RL78ファミリ

RL78用デジタル信号コントローラライブラリ - フィルタ

要旨

このアプリケーションノートには、Renesas RL78 用デジタル信号コントローラライブラリ（DSCL）の関数ライブラリの仕様、フィルタアルゴリズムカーネルの詳細な仕様、そして DSCL ライブラリ API のガイドラインが記載されています。このアプリケーションノートでは、*カーネル*という用語は FIR フィルタなどの共通 DSCL 関数を意味します。DSCL ライブラリでは、いくつかの異なる C 言語の関数呼び出しが同じ DSP カーネルに関連付けられることがあります。混乱を避けるため、*カーネル*は、DSCL ライブラリで DSP アルゴリズムを実装するための関数コレクションを含む DSP アルゴリズムを指すものとします。個別の関数に言及する場合は、特定の DSCL ライブラリ関数名を使用します。このアプリケーションの仕様は、RX DSP ライブラリ V1.0 と同様であるため、新しいバージョンの RX DSP ライブラリでは仕様が異なる場合があります。

動作確認デバイス

RL78 コア S3 - 未指定

目次

1. DSCL ライブラリカーネル

このアプリケーションノートで定義されているフィルタカーネルは次のとおりです。

1. 汎用 FIR
2. IIR バイクワッド
3. 単極 IIR

1.1 略称および頭字語一覧

略称	完全形
DSC	デジタル信号コントローラ
DSP	デジタル信号プロセッサ
FIR	有限インパルス応答
GPIO	汎用I/O
I/O	入出力
LSB	最下位ビット
MSB	最上位ビット

1.2 DSCL ライブラリビルド情報

1.2.1 ツールチェーン情報

DSCL ライブラリは、以下のツールを使用してビルドおよびテストされています。

- CS+バージョン 8.05.00、ビルドツール: CC-RL バージョン 1.10.00

2. DSCL ライブラリ API

このアプリケーションノートでは、DSCL ライブラリのすべての関数に共通するルネサス DSCL ライブラリ API デザインの側面について説明します。

2.1 用語

このアプリケーションノートでは、「カーネル」という用語は、DSCL ライブラリで実装されている DSP アルゴリズム（またはそのバリエーション）を指します。「関数」という用語は、DSCL ライブラリ API 内の特定かつ単独の関数呼び出しを指します。カーネルの実装には複数の関数が必要な場合もあります。たとえば、フィルタカーネルでは、初期化と他の整理タスクに 1 つまたは複数の関数が必要で、それ以外にもフィルタ処理用のメイン関数が必要です。

2.2 データ構造

ライブラリでは以下の種類のデータ構造を定義しています。

- ベクタ
- アルゴリズムカーネルハンドル

2.2.1 ベクタ

ベクタデータ構造には、ベクタディメンションと実際のデータアレイへのポインタが含まれます。

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

【注】 ベクタデータ用のバッファメモリの割り当てはユーザの責任となります。さらに、ベクタ構造の「データ」メンバーは(void *)として宣言されるため、ライブラリでサポートされるデータ型ごとに個別のベクタ構造を用意する必要はありません。

2.2.2　アルゴリズムカーネルハンドル

状態情報、定数データ、そして各種実行時パラメータを必要とするカーネル関数では、これらのデータは、すべてカーネル関数（または変換などの関数クラス）に特有の「ハンドル」データ構造に集約されます。たとえば、FIR フィルタハンドルは次のように定義されます。

```
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; // option flags that may specify rounding,
                              // saturation, or other behaviors
} r_dscl_firfilter_t;
```

【注】　ハンドルデータ構造には、ユーザに表示する必要のあるメンバーのみが含まれます。一部のカーネルでは、追加の実装特有の状態も管理する必要があります。

カーネル「ハンドル」データ構造のすべてのメンバーは、ユーザが初期化する必要があります。係数や状態メモリへのポインタも含まれます。係数や状態メモリは、ユーザが割り当てる必要があります。一部の DSP カーネルには、実装に依存した状態や係数のメモリ要件があります。この場合は、カーネルの必須パラメータを指定することで割り当てられるメモリ量を返す API 関数が提供されます。

多くの関数は、与えられたパラメータによって適切なカーネル実装に分岐するために、*'options,'*などのハンドル構造メンバーに対してランタイムチェックを実行する必要があります。最も一般的な実装の選択においてこれらのランタイムチェックのオーバーヘッドを最小限に抑えるため、可能な限り、デフォルト値の NULL が定義されています。このデフォルト値により、最も一般的な必須動作（主にカーネルの最速実装）が提供されます。

ユーザは、カーネルを再初期化せずにハンドル構造で提供されたカーネルパラメータを変更してはなりません。たとえば、動作中に FIR フィルタの丸めモードやタップ数を変更することは禁止されています。これらのカーネルパラメータを変更する場合は、カーネルの内部状態を格納するのに十分な量のメモリが割り当てられていることを確認してから、新しいパラメータでカーネルを再初期化する必要があります。フィルタ係数値を変更する場合は、この制限は適用されません。フィルタ係数はいつでも変更できます。

2.3 関数の引数

すべての関数は次の順序で引数を受け入れます。

<handle>：カーネル特有の状態、係数、パラメータ、そしてオプションが含まれているカーネルハンドルデータ構造へのポインタ。
<input1>…<inputN>：スカラデータを除くほとんどのデータ型のポインタとして渡される 1 つまたは複数の入力引数。スカラデータ値は直接渡すことができます。
<output1>…<outputN>：1 つまたは複数の出力ポインタ。
<追加オプション>：カーネルハンドルデータ構造には含まれない任意のカーネルパラメータまたはオプション。

【注】 関数呼び出しでは上記の要素がすべて含まれている必要はありません。たとえば、FIR フィルタ初期化関数には入力や出力はありません。

ほとんどの関数は、16 ビットの整数結果を返します。整数結果には、アプリケーションの整理タスクに必要なエラーコードや他の情報が含まれている場合もあります。たとえば、カーネルの内部状態を格納するために割り当てる必要のあるメモリ量を返したり、カーネル特有の特別な条件の発生を示したりします。このルールの例外は、単一の実数値のスカラ結果を計算し、エラー状態が発生しえない場合です。このような場合には、関数はステータスコードではなく結果を返します。

何らかのタスクに割り当てる必要のあるメモリ量を示す値を返す関数が負の値を返した場合は、エラー状態を示しています（「2.4 エラー処理」参照）。C99 以降、*malloc()*関数は符号なしのデータ型である *size_t* を想定します。*size_t* の実際のビット幅はプラットフォームに依存します。そのため、DSCL ライブラリ関数から有効な（エラーではない）結果が返されたのを確認してから、結果を *malloc()*に渡すようにしてください。

ほとんどの関数呼び出しの形式は次のようになります。

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

ほとんどの関数には上記の引数クラスの一部のみが含まれます。

2.4 エラー処理

すべての関数は入力引数とカーネルパラメータに対して可能な限り最大限のチェックを実行します。ほとんどの関数は、16 ビットの整数ステータスコードを返します。カーネルの内部状態を（メモリ割り当てのために）返す関数は例外です。たとえば、R_DSCL_FIR_stateSize_i16i16 関数などです。

すべての関数は、負の整数値でエラー状態を表します。特定のエラー状態には、関数ごとに指定された一意の負の整数値が割り当てられています。関数が成功した場合は、0 を返すか、または正の整数値を返すことで、エラー以外の結果または特別な状態を示します。たとえば、R_DSCL_FIR_stateSize_i16i16 関数は、FIR フィルタの状態を格納するためのメモリサイズ要件を返します。他の関数は、特別な非エラー状態（オーバフローの発生など）を示す正の値を返します。

メモリを割り当てるためにメモリサイズを結果として返す関数が 0 を返した場合は、指定されたカーネルパラメータにはメモリが不要であることを意味します。

エラー状態とステータス状態の違いに注意してください。エラー状態（R_DSCL_ERR_<description>で宣言）は常に負の整数値を持ち、カーネルの動作を妨げる状態を示します（例：NULL 入力ポインタ）。これに対してステータス状態（R_DSCL_STATUS_<description>で宣言）は正の整数値（R_DSCL_STATUS_OK の場合は 0）を持ち、カーネルの出力に影響する可能性があるものの、カーネルによる算術演算の進行は妨げない状態を示します。たとえば、演算オーバフローはステータス状態として示されます。したがって、一部のアプリケーションではステータス状態は無視してもかまいませんが、エラー状態は常に対処が必要です。エラーコードには負の値、ステータスコードには正の値（または 0）が割り当てられているため、ユーザのコードではこれら 2 種類の状態を容易に区別できます。

エラーコードとステータスコードは、ヘッダファイル *r_dscl_types.h* の *enum* 宣言で定義されます。

すべての関数のエラーコードで使用される一般的な形式は以下のとおりです。

R_DSCL_STATUS_OK：問題が発生していません。このコードの値は 0 です。
R_DSCL_ERR_<pointer>_NULL：関数が NULL ポインタに遭遇しました。<pointer>は、エラーとなったポインタの名前です。たとえば、R_DSCL_ERR_INPUT_NULL は、入力引数へのポインタが NULL であることを意味します。*input* がベクタまたはマトリクスであり、ベクタ/マトリクス構文中のデータポインタが NULL である場合にも、このコードが使用されます。
R_DSCL_ERR_INVALID_<x>：関数に（ハンドル経由で、または直接）渡されたオプションまたはパラメータが実装でサポートされていません。<x>は、問題のある関数引数、カーネルパラメータ、または構文メンバーを識別します。たとえば、フィルタのハンドル構造体に丸めモードを指定した'options'メンバーが存在し、サポートされていない値がこのメンバーで指定されている場合には、R_DSCL_ERR_INVALID_OPTIONS コードが使用されます。

また、一部の関数特有のエラーコードやステータスコードが定義されています。DSCL ライブラリ仕様のフェーズ 1 では、以下のエラーコードとステータスコードが定義されています。

R_DSCL_STATUS_OK = ステータス OK、問題は発生していません。
R_DSCL_ERR_HANDLE_NULL = ハンドルへのポインタが NULL です。
R_DSCL_ERR_INPUT_NULL = 入力ベクタまたはその中のデータへのポインタが NULL です。
R_DSCL_ERR_OUTPUT_NULL = 出力ベクタまたはその中のデータへのポインタが NULL です。
R_DSCL_ERR_STATE_NULL = FIR または IIR フィルタの内部状態へのポインタが NULL です。
R_DSCL_ERR_COEFF_NULL = 係数アレイへのポインタが NULL です。
R_DSCL_ERR_INVALID_TAPS = フィルタタップ数が 0 または実装でサポートされていません。
R_DSCL_ERR_INVALID_STAGES =フィルタステージ数が 0 または実装でサポートされていません。
R_DSCL_ERR_INVALID_OPTIONS = handle の options 値で現在サポートされていないモードが指定されています。

2.5　丸めのサポート

DSCL ライブラリの一部のカーネルでは、複数の丸めモードをサポートしています。これらのモードは、固定小数点データ型に適用されます。

丸めモードは、カーネルのハンドル構造体の *options* 要素でサポートされます。*options* の以下のビットフィールドは、丸めモードと飽和モード用に予約されています。

- ビット 0-2：丸めモード
  - R_DSCL_ROUNDING_DEFAULT = 0
  - R_DSCL_ROUNDING_TRUNC = 1
  - R_DSCL_ROUNDING_NEAREST = 2
  - 予約済み = 3-7

【注】 R_DSCL_ROUNDING_DEFAULT は、カーネルのデフォルト動作です。ライブラリのすべてのフィルタタイプにおいて、デフォルトの動作は切り詰めです。

3. フィルタ関数 API

このセクションでは、RL78 DSCL ライブラリで実装されているフィルタ関数について説明します。

3.1 FIR データ構造体の定義

FIR カーネルは、r_dscl_firfilter_t タイプのフィルタへのハンドルを使用します。このハンドルは、フィルタ呼び出しの一環として渡されます。ハンドルタイプのデータ構造体は次のようになります。

```
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;
```

データ構造体の各メンバーについて以下に説明します。

taps = フィルタタップ数

coefs = 係数ベクタへのポインタ（入力ベクタと同じデータ型である必要があります）。アレイの内容はユーザが管理します。

state = フィルタの内部状態へのポインタ、遅延ラインと他の実装に依存する状態を含みます。内部状態を格納するためのメモリはユーザによって割り当てられ、内部状態の内容はカーネルによって管理されます。

options = ビットマップされたパラメータ制御オプション。使用できるモードの概要は、ソフトウェア概要セクションの「丸めのサポート」を参照してください。

3.2 FIR 初期化 API

ハンドルで指定されたオプションに従ってフィルタ状態を初期化（遅延ラインと他のパラメータの 0 リセットも含む）するための関数です。ランタイム呼び出し関数を呼び出す前に、この関数を 1 回呼び出す必要があります。

形式

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

パラメータ

handle	r_dscl_firfilter_t データ構造体のインスタンスへのポインタ。
handle →state	入力データと同じアレイ上にある遅延ラインの開始アドレスへのポインタです。

戻り値

R_DSCL_STATUS_OK = ステータス OK、問題は発生していません。
R_DSCL_ERR_HANDLE_NULL = ハンドルへのポインタが NULL です。
R_DSCL_ERR_STATE_NULL = 遅延ラインへのポインタが NULL です。
R_DSCL_ERR_INVALID_TAPS = タップ数が 0 です。
R_DSCL_ERR_INVALID_OPTIONS = handle の options 値で現在サポートされていないモードが指定されています。
その他 = 予約済み。

【注】 この関数は､ハンドル構造体の状態要素によって参照されている FIR 状態の内容のみを初期化します。フィルタ係数や、他のハンドル構造体の内容は初期化しないため、別に初期化する必要があります。

例

この関数は単体では使用しませんので、実際の使用例については FIR フィルタの例を参照してください。

制限

ハンドルが事前にインスタンス化されている必要があります。詳細は FIR の例を参照してください。

3.3 FIR フィルタ API

ブロック有限インパルス応答（FIR）フィルタカーネルは、呼び出されるたびに、ユーザが選択可能な入力サンプル数に対して同じ数の出力サンプルを生成します。

形式

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

パラメータ

handle	r_dscl_firfilter_t データ構造体のインスタンスへのポインタ。
input	入力データの vector_t データ構造体へのポインタ。この関数は、インスタンスも実際の入力データも変更しません。
input→n	関数が処理する入力サンプル数。この値は、関数を呼び出す前に設定する必要があります。
input→data	入力データの開始アドレスへのポインタ。このポインタは、関数を呼び出す前に設定する必要があります。
output	出力データの vector_t データ構造体へのポインタ。この関数は、インスタンスと実際の出力データの両方を変更します。
output→n	関数が生成する出力サンプル数。この値は関数によって書き込まれます。
output→data	出力データバッファへのポインタ。このポインタは、関数を呼び出す前に設定する必要があります。出力データバッファは関数によって書き込まれます。

戻り値

R_DSCL_STATUS_OK = ステータス OK、問題は発生していません。
R_DSCL_ERR_HANDLE_NULL = ハンドルへのポインタが NULL です。
R_DSCL_ERR_INPUT_NULL = 入力ベクタまたはその中のデータへのポインタが NULL です。
R_DSCL_ERR_OUTPUT_NULL = 出力ベクタまたはその中のデータへのポインタが NULL です。
R_DSCL_ERR_STATE_NULL = フィルタの内部状態へのポインタが NULL です。
R_DSCL_ERR_COEFF_NULL = 係数アレイへのポインタが NULL です。
R_DSCL_ERR_INVALID_TAPS = フィルタタップ数が 0 です。
R_DSCL_ERR_INVALID_OPTIONS = handle の options 値で現在サポートされていないモードが指定されています。
その他 = 予約済み。

説明

ブロック FIR フィルタカーネルは、各入力サンプルに対して有限インパルス応答フィルタを実装します。次の数式は、T タップ FIR フィルタの一般構造を示しており、h は係数、x は入力データ、y は出力データを表しています。

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

各出力サンプルは、n タップの FIR フィルタを実行した結果です。これを**図1**に示します。

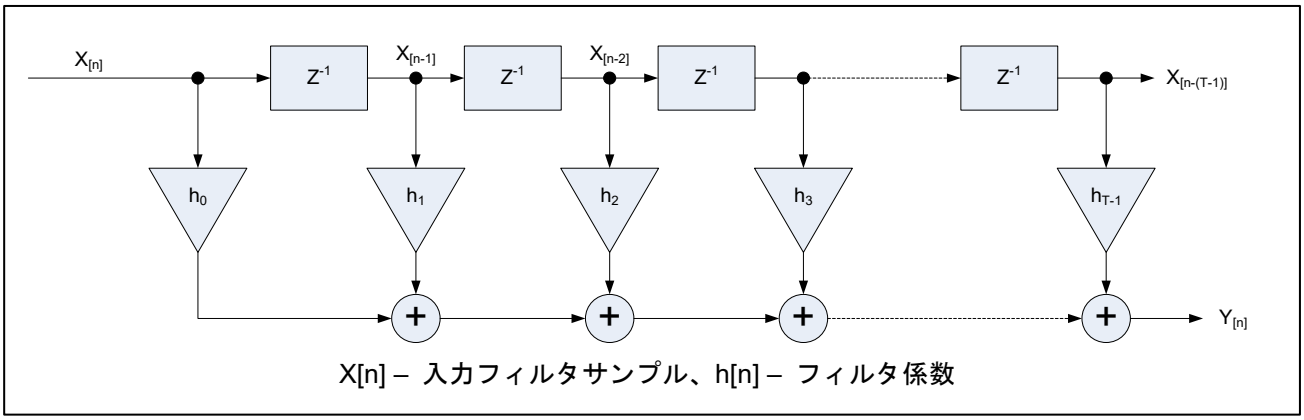

図1　FIR フィルタ

固定小数点動作

関数は固定小数点で実装されているため、固定小数点の動作に注意する必要があります。次の問題を考慮する必要があります。

- スケーリング
- オーバフロー

スケーリング：出力データのスケーリングファクタ「*FIR_SCALE_A*」は「*r_dscl_filter_asm.inc*」で定義されています。出力をメモリに書き込む前に、結果がスケールによって右にシフトされます。

スケールは、係数の小数位ビット数と等しくなければなりません。例：

フィルタ係数が Q4.12 形式で、フィルタ入力が Q2.14 形式である場合、各出力サンプルの累積結果は Q6.26 形式になります。スケール値に 12 を設定することで、累積結果の最下位 12 ビットを切り捨て、最終的な出力ワードでは小数部の 14 ビットのみを残すことによって、必要な変換を実行します。

スケーリングファクタのデフォルト値は 15 です。この値を変更した場合は、ライブラリを再コンパイルする必要があります。

オーバフロー：この関数は、精度とオーバフロー保護を犠牲にしてスピードが最適化されています。この関数は、一連の乗算/累算演算を使用して実装されています。アキュムレータは 32 ビットしかないため、オーバフローが発生します。累算後の最終結果は 16 ビットに変換されるため、精度も失われます。オーバフローを完全に回避するためには、入力データを log2(taps)ビットだけスケールダウンする必要があります（最大 15 ビット）。

例

FIR フィルタの初期化とランタイム使用の例を示します。

```
#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.
*-----------------------------------------------------------*/
```

処理フロー

上の例は、すべてのフィルタサンプルを 1 回だけ実行しており、この場合は入力バッファと出力バッファがすべてのデータを格納するのに十分なサイズを持っている必要があります。そうでない場合は、入力サンプルを何回かフィルタする必要があります。処理フローとスケーリングファクタを**図2**に示します。

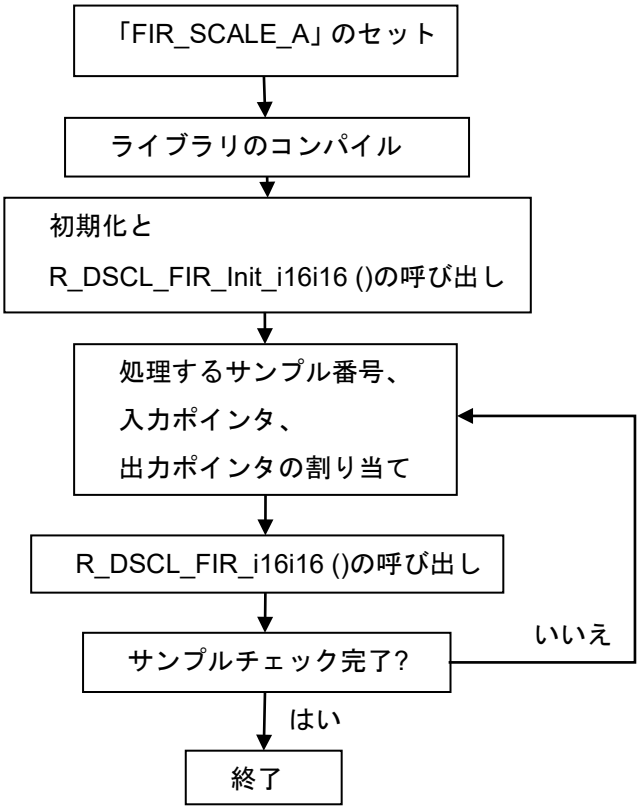

図2　処理フロー

制限

係数ベクタのタップ数は、フィルタハンドルで指定されている値と一致する必要があります。

3.4 IIR バイクワッドデータ構造体の定義

フィルタハンドル r_dscl_iirbiquad_t の定義を以下に示します。

```
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;
```

データ構造体の各メンバーについて以下に説明します。

stages = バイクワッドステージ数

coefs = 係数ベクタへのポインタ（入力ベクタと同じデータ型である必要があります）。アレイの内容はユーザが管理します。

state = フィルタの内部状態へのポインタ、遅延ラインと他の実装に依存する状態を含みます。内部状態を格納するためのメモリはユーザによって割り当てられ、内部状態の内容はカーネルによって管理されます。

options = ビットマップされたパラメータ制御オプション。使用できるモードの概要は、「丸めのサポート」を参照してください。

3.5 IIR バイクワッド状態サイズ API

これは IIR フィルタの「メンテナンス」関数です。この関数は、フィルタの内部状態（遅延ラインを含む）を格納するためにユーザが割り当てる必要のあるサイズ（バイト数）を返します。

形式

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

パラメータ

handle　　r_dscl_iirbiquad_t データ構造体のインスタンスへのポインタ。

戻り値

フィルタで必要なバイト数でのバッファサイズ（type int16_t）。

【注】 この関数から返されるバッファサイズは、実装側がフィルタに関連する非公開情報（ポインタ、入力および出力データ型のレコードなど）を管理するのに十分なサイズである必要があります。また、返されるサイズには、フィルタハンドルや係数アレイは含まれません。

説明

この関数をフィルタの初期化中に使用することで、ユーザが割り当てるべきバッファサイズを決定できます。あるいは、ユーザが開発中にこの関数を使用して必要なメモリサイズを決定し、そのサイズの静的アレイを（高速オンチップ RAM 上などで）内部状態用に割り当てることができます。

【注】 C99 以降、malloc()関数は符号なしのデータ型である size_t を想定します。size_t の実際のビット幅はプラットフォームに依存します。malloc(R_DSCL_IIRBiquad_StateSize_i16i16())を使用して内部状態を格納するためのメモリを割り当てたときに、R_DSCL_IIRBiquad_StateSize_i16i16()が負の値を返した場合は、予期しない動作が発生することがあります。

例

この関数は単体では使用しませんので、実際の使用例については IIR フィルタの例を参照してください。

制限

IIR ハンドルが事前にインスタンス化されている必要があります。詳細は IIR の例を参照してください。

3.6　IIR バイクワッド初期化 API

ハンドルで指定されたオプションに従ってフィルタ状態を初期化（遅延ラインと他のパラメータの 0 リセットも含む）するための関数です。ランタイム呼び出し関数を呼び出す前に、この関数を 1 回呼び出す必要があります。

形式

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

パラメータ

handle　　r_dscl_iirbiquad_t データ構造体のインスタンスへのポインタ。

戻り値

R_DSCL_STATUS_OK = ステータス OK、問題は発生していません。
R_DSCL_ERR_HANDLE_NULL = ハンドルへのポインタが NULL です。
R_DSCL_ERR_STATE_NULL = 遅延ラインへのポインタが NULL です。
R_DSCL_ERR_INVALID_STAGES= バイクワッドステージ数が 0 です。
R_DSCL_ERR_INVALID_OPTIONS = handle の options 値で現在サポートされていないモードが指定されています。
その他 = 予約済み。

【注】　この関数は、ハンドル構造体の状態要素によって参照されている IIR 状態の内容のみを初期化します。フィルタ係数や、他のハンドル構造体の内容は初期化しないため、別に初期化する必要があります。

説明

フィルタ状態を初期化（遅延ラインと他の実装特有パラメータの 0 リセットも含む）するための関数です。ランタイム呼び出し関数を呼び出す前に、この関数を 1 回呼び出す必要があります。

例

この関数は単体では使用しませんので、実際の使用例については IIR フィルタの例を参照してください。

制限

IIR ハンドルが事前にインスタンス化されている必要があります。詳細は IIR の例を参照してください。

3.7　IIR バイクワッドフィルタ API

このカーネルは、IIR（無限インパルス応答）フィルタを、カスケードバイクワッドの形式で実装します。バイクワッドは、2 次 IIR フィルタのセクションの 1 つです。さらに高次の IIR フィルタでは、カスケードバイクワッドの方が線形実装と比較して数値誤差が小さくなる傾向があります。

バイクワッドには、直接型 I および II や転置型 I および II など、さまざまな型があります。それぞれに長所と短所があります。IIR バイクワッド API は、直接型 I を使用して設計されています。

このカーネルは、呼び出されるたびに、ユーザが選択可能な入力サンプル数に対して同じ数の出力サンプルを生成します。カスケードバイクワッド数もユーザが選択できます。

形式

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

パラメータ

handle	r_dscl_iirbiquad_t データ構造体のインスタンスへのポインタ。
input	入力データの vector_t データ構造体へのポインタ。この関数は、インスタンスも実際の入力データも変更しません。
input→n	関数が処理する入力サンプル数。この値は、関数を呼び出す前に設定する必要があります。
input→data	入力データバッファへのポインタ。このポインタは、関数を呼び出す前に設定する必要があります。
output	出力データの vector_t データ構造体へのポインタ。　この関数は、インスタンスと実際の出力データの両方を変更します。
output→n	関数が生成する出力サンプル数。この値は関数によって書き込まれます。
output→data	出力データバッファへのポインタ。このポインタは、関数を呼び出す前に設定する必要があります。出力データバッファは関数によって書き込まれます。

戻り値

R_DSCL_STATUS_OK = ステータス OK、問題は発生していません。
R_DSCL_ERR_HANDLE_NULL = ハンドルへのポインタが NULL です。
R_DSCL_ERR_INPUT_NULL = 入力ベクタまたはその中のデータへのポインタが NULL です。
R_DSCL_ERR_OUTPUT_NULL = 出力ベクタまたはその中のデータへのポインタが NULL です。
R_DSCL_ERR_STATE_NULL = フィルタの内部状態へのポインタが NULL です。
R_DSCL_ERR_COEFF_NULL = 係数アレイへのポインタが NULL です。
R_DSCL_ERR_INVALID_STAGES= バイクワッドステージ数が 0 です。
R_DSCL_ERR_INVALID_OPTIONS = handle の options 値で現在サポートされていないモードが指定されています。
その他 = 予約済み。

説明

IIR バイクワッドフィルタは、カスケードバイクワッド型です。各バイクワッドは、入力と出力の関係が次の数式で表される 2 次 IIR フィルタのセクションとなります。

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

ただし、y(n)は出力サンプル、x(n)は入力サンプル、y(n-1)と x(n-1)はそれぞれ 1 サンプリング周期だけ遅延した出力サンプルと入力サンプル、y(n-2)と x(n-2)はそれぞれ 2 サンプリング周期だけ遅延した出力サンプルと入力サンプル、b0、b1、b2 はフィードフォワード係数、そして a1 と a2 はフィードバック係数です。

伝達関数全体は次のようになります。

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

ただし、N はカスケードバイクワッドステージ数です。各ステージには、係数 b0、b1、b2、a1、a2 の異なるセットがあります。

図3に、IIR バイクワッド直接型 I を示します。

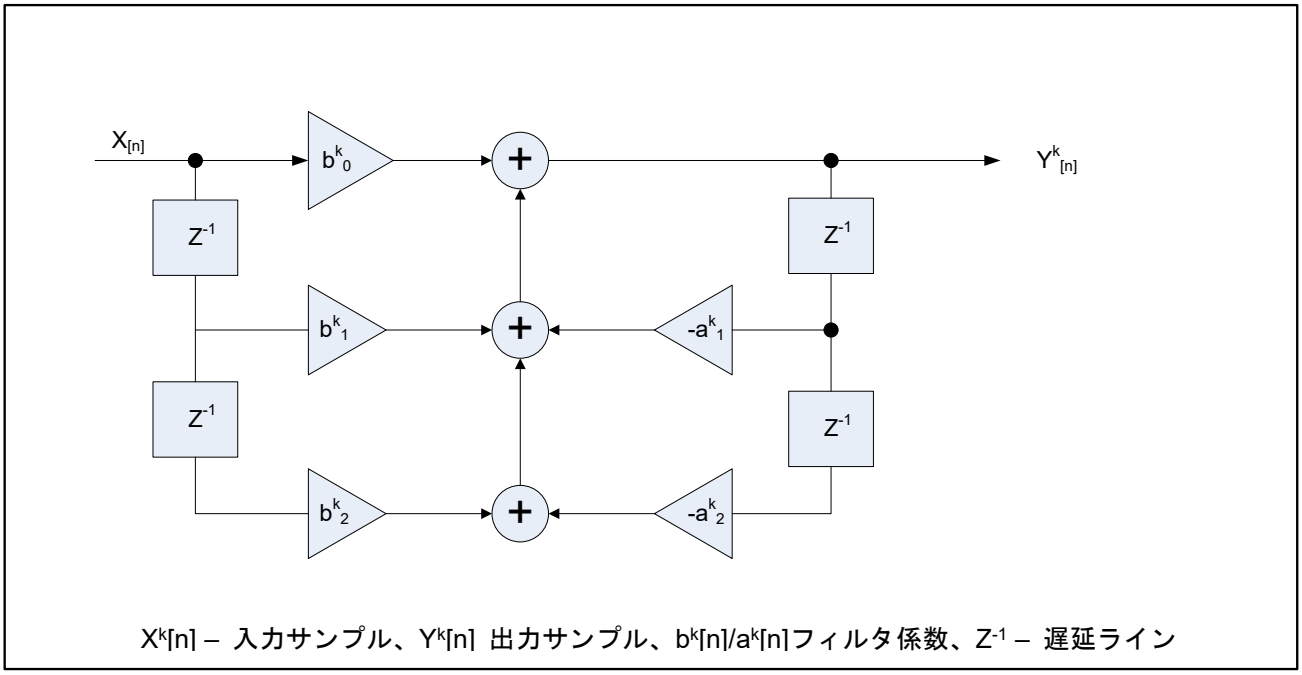

図3　IIR バイクワッド、直接型 I

固定小数点動作

関数は固定小数点で実装されているため、固定小数点の動作に注意する必要があります。次の問題を考慮する必要があります。

- スケーリング
- オーバフロー

スケーリング：出力データのスケーリングファクタ「*IIR_BQ_SCALE_A*」は「*r_dscl_filter_asm.inc*」で定義されています。出力をメモリに書き込む前に、結果がスケールによって右にシフトされます。

スケールは、係数の小数位ビット数と等しくなければなりません。例：

フィルタ係数が Q4.12 形式で、フィルタ入力が Q2.14 形式である場合、各出力サンプルの累積結果は Q6.26 形式になります。スケール値に 12 を設定することで、累積結果の最下位 12 ビットを切り捨て、最終的な出力ワードでは小数部の 14 ビットのみを残すことによって、必要な変換を実行します。

スケーリングファクタのデフォルト値は 14 です。つまり、係数によって[-2, 2)の範囲の値を表現できます。すべての係数値が[-1, 1)の範囲内にあれば、スケーリングファクタを 15 に変更できます。この場合は、ライブラリを再コンパイルする必要があります。

オーバフロー：この関数は、精度とオーバフロー保護を犠牲にしてスピードが最適化されています。この関数は、一連の乗算/累算演算を使用して実装されています。アキュムレータは 32 ビットしかないため、オー

バフローが発生します。累算後の最終結果は 16 ビットに変換されるため、精度も失われます。オーバフローを完全に回避するためには、入力データを 3 ビットだけスケールダウンする必要があります（最大 15 ビット）。

例

IIR バイクワッド関数の使用例を示します。

```
#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.
*-------------------------------------------------------------*/
```

処理フロー

上の例は、すべてのフィルタサンプルを 1 回だけ実行しており、この場合は入力バッファと出力バッファがすべてのデータを格納するのに十分なサイズを持っている必要があります。そうでない場合は、入力サンプルを何回かフィルタする必要があります。処理フローとスケーリングファクタを**図4**に示します

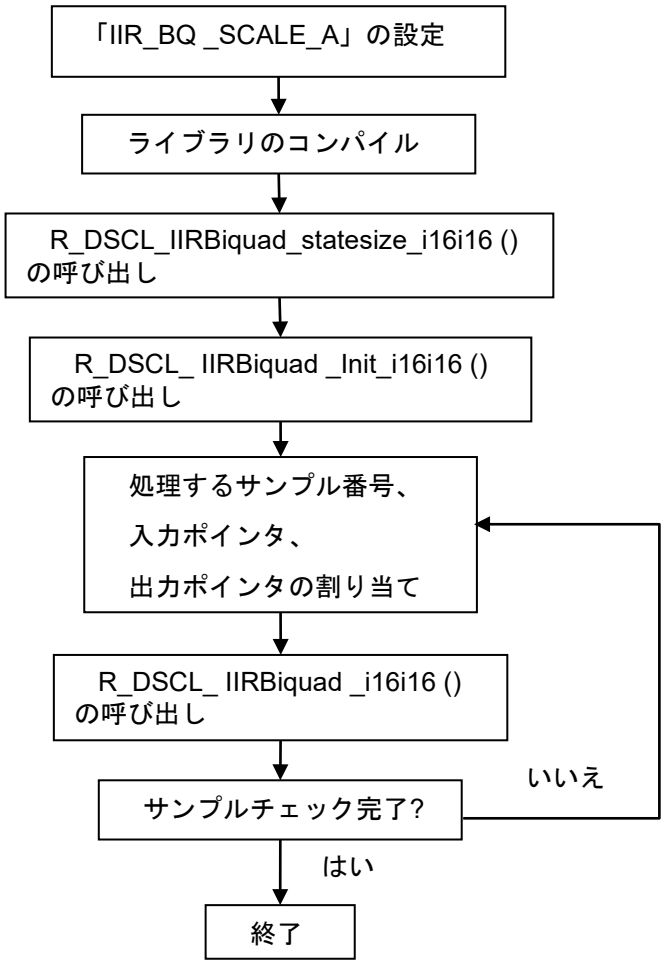

図4　処理フロー

制限

遅延ラインの長さは、カスケードステージ数によって決まります。そのため、このパラメータは R_DSCL_IIRBiquad_StateSize_i16i16 ()関数を呼び出す前に設定する必要があります。

3.8　単極 IIR データ構造体の定義

単極フィルタカーネルのすべてのバリエーションで使用されるフィルタハンドル r_dscl_iirsinglepole_t の定義は次のとおりです。

```
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;
```

データ構造体の各メンバーについて以下に説明します。

coefs = フィードバックタップの係数へのポインタ（入力と同じデータ型である必要があります）。係数はユーザが管理します。

state = フィードバックタップの状態へのポインタ。状態はカーネルが管理します。

options = ビットマップされたパラメータ制御オプション。使用できるモードの概要は、「丸めのサポート」を参照してください。

3.9　単極 IIR フィルタ API

このカーネルは、フィードバックタップが 1 つの IIR（無限インパルス応答）フィルタである単極フィルタを実装します。最大ゲインはユニティです。

形式

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

パラメータ

handle	r_dscl_iirsinglepole_t データ構造体のインスタンスへのポインタ。
input	入力データの vector_t データ構造体のインスタンスへのポインタ。この関数は、インスタンスも実際の入力データも変更しません。
input→n	関数が処理する入力サンプル数。この値は、関数を呼び出す前に設定する必要があります。
input→data	入力データバッファへのポインタ。このポインタは、関数を呼び出す前に設定する必要があります。
output	出力データの vector_t データ構造体のインスタンスへのポインタ。　この関数は、インスタンスと実際の出力データの両方を変更します。
output→n	関数が生成する出力サンプル数。この値は関数によって書き込まれます。
output→data	出力データバッファへのポインタ。このポインタは、関数を呼び出す前に設定する必要があります。出力データバッファは関数によって書き込まれます。

戻り値

R_DSCL_STATUS_OK = ステータス OK、問題は発生していません。
R_DSCL_ERR_HANDLE_NULL = ハンドルへのポインタが NULL です。
R_DSCL_ERR_INPUT_NULL = 入力ベクタまたはその中のデータへのポインタが NULL です。
R_DSCL_ERR_OUTPUT_NULL = 出力ベクタまたはその中のデータへのポインタが NULL です。
R_DSCL_ERR_INVALID_OPTIONS = handle の options 値で現在サポートされていないモードが指定されています。
その他 = 予約済み。

【注】　このカーネルには初期化関数はありません。内部状態を 0 に初期化するのはユーザの責任です。

説明

ローパスとハイパスの単極 IIR フィルタを、それぞれ**図5**と**図6**に示します。

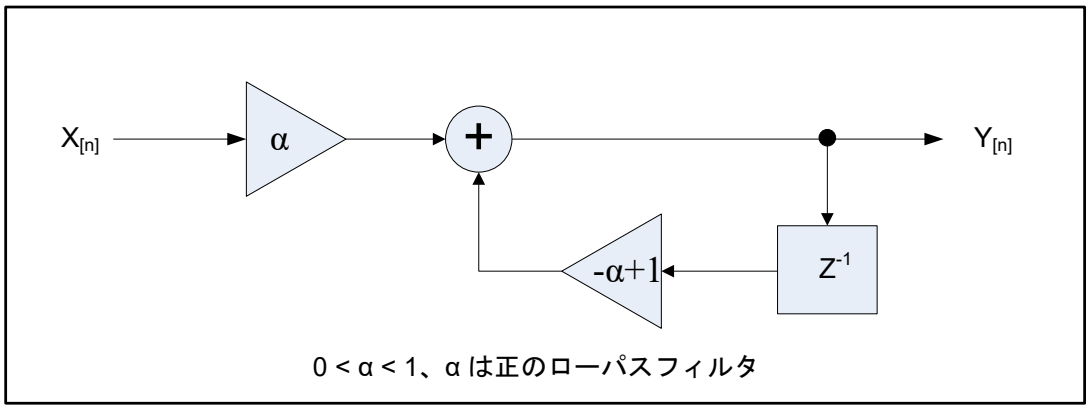

図5　ローパス単極 IIR

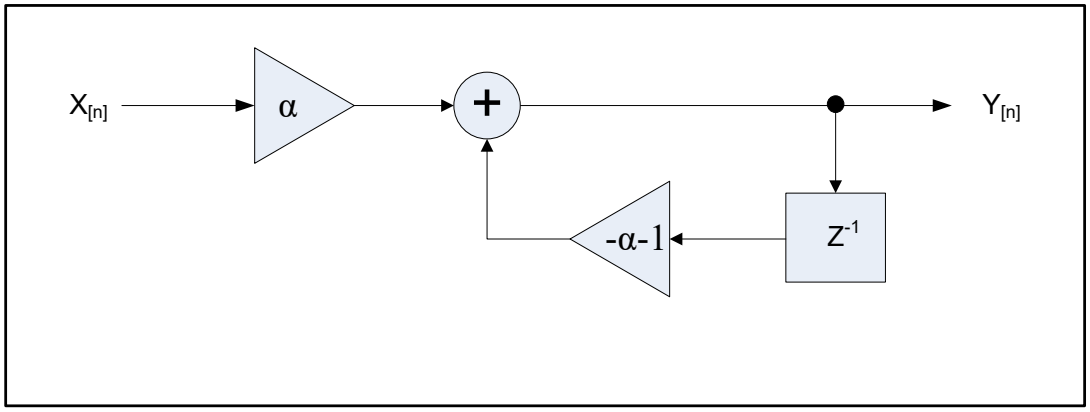

図6　ハイパス単極 IIR

単極ローパス IIR フィルタには次の伝達関数があります。

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

ただし、α は常に正で、フィルタ特性を決定します。α が 1.0 であれば、フィルタは入力信号を変更せずに通します。α が 0 に向かって減少すると、高周波成分の減衰が大きくなります。単極ローパスフィルタの出力は、次のように計算できます。

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

または

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

ただし、x_n は入力信号、y_n はフィルタ出力です。

単極ハイパスフィルタは次の伝達関数で実装できます。

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

ただし、このハイパスフィルタは、α が 0 に近づくとナイキスト周波数で発振するようになります。フィルタのこの性質は多くのアプリケーションでは望ましくないため、多くの単極ハイパスフィルタは、単極ローパスフィルタの出力を入力信号から減算することによって実装されています。そのため、ハイパス出力はシンプルな差分となります。

$$y'_n = x_n - y_n$$

ただし、x_n は入力信号、y_n は上記のように計算されたローパスフィルタ出力、そして y'_n はハイパスフィルタ出力です。

固定小数点動作

関数は固定小数点で実装されているため、固定小数点の動作に注意する必要があります。次の問題を考慮する必要があります。

- スケーリング
- オーバフロー

スケーリング：出力データのスケーリングファクタ「*IIR_SP_SCALE_A*」は「*r_dscl_filter_asm.inc*」で定義されています。出力をメモリに書き込む前に、結果がスケールによって右にシフトされます。

スケールは、係数の小数位ビット数と等しくなければなりません。例：

フィルタ係数が Q4.12 形式で、フィルタ入力が Q2.14 形式である場合、各出力サンプルの累積結果は Q6.26 形式になります。スケール値に 12 を設定することで、累積結果の最下位 12 ビットを切り捨て、最終的な出力ワードでは小数部の 14 ビットのみを残すことによって、必要な変換を実行します。

スケーリングファクタのデフォルト値は 15 です。この値を変更した場合は、ライブラリを再コンパイルする必要があります。

オーバフロー：この関数は、精度とオーバフロー保護を犠牲にしてスピードが最適化されています。この関数は、乗算/累算演算を使用して実装されています。アキュムレータは 32 ビットしかないため、オーバフローが発生します。累算後の最終結果は 16 ビットに変換されるため、精度も失われます。オーバフローを完全に回避するためには、入力データを 1 ビットだけスケールダウンする必要があります（最大 15 ビット）。

例

実際の 16 ビット固定小数点入力および出力データでの単極 IIR 関数の使用例を以下に示します。

```
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.
*-------------------------------------------------------------*/
```

処理フロー

上の例は、すべてのフィルタサンプルを 1 回だけ実行しており、この場合は入力バッファと出力バッファがすべてのデータを格納するのに十分なサイズを持っている必要があります。そうでない場合は、入力サンプルを何回かフィルタする必要があります。処理フローとスケーリングファクタを**図7**に示します。

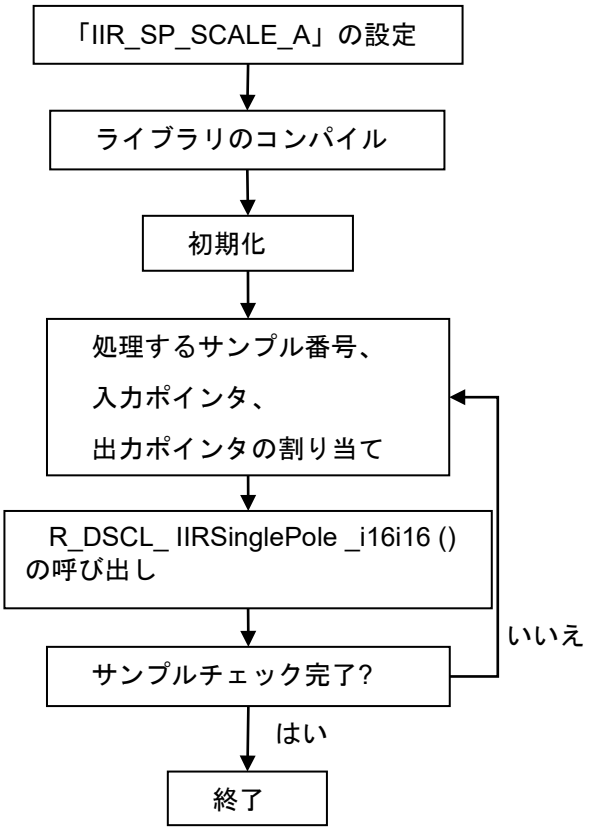

図7　処理フロー

制限

- 係数の大きさは 1.0 未満でなければなりません。

4. CS+でのサンプルワークスペース

4.1 16 ビット固定小数点ライブラリ単体

下記のインクルードファイルとライブラリファイルが提供されています。

このライブラリを単体で使用する場合は、表 1 に示されているファイルをインクルードして、表 2 の（コンパイラオプションに対応する）ライブラリファイルをリンクしてください。

表 1.　固定小数点ライブラリのインクルードファイル

ライブラリ	機能	
固定小数点ライブラリ	固定小数点演算を実装	「r_dscl_filters.h」

表 2.　固定小数点ライブラリ

ライブラリ名	コンパイラオプション
	Cpu
R_dscl_filter_rl78.lib	RL78/G14

これらのファイルを使用する前に、ローカルのインクルードまたはライブラリディレクトリにコピーしてください。

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

library    ——————————      R_dsp_rl78.lib
```

使用例

IIR 単極フィルタを使用したプログラムの例で、CS+でライブラリを指定する方法を示します。

[ソースプログラム]

```
#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);

  }
```

[CS+でライブラリを指定する方法]

プロジェクトツリーメニューで[CC-RL]の[Property]（プロパティ）を選択します。[Property]（プロパティ）ダイアログボックスで、[Frequently Used Options (for Link)]（よく使用するオプション（リンク））タブを選択し、「Using libraries」（ライブラリの使用）でライブラリを指定し、「Additional library paths」（追加のライブラリパス）でライブラリパスを指定します。

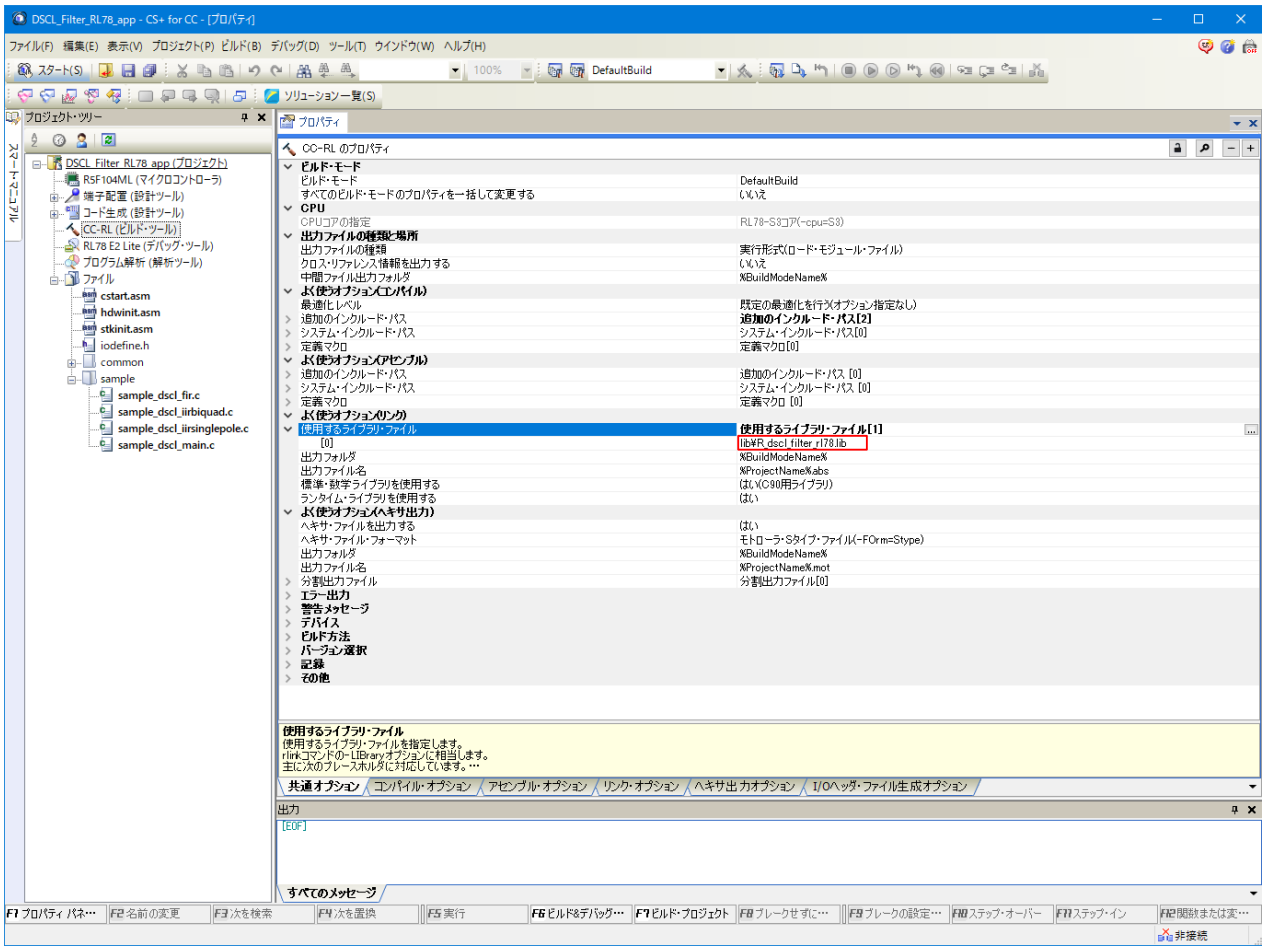

図 8　ライブラリの指定

4.2　リソースの要件

4.2.1　コードサイズとスタックサイズ

No.	カーネルカテゴリ	カーネルタイプ	入出力形式	関数	オプション	コードサイズ (Dec)	合計コードサイズ (Dec)	スタックサイズ (Dec)	全体スタックサイズ (Dec)
1	フィルター	汎用FIR	i16i16	R_DSP_FIR_StateSize_i16i16	-	13	13	4	4
				R_DSP_FIR_Init_i16i16	-	111	111	8	8
				R_DSP_FIR_i16i16	c interface	189	477	4	26
				R_DSP_FIR_i16i16	nr	137		20	
				R_DSP_FIR_i16i16	r	151		22	
		IIRバイクワッド	i16i16	R_DSP_IIRBiquad_StateSize_i16i16	-	8	8	2	4
				R_DSP_IIRBiquad_Init_i16i16	-	109	109	12	4
				R_DSP_IIRBiquad_i16i16	c interface	174	635	4	34
				R_DSP_IIRBiquad_i16i16	nr	222		28	
				R_DSP_IIRBiquad_i16i16	r	239		30	
		単極IIR	i16i16	R_DSP_IIRSinglePole_i16i16	c interface	173	488	6	32
				R_DSP_IIRSinglePole_i16i16	nr	143		22	
				R_DSP_IIRSinglePole_i16i16	r	172		26	

【注】　nr = R_DSP_ROUNDING_TRUNC（またはオプションなし）
　　r = R_DSP_ROUNDING_NEAREST

4.2.2　サイクルと精度

No.	フィルター		サンプル	タップ	オプション	サイクル	最大エラー	平均エラー
1	汎用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	単極IIR	ローパス	200	1	nr	8,191	3.02E-04	2.20E-04
4			200	1	r	9,915	4.44E-05	1.99E-05
5		ハイパス	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バイクワッド		200	4	nr	81,235	5.32E-04	4.00E-04
8			200	4	r	82,131	1.66E-04	4.82E-05

【注】　nr = R_DSP_ROUNDING_TRUNC（またはオプションなし）
　　r = R_DSP_ROUNDING_NEAREST

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2012 年 5 月 7 日	-	初版
1.01	2015 年 3 月 6 日	-	第 2 版
2.00	2021 年 4 月 13 日	3、27、29	ツールを「CubeSuite+、CA78K0R」から「CS+、CC-RL」に変更。
		27	サンプルコードの変更に併せてインクルードファイルを変更。

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。
5. クロックについて
   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。回路、ソフトウェアおよびこれらに関連する情報を使用する場合、お客様の責任において、お客様の機器・システムを設計ください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品または本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
6. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。

   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等

   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等

   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
10. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
12. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
13. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
14. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.5.0-1　2020.10)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。