Implementing Real-Time Cardiac Imaging Using the TMS320C3x DSP

APPLICATION REPORT: SPRA192

Authors: Peter Chou
Hong Jiang
Advisor: Professor Z.-P. Liang
Department of Electrical Engineering
University of Illinois at Urbana-Champaign

Digital Signal Processing Solutions
July 1997

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 application 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.

TRADEMARKS

Contents

Implementing Real-Time Cardiac Imaging Using the TMS320C3x DSP

Abstract

Magnetic resonance imaging (MRI) is a powerful medical diagnostic tool providing very detailed three-dimensional images of both stationary and moving objects. Unfortunately, current MRI systems cannot produce high definition images of moving objects in real time because of errors and artifacts caused by movement in the images. One method used to correct these artifacts collects position data using *navigator echoes* and post-processes the data to reverse the effects of object motion. Nevertheless, the physician still must wait for the data to be processed before viewing the images.

This application report describes a novel dynamic MRI method for real-time, high-resolution cardiac imaging using the Texas Instruments (TI™) TMS320C3x digital signal processor (DSP). The TMS320C3x processes the data as it is being received, creating real-time images of moving organs. Applications for real-time dynamic imaging include cardiac imaging, angiography, and abdominal imaging.

1. Introduction

Despite the impressive advances in medical technology through the ages, proper medical diagnosis still requires the human body to be examined directly. Physicians require a tool that allows them to peer deeply within the human anatomy to diagnose illness and suggest courses of action. To date, this need has been met using various imaging techniques, such as x-ray computed tomography (CT), positron emission tomography (PET), ultrasound, and MRI. Out of these modalities, MRI produces images with superior soft-tissue contrast and resolution. Moreover, MRI is minimally invasive and uses non-ionizing radiation, so it is not known to be harmful to human tissue. As a result, MRI holds tremendous promise in medical applications.

MRI possesses great potential as a medical imaging tool scanning stationary objects to produce high-definition images. However, in many applications, physicians are more concerned with moving objects than with static ones. For example, it is often necessary to acquire images of a beating heart, a breathing lung, or a moving abdomen. Unfortunately, the movement associated with the organ causes errors, motion artifacts known as *ghosting*, and general degradation in the resulting images. One method to cope with these image artifacts uses a computer to post-process the data by performing time-consuming calculations after all the data is acquired. The desired image can be produced only after a lengthy amount of calculation time. Current MRI systems cannot manipulate the data quickly enough for real-time dynamic images.

Hong Jiang and Professor Z.-P. Liang at the University of Illinois have recently developed a dynamic magnetic resonance imaging method for real-time high-resolution cardiac imaging. Present day MRI machines require collecting extra position data and post-processing them to achieve the desired result. We propose that dynamic MRI systems could be improved by using a dedicated DSP to continuously process and correct data as it is acquired, rather than collecting all the data first and then post-processing. Implementing an existing MRI system with the Texas Instruments TMS32OC3x DSP chip for real-time data acquisition control and processing makes real-time dynamic imaging possible. The new system has amazing potential for wide-scale implementation in all MRI systems and facilitates accurate diagnoses of current medical problems.

2. MRI and the Problem of Motion

This section explains the principles of MRI and describes how motion affects MRI results.

2.1 How MRI Works

MRI is based on the concept of nuclear magnetic resonance (NMR). NMR is observed when a strong magnet creates a strong, steady magnetic field that causes some of the protons of the atoms making up the target tissue to line up and spin together in the same direction.

A radio-frequency signal is applied to disrupt the spin of the protons. When the signal stops, the protons release energy as they move back to their aligned positions. A receiver coil measures the energy released as well as the time it takes for the protons to return to their original states. These radio-frequency pulses are applied multiple times in various ways to acquire the imaging data.

A computer then manipulates the data using various mathematical techniques to form an image. Because the image contrast of the picture depends on proton density and other tissue characteristics, MRI images possess great soft tissue contrast.

MRI systems form images by relating the magnitude of the received signal from each volume element, or *voxel*, of the object to the proton NMR properties of the tissues in that voxel. To form an image, it must be possible to identify the spatial location of the voxels corresponding to the received signals. This is accomplished by using varying magnetic fields to spatially encode the signals from different voxels with varying frequency and phase shift so that each unique location corresponds to a unique frequency and phase-shift value.

To frequency encode data along one dimension, a gradient magnetic field is applied during data acquisition along the dimension. The received signal is a composite consisting of many different frequencies, each frequency corresponding to a different spatial location along the frequency-encoded dimension.

The Fourier transform can be applied to the composite signal to give its frequency distribution, allowing extraction of the individual frequencies and amplitudes. This information gives the spin density at the particular spatial location along one dimension, which is used as the image data for the dimension.

To form a two-dimensional image, phase encoding must be used to provide image data along the other dimension. Unlike frequency encoding, which is applied with a single gradient and scan, phase encoding involves applying numerous gradients and taking numerous scans. Thus, the process of phase encoding is responsible for the long period of time required to generate an image.

2.2 The Problem of Motion

Medical professionals often need to acquire detailed images of moving tissue, such as a beating heart, moving abdomen, or breathing lungs. The long data acquisition time of MRI causes the resultant images of moving objects to be degraded, containing blurs, motion artifacts, and ghosts. One method to measure and correct the effects of motion involves accompanying the imaging data with *navigator echoes*[1] to provide a record of displacements during imaging. A computer can then be used to apply a set of algorithms to reverse the effects of the displacements. Using a computer to post-process the data is an intensive and time-consuming task that can only be performed after all the data is acquired. Thus, corrected images can be produced only after lengthy amounts of time spent on calculations. Because of the time and amount of processing needed to remove and correct these errors, the ability to image motion is the most significant limitation in current MRI systems.

3. Incorporating the TMS320C3x in an MRI System

Because of the slow nature of the phase-encoding process during MRI acquisition, motion causes changes in the position of structures between phase-encoding measurements. We used the Texas Instruments TMS32OC3x DSP to monitor the image data as it is being acquired to detect this motion. Our goal was to process the image data as it occurs on the fly rather than after all the data is collected. The TMS320C3x DSP must perform the computations quickly enough to keep up with the data acquisition rate and allow the scanner to produce images of moving objects with no delay.

Essentially, the TMS320C3x allows the MRI to produce dynamic images of moving objects in real-time. This breakthrough has direct applications in cardiac imaging, angiography, and abdominal imaging. The effects are wide and far-reaching, having potential applications in functional imaging, reduced time required for imaging, increased utilization in medical facilities, and possibly a reduction in the costs of the scans.

We denote the navigator signal from the first phase-encoding scan, $X_R[k]$. This is the reference data, which provides the original reference position and is in the frequency domain, or *k-space*. We denote the data acquired in each of the subsequent phase-encoding scans, $X_D[k]$. This is the dynamic data, which provides the new position of the structure, which may or may not have moved.

The movement of the specimen causes linear phase shifts in $X_D[k]$, the frequency data received. According to the Fourier transform, a phase shift in the frequency domain corresponds to a displacement in the spatial domain. Thus, to detect phase shifts in the frequency data, we transform the data to the spatial domain and calculate the spatial shift. Let $x_R[t]$ denote the inverse Fourier transform of $X_R[k]$, *and* $x_D[t]$ the inverse Fourier transform of $X_D[k]$. Then, the amount of displacement is determined by first performing a cross-correlation of the two signals, given by:

$$y[t] = \sum_{m=0} x_R[m] x_D[m-t]$$

The amount of spatial displacement is now simply given by the location of the peak of the cross-correlation function $y[t]$. If this value is greater than some threshold, there was too much motion and the DSP signals the computer controlling the data acquisition to re-scan. Otherwise, if the amount of displacement is smaller than the threshold, the motion is within tolerable limits and the next scan is made. Since the motion is normally constrained within a certain range, only a partial cross-correlation need be computed.

To provide different soft-tissue contrast, the data acquisition delay may vary in the millisecond to second range. For example, imaging techniques based on a gradient-recall echo usually have time delays ranging from 5 to 15 milliseconds. For the real-time system to be useful, the on-line data processing must occur during this short time. We can see from above that the most computational-intensive processes are the complex-value FFT and the cross-correlation.
For data of length 256, using the TMS32OC3x on-chip memory, the FFT can be computed in about 1 millisecond[2], and the partial cross - correlation can also be done in approximately the same time. Thus, we can achieve this speed requirement and successfully measure high-resolution motion on the fly.

4. Medical Applications

Using MRI to probe deep inside the human body has many benefits. It is safe and produces high quality images containing unprecedented levels of soft-tissue information. As a result, MRI is finding increased applications in medicine. MRI systems equipped with the TMS320C3x will be even more versatile.

The precision of MRI pictures means that physicians can get as much information from the image as from looking directly at the tissue. Thus, MRI has the potential to reduce the number of certain diagnostic surgeries. Because MRI is especially valuable in clearly defining soft tissue, it has applications in diagnosing brain and nervous system disorders. MRI images are used in oncology to identify diseased tissue and tumors by locating accumulations of fluids. Additionally, the effectiveness of a treatment can be evaluated without having to perform invasive surgery or expose the patient to harmful radiation. The TMSC320C3x enables an MRI system to provide real-time images of these moving tissues.

In addition, the outstanding contrast and spatial resolution of MRI has led to its increased use in functional imaging to view and map brain function. The real-time imaging method is certainly applicable in this imaging application due to its dynamic nature.

Magnetic resonance angiography is also increasing in viability. Because there is no harm of radiation and MRI does not require the injection of contrast agents, it is a safer alternative to x-rays. Furthermore, computers can generate for cardiologists views of different cross sections from different angles based on MRI data. Cardiovascular disease is better diagnosed, because MRI can see right into the heart and blood vessels, making it possible to measure blood flow and the effects of plaque in the arteries. Blood flow and flow effects can be even better imaged with our real-time implementation.

Cost is certainly a factor in the scope of an MRI application. As MRI increases in speed, the cost per procedure will most likely decrease, thus enabling it to be used in more procedures. MRI clearly already enjoys a wide array of applications; using the TMS320C3x enhancement broadens the scope of applications even further.

5. Conclusion

The phenomenon of NMR and its use in MRI has far-reaching effects in medicine. The main advantage of MRI is that it offers strong contrast resolution in scans of soft tissues and has the ability to probe the molecular characteristics of nuclear species in healthy and diseased tissue. Furthermore, MRI is safe, unlike other imaging modalities (such as the x-ray).

The main limitation of MRI is its inability to produce high-quality images of moving organs. This results from the significant image degradation caused by motion artifacts and the large amount of data processing needed to correct these image artifacts. By implementing the Texas Instruments TMS320C3x DSP as a controller and on-line data processor during the data acquisition process, it is possible to monitor the data for motion and generate an image unaffected by motion artifacts in real-time. The speed of the DSP makes it possible to perform the required computations in real-time during the data acquisition.

The greatest application of real-time MRI is in generating images of moving organs such as the heart. Because the chip runs more quickly, scans that once took minutes can now be done in seconds. This has great implications on the general populace by allowing more patients to be screened in shorter times. In the future, we expect to be able to use real-time MRI to generate high-quality cardiac and abdominal images, as well as more precise images of blood flow. Functional imaging will also become more advanced with the capability of generating high-quality real-time images. The additional applications of MRI are endless. MRI has definitely made a great impact and will continue to play a major role in the field of medicine in the future.

Appendix A. Source Code

```
#define MAG-SQ(x,y)                            (x*x+y*y)

extem fft(int, int, float *);
extem volatile float *input, *output;
extem int r_buffer, t_buffer;
extem volatile int buffer_rcvd, buffer_xmtd;

main()
{
float ref_data[2*N], dyn_data[2*Nj;
float max, sum;
int i, j, peak_location;

r_buffer = 2*N;
t_buffer = OUTSIG_LEN;

init-c3O();
init_arrays(t_buffer, r_buffer);
init_aic();

/* read in reference data */
while (!buffer_rcvd);
for (i = 0; i < 2*N; i++)
ref_data[i] = input[i];
buffer_rcvd = 0;

/* fft reference data */
fft(N, M, ref_data);

/* Compute magnitudes (squared) and store in ref_data */
for (i = 0; i < 2*N; i+=2)
ref_data[i/2) = MAG_SQ(ref_data[i], ref_data[i+l]);

while(1)
{
    /* read in dynamic data */
    while(!buffer_rcvd);
    for (i = 0; i < 2*N; i++)
       dyn_data[i] = input[I];
    buffer_rcvd = 0;

    /*fft dynamic data */
    fft(N, M, dyn_data);

    /* Compute magnitudes (squared) and store in dyn_data */
    for (i = 0; i < 2*N; i+=2)
    dyn_data[i/21 = MAG_SQ(dyn_data[i], dyn_data[i+ I]);

    /*Correlate dyn_data and ref_data, and put result in dyn_data */
    for (i = 0; i < N; i++)
    {
    sum = O.;
    for (j = 0; j < N - i; j++)
      sum + = ref_data[j] * dyn_data[j+I];
```

```
    dyn_data[I] = sum;
  {
  /*Find the location of the peak in dyn_data,
    and decide whether to accept or reject the data */
  peak_location = 0;
  max = dyn_data[0];

  for (i = 1; i < N; i++)
    if (dyn_data[I] > max) peak_location = i;

 if (peak_location < THRESHOLD)
 {
  output = continue_signal;
  buffer_xmtd = 0;
 }
 else
 {
   output = repeat_signal;
   buffer_xmtd = 0;
 {
 {
{
```

Appendix B. Radix-2 FFT Source Code

```
***********************************************************************
*Name:
* fft --- radix-2 complex FFF to be called as a C function.
*
*Synopsis:
*  int fft(N, M, data)
*  int N      FFT size: N-2**M
*  int M      Number of stages - log2(N)
*  float*data    Array with input and output data
*
*Description:
*  Generic function to do a radix-2 FFT computation on the 320C30.
*  The data array is 2*N-long, with real and imaginary values alternating.
*  The program is based on the FORTRAN program in the Burrus and Parks
*  book, p.111.
*  The computation is done in place, and the original data is destroyed.
*  Bit reversal is implemented at the end of the function.  If this is not
*  necessary, this part can be commented out.
*  The sine/cosine table for the twiddle factors is expected to be supplied
*  during link time, and it should have the following format:
*
*                  .global        _sine
*                  .data
* -sine        .float             value1         = sin(0*2*pi/N)
*                  .float         value2         = sin(I *2*pi/N)
*                  ......
*                  .float         value(5N/4)       = sin((5*N/4-I)*2*pi/N)
*
*  The values valuel, value2, etc., are the sine wave values.  For an
*  N-point FFT, there are N+N/4 values for a full and a quarter period of
*  the sine wave.  In this way, a full sine and cosine period are available
*  (superimposed).
*
*Stack structure upon the call:
*               +----------------+
*  -FP(4)   |   data  |
*  -FP(3)   |   M                    |
*  -FP(2)   |   N                    |
*  -FP(l)   |return addr |
*  -FP(0)   |  old FP  |
*               +----------------+
*
*Registers used: RO, RI, R2, R3, R4, R5, R6, R7, ARO, AR], AR2, AR4, AR5
*  AR6,AR7,IRO,IRI,RS,RE,RC
*
*AUTHOR: PANOS E. PAPAMICHALIS
*       TEXAS INSTRUMENTS                                    OCTOBER 13,1987
*
**********************************************************************
*
FP .set  AR3

         .GLOBL       _fft                          ; ENTRY POINT FOR EXECUTION
         .GLOBL       _sine                         ; ADDRESS OF SINE TABLE

         .BSS FFTSIZ,l
         .BSS LOGFFT,l
```

```
         .BSS INPUT,l

  .TEXT

SINTAB    .word _sine

;INITIALIZE C FUNCTION

_fft: PUSH            FP               ;SAVE DEDICATED REGISTERS
         LDI          SP,FP
         PUSH         R4
         PUSH         R5
         PUSHF  R6
         PUSHF  R7
         PUSH         AR4
         PUSH         AR5
         PUSH         AR6
         PUSH         AR7

         LDI          *-FP(2),R0       ;MOVE ARGUMENTS TO LOCATIONS MATCHING
         STI          R0,@FFTSIZ       ;            THE NAMES OF THE PROGRAM
         LDI          *-FP(3),R0
         STI          R0,@LOGFFT
         LDI          *-FP(4),R0
         STI          R0,@INPUT

;INITIALIZE FFT ROUTINE

         LDI       @FFTSIZ,IR1
         LSH       -2,IR1                    ;IR1=N/4,POINTER FOR SIN/COS TABLE
     LDI 0,AR6  ;AR6 HOLDS THE CURRENT STAGE NUMBER
         LDI       @FFTSIZ,IR0
         LSH       1,IR0                     ;IR0=2*N1 (BECAUSE OF REAL/IMAG)
         LDI       @FFTSIZ,R7  ;R7=N2
         LDI       1,AR7                ;INITIALIZE REPEAT COUNTER OF FIRST LOOP
     LDI  1,AR5  ;INITIALIZE IE INDEX (AR5=IE)

; OUTER LOOP

LOOP:    NOP       *++AR6(1)    ;CURRENT FFT STAGE
         LDI       @INPUT,AR0   ;AR0POINTS TO X(I)
         ADDI      R7,AR0,AR2   ;AR2 POINTS TO X(L)
      LDI AR7,RC
         SUBI      1,RC                      ;RC SHOULD BE ONE LESS THAN DESIRED#

;FIST LOOP

      RPTB  BLK1
        ADDF       *AR0,*AR2,R0    ;R0=X(I)+X(L)
        SUBF       *AR2++,*AR0++,R1   ;R1=X(I)-X(L)
        ADDF       *AR2,*AR0,R2    ;R2=Y(I)+Y(L)
        SUBF       *AR2,*AR0,R3    ;R3=Y(I)-Y(L)
        STF        R2,*AR0--       ;Y(I)=R2 AND…
‖       STF        R3,*AR2--       ;Y(L)=R3
BLK1    STF        R0,*AR0++(IR0)  ;X(I)=R0 AND…
‖       STF        R1,*AR2++(IR0)  ;X(L)=R1 AND AR0,2 = AR0,2+2*N1

;IF THIS IS THE LAST STAGE, YOU ARE DONE
```

```
        CMPI       @LOGFFT,AR6
        BZD        END

;MAIN INNER LOOP

        LDI        2,AR1                        ;INIT LOOP COUNTER FOR INNER
LOOP

        LDI        @SINTAB,AR4            ;INITIALIZE 1A INDEX (AR4=IA)
INLOP:  ADDI AR5,AR4  ;IA=IA+IE;AR4 POINTS TO COSINE
        LDI        AR1,AR0
        ADDI       2,AR1                        ;INCREMENT INNER LOOP COUNTER

        ADDI       @INPUT,AR0             ;(X(1),y(I)) POINTER
        ADDI       R7,AR0,AR2             ;(X(L),Y(L)) POINTER
    LDI  AR7,RC
        SUBI       1,RC                   ;RC SHOULD ONE LESS THAN DESIRED#
        LDF        *AR4,R6                ;R6=SIN

; SECOND LOOP

    RPTB BLK2
        SUBF       *AR2,*ARO,R2           ;R2=X(I)-X(L)
        SUBF       *+AR2,*+ARO,Rl         ;R1=Y(I)-Y(L)
        MPYF       R2,R6,RO               ;R0=R2*SIN AND…
||      ADDF       *+AR2,*+ARO,R3         ;R3=Y(I)+Y(L)
        MPYF       Rl,*+AR4(IRI),R3      ;R3=R1*COS AND…
||      STF        R3,*+ARO               ;Y(I)=Y(I)+Y(L)
        SUBF       RO,R3,R4               ;R4=R1*COS-R2*SIN
        MPYF       R1,R6,RO               ;R0=R1*SIN AND…
||      ADDF       *AR2,*ARO,R3           ;R3=X(I)+X(L)
        MPYF       R2,*+AR4(IRl),R3      ;R3=R2*COS AND…
||      STF        R3,*ARO++(IRO)         ;X(I)=X(I)+X(L) AND AR0=AR0+2*N1
        ADDF       RO,R3,R5               ;R5=R2*COS+R1*SIN
BLK2    STF        R5,*AR2++(IRO)         ;X(L)=R2*COS+R1*SIN, INCR AR2 AND…
||      STF        R4,*+AR2               ;Y(L)=R1*COS-R2*SIN

        CMPI       R7,AR1
        BNE        INLOP                        ;LOOP BACK TO THE INNER LOOP

    LSH 1,AR7      ;INCREMENT LOOP COUNTER FOR NEXT TIME

    LSH 1,AR5      ;IE=2*IE
        LDI        R7,IR0                       ;N1=N2
        LSH        -1,R7               ; N2=N2/2
        BR         LOOP                         ;NEXT FFT STAGE

;DO THE BIT-REVERSING OF THE OUTPUT

END:    LDI        @FFTSIZ,RC             ;RC=N
        SUBI       I,RC                   ;RC SHOULD BE ONE LESS THAN DESIRED#
        LDI        @FFTSIZ,IR0            ;IR0 = SIZE OF FFT=N
        LDI        @INPUT,AR0
        LDI        @INPUT,AR1

RPTB BITRV
        CMPI       AR0,AR1
        BGE        CONT
```

```
        LDF        *AR0,R0
||      LDF        *AR1,R1
        STF        R0,*AR1
||      STF        R1,*AR0
        LDF        *+AR0(1),R0
||      LDF        *+AR1(1),R1
        STF        R0,*+AR1(I)
||      STF        R1,*+AR0(I)
CONT    NOP        *++AR0(2)
BITRV   NOP        *AR1++(IR0)B

; RESTORE THE REGISTER VALUES AND RETURN

        POP        AR7
        POP        AR6
        POP        AR5
        POP        AR4
        POPF       R7
        POPF       R6
        POP        R5
        POP        R4
        POP        FP
RETS
```

Appendix C. AIC Library Source Code

```
!<arch>
aicdrvr.c/ 697924910 0     0   0   17365   '
/***********************************************************************/
/* AICDRVR.C                                       */
/*                                     */
/* TMS320C3x - AIC DRIVER                                    */
/*  :TMS320C3x CODE                                    */
/*  Compile and archive into aic.lib              */
/*
/* (C) 1991 TEXAS INSTRUMENTS, HOUSTON                          */
/***********************************************************************/
#include <math.h>
#include <stdlib.h>
#include <aic.h>

/**********************************************************************/
/* GLOBAL VARIABLES                                     */
/**********************************************************************/
int t_buffer = BLOCK_SIZE; /* SIZE OF I/O BUFFER(S)        */
int r_buffer = BLOCK_SIZE; /* SIZE OF I/O BUFFER(S)        */
VPVF output        /* OUTPUT DATA BUFFER FOR PROCESSOR        */
VPVF input;        /* INPUT DATA BUFFER FOR PROCESSOR        */
VPVF output_xfer;  /* OUTPUT DATA BUFFER FOR ISR/AIC        */
VPVF input_xfer,   /* INPUT DATA BUFFER FOR ISR/AIC        */
VI buffer_rcvd = FALSE   ; /* CPU-ISR COMM FLAG (INPUT)    */
VI buffer_xmtd = FALSE   ; /* CPU-ISR COMM FLAG (OUTPUT)    */
VI r_index = 0;    /* INDEX INTO INPUT AND OUTPUT DATA ARRAYS */
VI t_index = 0;    /* INDEX INTO INPUT AND OUTPUT DATA ARRAYS */
VI i;              /* GENERIC COUNTER VARIABLE        */

/**********************************************************************/
/* AIC CONTROL VARIABLES
        */
/**********************************************************************/
AIC_COMMAND_0      aic_command_0;  /* AIC COMMAND WORD 0 */
AIC_COMMAND_1      aic_command_1;  /* AIC COMMAND WORD 1 */
AIC_COMMAND_2      aic_command_2;  /* AIC COMMAND WORD 2 */
AIC_COMMAND_3      aic_command_3;  /* AIC COMMAND WORD 3 */
volatile AIC_PRIMARY  aic_primary;
VI secondary_transmit = OFF-, /* FLAG TO SENT SECONDARY TRANSMIT*/
VI aic_secondary = 0;    /* COMMAND SENT ON SECONDARY TRANSMIT
int ie;              /* ENABLED INTERRUPTS TEMP VARIABLE */

#ifASYNC
/**********************************************************************/
/* C_INT05() OR C_INT07()
/*    SERIAL PORT0/1 TRANSMIT INTERRUPT SERVICE ROUTINE   */
/*    1. IF SECONDARY TRANSMISSION SEND AIC COMMAND WORD*/
/*    2. OTHERWISE IF COMMAND SEND REQUESTED SETUP FOR SECONDARY */
/*       TRANSMISSION ON NEXT INTERRUPT    */
/*    3. OTHERWISE WRITE OUT OUTPUT DATA    */
/*    4. RESET SAMPLE INDEX AND FLAG IF FRAME IS FULL    */
/*       AND SWAP BUFFER POINTERS    */
/*    5. IF REAL TIME IS NOT MET GO TO ERROR HANDLER    */
/**********************************************************************/
#if SER-NUM
void c_int05(void) {}
```

```
void c_int07(void)
#else
void c_int07(void) {}
void c_int05(void)
#endif
{
VPVF swap;

if (secondary_transmit)
{
 SERIAL_PORT_ADDR(SER_NUM)->x_data = aic_secondary;
 Secondary_transmit = OFF;
 put_ie(ie);          /* RESTORE GET ENABLED INTS */
}
else
{
 if (aic_primary._bitval.command == SECONDARY_REQ)
        {
    ie = get_ie();         /* GET ENABLED INTS */
    if(SER_NUM)
       put_ie(0x40);        /* ENABLE SERIAL PORT INT */
          else
                            put_ie(0x10);
          secondary_transmit   = ON;
       }


aic_primary._bitval.data = outpuT_xfer[t_index];
SERIAL_PORT_ADDR(SER_NUM)->x_data = aic_primary._intval:

     if (++t_index == t_buffer)
     {
#ifERROR_CHECK
    if(buffer_xmtd == TRUE) error_in_real_time();
#endif
       swap    = output_xfer;
            output_xfer = output;
       output  = swap;
             t_index = 0;
             buffer_xmtd = TRUE;
           }
}
aic_primary._bitval.command = STANDARD;
}

/***********************************************************************/
/* C-INT06() OR C_INT08()                                          */
      SERIAL PORT0/1 RECEIVE INTERRUPT SERVICE ROUTINE
      1. READ INPUT DATA
      2. RESET SAMPLE INDEX AND FLAG IF FRAME IS FULL
         AND SWAP BUFFER POINTERS
      3. IF REAL TIME IS NOT MET GO TO ERROR HANDLER
/***********************************************************************/
#if SER_NUM
void c_int06(void) {}
void c_int08(void)
#else
void c_int08(void) {}
void c_int06(void)
```

```
#endif
{
  VPVF swap;

  aic_primary._intval = SERIAL_PORT_ADDR(SER_NUM)->r_data & 0x0FFFF;
  input_xfer[r_index] = aic_primary._bitval.data;

  if (++r_index == r_buffer)
  {
#ifERROR_CHECK
    if(buffer_rcvd == TRUE) error_in_real_time()
#endif

    swap      = input;
          input                   = input_xfer;
    input_xfer = swap;
          r_index = 0
          buffer_rcvd = TRUE;
  }
}
#else /* IF NOT ASYNC */

/***********************************************************************/
/*C_INT05() OR C_Int07()                         /*
/*   SERIAL PORT 0/1 TRANSMIT AND RECEIVE INTERRUPT SERVICE ROUTINE/*
/*   1. IF SECONDARY TRANSMISSION SEND AIC COMMAND WORD  /*
/*   2. OTHERWISE IF COMMAND SEND REQUESTED SETUP FOR SECONDARY/*
/*      TRANSMISSION ON NEXT INTERRUPT AND RECEIVE DATA  /*
/*   3. OTHERWISE WRIT7E OUT OUTPUT DATA AND RECEIVE INPUT DATA/*
/*   4. RESET SAMPLE INDEX AND FLAG IF FRAME IS FULL  /*
/*      AND SWAP BUFFER POINTERS         /*
/*   5. IF REAL TIME IS NOT MET GO TO ERROR HANDLER   /*
/***********************************************************************/
#if SER_NUM
void c_int05(void) {}
void c_int06(void) {}
void c_int08(void) {}
void c_int07(void) {}
#else
void c_int06(void) {}
void c_int07(void) {}
void c_int08(void) {}
void c_int05(void)
#endif
{
 VPVF swap;

if (secondary_transmit)
{
   SERIAL_PORT_ADDR(SER_NUM)->x_data = aic_secondary;
   secondary_transmit = OFF;
   put_ie(ie);         /* RESTORE GET ENABLED INTS
}
else
}
 if (aic_primary._bitval.command == SECONDARY_REQ)
        {
    ie = get_ie();         /*GET ENABLED INTS */
    if(SER_NUM)
```

```
       put_ie(0x40);      /* ENABLE SERIAL PORT INT */
          else
                         put_ie(0x10);
          secondary_transmit = ON;
       }

  aic_primary._bitval.data = output_xfer[t_index];
  SERIAL_PORT_ADDR(SER_NUM)->x_data = aic_primary._intval;

  aic_primary._intval = SERIAL_PORT_ADDR(SER_NUM)->R_data & 0x0FFFF;
  input_xfer[r_index] = aic_primary._bitval.data;

        if (++r_index == r_buffer)
        {
#if ERROR_CHECK
       if(buffer_rcvd == TRUE) error_in_real_time();
#endif
       swap      = input;
            input               = input_xfer;
       input_xfer = swap;
            r_index = 0;
            buffer_rcvd = TRUE;
         }
         if (++t_index == t_buffer)
         {
#ifERROR_CHECK
     if(buffer_xmtd == TRUE) error_in_real_time()
#endif
       swap    = output_xfer;
           output_xfer = output;
       output  = swap;
             t_index = 0;
             buffer_xmtd = TRUE;
          }
  }
  aic_primary._bitval.command = STANDARD:
}
#endif

/*********************************************************************/
/* INIT_ARRAYS(): INITIALIZE DATA ARRAY PARAMETERS      /*
/*********************************************************************/
void init_arrays(int t_buffer, int r_buffer)
{
  int i;
  /* --------------------------------------------------- */
  /* INITIALIZE AND ZERO FILL ARRAYS
  /* --------------------------------------------------- */
  if(!(input  = (float *) malloc(r_buffer))) heap_overflow();
  if(!(output  = (float *) malloc(r_buffer))) heap_overflow();
  if(!(input_xfer  = (float *) malloc(r_buffer))) heap_overflow();
  if(!(output_xfer  = (float *) malloc(r_buffer))) heap_overflow();
  for(i = 0; i < t_buffer; i++) output[i] = output_xfer[i] = 0.0;
}

/*********************************************************************/
/* INIT_AIC(): INMALIZE COMMUNICATIONS TO AIC           */
/*      NOTE: i IS A VOLAIILE TO FORCE TIME DELAYS AND TO FORCE */
/*       READS OF SERIAL PORT DATA RECEIVE REGISTER TO CLEAR */
```

```
/*       THE RECEIVE INTERRUPT FLAG              */
/***********************************************************************/
void init_aic(void)
{
  RESET_AIC;                       /* RESET AIC */
  WAIT(50);         /* KEEP RESET LOW FOR SOME PERIOD OF TIME */

  /* --------------------------------------------- /*
  /* SET AIC CONFIGURATION CHIP
  /* 1. ALLOW 8 KhZ SAMPLING RATE AND 3.6 KHZ ANTIALIASING FILTER */
  /*   GIVEN A 7.5 MHZ MCLK TO THE AIC FROM A 30 MHZ TMS320C30    */
  /* 2. ENABLE A/D HIGHPASS FILTER              */
  /*     3. SET SYNCHRONOUS TRANSMIT AND RECEIVE IF NOT ASYNC              */
  /* 4. ENABLE SINX/X D/A CORRECTION FILTER                  */
  /* 5. SET AIC FOR +/- 1.5 V INPUT         */
  /* --------------------------------------------- */
  aic_primary._bitval.command = STANDARD;
  aic_primary._bitval.data = 0;
  aic_primary._bitval.unused = 0;

#if TLC32046
  aic_command_0.command = 0;  /* SETUP AIC COMMAND WORD ZERO */
  aic_command_0.ra  = 26; /* ADJUST SAMPLING RATE TO 8 kHz */
  aic_command_0.ta  = 26;  /* AND 3.6 kHz ANTIALIAS FILTER */

  aic_command_1.command = 1; /* SETUP DEFAULT AIC COMMAND WORD 1 */
  aic_command_1.ra_prime = 1;
  aic_command_1.ta_prime = 1;
  aic_command_1.d_f = 0;

  aic_command_2.command = 2;  /* SETUP DEFAULT AIC COMMAND WORD 2 */
  aic_command_2.rb            = 18;
  aic_command_2.tb              = 18;

  aic_command_3.command = 3;
  aic_command_3.highpass = ON; /* TURN ON INPUT HIGHPASS FILTER     */
  aic_command_3.loopback = OFF; /* DISABLE AIC LOOPBACK    */
  aic_command_3.aux = OFF; /* DISABLE AUX INPUT           */
#ifASYNC
  aic_command_3.sync        = OFF; /* DISABLE SYNCHRONOUS A/D AND D/A */
#else
  aic_command_3.sync        = ON; /* ENABLE SYNCHRONOUS A/D AND D/A */
#endif
  aic_command_3.gain = THREE_V; /* SET FOR LINE-LEVEL INPUT     */
  aic_command_3.d_8       = 0;
  aic_command_3.sinx      = ON; /* ENABLE SIN x/x CORRECTION FILTER */
  aic_command_3.dl0out  =OFF; /*DISABLED10OUT(TEL-I/F MODE)   */
  aic_command_3.dllout  =OFF; /*DISABLED11OUT(TEL-I/F MODE)   */
  aic_command_3.d_cdef = 0;
#else
  aic_command_0.command = 0;  /* SETUP AIC COMMAND WORD ZERO */
  aic_command_0.ra         = 13; /* ADJUST SAMPLING RATE TO 8 kHz   */
  aic_command_0.ta          = 13;  /* AND 3.6 kHz ANTIALIAS FILTER */

  aic_command_1.command = 1; /* SETUP DEFAULT AIC COMMAND WORD I */
  aic_command_1.ra_prime = 1;
  aic_command_1.ta_prime = 1;
  aic_command_1.d_f    = 0;
```

```
  aic_command_2.command = 2;     /* SETUP DEFAULT AIC COMMAND WORD 2 */
  aic_command_2.rb             = 36;
  aic_command_2.tb             = 36;

  aic_command_3.command = 3;
  aic_command_3.highpass = ON; /* TURN ON INPUT HIGHPASS FILTER        */
  aic_command_3.loopback = OFF; /* DISABLE AIC LOOPBACK          */
  aic_command_3.aux    = OFF; /* DISABLE AUX INPUT    */
#ifASYNC
  aic_command_3.sync        = OFF; /* DISABLE SYNCHRONOUS A/D AND D/A  */
#else
  aic_command_3.sync        = ON; /* ENABLE SYNCHRONOUS A/D AND D/A   */
#endif
  aic_command_3.gain  = LINE_V; /* SET FOR LINE-LEVEL INPUT
  aic_command_3.d_8        = 0;
  aic_command_3.sinx       = ON; /* ENABLE SIN x/x CORRECTION FILTER
  aic_command_3.d_abcdef  = 0;
#endif

#if MSTR-CLOCK
  /* ------------------------------------------------ */
  /* CONFIGURE TIMER 0 TO ACT AS AIC MCLK        */
  /* THE TIMER IS CONFIGURED IN THE FOLLOWING WAY         */
  /*      1. THE TIMER’S VALUE DRIVES AN ACTIVE-HIGH TCLK 0 PIN */
  /*      2. THE TIMER IS RESET AND ENABLED           */
  /*      3. THE TIMER’S IS IN PULSE MODE      */
  /*      4. THE TIMER HAS A PERIOD OF TWO INSTRUCTION CYCLES     */
  /* ------------------------------------------------ */
  TIMER_ADDR(TIMER_NUM)->period = 1;
  TIMER_ADDR(TIMER_NUM)->gcontrol = FUNC|HLD|GO |CLKSRC;
#endif
  /* --------------------------------------- */
  /* CONFIGURE SERIAL PORT 0
  /*      1. EXTERNAL FSX, FSR, CLKX, CLKR        */
  /*      2. VARIABLE DATA RATE TRANSMIT AND RECEIVE           */
  /*      3. HANDSHAKE DISABLED            */
  /*      4. ACTIVE HIGH DATA AND CLK                  */
  /*      5. ACTIVE LOW FSX,FSR                 */
  /*      6. 16 BIT TRANSMIT AND RECEIVE WORD       */
  /*      7. TRANSMIT INTERRUPT        */
  /*      8. RECEIVE INTERRUPT ENABLED/RECEIVE          */
  /*      9. FSX, FSR, CLKX, CLKR, DX, DR CONFIGURED AS SERIAL */
  /*         PORT PINS         /*
/* --------------------------------------- */
  SERIAL_PORT_ADDR(SER_NUM)->gcontrol - 0x0;

  SERIAL_PORT_ADDR(SER_NUM)->s_x_control = CLKXFUNC | DXFUNC | FSXFUNC;
  SERIAL_PORT_ADDR(SER_NUM)->s_r_control = CLKRFUNC | DRFUNC | FSRFUNC;

  SERIAL_PORT_ADDR(SER_NUM)->gcontrol = XVAREN | RVAREN | FSXP | FSRP |
               XLEN_16 | RLEN_16 | XINT | RINT |
               RRESET | XRESET;

  /* CLEAR SERIAL TRANSMIT DATA */
  SERIAL_PORT_ADDR(SER_NUM)->x_data = 0x0;

UN_RESET_AIC;
CL_INT_FL_REG;
```

```
#if SER_NUM
  EN_SER_PORT_XMT_INT_1;
#else
  EN_SER_PORT_XMT_INT_0;
#endif

#ifASYNC
#if SER_NUM
  EN_SER_PORT_RCV_INT_1;
#else
  EN_SER_PORT_RCV_INT_0;
#endif
#endif

  EN_GLOBAL_INTS;               /* SET GLOBAL INTERRUPT ENABLE BIT         */
  /* -------------------------------------------------- */
  /* MODIFY AIC CONFIGURATION                                 */
  /* -------------------------------------------------- */
  configure_aic(*((int *) &aic_command_0));
  configure_aic(*((int *) &aic_command_3));
}

/***********************************************************************/
/* CONFIGURE_AICO: INITIATE AIC CONFIGURATION WORD TRANSMISSION ON NEXT */
INTERRUPT AFTER ALL PREVIOUS COMMANDS ARE SENT        */
/***********************************************************************/
void configure_aic(int i)
{
 while((aic_primary._bitval.command==SECONDARY_REQ) || secondary_transmit);
 aic_secondary = i;
 aic_primary._bitval.command = SECONDARY_REQ;
}

/***********************************************************************/
/* GET_IE(): read ie register          */
/***********************************************************************/
int get_ie(void)
{
 asm(" LDI IE,R0");
}
/***********************************************************************/
/* PUT_IE(): write ie register           */
/***********************************************************************/
void put_ie(int ie_value)

#if_REGPARM
  asm(" LDI AR2,IE");
#else
  asm(" LDI *-FP(2),IE");
#endif /* _REGPARM
}
```

References

[1] R. L. Ehman and J. F. Felmlee, "Adaptive Technique for High-Definition MR Imaging of Moving Structures," *Radiology,* vol. 173, no. 1, pp. 255-263, 1989.
[2] *Texas Instruments TMS32OC3x User's Guide*, p.11-125, 1994.