Application Report
SLAA130 –JUNE 2001

Interfacing the ADS7842 ADC to the TMS320C5400™ and TMS320C6000™ DSP Platforms

Robert Finger Texas Instruments, Inc.

ABSTRACT

This report describes various methods for connecting the ADS7842 to Texas Instruments TMS320C5400™ DSP and TMS320C6000™ DSP platforms. The ADS7842 is a 4-channel, 12-bit analog-to-digital converter (ADC) with a maximum sampling rate of 200 kHz.

Contents

Figures

1 Introduction

This report describes how to connect the ADS7842 data converter to Texas Instruments DSP devices of the C5000™ DSP and C6000™ DSP families. Depending on the application, some glue logic might be needed to make use of all the features of the ADC. Several application-specific methods are described. Software examples show how the converter can be controlled and read by the DSP.

2 ADS7842 ADC

The ADS7842 is a complete, 4-channel, 12-bit analog-to-digital converter (ADC) with a maximum sample rate of 200 kHz. It contains a 12-bit, capacitor-based, SAR A/D with a sample-and-hold amplifier, an interface for microprocessors and parallel, 3-state output drivers. The reference voltage can be varied from 100 mV to V_{CC} with a corresponding LSB resolution from 24 mV to 1.22 mV. The ADS7842 is assured down to 2.7-V operation.

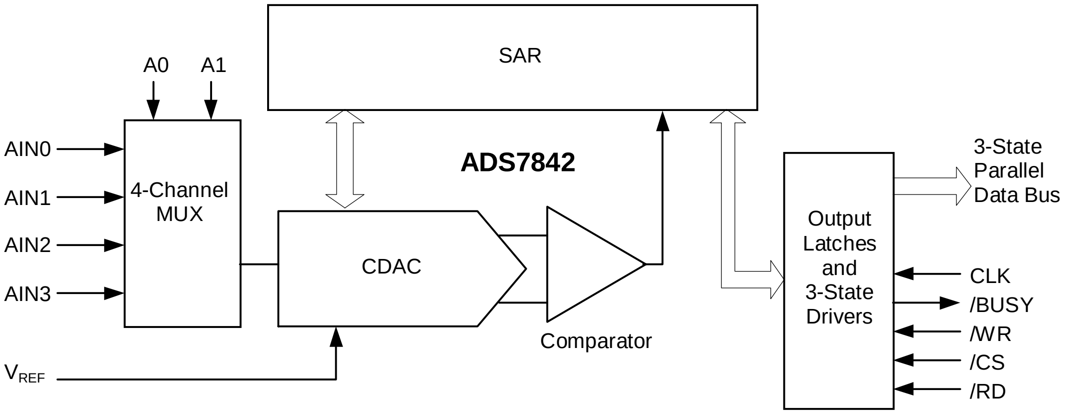

Figure 1. ADS7842 Block Diagram

The ADS7842 requires an external clock to run the conversion process. This clock can vary between 200 kHz (12 kHz throughput) and 3.2 MHz (200 kHz throughput). A conversion can be initiated by bringing the /WR pin LOW for a minimum of 25 ns. /BUSY will go LOW 20 ns after the falling edge of the /WR pin and return HIGH just after the ADS7842 has finished a conversion. The conversion is initiated on the falling edge of the /WR input, with valid signals on A0, A1, and /CS. The ADS7842 will enter the conversion mode on the first rising edge of the external clock following the /WR pin going LOW. The conversion will start on the first clock cycle. The MSB will be approximated by the capacitive digital-to-analog converter (CDAC) on the first clock cycle, the 2nd MSB on the second cycle, and so on until the LSB has been decided on the 12th clock cycle.

After /BUSY goes HIGH, the /CS and /RD pins may be brought LOW to enable the 12-bit output bus. /CS and /RD must be held LOW for at least 25 ns following the rising edge of /BUSY. Data will be valid 25 ns after the falling edge of both /CS and /RD and remain valid for 25 ns following the rising edge of /CS and /RD. Both have to stay low for at least 25 ns.

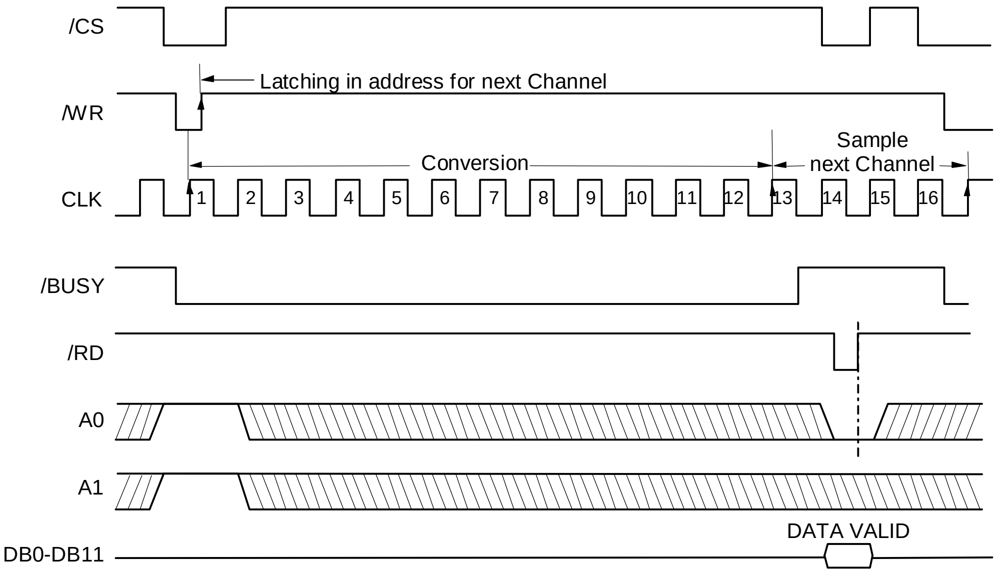

Figure 2. Normal Operation of the ADS7842

Channel selection of the four analog inputs is done via the A0 and A1 address inputs. These are latched on the rising edge of /WR. This selection does not choose the channel for the current conversion (started with the rising clock edge during the low-phase of /WR), but for the next conversion. A0 has to be low on the rising edge of /RD. Otherwise the device will enter power-down mode. The analog inputs are sampled during the sample phase in cycle 13 to 16. The internal sample and hold capacitor is loaded during this time.

3 How to Connect the ADS7842 to the TMS320C6000 DSP

3.1 Glueless Interface; External Memory Interface (EMIF) Only

This section describes a simple way to connect the ADS7842 to a C6000 DSP. In this configuration, the asynchronous memory interface of the C6000 DSP is used. If the ADS7842 is the only device connected to a certain CE space, no external address decode logic is needed.

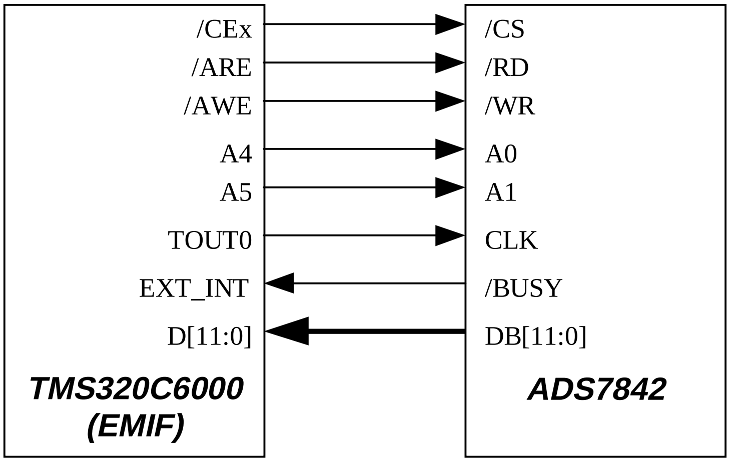

Figure 3. Direct C6000 DSP/ADS7842 Interface

In this example, timer 0 of the DSP generates the conversion clock. It could also be generated by an external clock source. A4 and A5 of the DSP's address lines are used for channel selection, as there are no A0 and A1 lines on the C6000 DSP. These lines are internally decoded to byte enable signals.

The DSP has to write to the data converter in order to start a conversion. At conversion start, /BUSY will go low and back high when the conversion is finished. This rising edge is recognized by the DSP as external interrupt. In the interrupt service routine, the DSP now reads one value from the ADC. To prevent the ADS7842 from going into power down mode, A4 (A0 on the ADS7842) has to be low during the read.

As the write can occur at any time and has no direct relation to the conversion clock, it has to be certain that during the low phase of /AWE, a low to high transition of the clock takes place. Otherwise the conversion might not start. A look in the data sheet of the ADS7842 shows, that /WR has to stay low 25 ns (t_{WR_hld}) after the rising edge of the clock. On the C6000 DSP this can easily be achieved. The low time of /AWE (strobe) can be programmed via the CE space control registers of the EMIF. The length of the strobe has to be longer than the clock cycle time of the conversion clock (t_{CLK}) plus 25 ns.

(1) $t_{strobe} \geq t_{CLK} + t_{WR\_hld}$

(2) $t_{strobe} = STROBE \cdot t_{CLK\_EMIF}$

The EMIF CE space control register of the C6000 DSP provides 6 bits for programming the strobe phase (STROBE in equation *(2)*). This value represents the number of clock cycles that /AWE stays low. Clock cycles are in terms of CPU clock for the C6x0x DSP and ECLKOUT cycles for the C6x1x DSP. With the 6 bits, the strobe can be 64 clock cycles maximum. For a C6x1x DSP with an external memory clock of 100 MHz (t_{CLK_EMIF} = 10ns), this leads to 64 · 10 ns = 640 ns. With equation *(1)* we get a maximum clock cycle time of 615 ns (1.62 MHz). This means the clock input of the ADS7842 has to run with at least 1.62 MHz.

On a 200 MHz C6x0x DSP, the resulting minimum input clock frequency would be 3.39 MHz (64 · 5 ns). This exceeds the specified maximum clock frequency of 3.33 MHz for the ADS7842. Smaller values for the strobe phase can be used, if it is certain that a rising edge will occur during the low phase of /WR. Connecting the conversion clock to a general-purpose input pin of the C6000 DSP and checking the status of the pin via software might do this. The program has to wait until CLK goes low and initiates the write command. In this case the programmer has to know the exact number of cycles it takes from detection of the falling clock-edge to the write command.

There is no special output enable signal on the ADS7842. The /CS and /RD signals are internally decoded and the outputs of the ADC will only be active, when both control signals are low. Therefore, no bus contention will occur if /ARE goes low due to an access to any other asynchronous device. Address decode logic would be necessary, if other devices are connected to the same CE space.

If a continuous conversion is needed, this configuration has one drawback. A new conversion has to be started periodically, and always at the same time-difference. The DSP starts the conversion by writing to the ADS7842. If the DSP is busy and can not start a conversion at a certain moment of time, the conversion starts later and jitter will occur. To avoid this, the scheme described in Section 3.2 can be used. But there, an external clock signal or additional timer of the DSP has to be used to provide the data converter clock.

3.2 Timer Used for /WR Generation

The timer of the C6000 DSP is very flexible and can be used to generate the desired timing behavior for the /WR signal. An external clock source, an additional timer of the DSP, or a clock signal from one of the serial ports is needed as input for the timer.

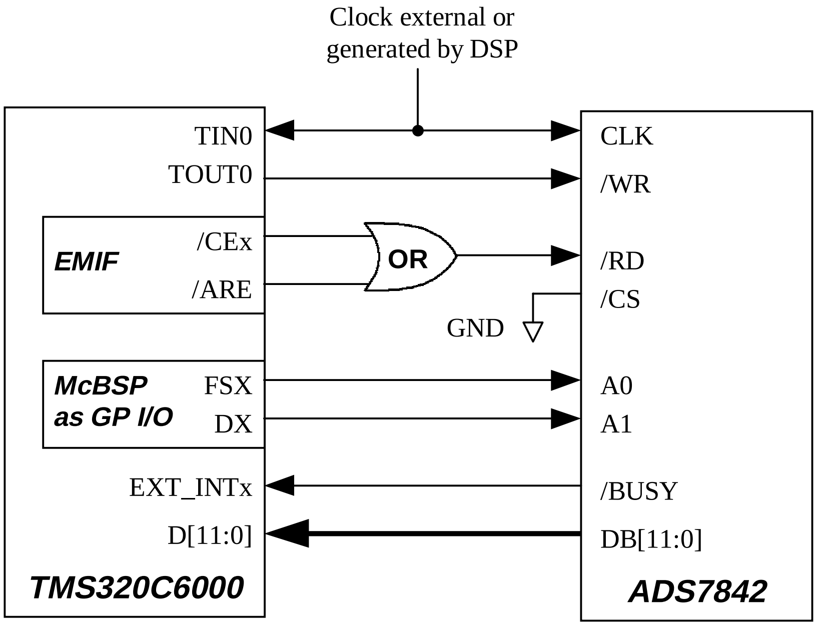

Figure 4. Timer Used for /WR Generation

Two pins of an unused serial port are used as general-purpose output pins in this example schematic. They control the address lines of the ADC to select the desired analog channel. If only one analog channel is needed, both address lines could be tied to ground. In the example, the ADS7842 is the only device connected to /CEx. Additional address decode logic has to be used, if other devices are also connected to the same CE-space.

The C6000 DSP timers are programmed via 2 registers, a third register holds the current count value:

Table 1. C6000 DSP Timer Registers

Name	Description
Timer control register	Determines the operating mode of the timer, monitors the timer status, and controls the function of the TOUT pin.
Timer period register	Contains the number of timer input clock cycles to count. This number controls the signal frequency
Timer counter register	Current value of the incrementing counter.

The data converter clock is used as the clock source for the timer. A low phase of /WR is needed every 16 clock cycles. Therefore, the timer period register will be set to 16 in order to achieve the maximum sample rate. Bigger values can also be used, if a slower sample rate is needed.

A detailed description of the timer control register can be found in the *TMS320C6000 Peripherals Reference Guide*. It has to be programmed to work in pulse mode and to invert the timer input and the timer output in order to generate the desired /WR signal timing. The following code segment shows an example of how to setup timer 0 for the desired operating mode:

```
#define TIMER0_PRD          *(unsigned volatile int *)0x1940004

#define TIMER0_CTRL         *(unsigned volatile int *)0x1940000

#define TIMER0_COUNT        *(unsigned volatile int *)0x1940008

...

    TIMER0_CTRL   = 0x00000403;   /* TOUT as timer output; Timer operates in pulse */
                                  /* mode; external clock source; Inverted TINP    */
                                  /* drives timer; Inverted TSTAT drives TOUT;     */
                                  /* Timer halted                                  */

    TIMER0_COUNT  = 0;            /* set count register to zero                    */

    TIMER0_PRD    = 16;           /* period register=16 (divide input clock by 16) */

    TIMER0_CTRL  |= 0xc0;         /* start timer 0                                 */

...
```

Section 5.1 shows a complete example program for this configuration. The program is written for a C6211/C6711 DSP with the ADS7842 connected to CE space /CE2. External interrupt 4 is used as input for the /BUSY signal. The analog input channel selection is done via the FSX and DX pins of McBSP 1. These pins are configured as general-purpose output pins. The resulting signal timing of this example program is shown in Figure 5.

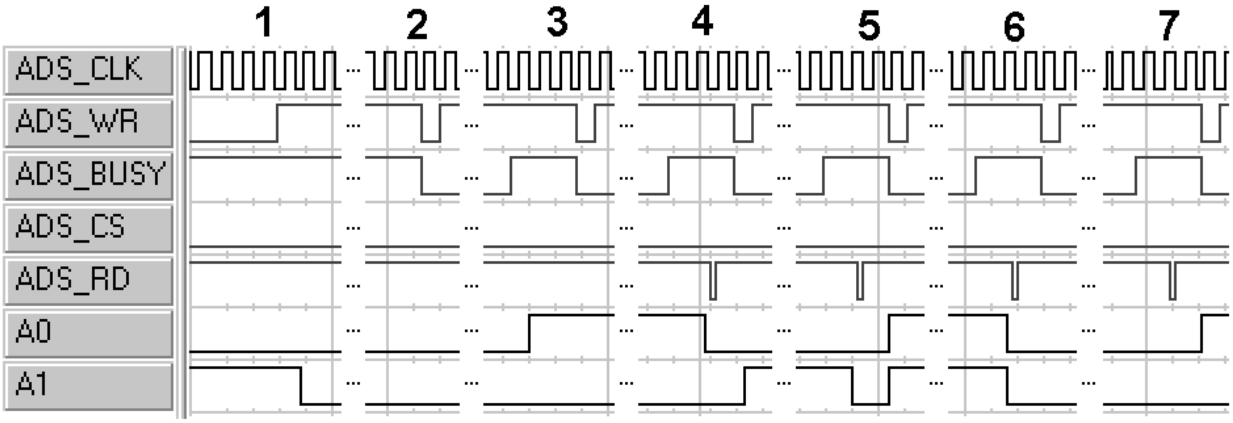

Figure 5. Signal Timing; /WR Generated by Timer Output

Phase 1: The DSP outputs are in their reset state at the beginning. The timer output (ADS_WR) will go high and both general-purpose McBSP outputs will be set to low after initialization. The timer is started and begins to work. The DSP now waits for the first rising edge of /BUSY.

Phase 2: The timer internally counts 16 clock cycles. After that, /WR goes low for one cycle. This will start the first conversion. Which analog input channel is sampled is not defined, as this is the first conversion. /BUSY goes low with the falling edge of /WR. The address for next channel (channel 0) is latched with the rising edge of /WR. The program is still waiting for the first rising edge of /BUSY.

Phase 3: The first conversion is finished, /BUSY goes high. This rising edge sets the interrupt flag of the DSP, and the program detects this flag and continues. No value is read from the ADC, as the first conversion is only a dummy conversion. The address of the next analog channel (channel 1) is set and interrupts will now be enabled. The second /WR low pulse starts the next conversion (now for channel 0) and the new address (1) is latched.

Phase 4: The rising edge of /BUSY will now initiate an interrupt. First, both ADS7842 address lines are set to zero in the interrupt service routine in order to prevent the ADC from entering power-down mode. Afterwards the first value is read (channel 0) and the ADC address lines are set to the next channel number (2).

Phase 5: The same as in phase 4 will happen for the next interrupts. After each read, the address lines are set for the next channel.

The interrupt service routine will set a flag when a complete block of data has been sampled. This signals the main program that the block contains valid data. In a real-time application, different blocks of data would be used. One block of data would be filled with sampled data. At the same time, the CPU could work on another block with previously sampled values. These implementations will always be application specific.

3.3 EMIF and One Multichannel Buffered Serial Port (McBSP) Transfer Channel Used

This section describes an interface, in which the McBSP generates the clock and conversion start signal for the data converter. Figure 6 shows the schematic.

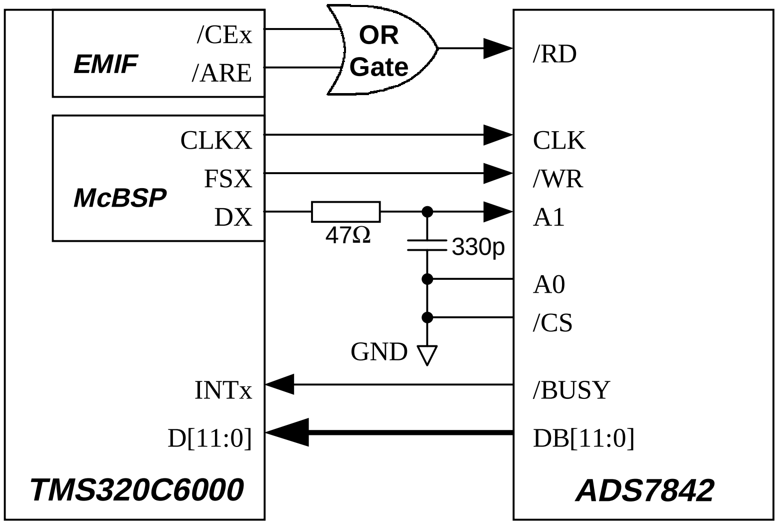

Figure 6. C6000 DSP/ADS7842 Interface With Signals Generated by McBSP

A look at the timing diagram of the ADS7842 reveals that this device needs a /WR low pulse every 16 clock cycles to operate at maximum conversion rate. The McBSP of the C6000 DSP can easily generate both signals, clock and /WR. For the /WR signal, frame sync of a transfer channel (FSX) can be used and programmed to stay high for 15 clock cycles and afterwards go low for one clock cycle. A new conversion will be started every 16 clock cycles, without the need to interrupt the DSP. Unfortunately, the DSP now can not select an analog input channel and only one analog input can be used. But with a simple trick, it is also possible to automatically switch between two channels; the data output signal DX can be used for channel selection.

The McBSP is configured to transfer 32-bit words, but the frame sync period will only be set to 16. This generates two frame sync signals every 32 bits. If the XFIG bit (transmit frame ignore) in the transmit control register (XCR) is set to 1, the second frame sync after at bit 16 will be ignored and a full 32-bit word will be transferred. The address bits have to be valid at the rising edge of /WR. This edge is exactly *between two bits* of the serial output. The output might go to a high impedance state at that moment. Therefore an R-C combination is used to keep the input level at A1 between the bits at a constant level. A0 is tied to low in order to prevent the device from going into power-down mode during reads. Analog inputs 1 and 3 will be sampled priodically in this configuration. The resulting signal timing is shown in Figure 7.

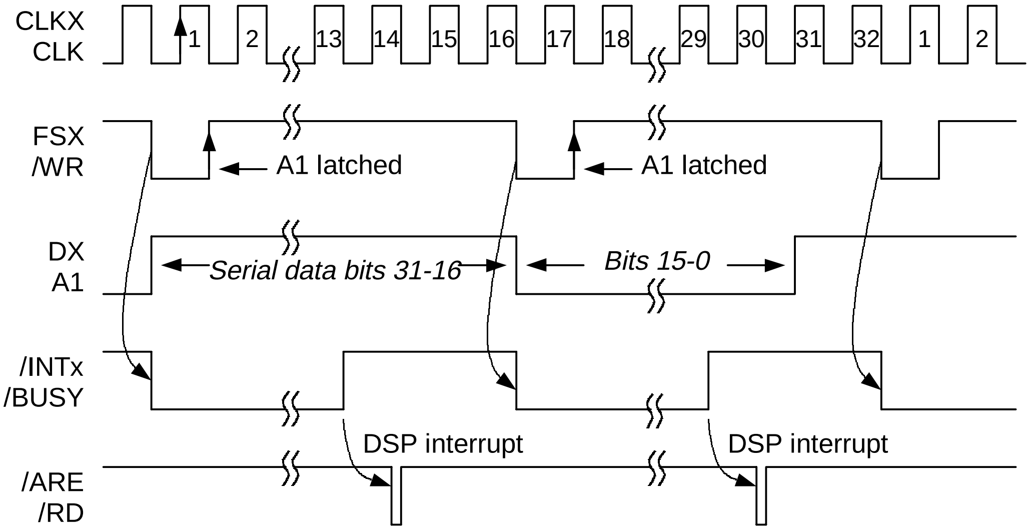

Figure 7. McBSP Timing

The whole period lasts 32 clock cycles and one bit per clock cycle will be transferred on the DX line. To get the desired bit pattern, a value of 0xFFFF0000 has to be written once into the data transmit register of the McBSP. The serial port will transfer this value continuously. The rising edge of /BUSY is used to generate the DSP interrupt. Either an interrupt service routine will be executed, in which one value is read from the ADC, or the /INTx signal might be used to start a DMA transfer of one word.

Section 5.2 shows in an example program for how the McBSP must be configured. It uses serial port one. The CLKX frequency, which must be 16 times the desired sampling rate, is programmed via the clock division factor CLKGDV in the sample rate generator register. The value depends on the DSP frequency and the DSP type used. The CPU clock will be used as source for the divider (f_{McBSP_clock}) on the C6201 DSP and half the CPU clock on the C6211 DSP.

(3) $CLKGDV = f_{McBSP\_clock} / (f_{sample} \cdot 16)$

Examle:

150 MHz C6211 → f_{McBSP_clock} = 75 MHz
desired ADS7842 sample rate (f_{sample}): 100 kHz

CLKGDV = 75 000 000 / (100 000 · 16) = 46.875

As fractional numbers are not allowed, a value of 46 or 47 must be used. Exactly 100 kHz can not be achieved this way. The following values are possible:

CLKGDV = 46 → f_{sample} = 101 902 Hz

CLKGDV = 47 → f_{sample} = 99 734 Hz

The software example in Section 5.2 uses the function provided by the chip support library (CSL). A global #define called SP1_CLKGDV is used in the example to set the clock divider of the serial port.

3.4 Parallel Interface With External Control Logic

There are various ways that external logic could control the ADS7842. The logic would have to implement a counter, which starts a conversion every 16 clock cycles and periodically switches between the channels. One problem is that the DSP does not know which channel is sampled at a certain moment of time. General-purpose I/O-pins of the DSP could be used by the control logic to indicate to the DSP what channel is currently available. As the ADS7842 only uses 12 data bits, unused data bits could also be used for transmitting channel information. In this case, the control logic would have to drive these data pins at every read that goes to the ADS7842.

One possible solution is shown in Figure 8:

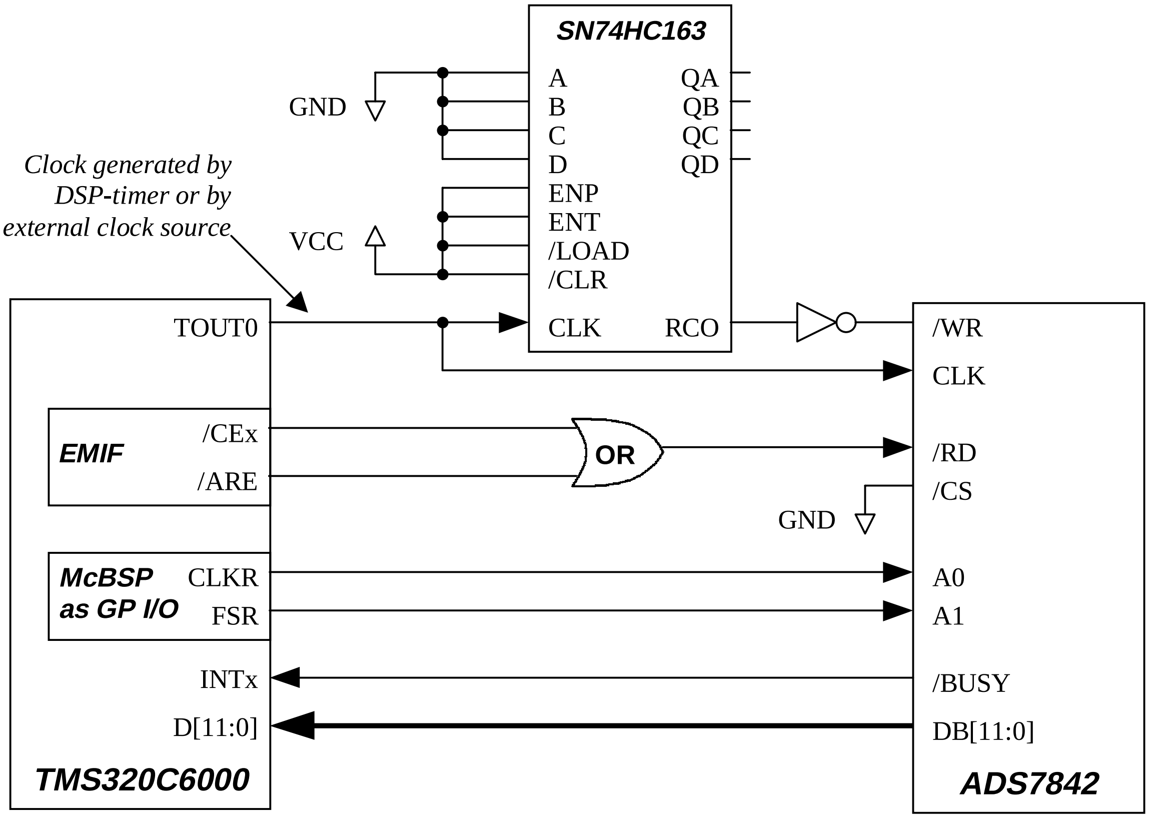

Figure 8. C6000 DSP Parallel Interface With External Control Logic

In this example, an SN74HC163 counter is used to generate the /WR signal for the ADS7842. The RCO output goes high every 16 clock cycles and stays high for one cycle. With the inverter, this signal can be used for the /WR to start a new conversion. Two general-purpose output pins of the C6000 DSP must be used for the ADS7842 address lines, if all four channels of the ADS7842 are needed. Two pins of the serial port have been used in the example. Alternatively, additional glue logic could be used to automatically switch between channels after each conversion. The timing looks quite similar to Figure 7.

When the ADS7842 is finished with a conversion, it will initiate a DSP interrupt. At the first step in the interrupt service routine, the general-purpose pin for A0 has to be set to low. Otherwise the ADC would enter power-down mode after the read. Now the DSP reads one value from the ADC. After the read, the general-purpose outputs pins must be set to make the channel selection for the conversion following the next conversion. If only one channel is used, both address lines of the ADS7842 can be tied to ground and no general-purpose output pin is needed.

4 How to Connect the ADS7842 to the TMS320C5000 DSP

4.1 Direct Connection Between DSP and ADC

A direct connection between the ADS7842 and a C5000 DSP is not as easy as with the C6000 DSP. The timer output pin of the C54x can not be used as clock source for the ADC, as the C54x timer output only operates in pulse mode. The pulse is too short for the ADS7842, which requires a low phase of at least 150 ns. A clock signal generated by the serial port could still be used.

The timing parameters of the C54x external bus are programmable only in a limited range. For example, on a C5402 DSP the maximum programmable value for the software wait states is 14. Wait states are programmable in terms of CPU clock cycles. This leads to a 140 ns (14 · 10 ns) low phase on a 100 MHz DSP (cycle time 10 ns). But this is not enough for an ADS7842 operating at a sample rate of 200 kHz. For a 200 kHz sample rate, the ADC clock has to run at 3.2 MHz (cycle time 313 ns). This would require a low phase of 338 ns (313 ns + 15 ns hold time) to securely start a conversion (see also Section 3.1 for a detailed description).

One way to safely start a conversion, is to use a general-purpose output pin of the C54x to control the /WR input of the ADC. The clock signal is fed back to the DSP via a general-purpose input pin. Figure 9 shows the schematics, the software flow to start a conversion is shown in Figure 10.

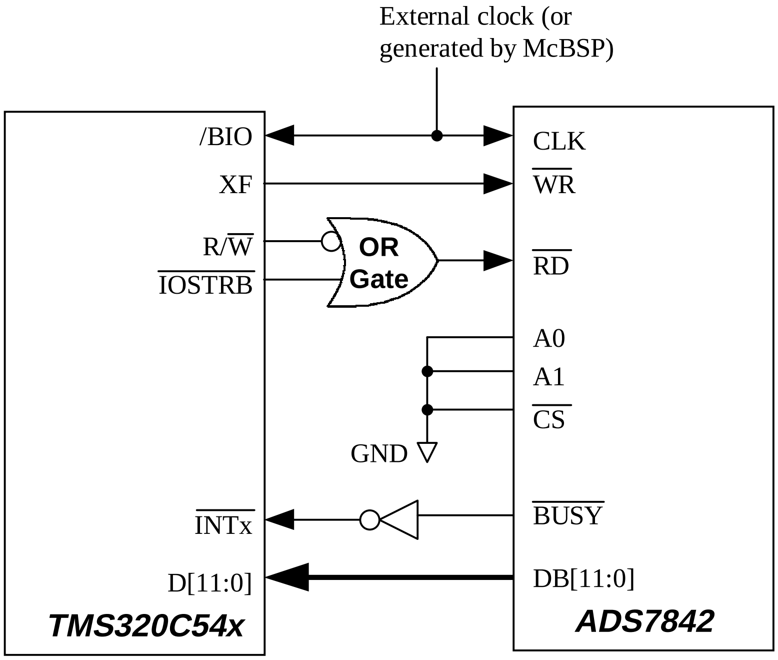

Figure 9. Direct Connection Between TMS320C54x™ and ADS7842

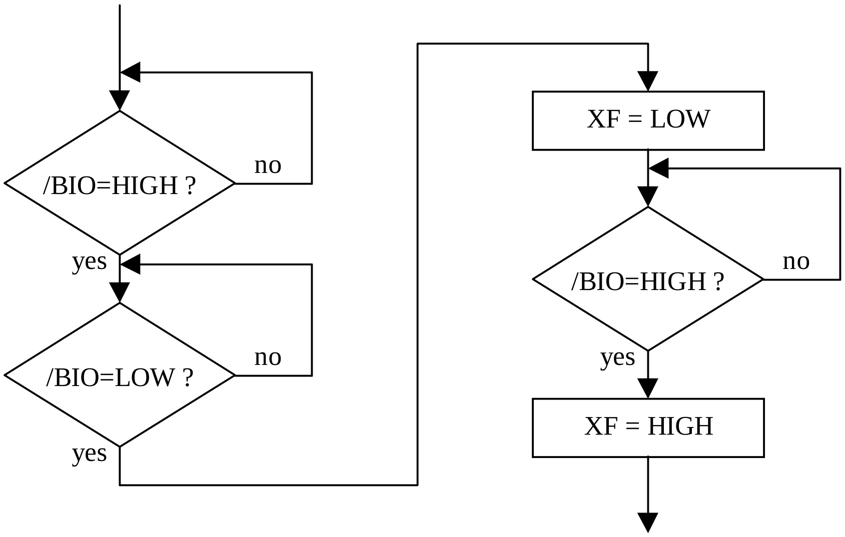

Figure 10. Software Flow for /WR Controlled by XF-pin

The program first waits for /BIO to be high and then low. This makes certain that there is enough time to set /WR low before the next rising edge of the conversion clock. After /WR has been set low, the program waits for the rising edge of the clock. Afterwards it sets /WR back to high.

4.2 I/O-Port and One McBSP Transfer Channel Used

The signals of the McBSP can be used to generate the clock and the conversion start signal, just as described in Section 3.3 for the C6000 DSP. The following example refers to devices that feature a 'multichannel buffered serial port' (McBSP). On devices which only have a BSP (buffered serial port) instead of a McBSP, this scheme could also be used, but without the option to make a channel selection between two channels. The BSP only allows transferring 16-bit values. Devices with a standard serial port (SP) would need external logic. On this device frame sync can not be programmed to be low active, but this is needed. Figure 11 shows the schematic for a C5000 DSP with a McBSP.

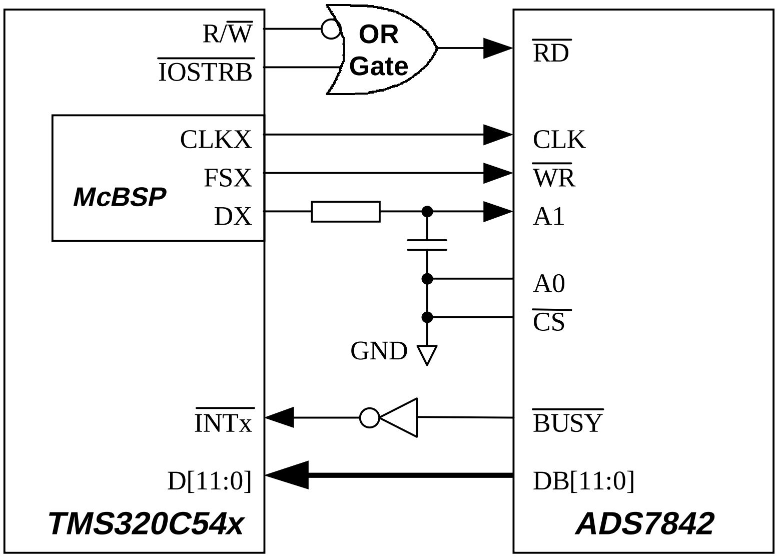

Figure 11. C54x™ DSP/ADS7842 Interface With Control Signals Generated by McBSP

For a detailed description how the McBSP has to be configured to generate the necessary signals see Section 3.3. In the example, the ADS7842 is connected to the I/O-space of the C54x. Additional address decode logic will be required if other devices are also connected to the I/O-space. Interrupts on the C54x are low active and react on a high to low transition on the interrupt pin. The polarity is not programmable. An inverter between /BUSY and /INTx is therefore needed.

Section 5.3 shows an example of how the McBSP has to be set up. The example uses the serial port functions provided by the mcbsp54.h include file. This file comes with Code Composer Studio™ and can be found in the directory \ti\c5400\dsk\include. The example code allows an automatic switching between two channels of the ADS7842.

4.3 Parallel Interface With External Control Logic

An external counter could be used to initiate a new conversion start every 16 clock cycles, just like described in Section 3.4. Various other methods are possible.

5 Sample Code

5.1 C6x11 DSP Example Program - /WR Generated by Timer

```
/******************************************************************************/
/* ADS7842 example program for the C6x11                                      */
/* This program assumes that the ADC is connected to CE2. An external clock   */
/* is used as conversion clock for the ADC. The clock is into the DSP via the */
/* timer 0 input pin, the timer outpin pin is used as /WR signal for the ADC. */
/* Two pins of serial port 1 (DX, FSX) are used as general purpose output     */
/* pins and are connected to A0 (FSX) and A1 (DX) of the ADC.                 */
/******************************************************************************/

/* define peripherals control registers                                       */
#define EMIF_CE2      *(unsigned volatile int *)0x1800010 /* EMIF CE2 control */

#define TIMER0_PRD    *(unsigned volatile int *)0x1940004 /* Timer 0 period   */
#define TIMER0_CTRL   *(unsigned volatile int *)0x1940000 /* Timer 0 control  */
#define TIMER0_COUNT  *(unsigned volatile int *)0x1940008 /* Timer 0 count    */

#define McBSP1_SPCR   *(unsigned volatile int *)0x1900008 /* McBSP 1 SPCR     */
#define McBSP1_PCR    *(unsigned volatile int *)0x1900024 /* McBSP 1 PCR      */

/* define interrupt control registers                                         */
extern cregister volatile unsigned int CSR;     /* Control Status Register    */
extern cregister volatile unsigned int IFR;     /* Interrupt Flag Register    */
extern cregister volatile unsigned int ICR;     /* Interrupt Clear Register   */
extern cregister volatile unsigned int IER;     /* Interrupt Enable Register  */

#define ADS7842_ADDR 0xA0000000   /* define ADC address; uses whole CE2 space */
#define SMPL_BLK_SZ 128           /* define sample block size                 */

short ad_buffer[4][SMPL_BLK_SZ];  /* array for the sampled data               */

volatile int channel;   /* variable for switching between analog channels     */
volatile int cnt;       /* holds the current offset in the data array         */
volatile int finished;  /* flag to show that a block of data has been aquired */

int ChSelValue[4] = {   /* array for the serial port control register to      */
  0x00002800,           /*   program the general purpose output pins for      */
  0x00002808,           /*   the channel selection (channels 0 to 3)          */
  0x00002820,
  0x00002828
};



/******************************************************************************/
/* void main(void)                                                            */
/* The main routine performs the setup of interrupts, the CE space control    */
/* register, timer and the pins of the serial port, used as GPIO pins.        */
/* After enabling interrupts the program will sample data in an endless loop. */
/******************************************************************************/
void main(void)
{
  CSR = 0x0100;                 /* disable all interrupts                     */
  IER = 0x0001;                 /* disable all interrupts except NMI          */
  ICR = 0xffff;                 /* clear all pending interrupts               */

  /* Configure CE2 - CE2 space control register, 32-bit asynch                */
  /* Read  setup = 1, strobe = 2, hold = 1                                    */
  /* Write setup = 1, strobe = 2, hold = 1                                    */
  EMIF_CE2 = 0x2A340922;

  /* program timer 0 to pulse mode, clocked by external signal                */
  TIMER0_CTRL  = 0x00000403;    /* invert input and outut, TOUT = timer out   */
  TIMER0_COUNT = 0;             /* timer count register set to zero           */
  TIMER0_PRD   = 16;            /* devide input frequency by 16               */

  channel = 1;
  cnt = 0;
  finished = 0;
  IER = 0x00000012;             /* enable INT4 (ADS7842 /BUSY) and NMI        */

  McBSP1_SPCR = 0x00000000;     /* configure McBSP as general purpose I/O     */
  McBSP1_PCR  = ChSelValue[0];  /* select channel 0 for the first conversion  */

  TIMER0_CTRL |= 0xc0;          /* start timer 0                              */

  /* After reset it's not defined, which analog channel will be converted     */
  /* first. Therefore the program waits for the first conversion to be        */
  /* finished but doesn't read the resulting data word.                       */

  while (! (IFR & 0x0010));     /* wait for interrupt flag (/BUSY)            */
  ICR=0x0010;                   /* clear  pending interrupts                  */

  McBSP1_PCR  = ChSelValue[1];  /* select channel 1 for the next conversion   */
  CSR = CSR | 0x01;             /* int global enable, ready for int's now     */

  /* endless loop, just as demo...                                            */
  while(1) {
    while(!finished);           /* wait for the whole block of data           */
    /* ...data processing would start here...                                 */
    finished = 0;
  }
}
```

5.2 C6000 DSP McBSP Setup

```
/******************************************************************************/
/* void init_McBSP1(void)                                                     */
/* This routine sets up McBSP channel 1 to generate the desired signals as    */
/* described in section xxx                                                   */
/* It uses functions of the Chip Support Library V1.2. CSL_Init() has to be   */
/* called prior calling this routine.                                         */
/******************************************************************************/
void init_McBSP1(void)
{
    static MCBSP_CONFIG MyConfig = {
      MCBSP_SPCR_DEFAULT,
      MCBSP_RCR_DEFAULT,
      MCBSP_MK_XCR(
        MCBSP_XCR_XWDREVRS_NA,             /* no bit reversal                 */
        MCBSP_XCR_XWDLEN1_32BIT,           /* word length 32 bit              */
        MCBSP_XCR_XFRLEN1_OF(0),           /* one word in phase 1             */
        MCBSP_XCR_XPHASE2_NA,              /* don't care                      */
        MCBSP_XCR_XDATDLY_0BIT,            /* transmit data delay 0           */
        1,                                 /* MCBSP_XCR_XFIG hard coded due   */
                                           /* to a bug in CSL Version 1.2     */
        MCBSP_XCR_XCOMPAND_MSB,            /* no companding, MSB first        */
        MCBSP_XCR_XWDLEN2_8BIT,            /* don't care                      */
        MCBSP_XCR_XFRLEN2_OF(0),           /* don't care                      */
        MCBSP_XCR_XPHASE_SINGLE            /* single phase frame              */
      ),
      MCBSP_MK_SRGR(
        MCBSP_SRGR_CLKGDV_OF(SP1_CLKGDV),  /* clock divider - gloabl define   */
        MCBSP_SRGR_FWID_OF(0),             /* FS width - one cycle low (0+1)  */
        MCBSP_SRGR_FPER_OF(15),            /* FS period - 16 (15+1) cycles    */
        MCBSP_SRGR_FSGM_FSG,               /* FS driven by sample rate gen.   */
        MCBSP_SRGR_CLKSM_INTERNAL,         /* SRG driven by internal clock    */
        MCBSP_SRGR_CLKSP_RISING,           /* rising edge of CLKS used        */
        MCBSP_SRGR_GSYNC_FREE              /* CLKG running free               */
      ),
      MCBSP_MCR_DEFAULT,                  /* don't care                      */
      MCBSP_RCER_DEFAULT,                 /* don't care                      */
      MCBSP_XCER_DEFAULT,                 /* don't care                      */
      MCBSP_MK_PCR(
        MCBSP_PCR_CLKRP_FALLING,          /* don't care                      */
        MCBSP_PCR_CLKXP_FALLING,          /* xmit data on falling edge       */
        MCBSP_PCR_FSRP_ACTIVEHIGH,        /* don't care                      */
        MCBSP_PCR_FSXP_ACTIVELOW,         /* transmit frame sync active low  */
        MCBSP_PCR_DXSTAT_0,               /* don't care                      */
        MCBSP_PCR_CLKSSTAT_0,             /* don't care                      */
        MCBSP_PCR_CLKRM_INPUT,            /* don't care                      */
        MCBSP_PCR_CLKXM_OUTPUT,           /* CLKX used as clock output pin   */
        MCBSP_PCR_FSRM_EXTERNAL,          /* don't care                      */
        MCBSP_PCR_FSXM_INTERNAL,          /* xmit FS generated internally    */
        MCBSP_PCR_RIOEN_SP,               /* don't care                      */
        MCBSP_PCR_XIOEN_SP                /* transmitter in serial port mode */
      )
    };

    /* Call MCBSP_Open and open serial port 1. Handle will be returned.       */
    hMcbsp = MCBSP_Open(MCBSP_DEV1, MCBSP_OPEN_RESET);

    /* Write above configuration in McBSP1 configuration registers.          */
    MCBSP_ConfigA(hMcbsp,&MyConfig);

    /* Port is setup, let's enable it in steps.                              */
    MCBSP_EnableSrgr(hMcbsp);
    MCBSP_EnableXmt(hMcbsp);
    MCBSP_EnableFsync(hMcbsp);

    /* wait until the transmitter is ready for a sample, then write to it.    */
    /* The 32 bit value contains of 16 high bits, followed by 16 low bits.    */
    while (!MCBSP_Xrdy(hMcbsp));
    MCBSP_Write(hMcbsp,0xFFFF0000);
}
```

5.3 C5000 DSP McBSP Setup

```
#define DSP_FRQ    100000                /* DSP frequency in kHz              */
#define AD_FRQ     150                   /* AD converter sampling freq in kHz */
#define SP1_CLKGDV (DSP_FRQ/AD_FRQ/16)   /* value for the clock divider       */



/******************************************************************************/
/* void init_McBSP(short SP_Channel)                                          */
/*                                                                            */
/* This routine sets up the McBSP channel SP_channel to generate the desired  */
/* signals as described in section xxx                                        */
/******************************************************************************/
void init_McBSP(short SP_Channel)
{
    /* Reset transmitter and receiver of the serial port                      */
    MCBSP_SUBREG_BITWRITE(SP_Channel, SPCR1_SUBADDR, RRST, RRST_SZ, 0);
    MCBSP_SUBREG_BITWRITE(SP_Channel, SPCR2_SUBADDR, XRST, XRST_SZ, 0);

    /* Set the transmit word length 1 to 32 bits                              */
    MCBSP_SUBREG_WRITE(SP_Channel, XCR1_SUBADDR,  0x00A0);

    /* XFIG=1, transmit frame ignore set to 1                                 */
    MCBSP_SUBREG_WRITE(SP_Channel, XCR2_SUBADDR,  0x0004);

    /* Sample rate generator - use the defined clock division factor          */
    MCBSP_SUBREG_WRITE(SP_Channel, SRGR1_SUBADDR, SP1_CLKGDV);

    /* CLK and FS generated internally. Frame period = 16 (15+1)              */
    MCBSP_SUBREG_WRITE(SP_Channel, SRGR2_SUBADDR, 0x300F);

    /* CLKX is an output, FS generated by sample rate generator, FS active    */
    /* low, data sampled on falling edge of CLKX                              */
    MCBSP_SUBREG_WRITE(SP_Channel, PCR_SUBADDR,   0x0A0A);
    /* Enable sample rate generator and frame sync                            */
    MCBSP_SAMPLE_RATE_ENABLE(SP_Channel);
    MCBSP_FRAME_SYNC_ENABLE(SP_Channel);

    /* Release the transmitter from reset and enable it                       */
    MCBSP_ENABLE(SP_Channel,2);

    /* Write data in the DXR register                                         */
    while (!MCBSP_XRDY(SP_Channel));
    MCBSP_DXR12_WRITE(SP_Channel,0xFFFF0000);
}
```

References

1. Data sheet, *TMS320C6201 Fixed-Point Digital Signal Processor* (SPRS051)
2. *TMS320C6000 Peripherals Reference Guide* (SPRU190)
3. *TMS320C54x DSP CPU and Peripherals Reference Set Volume 1* (SPRU131)
4. Data sheet, *ADS7842 12-Bit, 4-Channel Analog To Digital Converter* (SDAS103)

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 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 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, license, warranty or endorsement thereof.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations and notices. Representation or reproduction of this information with alteration voids all warranties provided for an associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Resale of TI's products or services with *statements different from or beyond the parameters* stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Also see: Standard Terms and Conditions of Sale for Semiconductor Products. www.ti.com/sc/docs/stdterms.htm

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

Copyright © 2001, Texas Instruments Incorporated