



Application Note

AN2312

n^{th} Order IIR Filtering Graphical Design Tool for PSoC™

Authors: Somsak Sukittanon, Ph.D., Stephen G. Dame, MSEE

Associated Project: Yes

Associated Part Family: CY8C21xxx, CY8C24xxxA, CY8C24794, CY8C27xxx, CY8C29xxx

PSoC Designer Version: 4.2

Associated Application Notes: AN2038

Abstract

High performance filtering has been implemented in a variety of industry standard Digital Signal Processing (DSP) chips. However, many applications don't require the high sampling rates and cost normally associated with these dedicated DSP chips. This Application Note presents an alternative for implementing high order IIR DSP filtering using inexpensive PSoC chips. A Biquad IIR filtering algorithm and MATLAB® Graphical User Interface (GUI) code and coefficient generation tools are developed that can dramatically reduce the time to deploy a digital filtering project for the PSoC. Links are provided for an example project and filter generation MATLAB source code.

Introduction

Many developers of small systems are faced with signal processing challenges during the design process. PSoC is a highly flexible electrical engineering environment in which to creatively approach and solve all kinds of systems' problems. With the on-chip ADC and DAC blocks of the PSoC, it is possible to sample analog signals and represent their amplitude values with as much digital precision as 8 – 14 bits at a variety of sample rates. Many system designs don't require video or even high kHz sample rates. The low cost of PSoC chips, combined with a reasonable MIPS rate and Flash program memory storage, enables some powerful traditional IIR filtering methods if only a library of DSP building blocks were available. A powerful filter design tool that can automatically generate PSoC code and filter coefficients would also be a benefit once the filtering code has been implemented.

This application presents flexible and powerful IIR Biquad filtering methods typically only used on much more powerful and expensive DSP chips. This Application Note also includes a project example and source code for a MATLAB tool that can automatically generate filter coefficients and PSoC filtering code.

Digital IIR Filter

For a linear time invariant system, an infinite impulse response (IIR) filter can be described in this form:

$$y[n] = b_0x[n] + b_1x[n-1] + \dots + b_Mx[n-M] - a_1y[n-1] - \dots - a_Ny[n-N]. \quad (1)$$

An output at time n , $y[n]$, can be computed from multiplying and summation of the current input, $x[n]$, and its previous values, $x[n-d]$, with weight coefficients b_i , also summation together with previous outputs, $y[n-d]$, with weight coefficients $-a_i$. M and N determine the number of zeros and poles, respectively, in the filter. For example, the Biquad IIR filter, referred to a second order filter, can be described by the following differential equation:

$$y[n] = b_0x[n] + b_1x[n-1] + b_2x[n-2] - a_1y[n-1] - a_2y[n-2]. \quad (2)$$

Applying Z -transform to Equation (2), the corresponding transfer function $H(Z)$ becomes:

$$H(Z) = \frac{b_0 + b_1Z^{-1} + b_2Z^{-2}}{1 + a_1Z^{-1} + a_2Z^{-2}}. \quad (3)$$

Efficient implementation, minimum delays and computation, are illustrated in Figure 1 below.

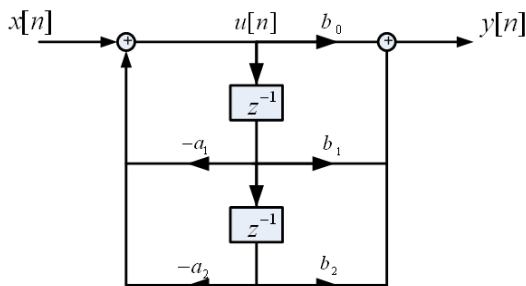


Figure 1. Second Order IIR Filter

There are several issues that need to be taken into account when using Equation (3) and Figure 1 in the PSoC ALU and MAC architecture (e.g., signed number representation, overflow). A signed integer number can represent a floating-point number between minus 1 and 1 to a specified bit resolution. How do we guarantee that b_i and a_i can be represented using a signed integer number format? If we are designing a stable IIR filter, this issue of stability must be addressed. From Equation (3), the transfer function in the form of zeros, z_i , and poles, p_i , can be described as:

$$H(Z) = \frac{k(1 - z_1Z^{-1})(1 - z_2Z^{-1})}{(1 - p_1Z^{-1})(1 - p_2Z^{-1})} \quad (4)$$

k is a constant. For the real filter, z_1 is the conjugate of z_2 and p_1 is the conjugate of p_2 .

$$H(Z) = \frac{k(1 - z_1Z^{-1})(1 - z_1^*Z^{-1})}{(1 - p_1Z^{-1})(1 - p_1^*Z^{-1})} \quad (5)$$

$$= \frac{k(1 - 2\text{Re}\{z_1\}Z^{-1} + |z_1|^2Z^{-2})}{(1 - 2\text{Re}\{p_1\}Z^{-1} + |p_1|^2Z^{-2})}$$

For a stable and minimum phase IIR filter, $\text{Re}\{p_1\}$, $\text{Re}\{z_1\}$, $|p_1|^2$ and $|z_1|^2$ must be less than 1. Comparing Equation (5) to (3), $|a_1|$ is always less than 2 and $|a_2|$ is always less than 1.

Biquad Implementation

In this section we describe the implementation of one Biquad filter in PSoC. The input, output, delays, and coefficient values are 16 bit signed numbers. The multiplication function is adopted from AN2038 ("Signed Multi-Byte Multiplication"). One shift left is also required to adjust the result to be 1.31-signed fractional numerical format.

Feedback Algorithm

Since $|a_1|$ is always less than 2, it is prescaled by 0.5 and then the result after multiplication with $u[n-1]$ is scaled back by 2 (using a left shift operation). For $|a_2|$, it is always less than 1, therefore it can be used directly. The following is PSoC assembly code for computing $u[n]$.

```

BiquadFiltering:
_BiquadFiltering:
    ;; retrieve the parameters
    mov X,SP
    ; x[n]
    mov A,[X-3]
    mov [iUn+1],A
    mov A,[X-4]
    mov [iUn],A
    ; statenumber
    mov A,[X-5]
    call LoadBiquadState

    ;; FEEDBACK PART
    ; ((-a1/2) * u[n-1])*2
    Multiply32s_16s_16s (cTemp), (iACoff+0),
    (iDelay+0)
    call ShiftLeft32BitsBuffer

    ; -a2 * u[n-2]
    MultiplyAndSum32s_16s_16s (cTemp),
    (iACoff+2), (iDelay+2)
    call ShiftLeft32BitsBuffer

    ; u[n] = x[n]-(a1*u[n-1])-(a2*u[n-2])
    mov A,[cTemp+1]
    add [iUn+1],A
    mov A,[cTemp]
    adc [iUn],A

```

Assembly Code 1. Feedback Operation

Feedforward Algorithm

Similar to $|a_1|$, $|b_1|$ is always less than 2 so we prescale all b_i by 0.5 and post scale the final result back by a factor of 2 (using a left shift operation). This method also prevents filter overflow. The filter state delays are updated after the feedforward operation. The following is PSoC assembly code for computing $y[n]$:

```

    ;; FORWARD PART
    ; (b0/2) * u[n]
    Multiply32s_16s_16s cTemp, (iBCoff+0),
    (iUn+0)

    ; (b1/2) * u[n-1]
    MultiplyAndSum32s_16s_16s (cTemp),
    (iBCoff+2), (iDelay+0)

    ; (b2/2) * u[n-2]
    MultiplyAndSum32s_16s_16s (cTemp),
    (iBCoff+4), (iDelay+2)
    call ShiftLeft32BitsBuffer
    call ShiftLeft32BitsBuffer
    mov [iBuffer+1],[cTemp+1]
    mov [iBuffer],[cTemp]

    ; swap delays
    ; delay1 <--- delay 0
    ; delay0 <--- u[n]
    mov [iDelay+2],[iDelay+0]
    mov [iDelay+3],[iDelay+1]
    mov [iDelay+0],[iUn+0]
    mov [iDelay+1],[iUn+1]
    mov A,iDn
    add A,[cStateDelay]
    mov [cTemp],A
    mov [cTemp+1],iDelay
    call SwapDelays

```

```

    ; y[n] = 2*{((b0/2) * u[n])+((b1/2) * u[n-1])
    ; +((b2/2) * u[n-2])}
    mov X,[iBuffer]
    mov A,[iBuffer+1]
ret

```

Assembly Code 2. Feedforward Operation

n^{th} Order IIR Filter

In the previous section, we discussed the implementation of a second order IIR filter. In this section, we discuss the extension of the Biquad to higher order IIR filters. Suppose we are required to design a 6th order IIR filter. The transfer function can be written as:

$$H(Z) = \frac{b_0 + b_1 Z^{-1} + \dots + b_6 Z^{-6}}{1 + a_1 Z^{-1} + \dots + a_6 Z^{-6}} \quad (6)$$

Equation (6) can also be represented in pole-zero form as:

$$\begin{aligned}
 H(Z) &= \frac{k(1 - z_1 Z^{-1})(1 - z_2^* Z^{-1}) \dots (1 - z_3^* Z^{-1})}{(1 - p_1 Z^{-1})(1 - p_1^* Z^{-1}) \dots (1 - p_3^* Z^{-1})} \\
 &= \left(\frac{k_1(1 - z_1 Z^{-1})(1 - z_1^* Z^{-1})}{(1 - p_1 Z^{-1})(1 - p_1^* Z^{-1})} \right) \\
 &\quad \times \left(\frac{k_2(1 - z_2 Z^{-1})(1 - z_2^* Z^{-1})}{(1 - p_2 Z^{-1})(1 - p_2^* Z^{-1})} \right) \\
 &\quad \times \left(\frac{k_3(1 - z_3 Z^{-1})(1 - z_3^* Z^{-1})}{(1 - p_3 Z^{-1})(1 - p_3^* Z^{-1})} \right).
 \end{aligned} \quad (7)$$

Equivalently, Equation (6) can be expressed as 3 cascaded Biquad IIR filters. Each Biquad structure requires a different set of a_i, b_i, k_i , and filter state delays. To implement a cascaded Biquad IIR filter, we can use the same previous PSoC assembly code but the Biquad structure is dynamically loaded each time prior to each Biquad filtering iteration. The filter state delays are saved after each Biquad operation. The following is the PSoC assembly code for dynamically loading the Biquad structure:

```

;compute the beginning of tables and
delays
mov X, NcCoeffs
call GetStateHeader
mov [cStateCoeff],[cTemp]
mov X, NcDelays
call GetStateHeader
mov [cStateDelay],[cTemp]

;load coefficients
mov [cTemp+0], iBCoff
mov [cTemp+1], NcBCoffs
mov A, 0
add A, [cStateCoeff]
call LoadTableItems
mov [cTemp+0], iACoff
mov [cTemp+1], NcACoffs
mov A, NcBCoffs
add A, [cStateCoeff]
call LoadTableItems

;load delays
mov A, iDn
add A, [cStateDelay]
mov [cTemp+1], A
mov [cTemp], iDelay
call SwapDelays

```

Assembly Code 3. Dynamic Load Biquad State

Experiments and Results

We compare the results of filtering between 16-bit integer precision data from PSoC and 64-bit precision data from MATLAB®. Different inputs are tested for stability and round off error.

6th Order IIR filter

Design Specification: 6th order digital low pass filter using Butterworth design. Cutoff frequency at 500 Hz, with a sampling rate of 4000 Hz.

Equation (8) shows the transfer function of a 3-Biquad IIR filter that implements the above specification. Figure 2 shows the pole-zero plot and frequency response of this filter.

$$\begin{aligned}
 H(Z) = & 2 \left(\frac{0.0508 + 0.1021Z^{-1} + 0.0513Z^{-2}}{1 - 0.8403Z^{-1} + 0.1883Z^{-2}} \right) \quad (8) \\
 & \times 2 \left(\frac{0.0508 + 0.1017Z^{-1} + 0.0508Z^{-2}}{1 - 0.9428Z^{-1} + 0.3333Z^{-2}} \right) \\
 & \times 2 \left(\frac{0.0508 + 0.1013Z^{-1} + 0.0504Z^{-2}}{1 - 1.1954Z^{-1} + 0.6906Z^{-2}} \right).
 \end{aligned}$$

Inputs

The impulse and unit response are used for testing. Assuming that we are using 12-bit samples from the ADC, the filtering results from PSoC and MATLAB are shown in Table 1.

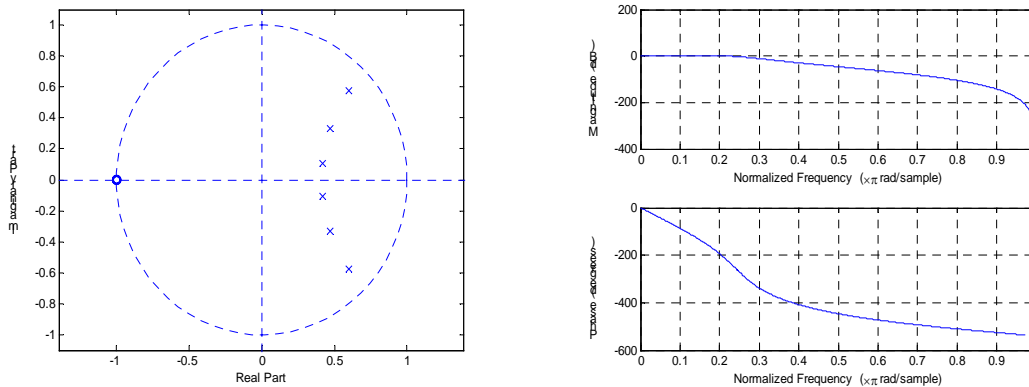


Figure 2. Pole-Zero Plot (Left) and Frequency Response of 6th Order IIR Filter (Right)

Table 1. Comparison Results

Time	Input		Output	
	Impulse	MATLAB	MATLAB	PSoC
0	0x07FF	0x0002	0x0002	0x0002
1	0x0000	0x0013	0x0013	0x0013
2	0x0000	0x0051	0x0050	0x0050
3	0x0000	0x00D3	0x00D2	0x00D2
4	0x0000	0x0183	0x0180	0x0180
5	0x0000	0x020E	0x020B	0x020B
6	0x0000	0x021F	0x021B	0x021B
7	0x0000	0x019C	0x0199	0x0199
8	0x0000	0x00BE	0x00BB	0x00BB
9	0x0000	0xFFE3	0xFFE2	0xFFE2
10	0x0000	0xFF63	0xFF62	0xFF62
11	0x0000	0xFF55	0xFF55	0xFF55
12	0x0000	0xFF9C	0xFF9B	0xFF9B
13	0x0000	0xFFFA	0xFFFA	0xFFFA
14	0x0000	0x003D	0x003C	0x003C
15	0x0000	0x004C	0x004A	0x004A

Input	Output	
	MATLAB	PSoC
0x07FF	0x0002	0x0002
0x07FF	0x0015	0x0015
0x07FF	0x0066	0x0065
0x07FF	0x013A	0x0138
0x07FF	0x02BC	0x02BA
0x07FF	0x04CA	0x04C7
0x07FF	0x06E9	0x06E5
0x07FF	0x0885	0x0881
0x07FF	0x0943	0x093E
0x07FF	0x0928	0x0923
0x07FF	0x088D	0x0888
0x07FF	0x07E4	0x07DF
0x07FF	0x0782	0x077D
0x07FF	0x077E	0x0779
0x07FF	0x07BB	0x07B7
0x07FF	0x0807	0x0803

Matlab Graphic Tool

To help PSoC developers doing DSP development on their systems, we have implemented an IIR filtering Graphical Design Tool, which can directly generate the PSoC files (*main.c* and *iirbiquad.inc*). This tool and its outputs are shown in Figure 3. Developers may simply substitute *main.c* and *iirbiquad.inc* with the associated project. The *main.c* file will need some minor adjustments that correspond to the user project environment.

MATLAB® requirements are to be running at least version 6. After launching MATLAB and changing to the project directory, you type `>>runGUIPSOC` at the command line. Four types of filters can be designed. The cutoff frequency is chosen after entering a sampling frequency. The output graph can be shown in different modes (e.g., Magnitude, phase).

More information about the PSoC signal-processing tool is available for free downloading at www.virtual-dsp.com.

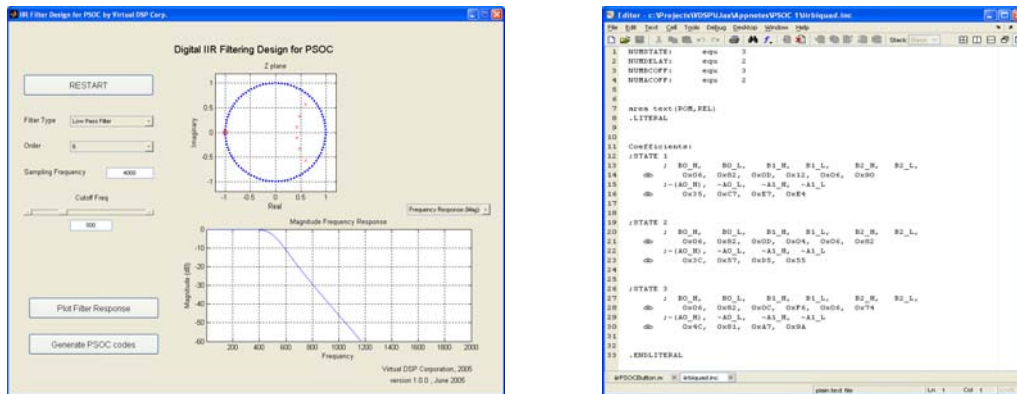


Figure 3. (Left) IIR Filtering Design Graphical Tool using MATLAB (Right) an Output File Used in PSoC

Conclusions

In this Application Note we present the implementation of a high order IIR filter for PSoC. Many practical design and implementation issues, signed representation or overflow, were discussed. We also presented an easy way to generate several filters, i.e., low pass, high pass, band pass, or notch filters, and plug coefficients into PSoC projects. The graphical design tool is currently running on MATLAB® and can generate PSoC filter code in one click.

Appendix

The multiply and sum operation is modified from Multiply32s_16s_16s (Application Note AN2038). Instead of storing into the first parameter, we add to the previous value.

```
macro MultiplyAndSum32s_16s_16s
; @0 = @0+(@1 * @2)
; 16bit by 16 bit signed multiply
; with 32 bit signed result and added to prior 32
bits
;
;          X(s) X+1(u)
;          +-----+
;          | / | / |
;          | / | / |
; R | (1) | (4) | Y(s)   Napier Matrix
;    | /v | / ^ |
;    | / v | / ^ |
;    +---v---| ^ ^ |
;    | v / | ^ / |
;    | v / | ^ / |
; R+1 | (2)>>>(3) | Y+1(u)
;    | / | / |
;    | / | / |
;    +-----+
;          R+2   R+3
; @0 = @0+(@1* @2)
;
; (1)
PushMulX @1
PushMulY @2
GetLSB
add [@0 + 1],A
GetSignedMSB
adc [@0],A
;
; (2)
PushMulY (@2 + 1)
GetXsYuMSB (@1 ), (@2+1)
cmp A,128
jc . + 4 ;pass on carry
dec [@0]
add [@0 + 1],A
adc [@0],0
GetLSB
add [@0+2],A
adc [@0+1],0
adc [@0],0
;
; (3)
PushMulX (@1 + 1)
GetUnsignedMSB (@1+1),(@2 + 1)
add [@0+2],A
adc [@0+1],0
adc [@0],0
GetLSB
add [@0+3],A
adc [@0+2],0
adc [@0+1],0
adc [@0],0
;
; (4)
PushMulY (@2)
GetXsYuMSB, (@1 + 1),(@2)
push A
cmp A,128
jc . + 4
dec [@0]
GetLSB
add [@0 + 2],A
pop A
adc [@0 + 1],A
adc [@0],0
endm
```

Assembly Code 4. Multiply and Sum Code

About the Authors

Name: Somsak Sukittanon, Ph.D.

Title: Principal R&D Engineer

Background: Dr. Sukittanon is a principal R&D engineer at Virtual DSP Corporation. He has led several investigations in the area of software development of signal processing and embedded systems. He is also a part-time lecturer at University of Washington. He has taught many classes, e.g., circuit analysis, DSP embedded design. He is the recipient of the 2005 outstanding teaching award for Electrical Engineering Department at the University of Washington.

Contact: somsak@virtual-dsp.com

Name: Stephen G. Dame, MSEE

Title: President and CEO

Background: Mr. Dame is president and founder of Virtual DSP Corporation, which is a research and development and manufacturing company in the area of digital signal processing, wireless and internet computing devices. He holds a masters degree in electrical engineering and is a successful entrepreneur with more than 25 years of experience in developing products in the medical, aerospace and consumer electronic markets. He is a recipient of 1994 Technical Fellow award for his work in Doppler Ultrasound at Advanced Technology Labs (ATL).

Contact: steve@virtual-dsp.com

Cypress Semiconductor
2700 162nd Street SW, Building D
Lynnwood, WA 98087
Phone: 800.669.0557
Fax: 425.787.4641

<http://www.cypress.com/>

Copyright © 2005 Virtual DSP Corporation. All rights reserved. Licensed to Cypress Semiconductor Corp.
"Programmable System-on-Chip," PSoC, PSoC Designer and PSoC Express are trademarks of Cypress Semiconductor Corp.
MATLAB® is a registered trademark of MathWorks, Inc.

All other trademarks or registered trademarks referenced herein are the property of their respective owners.
The information contained herein is subject to change without notice. Made in the U.S.A.