The TMS320C2xx Sum-of-Products Methodology

Todd Anderson

SPRA068
May 1996

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE–SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Contents

ABSTRACT

This application report shows a series of programs transforming a normal sum-of-products routine to one using the MACD instruction. The sum of products operation creates an IIR filter. Each program shows an incremental change from the previous program that leads to the final filter implementation. The series of six stand-alone programs run on a simulator. Using each one in turn assures that the final program operates as the original.

INTRODUCTION

This application uses the TMS320C2xx to program a fourth-or higher order filter. A series of stand-alone programs shows the progression from direct addressing with the load accumulator and multiply (LT/MPY) instructions, to indirect addressing with the multiply and accumulate with data move (MAC/MACD) instruction. Each program runs on the ‘C2xx simulator.

The first program uses the LT/MPY instructions and the direct addressing mode to program the IIR. The second builds upon the first, changing from the direct to the indirect addressing mode. The third program of the series rearranges the memory accesses for full indirection. The fourth introduces the MAC/MACD instruction to the filter program. The fifth eliminates repetitive MACD instructions by using the repeat (RPT) instruction. The last program optimizes the RPT instructions. This step-by-step progression verifies that the final program does indeed duplicate the results of the first program. All the programs are stand-alone programs for the ‘C2xx simulator.

PROGRAMMING A FOURTH-ORDER FILTER

The software program for a fourth-order filter derives from the IIR signal flow diagram of Figure 1. The diagram shows the IIR filter as two finite impulse response (FIR) filters, one an all-pole system and the other an all-zero system. This view of the IIR provides a straight forward way of programming the overall system.

The method for programming the signal flow diagram of Figure 1 requires the summation of the all-zero section, then the summation of the all-pole system. The all-zero system provides the term needed by the all-pole system to compute the sum-of-products operation. The output of the all-pole section is the sum of products:

$$y0 = u4(A4) + u3(A3) + u2(A2) + u1(A1) + u0(A0) + y4(B4) + y3(B3) + y2(B2) + y1(B1)$$

The general expression for the sum of products of an n-order filter is:

$$y(n) = \sum_{k=0}^{n} a(k)u(k) - \sum_{k=1}^{n} b(k)y(k)$$

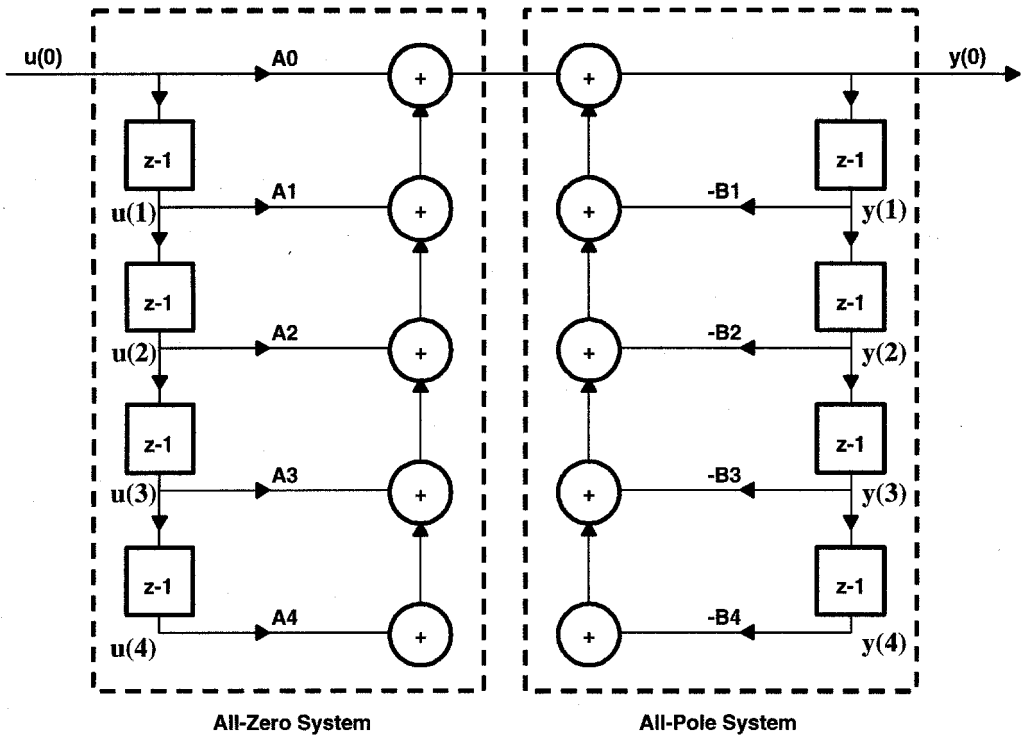

Figure 1. IIR Realization

IIR Filter Initialization

The programs for a fourth-order filter share a basic initialization of elements. This initialization assures correct program operation. The program sets the CPU resources, the memory, and the registers to known values.

The initialization sets the CPU configurations, such as product shift mode, sign extension, overflow, and memory, to an appropriate state. If the reset state of the configurations conforms with the program requirements, then the configurations remain the same.

Program memory contains the coefficients and data. Coefficients are either immediate data or reside in the program memory. The initialization sets the data and coefficients to the reset state.

The program initializes registers used for the addressing or storage of intermediate results. These registers include the auxiliary registers, the AR pointer, the accumulator, and the P registers.

IIR Filter using LTD/MPY and the Direct Addressing Mode

The first stand-alone program uses the LTD and MPY instructions of the ‘C2xx to execute the sum of products. Both instructions use the direct addressing mode to access data and coefficient tables. The LTD instruction (load the temporary register and accumulate) contains a DMOV instruction. The DMOV instruction moves data from its direct memory address (DMA) to the next location (DMA + 1). After the program calculates the product, it copies the data to the next memory (DMA + 1) location. This creates the z^{-1} delay shown in Figure 1. The ‘C2xx allows this type of buffering, known as ripple buffering, with on-chip RAM block 0-2. The program configures the RAM as data memory through the linker command (refer to the Linker Command File).

```
;-------------------------------------------------------
;Program Name:  MULT1.ASM
;
;This program implements a fourth-order filter using the
;load and multiply instruction with direct addressing.
;This program is the first in a six part series.
;-------------------------------------------------------

;Reserve Data Memory
;The following section of code reserves data memory for
;the coefficients and data

       .bss   u0,5,1
u1     .equ   u0+1
u2     .equ   u0+2
u3     .equ   u0+3
u4     .equ   u0+4

       .bss   A0,5,1
A1     .equ   A0+1
A2     .equ   A0+2
A3     .equ   A0+3
A4     .equ   A0+4

       .bss   y1,4,1
y2     .equ   y1+2
y3     .equ   y1+3
y4     .equ   y1+4

       .bss   B1,4,1
B2     .equ   B1+1
B3     .equ   B1+2
B4     .equ   B1+3

.text
b      _begin

A_tbl:
   .word 1, 2, 3, 4, 5        ;Coefficient table: A
B_tbl:
   .word 9, 8, 7, 6           ;Coefficient table: B
u_tbl:
   .word 2, 4, 6, 8, 10       ;Data table: u
y_tbl:
   .word 20, 18, 16, 14       ;Data table: y
;-------------------------------------------------------
;transfer coefficients and values from program memory to
;data memory
;-------------------------------------------------------
_begin:
setc   INTM                   ;global interrupt disable

mar    *,ar1                  ;auxiliary register =1

lar    ar1,#A0                ;move coefficients from
rpt    #5-1                   ;program memory to data
blpd   #A_tbl,*+              ;memory

lar    ar1,#B1
rpt    #4-1
blpd   #B_tbl,*+

lar    ar1,#u0                ;move data from program memory
rpt    #5-1                   ;to data memory
blpd   #u_tbl,*+

lar    ar1,#y1
rpt    #4-1
blpd   #y_tbl,*+

;-------------------------------------------------------------
;begin filter calculations
;-------------------------------------------------------------
```

```
ldp    #u0
lacl   #0                  ;set the accumulator to zero
lt     u4                  ;begin the sum of products
mpy    A4      ;(u4*A4) +
ltd    u3
mpy    A3      ;(u3*A3) +
ltd    u2
mpy    A2      ;(u2*A2) +
ltd    u1
mpy    A1      ;(u1*A1) +
ltd    u0
mpy    A0      ;(u0*A0)

lta    y4
mpy    B4      ;(y4*B4)
ltd    y3
mpy    B3      ;(y3*B3)
ltd    y2
mpy    B2      ;(y2*B2)
ltd    y1
mpy    B1      ;(y1*B1)
apac

sacl   y1                  ;store result
b      _begin              ;continuous loop-no exit
```

IIR Filter Using LTD and the Indirect Addressing Mode

The second program uses the indirect addressing mode. The indirect addressing mode allows access to any location in the 64K data memory space through the auxiliary registers. The LT/LTD instructions use an auto decrement (*–) option. This option decrements the DMA after the memory access. The LTD with an auto decrement option allows the rippling of data through the delay line of the filter. Note the array access is from the bottom.

```
;--------------------------------------------------------
;Program Name:  MULT2.ASM
;This program implements a fourth–order filter using the
;load and multiply instruction with indirect addressing
;on the arguments of LT.
;This program is the second in a six part series.
;--------------------------------------------------------

;Reserve Data Memory
;The following section of code reserves data memory for
;the coefficients and data
       .bss   u0,5,1
u1     .equ   u0+1
u2     .equ   u0+2
u3     .equ   u0+3
u4     .equ   u0+4

       .bss   A0,5,1
A1     .equ   A0+1
A2     .equ   A0+2
A3     .equ   A0+3
A4     .equ   A0+4

       .bss   y1,4,1
y2     .equ   y1+2
y3     .equ   y1+3
y4     .equ   y1+4

       .bss   B1,4,1
B2     .equ   B1+1
B3     .equ   B1+2
B4     .equ   B1+3

.text
b      _begin

A_tbl:
    .word 1, 2, 3, 4, 5        ;Coefficient table: A
```

```
B_tbl:
   .word 9, 8, 7, 6          ;Coefficient table: B
u_tbl:
   .word 2, 4, 6, 8, 10      ;Data table: u
y_tbl:
   .word 20, 18, 16, 14      ;Data table: y
;-----------------------------------------------------------
;transfer coefficients and values from program memory to
;data memory
;-----------------------------------------------------------
_begin:
setc   INTM                  ;global interrupt disable

mar    *,ar1                 ;load address pointer

lar    ar1,#A0               ;move coefficients from
rpt    #5-1                  ;program memory to data
blpd   #A_tbl,*+             ;memory

lar    ar1,#B1
rpt    #4-1
blpd   #B_tbl,*+

lar    ar1,#u0               ;move data from program memory
rpt    #5-1                  ;to data memory
blpd   #u_tbl,*+

lar    ar1,#y1
rpt    #4-1
blpd   #y_tbl,*+

;-----------------------------------------------------------
;begin filter calculations
;-----------------------------------------------------------
ldp    #u0                   ;set up page register
lacl   #0                    ;set the accumulator to zero

mar    *,ar2                 ;point at auxiliary register 2
lar    ar2,#u4               ;point at u4

                             ;begin the sum of products
lt     *-                    ;load u4, point at u3
mpy    A4                    ;(u4*A4)
ltd    *-                    ;load u3, point at u2
mpy    A3                    ;(u3+A3)
ltd    *-                    ;load u2, point at u1
mpy    A2                    ;(u2+A2)
ltd    *-                    ;load u1, point at u0
mpy    A1                    ;(u1+A1)
ltd    *                     ;load u0
mpy    A0                    ;(u0+A0)
lar    ar2,#y4               ;point at y4
lta    *-                    ;load y4, point at y3
mpy    B4                    ;(y4*B4)
ltd    *-                    ;load y3, point at y2
mpy    B3                    ;(y3*B3)
ltd    *-                    ;load y2, point at y1
mpy    B2                    ;(y2*B2)
ltd    *-                    ;load y1
mpy    B1                    ;(y1*B1)
apac

sacl   *                     ;store result
b      _begin                ;continuous loop-no exit
```

IIR Filter Using Full Indirection

This program extends the indirect addressing of MULT2.ASM by rearranging the placement of the tables in memory. The location of the `y_tbl` is above the `u_tbl` in data memory. This eliminates the instruction to load auxiliary register 2. The LTD instruction ripples through the two tables in the data buffer. The program now uses the full indirection of the ‘C2xx..

```
;-------------------------------------------------------
;Program Name:  MULT3.ASM
;This program creates a fourth-order filter using the
;load and multiply instruction with indirect addressing on ;argument of LT. This
program rearranges memory for full ;indirection.
;
;This program is third in a six part series.
;-------------------------------------------------------

;Reserve Data Memory
;The following section of code reserves data memory for
;the coefficients and data

       .bss   u0,5,1
u1     .equ   u0+1
u2     .equ   u0+2
u3     .equ   u0+3
u4     .equ   u0+4

       .bss   A0,5,1
A1     .equ   A0+1
A2     .equ   A0+2
A3     .equ   A0+3
A4     .equ   A0+4

       .bss   y1,4,1
y2     .equ   y1+2
y3     .equ   y1+3
y4     .equ   y1+4

       .bss   B1,4,1
B2     .equ   B1+1
B3     .equ   B1+2
B4     .equ   B1+3

.text
b      _begin

A_tbl:
    .word 1, 2, 3, 4, 5     ;Coefficient table: A
B_tbl:
    .word 9, 8, 7, 6        ;Coefficient table: B
u_tbl:
    .word 2, 4, 6, 8, 10    ;Data table: u
y_tbl:
    .word 20, 18, 16, 14    ;Data table: y

;-------------------------------------------------------
;transfer coefficients and values from program memory to
;data memory
;-------------------------------------------------------
_begin:
setc   INTM                ;global interrupt disable

mar    *,ar1               ;auxiliary register = 1

lar    ar1,#A0             ;move coefficients from
rpt    #5-1                ;program memory to data
blpd   #A_tbl,*+

lar    ar1,#B1
rpt    #4-1
blpd   #B_tbl,*+

lar    ar1,#u0             ;move data from program memory
```

```
rpt    #5-1      ;to data
blpd   #u_tbl,*+

lar    ar1,#y1
rpt    #4-1
blpd   #y_tbl,*+

;-------------------------------------------------------
;begin filter calculations
;-------------------------------------------------------
ldp    #u0                ;set up page register
lacl   #0                 ;set the accumulator to zero
mar    *,ar2              ;point at auxillary register 2
lar    ar2,#u4            ;point at u4

                          ;begin the sum of products
lt     *-                 ;load u4, point at u3
mpy    A4                 ;(u4*A4)
ltd    *-                 ;load u3, point at u2
mpy    A3                 ;(u3+A3)
ltd    *-                 ;load u2, point at u1
mpy    A2                 ;(u2+A2)
ltd    *-                 ;load u1, point at u0
mpy    A1                 ;(u1+A1)
ltd    *-                 ;load u0, point at y4
mpy    A0                 ;(u0+A0)

lta    *-                 ;load y4, point at y3
mpy    B4                 ;(y4*B4)
ltd    *-                 ;load y3, point at y2
mpy    B3                 ;(y3*B3)
ltd    *-                 ;load y2, point at y1
mpy    B2                 ;(y2*B2)
ltd    *-                 ;load y1
mpy    B1                 ;(y1*B1)
apac
sacl   *                  ;store result
b      _begin             ;continuous loop no exit
```

IIR Filter Using MAC/MACD

The fourth program replaces the LTD/MPY instructions with the MAC/MACD instructions. The MAC instruction multiplies the `u_tbl` or `y_tbl` (data memory value) by the coefficient (program memory value). The MAC option, *–, decrements the DMA and retrieves the next value. The MACD instruction replaces the LTD and MPY instructions. The MACD instruction adds a data move to the MAC instruction. This compensates for the z^{-1} delay (Figure 1) by rippling the data buffer.

```
;-------------------------------------------------------
;Program Name:  MULT4.ASM
;This program implements a fourth-order filter using the
;MACD instruction.
;This program is the fourth in a six part series.
;-------------------------------------------------------

;Reserve Data Memory
;The following section of code reserves data memory for
;the coefficients and data

       .bss   y1,9,1
y2     .equ   y1+1
y3     .equ   y1+2
y4     .equ   y1+3

u1     .equ   y1+4
u2     .equ   y1+5
u3     .equ   y1+6
u4     .equ   y1+7

.text
b      _begin
```

```
A_tbl:
   .word 1, 2, 3, 4, 5    ;Coefficient table: A
B_tbl:
   .word 9, 8, 7, 6       ;Coefficient table: B
u_tbl:
   .word 2, 4, 6, 8, 10   ;Data table: u
y_tbl:
   .word 20, 18, 16, 14   ;Data table: y

;---------------------------------------------------------
;transfer coefficients and values from program memory to

;data memory
;---------------------------------------------------------
_begin:
setc   INTM               ;global interrupt disable

ldp    #A4                ;be on A4's data page
blpd   #A_tbl+4,A4        ;initialize A4

mar    *,ar1              ;auxiliary register = 1

lar    ar1,#u0            ;move data from program memory
rpt    #5-1               ;to data memory
blpd   #u_tbl,*+

lar    ar1,#y1
rpt    #4-1
blpd   #y_tbl,*+

;---------------------------------------------------------
;begin filter calculations
;---------------------------------------------------------
ldp    #u0                ;set up page register
lacl   #0                 ;set the accumulator to zero

mar    *,ar2              ;point at auxiliary register 2
lar    ar2,#u4            ;point at u4

                          ;begin sum of products
lt     *-                 ;load u4, point at u3
mpy    An4                ;(u4*A4) +

macd   A_tbl+3,*-         ;load u3, point at u2
              ;(u3*A3) +
macd   A_tbl+2,*-         ;load u2, point at u1
              ;(u2*A2) +
macd   A_tbl+1,*-         ;load u1, point at u0
              ;(u1*A1) +
macd   A_tbl+0,*-         ;load u0, point at y4
              ;(u0 *A0) +
macd   B_tbl+3,*-         ;load y4, point at y3
              ;(y4*B4) +
macd   B_tbl+2,*-         ;load y3, point at y2
              ;(y3*B3) +
macd   B_tbl+1,*-         ;load y2, point at y1
              ;(y2*B2) +
macd   B_tbl+0,*-         ;load y1, point at y0
              ;(y1*B1) +
apac

sacl   *                  ;store result
b      _begin             ;continuous loop-no exit
```

IIR Filter Using the Repeat Instruction

MULT5.ASM uses repetitive MACD instructions. This program eliminates the multiple MACDs by using the ‘C2xx repeat (RPT) instruction. The RPT instruction executes the instruction following it n + 1 times. The repeat instruction does not allow interrupts during execution.

```
;-------------------------------------------------------
;Program Name:  MULT5.ASM
;This program implements a fourth–order filter using the
;rpt and macd instructions.
;This program is the fifth in a six part series.
;-------------------------------------------------------
;
;Reserve Data Memory
;The following section of code reserves data memory for
;the coefficients and data

   .bss   y1,9,1
y2 .equ   y1+1
y3 .equ   y1+2
y4 .equ   y1+3

u0 .equ   y1+4
u1 .equ   y1+5
u2 .equ   y1+6
u3 .equ   y1+7
u4 .equ   y1+8

   .bss   A4,1

.text
b  _begin

A_tbl:
   .word 1, 2, 3, 4, 5        ;Coefficient table: A
B_tbl:
   .word 9, 8, 7, 6           ;Coefficient table: B
u_tbl:
   .word 2, 4, 6, 8, 10       ;Data table: u
y_tbl:
   .word 20, 18, 16, 14       ;Data table: y

;-------------------------------------------------------
;transfer coefficients and values from program memory to
;data memory
;-------------------------------------------------------
_begin:
setc   INTM                   ;global interrupt disable

ldp    #A4                    ;be on A4’s page
blpd   #A_tbl,A4              ;initialize A4

mar    *,ar1                  ;load address pointer

lar    ar1,#u0                ;move data from program memory
rpt    #5–1                   ;to data memory
blpd   #u_tbl,*+

lar    ar1,#y1
rpt    #4–1
blpd   #b_tbl,*+

;-------------------------------------------------------
;begin filter calculations
;-------------------------------------------------------
ldp    #u0                    ;set up page register
lacl   #0                     ;set the accumulator to zero

mar    *,ar2                  ;point to auxiliary register 2
lar    ar2,#u4                ;point at u4

                              ;begin the sum of products
```

```
lt     *–                     ;load u4, point at u4
mpy    An4                    ;(u4*A4) +

rpt    #4–1
macd   A_tbl+1,*–             ;+ (u3*A3) + (u2*A2) + (u1*A1)
                              ;+ (u0*A0)
mac    B_tbl,*–               ;load y4, point at y3
                              ;(y4*B4) +
rpt    #3–1
macd   B_tbl+1,*–             ;(y3*B3) + (y2*B2) + (y1*B1)

apac
sacl   *                      ;store result
b  _begin                     ;continuous loop–no exit
```

Optimizing the Repeat Instruction

This program optimizes the code of MULT5.ASM by combining the two RPT/MACD sequences into one. This optimization depends on `y_tbl` and `u_tbl` being stored in sequential memory locations. This sequential storage of data allows the z^{-1} delay through the ripple buffer.

```
;–––––––––––––––––––––––––––––––––––––––––––––––––––––
;Program Name:  MULT6.ASM
;This program implements a fourth–order filter by
;optimizing the use of macd and the repeat instructions.
;This program is the sixth in a six part series.
;–––––––––––––––––––––––––––––––––––––––––––––––––––––

;Reserve Data Memory
;The following section of code reserves data memory for
;the coefficients and data

   .bss   y1,9,1
y2 .equ   y1+1
y3 .equ   y1+2
y4 .equ   y1+3

u0 .equ   y1+4
u1 .equ   y1+5
u2 .equ   y1+6
u3 .equ   y1+7
u4 .equ   y1+8

   .bss   A4,1

.text
b  _begin

A_tbl:
   .word 1, 2, 3, 4, 5        ;Coefficient table: A
B_tbl:
   .word 6, 7, 8, 9           ;Coefficient table: B
u_tbl:
   .word 2, 4, 6, 8, 10       ;Data table: u
y_tbl:
   .word 20, 18, 16, 14       ;Data table: y

;–––––––––––––––––––––––––––––––––––––––––––––––––––––
;transfer coefficients and values from program memory to
;data memory
;–––––––––––––––––––––––––––––––––––––––––––––––––––––
_begin:
setc   INTM                   ;global interrupt disable

ldp    #A4                    ;be on A4's page
blpd   #A_tbl,A4              ;initialize A4

mar    *,ar1                  ;load address pointer
```

```
lar    ar1,#u0                ;move data from program memory
rpt    #4-1                   ;to data memory
blpd   #u_tbl,*+

lar    ar1,#y1
rpt    #4-1
blpd   #y_tbl,*+

;-------------------------------------------------------
;begin filter calculations
;-------------------------------------------------------
ldp    #u0                    ;set up page register
lacl   #0                     ;set the accumulator to zero

mar    *,ar2                  ;point to auxiliary register 2
lar    ar2,#u4                ;point at u4

                              ;begin the sum of products
lt     *-                     ;load u4, point at u4
mpy    An4                    ;(u4*A4) +

rpt    #8-1
macd   A_tbl+1,*-             ;+ (u3*A3) + (u2*A2) + (u1*A1)
                              ;+ (u0*A0) +(y4*B4) +
                              ;(y3*B3) + (y2*B2) + (y1*B1)

apac
sacl   *                      ;store result
b   _begin                    ;continuous loop-no exit
```

LINKER COMMAND FILE

The linker command file generates an output file, object file, and map file of each program. The **X** in the filename is a variable number denoting the program number from the series. The linker command file specifies location of data and program memory.

```
-o mult X.out        /*output file name*/
-m mult X.map        /*map file name*/

mult X.obj           /*object file name*/

MEMORY          /*program memory*/
{
   PAGE0:
      ROM:   origin = 00000h, length = 00100h

   PAGE1:           /*data memory*/
      B2: origin = 00060h, length = 00020h
      B0: origin = 00200h, length = 00100h
      B1: origin = 00300h, length = 00100h
}
SECTIONS
{
   .text:> ROM   PAGE0
   .bss:> B2 PAGE1
}
```

For the first program (MULT1.ASM), the linker file is:

```
-o mult1.out        /*output file name*/
-m mult1.map        /*map file name*/

mult1.obj.
MEMORY          /*program memory*/
{
   PAGE0:
      ROM:   origin = 00000h, length = 00100h

   PAGE1:           /*data memory*/
      B2: origin = 00060h, length = 00020h
```

```
       B0: origin = 00200h, length = 00100h
       B1: origin = 00300h, length = 00100h
}
SECTIONS
{
    .text:> ROM   PAGE0
    .bss:> B2 PAGE1
}
```

IIR PERFORMANCE CONSIDERATIONS

The series of six programs shows the IIR used in several different ways. Though the ultimate goal is to program an IIR with the MACD instruction, each particular program provides a valid approach to programming the filter on the ‘C2xx. The series of programs allow the user to make tradeoffs between performance, program memory utilization, and data memory utilization. Some conditions that affect the system performance are: coefficient storage, in-line or looped code, repeat instruction, and multiplier operations.

The program locates coefficients in either program memory or data memory. If the coefficients reside in program memory they are accessed as immediate values (shown in MULT1.ASM) or via the MAC group of instructions (shown in MULT4.ASM). The program achieves its highest performance when the coefficients reside in program memory because the program passes the data and program operands simultaneously to the multiplier.

Filter programs using in-line code provide a higher performance than those using looped code. The cost of using this in-line code is the increase in program memory usage. MULT5.ASM and MULT6.ASM use considerably less program memory than the first four programs of the series. The block capability of the ‘C2xx allows the use of the compact looped code without the software overhead required with looped code.

The ‘C2xx supports single-instruction repeats (RPT) as shown in MULT5.ASM and MULT6.ASM. The filter program achieves its highest performance using the repeat instruction with the coefficients residing in program memory. The repeat instruction executes completely without interruption; whether this is an advantage or disadvantage depends upon the application.

The ‘C2xx multiplier operations require arguments from data memory or data and program memory. For longer filters the MAC instructions used in repeat mode provide the highest performance (MULT5.ASM and MULT6.ASM). For shorter filters the LT/MPY instructions require less initial overhead.

SUMMARY

This application report provides a series of stand-alone filter programs. The programs create a fourth-order filter (IIR) on the TMS320C2xx. Each program improves upon the previous program by using the distinct features and options of the ‘C2xx software. The programs provide a software progression from a program using the direct addressing mode with the LT/MPY instruction to one using the indirect addressing mode with the MACD instruction. The final program details a high performance IIR filter.