FIR Filtering in PSoC with Application to Fast Hilbert Transform - Standard

AN2328

Authors: Somsak Sukittanon, Ph.D., Stephen G. Dame, MSEE
Associated Project: Yes
Associated Part Family: CY8C24xxxA, CY8C24794, CY8C27xxx, CY8C29xxx
GET FREE SAMPLES HERE
Software Version: PSoC® Designer™ 4.2
Associated Application Notes: AN2038, AN2313

Application Note 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 dedicated DSP chips. This Application Note presents an alternative for implementing FIR DSP filtering using inexpensive PSoC chips. A finite impulse response (FIR) filtering algorithm, MATLAB® Graphical User Interface (GUI) code, and coefficient generation tools are developed that dramatically reduce the time to deploy an FIR filtering project for the PSoC device. Links are provided for an example project and filter generation MATLAB source code.

Introduction
In power and energy designs, it is useful to be able to detect the instantaneous frequency of low frequency sinewave signals (such as the AC line voltage) at more data points than simply the zero crossings. Input voltages of 110 or 220 volts can be divided down using a simple resistor divider and input to a PSoC analog input for sampling into digital signals using the on-chip ADC facilities. An FIR filter can be employed with a coefficient set comprising the “Hilbert Transform” as a means to create a complex signal that can then be used to determine the instantaneous phase angle of the sinewave.

A powerful filter design tool that can automatically generate PSoC code and filter coefficients would also be a benefit once the FIR algorithm has been designed and implemented. This application presents a cookbook method to design an FIR filter building block and automatically generate coefficient files for a specific designed filter characteristic.

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 FIR Filter
For a linear time invariant system, a finite impulse response (FIR) filter can be described in this form:
\[ y[n] = b_0 x[n] + b_1 x[n-1] + \cdots + b_N x[n-N+1]. \]  
Equation 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 \( b_i \) where \( N \) determines the number of zeros in the filter. For example, the moving-average filter, with \( N \) number of points, can be described by the following differential equation:
\[ y[n] = \frac{1}{N} \sum_{m=0}^{N-1} x[n-m] \]  
Equation 2

The frequency response of this system, after applying the discrete Fourier transform, is:
\[ H(e^{jω}) = \frac{1}{N} \sum_{m=0}^{N-1} e^{-jωm} = \frac{1 - e^{-jωN}}{N - e^{-jω}} \]  
Equation 3

\[ = e^{-jω(N-1)/2} \frac{\sin(ωN/2)}{N \sin(ω/2)} \]
The system described in Equation Error! Reference source not found. can be viewed as a low-pass filter where the cut-off frequency depends on the length of the filter, $N$. The longer the filter is, the narrower the bandwidth will be. The generic direct form FIR structure is illustrated in Figure 1.

Figure 1. Generic FIR Structure

Unlike an IIR filter, FIR filters always produce a stable output. For certain types (i.e., symmetric or antisymmetric shapes), they also provide a linear phase output. When an FIR filter is implemented in PSoC, the absolute value of $b_i$ must be less than one, $\forall |b_i| \leq 1$. Prescale and postscale values can be used if the number is over that range.

**FIR Implementation**

In this section, we describe the implementation of one FIR filter in PSoC. The input, output, delays, and coefficient values are 16-bit signed numbers. The multiplication function is adopted from Application Note AN2038 “Signed Multi-Byte Multiplication” and the feedforward algorithm is adapted from Application Note AN2313 “$n^{th}$ order IIR Filtering Graphical Design Tool for PSoC.” Readers are encouraged to read those Application Notes first. The most important issue in FIR implementation is circular buffering, which we will discuss in the next section.

**Circular Buffering**

FIR can be viewed as the convolution between the filter, length $N$, with the current input and $N-1$ delayed inputs.

$$y[n] = h[n] \otimes x[n]$$

$$= \sum_{m=0}^{N-1} h[m]x[n-m]$$

Equation 4

For example, the output at time $n$, $y[n]$ can be computed as:


Equation 5

The memory organization of Equation 5 is illustrated in Figure 2.

Figure 2. Memory Organization of the Output at Time $n$, $y[n]$

Before we see the importance of circular buffering, let’s compute the output at time $n+1$, $y[n+1]$.


Equation 6

And the memory organization of Equation 6 is illustrated in Figure 3.

Figure 3. Memory Organization of the Output at Time $n+1$, $y[n+1]$

There are some observations from Figures 2 and 3. 1) At any given time, the output only involves $N$ multiplication between the input, its delays and filter coefficients. 2) The first filter coefficient, $h[0]$, is multiplied to the most current input while the last filter coefficient, $h[N-1]$, is multiplied to the oldest memory in the delayed buffer. The efficient way to manage these stored samples is through circular buffering.

Figure 4. Circular Buffering Applied to the Delayed Input
First, we assign a pointer to consecutive memory locations. As illustrated in Figure 4 at time \( t \), the pointer is located to the lowest memory location (left figure), usually this is the oldest input that will be thrown away. When the new input is obtained from the ADC, the value will be put into this place and the pointer will move forward by one (middle figure) before the start of convolution. Notice that the pointer is now located in the oldest memory again (since we just replaced it with the new one). The convolution will start by multiplying the delay and filter coefficient one by one. So the filter coefficient must be arranged in descending order, \( h[N-1], \ldots, h[0] \). If the pointer in a circular buffer reaches the end of memory, it will wrap back to the beginning. After convolution, the pointer will move back to the same location, the oldest memory, and wait for the new input. For the next time sample, the input is saved and the process repeats itself (right figure).

**Convolution Algorithm**

First, we start by clearing the circular buffer and setting the pointer to the beginning of the delay taps.

**Code 1. Circular Buffer Initialization**

```assembly
;;-----------------------------------------
;;  FIRInit:
;;  Initializes FIR by clearing all delays
;;  and setting pointers to the right position
;;-----------------------------------------
_FIRInit:
;clear all delay buffers
mov A, 0
mov [cTemp+0], iDn
mov [cTemp+1], Ncdelaytaps
call BufferInit

;move one pointer to the beginning
mov [ptr_DelayTaps], iDn
;current address of FIR delay (16 bits)
mov [ptr_DelayTaps+1], 0
```

When the new sample is retrieved from the ADC, it is saved to the circular buffer and the pointer is moved to the next sample position. We have to check if the pointer falls off the circular buffer every time it is incremented.

**Code 2. Generic FIR Filtering**

```assembly
;;-----------------------------------------
;;  ;;FIRFiltering:
;;  Generic FIR filtering
;;-----------------------------------------
_FIRFiltering:
;
;Convolution
mov [ctr_Hn], 0 ;reset the counter of filter
mov [cTemp+0], 0 ;reset 32 bits ACC
mov [cTemp+1], 0
mov [cTemp+2], 0
mov [cTemp+3], 0
mov [ctrl], NUMFIRDELAY ;load number of multiplication
STARTConvolution:
cmp [ctrl], 0
jz EXITConvolution

```

November 17, 2005

Document No. 001-26759 Rev. **

3

[+] Feedback
add [ctr_Hn], 1
mov A,[ctr_Hn]
index FIRCoefficients
mvi [ptr_curHn], A ; load 16 bit (LOW)
add [ctr_Hn], 1
MultiplyAndSum32s_16s_16s (cTemp), (icurXn), (icurHn) // 16 bits multiply with 16 bits
dec [ctrl]  
jmp STARTConvolution
EXITConvolution:
call ShiftLeft32BitsBuffer //adjust to 1.31
mov X,[cTemp]
mov A,[cTemp+1]
ret

The following code is the method for incrementing the circular buffer. The content inside ptr_DelayTaps+1 is the counter for keeping track of whether or not the pointer reaches the bottom.

Code 3. Increment the Circular Buffer Pointer

;-----------------------------------------

class IncrementwithCircular
push A
inc [@0+1] ;increase the counter
mov A,02
sub A,[@0+1]
jen ENDIncrementwithCircular
mov [@0],@1 ;reach the end, go back to the beginning
mov [@0+1],0 ;load the counter with zero
ENDIncrementwithCircular:
pop A
endm

Experiments

We compare the results of filtering between 16-bit integer precision data from the PSoC and 64-bit precision from MATLAB. Different inputs are tested for stability and round-off errors. The FIR filter used in this experiment is a Hilbert filter. The details of its properties are discussed in the next section. The Hilbert filter, length \( N \) (odd number), is defined as:

\[
h[n] = \begin{cases} 
2\sin\left(\frac{\pi (n-\alpha)/2}{n-\alpha}\right), & n \neq \alpha \\
0, & n = \alpha
\end{cases}
\]  

Equation 7

\( \alpha = \left(\frac{N-1}{2}\right) \). The filter length is 65 and a Hanning window is used to reduce the sidelobe effects. These coefficients are reformatted before they are used in the PSoC. As shown in fircoeffs.inc, they are arranged in descending order starting with the high byte. The length of the filters and the size of the input are chosen such that the output does not overflow. 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.
Table 1. Comparison Results

<table>
<thead>
<tr>
<th>Time</th>
<th>Input</th>
<th>Output</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Impulse</td>
<td>MATLAB</td>
<td>PSoc</td>
<td>Step Resp.</td>
</tr>
<tr>
<td>0</td>
<td>0x07FF</td>
<td>0x0000</td>
<td>0x0000</td>
<td>0x07FF</td>
</tr>
<tr>
<td>1</td>
<td>0x0000</td>
<td>0x0000</td>
<td>0xFFFF</td>
<td>0x07FF</td>
</tr>
<tr>
<td>2</td>
<td>0x0000</td>
<td>0xFFFFE</td>
<td>0x0000</td>
<td>0x07FF</td>
</tr>
<tr>
<td>3</td>
<td>0x0000</td>
<td>0x0FFFF</td>
<td>0x0000</td>
<td>0x07FF</td>
</tr>
<tr>
<td>4</td>
<td>0x0000</td>
<td>0x0000</td>
<td>0xFFFF</td>
<td>0x07FF</td>
</tr>
<tr>
<td>5</td>
<td>0x0000</td>
<td>0xFFFC</td>
<td>0x0000</td>
<td>0x07FF</td>
</tr>
<tr>
<td>6</td>
<td>0x0000</td>
<td>0x0000</td>
<td>0x0000</td>
<td>0x07FF</td>
</tr>
<tr>
<td>7</td>
<td>0x0000</td>
<td>0xFFF9</td>
<td>0xFFF8</td>
<td>0x07FF</td>
</tr>
<tr>
<td>8</td>
<td>0x0000</td>
<td>0x0000</td>
<td>0x0000</td>
<td>0x07FF</td>
</tr>
<tr>
<td>9</td>
<td>0x0000</td>
<td>0xFFF4</td>
<td>0xFFF4</td>
<td>0x07FF</td>
</tr>
<tr>
<td>10</td>
<td>0x0000</td>
<td>0xFFFFF</td>
<td>0xFFFF</td>
<td>0x07FF</td>
</tr>
<tr>
<td>11</td>
<td>0x0000</td>
<td>0x0000</td>
<td>0x0000</td>
<td>0x07FF</td>
</tr>
<tr>
<td>12</td>
<td>0x0000</td>
<td>0x0FFFF</td>
<td>0x0FFFF</td>
<td>0x07FF</td>
</tr>
<tr>
<td>13</td>
<td>0x0000</td>
<td>0x0000</td>
<td>0x0000</td>
<td>0x07FF</td>
</tr>
<tr>
<td>14</td>
<td>0x0000</td>
<td>0x0FFFF</td>
<td>0x0FFFF</td>
<td>0x07FF</td>
</tr>
<tr>
<td>15</td>
<td>0x0000</td>
<td>0x0000</td>
<td>0x0000</td>
<td>0x07FF</td>
</tr>
<tr>
<td>16</td>
<td>0x0000</td>
<td>0x0FFFF</td>
<td>0x0FFFF</td>
<td>0x07FF</td>
</tr>
<tr>
<td>17</td>
<td>0x0000</td>
<td>0x0000</td>
<td>0x0000</td>
<td>0x07FF</td>
</tr>
<tr>
<td>18</td>
<td>0x0000</td>
<td>0x0FFFF</td>
<td>0x0FFFF</td>
<td>0x07FF</td>
</tr>
<tr>
<td>19</td>
<td>0x0000</td>
<td>0x0FFFF</td>
<td>0x0FFFF</td>
<td>0x07FF</td>
</tr>
</tbody>
</table>

Fast Hilbert Transformer

A Hilbert Transformer is a filter that provides an output with phase shift of 90 degrees, for all frequencies (all-pass filter). The impulse response is estimated using Equation Error! Reference source not found.. Observation is that the values will alternate between zero and nonzero when index n is even and odd, respectively. To reduce the computational time by half, we can skip the multiplying and summation of these zero terms. There are two changes for using a fast Hilbert Transform. First, the coefficients will reduce by half. Only nonzero terms will be listed in the fircoeffs.inc file. Note that the filter coefficients are still in descending order. Second, the generic FIR code needs to increment the circular buffer pointer twice.

Code 4. Fast Hilbert Transform

```
mov [cTemp],A
mov A,X
mvi [ptr_DelayTaps],A
mov A,[cTemp]
mvi [ptr_DelayTaps],A
IncrementwithCircular (ptr_DelayTaps+0), (iDn+0), NUMFIRDELAY

;Convolution
mov [ctr_Hn], 0
mov [cTemp+0], 0
mov [cTemp+1], 0
mov [cTemp+2], 0
mov [cTemp+3], 0
mov [ctrl],ALPHA
add [ptr_DelayTaps],2
IncrementwithCircular (ptr_DelayTaps+0), (iDn+0), NUMFIRDELAY ;skip zero
STARTFastHilbert:
cmp [ctrl], 0
```
To aid in cookbook deployment of this FIR filter, we provide a code generator using a MATLAB® program. MATLAB’s minimum requirement is at least version 6. After launching MATLAB and changing to the project directory, the user types `>>runGUIPSOC` at the command line. As shown in Figure 5, five different types of FIR filters can be designed based on Hanning window method. The cut-off frequency is chosen, based on the filter type, after entering a sampling frequency. The output graph can be shown in different modes (e.g., magnitude, phase, impulse or unit step response).

More information about PSoC signal processing tools is available for free downloading at [www.virtual-dsp.com](http://www.virtual-dsp.com).

### Summary

In this note, we present the implementation of an FIR filter for PSoC. Theory and implementation issues, e.g., circular buffering and overflow, were discussed. A graphical design tool, based on MATLAB program, is provided along with this note to help PSoC developers design the filter coefficients. The output of this tool can be used directly with PSoC Designer projects.
Figure 5. FIR Graphical Design Tool
About the Authors

**Name:** Somsak Sukittanon  
**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 areas 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 and DSP embedded design. He is the recipient of the 2005 outstanding teaching award for Electrical Engineering Department at the University of Washington.

**Contact:** soma@virtual-dsp.com

**Name:** Stephen G. Dame  
**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 areas 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.

**Contact:** steve@virtual-dsp.com

In March of 2007, Cypress recataloged all of its Application Notes using a new documentation number and revision code. This new documentation number and revision code (001-xxxxx, beginning with rev. **), located in the footer of the document, will be used in all subsequent revisions.

PSoC is a registered trademark of Cypress Semiconductor Corp. “Programmable System-on-Chip,” PSoC Designer, and PSoC Express are trademarks of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are the property of their respective owners.