Architectural Description of Embedded Signal Processors and DSP Algorithm Implementation

Navin Govind
Intel Corporation
Chandler AZ 85226
navin_govind@ccm.ch.intel.com

ABSTRACT

High speed calculations consisting of arithmetic and fast input/output operations are handled well by standard embedded signal processors. This paper describes the pipeline architecture of a high performance embedded signal processor and the register to register architectural feature that speeds up processing and minimizes context switching. To summarize the architectural features, the design and implementation of a FIR filter and PID routines are described.

ARCHITECTURAL OVERVIEW

The 80296SA device used as an example in this study has a pipeline architecture with four stages. Fetch, decode, read/execute and execute/write. This design achieves a faster throughput of instructions than devices without a pipeline architecture. The device has a Central Processing Unit (CPU) that connects to both an interrupt controller and a memory controller. The CPU has both 16-bit and 32-bit functions. The multiply/accumulate unit is 40-bits wide. An extension of the CPU bus connects the CPU to the internal peripheral modules. The 80296SA has a 24-bit internal address bus and bonds out 20 address lines, therefore this device can physically address 1 Mbytes of memory. The chip select logic matches the 24 bit address and can map up to 6 memory regions, each with 1-Megabyte of address space. The device has 512 bytes of register RAM and 2K bytes of code RAM, register to register architecture, serial port with a synchronous baud rate of 12.5 Mbaud maximum and a maximum asynchronous baud rate of 3.1 Mbaud, a Pulse Width Modulator (PWM) with a 195.3 KHz maximum frequency, an Event Processor Array (EPA) with 80ns resolution. A phase lock loop (PLL) allows an external clock to drive the device at one half or one quarter the maximum internal clock frequency. In a pipeline architecture, a different instruction is in each of the four pipeline stages. One instruction moves sequentially through the pipeline stages. As one instruction moves from the fetch to the decode stage, the next instruction moves into the fetch stage. Similarly, the previous two instructions are in the read/execute and execute/write stages. In summary, a pipeline architecture means that four instructions are in one of the four stages of execution at any one time. The size of the on-chip register RAM is 512 bytes and an additional 2kbytes of code/data RAM in address region 0F800h to 0FFFFh in extended mode (1Mbyte addressing mode) or in small memory mode it is mapped into address region 0FFF800h to 0FFFFFF (64kbyte addressing mode). This RAM may be used for time critical code such as interrupt service routines, time critical data such as digital signal processing routines, data tables, stack, or interrupt vector table. The user must determine appropriate allocation of this RAM for the system's time critical use. The 80296SA CPU does not stall instruction execution due to its four-stage pipeline architecture. It will continue execution of the instructions in the other three stages of the four stage pipeline.

INSTRUCTION SET MODIFICATIONS

Several instructions implemented in the CPU handle embedded digital signal processing routines efficiently and manipulates the 40-bit accumulator. The basic functions of the accumulator instructions include:

- clearing the accumulator before execution (indicated by 'Z' suffix)
- relocating the source within a data table (indicated by 'R' suffix)
- signed or unsigned math (signed indicated by 'S' prefix)
- saturating the accumulator value based on the result.

The opcodes for the multiply-accumulate instructions have the same opcodes as the multiply instructions. The CPU differentiates the instructions as follows:

- If the destination operand is less than 10h, the CPU executes one of the multiply-accumulate instructions.
- If the destination operand is greater than or equal to 10h, the CPU executes one of the multiply instructions.

This convention works because addresses below 17h are special function registers (SFRs), and therefore should never be the destination of a multiply instruction. The assembler translates the multiply-accumulate mnemonic into the appropriate opcode and destination operand. Additionally, the repeat (RPT) instructions make handling repeated multiply-accumulates easier to implement in code. Also the return from interrupt (RETI) instruction reduces interrupt latency. The opcode for the repeat instruction is the same opcode as the AND instructions. The CPU differentiates the instructions by the destination operand.

ACCUMULATOR

The device under study has a signal processing operational instruction which can handle a multiply and accumulate with an accumulator. The multiply/accumulate instruction can operate on signed-integer, unsigned-integer and signed-fractional data. 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. Single precision filters can be implemented efficiently by choosing to round off the 32-bit numbers rather than truncation. FIR filter coefficients are stored in windowed memory so that register direct addressing can be used for short fast-executing instruction. 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 window size can be selected to be 32-bytes, 64-bytes and 128-bytes. Code can be executed from any page in the address space supplied by the 16-bit data address register. The upper 8-bits which holds the page number come from sources for extended and non-extended instructions. Data can be accessed in any page with data accesses to page 00H in non-extended mode and accesses to all other pages in extended mode.

INTERRUPTS

After reset the device and peripheral interrupts operate in the default priority mode, but the interrupt structure may be modified to support a hardware prioritized model. The inclusion of a programmable priority scheme assigned to the actual interrupt vectors ensures that when the device responds to an interrupt and enters that interrupt service routine, no interrupt of a lower or equal priority can interfere with that routine. The first instruction following an interrupt may not be interrupted so it is still possible to disable all

interrupts during the service routine. But, when interrupts are left enabled, only interrupts with a higher numerical priority can interrupt an active routine. Each interrupts can be assigned to a fixed priority/vector pair, i.e., each interrupt channel has a 16-way multiplexer in front of it and a selection register that assigns an interrupt source. This flexibility of assignment is typically used during initialization, not for dynamic alteration of interrupt selections and/or priorities. Pending interrupts are latched at the vector priority input before the interrupt mask. A benefit of this change is that the interrupt mask register may not need to be passed to the interrupt service routine, on the stack, so that the masks can be adjusted during that service routine. The Interrupt Unit supports positive and negative edge sensitive external interrupts and the addition of level sensitive external interrupts.

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 non recursive 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. Non recursive 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)

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

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 (2)

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

In the difference equation above denoted as (2), if $b_k = 0$ for k = 1, 2, ...N, the realized filter is known as a non recursive filter. Therefore the equation for a FIR filter will reduce to the first half of (2)

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

Now y(nT) in (3) 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) is given by (4)

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

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.

IMPLEMENTATION OF FILTERS

The 80296SA is used to implement the direct form FIR filter realized by (3) and (4). 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 instruction set with the MAC, RPT and LD/ST instructions capable of auto increment 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 device under discussion 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.

DIRECT-FORM FIR FILTER

The output of the FIR filter 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 page 00h memory from higher data address to lower data address. 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

$$y(n) = x(n)h(0) + x(n-1)h(1) + \ldots\ldots x(n-5)h(5) \qquad (6)$$

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 1 for the MCS® Kx/Nx version in comparison to the 80296SA version.

PID IMPLEMENTATION

The transfer function of a PID loop has individual proportional, integral and differential terms which appear as a summed output. As initial conditions, it is assumed that the sampling time, constant and any scaling factor used will remain unchanged. A typical PID loop consists of a plant, error sensor, actuator and a controller. The controller in this case is a embedded signal processor. Analog processes can be implement using a digital control system in the form of a software algorithm. The PID routine can be implemented using equation (7). The PID algorithm must be converted to discrete values for implementing the algorithm on a microprocessor with the rectangular approximation given by:

$$y(n) = y(n-1) + G1*e(n) + G2*e(n-1) + G3*e(n-2) + G4*e(n-3) \quad (7)$$

The code listing for the PID implementation is shown in Listing 2 and the control loop is described in Figure 1

CONCLUSION

The implementation of a direct-form 6-tap non recursive FIR filter and a PID routine was 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. Coefficient quantization and finite-length arithmetic data handling was described. The architectural and instruction set features for optimum response, based on code density and speed of code execution of the FIR filter and were summarized. The 80296SA device can be used to implement recursive filters, PID routines and other complex algorithms that require digital signal processing functionality and features. In the case of filters, coefficient quantization effects, stability and characteristics of the phase response should be given special attention.

BACKGROUND

The Intel MCS® 96 microcontrollers are 16-bit CMOS devices that are designed to handle high-speed calculations and fast I/O operations. The 80296SA is the latest addition to the MCS® 96 controller family. The complete core redesign enhances the performance of the 16-bit CHMOS microcontroller as well as maintain the binary code compatibility. The 80296SA is pin compatible with the 8XC196NP and the 80C196NU. The 80296SA has 512 bytes of register RAM and 2Kbytes of code RAM, register to register architecture, serial port with a maximum synchronous baud rate of 12.5 Mbaud and a maximum asynchronous baud rate of 3.1 Mbaud, Pulse Width Modulator (PWM) with a 195.3 KHz maximum frequency, Event Processor Array (EPA) with 80ns resolution. A phase lock loop (PLL) allows an external clock to drive the device at one half or one quarter the maximum internal clock frequency. The total addressable space using the chip selects is 6 Mbytes with 1 Mbyte maximum address space for each of the six chip selects. There are five I/O ports sharing a common pin layout with the 8XC196NP and the 80C196NU. The interrupt controller on the 80C296SA can be programmed to perform like previous MCS® 96 microcontrollers or it can be programmed with prioritized interrupts. The programmable interrupt priority and vector sources are assigned to various interrupts with fixed priorities.

REFERENCES

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

[2] Intel 80296SA Microcontroller User’s Manual, No. 272803

[3] N. Govind, “Digital Filter Design and Algorithm Implementation with Embedded Signal Processors”, Proceedings of the 6th International Conference on Signal Processing Applications and Technology, Oct 24-26, 1995, pp. 551-555

[4] N. Govind and A. R. Hasan, “Real Time Fuzzy Logic Speed Control Using Conventional, Assembly and Simulation Methods for Industrial DC Motors,” in Proceedings of the 1995 IEEE/IAS International Conference on Industrial Automation and Control, IA&C ’95, Library of Congress No. 94-77908, pp. 203-208

80296SA BLOCK DIAGRAM

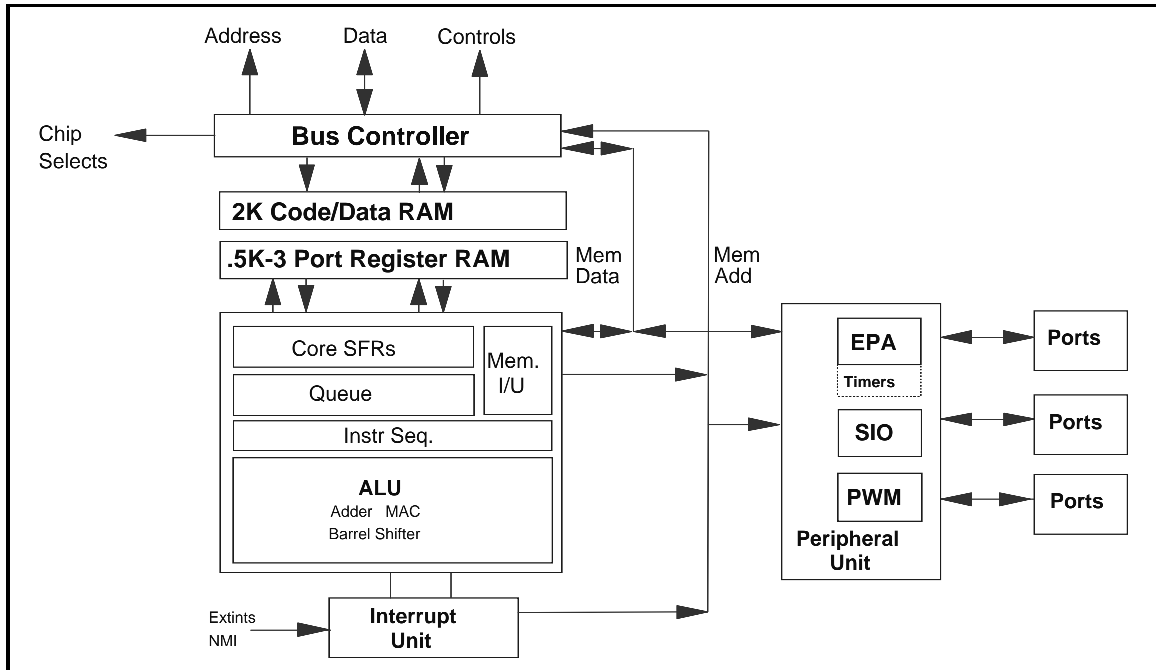


FIGURE 1. DIRECT FORM FIR FILTER

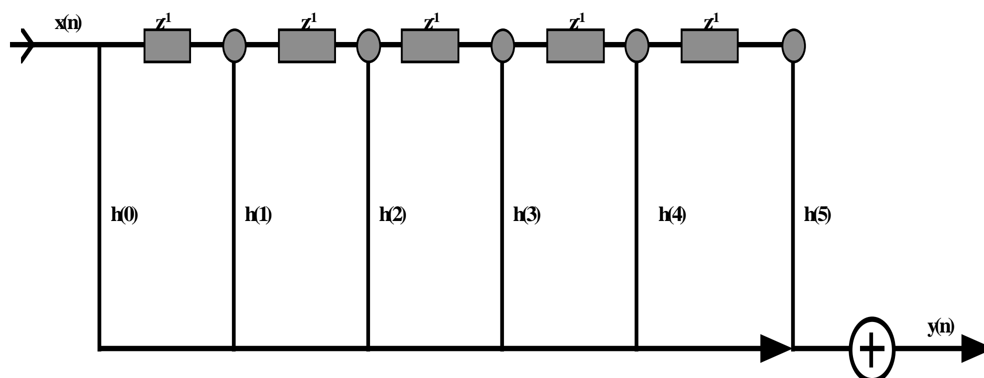
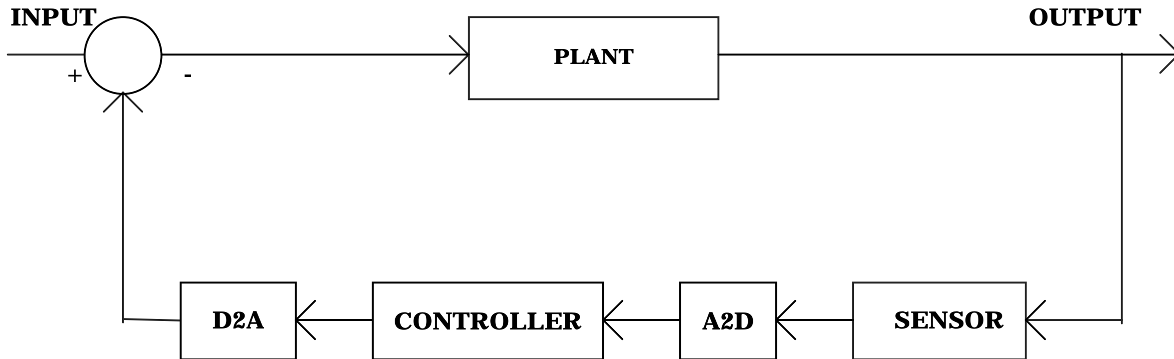


FIGURE 2. PID CONTROL LOOP



LISTING 1

```
;*****************************************************************
;* FIR 16-tap filter 80296SA  (State Time = 40nS)
;* By: Navin Govind
;* Intel Corporation
;*****************************************************************

;Perform FIR calculations 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)]


;          FIR FILTER USING MAC ("DEDICATED" FIR)

;This program illustrates the use of the MAC instruction in the implementation  of an FIR digital filter.

;The coefficient here were simply chosen as a portion of a ramp, 16 taps from 0 to 15.  This was done not to illustrate
;a useful filter, but to make it easier to recognize the coefficients and trace them through the memory sections
;where they appear.

;The program is for the 80296SA, and can be assembled with the Version 5 cross-assembler from Tasking.  After
;linking with LINKER, and conversion with OH196, the object file can be loaded and run on the 80296SA evaluation
;board.

;Another example using index registers is as follows

;      START:
;           LDB        ICB0,#01H          ;SET UP INCREMENT CONTROL BYTE REG
;           LDB        ICB1,#01H
;           LD         IDX0,SAMPLE        ;INIT SAMPLE POINTER
;           LD         IDX1,COEFF         ;INIT COEFFICIENT POINTER
;           SMACZ      ICX0,ICX1          ;DO INITIAL MPY, ZERO ACC
;           RPT        #0EH               ;REPEAT NEXT INSTR 15X
;           SMACR      ICX0,ICX1          ;DO 15 SUCCESSIVE MAC'S WITH INC
;           MSAC       YOUT,#018H         ;PLACE RESULTS IN YOUT
;      END

$include(_SFR_INC_)

YOUT       equ   0
XIN        equ   1
```

```
ERR        equ   2
YLAST      equ   3

           RSEG  AT  020h

;Working register declarations

temp:    dsw   2
temp_l   equ   temp
temp_h   equ   temp+2

out:     dsw   2
out_l    equ   out
out_h    equ   out+2

;Length of filter (N) = 15
loop:    dsb   1

;Locations of x(n-(N-1))
coeff:   dsw   2
coeff_0  equ   coeff
coeff_1  equ   coeff+2
coeff_2  equ   coeff+4
coeff_3  equ   coeff+6
coeff_4  equ   coeff+8
coeff_5  equ   coeff+10
coeff_6  equ   coeff+12
coeff_7  equ   coeff+14
coeff_8  equ   coeff+16
coeff_9  equ   coeff+18
coeff_10 equ   coeff+20
coeff_11 equ   coeff+22
coeff_12 equ   coeff+24
coeff_13 equ   coeff+26
coeff_14 equ   coeff+28

;Locations of h(n-1)
sample:  dsw   2
sample_0 equ   sample
sample_1 equ   sample+2
sample_2 equ   sample+4
sample_3 equ   sample+6
sample_4 equ   sample+8
sample_5 equ   sample+10
sample_6 equ   sample+12
sample_7 equ   sample+14
sample_8 equ   sample+16
sample_9 equ   sample+18
sample_10 equ   sample+20
sample_11 equ   sample+22
sample_12 equ   sample+24
sample_13 equ   sample+26
sample_14 equ   sample+28
;

CSEG

;MCS(r) 96 family of microcontrollers reset after power-up at 2080h
        CSEG AT 0FF2080H
        ejmp        START

;The following four 'clr' instructions may not be needed for 80926SA if the contents of temp and out registers are not
;read initially

START:
        ld          sp,#STACK                   ;set up stack depth
```

```
        ldb     wsr,#1Eh                    ;set 128 byte window

        clr     temp_l                      ;initialize temp register
        clr     temp_h

fir macro coef, samp, num
        mul     temp, coef&num, samp&num
        add     out_l, temp_l
        addc    out_h, temp_h
endm

        ldb     loop,#15
        ldbse   sample_0,#XIN

;Kx and Nx implementation
;+++++++++++++++++++++++++++++ FIR Calculation +++++++++++++++++++++++++++++++++
BEGIN:

irp  temp_count, <0,1,2,3,4,5,6,7,8,9,10,11,12,13,14>
    fir  coeff_, sample_, temp_count
endm

        mul     temp,coeff_0,sample_0       ;h(N-1) * x(n-(N-1))
        add     out_l,temp_l                ;accumulate
        addc    out_h,temp_h

        shll    temp,#1
        ld      out,temp                    ;save final output
        sjmp    fir_optz

;+++++++++++++++++++++++++++++ FIR Calculation +++++++++++++++++++++++++++++++++

;80296SA implementation
;+++++++++++++++++++++++++++++ FIR Calculation +++++++++++++++++++++++++++++++++
fir_optz:
        smacz   coeff,sample                ;DO INITIAL MPY, INIT ACC
        rpt     #0eh                        ;REPEAT NEXT INSTR 15X
        smacr   coeff+2,sample+2            ;DO 15 SUCCESSIVE MAC'S
        mvac    YOUT,#18                    ;ROTATE ACC INTO YOUT, BIT-18 = BIT-15
;+++++++++++++++++++++++++++++ FIR Calculation +++++++++++++++++++++++++++++++++

DONE:
        sjmp  DONE

END
```

LISTING 2

```
;*****************************************************************
;* PID routine for 80296SA  (State Time = 40nS)
;* By: Navin Govind
;* Intel Corporation
;*****************************************************************

; Perform PID calculations using the following equation

; The PID algorithm must be converted to discrete values for implementing the algorithm on a microprocessor
; The rectangular approximation is given by:
; y(n) = y(n-1) + G1*e(n) + G2*e(n-1) + G3*e(n-2) + G4*e(n-3)

$include(_SFR_INC_)

        RSEG  AT  040h
```

```
;Working register declarations

out:                dsw     2
out_l               equ     out
out_h               equ     out+2


temp:               dsw     2
temp_l              equ     temp
temp_h              equ     temp+2


EN:       dsw       1       ;
ENM1:     dsw       1       ; Previous error sample
ENM2:     dsw       1       ; Latest error sample
ENM3:     dsw       1       ; Oldest error sample

G1:       dsw       1       ; Constant for gain
G2:       dsw       1       ; constant for gain
G3:       dsw       1       ; Constant for gain
G4:       dsw       1       ; Constant for gain

;NOTE: The fraction notational bit in the respective SFR (special function register) must be set so that a shift left
;(shll) need not be done after every multiply instruction. If the afore mentioned SFR bit is not set a 'shll' instruction
;must follow a 3-op multiply to preserve data coherency.


;MCS(r) 96 family of microcontrollers reset after power-up at 2080h
          CSEG AT 0FF2080H
          ejmp      START

;The following 'clr' instructions may not be needed for 80926SA if the contents of temp and out regs are not read
;initially

START:
          ld        sp,#STACK                   ;set up stack depth
          ldb       wsr,#1Eh                    ;set 128 byte window

          clr       temp_l                      ;initialize temp register
          clr       temp_h


;++++++++++++++++++++++++++++ PID VARIABLES +++++++++++++++++++++++++++++++++++++
          ld        out_l,#000ah
          ld        out_h,#000ch

          clr       temp_l                      ;initialize temp register
          clr       temp_h

          ld        ENM3,#000ah
          ld        ENM2,#000bh
          ld        ENM1,#000ch
          ld        EN,#000dh

          ld        G1,#000ah
          ld        G2,#000bh
          ld        G3,#000ch
          ld        G4,#000dh

;Kx and Nx implementation
;++++++++++++++++++++++++++++ PID +++++++++++++++++++++++++++++++++++++++++++++++
BEGIN:
          mul       temp,G1,EN                  ;multiply and accumulate      (States = 16+4+4 )
          add       out_l,temp_l
          addc      out_h,temp_h
```

```
            mul     temp,G2,ENM1
            add     out_l,temp_l
            addc    out_h,temp_h

            mul     temp,G3,ENM2
            add     out_l,temp_l
            addc    out_h,temp_h

            mul     temp,G4,ENM3
            add     out_l,temp_l
            addc    out_h,temp_h

;80296SA implementation
;++++++++++++++++++++++++++++++++++++++++++++++ PID ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

            smacz   G1,EN               ; Saturated multiply and accumulate    (States = 2)
            smac    G2,ENM1             ; Saturated multiply and accumulate
            smac    G3,ENM2             ; Saturated multiply and accumulate
            smac    G4,ENM3             ; Saturated multiply and accumulate

;++++++++++++++++++++++++++++++++++++++++++++++ PID ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
DONE:
            sjmp    DONE                ; endless loop

END
```