Application Report
SPRA491A - January 2001

TMS320C6000 McBSP as a TDM Highway

Shaku Anjanaiah
Scott Tater

Digital Signaling Processing Solutions

ABSTRACT

This document describes how the multichannel buffered serial ports (McBSP) in the TMS320C6000™ digital signal processors (DSP) are used to communicate on a time-division multiplexed (TDM) data highway.

TDM provides multiple devices a time slot to perform data transfer. Thus, multiple users operate various channels; however, each user has a set of channel(s) assigned for transmission and reception. The McBSP can support up to 128 channels in multichannel mode. Each of these 128 channels can be enabled or disabled via software to communicate only in those necessary time slots.

When multiple users are connected to the same data lines, bus contention can be an issue during transmission. Although each device connected to this TDM highway has its own channel assignment, it is possible that the next device getting onto the bus can contend with the last bit(s) of the previous channel. The workaround for the data contention problem is addressed in this document by considering multiple McBSPs connected to a single data line. This document presents three workarounds applicable to the different C6000™ devices.

Contents

List of Figures

List of Tables

1 Design Problem

How do I use the multichannel buffered serial port (McBSP) to communicate over a time-division multiplexed (TDM) data highway without bus contention?

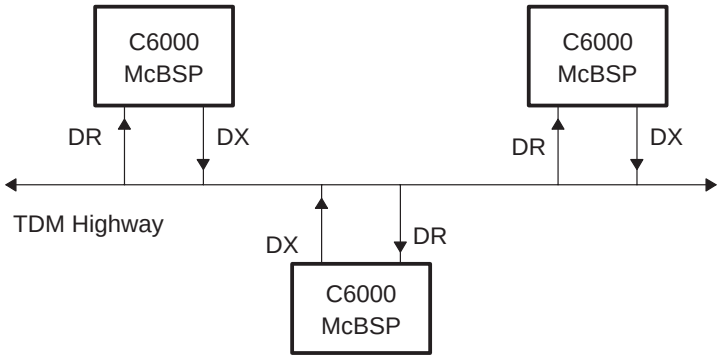

Figure 1. Multiple McBSPs on a Time-Division Multiplexed Bus

2 Application

Multiple DSPs can be connected via a single TDM bus to communicate specific information to a specific DSP or a group of DSPs. Typical applications would be messaging, broadcasting and passing channels of processed data to required DSPs for further processing or as data input. The TDM data transfer is achieved using the programmable McBSP and the DMA. The serial port pins DX and DR are used for data transfer, whereas CLK(R/X) and FS(R/X) serve as the control signals for clocking and synchronization. The DMA is responsible for retrieving data from the McBSP receiver and storing in an *in_buffer*. The CPU can then process the *in_buffer* data as required and make it available in the *out_buffer* for the DMA to write to the TDM bus via the transmitter.

It is important to configure each DSP to send and receive channels of data on certain assigned time slots. This TDM arrangement ensures proper allocation of data on particular channels and retrieval of this data by the required DSP. Since there are no address lines used, the multichannel operation of the McBSP helps in identifying the destination of data in transit. DSP-specific messages are sent on a known channel, and the destination DSP listens to this channel by enabling the appropriate receive channel enable bit(s).

3 Multichannel Operation Overview

The McBSP can perform multichannel selection operation for single-phase frames. Single-phase frames are characterized by a group of elements that have the same element size. The maximum number of elements per frame is the same as the number of channels, which is equal to 128. Therefore, each frame represents a time-division multiplexed data stream with up to 128 channels.

All C6000 devices are capable of implementing a TDM highway using the multichannel serial port and have the potential to transmit or receive from one up to 128 channels. The C64x™ McBSP supports an enhanced multichannel mode that allows for greater channel selection flexibility and requires fewer resources to transmit and receive over a large number of channels.

3.1 Multichannel Selection Mode (All C6000 devices)

For all C6000 devices, a set of programmable control registers in the McBSP specific for multichannel operation makes communication on a TDM highway possible. They are the Multichannel Control Register (MCR), the Transmit Channel Enable Register (XCER), and the Receive Channel Enable Register (RCER). These registers are explained in detail in the *TMS320C6000 Peripherals Reference Guide* (SPRU190).

The 128 channels in a frame are divided into eight 16-channel sub-frames as shown Figure 2. Each channel can have programmable data sizes (8-, 12-, 16-, 20-, or 32-bits). But all channels in a frame have to be of same data size. Odd-numbered sub-frames constitute Partition A (represented by (R/X)PABLK), and even-numbered belong to Partition B (represented by (R/X)PBBLK). Each of the 128 channels can be enabled or disabled for both transmit and receive. This is achieved by the 32-bit (R/X)CER register. Therefore, up to 32 channels can be enabled in two consecutive sub-frames, 16 in Partition A and 16 in Partition B.

Channels can be enabled or disabled at any time as long as it does not belong to the current (or active) sub-frame. An active sub-frame can be viewed as that time slot where transmission or reception is taking place. For example, if data transfer is ongoing in sub-frame 2 (channels 32–47 in (R/X)PABLK=1), the lower 16-bits of the channel enable registers (which correspond to Partition A) should not be changed since they would affect the enabling of the current channels. This can be ensured by using the current block status bits (R/X)CBLK in the MCR.

C64x is a trademark of Texas Instruments.

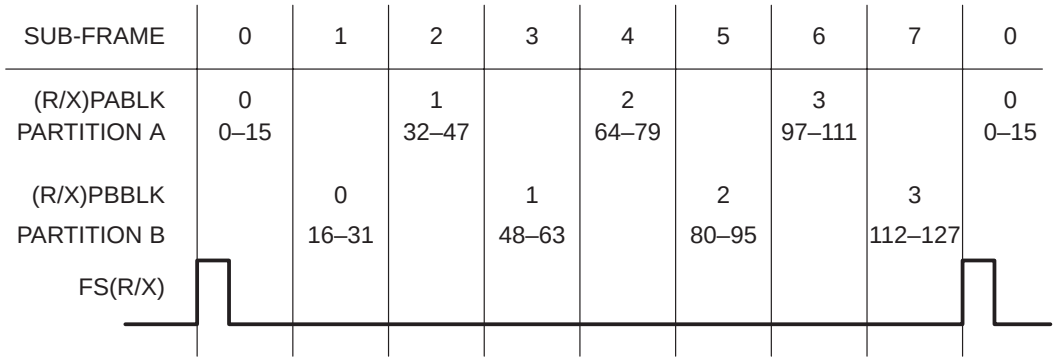

Figure 2. Sub-Frames, Partitions, and Channels in a Multichannel Frame

3.2 Enhanced Multichannel Selection Mode (C64x only)

The C64x series McBSP includes an enhanced multichannel selection mode that enables greater channel allocation flexibility, allowing up to 128 channels to be selected at any given time. This mode is accessed through two new bits in the MCR, the enhanced receive multichannel selection enable bit (RMCME) and the enhanced transmit multichannel selection enable bit (XMCME). This mode also adds six enhanced receive/transmit enable registers (RCERE1, RCERE2, RCERE3, XCERE1, XCERE2, and XCERE3) to the C64x McBSP. The XCER and RCER become XCERE0 and RCERE0 respectively when enhanced mode is enabled.

In enhanced multichannel mode, the registers RCERE0–RCERE3 and XCERE0–XCERE3 are used to enable up to 128 channels. Thus, the values in (R/X)PABLK and (R/X)PBBLK are don't cares. Table 1 shows the channel enable bits for a 128-channel data stream.

Table 1. Channel Enable Bits (in RCEREx/XCEREx) for a 128-Channel Data Stream

Channel Number of a 128-Channel Data Stream								
	0–15	**16–31**	**32–47**	**48– 63**	**64–79**	**80–95**	**96–111**	**112–127**
Register	RCERE0 XCERE0	RCERE0 XCERE0	RCERE1 XCERE1	RCERE1 XCERE1	RCERE2 XCERE2	RCERE2 XCERE2	RCERE3 XCERE3	RCERE3 XCERE3
Channel	R/XCE0 to R/XCE15	R/XCE16 to R/XCE31	R/XCE32 to R/XCE47	R/XCE48 to R/XCE63	R/XCE64 to R/XCE79	R/XCE80 to R/XCE95	R/XCE96 to R/XCE111	R/XCE112 to R/XCE127

4 Problem Description

When multiple devices transmit over the same line, care should be taken to avoid bus contention due to simultaneous or overlapped write accesses by two or more devices. Contention during transmission can be avoided by ensuring enough dead time between the last write of one device and the first write access of the next device.

To avoid data collision, the disable time (tdis(CKXH–DXZ)) of the DX output should be much smaller than the enable or delay time (td(CKXH–DX)) of the DX output for the next device. This is shown as Tdead in Figure 3 and is defined as:

Tdead = { td(CKXH–DX) – tdis(CKXH–DXZ) } >> 0

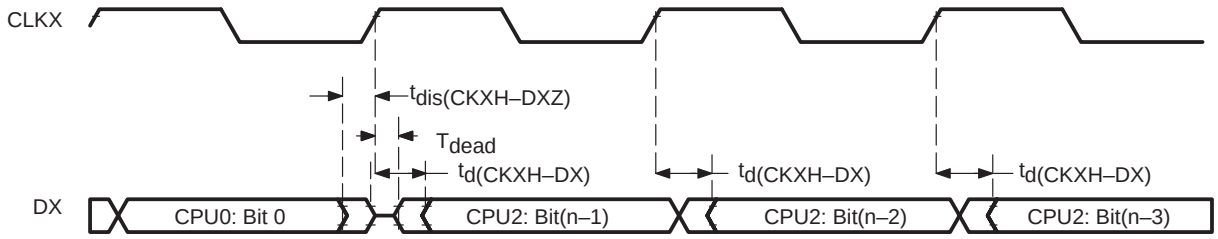

Figure 3. DX Timing for Multichannel Operation

The timing parameters and their values for the C6000 McBSP DX output, as per the datasheet, is shown in Table 2. It can be inferred from the timing numbers of the McBSP that the delay time and disable time for the DX pin is the same. Therefore, there can be bus contention if two McBSPs transmit on successive channels. The following sections will describe the workaround for this problem using the C6000 devices.

Table 2. Switching Characteristics of DX Output of C6201 McBSP

NO.		PARAMETER		MIN	MAX	UNIT
12	tdis(CKXH-DXZ)	Disable time, CLKX high to DX high impedance following last data bit	CLKX int	0	4	ns
			CLKX ext	3	16	ns
13	td(CKXH-DX)	Delay time, DX valid after CLKX high. For the first data bit, this is assured by design to be the delay time for data to become low impedance.	CLKX int	0	4	ns
			CLKX ext	3	16	ns

5 Solution Overview

The bus contention problem is due to lack of control on the DX output buffer's turn-on (delay) and turn-off (disable) time. This application report shows three solutions to the problem. Solution 1—dummy channel insertion—is applicable on all C6000 devices. Solution 2—DXENA—is recommended for C621x/C671x. Solution 3—DXENA plus enhanced multichannel selection mode—is the optimal solution for C64x TDM operations.

5.1 Solution 1: Dummy Channel Insertion (Recommended for C620x/C670x)

Contention can be avoided by programming the McBSP to transfer one extra (dummy) element or channel than what is necessary. This additional (dummy) channel should be disabled via the XCER register since it does not represent data of interest. A disabled channel has its DX in high-impedance state. The dummy channel will provide the necessary dead time between transfers from two McBSPs and thus prevent contention.

Consider the example where one McBSP in each of the three DSPs is connected to the TDM bus, as shown in Figure 4. Assume that DSP1 is the clock and frame master. As a result, DSP1 generates transmit and receive clocks and transmit and receive frame syncs.

In Figure 4, D(R/X)00 represents D(R/X) pins of McBSP0 in DSP0. Similarly D(R/X)01 and D(R/X)02 correspond to McBSP0 of DSP1 and DSP2. For this example, DX00 occupies sub-frame 0 and 1, DX01 occupies sub-frame 1, and DX02 occupies sub-frames 3, and 4. On the receive side, DR00 listens to sub-frame 1, 3, and 4, DR01 receives channels in sub-frame 0 and 3, and lastly, DR02 receives sub-frames 0 and 1.

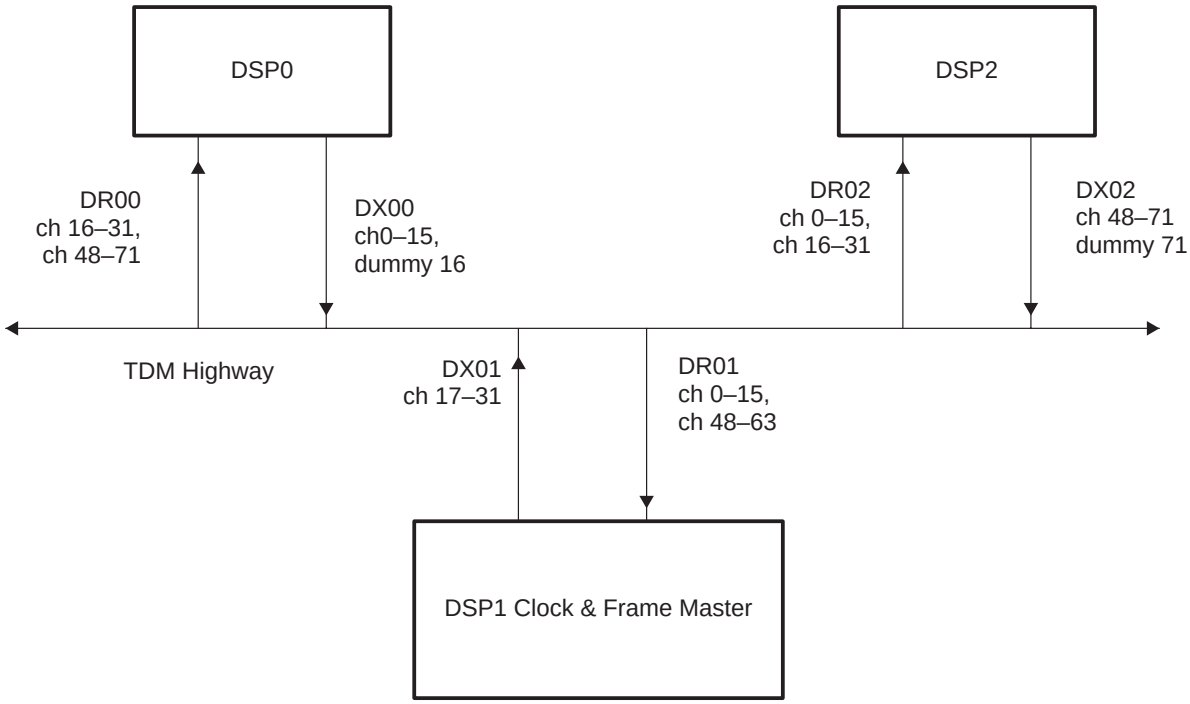

Figure 4. Three DSPs on a TDM Bus

The actual channels (or time slots) in which each of the McBSPs transmit and receive are listed in Table 3 and Table 4, respectively. In this TDM set up, up to 71 out of the available 128 channels are considered for data transfer. Among these 71 channels, only some are enabled for transmission and reception. In this example, some or all channels in sub-frame 0, 1, 3, and 4 are enabled, whereas sub-frame 2 corresponding to channels 32–47 is not used and is therefore disabled. To selectively choose channels, the appropriate multichannel mode has to be chosen. For this example, XMCM = RMCM = 1 will suit the application. The (R/X)MCM = 1 mode disables all 128 channels by default. The required channels are enabled via the 16-channel sub-frames (R/X)P(A/B)BLK and the channel enable registers (R/X)CER.

5.1.1 TDM Transmission

Since DSP1 is the clock and frame master, it will be programmed to generate frame sync for every 71 channels. The frame period (FPER) depends on the serial element size in a frame and the number of elements or channels in a frame. DSP0 and DSP2 are slaves and therefore will start their respective data transfer upon receiving the frame and clock from the master DSP1.

Table 3. TDM Data Transmission Setup

DSP#	Sub-frame/ XP(A/B)BLK	Channels Transmitted	Register Value
DSP#0 DX00	Sub-frame 0: XPABLK = 0 XPBBLK = 0	Sub-frame channels: 0–15 Enabled channels (XPABLK): 0–15 Dummy channel: 16 Disabled channels: all others	XCER = 0x0000FFFF
DSP#1 DX01	Sub-frame 1: XPBBLK = 0;	Sub-frame channels: 16–31 Enabled channels (XPBBLK): 17, 19 20–23, 25,27, 28–31 Disabled channels: all others	XCER = 0xFAFA0000
DSP#2 DX02	Sub-frames 3, and 4: XPBBLK = 1 XPABLK = 2	Sub-frame channels: 48–63 and 64–79 Enabled channels (XPBBLK): 48, 49, 51 52–54, 56, 58, 60, 61, 63 Enabled channels (XPABLK): 64–70 Dummy channel: 71 Disabled channels: all others	XCER = 0xB55B007F

Assume transmit and receive data delay to be 1. The frame sync from the master initiates data transmission on DSP0, since DSP0 is enabled for transmission on channels 0–15. The first data bit on DX is available one clock after the frame sync is active. At the same time, DSP1 and DSP2 are enabled to receive some channels on sub-frame 0. Note that more than one device can receive the same channel(s).

In the next sub-frame 1 (channels 16–31), DSP1 transmits starting from channel 17. DSP0 is programmed for 17 (channels 0–16) serial elements, and the last element will be disabled so that DX will be driven to high-impedance state. This configuration occurs because channel 16 has to be a dummy channel to prevent bus contention between DSP0 and 1.

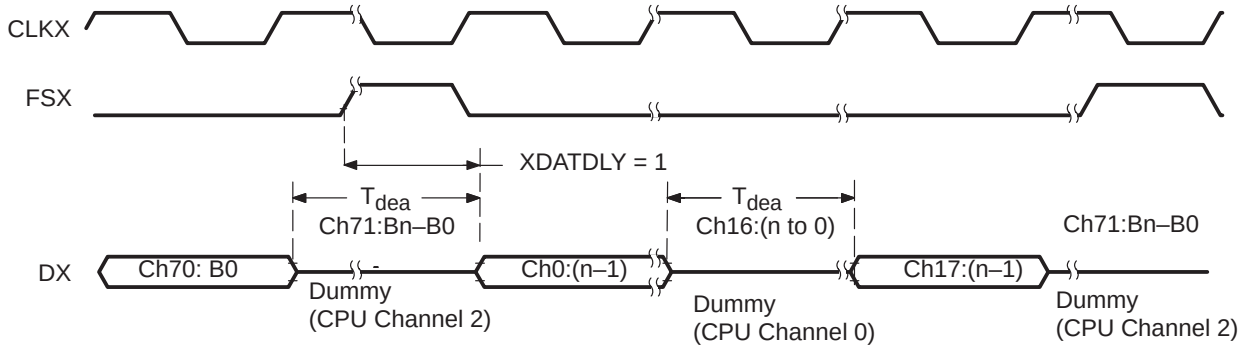

Figure 5. Dummy Channel on TDM Bus Used to Prevent Bus Contention

The transmission and reception continues on enabled channels up to channel 71. After channel 71 is transmitted on the TDM bus by DSP2, the next frame starts at channel 0. This configuration occurs because FPER is programmed for ((data-size in bits * 71) –1), and also because the data delay is 1. It provides for no gaps between two frames, which is known as maximum packet frequency. Since channel 0 is enabled, this again leads to successive channels (channel 71 and channel 0) being driven and resulting in bus contention. Therefore, channel 71 has to be a dummy channel and must be disabled.

5.1.2 TDM Reception

The receive section in DSP0, 1, and 2 can receive any channel of interest without any restrictions on overlapping of channels, as shown in Table 4. Note, however, that DSP0 receives channels from sub-frames 1, 3, and 4. Sub-frame 1 and 3 belong to partition B (RPBBLK) and sub-frame 4 to RPABLK. Note that the channels enabled for one sub-frame cannot be changed when it is the current one in transfer. Therefore, when the DSP0 receiver is in sub-frame 0 with a particular set of channels enabled (via RCER in RPBBLK 0), the RCER cannot be changed to be ready for the next sub-frame (3) which falls under the same partition. Before sub-frame 3 arrives, the receiver should be enabled for the appropriate channels (56, 58 in this case). To do this, the current block status bits RCBLK in the multichannel control register (MCR) can be probed to find out the current partition in progress. If the RCBLK does not point to block 0, then RCER can be programmed for channel enabling in sub-frame 3. This can also be done for the transmit side using the XCBLK, if required.

Table 4. TDM Data Reception Setup

DSP#	Sub-frame/ RP(A/B)BLK	Channels Received	Register Value
DSP#0 DR00	Sub-frames 1, 3, 4:	Sub-frame channels: 16–31, 48–63, and 64–71	
	RPBBLK = 0	Enabled channels (RPBBLK): 25, 27 Enabled channels (RPBBLK): 56, 58	RCER = 0x0A000000
	RPBBLK = 1	Enabled channels (RPABLK): 64–70	RCER = 0x05000000
	RPABLK = 1	Disabled channels: All others	RCER = 0x0000007F
DSP#1 DR01	Sub-frames 0, 3	Sub-frame channels: 0–15, 48–63	
	RPABLK = 0	Enabled channels (RPABLK): 0–7	
	RPBBLK = 1	Enabled channels (RPBBLK): 48, 49, 51, 52–54, 60, 61, 63 Disabled channels: all others	RCER = 0xB05B00FF
DSP#2 DR02	Sub-frames 0, 1:	Sub-frame channels: 0–15, 16–31	
	RPABLK = 0	Enabled channels (RPABLK): 0–15	
	RPBBLK = 0	Enabled channels (RPBBLK): 17, 19, 20–23, 28–31 Disabled channels: all others	RCER = 0xF0FAFFFF

5.1.3 McBSP Initialization

As in typical applications, consider that the DMA (C620x/C670x) or EDMA (C621x/C671x/C64x) services the McBSP. The McBSP initialization procedure for other typical applications is described in *TMS320C6000 McBSP Initialization* (SPRA488). The following steps describe the setup of interrupts, DMA or EDMA, and the McBSP in the required order for the TDM example above:

1. For DSP0, DSP1, and DSP2 McBSP0: Program the sample rate generator register (SRGR), serial port control register (SPCR), pin control register (PCR), receive control register (RCR), multichannel control register (MCR) and receive/transmit channel enable registers ((R/X)CER) to the required values.

   Caution: Do not set the $\overline{\text{GRST}}$ bit in SPCR in this step.

2. Take DSP1's sample rate generator out of reset by setting $\overline{\text{GRST}}$ = 1 in the SPCR. $\overline{\text{GRST}}$ is not required to be set for DSP0 and 2, since the clocks and frames are inputs. Therefore, the sample rate generator is not used for McBSP0.

3. Enabling Interrupts: To use interrupts, you have to set the global interrupt enable (GIE), and non-maskable interrupt enable (NMIE) bits in the IER. For the receiver of DSP0, an end-of-subframe interrupt (RINTM = 01b) can be used to determine the end of a 16-channel sub-frame, and thereby change the channel enabling for odd- or even-numbered sub-frame (see first row in Table 4).

   DMA: Select the DMA channel you want to use. Enable CPU interrupts corresponding to the DMA channel that will be used to service the McBSP. The default mapping of DMA channel-complete interrupts to the CPU can be used.

   EDMA: EDMA channels 12–15 are each associated with one specific McBSP transfer event. Unlike the DMA controller, one EDMA interrupt is shared for all channels.

   Choose EDMA channels based on the McBSP in use. Enable the EDMA interrupt to the CPU.

4. DMA initialization: Program the DMA source/destination registers and primary control register for required operation. Configure the R/WSYNC fields in the DMA primary control register to receive requests from the McBSP if event-synchronized operation is desired.

   EDMA initialization: EDMA initialization is similar to DMA except that configuration data is written to a parameter RAM entry instead of a memory-mapped register. Configure the event enable register (EER) if desired to receive requests from the McBSP.

5. Instruct the DMA to run if necessary. For example, set START = 01b in the DMA channel's primary control register to start the DMA without auto-initialization.

   When McBSP synchronization events are enabled, the DMA or EDMA will respond automatically to McBSP.

6. Take the transmitter and receiver of slave DSPs (0 and 2) out of reset. They will now await the clock (CLKR/X) and frame sync (FSR/X) from the master DSP1.

7. Take the transmitter and receiver of the master (DSP0) out of reset. Set $\overline{\text{FRST}}$ = 1 in DSP1. This causes the first frame sync output on FS(R/X) after 8-bit clocks.

5.2 Solution 2: DXENA (Recommended for C621x/C671x)

The C621x/C671x/C64x series of devices provides a hardware method to control the DX output buffer, called the DX enabler (DXENA). The DXENA field in the serial port control register (SPCR) controls the high impedance enable on the DX pin. When DXENA = 1, the DX pin turn-on time gains additional delay equal to two CPU clock cycles. Because the DXENA delay is independent from data transmission, the information seen on the TDM Highway is unchanged. The only change that occurs when DXENA = 1 is that the amount of time a transmitter's first bit is presented to the line is decreased. Figure 6 shows this extra delay time.

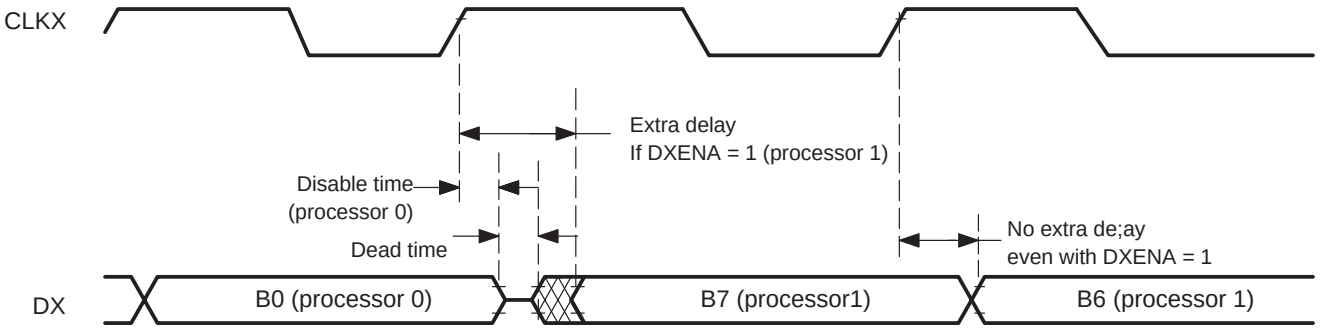

Figure 6. DX Timing for Multichannel Operation

Let's reconsider the previous example with DXENA = 1. We will keep the same three DSPs used in Figure 4, with DSP1 as the clock and frame master. There is no longer a need to insert wasted dummy channels. Notice that dummy channels 16 and 71 can be used to carry data. Figure 7 summarizes the new channel assignments. The DX enabler ensures that only one transmitter is active at any time; thus, no dummy channels are needed.

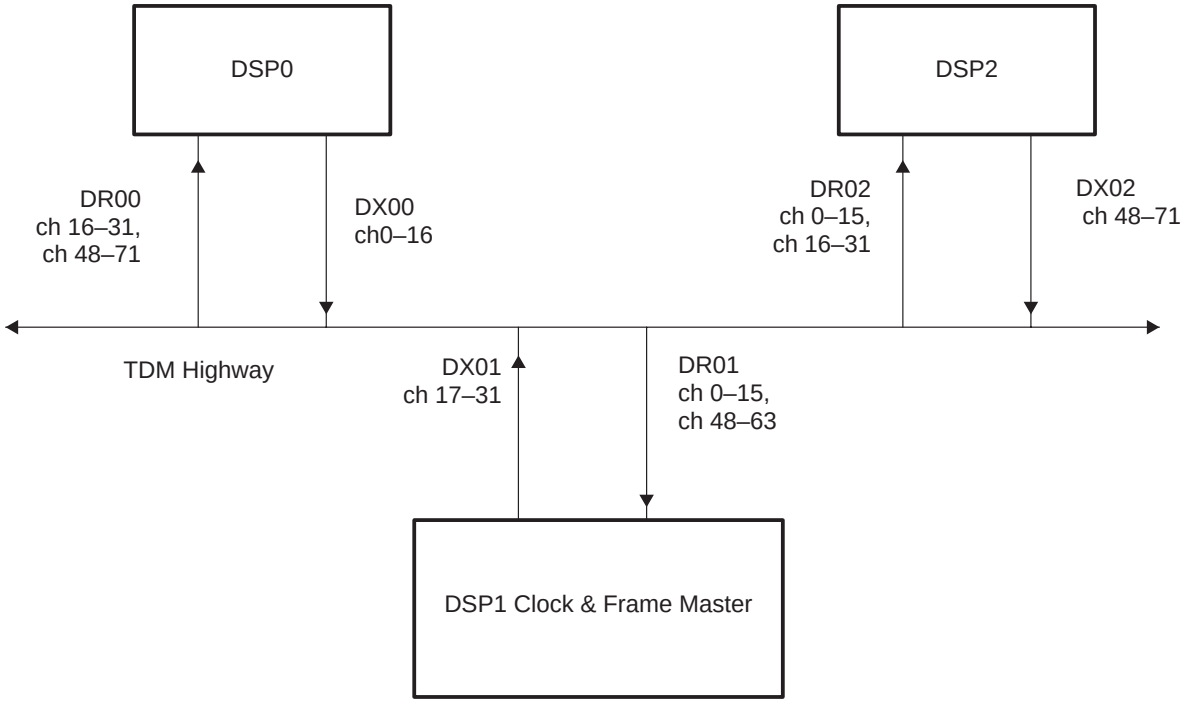

Figure 7. Three DSPs on a TDM Bus with DXENA = 1

5.3 Solution 3: DXENA and Enhanced Multichannel Selection Mode (Recommended for C64x)

A third solution is available to C64x users. In addition to the normal multichannel mode, this device adds the enhanced multichannel mode and features the DXENA.

Table 3 and Table 4 have also been updated to Table 5 and Table 6, respectively, with the new channel assignment information. Enhanced multichannel mode eliminates the need for (R/X)P(A/B)BLK fields, and these are shown as don't cares (X) in the table. There is no need to poll or update these fields to access multiple A or B blocks.

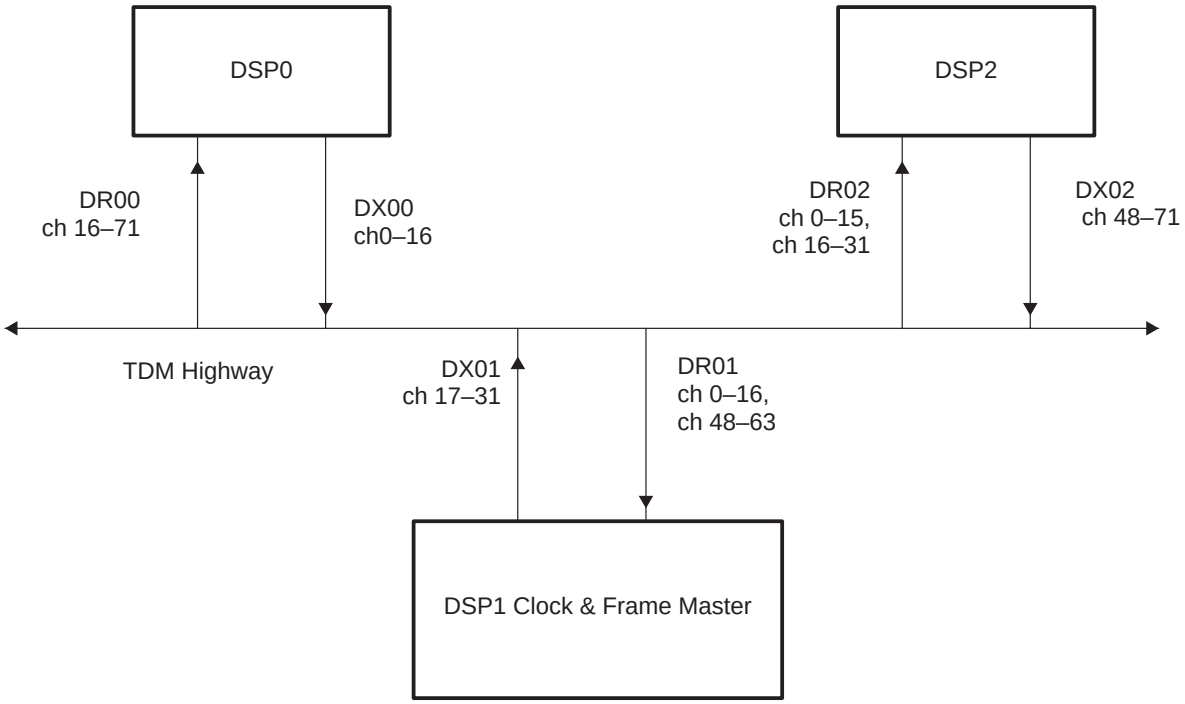

Figure 8. Three DSPs on a TDM Bus With DXENA = 1 and Enhanced Multichannel Mode

Table 5. TDM Data Transmission for Hardware Solution

DSP#	Channels Transmitted	Register Value
DSP#0 DX00	Enabled channels: 0–16 Dummy channel: None Disabled channels: all others	XCERE0 = 0x0001FFFF XCERE1 = 0x00000000 XCERE2 = 0x00000000 XCERE3 = 0x00000000
DSP#1 DX01	Enabled channels: 17, 19, 20–23, 25,27, 28–31 Disabled channels: all others	XCERE0 = 0xFAFA0000 XCERE1 = 0x00000000 XCERE2 = 0x00000000 XCERE3 = 0x00000000
DSP#2 DX02	Enabled channels: 32–49, 51–54, 56, 58, 60, 61, 63–71 Dummy channel: None Disabled channels: all others	XCERE0 = 0x00000000 XCERE1 = 0xB57BFFFF XCERE2 = 0x000000FF XCERE3 = 0x00000000

Table 6. TDM Data Reception for Hardware Solution

DSP#	Channels Received	Register Value
DSP#0 DR00	Enabled channels: 16,17, 19–23, 25, 27, 31, 56, 58, 64–71 Disabled channels: All others	RCERE0 = 0x8AFB0000 RCERE1 = 0x05000000 RCERE2 = 0x000000FF RCERE3 = 0x00000000
DSP#1 DR01	Enabled channels: 0–7, 48, 49, 51, 52–54, 60, 61, 63 Disabled channels: all others	RCERE0 = 0x000000FF RCERE1 = 0xB07B0000 RCERE2 = 0x00000000 RCERE3 = 0x00000000
DSP#2 DR02	Enabled channels: 0–15, 17, 19, 20–23, 28–31 Disabled channels: all others	RCERE0 = 0xF0FAFFFF RCERE1 = 0x00000000 RCERE2 = 0x00000000 RCERE3 = 0x00000000

Notice that McBSP initialization steps remain the same as before with two exceptions:

1. DXENA bit is now set to one in the SPCR in step one; and
2. RINTM does not have to be set in step three since subframes are not necessary in enhanced multichannel mode.

5.4 Conclusion

Multiple C6x™ devices can be connected in a multi-processor environment with one of them as a clock and frame master. This is achieved by using time-division multiplexing via the McBSP interface through the multichannel mode. Along with the standard multichannel mode, additional solutions are possible with certain C6000 family members. The DX enabler in the C621x/C671x/C64x allows increased TDM efficiency by eliminating dummy channels, and the C64x enhanced multichannel mode improves channel selection flexibility while decreasing operational complexity.

C6x is a trademark of Texas Instruments.

Appendix A Sample Code

```
/*****************************************************************************\
*           Copyright (C) 2000 Texas Instruments Incorporated.
*                          All Rights Reserved
*-----------------------------------------------------------------------------
* FILENAME...... XXXXXXXX
* DATE CREATED.. 11/28/2000
*-----------------------------------------------------------------------------
* This program demonstrates the McBSP in multichannel mode and is based on CSL v2.0.
* Functions used to configure the DMA or EDMA to service the McBSP are included.
* This program relies on DSP/BIOS to handle hardware and software interupts. These
* interupts service the Ping Pong buffers and maintain a count of the active
* frame for multichannel operation.
\*****************************************************************************/
#include <std.h>
#include <swi.h>
#include <log.h>
#include <clk.h>

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

/*-----------------------------------------------------------------------------*
/

/* declare DSP/BIOS objects created with the configuration tool                 */
extern far SWI_Obj SwiMain;
extern far LOG_Obj LogMain;
extern far SWI_Obj swiProcess;
extern far LOG_Obj trace;

/* Pick which EDMA transfer completion interrupt we want to use                 */
#define TCCINTNUM   10
```

```
/* define some constants                                                    */
#define BUFF_SZ  256               /* ping–pong buffer sizes in # of ints   */
#define BUFF_SZ_DR 32
#define BUFF_SZ_DX 32
#define EDMA_EIX 0
#define FCPU     150000000        /* CPU clock frequency                   */
#define SRATE    8000             /* data sample rate (simulated w/timer   */
#define TPRD     (FCPU/(4*SRATE)) /* timer period                          */

/* Create the buffers. We want to align the buffers to be cache friendly   */
/* by aligning them on an L2 cache line boundary.                           */
#pragma DATA_ALIGN(pingDX,128);
#pragma DATA_ALIGN(pongDX,128);
#pragma DATA_ALIGN(pingDR,128);
#pragma DATA_ALIGN(pongDR,128);

int pingDX[BUFF_SZ];
int pingDR[BUFF_SZ];
int pongDX[BUFF_SZ];
int pongDR[BUFF_SZ];

/* These two variables serve as the data sources for this example.          */
/* Also want to align these on a cache line boundary since they             */
/* sources of EDMA transfers.                                               */
#pragma DATA_ALIGN(ping_data,128);
#pragma DATA_ALIGN(pong_data,128);
static int ping_data;
static int pong_data;

/* global variable used to track the ping–pong'ing                          */
static int pingpong = 0;
int i=0;

/* declare some CSL objects                                                 */
#if (EDMA_SUPPORT)
EDMA_Handle hEdmaDR;     /* Handle for the EDMA channel                     */
EDMA_Handle hEdmaDX;     /* Handle for the EDMA channel                     */
EDMA_Handle hEdmaPing; /* Handle for the ping EDMA reload parameters        */
```

```
EDMA_Handle hEdmaPong; /* Handle for the pong EDMA reload parameters                */
EDMA_Config cfgEdma;  /* EDMA configuration structure                              */

EDMA_Handle hEdmaPingDR; /* Handle for the ping EDMA reload parameters             */
EDMA_Handle hEdmaPongDR; /* Handle for the pong EDMA reload parameters             */
EDMA_Config cfgEdmaDR;   /* EDMA configuration structure                           */

EDMA_Handle hEdmaPingDX; /* Handle for the ping EDMA reload parameters             */
EDMA_Handle hEdmaPongDX; /* Handle for the pong EDMA reload parameters             */
EDMA_Config cfgEdmaDX;   /* EDMA configuration structure                           */
#endif

#if (DMA_SUPPORT)
DMA_Handle hDmaDR;
DMA_Handle hDmaDX;
#endif

MCBSP_Handle hMcbsp;    /* Handle for the McBSP                                    */

/* Function prototype section                                                     */
void run_dma();
void run_edma();
void run_mcbsp();
void hwiEdmaIsr(int arg);
void HWI_rpblk(int arg);   /* ISR used to track subframes for multichannel mode   */
void swiProcessFunc();

/*--------------------------------------------------------------------------------*/

/*--------------------------------------------------------------------------------*/
void main(){

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

  /* initialize the input source data                                             */
  ping_data=0x00000000;
  pong_data=0x80000000;
```

```
  for (i=0;i<BUFF_SZ; i++) {
  pongDX[i]=i;
  pingDX[i]=i;
  pongDR[i]=0xF;
  pingDR[i]=0xF;
  }

#if (EDMA_SUPPORT)
  run_edma();
#endif

#if(DMA_SUPPORT)
  run_dma();
#endif
  run_mcbsp();
}

/*-------------------------------------------------------------------------------*/
void swiProcessFunc(){

  int *inbuff;
  int *outbuff;
  int x;

  if (pingpong){
    /* If pingpong is 0, then we own the ping input buffer                      */
    inbuff = pingDR;
    outbuff = pingDX;
  }else{
    /* If pingpong is 1, then we own the pong input buffer                      */
    inbuff = pongDR;
    outbuff = pongDX;
  }

  /* Now let's process the input buffer, for simplicity, we'll                  */
  /* just copy it to the output buffer.                                         */
  for (x=0; x<BUFF_SZ; x++) {
    outbuff[x] = ping_data+x;
  }
```

```
  ping_data++;

#if (EDMA_SUPPORT)
  /* If this example is enhanced to actually do something with the                */
  /* output buffer such as DMA it somewhere, you will want to flush               */
  /* it out of the cache first.                                                   */
  CACHE_flush(CACHE_L2,outbuff,BUFF_SZ);

  /* Since we're done processing the input buffer, clean it from cache,           */
  /* this invalidates it from cache to ensure we read a fresh version             */
  /* the next time.                                                               */
  CACHE_clean(CACHE_L2,inbuff,BUFF_SZ);
#endif
}

/*--------------------------------------------------------------------------------*/
void HWI_rpblk(int arg) {
  /* This ISR is called after every McBSP subframe                                */
  /* By changing the inactive RCER or XCER, >32 channels can be enabled           */
  LOG_printf(&trace,"Here is where to modify the XCER and RCER");
}
/*--------------------------------------------------------------------------------*/
void hwiEdmaIsr(int arg){
  /* Perform ping-pong                                                            */
  pingpong = (pingpong + 1) & 1;

#if (DMA_SUPPORT)
  /* Clear the pending condition from the DMA secondary control register          */
  DMA_CLEAR_CONDITION(hDmaDR,DMA_SECCTL_FRAMECOND);
#endif

#if (EDMA_SUPPORT)
  /* Clear the pending interrupt from the EDMA interrupt pending register         */
  EDMA_intClear(TCCINTNUM);

  /* Based on if we ping'ed or pong'ed, we need to set the EDMA channel           */
  /* link address for the NEXT frame.                                             */
  if (pingpong){
    /* Now filling pong so set link to ping                                       */
    EDMA_link(hEdmaDR,hEdmaPingDR);
```

```
    EDMA_link(hEdmaDX,hEdmaPingDX);
  }
  else{
    /* Now filling ping so set link to pong                                    */
    EDMA_link(hEdmaDR,hEdmaPongDR);
    EDMA_link(hEdmaDX,hEdmaPongDX);
  }
#endif
  /* Notify the app that we just ping–pong'ed                                  */
  SWI_post(&swiProcess);
}
/*--------------------------------------------------------------------------*/
void run_mcbsp() {
  /* create a config structure for the McBSP                                   */
  static MCBSP_Config ConfigLoopback = {
#if (EDMA_SUPPORT)
    MCBSP_SPCR_RMK(
         MCBSP_SPCR_FRST_NO,
         MCBSP_SPCR_GRST_NO,
   MCBSP_SPCR_XINTM_FRM,
         MCBSP_SPCR_XSYNCERR_NO,
         MCBSP_SPCR_XRST_NO,
         MCBSP_SPCR_DLB_OFF,
         MCBSP_SPCR_RJUST_RZF,
         MCBSP_SPCR_CLKSTP_DISABLE,
         MCBSP_SPCR_DXENA_ON,
         MCBSP_SPCR_RINTM_FRM,
         MCBSP_SPCR_RSYNCERR_NO,
         MCBSP_SPCR_RRST_NO
         ),
    MCBSP_RCR_RMK(
        MCBSP_RCR_RPHASE_SINGLE,
        MCBSP_RCR_RFRLEN2_OF(0),
        MCBSP_RCR_RWDLEN2_8BIT,
        MCBSP_RCR_RCOMPAND_MSB,
        MCBSP_RCR_RFIG_YES,
        MCBSP_RCR_RDATDLY_0BIT,
        MCBSP_RCR_RPHASE2_NORMAL,
        MCBSP_RCR_RFRLEN1_OF(31),
        MCBSP_RCR_RWDLEN1_32BIT,
```

```
        MCBSP_RCR_RWDREVRS_DISABLE
        ),
    MCBSP_XCR_RMK(
        MCBSP_XCR_XPHASE_SINGLE,
        MCBSP_XCR_XFRLEN2_OF(0),
        MCBSP_XCR_XWDLEN2_8BIT,
        MCBSP_XCR_XCOMPAND_MSB,
        MCBSP_XCR_XFIG_YES,
        MCBSP_XCR_XDATDLY_0BIT,
        MCBSP_XCR_XPHASE2_NORMAL,
        MCBSP_XCR_XFRLEN1_OF(31),
        MCBSP_XCR_XWDLEN1_32BIT,
        MCBSP_XCR_XWDREVRS_DISABLE
        ),
#endif
#if (DMA_SUPPORT)
    MCBSP_SPCR_RMK(
         MCBSP_SPCR_FRST_NO,
         MCBSP_SPCR_GRST_NO,
         MCBSP_SPCR_XINTM_FRM,
         MCBSP_SPCR_XSYNCERR_NO,
         MCBSP_SPCR_XRST_NO,
         MCBSP_SPCR_DLB_ON,
         MCBSP_SPCR_RJUST_RZF,
         MCBSP_SPCR_CLKSTP_DISABLE,
         MCBSP_SPCR_RINTM_EOS,
         MCBSP_SPCR_RSYNCERR_NO,
         MCBSP_SPCR_RRST_NO
         ),
    MCBSP_RCR_RMK(
        MCBSP_RCR_RPHASE_SINGLE,
        MCBSP_RCR_RFRLEN2_OF(0),
        MCBSP_RCR_RWDLEN2_8BIT,
        MCBSP_RCR_RCOMPAND_MSB,
        MCBSP_RCR_RFIG_YES,
        MCBSP_RCR_RDATDLY_0BIT,
        MCBSP_RCR_RFRLEN1_OF(31),
        MCBSP_RCR_RWDLEN1_32BIT
        ),
    MCBSP_XCR_RMK(
```

```
        MCBSP_XCR_XPHASE_SINGLE,
        MCBSP_XCR_XFRLEN2_OF(0),
        MCBSP_XCR_XWDLEN2_8BIT,
        MCBSP_XCR_XCOMPAND_MSB,
        MCBSP_XCR_XFIG_YES,
        MCBSP_XCR_XDATDLY_0BIT,
        MCBSP_XCR_XFRLEN1_OF(31),
        MCBSP_XCR_XWDLEN1_32BIT
        ),
#endif
    MCBSP_SRGR_RMK(
         MCBSP_SRGR_GSYNC_FREE,
         MCBSP_SRGR_CLKSP_RISING,
         MCBSP_SRGR_CLKSM_INTERNAL,
         MCBSP_SRGR_FSGM_DXR2XSR,
         MCBSP_SRGR_FPER_OF(1023),
         MCBSP_SRGR_FWID_OF(127),
         MCBSP_SRGR_CLKGDV_OF(15)
         ),
    MCBSP_MCR_RMK(
        MCBSP_MCR_XPBBLK_SF1,
        MCBSP_MCR_XPABLK_SF0,
        MCBSP_MCR_XMCM_DISXP,
        MCBSP_MCR_RPBBLK_SF1,
        MCBSP_MCR_RPABLK_SF0,
        MCBSP_MCR_RMCM_ELDISABLE
        ),
    MCBSP_RCER_RMK(
         /*Change the RCER to reflect desired enabled receive channels           */
         /*Example: Enable recv channs 31,30,28,27,26,25,23,22,19,17 in Partition A*/
         /*and recv channels 15,14,12,11,10,9,1,0 in Partition B                  */
         /*(Note that BUFF_SZ_DR should change to reflect the # of channels enabled*/
         //MCBSP_RCER_RCEB_OF(0xDE03),
         //MCBSP_RCER_RCEA_OF(0xDECA),
         /*Default to enable all channels                                         */
         MCBSP_RCER_RCEB_OF(0xFFFF),
         MCBSP_RCER_RCEA_OF(0xFFFF)
         ),
    MCBSP_XCER_RMK(
         /* Change the XCER to reflect desired enabled receive channels          */
```

```
        /* Example: to enable xmit channels 15,13,10,8,7,5,2,1 in Partition A    */
        /* and xmit channels 31,28,27,26,24,23,21,19,18,16 in Partition B        */
        /* (Note that BUFF_SZ_DX should change to reflect the # of channels enabled/
        //MCBSP_XCER_XCEB_OF(0x9DAD),
        //MCBSP_XCER_XCEA_OF(0xA5A6)
        /*Default to enable all channels                                         */
        MCBSP_XCER_XCEB_OF(0xFFFF),
        MCBSP_XCER_XCEA_OF(0xFFFF)
        ),
  MCBSP_PCR_RMK(
       MCBSP_PCR_XIOEN_SP,
       MCBSP_PCR_RIOEN_SP,
       MCBSP_PCR_FSXM_INTERNAL,
       MCBSP_PCR_FSRM_EXTERNAL,
       MCBSP_PCR_CLKXM_OUTPUT,
       MCBSP_PCR_CLKRM_INPUT,
       MCBSP_PCR_CLKSSTAT_0,
       MCBSP_PCR_DXSTAT_0,
       MCBSP_PCR_DXSTAT_0,
       MCBSP_PCR_FSRP_ACTIVEHIGH,
       MCBSP_PCR_CLKXP_RISING,
       MCBSP_PCR_CLKRP_FALLING
       )
};

/* Let's open up serial port 1                                               */
hMcbsp = MCBSP_open(MCBSP_DEV1, MCBSP_OPEN_RESET);

/* We'll set it up for digital loopback, 32bit mode. We have                 */
/* to setup the sample rate generator to allow self clocking.                */
MCBSP_config(hMcbsp,&ConfigLoopback);

/*Configure McBSP Receive interupt to 11                                     */
/*We'll use this to count subframes to enable >32 channels in multichannel mode  */
IRQ_disable(IRQ_EVT_RINT1);
IRQ_clear(IRQ_EVT_RINT1);
IRQ_map(IRQ_EVT_RINT1,11);
/* Now that the port is setup, let's enable it in steps.                     */
MCBSP_enableRcv(hMcbsp);
MCBSP_enableXmt(hMcbsp);
```

```
  MCBSP_enableSrgr(hMcbsp);

  IRQ_enable(IRQ_EVT_RINT1);
}

#if (DMA_SUPPORT)
void run_dma() {
  Uint32 PriCtlDR,SecCtlDR,SrcAddrDR,DstAddrDR,XfrCntDR;
  Uint32 PriCtlDX,SecCtlDX,SrcAddrDX,DstAddrDX,XfrCntDX;
  Uint32 GblIdx0,GblAddr0,GblAddr1,GblCnt0;

  /* Let's use the DMA to perform a simple data copy from                    */
  /* one buffer to another.                                                  */

  /* To start, we need to open up a DMA channel. Let's use                   */
  /* DMA channel 2 and also reset it upon opening.                           */
  hDmaDR = DMA_open(DMA_CHA1,DMA_OPEN_RESET);
  hDmaDX = DMA_open(DMA_CHA2,DMA_OPEN_RESET);

  /*Use CSL to allocate DMA registers                                        */
  /*Use address registers to reload the ping buffer after the end of each frame    */
  /*Use index register to advance from ping to pong                         */
  /*Use count register to reload the transfer control register              */
  GblAddr0 = DMA_allocGlobalReg(DMA_GBL_ADDRRLD,(Uint32)pingDR);
  GblAddr1 = DMA_allocGlobalReg(DMA_GBL_ADDRRLD,(Uint32)pingDX);
  GblIdx0  = DMA_allocGlobalReg(DMA_GBL_INDEX,((((Uint32)pongDR – (Uint32)pingDR
– (BUFF_SZ_DR * 4) + 4)<<16)| 4 ));
  GblCnt0  = DMA_allocGlobalReg(DMA_GBL_CNTRLD,( (2 <<16) | BUFF_SZ_DR));

  /* Generate discrete DMA parameters using 'make' macros                    */
  PriCtlDR  = DMA_PRICTL_RMK(
               DMA_PRICTL_DSTRLD_OF(GblAddr0),
               DMA_PRICTL_SRCRLD_NONE,
               DMA_PRICTL_EMOD_NOHALT,
               DMA_PRICTL_FS_DISABLE,
               DMA_PRICTL_TCINT_ENABLE,
               DMA_PRICTL_PRI_CPU,
               DMA_PRICTL_WSYNC_NONE,
               DMA_PRICTL_RSYNC_REVT1,
               DMA_PRICTL_INDEX_OF(GblIdx0),
               DMA_PRICTL_CNTRLD_OF(GblCnt0),
```

```
                DMA_PRICTL_SPLIT_DISABLE,
                DMA_PRICTL_ESIZE_32BIT,
                DMA_PRICTL_DSTDIR_IDX,
                DMA_PRICTL_SRCDIR_NONE,
                DMA_PRICTL_START_STOP
                );
  SecCtlDR  = DMA_SECCTL_RMK(
                DMA_SECCTL_DMACEN_LOW,
                DMA_SECCTL_WSYNCCLR_CLEAR,
                DMA_SECCTL_WSYNCSTAT_CLEAR,
                DMA_SECCTL_RSYNCCLR_CLEAR,
                DMA_SECCTL_RSYNCSTAT_CLEAR,
                DMA_SECCTL_WDROPIE_DISABLE,
                DMA_SECCTL_WDROPCOND_CLEAR,
                DMA_SECCTL_RDROPIE_DISABLE,
                DMA_SECCTL_RDROPCOND_CLEAR,
                DMA_SECCTL_BLOCKIE_DISABLE,
                DMA_SECCTL_BLOCKCOND_CLEAR,
                DMA_SECCTL_LASTIE_DISABLE,
                DMA_SECCTL_LASTCOND_CLEAR,
                DMA_SECCTL_FRAMEIE_ENABLE,
                DMA_SECCTL_FRAMECOND_CLEAR,
                DMA_SECCTL_SXIE_DISABLE,
                DMA_SECCTL_SXCOND_CLEAR
                );
  SrcAddrDR = (Uint32)MCBSP_getRcvAddr(hMcbsp);
  DstAddrDR = (Uint32)pingDR;
  XfrCntDR  = DMA_XFRCNT_RMK(
                DMA_XFRCNT_FRMCNT_OF(2),
                DMA_XFRCNT_ELECNT_OF(BUFF_SZ_DR)
                );

  PriCtlDX  = DMA_PRICTL_RMK(
                DMA_PRICTL_DSTRLD_NONE,
                DMA_PRICTL_SRCRLD_OF(GblAddr1),
                DMA_PRICTL_EMOD_NOHALT,
                DMA_PRICTL_FS_DISABLE,
                DMA_PRICTL_TCINT_DISABLE,
                DMA_PRICTL_PRI_CPU,
                DMA_PRICTL_WSYNC_XEVT1,
```

```
                DMA_PRICTL_RSYNC_NONE,
                DMA_PRICTL_INDEX_OF(GblIdx0),
                DMA_PRICTL_CNTRLD_OF(GblCnt0),
                DMA_PRICTL_SPLIT_DISABLE,
                DMA_PRICTL_ESIZE_32BIT,
                DMA_PRICTL_DSTDIR_NONE,
                DMA_PRICTL_SRCDIR_IDX,
                DMA_PRICTL_START_STOP
                );
  SecCtlDX  = DMA_SECCTL_RMK(
                DMA_SECCTL_DMACEN_LOW,
                DMA_SECCTL_WSYNCCLR_CLEAR,
                DMA_SECCTL_WSYNCSTAT_CLEAR,
                DMA_SECCTL_RSYNCCLR_CLEAR,
                DMA_SECCTL_RSYNCSTAT_CLEAR,
                DMA_SECCTL_WDROPIE_DISABLE,
                DMA_SECCTL_WDROPCOND_CLEAR,
                DMA_SECCTL_RDROPIE_DISABLE,
                DMA_SECCTL_RDROPCOND_CLEAR,
                DMA_SECCTL_BLOCKIE_DISABLE,
                DMA_SECCTL_BLOCKCOND_CLEAR,
                DMA_SECCTL_LASTIE_DISABLE,
                DMA_SECCTL_LASTCOND_CLEAR,
                DMA_SECCTL_FRAMEIE_DISABLE,
                DMA_SECCTL_FRAMECOND_CLEAR,
                DMA_SECCTL_SXIE_DISABLE,
                DMA_SECCTL_SXCOND_CLEAR
                );
  SrcAddrDX = (Uint32)pingDX;
  DstAddrDX = (Uint32)MCBSP_getXmtAddr(hMcbsp);
  XfrCntDX  = DMA_XFRCNT_RMK(
                DMA_XFRCNT_FRMCNT_OF(2),
                DMA_XFRCNT_ELECNT_OF(BUFF_SZ_DX)
                );

  /* Configure up the DMA channels                                          */
  DMA_configArgs(hDmaDR,PriCtlDR,SecCtlDR,SrcAddrDR,DstAddrDR,XfrCntDR);
```

```
   DMA_configArgs(hDmaDX,PriCtlDX,SecCtlDX,SrcAddrDX,DstAddrDX,XfrCntDX);

  /* Configure DMA interupt                                                  */
  IRQ_disable(IRQ_EVT_DMAINT1);
  IRQ_clear(IRQ_EVT_DMAINT1);
  IRQ_map(IRQ_EVT_DMAINT1,8);

  /* Start the DMA operation                                                 */
  DMA_autoStart(hDmaDR);
  DMA_autoStart(hDmaDX);

  /*Enable the DMA Interupt                                                  */
  IRQ_enable(IRQ_EVT_DMAINT1);
}
#endif

#if (EDMA_SUPPORT)
void run_edma()
{/* Create the EDMA configuration structure for ping transfers              */
  EDMA_Config cfgEdmaPingDX = {
    EDMA_OPT_RMK(
        EDMA_OPT_PRI_LOW,
        EDMA_OPT_ESIZE_32BIT,
        EDMA_OPT_2DS_NO,
        EDMA_OPT_SUM_INC,
        EDMA_OPT_2DD_NO,
        EDMA_OPT_DUM_NONE,
        EDMA_OPT_TCINT_NO,
        EDMA_OPT_TCC_OF(TCCINTNUM),
        EDMA_OPT_LINK_YES,
        EDMA_OPT_FS_NO
        ),
    EDMA_SRC_OF(pingDX),
    EDMA_CNT_OF(BUFF_SZ_DX),
    EDMA_DST_OF(0), /*Will set later using CSL                               */
    EDMA_IDX_OF(EDMA_EIX),
    EDMA_RLD_OF(0x00000000)
  };
```

```
/* Create the EDMA configuration structure for pong transfers                  */
EDMA_Config cfgEdmaPongDX = {
  EDMA_OPT_RMK(
     EDMA_OPT_PRI_LOW,
     EDMA_OPT_ESIZE_32BIT,
     EDMA_OPT_2DS_NO,
     EDMA_OPT_SUM_INC,
     EDMA_OPT_2DD_NO,
     EDMA_OPT_DUM_NONE,
     EDMA_OPT_TCINT_NO,
     EDMA_OPT_TCC_OF(TCCINTNUM),
     EDMA_OPT_LINK_YES,
     EDMA_OPT_FS_NO
     ),
  EDMA_SRC_OF(pongDX),
  EDMA_CNT_OF(BUFF_SZ_DX),
  EDMA_DST_OF(0), /*Will set later using CSL                                   */
  EDMA_IDX_OF(EDMA_EIX),
  EDMA_RLD_OF(0x00000000)
};

/* Create the EDMA configuration structure for ping transfers                  */
EDMA_Config cfgEdmaPingDR = {
  EDMA_OPT_RMK(
     EDMA_OPT_PRI_LOW,
     EDMA_OPT_ESIZE_32BIT,
     EDMA_OPT_2DS_NO,
     EDMA_OPT_SUM_NONE,
     EDMA_OPT_2DD_NO,
     EDMA_OPT_DUM_INC,
     EDMA_OPT_TCINT_YES,
     EDMA_OPT_TCC_OF(TCCINTNUM),
     EDMA_OPT_LINK_YES,
     EDMA_OPT_FS_NO
     ),
  EDMA_SRC_OF(0), /*Will set later using CSL                                   */
  EDMA_CNT_OF(BUFF_SZ_DR),
  EDMA_DST_OF(pingDR),
```

```
  EDMA_IDX_OF(EDMA_EIX),
  EDMA_RLD_OF(0x00000000)
};

/* Create the EDMA configuration structure for pong transfers                    */
EDMA_Config cfgEdmaPongDR = {
  EDMA_OPT_RMK(
      EDMA_OPT_PRI_LOW,
      EDMA_OPT_ESIZE_32BIT,
      EDMA_OPT_2DS_NO,
      EDMA_OPT_SUM_NONE,
      EDMA_OPT_2DD_NO,
      EDMA_OPT_DUM_INC,
      EDMA_OPT_TCINT_YES,
      EDMA_OPT_TCC_OF(TCCINTNUM),
      EDMA_OPT_LINK_YES,
      EDMA_OPT_FS_NO
      ),
  EDMA_SRC_OF(0), /*Will set later using CSL                                     */
  EDMA_CNT_OF(BUFF_SZ_DR),
  EDMA_DST_OF(pongDR),
  EDMA_IDX_OF(EDMA_EIX),
  EDMA_RLD_OF(0x00000000)
};

cfgEdmaPingDR.src = MCBSP_getRcvAddr(hMcbsp);
cfgEdmaPongDR.src = MCBSP_getRcvAddr(hMcbsp);
cfgEdmaPingDX.dst = MCBSP_getXmtAddr(hMcbsp);
cfgEdmaPongDX.dst = MCBSP_getXmtAddr(hMcbsp);

/* Although not required, let's clear all of the EDMA parameter RAM.            */
/* This makes it easier to view the RAM and see the changes as we configure it  */
EDMA_clearPram(0x00000000);

/* Lets open up the EDMA channel associated with timer #1.                      */
hEdmaDR = EDMA_open(EDMA_CHA_REVT1, EDMA_OPEN_RESET);
hEdmaDX = EDMA_open(EDMA_CHA_XEVT1, EDMA_OPEN_RESET);
```

```
/* We also need two EDMA reload parameters for each set of buffers sets so       */
/* let's allocate them here. Notice the –1, this means allocate any availale tale.*/
hEdmaPingDR = EDMA_allocTable(–1);
hEdmaPongDR = EDMA_allocTable(–1);
hEdmaPingDX = EDMA_allocTable(–1);
hEdmaPongDX = EDMA_allocTable(–1);

/* Let's copy the ping reload configuration structure to an                      */
/* intermediate configuration structure.                                         */
cfgEdmaDR = cfgEdmaPingDR;
cfgEdmaDX = cfgEdmaPingDX;

/* Let's initialize the link fields of the configuration structures              */
cfgEdmaPingDR.rld = EDMA_RLD_RMK(BUFF_SZ_DR,hEdmaPingDR);
cfgEdmaPongDR.rld = EDMA_RLD_RMK(BUFF_SZ_DR,hEdmaPongDR);
cfgEdmaDR.rld     = EDMA_RLD_RMK(BUFF_SZ_DR,hEdmaPongDR);

cfgEdmaPingDX.rld = EDMA_RLD_RMK(BUFF_SZ_DX,hEdmaPingDX);
cfgEdmaPongDX.rld = EDMA_RLD_RMK(BUFF_SZ_DX,hEdmaPongDX);
cfgEdmaDX.rld     = EDMA_RLD_RMK(BUFF_SZ_DX,hEdmaPongDX);

/* Now let's program up the EDMA channel with the configuration structure       */
EDMA_config(hEdmaDR, &cfgEdmaPongDR);
EDMA_config(hEdmaDX, &cfgEdmaPongDX);

/* Let's also configure the reload parameter tables in the EDMA PRAM             */
/* with the values in the configuration structures.                              */
EDMA_config(hEdmaPingDR, &cfgEdmaPingDR);
EDMA_config(hEdmaPongDR, &cfgEdmaPongDR);
EDMA_config(hEdmaPingDX, &cfgEdmaPingDX);
EDMA_config(hEdmaPongDX, &cfgEdmaPongDX);

/* Let's disable/clear related interrupts just in case they are pending          */
/* fram a previous run of the program.                                           */

IRQ_reset(IRQ_EVT_EDMAINT);
EDMA_intDisable(TCCINTNUM);
EDMA_intClear(TCCINTNUM);
```

```
  /* Enable the EDMA channel                                                  */
  EDMA_enableChannel(hEdmaDX);
  EDMA_enableChannel(hEdmaDR);

  /* Enable the related interrupts                                            */
  IRQ_enable(IRQ_EVT_EDMAINT);
  EDMA_intEnable(TCCINTNUM);
}
#endif
```

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement, and limitation of liability.

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

Customers are responsible for their applications using TI components.

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

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, license, warranty or endorsement thereof.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations and notices. Representation or reproduction of this information with alteration voids all warranties provided for an associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Resale of TI's products or services with *statements different from or beyond the parameters* stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Also see: Standard Terms and Conditions of Sale for Semiconductor Products. www.ti.com/sc/docs/stdterms.htm

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265