‘C6x McBSP Interface to a Voice-Band Audio Processor (VBAP)

APPLICATION REPORT: PRELIMINARY

Shaku Anjanaiah

Digital Signal Processing Solutions
June 1998

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain application using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

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

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

TRADEMARKS

TI is a trademark of Texas Instruments Incorporated.

Other brands and names are the property of their respective owners.

CONTACT INFORMATION

US TMS320 HOTLINE	(281) 274-2320
US TMS320 FAX	(281) 274-2324
US TMS320 BBS	(281) 274-2323
US TMS320 email	dsph@ti.com

Contents

Figures

Tables

‘C6x McBSP Interface to a Voice-Band Audio Processor (VBAP)

Abstract

This document describes how the multi-channel buffered serial port (McBSP) in the TMS320C6201 digital signal processor (DSP) interfaces to the VBAP. The VBAP under discussion is the TI TLV320AC36, 3V 2.048 MHz audio processor which is a μ-law companding device. The interface is also applicable to TI’s TLV320AC37, an A-law companding audio processor.

The highly programmable McBSP provides a glue-less interface to the VBAP. The VBAP processes analog signals from audio sources such as a microphone, converts it to digital data (in two choice formats) which is then transmitted to the DSP for processing. The DSP in turn will transmit digital data to the VBAP for conversion to analog data. The VBAP outputs this analog data to a device such as a speaker. The McBSP supports both companded (8-bit) and linear (16-bit) form of data that the VBAP supports. The hardware schematic discussed in this document is a possible solution for the VBAP interface, but it has not been tested.

Product Support

Related Documentation

The following list specifies relevant product names, part numbers, and literature numbers of TI documentation.

- *TMS320C6201 Digital Signal Processor* data sheet, March 1998, Literature number SPRS051C
- *TMS320C6201/C6701 Peripherals* Reference Guide, March 1998, Literature number SPRU190B
- *TCM320AC3x/4x Voice-Band Audio Processors* Application Report, June 1996, Literature Number SLWA001
- *TLV320AC36, TLV320AC37 Voice-Band Audio Processors (VBAP™)* data sheet, October 1997, Literature number SLWS006B

World Wide Web

Our World Wide Web site at **www.ti.com** contains the most up to date product information, revisions, and additions. Users registering with TI&ME can build custom information pages and receive new product updates automatically via email.

Email

For technical issues or clarification on switching products, please send a detailed email to **dsph@ti.com**. Questions receive prompt attention and are usually answered within one business day.

Design Problem

How do I interface a Voice-Band Audio Processor (VBAP) to the TMS320C6201 Multi-channel Buffered Serial Port (McBSP)?

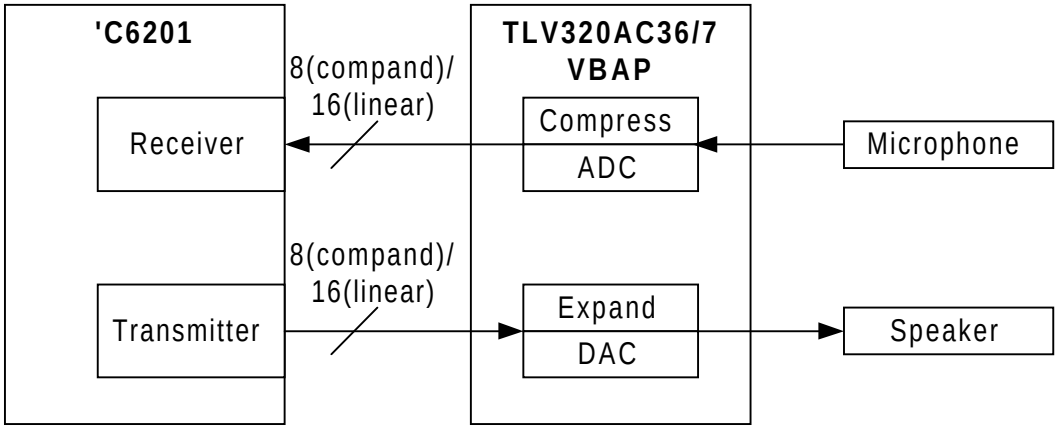
Overview

The block diagram interface between the ‘C6201 and the VBAP is shown in Figure 1. The analog voice-band input from the microphone is processed by the Analog-to-Digital Converter (ADC) in the VBAP. The ADC either compresses the analog signal to an 8-bit data format (if compand operation is chosen in the VBAP) or transforms the analog data to 13-bit linear data (if linear data format is chosen) with the three LSBs padded with zeroes or volume control data. The VBAP thus performs a transmit encoding on the analog data and provides digital data to the McBSP receiver. The DSP can now process the digital data as necessary.

The DSP (McBSP) can either transmit 8-bit companded data (in μ-law or A-law format) or 16-bit linear data to the receive section of the VBAP. The Digital-to-Analog Converter (DAC) in the VBAP expands the serial data to its analog form and sends it to the speaker.

In both cases of transmit and receive, the McBSP can be programmed to either transmit companded data in either A-law format if interfacing to the TLV320AC37 or μ-law format if interfacing to TLV320AC36. μ-law and A-law companding standard is part of the CCITT G.711 recommendation.

Figure 1. ‘C6201 McBSP and VBAP interface block diagram



VBAP Operation

The VBAP offers two modes of operation, which are pin-selectable. They are:

- *Fixed Data Rate Mode*: A single master clock (CLK) is used to clock both receive and transmit data. Data is transmitted to the DSP on rising edge of clock and receive data sampled on falling edge of CLK. Master clock frequency is 2.048 MHz. Frame sync signal inputs, FSX and FSR are active high for one CLK period to indicate starting of a frame. Frame syncs must have an 8KHz sampling rate.

- *Variable Data Rate Mode*: The received data on DIN is sampled on falling edge of DCLKR and transmit data is output on rising edge of DCLKX. These clocks can be slower than the master CLK rate of 2.048MHz.

  The DCLKR and DCLKX can have a range between 64KHz to 2.048MHz for 8-b compand mode and 128KHz to 2.048Mhz for 16-b linear data. The master clock is not used for clocking data, but for internal filters. FSX and FSR inputs remain high for the entire frame and remain at 8KHz sampling rate.

More information on the VBAP itself can be obtained from the literature specified under Related Documentation in this document.

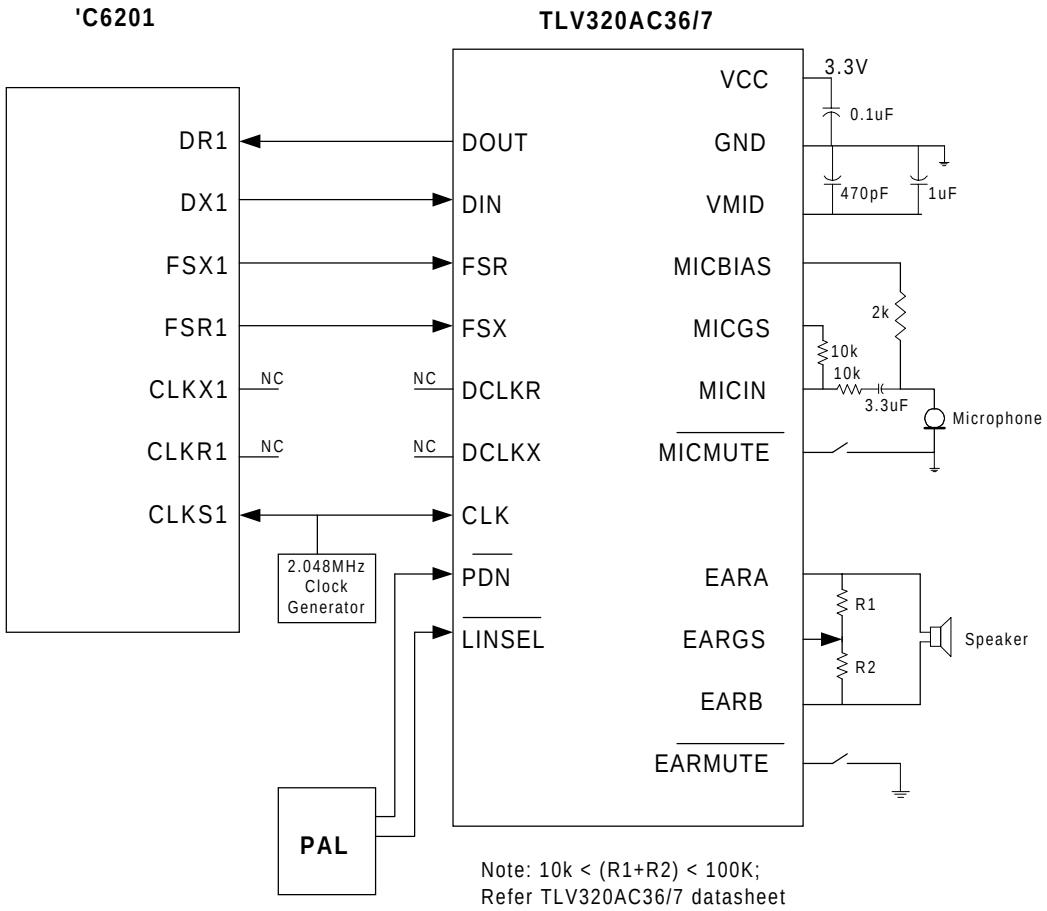
Hardware Interface

This application note describes the Fixed Data Rate Mode VBAP interface to the McBSP. Accordingly, the pin interface and schematic for this set up is shown in Figure 2. Since the VBAP is a 3V device, no voltage translation is necessary to interface to the ‘C6201. Also, the electret type microphone and speaker (or headset) should be a 3V part.

- **Clock Generation:** A 2.048 MHz clock generator drives the master clock, CLK, of the VBAP and external clock CLKS of the McBSP. The McBSP uses this external clock source to generate internal transmit and receive clocks to shift data. CLKR and CLKX of the McBSP should be configured as outputs. DCLKR should be tied to Vcc or left unconnected (logic high) for fixed data rate mode.

- **Frame Sync Generation:** The McBSP generates an 8KHz sampling rate frame sync signal. The frame sync parameters, FPER and FWID are programmable in the McBSP. In order to generate 125 μS period (8kHz) frame sync based on a 2.048 MHz clock, the frame period (not FPER) should be 256 bit clocks. Therefore FPER is 255. FWID is zero since the frame sync is 1 bit-clock high at the beginning of a frame. Note that (FPER+1) and (FWID+1) represent the actual values. Both FSX and FSR are driven by the same internal frame sync generator signal (FSG).

Figure 2. 'C6201 and VBAP Schematic for Fixed Data Rate Mode

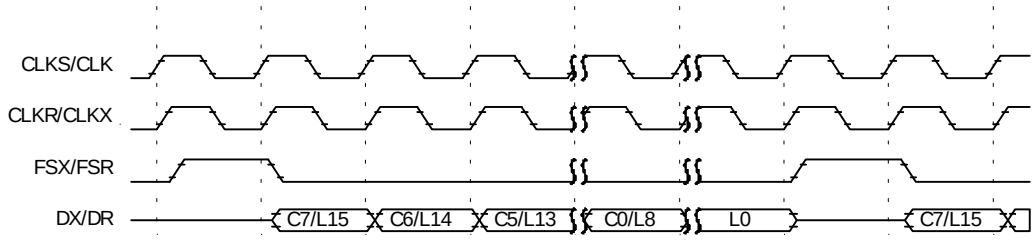


- **Data Format:** The McBSP drives data on DX and receives data on DR to/from the VBAP in either 8-bit compand format or 16-bit linear format. The /LINSEL pin on the VBAP when pulled low enables the data in linear format. Accordingly the (R/X)COMPAND bit in the McBSP should be programmed for receiving non-companded data. The schematic shows /LINSEL being driven by the PAL and therefore can be controlled by software.
- **Electret Microphone Interface:** A 3V electret-type microphone is chosen for this interface. It has the most effective noise cancellation and produces clear voice transmission (compared to the carbon and dynamic type). The passive components chosen for this interface is based on the information provided in the VBAP datasheet.
- **Speaker Interface:** R1 and R2 resistor network in the schematic is for controlling the power amplifier gain. The parallel combination of (R1+R2) and load resistance (RL) sets the total loading. See VBAP datasheet for more details.
- **Power Down:** The active low /PDN pin on the VBAP can be driven low by the PAL when reduced power consumption is required and the VBAP is not in use. A simple switch connected to /PDN and GND will also suffice.

McBSP Register Configuration

The timing diagram applicable for a fixed data rate mode of the VBAP is shown in Figure 3. The master clock from a 2.048 MHz clock generator drives McBSP's CLKS and VBAP's CLK. CLKS is used to generate the bit-clocks CLKR and CLKX for McBSP data transfer. Frame syncs are generated on the rising edge of CLKR/CLKX. First data bit (MSB) is transmitted/received with a 1 bit-clock delay from FSX/FSR. Therefore (R/X)DATDLY=1.

Figure 3. Fixed Data Rate Mode Timing Diagram



Cx/Ly: CompandData X / LinearData Y

The programmable registers of the McBSP should take on values based on the timing diagram and frame sync requirements, which were described earlier. Figure 4 through Figure 8 and Table 1 shows the register values for the application.

Figure 4. Receive Control Register (RCR)

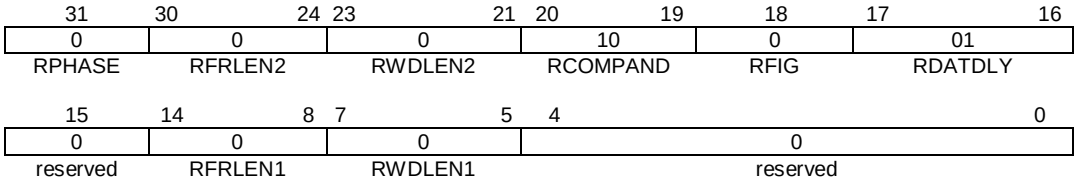


Figure 5. Transmit Control Register (XCR)

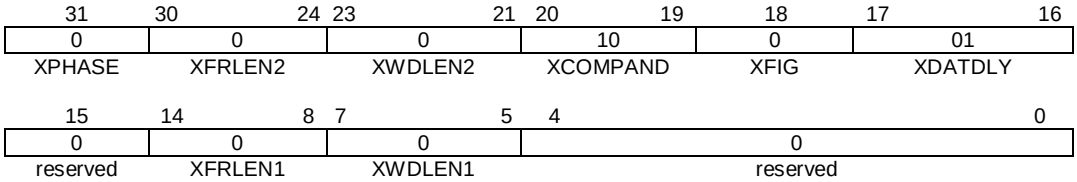


Figure 6. Sample Rate Generator Register (SRGR)

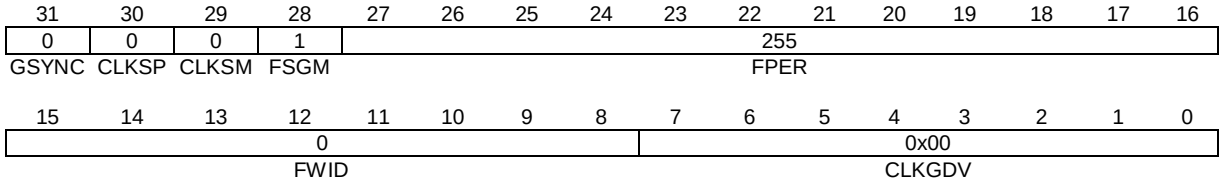


Figure 7. Pin Control Register (PCR)

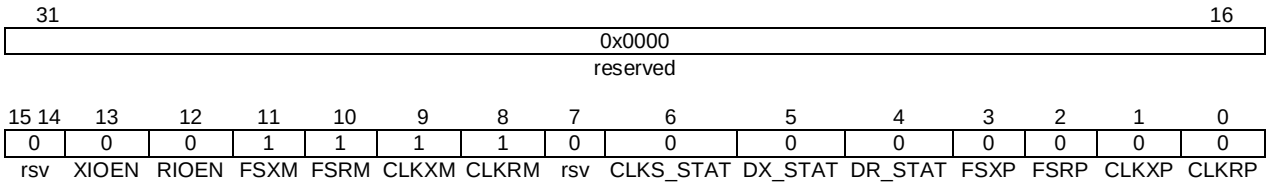


Figure 8. Serial Port Control Register (SPCR)

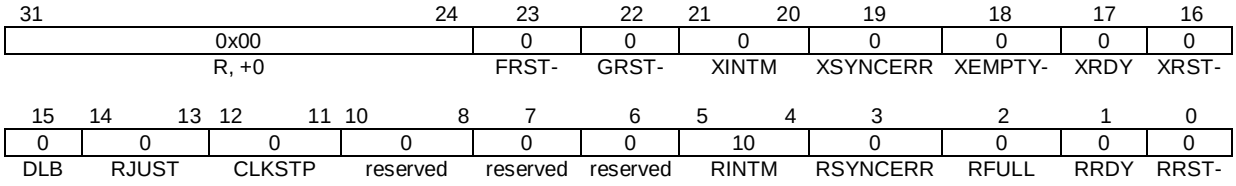


Table 1. McBSP Register Configuration Values

Register [bit-field #]	Bit-field Name	Value	Description
RCR[20:19]	RCOMPAND	10b	Receive μ-law companding
RCR[17:16]	RDATDLY	01b	I bit-clock data delay
XCR[20:19]	XCOMPAND	10b	Transmit μ-law companding
XCR[17:16]	XDATDLY	01b	I bit-clock data delay
SRGR[28]	FSGM	1	FSX generated by FSG
SRGR[27:16]	FPER	0xFF	Frame period of 256 2.048Mhz clock periods to get 8KHz frame sync sampling rate
SRGR[15:8]	FWID	0	Generates 1 clock period active-high pulse.
SRGR[7:0]	CLKGDV	0	2.048 CLKS drives CLKR/X with no divide-down
PCR[11]	FSXM	1	FSX is an output
PCR[10]	FSRM	1	FSR is an output
PCR[9]	CLKXM	1	CLKX is an output generated by CLKS
PCR[8]	CLKRM	1	CLKR is an output generated by CLKS
SPCR[31:0]	all	default	Reset bits will be driven low as per initialization procedure

McBSP and VBAP Initialization

Typically the DMA services the McBSP as and when read/write data events occur. The VBAP and the ‘C6201 (except the McBSP is not out of reset) is assumed to be powered up before the following steps are taken. Power is applied and all clock sources are connected. The /PDN pin on the VBAP is held low in power down state.

The following steps describe the setup of interrupts, DMA, and the McBSP in the required order.

1. Program the Sample Rate Generator Register (SRGR), Serial Port Control Register (SPCR), Pin Control Register (PCR), and Receive Control Register (RCR) to the values shown above.

   Caution: Do not set the /GRST bit in SPCR in this step.

2. Take the sample rate generator out of reset by setting /GRST=1 in the SPCR.

3. Enabling Interrupts: To use interrupts, you have to set the Global Interrupt Enable (GIE), and Non-Maskable Interrupt Enable (NMIE) bits in the IER.

Select the DMA channel(s) you want to use. Enable CPU interrupts that correspond to the DMA channel that will be used to service the McBSP. The default mapping of DMA channel-complete interrupts to CPU is as follows:

DMA channel 0 ➔ CPU interrupt 8

DMA channel 1 ➔ CPU interrupt 9

DMA channel 2 ➔ CPU interrupt 11

DMA channel 3 ➔ CPU interrupt 12

4. DMA initialization: Program the DMA channel for required operation. Following would be a typical set up:

   Source address = DRR

   Destination address = internal memory or as required.

   Transfer counter = number of elements to be transferred.

   Receive synchronization event, *RSYNC* = REVT from McBSP

   DMA interrupt bit, *TCINT* = enabled

   Priority bit, *PRI* = 1; optional, but recommended.

5. Instruct the DMA channel(s) to run. For example, set START=01b in the DMA channel's primary control register to start the DMA without auto-initialization. The DMA will start the first transfer upon receiving the first read/write sync event.

6. Wake up VBAP by pulling /PDN to a logic high state (if this done by software). Alternatively, this can also be done after power up (before step 1 above) using a switch. It is assumed here that an appropriate /LINSEL value is chosen for the desired operation. The VBAP is now ready for frame sync pulses from the McBSP to start transmission and reception

7. Set /XRST=/RRST=1 to wake up the McBSP. Set /FRST=1 to start the frame sync generator in the McBSP. This causes frame sync pulses on FSX/FSR pins which eventually drive the VBAP.

Conclusion

The programmability of the ‘C6x McBSP provides ease of interface to the VBAP. Although this application note only describes the fixed data rate mode of the VBAP, it is equally simple to interface in the variable-data rate mode. Variable data rate mode allows for varying data rates up to a maximum of 2.048MHz. An example code in the appendix shows McBSP1 initialization and operation.

Appendix A. Sample Source Code

```
/*

  TI Proprietary Information
  Internal Data

        06/22/98: Shaku Anjanaiah

  vbap.c: Uses the peripheral support library

  McBSP1 is used to communicate to the VBAP in 8-b u-law
  companded fixed data rate mode.
  DMA channel 0 and 2 is used to service the McBSP.
  CLKX, CLKR are generated using CLKS clock.
  FSX, FSR are outputs that drive the VBAP's frame syncs.
*/

#include "common.h"

#define M0TO1 FALSE /* McBSP0 unused */
#define M1TO0 TRUE /* McBSP1 used */

#define CLKGDV1             0
#define FPER1         0xFF
#define FWID1         0
#define CLKSM1              CLK_MODE_CLKS

#define M1TO0_MSTR  TRUE
#define XFER_SIZE     32 /* number of 8-b data transferred */
#define XFER_TYPE     DMA_XFER


void init_m0to1(void);
void init_M0_srgr(void);
void init_m1to0(void);
void init_M1_srgr(void);

void
main(void)
{
        int    xfer_size;
        int    xfer_type;
        int    mcsp0to1_rate;
        int    mcsp1to0_rate;

        recv1_done = FALSE;
        xmit1_done = FALSE;

        mcsp1to0 = M1TO0;
        mcsp0to1 = M0TO1;
        xfer_size = XFER_SIZE;
        xfer_type = XFER_TYPE;

        /* Set up SRGR values as needed */
```

```
        init_M1_srgr();

        /* Now, initialize other control registers for McBSP operation */
        if (mcsp1to0)
                init_m1to0();

        /* Enable sample rate generator; /GRST=1 */
        MCBSP_SAMPLE_RATE_ENABLE(1);

        /* Reset all DMA channels */
        switch (xfer_type) {
        case DMA_XFER:
                dma_reset();
                set_interrupts();/* Initialize DMA to service McBSP */
                if (mcsp1to0) {/* uses ch2 for xmit and ch0 for recv*/
                        DMA0_SRC_ADDR = MCBSP_DRR_ADDR(0);
                        DMA0_DEST_ADDR = (unsigned int) in1;
                        REG_WRITE (DMA0_XFR_COUNTER_ADDR, xfer_size);
                        LOAD_FIELD (DMA0_PRIMARY_CTRL_ADDR, DMA_ESIZE32,
                                ESIZE, ESIZE_SZ);
                        SET_BIT (DMA0_PRIMARY_CTRL_ADDR, TCINT);
                        LOAD_FIELD (DMA0_PRIMARY_CTRL_ADDR, DMA_ADDR_INC,
                                DST_DIR, DST_DIR_SZ);
                        LOAD_FIELD (DMA0_PRIMARY_CTRL_ADDR, DMA_DMA_PRI,
                                PRI, 1);
                        LOAD_FIELD (DMA0_PRIMARY_CTRL_ADDR, SEN_REVT0,
                                RSYNC, RSYNC_SZ);
                        DMA_START(DMA_CH0);

                        DMA2_SRC_ADDR = (unsigned int) out1;
                        DMA2_DEST_ADDR = MCBSP_DXR_ADDR(0);
                        REG_WRITE (DMA2_XFR_COUNTER_ADDR, xfer_size);

                        LOAD_FIELD (DMA2_PRIMARY_CTRL_ADDR, DMA_ESIZE32,
                                ESIZE, ESIZE_SZ);
                        SET_BIT (DMA2_PRIMARY_CTRL_ADDR, TCINT);
                        LOAD_FIELD (DMA2_PRIMARY_CTRL_ADDR, DMA_ADDR_INC,
                                SRC_DIR, SRC_DIR_SZ);
                        LOAD_FIELD (DMA2_PRIMARY_CTRL_ADDR, DMA_DMA_PRI,
                                 PRI, 1);
                        LOAD_FIELD (DMA2_PRIMARY_CTRL_ADDR, SEN_XEVT0,
                                WSYNC, WSYNC_SZ);
                        DMA_START(DMA_CH2);
                }
                SET_BIT (MCBSP_SPCR_ADDR(1), RRST);
                SET_BIT (MCBSP_SPCR_ADDR(1), XRST);
                SET_BIT (MCBSP_SPCR_ADDR(1), FRST);

                while (!xmit1_done || !recv1_done);
                break;
        }
        reg_dump();
        CSR |= 0x00007000;      /* PowerDown PD3 to shut off MCSP */
}
```

```
void
init_m1to0(void)
{
        /* PCR setup*/
        LOAD_FIELD (MCBSP_PCR_ADDR(1), FSYNC_POL_HIGH, FSXP, 1);
        LOAD_FIELD (MCBSP_PCR_ADDR(1), M1TO0_MSTR, CLKXM, 1);
        LOAD_FIELD (MCBSP_PCR_ADDR(1), FSYNC_MODE_INT, FSXM, 1);
        /* SRGR setup */
        LOAD_FIELD (MCBSP_SRGR_ADDR(1), FSX_FSG, FSGM, 1);
        /* XCR setup */
        LOAD_FIELD (MCBSP_XCR_ADDR(1), SINGLE_PHASE, XPHASE, 1);
        LOAD_FIELD (MCBSP_XCR_ADDR(1), WORD_LENGTH_8,
                        XWDLEN1, XWDLEN1_SZ);
        LOAD_FIELD (MCBSP_XCR_ADDR(1), 0, XFRLEN1, XFRLEN1_SZ);
        LOAD_FIELD (MCBSP_XCR_ADDR(1), DATA_DELAY1, XDATDLY, XDATDLY_SZ);
        LOAD_FIELD (MCBSP_XCR_ADDR(1), COMPAND_ULAW,
                        XCOMPAND, XCOMPAND_SZ);

        /* PCR setup*/
        LOAD_FIELD (MCBSP_PCR_ADDR(1), FSYNC_POL_HIGH, FSRP, 1);
        LOAD_FIELD (MCBSP_PCR_ADDR(1), M1TO0_MSTR, CLKRM, 1);
        LOAD_FIELD (MCBSP_PCR_ADDR(1), FSYNC_MODE_INT, FSRM, 1);
        /* RCR setup */
        LOAD_FIELD (MCBSP_RCR_ADDR(1), SINGLE_PHASE, RPHASE, 1);
        LOAD_FIELD (MCBSP_RCR_ADDR(1), WORD_LENGTH_8,
                        RWDLEN1, RWDLEN1_SZ);
        LOAD_FIELD (MCBSP_RCR_ADDR(1), 0, RFRLEN1, RFRLEN1_SZ);
        LOAD_FIELD (MCBSP_RCR_ADDR(1), DATA_DELAY1, RDATDLY, RDATDLY_SZ);
        LOAD_FIELD (MCBSP_RCR_ADDR(1), COMPAND_ULAW, RCOMPAND,
                        RCOMPAND_SZ);
        /* SPCR */
        LOAD_FIELD (MCBSP_SPCR_ADDR(1), RXJUST_RJZF, RJUST, RJUST_SZ);
}


void
init_M1_srgr(void)
{
        LOAD_FIELD (MCBSP_SRGR_ADDR(1), CLKGDV1, CLKGDV, CLKGDV_SZ);
        LOAD_FIELD (MCBSP_SRGR_ADDR(1), FWID1, FWID, FWID_SZ);
        LOAD_FIELD (MCBSP_SRGR_ADDR(1), FPER1, FPER, FPER_SZ);
        LOAD_FIELD (MCBSP_SRGR_ADDR(1), CLKSM1, CLKSM, 1);
        LOAD_FIELD (MCBSP_SRGR_ADDR(1), CLKS_POL_RISING, CLKSP, 1);
        LOAD_FIELD (MCBSP_SRGR_ADDR(1), GSYNC_OFF, GSYNC, 1);
}


void
init_M0_srgr(void)
{}
```

```
void
reg_dump(void)
{
        int  i;
        int     *m0 = (int *) MCBSP_ADDR(0);
        int     *m1 = (int *) MCBSP_ADDR(1);

        for (i = 0; i < 10; i++) {
                regdump0[i] = m0[i];
                regdump1[i] = m1[i];
        }
}

void
set_interrupts(void)
{
        intr_init();
        INTR_MAP_RESET;
        /* Hook interrupt service routine to an interrupt */
        intr_hook (c_int11, CPU_INT11);
        intr_hook (c_int08, CPU_INT8);


        /* enable NMIE, default CPU interrrupt 11 correponding to DMA channel 2 */
        INTR_ENABLE(CPU_INT_NMI);/* Enable NMIE */
        INTR_GLOBAL_ENABLE; /* Set GIE in CSR*/
        INTR_ENABLE(11);
        /* default CPU interrrupt 8 correponding to DMA channel 0 */
        INTR_ENABLE(8);
        return;
}


/* ================================= */
/* DMA DATA TRANSFER COMPLETION ISRS */
/* ================================= */

interrupt void
c_int11(void) /* DMA ch2 */
{
        xmit1_done = TRUE;
        return;
}

interrupt void
c_int08(void) /* DMA ch0 */
{
        recv1_done = TRUE;
        return;
}
```

```
/*************************************************************************/
/* COMMON.H V1.00                                                        */
/* Copyright (c) 1997 Texas Instruments Incorporated                     */
/*************************************************************************/

#include <dma.h>
#include <emif.h>
#include <intr.h>
#include <timer.h>
#include <cache.h>
#include <hpi.h>
#include <mcbsp.h>
#include <regs.h>
#include <stdio.h>
#include <trgcio.h>
#include <stdlib.h>

/* variables used in tcase */
int     mcsp0to1;
int     mcsp1to0;
volatile int xmit1_done;
volatile int recv0_done;
volatile int xmit0_done;
volatile int recv1_done;


#define FALSE       0
#define TRUE        1

/* BUFFERS DEFINED IN data6201.asm */

#define BUFFER_SIZE 256
#define COMPAND_SIZE          4096

extern int in0[BUFFER_SIZE];
extern int in1[BUFFER_SIZE];
extern int out0[BUFFER_SIZE];
extern int out1[BUFFER_SIZE];
extern int regdump0[10];
extern int regdump1[10];
extern int ulawenc[BUFFER_SIZE];

extern cregister volatile unsigned int AMR;
extern cregister volatile unsigned int CSR;
extern cregister volatile unsigned int IFR;
extern cregister volatile unsigned int ISR;
extern cregister volatile unsigned int ICR;
extern cregister volatile unsigned int IER;


extern interrupt void c_nmi01(void);
extern interrupt void c_int04(void);
extern interrupt void c_int05(void);
extern interrupt void c_int06(void);
```

```
extern interrupt void c_int07(void);
extern interrupt void c_int08(void);
extern interrupt void c_int09(void);
extern interrupt void c_int10(void);
extern interrupt void c_int11(void);
extern interrupt void c_int12(void);
extern interrupt void c_int13(void);
extern interrupt void c_int14(void);
extern interrupt void c_int15(void);

#define DMA_XFER      0
#define POLL_XFER     1
#define INT_XFER      2
#define GPIO          3
#define DLB1          4
#define DLB2          5
#define SPLIT_XFER    6
#define HW_BYTE  7
#define DMA_SPI             8
#define DMA_STB             9
#define DMA_MCM_FLY         10
#define DMA_NEW_FRAMESYNC       11
#define AUTO_INIT     12
#define DMA_SORT      13
#define SPLIT_SORT    14
#define DMA_SYNCERR         15

#define DMA_BYTE      16
#define DMA_HALFWORD        17

extern void
reset_mcbsp(void)
{
        /* the slave has to wake up before the frame master so that frames
        are not lost */
        if (mcsp0to1) {
                if (GET_BIT(MCBSP_PCR_ADDR(1), FSRM)) {/*(mcsp1->pcr.fsrm) */
                        SET_BIT (MCBSP_SPCR_ADDR(0), RRST);
                        SET_BIT (MCBSP_SPCR_ADDR(1), XRST);
                        SET_BIT (MCBSP_SPCR_ADDR(1), FRST);
                } else {
                        SET_BIT (MCBSP_SPCR_ADDR(1), RRST);
                        SET_BIT (MCBSP_SPCR_ADDR(0), XRST);
                        SET_BIT (MCBSP_SPCR_ADDR(0), FRST);
                }
        }
        if (mcsp1to0) {
                if (GET_BIT(MCBSP_PCR_ADDR(0), FSRM)){ /*(mcsp0->pcr.fsrm) */
                        SET_BIT (MCBSP_SPCR_ADDR(1), XRST);
                        SET_BIT (MCBSP_SPCR_ADDR(0), RRST);
                        SET_BIT (MCBSP_SPCR_ADDR(0), FRST);
                } else {
                        SET_BIT (MCBSP_SPCR_ADDR(0), RRST);
                        SET_BIT (MCBSP_SPCR_ADDR(1), XRST);
```

```
                        SET_BIT (MCBSP_SPCR_ADDR(1), FRST);
                }
        }
}

extern void set_interrupts(void);
extern void reg_dump(void);
```