5.32E-04	4.00E-04
8			84	4	r	296,654	1.66E-04	4.82E-05

Notes:

     nr = R_DSCL_ROUNDING_TRUNC (or no option)

     r = R_DSCL_ROUNDING_NEAREST

6. Sample Workspace When the LLVM Compiler Is Used in e²studio

6.1 DSC Library

The following include file and library file are provided.
If you want to use this library alone, include the file shown in Table 6-1 and link the library file in Table 6-2 (corresponding to the compiler option).

Table 6-1 Include File for the DSC Library

Library	Function	Include File Name
DSC Library	Implements digital filters	r_dscl_filters.h

Table 6-2 DSC Library

Library name	Compiler option
	Cpu
libR_dscl_filter_rl78.a	RL78/G23
libR_dscl_filter_rl78_S2_NOMDA.a	RL78/G15

Before using these files, copy them to your local include or library directory.

```
include directory       r_dscl_filters.h, r_dscl_types.h, r_stdint.h

library                 libR_dscl_filter_rl78.a (RL78/G23)
                        libR_dscl_filter_rl78_S2_NOMDA.a (RL78/G15)
```

Example of use
The following shows an example of a program that uses an IIR single-pole filter. This example shows how to specify a library in e²studio.

[Source Program]

```c
#include <stdlib.h>
#include "sample_dscl_iirsinglepole.h"


/***********************************************************************
*
Macro definitions
************************************************************************
/
#define   INPUT_N      (10)



/***********************************************************************
*
Typedef definitions
************************************************************************
/
static int16_t sp_buff_out16[INPUT_N];



/***********************************************************************
*
Exported global variables (to be accessed by other files)
************************************************************************
/

/***********************************************************************
*
Private global variables and functions
************************************************************************
/
static const int16_t sp_buff_in[INPUT_N] =
{(int16_t)(1.0000000000000000 *0x7FFF)
,(int16_t)(0.0710197609601031 *0x7FFF)
,(int16_t)(0.5590169943749470 *0x7FFF)
,(int16_t)(0.4484011233337100 *0x7FFF)
,(int16_t)((-0.2500000000000000)*0x7FFF)
,(int16_t)(0.5000000000000000 *0x7FFF)
,(int16_t)((-0.5590169943749470)*0x7FFF)
,(int16_t)((-0.1393841289587630)*0x7FFF)
,(int16_t)((-0.2500000000000000)*0x7FFF)
,(int16_t)((-0.8800367553350520)*0x7FFF)
};


/***********************************************************************
*
* Function Name: sample_dscl_iirsinglepole
* Description  : Sample code to demonstrate single-pole IIR filter
* Arguments    : none
* Return Value : r_dsp_status_t   Function status code
************************************************************************
/

int16_t sample_dscl_iirsinglepole (void)
{

  int16_t    result;
  vector_t   input;
  vector_t * input_ptr;
```

```c
    vector_t   output;
    vector_t * output_ptr;

    int16_t state;
    int16_t coeff;


    /*--------------------------*/
    /* Single-pole IIR filter    */
    /*--------------------------*/
    r_dscl_iirsinglepole_t   sp_handle;
    r_dscl_iirsinglepole_t * sp_handle_ptr;


    /*--------------------------*/
    /* Single-pole IIR filter    */
    /*--------------------------*/
    state           = 0;
     coeff          = (int16_t) ((-0.15) * 0x7FFF);
    sp_handle.options     = R_DSCL_ROUNDING_TRUNC;
    sp_handle.coefs       = &coeff;
    sp_handle.state       = &state;
    sp_handle_ptr       = &sp_handle;

    input.n   = INPUT_N;
    input.data = (void*)(&sp_buff_in[0]);
    input_ptr = &input;

    output_ptr = &output;
     output.data = (void *)sp_buff_out16;
     result     = R_DSCL_IIRSinglePole_i16i16
(sp_handle_ptr,input_ptr,output_ptr);

    return (result);

}
```

[To specify a library in e²studio]

From the [Project] menu, select [Properties] to open the "Properties" window. In the tree view of the "Properties" window, under the [C/C++ Build] node, select [Settings]. Open the [Tool Settings] tab, and then, under the [Linker] node, expand the [Archives] node. Select "R_dscl_filter_rl78" in [Archive (library) file (-l)] and specify the library path in [Archive search directories (-L)].
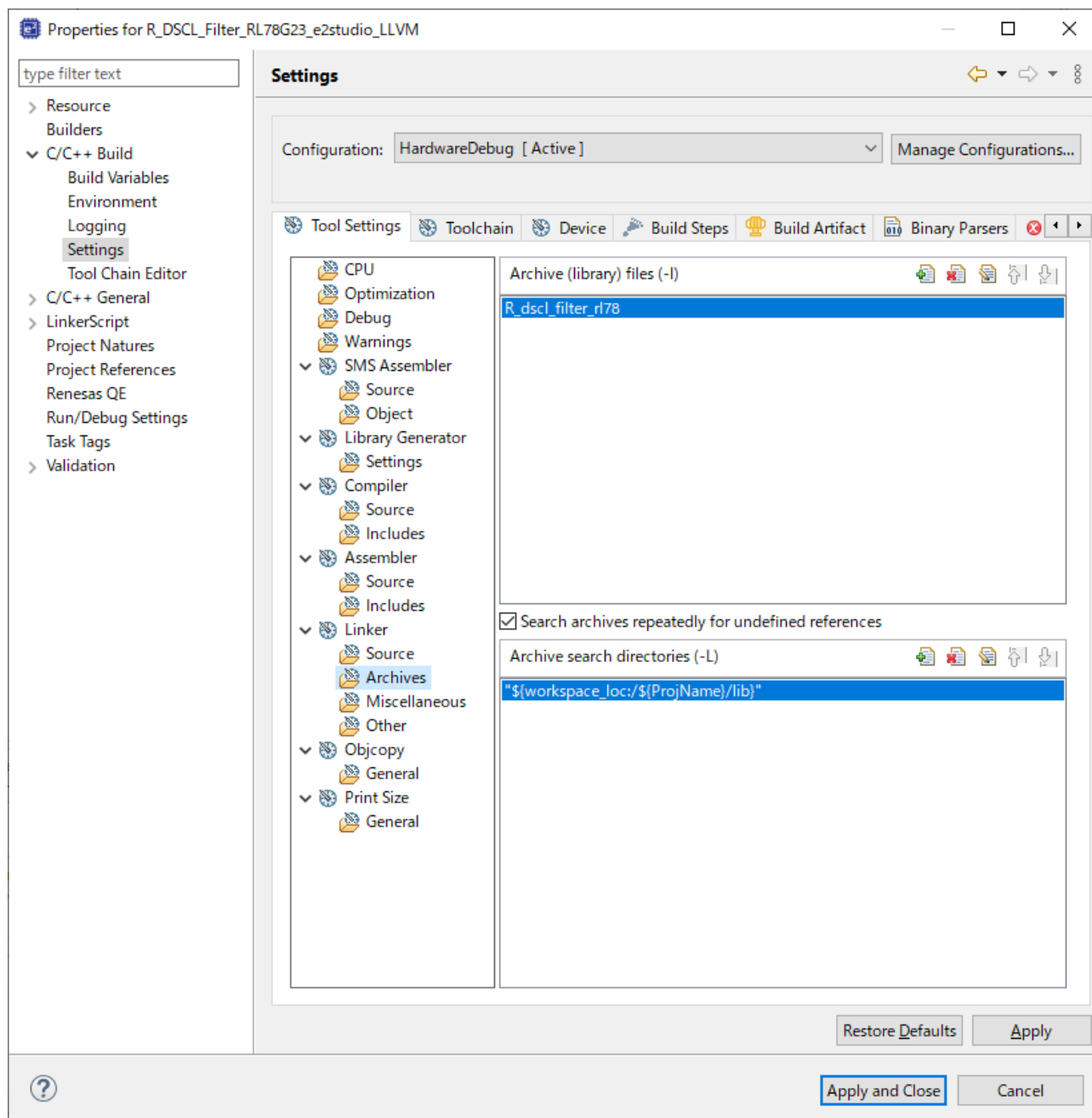
Figure 6-1 Specifying a library (e²studio)

6.2 Resource Requirements

6.2.1 Code Size and Stack Size

Table 6-3 DSC library for RL78/G23

No.	Kernel Category	Kernel Type	In/Out Format	Function	Options	Code size (DEC)	Total code size (DEC)	Stack size (DEC)	Overall stack size (DEC)
1	Filter	Generic FIR	i16i16	R_DCSL_FIR_StateSize_i16i16	-	12	12	4	4
				R_DCSL_FIR_Init_i16i16	-	89	89	16	16
				R_DCSL_FIR_i16i16	c interface	100	388	20	42
				R_DCSL_FIR_i16i16	nr	137		20	
				R_DCSL_FIR_i16i16	r	151		22	
		Biquad IIR	i16i16	R_DCSL_IIRBiquad_StateSize_i16i16	-	12	12	2	2
				R_DCSL_IIRBiquad_Init_i16i16	-	303	303	60	60
				R_DCSL_IIRBiquad_i16i16	c interface	109	570	22	52
				R_DCSL_IIRBiquad_i16i16	nr	222		28	
				R_DCSL_IIRBiquad_i16i16	r	239		30	
		Single-pole IIR	i16i16	R_DCSL_IIRSinglePole_i16i16	c interface	222	537	22	48
				R_DCSL_IIRSinglePole_i16i16	nr	143		22	
				R_DCSL_IIRSinglePole_i16i16	r	172		26	

Notes:

nr = R_DSCL_ROUNDING_TRUNC (or no option)

r = R_DSCL_ROUNDING_NEAREST

Table 6-4 DSC library for RL78/G15

No.	Kernel Category	Kernel Type	In/Out Format	Function	Options	Code size (DEC)	Total code size (DEC)	Stack size (DEC)	Overall stack size (DEC)
1	Filter	Generic FIR	i16i16	R_DCSL_FIR_StateSize_i16i16	-	12	12	4	4
				R_DCSL_FIR_Init_i16i16	-	100	100	20	20
				R_DCSL_FIR_i16i16	c interface	100	422	20	64
				R_DCSL_FIR_i16i16	nr	153		42	
				R_DCSL_FIR_i16i16	r	169		44	
		Biquad IIR	i16i16	R_DCSL_IIRBiquad_StateSize_i16i16	-	12	12	2	2
				R_DCSL_IIRBiquad_Init_i16i16	-	278	278	58	58
				R_DCSL_IIRBiquad_i16i16	c interface	109	717	22	76
				R_DCSL_IIRBiquad_i16i16	nr	288		50	
				R_DCSL_IIRBiquad_i16i16	r	320		54	
		Single-pole IIR	i16i16	R_DCSL_IIRSinglePole_i16i16	c interface	222	553	22	66
				R_DCSL_IIRSinglePole_i16i16	nr	151		40	
				R_DCSL_IIRSinglePole_i16i16	r	180		44	

Notes:

nr = R_DSCL_ROUNDING_TRUNC (or no option)

r = R_DSCL_ROUNDING_NEAREST

6.2.2   Cycles and Accuracy

Table 6-5 DSC library for RL78/G23

No.	Filter		Samples	Taps	Options	Cycles	Max. Error	Avg. Error
1	Generic FIR		200	64	nr	354,215	3.03E-05	1.58E-05
2			200	64	r	354,503	1.53E-05	8.43E-06
3	Single-pole IIR	Low-pass	200	1	nr	8,191	3.02E-04	2.20E-04
4			200	1	r	9,915	4.44E-05	1.99E-05
5		High-pass	200	1	nr	8,482	4.20E-05	1.86E-05
6			200	1	r	9,789	4.24E-05	1.48E-05
7	Biquad IIR		200	4	nr	81,235	5.32E-04	4.00E-04
8			200	4	r	82,131	1.66E-04	4.82E-05

Notes:

nr = R_DSCL_ROUNDING_TRUNC (or no option)

r = R_DSCL_ROUNDING_NEAREST

Table 6-6 DSC library for RL78/G15

No.	Filter		Samples	Taps	Options	Cycles	Max. Error	Avg. Error
1	Generic FIR		155	64	nr	17,352,000	3.03E-05	1.58E-05
2			155	64	r	17,356,000	1.53E-05	8.43E-06
3	Single-pole IIR	Low-pass	176	1	nr	500,800	3.02E-04	2.20E-04
4			176	1	r	524,800	4.44E-05	1.99E-05
5		High-pass	176	1	nr	512,000	4.20E-05	1.86E-05
6			176	1	r	536,000	4.24E-05	1.48E-05
7	Biquad IIR		100	4	nr	780,800	5.32E-04	4.00E-04
8			100	4	r	789,400	1.66E-04	4.82E-05

Notes:

nr = R_DSCL_ROUNDING_TRUNC (or no option)

r = R_DSCL_ROUNDING_NEAREST

7. DSC Library for RL78/G24 FAA

7.1 Sample Workspace on CS+, e²studio

7.1.1 DSC Library

The DSC Library (FIR Filter, IIR Biquad Filter) for FAA is generated by using smart configurator.
For basic operation of the Smart Configurator, refer to the User's Guide for basic Smart Configurator operations.

- RL78 Smart Configurator User's Guide: e² studio (R20AN0579)
- RL78 Smart Configurator User's Guide: CS+ (R20AN0580)

For the build process and debugger of the flexible application accelerator (FAA) contained in RL78/G24, please refer to the following user guides:

- RL78/G24 FAA Tool Guide for CS+ (R01AN7094)
- RL78/G24 FAA Tool Guide for e² studio (R01AN70945)

For details of FAA, refer to the document RL78/G24 User's Manual: hardware (R01UH0961E) Section4.

Before using the FIR filter and IIR biquad filter API, please copy the include files shown in Table 7-1 to your local directory.

Table 7-1 Include files for the DSC library for FAA

Include Directory Name	Include File Name
include	r_dscl_types.h

If you want to use an IIR Single Pole Filter, use the library R_dscl_filter_rl78.lib (RL78/G14, RL78/G23, RL78/G24 for Single-Pole IIR Filter) as described in section **4 Sample Workspace on CS+, e2studio**.

7.1.1.1 Example of use

The following shows an example of a program that uses a FIR filter. This example shows how to generate the DSC library for FAA. If you want to use the DSC library for FAA, generate it by using smart configurator, and need to be compiled and linked, according to **7.1.1.3 How to calculate the Data area size to be used** described later.

[Source Program]

```c
#include "sample_dscl_fir.h"

/***************************************************************************
*
Macro definitions
***************************************************************************
/
#define NUM_SAMPLES         (10)
#define NUM_TAPS            (10)
//#define FRACTION_BITS       (15)
//#define CONVERSION_CONST    ((1<<FRACTION_BITS)-1)



/***************************************************************************
*
Typedef definitions
***************************************************************************
/

/***************************************************************************
*
Exported global variables (to be accessed by other files)
***************************************************************************
/

/***************************************************************************
*
Private global variables and functions
***************************************************************************
/
/* coeffients stored in time-reversed order */
static int16_t myCoeffs[NUM_TAPS] = {
  95,         // h(9) = (int16_t)( 0.0029024*CONVERSION_CONST)
  330,    // h(8) = (int16_t)( 0.0100975*CONVERSION_CONST)
  323,    // h(7) = (int16_t)( 0.0098667*CONVERSION_CONST)
  33,         // h(6) = (int16_t)( 0.0010075*CONVERSION_CONST)
  (-488),    // h(5) = (int16_t)((-0.0149086)*CONVERSION_CONST)
  (-1101),   // h(4) = (int16_t)((-0.0336059)*CONVERSION_CONST)
  (-1605),   // h(3) = (int16_t)((-0.0490032)*CONVERSION_CONST)
  (-1794),   // h(2) = (int16_t)((-0.0547532)*CONVERSION_CONST)
  (-1508),   // h(1) = (int16_t)((-0.0460262)*CONVERSION_CONST)
  (-689),    // h(0) = (int16_t)((-0.0210426)*CONVERSION_CONST)
};

/* state & two blocks of input,
   stored in time-sequential order */
static int16_t inputData[(NUM_TAPS - 1) + (NUM_SAMPLES*2)] = {
    0, // x(-9), start of delayline
    0, // x(-8)
    0, // x(-7)
    0, // x(-6)
    0, // x(-5)
    0, // x(-4)
    0, // x(-3)
    0, // x(-2)
    0, // x(-1)
  32767,     // x(0) = (int16_t)( 1.0000*CONVERSION_CONST) ,start of 1st block
  input
```

```
  1736,       // x(1)  = (int16_t)( 0.0530*CONVERSION_CONST)
  25810,      // x(2)  = (int16_t)( 0.7877*CONVERSION_CONST)
  13368,      // x(3)  = (int16_t)( 0.4080*CONVERSION_CONST)
  10518,      // x(4)  = (int16_t)( 0.3210*CONVERSION_CONST)
  26721,      // x(5)  = (int16_t)( 0.8155*CONVERSION_CONST)
  (-983),     // x(6)  = (int16_t)((-0.0300)*CONVERSION_CONST)
  30521,      // x(7)  = (int16_t)( 0.9202*CONVERSION_CONST)
  0,          // x(8)  = (int16_t)( 0.0000*CONVERSION_CONST)
  19896,      // x(9)  = (int16_t)( 0.6072*CONVERSION_CONST)
  11586,      // x(10) = (int16_t)( 0.3536*CONVERSION_CONST) ,start of 2nd block
 input
  3201,       // x(11) = (int16_t)( 0.0977*CONVERSION_CONST)
  22884,      // x(12) = (int16_t)( 0.6984*CONVERSION_CONST)
  (-7621),    // x(13) = (int16_t)( 0.7025*CONVERSION_CONST)
  23018,      // x(14) = (int16_t)( 0.7025*CONVERSION_CONST)
  (-5314),    // x(15) = (int16_t)((-0.1622)*CONVERSION_CONST)
  10125,      // x(16) = (int16_t)( 0.3090*CONVERSION_CONST)
  6386,       // x(17) = (int16_t)( 0.1949*CONVERSION_CONST)
  (-7067),    // x(18) = (int16_t)((-0.2157)*CONVERSION_CONST)
  15882       // x(19) = (int16_t)( 0.4847*CONVERSION_CONST)
};

/* To store two blocks of output */
static int16_t outputData[NUM_SAMPLES*2];

/*************************************************************************
 *
 * Function Name: sample_dscl_fir
 * Description  : Sample code to demonstrate generic FIR filter
 * Arguments    : none
 * Return Value : r_dsp_status_t   Function status code
 *************************************************************************
/
int16_t sample_dscl_fir (void)
{

    r_dscl_firfilter_t myFilterHandle;
    vector_t           myInput;
    vector_t           myOutput;
    int16_t            myFIRFlags = R_DSCL_STATUS_OK;

    /*-------------- Set up the FIR filter ------------*/
    myFilterHandle.taps  = NUM_TAPS;
    myFilterHandle.options = R_DSCL_ROUNDING_NEAREST;

    /* No need to call StateSize API for FIR, as it always return 0. */
    /* The delayline & input share the same buffer.*/
    myFilterHandle.state = (void *)&inputData[0];        // start of delay line

    /*----- Initialize the coefficients and internal state ------*/
    myFilterHandle.coefs = (void *)myCoeffs;

    /*initialize delay line*/
    myFIRFlags = R_DSCL_FIR_Init_i16i16(&myFilterHandle);

    if(R_DSCL_STATUS_OK != myFIRFlags)
    {
        return myFIRFlags;
```

```
    }

    /*--------------- Set up the input/output ----------------*/
    myInput.n = NUM_SAMPLES;
    myInput.data  = (void *)&inputData[NUM_TAPS - 1]; //start of 1st block
 input
    myOutput.data = (void *)outputData;

    /*----------------- Wait for input data -----------------*/

    /*-------------- Main library function call --------------*/
    /* process 1st input block */
    myFIRFlags = R_DSCL_FIR_i16i16 (&myFilterHandle, &myInput, &myOutput);
    if(R_DSCL_STATUS_OK != myFIRFlags)
    {
        return myFIRFlags;
    }

    /* process 2nd input block */
    /* start of delay line for 2nd block */
    myFilterHandle.state = (void *)&inputData[NUM_SAMPLES];

    /* start of 2nd block input */
    myInput.data  = (void *)&inputData[(NUM_TAPS - 1) + NUM_SAMPLES];

    /* start of 2nd block output */
    myOutput.data = (void *)&outputData[NUM_SAMPLES];

    myFIRFlags = R_DSCL_FIR_i16i16 (&myFilterHandle, &myInput, &myOutput);
    if(R_DSCL_STATUS_OK != myFIRFlags)
    {
        return myFIRFlags;
    }

    /*-------------- Output data are now ready --------------
    * Note: At this point myOutput.n holds the number of output samples
    * generated bythe library,
    * where the data are written to the array pointed to by myOutput.data.
    *------------------------------------------------------------*/
    return myFIRFlags;

}
```

7.1.1.2 How to generate the DSC library for FAA

The code of the DSC library for FAA is generated by smart configurator.
In this section, the data memory containing the data necessary for the FAA to execute the program is called
Data area.
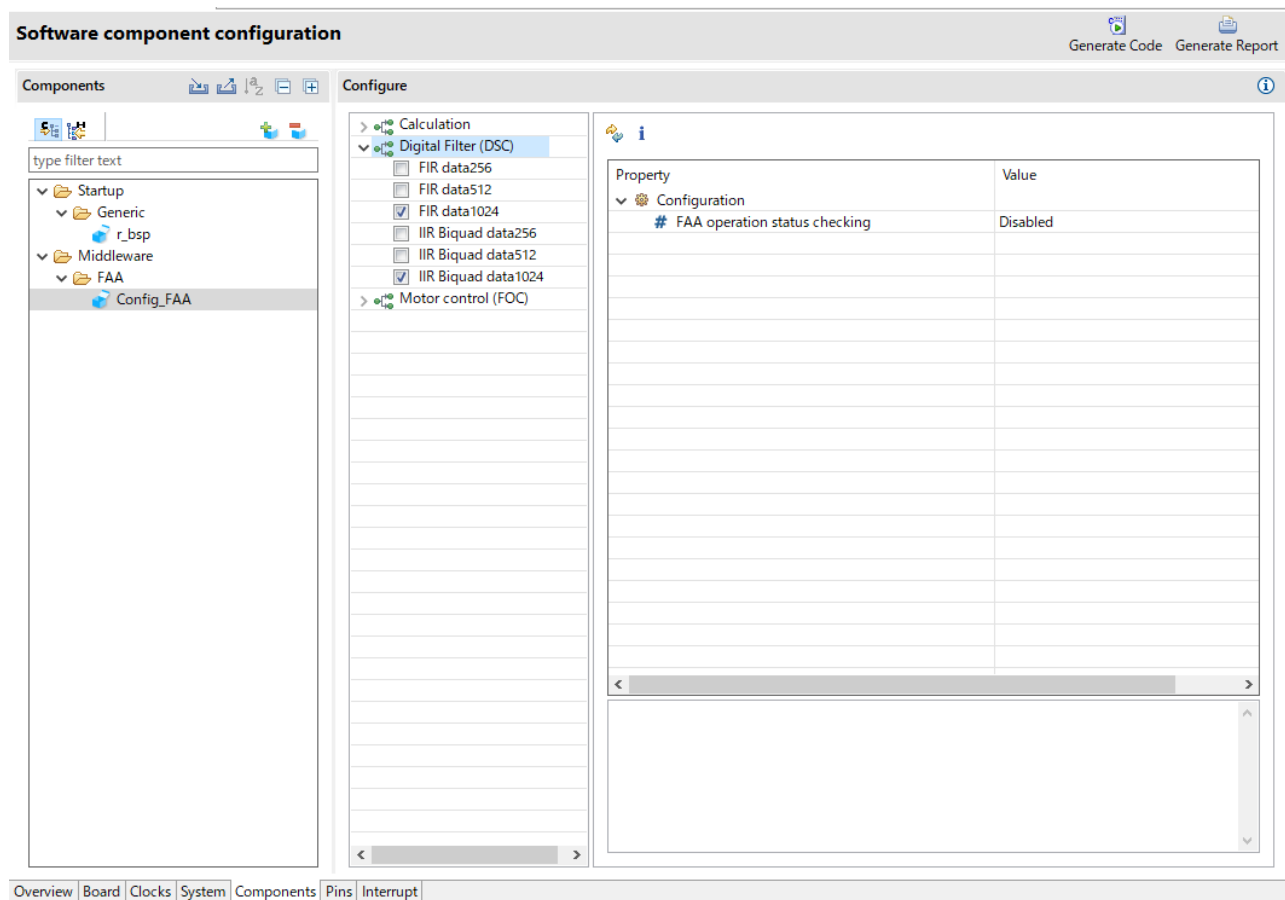
Figure 7-1Smart Configurator View

1. Select digital filters to be used in the Functions of smart configurator.

Note: You can select more than one but select only one required Data area size per digital filter.

2. After selecting a function, set its property.
3. The code is generated in ¥src¥smc_gen¥Config_FAA

Selectable Functions are shown in Table 7-2.

Table 7-2 Selectable Functions

Function	Description
FIR data256	FIR Filter : Data area size 256 bytes
FIR data512	FIR Filter : Data area size 512 bytes
FIR data1024	FIR Filter : Data area size 1024 bytes
IIR Biquad data256	IIR Biquad Filter : Data area size 256 bytes
IIR Biquad data512	IIR Biquad Filter : Data area size 512 bytes
IIR Biquad data1024	IIR Biquad Filter : Data area size 1024 bytes

Property Setting is shown in Table 7-3.

Table 7-3 Property Setting

Property	Description
FAA operation status checking	・Enabled : The operating state of FAA is checked when calling the API, and return an error code if the FAA is in operation by another function. (R_DSCL_ERR_FAA_ALREADY_RUNNING) ・Disabled : The operating state of FAA is not checked.

7.1.1.3 How to calculate the Data area size to be used

If you want to use the DSC library for FAA, in order to determine the Data area size of digital filter to be used, it is necessary to calculate user data size from each filter function API parameter. Since the FAA accesses data in 32 bits, the data size (1 data, 32-bit long word) that can be used for each specified size is below.
 ・When 256 / 512 / 1024 bytes are specified : 64 / 128 / 256 data can be used in the Data area.
Add the library usage data size to the user data size, since part of this data area is used as library local variables and stacks.

Table 7-4 shows how to calculate the data size required in order to determine the Data area size of digital filter to be used.

Table 7-4 How to calculate data size required

Filter	Parameter	Data size required for parameter	Library usage data size	How to calculate data size required
FIR	coefs	taps	9 + 2(stack size)	coefs + state + output + Library usage data size
	state	input→n		
	output	input→n		
IIR Biquad	coefs	stages * 5	12 + 3(stack size)	coefs + state + input + output + Library usage data size
	state	stages * 4		
	input	input→n		
	output	input→n		

Note: For details of parameter, refer to each filter function API parameter item.

7.1.1.4 Error codes added in the DSC library for FAA

The DSC library for FAA adds the error codes shown in Table 7-5.

Table 7-5 Error codes added

Error code	Description
R_DSCL_ERR_NO_MEMORY_AVAILABLE	Data area size is lacking. Output when the input data size is larger than the Data area size allocated by the FAA.
R_DSCL_ERR_FAA_ALREADY_RUNNING	FAA is operating.

7.1.2   Resource Requirements

7.1.2.1   Code Size and Stack Size

Table 7-6 DSC library for RL78/G24 for FAA

No.	Kernel Category	Kernel Type	In/Out Format	Function	Options	Code size (DEC)	Total code size (DEC)	Stack size (DEC)	Overall stack size (DEC)
1	Filter	Generic FIR	i16i16	R_DSCL_FIR_StateSize_i16i16	-	2	2	4	4
				R_DSCL_FIR_Init_i16i16	-	65	65	6	6
				R_DSCL_FIR_i16i16	c interface	119	680	8	46
				R_DSCL_FIR_i16i16	nr	39		10	
				R_DSCL_FIR_i16i16	r	39		10	
				R_Config_FAA_DSCL_FIR_Start	-	316		28	
				R_Config_FAA_DSCL_FIR_Get	-	167		14	
				P_DSCL_FIR	FAA	216	216	8	8
		Biquad IIR	i16i16	R_DSCL_IIRBiquad_StateSize_i16i16	-	5	5	4	4
				R_DSCL_IIRBiquad_Init_i16i16	-	65	65	10	10
				R_DSCL_IIRBiquad_i16i16	c interface	112	953	8	48
				R_DSCL_IIRBiquad_i16i16	nr	39		10	
				R_DSCL_IIRBiquad_i16i16	r	39		10	
				R_Config_FAA_DSCL_IIRBiquad_Start	-	525		30	
				R_Config_FAA_DSCL_IIRBiquad_Get	-	238		14	
				P_DSCL_IIRBiquad	FAA	336	336	12	12

Notes:

   nr = R_DSCL_ROUNDING_TRUNC (or no option)

   r = R_DSCL_ROUNDING_NEAREST

7.1.2.2   Cycles and Accuracy

Table 7-7 DSC library for RL78/G24 FAA

No.	Filter	Samples	Taps	Options	Cycles	Max. Error	Avg. Error
1	Generic FIR	50	64	nr	48,564	3.03E-05	1.58E-05
2		50	64	r	48,567	1.53E-05	8.43E-06
3	Biquad IIR	200	4	nr	15,398	5.32E-04	4.00E-04
4		200	4	r	15,598	1.66E-04	4.82E-05

Notes:

   nr = R_DSCL_ROUNDING_TRUNC (or no option)

   r = R_DSCL_ROUNDING_NEAREST

7.2　Sample Workspace on IAR Embedded Workbench

7.2.1　DSC Library

When using the FIR filter and IIR biquad filter APIs, please generate the FAA DSC library for FAA by using the Smart Configurator as shown below.

For basic operations of the Smart Configurator, please refer to the following user guide:

- RL78 Smart Configurator User Guide: IAR (R20AN0581)

For detailed information about FAA, please refer to Chapter 4 of the following user's manual:
- RL78/G24 User's Manual Hardware (R01UH0961)

Before using the FIR filter and IIR biquad filter API, please copy the include files shown in Table 7-8 to your local directory.

Table 7-8 Include files for the DSC library for FAA

Include Directory Name	Include File Name
include	r_dscl_types.h

The FAA DSC library does not support the IIR single-pole filter. Therefore, when using the IIR single-pole filter, please use the library R_dscl_filter_rl78.a (for RL78/G14, RL78/G23, RL78/G24 for IIR single-pole filter) described in section **5 Sample Workspace on IAR Embedded Workbench**.

7.2.1.1　Example of using FIR filter

The following shows an example of a program that uses a FIR filter. This example shows how to generate the DSC library for FAA. If you want to use the DSC library for FAA, generate it by using smart configurator, and need to be compiled and linked, according to **7.2.1.2 How to generate the DSC library for FAA**.

[Source Program]

```c
#include "sample_dscl_fir.h"

/***************************************************************************
*
Macro definitions
***************************************************************************
/
#define NUM_SAMPLES        (10)
#define NUM_TAPS           (10)
//#define FRACTION_BITS      (15)
//#define CONVERSION_CONST   ((1<<FRACTION_BITS)-1)



/***************************************************************************
*
Typedef definitions
***************************************************************************
/

/***************************************************************************
*
Exported global variables (to be accessed by other files)
***************************************************************************
/

/***************************************************************************
*
Private global variables and functions
***************************************************************************
/
/* coeffients stored in time-reversed order */
static int16_t myCoeffs[NUM_TAPS] = {
  95,      // h(9) = (int16_t)( 0.0029024*CONVERSION_CONST)
  330,    // h(8) = (int16_t)( 0.0100975*CONVERSION_CONST)
  323,    // h(7) = (int16_t)( 0.0098667*CONVERSION_CONST)
  33,      // h(6) = (int16_t)( 0.0010075*CONVERSION_CONST)
  (-488),    // h(5) = (int16_t)((-0.0149086)*CONVERSION_CONST)
  (-1101),   // h(4) = (int16_t)((-0.0336059)*CONVERSION_CONST)
  (-1605),   // h(3) = (int16_t)((-0.0490032)*CONVERSION_CONST)
  (-1794),   // h(2) = (int16_t)((-0.0547532)*CONVERSION_CONST)
  (-1508),   // h(1) = (int16_t)((-0.0460262)*CONVERSION_CONST)
  (-689),    // h(0) = (int16_t)((-0.0210426)*CONVERSION_CONST)
};

/* state & two blocks of input,
   stored in time-sequential order */
static int16_t inputData[(NUM_TAPS - 1) + (NUM_SAMPLES*2)] = {
    0, // x(-9), start of delayline
    0, // x(-8)
    0, // x(-7)
    0, // x(-6)
    0, // x(-5)
    0, // x(-4)
    0, // x(-3)
    0, // x(-2)
    0, // x(-1)
  32767,     // x(0) = (int16_t)( 1.0000*CONVERSION_CONST) ,start of 1st block
  input
```

```
    1736,        // x(1) = (int16_t)( 0.0530*CONVERSION_CONST)
    25810,       // x(2) = (int16_t)( 0.7877*CONVERSION_CONST)
    13368,       // x(3) = (int16_t)( 0.4080*CONVERSION_CONST)
    10518,       // x(4) = (int16_t)( 0.3210*CONVERSION_CONST)
    26721,       // x(5) = (int16_t)( 0.8155*CONVERSION_CONST)
    (-983),      // x(6) = (int16_t)((-0.0300)*CONVERSION_CONST)
    30521,       // x(7) = (int16_t)( 0.9202*CONVERSION_CONST)
    0,           // x(8) = (int16_t)( 0.0000*CONVERSION_CONST)
    19896,       // x(9) = (int16_t)( 0.6072*CONVERSION_CONST)
    11586,       // x(10) = (int16_t)( 0.3536*CONVERSION_CONST) ,start of 2nd block
 input
    3201,        // x(11) = (int16_t)( 0.0977*CONVERSION_CONST)
    22884,       // x(12) = (int16_t)( 0.6984*CONVERSION_CONST)
    (-7621),     // x(13) = (int16_t)( 0.7025*CONVERSION_CONST)
    23018,       // x(14) = (int16_t)( 0.7025*CONVERSION_CONST)
    (-5314),     // x(15) = (int16_t)((-0.1622)*CONVERSION_CONST)
    10125,       // x(16) = (int16_t)( 0.3090*CONVERSION_CONST)
    6386,        // x(17) = (int16_t)( 0.1949*CONVERSION_CONST)
    (-7067),     // x(18) = (int16_t)((-0.2157)*CONVERSION_CONST)
    15882        // x(19) = (int16_t)( 0.4847*CONVERSION_CONST)
};

/* To store two blocks of output */
static int16_t outputData[NUM_SAMPLES*2];

/****************************************************************************
*
* Function Name: sample_dscl_fir
* Description  : Sample code to demonstrate generic FIR filter
* Arguments    : none
* Return Value : r_dsp_status_t   Function status code
*****************************************************************************
/
int16_t sample_dscl_fir (void)
{

    r_dscl_firfilter_t myFilterHandle;
    vector_t           myInput;
    vector_t           myOutput;
    int16_t            myFIRFlags = R_DSCL_STATUS_OK;

    /*-------------  Set up the FIR filter ------------*/
    myFilterHandle.taps  = NUM_TAPS;
    myFilterHandle.options = R_DSCL_ROUNDING_NEAREST;

    /* No need to call StateSize API for FIR, as it always return 0. */
    /* The delayline & input share the same buffer.*/
    myFilterHandle.state = (void *)&inputData[0];       // start of delay line

    /*----- Initialize the coefficients and internal state ------*/
    myFilterHandle.coefs = (void *)myCoeffs;

    /*initialize delay line*/
    myFIRFlags = R_DSCL_FIR_Init_i16i16(&myFilterHandle);

    if(R_DSCL_STATUS_OK != myFIRFlags)
    {
        return myFIRFlags;
```

```c
    }

    /*--------------- Set up the input/output ---------------*/
    myInput.n = NUM_SAMPLES;
    myInput.data  = (void *)&inputData[NUM_TAPS - 1]; //start of 1st block
 input
    myOutput.data = (void *)outputData;

    /*----------------- Wait for input data -----------------*/

    /*-------------- Main library function call --------------*/
    /* process 1st input block */
    myFIRFlags = R_DSCL_FIR_i16i16 (&myFilterHandle, &myInput, &myOutput);
    if(R_DSCL_STATUS_OK != myFIRFlags)
    {
        return myFIRFlags;
    }

    /* process 2nd input block */
    /* start of delay line for 2nd block */
    myFilterHandle.state = (void *)&inputData[NUM_SAMPLES];

    /* start of 2nd block input */
    myInput.data  = (void *)&inputData[(NUM_TAPS - 1) + NUM_SAMPLES];

    /* start of 2nd block output */
    myOutput.data = (void *)&outputData[NUM_SAMPLES];

    myFIRFlags = R_DSCL_FIR_i16i16 (&myFilterHandle, &myInput, &myOutput);
    if(R_DSCL_STATUS_OK != myFIRFlags)
    {
        return myFIRFlags;
    }

    /*-------------- Output data are now ready --------------
    * Note: At this point myOutput.n holds the number of output samples
    * generated bythe library,
    * where the data are written to the array pointed to by myOutput.data.
    *-------------------------------------------------------*/
    return myFIRFlags;

}
```

7.2.1.2 How to generate the DSC library for FAA

1. Starting the Smart Configurator

Select [Smart Configurator for RL78 Vx.x.x] of [Renesas Electronics Smart Configurator] from the Windows start menu. The main window of the Smart Configurator will be starting.

Note: Please replace Vx.x.x with the user's version.
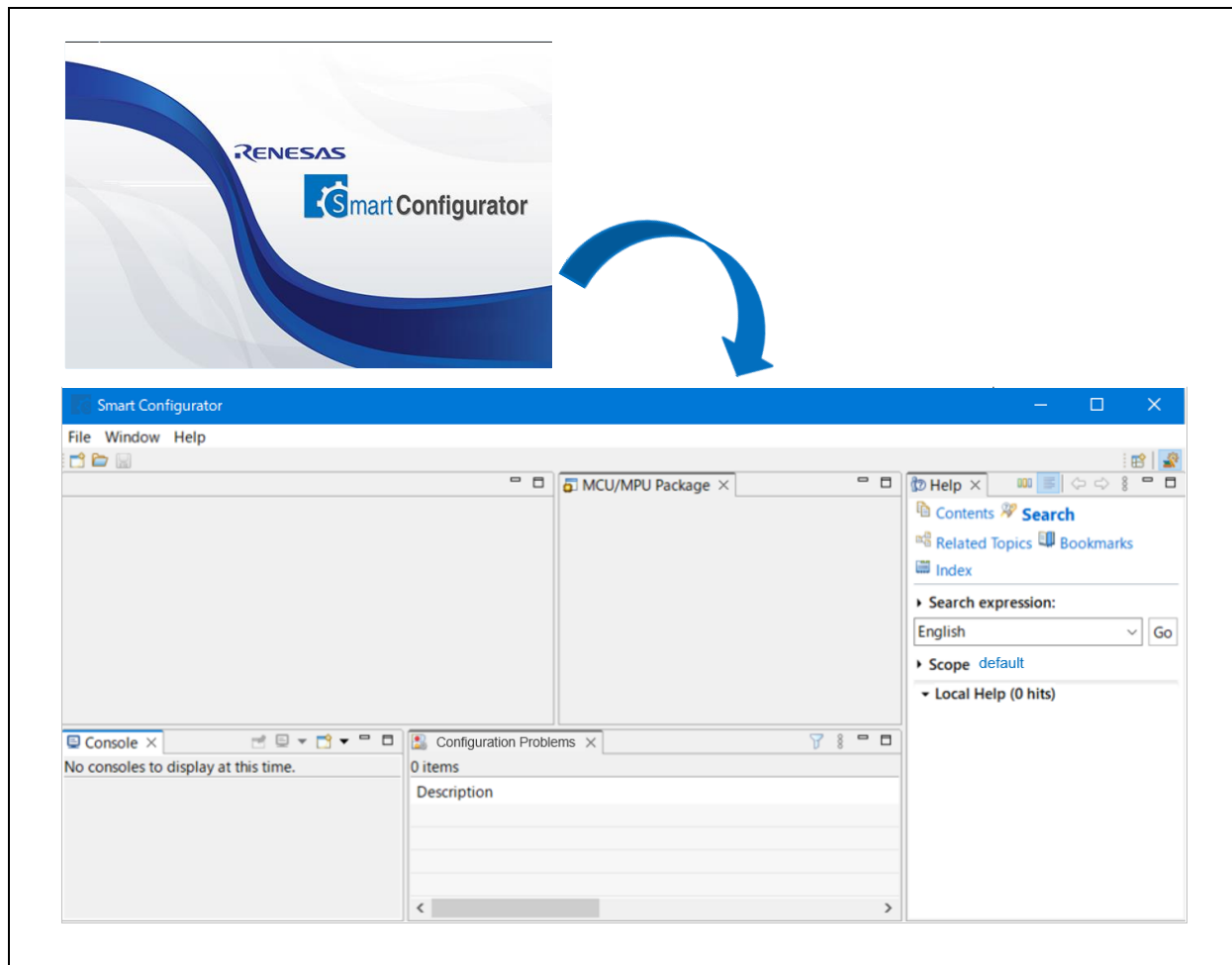
Figure 7-2 Starting of Smart Configurator

2. Creating a New Smart Configurator Configuration File

On the main window, click the [New Configuration File] button to display the [New Smart Configuration File] dialog box.

(1). In [Platform:], panel, select the device.
(2). In [Toolchain:], select [IAR RL78 Toolchain].
(3). In [File name:], enter the file name.
(4). Confirm [Location:], To change the location, please click [Browse] and select the save destination.

Note: The *.eww, *.ewp, *.ewd, main.c and buildinfo.ipcf files will be generated to this location after clicking "Generate Code" button.

(5). Click [Finish] to create the configuration file.
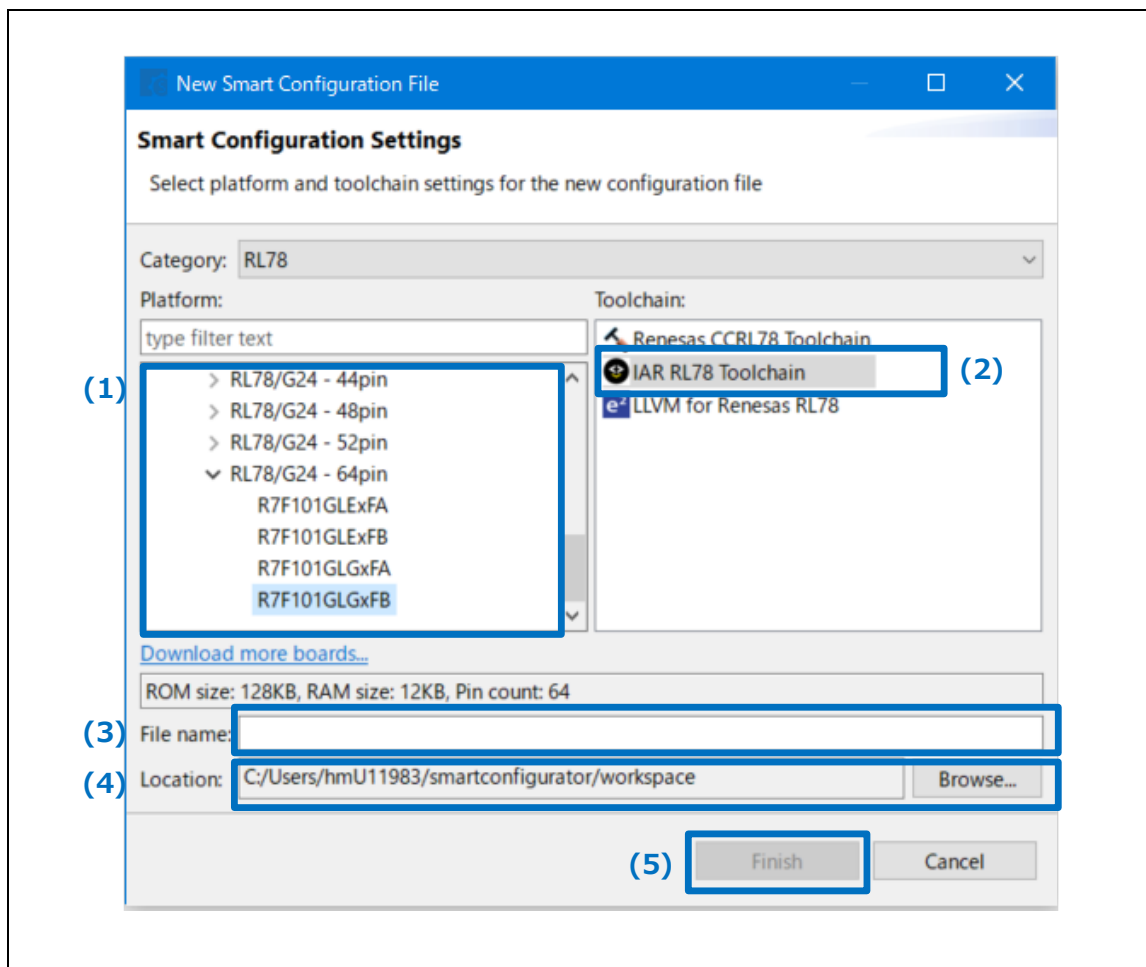
7-3 Create a Configuration File

(6). Add driver component, configure the setting, generate code, and save the project.

Note: The *.eww, *.ewp, *.ewd and main.c files will be generated only for the first-time code generation, while the buildinfo.ipcf file will be generated always for each code generation.

3.   Adding FAA Components

Select the "Components" page of the "Smart Configurator View" and click the "Add component" button. Next, add the "Flexible Application Accelerator" component from the "Software Component Selection" page.
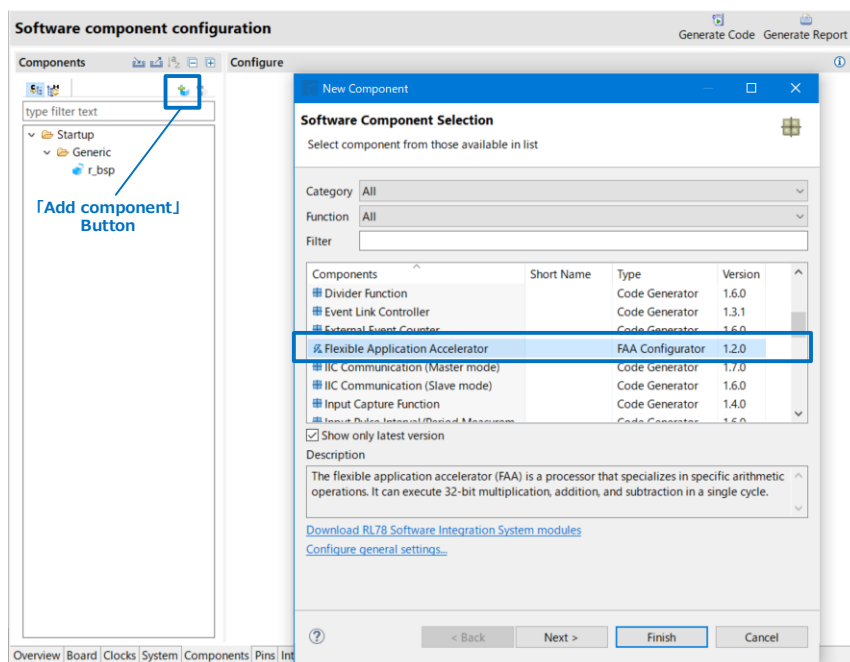
7-4 Adding FAA Components

4.   Downloading FAA Modules

Click on "Please download FAA data" displayed on the screen to see the FAA modules available for download. Select "Filter Library" to download.
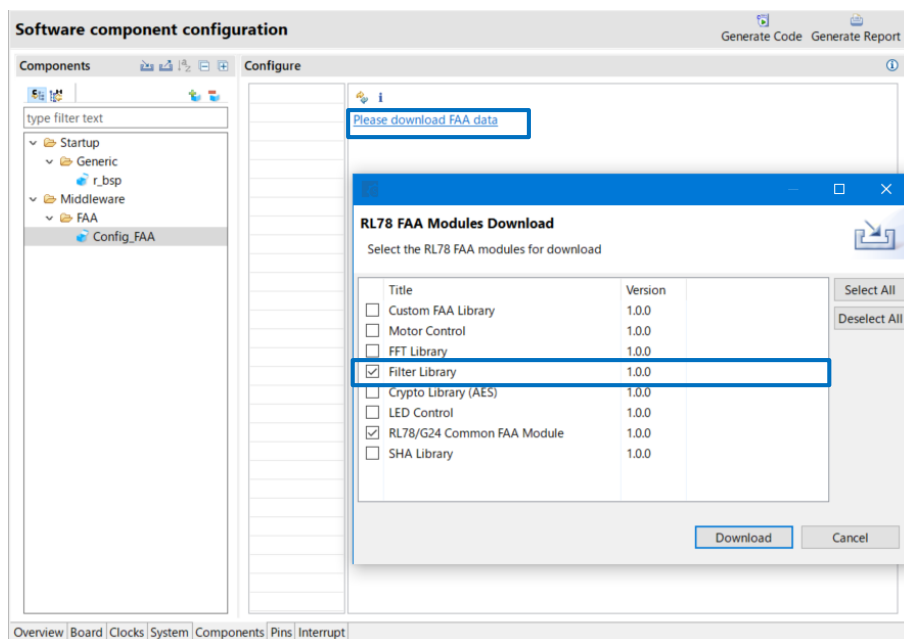
7-5 Downloading FAA Modules

5.   FAA Module Configuration

Select the "Digital Filter" module from the list of downloaded FAA modules to display the configuration screen. Configure the configuration settings according to the user environment.
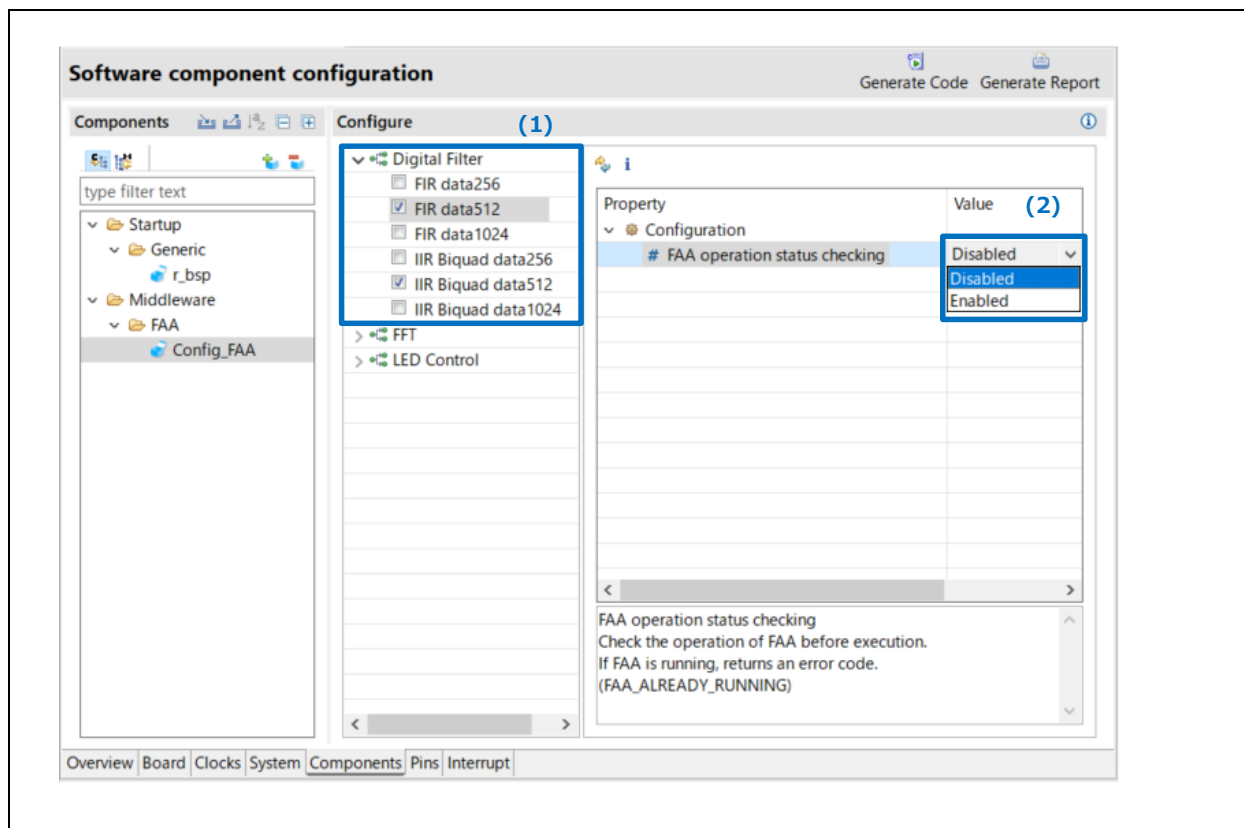
Figure 7-6 FAA Module Configuration

(1).   Select digital filters to be used in the Functions of smart configurator.

Note: You can select more than one but select only one required Data area size per digital filter.

Selectable Functions are shown in Table 7-9.

Table 7-9 Selectable Functions

Function	Description
FIR data256	FIR Filter : Data area size 256 bytes
FIR data512	FIR Filter : Data area size 512 bytes
FIR data1024	FIR Filter : Data area size 1024 bytes
IIR Biquad data256	IIR Biquad Filter : Data area size 256 bytes
IIR Biquad data512	IIR Biquad Filter : Data area size 512 bytes
IIR Biquad data1024	IIR Biquad Filter : Data area size 1024 bytes

(2).   You can configure properties to check the operational status of the FAA.

The properties to be configured are as follows Table 7-10.

Table 7-10 Properties for Checking FAA Operational Status

Property	Value	Description
FAA operation status checking	Enabled	The operating state of FAA is checked when calling the API and return an error code if the FAA is in operation by another function. (R_DSCL_ERR_FAA_ALREADY_RUNNING)
	Disabled	The operating state of FAA is not checked.

6.   Code Generation

Output a source file for the configured details by clicking on the "Generate Code" button in the Smart Configurator view.
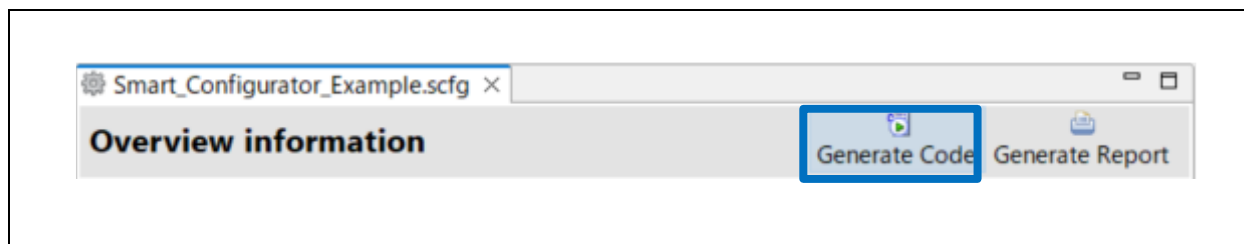
Figure 7-7 Generating a Source File

The Smart Configurator generates a source file in ¥<ProjectDir>¥src¥smc_gen and IAR related files in save location (refer to **2 Creating a New Smart Configurator Configuration File**)

7.   Loading in IAR Embedded Workbench

When IAR environment is selected for the compiler to be used, Smart Configurator outputs the related files (*.eww / *.ewp / *.ewd / main.c) together with the source file. It is not necessary for the user to create project files in IAR Embedded Workbench.

The usage procedure is as follows.

    (1). Select "Open Workspace…" from the "File" menu of IAR Embedded Workbench.

    (2). In the "Open Workspace" dialog box, browse to the folder where the project file is saved, select the project file (.eww), and click the "Open" button.
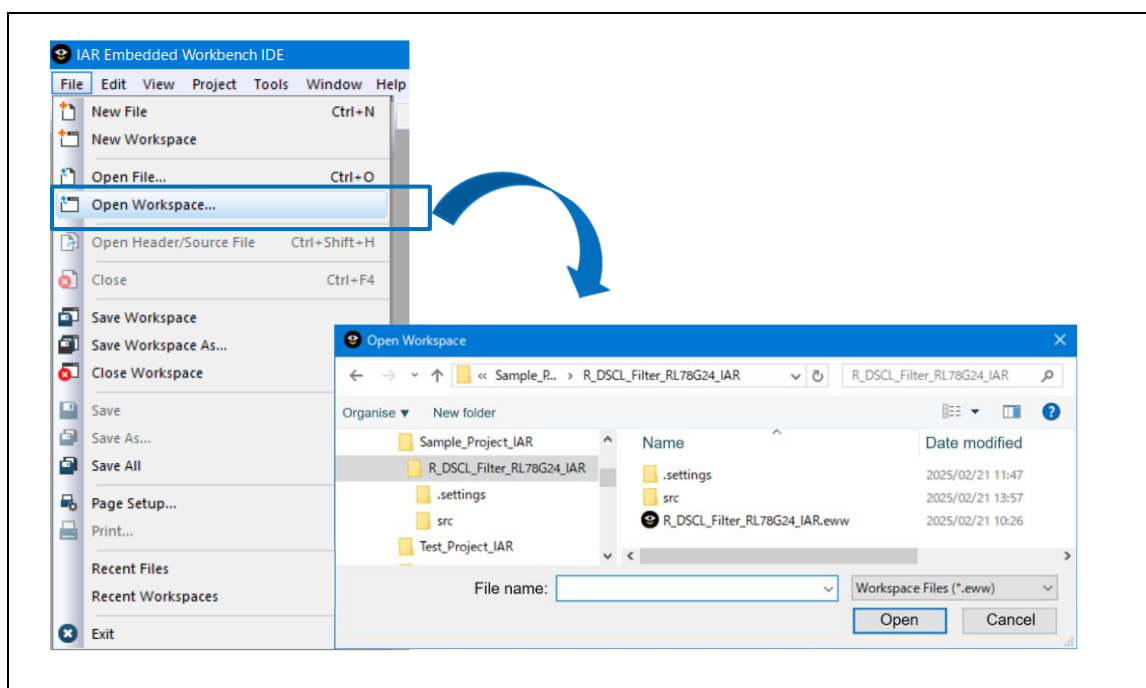
Figure 7-8 Load a *.eww File

(3). The source file output by the Smart Configurator is added to the IAR C project workspace.
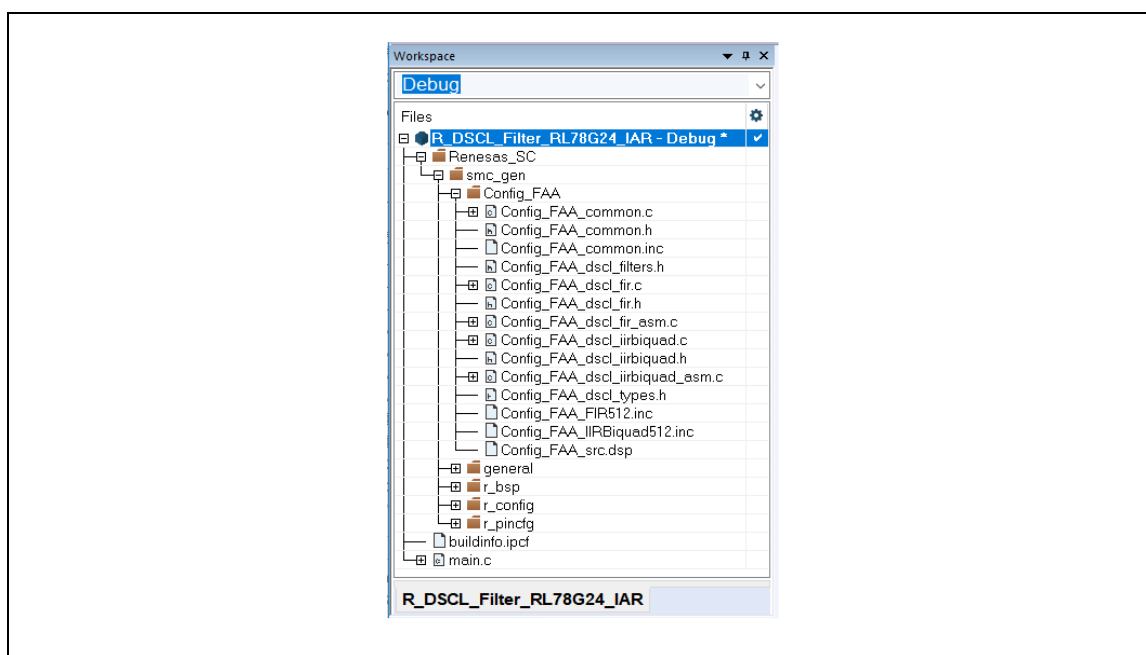
Figure 7-9 New Files Added to IAR Workspace

(4). Select "Options…" from the "Project" menu of IAR Embedded Workbench.

(5). In the "Options for node "Project Name" dialog box, change the target device to match with the target device selected when creating Smart Configurator's configuration file.
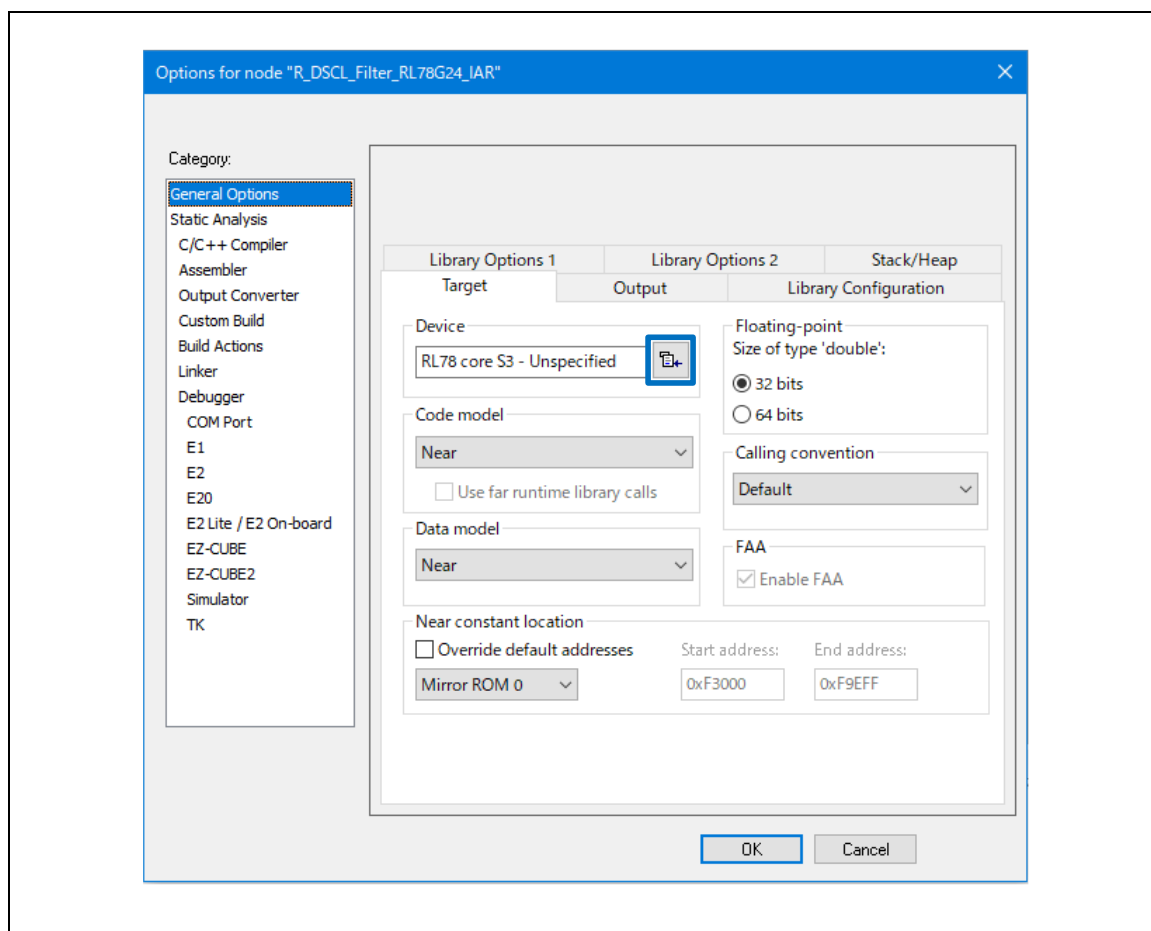

Figure 7-10 Change the target device

8. Adding the DSC Library Include Files for FAA

Before using the library, please copy the include files shown in Table 7-11 to your local directory.

Table 7-11 Include files for the DSC library for FAA

Include Directory Name	Include File Name
include	r_dscl_types.h

Please follow the steps below to add the include files that were copied to the project:

(1). From the "Project" menu in IAR Embedded Workbench, select "Options."

(2).   In the "[Node "Project Name" Options]" dialog box, click the "…" button next to "Additional include directories: (one per line)" under the "Preprocessor" option in the "C/C++ Compiler" tab.

(3). In the "Edit Include Directories" dialog box, add the path to the local directory to "<Click to add>".
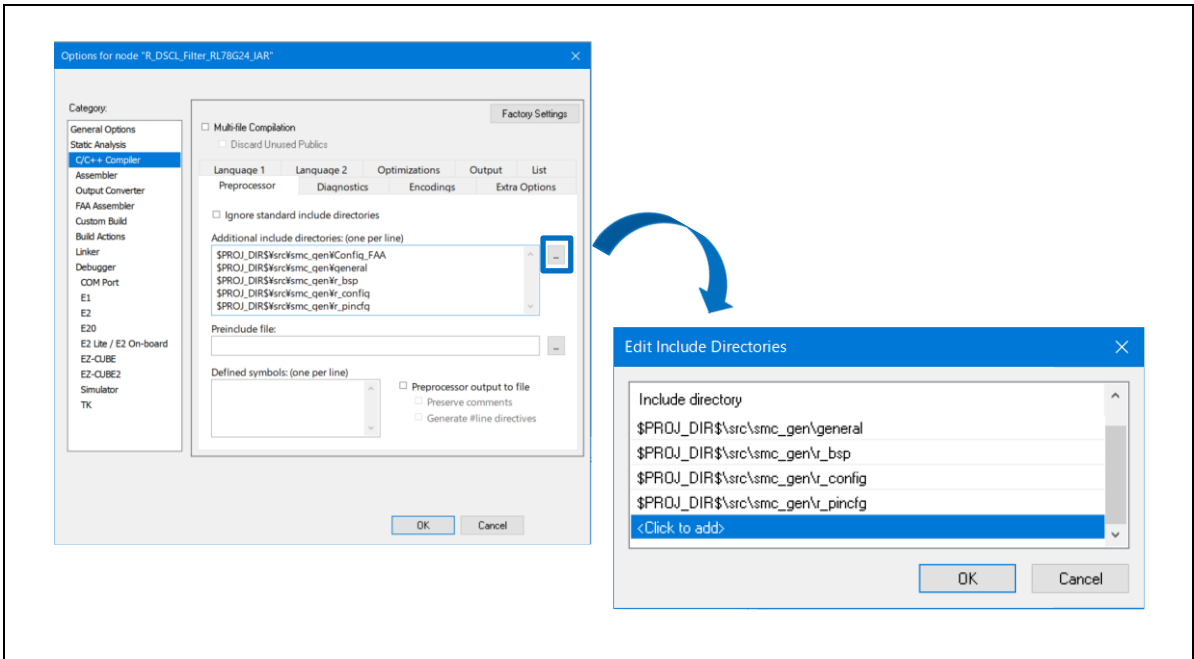
Figure 7-11 Adding the DSC Library Include Files for FAA

7.2.1.3 How to calculate the Data area size to be used

For details, please refer to section **7.1.1.3 How to calculate the Data area size to be used**.

7.2.1.4 Error codes added in the DSC library for FAA

For details, please refer to section **7.1.1.4 Error codes added in the DSC library for FAA**.

7.2.2 Resource Requirements

7.2.2.1 Code Size and Stack Size

Table 7-12 DSC library for RL78/G24 for FAA (IAR (RL78/G24 FAA))

No.	Kernel Category	Kernel Type	In/Out Format	Function	Options	Code size (DEC)	Total code size (DEC)	Stack size (DEC)	Overall stack size (DEC)
1	Filter	Generic FIR	i16i16	R_DSCL_FIR_StateSize_i16i16	-	2	2	4	4
				R_DSCL_FIR_Init_i16i16	-	41	41	8	8
				R_DSCL_FIR_i16i16	c interface	44	716	6	36
				R_DSCL_FIR_i16i16	nr	43		12	
				R_DSCL_FIR_i16i16	r	43		12	
				R_Config_FAA_DSCL_FIR_Start	-	374		18	
				R_Config_FAA_DSCL_FIR_Get	-	212		14	
				P_DSCL_FIR	FAA	216	216	14	14
		Biquad IIR	i16i16	R_DSCL_IIRBiquad_StateSize_i16i16	-	5	5	4	4
				R_DSCL_IIRBiquad_Init_i16i16	-	126	126	12	12
				R_DSCL_IIRBiquad_i16i16	c interface	64	1010	8	38
				R_DSCL_IIRBiquad_i16i16	nr	43		10	
				R_DSCL_IIRBiquad_i16i16	r	43		10	
				R_Config_FAA_DSCL_IIRBiquad_Start	-	560		20	
				R_Config_FAA_DSCL_IIRBiquad_Get	-	300		16	
				P_DSCL_IIRBiquad	FAA	336	336	16	16

Notes:

nr = R_DSCL_ROUNDING_TRUNC (or no option)

r = R_DSCL_ROUNDING_NEAREST

7.2.2.2 Cycles and Accuracy

Table 7-13 DSC library for RL78/G24 for FAA (IAR (RL78/G24 FAA))

No.	Filter	Samples	Taps	Options	Cycles	Max. Error	Avg. Error
1	Generic FIR	50	64	nr	49,776	3.03E-05	1.58E-05
2		50	64	r	49,776	1.53E-05	8.43E-06
3	Biquad IIR	200	4	nr	16,296	5.32E-04	4.00E-04
4		200	4	r	16,488	1.66E-04	4.82E-05

Notes:

nr = R_DSCL_ROUNDING_TRUNC (or no option)

r = R_DSCL_ROUNDING_NEAREST

Revision History

Rev.	Date	Description	
		Page	**Summary**
1.00	May 7, 2012	-	First Edition
1.01	March 6, 2015	-	Second Edition
2.00	Apr.13,21	3, 26, 28	Changed Tool from "CubeSuite+, CA78K0R" to "CS+, CC-RL"
		26	Changed the include file to match the changes in the sample code.
2.01	Sep.30,21	3, 26, 29	Added "e²studio" and "IAR Embedded Workbench" to the tools.
		32	Added chapter "5. Sample workspace on IAR Embedded Workbench"
2.02	June 27, 2022	3	Added the "LLVM" compiler and the "e²studio" tool.
		37	Added chapter "6. Sample Workspace When the LLVM Compiler Is Used in e²studio".
2.03	Sep.26,22	-	Added device RL78/G15
2.04	Mar.1,23	-	Added device RL78/G24 FAA
2.05	Mar.19,25	-	Added device RL78/G24 FAA for IAR In Table 7-6, "R_DSCL_Biquad" was changed to "P_DSCL_Biquad" In Table 7-7, the sample size for No. 3 and No. 4 was changed from "50" to "200".

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

    A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.

2. Processing at power-on

    The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.

3. Input of signal during power-off state

    Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.

4. Handling of unused pins

    Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.

5. Clock signals

    After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.

6. Voltage application waveform at input pin

    Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).

7. Prohibition of access to reserved addresses

    Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.

8. Differences between products

    Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.

2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.

3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.

4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.

5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

    "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.

    "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

    Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user's manual or other Renesas Electronics document.

6. When using Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.

7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.

8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.

9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.

10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.

11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.

12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 November 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics
Corporation. All trademarks and registered trademarks are the property
of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date
version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.