*Application Report*
*SPRA528A - February 2005*

# *TMS320C6000 McBSP: AC’97 Codec Interface (TLV320AIC27)*

*Joe Hansbauer*
*Rebecca Ma*
*Vassos Soteriou*

*Digital Signal Processing Solutions*

**ABSTRACT**

This document describes how to use the multichannel buffered serial ports (McBSPs) in the Texas Instruments TMS320C6000™ digital signal processor (DSP) as a digital controller for an audio codec 1997 device.

The McBSP is connected to a stereo audio codec 1997 device. This application report uses the TLV320AIC27 audio codec (AIC27) as an example. The audio codec 1997 (AC’97) standard specifies a five-signal digital connection between the McBSP and the AIC27. These five signals are the SYNC, BITCLK, SDATAOUT, SDATAIN, and RESETB signals of the AC’97 device. In this AC’97 interface, the McBSP operates as an audio codec controller. This application report discusses the configuration of these signals in detail. The hardware schematic discussed in this document is a possible solution for the AC’97 interface. The sample code, provided in Appendix NO TAG at the end of this application report, has been tested on a specially configured Texas Instruments TMS320C6000 board by connecting two McBSP ports together: one of the McBSPs has been set to act as a AC’97 device to emulate the presence of an AC’97 device interfaced to a McBSP port, but has not been tested using an actual TMS320C6000 McBSP-TLV320AIC27 interface. The sample code described in this application report can be downloaded from http://www.ti.com/lit/zip/SPRA528.

**Contents**

**List of Figures**

**List of Tables**

## 1 Design Problem

How can the multichannel buffered serial port (McBSP) in the TMS320C6000 be used as a digital controller for an audio codec 1997 (AC’97) device?

## 2 Overview

The McBSP can operate as a digital controller for an audio codec device that is compliant to the audio codec 1997 (AC’97) component specification. The AC’97 standard specifies a 5-wire digital serial link, or “AC-Link”, between the audio codec device and its digital controller. Figure 1 shows the block diagram for the AC’97 system architecture.

An AC’97 device can perform digital-to-analog conversion, analog-to-digital conversion, and analog input mixing. It supports different analog audio inputs/outputs and can communicate with a digital controller through the AC-Link, as mentioned above. This application report discusses the digital interface between the AC’97 device, TLV320AIC27, and the McBSP of the TMS320C6000 digital controller. Analog input/output descriptions are not within the scope of this application report. (See the *Stereo Audio Codec* data sheet (SLAS253) for information concerning the analog input/output of the audio codec.)

Because the TLV320AIC27 can interface to a 3.3-V digital controller, no voltage translation is necessary in the interface of the McBSP and the AIC27. The digital supply voltage, $DV_{DD}$, of the AIC27 must be 3.3 V.

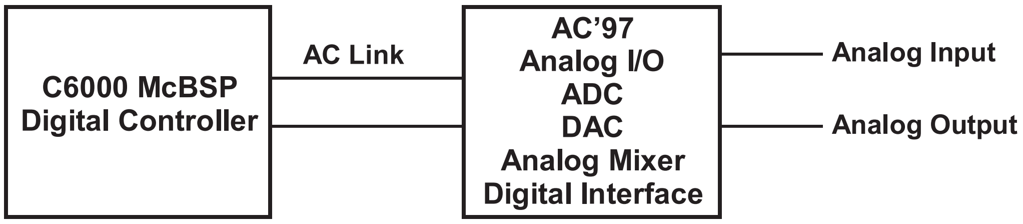

**Figure 1. AC’97 System Architecture**

## 3 Solution

### 3.1 Hardware Interface

To successfully use the McBSP as an audio codec digital controller, you must configure five AC'97 signals: SYNC, SDATAOUT, RESETB, BITCLK, and SDATAIN. The audio codec AIC27 generates two of these AC'97 signals, BITCLK and SDATAIN, which are connected to the McBSP CLKS and DR pins, respectively. The McBSP generates the remaining three AC'97 signals, SYNC, SDATAOUT, and RESETB. Table 1 summarizes the signals of this 5-wire digital serial link, or the AC-Link.

The audio codec device generates the clock for the AC'97 interface. A 24.576-MHz crystal is connected to the TLV320AIC27 audio codec. The AIC27 internally divides this fixed-rate clock signal by a factor of two to produce a 12.288-MHz clock signal output on the BITCLK pin. As shown in Figure 2, this BITCLK signal is connected to the CLKS pin of the McBSP and is used to drive the McBSP's sample rate generator.

The McBSP also derives the frame sync signal FSX from BITCLK. FSX is connected to the input pin SYNC of the AIC27. In compliance with the AC'97 protocol specification, you must set the period of the McBSP sample rate generator to 256 BITCLK cycles to generate a frame sync signal FSX every 256 CLKS cycles, at a fixed rate of 48 kHz. The width of the frame sync signal FSX is 16 BITCLK cycles. The period and width of the frame sync signal are defined in the FPER and FWID fields of the sample rate generator register (SRGR) of the McBSP.

The SDATAOUT, and SDATAIN pins of the AIC27 are connected to the DX and DR pins of the McBSP, respectively. The McBSP receives and transmits data in dual-phase frames. In each case, the first phase consists of one 16-bit element; the second phase consists of twelve 20-bit elements. The user can configure the receive and transmit operation of the McBSP in the receive and transmit control registers (RCR and XCR).

As the name implies, the RESETB signal to AIC27 resets the audio codec and brings the TLV320AIC27 out of the power-down mode. The AC'97 standard specifies two kinds of reset—Cold AC'97 Reset and Warm AC'97 Reset. Depending on the application, the user has the freedom to configure any of the general-purpose outputs of the C6000 DSP to drive the RESETB input to the AIC27. This application report shows the TOUT0 general-purpose output pin of the McBSP as the RESETB signal to the AIC27.

Figure 2 shows the 5-pin hardware interface between the McBSP and the AC'97 audio codec. Table 1 describes these AC-Link signals.

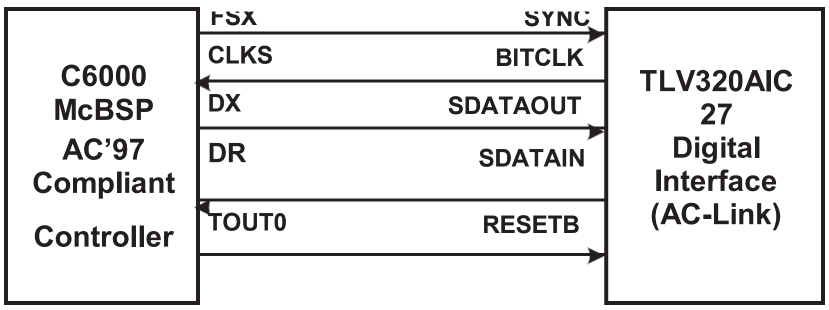

**Figure 2. Digital Interface Between McBSP and AC'97 Device**

**Table 1. McBSP and AIC27 Interface Pin Description**

| C6000 McBSP Pin | AIC27 Pin | Description |
|---|---|---|
| FSX (output) | SYNC (input) | FSX marks the beginning of each frame. FSX is generated by the McBSP at a fixed rate of 48 kHz. |
| CLKS (input) | BITCLK (output) | The AIC27 generates BITCLK at a fixed rate of 12.288 MHz. CLKS is the input clock to the McBSP's sample rate generator. |
| DX (output) | SDATAOUT (input) | Serial data goes from the McBSP to the AIC27 device. Data is captured by the AIC27 codec on every falling edge of BITCLK. |
| DR (input) | SDATAIN (output) | Serial data goes from the AIC27 codec to the McBSP. Data is transmitted by the AIC27 codec on every rising edge of BITCLK. |
| TOUT0 (output) | RESETB (input) | The reset signal generated by the McBSP wakes up the AIC27 codec from the power-down mode. |

## 3.2 McBSP Register Configuration

The setup of the programmable control registers of the McBSP for the McBSP and AIC27 interface is shown in Figure 3 through Figure 6. Table 2 lists and describes the register values for the interface. Note that although pin FSR is not used for the hardware interface, the FSRM field in the pin control register still must be set to 1 to indicate that the receive frame synchronization signals are generated internally by the sample rate generator.

| 31 | 30 – 24 | 23 – 21 | 20 – 19 | 18 | 17 – 16 |
|---|---|---|---|---|---|
| RPHASE | RFRLEN2 | RWDLEN2 | RCOMPAND | RFIG | RDATDLY |
| 1 | 0x0B | 0x3 | 0 | 0 | 0x1 |

| 15 | 14 – 8 | 7 – 5 | 4 | 3 – 0 |
|---|---|---|---|---|
| Reserved | RFRLEN1 | RWDLEN1 | RWDREVRS | Reserved |
| 0 | 0 | 0x2 | 0 | 0 |

**Figure 3. Receive Control Register (RCR)**

| 31 | 30 – 24 | 23 – 21 | 20 – 19 | 18 | 17 – 16 |
|---|---|---|---|---|---|
| XPHASE | XFRLEN2 | XWDLEN2 | XCOMPAND | XFIG | XDATDLY |
| 1 | 0x0B | 0x3 | 0 | 0 | 0x1 |

| 15 | 14 – 8 | 7 – 5 | 4 | 3 – 0 |
|---|---|---|---|---|
| Reserved | XFRLEN1 | XWDLEN1 | XWDREVRS | Reserved |
| 0 | 0 | 0x2 | 0 | 0 |

**Figure 4. Transmit Control Register (XCR)**

| 31 | 30 | 29 | 28 | 27 – 16 |
|---|---|---|---|---|
| GSYNC | CLKSP | CLKSM | FSGM | FPER |
| 0 | 0 | 0 | 1 | 0xFF |

| 15 – 8 | 7 – 0 |
|---|---|
| FWID | CLKGDV |
| 0x0F | 0x0 |

**Figure 5. Sample Rate Generator Register (SRGR)**

| 31–16 |
|---|
| Reserved |
| 0x0000 |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Rsvd | | XIOEN | RIOEN | FSXM | FSRM | CLKXM | CLKRM |
| 0 | | 0 | 0 | 1 | 1 | 0 | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Rsvd | CLKS_STAT | DX_STAT | DR_STAT | FSXP | FSRP | CLKXP | CLKRP |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Figure 6. Pin Control Register (PCR)**

**Table 2. Bit-Field Values for McBSP Registers**

| Register (Bit-Field No.) | Bit Field Name | Value in Hex | HAL Macro | Function Description |
|---|---|---|---|---|
| RCR[31] | RPHASE | 1 | MCBSP_RCR_RPHASE_DUAL | Dual phase receive frame |
| XCR[31] | XPHASE | 1 | MCBSP_XCR_XPHASE_DUAL | Dual phase transmit frame |
| RCR[30:24] | RFRLEN2 | B | MCBSP_RCR_RFRLEN2_OF(0xB) | Phase 2 receive frame length = 12 elements |
| XCR[30:24] | XFRLEN2 | B | MCBSP_XCR_XFRLEN2_OF(0xB) | Phase 2 transmit frame length = 12 elements |
| RCR[23:21] | RWDLEN2 | 3 | MCBSP_RCR_RWDLEN2_20BIT | Phase 2 receive elements = 20 bits |
| XCR[23:21] | XWDLEN2 | 3 | MCBSP_XCR_XWDLEN2_20BIT | Phase 2 Transmit elements = 20 bits |
| RCR[17:16] | RDATDLY | 1 | MCBSP_RCR_RDATDLY_1BIT | 1-bit receive data delay. TAG phase data begins after one BITCLK delay |
| XCR{17:16] | XDATDLY | 1 | MCBSP_XCR_XDATDLY_1BIT | 1-bit transmit data delay. TAG phase data begins after one BITCLK delay |
| RCR[14:8] | RFRLEN1 | 0 | MCBSP_RCR_RFRLEN1_OF(0x0) | Phase 1 receive frame length = 1 element |
| XCR[14:8] | XFRLEN1 | 0 | MCBSP_XCR_XFRLEN1_OF(0x0) | Phase 1 transmit frame length = 1 element |
| RCR[7:5] | RWDLEN1 | 2 | MCBSP_RCR_RWDLEN1_16BIT | Phase 1 receive elements = 16 bits |
| XCR[7:5] | XWDLEN1 | 2 | MCBSP_XCR_XWDLEN1_16BIT | Phase 1 transmit elements = 16 bits |
| SRGR[31] | GSYNC | 0 | MCBSP_SRGR_GSYNC_FREE | Sample rate generator clock is free running, driven by external clock CLKS |
| SRGR[29] | CLKSM | 0 | MCBSP_SRGR_CLKSM_CLKS | Sample rate generator clock is derived from external clock source CLKS |
| SRGR[28] | FSGM | 1 | MCBSP_SRGR_FSGM_FSG | FSX driven by sample rate generator frame sync signal |
| SRGR[27:16] | FPER | FF | MCBSP_SRGR_FPER_OF(0xFF) | Frame period = 256 CLKS(12.288 MHz) periods (to get 48-kHz FSX) |
| SRGR[15:8] | FWID | F | MCBSP_SRGR_FWID_OF(0xF) | Frame sync signal width = 16 BITCLK |

**Table 2. Bit-Field Values for McBSP Registers (Continued)**

| Register (Bit-Field No.) | Bit Field Name | Value in Hex | HAL Macro | Function Description |
|---|---|---|---|---|
| SRGR[7:0] | CLKGDV | 0 | MCBSP_SRGR_CLKDV_OF(0x0) | CLKG has same frequency as sample rate generator input clock CLKS |
| PCR[11] | FSXM | 1 | MCBSP_PCR_FSXM_INTERNAL | Frame synchronization generated internally |
| PCR[10] | FSRM | 1 | MCBSP_PCR_FSRM_INTERNAL | Frame synchronization generated internally |
| PCR[3] | FSXP | 0 | MCBSP_PCR_FSXP_ACTIVEHIG H | FSX is active-high |
| PCR[2] | FSRP | 0 | MCBSP_PCR_FSRP_ACTIVEHIG H | FSR is active-high |

The bit-fields and registers not listed in Table 2 assume their default values. The user is responsible to set some of the register fields if initial state is different from the default.

## 3.3 Timing Diagram

The AC-Link architecture of the TLV320AIC27 audio codec employs a dual-phase frame format. It divides each 256-bit frame (both transmit and receive) into two phases. The first phase is the 16-bit TAG Phase. As shown in Figure 7, for an audio output frame (SDATAOUT), where data goes from the McBSP to the codec, the first bit of the TAG Phase is the Valid Frame bit. When the Valid Frame bit is one, it indicates that there is at least one TDM slot containing valid data in this frame. For an audio input frame (SDATAIN) where data goes from the codec to the McBSP, the first bit of the TAG Phase indicates whether the codec is ready. The next twelve bits of the TAG Phase indicate whether there is valid data in the corresponding slot in the second phase.

The second phase is the DATA Phase. The DATA Phase consists of twelve 20-bit slots, for a total transmission of 12 X 20 = 240 bits. These slots contain control and audio data. (Refer to the *Stereo Audio Codec* data sheet (SLAS253) for a complete description of the frame contents.) Figure 7 shows the timing diagram for this AC’97 dual-phase frame format.

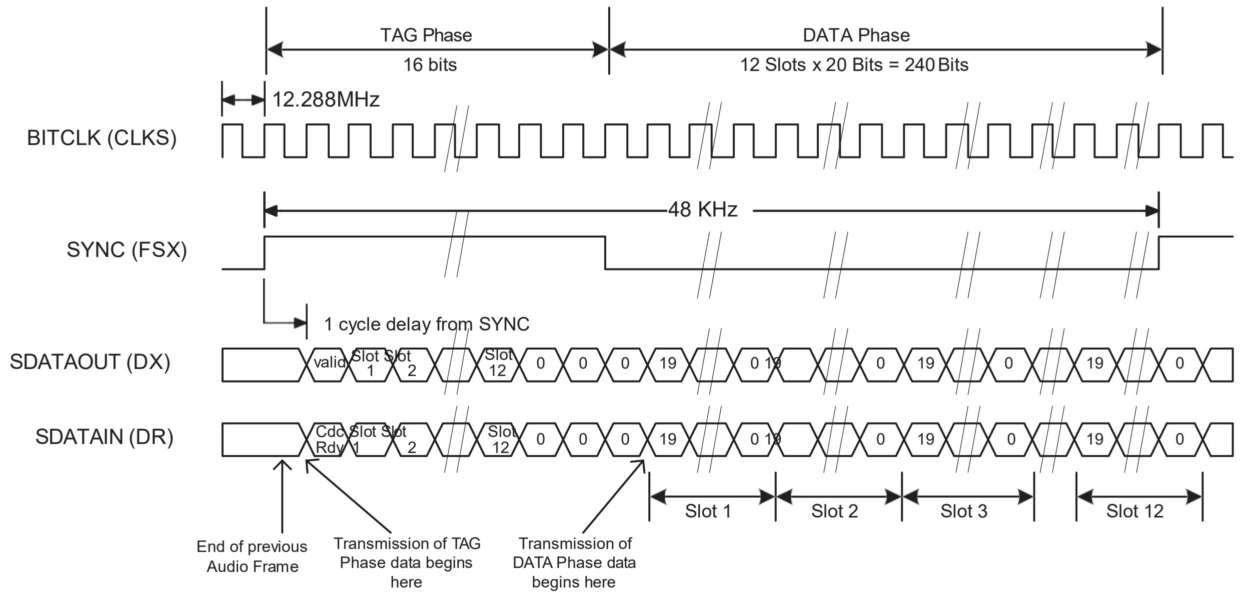

**Figure 7. Timing Diagram for AC'97 Dual-Phase Frame Format**

The FSX signal, generated by the McBSP, is synchronized to the rising edge of BITCLK, and is sampled by the AIC27 on the falling edge of BITCLK. The TAG phase data transfer begins immediately on the rising edge of the next BITCLK signal. For this reason, the RDATDLY and XDATDLY bits in the RCR and XCR registers, respectively, are set to 1 to indicate the 1 BITCLK delay from the rising edge of FSX to the beginning of the data transfer.

## 3.4 McBSP Initialization

Typically, the DMA or EDMA (depending on the TMS320C6000 DSP device used) service the McBSP by controlling internal data flow to and from the McBSP. The following steps describe the procedures necessary for initializing the DMA or EDMA, the McBSP, and the interrupts.

1. Reset the audio codec by asserting the AIC27 RESETB signal low for a minimum of 1us. In this application report, the C6000™ general purpose output pin, TOUT0, is used as the reset signal.
2. If the McBSP is not in reset state, set $\overline{\text{XRST}}$ = $\overline{\text{RRST}}$ =0 in SPCR
3. Program the McBSP configuration registers XCR, PCR, RCR and SRGR to the values shown in Table 2. Do not set the $\overline{\text{GRST}}$ bit in the SPCR in this step.
4. Take the sample rate generator (SRGR) out of reset by setting $\overline{\text{GRST}}$ = 1.
5. Hookup the interrupt service routines to the DMA or EDMA (interrupt 8 used for EDMA transfers, interrupt 9 used for DMA channel 1 transfers, and interrupt 11 used for DMA channel 2 transfers in the sample code provided). Enable the CPU interrupts that correspond to the DMA or EDMA channel that will be used to service the McBSP. Note that the EDMA controller generates a single interrupt, CPU_INT8, to the CPU (EDMA_INT) on behalf of all 16 channels (C621x/C671x) or 64 channels (C64x). The

C6000 is a trademark of Texas Instruments.

various control registers and bit fields facilitate EDMA interrupt generation. The default mapping of DMA channel-complete interrupts to the CPU is the following:

− DMA channel 0 $\rightarrow$ CPU interrupt 8

− DMA channel 1 $\rightarrow$ CPU interrupt 9

− DMA channel 2 $\rightarrow$ CPU interrupt 11

− DMA channel 3 $\rightarrow$ CPU interrupt 12

6. Either (a) or (b) should be followed:

   a. If the DMA is used to perform data transfers, it should first be initialized with the appropriate read/write syncs, src/dst addresses, and their update modes, transfer complete interrupt, and any other feature suitable for the application. Lastly, set the START bit. The DMA is now in the START state and waits for the synchronization events to occur. Wake up the AIC27 codec by setting the AIC27 RESETB signal inactive-high. BITCLK of the AIC27 starts running after some delay. BITCLK drives the McBSP's sample rate generator clock CLKG. Then, pull the McBSP out of reset (Set $\overline{\text{XRST}}$ = $\overline{\text{RRST}}$ = 1 to enable the McBSP). Set $\overline{\text{FRST}}$ = 1 to start the frame sync generator in the McBSP. The first frame sync signal (FSX) is generated by the McBSP after 8 CLKG clocks. This FSX signal is captured by the AIC27 codec on the falling edge of BITCLK. SDATAIN and SDATAOUT are transmitted at the next rising edge of BITCLK. For details on DMA initialization for servicing the McBSP, refer to *TMS320C6000 McBSP Initialization* (SPRA488) and *TMS320C6000 DMA Application*s (SPRA529).

   b. If the enhanced DMA (EDMA) is used to perform data transfers, the channels associated to the McBSP transmit and receive synchronization events should first be configured with the appropriate priority levels, element size, src/dst addresses, address update modes, transfer complete code, transfer-complete interrupt enable, source and destination dimensions, and any other feature suitable for the application in the PaRAM parameter fields. The events are latched in the event register (ER), even if the events are disabled. Enabling the corresponding event bit in the event enable register (EER) starts the data transfer by setting this bit to a 1. Wake up the AIC27 codec by setting the AIC27 RESETB signal inactive-high. BITCLK of the AIC27 starts running after some delay. BITCLK drives the McBSP's sample rate generator clock CLKG. Then, pull the McBSP out of reset (Set $\overline{\text{XRST}}$ = $\overline{\text{RRST}}$ = 1 to enable the McBSP). Set $\overline{\text{FRST}}$ = 1 to start the frame sync generator in the McBSP. The first frame sync signal (FSX) is generated by the McBSP after 8 CLKG clocks. This FSX signal is captured by the AIC27 codec on the falling edge of BITCLK. SDATAIN and SDATAOUT are transmitted at the next rising edge of BITCLK. For details on EDMA initialization for servicing the McBSP, refer to *TMS320C6000 McBSP Initialization* (SPRA488) and *TMS320C6000 Enhanced DMA: Example Applications* (SPRA636).

## 3.5 Sample C Functions

Appendix NO TAG contains a sample C code that sets up the TMS320C6000 as a digital controller for an audio codec 1997 device. (The *TMS320C6000 Chip Support Library API Reference Guide* (SPRU401) provides a detailed description of the header files and Hardware Abstraction Language Macros used in this code.) This code has been tested on a specially configured Texas Instruments TMS320C6000 board by connecting two McBSP ports together, one of the McBSPs set to act as a AC'97 device, to emulate the presence of an AC'97 device interfaced to a McBSP port, but has not been tested using an actual TMS320C6000 McBSP-TLV320AIC27 interface.

## 4 Conclusion

The McBSP can function as an AC’97 digital controller by correctly connecting just five signals. This AC’97 interface runs at a fixed-sample rate of 48 kHz. The McBSP generates this 48-kHz SYNC signal (or FSX) by dividing the 12.288-MHz BITCLK input by 256. Each of the audio frames consists of two phases. The first phase is a 16-bit phase that defines the TAG portion of the AC’97 signal. The second phase, which consists of twelve 20-bit elements, defines the 240-bit DATA Phase. With signals configured as described in this application report, the TMS320C6000 operates as an AC’97 digital controller.

## 5 References

1. *Audio Codec ’97 Component Specification, Revision 2.2*, Intel Corporation, September 2000.
2. *Stereo Audio Codec* (SLAS253).
3. *TMS320C6000 Peripherals Reference Guide* (SPRU190).
4. *TMS320C6211, TMS320C6211B Fixed-Point DSPs* (SPRS073).
5. *TMS320C6203 Fixed-Point Digital Signal Processor* (SPRS086).
6. *TMS320C6000 McBSP Initialization* (SPRA488).
7. *TMS320C6000 Enhanced DMA: Example Applications* (SPRA636).
8. *TMS320C6000 Chip Support Library API Reference Guide* (SPRU401).

## Appendix A  Sample Source Code for AIC27 Interface

```
/*-------------------------------------------------------------------------------*/
/* ac97codec.c V1.00                                                             */
/*  Copyright (c) 2001 Texas Instruments Incorporated                            */
/*-------------------------------------------------------------------------------*/
/*
    6/12/01:  Vassos S. Soteriou

   ac97codec.c:
   This program sets the McBSP of the TMS320C6000 devices as a digital
   controller for an Audio Codec 1997 device. This program supports
   all the Texas Instruments TMS320C6000 DSPs, those that use the DMA
   controller or the Enhanced DMA controller (EDMA). For those that use
   the DMA controller, DMA channels 1 and 2 service the McBSP.
   CLKX, CLKR are generated using the CLKS clock.
   FSX is the output that drives the codec's frame syncs.
   In the case of a DMA transfer, the vecs.asm assembly code file is
   used to hookup the c_int11() and c_int09() ISRs to the corresponding
   interrupts. Channel 1 is hooked up to interrupt 9 for data receive,
   channel 2 is hooked up to interrupt 11 for data transmit, the DMA
   controller has individual interrupts for each DMA channel. The EDMA
   controller, however, generates a single interrupt to the CPU (EDMA_INT)
   on behalf of all 16 channels (C621x/C671x) or 64 channels (C64x). The
   various control registers and bit fields facilitate EDMA interrupt
   generation. CPU_INT8 is responsible for all the EDMA channels
   The sample code is based on TI's CSL 2.0. Please refer to the TMS320C6000
   Chip Support Library API User's Guide for further information.
*/

/* Chip definition, change this accordingly */
#define CHIP_6415 1
/* Include files */
#include <c6x.h>
#include <csl.h>        /* CSL library   */
#include <csl_dma.h>    /* DMA_SUPPORT   */
#include <csl_edma.h>   /* EDMA_SUPPORT  */
#include <csl_irq.h>    /* IRQ_SUPPORT   */
#include <csl_mcbsp.h>     /* MCBSP_SUPPORT */
#include <csl_timer.h>     /* TIMER_SUPPORT */
```

```
/*----------------------------------------------------------------------------*/
/* Define constants */
#define FALSE 0
#define TRUE 1
#define DMA_AC97 8
#define XFER_TYPE DMA_AC97
#define BUFFER_SIZE 256
#define ELEMENT_COUNT 13 /* Do not change this, AC'97 protocol */
#define FRAME_COUNT 10   /* Change this to desired value       */
/* Global variables used in interrupt ISRs */
volatile int recv0_done = FALSE;
volatile int xmit0_done = FALSE;
/*----------------------------------------------------------------------------*/
/* Declare CSL objects */
MCBSP_Handle hMcbsp0;       /* Handles for McBSP */
#if (DMA_SUPPORT)
DMA_Handle hDma1;           /* Handles for DMA   */
DMA_Handle hDma2;
#endif
#if (EDMA_SUPPORT)          /* Handles for EDMA  */
EDMA_Handle hEdma1;
EDMA_Handle hEdma2;
EDMA_Handle hEdmadummy;
#endif
TIMER_Handle hTimer0;      /* Handle for TIMER0 */
/*----------------------------------------------------------------------------*/
/* External functions and function prototypes */
void init_mcbsp0_ac97(void);   /* Function prototypes */
void set_interrupts_dma(void);
void set_interrupts_edma(void);
/* Inlcude the vector table to call the IRQ ISRs hookup */
extern far void vectors();
/*----------------------------------------------------------------------------*/
/* main()                                                                     */
/*----------------------------------------------------------------------------*/
void main(void)
{
/* Declaration of local variables */
static int element_count, frame_count, xfer_type;
int delay_count = 0;
```

```
static Uint32 dmaInbuff[BUFFER_SIZE];   /* buffer for DMA supporting devices  */
static Uint32 dmaOutbuff[BUFFER_SIZE];
static Uint32 edmaInbuff[BUFFER_SIZE];  /* buffer for EDMA supporting devices */
static Uint32 edmaOutbuff[BUFFER_SIZE];
IRQ_setVecs(vectors); /* point to the IRQ vector table */
element_count = ELEMENT_COUNT;
frame_count = FRAME_COUNT;
xfer_type = XFER_TYPE;
/* initialize the CSL library */
CSL_init();
/* Reset AC97 device */
/* Handle to TIMER 0, reset it upon open */
hTimer0 = TIMER_open(TIMER_DEV0, TIMER_OPEN_RESET);
TIMER_setDataOut(hTimer0,0);           /* Write a 0 to the TOUT0           */
TIMER_start(hTimer0);                  /* Need to be 0 for at least 1 usec */
init_mcbsp0_ac97();
/* Enable sample rate generator   GRST=1                                   */
MCBSP_enableSrgr(hMcbsp0);             /* Handle to SRGR `                 */
switch (xfer_type) {
case DMA_AC97:
   #if (DMA_SUPPORT)                   /* for DMA supporting devices       */
   DMA_reset(INV);                     /* reset all DMA channels           */
   #endif
   #if (EDMA_SUPPORT)                  /* For EDMA supporting devices      */
   EDMA_clearPram(0x00000000);         /* Clear PaRAM RAM of the EDMA      */
   set_interrupts_edma();
   #endif
/*--------------------------------------------------------------------------*/
/* DMA channels 1 and 2 config structures                                   */
/*--------------------------------------------------------------------------*/
#if (DMA_SUPPORT)                                /* for DMA supporting devices */
  /* Channel 1 receives the data */
  hDma1 = DMA_open(DMA_CHA1, DMA_OPEN_RESET);   /* Handle to DMA channel 1    */
    DMA_configArgs(hDma1,
       DMA_PRICTL_RMK(
           DMA_PRICTL_DSTRLD_NONE,
           DMA_PRICTL_SRCRLD_NONE,
           DMA_PRICTL_EMOD_NOHALT,
           DMA_PRICTL_FS_DISABLE,
           DMA_PRICTL_TCINT_ENABLE, /* TCINT =1                             */
           DMA_PRICTL_PRI_DMA,      /* DMA high priority                     */
           DMA_PRICTL_WSYNC_NONE,
           DMA_PRICTL_RSYNC_REVT0,  /* Set synchronization event REVT1=01111 */
           DMA_PRICTL_INDEX_NA,
           DMA_PRICTL_CNTRLD_NA,
           DMA_PRICTL_SPLIT_DISABLE,
           DMA_PRICTL_ESIZE_32BIT,  /* Element size 32 bits                  */
           DMA_PRICTL_DSTDIR_INC, /* Increment destination by element size */
           DMA_PRICTL_SRCDIR_NONE,
           DMA_PRICTL_START_STOP
           ),
        DMA_SECCTL_RMK(
           DMA_SECCTL_WSPOL_NA,    /*only available for 6202 and 6203 devices */
           DMA_SECCTL_RSPOL_NA,    /*only available for 6202 and 6203 devices */
           DMA_SECCTL_FSIG_NA,     /*only available for 6202 and 6203 devices */
           DMA_SECCTL_DMACEN_LOW,
           DMA_SECCTL_WSYNCCLR_NOTHING,
           DMA_SECCTL_WSYNCSTAT_CLEAR,
           DMA_SECCTL_RSYNCCLR_NOTHING,
           DMA_SECCTL_RSYNCSTAT_CLEAR,
           DMA_SECCTL_WDROPIE_DISABLE,
           DMA_SECCTL_WDROPCOND_CLEAR,
           DMA_SECCTL_RDROPIE_DISABLE,
           DMA_SECCTL_RDROPCOND_CLEAR,
           DMA_SECCTL_BLOCKIE_ENABLE, /* BLOCK IE=1 enables DMA channel int */
           DMA_SECCTL_BLOCKCOND_CLEAR,
           DMA_SECCTL_LASTIE_DISABLE,
           DMA_SECCTL_LASTCOND_CLEAR,
           DMA_SECCTL_FRAMEIE_DISABLE,
           DMA_SECCTL_FRAMECOND_CLEAR,
           DMA_SECCTL_SXIE_DISABLE,
           DMA_SECCTL_SXCOND_CLEAR
           ),
        DMA_SRC_RMK(MCBSP_ADDRH(hMcbsp0, DRR)),
        DMA_DST_RMK((Uint32)dmaInbuff),
        DMA_XFRCNT_RMK(
           DMA_XFRCNT_FRMCNT_DEFAULT,          /* Set number of elements to
                                                  transfer                  */
           DMA_XFRCNT_ELECNT_OF(element_count * frame_count)
/* Optionally replace the above 2 lines with                                */
/*         DMA_XFRCNT_FRMCNT_OF(frame_count),                               */
/*         DMA_XFRCNT_ELECNT_OF(element_count)                           */
         )
);
  /* Channel 2 transmits the data */
  hDma2 = DMA_open(DMA_CHA2, DMA_OPEN_RESET);  /* Handle to DMA channel 2     */
    DMA_configArgs(hDma2,
       DMA_PRICTL_RMK(
           DMA_PRICTL_DSTRLD_NONE,
           DMA_PRICTL_SRCRLD_NONE,
           DMA_PRICTL_EMOD_NOHALT,
           DMA_PRICTL_FS_DISABLE,
           DMA_PRICTL_TCINT_ENABLE, /* TCINT =1                          */
           DMA_PRICTL_PRI_DMA,      /* DMA high priority                 */
           DMA_PRICTL_WSYNC_XEVT0, /* Set synchronization event XEVT0=01100 */
           DMA_PRICTL_RSYNC_NONE,
           DMA_PRICTL_INDEX_NA,
           DMA_PRICTL_CNTRLD_NA,
           DMA_PRICTL_SPLIT_DISABLE,
           DMA_PRICTL_ESIZE_32BIT,  /* Element size 32 bits              */
           DMA_PRICTL_DSTDIR_NONE,
           DMA_PRICTL_SRCDIR_INC,   /* Increment source by element size  */
           DMA_PRICTL_START_STOP
           ),
        DMA_SECCTL_RMK(
           DMA_SECCTL_WSPOL_NA,     /*only available for 6202 and 6203 devices */
           DMA_SECCTL_RSPOL_NA,     /*only available for 6202 and 6203 devices */
           DMA_SECCTL_FSIG_NA,      /*only available for 6202 and 6203 devices */
           DMA_SECCTL_DMACEN_LOW,
           DMA_SECCTL_WSYNCCLR_NOTHING,
           DMA_SECCTL_WSYNCSTAT_CLEAR,
           DMA_SECCTL_RSYNCCLR_NOTHING,
           DMA_SECCTL_RSYNCSTAT_CLEAR,
           DMA_SECCTL_WDROPIE_DISABLE,
           DMA_SECCTL_WDROPCOND_CLEAR,
           DMA_SECCTL_RDROPIE_DISABLE,
           DMA_SECCTL_RDROPCOND_CLEAR,
           DMA_SECCTL_BLOCKIE_ENABLE, /* BLOCK IE=1 enables DMA channel int */
           DMA_SECCTL_BLOCKCOND_CLEAR,
           DMA_SECCTL_LASTIE_DISABLE,
           DMA_SECCTL_LASTCOND_CLEAR,
           DMA_SECCTL_FRAMEIE_DISABLE,
           DMA_SECCTL_FRAMECOND_CLEAR,
           DMA_SECCTL_SXIE_DISABLE,
           DMA_SECCTL_SXCOND_CLEAR
           ),
        DMA_SRC_RMK((Uint32)dmaOutbuff),
        DMA_DST_RMK(MCBSP_ADDRH(hMcbsp0, DXR)),
        DMA_XFRCNT_RMK(
           DMA_XFRCNT_FRMCNT_DEFAULT,  /* Set number of elements to transfer */
           DMA_XFRCNT_ELECNT_OF(element_count * frame_count)
/* Optionally replace the above 2 lines with:  */
/*         DMA_XFRCNT_FRMCNT_OF(frame_count),  */
/*         DMA_XFRCNT_ELECNT_OF(element_count) */
           )
);
set_interrupts_dma(); /* Initialize the interrupts */
DMA_start(hDma1);     /* Start DMA channels 1 and 2 */
DMA_start(hDma2);
#endif  /* end for dma supporting devices */
/*---------------------------------------------------------------------------------*/
/* EDMA channels 12 and 13 config structures                                        */
/*---------------------------------------------------------------------------------*/
  #if (EDMA_SUPPORT)             /* for EDMA supporting devices  */

hEdma1 = EDMA_open(EDMA_CHA_REVT0, EDMA_OPEN_RESET);
EDMA_configArgs(hEdma1,

#if (!C64_SUPPORT)
     EDMA_OPT_RMK(
        EDMA_OPT_PRI_HIGH,       /* High priority EDMA */
        EDMA_OPT_ESIZE_32BIT,    /* Element size 32 bits */
        EDMA_OPT_2DS_DEFAULT,
        EDMA_OPT_SUM_DEFAULT,
        EDMA_OPT_2DD_DEFAULT,
        EDMA_OPT_DUM_INC,        /* Destination increment by element size */
        EDMA_OPT_TCINT_YES,      /* Enable Transfer Complete Interrupt    */
        EDMA_OPT_TCC_OF(13),     /* TCCINT = 0xD, REVT0                   */
        EDMA_OPT_LINK_YES,       /* Enable linking to NULL table          */
        EDMA_OPT_FS_NO
        ),
#endif
#if (C64_SUPPORT)
     EDMA_OPT_RMK(
        EDMA_OPT_PRI_HIGH,       /* High priority EDMA */
        EDMA_OPT_ESIZE_32BIT,    /* Element size 32 bits */
        EDMA_OPT_2DS_DEFAULT,
        EDMA_OPT_SUM_DEFAULT,
        EDMA_OPT_2DD_DEFAULT,
        EDMA_OPT_DUM_INC,  /*  Destination  increment  by  element  size  */
        EDMA_OPT_TCINT_YES,  /*  Enable  Transfer  Complete  Interrupt  */
        EDMA_OPT_TCC_OF(13),     /* TCCINT = 0xD, REVT0                    */
        EDMA_OPT_TCCM_DEFAULT,
        EDMA_OPT_ATCINT_DEFAULT,
        EDMA_OPT_ATCC_DEFAULT,
        EDMA_OPT_PDTS_DEFAULT,
        EDMA_OPT_PDTD_DEFAULT,
        EDMA_OPT_LINK_YES,       /* Enable linking to NULL table           */
        EDMA_OPT_FS_NO
        ),
#endif

        EDMA_SRC_RMK(MCBSP_ADDRH(hMcbsp0, DRR)), /* src to DRR0            */
        EDMA_CNT_RMK(0, frame_count * element_count),/* no. of elements    */
/* Optionally replace the above with the following line                    */
/*      EDMA_CNT_RMK(frame_count, element_count),                          */
        EDMA_DST_RMK((Uint32)edmaInbuff),  /*  dst  addr  to  edmaInbuff  */
        EDMA_IDX_RMK(0,0),
        EDMA_RLD_RMK(0,0)
        );

  hEdma2 = EDMA_open(EDMA_CHA_XEVT0, EDMA_OPEN_RESET);
  EDMA_configArgs(hEdma2,
#if(!C64_SUPPORT)                /* For 671X and 621x devices              */
      EDMA_OPT_RMK(
        EDMA_OPT_PRI_HIGH,       /* High priority EDMA                     */
        EDMA_OPT_ESIZE_32BIT,    /* Element size 32 bits                   */
        EDMA_OPT_2DS_DEFAULT,
        EDMA_OPT_SUM_INC,        /* Source increment by element size       */
        EDMA_OPT_2DD_DEFAULT,
        EDMA_OPT_DUM_DEFAULT,
        EDMA_OPT_TCINT_YES,       /*  Enable Transfer Complete Interrupt      */
        EDMA_OPT_TCC_OF(12),      /*  TCCINT = 0xC, XEVT0                     */
        EDMA_OPT_LINK_YES,        /* Enable linking to NULL table             */
        EDMA_OPT_FS_NO
        ),
#endif
#if(C64_SUPPORT)
        EDMA_OPT_RMK(             /* For 64x devices only                     */
        EDMA_OPT_PRI_HIGH,        /* High priority EDMA                       */
        EDMA_OPT_ESIZE_32BIT,     /* Element size 32 bits                     */
        EDMA_OPT_2DS_DEFAULT,
        EDMA_OPT_SUM_INC,         /* Source increment by element size */
        EDMA_OPT_2DD_DEFAULT,
        EDMA_OPT_DUM_DEFAULT,
        EDMA_OPT_TCINT_YES,   /*   Enable   Transfer   Complete   Interrupt   */
        EDMA_OPT_TCC_OF(12),      /* TCCINT = 0xC, XEVT0                      */
        EDMA_OPT_TCCM_DEFAULT,
        EDMA_OPT_ATCINT_DEFAULT,
        EDMA_OPT_ATCC_DEFAULT,
        EDMA_OPT_PDTS_DEFAULT,
        EDMA_OPT_PDTD_DEFAULT,
           EDMA_OPT_LINK_YES,        /* Enable linking to NULL table          */
        EDMA_OPT_FS_NO

        ),
#endif
        EDMA_SRC_RMK((Uint32)edmaOutbuff), /*src to edmaOutbuff               */
        EDMA_CNT_RMK(0,frame_count * element_count), /* set no. of elements */
/* Optionally replace the above with the following line                       */
/*      EDMA_CNT_RMK(frame_count, element_count),                             */
        EDMA_DST_RMK(MCBSP_ADDRH(hMcbsp0,  DXR)),  /*  dst  addr  to  DXR0  */
        EDMA_IDX_RMK(0,0),
        EDMA_RLD_RMK(0,0)
        );
  hEdmadummy = EDMA_allocTable(-1); /* Dynamically allocates PaRAM RAM table */
  EDMA_configArgs(hEdmadummy, /* Dummy or Terminating Table in PaRAM         */
         0x00000000,          /* Terminate EDMA transfers by linking to      */
         0x00000000,          /* this NULL table                             */
         0x00000000,
         0x00000000,
        0x00000000,
        0x00000000
        );
  EDMA_link(hEdma1, hEdmadummy); /* Link terminating event to the EDMA event */
  EDMA_link(hEdma2, hEdmadummy);
  EDMA_enableChannel(hEdma1);    /* Enable EDMA channels */
  EDMA_enableChannel(hEdma2);
  #endif  /* end for EDMA supporting devices */
}
/* make sure TOUT0 was low for  >++ 1 usec */
for (delay_count = 0 ; delay_count < 500 ; delay_count++);
TIMER_setDataOut(hTimer0,1);      /* Write a 1 to the TOUT0 */
/* wait for BITCLK to start */
for (delay_count = 0 ; delay_count <100 ; delay_count++);
MCBSP_enableRcv(hMcbsp0); /* Enable McBSP channel                     */
MCBSP_enableXmt(hMcbsp0); /* McBSP port 0 as the transmitter/receiver */
MCBSP_enableFsync(hMcbsp0); /* Enable frame sync for the McBSP        */
/* To flag an interrupt to the CPU when DMA transfer/receive is done */
#if (DMA_SUPPORT)
  while (!xmit0_done || !recv0_done);
#endif
/* To flag an interrupt to the CPU when EDMA transfer/receive is done */
/* Transfer completion interrupt 12 and 13 set flag = 1 when set      */
#if (EDMA_SUPPORT)
  while (!xmit0_done || !recv0_done);
#endif
MCBSP_close(hMcbsp0);  /* close McBSP port */
#if (DMA_SUPPORT)      /* close DMA channels */
DMA_close(hDma1);
DMA_close(hDma2);
#endif
#if (EDMA_SUPPPORT)
EDMA_close(hEdma1);    /* close EDMA channels */
EDMA_close(hEdma2);
EDMA_close(hEdmadummy);
#endif
TIMER_close(hTimer0);  /* close TIMER 0 */
} /* end main, progam ends here */
```

```
/*-------------------------------------------------------------------------------*/
/* init_mcbsp0_ac97()                                                            */
/*-------------------------------------------------------------------------------*/
/* MCBSP Config structure */
/* Setup the MCBSP_0 for transfers with the AC97 codec*/
void
init_mcbsp0_ac97(void)
{
MCBSP_Config mcbspCfg0 = {
#if (EDMA_SUPPORT)
    MCBSP_SPCR_RMK(
         MCBSP_SPCR_FREE_DEFAULT,  /* All fields in SPCR set to default values */
         MCBSP_SPCR_SOFT_DEFAULT,
           MCBSP_SPCR_FRST_DEFAULT,
           MCBSP_SPCR_GRST_DEFAULT,
           MCBSP_SPCR_XINTM_DEFAULT,
           MCBSP_SPCR_XSYNCERR_DEFAULT,
           MCBSP_SPCR_XRST_DEFAULT,
           MCBSP_SPCR_DLB_DEFAULT,
           MCBSP_SPCR_RJUST_DEFAULT,
           MCBSP_SPCR_CLKSTP_DEFAULT,
           MCBSP_SPCR_DXENA_DEFAULT,
           MCBSP_SPCR_RINTM_DEFAULT,
           MCBSP_SPCR_RSYNCERR_DEFAULT,
           MCBSP_SPCR_RRST_DEFAULT
           ),
#endif
#if (DMA_SUPPORT)
   MCBSP_SPCR_RMK(
          MCBSP_SPCR_FRST_DEFAULT,   /* All fields in SPCR set to default values */
          MCBSP_SPCR_GRST_DEFAULT,
          MCBSP_SPCR_XINTM_DEFAULT,
          MCBSP_SPCR_XSYNCERR_DEFAULT,
          MCBSP_SPCR_XRST_DEFAULT,
          MCBSP_SPCR_DLB_DEFAULT,
          MCBSP_SPCR_RJUST_DEFAULT,
          MCBSP_SPCR_CLKSTP_DEFAULT,
          MCBSP_SPCR_RINTM_DEFAULT,
          MCBSP_SPCR_RSYNCERR_DEFAULT,
          MCBSP_SPCR_RRST_DEFAULT
          ),
#endif
#if (EDMA_SUPPORT)
    MCBSP_RCR_RMK(
           MCBSP_RCR_RPHASE_DUAL,  /*  Dual  phase  receive  frame  */
           MCBSP_RCR_RFRLEN2_OF(0xB), /* frame length =12 elements */
           MCBSP_RCR_RWDLEN2_20BIT,  /*  receive  elements  =  20  bits*/
           MCBSP_RCR_RCOMPAND_DEFAULT,
           MCBSP_RCR_RFIG_DEFAULT,
           MCBSP_RCR_RDATDLY_1BIT,  /*  1-bit  receive  data  delay  */
           MCBSP_RCR_RFRLEN1_OF(0x0), /* frame length of 1 element */
           MCBSP_RCR_RWDLEN1_16BIT,  /*  receive  elements  =  16bits  */
           MCBSP_RCR_RWDREVRS_DEFAULT
           ),
#endif
#if (DMA_SUPPORT)
    MCBSP_RCR_RMK(
           MCBSP_RCR_RPHASE_DUAL,  /*  Dual  phase  receive  frame  */
           MCBSP_RCR_RFRLEN2_OF(0xB), /* frame length =12 elements */
           MCBSP_RCR_RWDLEN2_20BIT,  /*  receive  elements  =  20  bits*/
           MCBSP_RCR_RCOMPAND_DEFAULT,
           MCBSP_RCR_RFIG_DEFAULT,
           MCBSP_RCR_RDATDLY_1BIT,  /*  1-bit  receive  data  delay  */
           MCBSP_RCR_RFRLEN1_OF(0x0), /* frame length of 1 element */
           MCBSP_RCR_RWDLEN1_16BIT /* receive elements = 16bits */
           ),
#endif
#if (EDMA_SUPPORT)
    MCBSP_XCR_RMK(
           MCBSP_XCR_XPHASE_DUAL,  /*  Dual  phase  transmit  frame  */
           MCBSP_XCR_XFRLEN2_OF(0xB), /* frame length =12 elements */
           MCBSP_XCR_XWDLEN2_20BIT,
           MCBSP_XCR_XCOMPAND_DEFAULT,
           MCBSP_XCR_XFIG_DEFAULT,
           MCBSP_XCR_XDATDLY_1BIT,  /*  1-bit  transmit  data  delay  */
           MCBSP_XCR_XFRLEN1_OF(0x0), /* frame length of 1 element */
           MCBSP_XCR_XWDLEN1_16BIT,  /*  receive  elements  =  16bits  */
           MCBSP_XCR_XWDREVRS_DEFAULT
           ),
#endif
```

```
#if (DMA_SUPPORT)
    MCBSP_XCR_RMK(
           MCBSP_XCR_XPHASE_DUAL,  /*  Dual  phase  transmit  frame  */
           MCBSP_XCR_XFRLEN2_OF(0xB), /* frame length =12 elements */
           MCBSP_XCR_XWDLEN2_20BIT,   /* 20bit elements            */
           MCBSP_XCR_XCOMPAND_DEFAULT,
           MCBSP_XCR_XFIG_DEFAULT,
           MCBSP_XCR_XDATDLY_1BIT,  /*  1-bit  transmit  data  delay */
           MCBSP_XCR_XFRLEN1_OF(0x0), /* frame length of 1 element */
           MCBSP_XCR_XWDLEN1_16BIT /* receive elements = 16bits */
           ),
#endif
     MCBSP_SRGR_RMK(
           MCBSP_SRGR_GSYNC_FREE,    /* Free running driven by CLKS              */
           MCBSP_SRGR_CLKSP_DEFAULT,
           MCBSP_SRGR_CLKSM_CLKS,    /* External clock source, CLKS, deriv
                                        CLKSM                                    */
           MCBSP_SRGR_FSGM_FSG,      /* FSX driven by SRG frame sync signal      */
           MCBSP_SRGR_FPER_OF(0xFF), /* Frame period is 256 CLKS 12.288MHz
                                        periods                                  */
           MCBSP_SRGR_FWID_OF(0xF),  /* Frame sync signal width = 16 BITCLK      */
           MCBSP_SRGR_CLKGDV_OF(0)   /* CLKG same freq. as SRGR input clock CLKS */
           ),
#if (C64_SUPPORT)
     MCBSP_MCR_RMK(                  /* only for 64x                             */
           MCBSP_MCR_XMCME_DEFAULT, /* All fields in MCR set to default values */
           MCBSP_MCR_XPBBLK_DEFAULT,
           MCBSP_MCR_XPABLK_DEFAULT,
           MCBSP_MCR_XMCM_DEFAULT,
           MCBSP_MCR_RPBBLK_DEFAULT,
           MCBSP_MCR_RMCME_DEFAULT,
           MCBSP_MCR_RPABLK_DEFAULT,
           MCBSP_MCR_RMCM_DEFAULT
           ),
#else
     MCBSP_MCR_RMK(
           MCBSP_MCR_XPBBLK_DEFAULT, /* All fields in MCR set to default values  */
           MCBSP_MCR_XPABLK_DEFAULT,
           MCBSP_MCR_XMCM_DEFAULT,
           MCBSP_MCR_RPBBLK_DEFAULT,
           MCBSP_MCR_RPABLK_DEFAULT,
           MCBSP_MCR_RMCM_DEFAULT
           ),
#endif
#if(!C64_SUPPORT)
     MCBSP_RCER_RMK(
           MCBSP_RCER_RCEB_DEFAULT,  /* All fields in RCER set to default values */
           MCBSP_RCER_RCEA_DEFAULT
           ),
#endif
#if(!C64_SUPPORT)
     MCBSP_XCER_RMK(
           MCBSP_XCER_XCEB_DEFAULT,  /* All fields in XCER set to default values */
           MCBSP_XCER_XCEA_DEFAULT
           ),
#endif
#if (C64_SUPPORT)
           MCBSP_RCERE0_RMK(0),      /* Additional registers only for 64x        */
           MCBSP_RCERE1_RMK(0),
           MCBSP_RCERE2_RMK(0),
           MCBSP_RCERE3_RMK(0),
#endif

#if (C64_SUPPORT)
           MCBSP_XCERE0_RMK(0),       /* Additional registers only for 64x       */
           MCBSP_XCERE1_RMK(0),
           MCBSP_XCERE2_RMK(0),
           MCBSP_XCERE3_RMK(0),
#endif
     MCBSP_PCR_RMK(
           MCBSP_PCR_XIOEN_DEFAULT,
           MCBSP_PCR_RIOEN_DEFAULT,
           MCBSP_PCR_FSXM_INTERNAL,   /*   Frame   sync   generated   internally   */
           MCBSP_PCR_FSRM_INTERNAL,   /*   Frame   sync   generated   internally   */
           MCBSP_PCR_CLKXM_DEFAULT,
           MCBSP_PCR_CLKRM_DEFAULT,
         MCBSP_PCR_CLKSSTAT_DEFAULT,
           MCBSP_PCR_DXSTAT_DEFAULT,
           MCBSP_PCR_FSXP_ACTIVEHIGH, /* FSX is active high                      */
           MCBSP_PCR_FSRP_ACTIVEHIGH, /* FSR is active high                      */
           MCBSP_PCR_CLKXP_DEFAULT,
           MCBSP_PCR_CLKRP_DEFAULT
           )
};
hMcbsp0 = MCBSP_open(MCBSP_DEV0, MCBSP_OPEN_RESET); /* McBSP port 0              */
MCBSP_config(hMcbsp0, &mcbspCfg0);
}
/*----------------------------------------------------------------------------*/
/* set_interrupts_dma()                                                       */
/*----------------------------------------------------------------------------*/
#if (DMA_SUPPORT)
void                           /* Set the interrupts         */
set_interrupts_dma(void)       /* if the device supports DMA */
{
 IRQ_nmiEnable();
 IRQ_globalEnable();
 IRQ_disable(IRQ_EVT_DMAINT2);
 IRQ_disable(IRQ_EVT_DMAINT1);  /* INT11 and INT9 */
 IRQ_clear(IRQ_EVT_DMAINT2);
 IRQ_clear(IRQ_EVT_DMAINT1);
 IRQ_enable(IRQ_EVT_DMAINT2);
 IRQ_enable(IRQ_EVT_DMAINT1);
 return;
}
#endif
/*----------------------------------------------------------------------------*/
/* set_interrupts_edma()                                                      */
/*----------------------------------------------------------------------------*/
#if (EDMA_SUPPORT)
void                          /* Set the interrupts          */
set_interrupts_edma(void)     /* if the device supports EDMA */
{
 IRQ_nmiEnable();
 IRQ_globalEnable();
 IRQ_reset(IRQ_EVT_EDMAINT);
 IRQ_disable(IRQ_EVT_EDMAINT);
 EDMA_intDisable(12);  /*  ch  12  for  McBSP  transmit  event  XEVT0  */
 EDMA_intDisable(13);  /*  ch  13  for  McBSP  receive  event  REVT0  */
 IRQ_clear(IRQ_EVT_EDMAINT);
 EDMA_intClear(12);
 EDMA_intClear(13);
 IRQ_enable(IRQ_EVT_EDMAINT);
 EDMA_intEnable(12);
 EDMA_intEnable(13);

 return;
}
#endif
/*-------------------------------------------------------------------------------*/
/*    DMA DATA TRANSFER COMPLETION ISRs                                          */
/*-------------------------------------------------------------------------------*/
interrupt void          /* vecs.asm hooks this up to IRQ 11 */
c_int11(void)           /* DMA ch2 */
{
xmit0_done = TRUE;
return;
}
interrupt void          /* vecs.asm hooks this up to IRQ 09 */
c_int09(void)           /* DMA ch1                          */
{
recv0_done = TRUE;
return;
}
interrupt void          /* vecs.asm hooks this up to IRQ 08 */
c_int08(void)           /* for the EDMA */
{
 #if (EDMA_SUPPORT)
  if (EDMA_intTest(12))
  {
  xmit0_done = TRUE;
  EDMA_intClear(12);    /* clear CIPR bit so future interrupts can be recognized */
  }
  else if (EDMA_intTest(13))
  {
  recv0_done = TRUE;
  EDMA_intClear(13); /* clear CIPR bit so future interrupts can be recognized   */
  }
#endif
return;
}
/*----------------------End of ac97codec.c------------------------------------*/
```

```
********************************************************************************
*           Copyright (C) 2000 Texas Instruments Incorporated.
*                           All Rights Reserved
*------------------------------------------------------------------------------
* FILENAME.......vecs.asm
* DATE CREATED. 12/06/2000
* LAST MODIFIED. 12/06/2000
********************************************************************************
*------------------------------------------------------------------------------
* Global symbols defined here and exported out of this file
*------------------------------------------------------------------------------
   .global _vectors
   .global _vector0
   .global _vector1
   .global _vector2
   .global _vector3
   .global _vector4
   .global _vector5
   .global _vector6
   .global _vector7
   .global _c_int08   ; Hookup the c_int08 ISR in main() for EDMA
   .global _c_int09   ; Hookup the c_int09 ISR in main() for DMA
   .global _vector10
   .global _c_int11   ; Hookup the c_int11 ISR in main() for DMA
   .global _vector12
   .global _vector13
   .global _vector14
   .global _vector15
*------------------------------------------------------------------------------
* Global symbols referenced in this file but defined somewhere else.
* Remember that your interrupt service routines need to be referenced here.
*------------------------------------------------------------------------------
   .ref _c_int00
*------------------------------------------------------------------------------
* This is a macro that instantiates one entry in the interrupt service table.
*------------------------------------------------------------------------------
VEC_ENTRY .macro addr
    STW   B0,*--B15
    MVKL  addr,B0
    MVKH  addr,B0
    B     B0
    LDW   *B15++,B0
    NOP   2
    NOP
    NOP
   .endm
*------------------------------------------------------------------------------
* This is a dummy interrupt service routine used to initialize the IST.
*------------------------------------------------------------------------------
_vec_dummy:
  B    B3
  NOP  5
*------------------------------------------------------------------------------
* This is the actual interrupt service table (IST). It is properly aligned and
* is located in the subsection .text:vecs. This means if you don’t explicitly
* specify this section in your linker command file, it will default and link
* into the .text section. Remember to set the ISTP register to point to this
* table.
*------------------------------------------------------------------------------
 .sect ”.text:vecs”
 .align 1024
_vectors:
_vector0:   VEC_ENTRY _vec_dummy
_vector1:   VEC_ENTRY _vec_dummy
_vector2:   VEC_ENTRY _vec_dummy
_vector3:   VEC_ENTRY _vec_dummy
_vector4:   VEC_ENTRY _vec_dummy
_vector5:   VEC_ENTRY _vec_dummy
_vector6:   VEC_ENTRY _vec_dummy
_vector7:   VEC_ENTRY _vec_dummy
_vector8:   VEC_ENTRY _c_int08    ; Hookup the c_int08 ISR in main() for EDMA
_vector9:   VEC_ENTRY _c_int09    ; Hookup the c_int09 ISR in main() for DMA
_vector10:  VEC_ENTRY _vec_dummy
_vector11:  VEC_ENTRY _c_int11    ; Hookup the c_int11 ISR in main() for DMA
_vector12:  VEC_ENTRY _vec_dummy
_vector13:  VEC_ENTRY _vec_dummy
_vector14:  VEC_ENTRY _vec_dummy
_vector15:  VEC_ENTRY _vec_dummy
*------------------------------------------------------------------------------
********************************************************************************
* End of vecs.asm
********************************************************************************
```