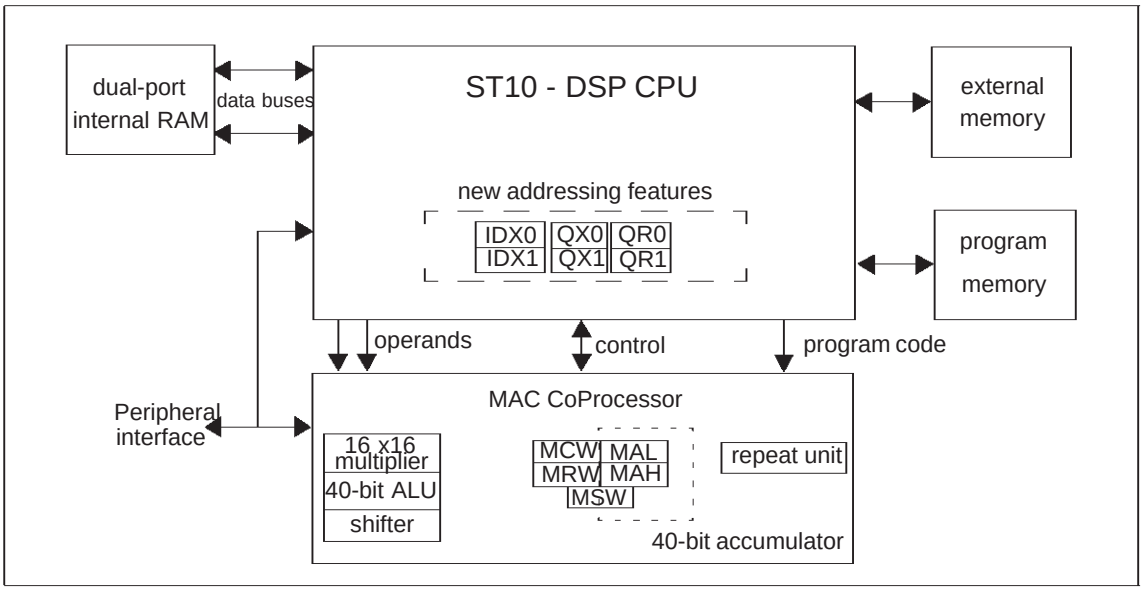
ST10 DSP MAC SIGNAL PROCESSING ALGORITHMS

The ST10 multiply-accumulate co-processor (MAC) performs common signal processing functions. The MAC carries out single-cycle instructions including 32-bit signed arithmetic (addition, subtraction, shift,...), 16 by 16-bit multiplication, and multiplication with cumulative subtraction/addition. The MAC includes the following components: 16 by 16 signed/unsigned parallel multiplier, scaler (one-bit left shifter), 40-bit signed arithmetic unit, 40-bit accumulator register, data limiter, 8-bit left/right shifter and a repeat unit.

A full description of the MAC co-processor, including the registers and instruction summary is given in the ST10R262 datasheet and the ST10R262/272l User's Manuals.

This application note describes how to use these common signal processing algorithms, using digital filters and matrix operations as examples. The sample codes contained in this application note can be cut and pasted into your application from the pdf document format..



Contents

1 Co-Processor Initialization

This routine initializes the co-processor registers:

```
;
; Control Registers Initialization.
;
MOV     MCW,    #mcw        ; (MCW) ← mcw.
MOV     MRW,    #mrw        ; (MRW) ← mrw.
;
; Accumulator Initialization.
;
MOV     MAH,    #data16     ; (MAH) ← #data16,
                            ; (MAE) ← 8 times (MAH15),
                            ; (MAL) ← 0000H.
;
; Core SFRs Initialization.
;
EXTR    #6                  ; Next 6 instructions will utilize the ESFR space.
MOV     IDX0,   #idx0       ; (IDX0) ← idx0.
MOV     IDX1,   #idx1       ; (IDX1) ← idx1.
MOV     QX0,    #qx0        ; (QX0) ← qx0.
MOV     QX1,    #qx1        ; (QX1) ← qx1.
MOV     QR0,    #qr0        ; (QR0) ← qr0.
MOV     QR1,    #qr1        ; (QR1) ← qr1.
```

	Instruction Cycles	Program Words
Total	10	19

2 Mathematics

2.1 Double precision multiplication

This routine assumes that:

- X_L (LSW) and X_H (MSW) are stored in R0 and R1, respectively.
- Y_L (LSW) and Y_H (MSW) are stored in R2 and R3, respectively.
- MP and MS are cleared.
- t performs P=X*Y.

After computation, the 64-bit product P is stored in R4-R7, where R7 contains the most significant word and R4 the least significant word.

```
;
; XL*YL multiplication (unsigned)
;
CoMULu      R0,         R2          ; (ACC) ← XL*YL.
CoSTORE     R4,         MAL         ; (R4) ← (ACC)L.
;
; XL*YH multiplication (unsigned/signed) and XH*YL multiplication (signed/unsigned).
;
CoSHR       #8                      ; (ACC) ← (ACC) >> 8.
CoSHR       #8                      ; (ACC) ← (ACC) >> 8.
CoMACus     R0,         R3          ; (ACC) ← (ACC) + XL*YH.
CoMACsu     R1,         R2          ; (ACC) ← (ACC) + XH*YL.
CoSTORE     R5,         MAL         ; (R5) ← (ACC)L.
;
; XHL*YH multiplication (signed/signed)
;
CoASHR      #8                      ; (ACC) ← (ACC) >>a 8.
CoASHR      #8                      ; (ACC) ← (ACC) >>a 8.
CoMAC       R1,         R3          ; (ACC) ← (ACC) + XH*YH.
CoSTORE     R5,         MAL         ; (R6) ← (ACC)L.
CoSTORE     R5,         MAH         ; (R7) ← (ACC)H.
```

	Instruction Cycles	Program Words
Total	12	24

2.2 Nth order power series

The formula is:

$$\left[y = \sum_{i=0}^{n} a(i)\, x^{i} = [[[[a(n)\, x + a(n-1)]\, x + a(n-2)]\, x + a(n-3)] + \ldots] \right]$$

The associated pseudo code is:

```
;                   x = input.
;                   y = output.
;                   a(i) for i=0,1,...,n, are the coefficients.
y = a(n);
for (i=1 to n) {
                    y = y*x+a(n-i);
}
```

Assuming that:

- x is a fractional and is located in R1;
- a(i) are fractional;
- y can be represented by a 16-bit data.

The final result is contained in the ACCumulator (ACC). To minimize the loop overhead, the program uses “loop unrolling” and assumes that n is even.

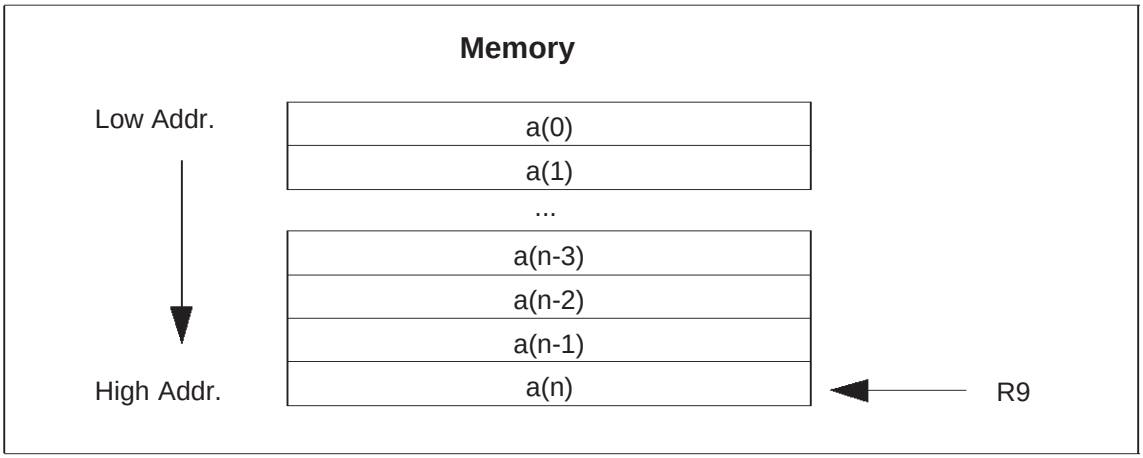


Figure 1 Memory Map

```
                ;
                ; Initialization.
                ;
                MOV         MCW,        #mcw            ; (MCW) ← mcw, MS and MP are cleared.
                MOV         MRW,        #mrw            ; (MRW) ← mrw.
                MOV         R0,         #0              ; (R0) ← 0
                MOV         R9,         a(n)_address    ; (R9) ← address of a(n).
                ;
                ; Initialize the Loop Count
                ;
                MOV         R3          #n/2            ; (R3) ← n/2.
                ;
                ; Loop Prolog
                ;
                CoMUL       R1,         [R9-]           ; (ACC) ← a(n)*x;
                                                        ; (R9)  ← (R9)-2
                ;
                ; Unrolled Loop
                ;
SERIE_LOOP      CoADD       R0,         [R9-]           ; (ACC) ← (ACC) + a(i-1);
                                                        ; (R9)  ← (R9)-2.
                CoSTORE     R2,         MAS             ; (R2) ← limited (ACC)
                CoMUL       R1,         R2              ; (ACC) ← (R2)*x
                                                        ; (R9)  ← (R9)-2
```

```
CoADD     R0,     [R9-]      ; (ACC) ← (ACC)+ a(i-2);
                             ; (R9)  ← (R9)-2.
CoSTORE   R2,     MAS        ; (R2) ← limited (ACC)
CoMUL     R1,     R2         ; (ACC) ← (R2)*x
                             ; (R9)  ← (R9)-2
;
; End_of_loop Checking.
;
CMPD1     R3      #0h        ; (R3)  ← (R3)-1.
JMPR      cc_Z    SERIE_LOOP ; End-of-Loop test & branch.
;
; Loop Epilog
;
CoADD     R0,     [R9]       ; (ACC) ← (ACC) + a(0);
```

	Instruction Cycles	**Program Words**
Total	4N+8	28

2.3 [NxN][Nx1] matrix multiply

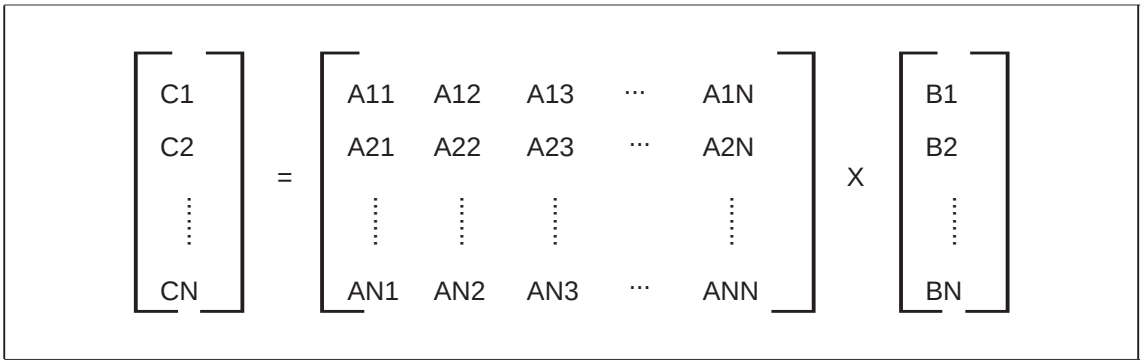


Figure 2 [NxN][Nx1] matrix multiply

The [NxN][Nx1] matrix multiply memory map is shown below:

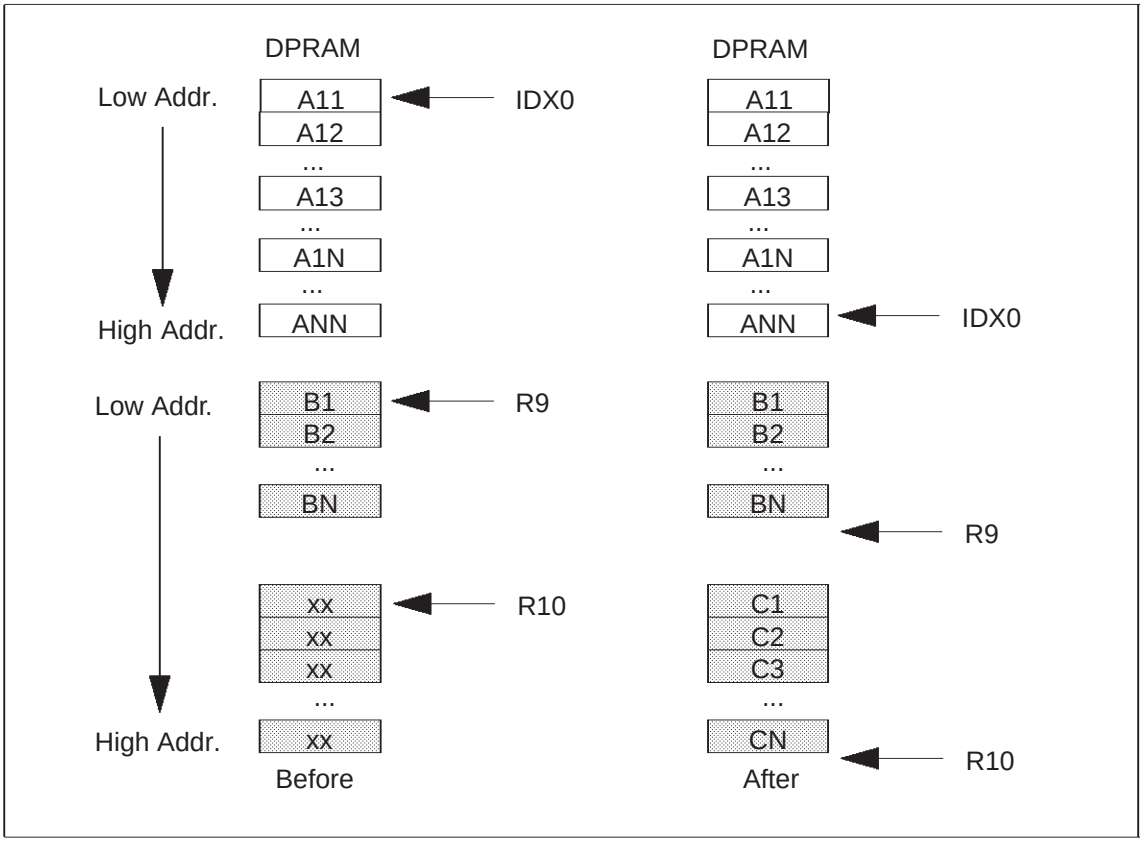


Figure 3 Memory map

N is assumed to be less than 31.

```
                ;
                ; MAC dedicated registers' initialization:
                ;
                EXTR    #2                      ; 2 next instructions use ESFR
                ;                               ; space.
                MOV     IDX0,   @A11            ; (IDX0) ← A11_addr.
                MOV     QR0,    #N-1            ; (QR0) ← N-1.
                ;
                ; GPRs initialization:
                ;       - R7 is used as loop counter.
                ;       - R9 contains B1Address.
                ;       - R10 contains C1 Address.
                ;
                MOV     R7,     #N              ; (R7) ← N
                MOV     R9,     @B1             ; (R9) B1_addr
                MOV     R10,    @C1             ; (R10)) ← C1_addr

                MATRIX_LOOP:
                ;
                ; Dot Product prolog
                ;
                CoMUL   [IDX0+],    [R9+]       ; (ACC) ← Ai1.B1
                                                ; (IDX0) ← (IDX0)+2
                                                ; (R9) ← (R9)+2.
                ;
                ; DOT Product loop.
                ;
REPEAT N-2 TIMES CoMAC  [IDX0+],    [R9+]       ; (ACC) ← (ACC) + Aij*Bj
                                                ; (IDX0) ← (IDX0)+2
                                                ; (R9) ← (R9)+2.
                ;
                ; Dot Product epilog (provide Ci in an appropriate format).
                ;
                CoMAC   [IDX0+],    [R9-QR0]    ; (ACC) ← (ACC) + AiN*Bn
                                                ; (IDX0) ← (IDX0)+2
                                                ; (R9) ← (R9)-(N-1).
                ;
                ; Shift & Rounding
```

```
;
CoASHR      #data3,     rnd             ; (ACC)=(ACC)>>#data3+rnd
;
; Write Ci into memory.
;
CoSTORE     [R10+]      MAS             ; ((R10)) ← Ci.
                                        ; (R10) ← (R10)+2.
;
; End_of_loop Checking.
;
CMPD1       R7          #0h             ; (R7) ← (R7) -1.
JMPR        cc_Z        MATRIX_LOOP     ; End-of-Loop test & branch
```

	Instruction Cycles	**Program Words**
Total	N^2+4N+7	24

2.4 N-real multiply (windowing)

The formula is: $y(i) = x(i)\ w(i)$ for i =0,1,...,N-1

The memory mapping is shown in Figure 4.To minimize the loop overhead, this program uses “loop unrolling”.The associated pseudo code is:

```
;           x(n) = input signal at time n.
;           w(n) = window coefficient at time n.
;
for (i=0 to N-1) {
            y(i)= x(i)*w(i);
}
```

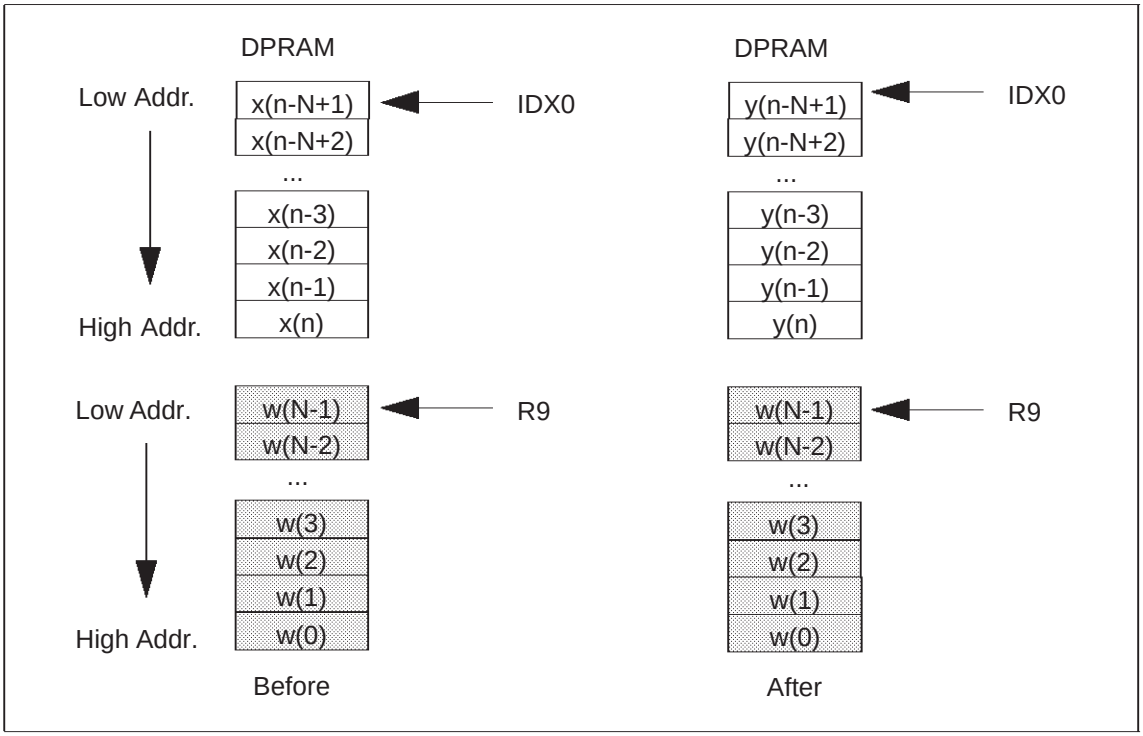


Figure 4 Memory map

This routines assumes that the following general purpose and co-processor registers (SFRs) have been initialized once for ever and L is a multiple of 4:

- R9 contains the w(N-1) address.
- IDX0 contains the x(n-N+1) address
- QX0 and QR0 with N-1

```
                ;
                ; Initialize the Loop Count
                ;
                MOV       R3        #N/4          ; (R3) ← N/4.
                ;
                ; Unrolled Loop
                ;
WINDOW_LOOP     CoMUL     [IDX0],   [R9-]   rnd   ; (ACC) ← w(i)*x(i) + rnd;
                                                  ; (R9)  ← (R9)+2
                CoSTORE   [IDX0+],  MAH           ; (R2) ← (ACC)
                                                  ; (IDX0) ← (IDX0)+2
                CoMUL     [IDX0],   [R9-]   rnd   ; (ACC) ← w(i+1)*x(i+1)+rnd;
                                                  ; (R9)  ← (R9)+2
```

```
CoSTORE   [IDX0+],   MAH          ; (R2) ← (ACC)
                                  ; (IDX0) ← (IDX0)+2
CoMUL     [IDX0],    [R9-]   rnd  ; (ACC) ← w(i+2)*x(i+2)+rnd;
                                  ; (R9) ← (R9)+2
CoSTORE   [IDX0+],   MAH          ; (R2) ← (ACC)
                                  ; (IDX0) ← (IDX0)+2
CoMUL     [IDX0],    [R9-]   rnd  ; (ACC) ← w(i+3)*x(i+3)+rnd;
                                  ; (R9) ← (R9)+2
CoSTORE   [IDX0+],   MAH          ; (R2) ← (ACC)
                                  ; (IDX0) ← (IDX0)+2
;
; End_of_loop Checking.
;
CMPD1     R3         #0h          ; (R3) ← (R3)-1.
JMPR      cc_Z       WINDOW_LOOP  ; End-of-Loop test & branch.
```

	Instruction Cycles	**Program Words**
Total	2N + 2N/4+2	5+ (2*2)*4

*Note The number of Instruction Cycles and Program Words required for this application depends on the "unrolling factor". "2N" corresponds to the number of cycles per coefficient, "2N/4" corresponds to the branch penalty when the "unrolling factor" is 4. Similarly, "(2*2)*4-4" corresponds to the increase in program words when the "unrolling factor" is 4.*
*Typically, if URF defines the factor, the execution time and number of program words becomes: 2N+2N/URF + 2 instruction cycles, and 5+ 4*URF program words.*

3 Fir Filter-Real Correlation-Convolution

3.1 Simple precision FIR filter

The pseudo code is:

```
;       x(n) = input signal at time n.
;       y(n) = output signal at time n.
;       h(k) = k'th coefficient.
;       L = Number of coefficient taps in the filter.
;
y(n)=0;
for (k=0 to L-1) {
        y(n)= y(n) + h(k)*x(n-k);
}
```

This program illustrates the use of multiply/multiply-accumulate instructions, “CoMIN & CoMAX” (performing a programmable saturation), and a shift instruction. The corresponding

memory map is shown below. It is assumed that the coefficients and samples have been initialized by another routine.

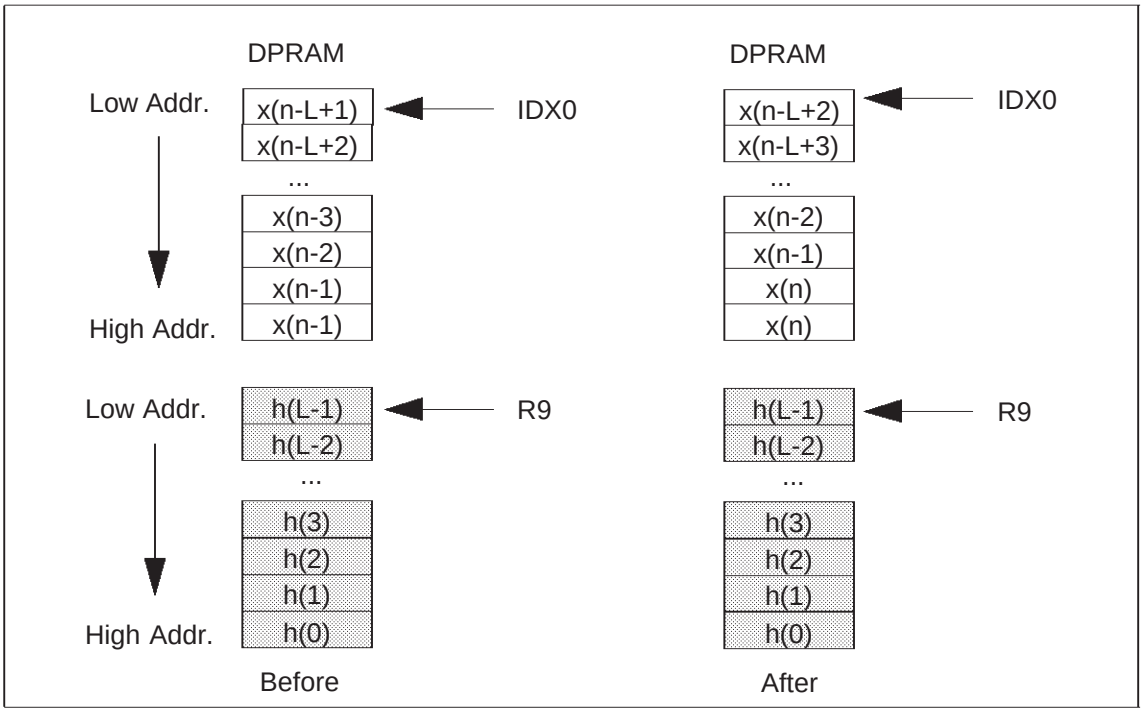


Figure 5 Memory Map

This routine assumes that the following general purpose registers and co-processor registers (SFRs) have been initialized:

- R0 with 0000_H
- R1 with the 16-bit MAXimum tolerated value
- R2 contains the 16-bit MINimum tolerated value
- R9 contains the h(L-1) address
- IDX0 contains the x(n-L+1) address
- QX0 and QR0 with L-1

```
;
; Repeat Count Initialization (repeat count > 31)
;
MOV             MRW,            #L-4            ; (MRW) ← L-4.
;
; Read the new filter input from a (E)SFR and move it into the DPRAM at x(n-1)
```

```
                ; address overwriting thus x(n-1)
                ;
                MOV         @x(n),      ADC_sfr     ; move the new input x(n)
                ;
                ; FIR prolog: first multiplication
                ;
                CoMUL       [IDX0+],    [R9+]       ; (ACC) ← h(L-1)*x(n-L+1)
                                                    ; (IDX0) ← (IDX0)+2,
                                                    ; (R9) ← (R9)+2.
                ;
                ; FIR loop: Repeat L-2 times the same MAC instruction.
; REPEAT MRW TIMES CoMACM     [IDX0+],    [R9+]       ; (ACC) ← (ACC) + h(i)*x(n-i)
                                                    ; & x(n-i-1) ← x(n-i),
                                                    ; (IDX0) ←(IDX0)+2,
                                                    ; (R9) ← (R9)+2.
                ;
                ; FIR epilog: last MAC instruction and provide y(n) in an appropriate format
                ;
                CoMACM      [IDX0-QX0], [R9-QR0]    ; (ACC) ←(ACC)+h(0)*x(n)
                                                    ; & x(n-l+1) ← x(n-L+2),
                                                    ; (IDX0) ← (IDX0)-2*(L-1),
                                                    ; (R9) ← (R9)-2*(L-1).
                ;
                ; Shift & Rounding
                ;
                CoASHR      #data3,     rnd         ; (ACC) ← (ACC)>>a #data3
                                                    ;+rnd
                ;
                ; Limiting
                ;
                CoMIN       R0,         R1          ; (ACC) ←Min((ACC), MAX).
                CoMAX       R0,         R2          ; (ACC) ← Max((ACC),MIN).
                ;
                ;Write the new filter output y(n) into a (E)SFR.
                ;
                NOP                                 ; Pipeline Effect.
                MOV         DAC_sfr,    MAH         ; move the new output y(n).
```

	Instruction Cycles	Program Words
Read Input sample	1	2
Initialization	1	2
FIR Loop	L	6
Post -Processing	4	7
Write Output sample	1	2
Total	L+7	19

3.2 Extended-precision FIR filter

This routine describes a FIR filter using a 32-bit coefficient and a 16-bit input sample.

- The extended-precision FIR filter uses the same naming convention as the single-precision FIR filter.
- $h_L(k)$ and $h_H(k)$ stand for the least significant (LS) and most significant (MS) word of the k'th coefficient, respectively.
- For simplicity, MP from MSC should be cleared.

Figure 6 shows the memory map. It is assumed that both coefficients and samples have been initialized by another routine. This program illustrates the use of signed/unsigned multiplications.

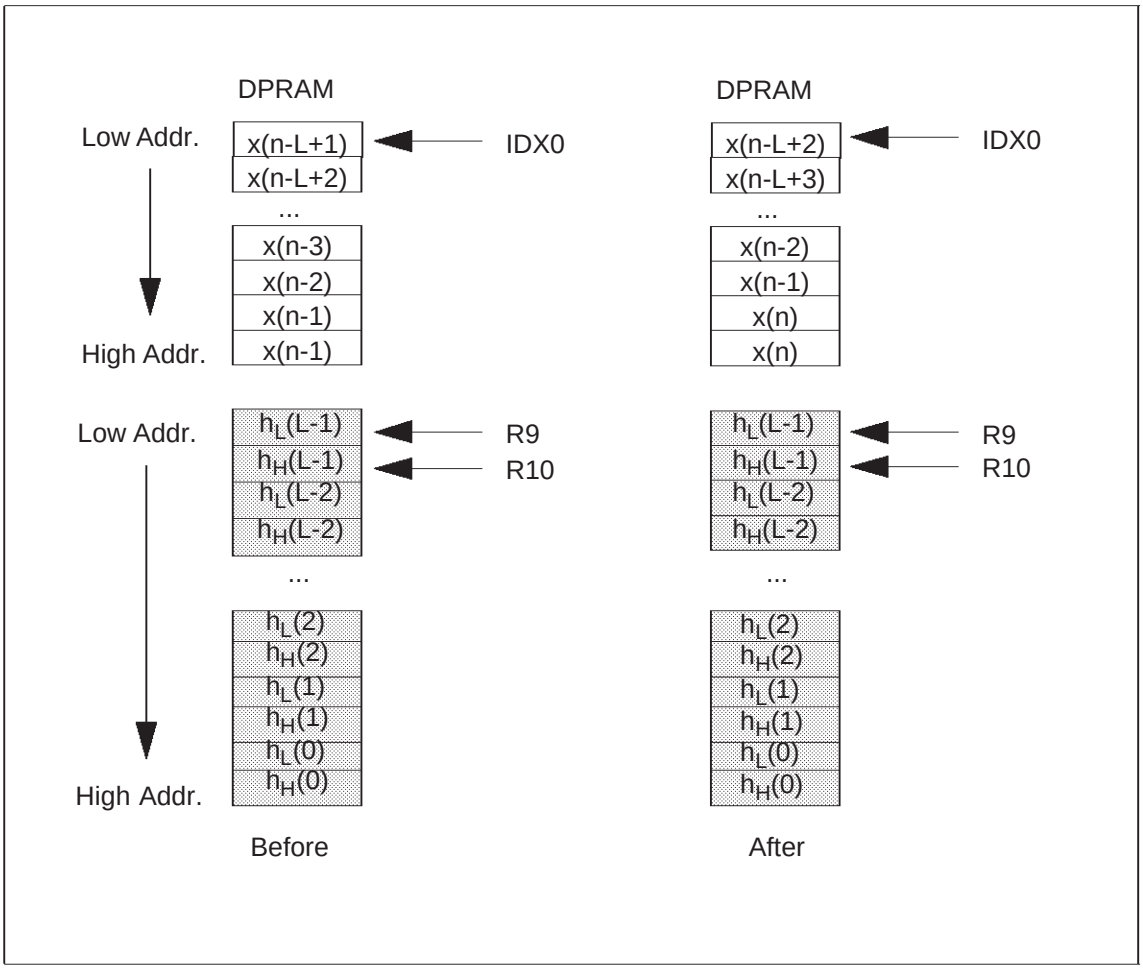


Figure 6 Memory map

This routines assumes that the following general purpose registers and co-processor registers (SFRs) have been initialized:

- R0 with 0000_H
- R1 with the 16-bit MAXimum tolerated value
- R2 contains the 16-bit MINimum tolerated value
- R9 contains the $h_L(L-1)$ address
- R10 contains the $h_H(L-1)$ address

- IDX0 contains the x(n-L+1) address
- QX0 with L-1
- QR0 with 2
- QR1 with 2*L-1

```
                ;
                ; Repeat Count Initialization (repeat count > 31)
                ;
                MOV         MRW,        #L-4        ; (MRW) ← L-4.
                ;
                ; Read the new filter input from a (E)SFR and move it into the DPRAM
                ; at x(n-1) address therefore overwriting x(n-1).
                ;
                MOV         @x(n),      ADC_sfr     ; move the new input x(n)
                ;
                ; FIR prolog (LSWs of Impulse response): first multiplication
                ;
                CoMULus     [IDX0+],    [R9+QR0]    ; (ACC) ← hL(L-1)*x(n-L+1)
                                                    ; (IDX0) ← (IDX0)+2,
                                                    ; (R9) ← (R9)+4.
                ;
                ; FIR loop (LSWs of Impulse response) Repeat the same MAC instruction L-2 times
                ;
REPEAT MRW TIMES CoMACsu    [IDX0+],    [R9+QR0]    ; (ACC) ← (ACC) + hL(i)*x(n-i)
                                                    ; (IDX0) ← (IDX0)+2,
                                                    ; (R9) ← (R9)+4.
                ;
                ; FIR epilog (LSWs of Impulse response): last MAC instruction and provide y(n)
                ; in an appropriate format
                ;
                CoMACsu     [IDX0-QX1], [R9-QR1]    ; (ACC) ← (ACC) + hL(0)*x(n)
                                                    ; & x(n-l+1) ← x(n-L+2),
                                                    ; (IDX0) ← (IDX0)-2*(L-1),
                                                    ; (R9) ← (R9)-2*(2L-1).
                ;
                ; Rounding & Shift
                ;
                MOV         MRW,        #L-4        ; (MRW)=L-4.
                CoRND                               ; (ACC)=(ACC)+rnd
```

```
                CoASHR      8,                          ; (ACC)=(ACC)>>8
                CoASHR      8,                          ; (ACC)=(ACC)>>8
                            ;
; FIR prolog (MSWs of Impulse response): first multiplication
                ;
                CoMAC       '[IDX0+],       [R10+QR0]   ; (ACC) ← hH(L-1)*x(n-L+1)
                                                        ; (IDX0) ← (IDX0)+2,
                                                        ; (R10) ← (R10)+4.
                ;
; FIR loop (MSWs of Impulse response)Repeat the same MAC instruction L-2 times
                ;
REPEAT MRW TIMES CoMACM     [IDX0+],        [R10+QR0]   ; (ACC) ← (ACC) + hH(i)*x(n-i)
                                                        ; & x(n-i-1) ← x(n-i),
                                                        ; (IDX0) ← (IDX0)+2,
                                                        ; (R10) ← (R10)+4.
                ;
; FIR epilog (MSWs of Impulse response): last MAC instruction and provide
                ; y(n) in an appropriate format
                ;
                CoMACM      [IDX0-QX1],     [R10-QR1]   ; (ACC) ← (ACC) + hH(0)*x(n)
                                                        ; & x(n-l+1) ← x(n-L+2),
                                                        ; (IDX0) ← (IDX0)-2*(L-1),
                                                        ; (R10) ← (R10)-2*(2L-1).
                ;
; Shift & Rounding
                ;
                CoASHR      #data3,         rnd         ; (ACC) ← (ACC)>>a #data3 +rnd
                ;
; Limiting
                ;
                CoMIN       R0,             R1          ; (ACC) ← Min((ACC), MAX).
                CoMAX       R0,             R2          ; (ACC) ← Max((ACC),MIN).
                ;
;Write the new filter output y(n) into a (E)SFR.
                ;
                NOP                                     ; Pipeline Effect.
                MOV         DAC_sfr,        MAH         ; move the new output y(n).
```

	Instruction Cycles	Program Words
Read Input sample	1	2
Initialization	2	4
FIR Loop	2L+3	18
Post -Processing	4	7
Write Output sample	1	2
Total	2L+11	33

4 IIR Filters

4.1 Nth Order IIR filter: direct form 1

The rules for the implementation of FIR filters can be extended to IIR filters. The Nth-order difference equation is:

$$y(n) = \sum_{k=1}^{N} a(k)\, y(n-k) + \sum_{k=0}^{M} b(k)\, x(n-k)$$

This can be called “Direct Form 1”. The associated pseudo code is:

```
;       x(n) = input signal at time n
;       y(n) = output signal at time n
;       a(k), b(k)= IIR coefficients
;       N, M refer to the above equation
y(n)=0;
for (k=0 to M) {
                                        y(n)= y(n) +b(k)*x(n-k)
}
for (k=1 to N) {
                                        y(n)= y(n) +a(k)*y(n-k);
}
```

Figure 7 shows the memory map. It has been assumed that the coefficients and samples have been initialized by another routine.

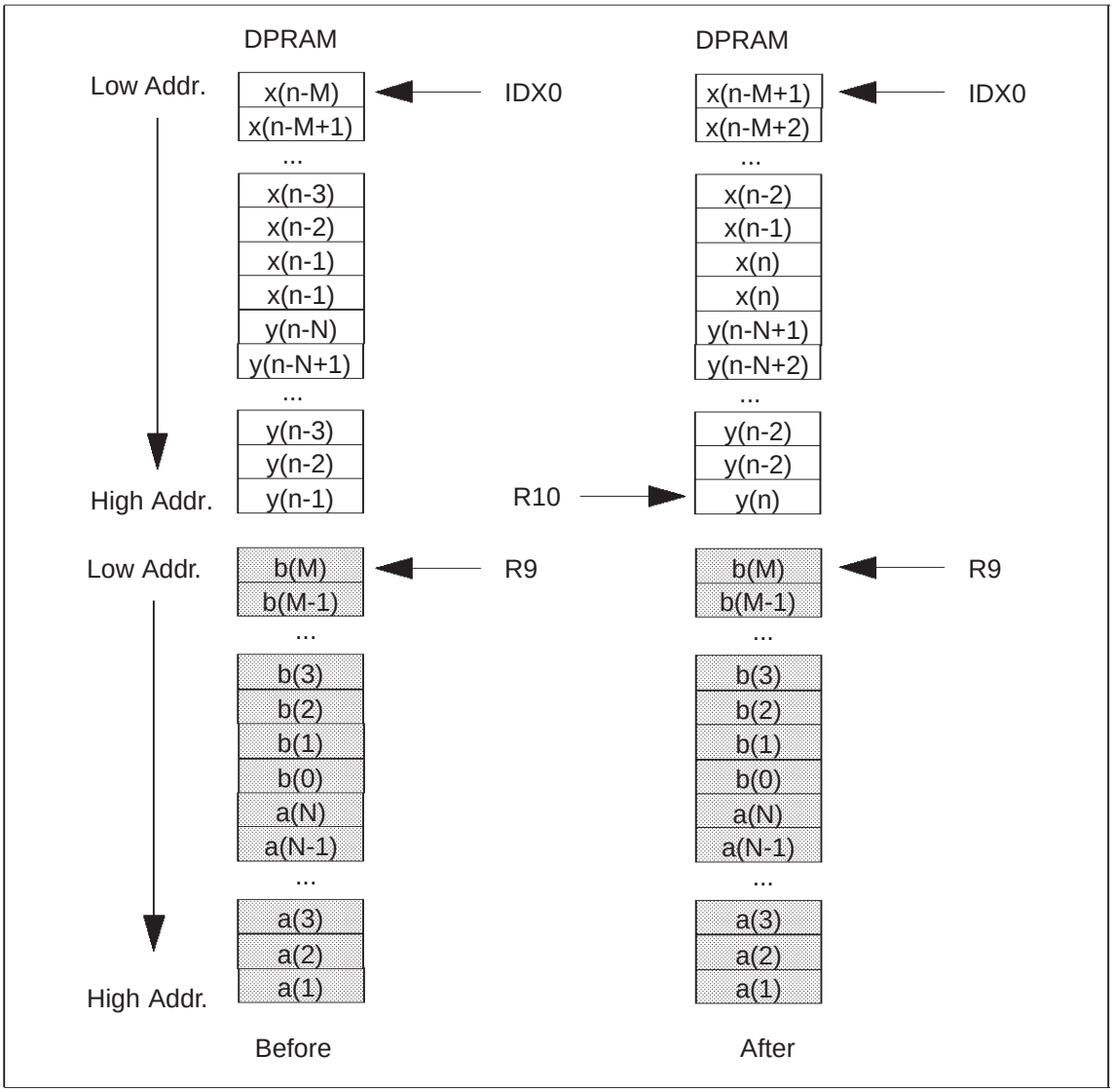


Figure 7 Memory map

This routines assumes that the following general purpose and co-processor registers (SFRs) have been initialized:

- R0 with 0000_H
- R1 with the 16-bit MAXimum tolerated value
- R2 contains the 16-bit MINimum tolerated value

- R9 contains the b(M) address
- R10 contains the y(n) address
- IDX0 contains the x(n-M) address
- QX0 with N+M
- QR0 with N+M

```
                ;
                ; Repeat Count Initialization (repeat count > 31) for the first IIR Loop
                ;
                MOV             MRW,        #M-2        ; (MRW) ← M-2.
                ;
                ; Read the new filter input from a (E)SFR & move it into the DPRAM
                ; at x(n-1) address, overwriting x(n-1).
                ;
                MOV             @x(n),      ADC_sfr     ; move the new input x(n)
                ;
                ; Prolog of the First IIR loop.
                ;
                CoMUL           '[IDX0+],   [R9+]       ; (ACC) ← b(M)*x(n-M)
                                                        ; (IDX0) ←(IDX0)+2,
                                                        ; (R9) ← (R9)+2.
                ;
                ; First IIR loop.
                ;
REPEAT MRW TIMES CoMACM         [IDX0+],    [R9+]       ; (ACC) ← (ACC)+b(i)*x(n-i)
                                                        ; & x(n-i-1) ← x(n-i),
                                                        ; (IDX0) ← (IDX0)+2,
                                                        ; (R9) ← (R9)+2.
                ;
                ; Repeat Count Initialization (repeat count > 31) for the second.
                ;
                MOV             MRW,        #N-43       ; (MRW) ← N-4.
                ;
                ;
                ; prolog of the Second IIR loop.
                ;
                CoMAC           [IDX0+],    [R9+]       ; (ACC) ← a(N)*y(n-N)
                                                        ; (IDX0) ← (IDX0)+2,
                                                        ; (R9) ← (R9)+2.
```

```
                ;
                ; Second IIR loop.
                ;
REPEAT MRW TIMES CoMACM         [IDX0+],        [R9+]       ; (ACC) ← (ACC) + a(i)*y(n-i)
                                                            ; & y(n-i-1) ← y(n-i),
                                                            ; (IDX0) ← (IDX0)+2,
                                                            ; (R9) ← (R9)+2.
                ;
                ; Epilog of the second IIR loop.
                ;
                CoMACM          [IDX0-QX0],     [R9-QR0]    ; (ACC) ← (ACC)+h(0)*x(n)
                                                            ; & y(n-2) ← y(n-1),
                                                            ; (IDX0) ← (IDX0)-2*(N+M),
                                                            ; (R9) ← (R9)-2*(N+M).
                ;
                ; Rounding
                ;
                CoRND                                       ; (ACC) ← (ACC) + rnd
                ;
                ; Limiting
                ;
                CoMIN           R0,             R1          ; (ACC) ← Min((ACC), MAX).
                CoMAX           R0,             R2          ; (ACC) ← Max((ACC),MIN).
                ;
                ; Write the new filter output, y(n), into memory.
                ;
                CoSTORE         [R10]           MAH         ; ((R10)) ← y(n).
                ;
                ;Write the new filter output y(n) into a (E)SFR.
                ;
                NOP                                         ; Pipeline Effect.
                MOV             DAC_sfr,        MAH         ; move the new output y(n).
```

	Instruction Cycles	**Program Words**
Read Input sample	1	2
Initialization	2	4
DF1 IIR Loop	N+M	10
Post -Processing	5	9

	Instruction Cycles	Program Words
Write Output sample	1	2
Total	N+M+9	27

4.2 N^{th} Order IIR filter: direct form 2

The following equations equally represent the Nth Order IIR filter:

$$u(n) = x(n) + \sum_{k=1}^{N} a(k)\, u(n-k)$$

$$u(n) = \sum_{k=0}^{N} b(k)\, u(n-k)$$

These equations use the intermediate state variable vector U={u(n), u(n-1), u(n-2),..., u(n-N)}. This representation is called “Direct Form 2” and is illustrated by Figure 8. Direct Form 2 has an advantage over Direct Form 1 as it requires less data memory.

The associated pseudo code is:

```
;                   x(n) = input signal at time n.
;                   u(n) = state variable at time n.
;                   y(n) = output signal at time n.
;                   a(k), b(k)= IIR coeffi cients.
;                   It is assumed N = M.
;
u(n)=x(n);
for (k=1 to N) {
                    u(n)= u(n) +a(k)*u(n-k);
}
y(n)=b(0)*u(n);
for (k=1 to N) {
                    y(n)= y(n) +b(k)*u(n-k);
}
```

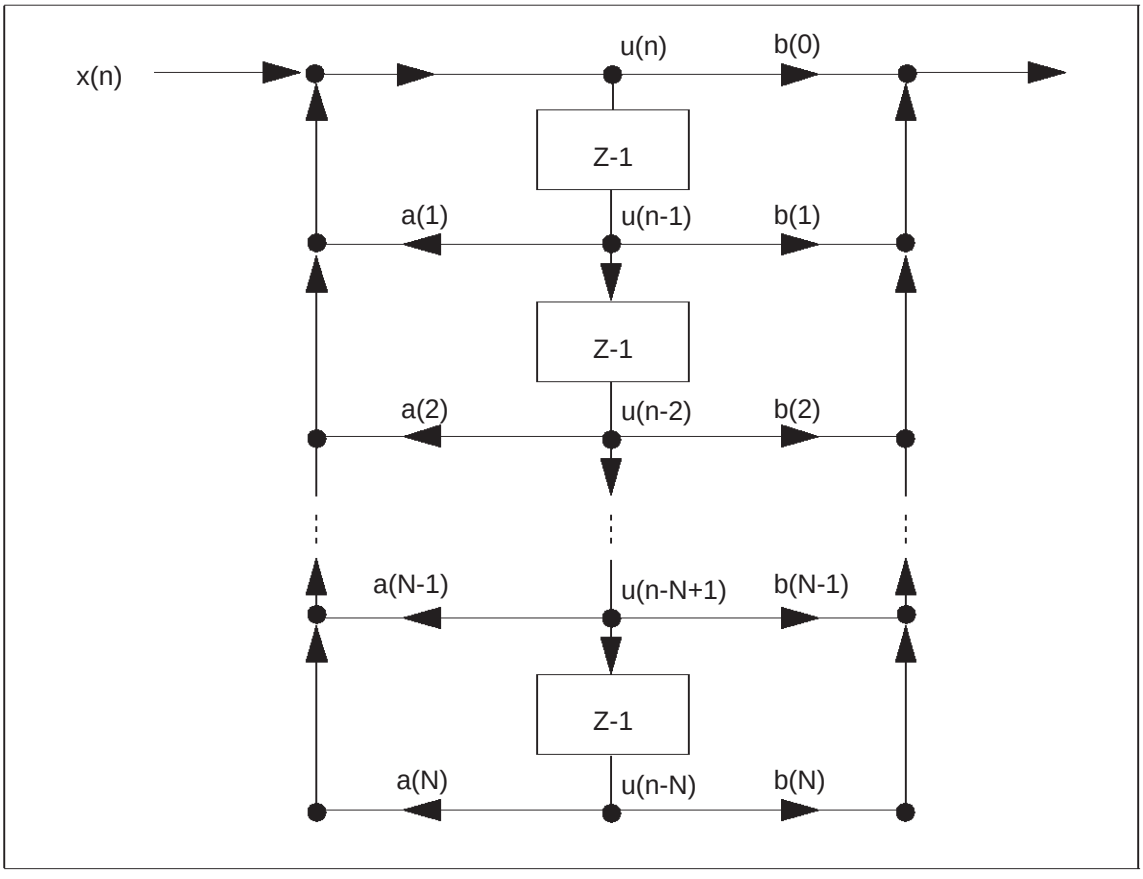


Figure 8 IIR Direct Form 2

Figure 9 shows the corresponding memory map. It has been assumed that the coefficients and samples have been initialized by another routine.

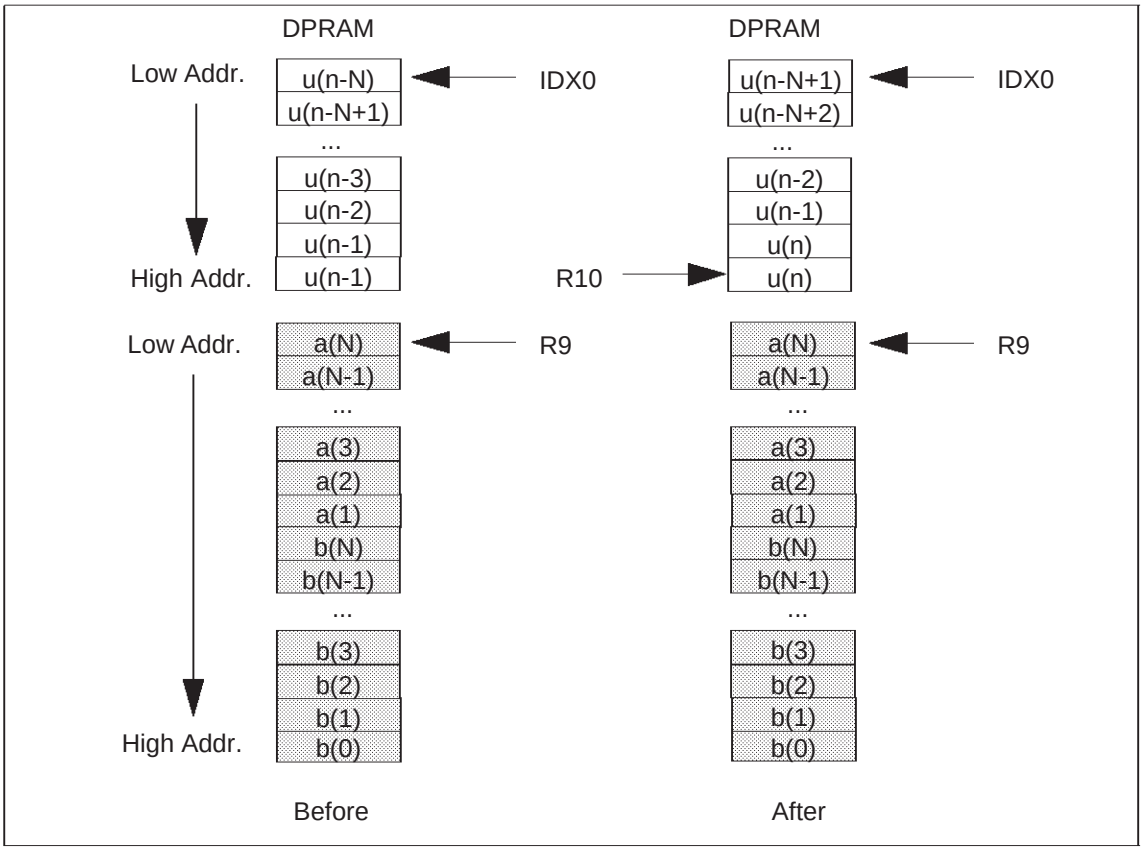


Figure 9 Memory map

This routines assumes that the following general purpose and co-processor registers (SFRs) have been initialized:

- R0 with 0000_H
- R1 with the 16-bit MAXimum tolerated value
- R2 contains the 16-bit MINimum tolerated value
- R9 contains the a(N) address
- R10 contains the y(n) address
- IDX0 contains the u(n-N) address
- QX0 with N-1 and QX1 with N
- QR0 with 2N

```
                ;
                ; Repeat Count Initialization (repeat count > 31) for the first IIR Loop
                ;
                MOV         MRW,        #N-3        ; (MRW) ← N-3.
                ;
                ; Read the new filter input from a (E)SFR and move it into the Accumulator.
                ;
                MOV         MAH,        ADC_sfr     ; (MAH) ← x(n),
                                                    ; (MAE) ← 8 times (MAH15),
                                                    ; (MAL) ← 0000H.
                ;
                ; First IIR loop.
                ;
REPEAT MRW TIMES CoMAC      [IDX0+],    [R9+]       ; (ACC) ← (ACC) + a(i)*u(n-i)
                                                    ; (IDX0) ← (IDX0) + 2,
                                                    ; (R9) ← (R9)+2.
                ;
                ; Epilog of the first IIR loop.
                ;
                CoMAC       [IDX0-QX0], [R9+], rnd  ; (ACC) ← (ACC)+a(1)*u(n-1)
                                                    ; +rnd
                                                    ; (IDX0) ← (IDX0)-2*(N-1),
                                                    ; (R9) ← (R9)+2.
                ;
                ; Repeat Count Initialization (repeat count > 31) for the second.
                ;
                MOV         MRW,        #N-4        ; (MRW) ← N-4.
                ;
                ; Move u(n) into memory.
                ;
                CoSTORE     [R10],      MAS         ; ((R10)) ← u(n)
                ;
                ; Prolog of the Second IIR loop.
                ;
                CoMAC       ‘           [IDX0+],[R9+] ; (ACC) ← b(N)*u(n-N)
                                                    ; (IDX0) ← (IDX0)+2,
                                                    ; (R9) ← (R9)+2.
                ;
```

```
                    ; Second IIR loop.
                    ;
REPEAT MRW TIMES    CoMACM      [IDX0+],      [R9+]          ; (ACC)← (ACC)+b(i)*u(n-i)
                                                             ; & u(n-i-1)← u(n-i),
                                                             ; (IDX0)← (IDX0)+2,
                                                             ; (R9)← (R9)+2.
                    ;
                    ; Epilog of the Second IIR loop.
                    ;
                    CoMACM      [IDX0-QX1],   [R9-QR0], rnd  ; (ACC)← b(0)*u(n)+md
                                                             ; & u(n-1)← u(n),
                                                             ; (IDX0)← (IDX0)-2N,
                                                             ; (R9)← (R9)-2N.
                    ;
                    ; Limiting
                    ;
                    CoMIN       R0,           R1             ; (ACC)← Min((ACC), MAX).
                    CoMAX       R0,           R2             ; (ACC)← Max((ACC),MIN).
                    ;
                    ;Write the new filter output y(n) into a (E)SFR.
                    ;
                    NOP                                      ; Pipeline Effect.
                    MOV         DAC_sfr,      MAH            ; move the new output y(n).
```

	Instruction Cycles	**Program Words**
Read Input sample	1	2
Initialization	2	4
DF2 IIR Loop	2N+2	14
Post -Processing	2	3
Write Output sample	1	2
Total	2N+8	25

4.3 N-cascaded real biquads (direct form 2)

A high-order filter can be implemented, either as a single section, or as a combination of first and second order sections. The single section form is quicker and easier to implement, but generates a larger numerical error. This increased error occurs for two reasons:

- The long filter computation process accumulates errors from multiplication with quantized coefficients.
- The roots of high-order polynomials are increasingly sensitive to changes in their quantized coefficients.

Therefore, the single section form is not recommended except for a very low order controller. (see “Nth Order IIR filter: direct form 1” on page 21)

To implement a high-order transfer function, first decompose it into first order and second order blocks (biquads), and then connect these blocks in a cascade. The following paragraphs illustrate this technique for an even numbers of cascaded biquads.

Unlike conventional digital signal processors, the MAC co-processor is able to repeat a single instruction at high speed but does not offer flexible and fast hardware looping. Consequently, to perform a loop containing more than one instruction the programmer must use the regular instruction set incurring a several cycle penalty for the end-of-loop detection. “Loop Unrolling” minimizes this penalty but increases the number of instructions. In the following section the loop unrolling technique will not be employed.

Equations of a Direct Form 2 N^{th} Order IIR filter applied to a second order filter (N=2) yield:

$$u^i(n) = x^i(n) - a^i(1)\ u^i(n-1) - a^i(2)\ u^i(n-2)$$

$$(y^i(n) = b^i(0)\ u^i(n) + b^i(1)\ u^i(n-1) + b^i(2)\ u^i(n-2))x$$

Where “i” specifies the biquad number. Note that $y^i(n)=x^{i+1}(n)$.

For simplicity, it has been assumed that no overflow occurs on $u^i(n)$ and $y^i(n)$.

The naming convention is:

; xi(n) = input signal at time n of biquad number i.

; ui(n) = state variable at time n of biquad number i.

; y(n) = output signal at time n of biquad number i.

; ai(k), bi(k)= Coefficients of biquad number i.

Figure 10 shows the corresponding memory map and assumes that both coefficients and samples have been initialized by another routine.

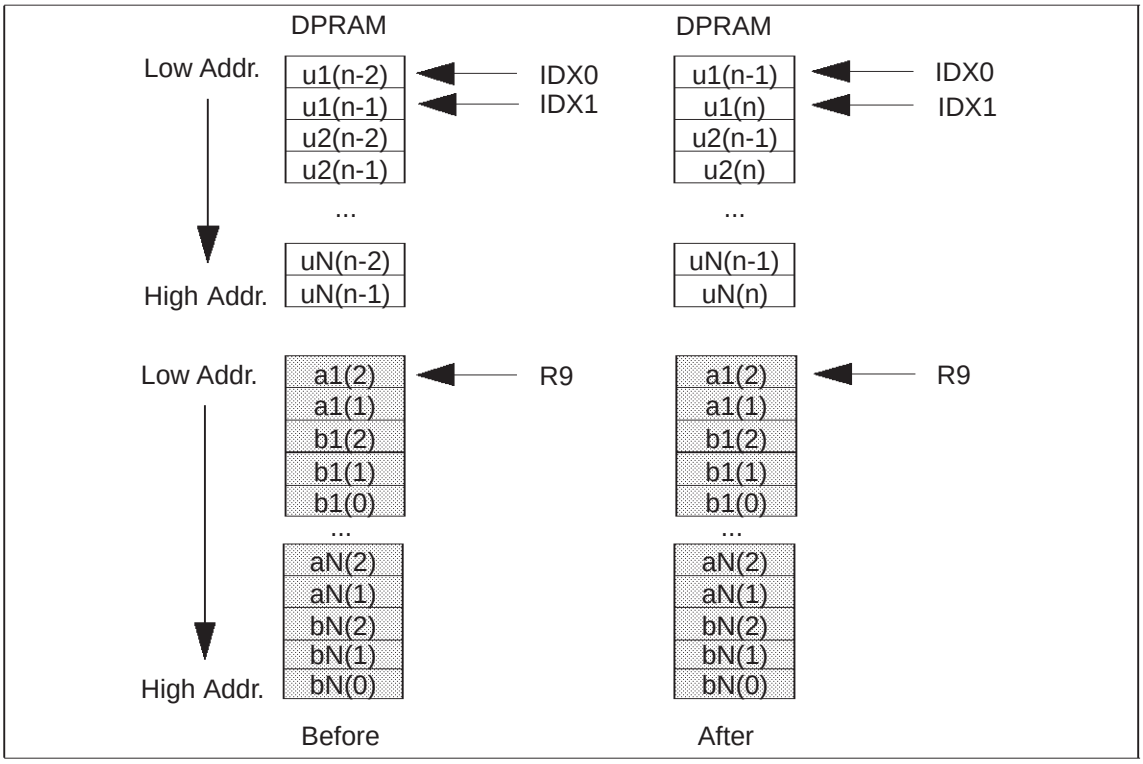


Figure 10 Memory map

This routines assumes that the following general purpose and co-processor registers have been initialized:

- R0 with 0000_H
- R1 with the 16-bit MAXimum tolerated value
- R2 contains the 16-bit MINimum tolerated value
- R9 contains the a1(2) address
- R10 contains the R5 physical address
- IDX0 contains the u1(n-2) address
- IDX1 contains the u1(n-1) address
- QX0 with 2
- QX1 with 2N-1
- QR0 with 5N-1.

```
;
; Initialize the Loop Count: N
;
MOV         R3          #N-1                ; (R3) ← N-1.
;
; Read the new filter input from a (E)SFR and move it into the Accumulator.
;
MOV         MAH,        ADC_sfr             ; (MAH) ← x(n),
                                            ; (MAE) ← 8 times (MAH15),
                                            ; (MAL) ← 0000H.
                                            DF2_BIQUAD_LOOP
;
; First Biquad iteration
;
CoMAC-      [IDX0+],    [R9+]               ; (ACC) ← (ACC)-a^i(2)*u^i(n-2)
                                            ; (IDX0) ← (IDX0)+2,
                                            ; (R9) ← (R9)+2.
CoMAC-      [IDX0-],    [R9+]               ; (ACC) ← (ACC)-a^i(2)*u^i(n-1)
                                            ; (IDX0) ← (IDX0)-2,
                                            ; (R9) ← (R9)+2.
CoRND                                       ; (ACC) ← (ACC)+rnd
;
; Write u^i(n), into a GPR (R5)
;
CoSTORE     R5,         MAS                 ; (R5) ← u^i(n).
;
; Second Biquad iteration.
;
CoMAC       [IDX0+],    [R9+]               ; (ACC) ← (ACC) + b^i(2)*u^i(n-2)
                                            ; (IDX0) ← (IDX0)+2,
                                            ; (R9) ← (R9)+2.
CoMACM      [IDX0+],    [R9+]               ; (ACC) ← (ACC)+b^i(1)*u^i(n-1)
                                            ; & u^i(n-2) ← u^i(n-1)
                                            ; (IDX0) ← (IDX0)+2,
                                            ; (R9) ← (R9)+2.
CoMAC       R5,         [R9+]               ; (ACC) ← (ACC) + b^i(0)*u^i(n)
                                            ; (R9) ← (R9)+2.
```

```
;
; Write u^j(n) to memory.
;
CoMOV     [IDX1+QX0]  [R10]                 ; u^j(n-1) ← u^j(n).
                                            ; (IDX1) ← (IDX1)+4,

;
; End_of_loop Checking.
;
CMPD1     R3          #0h                   ; (R3) ← (R3)-1.
JMPR      cc_Z        DF2_BIQUAD_LOOP       ; End-of-Loop test & branch.
;
; First iteration of the last biquad
;
CoMAC-    [IDX0+],    [R9+]                 ; (ACC) ← (ACC)-a^j(2)*u^j(n-2)
                                            ; (IDX0) ← (IDX0)+2,
                                            ; (R9) ← (R9)+2.
CoMAC-    [IDX0-],    [R9+]                 ; (ACC) ← (ACC)-a^j(2)*u^j(n-1)
                                            ; (IDX0) ← (IDX0)-2,
                                            ; (R9) ← (R9)+2.
; Note that CoMAC- and CoRND cannot be combined.
CoRND                                       ; (ACC) ← (ACC)+rnd
;
; Write u^N(n), into a GPR (R5)
;
CoSTORE   R5,         MAS                   ; (R5) ← u^j(n).
;
; Second iteration of the last biquad.
;
CoMAC     [IDX0+],    [R9+]                 ; (ACC) ← (ACC) + b^N(2)*u^N(n-2)
                                            ; (IDX0) ← (IDX0)+2,
                                            ; (R9) ← (R9)+2.
CoMACM    [IDX0-QX1], [R9+]                 ; (ACC) ← (ACC)+b^N(1)*u^N(n-1)
                                            ; & u^N(n-2) ← u^N(n-1)
                                            ; (IDX0) ← (IDX0)-2*(2N-1),
                                            ; (R9) ← (R9)+2.
CoMAC     R5,         [R9-QR0]      rnd     ; (ACC) ← (ACC) + b^N(0)*u^N(n)
                                            ; +rnd
```

```
                                                    ; (R9) ← (R9)-2*(5N-1).
;
; Write u^i(n) to memory.
;
CoMOV       [IDX1-QX1]   [R10]                      ; u^N(n-1) ← u^N(n).
                                                    ; (IDX1) ← (IDX1)-2*(2N-1),
;
; Limiting
;
CoMIN       R0,          R1                         ; (ACC) ← Min((ACC), MAX).
CoMAX       R0,          R2                         ; (ACC) ← Max((ACC),MIN).
;
;Write the new filter output y(n) into a (E)SFR.
;
NOP                                                 ; Pipeline Effect.
MOV         DAC_sfr,     MAH                        ; move the new output y(n).
```

	Instruction Cycles	**Program Words**
Read Input sample	1	2
Initialization	1	2
DF2 Biquad Loop	10N-1	19
Post -Processing	3	5
Write Output sample	1	2
Total	10N+5	31

4.4 N-cascaded real biquads: transpose form

The equations of a Direct Form 2 N^{th} Order IIR filter applied to a second order filter (N=2) can yield:

$$y^i(n) = b^i(0)\ x^i(n) + u^i(n-1)$$

$$(u^i(n) = b^i(1)\ x^i(n-1) - a^i(1)\ y^i(n) + w^i(n-1))x$$

$$w^i(n) = b^i(2)\ x^i(n) - a^i(2)\ y^i(n)$$

Where “i” is the biquad number. Note that $y^i(n)=x^{i+1}(n)$. This form is also called the “Transpose Form”.

For simplicity, it has been assumed that no overflow occurs on $u^i(n)$ or $y^i(n)$. This form is suitable when the input-to-output delay must be minimized. The naming convention is:

- ; xi(n) = input signal at time n of biquad number i.
- ; ui(n), wi(n) = state variables at time n of biquad number i.
- ; yi(n) = output signal at time n of biquad number i.
- ; ai(k), bi(k)= Coefficients of biquad number i.

Figure 11 shows the corresponding memory map. It is assumed that both coefficients and samples have been initialized by another routine.

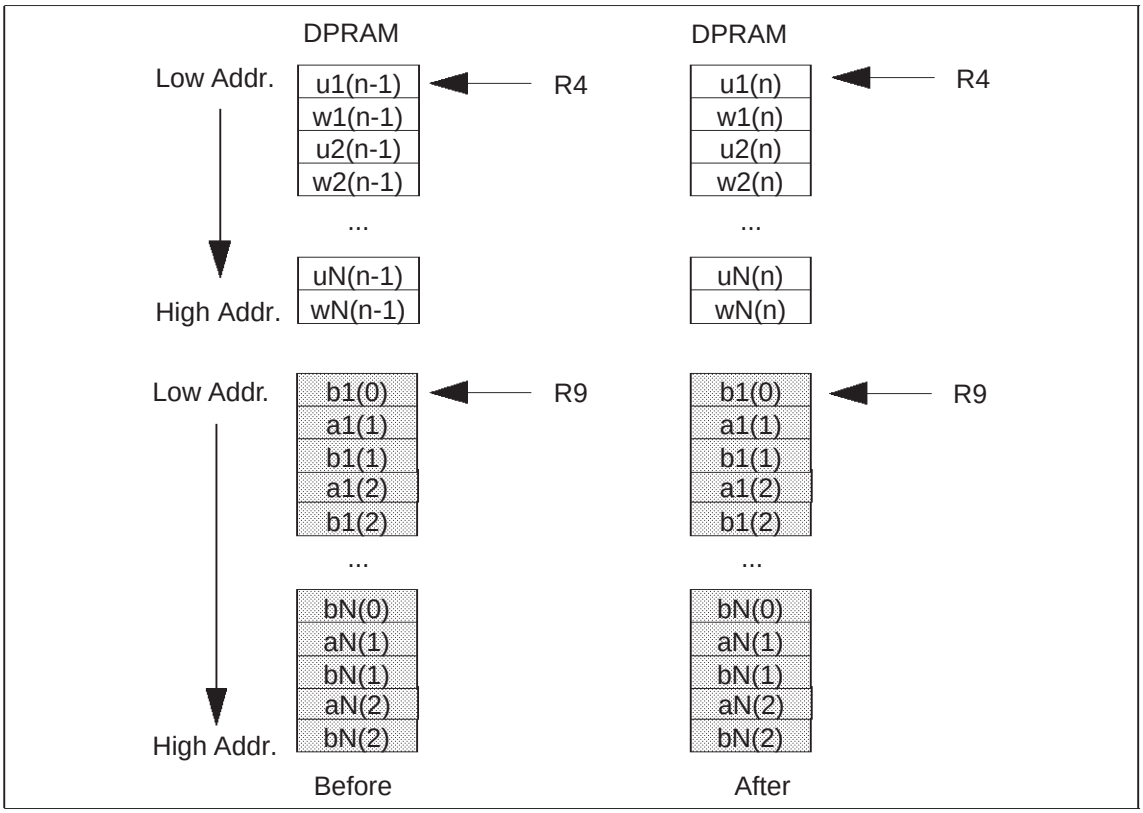


Figure 11 Memory map

This routines assumes that the following general purpose and co-processor registers (SFRs) have been initialized:

- R0 with 0000_H
- R1 with the 16-bit MAXimum tolerated value
- R2 contains the 16-bit MINimum tolerated value

- R4 contains the u1(n-1) address
- R9 contains the b1(0) address
- R10 contains the R5 physical address
- QR0 with 2N-1
- QR1 with 5N-1

```
                ;
                ; Initialize the Loop Count: N
                ;
                MOV     R3      #N-1            ; (R3) ← N-1.
                ;
                ; Read the new filter input from a (E)SFR and move it into a GPR (R5).
                ;
                MOV     R5,     ADC_sfr         ; (R5) ← x(n)
                ;
TF_BIQUAD_LOOP:
                ;
                ; Compute y^j(n)
                ;
                CoLOAD  [R4+],  R0              ; (ACC) ← u^j(n-1)
                                                ; (R4) ← (R4)+2.
                CoMAC   [R9+],  R5      rnd     ; (ACC) ← (ACC)+b^j(0)*x^j(n)
                                                ; +rnd
                                                ; (R9) ← (R9)+2.
                ;
                ; Write y^j(n) into R8.
                ;
                CoSTORE R8,     MAS             ; (R8) ← limited(y^j(n)).
                ;
                ; Compute u^j(n)
                ;
                CoLOAD  [R4],   R0              ; (ACC) ← w^j(n-1)
                CoMAC-  [R9+],  R8              ; (ACC) ← (ACC)-a^j(1)*y^j(n)+
                                                ; (R9) ← (R9)+2.
                CoMAC   [R9+],  R5      rnd     ; (ACC) ← (ACC)+b^j(1)*x^j(n)+rnd
                                                ; (R9) ← (R9)+2.
                ;
```

```
; Write u^i(n) into memory.
;
CoSTORE   [R4+],     MAS              ; u^i(n-1) ← u^i(n).
                                      ; (R4) ← (R4)+2.
;
; Compute w^i(n)
;
CoMUL-    [R9+],     R8               ; (ACC) ← -a^i(2)*y^i(n)
                                      ; (R9) ← (R9)+2.
CoMAC     [R9+],     R5     rnd       ; (ACC) ← (ACC)+b^i(2)*x^i(n)+rnd
                                      ; (R9) ← (R9)+2.
;
; Write w^i(n) into memory.
;
CoSTORE   [R4+],     MAS              ; w^i(n-1) ← w^i(n).
                                      ; (R4) ← (R4)+2.
;
; Write y^i(n) into R5.
;
MOV       R5         R8               ; x^(i+1)(n) ← y^i(n).
;
; End_of_loop Checking.
;
CMPD1     R3         #0h              ; (R3) ← (R3) -1.
JMPR      cc_Z       DF2_BIQUAD_LOOP  ; End-of-Loop test & branch.
;
; Compute y^N(n)
;
CoLOAD    [R4+],     R0               ; (ACC) ← u^N(n-1)
                                      ; (R4) ← (R4)+2.
CoMAC     [R9+],     R5     rnd       ; (ACC) ← (ACC)+b^N(0)*x^N(n)
                                      ; +rnd
                                      ; (R9) ← (R9)+2.
;
; Write y^N(n) into R8.
;
CoSTORE   R8,        MAS              ; (R8) ← limited(y(n)).
```

```
;
; Limiting
;
CoMIN     R0,          R1              ; (ACC) ← Min((ACC), MAX).
CoMAX     R0,          R2              ; (ACC) ← Max((ACC),MIN).
;
;Write the new filter output y(n) into a (E)SFR.
;
CoLOAD    [R4],        R0              ; (ACC) ← w^N(n-1)
MOV       DAC_sfr,     MAH             ; move the new output y(n).
;
; Compute u^i(n)
;
CoMAC-    [R9+],       R8              ; (ACC) ← (ACC)-a^N(1)*y^N(n)+
                                       ; (R9) ← (R9)+2.
CoMAC     [R9+],       R5    rnd       ; (ACC) ← (ACC)+b^N(1)*x^N(n)
                                       ; +rnd
                                       ; (R9) ← (R9)+2.
;
; Write u^i(n) into memory.
;
CoSTORE   [R4+],       MAS             ; u^N(n-1) ← u^N(n).
                                       ; (R4) ← (R4)+2.
;
; Compute w^i(n)
;
CoMUL-    [R9+],       R8              ; (ACC) ← -a^N(2)*y^N(n)
                                       ; (R9) ← (R9)+2.
CoMAC     [R9-QR1],    R5    rnd       ; (ACC) ← (ACC)+b^i(2)*x^i(n)+rnd
                                       ; (R9) ← (R9)-2*(5N-1).
;
; Write w^i(n) into memory.
;
CoSTORE   [R4-QR0],    MAS             ; w^N(n-1) ← w^N(n).
                                       ; (R4) ← (R4)-2*(2N-1).
```

	Instruction Cycles	Program Words
Read Input sample	1	2
Initialization	1	2
TF Biquad Loop	13N-1	44
Post -Processing	2	4
Write Output sample	1	2
Total	13N+4	54

5 LMS Adaptive Filter

5.1 Single-precision LMS adaptive filter

An adaptive filter contains coefficients that are updated by an adaptive algorithm to optimize the filter's response to a desired performance criterion. In general, adaptive filters have two distinct parts: a filter whose structure is designed to perform a processing function, and an adaptive algorithm for adjusting the coefficients of that filter to improve its performance. The incoming signal x(n) is weighted in a digital filter to produce an output y(n). The adaptive algorithm adjusts the filter weights to minimize the error e(n) between the filter output y(n) and the desired response of the filter d(n).

The Single-Precision LMS Adaptive Filter is a FIR filter whose coefficients are updated at each iteration according to an error signal e(n) equal to d(n)-y(n), where d(n) is the desired signal at time n and y(n) is the FIR output. Figure 12 illustrates this filter.

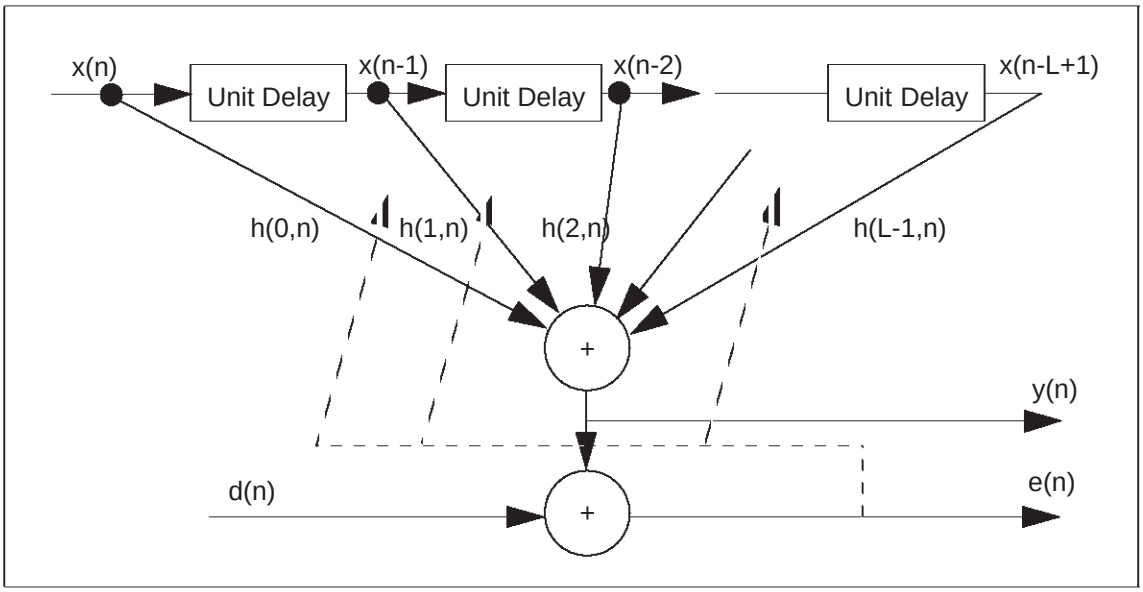


Figure 12 LMS Adaptive Filter

The corresponding pseudo code is:

```
;   x(n) = input signal at time n.
;   d(n) = desired signal at time n.
;   y(n) = output signal at time n.
;   h(k, n) = k'th coefficient at time n.
;   Mu= adaptive gain.
;   L = Number of coefficient taps in the filter.
```

```
;
y(n)=0;
for (k=0 to L-1) {
                        y(n)= y(n) + h(k,n)*x(n-k);
}

e(n)=d(n)-y(n);
for (k=0 to L-1) {
                        h(k,n+1)= h(k,n) - Mu*x(n-k)*e(n);
}
```

Figure 13 shows the corresponding memory map. It has been assumed that both the coefficients and samples have been initialized by another routine.

Unlike pure DSP filters, this filter is implemented in two steps, FIR output computation is followed by an update of the coefficients.

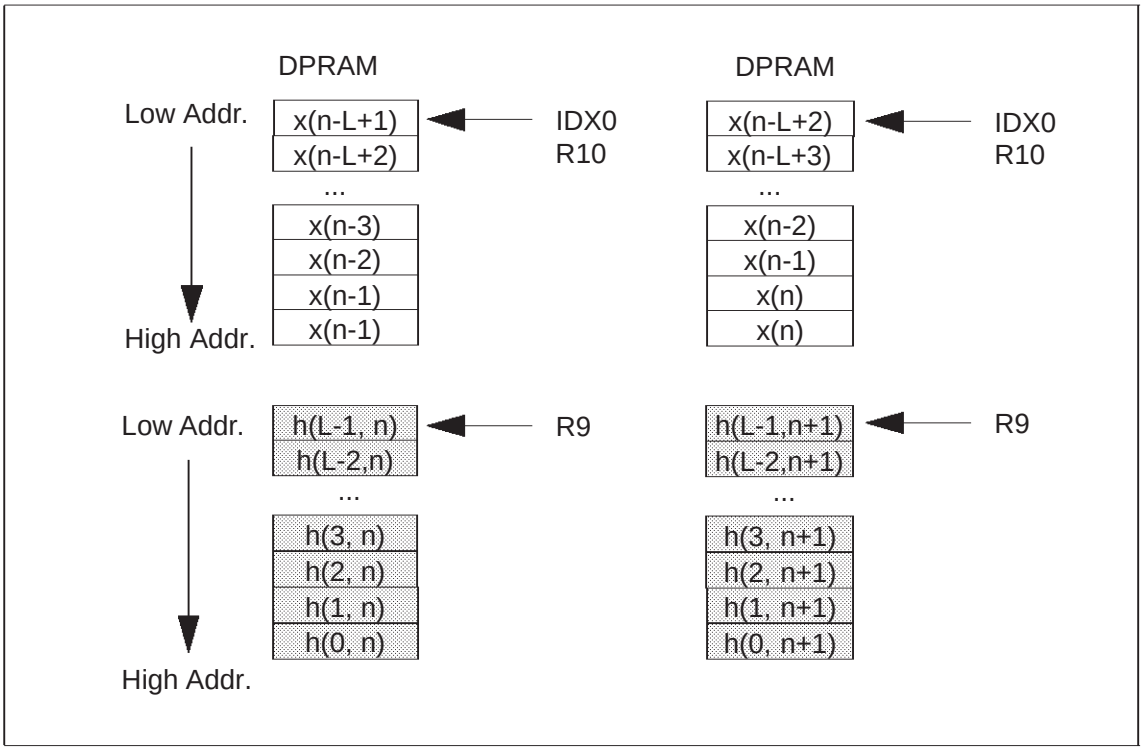


Figure 13 Memory map

This routines assumes that the following general purpose and co-processor registers (SFRs) have been initialized once for ever and that L is less than 31:

- R0 with 0000_H
- R1 with the 16-bit MAXimum tolerated value
- R2 contains the 16-bit MINimum tolerated value
- R9 contains the h(L-1,n) address
- R10 contains the x(n-L+1) address
- IDX0 contains the x(n-L+1) address
- QX0 and QR0 with L-1

```
                ;
                ; Read the new filter input from a (E)SFR and move it into the DPRAM
                ; at x(n-1) address overwriting therefore x(n-1).
                ;
                MOV         @x(n),      ADC_sfr     ; move the new input x(n)
                ;
                ; FIR prolog: first multiplication
                ;
                CoMUL       [IDX0+],    [R9+]       ; (ACC) ← h(L-1)*x(n-L+1)
                                                    ; (IDX0) ← (IDX0)+2,
                                                    ; (R9) ← (R9)+2.
                ;
                ; FIR loop: Repeat L-2 times the same MAC instruction.
                ;
REPEAT L-3 TIMES CoMAC      [IDX0+],    [R9+]       ; (ACC) ← (ACC)+h(i)*x(n-i)
                                                    ; (IDX0) ← (IDX0)+2,
                                                    ; (R9) ← (R9)+2.
                ;
                ; FIR epilog: last MAC instruction and provide y(n) in an appropriate format
                ;
                CoMAC       [IDX0-QX0], [R9-QR0]    ; (ACC) ← (ACC)+h(0)*x(n)
                                                    ; (IDX0) ← (IDX0)-2*(L-1),
                                                    ; (R9) ← (R9)-2*(L-1).
                ;
                ; Shift & Rounding
                ;
                CoASHR      #data3,         rnd     ; (ACC) ← (ACC)>>a #data3
                                                    ;+rnd
```

```
                ;
                ; Limiting
                ;
                CoMIN       R0,          R1              ; (ACC) ← Min((ACC), MAX).
                CoMAX       R0,          R2              ; (ACC) ← Max((ACC),MIN).
                ;
                ;Write the new filter output y(n) into an (E)SFR.
                ;
                NOP                                      ; Pipeline Effect.
                MOV         DAC_sfr,     MAH             ; move the new output y(n).
                ;
                ;Read d(n) and move it into a GPR.
                ;
                MOV         R5,          @d(n)           ; (R15) ¬ d(n)
                ;
                ; Error, e(n), Calculation
                ;
                SUB         R5,          R6              ; (R5) ← d(n)-y(n)=e(n)
                MOV         @e(n),       R5              ; e(n-1) ← e(n)
                CoMUL       R5,          R8              ; (ACC) ← Mu*e(n)
                CoNEG                                rnd ; (ACC) ← -(ACC)+rnd
                CoSTORE     R11,         MAS             ; (R11) ← -Mu*e(n).
                ;
                ; Coefficients' Updating
                ;
                MOV         R3,          #L-2            ; (R3) ← L
                ;
                ; Coefficient Update Prolog.
                ;
                CoLOAD      [R9],         R0             ; (ACC) ← h(L-1,n)
                CoMAC       R11          [R10+],    rnd  ; (ACC) ← h(L-1,n) -
                                                         ; Mu.e(n)*x(n-L+1)+md
                                                         ; (R10) ← (R10)+2.
                CoSTORE     [R9+],       MAS             ; h(L-1,n) ← h(L-1,n+1).
                                                         ; (R9) ← (R9)+2.
                ;
                ; Coefficient Update Loop.
                ;
LMS_LOOP:
                ;
```

```
CoLOAD    [R9],      R0          ; (ACC) ← h(k,n)
CoMACM    R11        [R10+],  rnd ; (ACC) ← h(k,n) -Mu.e(n)*x(n-k)
                                  ;+rnd
                                  ; x(n-k-1) ← x(n-k).
                                  ; (R10) ← (R10)+2.
CoSTORE   [R9+],     MAS          ; h(k,n) ← h(k,n+1).
                                  ; (R9) ← (R9)+2.
;
; End_of_loop Checking.
;
CMPD1     R3         #0h          ; (R3) ← (R3) -1.
JMPR      cc_Z       LMS_LOOP     ; End-of-Loop test & branch.
;
; Coeffi cient Update epilog.
;
CoLOAD    [R9],      R0           ; (ACC) ← h(0,n)
CoMACM    [R10-QR0], R11     rnd  ; (ACC) ← h(0,n) - Mu.e(n)*x(n)+
                                  ; rnd
                                  ; x(n-1) ← x(n).
                                  ; (R10) ← (R10)-2*(L-1).
CoSTORE   [R9-QR0],  MAS          ; h(0,n) ← h(0,n+1).
                                  ; (R9) ← (R9)-2*(L-1).
```

	Instruction Cycles	**Program Words**
Read Input samples	2	4
Initialization	1	2
LMS Loop	4L+2(L-2)+1	25
Post/Pre -Processing	9	16
Write Output sample	2	4
Total	4L+2(L-2)+15	51

Note *The branch penalty in the LMS loop is roughly one third of the execution time of the LMS loop. Nevertheless, as shown in “N-real multiply (windowing)” on page 10, it is possible to minimize the branch penalty by “unrolling” instructions. Therefore, if URF is the UnRolling Factor, the execution times and program words count become respectively:*
*4L+2(L-2)/URF+1 instruction cycles, and 51 +(URF-1)*2 Program words.*

5.2 Extended-precision LMS adaptive filter

16-bit coefficients can be insufficient for LMS filtering. The following routine describes a LMS filter with 32-bit coefficients and 16-bit samples. In most applications 24-bit coefficients provide good results.

The Extended-precision LMS Adaptive filter uses the same naming convention as the single-precision LMS Adaptive filter: $h_L(k,n)$ and $h_H(k,n)$ represent the LS word and MS word (respectively) of the k'th coefficient at time n.

For simplicity, you are advised to clear MP of MSC.

Figure 14 shows the corresponding memory map. It is assumed that both coefficients and samples have been initialized by another routine. Note that unlike the "Single-precision LMS adaptive filter" on page 40, this loop is not "unrolled".

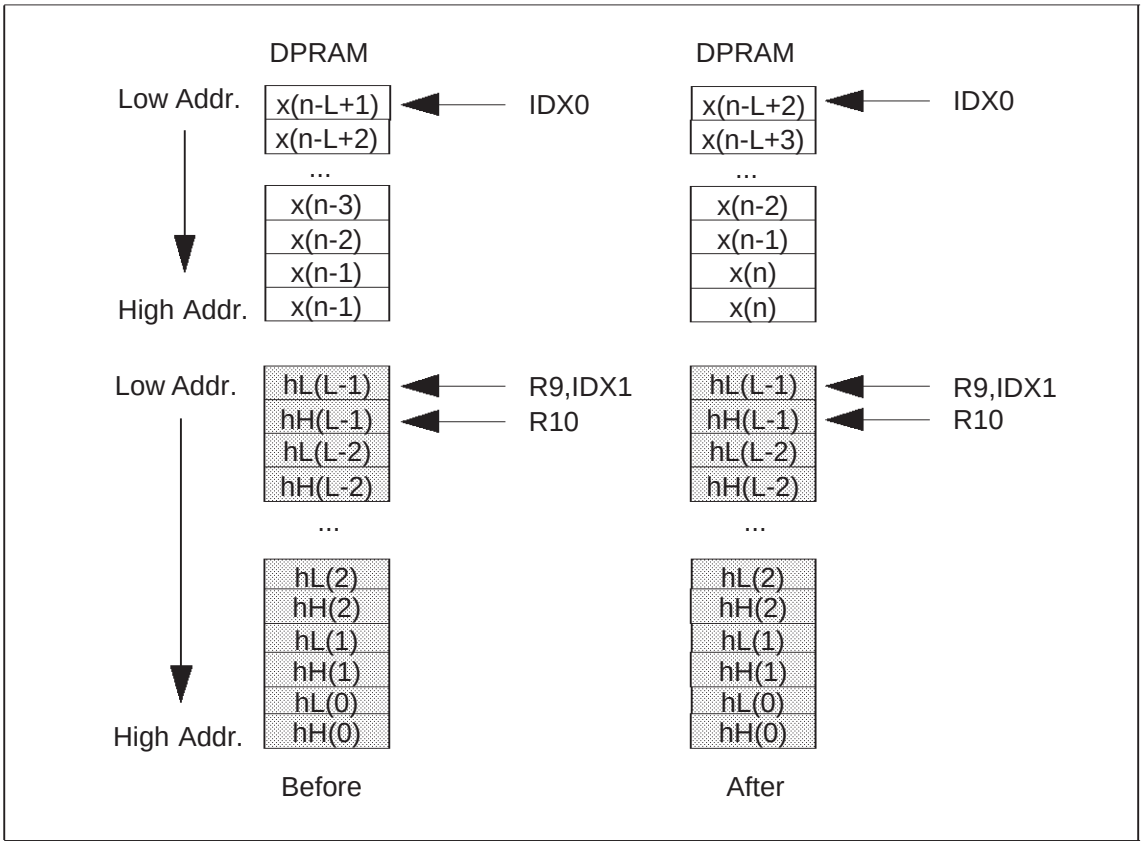


Figure 14 Memory map

This routines assumes that the following general purpose and co-processor registers (SFRs) have been initialized once for ever and that L is less than 31:

- R0 with 0000_H
- R1 with the 16-bit MAXimum tolerated value
- R2 contains the 16-bit MINimum tolerated value
- R9 contains the h_L(L-1) address
- R10 contains the h_H(L-1) address
- IDX0 contains the x(n-L+1) address
- QX0 with L-1
- QX1 and QR0 with 2
- QR1 with 2*L-1

```
                ;
                ; Read the new filter input from a (E)SFR and move it into the DPRAM at
                ; x(n-1) address therefore overwriting x(n-1).
                ;
                MOV         @x(n),      ADC_sfr       ; move the new input x(n)
                ;
                ; FIR prolog (LSWs of Impulse response): first multiplication
                ;
                CoMULsu     [IDX0+],    [R9+QR0]      ; (ACC) ← hL(L-1)*x(n-L+1)
                                                      ; (IDX0) ← (IDX0)+2,
                                                      ; (R9) ← (R9)+4.
                ;
                ; FIR loop (LSWs of Impulse response): Repeat L-2 times the same MAC instruction.
                ;
REPEAT L-3 TIMES CoMACsu    [IDX0+],    [R9+QR0]      ; (ACC) ← (ACC)+hL(i)*x(n-i)
                                                      ; (IDX0) ← (IDX0)+2,
                                                      ; (R9) ← (R9)+4.
                ;
                ; FIR epilog (LSWs of Impulse response): last MAC instruction and provide
                ; y(n) in an appropriate format
                ;
                CoMACsu     [IDX0-QX1], [R9-QR1], rnd ; (ACC) ← (ACC)+hL(0)*x(n)+rnd
                                                      ; & x(n-l+1) ← x(n-L+2),
                                                      ; (IDX0) ← (IDX0)-2*(L-1),
                                                      ; (R9) ← (R9)-2*(2L-1).
                ;
```

```
                ; Shift
                ;
                CoASHR      8,                              ; (ACC)=(ACC)>>8
                CoASHR      8,                              ; (ACC)=(ACC)>>8
                ;
                ; FIR prolog (MSWs of Impulse response): first multiplication
                ;
                CoMAC       [IDX0+],    [R10+QR0]           ; (ACC) ← hH(L-1)*x(n-L+1)
                                                            ; (IDX0) ← (IDX0)+2,
                                                            ; (R10) ← (R10)+4.
                ;
                ; FIR loop (MSWs of Impulse response): Repeat L-2 times the same MAC instruction.
                ;
REPEAT L-3 TIMES CoMAC      [IDX0+],    [R10+QR0]           ; (ACC) ← (ACC)+hH(i)*x(n-i)
                                                            ; & x(n-i-1) ← x(n-i),
                                                            ; (IDX0) ← (IDX0)+2,
                                                            ; (R10) ← (R10)+4.
                ;
                ; FIR epilog (MSWs of Impulse response): last MAC instruction and provide
                ; y(n) in an appropriate format
                ;
                CoMAC       [IDX0-QX1],  [R10-QR1]          ; (ACC) ← (ACC)+hH(0)*x(n)
                                                            ; & x(n-l+1) ← x(n-L+2),
                                                            ; (IDX0) ← (IDX0)-2*(L-1),
                                                            ; (R10) ← (R10)-2*(2L-1).
                ;
                ; Shift & Rounding
                ;
                CoASHR      #data3,          rnd            ; (ACC) ← (ACC)>>a #data3
                                                            ;+rnd
                ;
                ; Limiting
                ;
                CoMIN       R0,         R1                  ; (ACC) ← Min((ACC), MAX).
                CoMAX       R0,         R2                  ; (ACC) ← Max((ACC),MIN).
                ;
                ;Write the new filter output y(n) into a (E)SFR.
                ;
                NOP                                         ; Pipeline Effect.
```

```
                MOV         DAC_sfr,    MAH         ; move the new output y(n).
                ;
                ;Read d(n) and move it into a GPR.
                ;
                MOV         R5,         @d(n)       ; (R5) ← d(n)
                ;
                ; Error, e(n), Calculation
                ;
                SUB         R5,         R6          ; (R5) ← d(n)-y(n)=e(n)
                MOV         @e(n),      R5          ; e(n-1) ← e(n)
                CoMUL       R5,         R8          ; (ACC) ← Mu*e(n)
                CoNEG                               rnd ; (ACC) ← -(ACC)+rnd
                CoSTORE     R11,        MAS         ; (R11) ← -Mu*e(n).
                ;
                ; Coeffi cients' Updating
                ;
                MOV         R12,        IDX0        ; (R12) ← (IDX0)
                MOV         IDX1,       R9          ; (IDX1) ← (R9)
                MOV         R3,         #L-2        ; (R3) ← L
                ;
                ; Coeffi cient Update Prolog.
                ;
                CoLOAD      [IDX1+QX1], [R10-]      ; (ACC) ← h(L-1,n)
                                                    ; (IDX1) ← (IDX1)+4.
                                                    ; (R10) ← (R10)-2.
                CoMAC       R11,        [R12+]      ; (ACC) ← h(L-1,n) -
                                                    ; Mu.e(n)*x(n-L+1)+rnd
                                                    ; (R12) ← (R12)+2.
                CoSTORE     [R10+],     MAL         ; h_L(L-1,n) ← h_L(L-1,n+1).
                                                    ; (R10) ← (R10)+2.
                CoSTORE     [R10+QR0],  MAH         ; h_H(L-1,n) ← h_H(L-1,n+1).
                                                    ; (R10) ← (R10)+2.
                ;
                ; Coeffi cient Update Loop.
                ;
EXT_LMS_LOOP:
                ;
                CoLOAD      [IDX1+QX1], [R10-]      ; (ACC) ← h(k,n)
                                                    ; (IDX1) ← (IDX1)+4.
```

```
                                                    ; (R10) ← (R10)-2.
CoMACM      R11,          [R12+]                    ; (ACC) ← h(k,n) -Mu.e(n)*x(n-k)
                                                    ;+rn
                                                    ; x(n-k-1) ← x(n-k).
                                                    ; (R12) ← (R12)+2.
CoSTORE     [R10+],       MAL                       ; h_L(k,n) ← h_L(k,n+1).
                                                    ; (R10) ← (R10)+2.
CoSTORE     [R10+QR0],    MAH                       ; h_H(k,n) ← h_H(k,n+1).
                                                    ; (R10) ← (R10)+2.
;
; End_of_loop
Checking.
;
CMPD1       R3            #0h                       ; (R3) ← (R3) -1.
JMPR        cc_Z          EXT_LMS_LOOP              ; End-of-Loop test & branch.
;
; Coeffi cient Update epilog.
;
CoLOAD      [IDX1+QX1],   [R10-]                    ; (ACC) ← h(0,n)
                                                    ; (IDX1) ← (IDX1)+4.
                                                    ; (R10) ← (R10)-2.
CoMACM      R11,          [R12+]                    ; (ACC) ← h(0,n) -Mu.e(n)*x(n)
                                                    ;+rnd
                                                    ; x(n-1) ← x(n).
                                                    ; (R12) ← (R12)+2.
CoSTORE     [R10+],       MAL                       ; h_L(0,n) ← h_L(0,n+1).
                                                    ; (R10) ← (R10)+2.
CoSTORE     [R10-QR0],    MAH                       ; h_H(0,n) ← h_H(0,n+1).
                                                    ; (R10) ← (R10)-(2L-1).
```

	Instruction Cycles	**Program Words**
Read Input samples	2	4
Initialization	4	8
EXT LMS Loop	6L+2(L-2)+3	39
Post/Pre -Processing	9	16
Write Output sample	2	4
Total	6L+2(L-2)+20	71

6 Operations on Tables

6.1 Table move

This routine moves a table of L 16-bit data items from one memory location to another (where L is the number of data items). “Orig_Address” is the location of the first element of the table and “Dest_Address” is its location after the table move.

```
;
; MAC dedicated registers' initialization:
;
MOV                 MRW,    #L-1                ; (MRW) ← L-1.
EXTR                #1                          ; next instruction will
                                                ; utilize the ESFR space.
MOV                 IDX0,   #Dst_Address        ; (IDX0) ← Dst_Address.
;
; GPR initialization:
;
MOV                 R1,     #Orig_Address       ; (R1) ← Orig_Address
;
; Move
;
REPEAT MRW TIMES    CoMOV   [IDX0+],    [R1+]   ; ((IDX0)) ← ((R1))
                                                ; (IDX0) ← (IDX0)+2
                                                ; (R1) ← (R1)+2.
```

	Instruction Cycles	Program Words
Total	L+4	9

6.2 Find the index of a maximum value in a table

This routine finds the index of the maximum value of data x(i) for i=1 to L, contained in a table. The first element of the index is located at “Orig_Address”. The operation is performed in two steps, the maximum value is detected, and then the corresponding index is detected. At the end of the routine, the maximum value is stored in the co-processor accumulator and the index is stored in R1 (GPR).

```
;
; MAC dedicated registers' initialization:
```

```
;
MOV                 MRW,    #L-1                    ; (MRW) ← L-1.
EXTR                #1                              ; next instruction will
                                                    ; utilize the ESFR space.
;
; Accumulator Initialization.
;
MOV                 MAH,    #FFFF_H                 ; (MAH) ← FFFF_H,
                                                    ; (MAE) ← FF_H,
                                                    ; (MAL) ← 0000_H.
MOV                 MAL,    #FFFF_H                 ; (MAL) ← FFFF_H,
;
; GPRs initialization:
;
MOV                 R0,     #0000_H                 ; (R0) ← 0000_H
MOV                 R1,     #Orig_Address           ; (R1) ← Orig_Address
;
; First Iteration: Detection of the maximum value
;
REPEAT MRW TIMES    CoMAX   R0,             [R1+]   ; (ACC) ← Max((ACC),x(i))
                                                    ; (R1) ← (R1)+2
;
; Re-initialization:
;
MOV                 MRW,    #L-1                    ; (MRW) ← L-1.
EXTR                #1                              ; next instruction will
                                                    ; utilize the ESFR space.
MOV                 IDX0,   #Dst_Address            ; (IDX0) ← Dst_Address.
MOV                 R1,     #Orig_Address           ; (R1) ← Orig_Address
;
; Second Iteration: Detection of the corresponding index
;
REPEAT MRW TIMES    CoCMP   cc_EQU  R0      [R1+]   ; (MSW) ← (ACC)-x(i)
                                                    ; (R1) ← (R1)+2
```

	Instruction Cycles	Program Words
Total	3L/2+10[1]	22

1. On average

6.3 Compare for search

This routine finds the index of the first piece of data in a table which matches a specified condition “cc_cond” when compared to the contents of the accumulator. It assumes that data is stored in numerical order in the table. The same assumptions are made as for Section 6.1 and Section 6.2. When a match is made, the index is stored in R1.

```
;
; Initialization:
;
MOV             MRW,    #L-1                    ; (MRW) ← L-1.
MOV             R0,     #0000H                  ; (R0) ← 0000H
MOV             R1,     #Orig_Address           ; (R1) ← Orig_Address
;
; Accumulator Initialization.
;
MOV             MAH,    #data16                 ; (MAH) ← #data16,
                                                ; (MAE) ← 8 times (MAH15),
                                                ; (MAL) ← 0000H.
;
; Second Iteration: Detection of the corresponding index
;
REPEAT MRW TIMES CoCMP  cc_GT   R0      [R1+]   ; (MSW) ← (ACC)-x(i)
                                                ; (R1) ← (R1)+2

JNB MSW.12      NO_MATCH        ; test C-flag of MSW and
                                ; jump if no match
...
NO_MATCH
...
```

	Instruction Cycles	Program Words
Total	L/2+9[1]	11

1. On average

7 Summary of Routines

		Instruction Cycles	Program Words
Co-Processor Initialization		10	19
Mathematics	32 by 32 signed multiplication	12	24
	Nth Order Power Series	4N+8	28
	[NxN][Nx1] Matrix Multiply	N^2+4N+7	24
	N-Real Multiply (Windowing)	2.N+2.N/URF[1]+2	5+4.URF
DSP Routines[2]	16x16 L-tap FIR	L	6
	32x16 L-tap FIR	2L+3	18
	DF1[3] Nth Order IIR filter	2N	10
	DF2[4] Nth Order IIR filter	2N+2	14
	DF2 N-cascaded Biquads	10N-1	19
	TF[5] N-cascaded Biquads	13N-1	44
	16x16 L-tap LMS	4L+2(L-2)/URF +1	51+2(URF-1)
	32x16 L-tap LMS	6L+2(L-2)/URF +20	71+2(URF-1)
Operations on Tables	Table Move (L items)	L+4	9
	Find the Index of a Maximum Value in a table (L items)	3L/2+10[6]	22
	"Compare For Search"[7] (L items)	L/2+7[8]	11

Table 1 Summary of routines

1. "URF" stands for "UnRolling Factor".
2. Representative part of the routine only.
3. Direct Form 1
4. Direct Form 2
5. Transpose Form
6. On average
7. First data in a table that matches a specified condition.
8. On average

littlediode
COMPONENTS FOR ALL

Littlediode supplies new, hard to find or obsolete electronic components and semiconductors all over the world.

With over two million different components listed you are sure to find the part you need.

Feel free to visit us today at our online store:

LittleDiode.com

Looking forward to providing you with the best possible service.