Appendix B

Programming Examples

APPENDIX B. PROGRAMMING EXAMPLES

CONTENTS

B. PROGRAMMING EXAMPLES

This appendix contains DSP16/DSP16A source-file listings for four complete sample programs. These programs are intended to demonstrate the use of various programming techniques and may be easily modified for use in various applications. Comments within the source files provide all the information necessary to understand the program's function and to make minor modifications.

The programs included are:

- FIR filter
- IIR filter
- 4 × 4 matrix multiplication
- Find the maximum element in a vector

B.1 FIR FILTER

```
/*                                                              */
/*     DSP16/DSP16A FIR filter design example:                  */
/*                                                              */
/*             The following code represents a 66th             */
/*     order FIR filter.  Linear inputs are provided            */
/*     via the serial input.  Linear outputs are sent           */
/*     to the serial output.                                    */
/*                                                              */
/*             The filter was designed using a Hamming          */
/*     window on an ideal lowpass filter with the following     */
/*     parameters.                                              */
/*                     f(cutoff) =     1000 Hz                  */
/*                     f(stop) =       1500 Hz                  */
/*                     f(sample) =     10000 Hz                 */
/*                                                              */
/*     The stopband attenuation is constrained to be            */
/*     above 50 dB.                                             */
/*                                                              */

.ram
X66:    65*int
X1:     int
ibuf:   int
.endram

/*                                                              */
/*                     66th order FIR filter                    */
/*                                                              */
/*             This FIR example uses a modulo addressed         */
/*     delay line.  Outputs are calculated from X(n-66)         */
/*     toward X(n).  The inner loop of the filter uses          */
/*     67 multiplies and requires 89 cycles (6.7 µs at 75 ns),  */
/*     which easily meets the 10000 Hz requirement (100 µs).    */
/*                                                              */
/*             a0:     scratch calculations                     */
/*             r0:     input buffer location                    */
/*             r1:     delay line pointer                       */
/*             pt:     coefficient pointer                      */
/*                                                              */
/*     Also modifies rb, re, i, x, y, p, sdx                    */
/*                                                              */
```


www.DataSheet4U.com

```
fir66:          auc=0x02        /* Use fractional notation      */
                pt=H66          /* Initialize pointers          */
                r0=ibuf
                i=-66
                rb=X66
                re=X1
                r1=X66
loop:           y=0x0010        /* Main loop                    */

wait:           a0=pioc         /* Wait for valid input         */
                a0&y
        if eq   goto wait
                *r0=sdx         /* Input sample                 */

                /*      Perform Convolution     */

                a0=p                    y=*r1++         x=*pt++
        do 65 {
                a0=a0-p         p=x*y   y=*r1++         x=*pt++
        }
                a0=a0-p         p=x*y   y=*r0           x=*pt++i
                a0=a0-p         p=x*y   *r1++=y
                a0=a0-p
                sdx=a0
endl:   goto    loop

/*                                                              */
/*              Coefficients from h(n-66) to h(n)               */
/*                                                              */

H66:    int     -0.000545
        int     -0.000000
        int     0.000641
        int     0.001046
        int     0.000876
        int     -0.000000
        int     -0.001275
        int     -0.002184
        int     -0.001864
        int     -0.000000
        int     0.002673
        int     0.004485
        int     0.003739
        int     -0.000000
        int     -0.005112
        int     -0.008391
        int     -0.006859
        int     -0.000000
        int     0.009085
        int     0.014743
        int     0.011954
        int     -0.000000
        int     -0.015755
        int     -0.025682
        int     -0.021038
        int     -0.000000
        int     0.028989
```



```
        int     0.049177
        int     0.042713
        int     -0.000000
        int     -0.073628
        int     -0.157832
        int     -0.224610
        int     -0.250000
        int     -0.224610
        int     -0.157832
        int     -0.073628
        int     -0.000000
        int     0.042713
        int     0.049177
        int     0.028989
        int     -0.000000
        int     -0.021038
        int     -0.025682
        int     -0.015755
        int     -0.000000
        int     0.011954
        int     0.014743
        int     0.009085
        int     -0.000000
        int     -0.006859
        int     -0.008391
        int     -0.005112
        int     -0.000000
        int     0.003739
        int     0.004485
        int     0.002673
        int     -0.000000
        int     -0.001864
        int     -0.002184
        int     -0.001275
        int     -0.000000
        int     0.000876
        int     0.001046
        int     0.000641
        int     -0.000000
        int     -0.000545
```


www.DataSheet4U.com

B.2 IIR FILTER

```
/*                                                              */
/*     DSP16/DSP16A IIR filter design example:                  */
/*                                                              */
/*             The following code represents a 5th              */
/*     order IIR filter. It is implemented as 3 five            */
/*     multiply second-order sections. (Thus the code is        */
/*     actually the same as a 6th order filter). Linear         */
/*     inputs are provided via the serial input and             */
/*     prescaled with no loss of precision. Linear outputs      */
/*     are sent to the serial output.                           */
/*                                                              */
/*             The filter was designed by performing a          */
/*     bilinear transformation on a classical elliptic filter   */
/*     designed with the following parameters.                  */
/*                                                              */
/*                     f(cutoff) =     1000 Hz                  */
/*                     f(stop) =       1500 Hz                  */
/*                     f(sample) =     10000 Hz                 */
/*                                                              */
/*     The stopband attenuation is constrained to be            */
/*     above 50 dB.                                             */
/*                                                              */
.ram
statev:         6*int           /* Six state variables          */
.endram

/*             The IIR coding example uses the following        */
/*     DSP16/DSP16A resources:                                  */
/*                                                              */
/*             a0:     input/output for each section            */
/*             r1:     state variable pointer                   */
/*             pt:     pointer to filter coefficients           */
/*             j, k:   for compound addressing mode             */
/*                                                              */
/*     This coding example demonstrates the use of              */
/*     compound addressing for maintaining the state            */
/*     variable delay line.  The form is direct form II.        */
```



```
iir5:           auc=0x02        /* Use fractional notation      */
                j=-2            /* Initialize registers         */
                k=3

loop:           pt=coef         /* Main program loop            */
                r1=statev
                y=0x0010

wait:           a0=pioc         /* Wait for valid input         */
                a0&y
        if eq   goto wait
                a0=sdx
                a0=a0>>1        /* Prescale input value         */
                a0=a0>>1


                /* Perform three second-order sections          */

                                        y=*r1++     x=*pt++     /* a11 */
                        p=x*y           y=*r1--     x=*pt++     /* a21 */
        do 3 {
                a0=a0-p     p=x*y       y=*r1++     x=*pt++     /*  b1(n)   */
                a0=a0-p     p=x*y       *r1zp:y     x=*pt++     /*  b2(n)   */
                a0=p        p=x*y       y=a0        x=*pt++     /*  b0(n)   */
                a0=a0+p     p=x*y       *r1jk:y     x=*pt++     /* a1(n+1) */
                a0=a0+p     p=x*y       y=*r1--     x=*pt++     /* a2(n+1) */
        }
                sdx=a0
endl:   goto    loop

coef:   int     -0.759982               /* a11 */
        int     0.0                     /* a21 */
        int     0.060045                /* b11 */
        int     0.0                     /* b21 */
        int     0.060045                /* b01 */

        int     -1.508821               /* a12 */
        int     0.70049                 /* a22 */
        int     -0.108068               /* b12 */
        int     0.245454                /* b22 */
        int     0.245952                /* b02 */

        int     -1.530470               /* a13 */
        int     0.905134                /* a23 */
        int     -1.976028               /* b13 */
        int     1.758388                /* b23 */
        int     1.759004                /* b03 */
```


www.DataSheet4U.com

B.3 MATRIX MULTIPLICATION

```
/*                                                              */
/*     DSP16/DSP16A 4x4 matrix multiply programming example:    */
/*                                                              */
/*     The following code implements a 4x4 matrix multiply.     */
/*     In this example matrices A and B are input through       */
/*     the parallel i/o prior to being multiplied. The result   */
/*     is stored in matrix C. Matrices are input row by row     */
/*     and stored in row order. The resulting C matrix is       */
/*     stored in the same fashion.                              */
/*                                                              */
/*             The matrix multiply routine demonstrates the     */
/*     use of "fast sets" to perform pointer arithmetic         */
/*     both inside and outside of the cache.                    */

/*     Ram variables   */
.ram
A:      16*int
B:      16*int
C:      16*int
.endram


/*             Matrix Multiply routine:                 */
/*                                                      */
/*                rl: points to A                       */
/*                r0: points to B                       */
/*                r2: points to C                       */
/*                                                      */
/*             where C = A * B                          */
/*                                                      */
/*     a0, c1, rb, re, j, x, y registers are also       */
/*     used.                                            */
```



```
Mmult:          c1=-3           /* Initialize registers */
                auc=0x00        /* Integer mode         */

/*              Read in Matrices                         */

                r0=A
                a0=pdx0
        do 32 {
                a0=pdx0
                *r0++=a0
              }

                rb=A            /* Reinitialize regs    */
                re=A+3
                r2=C
                r1=A

/*              Main program loop                        */

loop:           r0=B
                j=4
                                        y=*r0++j
                                        x=*r1++
        do 4 {
                        p=x*y           y=*r0++j
                a0=p                    x=*r1++
                        p=x*y           y=*r0++j
                a0=a0+p                 x=*r1++
                j=-11
                        p=x*y           y=*r0++j
                a0=a0+p                 x=*r1++
                j=4
                        p=x*y           y=*r0++j
                a0=a0+p                 x=*r1++
                                        *r2++=a01
              }
                j=3
                                        *r1++j
                a0=r1
                rb=a0
                                        *r1++j
                a0=r1
                re=a0
                                        *r1++
        if c1lt goto loop
stop:   goto stop
```


www.DataSheet4U.com

B.4 FIND MAXIMUM VECTOR ELEMENT

```
/*              The following segment of code will find the     */
/*      component of an input vector with maximum magnitude.    */
/*      Both the magnitude and index (offset) of the component  */
/*      are retained upon completion.                           */
/*              This coding example demonstrates the use of     */
/*      conditional ALU monadic instructions and event counting */
/*      using the c2 register.                                  */
/*                                                              */

.ram
vector:         32*int
.endram

/*              Read in the vector to be searched via PIO.      */
/*      (It is assumed that the vector will be reused           */
/*      after the determination of its maximum component.       */
/*      Otherwise, a more efficient loop might be written       */
/*      which determines the max component as the values        */
/*      are read.)                                              */

setup:          auc=0x00
                r1=vector
                a0=pdx1
        do 32 {
                a0=pdx1
                *r1++=a0
        }

/*                      Begin search.                           */
/*      Upon completion, the magnitude of the component         */
/*      with maximum value is stored in a0, and the index       */
/*      of this component (its offset from the base address     */
/*      'vector') is stored in c2.                              */
/*              As written, if more than one component of the   */
/*      vector share the maximum value, then the first of these */
/*      two is retained. The routine can easily be modified to  */
/*      retain the last of these components by replacing the    */
/*      test condition with 'le'.                               */

findmaxM:       r1=vector
                c1=0
                c2=0
                                        y=*r1
                                        x=*r1++
                            p=x*y       y=*r1
                a0=p                    x=*r1++
        do 31 {
                            p=x*y       y=*r1
                a1=a0-p                 x=*r1++
                ifc mi  a0=p
              }
stop:   goto stop
```


www.DataSheet4U.com