GigaDevice Semiconductor Inc.

GD32G5x3 FIR / IIR User Guide

Application Note

AN208

Revision 1.0

(May. 2024)

Table of Contents

List of Figures

List of Tables

1. Introduction

FIR Filter (Finite Impulse Response Filter): FIR filter is a type of linear time-invariant filter, whose output depends only on the current output and past inputs. FIR filters achieve filtering effects by weighted averaging of the input signal. Common FIR filter design methods include windowing, frequency sampling, and optimization methods (such as least squares).

IIR Filter (Infinite Impulse Response Filter): IIR filter is a type of filter with infinite impulse response, whose output depends not only on the current input and past outputs but also on past inputs. IIR filters have narrower transition bands and steeper stopband edges, but they introduce some time-domain and frequency-domain nonlinear distortion. Common IIR filters include impulse invariant method, bilinear transformation, and frequency response matching method.

GD32G533_553 Filter Algorithm Accelerator (FAC) contains multiplier, accumulator, and address generation logic units, which can implement vector elements stored in local storage indexed. FAC supports loop buffers at both input and output ends, facilitating the implementation of digital filters including FIR (Finite Impulse Response) filters and IIR (Infinite Impulse Response) filters. FAC frees the CPU from frequent or lengthy filtering operations, accelerating computation compared to software filtering and enhancing the processing speed of critical tasks.

2. Filter principles

Digital signal processing (DSP) is widely utilized in various fields such as audio signal processing, motor control, image processing, and video processing. Here are some examples illustrating the practical effects of using filters in these fields:

Figure 2-1. The original signal without overlaid noise

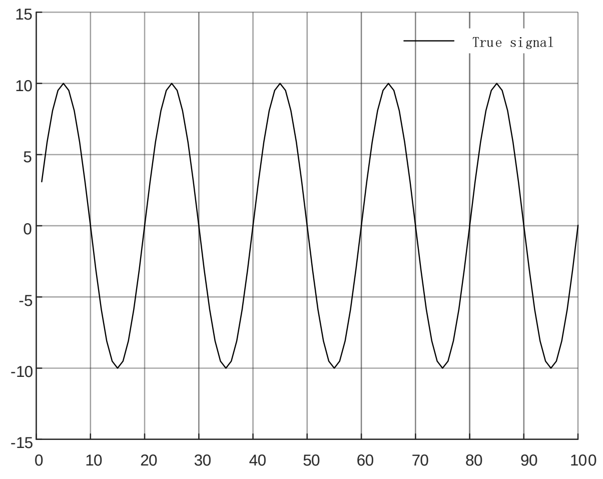


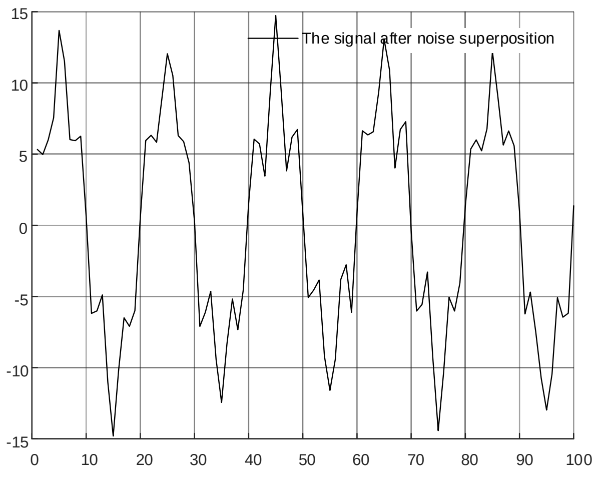
Figure 2-1. The original signal without overlaid noise is a standard sine wave. The sine wave has a peak amplitude A of 10, a frequency f of 500 Hz, and a sampling frequency Fs of 10 kHz. This signal does not contain any noise, and its formula can be expressed as:

$$x(i) = A \times \sin(2\pi \times i \times f / F_s) \tag{2-1}$$

In practical control scenarios, signals often contain various types of noise. Therefore, to simulate the above analysis conditions, Gaussian white noise (denoted as 'gussi') and noise components with amplitudes A_1 = 3 at three times the frequency and A_2 = 1 at five times the frequency are added to the signal. The signal formula can be expressed as:

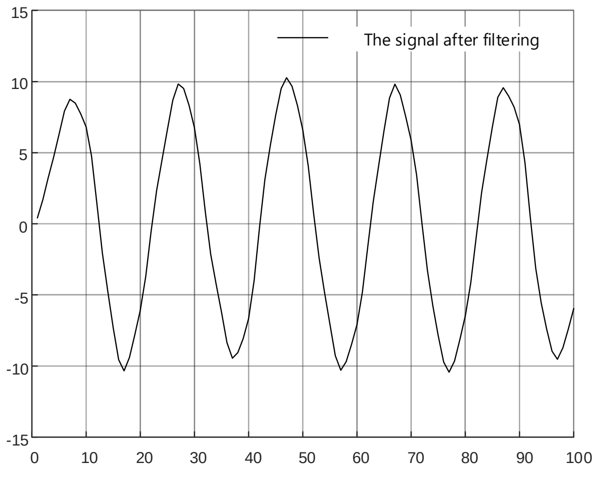
$$x\_noise(i) = A \times \sin\left(2\pi \times i \times \frac{f}{F_s}\right) + A_1 \times \sin\left(3 \times 2\pi \times i \times \frac{f}{F_s}\right) + A_2 \times \sin\left(5 \times 2\pi \times i \times \frac{f}{F_s}\right) + gussi(i) \tag{2-2}$$

Figure 2-2. The signal after noise superposition



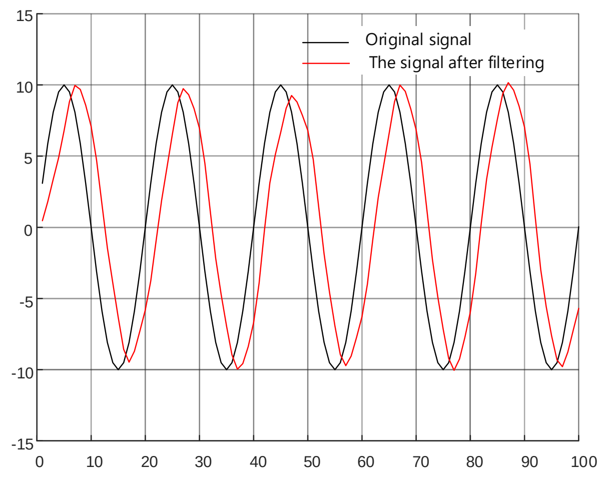
After adding noise to the signal, the characteristics of the original sine wave become less distinguishable. Therefore, to obtain the outline of the original signal more accurately, it is necessary to filter the signal after noise superposition. For this purpose, a second-order Butterworth low-pass filter (IIR) is designed to filter the signal, with a cutoff frequency of 1000 Hz. The filtered signal is shown in ***Figure 2-3. The signal after filtering***.

Figure 2-3. The signal after filtering



Compare the filtered signal with the original signal, as shown in ***Figure 2-4. Compare the signal after filtering with original signal***.

Figure 2-4. Compare the signal after filtering with original signal



It is evident that the amplitude of the signal has been attenuated after filtering, and the phase of the signal has lagged. In comparison to the original signal, although the filter has removed the majority of the signal noise, it inevitably introduces adverse effects such as phase lag and amplitude attenuation.

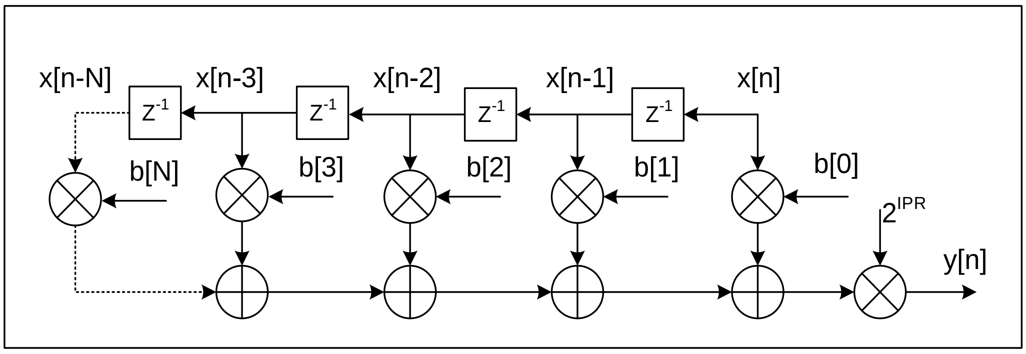
3. Finite Impulse Response

The FIR (Finite Impulse Response) filter is a type of linear phase filter, and its formula is:

$$y_n = 2^{IPR} \sum_{k=0}^{N} (x_{n-k} b_k) \tag{3-1}$$

In vector form, it is represented as $\vec{Y} = \vec{B} \times \vec{X}$, where $b_0 \sim b_k$ are N+1 filter coefficients. $\vec{X}$ contains an infinite number of input samples, and the elements of $\vec{Y}$ are obtained by dot product calculation $\overrightarrow{X_n} = [x_{n-N}, \dots, x_n]$. The structure of the FIR filter is shown in ***Figure 3-1. The structure of FIR filter***.

Figure 3-1. The structure of FIR filter



According to the formula of the FIR filter, we can see that the output of the FIR filter at any given moment depends entirely on the input from the previous time period and is independent of the output. This structure gives the FIR filter strong stability and does not introduce phase distortion, unlike the IIR filter, which may introduce phase distortion. However, because the FIR filter lacks feedback loops, it typically requires higher-order filters to achieve the same filtering effect as an IIR filter, meaning it requires more computational resources.

Next, let's discuss the design process of the FIR filter. Taking the signal after noise superposition in***Figure 2-2. The signal after noise superposition,*** we aim to extract a signal with a frequency of 500 Hz and remove noise signals at the third harmonic (1500 Hz), fifth harmonic (2500 Hz), and Gaussian white noise. To minimize noise, we design a 20th-order FIR low-pass filter. The filter parameters can be designed using corresponding software, and the final filter parameters are as follows: $\vec{B}$ = [0, -69, -207, -380, -404, 0, 1041, 2668, 4505, 5967, 6526, 5967, 4505, 2668, 1041, 0, -404, -380, -207, -69, 0].

Now, we need to use these parameters in the GD32G533_553. In the GD32G533_553, the data length is 256*32 bits, and for fixed-point operations, the maximum length is 256*16 bits. This means that in fixed-point mode, the maximum length of the data input to the filter is 256 data points (including the filter coefficients $\vec{B}$ and the data to be filtered). Before using it, we need to configure the data loading address and load the filter coefficients $\vec{B}$ to the initial address 0. The length of the filter coefficient is fir_coeffb_size = 20.

Table 3-1. Configuration of FIR Filter Filtering Parameters

```
/* Configure Coefficient buffer */
facconfig.coeff_addr             = 0;
facconfig.coeff_size             = fir_coeffb_size;
```

Configure the input data address length and threshold. The starting address for configuring input data has already been set for the filter parameters, so the starting address is fir_coeffb_size, and the length is the sum of the configured filter parameter length and the actual input data length.

Table 3-2. FIR Filter Input Parameter Configuration

```
/* configure input buffer */
facconfig.input_addr             = fir_coeffb_size;
facconfig.input_size             = fir_coeffb_size + fir_d0;
facconfig.input_threshold        = FAC_THRESHOLD_1;
```

After completing the allocation of the lengths and addresses for the filter parameter data and the input data to be filtered, write this configuration information into the corresponding buffer configuration registers: FAC_X0BCFG for X0 buffer, FAC_X1BCFG for X1 buffer, and FAC_YBCFG for Y buffer.

Table 3-3. FIR Input Data Address and Length Configuration

```
fac_init(&facconfig);
```

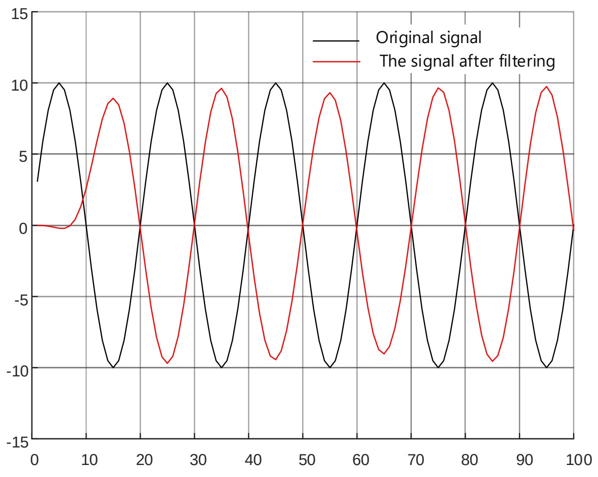
Next, preload the register data. The preload function is as follows:

Table 3-4. Preloading FIR Filter Data

```
faccoeff.coeffa_ctx              = NULL;
faccoeff.coeffa_size             = 0;
faccoeff.coeffb_ctx              = fir_coeffb;
faccoeff.coeffb_size             = fir_coeffb_size;
faccoeff.input_ctx               = input_data;
faccoeff.input_size              = input_array_size;
faccoeff.output_ctx              = NULL;
faccoeff.output_size             = 0;
/* preload X0   X1 Y buffer with coefficient */
fac_fixed_buffer_preload(&faccoeff);
```

After completing the above configuration, the filter parameters are configured. The effect after filtering with the above filter is as follows:

Figure 3-2. Effectiveness figure After FIR Filtering



It should be noted that since a fixed-point filter is being used, the filter parameters input to the filter should be scaled up by a factor of 32768. This scaling compensates for the fixed-point representation used in the filter processing.

When using float filter.

Table 3-5. FIR float Filtering enable

fac_float_enable();

At this point, the filter parameters input to the filter are:

$\vec{B}$ =[-0.000000000000000, -0.002122271148825, -0.006325353991514, -0.011611810377621, -0.012354656748982, 0.000000000000000, 0.031774497558567, 0.081435907564218, 0.137493781701943, 0.182125490388735, 0.199168830106960, 0.182125490388735, 0.137493781701943, 0.081435907564218, 0.031774497558567, 0.000000000000000, -0.012354656748982, -0.011611810377621, -0.006325353991514, -0.002122271148825, -0.000000000000000];

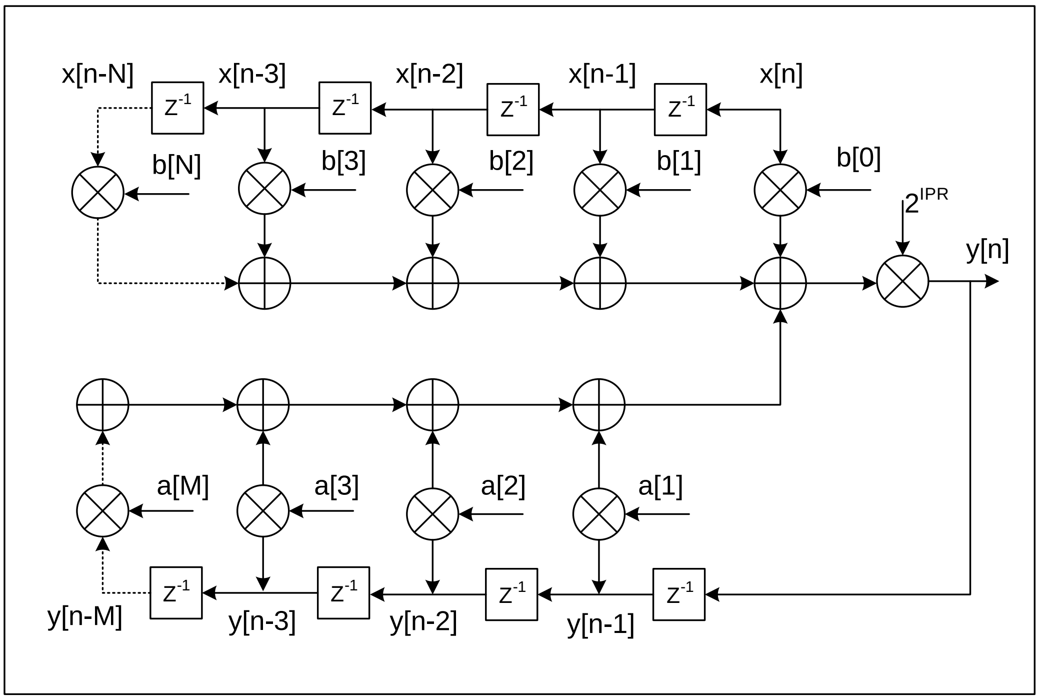
4. Infinite Impulse Response

The IIR (Infinite Impulse Response) filter is a type of non-linear phase filter, and its formula is:

$$y_n = 2^{IPR}(\sum_{k=0}^{N}(x_{n-k} \times b_k) + \sum_{k=1}^{M}(y_{n-k} \times a_k)) \qquad (4\text{-}1)$$

In vector form, it is represented as $\vec{Y} = \vec{B} \times \vec{X} + \vec{A} \times \vec{Y}$, where $b_0 \sim b_k$ are N+1 filter coefficients, $a_0 \sim a_M$ are M feedback filter coefficients. $\vec{X}$ contains an infinite number of input samples, and the elements of $\vec{Y}$ are obtained by dot product calculation $\overrightarrow{X_n} = [x_{n-N}, \ldots, x_n]$., and the elements of $\vec{Y}$ are obtained by dot product calculation.The structure of the IIR filter is shown in ***Figure 4-1. The structure of IIR filter***

Figure 4-1. The structure of IIR filter



From the IIR filtering formula, it can be inferred that the IIR filter includes feedback loops. At any given moment, the output not only depends on the input from the previous time period $\vec{X}$, but also on the output $\vec{Y}$, Due to the presence of feedback loops, the stability of IIR filters is weaker compared to FIR filters. Additionally, IIR filters introduce phase distortion during signal processing. However, due to the use of feedback loops, IIR filters typically require lower order to achieve the same filtering effect as FIR filters, resulting in lower computational requirements.

Next, let's discuss the design process of IIR filters. Taking the signal after noise superposition in ***Figure 2-2. The signal after noise superposition***, we aim to extract a signal with a frequency of 500 Hz and remove noise signals at the third harmonic (1500 Hz), fifth harmonic (2500 Hz), and Gaussian white noise. To minimize the noise signal, we design a 2nd-order IIR low-pass filter. The filter parameters can be designed using corresponding software. The filter parameters obtained from the software design are floating-point filter parameters: $\vec{B}$

=[0.020083365564211, 0.040166731128423, 0.020083365564211]; $\vec{A}$ =[1.000000000000000, -1.561018075800718, 0.641351538057563]; At this point, all the filter parameters designed are floating-point parameters. When using fixed-point IIR filters, the corresponding filter parameters need to be processed. For 16-bit signed fixed-point data, the range is -32768 to 32767. To make full use of these data, the above parameters are scaled up by 16384 (for IIR filters, both the feedforward coefficients$\vec{B}$ and feedback coefficients $\vec{A}$ are scaled up, maintaining the filtering effect). $\vec{B}$=[329, 658, 329]; $\vec{A}$ =[16384, -25575, 10507]. It is important to note that in conjunction with the designed filter parameters, the actual filter parameters input to the MCU are: $\vec{B}$=[329, 658, 329]; $\vec{A}$ =[25575, -10507]; The first value of$\vec{A}$ needs to be removed (as it is fixed and not input by the user), (as it is fixed and not input by the user), and the remaining two values need to be processed in reverse. This difference arises from the difference in the form of the transfer function used to generate the filter parameters and the actual filter transfer function used in the MCU.

Next, we need to use the above parameters in the GD32G533_553. In the GD32G533_553, the data length is 256*32, and for fixed-point operations, the maximum length is 256*16. Therefore in fixed-point mode, the maximum length of the data input to the filter is 256 data points (including the feedforward filter coefficients $\vec{B}$ feedback filter coefficients $\vec{A}$ and the data to be filtered). When using it, first configure the data loading address and load the filter parameters $\vec{B}$.

The initial address is set to 0, and the filter parameter length is iir_coeffa_size + iir_coeffb_size; The IIR filter simultaneously uses feedforward filter coefficients $\vec{B}$ and feedback filter coefficients $\vec{A}$, so the filter coefficient length is the sum of the two.

Table 4-1. Configuration of IIR Filter Filtering Parameters

```
/* Configure Coefficient buffer */
facconfig.coeff_addr          = 0;
facconfig.coeff_size          = iir_coeffa_size + iir_coeffb_size;
```

Configure the input data address length and threshold. Since the filter parameters have already been configured during the input data configuration, the starting address is iir_coeffa_size + iir_coeffb_size, and the length is the sum of the configured filter parameter length and the actual input data length.

Table 4-2. IIR Filter Input Parameter Configuration

```
/* Configure input buffer */
facconfig.input_addr          = iir_coeffa_size + iir_coeffb_size;
facconfig.input_size          = iir_coeffb_size + iir_d0;
facconfig.input_threshold     = FAC_THRESHOLD_1;
```

Unlike FIR filters, IIR filters also require configuring the output array because they involve feedback filtering.

Table 4-3. IIR Filter feedback Parameter Configuration

```
/* Configure input buffer */
facconfig.output_addr         = iir_coeffa_size + iir_coeffb_size + iir_coeffb_size + iir_d0;
facconfig.output_size          = iir_coeffa_size + iir_d1;
facconfig.output_threshold     = FAC_THRESHOLD_1;
```

After completing the allocation of the lengths and addresses for the filter parameter data, input data to be filtered, and feedback parameters, write this configuration information into the corresponding buffer configuration registers: FAC_X0BCFG for X0 buffer, FAC_X1BCFG for X1 buffer, and FAC_YBCFG for Y buffer.

Table 4-4. IIR Input Data Address and Length Configuration

```
fac_init(&facconfig);
```

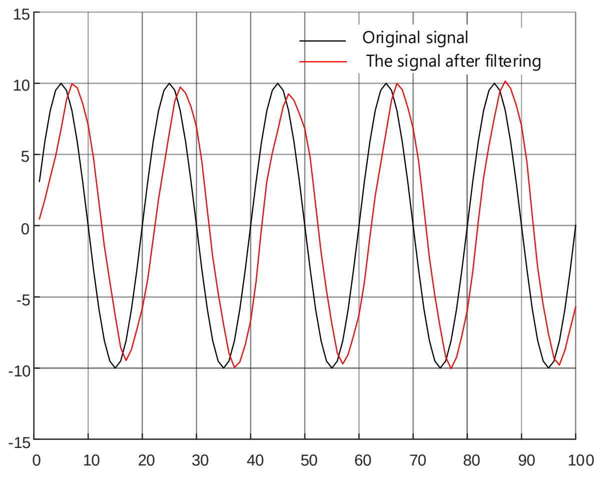
Next, preload the register data. The preload function is as follows:

Table 4-5. Preloading IIR Filter Data

```
faccoeff.coeffa_ctx          = iir_coeffa;
faccoeff.coeffa_size         = iir_coeffa_size;
faccoeff.coeffb_ctx          = iir_coeffb;
faccoeff.coeffb_size         = iir_coeffb_size;
faccoeff.input_ctx           = input_data;
faccoeff.input_size          = input_array_size;
faccoeff.output_ctx          = output_data;
faccoeff.output_size         = iir_coeffa_size;
/* preload X0   X1 Y buffer with coefficient */
fac_fixed_buffer_preload(&faccoeff);
```

After completing the above configuration, the filter parameters are configured. The effect after filtering with the above filter is as follows:

Figure 4-2. Effectiveness figure After IIR Filtering



It should be noted that since a fixed-point filter is being used, the filter parameters input to the filter should be scaled up by a factor of 32768. This scaling compensates for the fixed-point representation used in the filter processing.

When using float filter.

Table 4-6. IIR float Filtering enable

```
fac_float_enable();
```

At this point, the filter parameters input to the filter are:

$\vec{B}$ =[0.020083365564211, 0.040166731128423, 0.020083365564211]; $\vec{A}$ =[1.561018075800718, -0.641351538057563];

5. Revision history

Table 5-1. Revision history

Revision No.	Description	Date
1.0	Initial Release	May.1 2024