TMS320C620x/C642x McBSP: UART

Todd Hiers

ABSTRACT

This document describes how to use the multichannel buffered serial port (McBSP) in the Texas Instruments (TI) TMS320C6000™ (C6000™) digital signal processors (DSP) to interface to a universal asynchronous receiver/transmitter (UART). Descriptions of the hardware configuration and software routines necessary for proper functionality are included.

The McBSP is not capable of supporting UART standards natively. However, by simple modification of the serial control registers, there are two methods by which the McBSP can be configured to receive and transmit data that is understandable to a UART. The McBSP can be used in either the serial port mode or the general-purpose input/output (GPIO) mode. This application report discusses both methods. In addition, this application report demonstrates the hardware interface between the McBSP and a UART.

Project collateral and source code discussed in this application report can be downloaded from the following URL: http://www.ti.com/lit/zip/SPRA633.

Contents

List of Figures

List of Tables

1 Design Problem

How can the McBSP in a TMS320C6000 digital signal processor be used for transmitting data to and receiving data from a UART?

2 Overview

The UART standard is a well-established protocol for the exchange of serial data. Since it is asynchronous, the communications link requires no clock signal to be transmitted. Instead, the receiver and transmitter each have their own serial clocks that run at a preset frequency. The UART transmission protocol includes start and stop bits to help the receiver synchronize to the incoming data. The UART timing specification is shown in Figure 1. A high-to-low transition on the data line signifies the beginning of a transmission. After this Start condition, the data bits are sent serially with the least significant bit (LSB) first. The parity bit is optional, depending on the UART format. Each data frame ends with the Stop bit (logic high).

To interface a UART to the RS-232 port of the computer, the data signal needs to go through a RS-232 level converter to translate from the complementary metal-oxide-semiconductor (CMOS) logic levels to the RS-232 logic levels. The RS-232 logic levels use +3 to +25 V to signify a *Space* (logic 0) and -3 to -25 V for a *Mark* (logic 1). Any voltage in between these regions (i.e., between +3 and -3 V) is undefined.

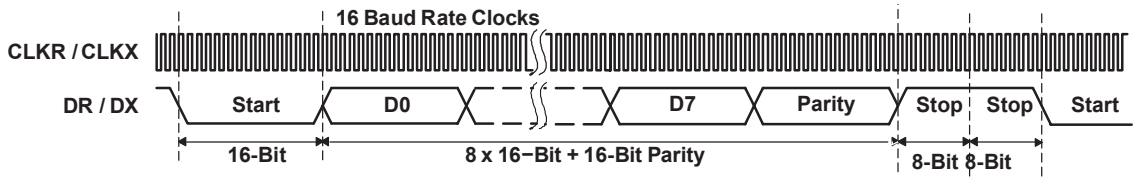


Figure 1. UART Timing

The McBSPs on the C6000 devices are synchronous serial ports, and are not capable of interfacing to a UART natively. UART functionality can be implemented in software, however. This application report discusses two methods to interface a UART to the McBSP. The first method uses the McBSP in normal serial port mode. The second method uses the McBSP in the GPIO mode.

3 UART Interface Method 1: McBSP in Serial Port Mode

To interface a UART to the McBSP in serial port mode, the UART's transmit data line is connected to both the data input and the frame synchronization input on the McBSP. This is because the UART serial data line contains both framing and data information. The UART's receive data line is connected to the data output of the McBSP. Figure 2 illustrates the UART to McBSP connection.

By using the McBSP's internal sample rate generator to clock itself, the McBSP can be configured to receive and transmit each UART bit as a 16-bit word. Software must expand each bit to be transmitted to a 16-bit word and compress each 16-bit word received to a single bit, as well as handle the necessary framing data.

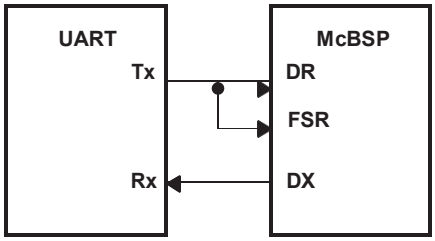


Figure 2. UART Connection - Serial Port Implementation

3.1 McBSP Setup: Serial Port Implementation

The C6000 treats each UART bit as a 16-bit word. The sample rate generator is configured to create an internal serial clock of 16 times the serial baud rate, thus duplicating the UART's internal timing. Since each UART word starts with a falling edge to indicate the start bit, this edge can be used as the active-low frame sync input. This is why both the data and frame sync inputs are connected to the UART's output. Notice that to prevent the McBSP from re-triggering, it is set to ignore all frame syncs during the receive packet.

To send a byte to a UART in 8N1 mode (eight data bits, no parity bit and one stop bit) the transfer should be in two phases, one consisting of nine 16-bit words and the other of two 8-bit words. Figure 3 shows the McBSP in 8N1 mode. The first half of the frame corresponds to the start bit and the eight data bits; the second half of the frame is the stop bit. Other UART modes can be accommodated by adjusting the frame word counts. When transmitting single UART bits as 16-bit words, '1' UART bits are encoded as 0×FFFF and '0' UART bits are encoded as 0×0000. The stop bit should be encoded in 8-bit words to allow for easy modification to the 1.5 stop bits setting in other UART modes, if desired.

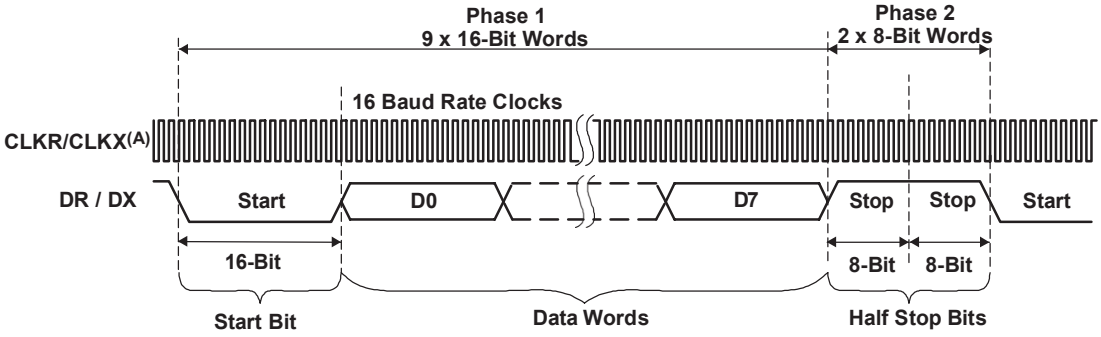


Figure 3. McBSP Transfer in UART 8N1 Mode

Several McBSP parameters have to be configured for the UART connection. Figure 4 through Figure 7 show the McBSP registers setup. Table 1 summarizes the McBSP setup.

3.1.1 Pin Control Register (PCR)

- FSXM = 1 and FSXP = 1. This allows the sample rate generator to generate active-low start bits.
- FSRM = 0 and FSRP = 1. The active-low start bit is the frame sync input to the McBSP.
- CLKRM = CLKXM = 1. Internal sample rate generator generates the serial clock.

The pin control register (PCR) is shown in Figure 4 and described in Table 1.

Figure 4. Pin Control Register (PCR)

31	30	29	28	27	26	25	16
Reserved							
0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8
Reserved		XIOEN	RIOEN	FSXM	FSRM	CLKXM	CLKRM
0	0	0	0	1	0	1	1

7	6	5	4	3	2	1	0
Reserved	CLKS_STAT	DX_STAT	DR_STAT	FSXP	FSRP	CLKXP	CLKRP
0	0	0	0	1	1	0	0

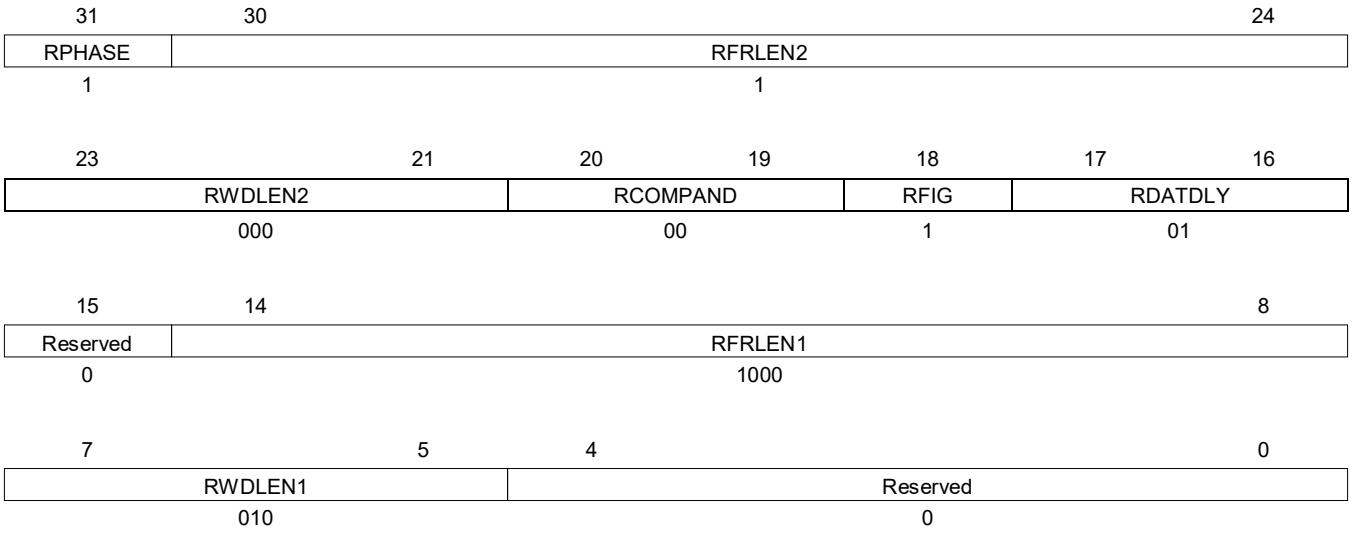
LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

3.1.2 Receive/Transmit Control Registers (RCR/XCR)

- (R/X)PHASE = 1. Enable dual-phase frame mode.
- (R/X)FRLEN1 = 8. Nine elements in the first phase of the frame.
- (R/X)FRLEN2 = 1. Two elements in the second phase of the frame.
- (R/X)WDLEN1 = 2. 16-bit words in the first phase (Start bit, data bits).
- (R/X)WDLEN2=0. 8-bit words in the second phase (Stop bits).
- (R/X)COMPAND=0. No companding.
- (R/X)FIG = 1. For reception, since data line transitions are seen on the FSR pin, unexpected frame sync signals must be ignored. For transmission, since the transmit frame sync signal FSX is generated on every DXR-to-XSR copy (see SRGR setup below), it occurs more frequently than desired for a UART frame. Unexpected frame syncs are ignored.
- XDATDLY=0. No data delay.
- RDATDLY=1. 1-bit data delay.

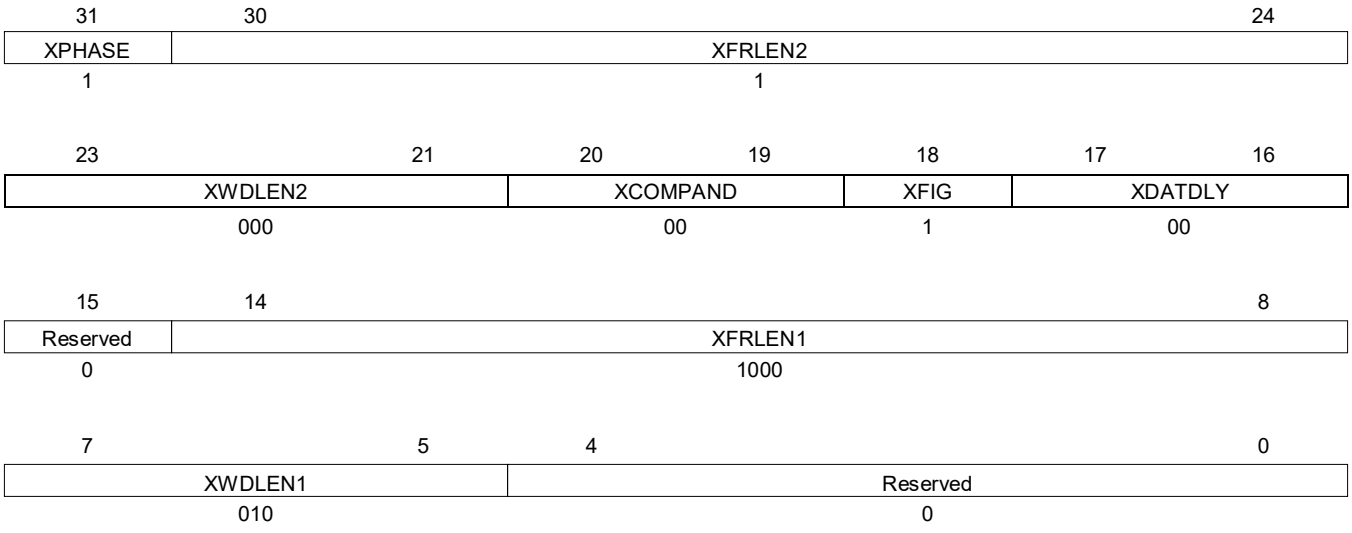
The receive/transmit control registers (RCR/XCR) are shown in Figure 5 and Figure 6 and described in Table 1.

Figure 5. Receive Control Register (RCR)



LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 6. Transmit Control Register (XCR)



LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

3.1.3 Sample Rate Generator Register (SRGR)

- FSGM = 0. The transmit frame sync signal (FSX) is generated on every DXR-to-XSR copy.
- CLKSM = 0 if the sample rate generator clock is derived from an external clock on the CLKS pin. CLKSM = 1 if the sample rate generator clock is derived from the internal CPU clock.
- CLKGDV= (CPU Clock frequency) / (16 * baud rate) -1. The clock divide ratio must be appropriately set so that the rate generated is 16 times the baud rate. For example, a CPU clock frequency of 200 MHz and a desired baud rate of 115,200 bps would result in an approximate CLKGDV value of 108. Note that when the sample rate generator clock is derived from the internal clock source, you may not be able to get a serial clock that is exactly 16 times the desired baud rate. In addition, the limited size of the CLKGDV field creates a minimum baud rate that the serial port is capable of clocking. If a baud rate slower than the minimum or an exact baud rate is desired, you should use an external clock on the CLKS pin to drive the sample rate generator.

The sample rate generator register (SRGR) is shown in Figure 7 and described in Table 1.

Figure 7. Sample Rate Generator Register (SRGR)

31	30	29	28	27	16
GSYNC	CLKSP	CLKSM	FSGM	FPER	
0	0	1	0	0	

15	8	7	0
FWID		CLKGDV	
0		108	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 1. Field Descriptions for McBSP Registers

Register [Bit-Field No.]	Field	Value (in Binary)	Description
RCR[31]	RPHASE	1	Dual Phase Receive
RCR[30-24]	RFRLEN2	1	2 Word Receive Frame Length (Phase 2)
RCR[23-21]	RWDLEN2	000	8 Bits Receive Word Length (Phase 2)
RCR[20-19]	RCOMPAND	00	No Companding
RCR[18]	RFIG	1	Unexpected FSR Ignored
RCR[17-16]	RDATDLY	01	1-Bit Data Delay
RCR[14-8]	RFRLEN1	1000	9 Word Receive Frame Length (Phase 1)
RCR[7-5]	RWDLEN1	010	16 Bits Receive Word Length (Phase 1)
PCR[10]	FSRM	1	FSR is Input Pin
PCR[8]	CLKRM	1	CLKR is Output Pin
PCR[2]	FSRP	1	Active-Low Frame Sync
SRGR[29]	CLKSM	1	SRGR Derived From CPU Clock
	FSGM	0	Frame Sync is Generated on Every DXR-to-XSR
SRGR[7-0]	CLKGDV	1101011 (107)	CLKX = CPU Clock Divided by 108 (107+1)
XCR[31]	XPHASE	1	Dual Phase Transmit
XCR[30-24]	XFRLEN2	1	2 Word Transmit Frame Length (Phase 2)
XCR[23-21]	XWDLEN2	000	8 Bits Transmit Word Length (Phase 2)
XCR[20-19]	XCOMPAND	00	No Companding
XCR[18]	XFIG	1	Unexpected FSX Ignored
XCR[17-16]	XDATDLY	0	No Data Delay
XCR[14-8]	XFRLEN1	1000	9 Word Transmit Frame Length (Phase 1)
XCR[7-5]	XWDLEN1	010	16 Bits Transmit Word Length (Phase 1)
SRGR[28]	FSGM	0	FSX Generated on DXR-to-XSR Copy
PCR[11]	FSXM	1	FSX is Output pin
PCR[9]	CLKXM	1	CLKX is Output pin
PCR[3]	FSXP	1	Active-Low Frame Sync

3.2 Receiving/Transmitting UART Data

Once the serial port has been configured to interface to a UART, the software routines that do the necessary data conversions must be implemented. When using the McBSP in serial port mode, there are two possible software implementations. You can either process the UART data word-by-word or in blocks. This application report discusses the more efficient implementation of the two-UART data processing in blocks. With simple modification, the software can handle word-by-word data processing.

The sample C program in Appendix A shows block UART data processing. In this implementation, the enhanced direct memory access (EDMA) or direct memory access (DMA) services the McBSP by transferring data between the McBSP and the receive/transmit buffers.

For transmits, a transmit conversion subroutine converts a block of data into UART transmission words by expanding each data bit into a 16-bit word. The transmit conversion subroutine places this block of transmission words in a transmit buffer, along with the framing Start (0×0000) and Stop (0×FFFF) bits in the proper locations. Figure 8 shows a sample transmit buffer. Afterward, the EDMA is setup to transfer the data from the transmit buffer to the McBSP. Since the data in the transmit buffer is already in the proper UART format, the McBSP frame sync generator can be used to continuously shift out this data.

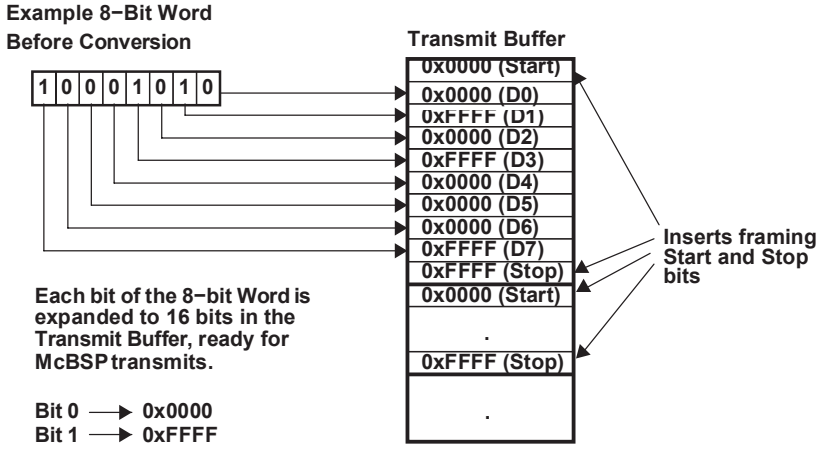


Figure 8. Block Data Processing of Transmit Buffer

4 UART Interface Method 2: McBSP in GPIO Mode

The C6000 DSP can also interface to a UART using its GPIO pins. The McBSP pins CLKX, FSX, DX, CLKR, FSR, DR, and CLKS can be used as GPIO pins when the following two conditions are true:

- The related portion (transmitter or receiver) of the serial port is in reset: ($\overline{R/X}$)RST = 0 in the Serial Port Control Register (SPCR).
- The GPIO is enabled for the related portion of the serial port: (R/X)IOEN=1 in the PCR.

Table 2 shows how to setup the McBSP pins as GPIO pins.

Table 2. Configuration of McBSP Pins as GPIO

Pin	GPIO Enabled When 0	Selected as Output When …	Output Value Driven From	Selected as Input When …	Input Value Readable on
CLKX	XRST = 0 XIOEN = 1	CLKXM = 1	CLKXP	CLKXM = 0	CLKXP
FSX	$\overline{\text{XRST}}$ = 0 XIOEN = 1	FSXM = 1	FSXP	FSXM = 0	FSXP
DX	$\overline{\text{XRST}}$ = 0 XIOEN = 1	Always	DX_STAT	Never	N/A
CLKR	$\overline{\text{RRST}}$ = 0 RIOEN = 1	CLKRM = 1	CLKRP	CLKRM = 0	CLKRP
FSR	$\overline{\text{RRST}}$ = 0 RIOEN = 1	FSRM = 1	FSRP	FSRM = 0	FSRP
DR	$\overline{\text{RRST}}$ = 0 RIOEN = 1	Never	N/A	Always	DR_STAT
CLKS	$\overline{\text{RRST}}$ = $\overline{\text{XRST}}$ = 0 RIOEN = XIOEN = 1	Never	N/A	Always	CLKS_STAT

4.1 McBSP Setup: GPIO Implementation

Although different GPIO pins on the C6000 DSP can be used as GPIO pins, this application report discusses an example UART implementation when the McBSP DX and DR pin are used as general-purpose output and input pins, respectively. Figure 9 illustrates the McBSP to UART connection. All other McBSP pin connections are don't cares in this example.

Figure 10 and Figure 11 show the SPCR and PCR setup specific to this example. The setups of all other McBSP registers are don't cares because the McBSP is in GPIO mode. The values in shaded bit fields are don't cares and are left at default.

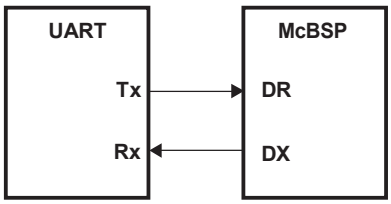


Figure 9. UART Connection - GPIO Implementation

Figure 10. Serial Port Control Register (SPCR)

31	30	29	28	27	26	25	24
Reserved							
000000000							

23	22	21	20	19	18	17	16
FRST	GRST	XINTM		XSYNCERR	XEMPTY	XRDY	XRST
0	0	00		0	0	0	0

15	14	13	12	11	10	9	8
DLB	RJUST		CLKSTP		Reserved		
0	0		00		000		

7	6	5	4	3	2	1	0
DXENA	Reserved	RINTM		RSYNCERR	RFULL	RRDY	RRST
0	0	0		0	0	0	0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 11. Pin Control Register (PCR)

31							16
Reserved							
0×0000							
15	14	13	12	11	10	9	8
Reserved		XIOEN	RIOEN	FSXM	FSRM	CLKXM	CLKRM
0		0	0	0	0	0	0
7	6	5	4	3	2	1	0
Reserved	CLKSSTAT	DXSTAT	DRSTAT	FSXP	FSRP	CLKXP	CLKRP
0	0	0	0	0	0	0	0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

4.2 GPIO UART Software

Appendix B contains three low level routines that can be called by higher level programs to perform UART data transmit and receive. The three functions are:

```
unsigned int SoftUartSpeedDetect(void);
void SoftUartOutchar(int, char);
char SoftUartInchar(int);
```

Function SoftUartSpeedDetect( ) sets the McBSP in GPIO mode and detects the UART transmission rate. Function SoftUartOutchar(int, char) transmits UART data from the McBSP to the UART. Function SoftUartInchar(int) receives UART data that comes from the UART to the McBSP. The following sections discuss these functions in detail.

4.2.1 SoftUartSpeedDetect - Subroutine for Auto-Baud Detection

The subroutine SoftUartSpeedDetect sets the McBSP in GPIO mode with the SPCR and PCR registers. It performs auto-baud detection by measuring the length of the Start bit, plus the length of the first data bit (logic high) in the character <cr> (carriage return). You need to ensure (in software) that the first character sent is <cr> because the first data bit has to be a logical one. This is shown as `T' in Figure 12.

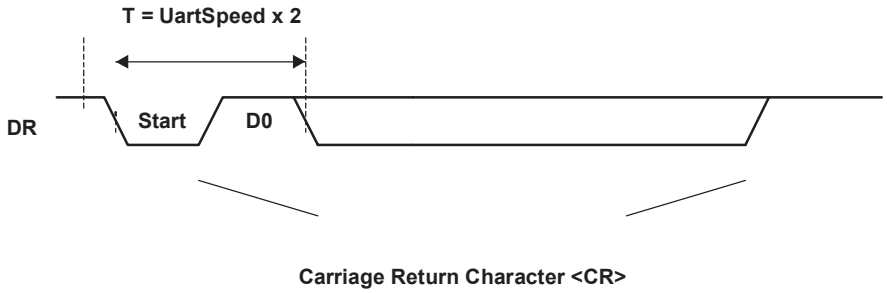


Figure 12. UART Auto-Baud Detection

The time T is determined by a software counter incremented by one until the second transition from high to low is detected by reading the DRSTAT bit in the PCR register. T represents twice the time of a bit length.

This measurement is required because the RX signal from UART is not always very clean. Simply measuring the length of the Start bit to determine the baud rate is not accurate enough.

The time reference value UartSpeed = T >>1 (i.e., T/2) is returned from SoftUartSpeedDetect() and used in the character input detection and character output send routines, SoftUartInchar and SoftUartOutchar.

This software UART is a basic emulation and can be customized in several ways by using timers and interrupts.

Current implementations of software UART are used for debugging purpose or application monitoring in places where a VT100 or an ANSI terminal is mandatory.

4.2.2 SoftUartInchar - Subroutine for UART Data Receive

Subroutine SoftUartInchar takes the input argument UartSpeed - the UART speed output returned from function SoftUartSpeedDetect.

This subroutine parses bit-by-bit the UART data on the DR line. It detects the Start bit by polling for the first DR line transition from inactive (logic 1) to active (logic 0) state. The eight data bits are transmitted by the UART device immediately after the Start bit. The best time to fetch the right value of each data bit is in the middle of the data bit waveform. Figure 13 shows how subroutine SoftUartInchar achieves this. It waits for half of the UartSpeed time value (P/2 in) during the Start bit. Then for each of the eight valid data bits, it samples the DR line status (DR_STAT bit in the PCR) in the middle of the data bit waveform. The subroutine shifts each binary bit result into a single register value to retrieve the original 8-bit character. All the delays are generated with polling loops to minimize the use of resources in the DSP.

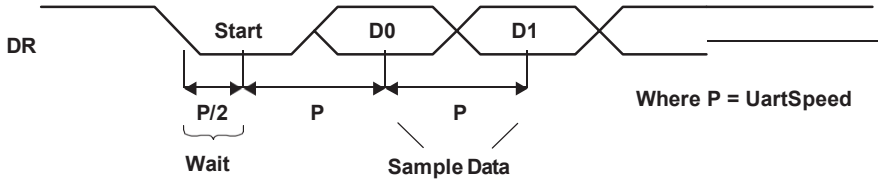


Figure 13. SoftUartInchar UART Data Fetch

4.2.3 SoftUartOutchar - Subroutine for UART Data Transmit

Subroutine SoftUartOutchar is based on the same mechanism as the SoftUartInchar subroutine. It takes the input argument UartSpeed - the UART speed output returned from function SoftUartSpeedDetect.

This subroutine drives the transmit data on the DX line through writing the DX_STAT bit in the PCR. At the beginning of a transfer, SoftUartOutchar writes a `0' to the DX line (Start bit). Subsequently, it transmits each data bit on the DX line. The time spent on each bit is derived from the UartSpeed input, processed by a polling loop.

The transmit character is first placed into the least significant eight bits in a register padded with three Stop bits (0x00000700). For example, the character `A' (ASCII character 0×41) is placed in the padded register to become 0×00000741. Each time through the loop, the LSB of the padded register is driven on the DX line, and is right-shifted to be ready for the next bit transmit. In the character `A' example, after the first bit `1' is driven on the DX line, the padded register is right-shifted by one to 0×000003A0.

5 Hardware UART Adapter for the C6000 Processors

The hardware adapter for the software UART support consists of a single SN75LV4737A multichannel RS232 line driver/receiver which transmit/receives the binary stream from/to the McBSP lines.

The SN75LV4737A consists of three line drivers, five line receivers, and a charge-pump circuit. It provides the electrical interface between an asynchronous communication controller and the serial port connector and meets the requirements of EIA/TIA-232-E. This combination of drivers and receivers matches those needed for the typical serial port used in an IBM PC/AT or compatibles.

The device has flexible control options for power management when the serial port is inactive. A common disable for all of the drivers and receivers is provided with the active-high STBY input. The active-low EN input is an enable for one receiver to implement a wake-up feature for the serial port. All the logic inputs can accept signals from controllers operating from a 5 V supply even though the SN75LV4737A is operating from 3.3 V.

Figure 14 presents an adapter board for the C6000 DSP. The connection in this figure demonstrates the GPIO mode implementation. For serial port mode implementation described in this application report, pin DR0 needs to be tied to pin FSR0.

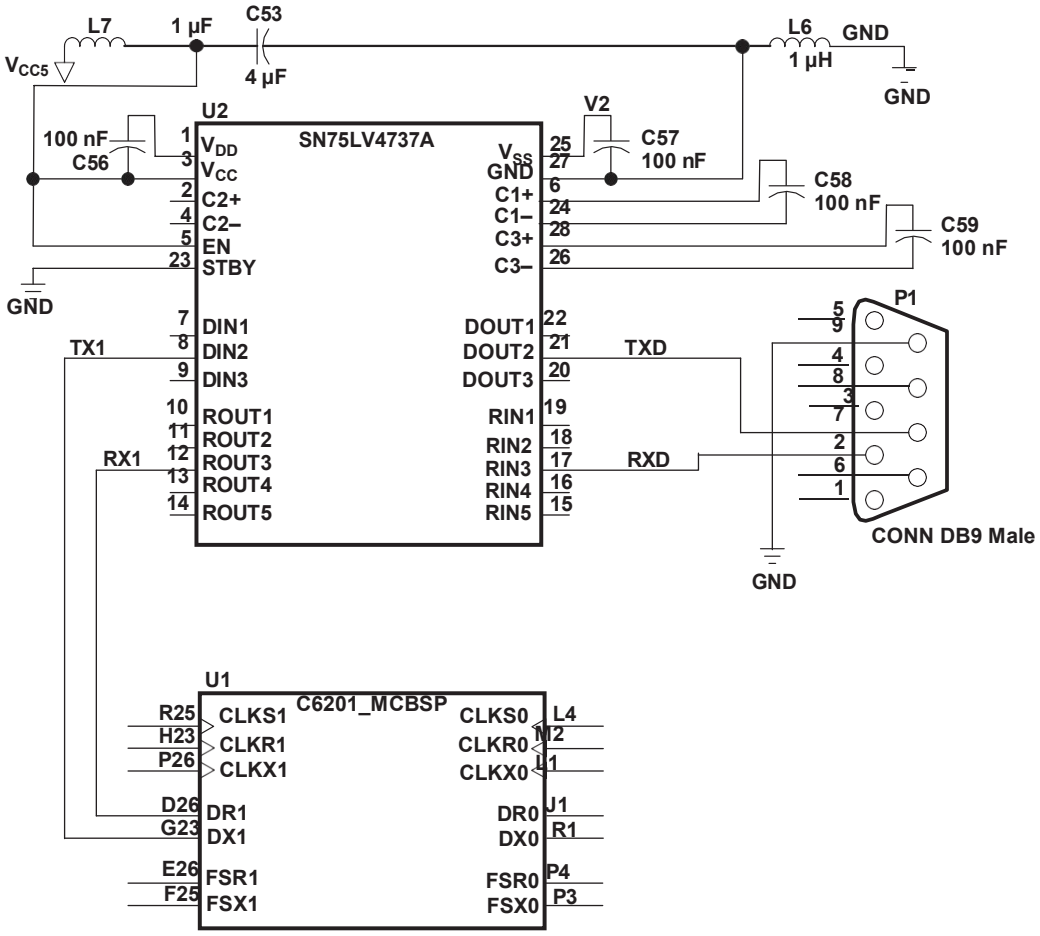


Figure 14. UART Adapter Board

6 Conclusion

The McBSP on the TMS320C6000 DSP is not natively capable of interfacing to a UART; however, with software control, communication between a McBSP and a UART is possible. The McBSP is easy to configure, and the compression/expansion software routines are straightforward for this purpose.

7 References

- *TMS320C6201 Digital Signal Processor* (SPRS051)
- *TMS320C6000 DSP Peripherals Overview Reference Guide* (SPRU190)
- *TMS320C6x Peripheral Support Library Programmer's Reference* (SPRU273)
- *TMS320C6000 CPU and Instruction Set Reference Guide* (SPRU189)
- Lammert Bies, RS232 general info, http://www.lammertbies.nl/comm/info/RS-232.html
- *3.3-V/5-V Multichannel RS-232 Line Driver/Receiver* (SLLS178)

Appendix A Sample C Code

A.1 Serial Port Mode

```
/*****************************************************************/
/* TEXAS INSTRUMENTS, INC.                                       */
/* Date Created: 06/22/2001                                      */
/* Date Last Modified: 07/9/2001                                 */
/* Source File: uart.c                                           */
/* Original Author: Todd Hiers                                   */
/* Author: Scott Chen                                            */
/*                                                               */
/* This code describes how to initialize the C6000 McBSP to      */
/* communicate with a UART.  By modifying the CHIP definition,   */
/* this code can be used to run on 6x1x/6x0x/64x.  #if statements */
/* are included in this code which allow flexibility in different */
/* devices.                                                      */
/*                                                               */
/* On 6x0x devices, DMA channels 1 and 2 are used to service     */
/* McBSP 1 transmit and receive operations, respectively. On     */
/* 6x1x/64x devices, EDMA channels 14 and 15 are used to service */
/* McBSP 1 transmit and receive operations, respectively.        */
/*                                                               */
/* For this example, a data string is being transmitted from McBSP */
/* transmit (DX) to McBSP receive (DR).  Each bit of the 8-bit   */
/* ASCII character is expanded into 16-bit UART transmission word. */
/* Once being received, the 16-bit UART transmission words are   */
/* compressed back to binary bits and ASCII form.                */
/*                                                               */
/* For the code to work, DX, DR, and FSR of McBSP1 are shorted   */
/* together.                                                     */
/*                                                               */
/* This code has been verified for functionality on 6711, 6202,  */
/* and 6203 devices.                                             */
/*                                                               */
/* This program is based on CSL 2.0.  Please refer to the        */
/* TMS320C6000 Chip Support Library API User's Guide for further */
/* information.                                                  */
/*****************************************************************/

/* Chip definition - Please change this accordingly */
#define CHIP_6711 1

/* Include files */
#include <csl.h>
#include <csl_mcbsp.h>
#include <csl_edma.h>
#include <csl_dma.h>
#include <csl_irq.h>
#include <stdio.h>

/* Create buffers and aligning them on an L2 cache line boundary. */
#pragma DATA_SECTION(xmitbuf,"xmit_buf");
unsigned short xmitbuf[0x0400];
#pragma DATA_SECTION(recvbuf,"recv_buf");
unsigned short recvbuf[0x0400];

/* Definitions */
#define BUFFER_SIZE   27                /* total number of UART data words*/
#define TRUE 1
#define FALSE 0

/* Declare CSL objects */
MCBSP_Handle hMcbsp1;                    /* handle for McBSP1 */
#if (EDMA_SUPPORT)
      EDMA_Handle hEdma14;               /* handle for EDMA 14 */
      EDMA_Handle hEdma15;               /* handle for EDMA 15 */
#endif
#if (DMA_SUPPORT)
      DMA_Handle hDma1;                  /* handle for DMA 1   */
      DMA_Handle hDma2;                  /* handle for DMA 2   */
```

```
#endif

/* Global Variables  */
volatile int receive_done = FALSE;
volatile int transmit_done = TRUE;
char xmit_msg[BUFFER_SIZE] = "McBSP does UART on C6000!\n";
char recv_msg[BUFFER_SIZE] = "Transmission didn't work!\n";

/* Include the vector table to call the IRQ ISRs hookup */
extern far void vectors();

/* Prototypes     */
void ConfigMcBSP(void);
void ConfigEDMA(void);
void ConfigDMA(void);
void ProcessTransmitData(void);
void ProcessReceiveData(void);
short VoteLogic(unsigned short);
int CheckTestCase(void);
interrupt void c_int11(void);
interrupt void c_int09(void);
interrupt void c_int08(void);


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

      int waittime = 0;
      int works = FALSE;

      /* initialize the CSL library */
      CSL_init();
      /* enable NMI and GI */
      IRQ_nmiEnable();
      IRQ_globalEnable();

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

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

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

      #endif

      #if (DMA_SUPPORT)

           DMA_reset(INV);

      #endif

      /* process transmit data */
      printf("Processing Transmit string...\n");
      ProcessTransmitData();
      printf("String transmitted: %s \n", xmit_msg);
      #if (EDMA_SUPPORT)

           /* Open the EDMA channels - EDMA 14 for transmit, */
           /* EDMA 15 for receive */
           hEdma14 = EDMA_open(EDMA_CHA_XEVT1, EDMA_OPEN_RESET);
           hEdma15 = EDMA_open(EDMA_CHA_REVT1, EDMA_OPEN_RESET);

      #endif

      #if (DMA_SUPPORT)

           /* Open the DMA channels - DMA 1 for transmit,   */
           /* DMA 2 for receive */
```

```
        hDma1 = DMA_open(DMA_CHA1, DMA_OPEN_RESET);
        hDma2 = DMA_open(DMA_CHA2, DMA_OPEN_RESET);

#endif

/* Open the McBSP channel 1 */
hMcbsp1 = MCBSP_open(MCBSP_DEV1, MCBSP_OPEN_RESET);

#if (EDMA_SUPPORT)

        /* Configure the EDMA channels */
        ConfigEDMA();

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

        /* enable EDMA channel interrupt to CPU */
        EDMA_intEnable(14);
        EDMA_intEnable(15);

        /* Enable EDMA channels */
        EDMA_enableChannel(hEdma14);
        EDMA_enableChannel(hEdma15);

#endif

#if (DMA_SUPPORT)

        /* Configure the DMA channels */
        ConfigDMA();

        IRQ_disable(IRQ_EVT_DMAINT1);
        IRQ_disable(IRQ_EVT_DMAINT2);

        IRQ_clear(IRQ_EVT_DMAINT1);
        IRQ_clear(IRQ_EVT_DMAINT2);

        IRQ_enable(IRQ_EVT_DMAINT1);
        IRQ_enable(IRQ_EVT_DMAINT2);

        DMA_start(hDma1);           /*start DMA channel 1*/
        DMA_start(hDma2);           /*start DMA channel 2*/

#endif

/* Setup for McBSP */
ConfigMcBSP();

/* Start Sample Rate Generator: set /GRST = 1 */
MCBSP_enableSrgr(hMcbsp1);

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

/* Wake up the McBSP as transmitter and receiver */
MCBSP_enableRcv(hMcbsp1);
MCBSP_enableXmt(hMcbsp1);

/* Enable Frame Sync Generator for McBSP 1: set /FRST = 1 */
MCBSP_enableFsync(hMcbsp1);

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

/* Check to make sure the test case works */
works = CheckTestCase();
if (works != 0) printf("Transmission Error. .. \n\n");
else printf("Received data matched transmitted data!\n\n");

/* process received data */
printf("Processing Receive string. . \n");
ProcessReceiveData();
printf("String received: %s \n", recv_msg);
```

```
        #if (EDMA_SUPPORT)

             IRQ_disable(IRQ_EVT_EDMAINT);
             EDMA_RSET(CIER, 0x0);

        #endif

        #if (DMA_SUPPORT)

             IRQ_disable(IRQ_EVT_DMAINT1);
             IRQ_disable(IRQ_EVT_DMAINT2);

        #endif

        MCBSP_close(hMcbsp1);          /* close McBSP 1 */

        #if (EDMA_SUPPORT)

             EDMA_close(hEdma14);     /* close EDMA 14 */
             EDMA_close(hEdma15);     /* close EDMA 15 */

        #endif

        #if (DMA_SUPPORT)

             DMA_close(hDma1);        /* close DMA 1 */
             DMA_close(hDma2);        /* close DMA 2 */

        #endif
    }   /* End of main() */

    /*********************************************************************/
    /* void ConfigEDMA(void): set up EDMA channel 14/15 for UART Xmit  */
    /*********************************************************************/
    #if (EDMA_SUPPORT)

    void ConfigEDMA(void)
    {
        EDMA_configArgs(hEdma14,

             /* OPT Setup */
             #if (C64_SUPPORT)
                  EDMA_OPT_RMK(
                       EDMA_OPT_PRI_HIGH,        /* 1  */
                       EDMA_OPT_ESIZE_16BIT,     /* 01 */
                       EDMA_OPT_2DS_NO,          /* 0  */
                       EDMA_OPT_SUM_INC,         /* 01 */
                       EDMA_OPT_2DD_NO,          /* 0  */
                       EDMA_OPT_DUM_NONE,        /* 00 */
                       EDMA_OPT_TCINT_YES,       /* 1  */
                       EDMA_OPT_TCC_OF(14),      /* 14 */
                       EDMA_OPT_TCCM_DEFAULT,    /* 0  */
                       EDMA_OPT_ATCINT_DEFAULT   /* 0  */
                       EDMA_OPT_ATCC_DEFAULT,    /* 0  */
                       EDMA_OPT_PDTS_DEFAULT,    /* 0  */
                       EDMA_OPT_PDTD_DEFAULT,    /* 0  */
                       EDMA_OPT_LINK_NO,         /* 0  */
                       EDMA_OPT_FS_NO            /* 0  */
                  ),
             #else
                  EDMA_OPT_RMK(
                       EDMA_OPT_PRI_HIGH,        /* 1 */
                       EDMA_OPT_ESIZE_16BIT,     /* 01 */
                       EDMA_OPT_2DS_NO,          /* 0  */
                       EDMA_OPT_SUM_INC,         /* 01 */
                       EDMA_OPT_2DD_NO,          /* 0  */
                       EDMA_OPT_DUM_NONE,        /* 00 */
                       EDMA_OPT_TCINT_YES,       /* 1 */
                       EDMA_OPT_TCC_OF(14),      /* 14 */
                       EDMA_OPT_LINK_NO,         /* 0  */
                       EDMA_OPT_FS_NO            /* 0  */
                  ),
             #endif
```

```
            /* SRC Setup */
            EDMA_SRC_RMK((Uint32) xmitbuf),        /*xmitbuf address*/

            /* CNT Setup */
            EDMA_CNT_RMK(
                  EDMA_CNT_FRMCNT_DEFAULT,
                  EDMA_CNT_ELECNT_OF(BUFFER_SIZE*11)
            ),

            /* DST Setup */
            EDMA_DST_RMK(MCBSP_getXmtAddr(hMcbsp1)),

            /* IDX Setup */
            EDMA_IDX_RMK(0,0),

            /* RLD Setup */
            EDMA_RLD_RMK(0,0)
            );

      EDMA_configArgs(hEdma15,

            /* OPT Setup */
            #if (C64_SUPPORT)
            EDMA_OPT_RMK(
                        EDMA_OPT_PRI_HIGH,           /* 1  */
                        EDMA_OPT_ESIZE_16BIT,        /* 01 */
                        EDMA_OPT_2DS_NO,             /* 0  */
                        EDMA_OPT_SUM_NONE,           /* 00 */
                        EDMA_OPT_2DD_NO,             /* 0  */
                        EDMA_OPT_DUM_INC,            /* 01 */
                        EDMA_OPT_TCINT_YES,          /* 1  */
                        EDMA_OPT_TCC_OF(15),         /* 15 */
                        EDMA_OPT_TCCM_DEFAULT,       /* 0  */
                        EDMA_OPT_ATCINT_DEFAULT      /* 0  */
                        EDMA_OPT_ATCC_DEFAULT,       /* 0  */
                        EDMA_OPT_PDTS_DEFAULT,       /* 0  */
                        EDMA_OPT_PDTD_DEFAULT,       /* 0  */
                        EDMA_OPT_LINK_NO,            /* 0  */
                        EDMA_OPT_FS_NO               /* 0  */
                  ),
            #else
                  EDMA_OPT_RMK(
                        EDMA_OPT_PRI_HIGH,           /* 1  */
                        EDMA_OPT_ESIZE_16BIT,        /* 01 */
                        EDMA_OPT_2DS_NO,             /* 0  */
                        EDMA_OPT_SUM_NONE,           /* 00 */
                        EDMA_OPT_2DD_NO,             /* 0  */
                        EDMA_OPT_DUM_INC,            /* 01 */
                        EDMA_OPT_TCINT_YES,          /* 1  */
                        EDMA_OPT_TCC_OF(15),         /* 15 */
                        EDMA_OPT_LINK_NO,            /* 0  */
                        EDMA_OPT_FS_NO               /* 0  */
                  ),
            #endif

            /* SRC Setup */
            EDMA_SRC_RMK(MCBSP_getRcvAddr(hMcbsp1)),

            /* CNT Setup */
            EDMA_CNT_RMK(0, (BUFFER_SIZE * 11)),

            /* DST Setup */
            EDMA_DST_RMK((Uint32) recvbuf),        /*recvbuf address*/

            /* IDX Setup */
            EDMA_IDX_RMK(0,0),

            /* RLD Setup */
            EDMA_RLD_RMK(0,0)
      );

} /* End of ConfigEDMA() */
#endif
```

```
/*********************************************************************/
/* void ConfigDMA(void): set up DMA channels 1 & 2 for UART Xmit   */
/*********************************************************************/
#if (DMA_SUPPORT)

    void ConfigDMA(void)
{

DMA_configArgs(hDma1,

     /* PRICTL Setup */
     DMA_PRICTL_RMK(
     DMA_PRICTL_DSTRLD_NONE,
     DMA_PRICTL_SRCRLD_NONE,
     DMA_PRICTL_EMOD_HALT,
     DMA_PRICTL_FS_DISABLE,
     DMA_PRICTL_TCINT_ENABLE,
     DMA_PRICTL_PRI_DMA,
     DMA_PRICTL_WSYNC_XEVT1,
     DMA_PRICTL_RSYNC_NONE,
     DMA_PRICTL_INDEX_NA,
     DMA_PRICTL_CNTRLD_NA,
     DMA_PRICTL_SPLIT_DISABLE,
     DMA_PRICTL_ESIZE_16BIT,
     DMA_PRICTL_DSTDIR_NONE,
     DMA_PRICTL_SRCDIR_INC,
     DMA_PRICTL_START_STOP
),

/* SECCTL Setup */
     DMA_SECCTL_RMK(
     DMA_SECCTL_WSPOL_ACTIVEHIGH,
     DMA_SECCTL_RSPOL_ACTIVEHIGH,
     DMA_SECCTL_FSIG_NORMAL,
     DMA_SECCTL_DMACEN_FRAMECOND,
     DMA_SECCTL_WSYNCCLR_NOTHING,
     DMA_SECCTL_WSYNCSTAT_CLEAR,
     DMA_SECCTL_RSYNCCLR_NOTHING,
     DMA_SECCTL_RSYNCSTAT_CLEAR,
     DMA_SECCTL_WDROPIE_DISABLE,
     DMA_SECCTL_WDROPCOND_CLEAR,
     DMA_SECCTL_RDROPIE_DISABLE,
     DMA_SECCTL_RDROPCOND_CLEAR,
     DMA_SECCTL_BLOCKIE_ENABLE,
     DMA_SECCTL_BLOCKCOND_CLEAR,
     DMA_SECCTL_LASTIE_DISABLE,
     DMA_SECCTL_LASTCOND_CLEAR,
     DMA_SECCTL_FRAMEIE_DISABLE,
     DMA_SECCTL_FRAMECOND_CLEAR,
     DMA_SECCTL_SXIE_DISABLE,
     DMA_SECCTL_SXCOND_CLEAR
),
     /* SRC Setup */
     DMA_SRC_RMK((Uint32) xmitbuf),                    /*xmitbuf*/
     /* DST Setup */
     DMA_DST_RMK(MCBSP_getXmtAddr(hMcbsp1)),           /*McBSP DXR */

     /* XFRCNT Setup */
     DMA_XFRCNT_RMK(
            DMA_XFRCNT_FRMCNT_OF(1),
            DMA_XFRCNT_ELECNT_OF(BUFFER_SIZE*11)
     )
);

DMA_configArgs(hDma2,

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

```
                    DMA_PRICTL_WSYNC_NONE,
                    DMA_PRICTL_RSYNC_REVT1,
                    DMA_PRICTL_INDEX_NA,
                    DMA_PRICTL_CNTRLD_NA,
                    DMA_PRICTL_SPLIT_DISABLE,
                    DMA_PRICTL_ESIZE_16BIT,
                    DMA_PRICTL_DSTDIR_INC,
                    DMA_PRICTL_SRCDIR_NONE,
                    DMA_PRICTL_START_STOP
              ),
              /* SECCTL Setup */
              DMA_SECCTL_RMK(
                    DMA_SECCTL_WSPOL_ACTIVEHIGH,
                    DMA_SECCTL_RSPOL_ACTIVEHIGH,
                    DMA_SECCTL_FSIG_NORMAL,
                    DMA_SECCTL_DMACEN_FRAMECOND,
                    DMA_SECCTL_WSYNCCLR_NOTHING,
                    DMA_SECCTL_WSYNCSTAT_CLEAR,
                    DMA_SECCTL_RSYNCCLR_NOTHING,
                    DMA_SECCTL_RSYNCSTAT_CLEAR,
                    DMA_SECCTL_WDROPIE_DISABLE,
                    DMA_SECCTL_WDROPCOND_CLEAR,
                    DMA_SECCTL_RDROPIE_DISABLE,
                    DMA_SECCTL_RDROPCOND_CLEAR,
                    DMA_SECCTL_BLOCKIE_ENABLE,
                    DMA_SECCTL_BLOCKCOND_CLEAR,
                    DMA_SECCTL_LASTIE_DISABLE,
                    DMA_SECCTL_LASTCOND_CLEAR,
                    DMA_SECCTL_FRAMEIE_DISABLE,
                    DMA_SECCTL_FRAMECOND_CLEAR,
                    DMA_SECCTL_SXIE_DISABLE,
                    DMA_SECCTL_SXCOND_CLEAR
              ),

              /* SRC Setup */
              DMA_SRC_RMK(MCBSP_getRcvAddr(hMcbsp1)),          /*McBSP DRR */

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

              /* XFRCNT Setup */
              DMA_XFRCNT_RMK(
                    DMA_XFRCNT_FRMCNT_OF(1),
                    DMA_XFRCNT_ELECNT_OF(BUFFER_SIZE*11)
              )

        );

} /* End of ConfigDMA() */

#endif

/*********************************************************************/
/* void ConfigMcBSP(void): Setup for McBSP Configuration             */
/*********************************************************************/
void ConfigMcBSP(void)
{

        MCBSP_Config mcbspCfg1 = {

              /* SPCR Setup */
              #if (DMA_SUPPORT)
                    MCBSP_SPCR_RMK(
                          MCBSP_SPCR_FRST_DEFAULT,         /* 0  */
                          MCBSP_SPCR_GRST_DEFAULT,         /* 0  */
                          MCBSP_SPCR_XINTM_XRDY,           /* 00 */
                          MCBSP_SPCR_XSYNCERR_DEFAULT,     /* 0  */
                          MCBSP_SPCR_XRST_DEFAULT,         /* 0  */
                          MCBSP_SPCR_DLB_OFF,              /* 0  */
                          MCBSP_SPCR_RJUST_RZF,            /* 00 */
                          MCBSP_SPCR_CLKSTP_DISABLE,       /* 0x */
                          MCBSP_SPCR_RINTM_RRDY,           /* 00 */
                          MCBSP_SPCR_RSYNCERR_DEFAULT,     /* 0  */
                          MCBSP_SPCR_RRST_DEFAULT          /* 0  */
```

```
    ),
#endif
#if (EDMA_SUPPORT)
      MCBSP_SPCR_RMK(
            MCBSP_SPCR_FREE_YES,              /* 1  */
            MCBSP_SPCR_SOFT_DEFAULT,          /* 0  */
            MCBSP_SPCR_FRST_DEFAULT,          /* 0  */
            MCBSP_SPCR_GRST_DEFAULT,          /* 0  */
            MCBSP_SPCR_XINTM_XRDY,    /*    00    */
            MCBSP_SPCR_XSYNCERR_DEFAULT,  /*  0  */
            MCBSP_SPCR_XRST_DEFAULT,    /*    0    */
            MCBSP_SPCR_DLB_OFF,             /* 0  */
            MCBSP_SPCR_RJUST_RZF,     /*    00     */
            MCBSP_SPCR_CLKSTP_DISABLE,   /*   0   */
            MCBSP_SPCR_DXENA_OFF,           /* 0  */
            MCBSP_SPCR_RINTM_RRDY,    /*    00    */
            MCBSP_SPCR_RSYNCERR_DEFAULT,  /*  0  */
            MCBSP_SPCR_RRST_DEFAULT         /* 0  */
    ),
#endif

/* RCR Setup */
#if (DMA_SUPPORT)
      MCBSP_RCR_RMK(
            MCBSP_RCR_RPHASE_DUAL,      /*     1     */
            MCBSP_RCR_RFRLEN2_OF(1),    /*   00010   */
            MCBSP_RCR_RWDLEN2_8BIT,     /*    000    */
            MCBSP_RCR_RCOMPAND_MSB,     /*    00     */
            MCBSP_RCR_RFIG_YES,             /*   1   */
            MCBSP_RCR_RDATDLY_1BIT,     /*    01     */
            MCBSP_RCR_RFRLEN1_OF(8),    /*   01000   */
            MCBSP_RCR_RWDLEN1_16BIT /* 010  */
    ),
#endif
#if (EDMA_SUPPORT)
      MCBSP_RCR_RMK(
            MCBSP_RCR_RPHASE_DUAL,      /*     1     */
            MCBSP_RCR_RFRLEN2_OF(1),    /*   00010   */
            MCBSP_RCR_RWDLEN2_8BIT,     /*    000    */
            MCBSP_RCR_RCOMPAND_MSB,     /*    00     */
            MCBSP_RCR_RFIG_YES,             /*   1   */
            MCBSP_RCR_RDATDLY_1BIT,     /*    01     */
            MCBSP_RCR_RFRLEN1_OF(8),    /*   01000   */
            MCBSP_RCR_RWDLEN1_16BIT,    /*    010    */
            MCBSP_RCR_RWDREVRS_DISABLE /* 0  */
    ),
#endif

/* XCR Setup */
#if (DMA_SUPPORT)
      MCBSP_XCR_RMK(
            MCBSP_XCR_XPHASE_DUAL,      /*     1     */
            MCBSP_XCR_XFRLEN2_OF(1),    /*   00010   */
            MCBSP_XCR_XWDLEN2_8BIT,     /*    000    */
            MCBSP_XCR_XCOMPAND_MSB,     /*    00     */
            MCBSP_XCR_XFIG_YES,             /*   1   */
            MCBSP_XCR_XDATDLY_0BIT,     /*    00     */
            MCBSP_XCR_XFRLEN1_OF(8),    /*   01000   */
            MCBSP_XCR_XWDLEN1_16BIT /* 010  */
),
#endif
#if (EDMA_SUPPORT)
      MCBSP_XCR_RMK(
            MCBSP_XCR_XPHASE_DUAL,      /*     1     */
            MCBSP_XCR_XFRLEN2_OF(1)    /*   00010   */
            MCBSP_XCR_XWDLEN2_8BIT,     /*    000    */
            MCBSP_XCR_XCOMPAND_MSB,     /*    00     */
            MCBSP_XCR_XFIG_YES,             /*   1   */
            MCBSP_XCR_XDATDLY_0BIT,     /*    00     */
            MCBSP_XCR_XFRLEN1_OF(8),    /*   01000   */
            MCBSP_XCR_XWDLEN1_16BIT,    /*    010    */
            MCBSP_XCR_XWDREVRS_DISABLE /* 0  */
),
#endif
```

```
            /* SRGR Setup */
                  MCBSP_SRGR_RMK(
                        MCBSP_SRGR_GSYNC_FREE,      /*      0      */
                        MCBSP_SRGR_CLKSP_RISING,     /*     0     */
                        MCBSP_SRGR_CLKSM_INTERNAL,     /*     1     */
                        MCBSP_SRGR_FSGM_DXR2XSR,     /*     0     */
                        MCBSP_SRGR_FPER_DEFAULT,     /*     0     */
                        MCBSP_SRGR_FWID_DEFAULT,     /*     0     */
                        MCBSP_SRGR_CLKGDV_OF(108) /* CLKGDV */
            ),

            /* MCR Setup */
MCBSP_MCR_DEFAULT,                              /* default values */

            /* RCER Setup */
            #if (C64_SUPPORT)
                  MCBSP_RCERE0_DEFAULT,   /*   default   values   */
                  MCBSP_RCERE1_DEFAULT,   /*   default   values   */
                  MCBSP_RCERE2_DEFAULT   /*   default   values   */
                  MCBSP_RCERE3_DEFAULT, /* default values */
            #else
                  MCBSP_RCER_DEFAULT,           /* default values */
            #endif

            /* XCER Setup */
            #if (C64_SUPPORT)
                  MCBSP_XCERE0_DEFAULT,   /*   default   values   */
                  MCBSP_XCERE1_DEFAULT,   /*   default   values   */
                  MCBSP_XCERE2_DEFAULT,   /*   default   values   */
                  MCBSP_XCERE3_DEFAULT, /* default values */
            #else
                  MCBSP_XCER_DEFAULT,           /* default values */
            #endif

            /* PCR Setup */
            MCBSP_PCR_RMK(
                  MCBSP_PCR_XIOEN_SP,           /* 0 */
                  MCBSP_PCR_RIOEN_SP,           /* 0 */
                  MCBSP_PCR_FSXM_INTERNAL,           /* 1 */
                  MCBSP_PCR_FSRM_EXTERNAL,           /* 0 */
                  MCBSP_PCR_CLKXM_OUTPUT,            /* 1 */
                  MCBSP_PCR_CLKRM_OUTPUT,            /* 1 */
                  MCBSP_PCR_CLKSSTAT_0,              /* 0 */
                  MCBSP_PCR_DXSTAT_0,                /* 0 */
                  MCBSP_PCR_FSXP_ACTIVELOW,          /* 1 */
                  MCBSP_PCR_FSRP_ACTIVELOW,          /* 1 */
                  MCBSP_PCR_CLKXP_RISING,            /* 0 */
                  MCBSP_PCR_CLKRP_FALLING            /* 0 */
            )
      };

      MCBSP_config(hMcbsp1, &mcbspCfg1);

}     /* end of Config_McBSP(void) */


/*****************************************************************/
/* void ProcessTransmitData(void)                                */
/*                                                               */
/* This function expands each of the 8-bit ASCII characters in the */
/* transmit string "xmit_msg" into UART transmission 16-bit word   */
/* and place them in the transmit buffer "xmitbuf".  In addition,  */
/* 16-bit Start and 8-bit Stop framing words, respectively, are    */
/* inserted before and after each of the ASCII characters in the   */
/* buffer.                                                       */
/*****************************************************************/
void ProcessTransmitData(void)
{
      Int         i;
      Short cnt = 1;
      unsigned char    xmit_char;
      unsigned short   *xmitbufptr;
```

```
        /* point to Transmit Buffer */
        xmitbufptr = (unsigned short *)xmitbuf;

        for (i=0; i<(sizeof(xmitbuf)/sizeof(unsigned int)); i++)
        {
             xmitbufptr[i] = 0x0000;   /* zero fill buffer    */
        }

        xmitbufptr = (unsigned short *)xmitbuf;

        /* Process data BYTES in xmit_msg[] and put in xmit buffer   */
        for (i = 0; i < BUFFER_SIZE; i++)
        {
             /* Get transmit character (one byte) from xmit_msg[] */
             /* and put in xmit buffer */
             xmit_char = xmit_msg[i];

             /* Process each BYTE of transmit character    */
             for (cnt = -1; cnt < 10; cnt++)
             {
                  if (cnt == -1)
                       *xmitbufptr++ =  0x0000;

                  else if (cnt == 8 || cnt ==9)
                       *xmitbufptr++ = 0xFFFF;

                  else if (xmit_char & (1 << cnt))
                       *xmitbufptr++ = 0xFFFF;

                  else
                       *xmitbufptr++ = 0x0000;

                  } /* end for cnt               */

             }     /* end for I                 */

}            /* end ProcessTransmitData          */


/*********************************************************************/
/* void ProcessReceiveData(void)                                     */
/*                                                                   */
/* This function decodes the data in the receive buffer, "recvbuf" */
/* and strips the framing start (0x0000) and Stop (0xFFFF) words.  */
/* It calls the subroutine VoteLogic() to determine each bit of    */
/* the ASCII character.  It then puts the result in recv_msg.      */
/*********************************************************************/
void ProcessReceiveData(void)
{
     Int i;
     unsigned char recv_char = 0;
     short cnt = -1;
     short recv_val;
     unsigned short   raw_data;
     unsigned short   *recvbufptr;    /*receive buffer pointer*/

     /* Point to the receive buffer      */
     Recvbufptr = (unsigned short *)recvbuf;

     /* Process all data in the Receive buffer    */
     for (i = 0; i < BUFFER_SIZE; i++)
     {
          recv_char = 0;

          /* Process each UART frame    */
          for (cnt = -1; cnt < 10; cnt++)
          {
               if(cnt == -1 || cnt == 8 || cnt == 9)
               {
                    /* Ignore Start and Stop bits    */
                    *recvbufptr++;
               }
               else
               {
```

```
                        /* Get 16-bit data from receive buffer        */
                        raw_data = *recvbufptr;
                        recvbufptr++;

                        /* get the value of the majority of the bits */
                        recv_val = VoteLogic(raw_data);

                        put received bit into proper place            */
                        recv_char    += recv_val << cnt;
                  }
            }     /* end for cnt       */

                  /* A full BYTE is decoded. Put in result: recv_msg[i]     */
                  recv_msg[i] = recv_char;

            }     /* end for I                                              */

      }        /* end ProcessReceiveData() function                         */


      /*****************************************************************/
      /* void CheckTestCase(void)                                      */
      /*****************************************************************/
      int CheckTestCase(void)
      {
              unsigned short *source;
              unsigned short *result;
              unsigned int i = 0;
              short cnt = -1;
              int error = 0;

              source = (unsigned short *) xmitbuf;
              result = (unsigned short *) recvbuf;

              for (i = 0; i < BUFFER_SIZE ; i++)
              {
                   for (cnt = -1; cnt < 10; cnt++)
                   {
                        /* Ignore the start and stop bits */
                        if(cnt == -1 || cnt == 8 || cnt ==9)
                        {
                             source++;
                             result++;
                        }
                        else
                        {
                             if (*source != *result)
                             {
                                  error = i + 1;
                                  break;
                             }
                                  source++;
                                  result++;
                             }
                        }
                   }

                   return(error);

      }           /* end CheckTestCase() function */


      /*****************************************************************/
      /* short VoteLogic(unsigned short)                               */
      /*                                                               */
      /* This function decoded the received character by testing the   */
      /* center 4 bits of the baud.  A majority rule is used for the   */
      /* decoding.                                                     */
      /*****************************************************************/
      short VoteLogic(unsigned short value)
      {
            short returnvalue;

            switch    ((value    >> 6) & 0x0F)
```

```
        {
              Case   0:
              Case   1:
              Case   2:
              Case   3:
              Case   4:
              Case   5:
              Case   6:
              Case   8:
              Case   9:

              Case   10:
                                    returnvalue = 0;
                                    break;
              case   7:
              case   11:
              case   12:
              case   13:
              case   14:
              case   15:
                                    returnvalue = 1 ;
                                    break;
        }     /* end switch          */

        return (returnvalue);

}     /* end VoteLogic() function   */

/*********************************************************************/
/* EDMA Data Transfer Completion ISRs                                */
/*********************************************************************/

interrupt void c_int11(void) /* DMA 2 */
{
     #if (DMA_SUPPORT)
          transmit_done = TRUE;
          printf("Transmit Completed\n");
     #endif
}

interrupt void c_int09(void) /* DMA 1 */
{
     #if (DMA_SUPPORT)
          receive_done = TRUE;
          printf("Receive Completed\n");
     #endif
}

interrupt void c_int08(void)
{
     #if (EDMA_SUPPORT)
          if (EDMA_intTest(14))
          {
               EDMA_intClear(14);
               transmit_done = TRUE;
               printf("Transmit Completed\n");
          }

          if (EDMA_intTest(15))
          {
               EDMA_intClear(15);
               receive_done = TRUE;
               printf("Receive Completed\n\n");
          }
     #endif
}
```

Appendix B Sample C/Assembly Code

B.1 GPIO Mode

```
;*************************************************************************
;* TEXAS INSTRUMENTS, INC.                                               *
;* SOFTWARE UART EMULATION FOR TMS320C6000                               *
;* Revision Date: 18/5/99                                                *
;* Author : Philippe Malleth                                             *
;*                                                                       *
;* USAGE                                                                 *
;*    These routines are C-callable and can be called as:                *
;*                                                                       *
;*       unsigned int SoftUartSpeedDetect(void);                         *
;*       void         SoftUartOutchar(int, char);                        *
;*       char         SoftUartInchar(int);                               *
;*                                                                       *
;*    If the routine is not to be used as a C-callable function,         *
;*    then all instructions relating to the stack should be removed.     *
;*    See comments of individual instructions to determine if they are   *
;*    related to the stack. You also need to initialize all passed       *
;*    parameters since these are assumed to be in registers as defined by*
;*    the calling convention of the compiler, (See the C compiler        *
;*    reference guide.)                                                  *
;*                                                                       *
;* DESCRIPTION                                                           *
;*    These routines are used to emulate the RX/TX behavior of a RS232 UART. *
;*    The RX and TX waveforms are generated with the McBSP pins put in   *
;*    GPIO mode. See the Application report for further details.         *
;*                                                                       *
;* C CODE                                                                *
;*    This is the C equivalent of the assembly code without restrictions:*
;*    Note that the assembly code is hand optimized and restrictions may *
;*    apply.                                                             *
;*                                                                       *
;*                                                                       *
;*    Calling convention :                                               *
;*                                                                       *
;*    This is a very basic example that first gets the right baudrate from*
;*    subroutine SoftUartSpeedDetect(), then forever waits for a character*
;*    and send it out as soon as received.                               *
;*                                                                       *
;*       main()                                                          *
;*       {                                                               *
;*       unsigned int UartSpeed;                                         *
;*       char c;                                                         *
;*                                                                       *
;*           UartSpeed = SoftUartSpeedDetect();                          *
;*           for(;;) {                                                   *
;*               c = SoftUartInchar(UartSpeed);                          *
;*               SoftUartOutchar(UartSpeed,c);                           *
;*           }                                                           *
;*       }                                                               *
;*                                                                       *
;*                                                                       *
;*                                                                       *
                                                        *
;*                                                                       *
;*    Subroutine details :                                               *
;*                                                                       *
;* unsigned int SoftUartSpeedDetect(void)                                *
;* {                                                                     *
;* volatile unsigned int speedcounter, i;                                *
;*                                                                       *
;*     MCBSP_IO_ENABLE(1);                                               *
;*     speedcounter = 0;                                                 *
;*     while( (int)MCBSP_DRSTAT(1));                                     *
;*     while(!(int)MCBSP_DRSTAT(1)){   /* counts START bit             */   *
;*         speedcounter++;                                               *
;*     }                                                                 *
;*     while( (int)MCBSP_DRSTAT(1)){   /* counts DR high (from <cr>)   */   *
```

```
;*         speedcounter++;                                                  *
;*     }                                                                    *
;*     MCBSP_DX_IO_H(1);                                                    *
;*     for(i=11*speedcounter;i>0;i--); /* wait long enough for one char */  *
;*     speedcounter >>= 2;             /* speedcounter divide by 2      */  *
;*     return(speedcounter);                                                *
;* }                                                                        *
;*                                                                          *
;*                                                                          *
;* char SoftUartInchar(int speedcnt)                                        *
;* {                                                                        *
;* volatile unsigned int incomingChar,speedcounter,tmpcounter;              *
;* volatile unsigned int tmpreg,dxstat;                                     *
;* unsigned int lsb;                                                        *
;*                                                                          *
;*     incomingChar=0;                                                      *
;*     speedcounter = speedcnt;                                             *
;*     tmpcounter = speedcounter>>1;                                        *
;*     while((int)MCBSP_DRSTAT(1));                                         *
;*     while(--tmpcounter != 0)                                             *
;*         tmpreg = (unsigned int)MCBSP_DRSTAT(1); /* counts half bit time */*
;*     for(tmpreg = 9;tmpreg!=0;--tmpreg) {                                 *
;*         tmpcounter = speedcounter  ;                                     *
;*         while(--tmpcounter != 0)                                         *
;*            dxstat = (unsigned int)MCBSP_DRSTAT(1); /* counts bit time   */*
;*         if(dxstat==1) incomingChar++;                                    *
;*         lsb = incomingChar&1;                                            *
;*         incomingChar >>=1;                                               *
;*         if (lsb) incomingChar+=0x80000000;                               *
;*     }                                                                    *
;*     incomingChar >>=23;                                                  *
;*     return((char)incomingChar);                                          *
;* }                                                                        *
;*                                                                          *
;*                                                                          *
;* void SoftUartOutchar(int speedcnt, char outgoingChar)                    *
;* {                                                                        *
;* unsigned int c,carry;                                                    *
;* volatile unsigned int paddedChar,bitcounter,tmpcounter;                  *
;*                                                                          *
;*     MCBSP_DX_IO_L(1);                                                    *
;*     carry = 0;                                                           *
;*     paddedChar = ((unsigned int) outgoingChar) | 0x00000700;             *
;*     for(bitcounter=11;bitcounter>0;bitcounter--) {                       *
;*         tmpcounter = speedcnt;                                           *
;*         while(--tmpcounter != 0) (int)MCBSP_DRSTAT(1);                   *
;*         c = carry;                                                       *
;*         carry = paddedChar&1;                                            *
;*         paddedChar >>=1;                                                 *
;*         if (carry) MCBSP_DX_IO_H(1);                                     *
;*         else       MCBSP_DX_IO_L(1);                                     *
;*     }                                                                    *
;*     MCBSP_DX_IO_H(1);                                                    *
;* }                                                                        *
;*                                                                          *
;**************************************************************************

      .global _SoftUartSpeedDetect
          .global _SoftUartInchar
          .global _SoftUartOutchar
          .sect ".text"
;**************************************************************************
;* FUNCTION NAME: _SoftUartSpeedDetect                                      *
;*                                                                          *
;* USAGE                                                                    *
;*    This routines are C-callable and can be called as:                    *
;*                                                                          *
;*      unsigned int SoftUartSpeedDetect(void);                             *
;*                                                                          *
;*    The McBSP1 at address 0x1900000 is used here.                         *
;*    Argument 1    : None.                                                 *
;*    Return Value : ASCII coded character read from a terminal.            *
;*                                                                          *
;**************************************************************************
```

```
_SoftUartSpeedDetect:
;** ---------------- function prolog ------------------------------------- *
;** preserve "save-on-call" registers
           SUB             B15, 4, A0
           STW     .D2     A10, *B15--[2]    ; f
||         STW     .D1     B10, *A0--[2]     ; f
           STW     .D2     A11, *B15--[2]    ; f
||         STW     .D1     B11, *A0--[2]     ; f
           STW     .D2     A12, *B15--[2]    ; f
||         STW     .D1     B12, *A0--[2]     ; f
           STW     .D2     A13, *B15--[2]    ; f
||         STW     .D1     B13, *A0--[2]     ; f
||         MVC     .S2     CSR,B13           ; f
           STW     .D2     A14, *B15--[2]    ; f
||         STW     .D1     B14, *A0--[2]     ; f
||         AND     .L2     -2,B13,B13        ; f
           STW     .D2     A15, *B15--[2]    ; f
||         STW     .D1     B3,  *A0--[2]     ; f
||         MVC     .S2     B13,CSR           ; f disable global interrupts
;** ---------------------------------------------------------------------- *
           MVK     .S1     0x8,A0            ;   set offset to SPCR register
           MVKH    .S1     0x1900000,A0      ;   takes McBSP1 port address
           LDW     .D1T1   *A0,A3            ;   load SPCR register
           NOP             4
           CLR     .S1     A3,0x10,0x10,A3   ;
           AND     .L1     0xfffffffe,A3,A3  ;
           STW     .D1T1   A3,*A0            ;   store new SPCR config value
||         MVK     .S1     0x24,A0           ;   set offset for PCR register
           MVKH    .S1     0x1900000,A0      ;   takes McBSP1 port address
           LDW     .D1T1   *A0,A3            ;   load PCR register
           NOP             4
           SET     .S1     A3,0xc,0xd,A3     ;   set bit 12&13 for I/O mode
           STW     .D1T1   A3,*A0            ; store new PCR config value
           NOP  5
           LDW     .D1T1   *A0,A3            ;
           NOP             4
           EXTU    .S1     A3,0x1b,0x1f,A1   ;   wait while DRSTAT is high
;** ---------------------------------------------------------------------- *
      .align 32
L1:[ A1]   B       .S2     L1                ;
|| [ A1]   LDW     .D1T1   *A0,A3            ;
||         EXTU    .S1     A3,0x1b,0x1f,A1   ;   wait while DRSTAT is high
|| [!A1]   ZERO    .L2     B4                ;   initialize counter
           NOP             5                 ;   for StartBit measurement
;** ---------------------------------------------------------------------- *
           .align 32
L3:[ A1]   B       .S2     L3                ;
|| [ A1]   LDW     .D1T1   *A0,A3            ;
||         EXTU    .S1     A3,0x1b,0x1f,A1   ;
|| [!A1]   ADD     .L2     0x1,B4,B4         ;   increment counter while
           NOP             5                 ;   DRSTAT bit is low
;** ---------------------------------------------------------------------- *
           .align 32
L31B:[ A1] B       .S2     L31B              ;
|| [ A1]   LDW     .D1T1   *A0,A3            ;
||         EXTU    .S1     A3,0x1b,0x1f,A1   ;
|| [ A1]   ADD     .L2     0x1,B4,B4         ;   increment counter while
           NOP             5                 ;   DRSTAT bit is low
;** ---------------------------------------------------------------------- *
           .align 32
           SHRU    .S2     B4,0x1,B4         ;
           MVK     .S2     0x0b,B0           ;
           SET     .S1     A3,0x5,0x5,A3     ;   set DXSTAT bit to 1
||         MV      .L1X    B0,A4             ;
           MPYLHU  .M1X    A4,B4,A3          ;
||         STW     .D1T1   A3,*A0            ;   store new PCR config value
           MPYU    .M2     B0,B4,B0          ;
           SHL     .S1     A3,0x10,A3        ;
           ADD     .L2X    B0,A3,B0          ;
;** ---------------------------------------------------------------------- *
          .align 32
waitcnt: [ B0]   B .S1     waitcnt           ;
||    [ B0] SUB     .L2     B0,0x1,B0         ;
||    [ B0] LDW     .D1T1   *A0,A3            ; Dummy load
```

```
           NOP             5
           ; BRANCH OCCURS                 ;
;** ----------------- function epilog ------------------------------------- *
;** restore preserved by call registers
           SUB             B15, 4, A0
           LDW     .D1     *++A0[2], B3      ; f
||         LDW     .D2     *++B15[2], A15    ; f
||         MVC     .S2     CSR, B13          ; f
           LDW     .D1     *++A0[2], B14     ; f
||         LDW     .D2     *++B15[2], A14    ; f
||         OR      .L2     B13, 1, B13       ; f
           LDW     .D1     *++A0[2], B13     ; f
||         LDW     .D2     *++B15[2], A13    ; f
||         MVC     .S2     B13,CSR           ; f enable global interrupts
           LDW     .D1     *++A0[2], B12     ; f
||         LDW     .D2     *++B15[2], A12    ; f
           LDW     .D1     *++A0[2], B11     ; f
||         LDW     .D2     *++B15[2], A11    ; f
||         B       .S2     B3                ; f return();
||         MV      .L1X    B4,A4             ;
           LDW     .D2     *++B15[2], A10    ; f
||         LDW     .D1     *++A0[2], B10     ; f
           NOP             4                 ; f
;** --------------------------------------------------------------------------*
;****************************************************************************
;* FUNCTION NAME: _SoftUartInchar                                           *
;*                                                                          *
;* USAGE                                                                    *
;*    This routines are C-callable and can be called as:                    *
;*                                                                          *
;*      char SoftUartInchar(int speedcnt);                                  *
;*                                                                          *
;*    The McBSP1 at address 0x1900000 is used here.                         *
;*    Argument 1    : Speed counter value returned by SoftUartSpeedDetect(). *
;*    Return Value : ASCII coded character read from a terminal.            *
;*                                                                          *
;****************************************************************************
_SoftUartInchar:
;** ----------------- function prolog ------------------------------------- *
;** preserve "save-on-call" registers
           SUB             B15, 4, A0
           STW     .D2     A10, *B15--[2]    ; f
||         STW     .D1     B10, *A0--[2]     ; f
           STW     .D2     A11, *B15--[2]    ; f
||         STW     .D1     B11, *A0--[2]     ; f
           STW     .D2     A12, *B15--[2]    ; f
||         STW     .D1     B12, *A0--[2]     ; f
           STW     .D2     A13, *B15--[2]    ; f
||         STW     .D1     B13, *A0--[2]     ; f
||         MVC     .S2     CSR,B13           ; f
           STW     .D2     A14, *B15--[2]    ; f
||         STW     .D1     B14, *A0--[2]     ; f
||         AND     .L2     -2,B13,B13        ; f
           STW     .D2     A15, *B15--[2]    ; f
||         STW     .D1     B3,  *A0--[2]     ; f
||         MVC     .S2     B13,CSR           ; f disable global interrupts
;** --------------------------------------------------------------------------*
           MVK     .S2     0x24,B4           ; set offset to SPCR register
||         MV      .L1     A4,A0             ;
           MVKH    .S2     0x1900000,B4      ; takes McBSP1 port address
||         SHRU    .S1     A0,0x2,A1         ;
||         ZERO    .L1     A4                ;
           LDW     .D2T2   *B4,B5            ;
           NOP             4
           EXTU    .S2     B5,0x1b,0x1f,B0   ; wait while DRSTAT bit is high
;** --------------------------------------------------------------------------*
           .align 32
L2:[ B0]   B       .S1     L2                ;
|| [ B0]   LDW     .D2T2   *B4,B5            ;
||         EXTU    .S2     B5,0x1b,0x1f,B0   ; wait while DRSTAT bit is high
           NOP             5
           ; BRANCH OCCURS ; |16|
;** --------------------------------------------------------------------------*
           .align 32
```

```
L4:[ A1]   B       .S1     L4                ;
|| [ A1]   SUB     .L1     A1,0x1,A1         ; while (--tmpcounter !=0 )
|| [ A1]   LDW     .D2T2   *B4,B0            ; dummy load
           NOP             5
           MVK     .S2     0x8,B1            ;
||         MV      .L1     A0,A1             ;
;** --------------------------------------------------------------------------*
          .align 32
L8:[ A1]   B       .S1     L8                ;
|| [ A1]   SUB     .L1     A1,0x1,A1         ;   while (--tmpcounter !=0 )
|| [ A1]   LDW     .D2T2   *B4,B0            ;   do read DRSTAT bit
||         EXTU    .S2     B0,0x1b,0x1f,B0   ;
           NOP     5
           ; BRANCH OCCURS ; |33|
;** --------------------------------------------------------------------------*
          .align 32
   [ B1]   B       .S1     L8                ;
|| [ B0]   ADD     .L1     0x1,A4,A4         ;
           AND     .L1     0x1,A4,A1         ;   lsb = incomingChar&1
||         SHRU    .S1     A4,0x1,A4         ;   incomingChar >>= 1
   [ A1]   SET     .S1     A4,0x1f,0x1f,A4   ;   if (lsb) incomingChar
                                             ;         +=0x80000000
|| [ B1]   MV      .L1     A0,A1             ;
|| [ B1]   SUB     .L2     B1,0x1,B1         ;
           NOP     3
           ; BRANCH OCCURS ; |42|
;** ----------------- function epilog ------------------------------------- *
;** restore preserved by call registers
           SUB             B15, 4, A0
           LDW     .D1     *++A0[2], B3      ; f
||         LDW     .D2     *++B15[2], A15    ; f
||         MVC     .S2     CSR, B13          ; f
           LDW     .D1     *++A0[2], B14     ; f
||         LDW     .D2     *++B15[2], A14    ; f
||         OR      .L2     B13, 1, B13       ; f
           LDW     .D1     *++A0[2], B13     ; f
||         LDW     .D2     *++B15[2], A13    ; f
||         MVC     .S2     B13,CSR           ; f enable global interrupts
           LDW     .D1     *++A0[2], B12     ; f
||         LDW     .D2     *++B15[2], A12    ; f
           LDW     .D1     *++A0[2], B11     ; f
||         LDW     .D2     *++B15[2], A11    ; f
||         B       .S2     B3                ; f return();
||         SHRU    .S1     A4,0x17,A4        ;   incomingChar >>= 23
           LDW     .D2     *++B15[2], A10    ; f
||         LDW     .D1     *++A0[2], B10     ; f
           NOP             4                 ; f
;** --------------------------------------------------------------------------*


;****************************************************************************
;* FUNCTION NAME: _SoftUartOutchar                                          *
;*                                                                          *
;* USAGE                                                                    *
;*    This routine are C-callable and can be called as:                     *
;*                                                                          *
;*      void SoftUartOutchar(int speedcnt, char r1);                        *
;*                                                                          *
;*    The McBSP1 at address 0x1900000 is used here.                         *
;*    Argument 1   : Speed counter value returned by SoftUartSpeedDetect(). *
;*    Argument 2   : ASCII coded character to be send out to a terminal.    *
;*    Return Value : None                                                   *
;*                                                                          *
;****************************************************************************
_SoftUartOutchar:
;** ----------------- function prolog ------------------------------------- *
;** preserve "save-on-call" registers
           SUB             B15, 4, A0
           STW     .D2     A10, *B15--[2]    ; f
||         STW     .D1     B10, *A0--[2]     ; f
           STW     .D2     A11, *B15--[2]    ; f
||         STW     .D1     B11, *A0--[2]     ; f
           STW     .D2     A12, *B15--[2]    ; f
||         STW     .D1     B12, *A0--[2]     ; f
```

```
            STW     .D2     A13, *B15--[2]    ; f
||          STW     .D1     B13, *A0--[2]     ; f
||          MVC     .S2     CSR,B13           ; f
            STW     .D2     A14, *B15--[2]    ; f
||          STW     .D1     B14, *A0--[2]     ; f
||          AND     .L2     -2,B13,B13        ; f
            STW     .D2     A15, *B15--[2]    ; f
||          STW     .D1     B3,  *A0--[2]     ; f
||          MVC     .S2     B13,CSR           ; f disable global interrupts
;** --------------------------------------------------------------------------*
            MVK     .S1     0x24,A0
            MVKH    .S1     0x1900000,A0      ;   takes McBSP1 port address
            LDW     .D1T1   *A0,A3            ;
            ZERO    .L2     B1                ;   carry = 0
            MVK     .S2     0xb,B2            ;   bitcounter = 11 bit
                                              ;     (1st,8dt,2stp)
            SET     .S2     B4,0x8,0xA,B4     ;   paddedChar = outgoingChar
                                              ;        | 0x00000700
            MV      .L2X    A4,B0             ;
            CLR     .S1     A3,0x5,0x6,A3     ;   set DXSTAT bit to 0
            STW     .D1T1   A3,*A0            ;   store new PCR config value
;** --------------------------------------------------------------------------*
          .align 32
loopcnt4: [B0]   B       .S1     loopcnt4     ;
|| [ B0]   LDW     .D1T1   *A0,A3            ;   Dummy load
||          MV      .L1X    B1,A1             ;   c = carry
|| [ B0]   SUB     .L2     B0,0x1,B0         ;   while (--tmpcounter !=0 )
            NOP      5
;** --------------------------------------------------------------------------*
          .align 32
   [ B2]    B       .S1     loopcnt4          ;   wait 11 bits released
||          AND     .L2     0x1,B4,B1         ;   carry = paddedChar&1
            NOP      2
   [ B2]    SUB     .L2     B2,0x1,B2         ;   bitcounter--
||          SHRU    .S2     B4,0x1,B4         ;   paddedChar >>= 1
|| [ B1]   SET     .S1     A3,0x5,0x6,A3     ;   set DXSTAT bit to 1
   [!B1]   CLR     .S1     A3,0x5,0x6,A3     ;   set DXSTAT bit to 0
||          MV      .L2X    A4,B0             ;
            STW     .D1T1   A3,*A0            ;   store new PCR config value
            ; BRANCH OCCURS                   ;
;** --------------------------------------------------------------------------*
            LDW     .D1T1   *A0,A3            ;
            NOP             4
            SET     .S1     A3,0x5,0x6,A3     ;   set DXSTAT bit to 1
            STW     .D1T1   A3,*A0            ;   store new PCR config value
;** ----------------- function epilog ------------------------------------- *
;** restore preserved by call registers
            SUB             B15, 4, A0
            LDW     .D1     *++A0[2], B3      ; f
||          LDW     .D2     *++B15[2], A15    ; f
||          MVC     .S2     CSR, B13          ; f
            LDW     .D1     *++A0[2], B14     ; f
||          LDW     .D2     *++B15[2], A14    ; f
||          OR      .L2     B13, 1, B13       ; f
            LDW     .D1     *++A0[2], B13     ; f
||          LDW     .D2     *++B15[2], A13    ; f
||          MVC     .S2     B13,CSR           ; f enable global interrupts
            LDW     .D1     *++A0[2], B12     ; f
||          LDW     .D2     *++B15[2], A12    ; f
            LDW     .D1     *++A0[2], B11     ; f
||          LDW     .D2     *++B15[2], A11    ; f
||          B       .S2     B3                ; f return();
            LDW     .D2     *++B15[2], A10    ; f
||          LDW     .D1     *++A0[2], B10     ; f
            NOP             4                 ; f
;** --------------------------------------------------------------------------*
```

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