Application Report
SPRA633 - March 2000

TMS320C6000 McBSP: UART

Todd Hiers
Rebecca Ma
Philippe Malleth

Digital Signal Processing Solutions

ABSTRACT

This document describes how to use the multichannel buffered serial port (McBSP) in the Texas Instruments (TI™) TMS320C6000 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 in which the McBSP can be configured to receive and transmit data that is understandable to an UART. The McBSP can be used in either the serial port mode or the general purpose input/output mode. This application report discusses both methods. In addition, this application report demonstrates the hardware interface between the McBSP and a UART.

Contents

List of Figures

List of Tables

1 Design Problem

How can the multichannel buffered serial port (McBSP) in a TMS320C6000 digital signal processor be used for transmitting data to and receiving data from an UART?

2 Overview

The Universal Asynchronous Receiver/Transmitter (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 LSB (Least Significant Bit) first. The parity bit is optional, depending on the UART format. Each data frame ends with the Stop bit (logic high).

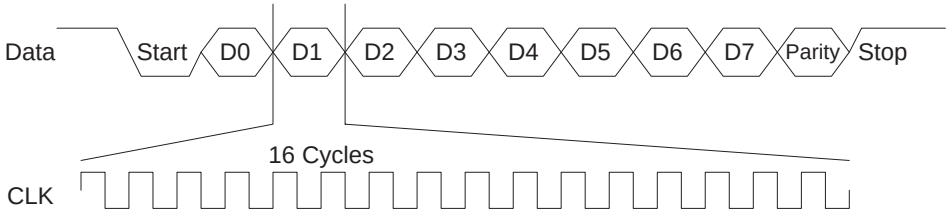


Figure 1. UART Timing

To interface an 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 CMOS logic levels to the RS-232 logic levels. The RS-232 logic levels use +3 to +25 volts to signify a “Space” (logic 0) and –3 to –25 volts for a “Mark” (logic 1). Any voltage in between these regions (i.e. between +3 and –3 Volts) is undefined.

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

3 UART Interface Method 1: McBSP in Serial Port Mode

To interface an 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.

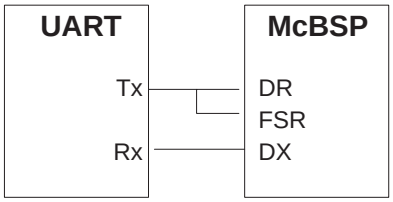
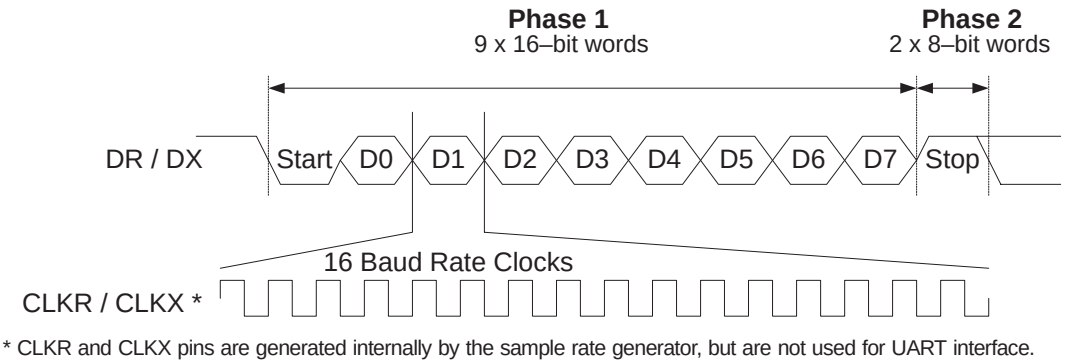


Figure 2. UART Connection—Serial Port Implementation

3.1 McBSP Setup: Serial Port Implementation

The C6000 McBSP 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.

To send a byte to an UART in 8N1 mode (eight data bits, no parity bit, and one stop bit), the transfer should be in two phases, one consisting of 9 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, and 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 0xffff and ‘0’ UART bits are encoded as 0x0000. 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.



* CLKR and CLKX pins are generated internally by the sample rate generator, but are not used for UART interface.

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. The values in the shaded bit-fields are “don’t cares”. Table 1 summarizes the McBSP setup.

- Pin Control Register (PCR)
  - FSXM = 1. This allows the sample rate generator to control the beginning of transmit frames.
  - 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.

31															16
0x0000															
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		0	0	1	0	1	1	0	0	0	0	0	1	0	0
rsv		XIOEN	RIOEN	FSXM	FSRM	CLKXM	CLKRM	rsv	CLKS_STAT	DX_STAT	DR_STAT	FSXP	FSRP	CLKXP	CLKRP

Figure 4. Pin Control Register (PCR)

- Receive/Transmit Control Registers (RCR/XCR)
  - R/XPHASE = 1. Enable dual-phase frame mode.
  - R/XFRLEN1 = 8. Nine elements in the first phase of the frame.
  - R/XFRLEN2 = 1. Two elements in the second phase of the frame.
  - R/XWDLEN1 = 2. 16-bit words in the first phase (Start bit, data bits).
  - R/XWDLEN2 = 0. 8-bit words in the second phase (Stop bit).
  - R/XFIG = 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 an UART frame. Unexpected frame syncs are ignored.

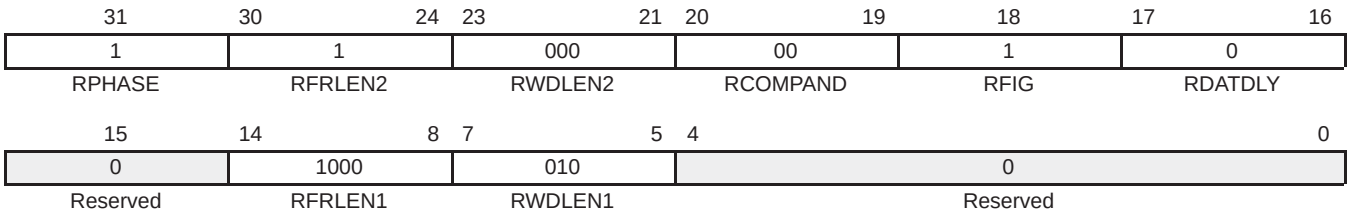


Figure 5. Pin Control Register (PCR)

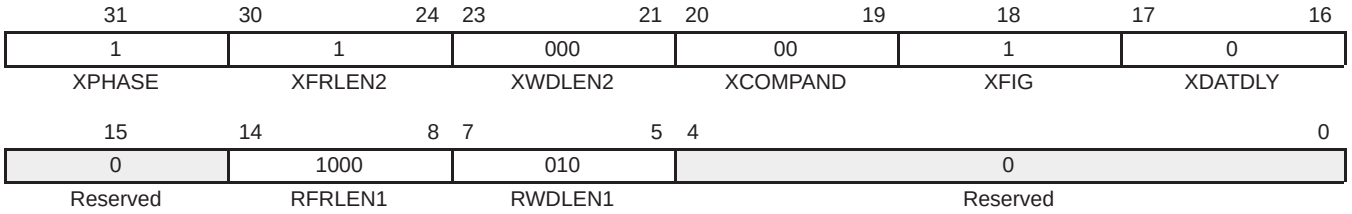


Figure 6. Transmit Control Register (XCR)

- 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.

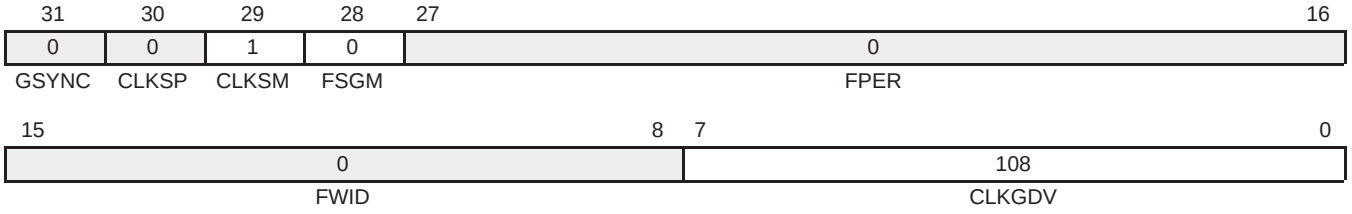


Figure 7. Sample Rate Generator Register (SRGR)

Table 1. Bit-Field Values for McBSP Registers

Register [Bit-Field No.]	Bit-Field Name	Value (in Binary)	Function
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[18]	RFIG	1	Unexpected FSR ignored
RCR[14–8]	RFRLEN1	1000	9 word Receive Frame Length (Phase 1)
RCR[7–5]	RWDLEN1	010	16 bits Receive Word Length (Phase 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[18]	XFIG	1	Unexpected FSX ignored
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
SRGR[7–0]	CLKGDV	1101100 (108)	CLKX = CPU clock divided by 109
PCR[11]	FSXM	1	FSX is output pin
PCR[9]	CLKXM	1	CLKX is output pin
PCR[8]	CLKRM	1	CLKR is output pin

3.2 Receiving/Transmitting UART Data

Once the serial port has been configured to interface to an 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 DMA services the McBSP by transferring data between the McBSP and the receive and transmit buffers. For the ‘C6211/6711 device, replace the subroutine run_dma() with run_edma() to set up the EDMA instead of the DMA to service the McBSP.

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 (0x0000) and Stop (0xffff) bits in the proper locations. Figure 8 shows an example transmit buffer. Afterward, the DMA 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.

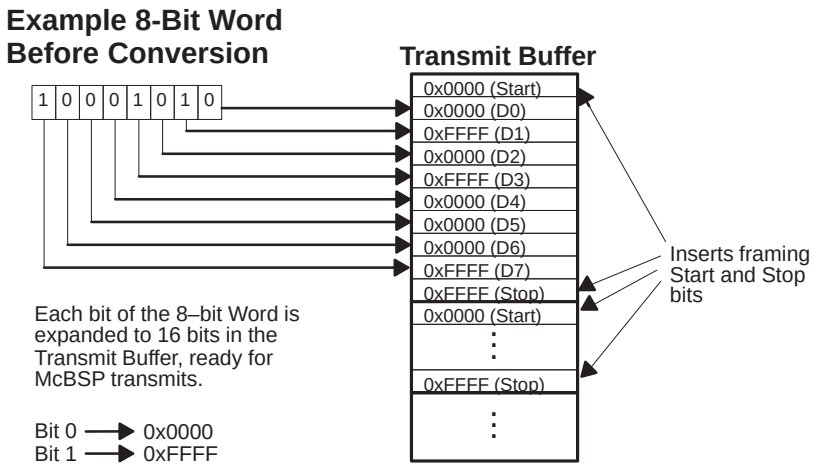


Figure 8. Block Data Processing Transmit Buffer

For receives, the DMA reads the expanded data from the McBSP receiver and writes this data to a receive buffer. The software holds off data processing until the DMA has finished moving a block of data, including the framing Start and Stop bits, to the receive buffer. A receive compression subroutine is then called to compress the received data into UART bytes.

4 UART Interface Method 2: McBSP in GPIO Mode

The C6000 DSP can also interface to an UART using its general purpose input/output pins. The McBSP pins CLKX, FSX, DX, CLKR, FSR, DR, and CLKS can be used as general purpose I/O pins when the following two conditions are true:

- The related portion (transmitter or receiver) of the serial port is in reset: /(R/X)RST = 0 in the serial port control register SPCR
- General-purpose I/O is enabled for the related portion of the serial port: (R/X)IOEN=1 in the pin control register PCR.

Table 2 shows how to setup the McBSP pins as general purpose I/O pins.

Table 2. Configuration of McBSP Pins as GPIO

Pin	GPIO Enabled When…	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	/XRST = 0 XIOEN = 1	FSXM = 1	FSXP	FSXM = 0	FSXP
DX	/XRST = 0 XIOEN = 1	Always	DX_STAT	Never	N/A
CLKR	/RRST = 0 RIOEN = 1	CLKRM = 1	CLKRP	CLKRM = 0	CLKRP
FSR	/RRST = 0 RIOEN = 1	FSRM = 1	FSRP	FSRM = 0	FSRP
DR	/RRST = 0 RIOEN = 1	Never	N/A	Always	DR_STAT
CLKS	/RRST = /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 setup of all other McBSP registers are don’t cares because the McBSP is in general purpose I/O mode. The values in shaded bit fields are don’t cares and are left at default.

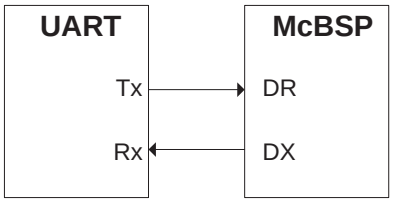


Figure 9. UART Connection – GPIO Implementation

31 – 24	23	22	21 – 20	19	18	17	16
000000000	0	0	00	0	0	0	0
Reserved	/FRST	/GRST	XINTM	XSYNCERR	XEMPTY	XRDY	/XRST

15	14 – 13	12 – 11	10 – 8	7	6	5 – 4	3	2	1	0
0	00	00	000	0	0	00	0	0	0	0
DLB	RJUST	CLKSTP	Rsvd	DXENA	Rsvd	RINTM	RSYNCERR	RFULL	RRDY	/RRST

Figure 10. Serial Port Control Register

31 – 16
0x0000
Reserved

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

Figure 11. Pin Control Register

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). Users 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.

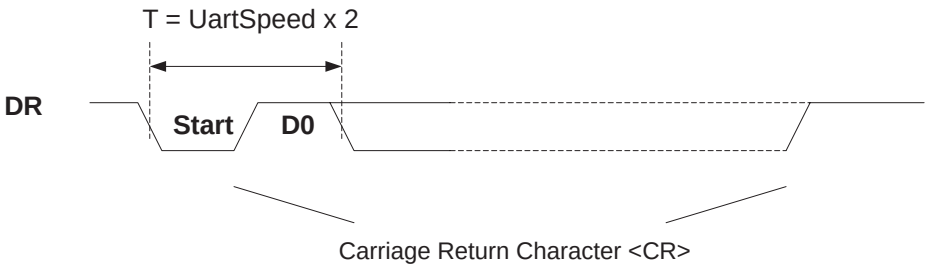


Figure 12. UART Auto-Baud Detection

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 8 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.

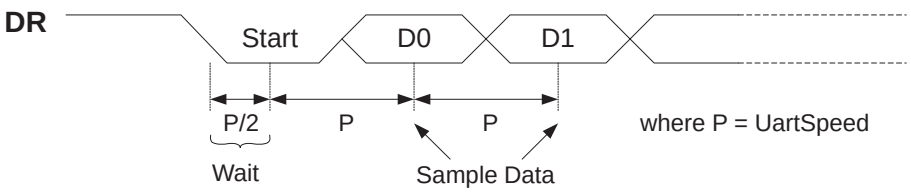


Figure 13. SoftUartInchar UART Data Fetch

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 spend on the each bit is derived from the UartSpeed input, processed by a polling loop.

The transmit character is first placed into the least significant 8 bits in a register padded with three Stop bits (0x00000700). For example, the character ‘A’ (ASCII character 0x41) will be placed in the padded register to become 0x00000741. Each time through the for loop, the least significant bit (LSB) of the padded register is driven on the DX line, and the padded register 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 0x000003A0.

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.

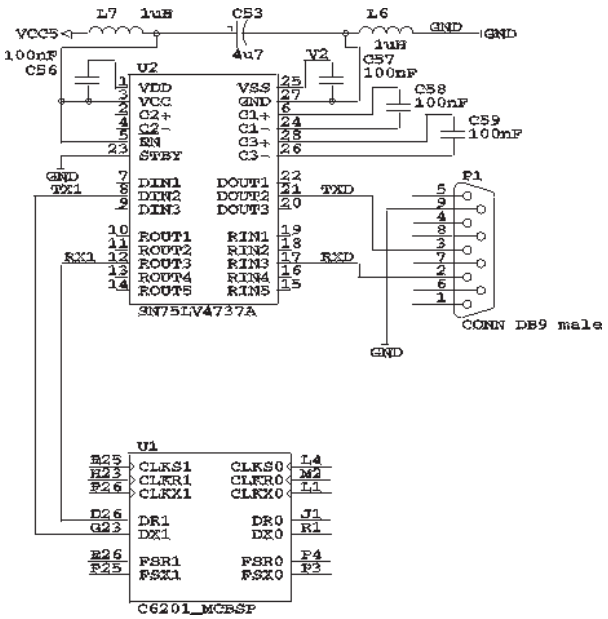


Figure 14. UART Adapter Board

6 Conclusion

The Multichannel Buffered Serial Port on the TMS320C6000 Digital Signal Processor is not natively capable of interfacing to a universal asynchronous receiver/transmitter. However, with software control, communication between a McBSP and an UART is possible. The McBSP is easy to configure, and the compression/expansion software routines are straightforward for this purpose.

7 References

1. *TMS320C6201 Digital Signal Processor Data Sheet*, SPRS051.
2. *TMS320C6000 Peripherals Reference Guide*, SPRU190.
3. *TMS320C6000 Peripheral Support Library Programmer's Reference*, SPRU273.
4. *TMS320C62x/67x CPU and Instruction Set*, SPRU189.
5. Lammert Bies, RS232 general info, http://www.lammertbies.nl/comm/info/RS-232.html.
6. *SN75LV4737A 3.3-V/5-V Multichannel RS-232 Line Driver/Receiver Datasheet*, SLLS178.

Appendix A Sample C Code: Serial Port Mode

This program demonstrates McBSP and DMA initialization to implement UART functionality. Function process_transmit_data() expands a block of data, adds framing bits, and stores the data in the transmit buffer. DMA channel 0 transfers the data from the transmit buffer to the McBSP Data Transmit Register DXR. DMA Channel 1 services the McBSP by transferring received data to the receive buffer. Upon DMA Channel 1 completion, function process_receive_data() is called to compress the UART data into ASCII data. The result is stored in recvmsg[].

McBSP 0 is configured to transmit UART data in 8N1 format at 115,200 bps, assuming a 200Mhz CPU clock. This program applies the 2–phase McBSP frame format discussed in this application report.

This code also contains function run_edma() that can be used for TMS320C6x1x devices.

```
/*****************************************************************/
/*    uart.c                                                     */
/*****************************************************************/
#include <mcbsp.h>
#include <dma.h>
#include <edma.h>
#include <stdlib.h>

/* Definitions */
#define XMITBUFADDR 0x80000000
#define RECVBUFADDR 0x80004000
#define NUMTXDATA   20        /* total number of UART data words */

/* Global variables */
volatile int DMA_done[4]= {0, 0, 0, 0};
char xmitmsg[NUMTXDATA] = ”McBSP can do UART!\n”;
char recvmsg[NUMTXDATA];

/* Prototypes */
extern void set_interrupts(void);
void start_mcbsp(void);
void process_transmit_data(void);
void run_dma(void);  /* for TMS320C6x0x devices */
void run_edma(void); /* for TMS320C6x1x devices */
void set_mcbsp(void);
void process_receive_data(void);
short vote_logic(unsigned short);


/*****************************************************
void run_dma(void)
This function sets up DMA channel 0 to service the
McBSP 0 transmitter.  It sets up DMA channel 1 to
service the McBSP 0 receiver.
*****************************************************/
void
run_dma (void)
{
```

```
unsigned int   dma_acr        = 0;
unsigned int   dma_gcra       = 0;
unsigned int   dma_gcrb       = 0;
unsigned int   dma_gndxa      = 0;
unsigned int   dma_gndxb      = 0;
unsigned int   dma_gaddra     = 0;
unsigned int   dma_gaddrb     = 0;
unsigned int   dma_gaddrc     = 0;
unsigned int   dma_gaddrd     = 0;
unsigned int   dma_pri_ctrl   = 0;
unsigned int   dma_sec_ctrl   = 0;
unsigned int   dma_src_addr   = 0;
unsigned int   dma_dst_addr   = 0;
unsigned int   dma_tcnt       = 0;
unsigned int   frame_cnt     = 1;
unsigned int   element_cnt    = 0;

    /* Reset DMA Control Registers */
    dma_reset();

    /* Set up Global Configuration Registers for the DMA */
    dma_global_init(dma_acr,dma_gcra,dma_gcrb,dma_gndxa,dma_gndxb,
                    dma_gaddra,dma_gaddrb,dma_gaddrc,dma_gaddrd);

    /* Set up DMA Primary Control Register */
    LOAD_FIELD(&dma_pri_ctrl, DMA_DMA_PRI, PRI  , 1       );
    SET_BIT(&dma_pri_ctrl,TCINT);         /* Allow Ch to interrupt CPU   */
    SET_BIT(&dma_pri_ctrl,EMOD);          /* Halt DMA with emu halt      */
    LOAD_FIELD(&dma_pri_ctrl, DMA_ADDR_NO_MOD, DST_DIR   , DST_DIR_SZ    );
    LOAD_FIELD(&dma_pri_ctrl, DMA_ADDR_INC   , SRC_DIR   , SRC_DIR_SZ    );
    LOAD_FIELD(&dma_pri_ctrl, DMA_ESIZE16    , ESIZE     , ESIZE_SZ      );
    LOAD_FIELD(&dma_pri_ctrl, SEN_XEVT0      , WSYNC     , WSYNC_SZ      );
    LOAD_FIELD(&dma_pri_ctrl, SEN_NONE       , RSYNC     , RSYNC_SZ      );

    /* Set up DMA Secondary Control Register */
    LOAD_FIELD(&dma_sec_ctrl, DMAC_FRAME_COND, DMAC_EN   , DMAC_EN_SZ    );
    SET_BIT(&dma_sec_ctrl, BLOCK_IE);

    /* Set up Source and Destination Address Registers */
    dma_src_addr = (unsigned int)XMITBUFADDR;
    dma_dst_addr = (unsigned int)MCBSP_DXR_ADDR(0);

    /* Calculate DMA element count based on total number of UART words */
    /* Each UART word is equivalent to 11 DMA elements:
       Start bit :  1 element
       UART data :  8 elements
       Stop bit  :  2 elements
                    (Stop bit is transmitted in Phase 2 of the
                     McBSP frame. The DMA transfers two 16–bit
                     0xffff to the McBSP for the Stop bit, although
                     each of these two 16–bit 0xffff are shifted
                     out as 8–bit elements from the McBSP DX pin
```

```
    */
    element_cnt = NUMTXDATA * 11;

    /* Set up DMA Tranfer Count Register */
    LOAD_FIELD(&dma_tcnt, frame_cnt   , FRAME_COUNT  , FRAME_COUNT_SZ  );
    LOAD_FIELD(&dma_tcnt, element_cnt , ELEMENT_COUNT, ELEMENT_COUNT_SZ);

    /* Store DMA Control registers (channel 0--service McBSP transmitter)*/
    dma_init(0, dma_pri_ctrl, dma_sec_ctrl, dma_src_addr, dma_dst_addr,
            dma_tcnt);

    /* Modify register fields for DMA channel 1 */
    LOAD_FIELD(&dma_pri_ctrl, SEN_NONE       , WSYNC     , WSYNC_SZ     );
    LOAD_FIELD(&dma_pri_ctrl, SEN_REVT0      , RSYNC     , RSYNC_SZ     );
    LOAD_FIELD(&dma_pri_ctrl, DMA_ADDR_INC   , DST_DIR   , DST_DIR_SZ   );
    LOAD_FIELD(&dma_pri_ctrl, DMA_ADDR_NO_MOD, SRC_DIR   , SRC_DIR_SZ   );
    dma_src_addr = (unsigned int)MCBSP_DRR_ADDR(0);
    dma_dst_addr = (unsigned int)RECVBUFADDR;

    /* Store DMA Control registers (channel 1--service McBSP receiver)*/
    dma_init(1, dma_pri_ctrl, dma_sec_ctrl, dma_src_addr, dma_dst_addr,
            dma_tcnt);

    /* Start DMA */
    DMA_START(1);
    DMA_START(0);
} /* end run_dma */

/*******************************************************
void run_edma(void)

This function sets up EDMA channel 12 to service the
McBSP 0 transmitter.  It sets up EDMA channel 13 to
service the McBSP 0 receiver.
*******************************************************/
void
run_edma(void)
{
int ch = 12;            /* XEVT0 synchronized channel */
int pri = 1;            /* EDMA high priority */
int esize = DMA_ESIZE16;
int ds = 0;             /* non 2D src */
int sum = 1;            /* src addr increment (XMITBUFADDR) */
int dd = 0;             /* non 2D dst */
int dum = 0;            /* dst addr no update (DXR) */
int interrupt_enable = 1;      /* transfer complete interrupt enabled */
int report = ch;        /* CIPR[report] bit set when channel complete */
int xfer_link = 0;      /* no linking */
int fs = 0;             /* fs needs to be disabled */
int element_cnt = 0;    /* initialize element_cnt variable */
unsigned int *edmaptr;
```

```
/* Calculate EDMA element count based on total number of UART words */
/* Each UART word is equivalent to 11 EDMA elements:
   Start bit :  1 element
   UART data :  8 elements
   Stop bit  :  2 elements
                (Stop bit is transmitted in Phase 2 of the
                 McBSP frame. The EDMA transfers two 16–bit
                 0xffff to the McBSP for the Stop bit, although
                 each of these two 16–bit 0xffff are shifted
                 out as 8–bit elements from the McBSP DX pin
*/
element_cnt = NUMTXDATA * 11;

   /* Clear completion flag DMA_done */
   DMA_done[0] = 0;
   DMA_done[1] = 0;
   /* Clear all pending EDMA syncs for McBSP channels */
EDMA_EER = 0;
EDMA_ECR = 0xFFFFFFFF;      /* clear all EDMA events in ECR */
EDMA_CIER = 0;     /* clear interrupt enable */

/* Set up EDMA channel to transfer from XMITBUFADDR to DXR*/
EDMA_OPTIONS(ch)= (unsigned int)((pri << PRI)
                | (esize << ESIZE) | (ds << DS)
                | (sum << SUM)     | (dd << DD)
                | (dum << DUM)     | (interrupt_enable << TCINT)
                | (report << TCC)  | (xfer_link << LINK)
                | (fs << FS));
EDMA_SRC_ADDR(ch) = XMITBUFADDR;
EDMA_DST_ADDR(ch) = MCBSP_DXR_ADDR(0);
EDMA_COUNT(ch) = element_cnt;
EDMA_INDEX(ch) = 0;
EDMA_REL_LNK(ch) = 0;

/* set EDMA channel for Receive */
/* value modification */
ch = 13;     /* REVT0 */
sum = 0;     /* src addr no update (DRR0) */
dum = 1;     /* dst addr increment (RECVBUFADDR) */
report = ch; /* CIPR[report] bit set when channel complete */

EDMA_OPTIONS(ch)= (unsigned int)((pri << PRI)
                | (esize << ESIZE) | (ds << DS)
                | (sum << SUM)     | (dd << DD)
                | (dum << DUM)     | (interrupt_enable << TCINT)
                | (report << TCC)  | (xfer_link << LINK)
                | (fs << FS));
EDMA_SRC_ADDR(ch) = MCBSP_DRR_ADDR(0);
EDMA_DST_ADDR(ch) = RECVBUFADDR;
EDMA_COUNT(ch) = element_cnt;
EDMA_INDEX(ch) = 0;
EDMA_REL_LNK(ch) = 0;
```

```
  /* enable EDMA interrupt */

  EDMA_CIER = (1 << 13);    /* enable 13 */
  /* enable EDMA channel 12 and 13 */
  EDMA_EER |= (unsigned int)((1 << 12) | (1 << 13));

} /* end run_edma() */


void
set_mcbsp(void)
{
unsigned int spcr = 0;
unsigned int rcr  = 0;
unsigned int xcr  = 0;
unsigned int srgr = 0;
unsigned int mcr  = 0;
unsigned int rcer = 0;
unsigned int xcer = 0;
unsigned int pcr  = 0;
int fsx_period    = 0;
int extra_time    = 0;

/* Set up Pin Control Register */
LOAD_FIELD(&pcr, FSYNC_MODE_INT  , FSXM , 1);
LOAD_FIELD(&pcr, FSYNC_MODE_EXT  , FSRM , 1);
LOAD_FIELD(&pcr, CLK_MODE_INT    , CLKXM, 1);
LOAD_FIELD(&pcr, CLK_MODE_INT    , CLKRM, 1);
LOAD_FIELD(&pcr, FSYNC_POL_LOW   , FSXP , 1);
LOAD_FIELD(&pcr, FSYNC_POL_LOW   , FSRP , 1);
LOAD_FIELD(&pcr, CLKX_POL_RISING , CLKXP, 1);
LOAD_FIELD(&pcr, CLKR_POL_RISING, CLKRP, 1);

/* Set up Receive Control Register */
LOAD_FIELD(&rcr, DUAL_PHASE    , RPHASE, 1);
LOAD_FIELD(&rcr, FRAME_IGNORE    , RFIG , 1);
LOAD_FIELD(&rcr, DATA_DELAY0     , RDATDLY, RDATDLY_SZ);
LOAD_FIELD(&rcr, 8               , RFRLEN1, RFRLEN1_SZ);
LOAD_FIELD(&rcr, 1               , RFRLEN2, RFRLEN2_SZ);
LOAD_FIELD(&rcr, WORD_LENGTH_16   , RWDLEN1, RWDLEN1_SZ);
LOAD_FIELD(&rcr, WORD_LENGTH_8   , RWDLEN2, RWDLEN2_SZ);
LOAD_FIELD(&rcr, NO_COMPAND_MSB_1ST    , RCOMPAND, RCOMPAND_SZ);

/* Set up Transmit Control Register */
LOAD_FIELD(&xcr, DUAL_PHASE    , XPHASE, 1);
LOAD_FIELD(&xcr, FRAME_IGNORE    , XFIG , 1);
LOAD_FIELD(&xcr, DATA_DELAY0     , XDATDLY, XDATDLY_SZ);
LOAD_FIELD(&xcr, 8               , XFRLEN1, XFRLEN1_SZ);
LOAD_FIELD(&xcr, 1               , XFRLEN2, XFRLEN2_SZ);
LOAD_FIELD(&xcr, WORD_LENGTH_16   , XWDLEN1, XWDLEN1_SZ);
LOAD_FIELD(&xcr, WORD_LENGTH_8   , XWDLEN2, XWDLEN2_SZ);
LOAD_FIELD(&xcr, NO_COMPAND_MSB_1ST, XCOMPAND, XCOMPAND_SZ);
```

```
/* Set up Sample Rate Generator Register */
extra_time = 16;
fsx_period = 10*16 + extra_time – 1;  /* start + 8bit_data + stop = 10 */
SET_BIT(&srgr, CLKSM);               /* CLKG derived from CPU clock */
LOAD_FIELD(&srgr, FSX_FSG    , FSGM    , 1);
LOAD_FIELD(&srgr, 108        , CLKGDV  , CLKGDV_SZ);
LOAD_FIELD(&srgr, fsx_period , FPER    , FPER_SZ);

/* Store McBSP 0 registers */
mcbsp_init(0, spcr, rcr, xcr, srgr, mcr, rcer, xcer, pcr);

} /* end set_mcbsp */

void
start_mcbsp(void)
{
      /* Bring McBSP out of reset */
      MCBSP_SAMPLE_RATE_ENABLE(0);    /* Start Sample Rate Generator */
      MCBSP_ENABLE(0, MCBSP_RX);      /* Bring Receive out of reset  */
      MCBSP_ENABLE(0, MCBSP_TX);      /* Bring Transmit out of reset */
      MCBSP_FRAME_SYNC_ENABLE(0);     /* Enable Frame Sync pulse     */

} /* End start_mcbsp */

/*****************************************************
void process_transmit_data(void)

This function expands each of the 8–bit ASCII character
in the message ”McBSP can do UART!\n” into UART transmission
16–bit words and places them in the Transmit Buffer.
In addition, 16–bit Start and Stop framing words, respectively,
are inserted before and after each of the ASCII character
in the Transmit Buffer.
*****************************************************/
void
process_transmit_data(void)
{
int i;
short cnt = –1;
unsigned char xmit_char;
unsigned short *xmitbufptr;  /* transmit buffer pointer */


       /* Point to Transmit Buffer */
       xmitbufptr = (unsigned short *)XMITBUFADDR;

      * Process data bytes in xmitmsg[] and put in xmit buffer */
       for (i=0; i < NUMTXDATA; i++){

              /* Get transmit character (one byte) from xmitmsg[] */
              xmit_char = xmitmsg[i];
```

```
                /* Process each byte of transmit character  */
                for (cnt=–1; cnt < 10; cnt++){
                        if (cnt == –1){
                             /* Put Start bit in xmit buffer */
                             *xmitbufptr++ = 0x0000;
                        } else if (cnt == 8 || cnt==9) {
                             /* Put Stop bit in xmit buffer */
                             *xmitbufptr++ = 0xffff;
                        } else if (xmit_char & (1 << cnt)) {
                             /* data bit is one */
                             *xmitbufptr++ = 0xffff;
                        } else {
                             /* data bit is zero */
                             *xmitbufptr++ = 0x0000;
                        }
                } /* end for cnt */

        } /* end for i */

} /* end process_transmit_data */

/*******************************************************
void process_receive_data(void)

This function decodes the received data in the
Receive Buffer.  It strips the framing Start (0x0000)
and Stop (0xffff) words.  It calls the subroutine
vote_logic() to determine each bit of the
ASCII character.  It puts the results in global
recvmsg[].
*******************************************************/
void
process_receive_data(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 Receive Buffer */
    recvbufptr = (unsigned short *)RECVBUFADDR;

    /* Process all data in Receive Buffer */
    for (i=0; i < NUMTXDATA; 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 +=1;
                    } else {
                         /* Get 16–bit data from Receive Buffer */
                         raw_data = *recvbufptr;
                         recvbufptr += 1;
                          /* Get the value of the majority of bits */
                          recv_val = vote_logic(raw_data);
                          /* Put received bit into proper place */
                          recv_char += recv_val << cnt;
                    }
               } /* end for cnt */

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

          } /* end for i */

     } /* end process_receive_data */


     /*****************************************************
     short vote_logic(unsigned short value)

     Data is decoded by testing the center 4 bits of
     the baud. A majority rule is used for the decode.
     *****************************************************/
     short
     vote_logic(unsigned short value) {
        switch ((value >> 6) & 0xf) {
           case 0:
           case 1:
           case 2:
           case 3:
           case 4:
           case 5:
           case 6:
           case 8:
           case 9:
           case 10:
                          return 0;
           case 7:
           case 11:
           case 12:
           case 13:
           case 14:
           case 15:
                          return 1;
         }
     }  /* end vote_logic */
```

```
/*******************************************************
For TMS320C6x1x devices, make the following changes:
-  Replace run_dma() with run_edma();
*******************************************************/
void main (void)
{
      set_interrupts();
      process_transmit_data();
      run_dma();
      set_mcbsp();
      start_mcbsp();

         /* wait for end of UART transmit and receive */
         while (!DMA_done[0] || !DMA_done[1]);

         /* Process received data */
         process_receive_data();

} /* end main */

/*****************************************************************/
/*    dma_int.c                                                  */
/*****************************************************************/

#include <intr.h>
#include <mcbsp.h>
#incldue <dma.h>
#include <edma.h>

/* Definitions */

/* Global variables */
extern volatile int DMA_done[4];

/* Prototypes */
interrupt void DMA_Ch0_ISR(void);
interrupt void DMA_Ch1_ISR(void);
interrupt void EDMA_ISR(void);
void set_interrupts(void);


/* DMA Ch0 ISR used to clear block condition and flag when the     */
/* transfer has completed.                                         */
interrupt void
DMA_CH0_ISR(void)
{
unsigned int sec_ctrl = 0x50000;

      sec_ctrl = REG_READ(DMA0_SECONDARY_CTRL_ADDR);
      RESET_BIT(&sec_ctrl, FRAME_COND);
      if (GET_BIT(&sec_ctrl, BLOCK_COND);
            DMA_done[0] = 1;
      RESET_BIT(&sec_ctrl, BLOCK_COND);
      }
      else SET_BIT (&sec_ctrl, RSYNC_STAT);
      REG_WRITE(DMA0_SECONDARY_CTRL_ADDR_sec_ctrl);
} /* End DMA_CH0_ISR */
```

```
/* DMA CH1 ISR used to clear block condition and flag when the     */
/* transfer has completed.                                         */
interrupt void
DMA_Ch1_ISR(void)
{
unsigned int sec_ctrl = 0;

      sec_ctrl = REG_READ(DMA1_SECONDARY_CTRL_ADDR);
      RESET_BIT(&sec_ctrl, FRAME_COND);
      if (GET_BIT(&sec_ctrl, BLOCK COND)){
            DMA_done[1] = 1;
            RESET_BIT(&sec_ctrl, BLOCK_COND);
      }
      else SET_BIT(&sec_ctrl, RSYNC_STAT);
      REG_WRITE(DMA1_SECONDARY_CTRL_ADDR, sec_ctrl);

} /* End DMA_Ch1_ISR */

/* EDMA ISR used to clear block condition and flag when the     */
/* transfer has completed.                                         */
interrupt void
EDMA_ISR(void)
{
 if(EDMA_CIPR & 1<<15){
  DMA_done[0] = 1;                      /* Set completion flag   */
  DMA_done[1] = 1;
  EDMA_CIPR = 1<<15;                        /* clear CIP from CIPR   */
 }
 if (EDMA)CIPR & 1<<13){
  DMA_done[0] = 1;                      /* Set completion flag */
  DMA_done[1] = 1;
  EDMA_CIPR = 1<<13;                        /* clear COP from CIPR */
 }
} /* End EDMA_ISR */


/*********************************************************
   Routine to enable DMA interrupt service routines
   For TMS320C6x1x devices, make the following changes:
   1. do not set INT9 interrupt
   2. Replace DMA_CH0_ISR with EDMA_ISR
*********************************************************/
void
set_interrupts(void)
{
      intr_init();
      intr_map(CPU_INT8, ISN_DMA_INT0);
      intr_hook(DMA_Ch0_ISR, CPU_INT8);
      intr_map(CPU_INT9, ISN_DMA_INT1);
      intr_hook(DMA_Ch1_ISR, CPU_INT9);

      INTR_GLOBAL_ENABLE();
      INTR_ENABLE(CPU_INT_NMI);
      INTR_ENABLE(CPU_INT8);
      INTR_ENABLE(CPU_INT9);

} /* End set_interrupts */
```

Appendix B Sample C/Assembly Code: 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

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 acknowledgement, 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.

CERTAIN APPLICATIONS USING SEMICONDUCTOR PRODUCTS MAY INVOLVE POTENTIAL RISKS OF DEATH, PERSONAL INJURY, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE ("CRITICAL APPLICATIONS"). TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF TI PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE FULLY AT THE CUSTOMER'S RISK.

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.