Digital Filter Design and Algorithm Implementation with Embedded Signal Processors

Navin Govind
Intel Corporation

Abstract

Electronic systems deal with signals that have a frequency spectrum with a wide range of frequency components. These frequency components require filter functions that may include isolation, rejection or attenuation depending on system implementation. Digital filters are systems that modify certain frequencies relative to others either by a digital computational process or by implementation of an algorithm. A band-limited signal which is continuos-time is sampled at the Nyquist rate and converted to discrete-time by a digital filter. This paper deals with the design of digital filters which include specification, approximation and realization of the desired properties of a causal discrete time system using a new generation of Intel 80C196 embedded controllers with digital signal processing capabilities. The 80C196, in addition to its register to register architecture, has a hardware based accumulator and multiply accumulate instructions suitable for signal processing. Key features of the instruction set to perform signal processing functions are described. Fast I/O operations and software implementation of digital filters in hard disk drive servo operations, optimized for performance and memory are discussed, along with a high level overview of both the architecture and software are discussed.

Signal Processing with Filters

Signals that have a frequency spectrum with a wide range of frequency components are transformed and manipulated with the input/output signal information during signal processing. Discrete time signal processing converts a continuos time signal into a sequence of samples which is a discrete time signal. Discrete time signals or digital signals appear mathematically as a sequence of numbers.

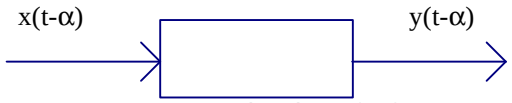
Digital filters involve processing continuos time signals using discrete time signal processing. Digital signals have an independent variable with discrete values for both time and amplitude. Filtering consists of the modification of an input signal to a desired output signal. Digital filters, unlike analog filters, can be implemented by an algorithm. A continuos time input signal which is sampled appears as a sequence of numbers and is transformed to an output signal which is a digital signal. This digital signal can be transformed back to a continuos time signal after the signal processing is completed. Filters in general are used in a variety of applications in the form of low pass filters passing the lower frequency components, high pass filters passing the high frequency components and band pass filters that pass a certain range of frequencies while attenuating other frequencies of signals that are continuos in time and are band limited. Digital filters eliminate variations normally present in the case of analog filters due to analog components. The variations may be in the form of noise and voltage variations affecting the phase and magnitude response of the filter being implemented. Presently, fast fixed point and floating point operations which are normally used to implement discrete time signal processing are available on silicon. The digital filter implemented on a single integrated circuit is an easy to use programmable microprocessor such as an 80C196NU, that follows the specified magnitude and phase response of the filter in design, accurately and consistently. Signal processing that require transformation of signals and are impossible to implement using analog components are realized in the form of digital filters on silicon.

The design of digital filters described here will be in the form of causal, time-invariant, linear systems. A system in which there is no output before the input is applied is stated to be a causal system. When an input h(t) in the form of an impulse response is applied to a system

$$h(t) = 0 \qquad \text{for all } t < 0$$

In a time-invariant system there will be a shift in the output sequence of numbers for a delay applied in the input sequence. For example, a system with input x(t) and output y(t), for a delay ‘n’ applied to

the input x(t-α) the output will be y(t-α) where α is the time shift.



TIME-INVARIANT SYSTEM

A linear system is defined by the principle of superposition. For systems with inputs $x_1(t)$ and $x_2(t)$ and outputs $y_1(t)$ and $y_2(t)$, the system is linear if

LINEAR SYSTEM

The properties described above apply to systems and are not descriptive of the input signals to the system. Another important property of a system is the stability of the system. A system is stable when for every bounded input a bounded output is generated with all poles of the system transfer function in the left half of the 'S' plane in the 'S' domain or all poles lie within the unit circle in the Z domain. Therefore if input x(t) is bounded then a fixed finite positive value of X(t) exists i.e.;

$$|x(t)| <= X(t) < \infty \qquad \text{for all t}$$

The bounded output y(t) of a stable system tends to zero for a bounded input when 't' is greater than zero. The description of linear time-invariant systems is important since these systems have important signal processing applications. The properties of linear time-invariant systems is used to represent this class of systems, more so, since filters are a significantly important part of this class.

Digital FIR Filter Design

The filter design considered in this paper will deal with causal filters. The design of a *finite duration impulse response* **(FIR)** filter also known as a nonrecursive filter will be used as a design example. The classical approach of specifying the properties of the system, approximating the specifications using the causal, time-invariant, linear and stable system and finally the realization of the filter will be taken here. Approximation is finding the transfer function of the filter with the desired frequency response of the filter in the frequency domain. Nonrecursive filter design is based on the approximation of the desired frequency response from a polynomial function. Since digital filters have a relationship between the input x(t) and the output y(t) a difference equation that establishes the input and output relationship of the filter can be derived from the rational transfer function (1)

$$H(Z) = \frac{a_0 + a_1 z^{-1} + a_1 z^{-2} \ldots\ldots\ldots a_M z^{-M}}{1 + b_1 z^{-1} + b_2 z^{-2} \ldots\ldots\ldots b_N z^{-N}} \qquad (1)$$

The difference equation derived from (1) is in the form of a linear difference equation. The difference equation realized from the transfer function is normally called a digital filter and is of the form

$$y(nT) = \sum_{k=0}^{M} a_k\, x(nT - kT) - \sum_{k=1}^{N} b_k y(nT - kT) \quad (2)$$

In the difference equation above denoted as (2), if

$b_k = 0$ for k = 1, 2, ...N, the realized filter is known as a nonrecursive filter. Therefore the equation for a FIR filter will reduce to the first half of (2)

$$y(nT) = \sum_{k=0}^{M} a_k\, x(nT - kT) \qquad (3)$$

which shows the finite length response of an FIR filter. If the filter has a unit sample input then the filter has a unit response of h(t)=a(t). To transform the continuos time signal x(t) into a discrete-time signal x(nT), the Z-transform is used. The Z-transform provides a relationship between the continuos time and discrete time signal processing. A sampled impulse signal x*(t) has a Laplace transform of X*(s) and is related to the Z-transform X(z) of a discrete-time signal x(nT) by the transformation $z = e^{st}$. The Z-transform maps the left half of the complex S-plane into a unit circle in the complex Z-plane. An analogy is used to analyze the complex Z-plane in the same way the properties of the Laplace transforms are used to analyze continuos-time systems. To complete the transfer function of a linear time-invariant system with respect to the input x(n), output y(n) and the impulse response h(n) in the Z domain

$$H(z) = \frac{Y(z)}{X(z)} \qquad (4)$$

with the Z-transform of a continuos-time signal x(t) denoted by

$$H(z) = \sum_{n=-\infty}^{\infty} h(nT)\, z^{-n} \qquad (5)$$

where "n -> from 0.....M" for a finite impulse response filter. To realize the filter, equations in the form of (3) and (4) are used and the filter is implemented using a direct form network structure.

80C196NU Implementation of Filters

The Intel 80C196NU is used to implement the direct form FIR filter realized by (3) and (4). The 196NU is a 16-bit high performance CHMOS microcontroller with the following features:
50 MHz @ 5V
Clock Doubling
16x16 multiply/accumulate
1 Mbyte Address Space
6 Programmable Chip Selects
Idle, Standby and Powerdown
3 Pulse Width Modulators
Peripheral Transaction Server
2 Timer/Counters
4 Event Processor Array
Full Duplex Serial I/O Unit

There is also an 8-byte prefetch queue implemented for increased program execution and a 32-bit accumulator peripheral for increased math performance. Saturation mode can be evoked to handle overflow and underflow when a 16x16 multiply takes place. Saturation occurs when two positive numbers generate a negative sign bit or when two negative numbers generate a positive sign bit. Saturation of this kind does not occur when this mode is enabled. The efficiency of the math performance and the fast I/O handling capability is required for signal processing applications. The memory addressing schemes are important since the input data values of x(n) and the coefficients h(n), when implementing a filter for example, are to be stored in successive memory locations for fast retrieval and computation. The powerful instruction set with the MAC and LD/ST instructions capable of autoincrement and the use of index pointers are taken advantage of to implement a FIR filter which can be optimized for code density, performance or a solution with the right balance of performance and code density.

There is generally a difference in the performance of a filter when implemented on a limited precision device as opposed to an infinite precision device due to finite word length arithmetic. The 80C196NU is a fixed point 16-bit device and the ideal filter coefficients are approximated as close as possible to include the coefficient quantization error present due to approximations. The effects due to coefficient quantization, finite word-lengths, truncation and rounding off products as well as A/D quantization noise is taken care off to a great extent by the 32-bit accumulator, shift instructions and the saturation mode enabled on the accumulator. The 80C196NU is the first member of the MCS96 architecture to have a signal processing operational instruction which can handle a multiply and accumulate with an accumulator. The logic for the accumulator includes a 16-bit adder, 3x1 multiplex logic and a 32-bit accumulator. The multiply/accumulate instruction can operate on signed-integer, unsigned-integer and signed-fractional data. The accumulator can be configured in four ways shown in Table 1

Table 1

Saturation Enable bit	Fractional Enable bit
0	**0**
0	**1**
1	**0**
1	**1**

With reference to Table 1, in the first case when both bits are zero the overflow flag is set if the sign bit of the accumulator and the signed bit of the addend are equal and the sign bit of the result is the opposite. In the second case a shift left by one bit (multiply x 2) is performed on the addend before the accumulation and the fractional accumulation proceeds as described in the first case. In the third case the 32-bit signed integer value is accumulated up or down to saturation. When positive overflow occurs during an accumulation, a value of 7FFFFFFFH is jammed into the accumulator and the saturation flag is set. When a negative overflow occurs during an accumulation a value of 80000000H is jammed into the accumulator with the saturation flag set. Accumulation proceeds normally after saturation is reached i.e. the accumulated value can decrease from positive saturation and increase from negative saturation. In the fourth case a shift left by one bit is performed on the addend before the accumulation and the

fractional accumulation proceeds as described in the third case.

Single precision filters can be implemented efficiently by choosing to round off the 32-bit numbers rather than truncation. The FIR filter coefficients are stored in windowed memory so that register direct addressing can be used for short fast-executing instruction. The 196NU memory map is shown in Table 2. The windowing feature expands the amount of memory that is accessible with register direct 8-bit addressing. The upper and lower register file and the peripheral SFR's can be windowed. The windows size can be selected to be 32-bytes, 64-bytes and 128-bytes. This limits the number of taps on the FIR filter being considered for implementation. Code can be executed from any page in the 1Mbyte address space. Internally the 196NU has 24 address lines with the lower 16-bits

Table 2 80C196NU Memory Map

ADDRESS	DESCRIPTION
FF FFFFH FF 0100H	EXTERNAL MEMORY
FF 00FFH FF 0000H	RSVD FOR ICE
FE FFFFH 10 0000H	OVERLAYED MEMORY
0E FFFFH 01 0000H	896K EXTERNAL MEMORY
00 FFFFH 00 2000H	EXTERNAL MEMORY
00 1FFFH 00 1FE0H	PERIPHERAL SPECIAL FUNCTION REGISTERS

supplied by the 16-bit data address register. The upper 8-bits which holds the page number come from sources for extended and nonextended instructions. Data can be accessed in any page with data accesses to page 00H in nonextended mode and accesses to all other pages in extended mode.

Direct-Form FIR Filter

The filter coefficients of a direct form FIR filter can be obtained from the difference equation (6).

$$y(nT) = \sum_{k=0}^{M} h(nT)\, x(nT - kT) \qquad (6)$$

The output of the FIR filter from (6) is a finite length weighted sum of the past inputs and the current input for a unit-sample response of h(nT). The filter inputs and coefficients are stored in 196NU page 00h memory from higher data address to lower data address as seen in Table 3 and Table 4. The input samples x(n) and the coefficient's h(n) can be stored in page 00h and y(n) computed for a 6-tap FIR using the relation

y(n) = x(n)h(0) + x(n-1)h(1) +..... x(n-5)h(5) (7)

The code listing for the macro version which has low code density but is execution limited and the in-line version which is optimized for fast execution but has higher code density is shown in Listing's 1 and 2 respectively. The Figure 1 shows the network structure of a direct-form 6-tap FIR filter.

Table 3

00 5FFFh	**x(n)**
.	**x(n-1)**
.	**x(n-2)**
.	**.**
.	**.**
00 4000h	**x[n-(N-1)]**

Table 4

00 AFFFh	**h(0)**
.	**h(1)**
.	**h(2)**
.	**.**
.	**.**
00 B000h	**h(N-1)**

Conclusion

The implementation of a direct-form 6-tap nonrecursive FIR filter on the 80C196NU has been described. The use of an accumulator to speed up real time computations using the multiply and accumulate instructions with the saturation mode shows techniques to approximate an ideal FIR filter coefficients to minimize the effects of coefficient quantization and finite-length arithmetic. The architectural and instruction set features for optimum response based on code density and speed of code execution of the FIR filter were summarized. Assembly code listings for the macro and in-line implementation are listed. The 80C196 device can be used to implement recursive filters with the required response and is not limited to just nonrecursive filters.

References

[1] A. V. Oppenheim and R. W. Schafer, "Discrete-Time Signal Processing," Prentice-Hall, NJ, 1989.

[2] Intel 80C196NP Microcontroller User's Manual

[3] Intel 80C196NU Supplement to the 8XC196NP Microcontroller User's Manual

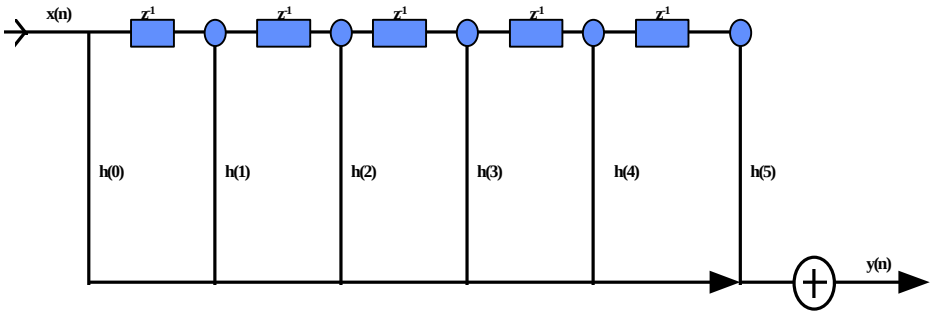


Figure 1. Direct Form FIR Filter

Listing 1

```
;FIR Filter Implementation in-line using the following equation
;For an impulse response of h(0), h(1).... h(N-1) and input x(n) at time 'n', the output
;y(n) at time n is given by: y(n) = h(0)*x(n) + h(1)*x(n-1)+....+h(N-1)*x[n-(N-1)]

        RSEG  AT  040h                  ;Register segment starts at 40h
        CSEG AT 0FF2080H                ;Code segment starts at 0FF 2080h
BEGIN:
FIR:    LD      sample,XIN              ;INPUT NEW SAMPLE
        MAC     coeff,sample            ;DO INITIAL MPY, INIT ACC
        RPT     #00EH                   ;REPEAT NEXT INSTR 15X
        MAC     coeff+,sample+          ;DO 15 SUCCESSIVE MAC'S
        ST      acc,ylast[00]           ;PLACE RESULTS IN YLAST
DONE:   SJMP    DONE                    ;WAIT FOR NEXT SAMPLE
        END
```

Listing 2

```
;FIR Filter macro implementation
        RSEG  AT  040h                  ;Register segment starts at 40h
        CSEG AT 0FF2080H
fir macro coef, samp, num
        MAC     accum, coef&num, samp&num
endm
        LD      loop,#15
        LDBSE   sample,#XIN
BEGIN:
        irp  accum_count, <0,1,2,3,4,5>
        fir  coeff_, sample_, accum_count
endm
        MAC     coeff+,sample+          ;h(N-1) * x(n-(N-1))
        ADD     sample,#5               ;ACCUMULATE LAST PRODUCT
```

```
DONE:                                   ;RESULTS IN ACCUMULATOR
        SJMP    DONE
END
```

Filename:	DSP_95.DOC
Directory:	C:\TESTDOCS\DOCS
Template:	C:\WINDOWS\WINWORD6\TEMPLATE\NORMAL.DOT
Title:	Introduction to the Next Generation MCS 96 Microcontroller
Subject:	
Author:	Navin Govind
Keywords:	
Comments:	
Creation Date:	07/05/95 10:56 AM
Revision Number:	62
Last Saved On:	07/28/95 1:28 PM
Last Saved By:	Navin Govind
Total Editing Time:	769 Minutes
Last Printed On:	12/18/95 3:58 PM

As of Last Complete Printing

Number of Pages:	6
Number of Words:	2,486 (approx.)
Number of Characters:	14,171 (approx.)