Application Report
SPRA569A - August 2001

TMS320C6000 McBSP: IOM-2 Interface

Eric Biscondi
Scott Chen

Digital Signal Processing Solutions

ABSTRACT

This document describes how the multi-channel buffered serial port (McBSP) in the Texas Instruments (TI) TMS320C6000™ (C6000™) digital signal processor (DSP) family is used to communicate to an ISDN Oriented Modular Interface Revision 2 (IOM-2) bus-compliant device. This document also describes the usage of McBSP registers and sample code to perform the above function. The sample code described in this application report can be downloaded from http://www.ti.com/lit/zip/SPRA569.

Contents

List of Figures

1 Design Problem

How do I use the multi-channel buffered serial port to communicate to an IOM-2 compliant device?

2 IOM-2 Serial Bus

The IOM-2 standard defines an industry-standard serial bus for interconnecting telecommunications ICs.[1] The serial bus IOM-2 provides a full-duplex communication link containing user control data, control/programming, and status channel.

The data clock (DCL) is used to clock data and operates at twice the data rate. IOM-2 frames are delimited by an 8-KHz frame sync signal (FSC). DU/DD (data upstream/data downstream) are the up/down serial information streams. The IOM-2 standard defines two operating modes:

- Line card mode
- Terminal mode

Both modes utilize the same basic frame and clocking structure but differ in the number and usage of the channels. The terminal mode consists of three sub-frames (channels). The line card mode consists of one to eight sub-frames (channels). The term *sub-frame* or *channel* in IOM-2 terminology refers to the number of serial words (elements) transferred per frame. A sub-frame contains information for an IOM-2 channel.

Figure 1 shows a simplified frame format of an IOM-2 serial bus in terminal mode. The data clock rate is 16 kHz x *N*, where *N* is the number of sub-frames.

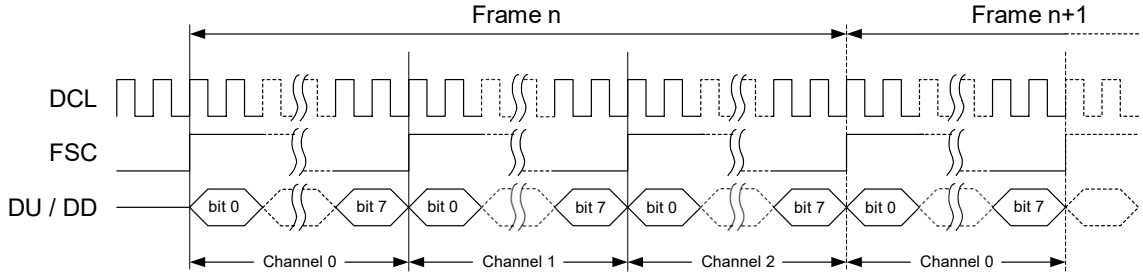

Figure 1. IOM-2 Terminal Mode Frame Format

3 McBSP Operation for IOM-2

The McBSP provides a direct interface to IOM-2-compliant devices. This document explains how to interface the McBSP to an IOM-2-compliant device.

Figure 2 shows the McBSP-to-IOM-2 interface. The IOM-2 clock DCL is used to drive McBSP's CLKS pin. The sample rate generator in the McBSP generates the internal clock signals CLKX and CLKR. The IOM-2 frame sync signal, FSC, is used to drive the McBSP receive and transmit frame sync (FSR and FSX). The IOM-2 data line, DU and DD, are connected to the McBSP DX and DR pins, respectively.

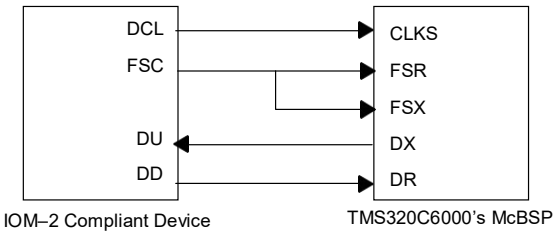

Figure 2. Connection Between the McBSP and an IOM-2 Compliant Device

4 McBSP Register Configuration

As shown in Figure 2, FSR, FSX, CLKS, and DR are inputs; DX is an output. To configure the McBSP pins, the pin control register (PCR) must be initialized as shown in Figure 3:

- (X/R)IOEN = 0. Serial port pins are not general-purpose I/Os.
- FS(X/R)M = 0. The frame sync signals (FSX/FSR) are driven by IOM-2 frame sync signal (FSC).
- CLK(X/R)M = 1. Receive and transmit clocks (CLKR and CLKX) are driven by the sample rate generator.
- FS(X/R)P = 0. Frame-sync signal is active high.
- CLKXP = 0. Transmit data is driven on rising edge of CLKX.
- CLKRP = 0. Receive data is sampled on falling edge of CLKR.

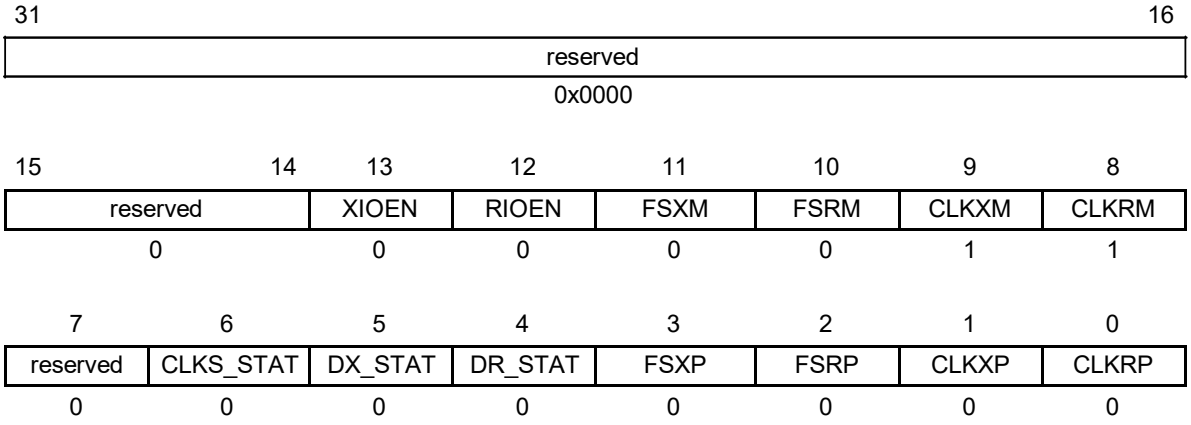

Figure 3. Pin Control Register (PCR)

As shown in Figure 4, the sample rate generator register (SRGR) is used to configure the internal McBSP frame and clock generator:[4, Chapter 8-5]

- CLKSP = 0. The rising edge of CLKS generates CLKG and thus CLK(R/X)_int.
- CLKSM = 0. The external clock (CLKS) drives the sample rate generator.
- CLKGDV = 1. The sample rate generator clock frequency is DCL divided by 2. This is to comprehend the IOM-2 data rate of 1-bit every two DCL clocks.
- The external frame FSC pulse dictates the arrival of a new frame; therefore, the frame period (FPER) and the frame width (FWID) are not programmed.

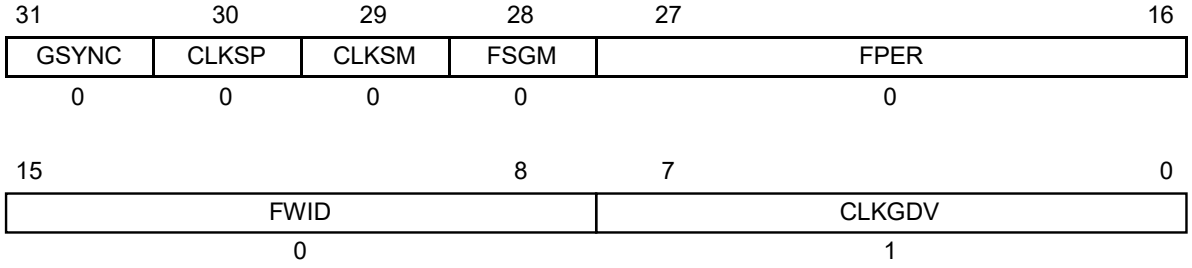

Figure 4. Sample Rate Generator Register (SRGR)

As shown in Figure 5, the receive and transmit control registers (RCR and XCR) initialize various parameters for the receive and transmit operations:

- (R/X)PHASE = 0. Single-phase frame (R/X)FRLEN2 and (R/X)WDLEN2 are not used.
- (R/X)WDLEN1 = 0. Element length is set to 8-bit (byte-base).
- (R/X)DATDLY = 0. No delay between the rising edge of the frame sync and the first data.
- (R/X)FRLEN1 = (4 * *N* - 1). Depends on which mode is used: one to eight channels when running in line–card mode; three channels when running in terminal mode.[1] The sample code in this document is using terminal mode, so N = 3.

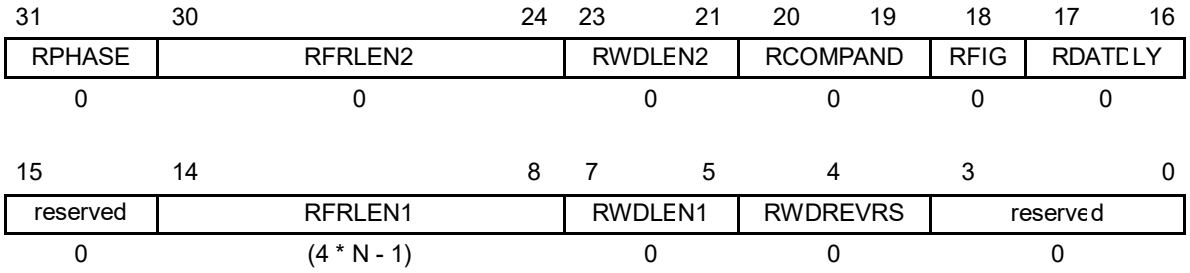

Figure 5. Receive Control Register (RCR)

Figure 6. Transmit Control Register (XCR)

As shown in Figure 7, the serial port is configured and initialized via the serial-port control register (SPCR). FREE is set to 1 to enable free running of serial port clocks when emulation halt occurs.

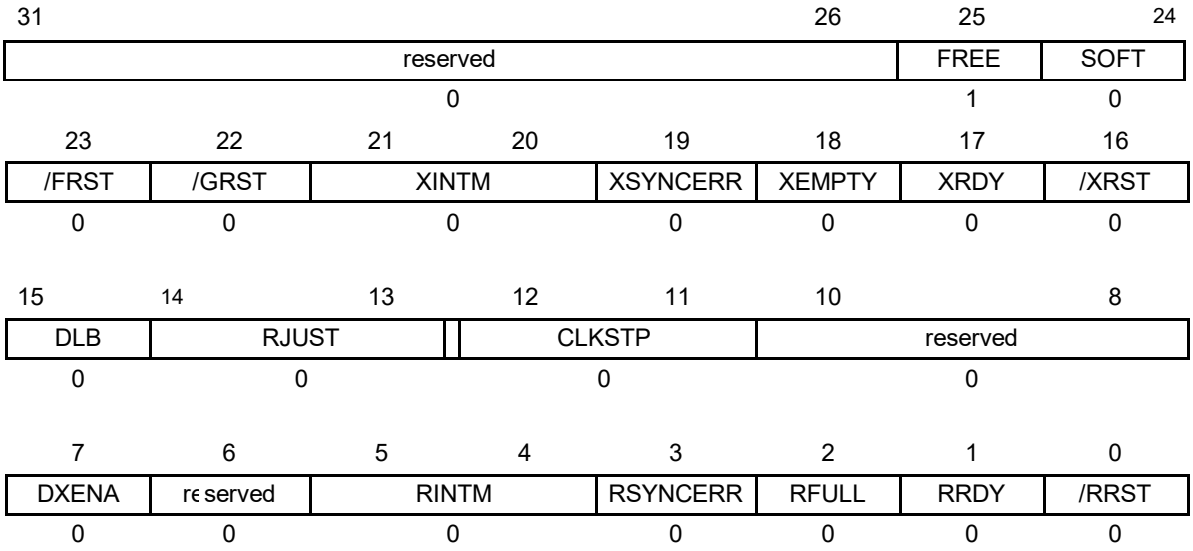

Figure 7. Serial Port Control Register (SPCR)

5 McBSP Initialization

Most applications require the DMA (Direct Memory Access) or EDMA (Enhanced DMA) to service the McBSP. In that case, the initialization steps must be followed in this order:

1. Enable interrupts by setting the Global Interrupt Enable (GIE) and Non-Maskable Interrupt Enable (NMIE) bits in the IER.
2. Configure DMA or EDMA as follows:
   a. If the DMA (TMS320C620x/C670x only) is used to perform data transfers, select and configure the DMA channel you want to use to service the McBSP. Enable CPU interrupts that correspond to the DMA channel. The default mapping of DMA channel-complete interrupts to CPU is as follows:

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

   Typical settings for the DMA channel would be as follows:

   – Source address = DRR for receive channel OR memory location for transmit buffer.
   – Destination address = memory location for receive buffer OR DXR for transmit channel.
   – Transfer counter = number of elements to be transferred.
   – Receive synchronization event, (R/W)SYNC = REVT from McBSP.
   – Transmit synchronization event, (R/W)SYNC = XEVT from McBSP.
   – DMA channel complete interrupt bit, TCINT = enabled.
   – Priority bit, PRI = 1; optional but recommended.

b. If the EDMA (TMS320C621x/C671x/C64x only) is used to perform data transfers, select and configure the EDMA channel you want to use to service the McBSP. Enable CPU interrupt that corresponds to the EDMA channels. The default mapping of EDMA channel-complete interrupt to CPU is 8.

The parameters for EDMA channel would be as follows:

- Source address = DRR for receive channel OR memory location for transmit buffer.
- Destination address = memory location for receive buffer OR DXR for transmit channel.
- Transfer counter = number of elements to be transferred.
- Transfer complete code = EDMA channel number.
- EDMA channel complete interrupt bit, TCINT = enabled.
- Priority bit, PRI = 1; optional but recommended.

3. Start DMA or EDMA.
4. Initialize all McBSP registers as described in the previous sections.
5. Take the sample rate generator out of reset by setting/GRST = 1 in SPCR.
6. Wake up the McBSP as transmitter and/or receiver by setting /XRST = 1 and/or /RRST = 1 in SPCR, respectively. Make sure the slave wakes up before the master.

The McBSP is now ready for transmit and receive as soon as it receives the external clocks and frame syncs.

6 Sample Code Setup

```
/*****************************************************************/
/* Date Created: 05/22/2001                                      */
/* Date Last Modified: 07/12/2001                                */
/* Source File: iom-2.c                                          */
/* Original Author: Eric Biscondi                                */
/* Author: Scott Chen                                            */
/*                                                               */
/* This code describes how to initialize the C6000 McBSP to      */
/* communicate with an IOM-2-compliant device operating in the   */
/* line-card mode.  On 6x0x devices, DMA channels 0 and 1 are used */
/* to service McBSP 0 transmit and receive operations,           */
/* respectively.  On 6x1x/64x devices, EDMA channels 12 and 13 are */
/* used to service McBSP 0 transmit and receive operations,      */
/* respectively.                                                 */
/*                                                               */
/* For this example, consider the terminal mode operation (each  */
/* frame contains three sub-frames of 4 bytes).  To simplify the */
/* example, a single frame is transferred/received.              */
/*                                                               */
/* Also in this code, McBSP 1 is configured as an IOM-2-Compliant */
/* device.  '_iom2' has been tagged onto items pertaining to IOM-2.*/
/* For 6x0x devices, DMA channels 2 and 3 are used to service    */
/* McBSP 1, while for 6x1x/64x devices, EDMA channel 14 and 15 are */
/* used.  This program has been simulated and verified for       */
```

```
/* functionality on C6711, C6202 and C6203 devices.                */
/*                                                                 */
/* This program is based on CSL 2.0.  Please refer to the          */
/* TMS320C6000 Chip Support Library API User's Guide for further   */
/* information.                                                    */
/*******************************************************************/
/* Chip definition - Please change this accordingly */
#define CHIP_6711 1
/* Include files */
#include <csl.h>        /* CSL Library   */
#include <csl_edma.h>   /* EDMA_Support  */
#include <csl_dma.h>    /* DMA_Support     */
#include <csl_irq.h>    /* IRQ_Support   */
#include <csl_mcbsp.h>  /* MCBSP_Support */

/* Define Constants */
#define FALSE 0
#define TRUE 1
#define N 3   /*set up the number of sub-frames */
#define BUFFER_SIZE 256
#define XFER_SIZE 9

/* Declare CSL objects */
MCBSP_Handle hMcbsp0;     /* Handles for McBSP */
MCBSP_Handle hMcbsp1_iom2;
#if (EDMA_SUPPORT)
      EDMA_Handle hEdma1;           /* Handles for EDMA */
      EDMA_Handle hEdma2;
      EDMA_Handle hEdma3_iom2;
      EDMA_Handle hEdma4_iom2;
      EDMA_Handle hEdmadummy;
#endif
#if (DMA_SUPPORT)
      DMA_Handle hDma0;       /* Handles for DMA */
      DMA_Handle hDma1;
      DMA_Handle hDma2_iom2;
      DMA_Handle hDma3_iom2;
#endif

/* Function Prototypes */
void InitMcBSP0(void);
void InitMcBSP1_iom2(void);
void config_edma1(void);
void config_edma2_iom2(void);
void config_dma1(void);
void config_dma2_iom2(void);
void init_data(void);
interrupt void c_int12(void);
interrupt void c_int11(void);
interrupt void c_int09(void);
interrupt void c_int08(void);

/* Include the vector table to call the IRQ ISRs hookup */
```

```
extern far void vectors();

/* Declaration of Global Variables */
volatile int receive_done  = FALSE;
volatile int transmit_done = FALSE;
volatile int receive_done_iom2  = FALSE;
volatile int transmit_done_iom2 = FALSE;
static Uint32 TransmitBuffer[BUFFER_SIZE];      /* transmit buffer for EDMA */
static Uint32 ReceiveBuffer[BUFFER_SIZE]; /* receive buffer for EDMA  */
static Uint32 TransmitBuffer_iom2[BUFFER_SIZE]; /* transmit buffer for EDMA */
static Uint32 ReceiveBuffer_iom2[BUFFER_SIZE];  /* receive buffer for EDMA  */


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

  int waittime = 0;

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

  /* enable NMI and GIE*/
  IRQ_nmiEnable();
  IRQ_globalEnable();

  /* point to the IRQ vector table */
  IRQ_setVecs(vectors);

  #if (EDMA_SUPPORT)
      /* disable and clear the event interrupt */
      IRQ_reset(IRQ_EVT_EDMAINT);

      /* clear Parameter RAM of EDMA */
      EDMA_clearPram(0x00000000);
  #endif

  #if (DMA_SUPPORT)
      DMA_reset(INV);
  #endif

  /* initialize the data for transmit and receive */
  init_data();

  #if (EDMA_SUPPORT)
      /* let's open up required EDMA channels */
      hEdma1 = EDMA_open(EDMA_CHA_REVT0, EDMA_OPEN_RESET);
      hEdma2 = EDMA_open(EDMA_CHA_XEVT0, EDMA_OPEN_RESET);
      hEdma3_iom2 = EDMA_open(EDMA_CHA_REVT1, EDMA_OPEN_RESET);
      hEdma4_iom2 = EDMA_open(EDMA_CHA_XEVT1, EDMA_OPEN_RESET);
  #endif
```

```
#if (DMA_SUPPORT)
    hDma0 = DMA_open(DMA_CHA0, DMA_OPEN_RESET);
    hDma1 = DMA_open(DMA_CHA1, DMA_OPEN_RESET);
    hDma2_iom2 = DMA_open(DMA_CHA2, DMA_OPEN_RESET);
    hDma3_iom2 = DMA_open(DMA_CHA3, DMA_OPEN_RESET);
#endif

/* let's open up required McBSP channels */
hMcbsp0 = MCBSP_open(MCBSP_DEV0, MCBSP_OPEN_RESET);      /*McBSP Port 0 */
hMcbsp1_iom2 = MCBSP_open(MCBSP_DEV1, MCBSP_OPEN_RESET); /*McBSP Port 1 */

#if (EDMA_SUPPORT)
    /* configure the EDMA channels */
    config_edma1();
    config_edma2_iom2();

    /* enable EDMA-CPU interrupt tied to McBSP */
    IRQ_enable(IRQ_EVT_EDMAINT);

    /* enable EDMA channel interrupt to CPU */
    EDMA_intEnable(12);
    EDMA_intEnable(13);
    EDMA_intEnable(14);     /* IOM-2 */
    EDMA_intEnable(15);     /* IOM-2 */

    /* Enable EDMA channels */
    EDMA_enableChannel(hEdma1);
    EDMA_enableChannel(hEdma2);
    EDMA_enableChannel(hEdma3_iom2);
    EDMA_enableChannel(hEdma4_iom2);
#endif

#if (DMA_SUPPORT)
    /* Configure the DMA Channels */
    config_dma1();
    config_dma2_iom2();

    IRQ_disable(IRQ_EVT_DMAINT0);
    IRQ_disable(IRQ_EVT_DMAINT1);
    IRQ_disable(IRQ_EVT_DMAINT2);
    IRQ_disable(IRQ_EVT_DMAINT3);

    IRQ_clear(IRQ_EVT_DMAINT0);
    IRQ_clear(IRQ_EVT_DMAINT1);
    IRQ_clear(IRQ_EVT_DMAINT2);
    IRQ_clear(IRQ_EVT_DMAINT3);

    IRQ_enable(IRQ_EVT_DMAINT0);
    IRQ_enable(IRQ_EVT_DMAINT1);
    IRQ_enable(IRQ_EVT_DMAINT2);
    IRQ_enable(IRQ_EVT_DMAINT3);
```

```
    DMA_start(hDma0);
    DMA_start(hDma1);
    DMA_start(hDma2_iom2);
    DMA_start(hDma3_iom2);
#endif

/* Set up McBSP ports */
InitMcBSP0();
InitMcBSP1_iom2();

/* Start Sample Rate Generator: set /GRST = 1 */
MCBSP_enableSrgr(hMcbsp0);
MCBSP_enableSrgr(hMcbsp1_iom2);   /* IOM-2 */

/* inserted wait time for McBSP to get ready */
for (waittime=0; waittime<0xF; waittime++);

/* Wake up the McBSP as transmitter and receiver, slave first  */
/* This set XRST = 1 and RRST = 1.                              */
MCBSP_enableRcv(hMcbsp0);        /* Enable McBSP port 0 as receiver   */
MCBSP_enableXmt(hMcbsp1_iom2); /* Enable McBSP port 1 as trasmitter */
MCBSP_enableRcv(hMcbsp1_iom2); /* Enable McBSP port 1 as receiver*/
MCBSP_enableXmt(hMcbsp0);      /* Enable McBSP port 0 as trasmitter */

/* Enable Frame Sync Generator for McBSP 1: set /FRST = 1 */
MCBSP_enableFsync(hMcbsp1_iom2);

/* To flag an interrupt to the CPU when EDMA transfer/receive is done */
while (!transmit_done || !receive_done
       || !transmit_done_iom2 || !receive_done_iom2);

#if (EDMA_SUPPORT)
    IRQ_disable(IRQ_EVT_EDMAINT);
    EDMA_RSET(CIER, 0x0);
#endif

#if (DMA_SUPPORT)
    IRQ_disable(IRQ_EVT_DMAINT0);
    IRQ_disable(IRQ_EVT_DMAINT1);
    IRQ_disable(IRQ_EVT_DMAINT2);
    IRQ_disable(IRQ_EVT_DMAINT3);
#endif

MCBSP_close(hMcbsp0);              /* close McBSP port 0    */
MCBSP_close(hMcbsp1_iom2);        /* close McBSP port 1    */

#if (EDMA_SUPPORT)
    EDMA_close(hEdma1);             /* close EDMA channel 1  */
    EDMA_close(hEdma2);             /* close EDMA channel 2  */
    EDMA_close(hEdma3_iom2);         /* close EDMA channel 3  */
    EDMA_close(hEdma4_iom2);         /* close EDMA channel 4  */
#endif
```

```
  #if (DMA_SUPPORT)
      DMA_close(hDma0);                   /* close DMA channel 0 */
      DMA_close(hDma1);                   /* close DMA channel 0 */
      DMA_close(hDma2_iom2);              /* close DMA channel 0 */
      DMA_close(hDma3_iom2);              /* close DMA channel 0 */
  #endif

}  /***** End of Main() *****/
/*********************************************************************/
/* init_data()  :  Initialize data for transmit and receive          */
/*********************************************************************/
void init_data(void)
{
  volatile Uint32 i=0;

  for (i = 0; i < XFER_SIZE; i=i+1){
      TransmitBuffer[i]      = 0xA2B000B0+i;
      TransmitBuffer_iom2[i] = 0xB2A000A0+i;
      ReceiveBuffer[i]       = 0xFFFFFFFF;
      ReceiveBuffer_iom2[i]  = 0xFFFFFFFF;
  }
} /* end of init_data() */

/*********************************************************************/
/* InitMcBSP0() :   MCBSP Port 0 Config structure                    */
/*********************************************************************/
void InitMcBSP0(void)
{
  MCBSP_Config mcbspCfg0 = {

      /* SPCR Setup */
      #if (DMA_SUPPORT)
            MCBSP_SPCR_RMK(
                  MCBSP_SPCR_FRST_DEFAULT,
                  MCBSP_SPCR_GRST_DEFAULT,
                  MCBSP_SPCR_XINTM_DEFAULT,
                  MCBSP_SPCR_XSYNCERR_NO,
                  MCBSP_SPCR_XRST_DEFAULT,
                  MCBSP_SPCR_DLB_OFF,
                  MCBSP_SPCR_RJUST_RZF,
                  MCBSP_SPCR_CLKSTP_DISABLE,
                  MCBSP_SPCR_RINTM_DEFAULT,
                  MCBSP_SPCR_RSYNCERR_NO,   /* a new frame synchronization */
                  MCBSP_SPCR_RRST_DEFAULT
                  ),
      #endif
      #if (EDMA_SUPPORT)
            MCBSP_SPCR_RMK(
                  MCBSP_SPCR_FREE_YES,    /* Only in C621x/C671x/C64x    */
                  MCBSP_SPCR_SOFT_DEFAULT, /* Only in C621x/C671x/C64x    */
                  MCBSP_SPCR_FRST_DEFAULT,
                  MCBSP_SPCR_GRST_DEFAULT,
                  MCBSP_SPCR_XINTM_DEFAULT,
```

```
            MCBSP_SPCR_XSYNCERR_NO,
            MCBSP_SPCR_XRST_DEFAULT,
            MCBSP_SPCR_DLB_OFF,
            MCBSP_SPCR_RJUST_RZF,
            MCBSP_SPCR_CLKSTP_DISABLE,
            MCBSP_SPCR_DXENA_OFF,   /* Only in C621x/C671x/C64x    */
            MCBSP_SPCR_RINTM_DEFAULT,
            MCBSP_SPCR_RSYNCERR_NO,  /* a new frame synchronization */
            MCBSP_SPCR_RRST_DEFAULT
            ),
#endif

/* RCR Setup */
#if (DMA_SUPPORT)
      MCBSP_RCR_RMK(
            MCBSP_RCR_RPHASE_SINGLE,      /* single-phase frame */
            MCBSP_RCR_RFRLEN2_DEFAULT,
            MCBSP_RCR_RWDLEN2_DEFAULT,
            MCBSP_RCR_RCOMPAND_DEFAULT,
            MCBSP_RCR_RFIG_NO,
            MCBSP_RCR_RDATDLY_DEFAULT,
            MCBSP_RCR_RFRLEN1_OF(4*N-1),/* user defines N at the top */
            MCBSP_RCR_RWDLEN1_8BIT        /* element length is 8 bits */
            ),
#endif
#if (EDMA_SUPPORT)
      MCBSP_RCR_RMK(
            MCBSP_RCR_RPHASE_SINGLE,      /* single-phase frame */
            MCBSP_RCR_RFRLEN2_DEFAULT,
            MCBSP_RCR_RWDLEN2_DEFAULT,
            MCBSP_RCR_RCOMPAND_DEFAULT,
            MCBSP_RCR_RFIG_NO,
            MCBSP_RCR_RDATDLY_DEFAULT,
            MCBSP_RCR_RFRLEN1_OF(4*N-1),/* user defines N at the top */
            MCBSP_RCR_RWDLEN1_8BIT,  /*  element  length  is  8  bits  */
            MCBSP_RCR_RWDREVRS_DISABLE /* Only in C6x1x/C64x */
            ),
#endif

/* XCR Setup */
#if (DMA_SUPPORT)
      MCBSP_XCR_RMK(
            MCBSP_XCR_XPHASE_SINGLE,      /* single-phase frame */
            MCBSP_XCR_XFRLEN2_DEFAULT,
            MCBSP_XCR_XWDLEN2_DEFAULT,
            MCBSP_XCR_XCOMPAND_DEFAULT,
            MCBSP_XCR_XFIG_NO,
            MCBSP_XCR_XDATDLY_DEFAULT,
            MCBSP_XCR_XFRLEN1_OF(4*N-1), /* user defines N at the top */
            MCBSP_XCR_XWDLEN1_8BIT      /* element length is 8 bits */
            ),
#endif
#if (EDMA_SUPPORT)
```

```
      MCBSP_XCR_RMK(
            MCBSP_XCR_XPHASE_SINGLE,      /* single-phase frame */
            MCBSP_XCR_XFRLEN2_DEFAULT,
            MCBSP_XCR_XWDLEN2_DEFAULT,
            MCBSP_XCR_XCOMPAND_DEFAULT,
            MCBSP_XCR_XFIG_NO,
            MCBSP_XCR_XDATDLY_DEFAULT,
            MCBSP_XCR_XFRLEN1_OF(4*N-1), /* user defines N at the top */
            MCBSP_XCR_XWDLEN1_8BIT,  /*  element  length  is  8  bits  */
            MCBSP_XCR_XWDREVRS_DISABLE /* Only in C6x1x/C64x */
                  ),
#endif

/* SRGR Setup */
MCBSP_SRGR_RMK(
      MCBSP_SRGR_GSYNC_FREE,
      MCBSP_SRGR_CLKSP_RISING,
      MCBSP_SRGR_CLKSM_CLKS,
      MCBSP_SRGR_FSGM_DEFAULT,
      MCBSP_SRGR_FPER_DEFAULT,
      MCBSP_SRGR_FWID_DEFAULT,
      MCBSP_SRGR_CLKGDV_OF(1)       /* divide clock by 2 */
      ),

/* MCR Setup */
MCBSP_MCR_DEFAULT,


/* RCER Setup */
#if (C64_SUPPORT)
      MCBSP_RCERE0_DEFAULT,
      MCBSP_RCERE1_DEFAULT,
      MCBSP_RCERE2_DEFAULT,
#else
      MCBSP_RCER_DEFAULT,
#endif

/* XCER Setup */
#if (C64_SUPPORT)
      MCBSP_XCERE0_DEFAULT,
      MCBSP_XCERE1_DEFAULT,
      MCBSP_XCERE2_DEFAULT,
#else
      MCBSP_XCER_DEFAULT,
#endif

/* PCR Setup */
MCBSP_PCR_RMK(
      MCBSP_PCR_XIOEN_SP,
      MCBSP_PCR_RIOEN_SP,
      MCBSP_PCR_FSXM_EXTERNAL,      /* FSXM = 0 */
      MCBSP_PCR_FSRM_EXTERNAL,      /* FSRM = 0 */
      MCBSP_PCR_CLKXM_OUTPUT,       /* clock mode from internal SRGR */
```

```
            MCBSP_PCR_CLKRM_OUTPUT,       /* clock mode from internal SRGR */
            MCBSP_PCR_CLKSSTAT_0,
            MCBSP_PCR_DXSTAT_0,
            MCBSP_PCR_FSXP_ACTIVEHIGH, /* active high frame sync.polarity */
            MCBSP_PCR_FSRP_ACTIVEHIGH, /* active high frame sync.polarity */
            MCBSP_PCR_CLKXP_RISING, /* clock polarity from rising edge */
            MCBSP_PCR_CLKRP_FALLING /* clock polarity from falling edge */
            ),
  };

  MCBSP_config(hMcbsp0, &mcbspCfg0);

}  /* end of InitMcBSP0(void) */

/*****************************************************************/
/* InitMcBSP1_iom2() :   MCBSP Port 1 Config structure           */
/*****************************************************************/
void InitMcBSP1_iom2(void)
{

  MCBSP_Config mcbspCfg_iom2 = {

      /* SPCR Setup */
      #if (DMA_SUPPORT)
            MCBSP_SPCR_RMK(
                  MCBSP_SPCR_FRST_DEFAULT,
                  MCBSP_SPCR_GRST_DEFAULT,
                  MCBSP_SPCR_XINTM_DEFAULT,
                  MCBSP_SPCR_XSYNCERR_NO,
                  MCBSP_SPCR_XRST_DEFAULT,
                  MCBSP_SPCR_DLB_OFF,
                  MCBSP_SPCR_RJUST_RZF,
                  MCBSP_SPCR_CLKSTP_DISABLE,
                  MCBSP_SPCR_RINTM_DEFAULT,
                  MCBSP_SPCR_RSYNCERR_NO,   /* a new frame synchronization */
                  MCBSP_SPCR_RRST_DEFAULT
                  ),
      #endif
      #if (EDMA_SUPPORT)
            MCBSP_SPCR_RMK(
                  MCBSP_SPCR_FREE_YES,     /* Only in C621x/C671x/C64x    */
                  MCBSP_SPCR_SOFT_DEFAULT,  /* Only in C621x/C671x/C64x    */
                  MCBSP_SPCR_FRST_DEFAULT,
                  MCBSP_SPCR_GRST_DEFAULT,
                  MCBSP_SPCR_XINTM_DEFAULT,
                  MCBSP_SPCR_XSYNCERR_NO,
                  MCBSP_SPCR_XRST_DEFAULT,
                  MCBSP_SPCR_DLB_OFF,
                  MCBSP_SPCR_RJUST_RZF,
                  MCBSP_SPCR_CLKSTP_DISABLE,
                  MCBSP_SPCR_DXENA_OFF,    /* Only in C621x/C671x/C64x    */
                  MCBSP_SPCR_RINTM_DEFAULT,
                  MCBSP_SPCR_RSYNCERR_NO,   /* a new frame synchronization */
```

```
            MCBSP_SPCR_RRST_DEFAULT
            ),
#endif

/* RCR Setup */
#if (DMA_SUPPORT)
      MCBSP_RCR_RMK(
            MCBSP_RCR_RPHASE_SINGLE,      /* single-phase frame */
            MCBSP_RCR_RFRLEN2_DEFAULT,
            MCBSP_RCR_RWDLEN2_DEFAULT,
            MCBSP_RCR_RCOMPAND_DEFAULT,
            MCBSP_RCR_RFIG_NO,
            MCBSP_RCR_RDATDLY_DEFAULT,
            MCBSP_RCR_RFRLEN1_OF(4*N-1), /* user defines N at the top */
            MCBSP_RCR_RWDLEN1_8BIT      /* element length is 32 bits */
            ),
#endif
#if (EDMA_SUPPORT)
      MCBSP_RCR_RMK(
            MCBSP_RCR_RPHASE_SINGLE,      /* single-phase frame */
            MCBSP_RCR_RFRLEN2_DEFAULT,
            MCBSP_RCR_RWDLEN2_DEFAULT,
            MCBSP_RCR_RCOMPAND_DEFAULT,
            MCBSP_RCR_RFIG_NO,
            MCBSP_RCR_RDATDLY_DEFAULT,
            MCBSP_RCR_RFRLEN1_OF(4*N-1), /* user defines N at the top */
            MCBSP_RCR_RWDLEN1_8BIT,  /*  element  length  is  32  bits  */
            MCBSP_RCR_RWDREVRS_DISABLE /* Only in C6x1x/C64x */
            ),
#endif

/* XCR Setup */
#if (DMA_SUPPORT)
      MCBSP_XCR_RMK(
            MCBSP_XCR_XPHASE_SINGLE,      /* single-phase frame */
            MCBSP_XCR_XFRLEN2_DEFAULT,
            MCBSP_XCR_XWDLEN2_DEFAULT,
            MCBSP_XCR_XCOMPAND_DEFAULT,
            MCBSP_XCR_XFIG_NO,
            MCBSP_XCR_XDATDLY_DEFAULT,
            MCBSP_XCR_XFRLEN1_OF(4*N-1), /* user defines N at the top */
            MCBSP_XCR_XWDLEN1_8BIT      /* element length is 8 bits  */
            ),
#endif
#if (EDMA_SUPPORT)
      MCBSP_XCR_RMK(
            MCBSP_XCR_XPHASE_SINGLE,      /* single-phase frame */
            MCBSP_XCR_XFRLEN2_DEFAULT,
            MCBSP_XCR_XWDLEN2_DEFAULT,
            MCBSP_XCR_XCOMPAND_DEFAULT,
            MCBSP_XCR_XFIG_NO,
            MCBSP_XCR_XDATDLY_DEFAULT,
            MCBSP_XCR_XFRLEN1_OF(4*N-1),  /* user defines N at the top
```

```
*/
                MCBSP_XCR_XWDLEN1_8BIT,      /* element length is 8 bits  */
                MCBSP_XCR_XWDREVRS_DISABLE   /* Only in C6x1x/C64x        */
                      ),
      #endif

      /* SRGR Setup */
      MCBSP_SRGR_RMK(
            MCBSP_SRGR_GSYNC_FREE,
            MCBSP_SRGR_CLKSP_RISING,
            MCBSP_SRGR_CLKSM_CLKS,
            MCBSP_SRGR_FSGM_FSG,
            MCBSP_SRGR_FPER_OF(32*N-1),         /* 32-bit per element   */
            MCBSP_SRGR_FWID_OF(2),              /* fram sync width = 3  */
            MCBSP_SRGR_CLKGDV_OF(1)             /* divide clock by 2    */
            ),

      /* MCR Setup */
      MCBSP_MCR_DEFAULT,

      /* RCER Setup */
      #if (C64_SUPPORT)
            MCBSP_RCERE0_DEFAULT,
            MCBSP_RCERE1_DEFAULT,
            MCBSP_RCERE2_DEFAULT,
      #else
            MCBSP_RCER_DEFAULT,
      #endif

      /* XCER Setup */
      #if (C64_SUPPORT)
            MCBSP_XCERE0_DEFAULT,
            MCBSP_XCERE1_DEFAULT,
            MCBSP_XCERE2_DEFAULT,
      #else
            MCBSP_XCER_DEFAULT,
      #endif

      /* PCR Setup */
      MCBSP_PCR_RMK(
            MCBSP_PCR_XIOEN_SP,
            MCBSP_PCR_RIOEN_SP,
            MCBSP_PCR_FSXM_INTERNAL,      /* FSXM = 1 */
            MCBSP_PCR_FSRM_EXTERNAL,      /* FSXM = 0 */
            MCBSP_PCR_CLKXM_OUTPUT,  /*  clock  mode  from  internal  SRGR  */
            MCBSP_PCR_CLKRM_OUTPUT,  /*  clock  mode  from  internal  SRGR  */
            MCBSP_PCR_CLKSSTAT_0,
            MCBSP_PCR_DXSTAT_0,
            MCBSP_PCR_FSXP_ACTIVEHIGH, /* active high frame sync. polarity */
            MCBSP_PCR_FSRP_ACTIVEHIGH, /* active high frame sync. polarity */
            MCBSP_PCR_CLKXP_RISING,  /*  clock  polarity  from  rising  edge  */
            MCBSP_PCR_CLKRP_FALLING /* clock polarity from falling edge */
            ),
```

```
  };

  MCBSP_config(hMcbsp1_iom2, &mcbspCfg_iom2);
}  /* end of InitMcBSP1(void) */

/*********************************************************************/
/* config_edma1() EDMA Channels (12&13) Configuration Structure    */
/*********************************************************************/
#if (EDMA_SUPPORT)
void config_edma1(void)
{

   EDMA_configArgs(hEdma1,

      /* OPT Setup */
      #if (C64_SUPPORT)
            EDMA_OPT_RMK(
                  EDMA_OPT_PRI_HIGH,       /* EDMA high priority  */
                  EDMA_OPT_ESIZE_8BIT,    /* Element size 8 bits */
                  EDMA_OPT_2DS_NO,
                  EDMA_OPT_SUM_NONE,
                  EDMA_OPT_2DD_NO,
                  EDMA_OPT_DUM_INC,  /*  Dest  increment  by  element  size  */
                  EDMA_OPT_TCINT_YES,   /*   Enable   Transfer   Interrupt   */
                  EDMA_OPT_TCC_OF(13),    /* TCCINT = 0xD, REVT0            */
                  EDMA_OPT_TCCM_DEFAULT,  /* only in C64x                  */
                  EDMA_OPT_ATCINT_DEFAULT,/* only in C64x                  */
                  EDMA_OPT_ATCC_DEFAULT,  /* only in C64x                  */
                  EDMA_OPT_PDTS_DEFAULT,  /* only in C64x                  */
                  EDMA_OPT_PDTD_DEFAULT,  /* only in C64x                  */
                  EDMA_OPT_LINK_YES,      /* Enable linking to NULL table */
                  EDMA_OPT_FS_NO
                  ),
      #else
            EDMA_OPT_RMK(
                  EDMA_OPT_PRI_HIGH,       /* EDMA high priority  */
                  EDMA_OPT_ESIZE_8BIT,    /* Element size 8 bits */
                  EDMA_OPT_2DS_NO,
                  EDMA_OPT_SUM_NONE,
                  EDMA_OPT_2DD_NO,
                  EDMA_OPT_DUM_INC,  /*  Dest  increment  by  element  size  */
                  EDMA_OPT_TCINT_YES,   /*   Enable   Transfer   Interrupt   */
                  EDMA_OPT_TCC_OF(13),    /* TCCINT = 0xD, REVT0            */
                  EDMA_OPT_LINK_YES,      /* Enable linking to NULL table   */
                  EDMA_OPT_FS_NO
                  ),
      #endif

      /* SRC Setup */
      EDMA_SRC_RMK(MCBSP_getRcvAddr(hMcbsp0)),

      /* CNT Setup */
      EDMA_CNT_RMK(
```

```
            EDMA_CNT_FRMCNT_DEFAULT,
            EDMA_CNT_ELECNT_OF(4*XFER_SIZE)
            ),

      /* DST Setup */
      EDMA_DST_RMK((Uint32) ReceiveBuffer),

      /* IDX Setup */
      EDMA_IDX_RMK(
            EDMA_IDX_FRMIDX_DEFAULT,
            EDMA_IDX_ELEIDX_DEFAULT
            ),

      /* RLD Setup */
      EDMA_RLD_RMK(
            EDMA_RLD_ELERLD_DEFAULT,
            EDMA_RLD_LINK_DEFAULT
            )
);
EDMA_configArgs(hEdma2,
      /* Opt Setup */
      #if (C64_SUPPORT)
            EDMA_OPT_RMK(
                  EDMA_OPT_PRI_HIGH,  /*  High  priority  EDMA  */
                  EDMA_OPT_ESIZE_8BIT, /* Element size 8 bits */
                  EDMA_OPT_2DS_NO,
                  EDMA_OPT_SUM_INC,    /* Source increment by element size */
                  EDMA_OPT_2DD_NO,
                  EDMA_OPT_DUM_NONE,
                  EDMA_OPT_TCINT_YES,     /* Enable Transfer Interrupt     */
                  EDMA_OPT_TCC_OF(12),    /* TCCINT = 0xC, XEVT0           */
                  EDMA_OPT_TCCM_DEFAULT   /* only in C64x                  */
                  EDMA_OPT_ATCINT_DEFAULT,/* only in C64x                */
                  EDMA_OPT_ATCC_DEFAULT,  /* only in C64x                  */
                  EDMA_OPT_PDTS_DEFAULT,  /* only in C64x                  */
                  EDMA_OPT_PDTD_DEFAULT,  /* only in C64x                  */
                  EDMA_OPT_LINK_YES,  /*  Enable  linking  to  NULL  table  */
                  EDMA_OPT_FS_NO
                  ),
      #else
            EDMA_OPT_RMK(
                  EDMA_OPT_PRI_HIGH,  /*  High  priority  EDMA  */
                  EDMA_OPT_ESIZE_8BIT,  /*  Element  size  8  bits  */
                  EDMA_OPT_2DS_NO,
                  EDMA_OPT_SUM_INC,     /* Source increment by element size */
                  EDMA_OPT_2DD_NO,
                  EDMA_OPT_DUM_NONE,
                  EDMA_OPT_TCINT_YES, /* Enable Transfer Complete Interrupt */
                  EDMA_OPT_TCC_OF(12), /* TCCINT = 0xC, XEVT0              */
                  EDMA_OPT_LINK_YES,   /* Enable linking to NULL table     */
                  EDMA_OPT_FS_NO
                  ),
      #endif
```

```
      /* SRC Setup */
      EDMA_SRC_RMK((Uint32) TransmitBuffer),

      /* CNT Setup */
      EDMA_CNT_RMK(
            EDMA_CNT_FRMCNT_DEFAULT,
            EDMA_CNT_ELECNT_OF(4*XFER_SIZE)
            ),

      /* DST Setup */
      EDMA_DST_RMK(MCBSP_getXmtAddr(hMcbsp0)),

      /* IDX Setup */
      EDMA_IDX_RMK(
            EDMA_IDX_FRMIDX_DEFAULT,
            EDMA_IDX_ELEIDX_DEFAULT
            ),

      /* RLD Setup */
      EDMA_RLD_RMK(
            EDMA_RLD_ELERLD_DEFAULT,
            EDMA_RLD_LINK_DEFAULT
            )
  );

  hEdmadummy = EDMA_allocTable(-1); /* Dynamically allocates PaRAM RAM table
*/
  EDMA_configArgs(hEdmadummy, /* Dummy or Terminating Table in PaRAM    */
      0x00000000,
      0x00000000,
      0x00000000,             /* Terminate EDMA transfers by linking to */
      0x00000000,             /* this NULL table  */
      0x00000000,
      0x00000000
      );

      EDMA_link(hEdma1, hEdmadummy); /* Link terminating event */
      EDMA_link(hEdma2, hEdmadummy);  /* to the EDMA event      */

}  /* end of config_edma1() */
#endif

/*****************************************************************/
/* config_edma2_iom2() EDMA Channels(14&15) Configuration Structure*/
/*****************************************************************/
#if (EDMA_SUPPORT)
void config_edma2_iom2(void)
{

  EDMA_configArgs(hEdma3_iom2,

      /* OPT Setup */
```

```
#if (C64_SUPPORT)
      EDMA_OPT_RMK(
            EDMA_OPT_PRI_HIGH,       /* EDMA high priority  */
            EDMA_OPT_ESIZE_8BIT,    /* Element size 8 bits */
            EDMA_OPT_2DS_NO,
            EDMA_OPT_SUM_NONE,
            EDMA_OPT_2DD_NO,
            EDMA_OPT_DUM_INC,  /*  Dest  increment  by  element  size*/
            EDMA_OPT_TCINT_YES,   /*   Enable   Transfer   Interrupt   */
            EDMA_OPT_TCC_OF(15),    /* TCCINT = 0xF, REVT1           */
            EDMA_OPT_TCCM_DEFAULT,  /* only in C64x                  */
            EDMA_OPT_ATCINT_DEFAULT,/* only in C64x                  */
            EDMA_OPT_ATCC_DEFAULT,  /* only in C64x                  */
            EDMA_OPT_PDTS_DEFAULT,  /* only in C64x                  */
            EDMA_OPT_PDTD_DEFAULT,  /* only in C64x                  */
            EDMA_OPT_LINK_YES,  /*  Enable  linking  to  NULL  table  */
            EDMA_OPT_FS_NO
            ),
#else
      EDMA_OPT_RMK(
            EDMA_OPT_PRI_HIGH,       /* EDMA high priority  */
            EDMA_OPT_ESIZE_8BIT,    /* Element size 8 bits */
            EDMA_OPT_2DS_NO,
            EDMA_OPT_SUM_NONE,
            EDMA_OPT_2DD_NO,
            EDMA_OPT_DUM_INC,  /*  Dest  increment  by  element  size  */
            EDMA_OPT_TCINT_YES,   /*   Enable   Transfer   Interrupt   */
            EDMA_OPT_TCC_OF(15),    /* TCCINT = 0xF, REVT1           */
            EDMA_OPT_LINK_YES,      /* Enable linking to NULL table   */
            EDMA_OPT_FS_NO
            ),
#endif

/* SRC Setup */
EDMA_SRC_RMK(MCBSP_getRcvAddr(hMcbsp1_iom2)),

/* CNT Setup */
EDMA_CNT_RMK(
      EDMA_CNT_FRMCNT_DEFAULT,
      EDMA_CNT_ELECNT_OF(4*XFER_SIZE)
      ),

/* DST Setup */
EDMA_DST_RMK((Uint32) ReceiveBuffer_iom2),

/* IDX Setup */
EDMA_IDX_RMK(
      EDMA_IDX_FRMIDX_DEFAULT,
      EDMA_IDX_ELEIDX_DEFAULT
      ),

/* RLD Setup */
EDMA_RLD_RMK(
```

```
            EDMA_RLD_ELERLD_DEFAULT,
            EDMA_RLD_LINK_DEFAULT
            )
);

EDMA_configArgs(hEdma4_iom2,

    /* Opt Setup */
    #if (C64_SUPPORT)
          EDMA_OPT_RMK(
                EDMA_OPT_PRI_HIGH,  /*  High  priority  EDMA  */
                EDMA_OPT_ESIZE_8BIT, /* Element size 8 bits */
                EDMA_OPT_2DS_NO,
                EDMA_OPT_SUM_INC,     /* Source increment by element size */
                EDMA_OPT_2DD_NO,
                EDMA_OPT_DUM_NONE,
                EDMA_OPT_TCINT_YES,     /* Enable Transfer Interrupt    */
                EDMA_OPT_TCC_OF(14),    /* TCCINT = 0xE, XEVT1          */
                EDMA_OPT_TCCM_DEFAULT,  /* only in C64x                 */
                EDMA_OPT_ATCINT_DEFAULT,/* only in C64x                 */
                EDMA_OPT_ATCC_DEFAULT,  /* only in C64x                 */
                EDMA_OPT_PDTS_DEFAULT,  /* only in C64x                 */
                EDMA_OPT_PDTD_DEFAULT,  /* only in C64x                 */
                EDMA_OPT_LINK_YES,  /*  Enable  linking  to  NULL  table  */
                EDMA_OPT_FS_NO
                ),
    #else
          EDMA_OPT_RMK(
                EDMA_OPT_PRI_HIGH,  /*  High  priority  EDMA  */
                EDMA_OPT_ESIZE_8BIT, /* Element size 8 bits */
                EDMA_OPT_2DS_NO,
                EDMA_OPT_SUM_INC,    /* Source increment by element size */
                EDMA_OPT_2DD_NO,
                EDMA_OPT_DUM_NONE,
                EDMA_OPT_TCINT_YES,  /* Enable Transfer Interrupt        */
                EDMA_OPT_TCC_OF(14), /* TCCINT = 0xE, XEVT1              */
                EDMA_OPT_LINK_YES,   /* Enable linking to NULL table     */
                EDMA_OPT_FS_NO
                ),
    #endif

    /* SRC Setup */
    EDMA_SRC_RMK((Uint32) TransmitBuffer_iom2),

    /* CNT Setup */
    EDMA_CNT_RMK(
          EDMA_CNT_FRMCNT_DEFAULT,
          EDMA_CNT_ELECNT_OF(4*XFER_SIZE)
          ),

    /* DST Setup */
    EDMA_DST_RMK(MCBSP_getXmtAddr(hMcbsp1_iom2)),
```

```
        /* IDX Setup */
        EDMA_IDX_RMK(
              EDMA_IDX_FRMIDX_DEFAULT,
              EDMA_IDX_ELEIDX_DEFAULT
              ),

        /* RLD Setup */
        EDMA_RLD_RMK(
              EDMA_RLD_ELERLD_DEFAULT,
              EDMA_RLD_LINK_DEFAULT
              )
    );

    hEdmadummy = EDMA_allocTable(-1);/* Dynamically allocates PaRAM RAM table */
    EDMA_configArgs(hEdmadummy,      /* Dummy or Terminating Table in PaRAM  */
        0x00000000,
        0x00000000,
        0x00000000,             /* Terminate EDMA transfers by */
        0x00000000,             /* linking to this NULL table  */
        0x00000000,
        0x00000000
        );

    EDMA_link(hEdma3_iom2, hEdmadummy);     /* Link terminating event to */
    EDMA_link(hEdma4_iom2, hEdmadummy);     /* the EDMA event            */

}  /* end of config_edma2_iom2() */
#endif

/*********************************************************************/
/* config_dma1_iom2() EDMA Channels(0&1) Configuration Structure   */
/*********************************************************************/
#if (DMA_SUPPORT)
void config_dma1(void)
{

    DMA_configArgs(hDma0,

        /* PRICTL Setup */
        DMA_PRICTL_RMK(
              DMA_PRICTL_DSTRLD_NONE,
              DMA_PRICTL_SRCRLD_NONE,
              DMA_PRICTL_EMOD_HALT,
              DMA_PRICTL_FS_DISABLE,
              DMA_PRICTL_TCINT_ENABLE,
              DMA_PRICTL_PRI_DMA,
              DMA_PRICTL_WSYNC_XEVT0,
              DMA_PRICTL_RSYNC_NONE,
              DMA_PRICTL_INDEX_NA,
              DMA_PRICTL_CNTRLD_NA,
              DMA_PRICTL_SPLIT_DISABLE,
              DMA_PRICTL_ESIZE_8BIT,
              DMA_PRICTL_DSTDIR_NONE,
```

```
            DMA_PRICTL_SRCDIR_INC,
            DMA_PRICTL_START_STOP
            ),

        /* SECCTL Setup */
        DMA_SECCTL_RMK(
            DMA_SECCTL_WSPOL_ACTIVEHIGH,
            DMA_SECCTL_RSPOL_ACTIVEHIGH,
            DMA_SECCTL_FSIG_NORMAL,
            DMA_SECCTL_DMACEN_FRAMECOND,
            DMA_SECCTL_WSYNCCLR_NOTHING,
            DMA_SECCTL_WSYNCSTAT_CLEAR,
            DMA_SECCTL_RSYNCCLR_NOTHING,
            DMA_SECCTL_RSYNCSTAT_CLEAR,
            DMA_SECCTL_WDROPIE_DISABLE,
            DMA_SECCTL_WDROPCOND_CLEAR,
            DMA_SECCTL_RDROPIE_DISABLE,
            DMA_SECCTL_RDROPCOND_CLEAR,
            DMA_SECCTL_BLOCKIE_ENABLE,
            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
            ),

        /* SRC Setup */
        DMA_SRC_RMK((Uint32) TransmitBuffer),

        /* DST Setup */
        DMA_DST_RMK(MCBSP_getXmtAddr(hMcbsp0)),         /*McBSP DXR */

        /* XFRCNT Setup */
        DMA_XFRCNT_RMK(
            DMA_XFRCNT_FRMCNT_DEFAULT,
            DMA_XFRCNT_ELECNT_OF(4*XFER_SIZE)
            )
    );

    DMA_configArgs(hDma1,

        /* PRICTL Setup */
        DMA_PRICTL_RMK(
            DMA_PRICTL_DSTRLD_NONE,
            DMA_PRICTL_SRCRLD_NONE,
            DMA_PRICTL_EMOD_HALT,
            DMA_PRICTL_FS_DISABLE,
            DMA_PRICTL_TCINT_ENABLE,
            DMA_PRICTL_PRI_DMA,
            DMA_PRICTL_WSYNC_NONE,
            DMA_PRICTL_RSYNC_REVT0,
```

```
            DMA_PRICTL_INDEX_NA,
            DMA_PRICTL_CNTRLD_NA,
            DMA_PRICTL_SPLIT_DISABLE,
            DMA_PRICTL_ESIZE_8BIT,
            DMA_PRICTL_DSTDIR_INC,
            DMA_PRICTL_SRCDIR_NONE,
            DMA_PRICTL_START_STOP
            ),

      /* SECCTL Setup */
      DMA_SECCTL_RMK(
            DMA_SECCTL_WSPOL_ACTIVEHIGH,
            DMA_SECCTL_RSPOL_ACTIVEHIGH,
            DMA_SECCTL_FSIG_NORMAL,
            DMA_SECCTL_DMACEN_FRAMECOND,
            DMA_SECCTL_WSYNCCLR_NOTHING,
            DMA_SECCTL_WSYNCSTAT_CLEAR,
            DMA_SECCTL_RSYNCCLR_NOTHING,
            DMA_SECCTL_RSYNCSTAT_CLEAR,
            DMA_SECCTL_WDROPIE_DISABLE,
            DMA_SECCTL_WDROPCOND_CLEAR,
            DMA_SECCTL_RDROPIE_DISABLE,
            DMA_SECCTL_RDROPCOND_CLEAR,
            DMA_SECCTL_BLOCKIE_ENABLE,
            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
            ),

      /* SRC Setup */
      DMA_SRC_RMK(MCBSP_getRcvAddr(hMcbsp0)),         /*McBSP DRR */

      /* DST Setup */
      DMA_DST_RMK((Uint32) ReceiveBuffer),

      /* XFRCNT Setup */
      DMA_XFRCNT_RMK(
            DMA_XFRCNT_FRMCNT_DEFAULT,
            DMA_XFRCNT_ELECNT_OF(4*XFER_SIZE)
            )

  );

}     /* end of config_dma1*/
#endif


/*****************************************************************/
/* config_dma2_iom2() EDMA Channels(2&3) Configuration Structure   */
```

```
/*********************************************************************/
#if (DMA_SUPPORT)
void config_dma2_iom2(void)
{

  DMA_configArgs(hDma2_iom2,

      /* PRICTL Setup */
      DMA_PRICTL_RMK(
            DMA_PRICTL_DSTRLD_NONE,
            DMA_PRICTL_SRCRLD_NONE,
            DMA_PRICTL_EMOD_HALT,
            DMA_PRICTL_FS_DISABLE,
            DMA_PRICTL_TCINT_ENABLE,
            DMA_PRICTL_PRI_DMA,
            DMA_PRICTL_WSYNC_XEVT1,
            DMA_PRICTL_RSYNC_NONE,
            DMA_PRICTL_INDEX_NA,
            DMA_PRICTL_CNTRLD_NA,
            DMA_PRICTL_SPLIT_DISABLE,
            DMA_PRICTL_ESIZE_8BIT,
            DMA_PRICTL_DSTDIR_NONE,
            DMA_PRICTL_SRCDIR_INC,
            DMA_PRICTL_START_STOP
            ),
      /* SECCTL Setup */
      DMA_SECCTL_RMK(
            DMA_SECCTL_WSPOL_ACTIVEHIGH,
            DMA_SECCTL_RSPOL_ACTIVEHIGH,
            DMA_SECCTL_FSIG_NORMAL,
            DMA_SECCTL_DMACEN_FRAMECOND,
            DMA_SECCTL_WSYNCCLR_NOTHING,
            DMA_SECCTL_WSYNCSTAT_CLEAR,
            DMA_SECCTL_RSYNCCLR_NOTHING,
            DMA_SECCTL_RSYNCSTAT_CLEAR,
            DMA_SECCTL_WDROPIE_DISABLE,
            DMA_SECCTL_WDROPCOND_CLEAR,
            DMA_SECCTL_RDROPIE_DISABLE,
            DMA_SECCTL_RDROPCOND_CLEAR,
            DMA_SECCTL_BLOCKIE_ENABLE,
            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
            ),
      /* SRC Setup */
      DMA_SRC_RMK((Uint32) TransmitBuffer_iom2),

      /* DST Setup */
      DMA_DST_RMK(MCBSP_getXmtAddr(hMcbsp1_iom2)),          /*McBSP DXR */
```

```
    /* XFRCNT Setup */
    DMA_XFRCNT_RMK(
          DMA_XFRCNT_FRMCNT_DEFAULT,
          DMA_XFRCNT_ELECNT_OF(4*XFER_SIZE)
          )
);

DMA_configArgs(hDma3_iom2,

    /* PRICTL Setup */
    DMA_PRICTL_RMK(
          DMA_PRICTL_DSTRLD_NONE,
          DMA_PRICTL_SRCRLD_NONE,
          DMA_PRICTL_EMOD_HALT,
          DMA_PRICTL_FS_DISABLE,
          DMA_PRICTL_TCINT_ENABLE,
          DMA_PRICTL_PRI_DMA,
          DMA_PRICTL_WSYNC_NONE,
          DMA_PRICTL_RSYNC_REVT1,
          DMA_PRICTL_INDEX_NA,
          DMA_PRICTL_CNTRLD_NA,
          DMA_PRICTL_SPLIT_DISABLE,
          DMA_PRICTL_ESIZE_8BIT,
          DMA_PRICTL_DSTDIR_INC,
          DMA_PRICTL_SRCDIR_NONE,
          DMA_PRICTL_START_STOP
          ),

    /* SECCTL Setup */
    DMA_SECCTL_RMK(
          DMA_SECCTL_WSPOL_ACTIVEHIGH,
          DMA_SECCTL_RSPOL_ACTIVEHIGH,
          DMA_SECCTL_FSIG_NORMAL,
          DMA_SECCTL_DMACEN_FRAMECOND,
          DMA_SECCTL_WSYNCCLR_NOTHING,
          DMA_SECCTL_WSYNCSTAT_CLEAR,
          DMA_SECCTL_RSYNCCLR_NOTHING,
          DMA_SECCTL_RSYNCSTAT_CLEAR,
          DMA_SECCTL_WDROPIE_DISABLE,
          DMA_SECCTL_WDROPCOND_CLEAR,
          DMA_SECCTL_RDROPIE_DISABLE,
          DMA_SECCTL_RDROPCOND_CLEAR,
          DMA_SECCTL_BLOCKIE_ENABLE,
          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
          ),
```

```
        /* SRC Setup */
         DMA_SRC_RMK(MCBSP_getRcvAddr(hMcbsp1_iom2)),         /*McBSP DRR */

        /* DST Setup */
        DMA_DST_RMK((Uint32) ReceiveBuffer_iom2),

        /* XFRCNT Setup */
        DMA_XFRCNT_RMK(
              DMA_XFRCNT_FRMCNT_DEFAULT,
              DMA_XFRCNT_ELECNT_OF(4*XFER_SIZE)
              )

  );

}     /* end of config_dma2_iom2 */
#endif


/*****************************************************************/
/* EDMA DATA TRANSFER COMPLETION ISRS                            */
/*****************************************************************/

interrupt void c_int08(void) /* EDMA and DMA */
{
  #if (EDMA_SUPPORT)
      if (EDMA_intTest(12))
      {
            /* clear CIPR bit so future interrupts can be recognized */
            EDMA_intClear(12);
            transmit_done = TRUE;
      }
  if (EDMA_intTest(13))
      {
            /* clear CIPR bit so future interrupts can be recognized */
            EDMA_intClear(13);
            receive_done = TRUE;
      }

  if (EDMA_intTest(14))
      {
            /* clear CIPR bit so future interrupts can be recognized */
            EDMA_intClear(14);
            transmit_done_iom2 = TRUE;
      }

  if (EDMA_intTest(15))
      {
            /* clear CIPR bit so future interrupts can be recognized */
            EDMA_intClear(15);
            receive_done_iom2 = TRUE;
      }
  #endif
```

```
  #if (DMA_SUPPORT)
      transmit_done = TRUE;
  #endif
}

interrupt void c_int09(void)  /* DMA 1 */
{
  #if (DMA_SUPPORT)
      receive_done = TRUE;
  #endif
}

interrupt void c_int11(void)  /* DMA 2 - for IOM-2 */
{
  #if (DMA_SUPPORT)
      transmit_done_iom2 = TRUE;
  #endif
}

interrupt void c_int12(void)  /* DMA 3 - for IOM-2 */
{
  #if (DMA_SUPPORT)
      receive_done_iom2 = TRUE;
  #endif
}

/*****************************************************************/
/ End of IOM-2.c                                                 */
/*****************************************************************/
```

7 References

1. "ICs for Communications," *IOM–2 Internal Reference Guide,* Siemens AG, 1991.
2. *IOM–2 Interfacing on TMS320C54x* (BPRA074).
3. *ICs for Communications, ISDN Subscriber Access Controller for Terminals, ISAC–STE, PSB2186 User's Manual*, 1994, Siemans AG.
4. *TMS320C6000 Peripherals Reference Guide* (SPRU190).

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