Using the TMS320C6000 McBSP as a High Speed Communication Port

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

ABSTRACT

This document describes how to use the mulit-channel buffered serial ports (McBSP) in the Texas Instruments (TI) TMS320C6000™ digital signal processor (DSP) as a high-speed data communication port.

One McBSP of one C6000™ DSP device can be connected to a McBSP on another C6000 DSP device to serve as a high-speed data communication port. Typically, McBSPs of similar device numbers are interconnected to achieve inter-communication at high speeds. This application note describes the maximum speed achieved using a similar setup for each of the C6000 devices. This is necessary since timing numbers vary between devices due to the process technology and speed grades. To achieve the maximum data rate, it is necessary to connect the two serial ports such that one device behaves both as a clock master and frame master. The term master refers to a device that generates the required signal (such as clocks and frames). Based on the assumptions made in this application report, the Timing Analysis Section of this report provides a list of the timing constraints and maximum clock and transfer rates of the McBSPs in the C6000 device series. Project collateral discussed in this application report can be downloaded from the following URL: http://www.ti.com/lit/zip/SPRA455.

Contents

List of Figures

List of Tables

1 Design Problem

How can the two multi-channel buffered serial ports (McBSP) in the TMS320C6000 DSP be used as high-speed data communication port?

2 Solution

Either one or more McBSPs of the TMS320C6000 devices can be connected to a McBSP of a different C6000 DSP device to serve as a high-speed data communication port. To achieve the maximum data rate it is necessary to connect the two serial ports so that one device behaves both as a clock master and a frame master. In other words, the McBSP transmitter that generates clocks for data transfer should also generate necessary frame synchronization signals. The other McBSP portion then acts as a slave awaiting these control signals from the master.

Figure 1 shows the block diagram of this arrangement. The transmit portion of McBSP0 of CPU0 is the master of clocks and frames to the receiver in McBSP1 of CPU1. Similarly, the McBSP1 of CPU1 transmitter is configured to be the clock and frame master to McBSP0 of CPU0.

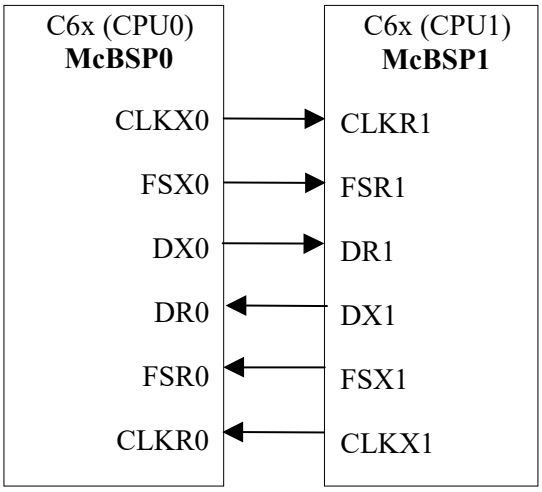

Figure 1. McBSP Connection for Maximum Data Rate

2.1 McBSP Register Configuration

The setup of bit-fields in the control registers for this operation is shown in Figure 2 through Figure 5 and listed in Table 1. Note that in this example the master derives its clock from the CPU clock (CLKOUT1) with a divide ratio of 2 to achieve maximum bit rate. Since the master transmitter is responsible for generating bit clocks and frame synchronization signals, CLKX and FSX are programmed as outputs. The data delay between the FSX output and first data bit has to be a non-zero value, because a data delay of zero does not provide maximum packet frequency. Therefore, both the transmitter and receiver are set for (R/X)DATDLY=1 in this example.

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

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

Figure 2. Receive Control Register (RCR)

31	30 – 24	23 – 21	20 – 19	18	17 – 16
XPHASE	XFRLEN2	XWDLEN2	XCC MPAND	XFIG	XDATDLY
0	0	0	0	0	0

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

Figure 3. Transmit Control Register (XCR)

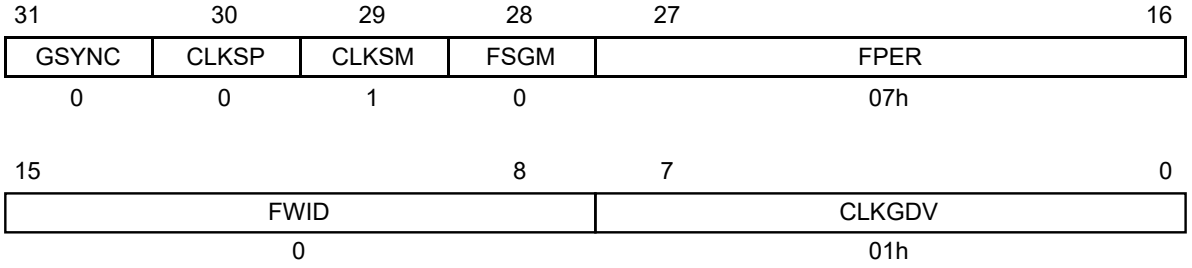

Figure 4. Sample Rate Generator Register (SRGR)

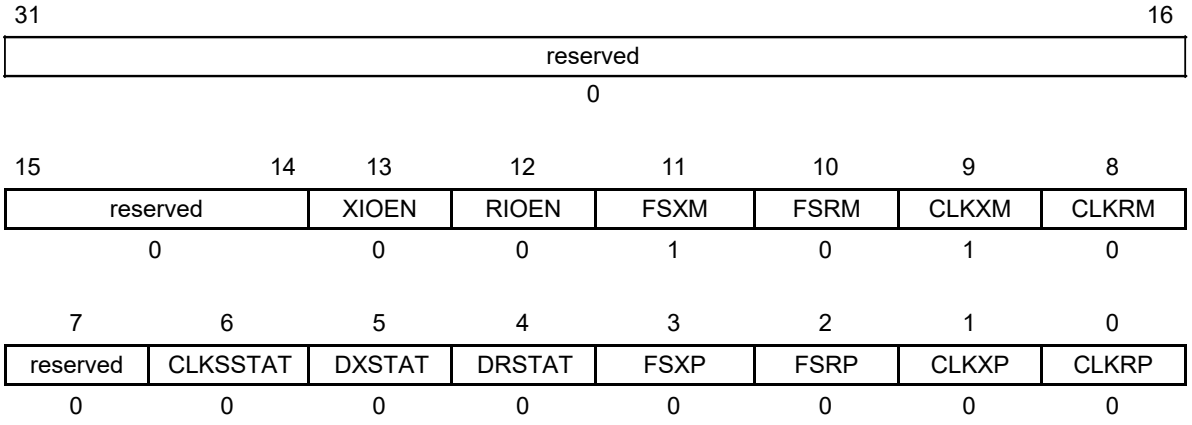

Figure 5. Pin Control Register (PCR)

Table 1. Bit-Field Values for McBSP Registers

Register [bit-field #]	Bit-field Name	Master (Transmitter)		Slave (Receiver)	
		Bin	HAL Macro MCBSP_	Bin	HAL Macro MCBSP_
RCR[17:16]	RDATDLY	0	RCR_RDATDLY_DEFAULT	1	RCR_RDATDLY_1BIT
XCR[17:16]	XDATDLY	1	XCR_XDATDLY_1BIT	0	XCR_XDATDLY_DEFAULT
SRGR[29]	CLKSM	1	SRGR_CLKSM_INTERNAL	0	SRGR_CLKSM_DEFAULT
SRGR[28]	FSGM	1	SRGR_FSGM_FSG	0	SRGR_FSGM_ DEFAULT
SRGR[7:0]	CLKGDV	1	SRGR_CLKGDV_OF(0x0)	0	SRGR_CLKGDV_ DEFAULT
PCR[11]	FSXM	1	PCR_FSXM_INTERNAL	0	PCR_FSXM_ DEFAULT
PCR[10]	FSRM	0	PCR_FSRM_DEFAULT	0	PCR_FSRM_ DEFAULT
PCR[9]	CLKXM	1	PCR_CLKXM_OUTPUT	0	PCR_CLKXM_ DEFAULT
PCR[8]	CLKRM	0	PCR_CLKRM_DEFAULT	0	PCR_CLKRM_ DEFAULT

The bit-fields not listed in Table 1 assume their default values. Note that the above HAL macros are merely sample code. The user can also set some other register field values, such as the number of phases, frame lengths, number of elements per frame, clock polarities and other parameters as required in a specific application. Appendices A and B provide full sample code, which uses the above macros, for setting up the McBSP on one DSP to transfer data to another McBSP of a similar C6000 device. The sample program in Appendix A sets up the McBSP0 of the master device as a transmitter. The sample code in Appendix B sets up the McBSP0 of the slave device as the receiver. This code can run on all C6000 devices, as both DMA and EDMA support is provided. The DMA/EDMA aid in the transfer of data to/from these two McBSPs. Again, this is just sample code, therefore the user can modify the different HAL macros (or the entire McBSP/DMA/EDMA/IRQ structures in the program) to fit a specific application. Please refer to the *TMS320C6000 Chip Support Library User's Guide* (SPRU401) for further information on the use of the functions and macros used in the code of Appendices A and B.

2.2 Timing Analysis

The parameters that need to be satisfied to achieve maximum data rates are listed in Table 2, Table 3, and Table 4 for the C6201, C6211/C6711, and TMS320C64x™ devices respectively. CLKPER is the clock period of the master clock (in this case, CLKX0/1), which can be varied in the analysis to verify the frequency at which the design margins are met. Note that a board route delay of 1 ns, or about 6 inches, is considered for propagation delay from source to destination.

Table 2 through Table 4 show the timing analysis for the McBSP of the C6201, C6211/C6711, and C64x™ DSPs respectively, indicating the maximum transfer rates achieved before any timing violations are met (refer to the conclusion), satisfying all timing requirements (no negative margins).

TMS320C64x and C64x are trademarks of Texas Instruments.

Table 2. Timing Requirements and Switching Characteristics for the C6201

Sl. No.	Parameter Type	Parameter Name	[Min, Max] (ns)	Margin (ns)	Description
1	Constraint	t_{su}(FRH-CKRL)	[2,]	<0>	Setup time, FSR valid to CLKR low
2	Constraint	t_h (CKRL-FRH)	[3,]	<0>	Hold time, FSR valid after CLKR low
3	Constraint	t_{su}(DRV-CKRL)	[0,]	<1>	Setup time, DR valid to CLKR low
4	Constraint	t_h (CKRL-DRV)	[4,]	<0>	Hold time, DR valid after CLKR low
5	Delay	t_d (CKXH-FXV)	[-2, 3]		Delay time, FSX valid after CLKX low
6	Delay	t_{dis} (CKXH-DXHZ)	[-1, 4]		Disable time, CLKX high to DX high impedance following last data bit.
7	Delay	t_d (CKXH-DXV)	[-1 4]		Delay time, DX valid after CLKX high
8	Variable	CLKPER	[10, 10]		Master clock period. Can be varied to satisfy parametric requirements.
9	Variable delay	t_{wd} (DX-DR)	[1, 1]		Wire delay from DX to DR. Set as per system needs.
10	Variable delay	t_{wd} (FSX-FSR)	[1, 1]		Wire delay from FSX to FSR. Set as per system needs.
11	Variable delay	t_{wd} (CLKX-CLKR)	[1, 1]		Wire delay from CLKX to CLKR. Set as per system needs.

Table 3. Timing Requirements and Switching Characteristics for the C6211/C6711

Sl. No.	Parameter Type	Parameter Name	[Min, Max] (ns)	Margin (ns)	Description
1	Constraint	t_{su}(FRH-CKRL)	[1,]	<10>	Setup time, FSR valid to CLKR low
2	Constraint	t_h (CKRL-FRH)	[3,]	<0>	Hold time, FSR valid after CLKR low
3	Constraint	t_{su}(DRV-CKRL)	[3,]	<7>	Setup time, DR valid to CLKR low
4	Constraint	t_h (CKRL-DRV)	[4,]	<1>	Hold time, DR valid after CLKR low
5	Delay	t_d (CKXH-FXV)	[-11, 3]		Delay time, FSX valid after CLKX low
6	Delay	t_{dis} (CKXH-DXHZ)	[-9, 4]		Disable time, CLKX high to DX high impedance following last data bit.
7	Delay	t_d (CKXH-DXV)	[-9, 4]		Delay time, DX valid after CLKX high
8	Variable	CLKPER	[28, 28]		Master clock period. Can be varied to satisfy parametric requirements.
9	Variable delay	t_{wd} (DX-DR)	[1, 1]		Wire delay from DX to DR. Set as per system needs.
10	Variable delay	t_{wd} (FSX-FSR)	[1, 1]		Wire delay from FSX to FSR. Set as per system needs.
11	Variable delay	t_{wd} (CLKX-CLKR)	[1, 1]		Wire delay from CLKX to CLKR. Set as per system needs.

Table 4. Timing Requirements and Switching Characteristics for the C64x

Sl. No.	Parameter Type	Parameter Name	[Min, Max] (ns)	Margin (ns)	Description
1	Constraint	t_{su}(FRH-CKRL)	[1,]	<0>	Setup time, FSR valid to CLKR low
2	Constraint	t_h (CKRL-FRH)	[3,]	<0>	Hold time, FSR valid after CLKR low
3	Constraint	t_{su}(DRV-CKRL)	[0,]	<0>	Setup time, DR valid to CLKR low
4	Constraint	t_h (CKRL-DRV)	[3,]	<0>	Hold time, DR valid after CLKR low
5	Delay	t_d (CKXH-FXV)	[-1, 3]		Delay time, FSX valid after CLKX low
6	Delay	t_{dis} (CKXH-DXHZ)	[-1, 4]		Disable time, CLKX high to DX high impedance following last data bit.
7	Delay	t_d (CKXH-DXV)	[-1, 4]		Delay time, DX valid after CLKX high
8	Variable	CLKPER	[8, 8]		Master clock period. Can be varied to satisfy parametric requirements.
9	Variable delay	t_{wd} (DX-DR)	[1, 1]		Wire delay from DX to DR. Set as per system needs.
10	Variable delay	t_{wd} (FSX-FSR)	[1, 1]		Wire delay from FSX to FSR. Set as per system needs.
11	Variable delay	t_{wd} (CLKX-CLKR)	[1, 1]		Wire delay from CLKX to CLKR. Set as per system needs.

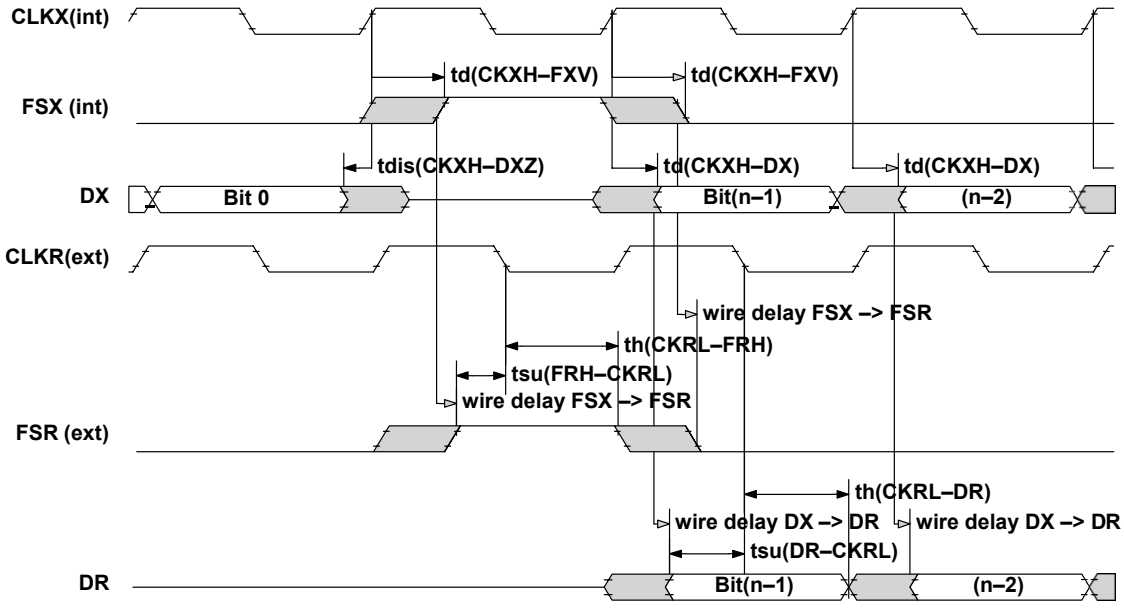

Figure 6. Signal Relationship for C6000 McBSP to McBSP Data Transfer With (R/X)DATDLY=1

2.3 Conclusion

Using the McBSP configuration as stated in the assumption, the following maximum transfer rates for the C6000 DSPs can be achieved:

Table 5. McBSP Maximum Transfer Rates for TMS320C6000 Devices

Device	Maximum Transfer Rate (Mbps)
C6201, C6202/B, C6203	100
C6204, C6205	83.33
C6211/B, C6711, C6712	35.71
C6701	55.56
C64x	125

The two McBSPs can be used as a high-speed communication port, with a data transfer rate that can be affected by the following factors:

- Method of data processing: For example, an interrupt-driven transfer is slower than a DMA/EDMA transfer. Therefore, the data transfer rate will have to be reduced so that all of the data is processed without missing any.
- Priority of data processing: The DMA/EDMA has a lower priority than the CPU, or the DMA channel used for servicing the McBSP has a lower priority than the other channels of the DMA/EDMA. Thus, the time taken to service a write or read to/from the McBSP can be extended, causing data over-write and/or receiver full error condition. To avoid this, either the bit-clock rate should be reduced, or appropriate priorities should be assigned for each channel.

NOTE: This application report discusses only two TMS320C6000 DSP McBSPs communicating with each other. Higher speeds can be achieved if the McBSP is connected to any other device that meets the timing parameters listed in the *TMS320C6000 Digital Signal Processor* data sheets found at www.ti.com.

3 References

1. TMS320C6000 Digital Signal Processor data sheets, found at www.ti.com
2. TMS320C6000 Peripherals Reference Guide (SPRU190).

Appendix A Sample Code for McBSP Master (Transmitter)

```
/*-------------------------------------------------------------------------
*/
/*  mcbsp_xmit_master.c V1.00
*/
/*  Copyright (c) 2001 Texas Instruments Incorporated
*/
/*-------------------------------------------------------------------------
--*/
/*
    6/26/01:  Vassos S. Soteriou
   mcbsp_xmit_master.c:
This program sets the McBSP0 of the TMS320C6000 devices in the data transmit
mode, to transfer data to the McBSP of another C6000 device.  This program
supports all the Texas Instruments TMS320C6000 DSPs, those that use the DMA
controller or the Enhanced DMA controller (EDMA). For those that use the DMA
controller, DMA channel 2 services the McBSP for data transfer. The vecs.asm
assembly code file is used to hookup the c_int11() ISR to the corresponding
interrupt.  Channel 2 is hooked up to interrupt 11 for data transmit, the DMA
controller has individual interrupts for each DMA channel.  The EDMA control-
ler, however, generates a single interrupt to the CPU (EDMA_INT) on behalf of
all 16 channels (C621x/C671x) or 64 channels(C64x). The various control regis-
ters and bit fields facilitate EDMA interrupt generation.  CPU_INT8 is  re-
sponsible for all the EDMA channels. The sample code is based on TI's CSL 2.0.
Please refer to the TMS320C6000 Chip Support Library API User's Guide for fur-
ther information.
Note that any DMA channel with any interrupt and any McBSP can be used for
this transfer, this is just sample code that can be used as a reference.
*/
/* Chip definition, change this accordingly */
#define CHIP_6414 1
/* Include files */
#include <c6x.h>
#include  <csl.h>    /*  CSL  library    */
#include   <csl_dma.h>   /*   DMA_SUPPORT   */
#include  <csl_edma.h>  /*  EDMA_SUPPORT  */
#include   <csl_irq.h>   /*   IRQ_SUPPORT   */
#include <csl_mcbsp.h> /* MCBSP_SUPPORT */
/*-------------------------------------------------------------------------*/
/* Define constants */
#define FALSE 0
#define TRUE 1
#define DMA_XFER 8
#define XFER_TYPE DMA_XFER
#define BUFFER_SIZE 256
#define ELEMENT_COUNT 5 /* set element_count =< buffer_size */
/* Global variables used in interrupt ISRs */
volatile int xmit0_done = FALSE;
/*-------------------------------------------------------------------------*/
/* Declare CSL objects */
MCBSP_Handle hMcbsp0;       /* Handles for McBSP */
#if (DMA_SUPPORT)
DMA_Handle hDma2;           /* Handle for DMA    */
```

```
#endif
#if (EDMA_SUPPORT)          /* Handles for EDMA  */
EDMA_Handle hEdma1;
EDMA_Handle hEdmadummy;
#endif
/*--------------------------------------------------------------------*/
/* External functions and function prototypes */
void init_mcbsp0_master(void);   /* Function prototypes */
void set_interrupts_dma(void);
void set_interrupts_edma(void);
/* Include the vector table to call the IRQ ISRs hookup */
extern far void vectors();
/*--------------------------------------------------------------------*/
/* main()
   */
/*--------------------------------------------------------------------*/
void main(void)
{
/* Declaration of local variables */
static int element_count, xfer_type;
static Uint32 dmaOutbuff[BUFFER_SIZE];  /* buffer for DMA supporting devices
*/
static Uint32 edmaOutbuff[BUFFER_SIZE]; /* buffer for EDMA supporting devices
*/
IRQ_setVecs(vectors); /* point to the IRQ vector table */
element_count = ELEMENT_COUNT;
xfer_type = XFER_TYPE;
/* initialize the CSL library */
CSL_init();
init_mcbsp0_master();
/* Enable sample rate generator GRST=1 */
MCBSP_enableSrgr(hMcbsp0);       /* Handle to SRGR */
switch (xfer_type) {
case DMA_XFER:
  #if (DMA_SUPPORT)       /* for DMA supporting devices */
  DMA_reset(INV);         /* reset all DMA channels */
  #endif
  #if (EDMA_SUPPORT)           /* for EDMA supporting devices */
  EDMA_clearPram(0x00000000); /* Clear PaRAM RAM of the EDMA */
  set_interrupts_edma();
  #endif
/*--------------------------------------------------------------------------*/
/* DMA channel 2 config structure
   */
/*--------------------------------------------------------------------------*/
#if (DMA_SUPPORT)       /* for DMA supporting devices */
  /* Channel 2 transmits the data */
  hDma2 = DMA_open(DMA_CHA2, DMA_OPEN_RESET);  /* Handle to DMA channel 2 */
    DMA_configArgs(hDma2,
       DMA_PRICTL_RMK(
           DMA_PRICTL_DSTRLD_DEFAULT,
           DMA_PRICTL_SRCRLD_DEFAULT,
           DMA_PRICTL_EMOD_DEFAULT,
```

```
            DMA_PRICTL_FS_DEFAULT,
            DMA_PRICTL_TCINT_ENABLE, /* TCINT =1                                */
            DMA_PRICTL_PRI_DMA,      /* DMA high priority                       */
            DMA_PRICTL_WSYNC_XEVT0, /* Set synchronization event XEVT0=01100*/
            DMA_PRICTL_RSYNC_DEFAULT,
            DMA_PRICTL_INDEX_DEFAULT,
            DMA_PRICTL_CNTRLD_DEFAULT,
            DMA_PRICTL_SPLIT_DEFAULT,
            DMA_PRICTL_ESIZE_32BIT,  /* Element size 32 bits                    */
            DMA_PRICTL_DSTDIR_DEFAULT,
            DMA_PRICTL_SRCDIR_INC,   /* Increment source by element size        */
            DMA_PRICTL_START_DEFAULT
            ),
         DMA_SECCTL_RMK(
            DMA_SECCTL_WSPOL_NA, /* only for 6202 and 6203 devices */
            DMA_SECCTL_RSPOL_NA, /* only for 6202 and 6203 devices */
            DMA_SECCTL_FSIG_NA, /* only for 6202 and 6203 devices */
            DMA_SECCTL_DMACEN_DEFAULT,
            DMA_SECCTL_WSYNCCLR_DEFAULT,
            DMA_SECCTL_WSYNCSTAT_DEFAULT,
            DMA_SECCTL_RSYNCCLR_DEFAULT,
            DMA_SECCTL_RSYNCSTAT_DEFAULT,
            DMA_SECCTL_WDROPIE_DEFAULT,
            DMA_SECCTL_WDROPCOND_DEFAULT,
            DMA_SECCTL_RDROPIE_DEFAULT,
            DMA_SECCTL_RDROPCOND_DEFAULT,
            DMA_SECCTL_BLOCKIE_ENABLE, /* BLOCK IE=1 enables DMA channel int */
            DMA_SECCTL_BLOCKCOND_DEFAULT,
            DMA_SECCTL_LASTIE_DEFAULT,
            DMA_SECCTL_LASTCOND_DEFAULT,
            DMA_SECCTL_FRAMEIE_DEFAULT,
            DMA_SECCTL_FRAMECOND_DEFAULT,
            DMA_SECCTL_SXIE_DEFAULT,
            DMA_SECCTL_SXCOND_DEFAULT
            ),
         DMA_SRC_RMK((Uint32)dmaOutbuff),
         DMA_DST_RMK(MCBSP_ADDRH(hMcbsp0, DXR)),
         DMA_XFRCNT_RMK(
            DMA_XFRCNT_FRMCNT_DEFAULT,
            DMA_XFRCNT_ELECNT_OF(element_count)  /* set xfer element count */
            )
);
set_interrupts_dma(); /* initialize the interrupt(s)                          */
                      /* enable the interrupt after DMA channels are          */
                      /* opened as the DMA_OPEN_RESET clears and disables     */
                      /* the channel interrupt once specified and clears      */
                      /* the corresponding inetrrupt bits in the IER. This    */
                      /* is not applicable for the EDMA channel open case     */
DMA_start(hDma2);     /* Start DMA channel 2 */
#endif  /* end for dma supporting devices */
/*--------------------------------------------------------------------------*/
/* EDMA channel 12 config structure                                         */
/*--------------------------------------------------------------------------*/
```

```
  #if (EDMA_SUPPORT)      /* for EDMA supporting devices  */

hEdma1 = EDMA_open(EDMA_CHA_XEVT0, EDMA_OPEN_RESET);
EDMA_configArgs(hEdma1,
#if(!C64_SUPPORT)               /* For 671X and 621x devices        */
      EDMA_OPT_RMK(
        EDMA_OPT_PRI_HIGH,      /* High priority EDMA               */
        EDMA_OPT_ESIZE_32BIT,   /* Element size 32 bits             */
        EDMA_OPT_2DS_DEFAULT,
        EDMA_OPT_SUM_INC,       /* Source increment by element size */
        EDMA_OPT_2DD_DEFAULT,
        EDMA_OPT_DUM_DEFAULT,
        EDMA_OPT_TCINT_YES,     /* Enable Transfer Complete Interrupt      */
        EDMA_OPT_TCC_OF(12),    /* TCCINT = 0xC, XEVT0                     */
        EDMA_OPT_LINK_YES,      /* Enable linking to NULL table            */
        EDMA_OPT_FS_NO
        ),
#endif
#if(C64_SUPPORT)
        EDMA_OPT_RMK(           /* For 64x devices only             */
        EDMA_OPT_PRI_HIGH,      /* High priority EDMA               */
        EDMA_OPT_ESIZE_32BIT,   /* Element size 32 bits             */
        EDMA_OPT_2DS_DEFAULT,
        EDMA_OPT_SUM_INC,       /* Source increment by element size */
        EDMA_OPT_2DD_DEFAULT,
        EDMA_OPT_DUM_DEFAULT,
        EDMA_OPT_TCINT_YES,   /*  Enable  Transfer  Complete  Interrupt  */
        EDMA_OPT_TCC_OF(12),    /* TCCINT = 0xC, XEVT0                     */
        EDMA_OPT_TCCM_DEFAULT,
        EDMA_OPT_ATCINT_DEFAULT,
        EDMA_OPT_ATCC_DEFAULT,
        EDMA_OPT_PDTS_DEFAULT,
        EDMA_OPT_PDTD_DEFAULT,
        EDMA_OPT_LINK_YES,      /* Enable linking to NULL table            */
        EDMA_OPT_FS_NO

        ),
#endif
        EDMA_SRC_RMK((Uint32)edmaOutbuff),  /*  dst  addr  to  edmaOutbuff  */
        EDMA_CNT_RMK(0, element_count), /* set count equal to element_count */
        EDMA_DST_RMK(MCBSP_ADDRH(hMcbsp0, DXR)), /* dst to DXR0            */
        EDMA_IDX_RMK(0,0),
        EDMA_RLD_RMK(0,0)
        );
  hEdmadummy = EDMA_allocTable(-1); /* Dynamically allocates PaRAM RAM table*/
  EDMA_configArgs(hEdmadummy, /* Dummy or Terminating Table in PaRAM       */
         0x00000000,          /* Terminate EDMA transfers by linking to    */

         0x00000000,          /* this NULL table                           */
         0x00000000,
         0x00000000,
         0x00000000,
         0x00000000
```

```
        );
  EDMA_link(hEdma1, hEdmadummy); /* Link terminating event to the EDMA event*/
  EDMA_enableChannel(hEdma1);    /* Enable EDMA channel */
  #endif  /* end for EDMA supporting devices */
}
/* Enable McBSP channel */
MCBSP_enableXmt(hMcbsp0); /* McBSP port 0 as the transmitter         */
MCBSP_enableFsync(hMcbsp0); /* Enable frame sync for the McBSP        */
/* To flag an interrupt to the CPU when DMA transmit is done */
#if (DMA_SUPPORT)
  while (!xmit0_done);
#endif
/* To flag an interrupt to the CPU when EDMA transfer is done  */
/* Transfer completion interrupt 12 sets flag = 1 when set     */
#if (EDMA_SUPPORT)
  while (!xmit0_done);
#endif
MCBSP_close(hMcbsp0);  /* close McBSP port */
#if (DMA_SUPPORT)      /* close DMA channels */
DMA_close(hDma2);
#endif
#if (EDMA_SUPPPORT)
EDMA_close(hEdma1);    /* close EDMA channel */
EDMA_close(hEdmadummy);
#endif
} /* end main, program ends here */
/*-------------------------------------------------------------------------*/
/* init_mcbsp0_master()                                                    */
/*-------------------------------------------------------------------------*/
/* MCBSP Config structure */
/* Setup the MCBSP_0 for data transfer */
void
init_mcbsp0_master(void)
{
MCBSP_Config mcbspCfg0 = {
#if (EDMA_SUPPORT)
    MCBSP_SPCR_RMK(
                MCBSP_SPCR_FREE_DEFAULT,  /* All fields in SPCR set to default
values */
                MCBSP_SPCR_SOFT_DEFAULT,
            MCBSP_SPCR_FRST_DEFAULT,
            MCBSP_SPCR_GRST_DEFAULT,
            MCBSP_SPCR_XINTM_DEFAULT,
            MCBSP_SPCR_XSYNCERR_DEFAULT,
            MCBSP_SPCR_XRST_DEFAULT,
            MCBSP_SPCR_DLB_DEFAULT,
            MCBSP_SPCR_RJUST_DEFAULT,
            MCBSP_SPCR_CLKSTP_DEFAULT,
            MCBSP_SPCR_DXENA_DEFAULT,
            MCBSP_SPCR_RINTM_DEFAULT,
            MCBSP_SPCR_RSYNCERR_DEFAULT,
            MCBSP_SPCR_RRST_DEFAULT
            ),
```

```
#endif
#if (DMA_SUPPORT)
   MCBSP_SPCR_RMK(
          MCBSP_SPCR_FRST_DEFAULT,   /* All fields in SPCR set to default val-
ues */
          MCBSP_SPCR_GRST_DEFAULT,
          MCBSP_SPCR_XINTM_DEFAULT,
          MCBSP_SPCR_XSYNCERR_DEFAULT,
          MCBSP_SPCR_XRST_DEFAULT,
          MCBSP_SPCR_DLB_DEFAULT,
          MCBSP_SPCR_RJUST_DEFAULT,
          MCBSP_SPCR_CLKSTP_DEFAULT,
          MCBSP_SPCR_RINTM_DEFAULT,
          MCBSP_SPCR_RSYNCERR_DEFAULT,
          MCBSP_SPCR_RRST_DEFAULT
          ),
#endif
#if (EDMA_SUPPORT)
    MCBSP_RCR_RMK(
           MCBSP_RCR_RPHASE_DEFAULT,  /* All fields in RCR set to default val-
ues  */
           MCBSP_RCR_RFRLEN2_DEFAULT,
           MCBSP_RCR_RWDLEN2_DEFAULT,
           MCBSP_RCR_RCOMPAND_DEFAULT,
           MCBSP_RCR_RFIG_DEFAULT,
           MCBSP_RCR_RDATDLY_DEFAULT,
           MCBSP_RCR_RFRLEN1_DEFAULT,
           MCBSP_RCR_RWDLEN1_DEFAULT,
           MCBSP_RCR_RWDREVRS_DEFAULT
           ),
#endif
#if (DMA_SUPPORT)
    MCBSP_RCR_RMK(
           MCBSP_RCR_RPHASE_DEFAULT,  /* All fields in RCR set to default val-
ues  */
           MCBSP_RCR_RFRLEN2_DEFAULT,
           MCBSP_RCR_RWDLEN2_DEFAULT,
           MCBSP_RCR_RCOMPAND_DEFAULT,
           MCBSP_RCR_RFIG_DEFAULT,
           MCBSP_RCR_RDATDLY_DEFAULT,
           MCBSP_RCR_RFRLEN1_DEFAULT,
           MCBSP_RCR_RWDLEN1_DEFAULT
           ),
#endif
#if (EDMA_SUPPORT)
    MCBSP_XCR_RMK(
           MCBSP_XCR_XPHASE_SINGLE,   /* Single phase transmit frame */
           MCBSP_XCR_XFRLEN2_DEFAULT,
           MCBSP_XCR_XWDLEN2_DEFAULT,
           MCBSP_XCR_XCOMPAND_DEFAULT,
           MCBSP_XCR_XFIG_DEFAULT,
           MCBSP_XCR_XDATDLY_1BIT,    /* 1-bit transmit data delay   */
           MCBSP_XCR_XFRLEN1_DEFAULT,
```

```
            MCBSP_XCR_XWDLEN1_DEFAULT,
            MCBSP_XCR_XWDREVRS_DEFAULT
            ),
#endif
#if (DMA_SUPPORT)
    MCBSP_XCR_RMK(
            MCBSP_XCR_XPHASE_SINGLE,   /* Single phase transmit frame */
            MCBSP_XCR_XFRLEN2_DEFAULT,
            MCBSP_XCR_XWDLEN2_DEFAULT,
            MCBSP_XCR_XCOMPAND_DEFAULT,
            MCBSP_XCR_XFIG_DEFAULT,
            MCBSP_XCR_XDATDLY_1BIT,    /* 1-bit transmit data delay   */
            MCBSP_XCR_XFRLEN1_DEFAULT,
            MCBSP_XCR_XWDLEN1_DEFAULT
            ),
#endif
     MCBSP_SRGR_RMK(
            MCBSP_SRGR_GSYNC_DEFAULT,
            MCBSP_SRGR_CLKSP_DEFAULT,
            MCBSP_SRGR_CLKSM_INTERNAL, /* Internal clock source              */
            MCBSP_SRGR_FSGM_FSG,       /* FSX driven by SRG frame sync signal*/

            MCBSP_SRGR_FPER_DEFAULT,
            MCBSP_SRGR_FWID_DEFAULT,
            MCBSP_SRGR_CLKGDV_OF(0x0) /* CLock divide of 1                   */
            ),
#if (C64_SUPPORT)
     MCBSP_MCR_RMK(                  /* only for 64x                        */
            MCBSP_MCR_XMCME_DEFAULT,  /* All fields in MCR set to default val-
ues    */
            MCBSP_MCR_XPBBLK_DEFAULT,
            MCBSP_MCR_XPABLK_DEFAULT,
            MCBSP_MCR_XMCM_DEFAULT,
            MCBSP_MCR_RPBBLK_DEFAULT,
            MCBSP_MCR_RMCME_DEFAULT,
            MCBSP_MCR_RPABLK_DEFAULT,
            MCBSP_MCR_RMCM_DEFAULT
            ),
#else
     MCBSP_MCR_RMK(
            MCBSP_MCR_XPBBLK_DEFAULT, /* All fields in MCR set to default val-
ues    */
            MCBSP_MCR_XPABLK_DEFAULT,
            MCBSP_MCR_XMCM_DEFAULT,
            MCBSP_MCR_RPBBLK_DEFAULT,
            MCBSP_MCR_RPABLK_DEFAULT,
            MCBSP_MCR_RMCM_DEFAULT
            ),
#endif
#if(!C64_SUPPORT)
     MCBSP_RCER_RMK(
            MCBSP_RCER_RCEB_DEFAULT,  /* All fields in RCER set to default val-
ues */
```

```
            MCBSP_RCER_RCEA_DEFAULT
            ),
#endif
#if(!C64_SUPPORT)
     MCBSP_XCER_RMK(
            MCBSP_XCER_XCEB_DEFAULT,  /* All fields in XCER set to default val-
ues */
            MCBSP_XCER_XCEA_DEFAULT
            ),
#endif
#if (C64_SUPPORT)
            MCBSP_RCERE0_RMK(0),      /* Additional registers only for 64x   */
            MCBSP_RCERE1_RMK(0),
            MCBSP_RCERE2_RMK(0),
            MCBSP_RCERE3_RMK(0),
#endif

#if (C64_SUPPORT)
            MCBSP_XCERE0_RMK(0),       /* Additional registers only for 64x  */
            MCBSP_XCERE1_RMK(0),
            MCBSP_XCERE2_RMK(0),
            MCBSP_XCERE3_RMK(0),
#endif
     MCBSP_PCR_RMK(
            MCBSP_PCR_XIOEN_DEFAULT,
            MCBSP_PCR_RIOEN_DEFAULT,
            MCBSP_PCR_FSXM_INTERNAL,  /* Frame sync generated internally     */
            MCBSP_PCR_FSRM_DEFAULT,
            MCBSP_PCR_CLKXM_OUTPUT,   /* tans. clock mode from internal SRGR */
            MCBSP_PCR_CLKRM_DEFAULT,
            MCBSP_PCR_CLKSSTAT_DEFAULT,
            MCBSP_PCR_DXSTAT_DEFAULT,
            MCBSP_PCR_FSXP_DEFAULT,
            MCBSP_PCR_FSRP_DEFAULT,
            MCBSP_PCR_CLKXP_DEFAULT,
            MCBSP_PCR_CLKRP_DEFAULT
            )
};
hMcbsp0 = MCBSP_open(MCBSP_DEV0, MCBSP_OPEN_RESET); /* McBSP port 0 */
MCBSP_config(hMcbsp0, &mcbspCfg0);
}
/*----------------------------------------------------------------------------*/
/* set_interrupts_dma()                                                       */
/*----------------------------------------------------------------------------*/
#if (DMA_SUPPORT)
void                            /* Set the interrupts         */
set_interrupts_dma(void)        /* if the device supports DMA */
{
 IRQ_nmiEnable();
 IRQ_globalEnable();
 IRQ_disable(IRQ_EVT_DMAINT2);  /* INT11 */
 IRQ_clear(IRQ_EVT_DMAINT2);
 IRQ_enable(IRQ_EVT_DMAINT2);
```

```
 return;
}
#endif
/*-------------------------------------------------------------------------*/
/* set_interrupts_edma()                                                   */
/*-------------------------------------------------------------------------*/
#if (EDMA_SUPPORT)
void                          /* Set the interrupt           */
set_interrupts_edma(void)     /* if the device supports EDMA */
{
 IRQ_nmiEnable();
 IRQ_globalEnable();
 IRQ_reset(IRQ_EVT_EDMAINT);
 IRQ_disable(IRQ_EVT_EDMAINT);
 EDMA_intDisable(12);         /* ch 12 for McBSP transmit event XEVT0 */
 IRQ_clear(IRQ_EVT_EDMAINT);
 EDMA_intClear(12);
 IRQ_enable(IRQ_EVT_EDMAINT);
 EDMA_intEnable(12);
 return;
}
#endif
/*-------------------------------------------------------------------------*/
/*    DMA DATA TRANSFER COMPLETION ISRs                                    */
/*-------------------------------------------------------------------------*/
interrupt void     /* vecs.asm hooks this up to IRQ 11 */
c_int11(void)      /* DMA ch2                          */
{
xmit0_done = TRUE;
return;
}
interrupt void    /* vecs.asm hooks this up to IRQ 08 */
c_int08(void)     /* for the EDMA                     */
{
 #if (EDMA_SUPPORT)
  if (EDMA_intTest(12))
  {
  xmit0_done = TRUE;
  EDMA_intClear(12); /* clear CIPR bit so future interrupts can be recognized
*/
  }
#endif
return;
}
/*------------------------End of mcbsp_xmit_master.c-----------------------*/
*****************************************************************************
*                                                                           *
*           Copyright (C) 2000 Texas Instruments Incorporated.              *
*                           All Rights Reserved                             *
*---------------------------------------------------------------------------*
* FILENAME...... vecs.asm                                                   *
* DATE CREATED.. 06/27/2001                                                 *
*****************************************************************************
```

```
*-------------------------------------------------------------------------*
* Global symbols defined here and exported out of this file              *
*-------------------------------------------------------------------------*
   .global _vectors
   .global _vector0
   .global _vector1
   .global _vector2
   .global _vector3
   .global _vector4
   .global _vector5
   .global _vector6
   .global _vector7
   .global _c_int08   ; Hookup the c_int08 ISR in main() for EDMA
   .global _vector9
   .global _vector10
   .global _c_int11   ; Hookup the c_int11 ISR in main() for DMA
   .global _vector12
   .global _vector13
   .global _vector14
   .global _vector15
*-------------------------------------------------------------------------*
* Global symbols referenced in this file but defined somewhere else.     *
* Remember that your interrupt service routines need to be referenced here.  *
*-------------------------------------------------------------------------*
   .ref _c_int00
*-------------------------------------------------------------------------*
* This is a macro that instantiates one entry in the interrupt service table.*
*-------------------------------------------------------------------------*
VEC_ENTRY .macro addr
    STW   B0,*--B15
    MVKL  addr,B0
    MVKH  addr,B0
    B B0
    LDW   *B15++,B0
    NOP   2
    NOP
    NOP
   .endm
*-------------------------------------------------------------------------*
* This is a dummy interrupt service routine used to initialize the IST.  *
*-------------------------------------------------------------------------*
_vec_dummy:
  B    B3
  NOP  5
*-------------------------------------------------------------------------*
* This is the actual interrupt service table (IST). It is properly aligned   *
* and is located in the subsection .text:vecs. This means if you don't       *
* explicitly specify this section in your linker command file, it will       *
* default and link into the .text section. Remember to set the ISTP          *
* register to point to this table.                                           *
*-------------------------------------------------------------------------*
 .sect ".text:vecs"
 .align 1024
```

```
_vectors:
_vector0:   VEC_ENTRY _vec_dummy
_vector1:   VEC_ENTRY _vec_dummy
_vector2:   VEC_ENTRY _vec_dummy
_vector3:   VEC_ENTRY _vec_dummy
_vector4:   VEC_ENTRY _vec_dummy
_vector5:   VEC_ENTRY _vec_dummy
_vector6:   VEC_ENTRY _vec_dummy
_vector7:   VEC_ENTRY _vec_dummy
_vector8:   VEC_ENTRY _c_int08    ; Hookup the c_int08 ISR in main() for EDMA
_vector9:   VEC_ENTRY _vec_dummy
_vector10:  VEC_ENTRY _vec_dummy
_vector11:  VEC_ENTRY _c_int11    ; Hookup the c_int11 ISR in main() for DMA
_vector12:  VEC_ENTRY _vec_dummy
_vector13:  VEC_ENTRY _vec_dummy
_vector14:  VEC_ENTRY _vec_dummy
_vector15:  VEC_ENTRY _vec_dummy
*------------------------------End of vecs.asm--------------------------------*
Title
```

Appendix B Sample Code for McBSP Slave (Receiver)

```
/*--------------------------------------------------------------------------*/
/*  mcbsp_recv_slave.c V1.00
    */
/*  Copyright (c) 2001 Texas Instruments Incorporated
    */
/*--------------------------------------------------------------------------*/
/*
    6/26/01:  Vassos S. Soteriou

   mcbsp_recv_slave.c:
This program sets the McBSP0 of the TMS320C6000 devices in the data receive
mode,to receive data from the McBSP of another C6000 device.  This program
supports all the Texas Instruments TMS320C6000 DSPs, those that use the DMA
controller or the Enhanced DMA controller (EDMA). For those that use the DMA
controller, DMA channel 1 services the McBSP for data receive. The vecs.asm
assembly code file is used to hookup the c_int09() ISR to the corresponding
interrupt.  Channel 1 is hooked up to interrupt 09 for data receive, the DMA
controller has individual interrupts for each DMA channel.  The EDMA control-
ler, however, generates a single interrupt to the CPU (EDMA_INT) on behalf of
all 16 channels (C621x/C671x) or 64 channelS (C64x). The various control reg-
isters and bit fields facilitate EDMA interrupt generation.  CPU_INT8 is re-
sponsible for all the EDMA channels. The sample code is based on TI's CSL 2.0.
Please refer to the TMS320C6000 Chip Support Library API User's Guide for fur-
ther information.
Note that any DMA channel with any interrupt and any McBSP can be used for
this transfer, this is just sample code that can be used as a reference.
*/
/* Chip definition, change this accordingly */
#define CHIP_6202 1
/* Include files */
#include <c6x.h>
#include  <csl.h>    /*  CSL  library    */
#include   <csl_dma.h>   /*   DMA_SUPPORT   */
#include  <csl_edma.h>  /*  EDMA_SUPPORT  */
#include   <csl_irq.h>   /*   IRQ_SUPPORT   */
#include <csl_mcbsp.h> /* MCBSP_SUPPORT */
/*--------------------------------------------------------------------------*/
/* Define constants */
#define FALSE 0
#define TRUE 1
#define DMA_XFER 8
#define XFER_TYPE DMA_XFER
#define BUFFER_SIZE 256 /* set same value as xmit */
#define ELEMENT_COUNT 5 /* set element_count =< buffer_size, set same value as
xmit*/
/* Global variables used in interrupt ISRs */
volatile int recv0_done = FALSE;
/*--------------------------------------------------------------------------*/
/* Declare CSL objects */
MCBSP_Handle hMcbsp0;       /* Handles for McBSP */
#if (DMA_SUPPORT)
DMA_Handle hDma1;           /* Handle for DMA    */
```

```
#endif
#if (EDMA_SUPPORT)          /* Handles for EDMA  */
EDMA_Handle hEdma1;
EDMA_Handle hEdmadummy;
#endif
/*--------------------------------------------------------------------------*/
/* External functions and function prototypes */
void init_mcbsp0_slave(void);   /* Function prototypes */
void set_interrupts_dma(void);
void set_interrupts_edma(void);
/* Include the vector table to call the IRQ ISRs hookup */
extern far void vectors();
/*--------------------------------------------------------------------------*/
/* main()
   */
/*--------------------------------------------------------------------------*/
void main(void)
{
/* Declaration of local variables */
static int element_count, xfer_type;
static Uint32 dmaInbuff[BUFFER_SIZE]; /* buffer for DMA supporting devices */
static Uint32 edmaInbuff[BUFFER_SIZE]; /* buffer for EDMA supporting devices*/
IRQ_setVecs(vectors); /* point to the IRQ vector table */
element_count = ELEMENT_COUNT;
xfer_type = XFER_TYPE;
/* initialize the CSL library */
CSL_init();
init_mcbsp0_slave();
/* Enable sample rate generator GRST=1 */
MCBSP_enableSrgr(hMcbsp0);       /* Handle to SRGR */
switch (xfer_type) {
case DMA_XFER:
  #if (DMA_SUPPORT)       /* for DMA supporting devices */
  DMA_reset(INV);         /* reset all DMA channels */
  #endif
  #if (EDMA_SUPPORT)           /* for EDMA supporting devices */
  EDMA_clearPram(0x00000000); /* Clear PaRAM RAM of the EDMA */
  set_interrupts_edma();
  #endif
/*--------------------------------------------------------------------------*/
/* DMA channel 1 config structure                                           */
/*--------------------------------------------------------------------------*/
#if (DMA_SUPPORT)       /* for DMA supporting devices */
  /* Channel 1 receives the data */
  hDma1 = DMA_open(DMA_CHA1, DMA_OPEN_RESET);  /* Handle to DMA channel 1 */
    DMA_configArgs(hDma1,
       DMA_PRICTL_RMK(
           DMA_PRICTL_DSTRLD_DEFAULT,
           DMA_PRICTL_SRCRLD_DEFAULT,
           DMA_PRICTL_EMOD_DEFAULT,
           DMA_PRICTL_FS_DEFAULT,
           DMA_PRICTL_TCINT_ENABLE, /* TCINT =1                            */
           DMA_PRICTL_PRI_DMA,      /* DMA high priority                   */
```

```
            DMA_PRICTL_WSYNC_REVT0,  /* Set synchronization event REVT0=01101*/
            DMA_PRICTL_RSYNC_DEFAULT,
            DMA_PRICTL_INDEX_DEFAULT,
            DMA_PRICTL_CNTRLD_DEFAULT,
            DMA_PRICTL_SPLIT_DEFAULT,
            DMA_PRICTL_ESIZE_32BIT,  /* Element size 32 bits                */
            DMA_PRICTL_DSTDIR_DEFAULT,
            DMA_PRICTL_SRCDIR_INC,   /* Increment source by element size    */
            DMA_PRICTL_START_DEFAULT
            ),
         DMA_SECCTL_RMK(
            DMA_SECCTL_WSPOL_NA,    /* only available for 6202 and 6203 devices
*/
            DMA_SECCTL_RSPOL_NA,    /* only available for 6202 and 6203 devices
*/
            DMA_SECCTL_FSIG_NA,     /* only available for 6202 and 6203 devices
*/
            DMA_SECCTL_DMACEN_DEFAULT,
            DMA_SECCTL_WSYNCCLR_DEFAULT,
            DMA_SECCTL_WSYNCSTAT_DEFAULT,
            DMA_SECCTL_RSYNCCLR_DEFAULT,
            DMA_SECCTL_RSYNCSTAT_DEFAULT,
            DMA_SECCTL_WDROPIE_DEFAULT,
            DMA_SECCTL_WDROPCOND_DEFAULT,
            DMA_SECCTL_RDROPIE_DEFAULT,
            DMA_SECCTL_RDROPCOND_DEFAULT,
            DMA_SECCTL_BLOCKIE_ENABLE, /* BLOCK IE=1 enables DMA channel int */
            DMA_SECCTL_BLOCKCOND_DEFAULT,
            DMA_SECCTL_LASTIE_DEFAULT,
            DMA_SECCTL_LASTCOND_DEFAULT,
            DMA_SECCTL_FRAMEIE_DEFAULT,
            DMA_SECCTL_FRAMECOND_DEFAULT,
            DMA_SECCTL_SXIE_DEFAULT,
            DMA_SECCTL_SXCOND_DEFAULT
            ),
         DMA_SRC_RMK(MCBSP_ADDRH(hMcbsp0, DRR)),
         DMA_DST_RMK((Uint32)dmaInbuff),
         DMA_XFRCNT_RMK(
            DMA_XFRCNT_FRMCNT_DEFAULT,
            DMA_XFRCNT_ELECNT_OF(element_count)  /* set recv element count */
            )
);
set_interrupts_dma(); /* initialize the interrupt(s)                        */
                      /* enable the interrupt after the DMA channels are    */
                      /* opened as the DMA_OPEN_RESET clears and disables   */
                      /* the channel interrupt once specified and clears    */
                      /* the corresponding interrupt bits in the IER. This  */
                      /* is not applicable for the EDMA channel open case   */
DMA_start(hDma1);     /* Start DMA channel 1 */
#endif  /* end for dma supporting devices */
/*--------------------------------------------------------------------------*/
/* EDMA channel 13 config structure                                         */
/*--------------------------------------------------------------------------*/
```

```
  #if (EDMA_SUPPORT)      /* for EDMA supporting devices  */

hEdma1 = EDMA_open(EDMA_CHA_REVT0, EDMA_OPEN_RESET);
EDMA_configArgs(hEdma1,
#if (!C64_SUPPORT)
     EDMA_OPT_RMK(
        EDMA_OPT_PRI_HIGH,       /* High priority EDMA */
        EDMA_OPT_ESIZE_32BIT,    /* Element size 32 bits */
        EDMA_OPT_2DS_DEFAULT,
        EDMA_OPT_SUM_DEFAULT,
        EDMA_OPT_2DD_DEFAULT,
        EDMA_OPT_DUM_INC,        /* Destination increment by element size */
        EDMA_OPT_TCINT_YES,      /* Enable Transfer Complete Interrupt    */
        EDMA_OPT_TCC_OF(13),     /* TCCINT = 0xD, REVT0                   */
        EDMA_OPT_LINK_YES,       /* Enable linking to NULL table          */
        EDMA_OPT_FS_NO
        ),
#endif
#if (C64_SUPPORT)
     EDMA_OPT_RMK(
        EDMA_OPT_PRI_HIGH,       /* High priority EDMA */
        EDMA_OPT_ESIZE_32BIT,    /* Element size 32 bits */
        EDMA_OPT_2DS_DEFAULT,
        EDMA_OPT_SUM_DEFAULT,
        EDMA_OPT_2DD_DEFAULT,
        EDMA_OPT_DUM_INC,  /*  Destination  increment  by  element  size  */
        EDMA_OPT_TCINT_YES,  /*  Enable  Transfer  Complete  Interrupt  */
        EDMA_OPT_TCC_OF(13),     /* TCCINT = 0xD, REVT0                   */
        EDMA_OPT_TCCM_DEFAULT,
        EDMA_OPT_ATCINT_DEFAULT,
        EDMA_OPT_ATCC_DEFAULT,
        EDMA_OPT_PDTS_DEFAULT,
        EDMA_OPT_PDTD_DEFAULT,
        EDMA_OPT_LINK_YES,       /* Enable linking to NULL table          */
        EDMA_OPT_FS_NO
        ),
#endif
        EDMA_SRC_RMK(MCBSP_ADDRH(hMcbsp0, DRR)), /* rcv addr to edmaInbuff */
        EDMA_CNT_RMK(0, element_count), /* set count equal to element_count */
        EDMA_DST_RMK((Uint32)edmaInbuff), /* dst to DRR0                   */
        EDMA_IDX_RMK(0,0),
        EDMA_RLD_RMK(0,0)
        );
  hEdmadummy = EDMA_allocTable(-1); /* Dynamically allocates PaRAM RAM table*/
  EDMA_configArgs(hEdmadummy, /* Dummy or Terminating Table in PaRAM        */
         0x00000000,         /* Terminate EDMA transfers by linking to     */

         0x00000000,         /* this NULL table                            */
         0x00000000,
         0x00000000,
         0x00000000,
         0x00000000
         );
```

```
  EDMA_link(hEdma1, hEdmadummy); /* Link terminating event to the EDMA event*/
  EDMA_enableChannel(hEdma1);    /* Enable EDMA channel */
  #endif  /* end for EDMA supporting devices */
}
/* Enable McBSP channel */
MCBSP_enableRcv(hMcbsp0); /* McBSP port 0 as the transmitter         */
/* To flag an interrupt to the CPU when DMA transfer/receive is done */
#if (DMA_SUPPORT)
  while (!recv0_done);
#endif
/* To flag an interrupt to the CPU when EDMA transfer is done  */
/* Transfer completion interrupt 13 sets flag = 1 when set     */
#if (EDMA_SUPPORT)
  while (!recv0_done);
#endif
MCBSP_close(hMcbsp0);  /* close McBSP port */
#if (DMA_SUPPORT)      /* close DMA channels */
DMA_close(hDma1);
#endif
#if (EDMA_SUPPPORT)
EDMA_close(hEdma1);    /* close EDMA channel */
EDMA_close(hEdmadummy);
#endif
} /* end main, program ends here */
/*-------------------------------------------------------------------------*/
/* init_mcbsp0_slave()                                                     */
/*-------------------------------------------------------------------------*/
/* MCBSP Config structure */
/* Setup the MCBSP_0 for data receive */
void
init_mcbsp0_slave(void)
{
MCBSP_Config mcbspCfg0 = {
#if (EDMA_SUPPORT)
    MCBSP_SPCR_RMK(
                MCBSP_SPCR_FREE_DEFAULT,  /* All fields in SPCR set to default
values */
                MCBSP_SPCR_SOFT_DEFAULT,
            MCBSP_SPCR_FRST_DEFAULT,
            MCBSP_SPCR_GRST_DEFAULT,
            MCBSP_SPCR_XINTM_DEFAULT,
            MCBSP_SPCR_XSYNCERR_DEFAULT,
            MCBSP_SPCR_XRST_DEFAULT,
            MCBSP_SPCR_DLB_DEFAULT,
            MCBSP_SPCR_RJUST_DEFAULT,
            MCBSP_SPCR_CLKSTP_DEFAULT,
            MCBSP_SPCR_DXENA_DEFAULT,
            MCBSP_SPCR_RINTM_DEFAULT,
            MCBSP_SPCR_RSYNCERR_DEFAULT,
            MCBSP_SPCR_RRST_DEFAULT
            ),
#endif
#if (DMA_SUPPORT)
```

```
   MCBSP_SPCR_RMK(
          MCBSP_SPCR_FRST_DEFAULT,   /* All fields in SPCR set to default val-
ues */
          MCBSP_SPCR_GRST_DEFAULT,
          MCBSP_SPCR_XINTM_DEFAULT,
          MCBSP_SPCR_XSYNCERR_DEFAULT,
          MCBSP_SPCR_XRST_DEFAULT,
          MCBSP_SPCR_DLB_DEFAULT,
          MCBSP_SPCR_RJUST_DEFAULT,
          MCBSP_SPCR_CLKSTP_DEFAULT,
          MCBSP_SPCR_RINTM_DEFAULT,
          MCBSP_SPCR_RSYNCERR_DEFAULT,
          MCBSP_SPCR_RRST_DEFAULT
          ),
#endif
#if (EDMA_SUPPORT)
    MCBSP_RCR_RMK(
           MCBSP_RCR_RPHASE_SINGLE,   /* Single phase receive frame */
           MCBSP_RCR_RFRLEN2_DEFAULT,
           MCBSP_RCR_RWDLEN2_DEFAULT,
           MCBSP_RCR_RCOMPAND_DEFAULT,
           MCBSP_RCR_RFIG_DEFAULT,
           MCBSP_RCR_RDATDLY_1BIT,    /* 1-bit receive data delay   */
           MCBSP_RCR_RFRLEN1_DEFAULT,
           MCBSP_RCR_RWDLEN1_DEFAULT,
           MCBSP_RCR_RWDREVRS_DEFAULT
           ),
#endif
#if (DMA_SUPPORT)
    MCBSP_RCR_RMK(
           MCBSP_RCR_RPHASE_SINGLE,   /* Single phase receive frame */
           MCBSP_RCR_RFRLEN2_DEFAULT,
           MCBSP_RCR_RWDLEN2_DEFAULT,
           MCBSP_RCR_RCOMPAND_DEFAULT,
           MCBSP_RCR_RFIG_DEFAULT,
           MCBSP_RCR_RDATDLY_1BIT,    /* 1-bit receive data delay   */
           MCBSP_RCR_RFRLEN1_DEFAULT,
           MCBSP_RCR_RWDLEN1_DEFAULT
           ),
#endif
#if (EDMA_SUPPORT)
    MCBSP_XCR_RMK(
           MCBSP_XCR_XPHASE_DEFAULT,  /* All fields in XCR set to default val-
ues    */
           MCBSP_XCR_XFRLEN2_DEFAULT,
           MCBSP_XCR_XWDLEN2_DEFAULT,
           MCBSP_XCR_XCOMPAND_DEFAULT,
           MCBSP_XCR_XFIG_DEFAULT,
           MCBSP_XCR_XDATDLY_DEFAULT,
           MCBSP_XCR_XFRLEN1_DEFAULT,
           MCBSP_XCR_XWDLEN1_DEFAULT,
           MCBSP_XCR_XWDREVRS_DEFAULT
           ),
```

```
#endif
#if (DMA_SUPPORT)
    MCBSP_XCR_RMK(
          MCBSP_XCR_XPHASE_DEFAULT,  /* All fields in XCR set to default val-
ues    */
          MCBSP_XCR_XFRLEN2_DEFAULT,
          MCBSP_XCR_XWDLEN2_DEFAULT,
          MCBSP_XCR_XCOMPAND_DEFAULT,
          MCBSP_XCR_XFIG_DEFAULT,
          MCBSP_XCR_XDATDLY_DEFAULT,
          MCBSP_XCR_XFRLEN1_DEFAULT,
          MCBSP_XCR_XWDLEN1_DEFAULT
          ),
#endif
     MCBSP_SRGR_RMK(
          MCBSP_SRGR_GSYNC_DEFAULT, /* All fields in SRGR set to default val-
ues    */
          MCBSP_SRGR_CLKSP_DEFAULT,
          MCBSP_SRGR_CLKSM_DEFAULT,
          MCBSP_SRGR_FSGM_DEFAULT,
          MCBSP_SRGR_FPER_DEFAULT,
          MCBSP_SRGR_FWID_DEFAULT,
          MCBSP_SRGR_CLKGDV_DEFAULT
          ),
#if (C64_SUPPORT)
     MCBSP_MCR_RMK(                  /* only for 64x                        */
          MCBSP_MCR_XMCME_DEFAULT,  /* All fields in MCR set to default val-
ues    */
          MCBSP_MCR_XPBBLK_DEFAULT,
          MCBSP_MCR_XPABLK_DEFAULT,
          MCBSP_MCR_XMCM_DEFAULT,
          MCBSP_MCR_RPBBLK_DEFAULT,
          MCBSP_MCR_RMCME_DEFAULT,
          MCBSP_MCR_RPABLK_DEFAULT,
          MCBSP_MCR_RMCM_DEFAULT
          ),
#else
     MCBSP_MCR_RMK(
          MCBSP_MCR_XPBBLK_DEFAULT, /* All fields in MCR set to default val-
ues    */
          MCBSP_MCR_XPABLK_DEFAULT,
          MCBSP_MCR_XMCM_DEFAULT,
          MCBSP_MCR_RPBBLK_DEFAULT,
          MCBSP_MCR_RPABLK_DEFAULT,
          MCBSP_MCR_RMCM_DEFAULT
          ),
#endif
#if(!C64_SUPPORT)
     MCBSP_RCER_RMK(
          MCBSP_RCER_RCEB_DEFAULT,  /* All fields in RCER set to default val-
ues */
          MCBSP_RCER_RCEA_DEFAULT
          ),
```

```
#endif
#if(!C64_SUPPORT)
     MCBSP_XCER_RMK(
           MCBSP_XCER_XCEB_DEFAULT,  /* All fields in XCER set to default val-
ues */
           MCBSP_XCER_XCEA_DEFAULT
           ),
#endif
#if (C64_SUPPORT)
           MCBSP_RCERE0_RMK(0),      /* Additional registers only for 64x   */
           MCBSP_RCERE1_RMK(0),
           MCBSP_RCERE2_RMK(0),
           MCBSP_RCERE3_RMK(0),
#endif

#if (C64_SUPPORT)
           MCBSP_XCERE0_RMK(0),       /* Additional registers only for 64x  */
           MCBSP_XCERE1_RMK(0),
           MCBSP_XCERE2_RMK(0),
           MCBSP_XCERE3_RMK(0),
#endif
     MCBSP_PCR_RMK(
           MCBSP_PCR_XIOEN_DEFAULT,
           MCBSP_PCR_RIOEN_DEFAULT,
           MCBSP_PCR_FSXM_DEFAULT,
           MCBSP_PCR_FSRM_DEFAULT,
           MCBSP_PCR_CLKXM_DEFAULT,
           MCBSP_PCR_CLKRM_DEFAULT,
           MCBSP_PCR_CLKSSTAT_DEFAULT,
           MCBSP_PCR_DXSTAT_DEFAULT,
           MCBSP_PCR_FSXP_DEFAULT,
           MCBSP_PCR_FSRP_DEFAULT,
           MCBSP_PCR_CLKXP_DEFAULT,
           MCBSP_PCR_CLKRP_DEFAULT
           )
};
hMcbsp0 = MCBSP_open(MCBSP_DEV0, MCBSP_OPEN_RESET); /* McBSP port 0 */
MCBSP_config(hMcbsp0, &mcbspCfg0);
}
/*----------------------------------------------------------------------------*/
/* set_interrupts_dma()                                                       */
/*----------------------------------------------------------------------------*/
#if (DMA_SUPPORT)
void                          /* Set the interrupts */
set_interrupts_dma(void)      /* if the device supports DMA */
{
 IRQ_nmiEnable();
 IRQ_globalEnable();
 IRQ_disable(IRQ_EVT_DMAINT1);  /* INT09 */
 IRQ_clear(IRQ_EVT_DMAINT1);
 IRQ_enable(IRQ_EVT_DMAINT1);
 return;
}
```

```
#endif
/*-------------------------------------------------------------------------*/
/* set_interrupts_edma()                                                   */
/*-------------------------------------------------------------------------*/
#if (EDMA_SUPPORT)
void                          /* Set the interrupt           */
set_interrupts_edma(void)     /* if the device supports EDMA */
{
 IRQ_nmiEnable();
 IRQ_globalEnable();
 IRQ_reset(IRQ_EVT_EDMAINT);
 IRQ_disable(IRQ_EVT_EDMAINT);
 EDMA_intDisable(13);         /* ch 13 for McBSP receive event REVT0 */
 IRQ_clear(IRQ_EVT_EDMAINT);
 EDMA_intClear(13);
 IRQ_enable(IRQ_EVT_EDMAINT);
 EDMA_intEnable(13);
 return;
}
#endif
/*-------------------------------------------------------------------------*/
/*    DMA DATA TRANSFER COMPLETION ISRs                                    */
/*-------------------------------------------------------------------------*/
interrupt void     /* vecs.asm hooks this up to IRQ 09 */
c_int09(void)      /* DMA ch2                          */
{
recv0_done = TRUE;
return;
}
interrupt void    /* vecs.asm hooks this up to IRQ 08 */
c_int08(void)     /* for the EDMA                     */
{
 #if (EDMA_SUPPORT)
  if (EDMA_intTest(13))
  {
  recv0_done = TRUE;
  EDMA_intClear(13); /* clear CIPR bit so future interrupts can be recognized
*/
  }
#endif
return;
}
/*----------------------End of mcbsp_recv_slave.c--------------------------*/
*****************************************************************************
*           Copyright (C) 2000 Texas Instruments Incorporated.              *
*                          All Rights Reserved                              *
*---------------------------------------------------------------------------*
* FILENAME...... vecs.asm                                                   *
* DATE CREATED.. 06/27/2001                                                 *
*****************************************************************************
*---------------------------------------------------------------------------*
* Global symbols defined here and exported out of this file                 *
*---------------------------------------------------------------------------*
```

```
   .global _vectors
   .global _vector0
   .global _vector1
   .global _vector2
   .global _vector3
   .global _vector4
   .global _vector5
   .global _vector6
   .global _vector7
   .global _c_int08   ; Hookup the c_int08 ISR in main() for EDMA
   .global _c_int09   ; Hookup the c_int09 ISR in main() for DMA
   .global _vector10
   .global _vector11
   .global _vector12
   .global _vector13
   .global _vector14
   .global _vector15
*-----------------------------------------------------------------------------*
* Global symbols referenced in this file but defined somewhere else.          *
* Remember that your interrupt service routines need to be referenced here.   *
*-----------------------------------------------------------------------------*
   .ref _c_int00
*-----------------------------------------------------------------------------*
* This is a macro that instantiates one entry in the interrupt service table.*
*-----------------------------------------------------------------------------*
VEC_ENTRY .macro addr
    STW   B0,*--B15
    MVKL  addr,B0
    MVKH  addr,B0
    B B0
    LDW   *B15++,B0
    NOP   2
    NOP
    NOP
   .endm
*-----------------------------------------------------------------------------*
* This is a dummy interrupt service routine used to initialize the IST.       *
*-----------------------------------------------------------------------------*
_vec_dummy:
  B    B3
  NOP  5
*-----------------------------------------------------------------------------*
* This is the actual interrupt service table (IST). It is properly aligned    *
* and is located in the subsection .text:vecs. This means if you don't        *
* explicitly specify this section in your linker command file, it will        *
* default and link into the .text section. Remember to set the ISTP register *
* to point to this table.                                                     *
*-----------------------------------------------------------------------------*
 .sect ".text:vecs"
 .align 1024
_vectors:
_vector0:   VEC_ENTRY _vec_dummy
_vector1:   VEC_ENTRY _vec_dummy
```

```
_vector2:   VEC_ENTRY _vec_dummy
_vector3:   VEC_ENTRY _vec_dummy
_vector4:   VEC_ENTRY _vec_dummy
_vector5:   VEC_ENTRY _vec_dummy
_vector6:   VEC_ENTRY _vec_dummy
_vector7:   VEC_ENTRY _vec_dummy
_vector8:   VEC_ENTRY _c_int08    ; Hookup the c_int08 ISR in main() for EDMA
_vector9:   VEC_ENTRY _c_int09    ; Hookup the c_int09 ISR in main() for DMA
_vector10:  VEC_ENTRY _vec_dummy
_vector11:  VEC_ENTRY _vec_dummy
_vector12:  VEC_ENTRY _vec_dummy
_vector13:  VEC_ENTRY _vec_dummy
_vector14:  VEC_ENTRY _vec_dummy
_vector15:  VEC_ENTRY _vec_dummy
*--------------------------End of vecs.asm-----------------------------------*
```

IMPORTANT NOTICE AND DISCLAIMER

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

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

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

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