Using a TMS320C6000 McBSP for Data Packing

Ivan Garcia *Digital Signal Processing Solutions*

ABSTRACT

This application report describes how to use the multichannel buffered serial port (McBSP) in the Texas Instruments TMS320C6000™ digital signal processor (DSP) for data packing. Data packing involves moving either multiple successive 8-bit elements to/from the McBSP as a single 16/24/32-bit element or multiple successive 16-bit words to/from the McBSP as a single 32-bit word.

The McBSP in the C6000™ DSP can implement data packing, thereby reducing the bus bandwidth. This application report provides two solutions by which the highly programmable McBSP performs data packing. The first solution manipulates the data frame length and element length. The second solution sets the frame sync ignore bits of the McBSP.

In addition, this application report contains sample data packing C code. The sample code described in this application report can be downloaded from http://www.ti.com/zip/SPRA551.

Contents

List of Figures

List of Tables

1 Design Problem

How can the multichannel buffered serial port (McBSP) in the TMS320C6000™ digital signal processor (DSP) be used for data packing?

2 Overview

A frame sync signal in the McBSP defines the beginning of a frame of a serial element transfer. By programming the (R/X)PHASE field in the receive/transmit control register (RCR/XCR), you can specify either a single-phase or dual-phase frame transfer. All elements in a phase must have the same number of bits. This application report focuses on single-phase frame operations, although dual-phase frame operations can also achieve data packing. The (R/X)WDLEN field in RCR/XCR defines the word (element) length in a frame, which can be 8, 12, 16, 20, 24, or 32 bits. The McBSP can handle up to 128 elements in a single-phase frame or up to 256 elements in a dual-phase frame. Frame length is programmable using the (R/X)FRLEN field in RCR/XCR.

A normal operation, in which six 8-bit elements are transmitted to and from the McBSP, requires six reads of the data receive register (DRR) and six writes to the data transmit register (DXR), respectively, to handle the 48 bits of receive data and the 48 bits of transmit data. Figure 1 shows this operation.

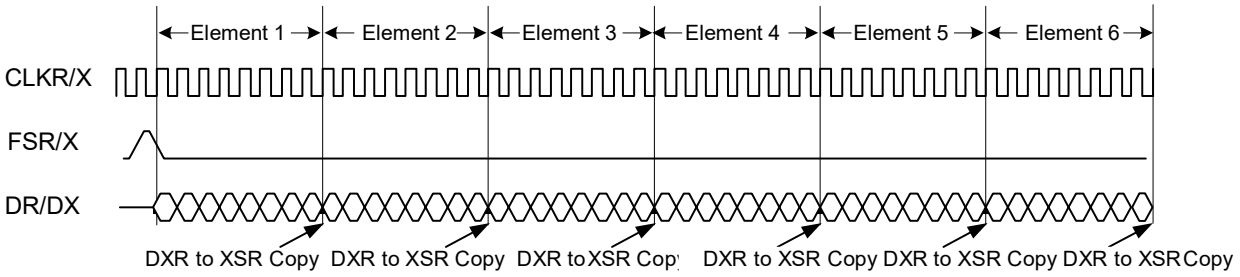


Figure 1. Timing Diagram for Data Transfer of Six 8-Bit Elements (With No Packing)

TMS320C6000 is a trademark of Texas Instruments.

In this configuration, the (E)DMA (direct memory access) needs to service the McBSP a total of 12 times—six 8-bit reads from the DRR and six 8-bit writes to the DXR. When the McBSP is operating at maximum frequency, as shown in Figure 1, there are ways to efficiently pack the transfer data so that the number of transfers required is reduced.

This application report discusses two solutions to the problem. Solution 1 achieves data packing by manipulating the data frame length and element length. Solution 2 shows how to pack data by setting the frame sync ignore bits of the McBSP.

3 Solution 1

Solution 1 achieves data packing by controlling the data receive/transmit frame length (R/X)FRLEN and receive/transmit word (element) length (R/X)WDLEN bits of the McBSP.

3.1 Data Packing by Controlling (R/X)FRLEN and (R/X)WDLEN

The first solution to the problem is to set the frame length and element length to pack the transfer data. The six 8-bit elements in Figure 1 can alternatively be viewed as a data stream of two 24-bit elements in a single frame. The McBSP is set up as follows:

- Receive Control Register (RCR)
  - RPHASE = 0, indicating a single-phase frame
  - RFRLEN1 = 000 0001, indicating a two-element frame
  - RWDLEN1 = 100, indicating 24-bit elements
- Transmit Control Register (XCR)
  - XPHASE = 0, indicating a single-phase frame
  - XFRLEN1 = 000 0001, indicating a two-element frame
  - XWDLEN1 = 100, indicating 24-bit elements

To handle the same 48 bits of receive and transmit data now requires only two 24-bit reads of the DRR and two 24-bit writes to the DXR. Therefore, handling the same 48-bit data now requires only one-third the previous number of internal data transfers. This reduces the amount of bus time required for internal serial port data movement.

Figure 2 shows this data packing operation. You can use the (E)DMA to service the McBSP. Because the (E)DMA can only transfer 8-, 16-, or 32-bit elements, you must set the DMA element length to 32 bits to transfer the 24-bit data to/from the McBSP. This wastes 8 bits of memory space per word and creates gaps in the memory arrays, as shown in Figure 3. If you view the 48-bit data as three 16-bit elements instead of two 24-bit elements and set the (E)DMA to perform three 16-bit element transfers, no gaps are created in the memory, as shown in Figure 3. However, the trade-off is that the (E)DMA must perform three transfers (16-bit elements) instead of two transfers (24-bit elements). Figure 4 shows the timing diagram for transferring three 16-bit elements.

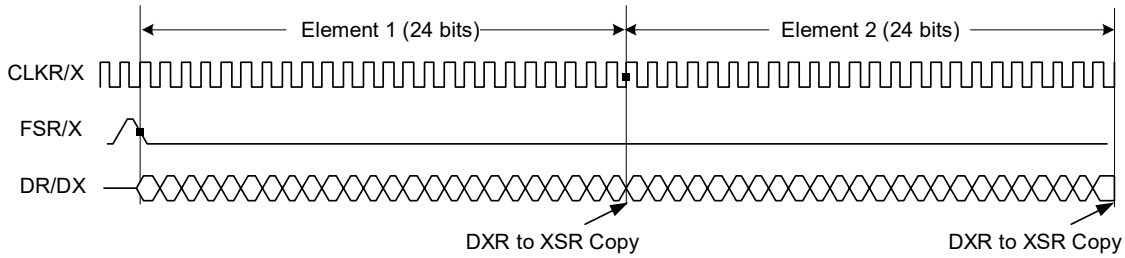


Figure 2. Timing Diagram for Data Transfer of Two 24-Bit Elements (With Data Packing)—Solution 1

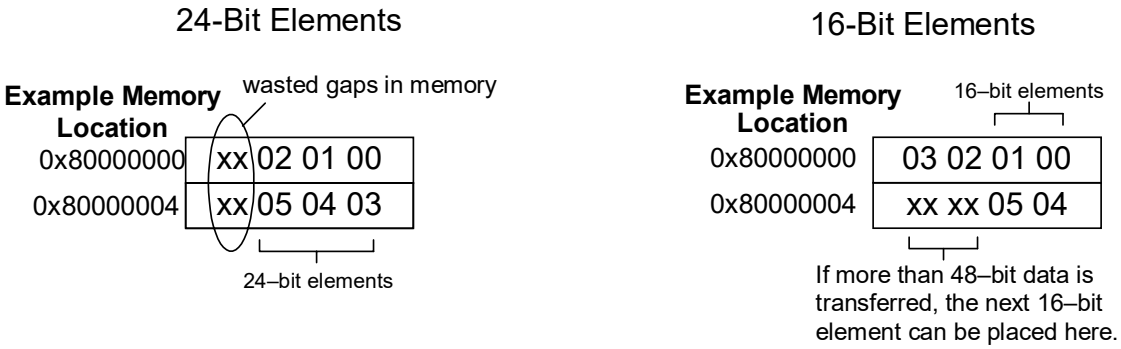


Figure 3. Transferred Data in Memory

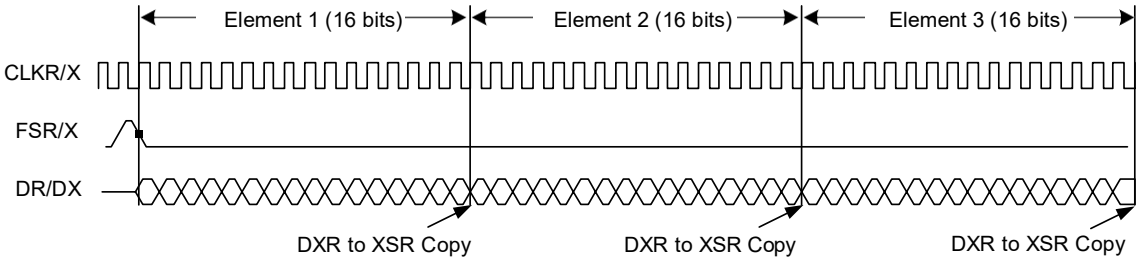


Figure 4. Timing Diagram for Data Transfer of Three 16-Bit Elements (With Data Packing)—Solution 1

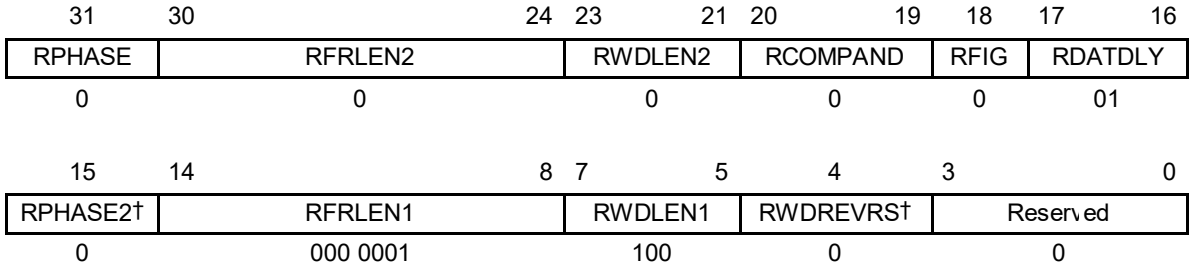
In Solution 1, the McBSP receive/transmit clocks (CLKR/CLKX) can be either internally or externally generated. Similarly, you can use either internal or external frame sync signals, FSR and FSX. The register setup discussed in section 3.2 and the sample code in Appendix A apply when the serial clock and frame sync signals are generated internally by the sample rate generator. By modifying the FSXM, FSRM, CLKXM, and CLKRM fields in the pin control register (PCR), you can alternatively configure the clock and frame sync signals to be inputs to the McBSP.

When an external device generates the frame sync signals, ensure that the frame sync signals occur only once per 48 bits of transferred data. Otherwise, you must refer to Solution 2 and set the receive/transmit frame ignore bits (RFIG/XFIG) in RCR/XCR to 1. This directs the McBSP to ignore unexpected receive/transmit frame sync pulses.

3.2 McBSP Registers Configuration

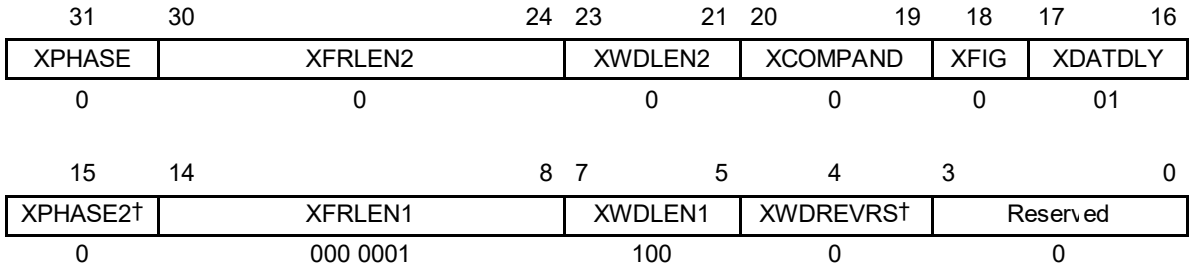
Figure 5, Figure 6, Figure 7, and Figure 8 show the bit field setup for the McBSP control registers for Solution 1. Table 1 lists and describes the bit fields. This solution assumes that the McBSP generates the frame sync and clock signals internally.

The bit fields and registers not listed in Table 1 assume default values. You are responsible to set some of the register fields, such as clock source, clock divide, and other parameters required by the application, if the initial state is different from the default.



† Available only on C621x/C671x and C64x devices.

Figure 5. Receive Control Register (RCR)—Solution 1



† Available only on C621x/C671x and C64x devices.

Figure 6. Transmit Control Register (XCR)—Solution 1

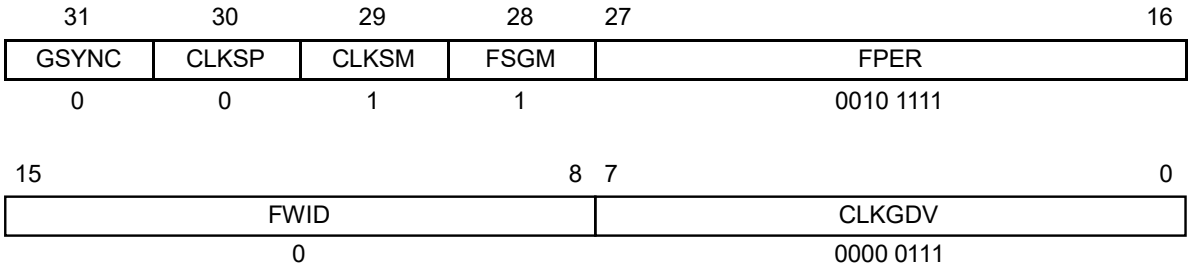


Figure 7. Sample Rate Generator Register (SRGR)—Solution 1

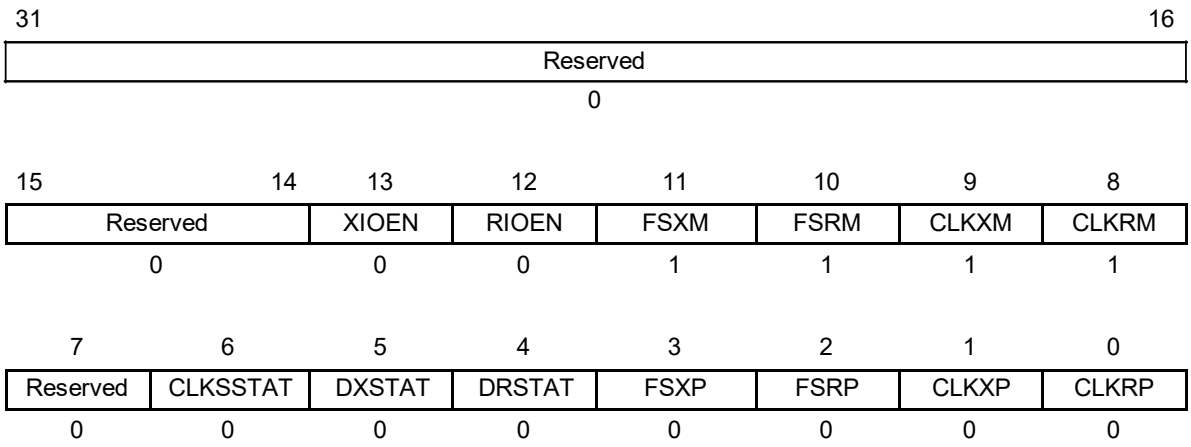


Figure 8. Pin Control Register (PCR)—Solution 1

Table 1. Bit-Field Values for McBSP Registers—Solution 1

Register	Bit Field			Function
	Bits	Name	Value (binary)	
RCR	17–16	RDATDLY	01	Receive data delay is 1 bit.
	14–8	RFRLEN1	000 0001	Receive frame length (number of elements) in phase 1 is 2 elements.
	7–5	RWDLEN1	100	Receive word length (number of bits) in phase 1 is 24 bits.
XCR	17–16	XDATDLY	01	Transmit data delay is 1 bit.
	14–8	XFRLEN1	000 0001	Transmit frame length (number of elements) in phase 1 is 2 elements.
	7–5	XWDLEN1	100	Transmit word length (number of bits) in phase 1 is 24 bits.
SRGR	29	CLKSM	1	Sample-rate generator clock is derived from CPU clock.
	28	FSGM	1	Transmit frame-sync signal (FSX) is driven by the sample-rate generator frame-sync signal (FSG).
	27–16	FPER	0010 1111	Frame period is 48 sample-rate generator clock (CLKG) periods (FPER + 1). The next frame-sync signal is active every 48 CLKG.
	7–0	CLKGDV	0000 0111	Sample-rate generator clock (CLKG) frequency is equal to 1/(CLKGDV + 1) of the internal clock source. The internal clock source is: □ CPU clock frequency (C620x/C670x) □ CPU clock frequency/2 (C621x/C671x) □ CPU clock frequency/4 (C64x) Actual CLKGDV used depends on applications and the frequency desired.

Table 1. Bit-Field Values for McBSP Registers—Solution 1 (Continued)

Register	Bit Field			Function
	Bits	Name	Value (binary)	
PCR	11	FSXM	1	Transmit frame-sync signal (FSX) is an output signal.
	10	FSRM	1	Frame-synchronization signal is generated internally by the sample-rate generator. FSR is an output signal.
	9	CLKXM	1	CLKX is an output pin and is driven by the internal sample-rate generator.
	8	CLKRM	1	CLKR is an output pin and is driven by the internal sample-rate generator.

4 Solution 2

Solution 2 shows how to pack data by setting the receive/transmit frame sync ignore (R/X)FIG bits of the McBSP.

4.1 Data Packing by Controlling (R/X)FIG

As shown in Figure 1, for a normal operation in which six 8-bit elements are transmitted to and from the McBSP, six reads of the DRR and six writes to the DXR, respectively, are required to handle the 48 bits of receive data and 48 bits of transmit data. Solution 1 presents an example of packing the transfer data by controlling the transfer frame and element length, provided that the serial data is being transferred at maximum packet frequency.

If the frame sync signal is generated by an external source, you can apply Solution 2 to pack the transfer data using the frame sync ignore (RFIG/XFIG) bits in the receive/transmit control registers (RCR/XCR). In this solution, an external serial device sends data in six 8-bit elements. In addition, this external device generates the frame sync signal. Solution 2 applies when either the McBSP or the external serial device generates the serial clocks, CLKR and CLKX. For data packing, the McBSP divides this 48-bit data into two 24-bit elements with the same data setup as Solution 1:

- Receive Control Register (RCR)
  - RPHASE = 0, indicating a single-phase frame
  - RFRLEN1 = 000 0001, indicating a two-element frame
  - RWDLEN1 = 100, indicating 24-bit elements
- Transmit Control Register (XCR)
  - XPHASE = 0, indicating a single-phase frame
  - XFRLEN1 = 000 0001, indicating a two-element frame
  - XWDLEN1 = 100, indicating 24-bit elements

In Solution 2, the external device sends one frame sync pulse for each 8-bit data element, as shown in Figure 9. However, to implement data packing, only one frame sync pulse is desired for every 24 bits of data. To ignore the extraneous frame syncs, the RFIG/XFIG bits should be set to 1. By setting the frame sync ignore bits, and the frame length and element length bits, only two reads of the DRR and two writes to the DXR are required to receive and transmit 48 bits of data. Figure 9 is the timing diagram for this operation.

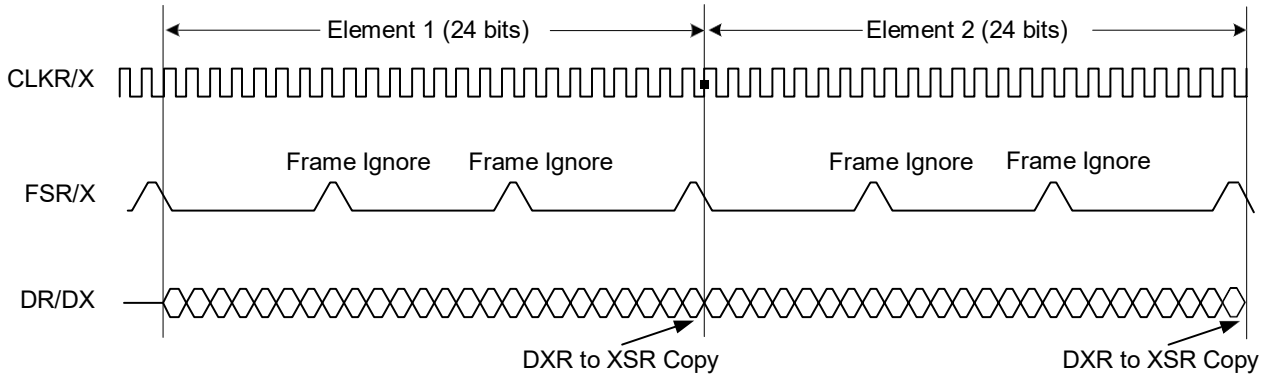


Figure 9. Timing Diagram for Data Packing With Frame Sync Ignore Operations—Solution 2

4.2 McBSP Registers Configuration

Figure 10, Figure 11, Figure 12, and Figure 13 show the bit field setup for the McBSP control registers for Solution 2. Table 2 lists and describes the bit fields. The RCR and XCR setup in this solution is similar to the setup in Solution 1, in addition to the frame sync ignore bits (RFIG and XFIG) being set.

The register setup in this solution shows the case in which the external device drives the serial clocks (CLKR and CLKX) and the frame sync signals (FSR and FSX). However, Solution 2 also applies if the McBSP generates the serial clocks and frame sync signals internally. If the frame sync signals are generated internally, setting the frame sync ignore bits (RFIG and XFIG) is optional because you should set the McBSP to generate the frame sync signals for only once every 24 bits of data.

In this solution with external clocks and frame sync signals, the sample rate generator register (SRGR) of the McBSP is configured with the default values. If you want the McBSP to generate the serial clocks instead, you need to set the SRGR as shown in Table 1. The following bits in the SRGR are don't cares, if the following external frame sync signals are used: GSYNC, FSGM, FPER, and FWID.

The bit fields and registers not listed in Table 2 assume default values. You are responsible to set some of the register fields, such as clock source, clock divide, and other parameters required by the application, if the initial state is different from the default.

31	30–24	23–21	20–19	18	17–16
RPHASE	RFRLEN2	RWDLEN2	RCOMPAND	RFIG	RDATDLY
0	0	0	0	1	01

15	14–8	7–5	4	3–0
RPHASE2†	RFRLEN1	RWDLEN1	RWDREVRS†	Reserved
0	000 0001	100	0	0

† Available only on C621x/C671x and C64x devices.

Figure 10. Receive Control Register (RCR)—Solution 2

31	30–24	23–21	20–19	18	17–16
XPHASE	XFRLEN2	XWDLEN2	XCOMPAND	XFIG	XDATDLY
0	0	0	0	1	01

15	14–8	7–5	4	3–0
XPHASE2†	XFRLEN1	XWDLEN1	XWDREVRS†	Reserved
0	000 0001	100	0	0

† Available only on C621x/C671x and C64x devices.

Figure 11. Transmit Control Register (XCR)—Solution 2

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

15–8	7–0
FWID	CLKGDV
0	0000 0111

Figure 12. Sample Rate Generator Register (SRGR)—Solution 2

31–16
Reserved
0

15–14	13	12	11	10	9	8
Reserved	XIOEN	RIOEN	FSXM	FSRM	CLKXM	CLKRM
0	0	0	0	0	0	0

7	6	5	4	3	2	1	0
Reserved	CLKSSTAT	DXSTAT	DRSTAT	FSXP	FSRP	CLKXP	CLKRP
0	0	0	0	0	0	0	0

Figure 13. Pin Control Register (PCR)—Solution 2

Table 2. Bit-Field Values for McBSP Registers—Solution 2

Register	Bit Field			Function
	Bits	Name	Value (binary)	
RCR	18	RFIG	1	Receive frame-sync pulses after the first pulse are ignored.
	17–16	RDATDLY	01	Receive data delay is 1 bit.
	14–8	RFRLEN1	000 0001	Receive frame length (number of elements) in phase 1 is 2 elements.
	7–5	RWDLEN1	100	Receive word length (number of bits) in phase 1 is 24 bits.
XCR	18	XFIG	1	Transmit frame-sync pulses after the first pulse are ignored.
	17–16	XDATDLY	01	Transmit data delay is 1 bit.
	14–8	XFRLEN1	000 0001	Transmit frame length (number of elements) in phase 1 is 2 elements.
	7–5	XWDLEN1	100	Transmit word length (number of bits) in phase 1 is 24 bits.
PCR	11	FSXM	0	Frame-sync signal is derived from an external source. Transmit frame-sync signal (FSX) is an input signal.
	10	FSRM	0	Frame-sync signal is derived from an external source. Receive frame-sync signal (FSR) is an input signal.
	9	CLKXM	0	CLKX is an input pin and is driven by an external clock.
	8	CLKRM	0	CLKR is an input pin and is driven by an external clock.

5 McBSP Initialization for Data Packing

The following example shows how to set up and initialize the McBSP to perform data packing. In this example, two multichannel buffered serial ports in one device are used. The first serial port, McBSP0, is used to transmit six 8-bit elements to the second serial port, McBSP1. McBSP1 is set up to use the method described in Solution 1 to pack data into two 24-bit elements. The (E)DMA in both devices service the corresponding McBSP by controlling internal data flow to and from the McBSP. The following steps describe the procedure necessary to initialize the (E)DMA, the McBSP, and the interrupts.

1. For McBSP0, program the SRGR, PCR, XCR, and RCR:

- SRGR: Default values (sample rate generator is not used)
- PCR: FSXM = FSRM = 0
- XCR: XWDLEN1 = 000 (8 bits)
  XFRLEN1 = 101 (6 elements)
- RCR: Default values (not used to receive)

2. For McBSP1, program the SRGR, PCR, XCR, and RCR to the values listed in Table 1.

Caution: Do not set the GRST bit in the serial port control register (SPCR) in the next step.

3. Take the sample rate generator of McBSP1 out of reset by setting GRST = 1 in the SPCR of McBSP1. The GRST bit in McBSP0 can remain at 0, because the sample rate generator of McBSP0 is not used.
4. Enabling interrupts. To use interrupts, you must set the global interrupt enable (GIE) in the control status register (CSR) and nonmaskable interrupt enable (NMIE) bits in the interrupt enable register (IER).

   For DMA (C620x/C670x)

   For the C62x™ and C67x™ DSPs, select the DMA channel(s) you want to use. Enable CPU interrupts that correspond to the DMA channel used to service the McBSP. The default mapping of DMA channel–complete interrupts to CPU is:

   - DMA channel 0 ⇒ CPU interrupt 8
   - DMA channel 1 ⇒ CPU interrupt 9
   - DMA channel 2 ⇒ CPU interrupt 11
   - DMA channel 3 ⇒ CPU interrupt 12

   For EDMA (C621x/C671x and C64x)

   Channels 12 and 15 were used to synchronize the EDMA transfers to the McBSP0 transmit and McBSP1 receive events, respectively. Unlike the C620x/C670x DMA controller which has individual interrupts for each DMA channel, the EDMA generates a single interrupt (EDMA_INT) to the CPU on behalf of all 16 channels (C621x and C671x DSPs) or 64 channels (C64x™ DSP).

   When the TCINT bit, in EDMA channel options register, is set to 1 for an EDMA channel and a specific transfer complete code (TCC) is provided, the EDMA controller sets a bit in the EDMA channel interrupt pending register (CIPR). The C64x DSP has two channel interrupt pending registers, channel interrupt pending low register (CIPRL) and channel interrupt pending high register (CIPRH), for the 64 channels.

   To configure the EDMA for any channel to interrupt the CPU:

   - In CIER, set CIEn to 1
   - In EDMA channel options register, set TCINT to 1
   - In EDMA channel options register, set TCC to n

   The transfer complete code is directly mapped to the CIPR bits in the C621x/C671x DSP. The transfer complete code is specified by the 4-bit TCC field.

   The transfer complete code is directly mapped to the CIPRL or CIPRH bits in the C64x DSP. The transfer complete code is expanded to a 6-bit value to accommodate the 64 channels. The transfer complete code is specified by the 2-bit TCCM field (MSBs) and the 4-bit TCC field.

   The CPU ISR should read the CIPR and determine what, if any, events/channels have completed and perform the operations necessary. The ISR should clear the bit in CIPR upon servicing the interrupt.

   The default mapping of EDMA channel–complete interrupts to CPU is:

   EDMA interrupt ⇒ CPU interrupt 8

C62x, C64x, and C67x are trademarks of Texas Instruments.

5. DMA initialization. Program the DMA channel for both transfers for the required operation. The following is a typical setup for the DMA channel corresponding to McBSP0:
    - Source address = internal memory or as required
    - Destination address = DXR
    - Transfer counter = number of elements to be transferred
    - In DMA channel primary control register (PRICTL):
        - DMA interrupt bit, TCINT = 1 (enabled)
        - Priority bit, PRI = 1 (DMA priority); optional, but recommended
        - Write sync event, WSYNC = 01100 (XEVT from McBSP)

    The following is a typical set up for the DMA channel corresponding to McBSP1:
    - Source address = DRR
    - Destination address = internal memory or as required
    - Transfer counter = number of elements to be transferred
    - In DMA channel primary control register (PRICTL):
        - DMA interrupt bit, TCINT = 1 (enabled)
        - Priority bit, PRI = 1 (DMA priority); optional, but recommended
        - Read sync event, RSYNC = 01111 (REVT from McBSP)

    EDMA initialization: Program the EDMA channel for both transfers for the required operation. The following is a typical setup for the EDMA channel corresponding to McBSP0:
    - Source address = internal memory or as required
    - Destination address = DXR
    - Transfer counter = number of elements to be transferred
    - In DMA channel primary control register (PRICTL):
        - Write sync event, WSYNC = 01100 (EDMA channel synchronized to McBSP0 transmit event, XEVT0)
    - In EDMA channel options register:
        - Priority, PRI = 001 (High); optional, but recommended
        - EDMA interrupt bit, TCINT = 1 (enabled)
        - Transfer complete code, TCC = 1100
        - 1D transfer, FS = 0

    The following is a typical set up for the EDMA channel corresponding to McBSP1:
    - Source address = DRR
    - Destination address = internal memory or as required
    - Transfer counter = number of elements to be transferred
    - In DMA channel primary control register (PRICTL):
        - Read sync event, RSYNC = 01111 (EDMA channel synchronized to McBSP1 receive event, REVT1)

- In EDMA channel options register:
  - Priority, PRI = 001 (High); optional, but recommended
  - EDMA interrupt bit, TCINT = 1 (enabled)
  - Transfer complete code, TCC = 1111
  - 1D transfer, FS = 0

6. Instruct the (E)DMA channel(s) in both devices to run. In the DMA channel primary control register (PRICTL), set START = 01 to start the DMA without autoinitialization. For the EDMA, set the corresponding bit in the EDMA event enable register. The (E)DMA starts the first transfer on receiving the first read/write sync event.
7. In the serial port control register (SPCR), set XRST=1 to wake up McBSP0. Note that McBSP0 must wake up before McBSP1 because McBSP0 must be ready to transmit as soon as it receives the frame sync signal from McBSP1.
8. In SPCR, set RRST = XRST = 1 to wake up McBSP1.
9. In SPCR, set FRST = 1 to start the frame sync generator in McBSP1. The first frame sync signal (FSX) is generated by McBSP1 after 8 CLKG clocks. This FSX signal from McBSP1 is captured by McBSP0 on the falling edge of McBSP1 internal signal CLKG. Data transfer between the two devices begins.

6 Sample C Functions

Appendix A contains sample C codes that perform data packing by applying Solution 1 from this application report. The C codes are tested on a board with a hardware setup shown in Figure 14. This example uses a single TMS320C6000 DSP. McBSP1 operates as the frame and clock master. McBSP0 operates as the external serial device mentioned in Solution 1.

The C code in Appendix A sets up both the external serial device (McBSP0) to transmit two frames of six 8-bit elements and sets up the frame and clock master (McBSP1) to pack the data from McBSP0 into two frames of two 24-bit elements. (See the *TMS320C6000 Chip Support Library API User's Guide* for a detailed description of the header files used in the C code.)

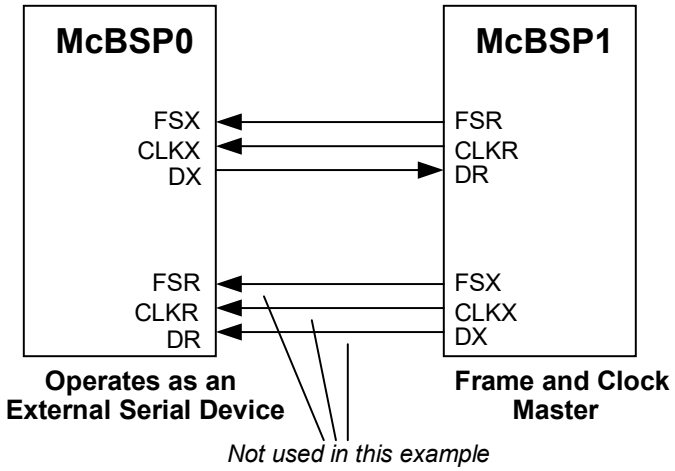


Figure 14. Data Packing Hardware Interface Example

7 Conclusion

Two different solutions are available to implement data packing using the TMS320C6000 McBSP. Solution 1 applies when either the McBSP or an external device generates the frame sync signals. Solution 2 applies only when an external source generates the frame sync signals. Both methods are equally effective in reducing the bus bandwidth for serial transfers. The programmable features of the McBSP, such as frame length, element length, and frame sync ignore help accomplish data packing.

8 References

1. *TMS320C6000 Peripherals Reference Guide*, literature number SPRU190, Texas Instruments
2. *TMS320C6000 Chip Support Library API User's Guide*, literature number SPRU401, Texas Instruments

Appendix A Data Packing Sample Source Code

```
TI Proprietary Information
Internal Data

Written by Rebecca Ma
3/9/98
Updated by Ivan Garcia
7/13/01

datapack.c:
This code sets up McBSP0 to transmit six 8-bit
elements to McBSP1.  DMA Channel 1 services McBSP0.
The sample rate generator for McBSP0 is not used.  FSX and CLKX
are both input pins, driven by McBSP1.

This code also sets up McBSP1 to pack received data (from McBSP0)
into two 24-bit data elements.  DMA Channel 2 services
McBSP1.  The sample rate generator for McBSP1 is used.  FSR and CLKR
are both output pins that drive the FSX and CLKX pins of McBSP0, respectively.

Datapacking only works if data is transferred at maximum packetfrequency.

NOTE: Since DMA can only transfer 8, 16, or 32 bits, we set the DMA transfer
element size to 32-bits, even though the data elements from DRR in McBSP1 are
24-bits long. We leave the RJUST bit in SPCR of McBSP1 to be 0 (default),
so that McBSP1 will right-justify and zero-fill MSBs in DRR.


*/

#define CHIP_6711    /* choose chip                                          */

#include <c6x.h>
#include <csl.h>
#include <csl_dma.h>
#include <csl_edma.h>
#include <csl_irq.h>
#include <csl_mcbsp.h>

/*************** DSP0/McBSP0 constants ***************/
#define XFER_ELEMENT_CNT0  6  /* number of 8-b elements transferred per frame */
#define XFER_FRAME_CNT0    2  /* total number of frames transferred           */
#define DMA_XFER_SIZE0     6  /* number of elements that DMA needs to transfer */
#define DMA_XFER_FRAME0    2  /* number of frames that DMA needs to transfer  */
#define XDATA  0x8000         /* location of data to be transmitted           */

/*************** DSP1/McBSP1 constants ***************/
#define X_ELEMENT_CNT1     2  /* number of 24-b elements transferred per frame */
#define X_FRAME_CNT1       2  /* total number of frames transferred           */
#define DMA_XFER_SIZE1     2  /* number of elements per frame that DMA needs  */
                              /* transfer                                     */
#define DMA_XFER_FRAME1    2  /* number of frames for DMA transfer            */
#define CLK_DIV            7  /* CLKG freq = 1/(7+1) = 1/8 the freq of        */
                              /* internal clk source. Internal clk source     */
                              /* changes according to device                  */
#define FR_PERIOD         47 /* frame period = 47+1 = 48 CLKG                */
#define RDATA  0x8100         /* location to put received data                */

MCBSP_Handle hMcbsp0;
MCBSP_Handle hMcbsp1;

/******** functions for McBSP0 ********/
void init_data(void);
void init_mcbsp0(void);
void set_interrupts(void);
void run_dma0(void);
void wake_mcbsp0(void);
```

```
/******** functions for McBSP1 ********/
void init_mcbsp1(void);
void run_dma1(void);
void wake_mcbsp1(void);

/* Inlcude the vector table to call the IRQ ISRs hookup                    */
extern far void vectors();

int   dma_done0 = FALSE;
int   dma_done1 = FALSE;

#if (EDMA_SUPPORT)
EDMA_Handle hEdma1;
EDMA_Handle hEdma2;
EDMA_Handle hEdma_dummy;
#endif

#if (DMA_SUPPORT)
DMA_Handle hDma1;
DMA_Handle hDma2;
#endif

void
main(void)
{
   CSL_init();          /* initialize the CSL library                      */
   init_data();
   init_mcbsp0();
   init_mcbsp1();

      #if (DMA_SUPPORT)
      DMA_reset(INV);   /* reset all DMA channels to power-on defaults     */
      hDma1 = DMA_open(DMA_CHA1, DMA_OPEN_RESET);
      hDma2 = DMA_open(DMA_CHA2, DMA_OPEN_RESET);
      #endif


   set_interrupts();    /* 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.                                           */

   run_dma0();
   run_dma1();

      #if (EDMA_SUPPORT)
      hEdma_dummy = EDMA_allocTable(-1);        /* Dynamically allocates   */
                                                /* PaRAM RAM table         */
         EDMA_configArgs(hEdma_dummy,           /* Dummy or Terminating Table */
            0x00000000,                         /* in PaRAM                */
            0x00000000,                         /* Terminate EDMA transfers by */
            0x00000000,                         /* linking to this NULL table */
            0x00000000,
            0x00000000,
            0x00000000
            );
         EDMA_link(hEdma1, hEdma_dummy);
         EDMA_link(hEdma2, hEdma_dummy);

      EDMA_enableChannel(hEdma1);
      EDMA_enableChannel(hEdma2);
      #endif

   wake_mcbsp0();
   wake_mcbsp1();

   while (!(dma_done0 & dma_done1));
```

```
   MCBSP_close(hMcbsp0);
   MCBSP_close(hMcbsp1);

      #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(hEdma_dummy);
      #endif
}  /* end main */
/* Initialize data to be transferred by (E)DMA Ch0 from memory to McBSP0     */
void
init_data(void)
{
   unsigned int i;
   int *xdata = (int *)XDATA;
      /* total elements for transfer */
   for (i=0; i < XFER_ELEMENT_CNT0*XFER_FRAME_CNT0; i++)
   {
      *xdata++ = i+1;
   }  /* end for*/
} /* End init_data */
void
init_mcbsp0(void)
{
MCBSP_Config mcbspCfg0 = {
      MCBSP_SPCR_DEFAULT,
      MCBSP_RCR_DEFAULT,
   #if (EDMA_SUPPORT)
      MCBSP_XCR_RMK(
         MCBSP_XCR_XPHASE_SINGLE,
         MCBSP_XCR_XFRLEN2_DEFAULT,
         MCBSP_XCR_XWDLEN2_DEFAULT,
         MCBSP_XCR_XCOMPAND_DEFAULT,
         MCBSP_XCR_XFIG_DEFAULT,
         MCBSP_XCR_XDATDLY_1BIT,
         MCBSP_XCR_XFRLEN1_OF(XFER_ELEMENT_CNT0-1),
         MCBSP_XCR_XWDLEN1_8BIT,
         MCBSP_XCR_XWDREVRS_DEFAULT
         ),
   #endif
   #if (DMA_SUPPORT)
      MCBSP_XCR_RMK(
         MCBSP_XCR_XPHASE_SINGLE,
         MCBSP_XCR_XFRLEN2_DEFAULT,
         MCBSP_XCR_XWDLEN2_DEFAULT,
         MCBSP_XCR_XCOMPAND_DEFAULT,
         MCBSP_XCR_XFIG_DEFAULT,
         MCBSP_XCR_XDATDLY_1BIT,
         MCBSP_XCR_XFRLEN1_OF(XFER_ELEMENT_CNT0-1),
         MCBSP_XCR_XWDLEN1_8BIT
         ),
   #endif
      MCBSP_SRGR_DEFAULT,              /* SRGR left at default value since    */
                                       /* McBSP0 sample rate generator not    */
                                       /* used                                */
      MCBSP_MCR_DEFAULT,
```

```
   #if(!C64_SUPPORT)
      MCBSP_RCER_RMK(
         MCBSP_RCER_RCEB_DEFAULT,      /* All fields in RCER set to default   */
         MCBSP_RCER_RCEA_DEFAULT
         ),
   #endif

   #if(!C64_SUPPORT)
      MCBSP_XCER_RMK(
         MCBSP_XCER_XCEB_DEFAULT,      /* All fields in XCER set to default   */
         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

   /* setup PCR */
      MCBSP_PCR_RMK(
         MCBSP_PCR_XIOEN_DEFAULT,
         MCBSP_PCR_RIOEN_DEFAULT,
         MCBSP_PCR_FSXM_EXTERNAL,
         MCBSP_PCR_FSRM_DEFAULT,
         MCBSP_PCR_CLKXM_INPUT,
         MCBSP_PCR_CLKRM_DEFAULT,
         MCBSP_PCR_CLKSSTAT_DEFAULT,
         MCBSP_PCR_DXSTAT_DEFAULT,
         MCBSP_PCR_FSXP_ACTIVELOW,
         MCBSP_PCR_FSRP_DEFAULT,
         MCBSP_PCR_CLKXP_DEFAULT,
         MCBSP_PCR_CLKRP_DEFAULT
         )
   };

   hMcbsp0 = MCBSP_open(MCBSP_DEV0, MCBSP_OPEN_RESET);
   MCBSP_config(hMcbsp0, &mcbspCfg0);

   /* enable sample rate generator */
   /* don't need to do this because CLKX is generated by McBSP1 */
}  /* end init_mcbsp0 */
```

```
void
init_mcbsp1(void)
{
   MCBSP_Config mcbspCfg1 = {
      MCBSP_SPCR_DEFAULT,
   #if (EDMA_SUPPORT)
      MCBSP_RCR_RMK(
         MCBSP_RCR_RPHASE_SINGLE,
         MCBSP_RCR_RFRLEN2_DEFAULT,
         MCBSP_RCR_RWDLEN2_DEFAULT,
         MCBSP_RCR_RCOMPAND_DEFAULT,
         MCBSP_RCR_RFIG_DEFAULT,
         MCBSP_RCR_RDATDLY_1BIT,
         MCBSP_RCR_RFRLEN1_OF(X_ELEMENT_CNT1-1),
         MCBSP_RCR_RWDLEN1_24BIT,
         MCBSP_RCR_RWDREVRS_DEFAULT
         ),
   #endif
   #if (DMA_SUPPORT)
      MCBSP_RCR_RMK(
         MCBSP_RCR_RPHASE_SINGLE,
         MCBSP_RCR_RFRLEN2_DEFAULT,
         MCBSP_RCR_RWDLEN2_DEFAULT,
         MCBSP_RCR_RCOMPAND_DEFAULT,
         MCBSP_RCR_RFIG_DEFAULT,
         MCBSP_RCR_RDATDLY_1BIT,
         MCBSP_RCR_RFRLEN1_OF(X_ELEMENT_CNT1-1),
         MCBSP_RCR_RWDLEN1_24BIT
         ),
   #endif
   MCBSP_XCR_DEFAULT,
      MCBSP_SRGR_RMK(
         MCBSP_SRGR_GSYNC_FREE,
         MCBSP_SRGR_CLKSP_DEFAULT,
         MCBSP_SRGR_CLKSM_INTERNAL,
         MCBSP_SRGR_FSGM_DEFAULT,
         MCBSP_SRGR_FPER_OF(FR_PERIOD),
         MCBSP_SRGR_FWID_DEFAULT,
         MCBSP_SRGR_CLKGDV_OF(CLK_DIV)
         ),
   MCBSP_MCR_DEFAULT,
   #if(!C64_SUPPORT)
      MCBSP_RCER_RMK(
         MCBSP_RCER_RCEB_DEFAULT,         /* All fields in RCER set to default */
         MCBSP_RCER_RCEA_DEFAULT
         ),
   #endif
   #if(!C64_SUPPORT)
      MCBSP_XCER_RMK(
         MCBSP_XCER_XCEB_DEFAULT,         /* All fields in XCER set to default */
         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
   /* setup PCR */
      MCBSP_PCR_RMK(
         MCBSP_PCR_XIOEN_DEFAULT,
         MCBSP_PCR_RIOEN_DEFAULT,
         MCBSP_PCR_FSXM_DEFAULT,
         MCBSP_PCR_FSRM_INTERNAL,
         MCBSP_PCR_CLKXM_DEFAULT,
         MCBSP_PCR_CLKRM_OUTPUT,
         MCBSP_PCR_CLKSSTAT_DEFAULT,
         MCBSP_PCR_DXSTAT_DEFAULT,
         MCBSP_PCR_FSXP_DEFAULT,
         MCBSP_PCR_FSRP_ACTIVELOW,
         MCBSP_PCR_CLKXP_DEFAULT,
         MCBSP_PCR_CLKRP_DEFAULT
         )
   };
   hMcbsp1 = MCBSP_open(MCBSP_DEV1, MCBSP_OPEN_RESET);
   MCBSP_config(hMcbsp1, &mcbspCfg1);
   /* Enable sample rate generator GRST=1                                    */
   MCBSP_enableSrgr(hMcbsp1);               /* Handle to SRGR               */
} /* end init_mcbsp */
/* Set up interrupts, such that DMA Channel 1 interrupt will cause          */
/* c_int09 to execute and DMA Channel 2 interrupt will cause                */
/* c_int11 to execute.                                                      */
#if (DMA_SUPPORT)
void
set_interrupts(void)
{
   IRQ_setVecs(vectors);              /* point to the IRQ vector table       */
      IRQ_nmiEnable();                /* enable NMIE                         */
      IRQ_globalEnable();             /* set GIE in CSR                      */
      IRQ_map(IRQ_EVT_DMAINT1, 9);
      IRQ_map(IRQ_EVT_DMAINT2, 11);
      IRQ_reset(IRQ_EVT_DMAINT1);     /* disable and clear                   */
      IRQ_reset(IRQ_EVT_DMAINT2);
      IRQ_enable(IRQ_EVT_DMAINT1);    /* enable DMA ch0 interrupt            */
      IRQ_enable(IRQ_EVT_DMAINT2);    /* enable DMA ch1 interrupt            */
} /* End set_interrupts */
#endif
```

```
/* Set up interrupt, such that EDMA interrupt will cause c_int08 to execute   */
#if (EDMA_SUPPORT)
void
set_interrupts(void)
{
   IRQ_setVecs(vectors);               /* point to the IRQ vector table       */

      IRQ_nmiEnable();
      IRQ_globalEnable();
      IRQ_map(IRQ_EVT_EDMAINT, 8);
      IRQ_reset(IRQ_EVT_EDMAINT);
      IRQ_disable(IRQ_EVT_EDMAINT);
      EDMA_intDisable(12);             /* ch 12 for McBSP transmit event XEVT0 */
      EDMA_intDisable(15);             /* ch 15 for McBSP receive event REVT1 */
      IRQ_clear(IRQ_EVT_EDMAINT);
      EDMA_intClear(12);
      EDMA_intClear(15);
      IRQ_enable(IRQ_EVT_EDMAINT);
      EDMA_intEnable(12);              /* enable a x-fer completion interrupt  */
      EDMA_intEnable(15);              /* by modifying the CIER register       */

} /* End set_interrupts */
#endif

/* DMA Channel Interrupt Service Routines will execute upon                   */
/* completion of a Block Transfer by a channel.                               */

interrupt void       /* vecs.asm hooks this up to IRQ 11                      */
c_int11(void)        /* DMA ch2                                               */
{
#if (DMA_SUPPORT)
dma_done1 = TRUE;    /* finished receiving?                                   */
return;
#endif
}

interrupt void       /* vecs.asm hooks this up to IRQ 09                      */
c_int09(void)        /* DMA ch1                                               */
{
#if (DMA_SUPPORT)
dma_done0 = TRUE;    /* finished transmitting?                                */
return;
#endif
}

interrupt void       /* vecs.asm hooks this up to IRQ 08                      */
c_int08(void)        /* for the EDMA                                          */
{
   #if (EDMA_SUPPORT)
   if (EDMA_intTest(12))
   {
   dma_done0 = TRUE;
   EDMA_intClear(12); /* clear CIPR bit so future interrupts can be recognized   */
   }
   else if (EDMA_intTest(15))
   {
   dma_done1 = TRUE;
   EDMA_intClear(15); /* clear CIPR bit so future interrupts can be recognized   */
   }
#endif
return;
}
```

```
/* Set up the DMA Control Registers to perform the data transfers            */
/* from XDATA to McBSP0. Channel 1 is used.                                  */
#if (DMA_SUPPORT)
void
run_dma0(void)
{


DMA_RSET(GBLCNTA,             /* count reload occurs at the end of each frame */
      DMA_GBLCNT_RMK(DMA_GBLCNT_FRMCNT_OF(0),
      DMA_GBLCNT_ELECNT_OF(6))
      );
   DMA_configArgs(hDma1,
      DMA_PRICTL_RMK(               /* initialize primary control register    */
         DMA_PRICTL_DSTRLD_NONE,
         DMA_PRICTL_SRCRLD_NONE,
         DMA_PRICTL_EMOD_HALT,
         DMA_PRICTL_FS_DISABLE,     /* need to disable frame sync             */
         DMA_PRICTL_TCINT_ENABLE,   /* enable interrupt                       */
         DMA_PRICTL_PRI_DMA,
         DMA_PRICTL_WSYNC_XEVT0,
         DMA_PRICTL_RSYNC_NONE,
         DMA_PRICTL_INDEX_A,
         DMA_PRICTL_CNTRLD_A,
         DMA_PRICTL_SPLIT_DISABLE,
         DMA_PRICTL_ESIZE_32BIT,
         DMA_PRICTL_DSTDIR_NONE,
         DMA_PRICTL_SRCDIR_INC,
         DMA_PRICTL_START_STOP
         ),
      DMA_SECCTL_RMK(            /* initialize DMA0 secondary control register  */
         DMA_SECCTL_WSPOL_DEFAULT,  /* only available for 6202/6203           */
         DMA_SECCTL_RSPOL_DEFAULT,  /* only available for 6202/6203           */
         DMA_SECCTL_FSIG_DEFAULT,   /* only available for 6202/6203           */
         DMA_SECCTL_DMACEN_BLOCKCOND,
         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,
         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(XDATA),
      DMA_DST_RMK(MCBSP_getXmtAddr(hMcbsp0)),
      DMA_XFRCNT_RMK(               /* init DMA1 transfer counter register    */
         DMA_XFRCNT_FRMCNT_OF(DMA_XFER_FRAME0),
         DMA_XFRCNT_ELECNT_OF(DMA_XFER_SIZE0)
         )
      );
   DMA_start(hDma1);                /* start DMA Channel 1                    */
}  /* end run_dma */
#endif
```

```
#if (EDMA_SUPPORT)
void
run_dma0(void)
{
/* channel tied to McBSP0 xmit */
hEdma1 = EDMA_open(EDMA_CHA_XEVT0, EDMA_OPEN_RESET);
EDMA_configArgs(hEdma1,
#if (!C64_SUPPORT)
   EDMA_OPT_RMK(
      EDMA_OPT_PRI_HIGH,
      EDMA_OPT_ESIZE_32BIT,         /* Element size 32 bits                  */
      EDMA_OPT_2DS_NO,
      EDMA_OPT_SUM_INC,
      EDMA_OPT_2DD_NO,
      EDMA_OPT_DUM_NONE,
      EDMA_OPT_TCINT_YES,           /* Enable Transfer Complete Interrupt    */
      EDMA_OPT_TCC_OF(12),
      EDMA_OPT_LINK_YES,            /* Enable linking to NULL table          */
      EDMA_OPT_FS_NO
      ),
#endif
#if (C64_SUPPORT)
   EDMA_OPT_RMK(
      EDMA_OPT_PRI_DEFAULT,
      EDMA_OPT_ESIZE_32BIT,         /* Element size 32 bits                  */
      EDMA_OPT_2DS_NO,
      EDMA_OPT_SUM_INC,
      EDMA_OPT_2DD_NO,
      EDMA_OPT_DUM_NONE,
      EDMA_OPT_TCINT_YES,           /* Enable Transfer Complete Interrupt    */
      EDMA_OPT_TCC_OF(12)
      EDMA_OPT_LINK_YES,            /* Enable linking to NULL table          */
      EDMA_OPT_FS_NO,
      EDMA_OPT_TCCM_DEFAULT,        /* TM = 00                               */
      EDMA_OPT_ATCINT_DEFAULT,
      EDMA_OPT_ATCC_DEFAULT,
      EDMA_OPT_PDTS_DEFAULT,
      EDMA_OPT_PDTD_DEFAULT
      ),
#endif


      EDMA_SRC_RMK(XDATA),
      EDMA_CNT_RMK(DMA_XFER_FRAME0-1, DMA_XFER_SIZE0),   /* no. of elements   */
      EDMA_DST_RMK(MCBSP_getXmtAddr(hMcbsp0)),
      EDMA_IDX_RMK(0,EDMA_IDX_FRMIDX_OF(0)),
      EDMA_RLD_RMK(EDMA_RLD_ELERLD_OF(6),0)
      );
   EDMA_enableChannel(hEdma1);
}  /* end run_dma0 */
#endif
```

```
/* Set up the DMA Control Registers to perform the data transfers            */
/* from McBSP1 to RDATA.  Channel 2 is used.                                 */
#if (DMA_SUPPORT)
void
run_dma1(void)
{

DMA_RSET(GBLCNTB,             /* channel tied to McBSP0 xmit                 */
      DMA_GBLCNT_RMK(DMA_GBLCNT_FRMCNT_OF(0),
      DMA_GBLCNT_ELECNT_OF(2))
      );
   DMA_configArgs(hDma2,
      DMA_PRICTL_RMK(               /* Init DMA1 primary control register    */
         DMA_PRICTL_DSTRLD_NONE,
         DMA_PRICTL_SRCRLD_NONE,
         DMA_PRICTL_EMOD_HALT,
         DMA_PRICTL_FS_DISABLE,     /* Need to disable frame sync            */
         DMA_PRICTL_TCINT_ENABLE,   /* enable interrupt                      */
         DMA_PRICTL_PRI_DMA,
         DMA_PRICTL_WSYNC_NONE,
         DMA_PRICTL_RSYNC_REVT1,
         DMA_PRICTL_INDEX_A,
         DMA_PRICTL_CNTRLD_B,
         DMA_PRICTL_SPLIT_DISABLE,
         DMA_PRICTL_ESIZE_32BIT,
         DMA_PRICTL_DSTDIR_INC,
         DMA_PRICTL_SRCDIR_NONE,
         DMA_PRICTL_START_STOP
         ),
      DMA_SECCTL_RMK(               /* Init DMA1 secondary control register */
         DMA_SECCTL_WSPOL_DEFAULT,  /* only available for 6202/6203          */
         DMA_SECCTL_RSPOL_DEFAULT,  /* only available for 6202/6203          */
         DMA_SECCTL_FSIG_DEFAULT,   /* only available for 6202/6203          */
         DMA_SECCTL_DMACEN_BLOCKCOND,
         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,
         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_getRcvAddr(hMcbsp1)),
      DMA_DST_RMK(RDATA),
      DMA_XFRCNT_RMK(               /* Init DMA1 transfer counter register   */
         DMA_XFRCNT_FRMCNT_OF(DMA_XFER_FRAME1),
         DMA_XFRCNT_ELECNT_OF(DMA_XFER_SIZE1)
         )
      );
   DMA_start(hDma2);
}  /* end run_dma1 */
#endif
```

```
#if (EDMA_SUPPORT)
void
run_dma1(void)
{
/* channel tied to McBSP1 rcv */
hEdma2 = EDMA_open(EDMA_CHA_REVT1, EDMA_OPEN_RESET);
EDMA_configArgs(hEdma2,
#if (!C64_SUPPORT)
   EDMA_OPT_RMK(
      EDMA_OPT_PRI_HIGH,
      EDMA_OPT_ESIZE_32BIT,         /* Element size 32 bits                  */
      EDMA_OPT_2DS_DEFAULT,
      EDMA_OPT_SUM_NONE,
      EDMA_OPT_2DD_NO,
      EDMA_OPT_DUM_INC,
      EDMA_OPT_TCINT_YES,           /* Enable Transfer Complete Interrupt    */
      EDMA_OPT_TCC_OF(15),
      EDMA_OPT_LINK_YES,            /* Enable linking to NULL table          */
      EDMA_OPT_FS_NO
      ),
#endif
#if (C64_SUPPORT)
   EDMA_OPT_RMK(
      EDMA_OPT_PRI_DEFAULT,
      EDMA_OPT_ESIZE_32BIT,
      EDMA_OPT_2DS_NO,
      EDMA_OPT_SUM_NONE,
      EDMA_OPT_2DD_NO,
      EDMA_OPT_DUM_INC,
      EDMA_OPT_TCINT_YES,           /* Enable Transfer Complete Interrupt    */
      EDMA_OPT_TCC_OF(15),
      EDMA_OPT_LINK_YES,            /* Enable linking to NULL table          */
      EDMA_OPT_FS_NO,
      EDMA_OPT_TCCM_DEFAULT,        /* TM = 00                               */
      EDMA_OPT_ATCINT_DEFAULT,
      EDMA_OPT_ATCC_DEFAULT,
      EDMA_OPT_PDTS_DEFAULT,
      EDMA_OPT_PDTD_DEFAULT
      ),
#endif
      EDMA_SRC_RMK(MCBSP_getRcvAddr(hMcbsp1)),
      EDMA_CNT_RMK(DMA_XFER_FRAME1-1, DMA_XFER_SIZE1), /* no. of elements    */
      EDMA_DST_RMK(RDATA),
      EDMA_IDX_RMK(0,0),
      EDMA_RLD_RMK(EDMA_RLD_ELERLD_OF(2),0)
      );
// EDMA_enableChannel(hEdma2);
}  /* end run_dma1 */
#endif
/* wake up mcbsp0 transmitter.  wait for frame sync from McBSP1             */
void
wake_mcbsp0(void)
{
   MCBSP_enableXmt(hMcbsp0);
} /* end wake_mcbsp0 */
/* wake up mcbsp1 receiver                                                  */
void
wake_mcbsp1(void)
{
   MCBSP_enableRcv(hMcbsp1);
   MCBSP_enableFsync(hMcbsp1);
} /* end wake_mcbsp1 */
```

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