Application Report
SPRA489A - August 2001

TMS320C6000 McBSP to Voice Band Audio Processor (VBAP) Interface

Shaku Anjanaiah and Vassos Soteriou — *Digital Signal Processing Solutions*

ABSTRACT

This document describes how to interface the multichannel buffered serial port (McBSP) in the TMS320C6000™ digital signal processor (DSP) to a voice band audio processor (VBAP). The VBAP under discussion is the TI TLV320AC56, 3V, 2.048 MHz audio processor which is a μ-law companding device. The interface is also applicable to TI's TLV320AC57, an A-law companding audio processor.

The highly programmable McBSP provides a glueless interface to the VBAP. The VBAP processes analog signals from audio sources such as a microphone, converts it to digital data (in two choices), 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 sample code described in this application report can be downloaded from http://www.ti.com/lit/zip/SPRA489.

Contents

List of Figures

List of Tables

1 Design Problem

How do I interface a voice-band audio processor (VBAP) to the TMS320C6000 multichannel buffered serial port (McBSP)?

1.1 Overview

The block diagram interface between the McBSP 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 the compand operation is chosen in the VBAP) or transforms the analog data to 13-bit linear data (if the 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 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 A-law format if interfacing to the TLV320AC57 or μ−law format if interfacing to TLV320AC56. The μ-law and A-law companding standards are part of the CCITT G.711 recommendation.

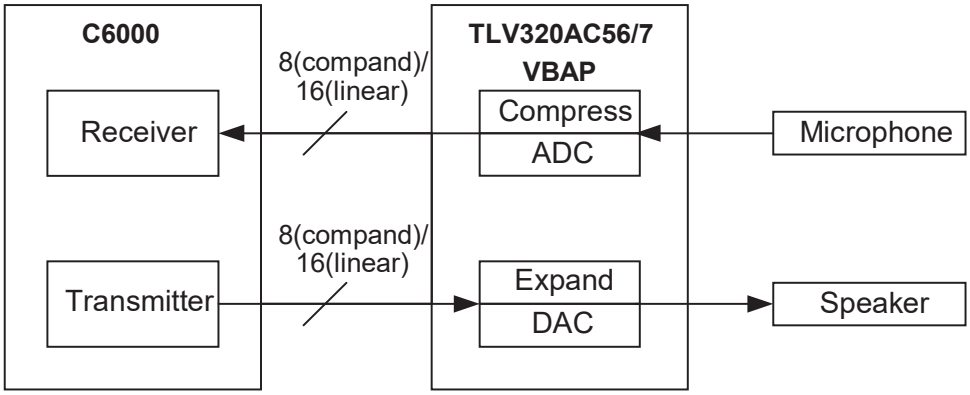


Figure 1. C6000 McBSP and VBAP Interface Block Diagram

1.2 VBAP Operation

The VBAP offers two pin-selectable modes of operation:

- 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 the rising edge of clock and receive data sampled on falling edge of CLK. The master clock frequency is 2.048 MHz. The 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 the falling edge of DCLKR and transmit data is output on the 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−bit 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. The 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 following literature:

- *Designing with the Voice−Band Audio Processor*, Application Report, February 1999, Literature Number SLWA001
- *TLV320AC56, TLV320AC57 Voice−Band Audio Processors (VBAP™)* Data Sheet, June 1996 – revised October 1997, Literature Number SLWS044

1.3 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 McBSP. Also, the electret type microphone and speaker (or headset) should be a 3V part. The interface includes:

- 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 with CLKGDV=0 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.

  Alternatively, the timer can be used to generate the 2.048MHz shift clock using the (CPU clock/4) as the timer clock source. For a 200MHz DSP, the timer CLKSRC would be 50MHz. The timer is used in its clock mode (50% duty cycle). To derive a 488 ns period for the shift clock, the timer period register value is calculated as follows:

  sclk_period = (2 * Period_register) / f(CLKSRC)

  Hence, period_register = 488 / 2*20 = 12 approx.
- 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 a 125 μS period (8kHz) frame sync based on a 2.048 MHz clock, the frame period (/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).

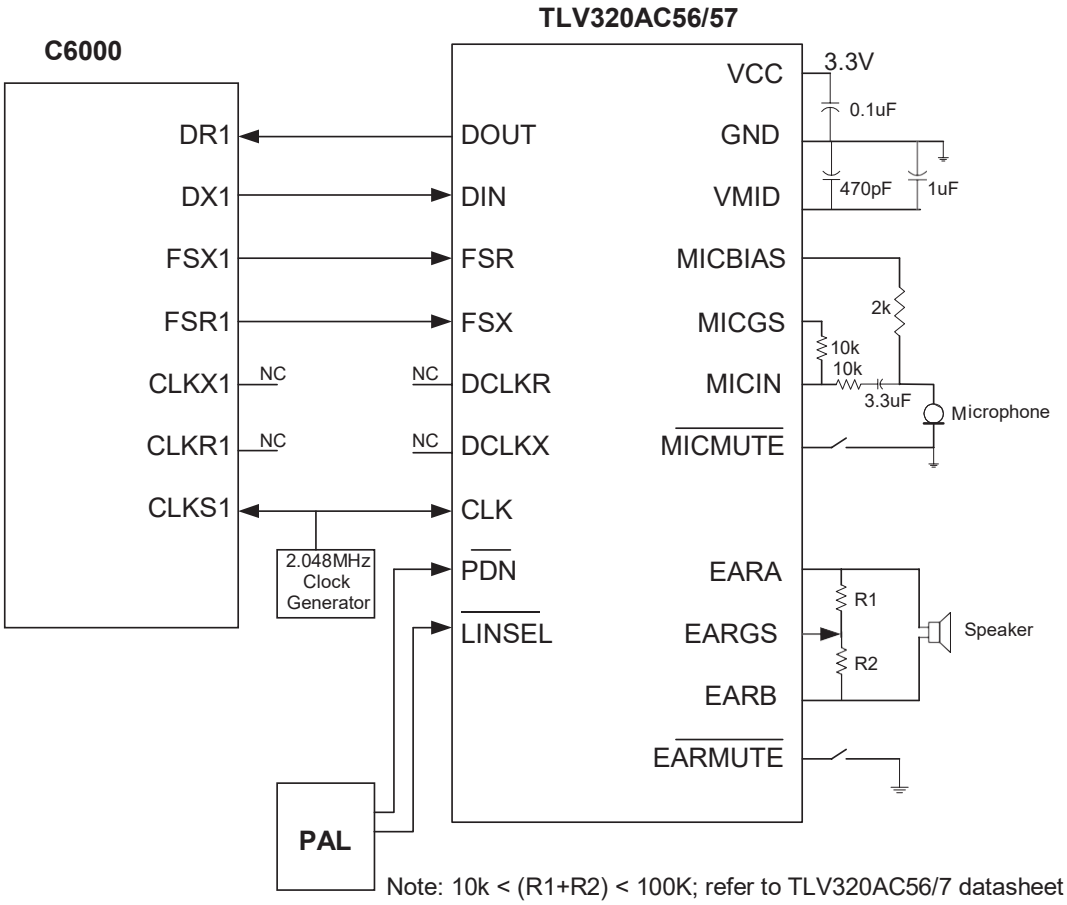
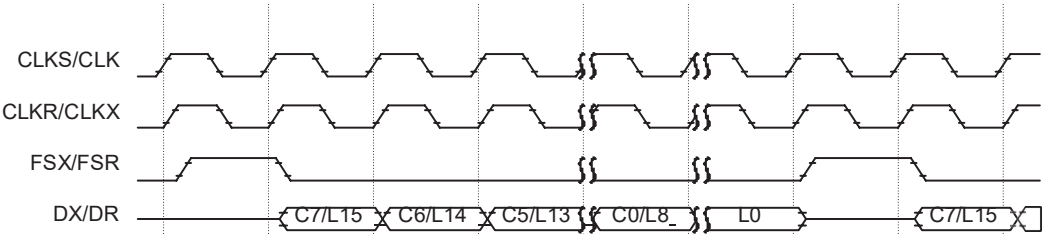


Figure 2. C6000 and VBAP Schematic for Fixed Rate Data Mode

- 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 are based on the information provided in the VBAP datasheet.
- Speaker Interface: The 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 the 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. The PD pin of the DSP can be inverted in the PAL to provide this control. Alternatively, a simple switch connected to /PDN and GND will also suffice.

1.4 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 the McBSP's CLKS pin and VBAP's CLK pin. The CLKS drives the bit−clocks CLKR and CLKX with no divide−downs for McBSP data transfer. Frame syncs are generated on the rising edge of CLKR/CLKX. The first data bit (MSB) is transmitted/received with a 1 bit−clock delay from FSX/FSR to be compliant with the Fixed Data Rate Mode. Therefore (R/X)DATDLY=1.



Cx/Ly: CompandData X / LinearData Y

Figure 3. Fixed Data Rate Mode Timing Diagram

The above functional requirements drive the values for the programmable McBSP registers. Figure 4 through Figure 8 and Table 1 show the register values for the application.

31	30 24	23 21	20 19	18	17 16
RPHASE	RFRLEN2	RWDLEN2	RCOMPAND	RFIG	RDATDLY
0	0	0	0x2	0	1

15	14 8	7 5	4	3 0
reserved	RFRLEN1	RWDLEN1	RWDREVRS	reserved
0	0	0	0	0

Figure 4. Receive Control Register (RCR)

31	30 24	23 21	20 19	18	17 16
XPHASE	XFRLEN2	XWDLEN2	XCOMPAND	XFIG	XDATDLY
0	0	0	0x2	0	1

15	14 8	7 5	4	3 0
reserved	XFRLEN1	XWDLEN1	XWDREVRS	reserved
0	0	010	0	0

Figure 5. Transmit Control Register (XCR)

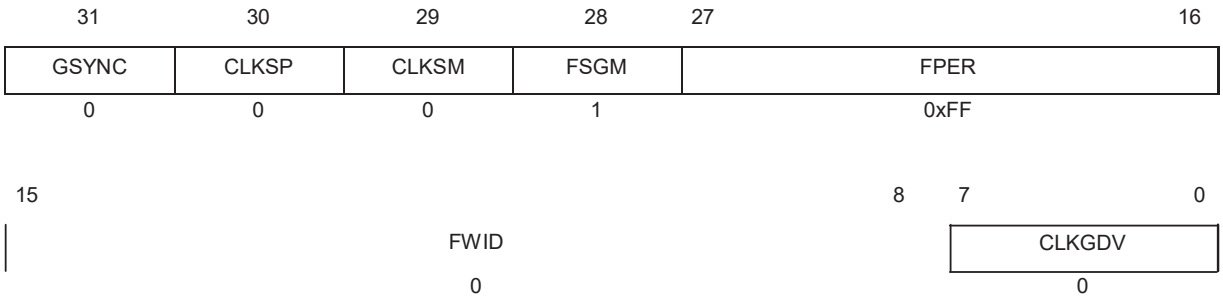


Figure 6. Sample Rate Generator Register (SRGR)

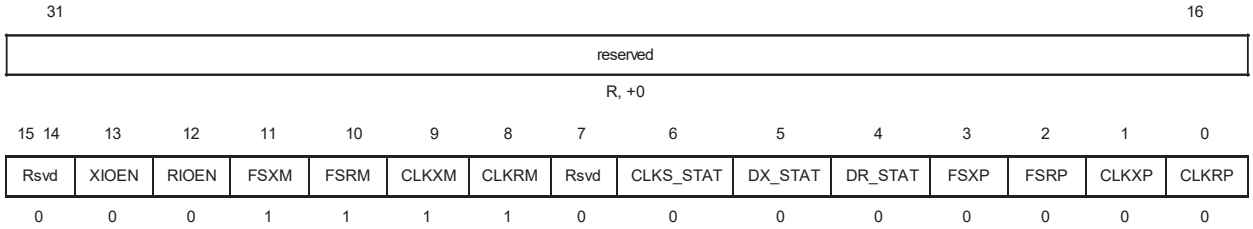


Figure 7. Pin Control Register (PCR)

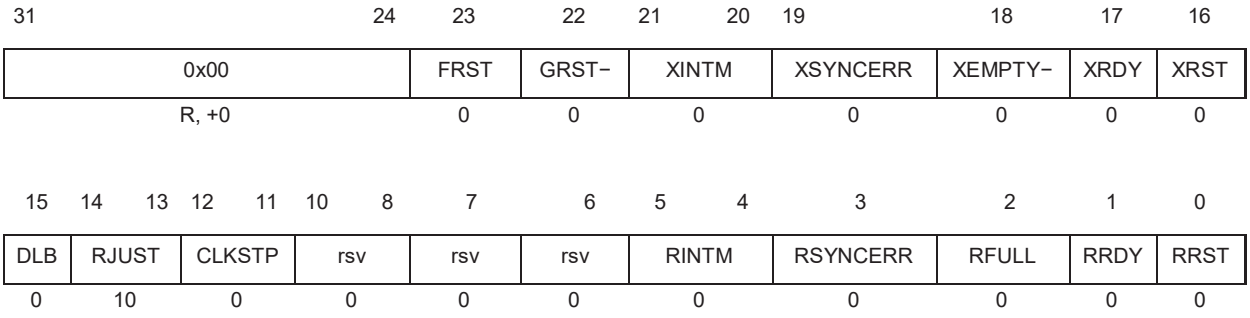


Figure 8. Serial Port Control Register (SPCR)

Table 1. McBSP Register Configuration Values

Register	Bit−field	Value	HAL Macro	Description
RCR[20:19]	RCOMPAND	0x2	MCBSP_RCR_RCOMPAND_ULAW	Receive μ−law companding
RCR[17:16]	RDATDLY	0x1	MCBSP_RCR_RDATDLY_1BIT	I bit−clock data delay
XCR[20:19]	XCOMPAND	0x2	MCBSP_XCR_XCOMPAND_ULAW	Transmit μ−law companding
XCR[17:16]	XDATDLY	0x1	MCBSP_RCR_XDATDLY_1BIT	I bit−clock data delay
SRGR[28]	FSGM	0x1	MCBSP_SRGR_FSGM_FSG	FSX generated by FSG
SRGR[27:16]	FPER	0xFF	MCBSP_SRGR_FPER_OF(0xFF)	Frame period of 256 2.048Mhz clock periods to get 8KHz frame sync sampling rate
SRGR[15:8]	FWID	0x0	MCBSP_SRGR_FWID_OF(0x0)	Generates 1 clock period active−high pulse.
SRGR[7:0]	CLKGDV	0x0	MCBSP_SRGR_FWID_OF(0x0)	2.048 CLKS drives CLKR/X with no divide−down
PCR[11]	FSXM	0x1	MCBSP_PCR_FSXM_INTERNAL	FSX is an output
PCR[10]	FSRM	0x1	MCBSP_PCR_FSRM_INTERNAL	FSR is an output
PCR[9]	CLKXM	0x1	MCBSP_PCR_CLKXM_OUTPUT	CLKX is an output generated by CLKS
PCR[8]	CLKRM	0x1	MCBSP_PCR_CLKRM_OUTPUT	CLKR is an output generated by CLKS
SPCR[31:0]	all	default	MCBSP_SPCR_(FIELD)_DEFAULT	Reset bits will be driven as per initialization procedure

1.5 McBSP and VBAP 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 VBAP and the DSP (except when the McBSP is not out of reset) are 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 can be driven low to send it to the power-down state when needed.

The following steps describe the setup of interrupts, DMA/EDMA, and the McBSP in required order. The C64x, C621x and C671x devices use the EDMA. The C620x and C670x devices use the DMA. The McBSP initialization procedure using the CPU or (E)DMA is also discussed in the Application Report SPRA488, *TMS320C6000 McBSP Initialization*.

1. If the McBSP is not in reset state, set /XRST=/RRST=0 in SPCR.
2. Program the McBSP configuration registers XCR, PCR, RCR and SRGR to the values shown in Table 1. Do not set the /GRST bit in the SPCR in this step.
3. Take the sample rate generator (SRGR) out of reset by setting /GRST=1
4. Either (a) or (b) should be followed:
   a. If the DMA is used:

      Hook up the DMA channels to the interrupt service routines. In the sample code provided, DMA channel 1 is hooked up to interrupt 9 for data receive and DMA channel 2 is hooked up to interrupt 11 for data transmit. The CPU interrupts that correspond to the DMA channels that will be used to service the McBSP should then be enabled. The default mapping of DMA channel-complete interrupts to the CPU is the following:

   − DMA channel 0 to CPU interrupt 8
   − DMA channel 1 to CPU interrupt 9

- DMA channel 2 to CPU interrupt 11
- DMA channel 3 to CPU interrupt 12

b. If the EDMA is used:

Hook up the EDMA channels to the interrupt service routine. 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 various control registers and bit fields facilitate EDMA interrupt generation. Interrupt 8 that is going to service the McBSP transfers via the EDMA should then be enabled.

5. 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 VBAP by pulling /PDN to a logic high state (if this is done using software) Alternatively, this can 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. Then, take the McBSP out of reset (Set /XRST=/RRST=1 to enable the McBSP). Set /FRST =1 to start the frame sync generator in the McBSP. This causes the frame sync pulses on the FSX/FSR pins which eventually drive the VBAP. For details on DMA initialization for servicing the McBSP, refer to TMS320C6000 McBSP Initialization (SPRA488) and TMS320C6000 DMA Applications (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 VBAP by pulling /PDN to a logic high state (if this is done using software) Alternatively, this can 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. Then, pull the McBSP out of reset (Set /XRST=/RRST=1 to enable the McBSP). Set /FRST =1 to start the frame sync generator in the McBSP. This causes the frame sync pulses on the FSX/FSR pins which eventually drive the VBAP. For details on EDMA initialization for servicing the McBSP, refer to TMS320C6000 McBSP Initialization (SPRA488) and TMS320C6000 Enhanced DMA: Example Applications (SPRA636).

1.6 Timing Analysis

As shown in Figure 2, C6000 and VBAP Schematic for Fixed Data Rate Mode, FSX and FSR are outputs of the McBSP driving the VBAP. CLKR and CLKX are generated internally via the external clock source CLKS without any divide-down. Figure 9 shows the timing relation of transmit data and frame sync, receive data and frame sync with respect to their respective clocks.

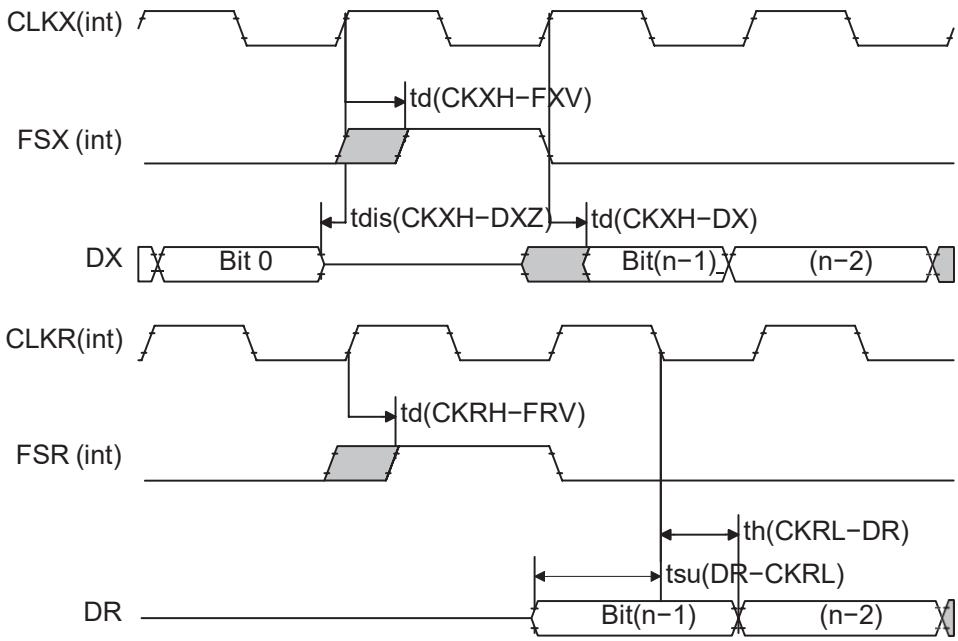


Figure 9. McBSP Timing for Internal Clocks and Frames

The timing analysis shown in Table 2 satisfies the hardware interface requirements. The timing numbers of the McBSP match with that of the VBAP with sufficient timing margins. The timing numbers for the VBAP correspond to the TLV320AC56/37 datasheet, Literature Number SLWS044B. The McBSP timings can be found in all C6000 DSP datasheets. Table 2 uses the McBSP timing parameters from the *TMS320C6701 Floating-Point Signal Processor* Data Sheet, Literature Number SPRS067E, May 1998, revised May 2000.

Table 2. Timing Analysis for McBSP and VBAP

VBAP Timing Requirements		C6701 Switching Characteristics		Unit
tsu (DIN)min Data In Setup Time	20	H − td(CKXH−DXV)max = 244.14 – 4	240.14	ns
th (DIN)min Data In Hold Time	20	L + td(CKXH−DXV)min = 244.14 − 2	242.14	ns
tsu (FSR)min FSR Setup Time	20	H − td(CKXH−FXV)max = 244.14 – 5	239.14	ns
th (FSR)min FSR Hold Time	20	L + td(CKXH−FXV)min = 244.14 − 4	240.14	ns
tsu (FSX)min FSX Setup Time	20	H − td(CKRH−FRV)max = 244.14 – 4	240.14	ns
th (FSX)min FSX Hold Time	20	L + td(CKRH−FRV)min = 244.14 – 4	240.14	ns
C6701 Timing Requirements		**VBAP Switching Characteristics**		
tsu(DRV−CKXL)min Data In Setup Time	10	H − tpd2(max) = 244.14 – 35	209.14	ns
th(CKXL−DRV)min Data In Hold Time	4	L + tpd2(min) = 244.14 + 0	244.14	ns

NOTE: The following is true for the above calculations:

1. Since CLKGDV = 0, CLK(R/X) derived from CPU clock will have a 50% duty cycle. Therefore H=L where H represents clock high duration and L represents clock low duration.
2. Period of CLK(R/X), T = H+L = 488.28 ns for a 2.048 MHz CLKS clock.

1.7 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 McBSP0 initialization and operation.

Appendix A Sample Source Code

```
/*-------------------------------------------------------------------------------*/
/*  mcbsp_to_vbap.c V1.00                                                        */
/*  Copyright (c) 2001 Texas Instruments Incorporated                            */
/*-------------------------------------------------------------------------------*/
/*
    Original by: Shaku Anjanaiah
    Modified by: Vassos S. Soteriou

   mcbsp_to_vbap.c:
   This program uses the McBSP of the TMS320C6000 devices to
   communicate with a VBAP in 8-bit ulaw companded fixed data
   rate mode. 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 (external) clock.
   FSX and FSR are outputs that drive the VPAB'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_6202 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 */

/*-------------------------------------------------------------------------------*/
/* Define constants */
#define FALSE 0
#define TRUE 1

#define DMA_VBAP 8
#define XFER_TYPE DMA_VBAP
#define BUFFER_SIZE 256 /* BUFFER_SIZE should be >= ELEMENT_COUNT */
#define ELEMENT_COUNT 32 /* Number of 8-bit datum to transfer      */

/* 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

/*---------------------------------------------------------------------------*/
/* External functions and function prototypes */

void init_mcbsp0_vbap(void);   /* Function prototypes */
void set_interrupts_dma(void);
void set_interrupts_edma(void);

/* Include 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, xfer_type;

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;
xfer_type = XFER_TYPE;

/* initialize the CSL library */
CSL_init();

init_mcbsp0_vbap();

/* Enable sample rate generator   GRST=1 */
MCBSP_enableSrgr(hMcbsp0);       /* Handle to SRGR */

switch (xfer_type) {
case DMA_VBAP:

  #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_DEFAULT,
           DMA_PRICTL_SRCRLD_DEFAULT,
           DMA_PRICTL_EMOD_DEFAULT,
           DMA_PRICTL_FS_DEFAULT,
           DMA_PRICTL_TCINT_ENABLE, /* TCINT =1                          */
           DMA_PRICTL_PRI_DMA,      /* DMA high priority                 */
```

```
            DMA_PRICTL_WSYNC_DEFAULT,
            DMA_PRICTL_RSYNC_REVT0, /* Set synchronization event REVT0=01101 */
            DMA_PRICTL_INDEX_DEFAULT,
            DMA_PRICTL_CNTRLD_DEFAULT,
            DMA_PRICTL_SPLIT_DISABLE,
            DMA_PRICTL_ESIZE_32BIT,  /* Element size is 32 bits              */
            DMA_PRICTL_DSTDIR_INC, /* Increment destination by element size */
            DMA_PRICTL_SRCDIR_DEFAULT,
            DMA_PRICTL_START_DEFAULT
            ),
         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_DEFAULT,
            DMA_SECCTL_WSYNCCLR_DEFAULT,
            DMA_SECCTL_WSYNCSTAT_DEFAULT,
            DMA_SECCTL_RSYNCCLR_DEFAULT,
            DMA_SECCTL_RSYNCSTAT_DEFAULT,
            DMA_SECCTL_WDROPIE_DEFAULT,
            DMA_SECCTL_WDROPCOND_DEFAULT,
            DMA_SECCTL_RDROPIE_DEFAULT,
            DMA_SECCTL_RDROPCOND_DEFAULT,
            DMA_SECCTL_BLOCKIE_ENABLE, /* BLOCK IE=1 enables DMA channel int */
            DMA_SECCTL_BLOCKCOND_DEFAULT,
            DMA_SECCTL_LASTIE_DEFAULT,
            DMA_SECCTL_LASTCOND_DEFAULT,
            DMA_SECCTL_FRAMEIE_DEFAULT,
            DMA_SECCTL_FRAMECOND_DEFAULT,
            DMA_SECCTL_SXIE_DEFAULT,
            DMA_SECCTL_SXCOND_DEFAULT
            ),
         DMA_SRC_RMK(MCBSP_ADDRH(hMcbsp0, DRR)),
         DMA_DST_RMK((Uint32)dmaInbuff),
         DMA_XFRCNT_RMK(
            DMA_XFRCNT_FRMCNT_DEFAULT,
            DMA_XFRCNT_ELECNT_OF(element_count) /* set xfer 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_DEFAULT,
            DMA_PRICTL_SRCRLD_DEFAULT,
            DMA_PRICTL_EMOD_DEFAULT,
            DMA_PRICTL_FS_DEFAULT,
            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_DEFAULT,
            DMA_PRICTL_INDEX_DEFAULT,
            DMA_PRICTL_CNTRLD_DEFAULT,
            DMA_PRICTL_SPLIT_DEFAULT,
            DMA_PRICTL_ESIZE_32BIT, /* Element size is 32 bits */
            DMA_PRICTL_DSTDIR_DEFAULT,
            DMA_PRICTL_SRCDIR_INC,   /* Increment source by element size */
            DMA_PRICTL_START_DEFAULT
            ),
         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_DEFAULT,
            DMA_SECCTL_WSYNCCLR_DEFAULT,
            DMA_SECCTL_WSYNCSTAT_DEFAULT,
            DMA_SECCTL_RSYNCCLR_DEFAULT,
```

```
            DMA_SECCTL_RSYNCSTAT_DEFAULT,
            DMA_SECCTL_WDROPIE_DEFAULT,
            DMA_SECCTL_WDROPCOND_DEFAULT,
            DMA_SECCTL_RDROPIE_DEFAULT,
            DMA_SECCTL_RDROPCOND_DEFAULT,
            DMA_SECCTL_BLOCKIE_ENABLE, /* BLOCK IE=1 enables DMA channel int    */
            DMA_SECCTL_BLOCKCOND_DEFAULT,
            DMA_SECCTL_LASTIE_DEFAULT,
            DMA_SECCTL_LASTCOND_DEFAULT,
            DMA_SECCTL_FRAMEIE_DEFAULT,
            DMA_SECCTL_FRAMECOND_DEFAULT,
            DMA_SECCTL_SXIE_DEFAULT,
            DMA_SECCTL_SXCOND_DEFAULT
            ),
         DMA_SRC_RMK((Uint32)dmaOutbuff),
         DMA_DST_RMK(MCBSP_ADDRH(hMcbsp0, DXR)),
         DMA_XFRCNT_RMK(
            DMA_XFRCNT_FRMCNT_DEFAULT,
            DMA_XFRCNT_ELECNT_OF(element_count)  /* set xfer element count      */
            )
);

set_interrupts_dma(); /* initialize the interrupts                              */
                      /* enable the interrupts after the DMA channels are opened */
                      /* as the DMA_OPEN_RESET clears and disables the channel   */
                      /* interrupt once specified and clears the corresponding   */
                      /* interrupt bits in the IER. This is not applicable for   */
                      /* the EDMA channel open case                              */
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, element_count), /* set count equal to 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, element_count), /* set count equal to 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 */
} /* end of switch here */


MCBSP_enableRcv(hMcbsp0); /* Enable McBSP0 rcv/xmt channel             */
MCBSP_enableXmt(hMcbsp0); /* McBSP port 0 as the transmitter/receiver  */

MCBSP_enableFsync(hMcbsp0); /* Enable frame sync generation for McBSP0 */

/* 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

} /* end main, program ends here */

/*-------------------------------------------------------------------------------*/
/* init_mcbsp0_vbap()                                                            */
/*-------------------------------------------------------------------------------*/
/* MCBSP Config structure */
/* Setup the MCBSP_0 for transfers with the VBAP */
void
init_mcbsp0_vbap(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_SINGLE, /* Single phase receive frame */
           MCBSP_RCR_RFRLEN2_DEFAULT,
           MCBSP_RCR_RWDLEN2_DEFAULT,
           MCBSP_RCR_RCOMPAND_ULAW,   /* ULAW companding             */
           MCBSP_RCR_RFIG_DEFAULT,
           MCBSP_RCR_RDATDLY_1BIT,  /*  1-bit  receive  data  delay  */
           MCBSP_RCR_RFRLEN1_OF(0x0), /* frame length of 0 elements */
           MCBSP_RCR_RWDLEN1_8BIT,  /*  receive  elements  =  8  bits  */
           MCBSP_RCR_RWDREVRS_DEFAULT
           ),
#endif
#if (DMA_SUPPORT)
    MCBSP_RCR_RMK(
           MCBSP_RCR_RPHASE_SINGLE, /* Single phase receive frame */
           MCBSP_RCR_RFRLEN2_DEFAULT,
           MCBSP_RCR_RWDLEN2_DEFAULT,
           MCBSP_RCR_RCOMPAND_ULAW,   /* ULAW companding             */
           MCBSP_RCR_RFIG_DEFAULT,
           MCBSP_RCR_RDATDLY_1BIT,  /*  1-bit  receive  data  delay  */
           MCBSP_RCR_RFRLEN1_OF(0x0), /* frame length of 0 elements */
           MCBSP_RCR_RWDLEN1_8BIT /* receive elements = 8 bits */
           ),
#endif
#if (EDMA_SUPPORT)
    MCBSP_XCR_RMK(
           MCBSP_XCR_XPHASE_SINGLE, /* Single phase transmit frame */
           MCBSP_XCR_XFRLEN2_DEFAULT,
           MCBSP_XCR_XWDLEN2_DEFAULT,
           MCBSP_XCR_XCOMPAND_ULAW,   /* ULAW companding             */
           MCBSP_XCR_XFIG_DEFAULT,
           MCBSP_XCR_XDATDLY_1BIT,  /*  1-bit  transmit  data  delay  */
           MCBSP_XCR_XFRLEN1_OF(0x0), /* frame length of 0 elements */
           MCBSP_XCR_XWDLEN1_8BIT,  /*  receive  elements  =  8bits  */
           MCBSP_XCR_XWDREVRS_DEFAULT
           ),
#endif
#if (DMA_SUPPORT)
    MCBSP_XCR_RMK(
           MCBSP_XCR_XPHASE_SINGLE, /* Single phase transmit frame */
           MCBSP_XCR_XFRLEN2_DEFAULT,
           MCBSP_XCR_XWDLEN2_DEFAULT,
           MCBSP_XCR_XCOMPAND_ULAW,   /* ULAW companding             */
           MCBSP_XCR_XFIG_DEFAULT,
           MCBSP_XCR_XDATDLY_1BIT,  /*  1-bit  transmit  data  delay  */
           MCBSP_XCR_XFRLEN1_OF(0x0), /* frame length of 0 elements */
           MCBSP_XCR_XWDLEN1_8BIT /* receive elements = 8bits */
           ),
#endif
     MCBSP_SRGR_RMK(
           MCBSP_SRGR_GSYNC_FREE,    /* Free running driven by CLKS                */
           MCBSP_SRGR_CLKSP_RISING,  /* On rising clock edge                       */
           MCBSP_SRGR_CLKSM_CLKS, /* External clock source, CLKS, derives 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(0x0),  /* Frame width is zero                        */
            MCBSP_SRGR_CLKGDV_OF(0x0) /* 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,   /*   FSX   is   an   output   */
            MCBSP_PCR_FSRM_INTERNAL,   /*   FSR   is   an   output   */
            MCBSP_PCR_CLKXM_OUTPUT,  /* output generated by CLKS */
            MCBSP_PCR_CLKRM_OUTPUT,  /* output generated by CLKS */
            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 mcbsp_to_vbap.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 mcros that instantiates one entry in the inetrrupt 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
*******************************************************************************
```

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265