Application Report
SPRA682 - September 2000

# T1 API for McBSP

*Mark Adams*
*Ramesh Iyer*

*Digital Signal Processing Solutions*

**ABSTRACT**

The multichannel buffered serial port (McBSP) is capable of directly interfacing to T1 framing devices and allows for individual channel selection. When used in conjunction with the DMA controller, multiple frames of selected channels can be sorted in memory before CPU intervention is needed. Such peripheral features reduce CPU interrupt latencies and leave precious DSP cycles free for other tasks.

The T1 API is designed to allow developers to take advantage of the McBSP and DMA for TDM streaming quickly and save time in development. By providing T1–E1 configuration parameters to the API through C callable routines the developer can focus on the application while the API configures the peripherals appropriately.

This report gives a brief overview of how the McBSP and DMA can handle TDM data, and provides detailed information on how to use the T1 API. A code example is given that uses the T1 API. For completeness, two hardware examples are given to show how to connect the McBSP to two common T1 framing devices.

**Contents**

**List of Figures**



**List of Tables**



# 1 McBSP Multichannel Functionality

The McBSP can operate in multichannel mode with a maximum of 128 channels. Individual channels may be selected for transmission and reception. If a channel is not selected, the McBSP will not generate an interrupt or synchronization event during that channel's time slot. This ensures that only selected channels will be processed while ignoring the others.

When the McBSP is synchronized to a DMA channel, selected channels can be gathered together in transmit or receive buffers and the entire buffer of channels serialized before the DSP is interrupted. Additionally, the DMA can sort channels from multiple frames into contiguous memory blocks. When the DMA has filled an entire buffer (ie. completed transfer of a block of frames), it issues an interrupt to the CPU. This approach significantly decreases interrupt latencies and processing time.

Table 1 depicts N channels grouped into M frames (one block). As an example assume that channels 0, 5 and 9 are selected and that the McBSP is synchronized to a DMA channel. Each time one of the selected channels is received in the McBSP the DMA transfers the channel element to a memory buffer. Since the DMA is configured for multiple frames, it will sort the incoming data by channel. The DMA issues an interrupt upon transfer of the last channel in a block.

**Table 1. Sorting Multiple Channels from Multiple Frames**

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 0 | 7 | 8 | 9 | 10 | …. | N–1 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 1 | 7 | 8 | 9 | 10 | …. | N–1 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 2 | 7 | 8 | 9 | 10 | …. | N–1 |
| … | … | … | … | … | … | … | … | … | … | … | … | …. | N–1 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | M–1 | 7 | 8 | 9 | 10 | …. | N–1 |

| |
|---|
| Frame 0 Channel 0 |
| Frame 1 Channel 0 |
| …. |
| Frame M–1 Channel 0 |
| Frame 0 Channel 5 |
| Frame 1 Channel 5 |
| … |
| Frame M–1 Channel 5 |
| Frame 0 Channel 9 |
| Frame 1 Channel 9 |
| …. |
| Frame M–1 Channel 9 |

Table 1 shows M frames of N channels, each row being a frame. A block is defined as M frames. The lower table shows the selected channels (0,5,9) sorted in memory.

At any given time only 32 of the available 128 channels may be enabled. Channels are divided into eight blocks (0–7) each block containing 16 contiguous channels. Even numbered blocks (0,2,4,6) belong to partition A while odd numbered blocks (1,3,5,7) belong to partition B.

Figure 1 is an illustrative example of what occurs during multi channel operation when synced with a DMA channel. This example assumes a 128 channel frame, and that even channels 32–46 are selected for reception and odd channels 33–35 are transmitted.

Figure 1, 1 frame, 128 channels, Select even channels 32–46 for Rx, Odd channels 33–35 for Tx. The down arrows indicate when a DMA transfer or DMA interrupt occurs.

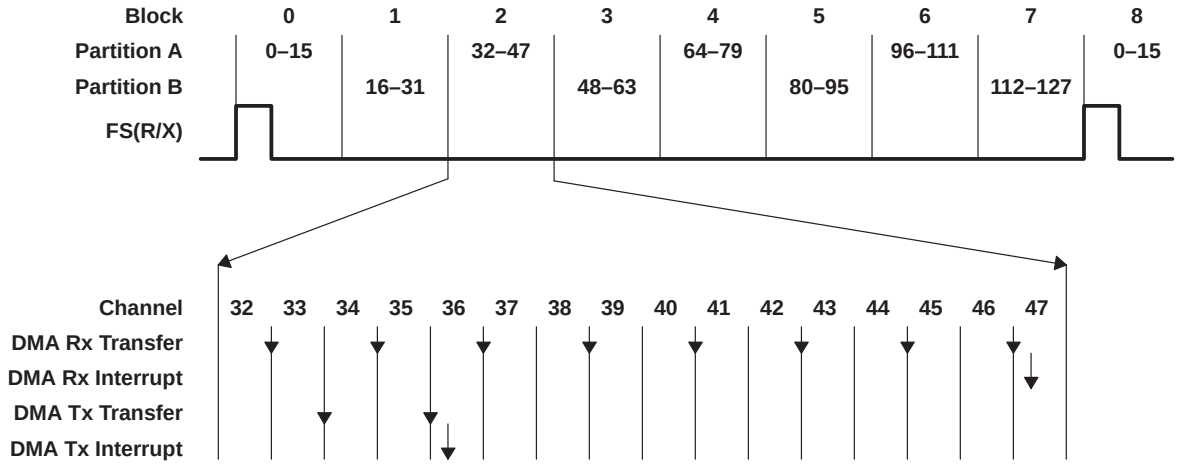


**Figure 1. Interrupt Timing Example**

## 2 How the API Works

The T1 API is designed to interface a DSP application to the McBSP and DMA peripherals in a simple and efficient manner. The API provides a user friendly C callable interface for T1/E1 streaming and handles the tedious task of setting up the peripheral control registers.

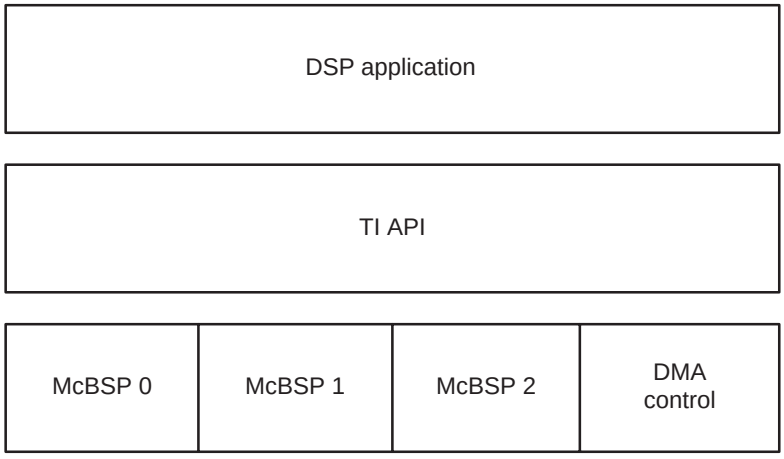


**Figure 2. Hierarchical Structue of the TDM API**

## 3 Double Buffering

The API manages four buffers for each T1 channel (two for rx, two for tx). Each buffer is designed to hold one block of information (a block consists of M frames). The application can access one while the McBSP and DMA operate on the other, alternating buffers every block.

# 4 Interrupts

There are two DMA interrupts that the user will be required to service: transmit complete and receive complete. Upon completion of transmitting or receiving an entire block of selected channels, the DMA will interrupt the CPU. The programmer must write an interrupt service routine to handle notification of a completed receive or transmit block of channel. During the DMA receive interrupt routine T1_init_rx( ) must be called. During the DMA transmit interrupt routine T1_init_tx( ) must be called. These two function re-initialize the DMA for the next block transfer.

# 5 API Overview

The T1 API consists of all the data structures and functions necessary to manage a T1 stream through a single McBSP and associated DMA channels. The T1 API manages the details of McBSP channel selection and DMA setup allowing developers to focus on the application. There may be as many instances of a T1 stream as there are McBSPs on a target device. The functions handle configuration control, selecting channels, and starting and stopping the T1 data stream.

# 6 API Structures

The **T1_CONFIG** structure contains all the fields the API needs to configure the McBSP and DMA. The following is a description of each field.

```
 typedef struct
{
         UINT16    mcbsp;
         TXRX_CFG  tx;
         TXRX_CFG  rx;
         UINT16    n_frames;
         UINT16    n_channels;
         UINT16    word_len;
         UINT16    pin_ctrl;
         UINT16    clk_gen;
         UINT16    frame_width;
         UINT16    clk_div;
         UINT16    dma_int_mux;
} T1_CONFIG;
```

## 6.1 McBSP Selection (mcbsp)

This field specifies which McBSP to use for T1 streaming. Possible choices are MCBSP0, MCBSP1 or MCBSP2. Note that not all C54x family members have 3 McBSPs.

## 6.2 Number of Frames (n_frames)

Specifies the number of frames in a block (See table 1). Valid values range from 1 to 128 frames.

## 6.3 Number of Channels (n_channels)

Specifies the number of channels in a frame (See table 1). Valid values range from 1 to 128 channels.

## 6.4 Word Length (word_len)

Defines the word length for tx/rx in bits. Possible values are LEN8, LEN12, LEN16, LEN20, LEN24 and LEN32, where the number represents length in bits.

## 6.5 Pin Control (pin_ctrl)

The pin control field configures the sources for the McBSP's clock pins (internal or external) and their polarity. One value for each mode must be selected. The pin control field is computed by a logical OR of all the desired values.

**Table 2. McBSP Frame Sync and Clock Configuration**

| Mode | Values | Description |
|---|---|---|
| FSXM (Transmit Frame Sync Mode) | FSXM_EXTERNAL, FSXM_INTERNAL | Tx frame sync source |
| FSRM (Receive Frame Sync Mode) | FSRM_EXTERNAL, FSRM_INTERNAL | Rx frame sync source |
| CLKXM (Transmitter Clock Mode) | CLKXM_EXTERNAL, CLKXM_INTERNAL | Tx clock source |
| CLKRM (Receiver Clock Mode) | CLKRM_EXTERNAL, CLKRM_INTERNAL | Rx clock source |
| FSXP | FSX_ACTIVE_HIGH, FSX_ACTIVE_LOW | Tx frame sync polarity |
| FSRP | FSR_ACTIVE_HIGH, FSR_ACTIVE_LOW | Rx frame sync polarity |
| CLKXP | CLKXP_RISING_EDGE, CLKXP_FALLING_EDGE | Tx clock polarity |
| CLXRP | CLKRP_RISING_EDGE, CLKRP_FALLING_EDGE | Rx clock polarity |

## 6.6 Clock Generator (clk_gen)

Frame and data clocks on the McBSP are generated from either the internal CPU clock or an external clock (CLKS pin) via the sample rate generator. The clock generation field specifies internal or external clock usage. For the case of external clocking, the polarity must also be specified.

**Table 3. McBSP Clock (Sample Rate Generator or External) Configuration**

| Mode | Values | Description |
|---|---|---|
| CLKSM (Sample rate generator clock mode) | CLKSM_EXTERNAL, CLKSM _INTERNAL | Sample rate generator source |
| CLKSP (Receive Frame Sync Mode) | CLKSP_RISING_EDGE, CLKSP_FALLING_EDGE | External clock source (CLKS) polarity |

## 6.7 Frame Width (frame_width)

Defines the width of the frame pulse when the McBSP is generating frame sync pulses. Possible values are 1 to 256, measured in bit clock cycles (CLKG).

## 6.8 Clock Divide Period (clk_div)

Sample rate generator clock divider. This value is used as the divide down number to generate the required bit clock (CLKG).

## 6.9 Multiplexed DMA Interrupt Selection (dma_int_mux)

Due to a limited number of interrupts in the 54x memory map, some DMA interrupts are multiplexed with other peripheral interrupts on the device. This field specifies which DMA interrupts should be available. Actual physical mappings vary by device, and can be found in the DMA controller documentation, as well as the which peripheral interrupt it is multiplexed with. This field affects INTOSEL in the DMPREC DMA register.

**Table 4. Multiplexed DMA Interrupt Assignment on VC5402**

| Value | Description |
|---|---|
| DMA_CH1_TO_CH5 (INTOSEL = 10b) | DMA Interrupts 1,2,3,4 and 5 mapped. (Maps 1,2 and 3 only on C5402) |
| DMA_CH2_TO_CH5 (INTOSEL = 01b) | DMA Interrupts 2,3,4 and 5 mapped. (Maps 2 and 3 only on C5402) |
| DMA_CH4_TO_CH5 (INTOSEL = 00b) | DMA Interrupts 4 and 5 mapped. (Not available on C5402). |

The **TXRX_CFG** structure defines configuration parameters that are unique to transmit and receive sections.

```
typedef struct
{
      UINT16      n_frames;
      UINT16      dma_chan;
      UINT16      dma_priority;
      UINT16      compand_mode;
      UINT16      data_delay;
} TXRX_CFG;
```

## 6.10 Number of Frames (n_frames)

Selects the number of frames in the block. Possible values are 1 to 128. This is related to the size the tx and rx buffers need to be.

## 6.11 DMA Channel (dma_chan)

Selects the DMA channel. There are six possible DMA channels (0–5) and 2 possible priorities, low and high.

**Table 5. DMA Channel Selection**

| Value | Description |
|---|---|
| DMA_CHAN_0 | Select DMA channel 0 |
| DMA_CHAN_1 | Select DMA channel 1 |
| DMA_CHAN_2 | Select DMA channel 2 |
| … | … |
| DMA_CHAN_5 | Select DMA channel 5 |

## 6.12 DMA Priority (dma_priority)

Selects the DMA channel priority. There are six possible DMA channels (0–5) and 2 possible priorities, low and high.

**Table 6. DMA Channel Priority Selection**

| Value | Description |
|---|---|
| DMA_CHAN_0_LOW | Select DMA channel 0, low priority |
| DMA_CHAN_0_HIGH | Select DMA channel 0, high priority |
| … | … |
| DMA_CHAN_5_LOW | Select DMA channel 5, low priority |
| DMA_CHAN_5_HIGH | Select DMA channel 5, high priority |

## 6.13 Companding Mode (compand_mode)

The McBSP can perform u-law or A-law companding internally. Note that word length must be 8 bits for companding to function properly. The following values define the companding mode.

**Table 7. McBSP Companding Mode Selection**

| Value | Description |
|---|---|
| MSB_FIRST | No companding. Data transfer starts with MSB first |
| LSB_FIRST | No companding. Data transfer starts with LSB first |
| ULAW | Compand using μ-law |
| ALAW | Compand using A-law |

## 6.14 Data Delay (data_delay)

The start of a frame is defined by the first clock cycle in which the frame synchronization is found to be active. The beginning of actual data reception or transmission with respect to the start of the frame can be delayed if required. This delay is called data delay. Typically, a 1-bit data delay is used as data often follows a once cycle active frame sync pulse. However, certain types of T1 framing devices insert framing bits in the data stream. During reception of such a stream a data delay of 2 is desirable: the framing bit appears after 1-bit delay and data appears after 2-bit delay. Therefore a delay of 2 causes the serial port to discard the framing bit. In transmission, by delaying the first transfer bit, the serial port inserts a blank period (high impedence) in place of the framing bit. Here it is expected that the framing device inserts its own framing bit.

**Table 8. McBSP Data Delay Selection**

| Value | Description |
|---|---|
| 0 | No data delay |
| 1 | 1-bit data delay |
| 2 | 2-bit data delay |

T1_BUFFER is simply a pointer to a buffer. The API requires two of these for transmit and two for reception.

Typedef UINT16* T1_BUFFER

The **T1_CHANNELS** structure tells the API what channels to select. The receiver and transmitter each have their own T1_CHANNELS structure.

```
typedef struct
{
      UINT16            num_chan_selected;
      UINT16            chan_bank_A;
      UINT16            chan_mask_A;
      UINT16            chan_bank_B;
      UINT16            chan_mask_B;
} T1_CHANNELS;
```

## 6.15 Number of Channels Selected (num_chan_selected)

The number of transmit or receive channels selected. Possible values range from 0 to 32.

## 6.16 Channel Bank Selection A (chan_bank_A)

Selects even channel banks. Channel bank selection and mask together choose which channels are enabled. Table 9 lists possible values.

**Table 9. McBSP Multichannel Mode Bank Selection A**

| Value | Description |
|---|---|
| CHAN_0_15 | Mask effects channels 0–15 |
| CHAN_32_47 | Mask effects channels 32–47 |
| CHAN_64_79 | Mask effects channels 64–79 |
| CHAN_96_111 | Mask effects channels 96–111 |

## 6.17 Channel Mask Selection A (chan_mask_A)

Masks the channels selected in the bank register. The lowest channel starts with the LSB. For example, if chan_bank_A = CHAN_0_15 and chan_mask_A = 0x0011 then channels 0 and 4 would be selected.

## 6.18 Channel Bank Selection B (chan_bank_B)

Selects odd channel banks. Channel bank selection and mask together choose which channels are enabled. Table 10 lists possible values.

**Table 10. McBSP Multichannel Mode Bank Selection B**

| Value | Description |
|---|---|
| CHAN_16_31 | Mask effects channels 16–31 |
| CHAN_48_63 | Mask effects channels 48–63 |
| CHAN_80_95 | Mask effects channels 80–95 |
| CHAN_112_127 | Mask effects channels 112–127 |

## 6.19 Channel Mask Selection B (chan_mask_B)

Masks the channels selected in the bank register. The lowest channel starts with the LSB. For example, if chan_bank_B = CHAN_16_31 and chan_mask_A = 0x0011 then channels 16 and 20 would be selected.

T1_STREAM is a structure defines everything needed for a single instance of a T1 stream.

```
Typedef struct
{
      T1_CONFIG         cfg;
      T1_BUFFER         txb1;
      T1_BUFFER         txb2;
      T1_BUFFER         rxb1;
      T1_BUFFER         rxb2;
      API_RESERVED      api;  // accessed by API only
} T1_STREAM;
```

## 6.20 API Functions

### *6.20.1 T1_config( T1_STREAM * )*

This function applies the information in T1_STREAM.cfg to configure the McBSP and DMA. The application should first write all the members of T1_STREAM.cfg as well as set the pointers to the transmit and receive buffers (txb1, txb2, rxb1, rxb2) before calling this function.

### *6.20.2 T1_init_tx( T1_STREAM * )*

This function is an extension of T1_config(). T1_init_tx() handles setting up the DMA for the next block of frames, and must be called immediately upon DMA tx interrupt. T1_init_tx() also handles any changes in channel selection.

### *6.20.3 T1_init_rx( T1_STREAM * )*

This function is an extension of T1_config(). T1_init_rx() handles setting up the DMA for the next block of frames, and must be called immediately upon DMA rx interrupt. T1_init_rx() also handles any changes in channel selection.

### *6.20.4 void T1_channel_sel( T1_STREAM*, T1_CHANNELS*, UINT16 sel )*

Call this function to enable/disable channels for transmit and receive. Channels may be changed “on the fly”, however, the API only makes channel changes at the end of each block – so changes to channel selection won’t occur until the next block of data.

Note that care must be taken to ensure that transmit and receive buffers are large enough to handle all the channels selected.

**Table 11. Description of API Structures**

| Parameter | Description |
|---|---|
| T1_STREAM * | Pointer to T1 stream affected. |
| T1_CHANNELS* | Pointer to channels structure that contains the new channel selection. |
| sel | Which channels in stream to change.<br>0 = TX, 1 = RX. |

### *6.20.5 T1_start( T1_STREAM * )*

Reset and start T1 frames, enabling interrupts. Enables required DMA interrupts and enables global interrupts.

### *6.20.6 T1_halt( T1_STREAM* )*

Halt T1 streaming, disabling interrupts. Disables required DMA interrupts, but does not disable global interrupts.

## 6.21 Making the API Work

T1 streaming is accomplished with the following steps:

**At compile time:**

1. Buffer allocation.
2. DMA interrupt service routines.

**At run time:**

1. Configure the T1 parameters (fill in T1_STREAM members).
2. Select the channels you want to tx/rx – call T1_chan_sel().
3. (Make sure buffers are allocated properly).
4. Call T1_config().
5. Start framing and interrupts – call T1_start().
6. Service DMA interrupts: during DMA tx ISR, call T1_init_tx() and in the DMA rx ISR call T1_init_rx(). These reset the DMA registers for the next block.

During framing channels selection may change by repeating step 2 without the need to stop and reconfigure the structure.

**Accessing Buffers:**

The program should always reference T1_STREAM members txbuf1 and rxbuf1 when accessing transmit and receive buffers. The API will use txbuf2 and rxbuf2 for DMA accesses. The API makes sure that txbuf1 and rxbuf1 point to user accessible buffers. Since the API changes txbuf1 and rxbuf1, applications must always access these pointers as members of the T1_STREAM data structure. Aliased versions of the pointer will not provide correct results.

**Interrupt Latencies:**

Because the DMA registers must be re-initialized after each transferred block, it is imperative that they are re-initialized before the DMA must transfer the next serial word. As a worse case example, consider the 2.048 Mbps E1 data stream. The application allows 8 bits clocks before the next DMA transfer will occur, or approximately 3.9 us. This means the DMA interrupts and calls to T1_init_tx() and T1_init_rx() must be able to occur in this time window. Careful consideration must be given to timings of other interrupts and/or non-interruptible code sections. T1_init_tx() requires consumes 84 clock cycles (including call and return) while T1_init_rx() requires 87 clock cycles.

## 6.22 T1 API Example Code

```
#include ”t1api.h”
void fill_tx_buffers();
#define NFRAMES         3
#define NCHANNELS 32
#define RX_BUFSIZE      (NFRAMES*NCHANNELS)
#define TX_BUFSIZE      (NFRAMES*NCHANNELS)
UINT16 rxbuf1[RX_BUFSIZE];
UINT16 rxbuf2[RX_BUFSIZE];
UINT16 txbuf1[TX_BUFSIZE];
UINT16 txbuf2[TX_BUFSIZE];
T1_STREAM t1;
UINT16 tx_test_val = 0;
UINT16 rx_test_val = 0;
UINT16 nRxBufsFail = 0;
UINT16 nTxBufs = 0;
UINT16 txflag = 0;
UINT16 rxflag = 0;
void main()
{
      UINT16 i;
    T1_CHANNELS ch;

    /* fill test buffers with something useful */
      fill_tx_buffers();

      /* assign double buffers to T1 stream    */
      t1.txbuf1 = txbuf1;
      t1.txbuf2 = txbuf2;
      t1.rxbuf1 = rxbuf1;
      t1.rxbuf2 = rxbuf2;
      /* fill in MsBSP and DMA cfg parameters  */
      t1.cfg.mcbsp = MCBSP0;

      t1.cfg.tx.dma_chan = DMA_CHAN_1;
      t1.cfg.tx.dma_priority = DMA_CHAN_1_LOW;
      t1.cfg.tx.compand_mode = MSB_FIRST;
      t1.cfg.tx.data_delay = 0;
      t1.cfg.rx.dma_chan = DMA_CHAN_2;
      t1.cfg.rx.dma_priority = DMA_CHAN_2_HIGH;
      t1.cfg.rx.compand_mode = MSB_FIRST;
      t1.cfg.rx.data_delay = 0;

      t1.cfg.n_frames = NFRAMES;
      t1.cfg.n_channels = NCHANNELS;
      t1.cfg.word_len = LEN8;

      t1.cfg.pin_ctrl = FSXM_INTERNAL |
                        FSRM_INTERNAL |
                        CLKXM_INTERNAL |
                        CLKRM_EXTERNAL |
```

```
                        FSX_ACTIVE_HIGH |
                        FSR_ACTIVE_HIGH |
                        CLKXP_RISING_EDGE |
                        CLKRP_FALLING_EDGE;
      t1.cfg.clk_gen = CLKSM_INTERNAL;

      t1.cfg.frame_width = 1;
      t1.cfg.clk_div = 48;
      t1.cfg.dma_int_mux = DMA_CH1_TO_CH5;

      /* Configure Tx channels (0–15, 16–31) */
      ch.num_chan_selected = 32;
      ch.chan_bank_A = CHAN_0_15;
      ch.chan_mask_A = 0xFFFF;
      ch.chan_bank_B = CHAN_16_31;
      ch.chan_mask_B = 0xFFFF;

      T1_channel_sel( &t1, &ch, 0 );

      /* Configure Rx channels (0–15, 16–31) */
      ch.num_chan_selected = 32;
      ch.chan_bank_A = CHAN_0_15;
      ch.chan_mask_A = 0xFFFF;
      ch.chan_bank_B = CHAN_16_31;
      ch.chan_mask_B = 0xFFFF;

    T1_channel_sel( &t1, &ch, 1 );
      /* Call API */
      T1_config( &t1 );
      T1_start( &t1 );

      while(1)
      {
            /* wait for interrupts */
            while(rxflag == 0 && txflag == 0);

            if(rxflag)
            {
            rxflag--;

            /* access rx buffer now (t1.rxbuf1)   */
            }

            if(txflag)
            {
              txflag--;

            /* access tx buffer now (t1.txbuf1)    */
            }
      }
}
```

```
void fill_tx_buffers()
{
      UINT16 i;

      for(i = 0; i < TX_BUFSIZE; i++)
      {
            txbuf1[i] = tx_test_val + i;
            txbuf2[i] = i + 1;
      }
      for(i = 0; i < RX_BUFSIZE; i++)
      {
            rxbuf1[i] = 0;
            rxbuf2[i] = 0;
      }
      rx_test_val = tx_test_val;
      tx_test_val += (1*2);
}
interrupt void dma_tx_isr()
{
      T1_init_tx(&t1);

      txflag++;
}
interrupt void dma_rx_isr()
{
      T1_init_rx(&t1);
      rxflag++;
}
```

## 6.23 T1 API Code Listing

The implementation of the T1 api consists of three files: t1apis.asm, t1api.c and t1api.h. These files should be compiled with optimization set at level –o0.

```
;------------------------------------------------------
;
; File:           t1apis.asm
; Copyright:      Texas Instruments, Inc.
;
;------------------------------------------------------
;------------------------------------------------------
;
; Function: T1_channel_sel( T1_STREAM* t1_strm,
;                           T1_CHANNELS* ch_sel
;                           UINT16 txrx_sel )
;
;------------------------------------------------------
      .global _T1_channel_sel

_T1_channel_sel:
      PSHM  AR1
      STLM  A,AR1       ; AR1 = &t1_strm
```

```
      LD    2h,A              ; A = &ch_sel
      LDU   3h,B              ; B = txrx_sel
      BC    TXSEL,BEQ
      LDM   AR1, B            ; B = &t1_strm
      BD    RXSEL
      ADD   #1Eh,0,B,B        ; B = t1_strm->api.rx_chans;
TXSEL:
      LDM   AR1, B            ; B = &t1_strm
      ADD   #19H,0,B,B        ; B = t1_strm->api.tx_chans;
RXSEL:
      STLM  B,AR1
      STLM  A,AR3
      NOP
      MVMM  AR1,AR2
      RPT   #4h                ; no need to diable interrupts,
      MVDD  *AR3+,*AR2+        ; RPT non-interruptable
      POPM  AR1
      RET


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

 File:            t1api.c
 Copyright: Texas Instruments, Inc.
-----------------------------------------------------*/
#include "t1api.h"
/*-----------------------------------------------------
 Function:  T1_config( T1_STREAM* t1_strm )

            Performs initializations for McBSP and DMA
            that must be done at startup.
-----------------------------------------------------*/
void T1_config( T1_STREAM* t1_strm )
{
      UINT16* subaddr_ptr;
      UINT16*     reg_ptr;
      UINT16 fper;

      subaddr_ptr = t1_strm->cfg.mcbsp;
      reg_ptr = t1_strm->cfg.mcbsp + 1;

      /*-------------------------------------------------
  Configure McBSP regs
      -------------------------------------------------*/
      /* SPCR1 */
      *subaddr_ptr = SPCR1;
      *reg_ptr = SPCR1_MASK_CFG;

      /* SPCR2 */
      *subaddr_ptr = SPCR2;
      *reg_ptr = SPCR2_MASK_CFG;
```

```
/* PCR */
*subaddr_ptr = PCR;
*reg_ptr = PCR_MASK_CFG | t1_strm->cfg.pin_ctrl;

/* RCR1 */
*subaddr_ptr = RCR1;
*reg_ptr =  RCR1_MASK_CFG |
            (t1_strm->cfg.n_channels-1 << 8) |
            (t1_strm->cfg.word_len << 5);

/* RCR2 */
*subaddr_ptr = RCR2;
*reg_ptr =  RCR2_MASK_CFG |
            (t1_strm->cfg.rx.compand_mode) |
            (t1_strm->cfg.rx.data_delay);

/* XCR1 */
*subaddr_ptr = XCR1;
*reg_ptr =  XCR1_MASK_CFG |
            (t1_strm->cfg.n_channels-1 << 8) |
            (t1_strm->cfg.word_len << 5);

/* XCR2 */
*subaddr_ptr = XCR2;
*reg_ptr =  XCR2_MASK_CFG |
            (t1_strm->cfg.tx.compand_mode) |
            (t1_strm->cfg.tx.data_delay);

/* SRGR1 */
*subaddr_ptr = SRGR1;
*reg_ptr =  t1_strm->cfg.frame_width - 1 |
                  t1_strm->cfg.clk_div;

/* SRGR2 */
switch( t1_strm->cfg.word_len )
{
      case LEN8:  fper = t1_strm->cfg.n_channels * 8;
                        break;
      case LEN12: fper = t1_strm->cfg.n_channels * 12;
                        break;
      case LEN16: fper = t1_strm->cfg.n_channels * 16;
                        break;
      case LEN20: fper = t1_strm->cfg.n_channels * 20;
                        break;
      case LEN24: fper = t1_strm->cfg.n_channels * 24;
                        break;
      case LEN32: fper = t1_strm->cfg.n_channels * 32;
                        break;
}
*subaddr_ptr = SRGR2;
```

```
*reg_ptr =  SRGR2_MASK_CFG |
                  (t1_strm->cfg.clk_gen) |
                  fper;
/* MCR1 */
*subaddr_ptr = MCR1;
*reg_ptr =  MCR1_MASK_CFG;

/* MCR2 */
*subaddr_ptr = MCR2;
*reg_ptr =  MCR2_MASK_CFG;

/*-------------------------------------------------
 API reserved setups (to save time in ISRs)
-------------------------------------------------*/

/* locate drr_reg in McBSP */
switch( (UINT16)(t1_strm->cfg.mcbsp) )
{
      case (UINT16)MCBSP0:    t1_strm->api.drr_reg = (UINT16*)DRR10;
                              break;
      case (UINT16)MCBSP1:    t1_strm->api.drr_reg = (UINT16*)DRR11;
                              break;
      case (UINT16)MCBSP2:    t1_strm->api.drr_reg = (UINT16*)DRR12;
                              break;
}
/* define tx sync event */
switch( (UINT16)(t1_strm->cfg.mcbsp) )
{
      case (UINT16)MCBSP0:    t1_strm->api.tx_sync_event = XEVT0;
                              break;
      case (UINT16)MCBSP1:    t1_strm->api.tx_sync_event = XEVT1;
                              break;
      case (UINT16)MCBSP2:    t1_strm->api.tx_sync_event = XEVT2;
                              break;
}

/* define rx sync event */
switch( (UINT16)(t1_strm->cfg.mcbsp) )
{
      case (UINT16)MCBSP0:    t1_strm->api.rx_sync_event = REVT0;
                              break;
      case (UINT16)MCBSP1:    t1_strm->api.rx_sync_event = REVT1;
                              break;
      case (UINT16)MCBSP2:    t1_strm->api.rx_sync_event = REVT2;
                              break;
}

T1_init_tx( t1_strm );
T1_init_rx( t1_strm );
```

```
      /*-------------------------------------------------
       Configure DMA regs
      -------------------------------------------------*/

    /* DMPREC */
    reg_ptr = (UINT16*)DMPREC;
    *reg_ptr =    *reg_ptr |
                  t1_strm->cfg.tx.dma_priority |
                  t1_strm->cfg.rx.dma_priority |
                  t1_strm->cfg.dma_int_mux;
      reg_ptr = (UINT16*)DMSBAR;
      subaddr_ptr = (UINT16*)DMABAI;

      *reg_ptr = DMSRCP;

      /* DMSRCP */
      *subaddr_ptr = 0;

      /* DMDSTP */
      *subaddr_ptr = 0;
      /* DMIDX0 */
      *subaddr_ptr = t1_strm->cfg.tx.n_frames;

      /* DMIDX1 */
      *subaddr_ptr = t1_strm->cfg.rx.n_frames;

}
/*-------------------------------------------------------
 Function:  T1_init_tx( T1_STREAM* t1_strm )

-------------------------------------------------------*/
void T1_init_tx( T1_STREAM* t1_strm )
{
      UINT16* subaddr_ptr;
      UINT16*     reg_ptr;
      /*-------------------------------------------------
       Configure tx McBSP regs
      -------------------------------------------------*/
      subaddr_ptr = t1_strm->cfg.mcbsp;
      reg_ptr = t1_strm->cfg.mcbsp + 1;

#if( PLATFORM != C5421 )
      /* XCERA */
      *subaddr_ptr = XCERA;
      *reg_ptr = t1_strm->api.tx_chans.chan_mask_A;

      /* XCERB */
      *subaddr_ptr = XCERB;
      *reg_ptr = t1_strm->api.tx_chans.chan_mask_B;
#endif
```

```
      /*-------------------------------------------------
       Configure tx DMA regs
      -------------------------------------------------*/
    /* DMPREC */
    reg_ptr = (UINT16*)DMPREC;
    *reg_ptr =    *reg_ptr |
                  t1_strm->cfg.tx.dma_priority;
      reg_ptr = (UINT16*)DMSBAR;
      subaddr_ptr = (UINT16*)DMABAI;
      *reg_ptr = t1_strm->cfg.tx.dma_chan;

      /* DMSRCx */
      *subaddr_ptr = (UINT16)(t1_strm->txbuf1);

      /* DMDSTx */
      *subaddr_ptr =  (UINT16)(t1_strm->api.drr_reg + 2);

      /* DMCTRx  - this is updated when channels are selected */
      *subaddr_ptr = t1_strm->api.tx_chans.num_chan_selected - 1;

      /* DMSFCx */
      *subaddr_ptr = t1_strm->api.tx_sync_event |
                           (t1_strm->cfg.tx.n_frames - 1);

      /* DMMCRx */
      *subaddr_ptr = DMMCR_TX_MASK_CFG;
      /* DMFRI0 */
      *reg_ptr = DMFRI0;
      *subaddr_ptr = -(t1_strm->cfg.tx.n_frames) *
                     (t1_strm->api.tx_chans.num_chan_selected-1) + 1;

      /* swap tx buffers */
      reg_ptr = t1_strm->txbuf1;
      t1_strm->txbuf1 = t1_strm->txbuf2;
      t1_strm->txbuf2 = reg_ptr;
}
/*-------------------------------------------------------
 Function:  T1_init_rx( T1_STREAM* t1_strm )

-------------------------------------------------------*/
void T1_init_rx( T1_STREAM* t1_strm )
{
      UINT16* subaddr_ptr;
      UINT16*     reg_ptr;
      /*-------------------------------------------------
       Configure tx McBSP regs
      -------------------------------------------------*/
      subaddr_ptr = t1_strm->cfg.mcbsp;
      reg_ptr = t1_strm->cfg.mcbsp + 1;
```

```
#if( PLATFORM != C5421 )
      /* RCERA */
      *subaddr_ptr = RCERA;
      *reg_ptr = t1_strm->api.rx_chans.chan_mask_A;

      /* RCERB */
      *subaddr_ptr = RCERB;
      *reg_ptr = t1_strm->api.rx_chans.chan_mask_B;
#endif
      /*-------------------------------------------------
       Configure rx DMA regs
      -------------------------------------------------*/

    /* DMPREC */
    reg_ptr = (UINT16*)DMPREC;
    *reg_ptr =    *reg_ptr |
                  t1_strm->cfg.rx.dma_priority;
      reg_ptr = (UINT16*)DMSBAR;
      subaddr_ptr = (UINT16*)DMABAI;
      *reg_ptr = t1_strm->cfg.rx.dma_chan;

      /* DMSRCx */
      *subaddr_ptr =  (UINT16)(t1_strm->api.drr_reg);

      /* DMDSTx */
      *subaddr_ptr = (UINT16)(t1_strm->rxbuf1);
      /* DMCTRx  - this is updated when channels are selected */
      *subaddr_ptr = t1_strm->api.rx_chans.num_chan_selected - 1;

      /* DMSFCx */
      *subaddr_ptr = t1_strm->api.rx_sync_event |
                              (t1_strm->cfg.rx.n_frames - 1);
      /* DMMCRx */
      *subaddr_ptr = DMMCR_RX_MASK_CFG;
      /* DMFRI1 */
      *reg_ptr = DMFRI1;
      *subaddr_ptr = -(t1_strm->cfg.rx.n_frames) *
                        (t1_strm->api.rx_chans.num_chan_selected-1) + 1;
      /* swap rx buffers */
      reg_ptr = t1_strm->rxbuf1;
      t1_strm->rxbuf1 = t1_strm->rxbuf2;
      t1_strm->rxbuf2 = reg_ptr;
}
/*-------------------------------------------------------
 Function:  T1_start( T1_STREAM* t1_strm )

-------------------------------------------------------*/
void T1_start( T1_STREAM* t1_strm )
{
      volatile UINT16* IMR = (UINT16*)IMR_REG;
      volatile UINT16* IFR = (UINT16*)IFR_REG;
```

```
UINT16* subaddr_ptr;
UINT16*     reg_ptr;
UINT16  mask = 0;

subaddr_ptr = t1_strm->cfg.mcbsp;
reg_ptr = t1_strm->cfg.mcbsp + 1;
/* Generate DMA interrupts mask and enable DMA ints */
switch( t1_strm->cfg.tx.dma_chan )
{
      case DMA_CHAN_0:  mask = DMACH0_INT;
                                      break;
      case DMA_CHAN_1:  mask = DMACH1_INT;
                                      break;
      case DMA_CHAN_2:  mask = DMACH2_INT;
                                      break;
      case DMA_CHAN_3:  mask = DMACH3_INT;
                                      break;
      case DMA_CHAN_4:  mask = DMACH4_INT;
                                      break;
      case DMA_CHAN_5:  mask = DMACH5_INT;
                                      break;
}
switch( t1_strm->cfg.rx.dma_chan )
{
      case DMA_CHAN_0:  mask |= DMACH0_INT;
                                      break;
      case DMA_CHAN_1:  mask |= DMACH1_INT;
                                      break;
      case DMA_CHAN_2:  mask |= DMACH2_INT;
                                      break;
      case DMA_CHAN_3:  mask |= DMACH3_INT;
                                      break;
      case DMA_CHAN_4:  mask |= DMACH4_INT;
                                      break;
      case DMA_CHAN_5:  mask |= DMACH5_INT;
                                      break;
}

*IFR = mask;      // clear pending DMA interrupts
*IMR |= mask;     // enable DMA interrupts

/* enable global interrupts – just in case */
asm(" RSBX INTM");
/* Start CLKG */
*subaddr_ptr = SPCR2;
*reg_ptr = SPCR2_MASK_STRT1;

/* Wait two bit clocks */
for(mask = 0; mask < 65000; mask++)
{
      asm(" nop");
```

```
            asm(" nop");
      }
      /* Pull Rx from reset */
      *subaddr_ptr = SPCR1;
      *reg_ptr = SPCR1_MASK_STRT;

      /* Pull Tx from reset */
      *subaddr_ptr = SPCR2;
      *reg_ptr = SPCR2_MASK_STRT2;

      /* FRST from reset */
      *subaddr_ptr = SPCR2;
      *reg_ptr = SPCR2_MASK_STRT3;

      /* Wait two bit clocks */
      for(mask = 0; mask < 65000; mask++)
      {
            asm(" nop");
            asm(" nop");
      }
}
/*-----------------------------------------------------

 File:            t1api.h
 Copyright: Texas Instruments, Inc.
-----------------------------------------------------*/
#ifndef __T1API__
#define __T1API__
/*-----------------------------------------------------
 T1 API Typedefs
-----------------------------------------------------*/
typedef unsigned int UINT16;
typedef UINT16 BOOL;
/*-----------------------------------------------------

 T1 API defines
-----------------------------------------------------*/
#define C5420           1       /* Older McBSP */
#define C5421           2       /* Newer McBSP */
#define PLATFORM  C5420
/*#define LOOPBACK      1*/
#define FALSE           0
#define TRUE            (!FALSE)
/*-----------------------------------------------------
 C54x Memory Mappes Regs
-----------------------------------------------------*/
#define IMR_REG         0
#define IFR_REG         1
/*-----------------------------------------------------
 McBSP Selection and Registers
```

```
-----------------------------------------------------*/
#define MCBSP0          ((UINT16*)0x38)
#define MCBSP1          ((UINT16*)0x48)
#define MCBSP2          ((UINT16*)0x34)
#define DXR10           0x23
#define DXR11           0x43
#define DXR12           0x33
#define DRR10           0x21
#define DRR11           0x41
#define DRR12           0x31
#define SPCR1           0x0
#define SPCR2           0x1
#define     RCR1        0x2
#define RCR2            0x3
#define XCR1            0x4
#define XCR2            0x5
#define SRGR1           0x6
#define SRGR2           0x7
#define MCR1            0x8
#define MCR2            0x9
#define RCERA           0xA
#define RCERB           0xB
#define XCERA           0xC
#define XCERB           0xD
#define PCR             0xE
/*-----------------------------------------------------
 DMA Registers
-----------------------------------------------------*/
#define DMPREC    0x54
#define DMSBAR    0x55
#define DMABAI    0x56
#define DMSBAN    0x57
#define DMSRCP    0x1e    // subaddr regs
#define DMDSTP  0x1f
#define DMIDX0    0x20
#define DMIDX1    0x21
#define DMFRI0    0x22
#define DMFRI1    0x23
/*-----------------------------------------------------
 McBSP/DMA Register Default Masks
-----------------------------------------------------*/
#ifdef LOOPBACK
#define SPCR1_MASK_CFG  0x8010      // DLB = 1, RJUST = 00
                                    // CLKSTP = 00, DXENA = 0
                                    // ABIS = 0, RINTM = 01
                                    // RSYNCERR = 0, RRST/ = 0
#else
#define SPCR1_MASK_CFG  0x0010      // DLB = 0, RJUST = 00
                                    // CLKSTP = 00, DXENA = 0
                                    // ABIS = 0, RINTM = 01
                                    // RSYNCERR = 0, RRST/ = 0
```

```
#endif

#define SPCR2_MASK_CFG  0x0010      // FREE = 0, SOFT = 0
                                    // FRST/ = 0, GRST/ = 0
                                    // XINTM = 01
                                    // XSYNCERR = 0, XRST/ = 0

#define PCR_MASK_CFG    0x0000      // XIOEN = 0, RIOEN = 0
#define RCR1_MASK_CFG   0x0000      // No care abouts (user define)
#define RCR2_MASK_CFG   0x0000      // RPHASE = 0, FRFLEN2 = 0
                                    // RWDLEN2 = 0, RFIG = 0
#define XCR1_MASK_CFG   0x0000      // No care abouts (user define)
#define XCR2_MASK_CFG   0x0000      // XPHASE = 0, XRFLEN2 = 0
                                    // XWDLEN2 = 0, XFIG = 0

#define SRGR2_MASK_CFG  0x9000      // GSYNC = 1, FSGM = 1
#define MCR1_MASK_CFG   0x0001      // RPBBLK = 00, RPABLK = 00
                                    // RMCM = 1

#define MCR2_MASK_CFG   0x0001      // XPABLK = 00, XPBBLK = 00
                                    // XMCM = 1

#ifdef LOOPBACK
#define SPCR1_MASK_STRT 0x8011      // DLB = 1, RJUST = 00
                                    // CLKSTP = 00, DXENA = 0
                                    // ABIS = 0, RINTM = 01
                                    // RSYNCERR = 0, RRST/ = 1
#else
#define SPCR1_MASK_STRT 0x0011      // DLB = 0, RJUST = 00
                                    // CLKSTP = 00, DXENA = 0
                                    // ABIS = 0, RINTM = 01
                                    // RSYNCERR = 0, RRST/ = 1
#endif
#define SPCR2_MASK_STRT1 0X0050     // FREE = 0, SOFT = 0
                                    // FRST/ = 0, GRST/ = 1
                                    // XINTM = 01
                                    // XSYNCERR = 0, XRST/ = 0

#define SPCR2_MASK_STRT2 0X0051     // FREE = 0, SOFT = 0
                                    // FRST/ = 0, GRST/ = 1
                                    // XINTM = 01
                                    // XSYNCERR = 0, XRST/ = 1
#define SPCR2_MASK_STRT3 0X00D1     // FREE = 0, SOFT = 0
                                    // FRST/ = 1, GRST/ = 1
                                    // XINTM = 01
                                    // XSYNCERR = 0, XRST/ = 1
#define DMMCR_TX_MASK_CFG     0x4541
                                    // AUTOINIT = 0, DINM = 1
                                    // IMOD = 0, CTMOD = 0,
                                    // SIND = 101, DMS = 01
                                    // DIND = 000, DMD = 01
```

```
#define DMMCR_RX_MASK_CFG     0x4059
                                    // AUTOINIT = 0, DINM = 1
                                    // IMOD = 0, CTMOD = 0
                                    // SIND = 000, DMS = 01
                                    // DIND = 110, DMD = 01

/*-----------------------------------------------------
 Word Length Selection
-----------------------------------------------------*/
#define LEN8      (0x0 << 5)
#define LEN12     (0x1 << 5)
#define LEN16     (0x2 << 5)
#define LEN20     (0x3 << 5)
#define LEN24     (0x4 << 5)
#define LEN32     (0x5 << 5)
/*-----------------------------------------------------
 Companding Mode Selection
-----------------------------------------------------*/
#define MSB_FIRST (0x0 << 3)
#define LSB_FIRST (0x1 << 3)
#define ULAW            (0x2 << 3)
#define ALAW            (0x3 << 3)
/*-----------------------------------------------------
 McBSP Clock Pin Configuration
-----------------------------------------------------*/
#define FSXM_EXTERNAL         (0x0 << 11)
#define FSXM_INTERNAL         (0x1 << 11)
#define FSRM_EXTERNAL         (0x0 << 10)
#define FSRM_INTERNAL         (0x1 << 10)
#define CLKXM_EXTERNAL        (0x0 << 9)
#define CLKXM_INTERNAL        (0x1 << 9)
#define CLKRM_EXTERNAL        (0x0 << 8)
#define CLKRM_INTERNAL        (0x1 << 8)
#define FSX_ACTIVE_HIGH       (0x0 << 3)
#define FSX_ACTIVE_LOW        (0x1 << 3)
#define FSR_ACTIVE_HIGH       (0x0 << 2)
#define FSR_ACTIVE_LOW        (0x1 << 2)
#define CLKXP_RISING_EDGE     (0x0 << 1)
#define CLKXP_FALLING_EDGE    (0x1 << 1)
#define CLKRP_RISING_EDGE     (0x1 << 0)
#define CLKRP_FALLING_EDGE    (0x0 << 0)
/*-----------------------------------------------------
 CLKS Configuration
-----------------------------------------------------*/
#define CLKSM_EXTERNAL        (0x0 << 13)
#define CLKSM_INTERNAL        (0x1 << 13)
#define CLKSP_RISING_EDGE     (0x0 << 14)
#define CLKSP_FALLING_EDGE    (0x1 << 14)
```

```
/*-----------------------------------------------------
 DMA Enable & Priority Selection
-----------------------------------------------------*/
#define DMA_CHAN_0_ENABLE     0x1
#define DMA_CHAN_1_ENABLE     0x2
#define DMA_CHAN_2_ENABLE     0x4
#define DMA_CHAN_3_ENABLE     0x8
#define DMA_CHAN_4_ENABLE     0x10
#define DMA_CHAN_5_ENABLE     0x20
#define DMA_CHAN_0_LOW        (0x0 << 8 | DMA_CHAN_0_ENABLE)
#define DMA_CHAN_0_HIGH       (0x1 << 8 | DMA_CHAN_0_ENABLE)
#define DMA_CHAN_1_LOW        (0x0 << 9 | DMA_CHAN_1_ENABLE)
#define DMA_CHAN_1_HIGH       (0x1 << 9 | DMA_CHAN_1_ENABLE)
#define DMA_CHAN_2_LOW        (0x0 << 10 | DMA_CHAN_2_ENABLE)
#define DMA_CHAN_2_HIGH       (0x1 << 10 | DMA_CHAN_2_ENABLE)
#define DMA_CHAN_3_LOW        (0x0 << 11 | DMA_CHAN_3_ENABLE)
#define DMA_CHAN_3_HIGH       (0x1 << 11 | DMA_CHAN_3_ENABLE)
#define DMA_CHAN_4_LOW        (0x0 << 12 | DMA_CHAN_4_ENABLE)
#define DMA_CHAN_4_HIGH       (0x1 << 12 | DMA_CHAN_4_ENABLE)
#define DMA_CHAN_5_LOW        (0x0 << 13 | DMA_CHAN_5_ENABLE)
#define DMA_CHAN_5_HIGH       (0x1 << 13 | DMA_CHAN_5_ENABLE)
#define DMA_CHAN_0            0x0
#define DMA_CHAN_1            0x5
#define DMA_CHAN_2            0xA
#define DMA_CHAN_3            0xF
#define DMA_CHAN_4            0x14
#define DMA_CHAN_5            0x19
/*-----------------------------------------------------
 DMA Sync Events
-----------------------------------------------------*/
#define REVT0           (0x1 << 12)
#define XEVT0           (0x2 << 12)
#define REVT1           (0x5 << 12)
#define XEVT1           (0x6 << 12)
#define REVT2           (0x3 << 12)
#define XEVT2           (0x4 << 12)
/*-----------------------------------------------------
 DMA Multiplexed Pin Selection
-----------------------------------------------------*/
#define DMA_CH4_TO_CH5  (0x0 << 6)
#define DMA_CH2_TO_CH5  (0x1 << 6)
#define DMA_CH1_TO_CH5  (0x2 << 6)
/*-----------------------------------------------------
 DMA Interrupt masks
-----------------------------------------------------*/
#define DMACH0_INT      (0x0 << 6)
#define DMACH1_INT      (0x1 << 7)
#define DMACH2_INT      (0x1 << 10)
#define DMACH3_INT      (0x1 << 11)
#define DMACH4_INT      (0x1 << 12)
#define DMACH5_INT      (0x1 << 13)
```

```
/*-------------------------------------------------------
 Channel Selection Indeces & Bit Masks
-------------------------------------------------------*/
#define MAX_NUM_CHANNELS      128
#define CHAN_0_15             0
#define CHAN_16_31            1
#define CHAN_32_47            2
#define CHAN_48_63            3
#define CHAN_64_79            4
#define CHAN_80_95            5
#define CHAN_96_111           6
#define CHAN_112_127          7
#define CHAN_0                0x0001
#define CHAN_1                0x0002
#define CHAN_2                0x0004
#define CHAN_3                0x0008
#define CHAN_4                0x0010
#define CHAN_5                0x0020
#define CHAN_6                0x0040
#define CHAN_7                0x0080
#define CHAN_8                0x0100
#define CHAN_9                0x0200
#define CHAN_10               0x0400
#define CHAN_11               0x0800
#define CHAN_12               0x1000
#define CHAN_13               0x2000
#define CHAN_14               0x4000
#define CHAN_15               0x8000

#define CHAN_16               0x0001
#define CHAN_17               0x0002
#define CHAN_18               0x0004
#define CHAN_19               0x0008
#define CHAN_20               0x0010
#define CHAN_21               0x0020
#define CHAN_22               0x0040
#define CHAN_23               0x0080
#define CHAN_24               0x0100
#define CHAN_25               0x0200
#define CHAN_26               0x0400
#define CHAN_27               0x0800
#define CHAN_28               0x1000
#define CHAN_29               0x2000
#define CHAN_30               0x4000
#define CHAN_31               0x8000
```

```
/*-------------------------------------------------------

 T1 API structures
-------------------------------------------------------*/
typedef struct
{
      UINT16            n_frames;
/* number of frames in a block */
      UINT16            dma_chan;
      UINT16            dma_priority;
      UINT16            compand_mode;
      UINT16            data_delay;
} TXRX_CFG;
typedef struct
{
      UINT16*           mcbsp;
      TXRX_CFG    tx;
      TXRX_CFG    rx;
      UINT16            n_channels;
/* number of channels in a frame */
      UINT16            word_len;
      UINT16            pin_ctrl;
      UINT16            clk_gen;
      UINT16            frame_width;
      UINT16            clk_div;
      UINT16            dma_int_mux;
} T1_CONFIG;
#if (PLATFORM == C5421)
typedef struct  /* All 128 channels may be selected on new McBSP */
{
      UINT16            num_chan_selected;
      UINT16            chan_sel[MAX_NUM_CHANNELS >> 4];
      BOOL        locked;     // Locked when selecting channels, init FALSE
} T1_CHANNELS;
#else             /* only 32 channels may be selected on the old McBSP */
typedef struct
{
      UINT16            num_chan_selected;
      UINT16            chan_bank_A;
      UINT16            chan_mask_A;
      UINT16            chan_bank_B;
      UINT16            chan_mask_B;
} T1_CHANNELS;
#endif

typedef struct
{
      UINT16*           drr_reg;
      UINT16            tx_sync_event;
      UINT16            rx_sync_event;
```

```
      T1_CHANNELS tx_chans;
      T1_CHANNELS rx_chans;
} API_RESERVED;

typedef struct
{
      UINT16*           txbuf1;
      UINT16*           txbuf2;
      UINT16*           rxbuf1;
      UINT16*           rxbuf2;
      T1_CONFIG         cfg;
      API_RESERVED      api;  // accessed by API only
} T1_STREAM;
/*-----------------------------------------------------

 T1 API function declarations

-----------------------------------------------------*/
void T1_config( T1_STREAM * );
void T1_init_tx( T1_STREAM * );
void T1_init_rx( T1_STREAM *);
void T1_channel_sel( T1_STREAM *, T1_CHANNELS *, UINT16 sel  );
void T1_start( T1_STREAM* );
void T1_halt( T1_STREAM* );
#endif
```

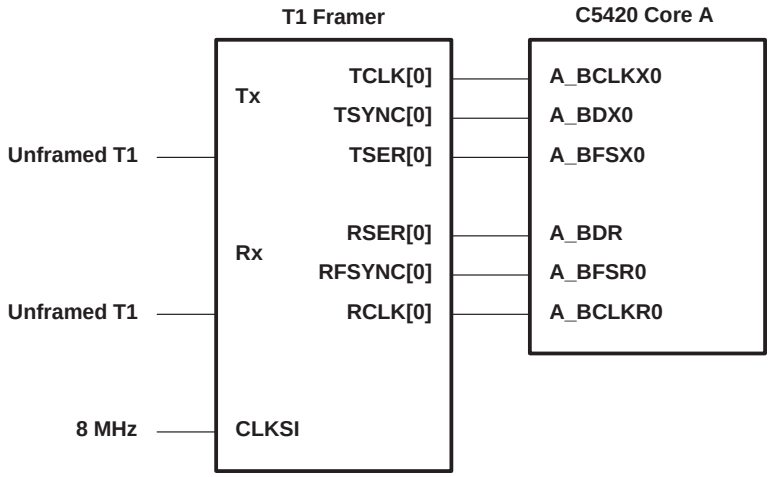


**Figure 3. McBSP to Framer Examples**

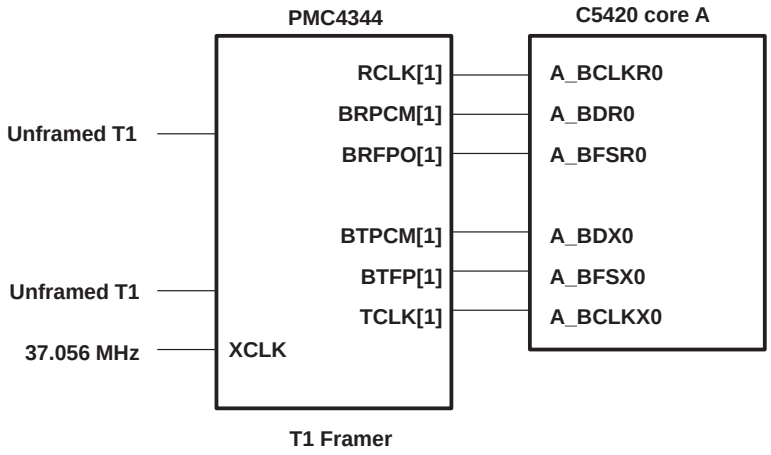


**Figure 4. PMC Sierra Framer to DSP**

## IMPORTANT NOTICE

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

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

Customers are responsible for their applications using TI components.

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

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