SIEMENS

Microcontrollers

ApNote AP1601

☐ additional file
APXXXX01.EXE available

Biquad IIR Filter - C166 Family with Signal Processing Capabilities

Many applications like disk drives need both, controller and signal processing capabilities.

K. Westerholz / Siemens HL MCB PD

AP1601 ApNote - Revision History		
Actual Revision : Rel.01	Previous Revison: Rel. none	
Page of actual Rel.	Page of prev. Rel.	Subjects changes since last release)

1 Abstract

Many applications like disk drives need both, controller and signal processing capabilities. For instance Infinite Impulse Response (IIR) filters are an application today performed by dedicated signal processors. They are capable of implementing rational transfer functions as required by band pass filters.

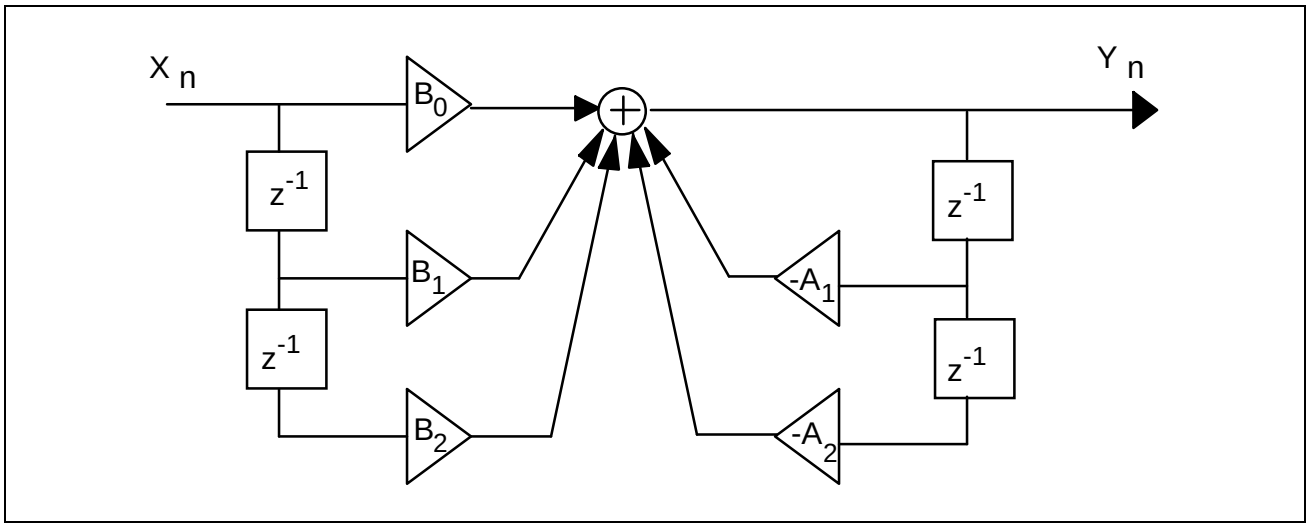

Figure 1:
Second-order Biquad IIR Filter Section

2 Transfer function / algorithm

The second order biquad IIR filter shown in the figure above and discussed below realizes the transfer function:

$$H(z) = \frac{B_0 + B_1 \cdot z^{-1} + B_2 \cdot z^{-2}}{1 + A_1 \cdot z^{-1} + A_2 \cdot z^{-2}}$$

where A1, A2, B0, B1, and B2 are coefficients that determine the desired impulse response. Furthermore, the corresponding difference equation for a biquad section is:

$$Y_n = B_0 \cdot X_n + B_1 \cdot X_{n-1} + B_2 \cdot X_{n-2} - A_1 \cdot Y_{n-1} - A_2 \cdot Y_{n-2}$$

This equation can be directly translated into a digital filtering algorithm. Higher-order filters can be obtained by cascading several biquad sections with appropriate coefficients. The C166 family of processors offers a fast multiply and divide unit that delivers results every 500 ns. Furthermore the C166 possesses a register file of 16 general purpose registers for each task. That allows to execute the biquad filter equation in 5.7us. The C166 family is therefore suitable for performing advanced signal processing tasks alongside other controller tasks. This application note demonstrates how to implement a biquad IIR filter by exploiting the signal processing capabilities of the C166 family. In combination with the algorithm depicted here the C166 architecture is suitable for sampling rates up to 160 KHz.

This application note addresses three problems:

* How does the processor store the filter coefficients and state variables within its memory ?
* How does the processor tackle the problem of register overflows?
* How do we implement the filter equation tailored to the capabilities of the C166 family ?

The algorithm presented assumes coefficients and variables normalised to the interval of [1,1) respectively [8000h,7FFFh]. This representation guarantees that the most significant bits also contain the leading part of the numbers. The interval is translated to the corresponding 16 bit hex representation by multiplying the fractional numbers by 32768 (8000h). Afterwards negative numbers have to be converted to their 2's complement. This representation means that the 16th bit indicates the sign and the implied radix point is located right behind the sign bit. Supposing we perform a signed multiplication the result will be a 32 bit signed number stored in the register pair of the multiply and divide unit, MDL and MDH. After the multiplication has been executed the implied radix point will be located between bit 14 and 13 of the MDH register. In order to adjust the imaginary radix point again right behind the sign bit a shift left is necessary.

In contrast, add instructions for accumulating the results do not affect the format but attention has to be paid to register overflows. Examining the overflow problem we see that two coefficients are positive and two are negative hence the maximum overflow would only amount to one bit. If we suspend the shift left to adjust the radix point till all multiplication results have been accumulated, we won't lose the most significant bit because it is stored in bit 30.

3 Program flow of the filter algorithm

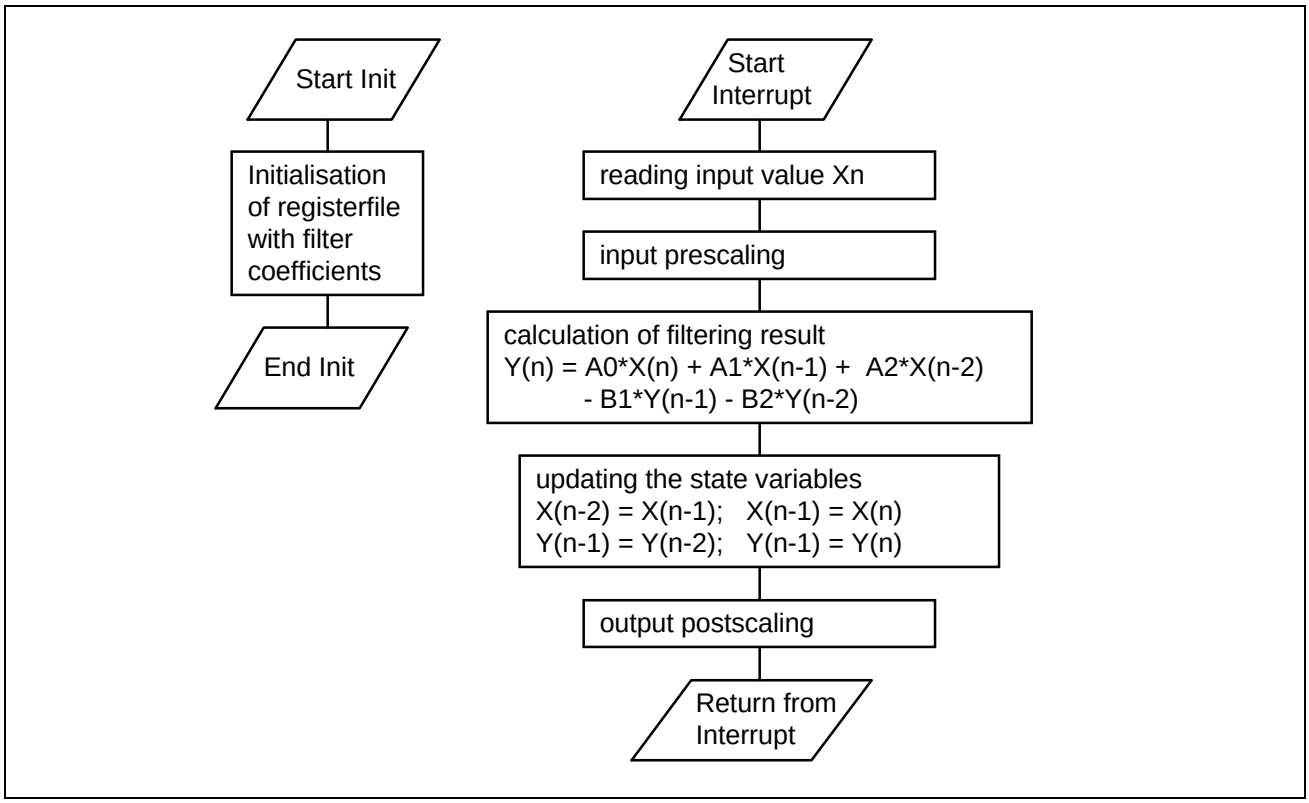

Figure 2:
Flow Chart Diagram of the Filter Algorithm

The algorithm itself comprises an initialization phase to preset the register file with the coefficients and a separate computing phase that is invoked by an interrupt. The input value is taken from a given memory location for instance the AD Converter's register. This value is then prescaled to reduce quantisation errors and after the computations it is postscaled.

The calculation of the output itself is performed by four 16 bit multiplications and the 32 bit results are accumulated in an general purpose register pair that contains the filter result. Due to the signed multiplication the accumulated 32 bit result has to be shifted one bit left to adjust the radix point. This operation only has to be performed once after all results have been accumulated. The general purpose register that contains the most significant bits is then used for updating the state variables. This has the effect that the 32 bit result is truncated to its 16 most significant bits. After a postscaling step the truncated result is stored to memory. Supposing all state variables and coefficients are kept in the register file and the code is executed out of the internal ROM an execution time of 5.7 us will be achieved for a C167 with 20 MHz. Assuming the program is executed out of an external memory accessed via a 16 bit non multiplexed bus without wait states, then the execution time amounts to 7.3 us. If several biquad filters are concatenated in order to get higher order filters, the register file can be easily switched by adjusting the context pointer to a further register file.

This application is directly coded in assembler because C as a HLL does not support the fixed point arithmetic required for an efficient implementation. Example code listing for the C167 (medium memory model):

4 Example code listing

```
;*****************************************************************
;* IIR Band Pass Filter                                          *
;*                                                               *
;* This example comprises an interrupt routine that performs the *
;* filtering algorithm and a main program that initializes the   *
;* register file with the filter coefficients                    *
;* The intention of this program is that the  AD Converter       *
;* runs continuously providing the input. The filter output      *
;* is stored to a memory location called _Yn. Each time a        *
;* conversion is completed an interrupt is generated that        *
;* invokes the _band_pass_filter interrupt routine.              *
;* The medium memory model is assumed in order to enable an      *
;* integration of this example into larger applications.         *
;*****************************************************************


$EXTEND
$NOMOD166
$STDNAMES(reg.def)
$SEGMENTED
$CASE
$MODEL(MEDIUM)

; definition of storage for the filter output Yn
    NAME    BPF
    ASSUME  DPP3:SYSTEM
BPF_1_NB    SECTION DATA WORD PUBLIC 'CNEAR'
    ASSUME  DPP2:BPF_1_NB
BPF_1_NB_ENTRY LABEL   BYTE
_Yn LABEL   WORD
    DS  4
    PUBLIC  _Yn
_Xn LABEL   WORD
    DS  2
    PUBLIC  _Xn
BPF_1_NB    ENDS

; definition of storage for the filter coefficients within the FARROM
; initialization of the coefficients B1, B2, A0, A1, A2 with arbitrarily
; chosen values

BPF_2_FC    SECTION DATA WORD PUBLIC 'CFARROM'
_B0 LABEL   WORD
    DW  05h     ;B0 = 0.0001526
    PUBLIC  _B0
_B1 LABEL   WORD
    DW  03h     ;B1 = 0.0000916
    PUBLIC  _B1
_B2 LABEL   WORD
    DW  01h     ;B2 = 0.0000305
    PUBLIC  _B2
_A1 LABEL   WORD
    DW  0FFFCh      ;A1 = -0.0001221
```

```
    PUBLIC  _A1
_A2 LABEL   WORD
    DW  0FFFEh      ;A2 = -0.0000610
    PUBLIC  _A2
BPF_2_FC    ENDS

;*************************************************************
;*                                                           *
;* Main program for pre-setting the register bank BPF_RB     *
;* with filter coefficients                                  *
;*                                                           *
;*************************************************************
    PUBLIC  _main

BPF_3_PR    SECTION CODE WORD PUBLIC 'CPROGRAM'
_main   PROC    NEAR

    MOV BUSCON0,#04BFh ;0 wait states, non multiplexed 16 bit
    MOV ADCON,#013h ;Single Channel Continuos, Channel 3
    MOV ADCIC,#044h ;Interrupt enable ADC, interrupt level 1
                ;group level 0
    MOV DPP3:BPF_RB,R0 ;Initialize register R0 of register
                ;bank BPF_RB with user Stack Pointer
    SCXT    CP,#DPP3:BPF_RB; Switching to the register bank of the
                ; interrupt routine "band_pass_filter"
    SCXT    DPP0,#PAG _B0
; initializing the register bank for the very first time with initial
; values for the state variables and the coefficients
    MOV R4,#00h     ;Yn_1 State variable in Q15 format
                ;decimal range -1,1
    MOV R7,#00h     ;Yn_2
    MOV R2,#00h     ;Xn_1
    MOV R3,#00h     ;Xn_2
    MOV R12,POF _B0 ;B0
    MOV R13,POF _B1 ;B1
    MOV R14,POF _B2 ;B2
    MOV R10,POF _A1 ;A1
    MOV R11,POF _A2 ;A2
    POP DPP0
    POP CP
    BSET    ADST        ; start AD Converter  ADST  = 1
    BSET    IEN     ; global interrupt enable IEN = 1
    ...         ; further application specific code
    RET
_main   ENDP
```

```
;*************************************************************
;*                                                           *
;* Interrupt task "band_pass_filter" with interrupt number 028h  *
;* It calculates the difference equation :                   *
;* It calculates the difference equation :                   *
;* Yn = B0*Xn + B1*Xn_1 + B2*Xn_2 + A1*Yn_1 + A2*Yn_2        *
;*                                                           *
;*************************************************************
_band_pass_filter  PROC TASK BPF_TASK INTNO BPF_INUM = 028h
    MOV DPP3:BPF_RB,R0     ;Initialize register R0 of register
                    ;bank BPF_RB with user Stack Pointer
    SCXT    CP,#DPP3:BPF_RB;save registers affected by the
    ;interrupt routine
    SCXT    MDC,#00h
    SCXT    DPP0,#PAG _Yn
    PUSH    MDL
    PUSH    MDH
    MOV R1,_Xn          ;_Xn contains the AD conversion result
    ;provided by ADDAT register

; calculation of the filter eqation
;  Yn = B0*Xn + B1*Xn_1 + B2*Xn_2 + A1*Yn_1 + A2*Yn_2;

    SHL R1,#05h     ; prescaling of _Xn by 2^5
                ; simultaneously this has the effect that the
                ; channel number provided in combination with
                ; the AD conversion result is masked out
    MUL R12,R1      ; B0*Xn
    MOV R4,MDL      ; register R4 and R5 are containing the 32 bit
                ; result
       MOV  R5,MDH      ; for the first time the previous content of R4
                ; and R5 will be overwritten
    MUL R13,R2      ; B1*Xn_1
    ADD R4,MDL      ; the multiplication result is added to the
    ADDC    R5,MDH      ; register pair R4 and R5
    MUL R14,R3      ; B2*Xn_2
    ADD R4,MDL      ; the multiplication result is added to the
    ADDC    R5,MDH      ; register pair R4 and R5
    MUL R10,R6      ; A1*Yn_1
    ADD R4,MDL      ; the multiplication result is added to the
    ADDC    R5,MDH      ; register pair R4 and R5
    MUL R11,R7      ; A2*Yn_2
    ADD R4,MDL      ; the multiplication result is added to the
    ADDC    R5,MDH      ; register pair R4 and R5
    MOV R7,R6       ;Yn_2 = Yn_1, update of the state variable Yn_2
    ADD R4,R4       ;one bit shift left to adjust the radix point
    ADDC    R5,R5       ;between bit 30 and 31
    MOV R6,R5       ;Yn_1 = Yn, due to the value range of -1,1
                ;only the MSW is stored to Yn_1
    ASHR    R5,#05h     ;postscaling of filter output by 2^5
    MOV _Yn,R5      ;due to the value range of -1,1
                ;only the MSW is stored
    MOV R3,R2       ;Xn_2 = Xn_1, update of the state variable Xn_2
    MOV R2,R1       ;Xn_1 = Xn, update of the state variable Xn_1
    POP MDH     ;restore saved registers
    POP MDL
    POP DPP0
    POP MDC
    POP CP      ; switch to old context
    RETI            ; return from interrupt
```

```
_band_pass_filter  ENDP
BPF_3_PR    ENDS

C166_BSS    SECTION DATA WORD GLOBAL 'CINITROM'
    DW  06h
    DPPTR   BPF_1_NB_ENTRY
    DW  0Eh
C166_BSS    ENDS

    EXTERN  __CSTART:FAR
C166_DGROUP DGROUP  BPF_1_NB
BPF_RB  REGDEF  R0-R15
    END
```