Using TMS320C5402 DMA Channels to Read from the TLV1570 ADC

Lijoy Philipose

AAP Data Conversion

ABSTRACT

This application report presents hardware and software solutions for using the DMA channels of the 16-bit, fixed-point TMS320C5402 DSP to collect digital samples from the TLV1570 10-bit, 1.25-MSPS, 8-channel, serial analog-to-digital converter. Project collateral discussed in this application report can be downloaded from the following URL: http://www.ti.com/lit/zip/SLAA097.

Contents

List of Figures

TEXAS INSTRUMENTS

List of Tables

1 Introduction

It is inefficient to waste DSP cycles to service slow serial analog-to-digital converters. One solution is to assign the nursing duties to DSP peripherals, specifically to the DMA controller, since it is designed to function without tasking the powerful DSP. The slower serial-data converter in this case is the TLV1570. The TLV1570 is an 8-channel, 10-bit, 1.25-MSPS ADC with a four-wire serial interface. After reading this report, users will be able to implement the hardware and software interface of this ADC to the TMS320C5402 DSP via the McBSP and DMA peripherals. The C-language source code used in developing this report is provided in the appendixes.

2 Hardware

The hardware interface consists of the TMS320C5402 DSK and the TLV1570 EVM.

2.1 TMS320C5402 DSK Starter Kit

The C5402 DSK is specifically designed for digital communications applications and comes complete with a TMS320C5402-based target board, DSK-specific *Code Composer Studio* debug tools, 32K application-size-limited C-compiler/assembler/linker, parallel-port interface, power supply, and cables.

The C5402 device features 100-MHz clock, 40-bit ALU, 16K x 16-bit dual-access on-chip RAM, 4K x 16-bit on-chip ROM, advanced multibus architecture with three separate 16-bit data-memory busses, and one program memory bus. In addition to these features, the DSK has an embedded JTAG emulation via the TBC and IEEE-1284 parallel ports. The onboard parallel-port controller allows the host PC to use the parallel port for emulation, or to directly access the host-port interface of the 'C5402. Other features include onboard standard JTAG-interface connection for optional emulation and expansion connectors for add-on accessories. Texas Instruments now provides expansion connectors or adapter boards to interface all data-converter EVMs to this DSK.

The enhanced peripherals of particular interest to this report are the McBSP serial ports. These ports are further explained in Chapter 3.

The 'C5402 DSK supports a TMS320VC5402 DSP which can operate at frequencies up to 100 MHz with a core voltage of 1.8 V and an I/O voltage of 3.3 V. The DSK provides support for all the DSP interfaces and control signals. The JTAG-emulation interface is used to support both embedded and external JTAG emulations. The control interface is used to reset the device and to provide external interrupts. The McBSP0, by default, is used to interface to a telephone DAA port; this port is also available to the daughterboard via an onboard multiplexer. The McBSP1, by default, is used on microphone/speaker interfaces; it is also brought to the peripheral-expansion connector for use on the daughterboard. The CPLD controls the source of McBSP0 and McBSP1.

2.2   TLV1570 EVM

The TLV1570 evaluation module (EVM) is a complete stand-alone board designed to allow quick and accurate evaluation of the TLV1570 analog-to-digital converter (ADC).

The TLV1570 is an 8-channel, 10-bit, 1.25-MSPS (megasamples-per-second) ADC with a four-wire serial interface. It is compatible with both 3-V and 5-V systems and directly interfaces to Texas Instruments digital-signal processors. This EVM provides a 12-bit digital-to-analog converter that can be used to loopback DAC output signals to ADC inputs. Circuits such as an external-voltage-reference source are provided for use with both the ADC and the DAC. An operational amplifier has been placed between the ADC multiplexer-output pin (MO) and the analog input pin (AIN) to allow for signal conditioning. The EVM provides control signals and power pins (VCC and GND) via connector J2. Analog-input signals can be provided via connector J1. Refer to the *TLV1570 Evaluation Module* user's guide for more information on this EVM.

3   Software

The DSP, McBSP, and CPLD devices must be initialized correctly before attempting to read or write to the data converter. The following sections explain the sample code included in the appendixes.

3.1   DSP

The 'VC5402 DSK provides the DSP with a single 20-MHz frequency reference via the DSP built-in crystal oscillator. The DSP's clock-mode pins are configured via dip-switch settings to allow for a number of different frequencies, up to the part's maximum rate of 100 MHz. The CLKMD register can also be changed after reset to select the DSP's operating frequency. The CPU clock frequency is 100 MHz when CLKMD is equal to 0x4007.

3.2   CPLD

There are seven DSP CPLD registers mapped into the DSP's lower I/O address space starting at address 0x0000. Only control register 2 (CNTL2) is of interest to this report.

3.2.1   *DSP CNTL2 Control Register (I/O Address = 0x0004)*

This register selects the source of data for both McBSPs. Bit 1 of this register needs to be modified; otherwise the McBSP1 defaults to the onboard device as its input source. The McBSP1 should be set to use the daughterboard as its source of data on this report. Table 2 shows the register-bit definitions.

Table 1. CNTL 2 Control Register Bit Definitions

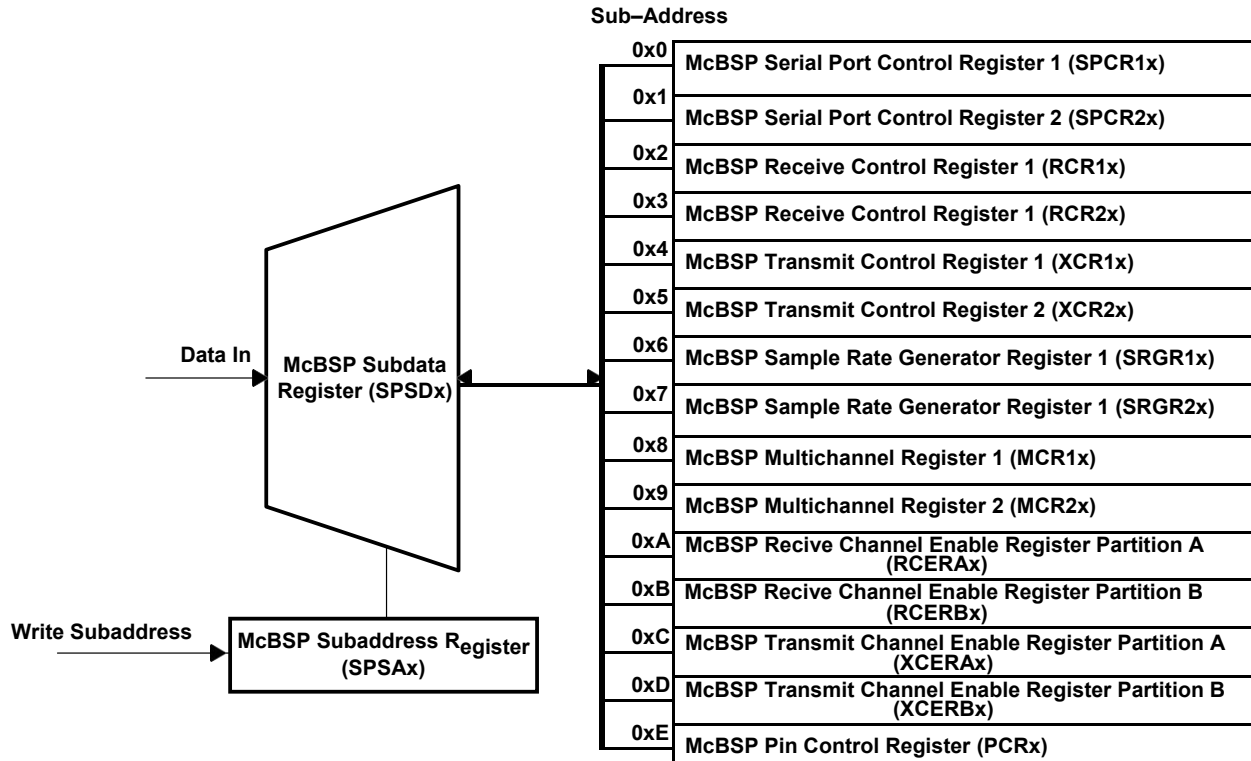| BIT	NAME	R/W	DESCRIPTION
7	DAAOH	RW	DAA off-hook control (0 = on-hook, 1 = off-hook)
6	DAACID	RW	DAA caller ID enable (0 = disabled, 1 = enabled)
5	FLASHENB–	RW	Select FLASH =1 (default) or SRAM (=0) for external memory (1)
4	INT1SEL	RW	Interrupt 1 source selection (0 = UART, 1 = daughterboard)
3	FC1CON	RW	MIC/Speaker AD50 FC control bit (0)
2	FC0CON	RW	DAA AD50 FC control bit (0)
1	BSPSEL1	RW	McBSP1 select control (0 = mic/speaker, 1 = daughterboard)
0	BSPSEL0	RW	McBSP0 select control (0 = TelSet DAA, 1 = daughterboard)

3.2.2 Write to CPLD Register CNTL2

Control register 2 should be set to enable the daughterboard as the source for McBSP1. This is accomplished by setting CNTL2=0x0002. This register is mapped to I/O space 4h; therefore, the port instruction is used to access the register.

```
ioport unsigned int port4;    /*Defines port4 as write to I/O space 0x4 */
port4 = 0x0002;               /*Write 0x0002 to CNTL2 register */
```

3.3 McBSP

The multichannel buffered serial port (McBSP) is a superset of the standard serial ports found on Texas Instrument's digital signal processors (DSP). In addition to features found on the previous serial-port interfaces, the McBSP is able to directly interface to TI/E1 framers, IOM-2-compliant devices, MVIP switching-compatible and ST-BUS-compliant devices, AC97-compliant devices, IIS-compliant devices, and SPI devices. It provides a wide selection of transmit/receive data sizes, μ-Law and A-Law companding, programmable polarity for both frame synchronization and data clocks, and highly-programmable internal clock and frame generation. These features and programming requirements are described in *TMS320C54x DSP Enhanced Peripherals Reference Set, Volume 5* by Texas Instruments.

First let us look at how the McBSP registers are accessed. The McBSP registers are memory-mapped using a register-subaddressing scheme. Figure 1 shows a visual representation of this scheme. Register subaddressing involves multiplexing a set of registers to a single location in the memory map. A sub-bank address register is used to control the multiplexer. A subdata register (SPSDx) is used to read or write data to the desired subaddressed register. To access a specific subaddressed register, the register's subaddress location is written to the subaddress register (SPSAx). This directs the multiplexer to connect to the desired physical location in memory. When a write access occurs, the data written to the subdata register is moved to the embedded data register specified by the subaddress register. Similarly for a read access, the contents of the register specified by the subaddress register is moved to the subdata register.

Sub–Address

0x0	McBSP Serial Port Control Register 1 (SPCR1x)
0x1	McBSP Serial Port Control Register 2 (SPCR2x)
0x2	McBSP Receive Control Register 1 (RCR1x)
0x3	McBSP Receive Control Register 1 (RCR2x)
0x4	McBSP Transmit Control Register 1 (XCR1x)
0x5	McBSP Transmit Control Register 2 (XCR2x)
0x6	McBSP Sample Rate Generator Register 1 (SRGR1x)
0x7	McBSP Sample Rate Generator Register 1 (SRGR2x)
0x8	McBSP Multichannel Register 1 (MCR1x)
0x9	McBSP Multichannel Register 2 (MCR2x)
0xA	McBSP Recive Channel Enable Register Partition A (RCERAx)
0xB	McBSP Recive Channel Enable Register Partition B (RCERBx)
0xC	McBSP Transmit Channel Enable Register Partition A (XCERAx)
0xD	McBSP Transmit Channel Enable Register Partition B (XCERBx)
0xE	McBSP Pin Control Register (PCRx)

Data In → McBSP Subdata Register (SPSDx)

Write Subaddress → McBSP Subaddress Register (SPSAx)

Figure 1. McBSP Register Addressing Scheme

Let us use the McBSP0 as an example: the subdata register (SPSD) is at location 0x039, and the subaddress register (SPSAx) is at location 0x038 in physical memory. The following assembly-code sample writes 0x000 to serial-port control register 1 of the McBSP0:

```
SPSA0        .set  038h  ;McBSP0 subaddress register
SPSD0        .set  039h  ;McBSP0 subdata register
SPCR10_SUB   .set  000h  ;McBSP0 serial port control register 1 subaddress

mmr(#SPSA0) =  #SPCR10_SUB
mmr(#SPSD0) = #000h
```

There are 16 registers associated with each McBSP. Interfacing a single TLV1570 ADC to the McBSP requires the proper configuration of only nine of these registers.

- The s*erial port control register 1 (SPCR1)* contains the McBSP receiver status bits and the main switch to enable or disable the receiver. This register includes the clock-stop mode bit, which sets the serial port for various clocking modes for SPI and non-SPI schemes. Also included in SPCR1 is the ABIS-mode bit and the receiver-interrupt mode bit.

- The s*erial port control register 2 (SPCR2)* contains the McBSP transmitter-status bits and the main switch to enable or disable the transmitter. This register also contains the bits to reset the frame-sync generator and the sample-rate generator.

- The *pin control register (PCR)* contains the bits to configure the McBSP pins as inputs or outputs during normal serial-port operation. This register is used to reconfigure the serial-port pins as general-purpose inputs or outputs when the receiver or transmitter is disabled. The PCR configures the transmitter and receiver clock and frame-sync modes. For

example, these bits determine whether CLKX/R and FSX/R are input or output pins and what their polarity is.

- *Receive control register 1 (RCR1)* contains the bits to configure various options of the receiver. The value of this register determines the receiver word size (between 8 and 32 bits) and the number of words per frame (1 to 128) expected per receiver event.

- *Receive control register 2 (RCR2)* determines the size of the word received and the bit delay after the frame-sync pulse. This register configuration plays an essential role when transfers greater than 16 bits and multiple phases are necessary. In this application the register bit of interest is the data-bit delay. The RCR2 register bits also select between μ-law, A-law companding, and whether the MSB or the LSB is transferred first for noncompanding 8-bit transfers.

- The *transmit control register 1 (XCR1)* contains the bits which determine the transmit-word length and frame size. A transfer can be 8 to 32 bits wide and anywhere from one to 128 words long.

- *Transmit control register 2 (XCR2)* contain the bits which determine the transmit-data delay and select between μ-law, A-law companding, and whether the MSB or the LSB is transferred first for noncompanding 8-bit transfers.

- *Sample-rate generator register one (SRGR1)* and sample-rate generator register 2 (SRGR2) control the sample-rate generator. The sample-rate generator is composed of a three-stage clock divider that allows programmable data clocks (CLKG) and framing signals (FSG). These are McBSP internal signals that can be programmed to drive the receive/transmit clock (CLKR/X) and the receive/transmit data framing (FSR/X). Sample-rate generator registers (SRGR[1,2]) control the operation of the various features of the sample-rate generator. These registers are used to control the width of the frame-sync pulse and to determine whether frame-sync is an external input driven by the sample-rate generator or a signal to indicate that data has been copied from DXR[1,2] to XSR[1,2]. These registers specify whether the sample-rate generator clock is derived from the CPU clock or from the CLKS pin, and by what value to divide the CPU clock to produce the desired serial clock (CLKX/R).

McBSP Register Bits By Control Function
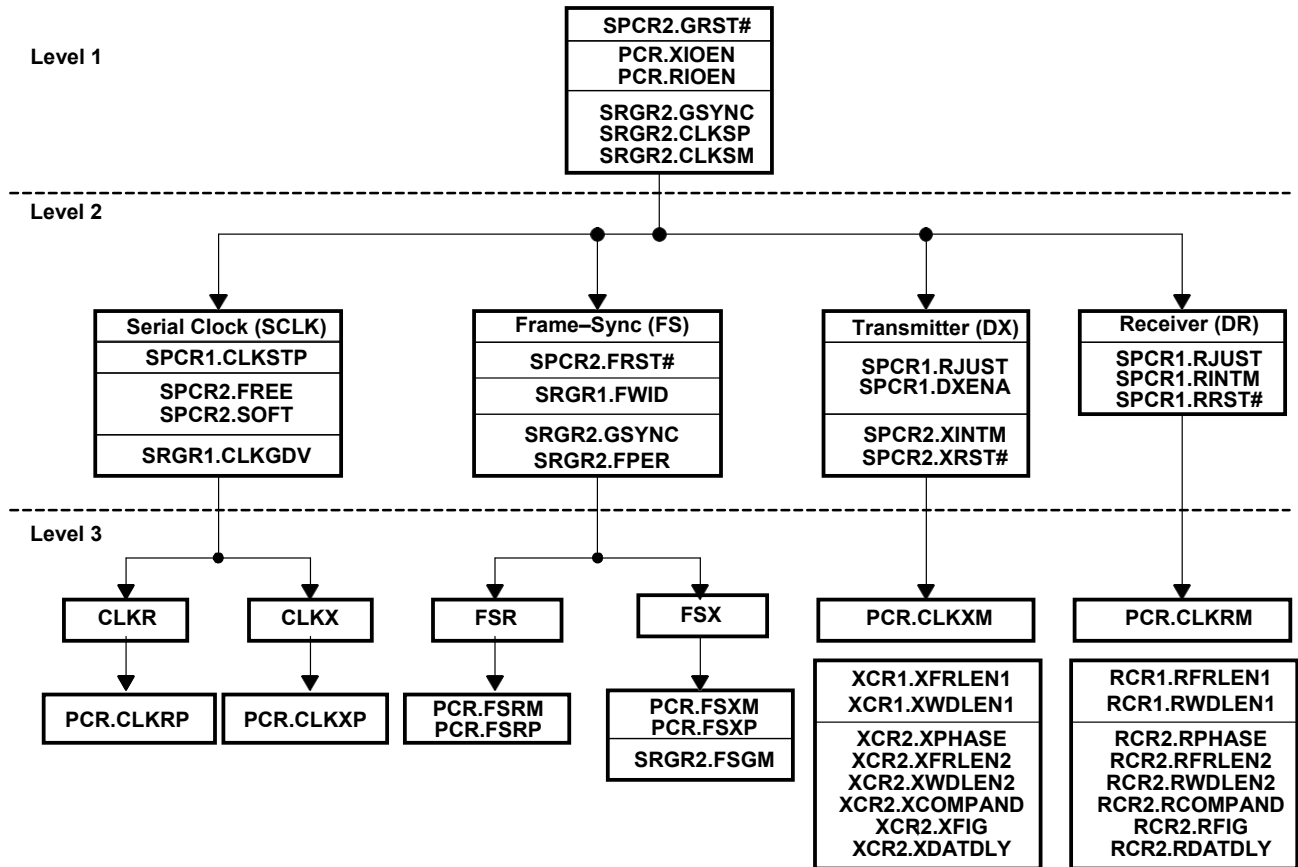
Figure 2. McBSP Register Bits With Respect to Serial-Port Function

Describing all nine registers in detail is not practical in this report. Figure 2 presents these register bits in relation to the serial lines. Table 3 summarizes the register configuration used in this application report. The multichannel registers are not included in this mapping, since they are not used for this particular interface. Figure 2 is divided into horizontal levels to illustrate the details of the bit control. Level-one register bits relate to the overall serial port. These register bits define the function of the serial-data pins, the state of the sample-rate generator, the clock input to the sample-rate generator, etc. Level-three bits define the specifics of that particular serial port signal line. Take for example the serial clock: level-three bits describe the polarity of the receive and transmit clocks. Detailed descriptions of all the register bits are found in the *TMS32054x DSP Enhanced Peripherals Reference Set, Volume 5* (Literature number SPRU302) from Texas Instruments.

Table 2. McBSP Register Settings

| McBSP1 ADDRESS	McBSP1 SUBADDRESS	ACRONYM	REGISTER INITIALIZED	COMMENT
0041		DRR11		Receive data register
0043		DXR11		Transmit data register
0048		SPSA1		McBSP1 subaddressing register
0049	0x0000	SPCR11	0x0001	The LSB bit must be 0 (0x0000) while configuring McBSP so that the transmitter is disabled.
	0x0001	SPCR21	0x02C1	The LSB bit must be 0 (0x02C0) while configuring McBSP so that the receiver is disabled.
	0x0002	RCR11	0x0040	Selects one 16-bit word transfer per frame
	0x0003	RCR21	0x0001	Sets 1-bit delay on receiver. Receiver assumes first MSB bit to arrive on the next clock cycle after FSR pulse.
	0x0004	XCR11	0x0040	Selects one 16-bit word transfer per frame.
	0x0005	XCR21	0x0001	Transmitter shifts out data immediately following the falling edge of FSX
	0x0006	SRGR11	0x0009	CLKX has a 10-MHz frequency, assuming a 100-MHz CPU clock. CLKX = CPU clock/(CLKGDV +1) when CLKGDV = SRGR1[7,0]
	0x0007	SRGR21	0x2000	The sample-rate generator clock is derived from the CPU clock
	0x000E	PCR1	0x0A00	FSX is determined from sample-rate generator frame-synchronization mode bit SRGR2.FSGM. CLKX output is driven by sample-rate generator.

3.4   DMA

The direct memory access (DMA) controller uses the same subaddressing scheme as the McBSP. So the same procedure used for writing to and reading from McBSP registers must be used here. The DMA provides one additional register which makes programming the registers easier. Figure 3 shows two registers at the input to the multiplexer. DMSDI is the sub-bank access register (DMSA) which increments the sub-bank address register after each read/write. Register DMSDN does not increment the sub-bank address register after read/write operations. The advantage in using the DMSDI is that the user no longer has to change the contents of DMSA to point to the next register. For example, programming the DMA channel 0 registers involves writing 0x00 to DMSA. The first value written to DMSDI is moved to DMSCR0, the second to DMDST0, the third to DMCTR0, etc. Register DMSDN should be used to write to the DMA register if that particular register is the only one to be modified.

Five channel-specific registers and the channel priority and enable control register (DMPREC) need to be configured for each DMA channel used:
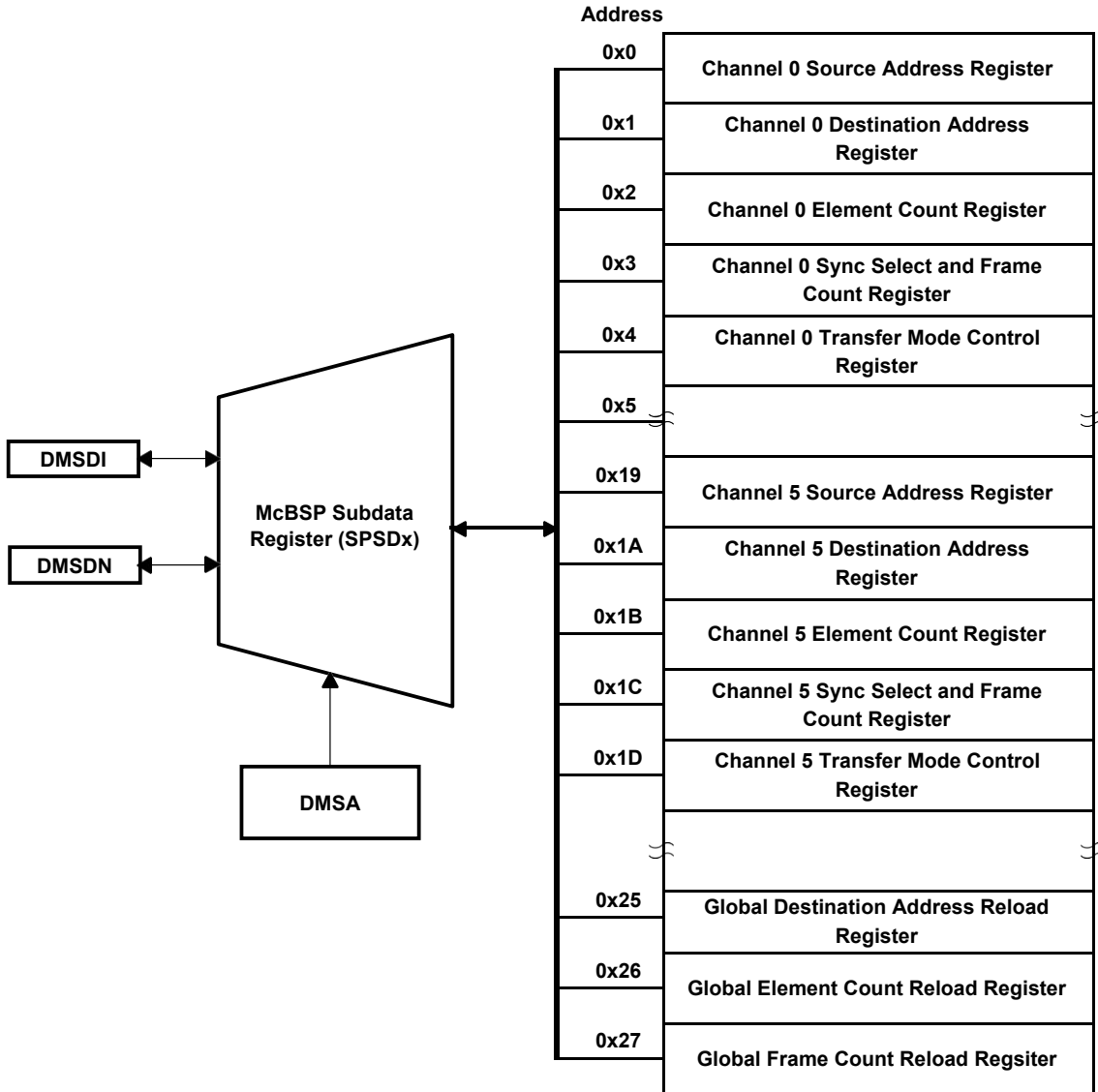
Figure 3. DMA Subaddressing Scheme

- *Source (DMSRCn) and destination (DMDSTn) registers* (where *n* is the DMA channel) store the address of the data to be read. Likewise, the address where data is to be written is stored in the DMSTn register.

- The *element count register(DMCTRn* is a 16-bit counter that keeps track of the number of DMA transfers to be completed. This register is always initialized to one less than the number of elements to be stored.

- The *DMA sync event and frame count (DMSFCn)* register controls three services: 1) the synchronization event used to trigger a DMA transfer; 2) the word size for each transfer, specified as either 16-bit or 32-bit words; 3) the number of frames to be transferred (one to 256); for example, if only one frame is desired, this register field should be written as zero (the desired value minus one).

- The *transfer-mode control register(DMMCRn)* controls the transfer mode of the channel. This register determines whether the source/destination address is postincrement or

postdecrement after each transfer. This register also determines whether the channel is operating in autobuffering mode (ABU) or multiframe mode, when the DMA will interrupt the DSP, and the address space where the source/destination addresses are located.

- The *channel priority and Enable Control Register (DMPREC)* controls the function of the overall DMA. Due to the limited number of interrupts available in the 'C54xx family, some DMA interrupts are multiplexed with other peripheral interrupts. Bits in this register determine which interrupts are assigned to the interrupt-flag register. The DMPREC also sets the priority given to each channel to either low or high so those channels with high priority are serviced before those with low priority. The DMA register settings used in this application report are summarized in Table 3.

Table 3. DMA Register Settings

REGISTER		COMMENT
NAME	VALUE	
DMSRC0	DRR11	DMA channel 0: source-address register Source-memory-mapped address to read from
DMDST0	&DataTable_0	DMA channel 0: destination-address register Destination address for sample storage
DMCTR0	NSAMPLES–1	DMA channel 0: element-count register Number of samples to store minus one
DMSFC0	0x5000	DMA channel 0: sync-select and frame-count register Synchronization transfers with McBSP1 receive event. CH0 reads data out of DRR1 when McBSP REVT occurs, then CH1 moves channel-command word to DXR1.
DMMCR0	0xC004	DMA channel 0: transfer-mode control register The content of the registers are reinitialized upon completion of block transfer. DMAC0 interrupt is generated after block transfer. No modification is made to source address register after each transfer. Destination address register is post-incremented after transfer.
DMSRC1	&ChnlSelCmd	DMA channel 1: source-address register Source-memory address to read command word from
DMDST1	DXR11	DMA channel 1: destination address register Destination address to store command, that is, McBSP1 transmitter
DMCTR1	NSAMPLES–1	DMA channel 1: element-count register Number of times to transfer command word to transmitter, minus 1
DMSFC1	0xE000	DMA channel 1: sync-select and frame-count register Synchronization transfers with INT3 receive event. When INT3 occurs, CH0 reads data out of DRR1, then CH1 moves channel-command word to DXR1.
DMMCR1	0x8000	DMA channel 1: transfer-mode control register The contents of the registers are reinitialized upon completion of block transfer. No modification is made to source-address register after each transfer. No modification is made to destination-address register after each transfer.
DMAPREC	0x0103	DMA priority and enable-control register Enable DMA CH0 and CH1. CH0 has high priority. CH0 reads data out of DRR1 and stores it before CH1 triggers another conversion cycle.

Now that the basic operation of the McBSP and DMA peripherals has been covered, let us look at the TLV1570 analog-to-digital converter.

3.5 TLV1570

3.5.1 Data Converter Operation

The TLV1570 accepts an analog-input range from 0 V to AVdd, and digitizes the input at a maximum throughout rate of 1.25 MSPS. To achieve this rate it must be clocked at 20 MHz with power supplied at 5 V. Using a 3-V supply and a maximum serial-clock input of 10 MHz, the throughput rate drops to 625 KSPS. The sampling rate is determined by dividing the serial-clock frequency by 16. This data converter requires 16 serial clocks for each conversion (sample and convert). The result from the current cycle is placed on the serial-data-out line at the start of the next conversion cycle. Before accepting valid conversion results, the device needs to be programmed to the desired operating state. This is done by writing to the 16-bit configuration register. Table 4 explains all the different operating states for this ADC.

Table 4. Configuration Register Definitions

BIT	DESCRIPTION		5 V	3 V
DI15	Software power down: 0: Normal 1: Power down enabled		X X	X X
DI14	Reads out values of the internal register, 1 – read. Only DI15 – DI1 are read out.		X	X
DI13, DI12	These two bits select the self-test voltage to be applied to the ADC input during next clock cycle: 00: Allow AIN to come in normally 01: Apply AGND to AIN 10: Apply VREF/2 to AIN 11: N/A		X	X
DI11	Choose speed application 0: High speed (higher power consumption) 1: Low speed (lower power consumption)		X	X
DI10	This bit enables channel autoscan function. 0: Autoscan disabled 1: Autoscan enabled		X	X
DI9, DI8, DI7	DI9 – DI7: These three bits select which of the eight channels is to be used (if DI10 = 0). 000: Channel 0 selected as input 001: Channel 1 selected as input 010: Channel 2 selected as input 011: Channel 3 selected as input 100: Channel 4 selected as input 101: Channel 5 selected as input 110: Channel 6 selected as input 111: Channel 7 selected as input	DI9, DI8: These two bits select the channel swept sequence used by auto scan mode (if DI10 = 1) 00: Analog inputs CH0, CH1, CH2, ….., CH7 sequentially selected 01: Analog inputs CH1, CH3, CH5, CH7 sequentially selected 10: Analog inputs CH0, CH2, CH4, CH6 sequentially selected 11: Analog inputs CH7, CH6, CH5, ….., CH0 sequentially selected DI7 Autoscan reset 0: No reset 1: Reset autoscan sequence	X	X
DI6	Selects Internal or external reference voltage: 0: External 1: Internal		X	X
DI5	Selects internal-reference voltage value to be applied to the ADC during next conversion cycle. 0: 2.3 V 1: 3.8 V		 X	 X

Table 4. Configuration Register Definitions (Continued)

BIT	DESCRIPTION	5 V	3 V
DI4	Enables/disables autopower-down function: 1: Enable 0: Disable	X	X
DI3	Performance optimizer – linearity 0: AV_{DD} = 5.5 V to 3.6 V 1: AV_{DD} = 3.5 V to 2.7 V	X	X
DI2	Always write 0 (reserved bit)	X	X
DI1	Always write 0 (reserved bit)	X	X
DI0	Always write 0 (reserved bit)	X	X

3.5.2   Hardware Overview

The block diagram of Figure 4 shows how to connect the TLV1570 EVM to the 'C5402 DSP. General-purpose I/O pin XF is used as the chip-select pin. FSR1 and CLKR1 are tied to EVM pins FSX1 and CLKX1, respectively. Pin SDOUT is tied to pin DRR1, and pin SDIN is connected to pin DXR1. This configuration indicates that the TLV1570 is interfaced to the McBSP1. Tying the FSR1 and FSX1 lines together ensures that the receiver expects data to arrive at the same time the ADC transmits it.
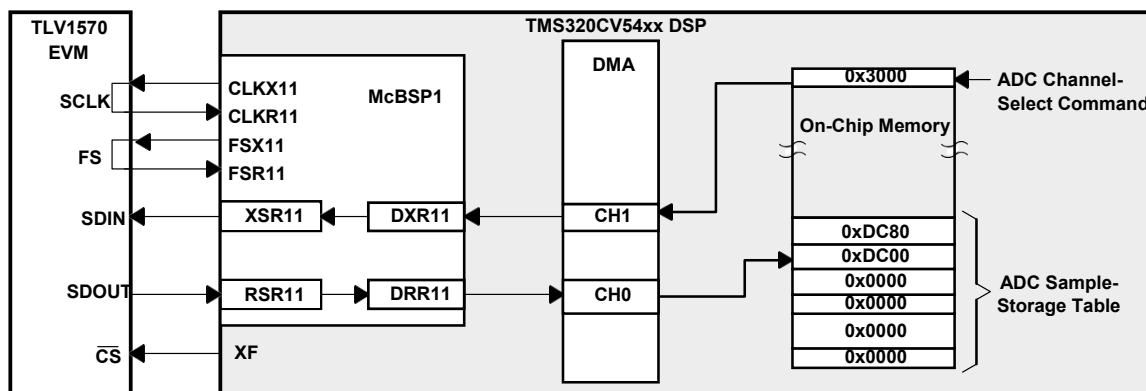
Figure 4. Overview of How DMA Is Used to Collect Samples

3.5.3   Software Overview

The software flowchart for this interface method is presented in Figure 5. Begin by setting the CPU-clock speed. On the 'C5402 DSK, the maximum CPU clock is 100 MHz. If there are any pending interrupts, they are cleared by writing all ones to IFR. Enable the DMAC0 interrupt in IMR. The interrupt service routine (ISR) associated with this interrupt can be used to initiate DSP processing of the collected ADC samples. The interrupt vector table (IVT) needs to be remapped so it points to the user ISR. The vector table should be placed at the beginning of a data page.

The 'C5402 DSK I/O lines are managed by a CPLD. The input to the McBSP1 must be set to arrive from the expansion bus.

Before configuring the McBSP registers, it is important to disable the transmitter and receiver portions. Once disabled, users need to configure the device for the desired operation; only then the transmitter and receiver may be enabled.

For this interface, the McBSP needs to be configured per Table 2. Note that Table 2 settings have the transmitter and receiver enabled. The transmitter and the receiver must be disabled during initialization of the McBSP.

The DMA channel 1 source register is initialized to the address of the ADC command word. Its destination register is initialized to the address of the McBSP1 transmit-data register. The count registers for both channels store the value for the number of samples (transfers) to read (implement). Both channel 1 and channel 0 are synchronized to the McBSP1 receive event. Therefore, when a receive event occurs, the DMA channel 0 reads the data from the DRR1 and stores it in memory. DMA channel 1 sends the next command word to the ADC to begin another conversion. Table 3 explains the DMA-register settings used in this application report.
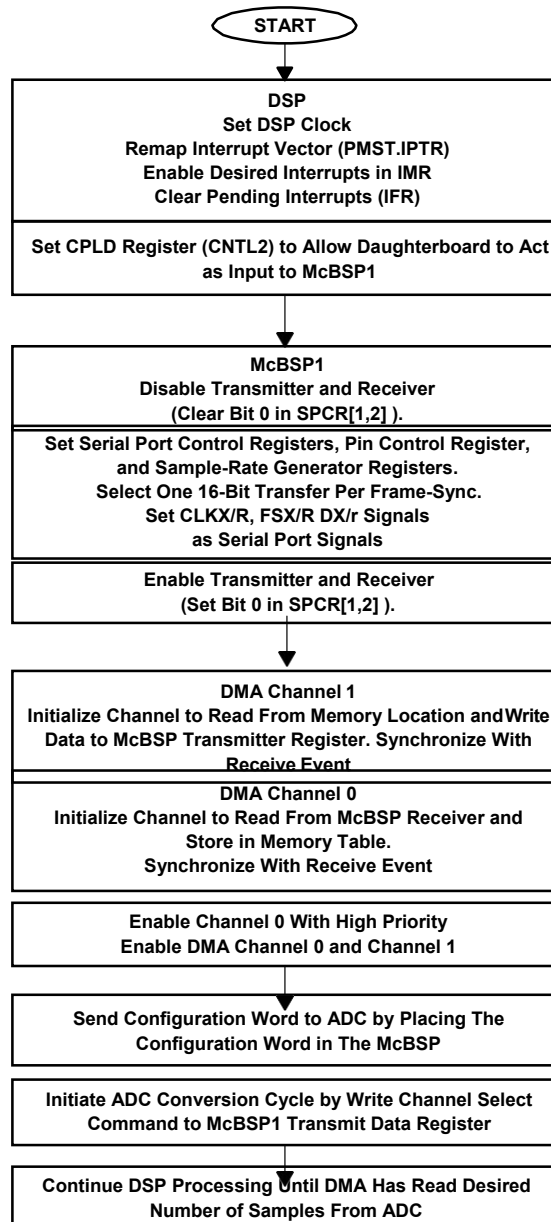
Figure 5. Program Flow Chart

Note that channel zero, where the received data is stored, is given the high priority. This selection ensures that received data is stored after every conversion. The only remaining step is to trigger the first conversion and enable the DMA channels.
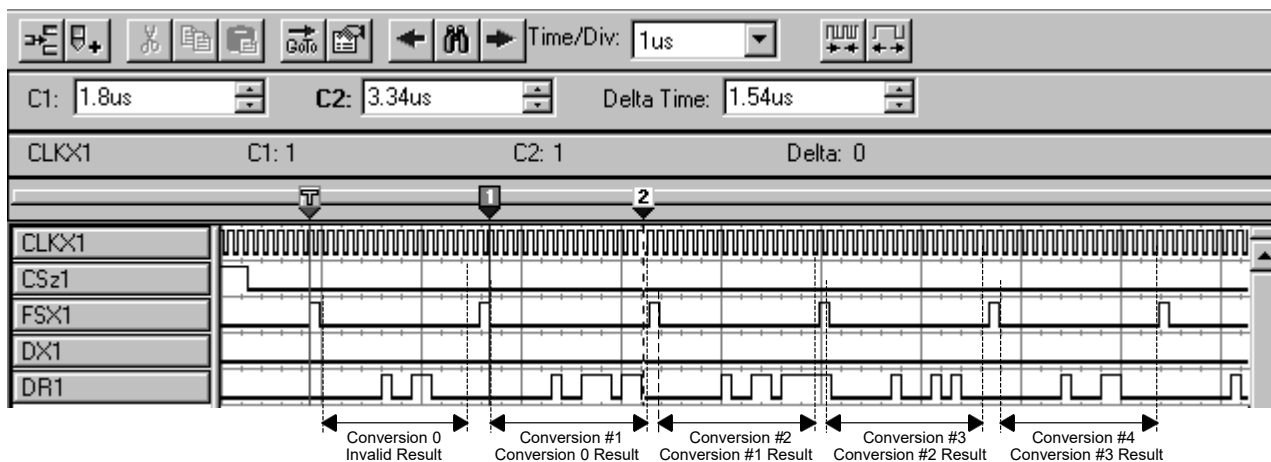
Figure 6. DMA Triggered Multiple Conversions

Figure 6 shows the behavior of the hardware in this application. The data read in from the initial conversion (conversion 0) cycle is discarded, since it comes from an unknown configuration. Conversion result 0 is put on the SDOUT line while the second conversion (conversion 1) cycle is proceeding. After the DMA has completed all its transfers, it notifies the DSP by generating a DMA-channel-zero interrupt event (DMAC0). When the DMAC0 interrupt occurs, the CPU begins processing the data. The DMA channels store the specified number of samples before producing the interrupt. Once the DMA completes the specified number of reads, it generates an interrupt (DMAC0) to signal the CPU to process the samples. In the mean time the CPU can be performing other tasks.

4   References

1. *TMS320C54X DSP CPU And Peripherals Reference Set, Vol I*, Literature number SPRU131
2. *TMS320C54X Optimizing C Compiler User's Guide*, Literature number SPRU103
3. *TMS320C54XX DSP Enhanced Peripherals Reference Set, Vol 5*, Literature number SPRU302
4. *TLV1570 Evaluation Module User's Guide*, Literature number SLAU024
5. TLV1570 data sheet, Literature number SLAS169

Appendix A   main.c

```
/*****************************************************************************/
/*Function:    main()                                              */
/*file name:   main.c                                              */
/*Description: Main function for using the DMA to collect MAX samples    */
/*    from TLV1570 ADC. Begin by setting up the DSP, CPLD, and McBSP.   */
/*    Then send configuration word and begin the first conversion cycle.  */
/*    Once this cycle is complete, the DMA will collect MAX number of    */
/*    samples from the ADC attached to McBSP1. The samples are        */
/*    stored in array DataTable_0. Once NSAMPLES samples have been      */
/*    collected, the DMA will produce a DMACO interrupt to the DSP.     */
/*Inputs:      None                                              */
/*Outputs:     None                                              */
/*Returns:     None                                              */
/*Note:        None                                              */
/* AUTHOR:     AAP Application Group, L. Philipose, Dallas          */
/*             CREATED 2000(C) BY TEXAS INSTRUMENTS INCORPORATED       */
/*****************************************************************************/
#include "c5402Reg.h" /*File Contains structures definitions for DSP, McBSP and DMA  */
#include "adc_const.h" /*File Contains ADC parameter data for each data converter(DC)*/

MCBSP McBSP1;                          /*Initialize McBSP Register Structures */
unsigned int n=0,l=0;                  /*Global Index useful DataTable array */
unsigned int DataTable_0[NSAMPLES];    /*Data Table                      */
ioport unsigned int port0;             /*Write to i/o address 0x0        */
ioport unsigned int port4;             /*Write to i/o address 0x4        */

/*Function used to communicate with ADC on McBSP1                        */

unsigned int McBSP1WriteRead(unsigned int value)
{
   /*Write McBSP1 */
   SPSA1_ADDR = SPCR21_SUB;
   while(SPCR21_ADDR->bitval.xrdy != 1);
   DXR11_ADDR = value;

   /*Read McBSP1 */
   SPSA1_ADDR = SPCR11_SUB;
   while(SPCR11_ADDR->bitval.rrdy != 1);
   return ( DRR11_ADDR );
}
void main(void)
{
 unsigned int CLKGDV,ChnlSelCmd; /*Divide down ratio for CPU to produce CLKX in SRGR1  */
   unsigned int channel0,source0, /*Variables used to store DMA register settings */
      destination0, count0,
      frame_sync0, control_mode0;
  unsigned int channel1,source1,
      destination1, count1,
      frame_sync1, control_mode1;

   port4 = 0x0003;   /*Set CNTL2, in CPLD, allow daughtercard as input to McBSP1 */
```

```
port0 = 0x00808;  /*Set CNTL1, in CPLD, allow INT3# from daughtercard to DSP */

ST1_ADDR->bitval.xf = 1;   /*set ADC CS# high                       */
ST1_ADDR->bitval.intm = 1; /*Disable System Interrupts              */

DMAPREC_ADDR -> value = 0x0000;  /* Reset DMA channels               */

IMR_ADDR->value = 0;       /*Mask out all interrupts                */
IFR_ADDR->value = 0xFFFF;  /*Clear all pending interrupts by writing ones */
                           /*to register                            */

PMST_ADDR->value = 0x3620; /*Configure PMST, set IPTR=0x3600. Remap  */
                           /*Interrupt Vector Table                 */

ST1_ADDR->bitval.intm = 0; /*Enable System Interrupts               */

while (n<=NSAMPLES)        /*Initialize Data Table 0 with 0xFFFF     */
{
    DataTable_0[n++]=0xFFFF;
}

CLKGDV=(DSP_FREQ/(SERCLOCK+1));  /*CPU clock (in MHz) divide by desired   */
                                 /*CLKX Freq (in MHz)                     */
                                 /*plus one = SRGR1.CLKGDV                */

                                 /*Initialize McBSP1 registers*/
McBSP1.RegVal.SPCR1_REG = 0x0000;   /*Receiver Off*/
McBSP1.RegVal.SPCR2_REG = 0x02C0;   /*Free running serial clock, transmitter
                                    /*off                           */
                                    /*Frame-Sync Generator and Sample-Rate */
                                    /*Generator Reset               */
McBSP1.RegVal.PCR_REG = 0x0A00;     /*DX,FSX,CLKX,DR,FSR,CLKR are serial   */
                                    /*port pins. CLKX and FSX/R output driven */
                                    /*by sample-rate generator.       */
McBSP1.RegVal.RCR1_REG = 0x0040;    /*Receive 16-bits per frame       */
McBSP1.RegVal.RCR2_REG = 0x0001;    /*Receiver 16-bits per frame, one-bit   */
                                    /*receive                       */
                                    /*delay. With this delay receiver assumes*/
                                    /*first bit arrives after falling edge of*/
                                    /*FSR                           */
McBSP1.RegVal.XCR1_REG = 0x0040;    /*Transmit 16-bits per frame      */
McBSP1.RegVal.XCR2_REG = 0x0001;    /*Transmit 16-bits per frame and one-bit */
                                    /*transmit delay                */
                                    /*With this delay transmitter sends first*/
                                    /*bit after falling edge of FSX  */
McBSP1.RegVal.SRGR1_REG =  CLKGDV;  /*FSX width is default on Clock.  */
                                    /*CPU divide-down number determines    */
                                    /*CLKX frequency.               */
McBSP1.RegVal.SRGR2_REG =  0x2000;  /*Sample-rate generator Clock derived  */
                                    /*from CPU clock                */
McBSP1.RegVal.MCR1_REG  =  0x0;     /*Not using this feature         */
McBSP1.RegVal.MCR2_REG  =  0x0;
McBSP1.RegVal.RCERA_REG =  0x0;
```

```
   McBSP1.RegVal.RCERB_REG  =  0x0;
   McBSP1.RegVal.XCERA_REG  =  0x0;
   McBSP1.RegVal.XCERB_REG  =  0x0;

   MCBSP1_init(&McBSP1);               /*Initialize McBSP1 registers with user */
                                       /*values                                */

   SPSA1_ADDR = SPCR21_SUB;            /*Choose Serial Port Control Register 2 */
   SPCR21_ADDR->bitval.xrst = 1;       /*Enable McBSP1 Transmitter        */

   SPSA1_ADDR = SPCR11_SUB;
   SPCR11_ADDR->bitval.rrst = 1;       /*Enable McBSP1 Receiver           */

   DMA_reset();                        /*Reset all DMA Channels           */

   channel0      =0x0;                 /*Set up DMA Channel 0 store Samples */
                                       /*from McBSP1                      */
   source0       =DRR11_BASE;          /*Source address to read from      */
   destination0  =(unsigned int)&DataTable_0;  /*Destination Address to    */
                                       /*store Sample to                  */
   count0        =NSAMPLES-1;          /*Number of Samples to Store       */
   frame_sync0   =0x5000;             /*Sync of McBSP1 receive event      */
   control_mode0 =0xC004;             /*Autoinitialization, interrupt     */
                                       /*after buffer                     */

   DMA_init(channel0, source0,         /*Initialize DMA Channel 0          */
        destination0,
        count0, frame_sync0,
        control_mode0);
/*Set up Transmitter channel 1 */
   channel1=0x1;                       /*Set up DMA Channel 1 to send      */
                                       /*Command  Word  to  Transmitter    */
   source1 =(unsigned int)&ChnlSelCmd;/*Source address to read command Word */
   destination1  =DXR11_BASE;          /*Destination Address to Store Command,  */
                                       /*that is, McBSP1 transmitter       */
   count1        =NSAMPLES-1;          /*Number of Times to transfer Command */
                                       /*Word  to transmitter.             */
   frame_sync1 =0x5000  ;  /*Sync  transfer  with  McBSP1  receive  event */
   control_mode1 =0x8000;              /*Autoinitialize registers from transfers */

   DMA_init(  channel1, source1,   /*Initialize DMA Channel 1           */
            destination1, count1,
            frame_sync1, control_mode1);
   DMAPREC_ADDR->value=0x0183;    /*Enable DMA CH0 & CH1. channel 0 has   */
                                       /*high priority                    */
                                       /*INT10 & 11 for McBSP1            */

   ST1_ADDR->bitval.xf = 0;       /*Set ADC CS# low                       */
   ChnlSelCmd=ADChan0;                 /*This Channel Select Command will be used */
   McBSP1WriteRead(ChnlSelCmd);       /*Send configuration word to ADC by the   */
                                       /*DMA for each conversion cycle    */
```

```
    while(IFR_ADDR->bitval.dmac0 != 1); /*Wait for DMA buffer to be filled    */
                                        /*before quitting                     */
    ST1_ADDR->bitval.xf = 1;            /*Desired samples collected. Deselect ADC*/

}
```

Appendix B   c5402Reg.h

```
/********************************************************************/
/*Function:     None                                           */
/*file name:    c5402Reg.h                                     */
/*Description: File defines ONLY those DSP, McBSP1, and DMA Registers    */
/*        objects used in conjunction with this Application Note.    */
/*Inputs:       None                                           */
/*Outputs:      None                                           */
/*Returns:      None                                           */
/* AUTHOR:      AAP Application Group, L. Philipose, Dallas      */
/*                CREATED 2000(C) BY TEXAS INSTRUMENTS INCORPORATED    */
/********************************************************************/
/****************************************************************/
/* Define Interrupt Flag and Interrupt Mask Registers        */
/****************************************************************/
#define IMR_BASE        0x00
#define IMR_ADDR        ((volatile IMR_REG *)   ((char *) IMR_BASE))
#define IFR_BASE        0x01
#define IFR_ADDR        ((volatile IFR_REG *)   ((char *) IFR_BASE))
typedef union {
        struct {
                unsigned int res        :2;
                unsigned int dmac5      :1;
                unsigned  int   dmac4   :1;
                unsigned  int   bxint1  :1;
                unsigned  int   brint1  :1;
                unsigned int hint       :1;
                unsigned int int3       :1;
                unsigned int tint1      :1;
                unsigned  int   dmac0   :1;
                unsigned  int   bxint0  :1;
                unsigned  int   brint0  :1;
                unsigned int tint0      :1;
                unsigned int int2       :1;
                unsigned int int1       :1;
                unsigned int int0       :1;
                } bitval;
        unsigned int value;
} IFR_REG;
typedef union {
```

```
        struct {
                unsigned int res        :2;
                unsigned int dmac5      :1;
                unsigned  int   dmac4   :1;
                unsigned  int   bxint1  :1;
                unsigned  int   brint1  :1;
                unsigned int hint       :1;
                unsigned int int3       :1;
                unsigned int tint1      :1;
                unsigned  int   dmac0   :1;
                unsigned  int   bxint0  :1;
                unsigned  int   brint0  :1;
                unsigned int tint0      :1;
                unsigned int int2       :1;
                unsigned int int1       :1;
                unsigned int int0       :1;
                } bitval;
        unsigned int value;
} IMR_REG;
/******************************************************************/
/*Status Registers */
/******************************************************************/
#define ST0_BASE        0x06
#define ST1_BASE        0x07
#define  ST0_ADDR   ((volatile  ST0_REG  *)   ((char  *)  ST0_BASE))
#define  ST1_ADDR   ((volatile  ST1_REG  *)   ((char  *)  ST1_BASE))
typedef union {
        struct {
                unsigned int arp        :3;
                unsigned int tc         :1;
                unsigned int c          :1;
                unsigned int ova        :1;
                unsigned int ovb        :1;
                unsigned int dp         :9;
                } bitval;
        unsigned int value;
} ST0_REG;
typedef union {
        struct {
                unsigned int braf       :1;
```

```
            unsigned int cpl         :1;
            unsigned int xf          :1;
            unsigned int hm          :1;
            unsigned int intm        :1;
            unsigned int zero        :1;
            unsigned int ovm         :1;
            unsigned int sxm         :1;
            unsigned int c16         :1;
            unsigned int frct        :1;
            unsigned int cmpt        :1;
            unsigned int asmm        :5;
            } bitval;
      unsigned int value;
} ST1_REG;


/********************************************************************/
/*PMST */
/********************************************************************/
#define PMST_BASE      0x1d
#define PMST_ADDR      ((volatile PMST_REG *)  ((char *) PMST_BASE))
typedef union {
      struct {
            unsigned int iptr        :9;
            unsigned int mpmc        :1;
            unsigned int ovly        :1;
            unsigned int avis        :1;
            unsigned  int   drom  :1;
            unsigned  int  clkoff  :1;
            unsigned int smul        :1;
            unsigned int sst         :1;
            } bitval;
      unsigned int value;
} PMST_REG;
/********************************************************************/
/* Structure for McBSP */
/********************************************************************/
/*-----------------------------------------------------------------*/
/* McBSP 1 */
/*-----------------------------------------------------------------*/
#define DRR21_BASE      0x40
```

```
#define DRR11_BASE        0x41
#define DXR21_BASE        0x42
#define DXR11_BASE        0x43
#define   SPSA1_BASE      0x48
#define   SPCR11_BASE     0x49
#define   SPCR21_BASE     0x49
#define RCR11_BASE        0x49
#define RCR21_BASE        0x49
#define XCR11_BASE        0x49
#define   XCR21_BASE      0x49
#define   SRGR11_BASE     0x49
#define   SRGR21_BASE     0x49
#define MCR11_BASE        0x49
#define   MCR21_BASE      0x49
#define   RCERA1_BASE     0x49
#define   RCERB1_BASE     0x49
#define   XCERA1_BASE     0x49
#define   XCERB1_BASE     0x49
#define PCR1_BASE         0x49
#define SPCR11_SUB        0x00
#define SPCR21_SUB        0x01
#define RCR11_SUB         0x02
#define RCR21_SUB         0x03
#define XCR11_SUB         0x04
#define XCR21_SUB         0x05
#define SRGR11_SUB        0x06
#define SRGR21_SUB        0x07
#define MCR11_SUB         0x08
#define MCR21_SUB         0x09
#define RCERA1_SUB        0x0A
#define RCERB1_SUB        0x0B
#define XCERA1_SUB        0x0C
#define XCERB1_SUB        0x0D
#define PCR1_SUB          0x0E
#define  DRR21_ADDR  (*(volatile  unsigned  int  *)DRR21_BASE)
#define  DRR11_ADDR  (*(volatile  unsigned  int  *)DRR11_BASE)
#define  DXR21_ADDR  (*(volatile  unsigned  int  *)DXR21_BASE)
#define  DXR11_ADDR  (*(volatile  unsigned  int  *)DXR11_BASE)
#define SPSA1_ADDR (*(volatile unsigned int *)SPSA1_BASE)
#define SPCR11_ADDR ((volatile SPCR1_REG *) ((char *) SPCR11_BASE))
```

```
#define SPCR21_ADDR      ((volatile SPCR2_REG *) ((char *) SPCR21_BASE))
#define RCR11_ADDR       ((volatile RCR1_REG *)  ((char *) RCR11_BASE))
#define RCR21_ADDR       ((volatile RCR2_REG *)  ((char *) RCR21_BASE))
#define XCR11_ADDR       ((volatile XCR1_REG *)  ((char *) XCR11_BASE))
#define XCR21_ADDR       ((volatile XCR2_REG *)  ((char *) XCR21_BASE))
#define SRGR11_ADDR      ((volatile SRGR1_REG *) ((char *) SRGR11_BASE))
#define SRGR21_ADDR      ((volatile SRGR2_REG *) ((char *) SRGR21_BASE))
#define MCR11_ADDR       ((volatile MCR1_REG *)  ((char *) MCR11_BASE))
#define MCR21_ADDR       ((volatile MCR2_REG *)  ((char *) MCR21_BASE))
#define RCERA1_ADDR      ((volatile RCERA_REG *) ((char *) RCERA1_BASE))
#define RCERB1_ADDR      ((volatile RCERB_REG *) ((char *) RCERB1_BASE))
#define XCERA1_ADDR      ((volatile XCERA_REG *) ((char *) XCERA1_BASE))
#define XCERB1_ADDR      ((volatile XCERB_REG *) ((char *) XCERB1_BASE))
#define PCR1_ADDR        ((volatile PCR_REG *)   ((char *) PCR1_BASE))
/*------------------------------------------------------------------*/
/* SPCR1 */
/*------------------------------------------------------------------*/
typedef union {
        struct {
                unsigned int dlb:1;
                unsigned int rjust:2;
                unsigned int clkstp:2;
                unsigned int rsrvd:3;
                unsigned int dxena:1;
                unsigned int abis:1;
                unsigned int rintm:2;
                unsigned int rsyncerr:1;
                unsigned int rfull:1;
                unsigned int rrdy:1;
                unsigned int rrst:1;
                } bitval;
        unsigned int value;
} SPCR1_REG;
/*------------------------------------------------------------------*/
/* SPCR2 */
/*------------------------------------------------------------------*/
typedef union {
        struct {
                unsigned int rsrvd:6;
                unsigned int free:1;
```

```
                unsigned int soft:1;
                unsigned int frst:1;
                unsigned int grst:1;
                unsigned int xintm:2;
                unsigned int xsyncerr:1;
                unsigned int xempty:1;
                unsigned int xrdy:1;
                unsigned int xrst:1;
                } bitval;
        unsigned int value;
} SPCR2_REG;
/*-------------------------------------------------------------------*/
/* PCR */
/*-------------------------------------------------------------------*/
typedef union {
        struct {
                unsigned int rsrvd1:2;
                unsigned int xioen:1;
                unsigned int rioen:1;
                unsigned int fsxm:1;
                unsigned int fsrm:1;
                unsigned int clkxm:1;
                unsigned int clkrm:1;
                unsigned int rsrvd2:1;
                unsigned int clks_stat:1;
                unsigned int dx_stat:1;
                unsigned int dr_stat:1;
                unsigned int fsxp:1;
                unsigned int fsrp:1;
                unsigned int clkxp:1;
                unsigned int clkrp:1;
                } bitval;
        unsigned int value;
} PCR_REG;
/*-------------------------------------------------------------------*/
/* RCR1 */
/*-------------------------------------------------------------------*/
typedef union {
        struct {
                unsigned int rsrvd1:1;
```

```
                unsigned int rfrlen1:7;

                unsigned int rwdlen1:3;

                unsigned int rsrvd2:5;

                } bitval;

        unsigned int value;

} RCR1_REG;
/*-----------------------------------------------------------------------*/
/* RCR2 */
/*-----------------------------------------------------------------------*/
typedef union {

        struct {

                unsigned int rphase:1;

                unsigned int rfrlen2:7;

                unsigned int rwdlen2:3;

                unsigned int rcompand:2;

                unsigned int rfig:1;

                unsigned int rdatdly:2;

                } bitval;

        unsigned int value;

} RCR2_REG;
/*-----------------------------------------------------------------------*/
/* XCR1 */
/*-----------------------------------------------------------------------*/
typedef union {

        struct {

                unsigned int rsrvd1:1;

                unsigned int xfrlen1:7;

                unsigned int xwdlen1:3;

                unsigned int rsrvd2:5;

                } bitval;

        unsigned int value;

} XCR1_REG;
/*-----------------------------------------------------------------------*/
/* XCR2 */
/*-----------------------------------------------------------------------*/
typedef union {

        struct {

                unsigned int xphase:1;

                unsigned int xfrlen2:7;

                unsigned int xwdlen2:3;
```

```
                  unsigned int xcompand:2;
                  unsigned int xfig:1;
                  unsigned int xdatdly:2;
                  } bitval;
          unsigned int value;
} XCR2_REG;
/*----------------------------------------------------------------------*/
/* SRGR1 */
/*----------------------------------------------------------------------*/
typedef union {
          struct {
                  unsigned int fwid:8;
                  unsigned int clkdiv:8;
                  } bitval;
          unsigned int value;
} SRGR1_REG;
/*----------------------------------------------------------------------*/
/* SRGR2 */
/*----------------------------------------------------------------------*/
typedef union {
          struct {
                  unsigned int gsync:1;
                  unsigned int clksp:1;
                  unsigned int clksm:1;
                  unsigned int fsgm:1;
                  unsigned int fper:12;
                  } bitval;
          unsigned int value;
} SRGR2_REG;
/*----------------------------------------------------------------------*/
/* MCR1 */
/*----------------------------------------------------------------------*/
typedef union {
          struct {
                  unsigned int rsrvd1:7;
                  unsigned int rpbblk:2;
                  unsigned int rpablk:2;
                  unsigned int rcblk:3;
                  unsigned int rsrvd2:1;
                  unsigned int rmcm:1;
```

```
                } bitval;
        unsigned int value;
} MCR1_REG;
/*------------------------------------------------------------------*/
/* MCR2 */
/*------------------------------------------------------------------*/
typedef union {
        struct {
                unsigned int rsrvd1:7;
                unsigned int xpbblk:2;
                unsigned int xpablk:2;
                unsigned int xcblk:3;
                unsigned int xmcm:2;
                } bitval;
        unsigned int value;
} MCR2_REG;
/*------------------------------------------------------------------*/
/* RCERA */
/*------------------------------------------------------------------*/
typedef union {
        struct {
                unsigned int RCEA15:1;
                unsigned int RCEA14:1;
                unsigned int RCEA13:1;
                unsigned int RCEA12:1;
                unsigned int RCEA11:1;
                unsigned int RCEA10:1;
                unsigned int RCEA9:1;
                unsigned int RCEA8:1;
                unsigned int RCEA7:1;
                unsigned int RCEA6:1;
                unsigned int RCEA5:1;
                unsigned int RCEA4:1;
                unsigned int RCEA3:1;
                unsigned int RCEA2:1;
                unsigned int RCEA1:1;
                unsigned int RCEA0:1;
                } bitval;
        unsigned int value;
} RCERA_REG;
```

```
/*-------------------------------------------------------------------------*/
/* RCERB */
/*-------------------------------------------------------------------------*/
typedef union {
        struct {
                unsigned int RCEB15:1;
                unsigned int RCEB14:1;
                unsigned int RCEB13:1;
                unsigned int RCEB12:1;
                unsigned int RCEB11:1;
                unsigned int RCEB10:1;
                unsigned int RCEB9:1;
                unsigned int RCEB8:1;
                unsigned int RCEB7:1;
                unsigned int RCEB6:1;
                unsigned int RCEB5:1;
                unsigned int RCEB4:1;
                unsigned int RCEB3:1;
                unsigned int RCEB2:1;
                unsigned int RCEB1:1;
                unsigned int RCEB0:1;
                } bitval;
        unsigned int value;
} RCERB_REG;
/*-------------------------------------------------------------------------*/
/* XCERA */
/*-------------------------------------------------------------------------*/
typedef union {
        struct {
                unsigned int XCEA15:1;
                unsigned int XCEA14:1;
                unsigned int XCEA13:1;
                unsigned int XCEA12:1;
                unsigned int XCEA11:1;
                unsigned int XCEA10:1;
                unsigned int XCEA9:1;
                unsigned int XCEA8:1;
                unsigned int XCEA7:1;
                unsigned int XCEA6:1;
                unsigned int XCEA5:1;
```

```
                unsigned int XCEA4:1;

                unsigned int XCEA3:1;

                unsigned int XCEA2:1;

                unsigned int XCEA1:1;

                unsigned int XCEA0:1;

                } bitval;

        unsigned int value;

} XCERA_REG;
/*----------------------------------------------------------------------*/
/* XCERB */
/*----------------------------------------------------------------------*/
typedef union {

        struct {

                unsigned int XCEB15:1;

                unsigned int XCEB14:1;

                unsigned int XCEB13:1;

                unsigned int XCEB12:1;

                unsigned int XCEB11:1;

                unsigned int XCEB10:1;

                unsigned int XCEB9:1;

                unsigned int XCEB8:1;

                unsigned int XCEB7:1;

                unsigned int XCEB6:1;

                unsigned int XCEB5:1;

                unsigned int XCEB4:1;

                unsigned int XCEB3:1;

                unsigned int XCEB2:1;

                unsigned int XCEB1:1;

                unsigned int XCEB0:1;

                } bitval;

        unsigned int value;

} XCERB_REG;
typedef union {

        struct {

                unsigned int SPCR1_REG :16;

                unsigned int SPCR2_REG :16;

                unsigned  int  PCR_REG  :16;

                unsigned  int  RCR1_REG :16;

                unsigned  int  RCR2_REG :16;

                unsigned int XCR1_REG :16;
```

```
                unsigned int XCR2_REG :16;
                unsigned int SRGR1_REG :16;
                unsigned int SRGR2_REG :16;
                unsigned int MCR1_REG :16;
                unsigned int MCR2_REG :16;
                unsigned int RCERA_REG :16;
                unsigned int RCERB_REG :16;
                unsigned int XCERA_REG :16;
                unsigned int XCERB_REG :16;
                } RegVal;
        unsigned int value;
} MCBSP;
/**********************************************************************/
/* Structure for DMA */
/**********************************************************************/
#define DMAPREC_BASE    0x54
#define DMAPREC_ADDR    ((volatile DMPREC_REG *)        ((char *) DMAPREC_BASE))
#define DMSBA_BASE      0x55
#define DMSBA_ADDR      (*(volatile unsigned int *) DMSBA_BASE)
#define DMSBAI_BASE     0x56    /* Autoincrementing Subaddress Register */
#define DMSBAI_ADDR     (*(volatile unsigned int *)     DMSBAI_BASE)
#define DMSBANOI_BASE   0x57    /* Subaddress Register without Autoincrement */
#define DMSBANOI_ADDR   (*(volatile unsigned int *)     DMSBANOI_BASE)
/* Sub addressing offsets */
#define DMSRC0_SUB      0x00
#define DMDST0_SUB      0x01
#define DMCTR0_SUB      0x02
#define DMSFC0_SUB      0x03
#define DMMCR0_SUB      0x04
#define DMSRC1_SUB      0x05
#define DMDST1_SUB      0x06
#define DMCTR1_SUB      0x07
#define DMSFC1_SUB      0x08
#define DMMCR1_SUB      0x09
#define DMSRC2_SUB      0x0A
#define DMDST2_SUB      0x0B
#define DMCTR2_SUB      0x0C
#define DMSFC2_SUB      0x0D
#define DMMCR2_SUB      0x0E
#define DMSRC3_SUB      0x0F
```

```
#define DMDST3_SUB      0x10
#define DMCTR3_SUB      0x11
#define DMSFC3_SUB      0x12
#define DMMCR3_SUB      0x13
#define DMSRC4_SUB      0x14
#define DMDST4_SUB      0x15
#define DMCTR4_SUB      0x16
#define DMSFC4_SUB      0x17
#define DMMCR4_SUB      0x18
#define DMSRC5_SUB      0x19
#define DMDST5_SUB      0x1A
#define DMCTR5_SUB      0x1B
#define DMSFC5_SUB      0x1C
#define DMMCR5_SUB      0x1D
#define DMSRCP_SUB      0x1E
#define DMDSTP_SUB      0x1F
#define DMIDX0_SUB      0x20
#define DMIDX1_SUB      0x21
#define DMFRI0_SUB      0x22
#define DMFRI1_SUB      0x23
#define DMGSA_SUB       0x24
#define DMGDA_SUB       0x25
#define DMGCR_SUB       0x26
#define DMGFR_SUB       0x27
/* Define the base addresses for autoincrementing */
/* Autoincrementing addresses will be denoted with an A ending */
#define DMSRC0_BASEA    0x56
#define DMSRC1_BASEA    0x56
#define DMSRC2_BASEA    0x56
#define DMSRC3_BASEA    0x56
#define DMSRC4_BASEA    0x56
#define DMSRC5_BASEA    0x56
#define DMDST0_BASEA    0x56
#define DMDST1_BASEA    0x56
#define DMDST2_BASEA    0x56
#define DMDST3_BASEA    0x56
#define DMDST4_BASEA    0x56
#define DMDST5_BASEA    0x56
#define DMCTR0_BASEA    0x56
#define DMCTR1_BASEA    0x56
```

```
#define DMCTR2_BASEA     0x56
#define DMCTR3_BASEA     0x56
#define DMCTR4_BASEA     0x56
#define DMCTR5_BASEA     0x56
#define DMSFC0_BASEA     0x56
#define DMSFC1_BASEA     0x56
#define DMSFC2_BASEA     0x56
#define DMSFC3_BASEA     0x56
#define DMSFC4_BASEA     0x56
#define DMSFC5_BASEA     0x56
#define DMMCR0_BASEA     0x56
#define DMMCR1_BASEA     0x56
#define DMMCR2_BASEA     0x56
#define DMMCR3_BASEA     0x56
#define DMMCR4_BASEA     0x56
#define DMMCR5_BASEA     0x56
#define DMSRCP_BASEA     0x56
#define DMDSTP_BASEA     0x56
#define DMIDX0_BASEA     0x56
#define DMIDX1_BASEA     0x56
#define DMFRI0_BASEA     0x56
#define DMFRI1_BASEA     0x56
#define DMSGA_BASEA              0x56
#define DMGDA_BASEA              0x56
#define DMGCR_BASEA              0x56
#define DMGFR_BASEA              0x56
#define DMSRC0_ADDRA     (*(volatile unsigned int *)     DMSRC0_BASEA)
#define DMSRC1_ADDRA     (*(volatile unsigned int *)     DMSRC1_BASEA)
#define DMSRC2_ADDRA     (*(volatile unsigned int *)     DMSRC2_BASEA)
#define DMSRC3_ADDRA     (*(volatile unsigned int *)     DMSRC3_BASEA)
#define DMSRC4_ADDRA     (*(volatile unsigned int *)     DMSRC4_BASEA)
#define DMSRC5_ADDRA     (*(volatile unsigned int *)     DMSRC5_BASEA)
#define DMDST0_ADDRA     (*(volatile unsigned int *)     DMDST0_BASEA)
#define DMDST1_ADDRA     (*(volatile unsigned int *)     DMDST1_BASEA)
#define DMDST2_ADDRA     (*(volatile unsigned int *)     DMDST2_BASEA)
#define DMDST3_ADDRA     (*(volatile unsigned int *)     DMDST3_BASEA)
#define DMDST4_ADDRA     (*(volatile unsigned int *)     DMDST4_BASEA)
#define DMDST5_ADDRA     (*(volatile unsigned int *)     DMDST5_BASEA)
#define DMCTR0_ADDRA     (*(volatile unsigned int *)     DMCTR0_BASEA)
#define DMCTR1_ADDRA     (*(volatile unsigned int *)     DMCTR0_BASEA)
```

```
#define DMCTR2_ADDRA      (*(volatile unsigned int *)      DMCTR0_BASEA)
#define DMCTR3_ADDRA      (*(volatile unsigned int *)      DMCTR0_BASEA)
#define DMCTR4_ADDRA      (*(volatile unsigned int *)      DMCTR0_BASEA)
#define DMCTR5_ADDRA      (*(volatile unsigned int *)      DMCTR0_BASEA)
#define DMSFC0_ADDRA      ((volatile DMSFCn_REG *)        ((char *) DMSFC0_BASEA))
#define DMSFC1_ADDRA      ((volatile DMSFCn_REG *)        ((char *) DMSFC1_BASEA))
#define DMSFC2_ADDRA      ((volatile DMSFCn_REG *)        ((char *) DMSFC2_BASEA))
#define DMSFC3_ADDRA      ((volatile DMSFCn_REG *)        ((char *) DMSFC3_BASEA))
#define DMSFC4_ADDRA      ((volatile DMSFCn_REG *)        ((char *) DMSFC4_BASEA))
#define DMSFC5_ADDRA      ((volatile DMSFCn_REG *)        ((char *) DMSFC5_BASEA))
#define DMMCR0_ADDRA      ((volatile DMMCRn_REG *)        ((char *) DMMCR0_BASEA))
#define DMMCR1_ADDRA      ((volatile DMMCRn_REG *)        ((char *) DMMCR1_BASEA))
#define DMMCR2_ADDRA      ((volatile DMMCRn_REG *)        ((char *) DMMCR2_BASEA))
#define DMMCR3_ADDRA      ((volatile DMMCRn_REG *)        ((char *) DMMCR3_BASEA))
#define DMMCR4_ADDRA      ((volatile DMMCRn_REG *)        ((char *) DMMCR4_BASEA))
#define DMMCR5_ADDRA      ((volatile DMMCRn_REG *)        ((char *) DMMCR5_BASEA))
#define DMSRCP_ADDRA      ((volatile DMSRCP_REG *)        ((char *) DMSRCP_BASEA))
#define DMDSTP_ADDRA      ((volatile DMDSTP_REG *)        ((char *) DMDSTP_BASEA))
#define DMIDX0_ADDRA      (*(volatile unsigned int *)      DMIDX0_BASEA)
#define DMIDX1_ADDRA      (*(volatile unsigned int *)      DMIDX1_BASEA)


#define DMFRI0_ADDRA      (*(volatile unsigned int *)      DMFRI0_BASEA)
#define DMFRI1_ADDRA      (*(volatile unsigned int *)      DMFRI1_BASEA)
#define DMSGA_ADDRA       (*(volatile unsigned int *)      DMSGA_BASEA)
#define DMGDA_ADDRA       (*(volatile unsigned int *)      DMGDA_BASEA)
#define DMGCR_ADDRA       (*(volatile unsigned int *)      DMGCR_BASEA)
#define DMGFR_ADDRA       (*(volatile unsigned int *)      DMGFR_BASEA)
/* Define the base addresses without autoincrementing  */
#define DMSRC0_BASE       0x57
#define   DMSRC1_BASE     0x57
#define   DMSRC2_BASE     0x57
#define   DMSRC3_BASE     0x57
#define   DMSRC4_BASE     0x57
#define   DMSRC5_BASE     0x57
#define   DMDST0_BASE     0x57
#define   DMDST1_BASE     0x57
#define   DMDST2_BASE     0x57
#define   DMDST3_BASE     0x57
#define   DMDST4_BASE     0x57
#define DMDST5_BASE       0x57
```

```
#define DMCTR0_BASE        0x57
#define DMCTR1_BASE        0x57
#define DMCTR2_BASE        0x57
#define DMCTR3_BASE        0x57
#define DMCTR4_BASE        0x57
#define DMCTR5_BASE        0x57
#define DMSFC0_BASE        0x57
#define DMSFC1_BASE        0x57
#define DMSFC2_BASE        0x57
#define DMSFC3_BASE        0x57
#define DMSFC4_BASE        0x57
#define DMSFC5_BASE        0x57
#define DMMCR0_BASE        0x57
#define DMMCR1_BASE        0x57
#define DMMCR2_BASE        0x57
#define DMMCR3_BASE        0x57
#define DMMCR4_BASE        0x57
#define DMMCR5_BASE        0x57
#define DMSRCP_BASE        0x57
#define DMDSTP_BASE        0x57
#define DMIDX0_BASE        0x57
#define DMIDX1_BASE        0x57
#define DMFRI0_BASE        0x57
#define DMFRI1_BASE        0x57
#define DMSGA_BASE         0x57
#define DMGDA_BASE         0x57
#define DMGCR_BASE         0x57
#define DMGFR_BASE         0x57
#define DMSRC0_ADDR        (*(volatile unsigned int *)    DMSRC0_BASE)
#define DMSRC1_ADDR        (*(volatile unsigned int *)    DMSRC1_BASE)
#define DMSRC2_ADDR        (*(volatile unsigned int *)    DMSRC2_BASE)
#define DMSRC3_ADDR        (*(volatile unsigned int *)    DMSRC3_BASE)
#define DMSRC4_ADDR        (*(volatile unsigned int *)    DMSRC4_BASE)
#define DMSRC5_ADDR        (*(volatile unsigned int *)    DMSRC5_BASE)

#define DMDST0_ADDR        (*(volatile unsigned int *)    DMDST0_BASE)
#define DMDST1_ADDR        (*(volatile unsigned int *)    DMDST1_BASE)
#define DMDST2_ADDR        (*(volatile unsigned int *)    DMDST2_BASE)
#define DMDST3_ADDR        (*(volatile unsigned int *)    DMDST3_BASE)
#define DMDST4_ADDR        (*(volatile unsigned int *)    DMDST4_BASE)
```

```
#define DMDST5_ADDR      (*(volatile unsigned int *)     DMDST5_BASE)


#define DMCTR0_ADDR      (*(volatile unsigned int *)     DMCTR0_BASE)
#define DMCTR1_ADDR      (*(volatile unsigned int *)     DMCTR1_BASE)
#define DMCTR2_ADDR      (*(volatile unsigned int *)     DMCTR2_BASE)
#define DMCTR3_ADDR      (*(volatile unsigned int *)     DMCTR3_BASE)
#define DMCTR4_ADDR      (*(volatile unsigned int *)     DMCTR4_BASE)
#define DMCTR5_ADDR      (*(volatile unsigned int *)     DMCTR5_BASE)
#define DMSFC0_ADDR      ((volatile DMSFCn_REG *)        ((char *) DMSFC0_BASE))
#define DMSFC1_ADDR      ((volatile DMSFCn_REG *)        ((char *) DMSFC1_BASE))
#define DMSFC2_ADDR      ((volatile DMSFCn_REG *)        ((char *) DMSFC2_BASE))
#define DMSFC3_ADDR      ((volatile DMSFCn_REG *)        ((char *) DMSFC3_BASE))
#define DMSFC4_ADDR      ((volatile DMSFCn_REG *)        ((char *) DMSFC4_BASE))
#define DMSFC5_ADDR      ((volatile DMSFCn_REG *)        ((char *) DMSFC5_BASE))
#define DMMCR0_ADDR      ((volatile DMMCRn_REG *)        ((char *) DMMCR0_BASE))
#define DMMCR1_ADDR      ((volatile DMMCRn_REG *)        ((char *) DMMCR1_BASE))
#define DMMCR2_ADDR      ((volatile DMMCRn_REG *)        ((char *) DMMCR2_BASE))
#define DMMCR3_ADDR      ((volatile DMMCRn_REG *)        ((char *) DMMCR3_BASE))
#define DMMCR4_ADDR      ((volatile DMMCRn_REG *)        ((char *) DMMCR4_BASE))
#define DMMCR5_ADDR      ((volatile DMMCRn_REG *)        ((char *) DMMCR5_BASE))
#define DMSRCP_ADDR      ((volatile DMSRCP_REG *)        ((char *) DMSRCP_BASE))
#define DMDSTP_ADDR      ((volatile DMDSTP_REG *)        ((char *) DMDSTP_BASE))
#define DMIDX0_ADDR      (*(volatile unsigned int *)     DMIDX0_BASE)
#define DMIDX1_ADDR      (*(volatile unsigned int *)     DMIDX1_BASE)


#define DMFRI0_ADDR      (*(volatile unsigned int *)     DMFRI0_BASE)
#define DMFRI1_ADDR      (*(volatile unsigned int *)     DMFRI1_BASE)
#define DMSGA_ADDR       (*(volatile unsigned int *)     DMSGA_BASE)
#define DMGDA_ADDR       (*(volatile unsigned int *)     DMGDA_BASE)
#define DMGCR_ADDR       (*(volatile unsigned int *)     DMGCR_BASE)
#define DMGFR_ADDR       (*(volatile unsigned int *)     DMGFR_BASE)
/*------------------------------------------------------------------*/
/* DMPREC */
/*------------------------------------------------------------------*/
typedef union {
        struct {
                unsigned int free:1;
                unsigned int rsvd:1;
                unsigned int dprc:6;
                unsigned int intosel:2;
```

```
                unsigned int de:6;
                } bitval;
        unsigned int value;
} DMPREC_REG;
/*-----------------------------------------------------------------------*/
/* DMFCn */
/*-----------------------------------------------------------------------*/
typedef union {
        struct {
                unsigned int dsyn:4;
                unsigned int dblw:1;
                unsigned int rsrvd:3;
                unsigned int framecount:8;
                } bitval;
        unsigned int value;
} DMSFCn_REG;
/*-----------------------------------------------------------------------*/
/* DMMCRn */
/*-----------------------------------------------------------------------*/
typedef union {
        struct {
                unsigned int autoinit:1;
                unsigned int dinm:1;
                unsigned int imod:1;
                unsigned int ctmod:1;
                unsigned int rsrvd1:1;
                unsigned int sind:2;
                unsigned int dms:2;
                unsigned int rsrvd2:1;
                unsigned int dind:3;
                unsigned int dmd:2;
                } bitval;
        unsigned int value;
} DMMCRn_REG;
/*-----------------------------------------------------------------------*/
/* DMSRCP */
/*-----------------------------------------------------------------------*/
typedef union {
        struct {
                unsigned int rsvd        :9;
```

```
                unsigned int source      :7;

                } bitval;

        unsigned int value;

} DMSRCP_REG;
/*--------------------------------------------------------------------*/
/* DMDSTP */
/*--------------------------------------------------------------------*/
typedef union {

        struct {

                unsigned int rsvd        :9;

                unsigned int dest        :7;

                } bitval;

        unsigned int value;

} DMDSTP_REG;
```

Appendix C   dma_src.c

```
/*******************************************************************/
/*                                                               */
/*Function:   DMA_reset()                                        */
/*file name:  dma_src.c                                          */
/*Description: This function performs a reset of all DMA  registers. */
/*Inputs:      None                                              */
/*Outputs:     None                                              */
/*Returns:     None                                              */
/* AUTHOR:     AAP Application Group, L. Philipose, Dallas        */
/*             CREATED 2000(C) BY TEXAS INSTRUMENTS INCORPORATED.  */
/*******************************************************************/
#include "c5402Reg.h"
void DMA_reset(void)
{  /* DMA Initialization */
  /* Zero all Frame and Sync Registers */

  /* Clear Channel 0 */
  DMSBA_ADDR = DMSFC0_SUB;        /* Subaddress Frame-Sync       */
  DMSFC0_ADDR->value = 0x0;

  DMSBA_ADDR = DMSRC0_SUB;        /* Subaddress Source           */
  DMSRC0_ADDR = 0x0;

  /* Clear Channel 1 */
  DMSBA_ADDR = DMSFC1_SUB;        /* Subaddress Frame/Sync       */
  DMSFC1_ADDR->value = 0x0;

  DMSBA_ADDR = DMSRC1_SUB;        /* Subaddress Source           */
  DMSRC1_ADDR = 0x0;

  /* Clear Channel 2 */
  DMSBA_ADDR = DMSFC2_SUB;        /* Subaddress Frame/Sync       */
  DMSFC2_ADDR->value = 0x0;

  DMSBA_ADDR = DMSRC2_SUB;        /* Subaddress Source           */
  DMSRC2_ADDR = 0x0;

  /* Clear Channel 3 */
  DMSBA_ADDR = DMSFC3_SUB;        /* Subaddress Frame/Sync       */
  DMSFC3_ADDR->value = 0x0;

  DMSBA_ADDR = DMSRC3_SUB;        /* Subaddress Source           */
  DMSRC3_ADDR = 0x0;
```

```
 /* Clear Channel 4 */
DMSBA_ADDR = DMSFC4_SUB;     /* Subaddress Frame/Sync        */
DMSFC4_ADDR->value = 0x0;

DMSBA_ADDR = DMSRC4_SUB;     /* Subaddress Source            */
DMSRC4_ADDR = 0x0;

/* Clear Channel 5 */
DMSBA_ADDR = DMSFC5_SUB;     /* Subaddress Frame/Sync        */
DMSFC5_ADDR->value = 0x0;

DMSBA_ADDR = DMSRC5_SUB;     /* Subaddress Source            */
DMSRC5_ADDR = 0x0;

/* General Stop/Reset of DMA */
DMAPREC_ADDR->value = 0x0;  /* STOP all DMA channels         */

DMSBA_ADDR = DMSRCP_SUB;     /* Subaddress Source Program Page Address      */
DMSRCP_ADDRA->value = 0x0;  /* Use autoincrement address                    */
DMDSTP_ADDRA->value = 0x0;  /* Destination page (Auto)                      */
DMIDX0_ADDRA = 0x0;          /* Index Register 0 (Auto)                      */
DMIDX1_ADDRA = 0x0;          /* Index Register 1 (Auto)                      */
DMFRI0_ADDRA = 0x0;          /* Frame Index Register 0 (Auto)                */
DMFRI1_ADDRA = 0x0;          /* Frame index Register 1 (Auto)                */
DMSGA_ADDRA = 0x0;           /* Global Source Address Reload Register (Auto) */
DMGDA_ADDRA = 0x0;         /* Global Destination Address Reload Register (Auto)*/
DMGCR_ADDRA = 0x0;         /* Global Count Reload Register (Auto)           */
DMGFR_ADDRA = 0x0;         /* Global Frame Count Reload Register (Auto)     */


/***********************************************************************/
/*                                                                     */
/*DMA_init                                                             */
/*file name:   dma_src.c                                               */
/*Description     This function performs the initialization of all     */
/*                DMA channels. It is capable of initializing the      */
/*                channels one at a time.                              */
/*                                                                     */
/*Inputs:       channel – selects the channel to be initialized 0,1    */
/*         source – source of data (input/read)                        */
/*         destination – destination of data (output/write)            */
/*         count – number of DMA transfers to be performed             */
/*         frame_sync – element that initiates each transfer           */
/*         control_mode – transfer mode control                        */
/*                                                                     */
/*Outputs:  None                                                       */
```

```
/*Returns:   None                                                    */
/*Note:      Although DMA_init sets up the channel, it does NOT start */
/*           the DMA channel running. This must be done separately by */
/*           modifying the DMPREC register.                           */
/*                                                                    */
/* AUTHOR: AAP Application Group, L. Philipose, Dallas                */
/*         CREATED 2000(C) BY TEXAS INSTRUMENTS INCORPORATED.         */
/*********************************************************************/
void DMA_init(  unsigned int channel,
                unsigned int source,
                unsigned int destination,
                unsigned int count,
                unsigned int frame_sync,
                unsigned int control_mode)
{
    switch(channel)
    {
        case 0: /*Initialize DMA channel 0 Registers*/
                DMSBA_ADDR = DMSRC0_SUB;        /* Sub Address Register */
                DMSRC0_ADDRA = source;
                DMDST0_ADDRA = destination;
                DMCTR0_ADDRA = count; DMSFC0_ADDRA-
                >value = frame_sync; DMMCR0_ADDRA-
                >value = control_mode; break;


        case 1:  /*Initialize DMA channel 1 Registers*/
                DMSBA_ADDR = DMSRC1_SUB;        /* Sub Address Register */
                DMSRC1_ADDRA = source;
                DMDST1_ADDRA = destination;
                DMCTR1_ADDRA = count; DMSFC1_ADDRA-
                >value = frame_sync; DMMCR1_ADDRA-
                >value = control_mode; break;



    }
}
```

Appendix D Mcbspsrc.c

```
/*******************************************************************/
/*                                                                 */
/*Function:   MCBSP1_init()                                         */
/*                                                                 */
/*file name:  mcbspsrc.c                                            */
/*Description: This function performs the initialization of McBSP1  */
/*    Registers. It begins by resetting the Frame-sync Generator,   */
/*    Sample-Rate Generator, Transmitter and Receiver. Once         */
/*    registers are programmed in enables the Frame-sync and Sample */
/*    rate generators only.                                         */
/*                                                                 */
/*Inputs:   Address of McBSP object. This structure contains all    */
/*          the McBSP Registers and their desired values.           */
/*Outputs:  None                                                    */
/*Returns:  None                                                    */
/*Note:  Although MCBSP_init sets up the channel, it does NOT        */
/*       Enable the Transmitter or Receiver. This is done by modifying */
/*       bit 0 of SPCR1(for Receiver) and SPCR2(for Transmitter).   */
/*       The Multi-channel registers are NOT currently not programmed. */
/*       Those lines were commented out simple to save cycles, because */
/*       we currently don't use the multi-channel McBSP features.   */
/*AUTHOR:   AAP Application Group, L. Philipose, Dallas             */
/*          CREATED 2000(C) BY TEXAS INSTRUMENTS INCORPORATED.      */
/*******************************************************************/
#include "c5402Reg.h"
void MCBSP1_init(MCBSP *pMcBSP)
{
   unsigned int i=0;

    /* McBSP 1    Place McBSP into reset*/
            SPSA1_ADDR = SPCR11_SUB;   /* Clear the RRST bit            */
             SPCR11_ADDR->value = pMcBSP->RegVal.SPCR1_REG & 0xFFFE;

            SPSA1_ADDR = SPCR21_SUB;   /* Clear the XRST FRST GRST bits */
             SPCR21_ADDR->value = pMcBSP->RegVal.SPCR2_REG & 0xFF3E;

            for (i=0;i<5;i++);          /*Wait for transmitter and Receiver */
                                        /*to Power Off                      */

            SPSA1_ADDR = RCR11_SUB;
            RCR11_ADDR->value = pMcBSP->RegVal.RCR1_REG  ;
```

```
        SPSA1_ADDR = RCR21_SUB;
        RCR21_ADDR->value = pMcBSP->RegVal.RCR2_REG;

        SPSA1_ADDR = XCR11_SUB;
        XCR11_ADDR->value = pMcBSP->RegVal.XCR1_REG;
        SPSA1_ADDR = XCR21_SUB;
        XCR21_ADDR->value = pMcBSP->RegVal.XCR2_REG;
        SPSA1_ADDR = SRGR11_SUB;
        SRGR11_ADDR->value = pMcBSP->RegVal.SRGR1_REG;
        SPSA1_ADDR = SRGR21_SUB;
        SRGR21_ADDR->value = pMcBSP->RegVal.SRGR2_REG;
/*  Since we are not currently using the multichannel
 * capabilities of the McBSP, these registers will not be
 *  programmed.
*/
/*              SPSA1_ADDR = MCR11_SUB;
                MCR11_ADDR->value = pMcBSP->RegVal.MCR1_REG ;

                SPSA1_ADDR = MCR21_SUB;
                MCR21_ADDR->value = pMcBSP->RegVal.MCR2_REG;

                SPSA1_ADDR = RCERA1_SUB;
                RCERA1_ADDR->value = pMcBSP->RegVal.RCERA_REG;

                SPSA1_ADDR = RCERB1_SUB;
                RCERB1_ADDR->value = pMcBSP->RegVal.RCERB_REG;

                SPSA1_ADDR = XCERA1_SUB;
                XCERA1_ADDR->value = pMcBSP->RegVal.XCERA_REG;

                SPSA1_ADDR = XCERB1_SUB;
                XCERB1_ADDR->value = pMcBSP->RegVal.XCERB_REG ;
                SPSA1_ADDR = PCR1_SUB;
                PCR1_ADDR->value = pMcBSP->RegVal.PCR_REG;

/*Enable Sample-Rate Generator and Frame-Sync Generator*/

     SPSA1_ADDR = SPCR21_SUB;       /*Choose Serial Port Control Register 2*/
     SPCR21_ADDR->bitval.frst = 1; /*Frame-Sync generator pulled out of Reset*/
     SPCR21_ADDR->bitval.grst = 1; /*Sample-Rate Generator pulled out of Reset*/

}
```

Appendix E   adc_const.h

```
/*********************************************************************/
/*                                                                  */
/*file name:    adc_const.h                                         */
/*Description: Contains various DSP and ADC parameters              */
/*Inputs:       None                                                */
/*Outputs:      None                                                */
/*Returns:      None                                                */
/*Note:         None                                                */
/*AUTHOR:       AAP Application Group, L. Philipose, Dallas         */
/*              CREATED 2000(C) BY TEXAS INSTRUMENTS INCORPORATED.  */
/*********************************************************************/


/*DSP Parameters*/
#define    DSP_FREQ      (100)    /* In MHz                         */
#define CPU_100MHZ   (0x4007) /* Set  C5402  DSK  to  run  at  100MHz   */
#define SERCLOCK (5) /* Serial clock speed in MHz  */


/*ADC control words*/
#define ADChan0 0x0000 //5V & ext ref
#define ADChan1 0x0080 //5V & ext ref
#define ADChan4 0x0200 //5V & ext ref
#define ADChan5 0x0280 //5V & ext ref
#define ADChan6 0x0300 //5V & ext ref
#define ADChan7 0x0380 //5V & ext ref
```

Appendix F   C5402 Memory Mapping

```
/*File:  dma_tlv1570.cmd                           */
/*Description: 'C5402 DSK Memory Mapping          */
MEMORY
{
    PAGE 0:  PRAM0:     origin = 0x0060 length = 0x04FF
    PAGE 0:  CODE:      origin = 0x0560 length = 0x3000
    PAGE 0:  VECTOR:    origin = 0x3600 length = 0x0080
    PAGE 1:  DRAM0:     origin = 0x0060 length = 0x0FA0
    PAGE 1:  DATA:      origin = 0x1000 length = 0x3000
}
SECTIONS
{
    .vectors  > VECTOR  PAGE 0   /*Typically where User Interrupt Vector  */
                                 /*Table is stored                        */
    .text    > CODE   PAGE 0     /*Typically where source code is located */
    .cinit   > CODE   PAGE 0
    .bss     > PRAM0  PAGE 0
}
```