TEXAS INSTRUMENTS

Application Report
SPRA669 - July 2000

TMS320C54x Digital Filters

C5000 Applications Team

Digital Signal Processing Solutions

ABSTRACT

Certain features of the TMS3320C54x™ architecture and instruction set facilitate the solution of numerically intensive problems. Some examples include filtering, encoding techniques in telecommunication applications, and speech recognition. This application report discusses digital filters that use both fixed and adaptive coefficients and fast Fourier transforms.

Contents

1 Finite Impulse Response (FIR) Filters . . . 2
2 Infinite Impulse Response (IIR) Filters . . . 7
3 Adaptive Filtering . . . 10
4 Fast Fourier Transforms (FFTs) . . . 15
4.1 Memory Allocation for Real FFT Example . . . 16
4.2 Real FFT Example . . . 18
4.2.1 Phase 1: Packing and Bit-Reversal of Input . . . 18
4.2.2 Phase 2: N-Point Complex FFT . . . 20
4.2.3 Phase 3: Separation of Odd and Even Parts . . . 22
4.2.4 Phase 4: Generation of Final Output . . . 24
Appendix A Main.inc . . . 26
Appendix B FFT Example Code . . . 27
B.1 256-Point Real FFT Initialization . . . 27
B.2 Bit Reversal Routine . . . 29
B.3 256-Point Real FFT Routine . . . 32
B.4 Unpack 256-Point Real FFT Output . . . 37

List of Figures

Figure 1. Data Memory Organization in an FIR Filter . . . 2
Figure 2. Block Diagram of an Nth-Order Symmetric FIR Filter . . . 4
Figure 3. Input Sequence Storage . . . 5
Figure 4. Nth-Order Direct-Form Type II IIR Filter . . . 7
Figure 5. Biquad IIR Filter . . . 8
Figure 6. Adaptive FIR Filter Implemented Using the Least-Mean-Squares (LMS) Algorithm . . . 10
Figure 7. System Identification Using Adaptive Filter . . . 11
Figure 8. Memory Allocation for Real FFT Example . . . 16
Figure 9. Data Processing Buffer . . . 17
Figure 10. Phase 1 Data Memory . . . 19
Figure 11. Phase 2 Data Memory . . . 21

TMS320C54x is a trademark of Texs Instruments.

Figure 12. Phase 3 Data Memory 23
Figure 13. Phase 4 Data Memory 25

List of Examples

Example 1. FIR Implementation Using Circular Addressing Mode With a Multiply and Accumulate (MAC) Instruction 3
Example 2. Symmetric FIR Implementation Using FIRS Instruction 6
Example 3. Two-Biquad Implementation of an IIR Filter 8
Example 4. System Identification Using Adaptive Filtering Techniques 11

1 Finite Impulse Response (FIR) Filters

Digital filters are a common requirement for digital signal processing systems. There are two types of digital filters: finite impulse response (FIR) and infinite impulse response (IIR). Each of these can have either fixed or adaptive coefficients.

If an FIR filter has an impulse response, h(0), h(1), ..., h(N–1), and x(n) represents the input of the filter at time n, the output y(n] at time n is given by the following equation:

$$y(n) = h(0)x(n) + h(1)x(n-1) + h(2)x(n-2) + ...+ h(N-1)x[n-(N-1)]$$

Figure 1 illustrates a method using circular addressing to implement an FIR filter. To set up circular addressing, initialize BK to block length N. The locations for d_data_buffer, and impulse responses, COFF_FIR, must start from memory locations whose addresses are multiples of the smallest power of 2 that is greater than N. For instance, if N = 11, the first address for d_data_buffer must be a multiple of 16. Thus, the lowest four bits of the beginning address must be 0.

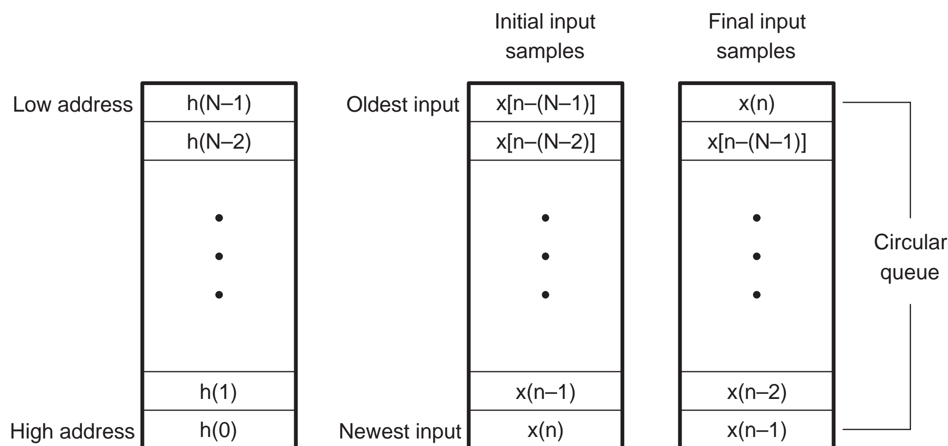

Figure 1. Data Memory Organization in an FIR Filter

In Example 1, N is 16 and the circular buffer starts at an address whose four LSBs are 0.

Example 1. FIR Implementation Using Circular Addressing Mode With a Multiply and Accumulate (MAC) Instruction

```
; TEXAS INSTRUMENTS INCORPORATED
   .mmregs
   .include ”main.inc”
; the 16 tap FIR coefficients
COFF_FIR_START .sect ”coff_fir”  ; filter coefficients
   .word 6Fh
   .word 0F3h
   .word 269h
   .word 50Dh
   .word 8A9h
   .word 0C99h
   .word 0FF8h
   .word 11EBh
   .word 11EBh
   .word 0FF8h
   .word 0C99h
   .word 8A9h
   .word 50Dh
   .word 269h
   .word 0F3h
   .word 6Fh
COFF_FIR_END
FIR_DP            .usect   ”fir_vars”,0
d_filin         .usect   ”fir_vars”,1
d_filout        .usect   ”fir_vars”,1
fir_coff_table .usect   ”fir_coff”,20
d_data_buffer     .usect   ”fir_bfr”,40   ; buffer size for the filter
               .def     fir_init    ; initialize FIR filter
               .def     fir_task    ; perform FIR filtering
;-----------------------------------------------------------------------------
; Functional Description
;  This routine initializes circular buffers both for data and coeffs.
;-----------------------------------------------------------------------------
         .asg     AR0, FIR_INDEX_P
         .asg     AR4,FIR_DATA_P
         .asg     AR5,FIR_COFF_P
         .sect    ”fir_prog”
fir_init:
         STM         #fir_coff_table,FIR_COFF_P
         RPT         #K_FIR_BFFR-1                    ; move FIR coeffs from program
         MVPD     #COFF_FIR_START,*FIR_COFF_P+      ; to data
         STM         #K_FIR_INDEX,FIR_INDEX_P
         STM         #d_data_buffer,FIR_DATA_P         ; load cir_bfr address for the
                                                ; recent samples
         RPTZ     A,#K_FIR_BFFR
         STL         A,*FIR_DATA_P+                ; reset the buffer
         STM         #(d_data_buffer+K_FIR_BFFR-1), FIR_DATA_P
         RETD
         STM         #fir_coff_table, FIR_COFF_P
;---------------------------------------------------------------------------
;  Functional Description
;
;  This subroutine performs FIR filtering using MAC instruction.
;  accumulator A (filter output) = h(n)*x(n-i) for i = 0,1...15
;-----------------------------------------------------------------------------
      .asg     AR6,INBUF_P
      .asg     AR7,OUTBUF_P
      .asg     AR4,FIR_DATA_P
      .asg     AR5,FIR_COFF_P
      .sect    ”fir_prog”
```

Example 1. FIR Implementation Using Circular Addressing Mode With a Multiply and Accumulate (MAC) Instruction (Continued)

```
fir_task:
;      LD       #FIR_DP,DP
       STM         #K_FRAME_SIZE-1,BRC           ; Repeat 256 times
       RPTBD    fir_filter_loop-1
       STM         #K_FIR_BFFR,BK             ; FIR circular bffr size
       LD       *INBUF_P+, A                  ; load the input value
fir_filter:
       STL         A,*FIR_DATA_P+%               ; replace oldest sample with newest
                                           ; sample
       RPTZ     A,(K_FIR_BFFR-1)
       MAC         *FIR_DATA_P+0%,*FIR_COFF_P+0%,A  ; filtering
       STH         A, *OUTBUF_P+                    ; replace the oldest bffr value
fir_filter_loop
    RET
```

In a second method, two features of the ’54x device facilitate implementation of the FIR filters: circular addressing and the FIRS instruction. The FIR filter shown in Figure 2, with symmetric impulse response about the center tap, is widely used in digital signal processing applications because of its linear phase response. In applications such as speech processing, linear phase response is required to avoid phase distortion, which degrades the quality of the signal waveforms. The output of the filter for length N is given by:

$$y(n) = \sum_{k=0}^{N/2-1} h(k)[x(n-k) + x(n-(N-1+k))] \qquad n = 0, 1, 2$$

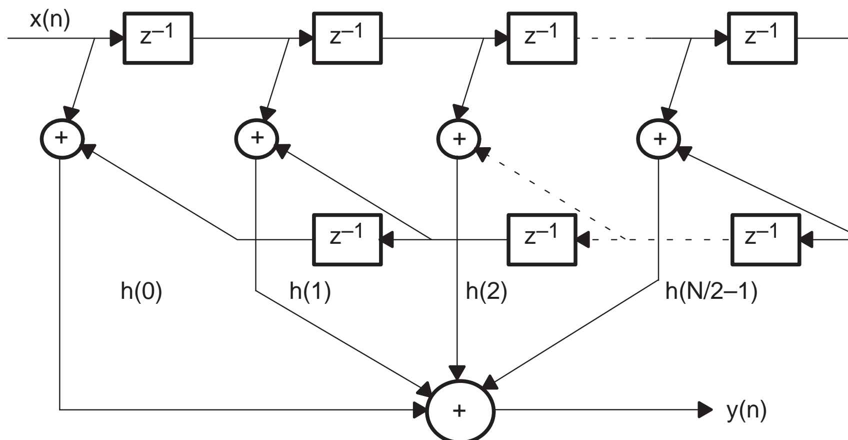

Figure 2. Block Diagram of an Nth-Order Symmetric FIR Filter

Figure 3 shows the storage diagram of the input sequence for two circular buffers. To build the buffers, the value of N/2 is loaded into a circular buffer size register. AR4 is set to point to the top of buffer 1 and AR5 points to the bottom of buffer 2. The data at the top of buffer 1 is moved to the bottom of buffer 2 for the delayed operation before storing new sample data in buffer 1. The processor then performs the adds and multiplies h(0){x(0)+x(–N+1)}. After each iteration of the filtering routine, AR4 points to the next time slot window for the data move and AR5 points to the next input sample. For the next iteration of the filtering routine, AR4 points to address 1 and AR5 points to address N/2–2.

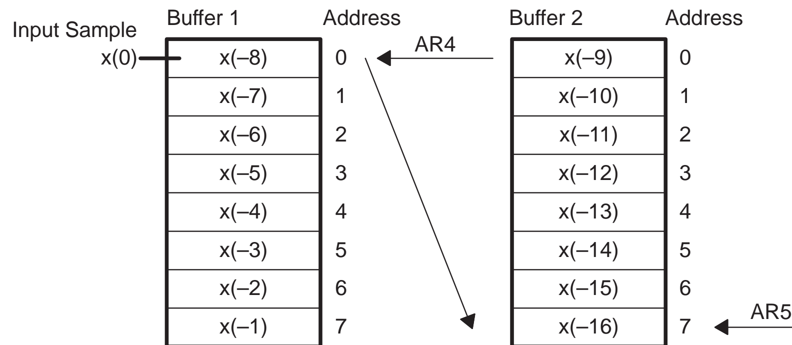

Figure 3. Input Sequence Storage

Example 2 shows how to implement a symmetric FIR filter on the ’54x. It uses the symmetrical finite impulse response (FIRS) instruction and the repeat next instruction and clear accumulator (RPTZ) instruction together. FIRS can add two data values (input sequences stored in data memory) in parallel with multiplication of the previous addition result using an impulse response stored in program memory. FIRS becomes a single-cycle instruction when used with the single-repeat instruction. To perform the delayed operation in this storage scheme, two circular buffers are used for the input sequence.

Example 2. Symmetric FIR Implementation Using FIRS Instruction

```
; TEXAS INSTRUMENTS INCORPORATED
   .mmregs
   .include    "main.inc"
FIR_COFF .sect    "sym_fir"   ; filter coefficients
   .word 6Fh
   .word 0F3h
   .word 269h
   .word 50Dh
   .word 8A9h
   .word 0C99h
   .word 0FF8h
   .word 11EBh
d_datax_buffer .usect  "cir_bfr",20
d_datay_buffer .usect  "cir_bfr1",20
   .def  sym_fir_init      ; initialize symmetric FIR
   .def  sym_fir_task
;-----------------------------------------------------------------------------
;  Functional Description
;  This routine initializes circular buffers both for data and coeffs.
;-----------------------------------------------------------------------------
   .asg  AR0,SYMFIR_INDEX_P
   .asg  AR4,SYMFIR_DATX_P
   .asg  AR5,SYMFIR_DATY_P
   .sect "sym_fir"
sym_fir_init:
   STM   #d_datax_buffer,SYMFIR_DATX_P ; load cir_bfr address
            ; for the 8 most
            ; recent samples  STM   #d_datay_buffer+K_FIR_BFFR/2–1,SYMFIR_DATY_P
            ; load cir_bfr1 address
            ; for the 8 old samples
   STM   #K_neg1,SYMFIR_INDEX_P     ; index offset –
            ; whenever the pointer
            ; hits the top of the bffr,
            ; it automatically hits
            ; bottom address of
            ; buffer and decrements
            ; the counter
   RPTZ  A,#K_FIR_BFFR
   STL   A,* SYMFIR_DATX_P+
   STM   #d_datax_buffer, SYMFIR_DATX_P
   RPTZ  A,#K_FIR_BFFR
   STL   A,* SYMFIR_DATY_P-
   RETD
   STM   #d_datay_buffer+K_FIR_BFFR/2–1, SYMFIR_DATY_P
;---------------------------------------------------------------------------------
;  Functional Description
;This program uses the FIRS instruction to implement symmetric FIR filter
;Circular addressing is used for data buffers. The input scheme for the data;
;samples is divided into two circular buffers. The first buffer contains
;samples from X(–N/2) to X(–1) and the second buffer contains samples from
;X(–N) to X(–N/2–1).
;---------------------------------------------------------------------------------
   .asg  AR6,INBUF_P
   .asg  AR7,OUTBUF_P
   .asg  AR4,SYMFIR_DATX_P
   .asg  AR5,SYMFIR_DATY_P
   .sect "sym_fir"
```

Example 2. Symmetric FIR Implementation Using FIRS Instruction (Continued)

```
sym_fir_task:
   STM   #K_FRAME_SIZE-1,BRC
   RPTBD sym_fir_filter_loop-1
   STM   #K_FIR_BFFR/2,BK
   LD *INBUF_P+, B
symmetric_fir:
   MVDD  *SYMFIR_DATX_P,*SYMFIR_DATY_P+0%    ; move X(-N/2) to X(-N)
   STL   B,*SYMFIR_DATX_P      ; replace oldest sample with newest
                            ; sample
   ADD   *SYMFIR_DATX_P+0%,*SYMFIR_DATY_P+0%,A ; add X(0)+X(-N/2-1)
   RPTZ  B,#(K_FIR_BFFR/2-1)
   FIRS   *SYMFIR_DATX_P+0%,*SYMFIR_DATY_P+0%,FIR_COFF
   MAR   *+SYMFIR_DATX_P(2)%  ; to load the next newest sample
   MAR   *SYMFIR_DATY_P+%     ; position for the X(-N/2) sample
   STH   B, *OUTBUF_P+
sym_fir_filter_loop
   RET
   .end
```

2 Infinite Impulse Response (IIR) Filters

IIR filters are widely used in digital signal processing applications. The transfer function of an IIR filter is given by:

$$H(z) = \frac{b_0 + b_1 z^{-1} + \ldots + b_M z^{-M}}{1 + a_1 z^{-1} + \ldots + a_N z^{-N}} = \frac{Y(z)}{x(z)}$$

The transfer function has both poles and zeros. Its output depends on both input and past output. IIR filters need less computation than FIR filters. However, IIR filters have stability problems. The coefficients are very sensitive to quantization error. Figure 4 shows a typical diagram of an IIR filter.

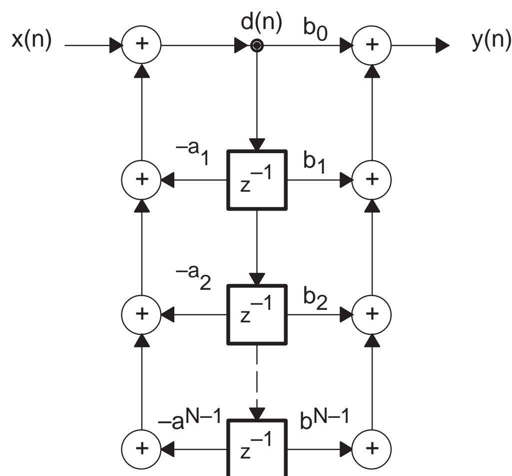

Figure 4. Nth-Order Direct-Form Type II IIR Filter

Most often, IIR filters are implemented as a cascade of second-order sections, called biquads. The block diagram is shown in Figure 5.

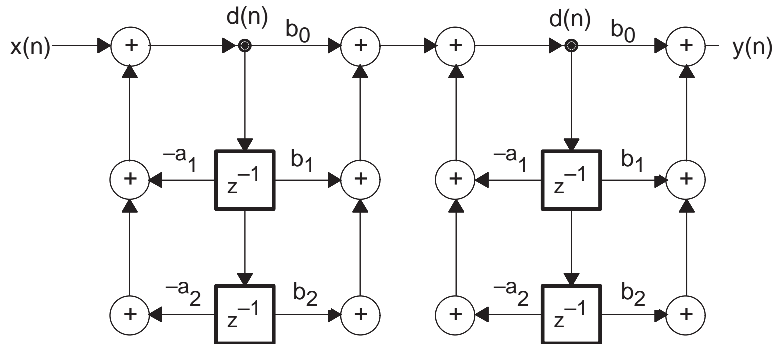

Figure 5. Biquad IIR Filter

Example 3. Two-Biquad Implementation of an IIR Filter

```
; TEXAS INSTRUMENTS INCORPORATED
         .mmregs
         .include       ”main.inc”
         .sect       ”iir_coff”
iir_table_start
*
*  second–order section # 01
*
         .word       –26778           ;A2
         .word       29529         ;A1/2
         .word       19381         ;B2
         .word       –23184           ;B1
         .word       19381         ;B0
*
*  second–order section # 02
*
         .word       –30497           ;A2
         .word       31131         ;A1/2
         .word       11363         ;B2
         .word       –20735           ;B1
         .word       11363         ;B0
iir_table_end
iir_coff_table       .usect   ”coff_iir”,16
IIR_DP                  .usect   ”iir_vars”,0
d_iir_d              .usect   ”iir_vars”,3*2
d_iir_y              .usect   ”iir_vars”,1
         .def        iir_init
         .def        iir_task
;–––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––
;  Functional Description
;  This routine initializes buffers both for data and coeffs.
;––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––
         .asg        AR5,IIR_DATA_P       ; data samples pointer
         .asg        AR4,IIR_COFF_P    ; IIR filter coffs pointer
         .sect       ”iir
iir_init:
   STM   #iir_coff_table,IIR_COFF_P
      RPT      #K_IIR_SIZE–1                 ; move IIR coeffs from program
      MVPD  #iir_table_start,*IIR_COFF_P+ ; to data
;     LD    #IIR_DP,DP
```

Example 3. Two-Biquad Implementation of an IIR Filter (Continued)

```
      STM      #d_iir_d,IIR_DATA_P           ;AR5:d(n),d(n–1),d(n–2)
      RPTZ  A,#5                        ;initial d(n),d(n–1),d(n–2)=0
      STL      A,*IIR_DATA_P+
      RET
;  Functional Description
;
;     This subroutine performs IIR filtering using biquad sections
;     IIR Low pass filter design
;     Filter type : Elliptic Filter
;     Filter order : 4 order (cascade: 2nd order + 2nd order)
;     cut freq. of pass band : 200 Hz
;     cut freq. of stop band : 500
;                                  B0
;     ... –––––> + ––––––> d(n) –––– x –> + –––––.... ––>
;     |          |                   |
;     |    A1    |         B1        |
;     + <– x –– d(n–1) –– x –> +     |
;     |          |                   |
;     |    A2    |         B2        |
;     + <– x –– d(n–2) –– x –> +
;
;         second order IIR
;–––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––
   .asg  AR5,IIR_DATA_P              ; data samples pointer
   .asg  AR4,IIR_COFF_P              ; IIR filter coffs pointer
   .asg  AR6,INBUF_P
   .asg  AR7,OUTBUF_P
   .asg  AR1,IIR_INDEX_P
   .sect ”iir”
iir_task:
   STM      #K_FRAME_SIZE–1,BRC           ; Perform filtering for 256 samples
   RPTB  iir_filter_loop–1
   LD    *INBUF_P+,8,A                  ; load the input value
iir_filter:
   STM      #d_iir_d+5,IIR_DATA_P         ;AR5:d(n),d(n–1),d(n–2)
   MVPD  #iir_table_start,*IIR_COFF_P+   ; to data
   STM      #iir_coff_table,IIR_COFF_P    ;AR4:coeff of IIR filter A2,A1,B2,B1,B0
   STM      #K_BIQUAD–1,IIR_INDEX_P
feedback_path:
   MAC      *IIR_COFF_P+,*IIR_DATA_P–,A   ;input+d(n–2)*A2
   MAC      *IIR_COFF_P,*IIR_DATA_P,A     ;input+d(n–2)*A2+d(n–1)*A1/2
   MAC      *IIR_COFF_P+,*IIR_DATA_P–,A   ; A = A+d(n–1)*A1/2
   STH      A,*IIR_DATA_P+              ;d(n) = input+d(n–2)*A2+d(n–1)*A1
   MAR      *IIR_DATA_P+
* Forward path
   MPY      *IIR_COFF_P+,*IIR_DATA_P–,A   ;d(n–2)*B2
   MAC      *IIR_COFF_P+,*IIR_DATA_P,A    ;d(n–2)*B2+d(n–1)*B1
   DELAY *IIR_DATA_P–                   ;d(n–2)=d(n–1)
eloop:
   BANZD feedback_path, *IIR_INDEX_P–
   MAC      *IIR_COFF_P+,*IIR_DATA_P,A    ;d(n–2)*B2+d(n–1)*B1+d(n)*B0
   DELAY *IIR_DATA_P–                   ;d(n–1)=d(n)
   STH      A,d_iir_y                     ;output=d(n–2)*B2+d(n–1)*B1+d(n)*B0
   LD    d_iir_y,2,A                 ; scale the output
   STL      A, *OUTBUF_P+                 ; replace the oldest bffr value
iir_filter_loop
   RET
   .end
```

3 Adaptive Filtering

Some applications for adaptive FIR and IIR filtering include echo and acoustic noise cancellation. In these applications, an adaptive filter tracks changing conditions in the environment. Although in theory, both FIR and IIR structures can be used as adaptive filters, stability problems and the local optimum points of IIR filters makes them less attractive for this use. FIR filters are used for all practical applications. The LMS, ST||MPY, and RPTBD instructions on the ’54x can reduce the execution time of code for adaptive filtering. The block diagram of an adaptive FIR filter is shown in Figure 6. The Adaptive filtering routine is shown in Example 4, page 11.

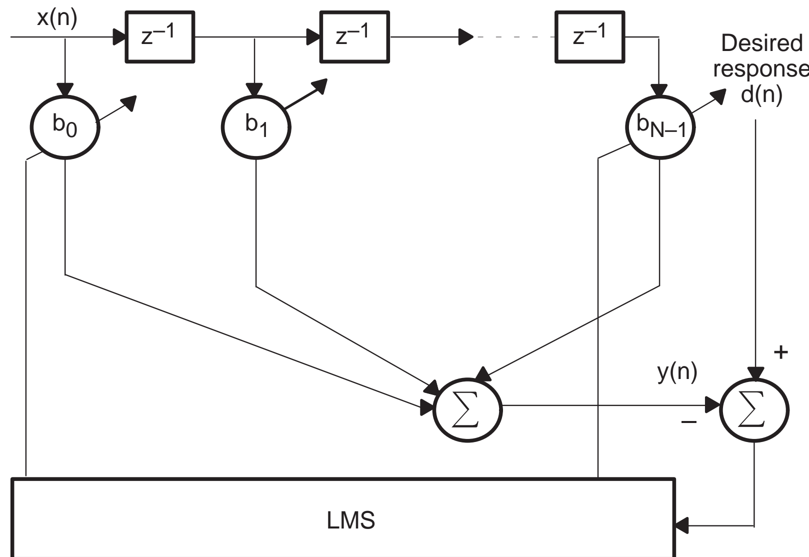

Figure 6. Adaptive FIR Filter Implemented Using the Least-Mean-Squares (LMS) Algorithm

On the ’54x, one coefficient can be adapted by using the least-mean-squares (LMS) algorithm, which is given by

$$b_k(i+1) = b_k(i) + 2\beta e(i) x(i-k),$$

where:

$e(i) = d(i) - y(i)$

The output of the adaptive filter is given by

$$y(i) = \sum_{k=0}^{N-1} b_k \, x(i-k)$$

The LMS instruction can perform a MAC instruction and an addition with rounding in parallel. The LMS algorithm calculates the filter output and updates each coefficient in the filter in parallel by using the LMS instruction, along with the ST||MPY and RPTBD instructions. For each coefficient in the filter at a given instant, $2\beta e(i)$ is a constant. This factor can be computed once and stored in the temporary register, T, to use in each of the updates. The ST||MPY instruction multiplies a data sample by this factor, then the LMS instruction updates a coefficient in the filter and accumulates the filtered output. Since the factor is stored in T, the adaptive filtering in a time-slot window is performed in N cycles.

An adaptive filter can be used in modeling to imitate the behavior of a physical dynamic system. Figure 7 shows a block diagram of system identification, where the adaptive filter adjusts itself to cause its output to match that of the unknown system. H(z) is the impulse response of an unknown system; for example, the echo response in the case of an acoustic echo cancellation system. The signal x(n) trains the system. The size of the adaptive filter is chosen to be N, where N is the number of taps (coefficients) of the known system, W(z).

Two circular buffers store the input sequence. AR3 points to the first buffer, AR2 points to the coefficients of H(z), AR4 points to the coefficients of W(z), and AR5 points to the second buffer. The newest sample is stored in a memory location that is input to the adaptive filter. The input sample is subtracted from the output of the adaptive filter to obtain the error data for the LMS algorithm. In this example, the adaptive filter output is computed for the newest sample and the filter coefficients are updated using the previous sample. Thus, there is an inherent delay between the update of the coefficient and the output of the adaptive filter.

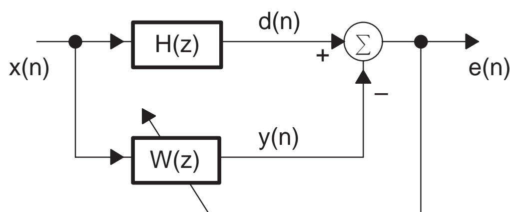

Figure 7. System Identification Using Adaptive Filter

Example 4. System Identification Using Adaptive Filtering Techniques

```
; TEXAS INSTRUMENTS INCORPORATED
         .mmregs
         .include ”main.inc”
scoff    .sect    ”coeffh”
         .include ”impulse.h”
         ADAPT_DP .usect  ”adpt_var”,0
d_primary         .usect  ”adpt_var”,1
d_output          .usect  ”adpt_var”,1
d_error           .usect  ”adpt_var”,1
d_mu              .usect  ”adpt_var”,1
d_mu_e            .usect  ”adpt_var”,1
d_new_x           .usect  ”adpt_var”,1
d_adapt_count        .usect  ”adpt_var”,1
hcoff             .usect ”bufferh”, H_FILT_SIZE     ; H(z) coeffs
wcoff             .usect ”bufferw”, ADPT_FILT_SIZE   W(z) coeffs
xh                .usect ”bufferx”, H_FILT_SIZE        ; input data to H(z)
xw                .usect ”bufferp”, ADPT_FILT_SIZE  ; input data–adaptive filter
            .def        adapt_init,adapt_task
;–––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––
;
;  Functional Description
;
;  This subroutine moves filter coefficients from program to data space.
;  Initializes the adaptive coefficients, buffers, vars, and sets the circular
;  buffer address for processing.
;–––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––
   .asg     AR0,INDEX_P
   .asg     AR1,INIT_P                          ; initialize buffer pointer
   .asg     AR3,XH_DATA_P                          ; data coeff buffer pointer
   .asg     AR5,XW_DATA_P                          ; data coeff buffer pointer
                                                ; for cal.y output
   .sect ”filter”
```

Example 4. System Identification Using Adaptive Filtering Techniques (Continued)

```
adapt_init:
;  initialize input data location, input to hybrid, with zero.
   STM      #xh,INIT_P
   RPTZ  A,#H_FILT_SIZE–1
   STL      A,*INIT_P+
;  initialize input data location, input to adaptive filter, with Zero.
   STM      #xw,INIT_P
   RPTZ  A,#ADPT_FILT_SIZE–1
   STL      A,*INIT_P+
;  initialize adaptive coefficient with Zero.
   STM      #wcoff,INIT_P
   RPTZ  A,#ADPT_FILT_SIZE–1
   STL      A,*INIT_P+
;  initialize temporary storage locations with zero
   STM      #d_primary,INIT_P
   RPTZ  A,#6
   STL      A,*INIT_P+
;  copy system coefficient into RAM location, Reverse order STM   #hcoff,INIT_P
   RPT      #H_FILT_SIZE–1
   MVPD  #scoff,*INIT_P+
;  LD #ADAPT_DP,DP                   ;set DP now and not worry about it
   ST #K_mu,d_mu
   STM   #1,INDEX_P                  ; increment value to be used by
                                  ; dual address
;  associate auxiliary registers for circular computation
   STM   #xh+H_FILT_SIZE–1,XH_DATA_P   ; last input of hybrid buffer
   RETD
   STM   #xw+ADPT_FILT_SIZE–1,XW_DATA_P ;last element of input buffer
;–––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––
;
;  Functional Description
;
;  This subroutine performs the adaptive filtering. The newest sample is stored
;  in a separate location since filtering and adaptation are performed at the
;  same time. Otherwise the oldest sample is over written before updating
;  the w(N–1) coefficient.
;
;  d_primary = xh *hcoff
;  d_output = xw *wcoff
;  LMS algorithm:
;  w(i+1) = w(i)+d*mu_error*xw(n–i) for i = 0,1,...127 and n = 0,1,.....
;–––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––
   .asg  AR2,H_COFF_P                ; H(Z) coeff buffer pointer
   .asg  AR3,XH_DATA_P               ; data coeff buffer pointer
   .asg  AR6,INBUF_P              ; input buffer address pointer
   .asg  AR7,OUTBUF_P                ; output buffer address pointer
                                  ; for cal. primary input
   .asg  AR4,W_COFF_P                ; W(z) coeff buffer pointer
   .asg  AR5,XW_DATA_P               ; data coeff buffer pointer
   .sect ”filter”
adapt_task:
   STM      #H_FILT_SIZE,BK             ; first circular buffer size
   STM      #hcoff,H_COFF_P             ; H_COFF_P ––> last of sys coeff
   ADDM  #1,d_adapt_count
   LD    *INBUF_P+, A                ; load the input sample
   STM      #wcoff,W_COFF_P             ; reset coeff buffer
   STL      A,d_new_x                   ; read in new data
   LD    d_new_x,A                   ;
   STL      A,*XH_DATA_P+0%             ; store in the buffer
   RPTZ  A,#H_FILT_SIZE–1            ; Repeat 128 times
   MAC      *H_COFF_P+0%,*XH_DATA_P+0%,A  ; mult & acc:a = a + (h * x)
```

```
   STH      A,d_primary              ; primary signal
;  start simultaneous filtering and updating the adaptive filter here.
   LD    d_mu_e,T                 ; T = step_size*error
   SUB      B,B                      ; zero acc B
   STM      #(ADPT_FILT_SIZE-2),BRC    ; set block repeat counter
   RPTBD lms_end-1
   MPY      *XW_DATA_P+0%, A           ; error * oldest sample
   LMS      *W_COFF_P, *XW_DATA_P      ; B = filtered output (y)
                                  ; Update filter coeff
   ST    A, *W_COFF_P+              ; save updated filter coeff
   || MPY *XW_DATA_P+0%,A           ; error *x[n-(N-1)]
   LMS   *W_COFF_P, *XW_DATA_P          ; B = accum filtered output y
                                  ; Update filter coeff
lms_end
   STH      A, *W_COFF_P                ; final coeff
   MPY      *XW_DATA_P,A              ; x(0)*h(0)
   MVKD  #d_new_x,*XW_DATA_P        ; store the newest sample
   LMS      *W_COFF_P,*XW_DATA_P+0%
   STH      B, d_output              ; store the filtered output
   LD    d_primary,A
   SUB      d_output,A
   STL      A, d_error               ; store the residual error signal
   LD    d_mu,T
   MPY      d_error,A                   ; A=u*e
   STH      A,d_mu_e                 ; save the error *step_size
   LD    d_error,A                ; residual error signal
   STL      A, *OUTBUF_P+
   LD    #K_FRAME_SIZE,A            ; check if a frame of samples
   SUB      d_adapt_count,A             ; have been processed
   BC    adapt_task,AGT
   RETD
   ST    #K_0,d_adapt_count          ; restore the count
   .end
* This is an input file used by the adaptive filter program.
* The transfer function is the system to be identified by the adaptive filter
   .word 0FFFDh
   .word 24h
   .word 6h
   .word 0FFFDh
   .word 3h
   .word 3h
   .word 0FFE9h
   .word 7h
   .word 12h
   .word 1Ch
   .word 0FFF3h
   .word 0FFE8h   .word 0Ch
   .word 3h
   .word 1Eh
   .word 1Ah
   .word 22h
   .word 0FFF5h
   .word 0FFE5h
   .word 0FFF1h
   .word 0FFC5h
   .word 0Ch
   .word 0FFE8h
   .word 37h
   .word 0FFE4h
   .word 0FFCAh
   .word 1Ch
   .word 0FFFDh
```

Example 4. System Identification Using Adaptive Filtering Techniques (Continued)

```
.word 21h
.word 0FFF7h
.word 2Eh
.word 28h
.word 0FFC6h
.word 53h
.word 0FFB0h
.word 55h
.word 0FF36h
.word 5h
.word 0FFCFh
.word 0FF99h
.word 64h
.word 41h
.word 0FFF1h
.word 0FFDFh
.word 0D1h
.word 6Ch
.word 57h
.word 36h
.word 0A0h
.word 0FEE3h
.word 6h
.word 0FEC5h
.word 0ABh
.word 185h
.word 0FFF6h
.word 93h
.word 1Fh
.word 10Eh
.word 59h
.word 0FEF0h
.word 96h
.word 0FFBFh
.word 0FF47h
.word 0FF76h
.word 0FF0Bh
.word 0FFAFh
.word 14Bh
.word 0FF3Bh
.word 132h
.word 289h
.word 8Dh
.word 0FE1Dh
.word 0FE1Bh
.word 0D4h
.word 0FF69h
.word 14Fh
.word 2AAh
.word 0FD43h
.word 0F98Fh
.word 451h
.word 13Ch
.word 0FEF7h
.word 0FE36h
.word 80h
.word 0FFBBh
.word 0FC8Eh
.word 10Eh
.word 37Dh
```

Example 4. System Identification Using Adaptive Filtering Techniques (Continued)

```
.word 6FAh
.word 1h
.word 0FD89h
.word 198h
.word 0FE4Ch
.word 0FE78h
.word 0F215h
.word 479h
.word 749h
.word 289h
.word 0F667h
.word 304h
.word 5F8h
.word 34Fh
.word 47Bh
.word 0FF7Fh
.word 85Bh
.word 0F837h
.word 0F77Eh
.word 0FF80h
.word 0B9Bh
.word 0F03Ah
.word 0EE66h
.word 0FE28h
.word 0FAD0h
.word 8C3h
.word 0F5D6h
.word 14DCh
.word 0F3A7h
.word 0E542h
.word 10F2h
.word 566h
.word 26AAh
.word 15Ah
.word 2853h
.word 0EE95h
.word 93Dh
.word 20Dh
.word 1230h
.word 238Ah
```

4 Fast Fourier Transforms (FFTs)

FFTs are an efficient class of algorithms for the digital computation of the N-point discrete Fourier transform (DFT). In general, their input sequences are assumed to be complex. When input is purely real, their symmetric properties compute the DFT very efficiently.

One such optimized real FFT algorithm is the packing algorithm. The original 2N-point real input sequence is packed into an N-point complex sequence. Next, an N-point FFT is performed on the complex sequence. Finally the resulting N-point complex output is unpacked into the 2N-point complex sequence, which corresponds to the DFT of the original 2N-point real input.

Using this strategy, the FFT size can be reduced by half, at the FFT cost function of O(N) operations to pack the input and unpack the output. Thus, the real FFT algorithm computes the DFT of a real input sequence almost twice as fast as the general FFT algorithm. The following subsections show how to perform a 16-point real FFT (2N = 16).

4.1 Memory Allocation for Real FFT Example

The following tables give the organization of values in data memory from the beginning of the real FFT algorithm to its end. Initially, the original 2N-point real input sequence, a(n), is stored in the lower half of the 4N-word data processing buffer, as shown in Figure 9. Figure 8 shows memory allocation for a real FFT example.

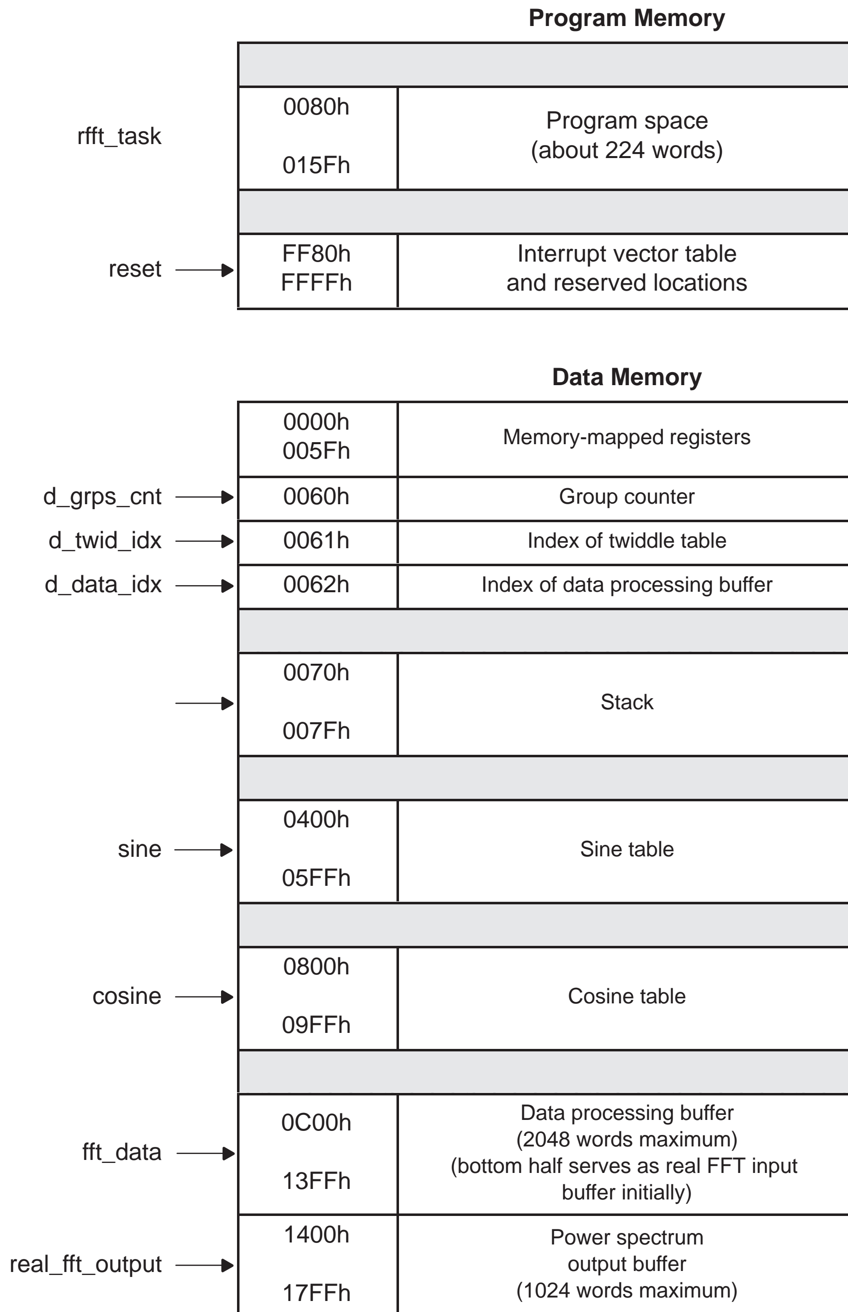

Figure 8. Memory Allocation for Real FFT Example

Address	Data Memory
0C00h	
0C01h	
0C02h	
0C03h	
0C04h	
0C05h	
0C06h	
0C07h	
0C08h	
0C09h	
0C0Ah	
0C0Bh	
0C0Ch	
0C0Dh	
0C0Eh	
0C0Fh	
0C10h	a(0)
0C11h	a(1)
0C12h	a(2)
0C13h	a(3)
0C14h	a(4)
0C15h	a(5)
0C16h	a(6)
0C17h	a(7)
0C18h	a(8)
0C19h	a(9)
0C1Ah	a(10)
0C1Bh	a(11)
0C1Ch	a(12)
0C1Dh	a(13)
0C1Eh	a(14)
0C1Fh	a(15)

Figure 9. Data Processing Buffer

4.2 Real FFT Example

The ’54x real FFT algorithm is a radix-2, in-place DFT algorithm. It is shown in the following subsections in four phases :

1. Packing and bit-reversal of input
2. N-point complex FFT
3. Separation of odd and even parts
4. Generation of final output

Initially, any real input sequences of 16 to 1024 points can be used by simply modifying the constants K_FFT_SIZE and K_LOGN appropriately, defined in file main.inc. (The real input size is described as 2N and the FFT size in phase two as N.) For a 256-point real input, for example, K_FFT_SIZE must be set to 128, not 256, and K_LOGN must be 7, not 8. Input data is assumed to be in Q15 format.

4.2.1 Phase 1: Packing and Bit-Reversal of Input

In phase 1, the input is bit-reversed so that the output at the end of the entire algorithm is in natural order. First, the original 2N-point real input sequence is copied into contiguous sections of memory labeled real_fft_input and interpreted as an N-point complex sequence, d[n]. The even-indexed real inputs form the real part of d[n] and the odd-indexed real inputs form the imaginary part. This process is called packing. (n is a variable indicating time and can vary from 0 to infinity, while N is a constant). Next, this complex sequence is bit-reversed and stored into the data processing buffer, labeled fft_data.

1. Arrange the real input sequence, a(n) for n = 0, 1, 2, ... n – 1, as shown in Figure 9. Divide a(n) into two sequences as shown in Figure 10. The first is the original input sequence from 0C10h to 0C1Fh. The other is a packed sequence:

   for n = 0, 1, 2, ... , N – 1

2. Form the complex FFT input, d(n), by using r(n) for the real part and i(n) for the imaginary part:

   $d(n) = r(n) + j\, i(n)$

3. Store d(n) in the upper half of the data processing buffer in bit-reversed order as shown in Figure 10.

Address	Contents
0C00h	r(0) = a(0)
0C01h	i(0) = a(1)
0C02h	r(4) = a(8)
0C03h	i(4) = a(9)
0C04h	r(2) = a(4)
0C05h	i(2) = a(5)
0C06h	r(6) = a(12)
0C07h	i(6) = a(13)
0C08h	r(1) = a(2)
0C09h	i(1) = a(3)
0C0Ah	r(5) = a(10)
0C0Bh	i(5) = a(11)
0C0Ch	r(3) = a(6)
0C0Dh	i(3) = a(7)
0C0Eh	r(7) = a(14)
0C0Fh	i(7) = a(15)
0C10h	a(0)
0C11h	a(1)
0C12h	a(2)
0C13h	a(3)
0C14h	a(4)
0C15h	a(5)
0C16h	a(6)
0C17h	a(7)
0C18h	a(8)
0C19h	a(9)
0C1Ah	a(10)
0C1Bh	a(11)
0C1Ch	a(12)
0C1Dh	a(13)
0C1Eh	a(14)
0C1Fh	a(15)

Figure 10. Phase 1 Data Memory

4.2.2 Phase 2: N-Point Complex FFT

In phase 2, an N-point complex FFT is performed in place in the data-processing buffer. The twiddle factors are in Q15 format and are stored in two separate tables, pointed to by sine and cosine. Each table contains 512 values, corresponding to angles ranging from 0 to almost 180 degrees. The indexing scheme used in this algorithm permits the same twiddle tables for inputs of different sizes. Since circular addressing indexes the tables, the starting address of each table must line up to an address with 0s in the eight LSBs.

1. Perform an N-point complex FFT on d(n). The resulting sequence is

   $D[k] = F\{d(n)\} = R[k] + j\, I[k]$

   where R[k] and I[k] are the real and imaginary parts of D[k], respectively.

2. Since the FFT computation is done in place, the resulting sequence, D[k], occupies the upper half of the data-processing buffer, as shown. The lower half of the data processing buffer still contains the original real input sequence, a(n). This is overwritten in phase 3.

3. All the information from the original 2N-point real sequence, a(n), is contained in this N-point complex sequence, D[k]. The remainder of the algorithm unpacks D[k] into the final 2N-point complex sequence, A[k] = F{a(n)}.

0C00h	R[0]
0C01h	I[0]
0C02h	R[1]
0C03h	I[1]
0C04h	R[2]
0C05h	I[2]
0C06h	R[3]
0C07h	I[3]
0C08h	R[4]
0C09h	I[4]
0C0Ah	R[5]
0C0Bh	I[5]
0C0Ch	R[6]
0C0Dh	I[6]
0C0Eh	R[7]
0C0Fh	I[7]
0C10h	a(0)
0C11h	a(1)
0C12h	a(2)
0C13h	a(3)
0C14h	a(4)
0C15h	a(5)
0C16h	a(6)
0C17h	a(7)
0C18h	a(8)
0C19h	a(9)
0C1Ah	a(10)
0C1Bh	a(11)
0C1Ch	a(12)
0C1Dh	a(13)
0C1Eh	a(14)
0C1Fh	a(15)

Figure 11. Phase 2 Data Memory

4.2.3 Phase 3: Separation of Odd and Even Parts

Phase 3 separates the FFT output to compute four independent sequences: RP, RM, IP, and IM, which are the even real, odd real, even imaginary, and the odd imaginary parts, respectively.

1. D[k] is separated into its real even part, RP[k], real odd part, RM[k], imaginary even part, IP[k], and imaginary odd part, IM[k,] according to the following equations:

   $RP[k] = RP[N-k] = 0.5 * (R[k] + R[N-k])$

   $RM[k] = -RM[N-k] = 0.5 * (R[k] - R[N-k])$

   $IP[k] = IP[N-k] = 0.5 * (I[k] + I[N-k])$

   $IM[k] = -IM[N-k] = 0.5 * (I[k] - I[N-k])$

   $RP[0] = R[0]$

   $IP[0] = I[0]$

   $RM[0] = IM[0] = RM[N/2] = IM[N/2] = 0$

   $RP[N/2] = R[N/2]$

   $IP[N/2] = I[N/2]$

2. Figure 12 shows the organization of the values at the end of phase three. The sequences RP[k] and IP[k] are stored in the upper half of the data processing buffer in ascending order; the sequences RM[k] and IM[k] are stored in the lower half in descending order.

0C00h	RP[0] = R[0]
0C01h	IP[0] = I[0]
0C02h	RP[1]
0C03h	IP[1]
0C04h	RP[2]
0C05h	IP[2]
0C06h	RP[3]
0C07h	IP[3]
0C08h	RP[4] = R[4]
0C09h	IP[4] = I[4]
0C0Ah	RP[5]
0C0Bh	IP[5]
0C0Ch	RP[6]
0C0Dh	IP[6]
0C0Eh	RP[7]
0C0Fh	IP[7]
0C10h	a(0)
0C11h	a(1)
0C12h	IM[7]
0C13h	RM[7]
0C14h	IM[6]
0C15h	RM[6]
0C16h	IM[5]
0C17h	RM[5]
0C18h	IM[4] = 0
0C19h	RM[4] = 0
0C1Ah	IM[3]
0C1Bh	RM[3]
0C1Ch	IM[2]
0C1Dh	RM[2]
0C1Eh	IM[1]
0C1Fh	RM[1]

Figure 12. Phase 3 Data Memory

4.2.4 Phase 4: Generation of Final Output

Phase 4 performs one more set of butterflies to generate the 2N-point complex output, which corresponds to the DFT of the original 2N-point real input sequence. The output resides in the data processing buffer.

3. The four sequences, RP[k], RM[k], IP[k], and IM[k], are used to compute the real FFT of a(n) according to the following equations.

   $AR[k] = AR[2N - k] = RP[k] + \cos(k\pi/N) * IP[k] - \sin(k\pi/N) * RM[k]$

   $AI[k] = -AI[2N - k] = IM[k] - \cos(k\pi/N) * RM[k] - \sin(k\pi/N) * IP[k]$

   $AR[0] = RP[0] + IP[0]$

   $AI[0] = IM[0] - RM[0]$

   $AR[N] = R[0] - I[0]$

   $AI[N] = 0$

   where:

   $A[k] = A[2N-k] = AR[k] + j\,AI[k] = F\{a(n)\}$

4. The real FFT outputs fill up the entire 4N-word data processing buffer. These outputs are real/imaginary, interleaved, and in natural order, as shown inFigure 13 . The values RM[0] and IM[0] are not stored because they are not used to compute the final outputs in phase 4.

0C00h	AR[0]
0C01h	AI[0]
0C02h	AR[1]
0C03h	AI[1]
0C04h	AR[2]
0C05h	AI[2]
0C06h	AR[3]
0C07h	AI[3]
0C08h	AR[4]
0C09h	AI[4]
0C0Ah	AR[5]
0C0Bh	AI[5]
0C0Ch	AR[6]
0C0Dh	AI[6]
0C0Eh	AR[7]
0C0Fh	AI[7]
0C10h	AR[8]
0C11h	AI[8]
0C12h	AR[9]
0C13h	AI[9]
0C14h	AR[10]
0C15h	AI[10]
0C16h	AR[11]
0C17h	AI[11]
0C18h	AR[12]
0C19h	AI[12]
0C1Ah	AR[13]
0C1Bh	AI[13]
0C1Ch	AR[14]
0C1Dh	AI[14]
0C1Eh	AR[15]
0C1Fh	AI[15]

Figure 13. Phase 4 Data Memory

Appendix A Main.inc

```
* Filename: Main.inc
* Includes all the constants that are used in the entire application
K_0                .set  0     ; constant
K_FIR_INDEX     .set  1     ; index count
K_FIR_BFFR      .set  16    ; FIR buffer size
K_neg1             .set  -1h      ; index count
K_BIQUAD        .set  2     ; there are 2 bi-quad sections
K_IIR_SIZE      .set  10    ; each bi-quad has 5 coeffs
K_STACK_SIZE       .set  200   ; stack size
K_FRAME_SIZE       .set  256      ; PING/PONG buffer size
K_FRAME_FLAG       .set  1     ; set after 256 collected
H_FILT_SIZE     .set  128      ; H(z) filter size
ADPT_FILT_SIZE .set  128      ; W(z) filter size
K_mu            .set  0h       ; initial step constant
K_HOST_FLAG     .set  1     ; Enable EVM_HOST interface
K_DEFAULT_AC01 .set  1h    ; default AC01 init
*  This include file sets the FFT size for the '54x Real FFT code
*  Note that the Real FFT size (i.e. the number of points in the
*  original real input sequence) is 2N; whereas the FFT size is
*  the number of complex points formed by packing the real inputs,
*  which is N.  For example, for a 256-pt Real FFT, K_FFT_SIZE
*  should be set to 128 and K_LOGN should be set to 7.
K_FFT_SIZE      .set  128      ; # of complex points (=N)
K_LOGN             .set  7     ; # of stages (=logN/log2)
K_ZERO_BK          .set  0
K_TWID_TBL_SIZE    .set  128      ; Twiddle table size
K_DATA_IDX_1       .set  2     ; Data index for Stage 1
K_DATA_IDX_2       .set  4     ; Data index for Stage 2
K_DATA_IDX_3       .set  8     ; Data index for Stage 3
K_FLY_COUNT_3      .set  4     ; Butterfly counter for Stage 3
K_TWID_IDX_3       .set  32    ; Twiddle index for Stage 3
```

Appendix B FFT Example Code

.

B.1 256-Point Real FFT Initialization

```
; TEXAS INSTRUMENTS INCORPORATED
; DSP Data Communication System Development / ASP
;
; Archives:    PVCS
; Filename:    rfft.asm
; Version:     1.0
; Status :     draft          ( )
;              proposal       (X)
;              accepted       ( ) dd-mm-yy/?acceptor.
;
; AUTHOR      Simon Lau and Nathan Baltz
;
;             Application Specific Products
;             Data Communication System Development
;             12203 SW Freeway, MS 701
;             Stafford, TX 77477
;{
; IPR statements description (can be collected).
;}
;(C) Copyright 1996. Texas Instruments. All rights reserved.
;
;{
; Change history:
;
;        VERSION      DATE     /      AUTHORS              COMMENT
;          1.0     July-17-96  /   Simon & Nathan      original created
;
;}
;{
;    1. ABSTRACT
;
;    1.1 Function Type
;        a.Core Routine
;        b.Subroutine
;
;    1.2 Functional Description
;        This file contains core routine:
;        rfft
;
;    1.3 Specification/Design Reference (optional)
;
;    1.4 Module Test Document Reference
;        Not done
;
;    1.5 Compilation Information
;        Compiler:       TMS320C54X  ASSEMBLER
;        Version:        1.02 (PC)
;        Activation:     asm500 -s rfft.asm
;
;    1.6 Notes and Special Considerations
;        -
;}
;{
;    2. VOCABULARY
;
;    2.1 Definition of Special Words, Keywords (optional)
```

```
;         -
;     2.2 Local Compiler Flags
;         -
;     2.3 Local Constants
;         -
;}
;{
;     3. EXTERNAL RESOURCES
;
;     3.1 Include Files
          .mmregs
          .include          "main.inc"
          .include          "init_54x.inc"
;     3.2 External Data
          .ref              bit_rev, fft, unpack
          .ref              power
          .ref              sine,cosine
          .ref              sine_table,cos_table
;     3.3 Import Functions
;}
;{
;     4. INTERNAL RESOURCES
;
;     4.1 Local Static Data
;         -
;     4.2 Global Static Data
;         -
;     4.3 Dynamic Data
;         -
;     4.4 Temporary Data
;         -
;     4.5 Export Functions
          .def              rfft_task
;}
;     5. SUBROUTINE CODE
;     HeaderBegin
;==============================================================================
;
;-----------------------------------------------------------------------------
;     5.1 rfft
;
;     5.2 Functional Description
;         The following code implements a Radix-2, DIT, 2N-point Real FFT for the
;         TMS320C54x.  This main program makes four function calls, each
;         corresponds to a different phase of the algorithm. For more details about
;         how each phase is implemented, see bit_rev.asm, fft.asm, unpack.asm, and
;         power.asm assembly files.
;-----------------------------------------------------------------------------
;


;     5.3 Activation
;         Activation example:
;         CALL    rfft
;         Reentrancy:     No
;         Recursive :     No
;     5.4 Inputs
;         NONE
;     5.5 Outputs
;         NONE
;
;     5.6 Global
;
;         Data structure: AR1
```

```
;         Data Format:     16-bit pointer
;         Modified:        No
;         Description:     used for moving the twiddle tables from
;                          program to data
;
;     5.7 Special considerations for data structure
;         -
;     5.8 Entry and Exit conditions
;
;    |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A |B |BK|BRC| T|TRN
;
;in  |U |  1|  1|NU |  1 |NU |NU |NU |NU |NU |NU |NU |NU |NU |NU|NU|NU|NU |NU|NU
;
;out |U |  1|  1|NU |  1 |U  |NU |UM |NU |NU |NU |NU |NU |NU |NU|NU|NU|UM |NU|NU
;
; Note : UM - Used & Modified, U - Used, NU - Not Used
;
;     5.9 Execution
;           Execution time: ?cycles
;           Call rate:      not applicable for this application
;
;==========================================================================
;HeaderEnd
;     5.10 Code
          .asg     AR1,FFT_TWID_P
          .sect    "rfft_prg"
rfft_task:
          STM      #sine,FFT_TWID_P
          RPT      #K_FFT_SIZE-1                 ; move FIR coeffs from program
          MVPD     #sine_table,*FFT_TWID_P+      ; to data
          STM      #cosine,FFT_TWID_P
          RPT      #K_FFT_SIZE-1                 ; move FIR coeffs from program
          MVPD     #cos_table,*FFT_TWID_P+       ; to data
          CALL     bit_rev
          CALL     fft
          CALL     unpack
          CALLD    power
          STM      #K_ST1,ST1                    ; restore the original contents of
                                                 ; ST1 since ASM field has changed
          RET                                    ; return to main program
          .end
```

B.2 Bit Reversal Routine

```
; TEXAS INSTRUMENTS INCORPORATED
; DSP Data Communication System Development / ASP ;
;
; Archives:     PVCS
; Filename:     bit_rev.asm
; Version:      1.0
; Status :      draft          ( )
;               proposal       (X)
;               accepted       ( ) dd-mm-yy/?acceptor.
;
; AUTHOR      Simon Lau and Nathan Baltz
;
;             Application Specific Products
;             Data Communication System Development
;             12203 SW Freeway, MS 701
;             Stafford, TX 77477
;{
; IPR statements description (can be collected).
;}
```

```
;(C) Copyright 1996. Texas Instruments. All rights reserved.
;
;{
; Change history:
;
;        VERSION       DATE     /      AUTHORS              COMMENT
;          1.0     July-17-96  /   Simon & Nathan      original created
;
;}
;{
;    1. ABSTRACT
;
;    1.1 Function Type
;        a.Core Routine
;        b.Subroutine
;
;    1.2 Functional Description
;        This file contains one subroutine:
;        bit_rev
;
;    1.3 Specification/Design Reference (optional)
;        called by rfft.asm depending upon the task thru CALA
;
;    1.4 Module Test Document Reference
;        Not done
;
;    1.5 Compilation Information
;        Compiler:        TMS320C54X  ASSEMBLER
;        Version:         1.02 (PC)
;        Activation:      asm500 -s bit_rev.asm
;
;    1.6 Notes and Special Considerations
;        -
;}
;{
;    2. VOCABULARY
;
;    2.1 Definition of Special Words, Keywords (optional)
;        -
;    2.2 Local Compiler Flags
;        -
;    2.3 Local Constants
;        -
;}
;{
;    3. EXTERNAL RESOURCES
;
;    3.1 Include Files
         .mmregs
         .include    "main.inc"
;    3.2 External Data
         .ref        d_input_addr, fft_data
;    3.3 Import Functions
;}
;{
;    4. INTERNAL RESOURCES
;
;    4.1 Local Static Data
;        -
;    4.2 Global Static Data
;        -
;    4.3 Dynamic Data
;        -
```

```
;     4.4 Temporary Data
;         -
;     4.5 Export Functions
          .def        bit_rev
;}
;     5. SUBROUTINE CODE
;     HeaderBegin
;===============================================================================
;
;-------------------------------------------------------------------------------
;     5.1 bit_rev
;
;     5.2 Functional Description
;         This function is called from the main module of the 'C54x Real FFT code.
;         It reorders the original 2N-point real input sequence by using
;         bit-reversed addressing.  This new sequence is stored into the data
;         processing buffer of size 2N, where FFT will be performed in-place
;         during Phase Two.
;
;-------------------------------------------------------------------------------
;


;     5.3 Activation
;         Activation example:
;         CALL    bit_rev
;         Reentrancy:     No
;         Recursive :     No
;
;     5.4 Inputs
;         NONE
;     5.5 Outputs
;         NONE
;
;     5.6 Global
;
;         Data structure: AR0
;         Data Format:    16-bit index pointer
;         Modified:       No
;         Description:    used for bit reversed addressing
;
;         Data structure: AR2
;         Data Format:    16-bit pointer
;         Modified:       Yes
;         Description:    pointer to processed data in bit-reversed order
;
;         Data structure: AR3
;         Data Format:    16-bit pointer
;         Modified:       Yes
;         Description:    pointer to original input data in natural order
;
;         Data structure: AR7
;         Data Format:    16-bit pointer
;         Modified:       Yes
;         Description:    starting addressing of data processing buffer
;
;     5.7 Special considerations for data structure
;         -
;     5.8 Entry and Exit conditions
;
```

```
;    |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A  |B  |BK|BRC|  T|TRN|
;
;in  |U |  1|  1|NU |  1 |NU |NU |NU |NU |NU |NU |NU |NU |NU |NU |NU |NU|NU |NU |NU |
;
;out |U |  1|  1|NU |  1 |NU |UM |NU |UM |UM |NU |NU |NU |UM |NU |NU |NU|UM |NU |NU |
;
; Note : UM – Used & Modified, U – Used, NU – Not Used
;
;     5.9 Execution
;         Execution time: ?cycles
;         Call rate:      not applicable for this application
;
;==============================================================================
;HeaderEnd
; 	5.10 Code
      .asg    AR2,REORDERED_DATA
      .asg    AR3,ORIGINAL_INPUT
      .asg    AR7,DATA_PROC_BUF
      .sect   ”rfft_prg”
bit_rev:
      SSBX    FRCT                              ; fractional mode is on
      MVDK    d_input_addr,ORIGINAL_INPUT       ; AR3 -> 1 st original input
      STM     #fft_data,DATA_PROC_BUF           ; AR7 -> data processing buffer
      MVMM    DATA_PROC_BUF,REORDERED_DATA      ; AR2 -> 1st bit-reversed data
      STM     #K_FFT_SIZE-1,BRC
      RPTBD   bit_rev_end-1
      STM     #K_FFT_SIZE,AR0                   ; AR0 = 1/2 size of circ buffer
      MVDD    *ORIGINAL_INPUT+,*REORDERED_DATA+
      MVDD    *ORIGINAL_INPUT-,*REORDERED_DATA+
      MAR     *ORIGINAL_INPUT+0B
bit_rev_end:
      RET                                       ; return to Real FFT main module
      end
```

B.3 256-Point Real FFT Routine

```
; TEXAS INSTRUMENTS INCORPORATED
; DSP Data Communication System Development / ASP
;
; Archives:   PVCS
; Filename:   fft.asm
; Version:    1.0
; Status :    draft         ( )
;             proposal      (X)
;             accepted      ( ) dd–mm–yy/?acceptor.
;
; AUTHOR    Simon Lau and Nathan Baltz
;
;            Application Specific Products
;            Data Communication System Development
;            12203 SW Freeway, MS 701
;            Stafford, TX 77477
;{
; IPR statements description (can be collected).
;}
;(C) Copyright 1996. Texas Instruments. All rights reserved.
;
;{
; Change history:
;
;        VERSION      DATE     /      AUTHORS              COMMENT
;          1.0     July–17–96  /   Simon & Nathan      original created
```

```
;
;}
;{
;     1. ABSTRACT
;
;     1.1 Function Type
;         a.Core Routine
;         b.Subroutine
;
;     1.2 Functional Description
;         This file contains one subroutine:
;         fft
;
;     1.3 Specification/Design Reference (optional)
;         called by rfft.asm depending upon the task thru CALA
;
;     1.4 Module Test Document Reference
;         Not done
;
;     1.5 Compilation Information
;         Compiler:       TMS320C54X  ASSEMBLER
;         Version:        1.02 (PC)
;         Activation:     asm500 -s fft.asm
;
;     1.6 Notes and Special Considerations
;         -
;}
;{
;     2. VOCABULARY
;
;     2.1 Definition of Special Words, Keywords (optional)
;         -
;     2.2 Local Compiler Flags
;         -
;     2.3 Local Constants
;         -
;}
;{
;     3. EXTERNAL RESOURCES
;
;     3.1 Include Files
          .mmregs
          .include    "main.inc"
;     3.2 External Data
          .ref        fft_data, d_grps_cnt, d_twid_idx, d_data_idx, sine, cosine
;     3.3 Import Functions
;}
;{
;     4. INTERNAL RESOURCES
;
;     4.1 Local Static Data
;         -
;     4.2 Global Static Data
;         -
;     4.3 Dynamic Data
;         -
;     4.4 Temporary Data
;         -
;     4.5 Export Functions
          .def        fft
;}
;     5. SUBROUTINE CODE
;     HeaderBegin
```

```
;===============================================================================
;
;-----------------------------------------------------------------------------
;    5.1 fft
;
;    5.2 Functional Description
;        PHASE TWO   (LogN)–Stage Complex FFT
;        This function is called from main module of the 'C54x Real FFT code.
;        Here we assume that the original 2N–point real input sequence is al
;        ready packed into an N–point complex sequence and stored into the
;        data processing buffer in bit–reversed order (as done in Phase One).
;        Now we perform an in–place, N–point complex FFT on the data proces
;        sing buffer, dividing the outputs by 2 at the end of each stage to
;        prevent overflow. The resulting N–point complex sequence will be un–
;        packed into a 2N–point complex sequencein Phase Three & Four.
;-----------------------------------------------------------------------------
;
;
;    5.3 Activation
;        Activation example:
;        CALL    fft
;        Reentrancy:     No
;        Recursive :     No
;
;    5.4 Inputs
;        NONE
;    5.5 Outputs
;        NONE
;
;    5.6 Global
;
;        Data structure:    AR0
;        Data Format:       16–bit index pointer
;        Modified:          No
;        Description:       index to twiddle tables
;
;        Data structure:    AR1
;        Data Format:       16–bit counter
;        Modified:          No
;        Description:       group counter
;
;        Data structure:    AR2
;        Data Format:       16–bit pointer
;        Modified:          Yes
;        Description:       pointer to 1st butterfly data PR,PI
;
;        Data structure:    AR3
;        Data Format:       16–bit pointer
;        Modified:          Yes
;        Description:       pointer to 2nd butterfly data QR,QI
;
;        Data structure:   AR4
;        Data Format:      16–bit pointer
;        Modified:         Yes
;        Description:      pointer to cosine value WR
;
;        Data structure:   AR5
;        Data Format:      16–bit pointer
;        Modified:         Yes
;        Description:      pointer to cosine value WI
;
;        Data structure:   AR6
;        Data Format:      16–bit counter
```

```
;       Modified:           Yes
;       Description:        butterfly counter
;
;       Data structure:     AR7
;       Data Format:        16-bit pointer
;       Modified:           Yes
;       Description:        start address of data processing buffer
;
;
;    5.7 Special considerations for data structure
;       -
;    5.8 Entry and Exit conditions
;
;    |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A |B |BK|BRC| T|TRN|
;
;in  |U |  1|  1|NU |  1 |  0|NU |NU |NU |NU |NU |NU |NU |NU |NU|NU|NU|NU |NU|NU |
;
;out |U |  1|  1|NU |  1 | -1|UM |UM |UM |UM |UM |UM |UM |UM |UM|UM|UM|UM |NU|NU |
;
; Note : UM - Used & Modified, U - Used, NU - Not Used
;
;    5.9 Execution
;       Execution time: ?cycles
;       Call rate: not applicable for this application
;
;==========================================================================
;    HeaderEnd
;    5.10 Code
          .asg     AR1,GROUP_COUNTER
          .asg     AR2,PX
          .asg     AR3,QX
          .asg     AR4,WR
          .asg     AR5,WI
          .asg     AR6,BUTTERFLY_COUNTER
          .asg     AR7,DATA_PROC_BUF              ; for Stages 1 & 2
          .asg     AR7,STAGE_COUNTER              ; for the remaining stages
          .sect    "rfft_prg"
fft:
;   Stage 1 -----------------------------------------------------------------
        STM        #K_ZERO_BK,BK                  ; BK=0 so that *ARn+0% == *ARn+0
        LD         #-1,ASM                        ; outputs div by 2 at each stage
        MVMM       DATA_PROC_BUF,PX               ; PX -> PR
        LD         *PX,A                          ; A  :=  PR
        STM        #fft_data+K_DATA_IDX_1,QX      ; QX -> QR
        STM        #K_FFT_SIZE/2-1,BRC
           RPTBD      stage1end-1
        STM        #K_DATA_IDX_1+1,AR0
        SUB        *QX,16,A,B                     ; B  :=  PR-QR
        ADD        *QX,16,A                       ; A  :=  PR+QR
        STH        A,ASM,*PX+                     ; PR':= (PR+QR)/2
        ST         B,*QX+                         ; QR':= (PR-QR)/2
        ||LD       *PX,A                          ; A  :=  PI
        SUB        *QX,16,A,B                     ; B  :=  PI-QI
           ADD        *QX,16,A                    ; A  :=  PI+QI
           STH        A,ASM,*PX+0                 ; PI':= (PI+QI)/2
        ST         B,*QX+0%                       ; QI':= (PI-QI)/2
        ||LD       *PX,A                          ; A  :=  next PR


stage1end:
;   Stage 2 -----------------------------------------------------------------
```

```
            MVMM        DATA_PROC_BUF,PX                ; PX -> PR
            STM         #fft_data+K_DATA_IDX_2,QX       ; QX -> QR
            STM         #K_FFT_SIZE/4-1,BRC
            LD          *PX,A                           ; A  :=  PR
            RPTBD       stage2end-1
         STM         #K_DATA_IDX_2+1,AR0
; 1st butterfly
         SUB         *QX,16,A,B                      ; B  :=  PR-QR
         ADD         *QX,16,A                        ; A  :=  PR+QR
         STH         A,ASM,*PX+                      ; PR':= (PR+QR)/2
            ST          B,*QX+                          ; QR':= (PR-QR)/2
         ||LD        *PX,A                           ; A  :=  PI
         SUB         *QX,16,A,B                      ; B  :=  PI-QI
         ADD         *QX,16,A                        ; A  :=  PI+QI
         STH         A,ASM,*PX+                      ; PI':= (PI+QI)/2
         STH         B,ASM,*QX+                      ; QI':= (PI-QI)/2
; 2nd butterfly
         MAR      *QX+
            ADD      *PX,*QX,A                          ; A  :=  PR+QI
            SUB      *PX,*QX-,B                         ; B  :=  PR-QI
         STH      A,ASM,*PX+                         ; PR':= (PR+QI)/2
         SUB      *PX,*QX,A                          ; A  :=  PI-QR
         ST       B,*QX                              ; QR':= (PR-QI)/2
         ||LD     *QX+,B                             ; B  :=     QR
            ST       A, *PX                             ; PI':= (PI-QR)/2
         ||ADD    *PX+0%,A                           ; A  :=  PI+QR
         ST       A,*QX+0%                           ; QI':= (PI+QR)/2
         ||LD     *PX,A                           ; A  :=  PR
stage2end:
; Stage 3 thru Stage logN-1 ---------------------------------------------------
      STM   #K_TWID_TBL_SIZE,BK                   ; BK = twiddle table size always
      ST    #K_TWID_IDX_3,d_twid_idx              ; init index of twiddle table
      STM   #K_TWID_IDX_3,AR0                        ; AR0 = index of twiddle table
      STM   #cosine,WR                               ; init WR pointer
      STM   #sine,WI                                 ; init WI pointer
      STM   #K_LOGN-2-1,STAGE_COUNTER                ; init stage counter
      ST       #K_FFT_SIZE/8-1,d_grps_cnt               ; init group counter
      STM   #K_FLY_COUNT_3-1,BUTTERFLY_COUNTER    ; init butterfly counter
      ST       #K_DATA_IDX_3,d_data_idx                 ; init index for input data
stage:
      STM      #fft_data,PX                             ; PX -> PR
      LD    d_data_idx, A
      ADD      *(PX),A
      STLM  A,QX                                     ; QX -> QR
      MVDK     d_grps_cnt,GROUP_COUNTER                 ; AR1 contains group counter

group:
      MVMD     BUTTERFLY_COUNTER,BRC                    ; # of butterflies in each
grp
      RPTBD    butterflyend-1
      LD       *WR,T                                    ; T  :=  WR
      MPY      *QX+,A                                ; A  :=  QR*WR  || QX->QI
      MACR     *WI+0%,*QX-,A                         ; A  :=  QR*WR+QI*WI
                                                     ; || QX->QR
      ADD      *PX,16,A,B                               ; B  := (QR*WR+QI*WI)+PR
      ST       B,*PX                                 ; PR':=((QR*WR+QI*WI)+PR)/2
      ||SUB    *PX+,B                                ; B  :=  PR-(QR*WR+QI*WI)
                                                     ; || PX->PI
      ST       B,*QX                                 ; QR':= (PR-(QR*WR+QI*WI))/2
      ||MPY    *QX+,A                                   ; A  :=  QR*WI [T=WI]
                                                     ; || QX->QI
      MASR     *QX,*WR+0%,A                             ; A  :=  QR*WI-QI*WR
      ADD      *PX,16,A,B                            ; B  := (QR*WI-QI*WR)+PI
```

```
      ST       B,*QX+                                     ; QI':=((QR*WI-QI*WR)+PI)/2
                                                          ; || QX->QR
      ||SUB    *PX,B                                      ; B  :=  PI-(QR*WI-QI*WR)
      LD       *WR,T                                      ; T  :=  WR
      ST       B,*PX+                                     ; PI':= (PI-(QR*WI-QI*WR))/2
                                                          ; || PX->PR
      ||MPY    *QX+,A                                     ; A  :=  QR*WR  || QX->QI
butterflyend:
; Update pointers for next group
      PSHM     AR0                                        ; preserve AR0
      MVDK     d_data_idx,AR0
     MAR       *PX+0                                         ; increment PX for next group
     MAR       *QX+0                                         ; increment QX for next group
     BANZD     group,*GROUP_COUNTER-
     POPM      AR0                                           ; restore AR0
     MAR       *QX-
; Update counters and indices for next stage
     LD        d_data_idx,A
     SUB       #1,A,B                                        ; B = A-1
     STLM      B,BUTTERFLY_COUNTER                           ; BUTTERFLY_COUNTER = #flies-1
     STL       A,1,d_data_idx                                ; double the index of data
     LD      d_grps_cnt,A
     STL     A,ASM,d_grps_cnt                            ; 1/2 the offset to next group
     LD      d_twid_idx,A
     STL     A,ASM,d_twid_idx                            ; 1/2 the index of twiddle table
     BANZ    D stage,*STAGE_COUNTER-
     MVDK    d_twid_idx,AR0                              ; AR0 = index of twiddle table
fft_end:
     RET                                                 ; return to Real FFT main module
    .end
```

B.4 Unpack 256-Point Real FFT Output

```
;    TEXAS INSTRUMENTS INCORPORATED
;    DSP Data Communication System Development / ASP
;    Archives:    PVCS
;    Filename:    unpack.asm
;    Version:     1.0
;    Status :     draft         ( )
;                 proposal      (X)
;                 accepted      ( ) dd-mm-yy/?acceptor.
;
;    AUTHOR    Simon Lau and Nathan Baltz
;
;            Application Specific Products
;            Data Communication System Development
;            12203 SW Freeway, MS 701
;            Stafford, TX 77477
;{
;    IPR statements description (can be collected).
;}
;    (C) Copyright 1996. Texas Instruments. All rights reserved.
;{
;    Change history:
;
;        VERSION      DATE     /      AUTHORS              COMMENT
;          1.0     July-17-96  /   Simon & Nathan      original created
;
;}
;{
;    1. ABSTRACT
;
;    1.1 Function Type
```

```
;         a.Core Routine
;         b.Subroutine
;
;     1.2 Functional Description
;         This file contains one subroutine:
;         unpack
;
;     1.3 Specification/Design Reference (optional)
;         called by rfft.asm depending upon the task thru CALA
;
;     1.4 Module Test Document Reference
;         Not done
;

;     1.5 Compilation Information
;         Compiler:       TMS320C54X  ASSEMBLER
;         Version:        1.02 (PC)
;         Activation:     asm500 -s unpack.asm
;
;     1.6 Notes and Special Considerations
;         -
;}
;{
;     2. VOCABULARY
;
;     2.1 Definition of Special Words, Keywords (optional)
;         -
;     2.2 Local Compiler Flags
;         -
;     2.3 Local Constants
;         -
;}
;{
;     3. EXTERNAL RESOURCES
;
;     3.1 Include Files
          .mmregs
          .include        "main.inc"
;     3.2 External Data
          .ref            fft_data,sine, cosine
;     3.3 Import Functions
;}
;{
;     4. INTERNAL RESOURCES
;
;     4.1 Local Static Data
;         -
;     4.2 Global Static Data
;         -
;     4.3 Dynamic Data
;         -
;     4.4 Temporary Data
;         -
;     4.5 Export Functions
          .def        unpack
;}
;     5. SUBROUTINE CODE
;     HeaderBegin
;==============================================================================
;
;     5.1  unpack
;
;     5.2 Functional Description
```

```
;
;        PHASE THREE & FOUR      Unpacking to 2N Outputs
;        This function is called from the main module of the 'C54x Real FFT
;        code. It first computes four intermediate sequences (RP,RM, IP, IM)
;        from the resulting complex sequence at the end of the previous phase.
;        Next, it uses the four intermediate sequences to form the FFT of the
;        original 2N-point real input. Again, the outputs are divided by 2 to
;        prevent overflow
;-----------------------------------------------------------------------------
;
;    5.3 Activation
;        Activation example:
;          CALL    unpack
;
;        Reentrancy:     No
;        Recursive :     No
;
;    5.4 Inputs
;        NONE
;    5.5 Outputs
;        NONE
;
;    5.6 Global
;
;    5.6.1 Phase Three Global
;
;          Data structure:    AR0
;          Data Format:       16-bit index pointer
;          Modified:          No
;          Description:       index to twiddle tables
;
;          Data structure:    AR2
;          Data Format:       16-bit pointer
;          Modified:          Yes
;          Description:       pointer to R[k], I[k], RP[k], IP[k]
;
;          Data structure:    AR3
;          Data Format:       16-bit pointer
;          Modified:          Yes
;          Description:       pointer to R[N-k], I[N-k], RP[N-k], IP[N-k]
;
;          Data structure:    AR6
;          Data Format:       16-bit pointer
;          Modified:          Yes
;          Description:       pointer to RM[k], IM[k]
;
;          Data structure:    AR7
;          Data Format:       16-bit pointer
;          Modified:          Yes
;          Description:       pointer to RM[n-k], IM[n-k]
;
;    5.6.2 Phase Four Global
;
;          Data structure: AR0
;          Data Format:       16-bit index pointer
;          Modified:          No
;          Description:       index to twiddle tables
;
;          Data structure: AR2
;          Data Format:       16-bit counter
;          Modified:          No
;          Description:       pointer to RP[k], IP[k], AR[k], AI[k], AR[0]
;
```

```
;           Data structure: AR3
;           Data Format:    16-bit pointer
;           Modified:       Yes
;           Description:    pointer to RM[k], IM[k], AR[2N-k], AI[2N-k]
;
;           Data structure: AR4
;           Data Format:    16-bit pointer
;           Modified:       Yes
;           Description:    pointer to cos(k*pi/N), AI[0]
;
;           Data structure: AR5
;           Data Format:    16-bit pointer
;           Modified:       Yes
;           Description:    pointer to sin(k*pi/N), AR[N], AI[N]
;
;     5.7 Special considerations for data structure
;         -
;     5.8 Entry and Exit conditions
;
;     5.8.1 Phase Three Entry and Exit Conditions
;
;    |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A  |B  |BK|BRC|  T|TRN
;
;in  |2 |  1|  1|  0|  1 |  0|NU |NU |NU |NU |NU |NU |NU |NU |NU|NU |0  |NU |NU |NU
;
;out |2 |  1|  1|  0|  1 |-1 |UM |NU |UM |UM |NU |NU |UM |UM |UM|UM |UM |UM |NU |NU
;
; Note : UM - Used & Modified, U - Used, NU - Not Used
;
;     5.8.2 Phase Four Entry and Exit Conditions
;
;    |DP|OVM|SXM|C16|FRCT|ASM|AR0|AR1|AR2|AR3|AR4|AR5|AR6|AR7|A  |B  |BK|BRC|  T|TRN
;
;in  |U |  1|  1|  0|  1 |-1 |NU |NU |NU |NU |NU |NU |NU |NU |NU|NU |NU |NU |NU |NU
;
;out |U |  1|  1|  0|  1 |-1 |UM |NU |UM |UM |UM |UM |NU |NU |UM|UM |UM |UM |NU |NU
;
; Note : UM - Used & Modified, U - Used, NU - Not Used
;
;     5.9 Execution
;         Execution time: ?cycles
;         Call rate:      not applicable for this application
;
;==========================================================================
;          HeaderEnd
;     5.10  Code
    .sect        "rfft_prg"
unpack:
; Compute intermediate values RP, RM, IP, IM
    .asg    AR2,XP_k
    .asg    AR3,XP_Nminusk
    .asg    AR6,XM_k
    .asg    AR7,XM_Nminusk

     STM    #fft_data+2,XP_k                    ; AR2 -> R[k] (temp RP[k])
     STM    #fft_data+2*K_FFT_SIZE-2,XP_Nminusk  ; AR3 -> R[N-k] (temp
                                                           RP[N-k])
     STM    #fft_data+2*K_FFT_SIZE+3,XM_Nminusk  ; AR7 -> temp RM[N-k]
     STM    #fft_data+4*K_FFT_SIZE-1,XM_k        ; AR6 -> temp RM[k]
     STM    #-2+K_FFT_SIZE/2,BRC
     RPTBD  phase3end-1
     STM    #3,AR0
     ADD    *XP_k,*XP_Nminusk,A                 ; A := R[k]+R[N-k] =
```

```
                                                         2*RP[k]
     SUB     *XP_k,*XP_Nminusk,B                       ; B := R[k]-R[N-k] =
                                                         2*RM[k]
     STH     A,ASM,*XP_k+                              ; store RP[k] at AR[k]
     STH     A,ASM,*XP_Nminusk+                        ; store RP[N-k]=RP[k] at
                                                         AR[N-k]
     STH     B,ASM,*XM_k-                              ; store RM[k] at AI[2N-k]
     NEG     B                                         ; B := R[N-k]-R[k] =
                                                         2*RM[N-k]
     STH     B,ASM,*XM_Nminusk-                        ; store RM[N-k] at AI[N+k]
     ADD     *XP_k,*XP_Nminusk,A                       ; A := I[k]+I[N-k] =
                                                         2*IP[k]
     SUB     *XP_k,*XP_Nminusk,B                       ; B := I[k]-I[N-k] =
                                                         2*IM[k]
     STH     A,ASM,*XP_k+                              ; store IP[k] at AI[k]
     STH     A,ASM,*XP_Nminusk-0                       ; store IP[N-k]=IP[k] at
                                                         AI[N-k]
     STH     B,ASM,*XM_k-                              ; store IM[k] at AR[2N-k]
     NEG     B                                         ; B := I[N-k]-I[k] =
                                                         2*IM[N-k]
     STH     B,ASM,*XM_Nminusk+0                       ; store IM[N-k] at AR[N+k]
phase3end:
     ST      #0,*XM_k-                                 ; RM[N/2]=0
     ST      #0,*XM_k                                  ; IM[N/2]=0
; Compute AR[0],AI[0], AR[N], AI[N]
     .asg    AR2,AX_k
     .asg    AR4,IP_0
     .asg    AR5,AX_N
     STM     #fft_data,AX_k                            ; AR2 -> AR[0] (temp
                                                             RP[0])
     STM     #fft_data+1,IP_0                          ; AR4 -> AI[0] (temp
                                                             IP[0])
     STM     #fft_data+2*K_FFT_SIZE+1,AX_N             ; AR5 -> AI[N]
     ADD     *AX_k,*IP_0,A                             ; A := RP[0]+IP[0]
     SUB     *AX_k,*IP_0,B                             ; B := RP[0]-IP[0]
     STH     A,ASM,*AX_k+                              ; AR[0] = (RP[0]+IP[0])/2
     ST      #0,*AX_k                                       ; AI[0] = 0
     MVDD    *AX_k+,*AX_N-                                  ; AI[N] = 0
     STH      B,ASM,*AX_N                                   ; AR[N] = (RP[0]-IP[0])/2
; Compute final output values AR[k], AI[k]
     .asg      AR3,AX_2Nminusk
     .asg      AR4,COS
     .asg      AR5,SIN

     STM       #fft_data+4*K_FFT_SIZE-1,AX_2Nminusk     ; AR3 -> AI[2N-1]
                                                             (temp RM[1])
     STM       #cosine+K_TWID_TBL_SIZE/K_FFT_SIZE,COS   ; AR4 -> cos(k*pi/N)
     STM       #sine+K_TWID_TBL_SIZE/K_FFT_SIZE,SIN     ; AR5 -> sin(k*pi/N)
     STM       #K_FFT_SIZE-2,BRC
     RPTBD     phase4end-1
     STM       #K_TWID_TBL_SIZE/K_FFT_SIZE,AR0          ; index of twiddle
                                                             tables
     LD        *AX_k+,16,A                              ; A := RP[k] ||
                                                             AR2->IP[k]
     MACR      *COS,*AX_k,A                             ; A :=A+cos(k*pi/N)
                                                              *IP[k]
     MASR      *SIN,*AX_2Nminusk-,A                     ; A := A-sin(k*pi/N)
                                                             *RM[k]
                                                        ; || AR3->IM[k]
     LD        *AX_2Nminusk+,16,B                       ; B := IM[k] ||
                                                             AR3->RM[k]
     MASR      *SIN+0%,*AX_k-,B                         ; B := B-sin(k*pi/N)
                                                             *IP[k]
```

```
                                                              ; || AR2->RP[k]
     MASR      *COS+0%,*AX_2Nminusk,B                         ; B := B-cos(k*pi/N)
                                                              ;        *RM[k]
     STH       A,ASM,*AX_k+                                   ; AR[k] = A/2
     STH       B,ASM,*AX_k+                                   ; AI[k] = B/2
     NEG       B                                              ; B := -B
     STH       B,ASM,*AX_2Nminusk-                            ; AI[2N-k] = -AI[k]
                                                              ;          = B/2
     STH       A,ASM,*AX_2Nminusk-                            ; AR[2N-k] = AR
                                                              ;      [k] = A/2
phase4end:
     RET                                                      ; returntoRealFFTmain module
     .end
```

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products 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.

Customers are responsible for their applications using TI components.

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

TI assumes no liability for applications assistance or customer product design. TI does not 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. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.

Copyright © 2000, Texas Instruments Incorporated