An Implementation of a Software UART Using the TMS320C25

APPLICATION REPORT: SPRA077

David Zalac
Digital Signal Processor Products-
Semiconductor Group
Texas Instruments

Digital Signal Processing Solutions

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software 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 application 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, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

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

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI 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.

TRADEMARKS

TI is a trademark of Texas Instruments Incorporated.

Other brands and names are the property of their respective owners.

CONTACT INFORMATION

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

An Implementation of a Software UART Using the TMS320C25

Abstract

When transmitting to and receiving data from a processing engine such as the TMS320C25 digital signal processor, it is common to encounter problems when interfacing to asynchronous devices. This chapter provides a look at software implementation of a Universal Asynchronous Receiver and Transmitter (UART) which enables communication with asynchronous serial devices with a minimum of external hardware. Topics covered include:

- Characteristics of asynchronous communications
- Limitations of and reasons where use of a UART are justifiable
- An overview of the functions provided by a UART
- UART Implementation
- A list of References
- An appendix of source code

Supporting figures illustrate the following:

- UART word format and architecture
- Timer interrupt service routine
- Transmitter routine
- UART status word
- Receiver routine
- RS-232 interface

Product Support

World Wide Web

Our World Wide Web site at www.ti.com contains the most up to date product information, revisions, and additions. New users must register with TI&ME before they can access the data sheet archive. TI&ME allows users to build custom information pages and receive new product updates automatically via email.

Email

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

Introduction

Interfacing to asynchronous devices is a common problem in transmitting to and receiving data from a processing engine such as the TMS320C25 digital signal processor. This report describes a software implementation of a Universal Asynchronous Receiver & Transmitter (UART) that provides the ability to communicate with asynchronous serial devices in a system with a minimum of external hardware.

Asynchronous communications are characterized by the absence of a timing reference such as a clock or framing signal. Various tradeoffs arise from this distinction from synchronous communications in terms of hardware and software requirements and data throughput capacity. Synchronous communications require a timing reference, but otherwise have minimal hardware and software requirements. Asynchronous communications require a mechanism for deriving a timing reference from the received signal. Additionally, various error-checking functions are typically implemented. These requirements impose hardware and/or software overhead that is not imposed in the synchronous case. Moreover, synchronous interfaces can typically support much higher data throughput rates than asynchronous interfaces.

Implementing a UART in software imposes CPU overhead whose acceptability is application-dependent. In applications where the overall data throughput rate is sufficiently low, or in cases in which a UART is to be used only for booting system memory at powerup, use of a software UART may be justifiable. A hardware solution (i.e., a UART IC) may be more appropriate in high data rate applications and in applications requiring low I/O overhead. A detailed analysis of overhead imposed by the TMS320C25 software UART is given later in this report.

A high-speed synchronous serial interface is provided by the on-chip serial port of the TMS320C25. A full description and specification of the serial port may be found in the *Second-Generation TMS320 User's Guide*. [4]

Overview

The functions provided by a UART are simply the transmission and reception of serial data and the checking and signalling of various error conditions. These functions are described in detail in the following sections.

Data Format

Shown in Figure 1 is the layout of a word in a format assumed by the UART. Bit 0 is a space (logic low) and is referred to as the start bit. Bits 1 through N are the N data bits of the word with the LSB occupying bit position 1. Typically, N has a value of 5, 6, 7, or 8. The maximum value of N is given by Nmax = 14−M, where M is the

number of stop bits. Bit N+1 is referred to as the parity bit and has a value such that the total number of ones in the word (bits 1 through N+1) is odd if odd parity is selected and even if even parity is selected. Bits N+2 through N+M+1 are referred to as stop bits, and each has a value of one. The total word length WORD_LEN is thus given by WORD_LEN = N+M+2.

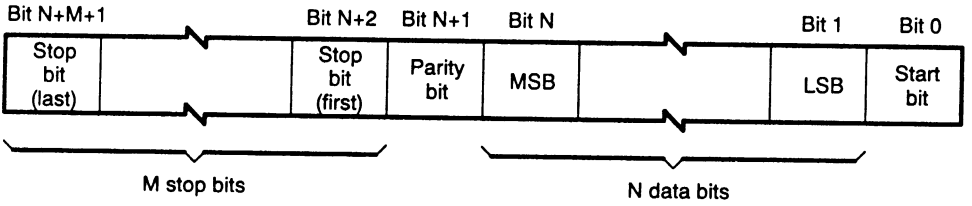

Figure 1. UART Word Format

Data Reception

Reception of a data word starts with detection of the start bit. One way of performing start bit detection is to sample the input data signal at a rate that is large compared to the bit rate, then testing each sample for a space (logic low). An optional check can be performed to verify that the first logic low detected represents a valid start bit and not just noise. This check is performed by testing that the input signal is low one-half-bit duration after the start bit has been detected.

Once the start bit has been detected, the UART simply recovers the data from the input signal and keeps track of the data parity. The parity is checked against the received parity bit after all the data bits have been received. Finally, the integrity of the word framing is checked by testing that the input signal is high when the first stop bit is expected.

Data Transmission

Transmission is considerably simpler than reception in that timing information does not have to be recovered from an asynchronous signal. Furthermore, no error checking is performed by the UART transmitter. Transmitting a data word is preceded by appropriately formatting the data to be transmitted; i.e., adding start, stop and parity bits. Formatting is done in TMS320C25 software. The output signal is generated from the data and appears on the UART's output signal line.

Implementation

The UART implementation described in this report makes use of two TMS320C25 general-purpose I/O pins (XF and BIO/) and the timer interrupt. The input signal is received on the BIO pin via the TMS320C25 BIOZ instruction. The output signal appears on the general-purpose flag pin XF. The state of XF is controlled in software via the SXF and RXF instructions. The TMS320C25 serial port is not used. As shown if Figure 2,

the transmitter and receiver are "serviced" each time a timer interrupt is generated. The timer interrupt rate is an integer multiple (K) of the bit rate.

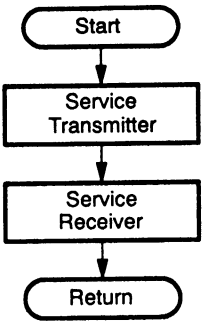

Figure 2. Timer Interrupt Service Routine

Several pieces of code comprise the UART software:

1. Timer interrupt service routine
2. UART_INIT initialization routine
3. XMT routine
4. RCV routine
5. PUT_DATA
6. GET_DATA
7. XCOMPOSE

The UART transmitter and receiver are located in the timer interrupt service routine. (No context save/restore is included in this interrupt service routine. Refer to the "Precautions" section for details.) UART_INIT initializes the UART with the values appearing in the assembly-time constants section of the source listing. XMT and RCV are user-written routines that interface the UART to the user's program. XCOMPOSE, PUT_DATA and GET_DATA are auxiliary routines available to the user for executing UART interface housekeeping tasks. Each of these seven routines is described in detail later in this report.

Figure 3 shows the UART structure and the transmit and receive data paths. The transmit and receive buffers TDATA and RDATA are 16-bit wide TMS320C25 on-chip data memory locations. The transmit and receive software shift registers TSHF and RSHF are also located in data memory. The UART status word USTAT maintains UART status and error information. USTAT will be discussed in detail later in this report. The value of USTAT is written to I/O port UARTPORT each time USTAT is updated, thus allowing the capability of externally monitoring the UART status. The input and output pins ($\overline{BIO}$ and XF) may be interfaced to RS-232-compatible transmit and receive lines. Finally, the locations in data memory for transmit and receive data are pointed at by two TMS320C25 auxiliary registers AR(OPT_PTR) and AR(INP_PTR), respectively.

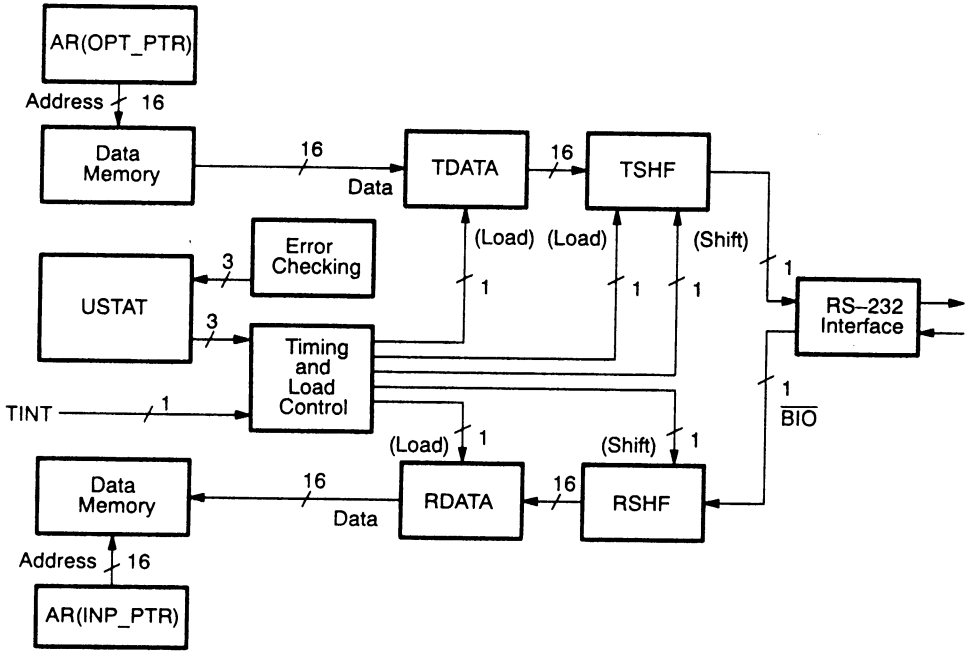

Figure 3. UART Architecture

All UART variables are mapped into TMS320C25 on-chip RAM block B2. The code size is 332 words and can be executed from on-chip ROM, EPROM (TMS320E25), or off-chip program memory. The maximum bit rate supported is 19.2 kilobits per second (full duplex).

Transmitter

Figure 4 shows a flowchart of the transmitter routine. When the transmitter completes transmission of the current word, a new word (TDATA) is loaded into the transmitter software shift register if the TDA (Transmit Data Available) flag is set.

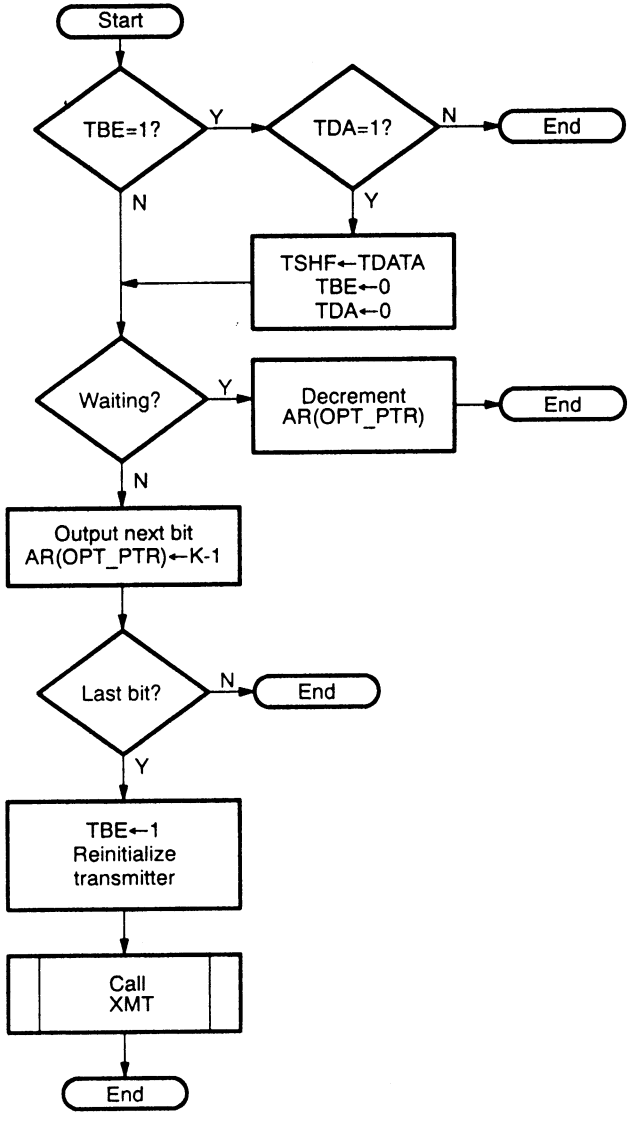

Figure 4. Transmitter Routine

TDA is one of 6 flags residing in the UART status word (see Figure 5). The word to be transmitted is shifted out on the XF pin at the user-specified bit rate.

Bit 7	Bit 6	Bit 5	Bit 4	Bit3	Bit 2	Bit 1	Bit 0
Not used	Not used	TDA	RDE	RDA	ROR	FRM	RPE

Figure 5. UART Status Word

Feeding data to the transmitter consists of three steps: appropriately reformatting the data to be transmitted (i.e., adding start, parity, and stop bits), loading the data into the UART variable TDATA, and indicating to the UART that valid transmit data is present in TDATA by setting the TDA flag. The first step can be accomplished by pointing AR(OPT__PTR) at the data to be transmitted and calling auxiliary routine XCOMPOSE. XCOMPOSE does an in-place reformatting of the data per the values in the UART parameter variables (see source listing). Routine PUT__DATA may be called to load TDATA and set TDA.

Each time the transmit shift register empties, a call to XMT is made. Management of AR(OPT__PTR) and calls to XCOMPOSE and PUT__DATA may be made from XMT. Alternatively, these functions can be executed from the user's program. This is the preferred approach, because all code in XMT adds to the maximum path length through the timer interrupt routine and thus decreases the maximum bit rate. However, calls to PUT__DATA from the user's program should be made only if TDA = 0. If this condition is not satisfied, the current word to be transmitted will be overwritten. (The condition TDA = 0 is guaranteed if PUT__DATA calls are made from XMT and needn't be checked.)

Initiation of transmission of the first word in a string of words (string = one or more words) must be made from the user's program by calling XCOMPOSE and PUT__DATA, as no XMT calls can be made until the transmitter is started.

The status of the transmitter can be ascertained by reading bit 5 of the UART status word, as shown in Figure 5:

USTAT

BIT 5 Transmit Data Available TDA = 1 indicates to the UART that valid transmit data is present in TDATA.

Receiver

Shown in Figure 6 is a flowchart of the receiver routine. The state of the receiver is indicated by the value of RSTAT and bits 0 - 4 of the UART status word as shown below.

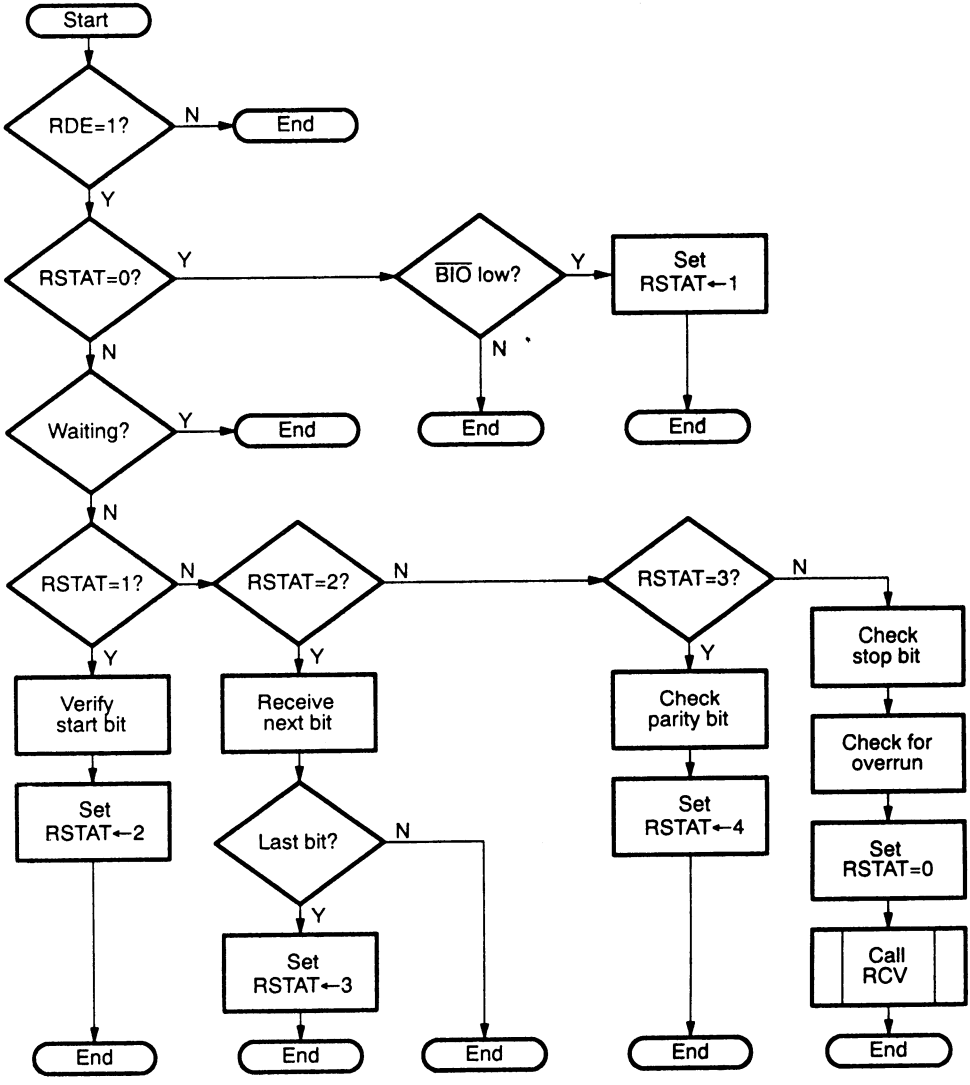

Figure 6. Receiver Routine

RSTAT	Receiver Status
0	Waiting for start bit
1	Waiting for start bit center
2	Waiting for data bit
3	Waiting for parity bit
4	Waiting for stop bit

USTAT

BIT 0	Receive Parity Error	If receive parity checking is active (RPACTIVE = 1), RPE is set if a parity error is detected.
BIT 1	Framing Error	This bit is set if a logic low is sensed on $\overline{BIO}$ when the first stop bit is expected.
BIT 2	Receiver Overrun	This bit is set if RDA is not cleared before reception of the next word is completed.
BIT 3	Receive Data Available	This bit is set when reception of a word is completed and indicates the presence of valid data in RDATA.
BIT 4	Receive Data Enable	The receiver routine is bypassed if this bit is a zero.

Reception is initiated by setting RDE. Each time a complete word is received, a call to RCV is made. RCV can call GET__DATA to copy the new data to the location pointed at by AR(INP__PTR) and to clear RDA. If RDA is indeed cleared by every call to RCV, some overhead can be eliminated by deleting the setting, clearing, and checking of RDA because the overrun detect function is superfluous in this case.

The received data in RDATA is right-justified with the LSB in the zero'th bit position and with start, stop, and parity bits stripped.

Overhead and Optimization

The overhead imposed by the UART is primarily determined by the length of the timer interrupt service routine and the timer interrupt rate (K * bit rate). An expression for overhead is given below:

$$\text{overhead } (\%) = K \times (\text{bit rate}) \times (T + R) \times T_{c(C)} \times 100\%$$

K = (timer interrupt rate)/(bit rate). $T_{c(C)}$ is the period of CLKOUT1 and CLKOUT2. In the timer interrupt service routine shown in the source listing, T and R are given by

$$T = (8 \times \text{WORD\_\_LEN} \times K + 14 \times \text{WORD\_\_LEN} + 15)/(\text{WORD\_\_LEN} \times K)$$

$$R = (17 \times \text{WORD\_\_LEN} \times K + 36 \times \text{WORD\_\_LEN} + 169)/(\text{WORD\_\_LEN} \times K)$$

The values of T and R at WORD__LEN = 10 and K = 16 are T = 9 cycles per timer interrupt and R = 20 cycles per timer interrupt. (These are the correct values when the timer interrupt service routine is in zero-waitstate external program memory. T and R have smaller values if the service routine is in internal program memory.)

T and R represent the average path lengths (in processor cycles) through the transmitter and receiver routines, respectively, in continuous full-duplex operation. Values of overhead for several bit rates and values of K are tabulated in Table 1. Overhead associated with XMT, RCV, GET__DATA, PUT__DATA, and XCOMPOSE code is not included. Note that continuous full-duplex operation constitutes worst-case scenario, a scenario unlikely in applications using asynchronous I/O.

Table 1. UART Overhead (%) vs. Bit Rate (r in bps) and K for WORD__LEN=10†

K/r	300	1200	2400	4800	9600	19,200
6	1	3	5	10	21	42
8	1	3	6	13	26	51
10	1	4	8	15	31	–
12	1	4	9	18	35	–
14	1	5	10	20	40	–
16	1	6	11	22	45	–

†Overhead imposed by the UART is relatively insensitive to WORD__LEN.

There are several ways the user can modify the UART code to reduce the values of T and/or R. Some of these involve eliminating the setting, clearing, and checking of flags in the USTAT register that are not necessary in a fixed configuration. Others involve streamlining of the interface between the UART software and the user's program.

UART Configuration

The initial values of the UART parameters (e.g., bit rate and parity type) appear in the assembly-time constants section of the source listing given in the appendix of this report. Note that an initial-value constant exists for each UART parameter and has the same name as its corresponding parameter, but with an "I" prefix. If the UART is to be run in a fixed configuration, the user needs only to modify the initial value parameters and re-assemble and link the program. However, if the configuration is to be modified "on the fly", the following measures need to be taken:

To Respecify	Modify
RPACTIVE	RPACTIVE (0 or 1)
K	KM1, K2M1, and TINTPER (KM1 = K-1; K2M1 = K/2-1)
N	N and WORDLEN, CALL PARINIT
WORD__LEN	WORD__LEN
ODD	ODD, CALL PARINIT (initialize parity templates)
TINTPER	TINTPER
# of stop bits	WORD__LEN (# of stop bits = WORD__LEN-N-2)

Precautions

No context save/restore is provided in the timer interrupt service routine as the user will want to write and optimize this part of the routine for his own application. The timer interrupt routine affects the following registers and memory locations:

CPU Registers	Memory Locations
Accumulator	60h-77h (RAM block B2)
T register	dma<AR(INP__PTR)>
P register	TIM register
Auxiliary registers 1-7	PRD register
Status regs ST0 & ST1	

The timer interrupt routine uses two levels of stack plus as many levels as are required to accommodate subroutine calls from XMT and RCV.

If the PRD register contains a value less than 64 (19.2 kbps @ K = 8 or 9.6 kbps @ K = 16), the sampling of some receive bits may be significantly delayed from bit interval centers and some transmit signal edges may be delayed.

The actual transmit and receiver-sampling bit rate r is given by

$$r = 1/[P \times K \times T_{c(C)}],$$

where P is the sum of the contents of the PRD register and one. If no integer value of P exists for a specified r, K, and $T_{c(C)}$, the receiver should typically be allowed to run at the rate closest to but greater than the ideal bit rate.

If the receiver bit rate is exactly equal to the transmit bit rate of the external transmitting equipment, the sampling of incoming bits will occur at times close to the centers of the corresponding bit intervals. Some error is introduced by the latency between the falling edge of the start bit and the time at which the start bit is detected. The maximum value of that error (e_1) is equal to one period of the timer interrupt.

Additional error is introduced if the receiver bit rate differs from the bit rate of the incoming data stream. Let the bit duration dictated by the timer interrupt rate be denoted by T_1 and let the bit duration of the incoming data be denoted by T_2. The error introduced by the inequality of T_1 and T_2 (e_2) for the n'th bit is given by

$$e_{2(n)} = (T_1 - T_2) \times (n - 1/2) \quad (1)$$

The start bit corresponds to n = 1. The cumulative error for one word is equal to e_2 evaluated at n=WORD_LEN.

Still another source of error is the latency associated with multicycle instructions. Should a timer interrupt occur during execution of a multicycle instruction or repeat loop, an error e_3 will delay the sampling of $\overline{BIO}$ by a minimum of zero and a maximum of I−1 cycles, where I is the length (in cycles) of the longest instruction or repeat loop.

The total difference between the sampling time and a corresponding bit interval center is the sum of e_1, e_2, and e_3. In general, the absolute value of the sum of e_1, e_2, and e_3 must be less than one-half the duration of one bit in the incoming data stream in order that all sampling instants fall in corresponding bit intervals; i.e.,

$$|e_1 + e_2 + e_3| < T_2/2 \quad (2)$$

The above constraint is appropriate for a receive signal having negligible rise and fall times and equal space and mark durations. If either of these conditions is not satisfied, the constraint expression should be modified accordingly.

Worst-Case Error Analysis

Following are descriptions of the two worst-case scenarios in terms of the three error components. The results of this analysis are then plugged in the constraint expression given in (2) to yield a description of the error constraint in terms of rate difference, K and I.

If the incoming data rate is higher than the receiver bit rate, e_1, e_2, and e_3 are all greater than or equal to zero. The worst-case value of e_1 is its maximum value given by

$$e_{1(max)} = t_{c(C)} \times [< \text{PRD reg} > + 1]$$
$$= T_1/K$$

The e_2 contribution is the cumulative error resulting from the inequality of T_1 and T_2 and is given by

$$e_{2(max)} = (T_1 - T_2) \times (\text{WORD\_LEN} - 1/2)$$

The worst-case value of e_3 is given by

$$e_{3(max)} = (I_{max} - 1) \times t_{c(C)}$$

If the incoming data rate is lower than the receiver bit rate, e_1, e_2, and e_3 are all less than or equal to zero. The worst-case value of e_1 is its minimum value given by

$$e_{1(min)} = 0$$

The e_2 contribution is the cumulative error resulting from the inequality of T_1 and T_2 and is given by

$$e_{2(min)} = (T_1 - T_2) \times (WORD\_LEN - 1/2)$$

The worst-case value of e_3 is given by

$$e_{3(min)} = 0$$

The error constraint (2) is thus satisfied if the following pair of inequalities is satisfied:

$$e_{1(min)} + e_{2(min)} + e_{3(min)} > -T_2/2 \qquad (3)$$

$$e_{1(max)} + e_{2(max)} + e_{3(max)} < T_2/2$$

where expressions for the extreme values of each error component are given above.

The inequalities in (3) specify the overall constraint on maximum rate difference, minimum value of K and maximum value of I. For example, suppose

T_2 = 0.100 ms
T_1 = 0.103 ms
I = 20
$t_{c(C)}$ = 100 ns
WORD_LEN = 10

Since T_1 is sufficiently close to T_2, the first inequality in (3) is satisfied:

$$0 + [(103 \times 10-6) - (100 \times 10^{-6})] \times (10-0.5) + 0 > (-100 \times 10^{-6})/2$$

Evaluation of the second inequality in (3) yields

$$(103 \times 10^{-6})/K + [(103 \times 10^{-6}) - (100 \times 10^{-6})] \times (10 - 0.5) + (20 - 1)$$

$$\times (100 \times 10^{-9}) < (100 \times 10^{-6})/2$$

or

$$K \geq 6$$

Thus (2) translates into a specification for the minimum value of K for a given T_1, T_2, I, $t_{c(C)}$, and WORD_LEN.

In summary, considerations must be made with respect to the data rate of the external transmitting equipment, the data rate resulting from the timer interrupt rate, and the latencies associated with start bit detection and multicycle instructions. The two inequalities in (2) must be satisfied for all bits for proper UART operation.

Loopback Test

In the source code given, the XMT and RCV routines are structured to implement a loopback test at 9600 bps, 7 data bits, 1 stop bit and odd parity. The circuit shown in Figure 7 can be used to interface to RS-232-compatible transmit and receive lines. No other RS-232 signals are supported.

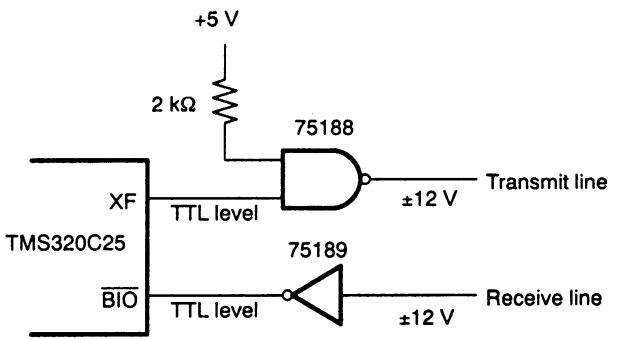

Figure 7. RS-232 Interface

References

[1] *TMS7000 Family Data Manual*, literature number SPND001C, Texas Instruments, 1989.

[2] McNamara, John, *Technical Aspects of Data Communications*, Digital Equipment Corporation, 1982.

[3] *Data Communications Standards*, McGraw-Hill, 1982.

[4] *Second-Generation TMS320 User's Guide*, literature number SPRU014A, Texas Instruments, 1989.

```
*****************************************************************
*
*   SOFTWARE UART USING THE TMS320C25
*
*   WRITTEN BY: DAVE ZALAC
*               TEXAS INSTRUMENTS, INC.
*               1/31/89
*
*   PROGRAM MEMORY REQUIREMENT: 332   WORDS
*   DATA MEMORY REQUIREMENT:    23    WORDS
*   MAXIMUM BIT RATE:           19.2 KBPS
*
*****************************************************************
*
*   ASSEMBLY-TIME CONSTANTS SECTION
*
*****************************************************************
*
*   STATUS BIT MASKS
*
*****************************************************************
*
RPE_MSK   .set    001h          ; RECEIVE PARITY ERROR
*
RPE_SET   .set    001h
RPE_CLR   .set    0FEh
*
FRMERR_MSK  .set  002h          ; RECEIVE FRAMING ERROR
FRMERR_SET  .set  002h
FRMERR_CLR  .set  0FDh
*
ROR_MSK   .set    004h          ; RECEIVE DATA OVERRUN
ROR_SET   .set    004h
ROR_CLR   .set    0FBh
*
RDA_MSK   .set    008h          ; RECEIVE DATA AVAILABLE
RDA_SET   .set    008h
RDA_CLR   .set    0F7h
*
RDE_MSK   .set    010h          ; RECEIVER ENABLE
RDE_SET   .set    010h
RDE_CLR   .set    0EFh
*
TDA_MSK   .set    020h          ; TRANSMIT DATA AVAILABLE
TDA_SET   .set    020h
TDA_CLR   .set    0DFh
*
*****************************************************************
*
*   USTAT REGISTER
*
*   BIT #          5     4     3     2     1     0
*   FLAG           TDA   RDE   RDA   ROR   FRM   RPE
*   INITIAL VALUE  0     0     0     0     0     0
*
*****************************************************************
*
UARTPORT  .set    0             ; PORT ADDRESS OF UART STATUS REGISTER
*
*****************************************************************
*
*   INITIAL VALUES OF UART PARAMETERS
*
*****************************************************************
*
IRPACTIVE .set    1             ; RPACTIVE = 1 INDICATES RCV PARITY
*                               ;   CHECKING ON
IKM1      .set    15            ; TINT RATE = K*BIT RATE; KM1 = K-1
IK2M1     .set    7             ; K2M1 = K/2-1
INM1      .set    6             ; NUMBER OF DATA BITS/WORD
IWORD_LEN .set    10            ; TOTAL # OF BITS/WORD
IODD      .set    1             ; ODD = 1 SELECTS ODD PARITY
*                               ; ODD = 0 SELECTS EVEN PARITY
ITINTPER  .set    259           ; TINTPER = TIMER INTERRUPT PERIOD IN
*                               ; CLKOUT1 CYCLES. TINTPER SHOULD BE SET TO
*                               ; (1/(K*R*TC(C)))-1, ROUNDED ROUNDED DOWN
*                               ; TO THE NEAREST INTEGER. R IS THE BIT RATE
*                               ; IN BPS.
*
*****************************************************************
*
*   REGISTER ASSIGNMENTS
*
*****************************************************************
*
TWG       .set    7             ; DIVIDES TINT RATE BY K (XMT)
RWG       .set    6             ; DIVIDES TINT RATE BY K (RCV)
XBITS_REG .set    5             ; USED BY TRANSMITTER TO COUNT DATA BITS
RBITS     .set    4             ; USED BY RECEIVER TO COUNT DATA BITS
XBREG     .set    3             ; USED BY XCOMPOSE TO COUNT DATA BITS
INP_PTR   .set    2             ; POINTS AT LOCATION TO PUT RECEIVED DATA
OPT_PTR   .set    1             ; POINTS AT DATA TO BE TRANSMITTED
*
*****************************************************************
*
*   END ASSEMBLY-TIME CONSTANTS SECTION
*
*****************************************************************
*
*
*   DATA MEMORY SPACE RESERVATION
*
*****************************************************************
*
*   VARIABLES
*
*****************************************************************
*
```

```
        .bss    RSTAT,1         ; RECEIVER STATUS
        .bss    USTAT,1         ; UART STATUS REGISTER
        .bss    SHF,1           ; SHIFT FACTOR
        .bss    INP,1           ; RECEIVED BIT
        .bss    DWORD,1         ; RECEIVER SHIFT REGISTER
        .bss    RPAR,1          ; LOCALLY-GENERATED PARITY
        .bss    RPARI,1         ; INITIAL VALUE OF RPAR
        .bss    XSTOP,1         ; STOP BIT STRING
        .bss    XPARI,1         ; INITIAL VALUE OF XSTOP
        .bss    XPAR,1          ; TRANSMIT APRITY
        .bss    RDATA,1         ; RECEIVE BUFFER
        .bss    XPARTOG,1       ; TRANSMIT PARITY TOGGLE MASK
        .bss    TDATA,1         ; TRANSMIT DATA
        .bss    TSHF,1          ; TRANSMIT SHIFT REGISTER
        .bss    TEMP,1          ; SCRATCH VARIABLE
        .bss    RPACTIVE,1      ; RECEIVE PARITY ACTIVE (0/1)
        .bss    KM1,1           ; K-1
        .bss    K2M1,1          ; K/2-1
        .bss    NM1,1           ; N-1
        .bss    WORD_LEN,1      ; OVERALL WORD LENGTH
        .bss    ODD,1           ; SELECTS ODD/EVEN PARITY (1/0)
        .bss    TINTPER,1       ; TIMER INTERRUPT PERIOD
        .bss    TBE,1           ; TRANSMIT BUFFER EMPTY
*
*****************************************************************
*
*   PROGRAM SECTION
*
*****************************************************************
*
        .sect   "vectors"
*
RS      B       MAIN
*
        .space  ((RS+24-$)*16) ; POSITION TINT VECTOR
*
TIMINT  B       XMIT           ; BRANCH TO XMT/RCV ROUTINE
*
        .text
*
MAIN    LDPK    0              ; INITIALIZE
        ZAC
        SACL    5              ; NO GLOBSL MEMORY
        SOVM                   ; SET OVERFLOW MODE
        SSXM                   ; SET SIGN-EXT. MODE
        SPM     0              ; P-REG SHIFT = 0 BITS
        CNFD                   ; CONFIGURE BLOCK B0 AS DATA MEMORY
        CALL    UART_INIT      ; INITIALIZE UART
*
        LACK    10h
        SACL    USTAT          ; ENABLE RECEIVER
*
*****************************************************************
*
*   THE USER'S PROGRAM SHOULD APPEAR HERE.
*
SELF    B       SELF
*
*   END PROGRAM SECTION
*
*****************************************************************
*
*   UART INITIALIZATION ROUTINE
*
*   THE FOLLOWING CODE INITIALIZES THE UART PER THE VALUES IN THE ASSEMBLY-
*   TIME CONSTANTS SECTION ABOVE. ROUTINE PARINIT IS A SUBSET OF UART_INIT
*   AND MAY BE CALLED INDEPENDENTLY.
*
*****************************************************************
*
UART_INIT  LDPK    0
*
        LALK    IRPACTIVE      ; INITIALIZE UART PARAMETER VARIABLES
        SACL    RPACTIVE
        LALK    IKM1
        SACL    KM1
        LALK    IK2M1
        SACL    K2M1
        LALK    INM1
        SACL    NM1
        LALK    IWORD_LEN
        SACL    WORD_LEN
        LACK    IODD
        SACL    ODD
        LALK    ITINTPER
        SACL    TINTPER
        LACK    1
*       SACL    TBE
        ZALS    TINTPER        ; INITIALIZE TIMER PERIOD
        SACL    2
        SACL    3
        LALK    OFFC8h         ; ENABLE TINT ONLY
        SACL    4
        ZAC
        SACL    RSTAT          ; RSTAT: = 0
        SACL    USTAT          ; USTAT: = 0
        OUT     USTAT,UARTPORT
        LAR     TWG, KM1       ; XMT WAIT: = 1/R
        LAR     XBITS_REG, WORD_LEN  ; INITIALIZE XBITS_REG
        ZAC
        SACL    DWORD          ; DWORD: = 0
        LRLK    INP_PTR, 0200h ; INITIALIZE DATA POINTERS
        LRLK    OPT_PTR, 0200h
*
*****************************************************************
*
*   CONFIGURE PARITY-RELATED CONSTANTS
*
*****************************************************************
```

```
*
PARINIT  ZALS    ODD
         SACL    RPAR
         SACL    XPARI
         LACK    1
         SACL    XPARTOG         ; INITIALIZE TOGGLE MASK TO BIT 0
         LALK    0FFFFh
         SACL    XSTOP           ; INITIALIZE STOP MASK TO BIT 0
         LARP    0
         LAR     0,NM1
         MAR     *+
SHIFX    LAC     XPARI,1         ; SHIFT PARITY BIT LOCATION BY N + 1 BITS
         SACL    XPARI
         LAC     XPARTOG,1       ; SHIFT TOGGLE MASK BY N + 1 BITS
         SACL    XPARTOG
         LAC     XSTOP,1         ; SHIFT STOP BIT STRING BY N + 1 BITS
         SACL    XSTOP
         BANZ    SHIFX
*
         LAC     XSTOP,1         ; TOTAL # OF SHIFTS FOR XSTOP = N + 2
         SACL    XSTOP
*
         EINT
         RET                     ; END UART INITIALIZATION
*
*****************************************************************
*
*   END UART_INIT
*
*****************************************************************
*
*
*   TRANSMITTER
*
*****************************************************************
*
XMIT     ZALS    TBE
         BZ      NOT_EMPTY       ; IF NOT ZERO THEN EMPTY
 *
EMPTY    ZALS    USTAT
         ANDK    TDA_MSK
         BZ      RCV             ; IF TDA = 0, THEN SKIP TO RECEIVER
*
         ZALS    TDATA           ; TSHF: = TDATA
         SACL    TSHF
*
         ZALS    USTAT           ; CLEAR TDA AND TBE
         ANDK    TDA_CLR
         SACL    USTAT
         OUT     USTAT,UARTPORT
         ZAC
         SACL    TBE
*
NOT_EMPTY LARP   TWG             ; IF WAITING, SKIP TO RECEIVER
         BANZ    RCV
*
         ZALS    TSHF            ; OUTPUT NEXT BIT
         ROR
         BC      XONE
         NOP                     ; TIME COMPENSATION
         RXF
         B       XZERO
XONE     SXF
XZERO    SACL    TSHF
         LAR     TWG, KM1        ; WAIT 1 BIT INTERVAL BEFORE OUTPUTTING
*                                ;   NEXT BIT
*
         LARP    XBITS_REG       ; IF LAST BIT, SET TBE
         BANZ    RCV
*
         LACK    1               ; SET TBE
         SACL    TBE
*
         LAR     XBITS_REG, WORD_LEN
*
         CALL    XMT             ; SIGNAL END-OF-WORD
*
*****************************************************************
*
*   END TRANSMITTER
*
*****************************************************************
*
*
*   RECEIVER
*
*****************************************************************
*
RCV      ZALS    USTAT           ; SKIP RECEIVE ROUTINE IF RDE = 0
         ANDK    RDE_MSK
         BNZ     CONT
         B       RETURN
*
CONT     ZALS    RSTAT           ; RSTAT = 0 IMPLIES WAITING FOR START BIT
         BNZ     NOTSTART
*
*****************************************************************
*
*   RSTAT = 0
*
*****************************************************************
*
         BIOZ    STARTBIT        ; LOOK FOR START BIT ON BIO/
         B       RETURN
*
STARTBIT LACK    1               ; UPDATE RSTAT
         SACL    RSTAT
         LAR     RWG, K2M1       ; WAIT 1/2 BAUD INTERVAL AFTER START BIT
```

```
        B       RETURN          ;  DETECTION
*
*****************************************************************
*
*    RSTAT = 0
*
*****************************************************************
*
NOTSTART  LARP  RWG             ; IF WAITING, RETURN
        BANZ    RETURN
*
        ZALS    RSTAT           ; RSTAT = 1 INDICATES START BIT
        XORK    1               ;   VERIFICATION PENDING
        BNZ     NOTVER
*
*****************************************************************
*
*   RSTAT = 1
*
*****************************************************************
*
        BIOZ    VALID_START     ; IF INVALID START BIT, START OVER
        ZAC
        SACL    RSTAT
        B       RETURN
*
VALID_START  LACK  1            ; INITIALIZE BIT POSITION INDICATOR
        SACL    SHF
*
        LACK    2               ; UPDATE RSTAT
        SACL    RSTAT
*
        LAR     RBITS, AR1      ; N BITS/CHARACTER
        LAR     RWG, KM1        ; WAIT 1 BAUD INTERVAL BEFORE SAMPLING DATA
*
        B       RETURN
*
NOTVER  ZALS    RSTAT
        XORK    2
        BNZ     NOTDATA
*
*****************************************************************
*
*   RSTAT = 2
*
*****************************************************************
*
        ZAC                     ; PRESET DATA TO ZERO
        SACL    INP
        BIOZ    DAT_ZERO
*
        LACK    1               ; SET DATA TO 1 IF MARK
        SACL    INP
*
DAT_ZERO  LT    SHF             ; SHIFT INPUT BIT TO APPROPRIATE POSITION
        MPY     INP             ;   AND APPEND TO INPUT STRING
        PAC
        OR      DWORD
        SACL    DWORD
*
        ZALS    INP
        BZ      NOCHANGE        ; NO PARITY CHANGE IF SPACE
*
        ZALS    RPAR            ; TOGGLE PARITY-GEN BIT IF MARK
        XORK    1
        SACL    RPAR
*
NOCHANGE  LAC   SHF,1           ; UPDATE SHIFT FACTOR
        SACL    SHF
*
        LAR     RWG, KM1        ; WAIT 1 BAUD INTERVAL BEFORE SAMPLING NEXT
*                               ;   BIT
*
        LARP    RBITS           ; END-OF-WORD DETECT
        BANZ    SKP3
*
        LACK    3               ; UPDATE RSTAT WHEN FULL WORD IS RECEIVED
        SACL    RSTAT
*
SKP3    B       RETURN
*
NOTDATA  ZALS   RSTAT           ; RSTAT = 3 IMPLIES WAITING FOR PARITY BIT
        XORK    3
        BNZ     NOTPAR
*
*****************************************************************
*
*   RSTAT = 3
*
*****************************************************************
*
        ZAC                     ; PRESET PARITY TO ZERO
        SACL    INP
*
        BIOZ    PZERO           ; SAMPLE PARITY BIT
        LACK    1
        SACL    INP
*
PZERO   ZALS    INP             ; CHECK AGAINST LOCALLY-GENREATED PARITY
        XOR     RPAR
*
        AND     RPACTIVE        ; IGNORE IF RECEIVE PARITY-CHECKING IS
        BZ      RCVPAR_OK       ;   DE-ACTIVATED
*
        ZALS    USTAT           ; REPORT PARITY ERROR
        ORK     RPE_SET
        SACL    USTAT
        OUT     USTAT,UARTPORT
```

```
*
RCVPAR_OK LAR     RWG,KM1       ; WAIT 1 BAUD INTERVAL
*
        LACK    4
        SACL    RSTAT
*
        B       RETURN
*
*****************************************************************
*
*   RSTAT = 4
*
*****************************************************************
*
NOTPAR  BIOZ    STOPO         ; INVALID STOP BIT
*
        B       STOP_OK
*
STOPO   ZALS    USTAT         ; REPORT FRAMING ERROR
        ORK     FRMERR_SET
        SACL    USTAT
        OUT     USTAT,UARTPORT
*
STOP_OK ZALS    USTAT         ; OVERRUN DETECT
        ANDK    RDA_MSK
        BZ      NO_OVERRUN
*
OVERRUN ZALS    USTAT         ; REPORT OVERRUN
        ORK     ROR_SET
        SACL    USTAT
        OUT     USTAT,UARTPORT
*
NO_OVERRUN ZALS DWORD         ; COPY DATA TO RDATA
        SACL    RDATA
*
        ZALS    USTAT         ; SET RDA
        ORK     RDA_SET
        SACL    USTAT
        OUT     USTAT,UARTPORT
*
        ZAC
        SACL    RSTAT
        SACL    DWORD
*
        ZALS    RPARI
        SACL    RPAR
*
        CALL    RECV
*
RETURN  EINT
        RET
*
*****************************************************************
*
*   END RECEIVER
*
*****************************************************************
*
*   TRANSMITTER DATA FEED ROUTINE
*
*   THE FOLLOWING XMT/RCV ROUTINES IMPLEMENT A LOOPBACK TEST.
*
*****************************************************************
*
XMT     RET
*
*****************************************************************
*
*   END XMT
*
*****************************************************************
*
*   RECEIVER DATA RECOVERY ROUTINE
*
*   THIS SET OF XMT/RCV ROUTINES IMPLEMENTS A LOOPBACK TEST.
*
*****************************************************************
*
RECV    CALL    GET_DATA      ; GET RECEIVED DATA
        CALL    XCOMPOSE      ; REFORMAT
        CALL    PUT_DATA      ; TRANSMIT SAME DATA
        RET
*
*****************************************************************
*
*   END RCV
*
*****************************************************************
*
*   TRANSMITTER INTERFACE ROUTINE
*
*   PUT_DATA COPIES <AR(OPT_PTR)> TO TDATA AND SET TDA
*
*****************************************************************
*
PUT_DATA LARP   OPT_PTR       ; COPY DATA TO TDATA
        ZALS    *
        SACL    TDATA
*
        ZALS    USTAT         ; SET AND REPORT TDA
        ORK     TDA_SET
        SACL    USTAT
        OUT     USTAT,UARTPORT
*
        RET
*
*****************************************************************
*
```

```
*    END PUT_DATA
*
*****************************************************************
*
*    RECEIVER INTERFACE ROUTINE
*
*    GET_DATA COPIES RDATA TO LOCATION POINTED AT BY AR(INP_PTR) AND CLEARS
*    RDA, ROR, FRM, AND RPE
*
*****************************************************************
*
GET_DATA  LARP     INP_PTR        ; COPY DATA
          ZALS     RDATA
          SACL     *
*
          ZALS     USTAT
          ANDK     RDA_CLR        ; CLEAR RDA
          ANDK     0FFF8h         ; CLEAR ERROR FLAGS
          SACL     USTAT
          OUT      USTAT,UARTPORT
*
          RET
*
*****************************************************************
*
*    END GET_DATA
*
*****************************************************************
*
*    TRANSMIT DATA COMPOSE ROUTINE
*
*    XCOMPOSE ADDS START, STOP, AND PARITY BITS TO DATA POINTED AT BY
*    AR(OPT_PTR)
*
*****************************************************************
*
XCOMPOSE  ZALS     XPARI          ; COMPOSE TRANSMIT WORD
          SACL     XPAR           ; INPUT IS <AR(OPT_PTR))
*                                 ; OUTPUT IS <AR(OPT_PTR))
*
          LARP     OPT_PTR
          ZALS     *
          SACL     TEMP
*
          LAR      XBREG, NM1     ; NUMBER OF TRANSMIT DATA BITS
          LARP     XBREG
*
C_PAR     ZALS     TEMP
          ROR                     ; DETERMINE PARITY BIT VALUE
          SACL     TEMP
          BNC      XNOCHANGE      ; NO PARITY CHANGE IF SPACE
*
          ZALS     XPAR           ; TOGGLE PARITY-GEN BIT IF MARK
          XOR      XPARTOG
          SACL     XPAR
*
XNOCHANGE BANZ     C_PAR
*
          LARP     OPT_PTR
          LAC      *,1            ; ADD START BIT
          OR       XPAR           ; ADD PARITY BIT
          OR       XSTOP          ; ADD STOP BITS
          SACL     *
*
          RET
*
*****************************************************************
*
*    END XCOMPOSE
*
*****************************************************************
*
```