‘C6x McBSP Interface to a Single rate ST bus Device

APPLICATION REPORT: PRELIMINARY

Shaku Anjanaiah

Digital Signal Processing Solutions
April 1998

TEXAS INSTRUMENTS

IMPORTANT NOTICE

TRADEMARKS

CONTACT INFORMATION

US TMS320 HOTLINE	(281) 274-2320
US TMS320 FAX	(281) 274-2324
US TMS320 BBS	(281) 274-2323
US TMS320 email	dsph@ti.com

Contents

Figures

Tables

‘C6x McBSP Interface to a Single rate ST bus Device

Abstract

This document describes how the multi-channel buffered serial ports (McBSP) in the Texas Instruments (TI™) TMS320C6201 digital signal processor (DSP) is used to communicate to a ST bus compliant device.

The McBSP receives the framing signal, clock, and data from the ST bus device and processes them to generate internal frame syncs and clocks for correct data reception. The highly programmable features of the McBSP make it easy to interface to ST bus signals. This application report focuses on the single rate ST bus wherein the ST bus system clock and the data rate (number of bits per sec) are equal. Hence the name single rate, which applies only to a 2.048 MHz system clock. The usage of McBSP registers and sample code to perform the above function is described in this document.

Product Support

Related Documentation

The following list specifies product names, part numbers, and literature numbers of corresponding TI documentation.

- ☐ *TMS320C6201 Digital Signal Processor* data sheet, March 1998, Literature number SPRS051C
- ☐ *TMS320C6201/C6701 Peripherals* Reference Guide, March 1998, Literature number SPRU190B

World Wide Web

Our World Wide Web site at **www.ti.com** contains the most up to date product information, revisions, and additions. Users registering with TI&ME can build custom information pages and receive new product updates automatically via email.

Email

For technical issues or clarification on switching products, please send a detailed email to **dsph@ti.com**. Questions receive prompt attention and are usually answered within one business day.

Design Problem

How do I use the multi-channel buffered serial port in the TMS320C6201 to communicate to a single rate Serial Telecom™ (ST) Bus compliant device?

ST Bus Requirements

The ST bus [1] is a synchronous serial bus with data transfer rates of 2.048, 4.096, or 8.192 Mbps. The interface to a ST-bus device comprises the clock, frame, and data signals. These signals are available on the McBSP and are programmable, thereby making it a glue-less interface.

The ST bus data stream comprises of frames with a period of 125μs or a frame rate of 8000 frames per sec. This 8kHz sampling rate (twice the highest signal frequency in order to retain all the information in the stream) corresponds to the 3.5 to 4kHz voice band frequency. The frame signal indicates the start of a frame and each frame carries blocks of 8-bit data.

The clocks for ST bus data can be 2.048, 4.096, 8.192, or 16.384 MHz. Note that these clocks are always twice the data rate except for 2.048 MHz. Since the 2.048 MHz ST bus clock rate can also be the data rate, it is referred to as a Single Rate ST bus. An example of a Double rate ST bus would be a 2.048Mbps data stream clocked by a 4.096MHz clock. The following sections describe the hardware and software interface of the ‘C6x McBSP to a single rate ST bus device.[2]

McBSP Operation

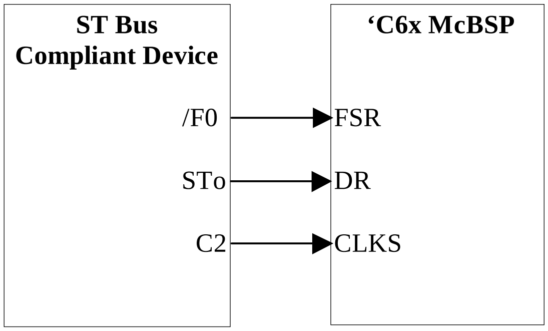
The ST bus compliant device that the McBSP is interfacing to is the master of frames and clock. This means that the ST bus device should provide a 2.048 MHz clock, C2, which becomes the external clock source to the McBSP via the CLKS pin. Also, the framing signal, /F0, generated by the ST bus device is used as the receive frame sync (FSR) input to the McBSP. The data transmitted on STo by the ST bus device is received on the DR pin of the McBSP. These connections are shown in Figure 1.

™ Mitel Semiconductor

[1] Reference: Mitel Application Note MSAN-126, ST-BUS Generic Device Specification (Rev. B)

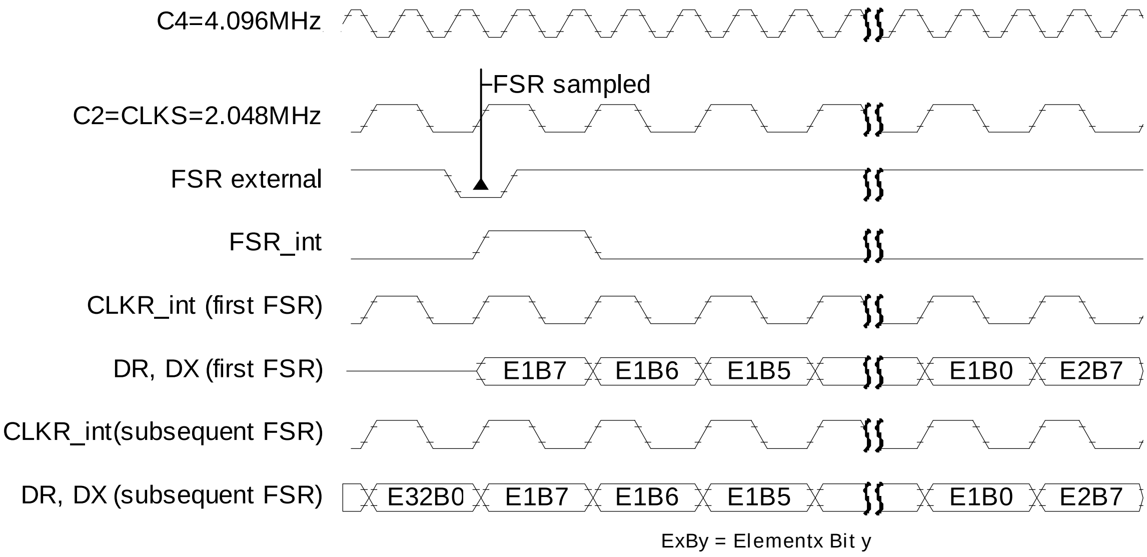
[2] It is recommended that the reader be familiar with the features of the ‘C6x McBSP by reading the TMS320C6201/C6701 Peripheral Reference Guide, especially Section 8.5.4

Figure 1. McBSP Connection for 2.048 MHz Single Rate ST Bus



In order for the McBSP to recognize a ST bus data stream, the GSYNC bit in the Sample Rate Generator Register has to be set. The ST bus-provided frame sync /F0 is an active low signal that is low for one half CLKS period. The GSYNC bit causes the external frame sync that arrives on FSR to be sampled on the rising edge of CLKS and in turn generates an internal frame sync in the McBSP that is active high for one CLKS clock period. This internal frame sync, FSR_int, is used as the reference for data reception. This is shown in Figure 2.

Figure 2. Single Rate ST Bus Example



In this 2.048MHz single rate ST bus example, the data stream comprises 32 elements of 8-bits each in each frame. Each new frame starts with a new frame sync signal /F0. Since the McBSP receiver does not know when the *first* frame will arrive on FSR, and therefore does not know when to get out of reset and receive data, special interrupts should be used. This is made easy with the new frame sync interrupt that is available on the McBSP and which works even when the receiver is in reset. The receive CPU interrupt (RINT) can be programmed to detect a new frame sync pulse, after which the CPU can safely take the receiver out of reset. Further initialization details are discussed in McBSP Initialization section in this document.

McBSP Register Configuration

As shown in Figure 1, FSR, CLKS, and DR are inputs.

- ***Framing Signal***: The polarity of the incoming frame sync signal (/F0) has to be inverted to provide the necessary active high input signal to the McBSP. The external frame sync pulse dictates the arrival of a new frame and therefore the frame period (FPER) and frame width (FWID) is not used/programmed.

  Although FSR is treated as input, FSRM bit in PCR and /FRST bit in SPCR has to be set to 1. /FRST bit has to be set to enable the frame sync signal generation. FSRM=1 indicates that the internally generated FSR_int will be used to detect the arrival of data, but will not be output on FSR pin because GSYNC=1 disables the output buffer.

  Since it is a single phase frame with each frame comprising 32 elements of 8-bits each, FRLEN1=31, and WDLEN1=0.

- ***Data Delay***: Since there is no delay between the arrival of the first bit of data and the generation of internal frame sync FSR_int, the receiver should be set up for a data delay of zero.

- ***Clocks***: Although CLKR pin of the McBSP is not used in this set up, it is necessary to configure it as an output (which is the default). Another important parameter is the polarity of CLKS signal. The CLKS polarity determines the edge that samples the incoming frame sync signal and also the edge that generates internal clocks CLKG, CLKR_int, CLKX_int, and internal frame sync signal FSG, FSR_int, and FSX_int. For the Single Rate ST bus case, rising edge of CLKS does the above function.

The various bit settings for the above requirements is shown in Figure 3, Figure 4, Figure 5, Figure 6, and Table 1.

Figure 3. Receive Control Register (RCR)

31	30 24	23 21	20 19	18	17 16
0	0	0	0	0	00
RPHASE	RFRLEN2	RWDLEN2	RCOMPAND	RFIG	RDATDLY

15	14 8	7 5	4 0
0	31	0	0
reserved	RFRLEN1	RWDLEN1	reserved

Figure 4. Sample Rate Generator Register (SRGR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	0	0	0	0											
GSYNC	CLKSP	CLKSM	FSGM	FPER											

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								0x00							
FWID								CLKGDV							

Figure 5. Pin Control Register (PCR)

31 16
0x0000
reserved

15 14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	0	0	0	1	0	0
rsv	XIOEN	RIOEN	FSXM	FSRM	CLKXM	CLKRM	rsv	CLKS_STAT	DX_STAT	DR_STAT	FSXP	FSRP	CLKXP	CLKRP

Figure 6. Serial Port Control Register (SPCR)

31 24	23	22	21 20	19	18	17	16
0x00	1	1	0	0	0	0	0
R, +0	FRST-	GRST-	XINTM	XSYNCERR	XEMPTY-	XRDY	XRST-

15	14 13	12 11	10 8	7	6	5 4	3	2	1	0
0	0	0	0	0	0	10	0	0	0	0
DLB	RJUST	CLKSTP	reserved	reserved	reserved	RINTM	RSYNCERR	RFULL	RRDY	RRST-

Table 1. Bit-Field Values for McBSP Registers

Register [bit-field #]	**Bit-field Name**	**Value (in binary)**
		Slave (Receiver)
RCR[17:16]	RDATDLY	0(default)
SPCR[5:4]	RINTM	10b
SPCR[23]	/FRST	1
SPCR[22]	/GRST	1
SRGR[31]	GSYNC	1
SRGR[29]	CLKSM	0(default)
SRGR[7:0]	CLKGDV	0(default)
PCR[10]	FSRM	1
PCR[8]	CLKRM	1
PCR[2]	FSRP	1

McBSP Initialization

Typical applications use the DMA to service the McBSP. The following steps describe the setup of interrupts, DMA, and the McBSP in the required order.

1. Program the Sample Rate Generator Register (SRGR), Serial Port Control Register (SPCR), Pin Control Register (PCR), and Receive Control Register (RCR) to the values shown above.

   Caution: Do not set the /GRST bit in SPCR in this step.

2. Take the sample rate generator out of reset by setting /GRST=1 in the SPCR.

3. Enabling Interrupts: To use interrupts, you have to set the Global Interrupt Enable (GIE), and Non-Maskable Interrupt Enable (NMIE) bits in the IER.

   Select the DMA channel you want to use. Enable CPU interrupts that correspond to the DMA channel that will be used to service the McBSP. 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

Since the McBSP has to be taken out of reset on the arrival of the very first frame sync, corresponding receive interrupt mode has to be selected in Step 1. Accordingly, RINT should be mapped to one of the CPU interrupts (say, CPU interrupt 13) and the corresponding enable bit set in the IER.

4. DMA initialization: Program the DMA channel for required operation. Following would be a typical set up:

   Source address = DRR

   Destination address = internal memory or as required.

   Transfer counter = number of elements to be transferred, typically at least 32.

   Receive synchronization event, *RSYNC* = REVT from McBSP

   DMA interrupt bit, *TCINT* = enabled

   Priority bit, *PRI* = 1; optional, but recommended.

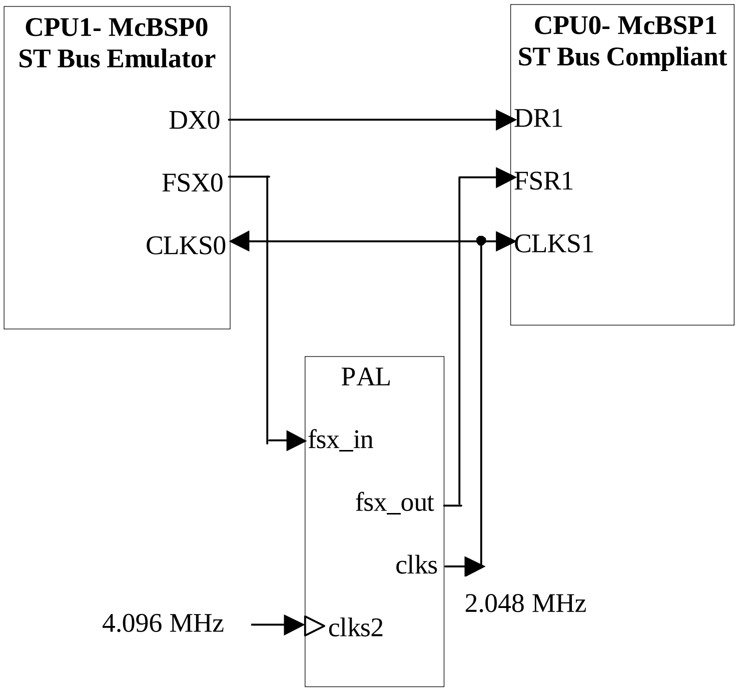
5. Instruct the DMA to run. For example, set START=01b in the DMA channel's primary control register to start the DMA without auto-initialization.
6. Now, the first frame sync that arrives on FSR will wake up the receiver. This is done in the ISR corresponding to CPU_INT13. The ISR should also disable this interrupt so that subsequent frame syncs do not cause unnecessary enabling of the receiver that has already been taken out of reset.

The receiver then provides the read sync event to the DMA, which causes transfer of data from DRR to specified destination address. The receiver will continue to receive data until the required number of frames have been received.

Sample Code Setup

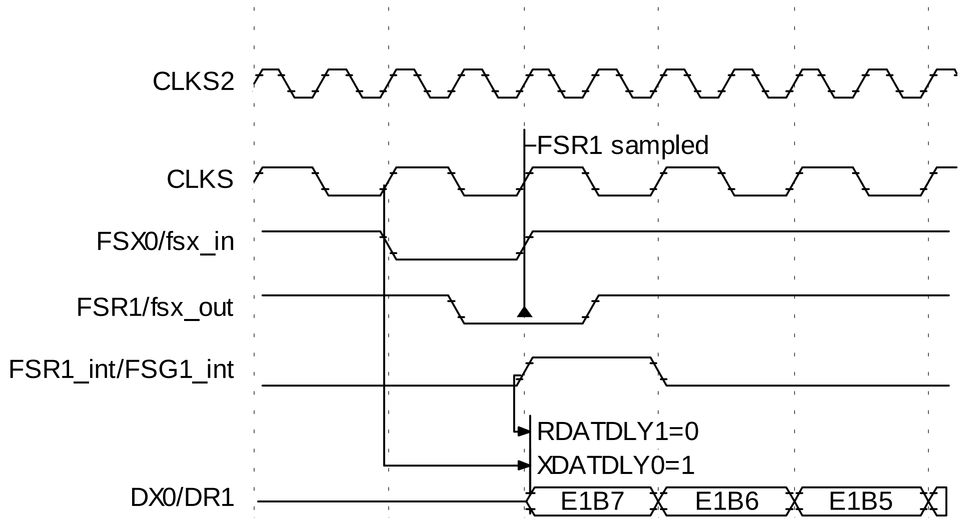
The example code in Appendix A through D was tested on two 'C6201 devices, CPU0 and CPU1, on a single board. The block diagram of this test setup is shown in Figure 7.

Figure 7 ST Bus emulator and the McBSP



McBSP0 in CPU1 is configured as the ST-bus transmitter (master) which provides the frame sync and data. The ST bus clock in this example is 2.048 MHz which is derived from a 4.096 MHz clock via a PAL. The PAL uses FSX0 (*fsx_in*) to generate the active low frame sync signal, *fsx_out*, at the appropriate rising edge of CLKS(0/1). Signal *fsx_out* is equivalent to FSR_ext signal shown in Figure 2. The PAL equations in VHDL are listed in Appendix E. The resulting signals due to the above set up is shown in Figure 8.

Figure 8 Timing Diagram for Example Single Rate ST Bus Setup



C Functions Overview

A sample C code that initializes the ‘C6x to interface to a ST bus device is enclosed in Appendices A through D. A brief description of some of the routines is described below:

```
init_M1_srgr();
init_M0_srgr();

/* Set up SRGR values in McBSP0/1 as needed */

init_m0to1();
init_m1to0();
/* Set up SPCR0/1, PCR0/1, XCR0, RCR1 registers in
CPU0_McBSP1 and CPU1_McBSP0 */

set_interrupts();
/* Maps interrupt sources to CPU interrupts.
Hooks the ISR to the relevant CPU interrupt.
Enables GIE, NMIE, CPU_INT8, and CPU_INT13. */

reg_dump();
/* Dumps the value of McBSP registers at the end of
data transfer. */

/* The following are Interrupt Service Routines */
interrupt void
c_int08(void)
/* This ISR is entered when DMA ch0 in McBSP1 (ST bus
slave) has completed data reception. Sets the
‘recv1_done’flag. */
```

```
interrupt void
c_int11(void)
/* This ISR is entered when DMA ch2 in McBSP0 (ST bus
emulator) has completed transmitting data. Sets the
‘xmit0_done’flag. */


interrupt void
c_int13(void)
/* This ISR is executed when the first frame sync
arrives. Wakes up the receiver (/RRST=1), sets the
‘new_fsr’ flag, disables this interrupt to prevent
unnecessary setting of /RRST bit. */
```

Conclusion

The ‘C6x McBSP offers ease of interface and ease of use to interface to a Single Rate ST bus. Although, double rate ST bus is not discussed in this document, it is a fairly straightforward interface too. This is discussed in Section 8.5.4.1 of the TMS320C6201/C6701 Peripheral Reference Guide.

Appendix A.

```
/*********************************************************************/
/* STB1_MASTER.C V1.00 */
/* Copyright (c) 1998 Texas Instruments Incorporated */
/*********************************************************************/
/*
  04/15/98: Shaku Anjanaiah

  stb1_master.c:
  McBSP0 (defined in stb1_master.c) acts as STbus emulator
  and McBSP1 (defined in stb1_slave.c) acts as single rate
  ST bus device.
*/
/*
This is a test to verify Single rate ST-bus operation using 2 C6201s.
This tcase configures CPU0_McBSP1 as ST bus master on the VDB.
FSX generated by the master has to be delayed (this is done in the
PAL) so that it is detected on the rising edge of CLKS.

CLKS drives both master and slave ST-bus devices.
*/

#include "common.h"

#define M0TO1     TRUE /* McBSP0 transmits to McBSP1 */
#define M1TO0     FALSE /* McBSP1 transmits to McBSP0 */

#define CLKGDV0         0
#define FPER0           511
#define FWID0           0

#define XFER_SIZE 128
#define XFER_TYPE DMA_STB

#define CLKSM0          CLK_MODE_CLKS
#define CLKSM1          CLK_MODE_CLKS

#define M0TO1_MSTR      TRUE
#define M1TO0_MSTR      TRUE

void init_m0to1(void);
void init_M0_srgr(void);

void
main(void)
{
      int     xfer_size;
      int     xfer_type;
      int     mcsp0to1_rate;
      int     mcsp1to0_rate;

      xmit1_done = mcsp1to0 ? FALSE : TRUE;
      recv0_done = mcsp1to0 ? FALSE : TRUE;
      xmit0_done = mcsp0to1 ? FALSE : TRUE;
      recv1_done = mcsp0to1 ? FALSE : TRUE;
      mcsp1to0 = M1TO0;
      mcsp0to1 = M0TO1;
      xfer_size = XFER_SIZE;
      xfer_type = XFER_TYPE;

      /* Set up SRGR values as needed */
      init_M0_srgr();

      /* Enable sample rate generator; /GRST=1 */
      MCBSP_SAMPLE_RATE_ENABLE(0);

      /* Now, initialize other control registers for McBSP operation */
      if (mcsp0to1)
            init_m0to1();
      if (mcsp1to0) { for (;;); }

      /* Reset all DMA channels */
      switch (xfer_type) {
      case DMA_STB:
            dma_reset();
            set_interrupts();
            /* Initialize DMA to service McBSP */
            if (mcsp0to1) {/* uses ch2 for xmit */
                  DMA2_SRC_ADDR = (unsigned int) out0;
                  DMA2_DEST_ADDR = MCBSP_DXR_ADDR(0);
                  REG_WRITE (DMA2_XFR_COUNTER_ADDR, xfer_size);

                  LOAD_FIELD (DMA2_PRIMARY_CTRL_ADDR, DMA_ESIZE32,
ESIZE, ESIZE_SZ);
                  SET_BIT (DMA2_PRIMARY_CTRL_ADDR, TCINT);
                  LOAD_FIELD (DMA2_PRIMARY_CTRL_ADDR, DMA_ADDR_INC,
SRC_DIR, SRC_DIR_SZ);
                  LOAD_FIELD (DMA2_PRIMARY_CTRL_ADDR, DMA_DMA_PRI, PRI,
1);
                  LOAD_FIELD (DMA2_PRIMARY_CTRL_ADDR, SEN_XEVT0, WSYNC,
WSYNC_SZ);
                  DMA_START(DMA_CH2);
            }
            if (mcsp1to0) { for (;;); }
            /* take transmitter (stb1 master) out of reset */
            SET_BIT (MCBSP_SPCR_ADDR(0), XRST);
            SET_BIT (MCBSP_SPCR_ADDR(0), FRST);

            while (!xmit0_done);
            break;
      }
      reg_dump();
      CSR |= 0x00007000;      /* PowerDown PD3 to shut off MCSP */
}


void
init_m0to1(void)
{
      /* PCR setup*/
      LOAD_FIELD (MCBSP_PCR_ADDR(0), FSYNC_POL_LOW, FSXP, 1); /* fsx
inverted */
      LOAD_FIELD (MCBSP_PCR_ADDR(0), M0TO1_MSTR, CLKXM, 1); /* clkx0 is
o/p */
      LOAD_FIELD (MCBSP_PCR_ADDR(0), FSYNC_MODE_INT, FSXM, 1); /* fsx0
is o/p */
      /* SRGR setup */
      LOAD_FIELD (MCBSP_SRGR_ADDR(0), FSX_FSG, FSGM, 1);
      /* XCR setup */
      LOAD_FIELD (MCBSP_XCR_ADDR(0), SINGLE_PHASE, XPHASE, 1);
      LOAD_FIELD (MCBSP_XCR_ADDR(0), WORD_LENGTH_8, XWDLEN1,
XWDLEN1_SZ);
      LOAD_FIELD (MCBSP_XCR_ADDR(0), 31, XFRLEN1, XFRLEN1_SZ);
      LOAD_FIELD (MCBSP_XCR_ADDR(0), DATA_DELAY1, XDATDLY, XDATDLY_SZ);
      LOAD_FIELD (MCBSP_XCR_ADDR(0), NO_COMPAND_MSB_1ST, XCOMPAND,
XCOMPAND_SZ);
}


void
init_M0_srgr(void)
{
      LOAD_FIELD (MCBSP_SRGR_ADDR(0), CLKGDV0, CLKGDV, CLKGDV_SZ);
      LOAD_FIELD (MCBSP_SRGR_ADDR(0), FWID0, FWID, FWID_SZ);
      LOAD_FIELD (MCBSP_SRGR_ADDR(0), FPER0, FPER, FPER_SZ);
      LOAD_FIELD (MCBSP_SRGR_ADDR(0), CLK_MODE_CLKS, CLKSM, 1);
      LOAD_FIELD (MCBSP_SRGR_ADDR(0), CLKS_POL_RISING, CLKSP, 1);
      LOAD_FIELD (MCBSP_SRGR_ADDR(0), GSYNC_OFF, GSYNC, 1);
}


void
reg_dump(void)
{
      int  i;
      int   *m0 = (int *) MCBSP_ADDR(0);
      int   *m1 = (int *) MCBSP_ADDR(1);

      for (i = 0; i < 10; i++) {
            regdump0[i] = m0[i];
            regdump1[i] = m1[i];
      }
}


void
set_interrupts(void)
{
      intr_init();
      /* Hook interrupt service routine to an interrupt */
      intr_hook (c_int11, CPU_INT11);

      /* enable NMIE, default CPU interrrupt 11 correponding to DMA
channel 2 */
```

```
      INTR_ENABLE(CPU_INT_NMI);/* Enable NMIE */
      INTR_GLOBAL_ENABLE; /* Set GIE in CSR*/
      INTR_ENABLE(11);
}


/* ================================= */
/* DMA DATA TRANSFER COMPLETION ISRS */
/* ================================= */

interrupt void
c_int11(void) /* DMA ch2 */
{
      xmit0_done = TRUE;
      return;
}
```

Appendix B.

```
/*********************************************************************/
/* STB1_SLAVE.C V1.00 */
/* Copyright (c) 1998 Texas Instruments Incorporated */
/*********************************************************************/
/*
  04/15/98: Shaku Anjanaiah

  stb1_slave.c:
  McBSP0 acts as STbus emulator and McBSP1 (defined in stb1_slave.c)
  acts as single rate ST bus device.
*/
/*
This is a test to verify Single rate ST-bus operation using 2 C6201s.
This tcase configures CPU1_McBSP0 as ST bus slave on the VDB.
External FSR is derived from FSX0 and some glue logic in the PAL
so that it is detected on the rising edge of CLKS.

CLKS drives both master and slave ST-bus devices.
*/

#include "common.h"

#define M0TO1     TRUE /* McBSP0 transmits to McBSP1 */
#define M1TO0     FALSE /* McBSP1 transmits to McBSP0 */

#define CLKGDV1         0
#define FPER1           0
#define FWID1           0

#define XFER_SIZE 128
#define XFER_TYPE DMA_STB

#define CLKSM0          CLK_MODE_CLKS
#define CLKSM1          CLK_MODE_CLKS

#define M0TO1_MSTR      TRUE
#define M1TO0_MSTR      TRUE

void init_m0to1(void);
void init_M1_srgr(void);
volatile int new_fsr;

void
main(void)
{
      int     xfer_size;
      int     xfer_type;

      xmit1_done = mcsp1to0 ? FALSE : TRUE;
      recv0_done = mcsp1to0 ? FALSE : TRUE;
      xmit0_done = mcsp0to1 ? FALSE : TRUE;
      recv1_done = mcsp0to1 ? FALSE : TRUE;
      new_fsr = FALSE;

      mcsp1to0 = M1TO0;
      mcsp0to1 = M0TO1;
      xfer_size = XFER_SIZE;
      xfer_type = XFER_TYPE;

      /* Set up SRGR values as needed */
      init_M1_srgr();

      /* Enable sample rate generator; /GRST=1 */
      MCBSP_SAMPLE_RATE_ENABLE(1);

      /* Now, initialize other control registers for McBSP operation */
      if (mcsp0to1)
            init_m0to1();
      if (mcsp1to0) { for (;;); }


      /* Reset all DMA channels */
      switch (xfer_type) {
      case DMA_STB:
            dma_reset();
            set_interrupts();

            /* Initialize DMA to service McBSP */
            if (mcsp0to1) {/* uses ch0 for recv */
                  DMA0_SRC_ADDR = MCBSP_DRR_ADDR(1);
                  DMA0_DEST_ADDR = (unsigned int) in1;
                  REG_WRITE (DMA0_XFR_COUNTER_ADDR, xfer_size);
                  LOAD_FIELD (DMA0_PRIMARY_CTRL_ADDR, DMA_ESIZE32,
ESIZE, ESIZE_SZ);
                  SET_BIT (DMA0_PRIMARY_CTRL_ADDR, TCINT);
                  LOAD_FIELD (DMA0_PRIMARY_CTRL_ADDR, DMA_ADDR_INC,
DST_DIR, DST_DIR_SZ);
                  LOAD_FIELD (DMA0_PRIMARY_CTRL_ADDR, DMA_DMA_PRI, PRI,
1);
                  LOAD_FIELD (DMA0_PRIMARY_CTRL_ADDR, SEN_REVT1, RSYNC,
RSYNC_SZ);
                  DMA_START(DMA_CH0);
            }
            if (mcsp1to0) { for (;;); }
            /* wait for first frame sync to enable receiver */
            while (!new_fsr);
            /* wait until all data is received */
            while (!recv1_done);
            break;
      }
      reg_dump();
      CSR |= 0x00007000;      /* PowerDown PD3 to shut off MCSP */
}

void
init_m0to1(void)
{
      /* PCR setup*/
      LOAD_FIELD (MCBSP_PCR_ADDR(1), FSYNC_POL_LOW, FSRP, 1);/* fsr
inverted */
      LOAD_FIELD (MCBSP_PCR_ADDR(1), M1TO0_MSTR, CLKRM, 1); /* clkr1 is
o/p */
      LOAD_FIELD (MCBSP_PCR_ADDR(1), FSYNC_MODE_INT, FSRM, 1); /* fsx0
is o/p */
      /* RCR setup */
      LOAD_FIELD (MCBSP_RCR_ADDR(1), SINGLE_PHASE, RPHASE, 1);
      LOAD_FIELD (MCBSP_RCR_ADDR(1), WORD_LENGTH_8, RWDLEN1,
RWDLEN1_SZ);
      LOAD_FIELD (MCBSP_RCR_ADDR(1), 31, RFRLEN1, RFRLEN1_SZ);
      LOAD_FIELD (MCBSP_RCR_ADDR(1), DATA_DELAY0, RDATDLY, RDATDLY_SZ);
      LOAD_FIELD (MCBSP_RCR_ADDR(1), NO_COMPAND_MSB_1ST, RCOMPAND,
RCOMPAND_SZ);
      /* SPCR */
      LOAD_FIELD (MCBSP_SPCR_ADDR(1), RXJUST_RJZF, RJUST, RJUST_SZ);
      LOAD_FIELD (MCBSP_SPCR_ADDR(1), INTM_FRAME, RINTM, RINTM_SZ);
      /* Very important to enable FRST so that internal FSR_int
      can be generated when GSYNC is ON */
      SET_BIT (MCBSP_SPCR_ADDR(1), FRST);
}

void
init_M1_srgr(void)
{
      LOAD_FIELD (MCBSP_SRGR_ADDR(1), CLKGDV1, CLKGDV, CLKGDV_SZ);
      LOAD_FIELD (MCBSP_SRGR_ADDR(1), FWID1, FWID, FWID_SZ);
      LOAD_FIELD (MCBSP_SRGR_ADDR(1), FPER1, FPER, FPER_SZ);
      LOAD_FIELD (MCBSP_SRGR_ADDR(1), CLKSM1, CLKSM, 1);
      LOAD_FIELD (MCBSP_SRGR_ADDR(1), CLKS_POL_RISING, CLKSP, 1);
      LOAD_FIELD (MCBSP_SRGR_ADDR(1), GSYNC_ON, GSYNC, 1);
}

void
reg_dump(void)
{
      int  i;
      int   *m0 = (int *) MCBSP_ADDR(0);
      int   *m1 = (int *) MCBSP_ADDR(1);

      for (i = 0; i < 10; i++) {
            regdump0[i] = m0[i];
            regdump1[i] = m1[i];
      }
}


void
set_interrupts(void)
{
      intr_init();
      INTR_MAP_RESET;
      intr_map(CPU_INT13, ISN_RINT1);
      /* Hook interrupt service routine to an interrupt */
      intr_hook (c_int08, CPU_INT8);
      intr_hook (c_int13, CPU_INT13);

      /* Enable NMIE and GIE */
      INTR_ENABLE(CPU_INT_NMI);
      INTR_GLOBAL_ENABLE;

      /* default CPU interrrupt 8 correponding to DMA channel 0 */
      INTR_ENABLE(8);
      /* enable new frame sync interrupt in McBSP1) */
      INTR_ENABLE(13);
}



/* DMA DATA TRANSFER COMPLETION ISRS */
interrupt void
c_int08(void) /* DMA ch0 */
{
      recv1_done = TRUE;
      return;
}

/* new frame sync interrupt wakes up the receiver */
interrupt void
c_int13(void)

{
      new_fsr = TRUE;
      SET_BIT (MCBSP_SPCR_ADDR(1), RRST);
      INTR_DISABLE(13);
      return;
}
```

Appendix C.

```
/*********************************************************************/
/* COMMON.H V1.00*/
/* Copyright (c) 1997 Texas Instruments Incorporated */
/*********************************************************************/

#include <dma.h>
#include <emif.h>
#include <intr.h>
#include <timer.h>
#include <cache.h>
#include <hpi.h>
#include <mcbsp.h>
#include <regs.h>
#include <stdio.h>
#include <trgcio.h>
#include <stdlib.h>

/* variables used in tcase */
int     mcsp0to1;
int     mcsp1to0;
volatile int xmit1_done;
volatile int recv0_done;
volatile int xmit0_done;
volatile int recv1_done;


#define FALSE       0
#define TRUE        1

/* BUFFERS DEFINED IN data6201.asm */

#define BUFFER_SIZE     256
#define COMPAND_SIZE    4096

extern int in0[BUFFER_SIZE];
extern int in1[BUFFER_SIZE];
extern int out0[BUFFER_SIZE];
extern int out1[BUFFER_SIZE];
extern int regdump0[10];
extern int regdump1[10];
extern int ulawenc[BUFFER_SIZE];

extern cregister volatile unsigned int AMR;
extern cregister volatile unsigned int CSR;
extern cregister volatile unsigned int IFR;
extern cregister volatile unsigned int ISR;
extern cregister volatile unsigned int ICR;
extern cregister volatile unsigned int IER;


extern interrupt void c_nmi01(void);
extern interrupt void c_int04(void);
extern interrupt void c_int05(void);
extern interrupt void c_int06(void);
extern interrupt void c_int07(void);
extern interrupt void c_int08(void);
extern interrupt void c_int09(void);
extern interrupt void c_int10(void);
extern interrupt void c_int11(void);
extern interrupt void c_int12(void);
extern interrupt void c_int13(void);
extern interrupt void c_int14(void);
extern interrupt void c_int15(void);

#define DMA_XFER  0
#define POLL_XFER 1
#define INT_XFER  2
#define GPIO            3
#define DLB1            4
#define DLB2            5
#define SPLIT_XFER      6
#define HW_BYTE  7
#define DMA_SPI         8
#define DMA_STB         9
#define DMA_MCM_FLY     10
#define DMA_NEW_FRAMESYNC     11
#define AUTO_INIT 12
#define DMA_SORT  13
#define SPLIT_SORT      14
#define DMA_SYNCERR     15

#define DMA_BYTE  16
#define DMA_HALFWORD    17

extern void set_interrupts(void);
extern void reg_dump(void);
```

Appendix D.

```
/**********************************************************************/
/* DATA6201.ASM V1.00*/
/* Copyright (c) 1997 Texas Instruments Incorporated */
/**********************************************************************/
      .global _in0, _in1, _out0, _out1
      .global _regdump0, _regdump1, _ulawenc

      .data
_out0:
      .eval   0, i
      .loop   128
      .word (i | 0x80888880)  ; sets bits to check sign extension
      .word i + 1             ; no sign extension
      .eval   i + 2, i
      .endloop
_out1:
      .eval   127, i
      .loop   128
      .word (i | 0x80888880)  ; sets bits to check sign extension
      .word i - 1             ; no sign extension
      .eval   i - 2, i
      .endloop
_in0:
      .loop   256
      .word 0xDEADFACE
      .endloop
_in1:
      .loop   256
      .word 0xDEADFACE
      .endloop

_regdump0:
      .loop 10
      .word 0xDEADFACE
      .endloop

_regdump1:
      .loop 10
      .word 0xDEADFACE
      .endloop

_ulawenc:
      .eval 0, i
      .loop 256
      .word i
      .eval i+1, i
      .endloop
```

Appendix E.

```
------------------------------------------------------------------------
˝
--
˝
-- Copyright: Texas Instruments, Inc.
˝
--
˝
--     TI Proprietary Information
˝
--            Internal Data
˝
--
------------------------------------------------------------------------
-- stb1b.vhd
--
-- Description : Generate a clean FSR using clks and clks
-- times 2 (clks2) signal. Added test bench.
--
-- Assumptions:
--
-- Operation:
--
-- Disclaimers:
--
-- Issues:
--
------------------------------------------------------------------------
--
--  Revision 1 5/11/98 Shaku Anjanaiah
--  Generate delayed FSX for single rate ST bus emulation
--
------------------------------------------------------------------------
--
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE WORK.std_arith.ALL;

entity stb1b_fsx is port(
            clks2: in std_logic;
            fsx_in: in std_logic;
            clks: out std_logic;
            fsx_out: out std_logic);
end stb1b_fsx;

architecture bhv of stb1b_fsx is
signal clks_q: std_logic;
signal clks_int: std_logic;

begin
clks_q <= NOT (clks_int);
DIVIDER     : PROCESS
-- Generate a divide-by-2 clock
      BEGIN
```

```
            wait until clks2'event AND clks2 = '1'; -- wait til rising
edge
            clks_int <= clks_q;
      END PROCESS DIVIDER;

clks <= clks_int;

LATCH : PROCESS
-- Generate fsx_out which is external FSR so that it
-- can be sampled on the rising edge of CLKS. Therefore,
-- rising edge of clks2 is used to latch fsx_in.
      BEGIN
            wait until clks2' event AND clks2 = '1';
            fsx_out <= fsx_in;
      END PROCESS LATCH;

end bhv;
```