General Digital Filters Library

User Reference Manual

56800E, 56800Ex
Digital Signal Controller

56800Ex_GDFLIB
Rev. 0
02/2014

freescale.com

freescale™

Chapter 1 License Agreement

NO WARRANTY. TO THE MAXIMUM EXTENT PERMITTED BY LAW, FREESCALE EXPRESSLY DISCLAIMS ANY WARRANTY FOR THE

SOFTWARE. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. YOU ASSUME THE ENTIRE RISK ARISING OUT OF THE USE OR PERFORMANCE OF THE SOFTWARE, OR ANY SYSTEMS YOU DESIGN USING THE SOFTWARE (IF ANY). NOTHING IN THIS AGREEMENT MAY BE CONSTRUED AS A WARRANTY OR REPRESENTATION BY FREESCALE THAT THE SOFTWARE OR ANY DERIVATIVE WORK DEVELOPED WITH OR INCORPORATING THE SOFTWARE WILL BE FREE FROM INFRINGEMENT OF THE INTELLECTUAL PROPERTY RIGHTS OF THIRD PARTIES.

INDEMNITY. You agree to fully defend and indemnify Freescale from any and all claims, liabilities, and costs (including reasonable attorney's fees) related to (1) your use (including your sublicensee's use, if permitted) of the Software or (2) your violation of the terms and conditions of this Agreement.

LIMITATION OF LIABILITY. IN NO EVENT WILL FREESCALE BE LIABLE, WHETHER IN CONTRACT, TORT, OR OTHERWISE, FOR ANY INCIDENTAL, SPECIAL, INDIRECT, CONSEQUENTIAL OR PUNITIVE DAMAGES, INCLUDING, BUT NOT LIMITED TO, DAMAGES FOR ANY LOSS OF USE, LOSS OF TIME, INCONVENIENCE, COMMERCIAL LOSS, OR LOST PROFITS, SAVINGS, OR REVENUES TO THE FULL EXTENT SUCH MAY BE DISCLAIMED BY LAW.

COMPLIANCE WITH LAWS; EXPORT RESTRICTIONS. You must use the Software in accordance with all applicable U.S. laws, regulations and statutes. You agree that neither you nor your licensees (if any) intend to or will, directly or indirectly, export or transmit the Software to any country in violation of U.S. export restrictions.

GOVERNMENT USE. Use of the Software and any corresponding documentation, if any, is provided with RESTRICTED RIGHTS. Use, duplication or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of The Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(l) and (2) of the Commercial Computer Software--Restricted Rights at 48 CFR 52.227-19, as applicable. Manufacturer is Freescale Semiconductor, Inc., 6501 William Cannon Drive West, Austin, TX, 78735.

HIGH RISK ACTIVITIES. You acknowledge that the Software is not fault tolerant and is not designed, manufactured or intended by Freescale for incorporation into products intended for use or resale in on-line control equipment in hazardous, dangerous to life or potentially life-threatening environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines or weapons systems, in which the failure of products could lead directly to death, personal injury or severe physical or environmental damage ("High Risk Activities"). You specifically represent and warrant that you will not use the Software or any derivative work of the Software for High Risk Activities.

CHOICE OF LAW; VENUE; LIMITATIONS. You agree that the statutes and laws of the United States and the State of Texas, USA, without regard to conflicts of laws principles, will apply to all matters relating to this Agreement or the Software, and you agree that any litigation will be subject to the exclusive jurisdiction of the state or federal courts in Texas, USA. You agree that regardless of any statute or law to the contrary, any claim or cause of action arising out of or related to this Agreement or the Software must be filed within one (1) year after such claim or cause of action arose or be forever barred.

PRODUCT LABELING. You are not authorized to use any Freescale trademarks, brand names, or logos.

ENTIRE AGREEMENT. This Agreement constitutes the entire agreement between you and Freescale regarding the subject matter of this Agreement, and supersedes all prior communications, negotiations, understandings, agreements or representations, either written or oral, if any. This Agreement may only be amended in written form, executed by you and Freescale.

SEVERABILITY. If any provision of this Agreement is held for any reason to be invalid or unenforceable, then the remaining provisions of this Agreement will be unimpaired and, unless a modification or replacement of the invalid or unenforceable provision is further held to deprive you or Freescale of a material benefit, in which case the Agreement will immediately terminate, the invalid or unenforceable provision will be replaced with a provision that is valid and enforceable and that comes closest to the intention underlying the invalid or unenforceable provision.

NO WAIVER. The waiver by Freescale of any breach of any provision of this Agreement will not operate or be construed as a waiver of any other or a subsequent breach of the same or a different provision.

Chapter 2 INTRODUCTION

2.1 Overview

This reference manual describes the General Digital Filters Library (GDFLIB) for the Freescale 56F800E(X) family of Digital Signal Controllers. This library contains optimized functions.

2.2 Supported Compilers

General Digital Filters Library (GDFLIB) is written in assembly language with C-callable interface. The library was built and tested using the CodeWarrior™ Development Studio version 10.3.

The library is delivered in library module 56800Ex_GDFLIB.lib and is intended for use in small data memory model projects. The interfaces to the algorithms included in this library have been combined into a single public interface include file, gdflib.h. This was done to simplify the number of files required for inclusion by application programs. Refer to the specific algorithm sections of this document for details on the software Application Programming Interface (API), defined and functionality provided for the individual algorithms.

2.3 Installation

If user wants to fully use this library, the CodeWarrior™ Development Studio should be installed prior to the General Digital Filters Library. In case that General Digital Filters Library is installed while CodeWarrior™ Development Studio is not present, users can only browse the installed software package, but will not be able to build, download and run code. The installation itself consists of copying the required files to the destination hard drive, checking the presence of CodeWarrior and creating the shortcut under the Start->Programs menu.

The General Digital Filters Library release is installed in its own folder named 56800Ex_GDFLIB.

To start the installation process, perform the following steps:

1. Execute 56800Ex_FSLESL_rXX.exe.
2. Follow the FSLESL software installation instructions on your screen.

2.4 Library Integration

The library integration is described in AN4586 which can be downloaded from www.freescale.com.

2.5 API Definition

The description of each function described in this General Digital Filters

Library user reference manual consists of a number of subsections:

Synopsis

This subsection gives the header files that should be included within a source file that references the function or macro. It also shows an appropriate declaration for the function or for a function that can be substituted by a macro. This declaration is not included in your program; only the header file(s) should be included.

Prototype

This subsection shows the original function prototype declaration with all its arguments.

Arguments

This optional subsection describes input arguments to a function or macro.

Description

This subsection is a description of the function or macro. It explains algorithms being used by functions or macros.

Return

This optional subsection describes the return value (if any) of the function or macro.

Range Issues

This optional subsection specifies the ranges of input variables.

Special Issues

This optional subsection specifies special assumptions that are mandatory for correct function calculation; for example saturation, rounding, and so on.

Implementation

This optional subsection specifies, whether a call of the function generates a library function call or a macro expansion.
This subsection also consists of one or more examples of the use of the function. The examples are often fragments of code (not completed programs) for illustration purposes.

See Also

This optional subsection provides a list of related functions or macros.

Performance

This section specifies the actual requirements of the function or macro in terms of required code memory, data memory, and number of clock cycles to execute.

2.6 Data Types

The 16-bit DSC core supports four types of two's-complement data formats:

- Signed integer
- Unsigned integer
- Signed fractional
- Unsigned fractional

Signed and unsigned integer data types are useful for general-purpose computation; they are familiar with the microprocessor and microcontroller programmers. Fractional data types allow powerful numeric and digital-signal-processing algorithms to be implemented.

2.6.1 Signed Integer (SI)

This format is used for processing data as integers. In this format, the N-bit operand is represented using the N.0 format (N integer bits). The signed integer numbers lie in the following range:

$$-2^{[N-1]} \le SI \le [2^{[N-1]} - 1]$$ **Eqn. 2-1**

This data format is available for bytes, words, and longs. The most negative, signed word that can be represented is –32,768 ($8000), and the most negative, signed long word is –2,147,483,648 ($80000000).

The most positive, signed word is 32,767 ($7FFF), and the most positive signed long word is 2,147,483,647 ($7FFFFFFF).

2.6.2 Unsigned Integer (UI)

The unsigned integer numbers are positive only, and they have nearly twice the magnitude of a signed number of the same size. The unsigned integer numbers lie in the following range:

$$0 \le UI \le [2^{[N-1]} - 1]$$ **Eqn. 2-2**

The binary word is interpreted as having a binary point immediately to the right of the integer's least significant bit. This data format is available for bytes, words, and long words. The most positive, 16-bit, unsigned integer is 65,535 ($FFFF), and the most positive, 32-bit, unsigned integer is 4,294,967,295 ($FFFFFFFF). The smallest unsigned integer number is zero ($0000), regardless of size.

2.6.3 Signed Fractional (SF)

In this format, the N-bit operand is represented using the 1.[N–1] format (one sign bit, N–1 fractional bits). The signed fractional numbers lie in the following range:

$$-1.0 \le SF \le 1.0 - 2^{-[N-1]} \quad \textbf{Eqn. 2-3}$$

This data format is available for words and long words. For both word and long-word signed fractions, the most negative number that can be represented is –1.0; its internal representation is $8000 (word) or $80000000 (long word). The most positive word is $7FFF ($1.0 - 2^{-15}$); its most positive long word is $7FFFFFFF ($1.0 - 2^{-31}$).

2.6.4 Unsigned Fractional (UF)

The unsigned fractional numbers can be positive only, and they have nearly twice the magnitude of a signed number with the same number of bits. The unsigned fractional numbers lie in the following range:

$$0.0 \le UF \le 2.0 - 2^{-[N-1]} \quad \textbf{Eqn. 2-4}$$

The binary word is interpreted as having a binary point after the MSB. This data format is available for words and longs. The most positive, 16-bit, unsigned number is $FFFF, or $\{1.0 + (1.0 - 2^{-[N-1]})\} = 1.99997$. The smallest unsigned fractional number is zero ($0000).

2.7 User Common Types

Table 2-1. User-Defined Typedefs in *56800E_types.h*

Mnemonics	Size — bits	Description
Word8	8	To represent 8-bit signed variable/value.
UWord8	8	To represent 16-bit unsigned variable/value.
Word16	16	To represent 16-bit signed variable/value.
UWord16	16	To represent 16-bit unsigned variable/value.
Word32	32	To represent 32-bit signed variable/value.
UWord32	32	To represent 16-bit unsigned variable/value.
Int8	8	To represent 8-bit signed variable/value.
UInt8	8	To represent 16-bit unsigned variable/value.
Int16	16	To represent 16-bit signed variable/value.
UInt16	16	To represent 16-bit unsigned variable/value.
Int32	32	To represent 32-bit signed variable/value.

Table 2-1. User-Defined Typedefs in *56800E_types.h* (continued)

UInt32	32	To represent 16-bit unsigned variable/value.
Frac16	16	To represent 16-bit signed variable/value.
Frac32	32	To represent 32-bit signed variable/value.
NULL	constant	Represents NULL pointer.
bool	16	Boolean variable.
false	constant	Represents false value.
true	constant	Represents true value.
FRAC16()	macro	Transforms float value from <–1, 1) range into fractional representation <–32768, 32767>.
FRAC32()	macro	Transforms float value from <–1, 1) range into fractional representation <–2147483648, 2147483648>.

2.8 V2 and V3 Core Support

The library has been written to support both 56800E (V2) and 56800Ex (V3) cores. The V3 core offers new set of math instructions which can simplify and accelarete the algorithm runtime. Therefore certain algorithms can have two prototypes.

If the library is used on the 56800Ex core, the V3 algorithms use is recommended because:

- the code is shorter
- the execution is faster
- the precision of 32-bit calculation is higher

The final algorithm is selected by a define. To select the correct algorithm implementation the user has to set up a define: OPTION_CORE_V3. If this define is not defined, it is automatically set up as 0. If its value is 0, the V2 algorithms are used. If its value is 1, the V3 algorithms are used.

The best way is to define this define is in the project properties (see Figure 2-1):

1. In the left hand tree, expand the C/C++ Build node
2. Click on the Settings node
3. Under the Tool Settings tab, click on the DSC Compiler/Input node
4. In the Defined Macros dialog box click on the first icon (+) and type the following: OPTION_CORE_V3=1
5. Click OK
6. Click OK on the Properties dialog box

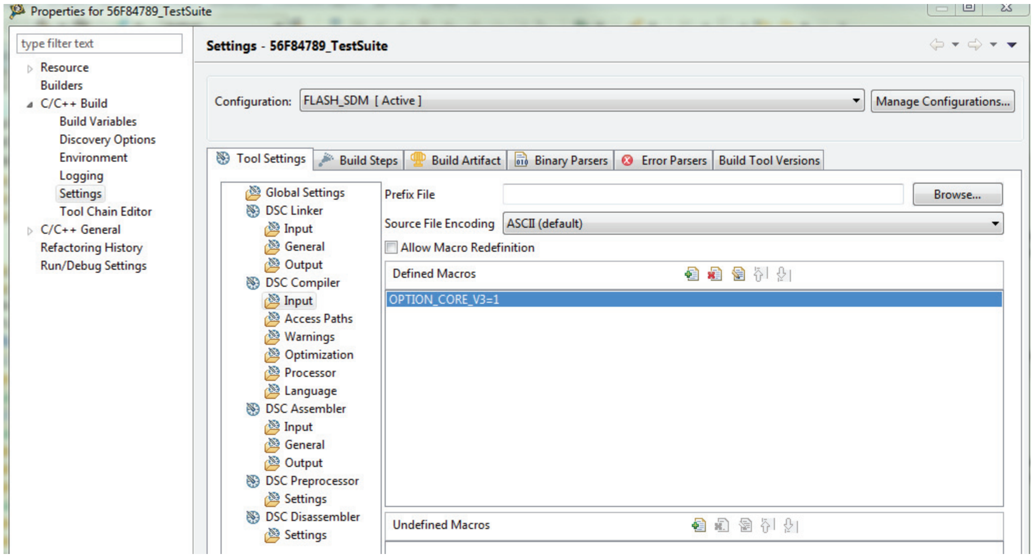

Figure 2-1. V2/V3 core option

2.9 Special Issues

All functions in the General Digital Filters Library are implemented without storing any of the volatile registers (refer to the compiler manual) used by the respective routine. Only non-volatile registers (C10, D10, R5) are saved by pushing the registers on the stack. Therefore, if the particular registers initialized before the library function call are to be used after the function call, it is necessary to save them manually.

Chapter 3 FUNCTION API

3.1 API Summary

Table 3-1. API Functions Summary

Name	Arguments	Output	Description
GDFLIB_FilterIIR1Init	GDFLIB_FILTER_IIR1_T *pudtFilter	Void	The function initializes internal variables of a first order IIR filter.
GDFLIB_FilterIIR1	Frac16 f16In GDFLIB_FILTER_IIR1_T *pudtFilter	Frac16	The function calculates first order Direct Form 1 IIR filter.
GDFLIB_FilterIIR2Init	GDFLIB_FILTER_IIR2_T *pudtFilter	Void	The function initializes internal variables of a second order IIR filter.
GDFLIB_FilterIIR2	Frac16 f16In GDFLIB_FILTER_IIR2_T *pudtFilter	Frac16	The function calculates second order Direct Form 1 IIR filter.
GDFLIB_FilterIIR3Init	GDFLIB_FILTER_IIR3_T *pudtFilter	Void	The function initializes internal variables of a third order IIR filter.
GDFLIB_FilterIIR3	Frac16 f16In GDFLIB_FILTER_IIR3_T *pudtFilter	Frac16	The function calculates third order Direct Form 1 IIR filter.
GDFLIB_FilterIIR4Init	GDFLIB_FILTER_IIR4_T *pudtFilter	Void	The function initializes internal variables of a fourth order IIR filter.
GDFLIB_FilterIIR4	Frac16 f16In GDFLIB_FILTER_IIR4_T *pudtFilter	Frac16	The function calculates fourth order Direct Form 1 IIR filter.
GDFLIB_FilterMA32Init	GDFLIB_FILTER_MA32_T *pudtFilter	Void	This function initializes the internal variables of of the GDFLIB_FilterMA32 function with zero.
GDFLIB_FilterMA32InitVal	Frac16 f16InitVal GDFLIB_FILTER_MA32_T *pudtFilter	Void	This function initializes the internal variables of of the GDFLIB_FilterMA32 function with a value.
GDFLIB_FilterMA32	Frac16 f16In GDFLIB_FILTER_MA32_T *pudtFilter	Frac16	The function calculates recursive form of an average filter.

3.2 GDFLIB_FilterIIR1Init

This function initializes the internal variables of a first order IIR filter.

3.2.1 Synopsis

```
#include "gdflib.h"
void GDFLIB_FilterIIR1Init(GDFLIB_FILTER_IIR1_T *pudtFilter)
```

3.2.2 Prototype

```
void GDFLIB_FilterIIR1InitFC(GDFLIB_FILTER_IIR1_T * const pudtFilter)
```

3.2.3 Arguments

Table 3-2. Function Arguments

Name	In/Out	Format	Range	Description
*pudtFilter	In/Out	N/A	N/A	Pointer to a filter structure, which contains filter coefficients and filter buffer; the GDFLIB_FILTER_IIR1_T data type is defined in header file GDFLIB_FilterIIRasm.h.

Table 3-3. User-Type Definitions

Typedef	Name	In/Out	Format	Range	Description
GDFLIB_FILTER_IIR1_T	f32FiltBufferY[1]	In/Out	SF32	0x80000000... 0x7FFFFFFF	filter buffer storing output values
	f16FiltBufferX[1]	In/Out	SF16	0x8000... 0x7FFF	filter buffer storing input values
	udtFiltCoeff	In	N/A	N/A	structure containing filter coefficients

Table 3-4. User-Type Definitions

Typedef	Name	In/ Out	Format	Range	Description
GDFLIB_FILTER_IIR_COEFF1_T	f16B1	In	SF16	$8000... $7FFF	B1 coefficient of the filter
	f16B2	In	SF16	$8000... $7FFF	B2 coefficient of the filter
	f16A2	In	SF16	$8000... $7FFF	A2 coefficient of the filter

3.2.4 Availability

This library module is available in the ANSI C format.

This library module is targeted for the 56800E and 56800Ex platforms.

3.2.5 Dependencies

List of all dependent files:

- GDFLIB_FilterIIRasm.h
- GDFLIB_types.h

3.2.6 Description

The **GDFLIB_FilterIIR1Init** function initializes the buffer and coefficients of the first order IIR filter. This function is called once, during the variable initialization, and since it clears the filter buffer, it must not be called together with the filter-calculation function.

3.2.7 Returns

This function initializes the filter structure pointed to by the pudtFilter pointer.

3.2.8 Range Issues

The filter coefficients must be defined prior to this function call. If the Matlab filter-design toolbox is used for the filter coefficients calculation, then all calculated coefficients must be divided by 2.0 in order to avoid saturation during filter calculation.

3.2.9 Special Issues

The function **GDFLIB_FilterIIR1Init** is the saturation mode independent.

3.2.10 Implementation

The **GDFLIB_FilterIIR1Init** function is implemented as a function call.

Example 3-1. Implementation Code

```
#include "gdflib.h"

static Frac16 mf16Value;
static Frac16 mf16FilteredValue;
static GDFLIB_FILTER_IIR1_T mudtFilterIIR1 = GDFLIB_FILTER_IIR1_DEFAULT;


void Isr(void);
```

```
void main(void)
{
        /* LPF 1st order butterworth 100Hz, Ts = 100us*/
        mudtFilterIIR1.udtFiltCoeff.f16B1 = FRAC16(0.0305 / (2.0));
        mudtFilterIIR1.udtFiltCoeff.f16B2 = FRAC16(0.0305 / (2.0));
        mudtFilterIIR1.udtFiltCoeff.f16A2 = FRAC16(-0.9391 / (2.0));

        /* Filter initialization */
        GDFLIB_FilterIIR1Init(&mudtFilterIIR1);
}

/* Periodical function or interrupt */
void Isr(void)
{
        /* Filter calculation */
        mf16FilteredValue = GDFLIB_FilterIIR1(mf16Value,
&mudtFilterIIR1);
}
```

3.2.11 Performance

Table 3-5. Performance of the GDFLIB_FilterIIR1Init Function

Code Size (words)	4	
Data Size (words)	0	
Execution Clock	Min	N/A
	Max	N/A

3.3 GDFLIB_FilterIIR1

This function calculates the first-order direct form one IIR filter.

3.3.1 Synopsis

```
#include ”gdflib.h”
Frac16 GDFLIB_FilterIIR1(Frac16 f16In, GDFLIB_FILTER_IIR1_T *pudtFilter)
```

3.3.2 Prototype

```
asm Frac16 GDFLIB_FilterIIR1FAsm(Frac16 f16In, GDFLIB_FILTER_IIR1_T *
const pudtFilter)
```

V3 core version:

```
asm Frac16 GDFLIB_V3FilterIIR1FAsm(Frac16 f16In, GDFLIB_FILTER_IIR1_T *
const pudtFilter)
```

3.3.3 Arguments

This subsection describes the input/output arguments to a function or a macro. It explains the algorithms being used by the functions or macro.

Table 3-6. Function Arguments

Name	In/Out	Format	Range	Description
f16In	In	SF16	0x8000... 0x7FFF	input signal to be filtered
*pudtFilter	In/Out	N/A	N/A	Pointer to a filter structure, which contains filter coefficients and filter buffer; the GDFLIB_FILTER_IIR_COEFF1_T data type is defined in header file GDFLIB_FilterIIRasm.h.

Table 3-7. User-Type Definitions

Typedef	Name	In/Out	Format	Range	Description
GDFLIB_FILTER_IIR1_T	f32FiltBufferY[1]	In/Out	SF32	0x80000000... 0x7FFFFFFF	filter buffer storing output values
	f16FiltBufferX[1]	In/Out	SF16	0x8000... 0x7FFF	filter buffer storing input values
	udtFiltCoeff	In	N/A	N/A	structure containing filter coefficients

Table 3-8. User-Type Definitions

Typedef	Name	In/ Out	Format	Range	Description
GDFLIB_FILTER_IIR_COEFF1_T	f16B1	In	SF16	0x8000... 0x7FFF	b1 coefficient of the filter
	f16B2	In	SF16	0x8000... 0x7FFF	b2 coefficient of the filter
	f16A2	In	SF16	0x8000... 0x7FFF	a2 coefficient of the filter

3.3.4 Availability

This library module is available in the C-callable interface assembly version format.

This library module is targeted for the 56800E and 56800Ex platforms.

3.3.5 Dependencies

The dependent files are:

- GDFLIB_FilterIIRasm.h
- GDFLIB_types.h

3.3.6 Description

The **GDFLIB_FilterIIR1Init** function calculates the first-order infinite impulse response (IIR) filter. The IIR filters are also called recursive filters, because both the input and the previously calculated output values are used for calculation. This form of feedback enables the transfer of energy from the output to the input, which theoretically leads to an infinitely long impulse response (IIR). A general form of the IIR filter, expressed as a transfer function in the Z-domain, is described as follows:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_1 + b_2 z^{-1} + b_3 z^{-2} + \ldots + b_{(N+1)} z^{-N}}{1 + a_2 z^{-1} + a_3 z^{-2} + \ldots + a_{(N+1)} z^{-N}} \qquad \textbf{Eqn. 3-1}$$

where N denotes the filter order. The first-order IIR filter in the Z-domain is therefore given as:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_1 + b_2 z^{-1}}{1 + a_2 z^{-1}} \qquad \textbf{Eqn. 3-2}$$

which is transformed into a time-domain difference equation as:

$$y(k) = b_1 x(k) + b_2 x(k-1) - a_2 y(k-1) \qquad \textit{Eqn. 3-3}$$

The filter difference equation is implemented in the digital signal controller directly, as written in Equation 3-3; this equation represents a direct-form one first-order IIR filter as depicted in Figure 3-1.

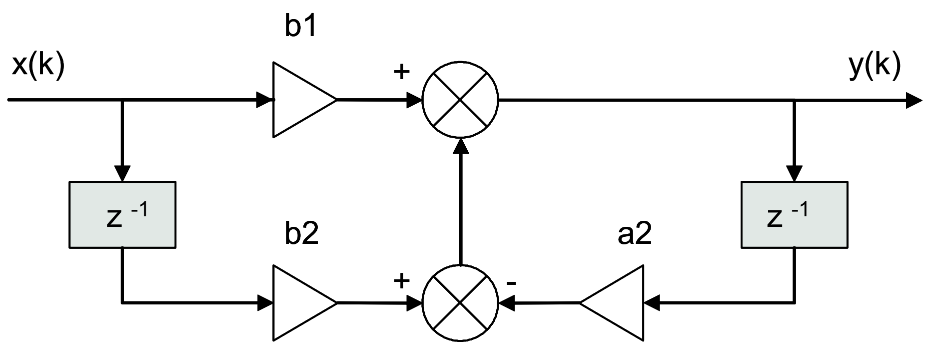

Figure 3-1. Direct-Form One First-Order IIR Filter

The coefficients of the filter depicted in Figure 3-1 can be designed to meet the requirements for the first-order low (LPF) or high-pass filter (HPF). The coefficient quantization error due to finite precision arithmetic can be neglected in the case of a first-order filter. A higher-order LPF or HPF can be obtained by connecting a number of the first-order filters in series. The number of connections gives the order of the resulting filter.

The filter coefficients are calculated using the Butterworth approximation. The Butterworth normalized transfer function in the s-plane is given as:

$$H_N(s) = \frac{1}{\prod_{k=1}^{n}(s - s_k)} \qquad \textit{Eqn. 0-1}$$

where

$$s_k = \begin{cases} e^{j(2k-1)\pi/2n} & \text{for even n} \\ e^{j(k-1)\pi/n} & \text{for odd n} \end{cases} \qquad \textit{Eqn. 3-4}$$

The normalized Butterworth first-order low-pass filter prototype is therefore given as:

$$H(s) = \frac{1}{s+1} \qquad \textit{Eqn. 3-5}$$

Transferring the prototype described in Equation 3-5 into a denormalized low-pass filter results in a transfer function:

$$H(s) = \frac{\omega_c}{s + \omega_c} \qquad \text{Eqn. 3-6}$$

This is a transfer function of Butterworth low-pass filter in the s-domain with the cutoff frequency given by the ω_c. Transformation of an analog filter described by Equation 3-6 into a discrete form is done using the bilinear transformation, resulting in the following transfer function:

$$H(z) = \frac{\frac{\omega_{cd}T_s}{2+\omega_{cd}T_s} + \frac{\omega_{cd}T_s}{2+\omega_{cd}T_s}z^{-1}}{1 + \frac{\omega_{cd}T_s - 2}{2+\omega_{cd}T_s}z^{-1}} \qquad \text{Eqn. 3-7}$$

where ω_{cd} is the cutoff frequency of the filter in the digital domain and T_s is the sampling period. However, mapping of the analog system into a digital domain using the bilinear transformation makes the relation between ω_c and ω_{cd} non-linear. This introduces a distortion in the frequency scale of the digital filter relative to that of the analog filter. This is known as warping effect. The warping effect can be eliminated by pre-warping the analog filter, and then transforming it into the digital domain, resulting in this transfer function:

$$H(z) = \frac{\frac{\omega_{cd\_p}T_{s\_p}}{2+\omega_{cd\_p}T_{s\_p}} + \frac{\omega_{cd\_p}T_{s\_p}}{2+\omega_{cd\_p}T_{s\_p}}z^{-1}}{1 + \frac{\omega_{cd\_p}T_{s\_p} - 2}{2+\omega_{cd\_p}T_{s\_p}}z^{-1}} \qquad \text{Eqn. 3-8}$$

where ω_{cd_p} is the pre-warped cutoff frequency of the filter in the digital domain, and T_{s_p} is the pre-warped sampling period. The pre-warped cutoff frequency is calculated as follows:

$$\omega_{cd\_p} = \frac{2}{T_{s\_p}}\tan\left(\frac{\omega_{cd}T_s}{2}\right) \qquad \text{Eqn. 3-9}$$

and the pre-warped sampling period is:

$$T_{s\_p} = 0.5 \quad \text{Eqn. 3-10}$$

Because the given filter equation is as described in Equation 3-3, the Butterworth low-pass filter coefficients are calculated as follows:

$$b_1 = \frac{\omega_{cd\_p} T_{s\_p}}{2 + \omega_{cd\_p} T_{s\_p}} \quad \text{Eqn. 3-11}$$

$$b_2 = \frac{\omega_{cd\_p} T_{s\_p}}{2 + \omega_{cd\_p} T_{s\_p}} \quad \text{Eqn. 3-12}$$

$$a_2 = \frac{\omega_{cd\_p} T_{s\_p} - 2}{2 + \omega_{cd\_p} T_{s\_p}} \quad \text{Eqn. 3-13}$$

A similar approach is adopted for a high-pass filter. Transferring the prototype described in Equation 3-5 into a denormalized high-pass filter results in this transfer function:

$$H(s) = \frac{s}{s + \omega_c} \quad \text{Eqn. 3-14}$$

Discretization of the analog filter given in Equation 3-14 by the bilinear transformation, with pre-warping the results is in the following transfer function:

$$H(z) = \frac{\dfrac{2}{2 + \omega_{cd\_p} T_{s\_p}} + \dfrac{-2}{2 + \omega_{cd\_p} T_{s\_p}} z^{-1}}{1 + \dfrac{\omega_{cd\_p} T_{s\_p} - 2}{2 + \omega_{cd\_p} T_{s\_p}} z^{-1}} \quad \text{Eqn. 3-15}$$

Because the given filter equation is as described in Equation 3-3, the Butterworth high-pass filter coefficients are calculated as follows:

$$b_1 = \frac{2}{2 + \omega_{cd\_p} T_{s\_p}}$$ **Eqn. 3-16**

$$b_2 = \frac{-2}{2 + \omega_{cd\_p} T_{s\_p}}$$ **Eqn. 3-17**

$$a_2 = \frac{\omega_{cd\_p} T_{s\_p} - 2}{2 + \omega_{cd\_p} T_{s\_p}}$$ **Eqn. 3-18**

3.3.7 Returns

The function returns the filtered value of the input f16In in the step k, and stores the input and the output values in the step k into the filter buffer.

3.3.8 Range Issues

The filter coefficients must be defined prior to this function call. All filter coefficients must be divided by 2.0 in order to avoid saturation during the filter calculation. Therefore in order to achieve the correct functionality, the filter output is multiplied by two. This is done automatically within the function.

3.3.9 Special Issues

The function **GDFLIB_FilterIIR1** requires the saturation mode to be turned off.

3.3.10 Implementation

The **GDFLIB_FilterIIR1** function is implemented as a function call.

Example 3-2. Implementation Code

```
#include "gdflib.h"

static Frac16 mf16Value;
static Frac16 mf16FilteredValue;
static GDFLIB_FILTER_IIR1_T mudtFilterIIR1 = GDFLIB_FILTER_IIR1_DEFAULT;


void Isr(void);

void main(void)
{
```

```
        /* LPF 1st order butterworth 100Hz, Ts = 100us*/
        mudtFilterIIR1.udtFiltCoeff.f16B1 = FRAC16(0.0305 / (2.0));
        mudtFilterIIR1.udtFiltCoeff.f16B2 = FRAC16(0.0305 / (2.0));
        mudtFilterIIR1.udtFiltCoeff.f16A2 = FRAC16(-0.9391 / (2.0));

        /* Filter initialization */
        GDFLIB_FilterIIR1Init(&mudtFilterIIR1);
}

/* Periodical function or interrupt */
void Isr(void)
{
        /* Filter calculation */
        mf16FilteredValue = GDFLIB_FilterIIR1(mf16Value,
&mudtFilterIIR1);
}
```

3.3.11 Performance

This section specifies actual requirements of the function or macro in terms of required code memory, data memory, and number of clock cycles to execute.

Table 3-9. Performance of the GDFLIB_FilterIIR1 Function

Code Size (words)	V2: 17, V3: 14	
Data Size (words)	0	
Execution Clock	Min	V2: 42, V3: 38 cycles
	Max	V2: 42, V3: 38 cycles

3.4 GDFLIB_FilterIIR2Init

The function initializes internal variables of a second order IIR filter.

3.4.1 Synopsis

```
#include "gdflib.h"
void GDFLIB_FilterIIR2Init(GDFLIB_FILTER_IIR2_T *pudtFilter)
```

3.4.2 Prototype

```
void GDFLIB_FilterIIR2InitFC(GDFLIB_FILTER_IIR2_T * const pudtFilter)
```

3.4.3 Arguments

This subsection describes input/output arguments to a function or a macro. It explains algorithms being used by functions or macro.

Table 3-10. Function Arguments

Name	In/Out	Format	Range	Description
*pudtFilter	in/out	N/A	N/A	Pointer to a filter structure, which contains filter coefficients and filter buffer; the GDFLIB_FILTER_IIR2_T data type is defined in header file GDFLIB_FilterIIRasm.h

Table 3-11. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
GDFLIB_FILTER_IIR2_T	f32FiltBufferY[2]	In/Out	SF32	0x80000000... 0x7FFFFFFF	Filter buffer storing output values
	f16FiltBufferX[2]	In/Out	SF16	0x8000... 0x7FFF	Filter buffer storing input values
	udtFiltCoeff	In	N/A	N/A	Structure containing filter coefficients

Table 3-12. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
GDFLIB_FILTER_IIR_COEFF2_T	f16B1	in	SF16	0x8000... 0x7FFF	B1 coefficient of the filter
	f16B2	in	SF16	0x8000... 0x7FFF	B2 coefficient of the filter
	f16A2	in	SF16	0x8000... 0x7FFF	A2 coefficient of the filter
	f16B3	in	SF16	0x8000... 0x7FFF	B3 coefficient of the filter
	f16A3	in	SF16	0x8000... 0x7FFF	A3 coefficient of the filter

3.4.4 Availability

This library module is available in the ANSI C version format.

This library module is targeted for the 56800E and 56800Ex platforms.

3.4.5 Dependencies

The dependent files are:

- GDFLIB_FilterIIRasm.h
- GDFLIB_types.h

3.4.6 Description

The **GDFLIB_FilterIIR2Init** function initializes the buffer and coefficients of a second order IIR filter. This function is called once, during variable initialization and since it clears the filter buffer it must not be called together with the filter calculation function.

3.4.7 Returns

The function initializes the filter structure pointed to by the pudtFilter pointer.

3.4.8 Range Issues

The filter coefficients must be defined prior to this function call. If Matlab filter design toolbox is used for the filter coefficients calculation then all calculated coefficients must be divided by 2.0 to avoid saturation during filter calculation.

3.4.9 Implementation

This optional subsection specifies whether call into function generates library function call or macro expansion. This subsection also consist of one or more examples of the use of the function. The examples are often fragments of code (not completed programs) for illustration purposes.

Example 3-3. Implementation Code

```
#include "gdflib.h"

static Frac16 mf16Value;
static Frac16 mf16FilteredValue;
static GDFLIB_FILTER_IIR2_T mudtFilterIIR2 = GDFLIB_FILTER_IIR2_DEFAULT;


void Isr(void);

void main(void)
{
        /* BPF Butterworth approximation fc=500Hz, bw=225Hz Ts = 100us
*/
        mudtFilterIIR2.udtFiltCoeff.f16B1= FRAC16(0.06612 / (2.0));
        mudtFilterIIR2.udtFiltCoeff.f16B2= FRAC16(0.0 / (2.0));
        mudtFilterIIR2.udtFiltCoeff.f16B3= FRAC16(-0.06612 / (2.0));
        mudtFilterIIR2.udtFiltCoeff.f16A2= FRAC16(-1.7762 / (2.0));
        mudtFilterIIR2.udtFiltCoeff.f16A3= FRAC16(0.8678 / (2.0));

        /* Filter initialization */
        GDFLIB_FilterIIR2Init(&mudtFilterIIR2);
}

/* Periodical function or interrupt at 100us*/
void Isr(void)
{
        /* Filter calculation */
        mf16FilteredValue = GDFLIB_FilterIIR2(mf16Value,
&mudtFilterIIR2);
}
```

3.4.10 Performance

This section specifies actual requirements of the function or macro in terms of required code memory, data memory and number of clock cycles to execute.

Table 3-13. Performance of GDFLIB_FilterIIR2Init Function

Code Size (words)	8	
Data Size (words)	0	
Execution Clock	Min	N/A cycles
	Max	N/A cycles

3.5 GDFLIB_FilterIIR2

The function calculates the second order Direct Form 1 IIR filter.

3.5.1 Synopsis

```
#include "gdflib.h"
Frac16 GDFLIB_FilterIIR2(Frac16 f16In, GDFLIB_FILTER_IIR2_T *pudtFilter)
```

3.5.2 Prototype

```
asm Frac16 GDFLIB_FilterIIR2FAsm(Frac16 f16In, GDFLIB_FILTER_IIR2_T *
const pudtFilter)
```

V3 core version:

```
asm Frac16 GDFLIB_V3FilterIIR2FAsm(Frac16 f16In, GDFLIB_FILTER_IIR2_T *
const pudtFilter)
```

3.5.3 Arguments

This subsection describes input/output arguments to a function or a macro. It explains algorithms being used by functions or macro.

Table 3-14. Function Arguments

Name	In/Out	Format	Range	Description
f16In	In	SF16	0x8000... 0x7FFF	Input signal to be filtered
*pudtFilter	In/Out	N/A	N/A	Pointer to a filter structure, which contains filter coefficients and filter buffer; the GDFLIB_FILTER_IIR2_T data type is defined in header file GDFLIB_FilterIIRasm.h

Table 3-15. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
GDFLIB_FILTER_IIR2_T	f32FiltBufferY[2]	In/Out	SF32	0x80000000... 0x7FFFFFFF	Filter buffer storing output values
	f16FiltBufferX[2]	In/Out	SF16	0x8000... 0x7FFF	Filter buffer storing input values
	udtFiltCoeff	In	N/A	N/A	Structure containing filter coefficients

Table 3-16. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
GDFLIB_FILTER_IIR_COEFF2_T	f16B1	In	SF16	0x8000... 0x7FFF	b1 coefficient of the filter
	f16B2	In	SF16	0x8000... 0x7FFF	b2 coefficient of the filter
	f16A2	In	SF16	0x8000... 0x7FFF	a2 coefficient of the filter
	f16B3	In	SF16	0x8000... 0x7FFF	b3 coefficient of the filter
	f16A3	In	SF16	0x8000... 0x7FFF	a3 coefficient of the filter

3.5.4 Availability

This library module is available in the C-callable interface assembly version formats.

This library module is targeted for the DSC 56F80xx platform.

3.5.5 Dependencies

The dependent files are:

- GDFLIB_FilterIIRasm.h
- GDFLIB_types.h

3.5.6 Description

The **GDFLIB_FilterIIR2** function calculates the second order infinite impulse response (IIR) filter. IIR filters are also called recursive filters because the input and the previously calculated output values are used for calculation. This form of feedback enables transfer of the energy from the output to the input, which theoretically leads to an infinitely long impulse response (IIR).

General form of the IIR filter expressed as a transfer function in the Z-domain is described as follows:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_1 + b_2 z^{-1} + b_3 z^{-2} + \ldots + b_{(N+1)} z^{-N}}{1 + a_2 z^{-1} + a_3 z^{-2} + \ldots + a_{(N+1)} z^{-N}}$$

Eqn. 3-19

where N denotes the filter order. The second order IIR filter in the Z-domain is therefore given as:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_1 + b_2 z^{-1} + b_3 z^{-2}}{1 + a_2 z^{-1} + a_3 z^{-2}} \qquad \textbf{Eqn. 3-20}$$

which is transformed into the time domain difference equation as:

$$y(k) = b_1 x(k) + b_2 x(k-1) + b_3 x(k-2) - a_2 y(k-1) - a_3 y(k-2) \qquad \textbf{Eqn. 3-21}$$

The filter difference equation is implemented in Digital Signal Controller directly as written in Equation 3-21. This represents a Direct-Form 1 second order IIR filter as depicted in Figure 3-2.

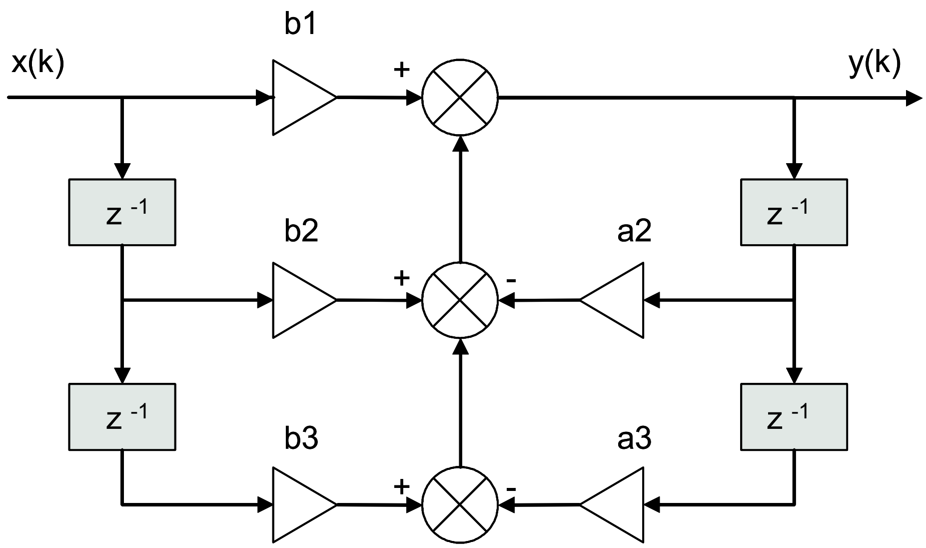

Figure 3-2. Direct Form 1 Second Order IIR filter

The coefficients of the filter depicted in Figure 3-2 can be designed to meet the requirements for the Band Pass (BPF) or the Band Stop filter (BSF). Although the filter is implemented as a second order filter, it is not recommended to use this implementation for the second order LPF or HPF due to the coefficient quantization error. This error arises because of the finite precision arithmetic used for the filter implementation. A higher order LPF or HPF can be obtained by connecting a number of the first order filters in series. The number of the connections gives the order of the resulting filter.

Filter coefficients are calculated using the Butterworth approximation. Butterworth normalized transfer function in ‘s’-plane is given as:

$$H_N(s) = \frac{1}{\prod_{k=1}^{n}(s - s_k)}$$ **Eqn. 3-22**

where

$$s_k = \begin{cases} e^{j(2k-1)\pi/2n} & \text{for even n} \\ e^{j(k-1)\pi/n} & \text{for odd n} \end{cases}$$ **Eqn. 3-23**

The normalized Butterworth second order low pass filter prototype is therefore given as:

$$H(s) = \frac{1}{s^2 + \sqrt{2}s + 1}$$ **Eqn. 3-24**

Transferring the prototype described in Equation 3-24 into a denormalized band-pass filter results in a transfer function:

$$H(s) = \frac{s\omega_{bw}}{s^2 + s\omega_{bw} + \omega_c^2}$$ **Eqn. 3-25**

which is a transfer function of Butterworth Band Pass Filter in ‘s’-domain with center frequency given by ω_c and bandwidth given by ω_{bw}. For the BPF center frequency and bandwidth relation, refer to the filter bode plot depicted on Figure 3-3.

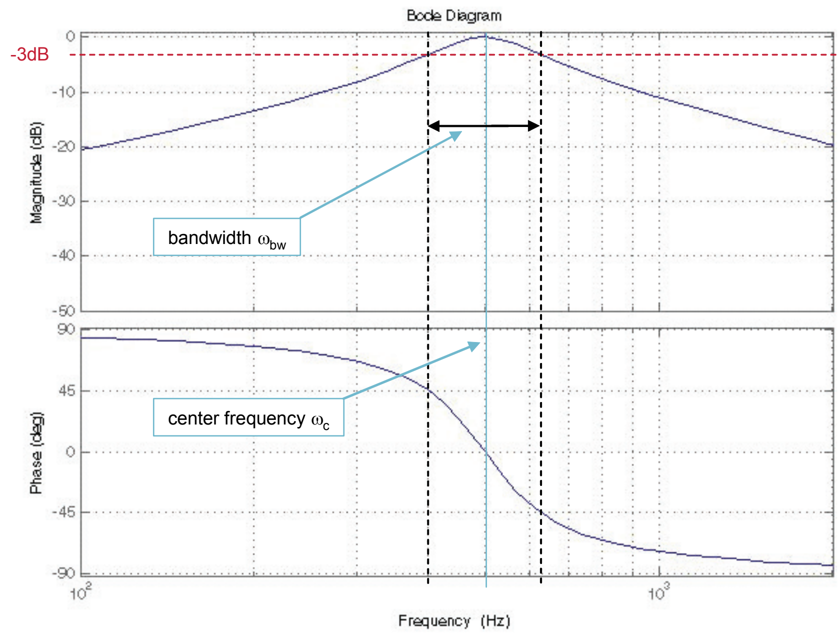

Figure 3-3. BPF Bode Plot (f_c=500Hz, f_{bw}=225Hz)

Transformation of an analog filter described by Equation 3-25 into a discrete form is done using Bilinear transformation, which results in the following transfer function:

$$H(z)=\frac{\frac{2T_s\omega_{bwd}}{C}+\frac{-2T_s\omega_{bwd}}{C}z^{-2}}{1+\frac{2T_s^2\omega_{cd}^2-8}{C}z^{-1}+\frac{4-2T_s\omega_{bwd}+T_s^2\omega_{cd}^2}{C}z^{-2}} \qquad \textbf{Eqn. 3-26}$$

$$C=4+2T_s\omega_{bwd}+T_s^2\omega_{cd}^2 \qquad \textbf{Eqn. 3-27}$$

where ω_{cd} is the center frequency, ω_{bw} is the bandwidth of the filter in the digital domain and T_s is the sampling period. However, mapping of the analog system into a digital domain using Bilinear transformation makes the relation between the analog and digital frequencies nonlinear. This introduces a distortion in the frequency scale of the digital filter relative to that of the analog filter, which is known as a warping effect. The warping effect can by eliminated by prewarping the analog filter and then transforming the prewarped transfer function into the digital domain. This results in the transfer function described as:

$$H(z) = \frac{\frac{2T_{s\_p}\omega_{bwd\_p}}{C} + \frac{-2T_{s\_p}\omega_{bwd\_p}}{C}z^{-2}}{1 + \frac{2T_{s\_p}^2\omega_{cd\_p}^2 - 8}{C}z^{-1} + \frac{4 - 2T_{s\_p}\omega_{bwd\_p} + T_{s\_p}^2\omega_{cd\_p}^2}{C}z^{-2}} \qquad \text{Eqn. 3-28}$$

$$C = 4 + 2T_{s\_p}\omega_{bwd\_p} + T_{s\_p}^2\omega_{cd\_p}^2 \qquad \text{Eqn. 3-29}$$

where ω_{cd-p} is the prewarped center frequency, ω_{bwd-p} is the prewarped bandwidth of the filter in digital domain and T_{s-p} is the prewarped sampling period. Prewarped center frequency is calculated as:

$$\omega_{cd\_p} = \frac{2}{T_{s\_p}}\tan\left(\frac{\omega_{cd}T_s}{2}\right) \qquad \text{Eqn. 3-30}$$

prewarped bandwidth

$$\omega_{bwd\_p} = \frac{2}{T_{s\_p}}\tan\left(\frac{\omega_{bwd}T_s}{2}\right) \qquad \text{Eqn. 3-31}$$

and prewarped sampling period:

$$T_{s\_p} = 0.5 \qquad \text{Eqn. 3-32}$$

Therefore given the filter equation Equation 3-21, the Butterworth band-pass filter coefficients are calculated as follows:

$$b_1 = \frac{2T_{s\_p}\omega_{bwd\_p}}{4 + 2T_{s\_p}\omega_{bwd\_p} + T_{s\_p}^2\omega_{cd\_p}^2} \qquad \text{Eqn. 3-33}$$

$$b_2 = 0 \qquad \text{Eqn. 3-34}$$

$$b_3 = \frac{-2T_{s\_p}\omega_{bwd\_p}}{4 + 2T_{s\_p}\omega_{bwd\_p} + T_{s\_p}^2\omega_{cd\_p}^2} \qquad \text{Eqn. 3-35}$$

$$a_2 = \frac{2T_{s\_p}^2 \omega_{cd\_p}^2 - 8}{4 + 2T_{s\_p}\omega_{bwd\_p} + T_{s\_p}^2 \omega_{cd\_p}^2} \qquad \textbf{Eqn. 3-36}$$

$$a_3 = \frac{4 - 2T_{s\_p}\omega_{bwd\_p} + T_{s\_p}^2 \omega_{cd\_p}^2}{4 + 2T_{s\_p}\omega_{bwd\_p} + T_{s\_p}^2 \omega_{cd\_p}^2} \qquad \textbf{Eqn. 3-37}$$

A similar approach is adopted for a bandstop filter. Transferring the normalized low pass filter prototype described in Equation 3-24 into a denormalized bandstop filter results in a transfer function:

$$H(s) = \frac{s^2 + \omega_c^2}{s^2 + \omega_{bw} s + \omega_c^2} \qquad \textbf{Eqn. 3-38}$$

Discretization of the analog filter given in Equation 3-38 by Bilinear transformation with prewarping results in a following transfer function:

$$H(z) = \frac{\frac{4 + \omega_{cd\_p}^2 T_{s\_p}^2}{C} + \frac{2\omega_{cd\_p}^2 T_{s\_p}^2 - 8}{C} z^{-1} + \frac{4 + \omega_{cd\_p}^2 T_{s\_p}^2}{C} z^{-2}}{1 + \frac{2\omega_{cd\_p}^2 T_{s\_p}^2 - 8}{C} z^{-1} + \frac{4 - 2\omega_{bwd\_p} T_{s\_p} + \omega_{cd\_p}^2 T_{s\_p}^2}{C} z^{-2}} \qquad \textbf{Eqn. 3-39}$$

where

$$C = 4 + 2T_{s\_p}\omega_{bwd\_p} + \omega_{cd\_p}^2 T_{s\_p}^2 \qquad \textbf{Eqn. 3-40}$$

For the BSF center frequency and bandwidth relation, refer to the filter bode plot depicted on Figure 3-4

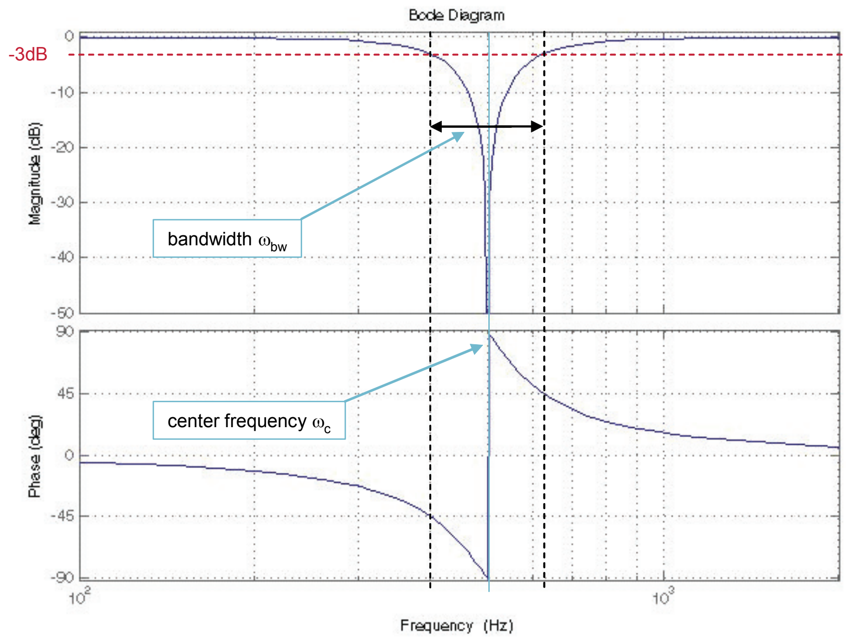

Figure 3-4. BSF Bode Plot(f_c=500Hz, f_{bw}=225Hz)

Therefore given the filter equation as described in Equation 3-21, the Butterworth bandstop filter coefficients are calculated as follows:

$$b_1 = \frac{4 + \omega_{cd\_p}^2 T_{s\_p}^2}{4 + 2T_{s\_p}\omega_{bwd\_p} + \omega_{cd\_p}^2 T_{s\_p}^2}$$ **Eqn. 3-41**

$$b_2 = \frac{2\omega_{cd\_p}^2 T_{s\_p}^2 - 8}{4 + 2T_{s\_p}\omega_{bwd\_p} + \omega_{cd\_p}^2 T_{s\_p}^2}$$ **Eqn. 3-42**

$$b_3 = \frac{4 + \omega_{cd\_p}^2 T_{s\_p}^2}{4 + 2T_{s\_p}\omega_{bwd\_p} + \omega_{cd\_p}^2 T_{s\_p}^2}$$ **Eqn. 3-43**

$$a_2 = \frac{2\omega_{cd\_p}^2 T_{s\_p}^2 - 8}{4 + 2T_{s\_p}\omega_{bwd\_p} + \omega_{cd\_p}^2 T_{s\_p}^2}$$ **Eqn. 3-44**

$$a_3 = \frac{4 - 2T_{s\_p}\omega_{bwd\_p} + \omega_{cd\_p}^2 T_{s\_p}^2}{4 + 2T_{s\_p}\omega_{bwd\_p} + \omega_{cd\_p}^2 T_{s\_p}^2} \quad \text{Eqn. 3-45}$$

To avoid saturation, all the filter coefficients given in Equation 3-33 - Equation 3-37 and Equation 3-41 - Equation 3-45 must be divided by two to be able to implement it on the embedded side. Moreover the coefficients are implemented as 16-bit numbers, therefore the coefficients calculated as fractional numbers must be transformed into integer numbers (to be used on the target platform). The transformations are given as follows:

$$B_1 = \frac{b_1}{2} * 2^{15} \quad \text{Eqn. 3-46}$$

$$B_2 = 0 \quad \text{Eqn. 3-47}$$

$$B_3 = \frac{b_3}{2} * 2^{15} \quad \text{Eqn. 3-48}$$

$$A_2 = \frac{a_2}{2} * 2^{15} \quad \text{Eqn. 3-49}$$

$$A_3 = \frac{a_3}{2} * 2^{15} \quad \text{Eqn. 3-50}$$

3.5.7 Returns

The function returns filtered value of input f16In in the step k and stores the input and output values in the step k into the filter buffer.

3.5.8 Range Issues

The filter coefficients must be defined prior to this function call. All filter coefficients must be divided by 2.0 to avoid saturation during filter calculation. Therefore, to achieve correct functionality, filter output is multiplied by two. This is done automatically within the function.

3.5.9 Special Issues

The function **GDFLIB_FilterIIR2** requires the saturation mode to be turned off.

3.5.10 Implementation

This optional subsection specifies whether call into function generates library function call or macro expansion. This subsection also consist of one or more examples of the use of the function.

Example 3-4. Implementation Code

```
#include "gdflib.h"

static Frac16 mf16Value;
static Frac16 mf16FilteredValue;
static GDFLIB_FILTER_IIR2_T mudtFilterIIR2 = GDFLIB_FILTER_IIR2_DEFAULT;


void Isr(void);

void main(void)
{
        /* BPF Butterworth approximation fc=500Hz, bw=225Hz Ts = 100us
*/
        mudtFilterIIR2.udtFiltCoeff.f16B1= FRAC16(0.06612 / (2.0));
        mudtFilterIIR2.udtFiltCoeff.f16B2= FRAC16(0.0 / (2.0));
        mudtFilterIIR2.udtFiltCoeff.f16B3= FRAC16(-0.06612 / (2.0));
        mudtFilterIIR2.udtFiltCoeff.f16A2= FRAC16(-1.7762 / (2.0));
        mudtFilterIIR2.udtFiltCoeff.f16A3= FRAC16(0.8678 / (2.0));

        /* Filter initialization */
        GDFLIB_FilterIIR2Init(&mudtFilterIIR2);
}

/* Periodical function or interrupt at 100us*/
void Isr(void)
{
        /* Filter calculation */
        mf16FilteredValue = GDFLIB_FilterIIR2(mf16Value,
&mudtFilterIIR2);
}
```

3.5.11 Performance

This section specifies actual requirements of the function or macro in terms of required code memory, data memory and number of clock cycles to execute.

Table 3-17. Performance of GDFLIB_FilterIIR2 function

Code Size (words)	V2: 30, V3: 25	
Data Size (words)	0	
Execution Clock	Min	V2: 58, V3: 54 cycles
	Max	V2: 58, V3: 54 cycles

3.6 GDFLIB_FilterIIR3Init

The function initializes internal variables of a third order IIR filter.

3.6.1 Synopsis

```
#include "gdflib.h"
void GDFLIB_FilterIIR3Init(GDFLIB_FILTER_IIR3_T *pudtFilter)
```

3.6.2 Prototype

```
void GDFLIB_FilterIIR2InitFC(GDFLIB_FILTER_IIR3_T * const pudtFilter)
```

3.6.3 Arguments

This subsection describes input/output arguments to a function or a macro. It explains algorithms being used by functions or macro.

Table 3-18. Function Arguments

Name	In/Out	Format	Range	Description
*pudtFilter	in/out	N/A	N/A	Pointer to a filter structure, which contains filter coefficients and filter buffer; the GDFLIB_FILTER_IIR3_T data type is defined in header file GDFLIB_FilterIIRasm.h

Table 3-19. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
GDFLIB_FILTER_IIR3_T	f32FiltBufferY[3]	In/Out	SF32	0x80000000... 0x7FFFFFFF	Filter buffer storing output values
	f16FiltBufferX[3]	In/Out	SF16	0x8000... 0x7FFF	Filter buffer storing input values
	udtFiltCoeff	In	N/A	N/A	Structure containing filter coefficients

Table 3-20. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
GDFLIB_FILTER_IIR_COEFF3_T	f16B1	In	SF16	0x8000... 0x7FFF	b1 coefficient of the filter
	f16B2	In	SF16	0x8000... 0x7FFF	b2 coefficient of the filter
	f16A2	In	SF16	0x8000... 0x7FFF	a2 coefficient of the filter
	f16B3	In	SF16	0x8000... 0x7FFF	b3 coefficient of the filter
	f16A3	In	SF16	0x8000... 0x7FFF	a3 coefficient of the filter
	f16B4	In	SF16	0x8000... 0x7FFF	b4 coefficient of the filter
	f16A4	In	SF16	0x8000... 0x7FFF	a4 coefficient of the filter

3.6.4 Availability

This library module is available in the ANSI C version format.

This library module is targeted for the 56800E and 56800Ex platforms.

3.6.5 Dependencies

The dependent files are:

- GDFLIB_FilterIIRasm.h
- GDFLIB_types.h

3.6.6 Description

The **GDFLIB_FilterIIR3Init** function initializes the buffer and coefficients of a second order IIR filter. This function is called once, during variable initialization and since it clears the filter buffer it must not be called together with the filter calculation function.

3.6.7 Returns

The function initializes the filter structure pointed to by the pudtFilter pointer.

3.6.8 Range Issues

The filter coefficients must be defined prior to this function call. If Matlab filter design toolbox is used for the filter coefficients calculation then all calculated coefficients must be divided by 4.0 to avoid saturation during filter calculation.

3.6.9 Implementation

This optional subsection specifies whether call into function generates library function call or macro expansion. This subsection also consist of one or more examples of the use of the function. The examples are often fragments of code (not completed programs) for illustration purposes.

Example 3-5. Implementation Code

```
#include "gdflib.h"

static Frac16 mf16Value;
static Frac16 mf16FilteredValue;
static GDFLIB_FILTER_IIR3_T mudtFilterIIR3 = GDFLIB_FILTER_IIR3_DEFAULT;


void Isr(void);

void main(void)
{
        /* HPF Butterworth approximation fc=2kHz, Ts = 100us*/

        mudtFilterIIR3.udtFiltCoeff.f16B1= FRAC16(0.2569 / (4.0));
        mudtFilterIIR3.udtFiltCoeff.f16B2= FRAC16(-0.7707 / (4.0));
        mudtFilterIIR3.udtFiltCoeff.f16B3= FRAC16(0.7707 / (4.0));
        mudtFilterIIR3.udtFiltCoeff.f16B4= FRAC16(-0.2569 / (4.0));
        mudtFilterIIR3.udtFiltCoeff.f16A2= FRAC16(-0.5772 / (4.0));
        mudtFilterIIR3.udtFiltCoeff.f16A3= FRAC16(0.4218 / (4.0));
        mudtFilterIIR3.udtFiltCoeff.f16A4= FRAC16(-0.0563 / (4.0));

        /* Filter initialization */
        GDFLIB_FilterIIR3Init(&mudtFilterIIR3);
}

/* Periodical function or interrupt at 100us*/
void Isr(void)
{
        /* Filter calculation */
        mf16FilteredValue = GDFLIB_FilterIIR3(mf16Value,
&mudtFilterIIR3);
}
```

3.6.10 Performance

This section specifies actual requirements of the function or macro in terms of required code memory, data memory and number of clock cycles to execute.

Table 3-21. Performance of GDFLIB_FilterIIR3Init Function

Code Size (words)	11 words	
Data Size (words)	0 words	
Execution Clock	Min	N/A cycles
	Max	N/A cycles

3.7 GDFLIB_FilterIIR3

The function calculates the third order Direct Form 1 IIR filter.

3.7.1 Synopsis

```
#include "gdflib.h"
Frac16 GDFLIB_FilterIIR3(Frac16 f16In, GDFLIB_FILTER_IIR3_T *pudtFilter)
```

3.7.2 Prototype

```
asm Frac16 GDFLIB_FilterIIR3FAsm(Frac16 f16In, GDFLIB_FILTER_IIR3_T *
const pudtFilter)
```

V3 core version:

```
asm Frac16 GDFLIB_V3FilterIIR3FAsm(Frac16 f16In, GDFLIB_FILTER_IIR3_T *
const pudtFilter)
```

3.7.3 Arguments

This subsection describes input/output arguments to a function or a macro. It explains algorithms being used by functions or macro.

Table 3-22. Function Arguments

Name	In/Out	Format	Range	Description
f16In	In	SF16	0x8000... 0x7FFF	Input signal to be filtered
*pudtFilter	In/Out	N/A	N/A	Pointer to a filter structure, which contains filter coefficients and filter buffer; the GDFLIB_FILTER_IIR3_T data type is defined in header file GDFLIB_FilterIIRasm.h

Table 3-23. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
GDFLIB_FILTER_IIR3_T	f32FiltBufferY[3]	In/Out	SF32	0x80000000... 0x7FFFFFFF	Filter buffer storing output values
	f16FiltBufferX[3]	In/Out	SF16	0x8000... 0x7FFF	Filter buffer storing input values
	udtFiltCoeff	In	N/A	N/A	Structure containing filter coefficients

Table 3-24. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
GDFLIB_FILTER_IIR_COEFF3_T	f16B1	In	SF16	0x8000... 0x7FFF	b1 coefficient of the filter
	f16B2	In	SF16	0x8000... 0x7FFF	b2 coefficient of the filter
	f16A2	In	SF16	0x8000... 0x7FFF	a2 coefficient of the filter
	f16B3	In	SF16	0x8000... 0x7FFF	b3 coefficient of the filter
	f16A3	In	SF16	0x8000... 0x7FFF	a3 coefficient of the filter
	f16B4	In	SF16	0x8000... 0x7FFF	b4 coefficient of the filter
	f16A4	In	SF16	0x8000... 0x7FFF	a4 coefficient of the filter

3.7.4 Availability

This library module is available in the C-callable interface assembly version formats.

This library module is targeted for the 56800E and 56800Ex platforms.

3.7.5 Dependencies

The dependent files are:

- GDFLIB_FilterIIRasm.h
- GDFLIB_types.h

3.7.6 Description

The **GDFLIB_FilterIIR3** function calculates the third order infinite impulse response (IIR) filter. IIR filters are also called recursive filters because the input and the previously calculated output values are used for calculation. This form of feedback enables transfer of the energy from the output to the input, which theoretically leads to an infinitely long impulse response (IIR).

General form of the IIR filter expressed as a transfer function in the Z-domain is described as follows:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_1 + b_2 z^{-1} + b_3 z^{-2} + \ldots + b_{(N+1)} z^{-N}}{1 + a_2 z^{-1} + a_3 z^{-2} + \ldots + a_{(N+1)} z^{-N}}$$

Eqn. 3-51

where N denotes the filter order. The second order IIR filter in the Z-domain is therefore given as:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_1 + b_2 z^{-1} + b_3 z^{-2} + b_4 z^{-3}}{1 + a_2 z^{-1} + a_3 z^{-2} + a_4 z^{-3}}$$

Eqn. 3-52

which is transformed into the time domain difference equation as:

$$y(k) = b_1 x(k) + b_2 x(k-1) + b_3 x(k-2) + b_4 x(k-3) - a_2 y(k-2) - a_3 y(k-3) - a_4 y(k-4)$$

Eqn. 3-53

The filter difference equation is implemented in Digital Signal Controller directly as written in Equation . This represents a Direct-Form 1 third order IIR filter as depicted in Figure 3-5.

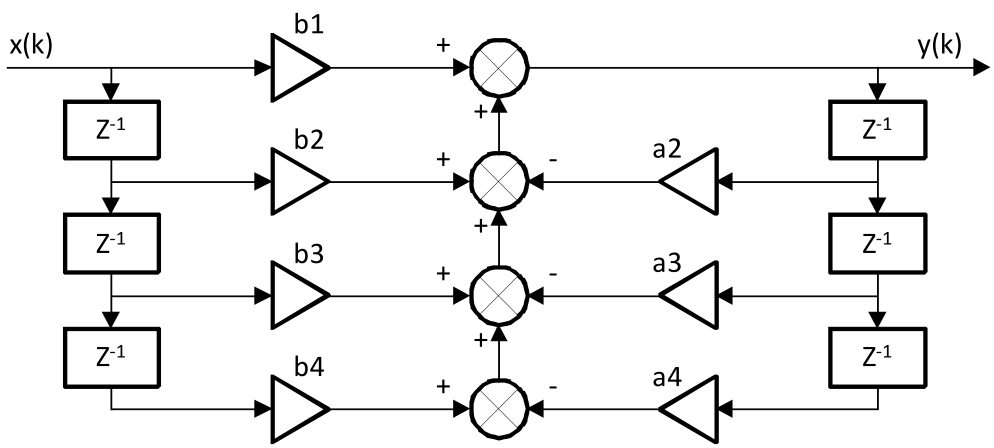

Figure 3-5. Direct Form 1 Third Order IIR filter

The coefficients of the filter depicted in Figure 3-5 can be designed to meet the requirements for the Band Pass (BPF) or the Band Stop filter (BSF).

Filter coefficients are calculated using the Butterworth approximation similarly as at the 1st and 2nd order filter.

3.7.7 Returns

The function returns filtered value of input f16In in the step k and stores the input and output values in the step k into the filter buffer.

3.7.8 Range Issues

The filter coefficients must be defined prior to this function call. All filter coefficients must be divided by 4.0 to avoid saturation during filter calculation. Therefore, to achieve correct functionality, filter output is multiplied by two. This is done automatically within the function.

3.7.9 Special Issues

The function **GDFLIB_FilterIIR3** requires the saturation mode to be turned off.

3.7.10 Implementation

This optional subsection specifies whether call into function generates library function call or macro expansion. This subsection also consist of one or more examples of the use of the function.

The following example is to set up a high-pass filter with the sampling frequency 10kHz, passing frequencies above 2kHz, i.e. the cut-off frequency with 2kHz and 60dB/dec.

Example 3-6. Filter Parameters by Matlab

```
% sampling frequency 10KHz
Ts = 1 / 10000

% cut-off frequency 2KHz, -3dB
Fc = 2000
Rp = 3

% stopped frequency 200Hz, -60dB
Fs = 200
Rs = 60

% checking order of the filter
n = buttord(2 * Ts * Fc, 2 * Ts * Fs, Rp, Rs)

% n = 3, i.e. the filter is achievable with the 3rd order

% getting the filter coefficients
[b, a] = butter(n, 2* Ts * Fc, 'high')

% the coefs are:
% b1 = 0.2569, b2 = -0.7707, b3 = 0.7707, b4 = -0.2569
% a1 = 1.0000, a2 = -0.5772, a3 = 0.4218, a4 = -0.0563
```

The desired filter response plot is visible in Figure 3-6.

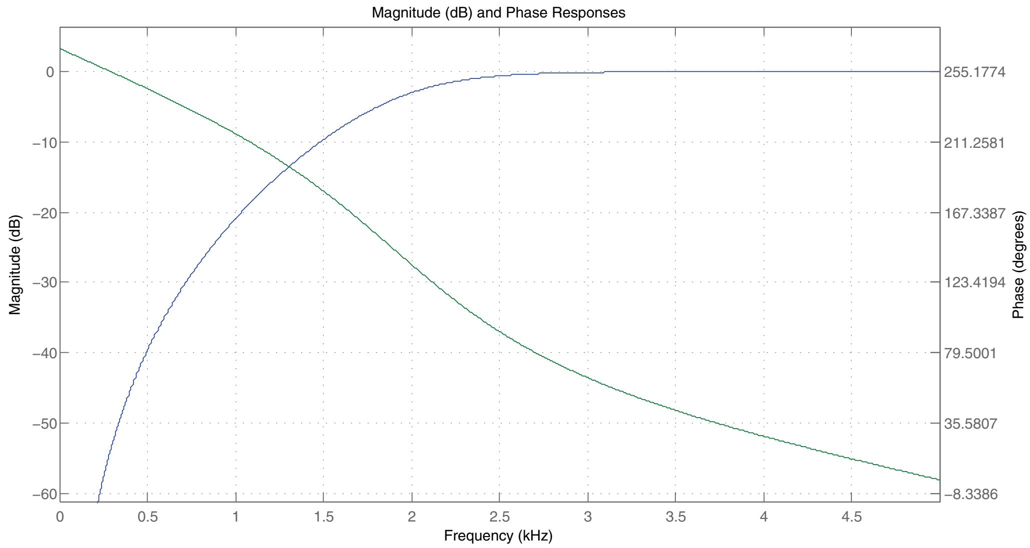

Figure 3-6. Filter Response

Example 3-7. Implementation Code

```
#include "gdflib.h"

static Frac16 mf16Value;
static Frac16 mf16FilteredValue;
static GDFLIB_FILTER_IIR3_T mudtFilterIIR3 = GDFLIB_FILTER_IIR3_DEFAULT;


void Isr(void);

void main(void)
{
        /* HPF Butterworth approximation fc=2kHz, Ts = 100us*/

        mudtFilterIIR3.udtFiltCoeff.f16B1= FRAC16(0.2569 / (4.0));
        mudtFilterIIR3.udtFiltCoeff.f16B2= FRAC16(-0.7707 / (4.0));
        mudtFilterIIR3.udtFiltCoeff.f16B3= FRAC16(0.7707 / (4.0));
        mudtFilterIIR3.udtFiltCoeff.f16B4= FRAC16(-0.2569 / (4.0));
        mudtFilterIIR3.udtFiltCoeff.f16A2= FRAC16(-0.5772 / (4.0));
        mudtFilterIIR3.udtFiltCoeff.f16A3= FRAC16(0.4218 / (4.0));
        mudtFilterIIR3.udtFiltCoeff.f16A4= FRAC16(-0.0563 / (4.0));

        /* Filter initialization */
        GDFLIB_FilterIIR3Init(&mudtFilterIIR3);
}

/* Periodical function or interrupt at 100us*/
void Isr(void)
{
        /* Filter calculation */
        mf16FilteredValue = GDFLIB_FilterIIR3(mf16Value,
&mudtFilterIIR3);
}
```

3.7.11 Performance

This section specifies actual requirements of the function or macro in terms of required code memory, data memory and number of clock cycles to execute.

Table 3-25. Performance of GDFLIB_FilterIIR3 function

Code Size (words)	V2: 44, V3: 37	
Data Size (words)	0	
Execution Clock	Min	V2: 74, V3: 70 cycles
	Max	V2: 74, V3: 70 cycles

3.8 GDFLIB_FilterIIR4Init

The function initializes internal variables of a fourth order IIR filter.

3.8.1 Synopsis

```
#include "gdflib.h"
void GDFLIB_FilterIIR4Init(GDFLIB_FILTER_IIR4_T *pudtFilter)
```

3.8.2 Prototype

```
void GDFLIB_FilterIIR2InitFC(GDFLIB_FILTER_IIR4_T * const pudtFilter)
```

3.8.3 Arguments

This subsection describes input/output arguments to a function or a macro. It explains algorithms being used by functions or macro.

Table 3-26. Function Arguments

Name	In/Out	Format	Range	Description
*pudtFilter	in/out	N/A	N/A	Pointer to a filter structure, which contains filter coefficients and filter buffer; the GDFLIB_FILTER_IIR4_T data type is defined in header file GDFLIB_FilterIIRasm.h

Table 3-27. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
GDFLIB_FILTER_IIR4_T	f32FiltBufferY[4]	In/Out	SF32	0x80000000... 0x7FFFFFFF	Filter buffer storing output values
	f16FiltBufferX[4]	In/Out	SF16	0x8000... 0x7FFF	Filter buffer storing input values
	udtFiltCoeff	In	N/A	N/A	Structure containing filter coefficients

Table 3-28. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
GDFLIB_FILTER_IIR_COEFF4_T	f16B1	In	SF16	0x8000... 0x7FFF	b1 coefficient of the filter
	f16B2	In	SF16	0x8000... 0x7FFF	b2 coefficient of the filter
	f16A2	In	SF16	0x8000... 0x7FFF	a2 coefficient of the filter
	f16B3	In	SF16	0x8000... 0x7FFF	b3 coefficient of the filter
	f16A3	In	SF16	0x8000... 0x7FFF	a3 coefficient of the filter
	f16B4	In	SF16	0x8000... 0x7FFF	b4 coefficient of the filter
	f16A4	In	SF16	0x8000... 0x7FFF	a4 coefficient of the filter
	f16B5	In	SF16	0x8000... 0x7FFF	b5 coefficient of the filter
	f16A5	In	SF16	0x8000... 0x7FFF	a5 coefficient of the filter

3.8.4 Availability

This library module is available in the ANSI C version format.

This library module is targeted for the 56800E and 56800Ex platforms.

3.8.5 Dependencies

The dependent files are:

- GDFLIB_FilterIIRasm.h
- GDFLIB_types.h

3.8.6 Description

The **GDFLIB_FilterIIR4Init** function initializes the buffer and coefficients of a second order IIR filter. This function is called once, during variable initialization and since it clears the filter buffer it must not be called together with the filter calculation function.

3.8.7 Returns

The function initializes the filter structure pointed to by the pudtFilter pointer.

3.8.8 Range Issues

The filter coefficients must be defined prior to this function call. If Matlab filter design toolbox is used for the filter coefficients calculation then all calculated coefficients must be divided by 8.0 to avoid saturation during filter calculation.

3.8.9 Implementation

This optional subsection specifies whether call into function generates library function call or macro expansion. This subsection also consist of one or more examples of the use of the function. The examples are often fragments of code (not completed programs) for illustration purposes.

Example 3-8. Implementation Code

```
#include "gdflib.h"

static Frac16 mf16Value;
static Frac16 mf16FilteredValue;
static GDFLIB_FILTER_IIR4_T mudtFilterIIR4 = GDFLIB_FILTER_IIR4_DEFAULT;


void Isr(void);

void main(void)
{
        /* BPF Butterworth approximation fc=1000Hz, bw=250Hz Ts = 100us
*/
        mudtFilterIIR4.udtFiltCoeff.f16B1= FRAC16(0.0055 / (8.0));
        mudtFilterIIR4.udtFiltCoeff.f16B2= FRAC16(0.0 / (8.0));
        mudtFilterIIR4.udtFiltCoeff.f16B3= FRAC16(-0.0111 / (8.0));
        mudtFilterIIR4.udtFiltCoeff.f16B4= FRAC16(0.0 / (8.0));
        mudtFilterIIR4.udtFiltCoeff.f16B5= FRAC16(0.0055 / (8.0));
        mudtFilterIIR4.udtFiltCoeff.f16A2= FRAC16(-3.0664 / (8.0));
        mudtFilterIIR4.udtFiltCoeff.f16A3= FRAC16(4.1359 / (8.0));
        mudtFilterIIR4.udtFiltCoeff.f16A4= FRAC16(-2.7431 / (8.0));
        mudtFilterIIR4.udtFiltCoeff.f16A5= FRAC16(0.8008 / (8.0));

        /* Filter initialization */
        GDFLIB_FilterIIR4Init(&mudtFilterIIR4);
}

/* Periodical function or interrupt at 100us*/
void Isr(void)
{
        /* Filter calculation */
        mf16FilteredValue = GDFLIB_FilterIIR4(mf16Value,
&mudtFilterIIR4);
}
```

3.8.10 Performance

This section specifies actual requirements of the function or macro in terms of required code memory, data memory and number of clock cycles to execute.

Table 3-29. Performance of GDFLIB_FilterIIR4Init Function

Code Size (words)	14	
Data Size (words)	0	
Execution Clock	Min	N/A cycles
	Max	N/A cycles

3.9 GDFLIB_FilterIIR4

The function calculates the fourth order Direct Form 1 IIR filter.

3.9.1 Synopsis

```
#include ”gdflib.h”
Frac16 GDFLIB_FilterIIR4(Frac16 f16In, GDFLIB_FILTER_IIR4_T *pudtFilter)
```

3.9.2 Prototype

```
asm Frac16 GDFLIB_FilterIIR4FAsm(Frac16 f16In, GDFLIB_FILTER_IIR4_T *
const pudtFilter)
```

V3 core version:

```
asm Frac16 GDFLIB_V3FilterIIR4FAsm(Frac16 f16In, GDFLIB_FILTER_IIR4_T *
const pudtFilter)
```

3.9.3 Arguments

This subsection describes input/output arguments to a function or a macro. It explains algorithms being used by functions or macro.

Table 3-30. Function Arguments

Name	In/Out	Format	Range	Description
f16In	In	SF16	0x8000... 0x7FFF	Input signal to be filtered
*pudtFilter	In/Out	N/A	N/A	Pointer to a filter structure, which contains filter coefficients and filter buffer; the GDFLIB_FILTER_IIR4_T data type is defined in header file GDFLIB_FilterIIRasm.h

Table 3-31. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
GDFLIB_FILTER_IIR4_T	f32FiltBufferY[4]	In/Out	SF32	0x80000000... 0x7FFFFFFF	Filter buffer storing output values
	f16FiltBufferX[4]	In/Out	SF16	0x8000... 0x7FFF	Filter buffer storing input values
	udtFiltCoeff	In	N/A	N/A	Structure containing filter coefficients

Table 3-32. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
GDFLIB_FILTER_IIR_COEFF4_T	f16B1	In	SF16	0x8000... 0x7FFF	b1 coefficient of the filter
	f16B2	In	SF16	0x8000... 0x7FFF	b2 coefficient of the filter
	f16A2	In	SF16	0x8000... 0x7FFF	a2 coefficient of the filter
	f16B3	In	SF16	0x8000... 0x7FFF	b3 coefficient of the filter
	f16A3	In	SF16	0x8000... 0x7FFF	a3 coefficient of the filter
	f16B4	In	SF16	0x8000... 0x7FFF	b4 coefficient of the filter
	f16A4	In	SF16	0x8000... 0x7FFF	a4 coefficient of the filter
	f16B5	In	SF16	0x8000... 0x7FFF	b5 coefficient of the filter
	f16A5	In	SF16	0x8000... 0x7FFF	a5 coefficient of the filter

3.9.4 Availability

This library module is available in the C-callable interface assembly version formats.

This library module is targeted for the 56800E and 56800Ex platforms.

3.9.5 Dependencies

The dependent files are:

- GDFLIB_FilterIIRasm.h
- GDFLIB_types.h

3.9.6 Description

The **GDFLIB_FilterIIR4** function calculates the third order infinite impulse response (IIR) filter. IIR filters are also called recursive filters because the input and the previously calculated output values are used for calculation. This form of feedback enables transfer of the energy from the output to the input, which theoretically leads to an infinitely long impulse response (IIR).

General form of the IIR filter expressed as a transfer function in the Z-domain is described as follows:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_1 + b_2 z^{-1} + b_3 z^{-2} + \ldots + b_{(N+1)} z^{-N}}{1 + a_2 z^{-1} + a_3 z^{-2} + \ldots + a_{(N+1)} z^{-N}} \quad \textbf{Eqn. 3-54}$$

where N denotes the filter order. The second order IIR filter in the Z-domain is therefore given as:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_1 + b_2 z^{-1} + b_3 z^{-2} + b_4 z^{-3} + b_5 z^{-4}}{1 + a_2 z^{-1} + a_3 z^{-2} + a_4 z^{-3} + a_5 z^{-4}} \quad \textbf{Eqn. 3-55}$$

which is transformed into the time domain difference equation as:

$$\begin{aligned} y(k) = {} & b_1 x(k) + b_2 x(k-1) + b_3 x(k-2) + b_4 x(k-3) + b_5 x(k-4) \\ & - a_2 y(k-2) - a_3 y(k-3) - a_4 y(k-4) - a_5 y(k-4) \end{aligned} \quad \textbf{Eqn. 3-56}$$

The filter difference equation is implemented in Digital Signal Controller directly as written in Equation . This represents a Direct-Form 1 fourth order IIR filter as depicted in Figure 3-7.

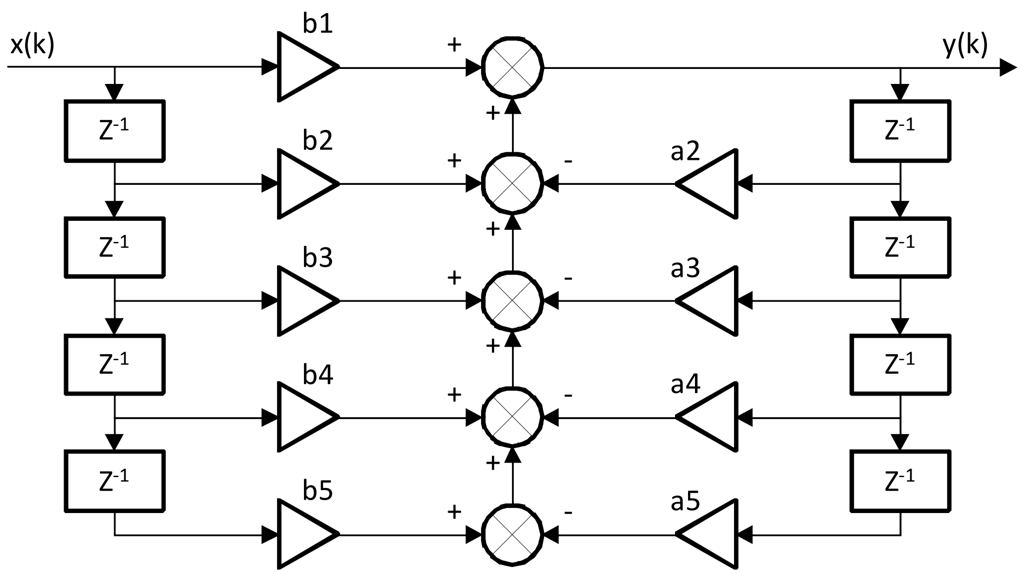

Figure 3-7. Direct Form 1 Fourth Order IIR filter

The coefficients of the filter depicted in Figure 3-7 can be designed to meet the requirements for the Band Pass (BPF) or the Band Stop filter (BSF).

Filter coefficients are calculated using the Butterworth approximation similarly as at the 1st and 2nd order filter.

3.9.7 Returns

The function returns filtered value of input f16In in the step k and stores the input and output values in the step k into the filter buffer.

3.9.8 Range Issues

The filter coefficients must be defined prior to this function call. All filter coefficients must be divided by 8.0 to avoid saturation during filter calculation. Therefore, to achieve correct functionality, filter output is multiplied by two. This is done automatically within the function.

3.9.9 Special Issues

The function **GDFLIB_FilterIIR4** requires the saturation mode to be turned off.

3.9.10 Implementation

This optional subsection specifies whether call into function generates library function call or macro expansion. This subsection also consist of one or more examples of the use of the function.

The following example is to set up a band-pass filter with the sampling frequency 10kHz, passing frequency 1kHz with the bandwidth of 250 Hz, i.e. the frequencies of 875 and 1125Hz will be at -3dB. Roughly at double bandwidth it should have the attenuation of 20dB.

Example 3-9. Filter Parameters by Matlab

```
% sampling frequency 10KHz
Ts = 1 / 10000

% Center frequency 1kHz, -3dB
Fc = 1000
Rp = 3

% Bandwidth 250Hz
Fbw = 250

% stopped frequencies at -20dB
Rs = 20

% checking order of the filter
n = buttord(2 * Ts * [Fc - Fbw /2 Fc + Fbw / 2], 2 * Ts * [Fc - Fbw Fc +
Fbw], Rp, Rs)

% n = 4, i.e. the filter is achievable with the 4th order
```

```
% getting the filter coefficients
[b, a] = butter(n / 2, 2 * Ts * [Fc - Fbw /2 Fc + Fbw / 2])

% the coefs are:
% b1 = 0.0055, b2 = 0.0000, b3 = -0.0111, b4 = 0.0000, b5 = 0.0055
% a1 = 1.0000, a2 = -3.0664, a3 = 4.1359, a4 = -2.7431, a5 = 0.8008
```

The desired filter response plot is visible in Figure 3-8.

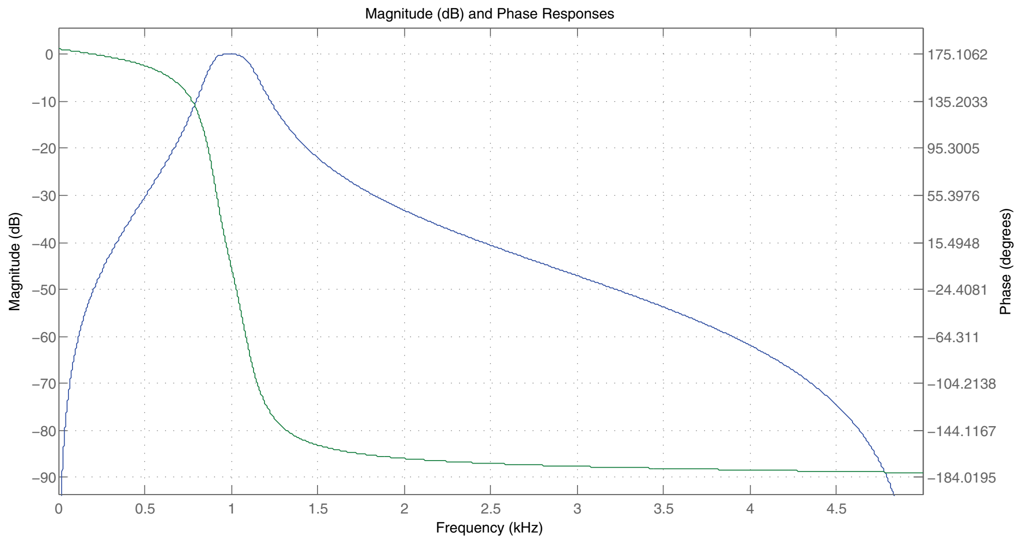

Figure 3-8. Filter Response

Example 3-10. Implementation Code

```
#include "gdflib.h"

static Frac16 mf16Value;
static Frac16 mf16FilteredValue;
static GDFLIB_FILTER_IIR4_T mudtFilterIIR4 = GDFLIB_FILTER_IIR4_DEFAULT;


void Isr(void);

void main(void)
{
        /* BPF Butterworth approximation fc=1000Hz, bw=250Hz Ts = 100us
*/
        mudtFilterIIR4.udtFiltCoeff.f16B1= FRAC16(0.0055 / (8.0));
        mudtFilterIIR4.udtFiltCoeff.f16B2= FRAC16(0.0 / (8.0));
        mudtFilterIIR4.udtFiltCoeff.f16B3= FRAC16(-0.0111 / (8.0));
        mudtFilterIIR4.udtFiltCoeff.f16B4= FRAC16(0.0 / (8.0));
        mudtFilterIIR4.udtFiltCoeff.f16B5= FRAC16(0.0055 / (8.0));
        mudtFilterIIR4.udtFiltCoeff.f16A2= FRAC16(-3.0664 / (8.0));
        mudtFilterIIR4.udtFiltCoeff.f16A3= FRAC16(4.1359 / (8.0));
```

```
        mudtFilterIIR4.udtFiltCoeff.f16A4= FRAC16(-2.7431 / (8.0));
        mudtFilterIIR4.udtFiltCoeff.f16A5= FRAC16(0.8008 / (8.0));

        /* Filter initialization */
        GDFLIB_FilterIIR4Init(&mudtFilterIIR4);
}

/* Periodical function or interrupt at 100us*/
void Isr(void)
{
        /* Filter calculation */
        mf16FilteredValue = GDFLIB_FilterIIR4(mf16Value,
&mudtFilterIIR4);
}
```

3.9.11 Performance

This section specifies actual requirements of the function or macro in terms of required code memory, data memory and number of clock cycles to execute.

Table 3-33. Performance of GDFLIB_FilterIIR4 function

Code Size (words)	V2: 55, V3: 46	
Data Size (words)	0	
Execution Clock	Min	V2: 86, V3: 78 cycles
	Max	V2: 86, V3: 78 cycles

3.10 GDFLIB_FilterMA32Init

This function initializes the internal variables of of the **GDFLIB_FilterMA32** function with zero.

3.10.1 Synopsis

```
#include "gdflib.h"
void GDFLIB_FilterMA32Init(GDFLIB_FILTER_MA32_T *pudtFilter)
```

3.10.2 Prototype

```
void GDFLIB_FilterMA32InitFC(GDFLIB_FILTER_MA32_T * const pudtFilter)
```

3.10.3 Arguments

This subsection describes the input/output arguments to a function or a macro. It explains the algorithms being used by the functions or macro.

Table 3-34. Function Arguments

Name	In/ Out	Format	Range	Description
*pudtFilter	in/out	N/A	N/A	Pointer to a filter structure, which contains filter coefficients and filter buffer; the GDFLIB_FILTER_MA32_T data type is defined in header file GDFLIB_FilterMA32asm.h.

Table 3-35. User-Type Definitions

Typedef	Name	In/ Out	Form at	Range	Description
GDFLIB_FILTER_MA32_T	f32Acc	in/ out	SF32	$80000000... $7FFFFFFF	internal filter accumulator
	w16N	in	SI16	$8000... $7FFF	number of filtered points (filter window size)

3.10.4 Availability

This library module is available in the The ANSI C version formats.

This library module is targeted for the 56800E and 56800Ex platforms.

3.10.5 Dependencies

The dependent files are:

- GDFLIB_FilterMA32asm.h
- GDFLIB_types.h

3.10.6 Description

The **GDFLIB_FilterMA32Init** function initializes the accumulator of a moving average filter. This function is called once, during the variable initialization, and since it clears the filter buffer, it must not be called together with the filter calculation function. The size of the filter window (number of filtered points) shall be defined prior to this function call. The number of the filtered points is defined by assigning a value to the pFilter.w16N variable, stored within the filter structure. This number represents the number of filtered points as a power of two, as follows:

$$n_p = 2^{(\mathrm{pudtFilter.w16N})} \quad , \mathrm{pudtFilter.w16N} \geq 0 \qquad \textit{Eqn. 3-57}$$

where n_p is the actual number of filtered points (size of the filter window).

3.10.7 Returns

This function initializes the filter accumulator in the filter structure pointed to by the pudtFilter pointer.

3.10.8 Special Issues

The function **GDFLIB_FilterMA32Init** is the saturation mode independent.

3.10.9 Implementation

The **GDFLIB_FilterMA32Init** function is implemented as a function call.

Example 3-11. Implementation Code

```
#include "gdflib.h"

static GDFLIB_FILTER_MA32_T mudtFilterMA32 = GDFLIB_FILTER_MA32_DEFAULT;
static Frac16 mf16Value;
static Frac16 mf16FilteredValue;

void Isr(void);

void main(void)
{
         /* filter window size 2 ^ 2 = 4 points */
         mudtFilterMA32.w16N = 2;

         /* Filter initialization */
         GDFLIB_FilterMA32Init(&mudtFilterMA32);
}

/* Periodical function or interrupt */
void Isr(void)
{
         /* Filter calculation */
```

```
        mf16FilteredValue = GDFLIB_FilterMA32(mf16Value,
&mudtFilterMA32);
}
```

3.10.10 Performance

This section specifies the actual requirements of the function or macro in terms of required code memory, data memory, and number of clock cycles to execute.

Table 3-36. Performance of the GDFLIB_FilterMA32Init Function

Code Size (bytes)	2	
Data Size (bytes)	0	
Execution Clock	Min	16
	Max	16

3.11 GDFLIB_FilterMA32InitVal

This function initializes the internal variables of the **GDFLIB_FilterMA32** function with a value.

3.11.1 Synopsis

```
#include ”gdflib.h”
void GDFLIB_FilterMA32InitVal(Frac16 f16InitVal, GDFLIB_FILTER_MA32_T
*pudtFilter)
```

3.11.2 Prototype

```
void GDFLIB_FilterMA32InitValFAsm(Frac16 f16InitVal,
GDFLIB_FILTER_MA32_T * const pudtFilter)
```

3.11.3 Arguments

This subsection describes the input/output arguments to a function or a macro. It explains the algorithms being used by the functions or macro.

Table 3-37. Function Arguments

Name	In/ Out	Format	Range	Description
f16InitVal	in	SF16	0x8000... 0x7FFF	Initial value of the filter
*pudtFilter	in/out	N/A	N/A	Pointer to a filter structure, which contains filter coefficients and filter buffer; the GDFLIB_FILTER_MA32_T data type is defined in header file GDFLIB_FilterMA32asm.h.

Table 3-38. User-Type Definitions

Typedef	Name	In/ Out	Form at	Range	Description
GDFLIB_FILTER_MA32_T	f32Acc	in/ out	SF32	$80000000... $7FFFFFFF	internal filter accumulator
	w16N	in	SI16	$8000... $7FFF	number of filtered points (filter window size)

3.11.4 Availability

This library module is available in the C-callable interface assembly format.

This library module is targeted for the 56800E and 56800Ex platforms.

3.11.5 Dependencies

The dependent files are:

- GDFLIB_FilterMA32asm.h
- GDFLIB_types.h

3.11.6 Description

The **GDFLIB_FilterMA32InitVal** function initializes the accumulator of a moving average filter with a predifened value. This function is called onced where the user wants the filter to be initialized. The buffer is set up in the manner the output to be the initial value in the first step.

The size of the filter window (number of filtered points) shall be defined prior to this function call. The number of the filtered points is defined by assigning a value to the pFilter.w16N variable, stored within the filter structure. This number represents the number of filtered points as a power of two, as follows:

$$n_p = 2^{(\mathrm{pudtFilter.w16N})} \quad , \mathrm{pudtFilter.w16N} \geq 0 \qquad \textit{Eqn. 3-58}$$

where n_p is the actual number of filtered points (size of the filter window).

3.11.7 Returns

This function initializes the filter accumulator in the filter structure pointed to by the pudtFilter pointer.

3.11.8 Special Issues

The function **GDFLIB_FilterMA32InitVal** is the saturation mode independent.

3.11.9 Implementation

The **GDFLIB_FilterMA32InitVal** function is implemented as a function call.

Example 3-12. Implementation Code

```
#include "gdflib.h"

static GDFLIB_FILTER_MA32_T mudtFilterMA32 = GDFLIB_FILTER_MA32_DEFAULT;
static Frac16 mf16Value;
static Frac16 mf16FilteredValue;

void Isr(void);

void main(void)
{
        /* filter window size 2 ^ 2 = 4 points */
        mudtFilterMA32.w16N = 2;
```

```
        /* Filter initialization to 0.5 */
        GDFLIB_FilterMA32InitVal(FRAC16(0.5), &mudtFilterMA32);
}

/* Periodical function or interrupt */
void Isr(void)
{
        /* Filter calculation */
        mf16FilteredValue = GDFLIB_FilterMA32(mf16Value,
&mudtFilterMA32);
}
```

3.11.10 Performance

This section specifies the actual requirements of the function or macro in terms of required code memory, data memory, and number of clock cycles to execute.

Table 3-39. Performance of the GDFLIB_FilterMA32InitVal Function

Code Size (words)	13	
Data Size (words)	0	
Execution Clock	Min	41
	Max	41

3.12 GDFLIB_FilterMA32

This function calculates a recursive form of an average filter. It also has an inline version.

3.12.1 Synopsis

```
#include ”gdflib.h”
Frac16 GDFLIB_FilterMA32(Frac16 f16In, GDFLIB_FILTER_MA32_T * const
pudtFilter)
```

3.12.2 Prototype

```
asm Frac16 GDFLIB_FilterMA32FAsm(Frac16 f16In, GDFLIB_FILTER_MA32_T *
const pudtFilter)
```

3.12.3 Arguments

This subsection describes the input/output arguments to a function or a macro. It explains the algorithms being used by functions or macro.

Table 3-40. Function Arguments

Name	In/Out	Format	Range	Description
f16In	in	SF16	0x8000... 0x7FFF	input signal to be filtered
*pudtFilter	in/out	N/A	N/A	Pointer to a filter structure, which contains filter coefficients and filter buffer; the GDFLIB_FILTER_MA32_T data type is defined in the header file GDFLIB_FilterMA32asm.h.

Table 3-41. User-Type Definitions

Typedef	Name	In/Out	Format	Range	Description
GDFLIB_FILTER_MA32_T	f32Acc	in/out	SF32	0x80000000... 0x7FFFFFFF	internal filter accumulator
	w16N	in	SF16	0x8000... 0x7FFF	number of filtered points (filter window size)

3.12.4 Availability

This library module is available in the C-callable interface assembly formats.

This library module is targeted for the 56800E and 56800Ex platforms.

3.12.5 Dependencies

The dependent files are:

- GDFLIB_FilterMA32asm.h
- GDFLIB_types.h

3.12.6 Description

The **GDFLIB_FilterMA32** function calculates a recursive form of an average filter. The filter calculation consists of the following equations:

$$acc(k) = acc(k-1) + x(k) \quad \text{Eqn. 3-59}$$

$$y(k) = \frac{acc(k)}{n_p} \quad \text{Eqn. 3-60}$$

$$acc(k) \leftarrow acc(k) - y(k) \quad \text{Eqn. 3-61}$$

where x(k) is the actual value of the input signal, acc(k) is the internal filter accumulator, y(k) is the actual filter output, and n_p is the number of points in the filtered window. The size of the filter window (number of filtered points) shall be defined prior to this function call. The number of filtered points is defined by assigning a value to the pFilter.w16N variable, stored within the filter structure. This number represents the number of filtered points as a power of 2, as follows:

$$n_p = 2^{(\text{pudtFilter.w16N})} \quad , \text{pudtFilter.w16N} \geq 0 \quad \text{Eqn. 3-62}$$

where n_p is the actual number of filtered points (size of filter window).

3.12.7 Returns

The function returns the filtered value of the input f16In in the step k, and stores the difference between the filter accumulator and the output in the step k into the filter accumulator.

3.12.8 Range Issues

The internal filter accumulator acc(k) is implemented as a 32-bit variable.

3.12.9 Special Issues

The size of the filter window (number of filtered points) must be defined prior to this function call and must be equal to or greater than zero.

The algorithm's structure should be initialized by the **GDFLIB_FilterMA32Init** or **GDFLIB_FilterMA32InitVal** functions prior the filter algorithm use.

The **GDFLIB_FilterMA32** function is the saturation mode independent.

3.12.10 Implementation

The **GDFLIB_FilterMA32** function is implemented as a function call

Example 3-13. Implementation Code

```
#include "gdflib.h"

static GDFLIB_FILTER_MA32_T mudtFilterMA32 = GDFLIB_FILTER_MA32_DEFAULT;
static Frac16 mf16Value;
static Frac16 mf16FilteredValue;

void Isr(void);

void main(void)
{
        /* filter window size 2 ^ 2 = 4 points */
        mudtFilterMA32.w16N = 2;

        /* Filter initialization */
        GDFLIB_FilterMA32Init(&mudtFilterMA32);
}

/* Periodical function or interrupt */
void Isr(void)
{
        /* Filter calculation */
        mf16FilteredValue = GDFLIB_FilterMA32(mf16Value,
&mudtFilterMA32);
}
```

3.12.11 Performance

This section specifies the actual requirements of the function or macro in terms of required code memory, data memory, and number of clock cycles to execute.

Table 3-42. Performance of the GDFLIB_FilterMA32 Function

Code Size (words)	23	
Data Size (words)	0	
Execution Clock	Min	43 cycles
	Max	43 cycles

Appendix A Revision History

Table 0-1. Revision history

Revision number	Date	Subsequent changes
0	02/2014	Initial release

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

56800Ex_GDFLIB
Rev. 0, 02/2014

freescale™