Software UART on the DSP56L811 Using GPIO Port B

by
Glen Wienecke
Motorola
Consumer DSP Applications

Mfax and OnCE are trademarks of Motorola, Inc.

How to reach us:

USA/Europe/Locations Not Listed:
Motorola Literature Distribution
P.O. Box 5405
Denver, Colorado 80217
1 (800) 441-2447

Mfax™:
RMFAX0@email.sps.mot.com
TOUCHTONE (602) 244-6609

Asia/Pacific:
Motorola Semiconductors H.K. Ltd.
8B Tai Ping Industrial Park
51 Ting Kok Road
Tai Po, N.T., Hong Kong
852-2662928

Technical Resource Center:
1 (800) 521-6274

DSP Helpline
dsphelp@dsp.sps.mot.com

Japan:
Nippon Motorola Ltd.
Tatsumi-SPD-JLDC
6F Seibu-Butsuryu-Center
3-14-2 Tatsumi Koto-Ku
Tokyo 135, Japan
81-3-3521-8315

Internet:
www.motorola-dsp.com

Ⓜ **MOTOROLA**

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SECTION 1

UART METHODOLOGY

The UART port is a common interface provided on a vast number of devices, including personal computers. By using this software module in conjunction with minimal external hardware, the DSP56L811 can support a UART interface.

1.1   INTRODUCTION

This application note describes a software module written for the DSP56L811 that allows it to emulate a Universal Asynchronous Receiver Transmitter (UART). Since the serial peripherals available on the DSP56L811 are synchronous devices, they do not have intrinsic UART compatibility. This software module extends the basic DSP56L811 serial communications capability by providing the UART interface code necessary to link to other UART ports.

1.2   BACKGROUND

The UART interface uses synchronization information embedded in serial data packets. These serial packets consist of:

- Start bit to indicate the beginning of a data packet

- Data bits

- Parity bit (optional)

- One or more stop bits to indicate the end of a data packet

In order for a serial transfer to occur between two UART-equipped devices, both the transmitter and receiver must be configured with the same serial communications parameters. These parameters are as follows:

- Baud rate (or bits per second) for the transfer

- Number of data bits in the transfer

- Parity check method, if any, to be used in the transfer

- Number of stop bits

1.3   START BIT

The start bit signifies the beginning of a serial packet. This bit transitions the serial data line from an idle high-level state to a low-level state. The width of the start bit is specified by the baud rate.

1.4   DATA BITS

The data bits immediately follow the start bit. The number of data bits in a packet can vary from five to eight, but eight data bits is the most common. The width of the data bits is specified by the baud rate.

1.5   PARITY BIT

An optional parity bit follows the data bits of the serial packet. Parity is a simple way to detect transmission errors. This error-detection method can only detect single-bit errors reliably, but is well suited for short, less error-prone paths. The most common types of parity used in serial transfers are as follows:

- **No Parity**—No parity bit is embedded into the serial packet; stop bits immediately follow the data bits.

- **Even Parity**—The parity bit in the packet is set or cleared so that the total number of 1s occurring in the data bits combined with the parity bit segment is an even number.

- **Odd Parity**—The parity bit in the packet is set or cleared so that the total number of 1s occurring in the data bits combined with the parity bit segment is an odd number.

1.6   STOP BITS

The stop bits in the packet indicate the end of a transfer. These stop bits are transferred as high-level pulses. The number of stop bits may vary, but at least one stop bit is required for proper synchronization.

1.7   BAUD RATE

The baud rate in this context is equivalent to bits per second (bps) and includes the start bit, parity, and stop bits. For example, a baud rate of 9600 would exhibit a serial bit width characteristic of $1/9600$ or $104.17\ \mu s$ for all bits in the serial packet. The total packet width is determined by the number of actual bits in the packet, not just the number of data bits.

1.8   SERIAL DATA PACKET

The figure below is an example of what a serial data packet might look like.
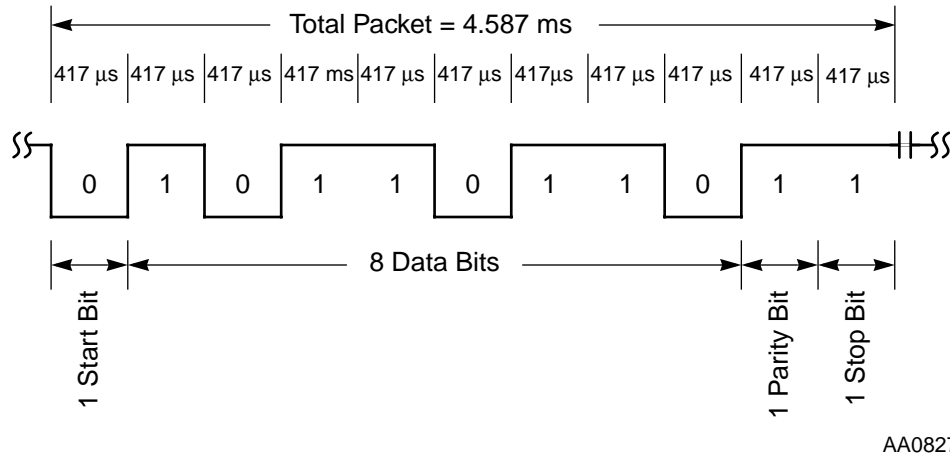
Figure 1-1 Serial Data Packet

The data packet above exhibits the following characteristics:

- Baud rate of 2400 (this results in a bit width of $1/2400 = 417 \mu s$)

- One start bit

- 8 data bits

- Even parity check bit

- One stop bit

SECTION 2

SYSTEM OVERVIEW

Timer interrupts are used at the appropriate bit intervals to sample or transmit the serial data. This interrupt-driven implementation greatly reduces the MIPS requirement of this module by keeping the core intervention to a minimum.

2.1 INTRODUCTION

This software module uses GPIO Port B interrupts in conjunction with timer interrupts to perform the necessary synchronization associated with the UART protocol. GPIO pins PB0–PB7 have the capability of generating an interrupt on a rising/falling edge. One of these pins configured as a receive pin allows for synchronization on each start bit of a serial packet.

2.2 UART EMULATION DETAILS

This section will describe, in detail, the approach used to emulate a hardware UART on the DSP56L811. The purpose of this section is to inform users about what the software is doing so that modifications to the code can be made, thus allowing integration of this software into a target system. **Figure 2-1** is an example of a serial data packet.
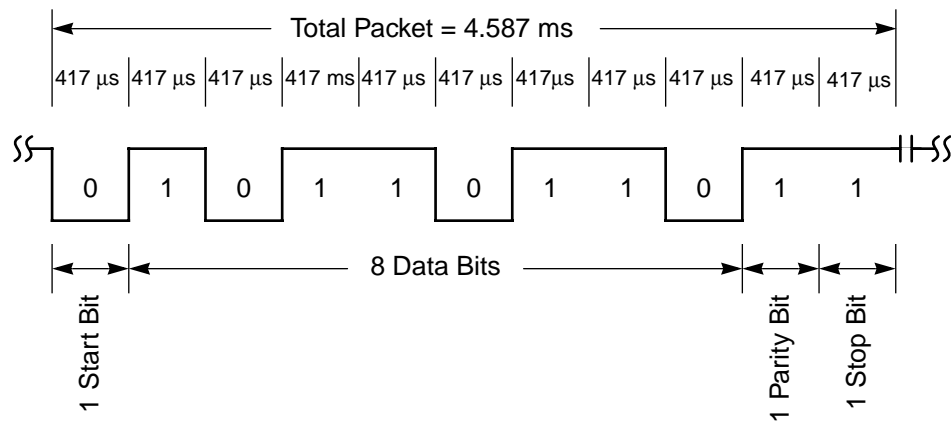
Figure 2-1 Serial Data Packet

2.2.1   Data Packet Description

The data packet above exhibits the following characteristics:

- Baud rate of 2400 (this results in a bit width of $1/2400 = 417\ \mu s$)

- One start bit

- 8 data bits

- Even parity check bit

- One stop bit

2.2.2   Receive Packet Example Description

Figure 2-2 indicates where the described events occur in the UART packet when receiving a data packet.
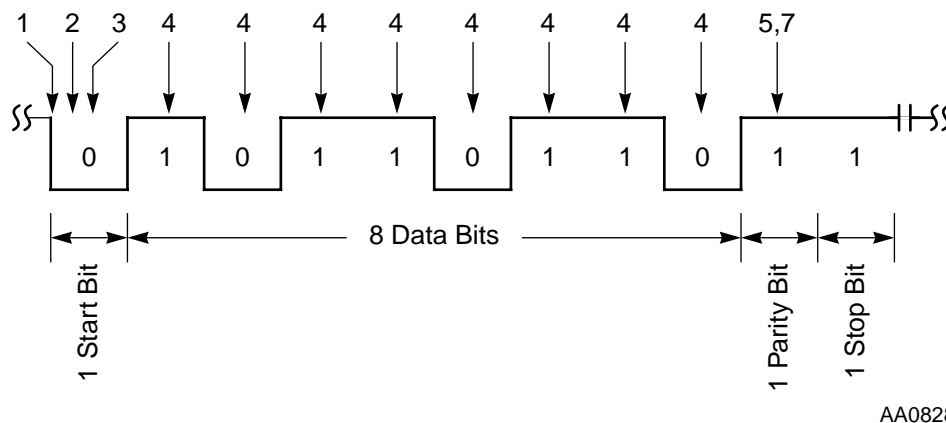
Figure 2-2 Receiving Serial Data Packets

By using this example data packet, we can evaluate how the asynchronous serial data stream is sampled and received. Receiving a serial packet from a connected external device involves the following sequence of events:

1. The falling edge of the receive line generates a GPIO Port B interrupt. The GPIO Port B interrupt service routine programs Timer0 to interrupt the DSP at $1/4$ of the serial bit interval and then disables further GPIO Port B interrupts.

2. The Timer0 interrupt service routine samples the GPIO Port B receive pin to verify the start bit. If the start bit is verified, Timer0 is set to interrupt at $1/4$ the serial bit interval.

3. The Timer0 interrupt service routine samples the GPIO Port B receive pin a second time to verify the start bit. If the start bit is verified, Timer0 is set to interrupt at a serial bit interval.

Note: This sample is made in the middle fo the start bit. This serves as a reference point for subsequent samples.

4. The Timer0 interrupt service routine samples data bits at the middle of each bit interval and stores the sample. Parity check is calculated as each sample is taken and repeated for each data bit in the serial packet.

5. With even parity check enabled (as in this example), the parity bit is sampled after receiving all 8 bits and verified against calculated parity check bit.

6. If the 16 bits of data are transmitted using this method, the process continues, beginning with step 1, until the second 8 bits of data have been sampled (the total data transmission from the connected device is sent as two serial packets with 8 data bits each).

7. After the data is received the Timer0 interrupt service routine disables further Timer0 interrupts and enables GPIO Port B interrupts.

Note: The stop bit is not sampled in order to reduce the MIPS requirement.

2.2.3   Transmit Packet Example Description

Figure 2-3 indicates where the described events occur in the UART packet when transmitting a data packet.

Figure 2-3 Transmitting Serial Data Packets

Asynchronous serial packet transmission is handled in a manner similar to receiving a serial packet. The sequence of events that occurs when a serial packet is transmitted to the connected device is as follows:

1. The user fills a buffer with data to be transmitted and invokes the UART transmit routine.

2. The UART transmit routine drives the GPIO Port B transmit pin low to signal the start bit. Timer1 is set to interrupt at the end of a bit interval.

3. The Timer1 interrupt service routine rotates the LSB out of the data buffer and drives the GPIO Port B transmit pin to the value of the rotated bit. The parity check is calculated as each bit is transmitted. Timer1 is set to interrupt at the end of each bit interval. This process repeats for each data bit in the serial packet.

4. If even parity check is enabled, the Timer1 interrupt service routine generates even parity check and the GPIO Port B transmit pin is driven to the value of the parity check bit. Timer1 is set to interrupt at the end of the bit interval.

5. The Timer1 interrupt service routine drives the GPIO Port B transmit pin high to signal the stop bit. Timer1 is set to interrupt at the end of the bit interval.

6. If 16 bits of data are being transmitted, the Timer1 interrupt service routine drives GPIO Port B transmit pin low to signal the start bit for the second data packet, and repeats the process beginning with step 3 until the second set of 8 data bits have been transmitted (i.e., 16 bits of data are transmitted to the connected device as two serial packets with 8 data bits each).

7. The Timer1 interrupt service routine disables further Timer1 interrupts, and data transmission is complete.

2.3 UART EMULATION CHARACTERISTICS

This software module was designed to meet the important communications configurations available on a variety of UART systems while keeping code as simple as possible. The following list provides the UART emulation characteristics of this module:

• The UART Status Register, UART Control Register, and UART Data Register are located in X data RAM.

• Bits in the UART Status Register are used for Parity Error (PERR) and Overrun (ORUN) error detection.

• Bits in the UART Status Register are used to detect Receive Data Register Full (RDRF) and Transmit Data Register Empty (TDRE).

- The UART emulation uses double-buffered input and single-buffered output.

- The UART emulation recognizes data sizes of 8 bits or 16 bits as seen from the DSP56L811 side (16-bit data is transmitted and received as two UART packets that include 8 data bits).

- The UART emulation uses either no parity check or even parity check for data integrity.

- The UART emulation supports a range of baud rates to support various connected UART devices.

- The maximum recommended baud rate for full-duplex transfers is 19200 baud.

- The maximum recommended baud rate for half-duplex transfers is 38400 baud.

- The emulation uses a single stop bit.

- The 2X oversample on the start bit protects against line noise.

- The emulation uses a single timer interrupt for each transmit/receive bit to minimize core interrupts and MIPS requirement.

- The dual timer interrupt scheme allows for full-duplex operation.

2.4 HARDWARE REQUIREMENTS

The UART hardware requirements vary according to the device that is being connected to the DSP56L811. Personal computer serial ports use the UART protocol with RS-232 voltage levels. The RS-232 lines also carry inverted signals because the typical UART transceiver inverts the RS-232 signal and shifts the voltage levels down to TTL specifications. An additional level shifter may be required to convert the TTL voltage levels down to the CMOS operating voltage levels of the DSP. **Figure 2-4** gives an example of how a PC could be connected to the DSP56L811.
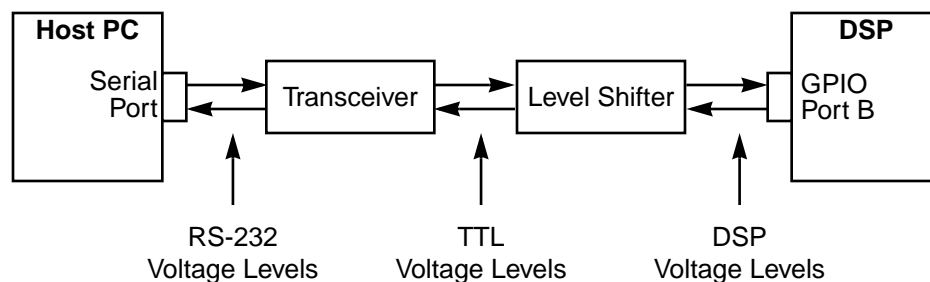
Figure 2-4 UART Hardware Diagram

The transmit and receive lines of the serial port on the PC are connected to the UART transceiver. These signals are then connected to a level shifter to drop the voltage down to DSP levels. Finally, the receive and transmit lines are connect to two GPIO Port B pins on the DSP.

Notes: **1.** The other handshaking lines on the Host PC serial port are not used, and this software module does not provide for hardware handshaking.

2. Some UART transceivers provide level shifting down to DSP voltage levels, thus eliminating the need for an additional level shifter.

2.4.1   Interrupt Restrictions

Because the UART uses a GPIO and timer interrupt scheme for bit synchronization, special care must be taken to assure that these interrupts are not masked for long durations. Masking can cause erroneous bit synchronizing that can result in loss of data integrity. Make sure that other user interrupt routines that can be executed during the a serial transfer are short, so that they do not mask the UART interrupts for long periods of time.

2.4.2   UART Sampling

To keep the implementation simple and reduce the MIPS requirement of this software module, the sampling of bits during a receive operation is limited. Most hardware UARTs oversample each data bit and then use some scheme to determine the bit value from the oversamples. This allows for greater data integrity. The sampling approach used in this module is a 2X oversample on the start bit, then single-sampling for each subsequent bit. The sampling is done in the middle of the serial bit. Noise on the line could produce an erroneous bit sampling, and there is no oversampling to protect against this. Even with the limited sampling scheme, an even parity check is available to reliably detect single bit errors. This scheme seems well suited for short, less error-prone serial data paths.

2.4.3   Transfer Speed Restrictions

The speed at which the software UART can transfer serial data is constrained by the DSP56L811 clock speed. The maximum recommended rates for full-duplex and half-duplex transfers are 19200 and 38400 baud, respectively for a DSP core clock of

40 MHz. Using a slower clock requires slower baud rates for proper bit synchronization.

Another transfer speed constraint is the total MIPS required for the software UART routines. Refer to **Section 2.5** for more information on the MIPS requirements.

2.5   RESOURCE REQUIREMENTS

Unlike a hardware UART peripheral, the software implementation requires more core intervention and resources. The following resources provide the necessary tools to allow the module to emulate a hardware UART:

- Timer0 for receive bit synchronization
- Timer0 interrupt vector
- Timer1 for transmit bit synchronization
- Timer1 interrupt vector
- Two GPIO Port B pins for transmit and receive lines (receive must be PB0–PB7)
- GPIO interrupt vector
- Seven words of X memory RAM space for UART registers and internal variable space
- 208 words of Program RAM space for UART code
- Core MIPS (requirement increases with faster baud rates)

To describe the MIPS requirement for this module accurately, we must look at the actual core cycles spent on executing instructions for the UART routines and the time that it takes to send a packet of serial data. **Table 2-1** provides an analysis of the DSP core clock cycle requirements for transmit and receive operations based on the particular communications configurations.

Note: These clock cycle counts were determined using the DSP56800 simulator.

Table 2-1   Instruction Cycle Requirements for UART Operations

Task	8-bit Data No Parity	8-bit Data Even Parity	16-bit Data No Parity	16-bit Data Even Parity
Receive	1018	1110	2014	2202
Transmit	1020	1118	2014	2208

Table 2-2 shows the packet interval information required to determine data transmission time.

Table 2-2   Packet Interval for UART Operations

	8-bit data No Parity	**8-bit Data Even Parity**	**16-bit Data No Parity**	**16-bit Data Even Parity**
Receive	$\dfrac{8.5}{\text{baud rate}}$	$\dfrac{9.5}{\text{baud rate}}$	$\dfrac{17}{\text{baud rate}}$	$\dfrac{19}{\text{baud rate}}$
Transmit	$\dfrac{10}{\text{baud rate}} + \dfrac{94}{\text{DSP core clk}}$	$\dfrac{11}{\text{baud rate}} + \dfrac{94}{\text{DSP core clk}}$	$\dfrac{20}{\text{baud rate}} + \dfrac{94}{\text{DSP core clk}}$	$\dfrac{22}{\text{baud rate}} + \dfrac{94}{\text{DSP core clk}}$

Note: The packet interval values were determined by looking at the number of bit intervals required for the transmit and receive operations.

Note: The transmit operation takes an additional 94 core clock cycles to initiate a transmission.

The values in the two tables above can be used to calculate the MIPS for a specific communications configuration. The following equation can be used to calculate the required MIPS:

$$\text{MIPS Requirement} = \frac{\text{Clock Cycle Requirement}}{\text{Packet Interval}} \times \frac{\text{Instruction}}{2 \text{ Cycles}} \times \frac{1}{10^6}$$

The following examples illustrate how to calculate the MIPS requirement.

Example 2-**1**   UART Receive Operation MIPS Requirement Calculation

- *Example 1*: 8-bit data
- Even parity
- 19200 baud
- DSP core clock = 40 MHz
- Clock Cycle Requirement = 1110 cycles (**Table 2-1** on page 2-9)
- Packet Interval = 9.5 bits / 19200 bps = 494.79 μs (**Table 2-2** on page 2-10)

$$\text{MIPS Requirement} = \frac{1110 \text{ Cycles}}{494.79 \text{ μs}} \times \frac{\text{Instruction}}{2 \text{ Cycles}} \times \frac{1}{10^6} = 1.12$$

Example 2-**2**   UART Transmit Operation MIPS Requirement Calculation

- 16-bit data
- No parity
- 9600 baud
- DSP core clock = 40 MHz
- Clock Cycle Requirement = 2014 cycles (**Table 2-1** on page 2-9)
- Packet Interval = 20 bits / 9600 bps + 94 / 40 MHz = 2.09 ms (**Table 2-2** on page 2-10)

$$\text{MIPS Requirement} = \frac{2014 \text{ Cycles}}{2.09 \text{ ms}} \times \frac{\text{Instruction}}{2 \text{ Cycles}} \times \frac{1}{10^6} = 0.483$$

Note: All of the calculations and formulas for the MIPS requirements are based on estimations of cycle counts provided by simulation. Actual MIPS requirements may vary according to the user's application of this software module.

SECTION 3

USING THE SOFTWARE UART

The programming model for this module was designed to provide UART configuration capability similar to that of a hardware UART. The programming model uses a UART Control Register, a UART Status Register, and a UART Data Register.

3.1   INTRODUCTION

The functionality of the UART registers introduces an important difference between a hardware UART and the software implementation. Certain bits of status registers in hardware UARTs are altered as the data registers are read or written. For example, a bit indicating that the receive buffer is full may be cleared when the receive buffer register is read. This is not the case in the software implementation. The *user is responsible* for setting or clearing the appropriate status bits to ensure proper operation, because the registers of the software UART are simply X memory locations.

This section provides information on the software UART register set. Each register can be referenced from inside user code by using the appropriate equate string from the UART software module with the X memory reference. For example, the UART Status Register equate is *uartSR*, so this register can be referenced by using *x:uartSR* as an operand for instructions.

Note: All registers are 16 bits wide.

3.2   UART CONTROL REGISTER—x:uartCR

The bits of this register are used to program the communications configuration for the software UART. **Table 3-1** describes the meaning of each bit.

Table 3-1 UART Control Register Bit Descriptions

Bit No.	Bit Name	Mask	Usage	= 0	= 1	Default
15–2	N/A	N/A	N/A	N/A	N/A	N/A
1	DSIZ	uartDSIZ	data size flag	8-bit data	16-bit data	0
0	PTYP	uartPTYP	parity type flag	no parity	even parity	0

Note: The default for both programmable bits is 0. The default values are set during the execution of the configure communications routine provided in this module.

3.2.1 PTYP Bit

The PTYP bit configures the Parity Type used by the receive and transmit operations of the UART. If programmed as a 0, no parity will be used on transmit and receive operations. If programmed as a 1, even parity will be used on transmit and receive operations. The two parity types available can be programmed as follows:

```
bfclr      #uartPTYP,x:uartSR     ; set PTYP = 0
bfset      #uartPTYP,x:uartSR     ; set PTYP = 1
```

When configured for even parity, the data received is checked against the parity bit for even parity. If an error occurs, the PERR bit in the UART Status Register is set. In addition, the even parity bit is calculated and transmitted during a UART transmission.

3.2.2 DSIZ Bit

The DSIZ bit is used to configure the Data Size for UART transfers. If programmed as a 0, a data size of 8 bits is used. If programmed as a 1, a data size of 16 bits is used. The data sizes available can be programmed as follows:

```
bfclr      #uartDSIZ,x:uartSR     ; set DSIZ = 0
bfset      #uartDSIZ,x:uartSR     ; set DSIZ = 1
```

Note: This configuration is only used on the DSP56L811 side. Because the data size in most UART devices is 8 bits and the data size in the DSP56L811 is 16 bits, this can be a useful setting. The 16-bit data is actually received and transmitted as two packets that include 8 data bits. Hence, the connected device must complete two packet transfers for every packet transfer on the DSP56L811.

3.3   UART STATUS REGISTER—x:uartSR

The bits of this register are used to determine the current status of the UART operations. **Table 3-2** describes the meaning of each bit.

Table 3-2 UART Control Register Bit Descriptions

Bit No.	Name	Mask	Usage	Bit = 0	Bit = 1	Default
15–8	N/A	N/A	N/A	N/A	N/A	N/A
7	TXPB	N/A	Transmit Parity Bit	DO NOT MODIFY	DO NOT MODIFY	N/A
6	TXHI	N/A	Transmit High/low byte flag	DO NOT MODIFY	DO NOT MODIFY	N/A
5	RXPB	N/A	Receive Parity Bit	DO NOT MODIFY	DO NOT MODIFY	N/A
4	RXHI	N/A	Receive Hi/low byte flag	DO NOT MODIFY	DO NOT MODIFY	N/A
3	PERR	uartPERR	Parity Error flag	no parity error	parity error	0
2	ORUN	uartORUN	Overrun error flag	no overrun	overrun error	0
1	TDRE	uartTDRE	Transmit Data Register Empty flag	Transmit Data Register full	Trans Data Register empty	1
0	RDRF	uartRDRF	Receive Data Register Full flag	Receive Data Register empty	Receive Data Register full	0

The default values are set during the execution of the configure communications routine provided in module.

3.3.1　PERR Bit

The PERR bit is the flag for Parity Errors. If even parity is being used (PTYP = 1), then this bit is set if a parity error occurs during the reception of a serial packet.

Note:　This bit is not cleared by the UART routines. The *user is responsible* for clearing this bit after a parity error has occurred.

An example of how this bit might be used is shown in the following code:

```
bftsth      #uartPERR,x:uartSR    ; if PERR = 1
bcs         parity_error          ; goto parity_error
```

3.3.2　ORUN Bit

The Overrun (ORUN) error bit is used to detect overrun errors. An overrun occurs when the UART data register is already full (RDRF = 1) and another serial packet is received. This means that the data currently in the UART data register will be overwritten with the newly received data. Since the receive routines feature double-buffering, the user has the time interval before the next serial packet is received to read the data register and clear the Receive Data Register Full flag.

An example of how this bit might be used is shown in the following code.

```
bftsth      #uartORUN,x:uartSR    ; if ORUN = 1
bcs         overrun_error         ; goto overrun_error
```

3.3.3　TDRE Bit

The Transmit Data Register Empty (TDRE) bit indicates when the Transmit Data Register is empty. If this bit is set (TDRE = 1), then transmission is in progress and the user should not invoke the UART transmit routine. If this bit is clear (TDRE = 0), then the transmit data register is empty and a new transmission can occur. Since the transmit routines are single-buffered, failure to poll the TDRE bit prior to performing a serial transmit will result in an erroneous transmission. An example of how this bit might be used is shown in the following code:

```
L1          brclr      #uartTDRE,x:uartSR,L1 ; if TDRE=0 goto L1
            move       #$FFFF,x0             ; x0 = $FFFF
            lea        (sp)+                 ; advance sp
            move       x0,x:(sp)                    ; pass parm on stack
            jsr        uart_Transmit         ; invoke transmit
            pop                              ; fix stack
```

3.3.4 RDRF Bit

The Receive Data Register Full (RDRF) bit is used to detect when new data has been put into the UART data register. By polling this bit, the user can detect when the UART data register contains valid data. When this bit is set (RDRF = 1) the UART data register contains new data. When this bit is clear (RDRF = 0) then no new data has arrived.

Note: The UART routines do not clear this bit. *The user is responsible for clearing this bit when data has be read out of the UART data register.*

An example of how this bit might be used is shown in the following code:

```
L1          brclr       #uartRDRF,x:uartSR,L1  ; if RDRF=0 goto L1
            move        x:uartDR,x0            ; x0 = receive data
            bfclr       #uartRDRF,x:uartSR     ; RDRF = 0
```

3.4 UART DATA REGISTER—x:uartDR

The UART Data Register is the receive data buffer that contains valid data when the RDRF (Receive Data Register Full) bit of the UART Status Register is set. The data in this buffer should be read and the RDRF bit should be cleared by the user before the next serial data is received to prevent the ORUN (Overrun) error bit from being set in the UART Status Register.

APPENDIX A

UART FILES

The uart.asm file contains all of the assembly code necessary for the software implementation. In addition, a large header comment describes the steps necessary to get the code up and running. The uartapp.asm file shows how to use the UART routines that configure the communications settings, receive serial data, and transmit serial data.

A.1   INTRODUCTION

The files provided to get the user started include *uart.asm*, the UART module itself, and *uartapp.asm*, an example application using the routines of the UART module. This appendix gives a full listing of both files.

A.2   uart.asm FILE LISTING

```
;-----------------------------------------------------------------------------
;                   (C) Motorola, Inc. 1996 -- All rights reserved.
;
; Module:       uart.asm
;
; Version:           1.0
;
; Purpose:      The purpose of this module is to emulate a hardware UART
;               (Universal Asynchronous Receiver Transmitter) communications
;               interface using software.  The 56800 GPIO port b is used
;               for the transmit and receive lines of the serial data
;               streams.  Additionally, GPIO and timer module interrupt
;               service routines provide the necessary synchronization to
;               a user-specified baud rate.
;
; Modification
;     History:  None
;
; Resources:    The implementation of the UART requires the following
;               resources:
;               - Timer 0 of timer module for receive bit synchronization
;               - Timer 0 interrupt vector
;               - Timer 1 of timer module for transmit bit synchronization
;               - Timer 1 interrupt vector
;               - Two GPIO port b pins for receive/transmit (must be PB0-PB7)
;               - GPIO interrupt vector
;               - 7 words of x-memory space for UART registers and variables
;               - 208 words of p-memory space for UART instructions
;               - Core MIPS increasing with faster baud rates
;
;                 Example:  9600 baud; 8-bit data; even parity
;                 MIPS Requirement = 0.54 MIPS
;                 (see application note for further MIPS analysis)
;
; Specs:        The UART routines can be configured by the user to
;               match the capabilities of the device connected to the 56800.
;               The following configurations are available:
;               - Variable baud rates (Note:  baud rates up to 38400
;                 have been successful in a lab environment)
;               - Even parity check or no parity check
```

```
;                       - Data sizes of 8-bit or 16-bit values as seen from the DSP
;                         (Note:  16-bit data is transmitted/received as two 8-bit
;                          values)
;                       - One stop bit
;
; Routines:      This module consists of two user-callable routines, three
;                interrupt service routines, and one utility routine.  See
;                comments above module routines for descriptions.
;
;                       User-Callable Routines:
;                       uart_Config_Com - configures UART to default settings
;                       uart_Transmit - initiates a UART transmit
;
;                       Interrupt Service Routines:
;                       uart_GPIO_ISR - interrupt service routine for gpio port b
;                       uart_TIMER0_ISR - interrupt service routine for timer 0
;                       uart_TIMER0_ISR - interrupt service routine for timer 1
;
;                       Utility Routine:
;                       uart_Start_TX - required for proper operation of UART code
;
; X-Mem Space:  X-memory space is used by this module for UART status,
;               control, state information, and data buffers.  The following
;               provides a description of each x-memory word used by this
;               module.
;
;       uartSR - UART Status Register
;       ----------------------------------------------------------------------
;       | ********** | TXPB | TXHI | RXPB | RXHI | PERR | ORUN | TDRE | RDRF |
;       ----------------------------------------------------------------------
;        15        8   7      6      5      4      3      2      1      0
;
;       ----------------------------------------------------------------------
;       BITS   NAME   USE               = 0                 = 1
;       ----------------------------------------------------------------------
;       15-8   N/A    N/A               N/A                 N/A
;          7   TXPB   tx parity         USED INTERNALLY; DO NOT MODIFY
;          6   TXHI   tx hi/lo byte     USED INTERNALLY; DO NOT MODIFY
;          5   RXPB   rx parity         USED INTERNALLY; DO NOT MODIFY
;          4   RXHI   rx hi/lo byte     USED INTERNALLY; DO NOT MODIFY
;          3   PERR   parity error      no parity error     parity error
;          2   ORUN   overrun error     no overrun          overrun error
;          1   TDRE   tx data reg empty tx data reg full    tx data reg empty
;          0   RDRF   rx data reg full  rx data reg empty   rx data reg full
;
;
;       uartCR - UART Control Register
;       ----------------------------------------------------------------------
;       | **************************************************** | DSIZ | PTYP |
;       ----------------------------------------------------------------------
;        15                                                      2    1    0
;
;       ----------------------------------------------------------------------
```

```
;        BITS   NAME   USE              = 0               = 1
;        ----------------------------------------------------------------
;        15-2  N/A    N/A              N/A               N/A
;           1  DSIZ   data size        8-bit data        16-bit data
;           0  PTYP   parity type      no parity         even parity
;
;
;        uartRXC - UART Receive Count Register (DO NOT MODIFY)
;
;        uartTXC - UART Transmit Count Register (DO NOT MODIFY)
;
;        uartRXR - UART Receive Shift Register (DO NOT MODIFY)
;
;        uartTXR - UART Transmit Shift Register (DO NOT MODIFY)
;
;        uartDR - UART Data Register (contains valid data when RDRF = 1)
;
; How to run    This module is designed to be included at the end of the
; this code:    existing user code.  The user must modify this module to
;               to allow for proper code integration.  The following
;               description lists the steps necessary prior to executing
;               this code.  The user can use a search of each step to find
;               the point in this code that needs modifications.  For
;               instance, do a text search on STEP 0 to find the point of
;               the module that needs modification for step 0.
;
;               STEP 1  Modify the equate value uartXBASE that determines
;                       the base address in x-memory where UART variables
;                       will be placed.
;
;                       Example - UART x-memory storage begins at x:$40
;                       uartXBASE    equ    $40
;
;               STEP 2  Specify the GPIO pins to be used for the receive
;                       and transmit lines of the UART.  Do this by
;                       modifying the equate values uartPBTX, uartPBRX,
;                       and uartPBIRUPT.  Note that only PB0-PB7 can be
;                       used as the receive pin since they are the only
;                       ones capable of generating transitional interrupts.
;                       The value of uartPBIRUPT is written to the PBINT
;                       port B control register.  The upper and lower bytes
;                       of the uartPBIRUPT value should coincide with the
;                       lower byte of uartPBRX.
;
;                       Example 1 - PB0 = Receive; PB1 = Transmit
;                       uartPBRX     equ    $0001
;                       uartPBTX     equ    $0002
;                       uartPBIRUPT  equ    $0101
;
;                       Example 2 - PB2 = Receive; PB3 = Transmit
;                       uartPBRX     equ    $0004
;                       uartPBTX     equ    $0008
;                       uartPBIRUPT  equ    $0404
```

```
;
;               STEP 3  Calculate the UART bit rate using the following
;                       formula and modify the uartBR equate.
;                       Note: round down to nearest integer
;
;                       uartBR = 1 / [baud rate * (4 / CPU clock rate)] - 1
;
;                       Example - DSP running at 40MHz, 19200 baud
;                       uartBR = 1 / [19200 * (4 / 40e6)] - 1 = 519
;
;               STEP 4  If desired add user code to uart_TIMER0_ISR that
;                       will be executed upon completion of a serial data
;                       receive.
;
;               STEP 5  If desired add user code to uart_TIMER1_ISR that
;                       will be executed upon completion of a serial data
;                       transmit.
;
;               STEP 6  Use the include directive to add the code in this
;                       module to user code.  This code should be added at
;                       the end of the user code.
;
;                       Example
;                       ; last line of user code
;                               include 'uart.asm'
;
;               STEP 7  Follow the calling requirements of the UART
;                       routines in user code.  See comments above module
;                       routines for more information.
;
;               See uartapp.asm for an application example of this module.
;-------------------------------------------------------------------------



;**************************
;* ********************** *
;* * UART Module Equates * *
;* ********************** *
;**************************


;***** Addresses *****;
uartIPR         equ     $fffb           ;addr of interrupt priority reg
uartPBD         equ     $ffec           ;addr of port b data reg
uartPBDDR       equ     $ffeb           ;addr of port b data direction reg
uartPBINT       equ     $ffea           ;addr of port b interrupt reg
uartTCR01       equ     $ffdf           ;addr of timer01 control reg
uartTPR0        equ     $ffde           ;addr of timer0 preload reg
uartTCT0        equ     $ffdd           ;addr of timer0 count reg
uartTPR1        equ     $ffdc           ;addr of timer1 preload reg
uartTCT1        equ     $ffdb           ;addr of timer1 count reg

;***** Constants *****;
```

```
;--------------------------------------------------------------------------------
;               STEP 1  Modify the equate value uartXBASE that determines
;                       the base address in x memory where storage for
;                       internal UART variables will be placed.
;--------------------------------------------------------------------------------
uartXBASE       equ     $100               ;base address for xmem var storage


;--------------------------------------------------------------------------------
;               STEP 2  Specify the GPIO pins to be used for the receive
;                       and transmit lines of the UART.  Do this by
;                       modifying the equate values uartPBTX,
;                       uartPBRX, and uartPBIRUPT.
;--------------------------------------------------------------------------------
uartPBTX        equ     $0001              ;PBDDR (data dir) serial transmit mask
uartPBRX        equ     $0002              ;PBDDR (data dir) serial receive mask
uartPBIRUPT     equ     $0202              ;PBINT (port b interrupt reg) setting

uartENABLET0    equ     $0090              ;timer0 control bit mask
uartENABLET1    equ     $9000              ;timer1 control bit mask

;***** uartCR MASKS *****;
uartPTYP        equ     $0001              ;mask for PTYP bit
uartDSIZ        equ     $0002              ;mask for DSIZ bit

;***** uartSR MASKS *****;
uartRDRF        equ     $0001              ;mask for RDRF bit
uartTDRE        equ     $0002              ;mask for TDRE bit
uartORUN        equ     $0004              ;mask for ORUN bit
uartPERR        equ     $0008              ;mask for PERR bit
uartRXHI        equ     $0010              ;mask for RXHI bit
uartRXPB        equ     $0020              ;mask for RXPB bit
uartTXHI        equ     $0040              ;mask for TXHI bit
uartTXPB        equ     $0080              ;mask for TXPB bit


;--------------------------------------------------------------------------------
;               STEP 3  Calculate the UART bit rate using the following
;                       formula and modify the uartBR equate.
;                       Note: round down to nearest integer
;
;                       uartBR = 1 / [baud rate * (4 / CPU clock rate)] - 1
;--------------------------------------------------------------------------------
uartBR          equ     519                ;UART rate is 19200 @ 40MHz


;*****************************
;* ************************* *
;* * UART Module Subroutines * *
;* ************************* *
;*****************************

;****************************************************************************
; Routine:      uart_Config_Com
; Purpose:      This routine configures the necessary peripherals and
```

```
;                   initializes UART status and control registers.
; Parameters:   N/A
;****************************************************************************
uart_Config_Com
        move    #$0000,x:uartTCR01              ;disable timer0/timer1
        move    #uartBR,x:uartTPR0              ;timer0 preload reg = uartBR
        move    #uartBR,x:uartTPR1              ;timer1 preload reg = uartBR


        bfset   #$0100,sr                       ;enable all levels of irupts
        bfclr   #$0200,sr                       ;enable all levels of irupts


        bfset   #$8800,x:uartIPR                ;enable GPIO/timer interrupts
        bfset   #uartPBTX,x:uartPBDDR           ;configure serial output pin
        bfclr   #uartPBRX,x:uartPBDDR           ;configure serial input pin
        move    #uartPBIRUPT,x:uartPBINT        ;set up port b interrupts
        bfset   #uartPBTX,x:uartPBD             ;drive serial output pin high
        move    #$0000,x:uartSR                 ;reset UART status reg
        move    #$0000,x:uartCR                 ;reset UART control reg
        bfset   #uartTDRE,x:uartSR              ;uartTRDE flag = 1
        rts


;****************************************************************************
; Routine:      uart_GPIO_ISR
; Purpose:      This routine is the interrupt service routine for the GPIO
;               port b pins.  When a falling edge occurs on the port b pin
;               configured as the receive line, an interrupt occurs and this
;               routine is invoked.
; Parameters:   N/A
; Notes:        DO NOT INVOKE DIRECTLY
;****************************************************************************
uart_GPIO_ISR
        move    #uartBR/4-4,x:uartTCT0          ;timer0 cnt reg = qrate
        bfset   #uartENABLET0,x:uartTCR01       ;enable timer0 with irupts
        move    #$0000,x:uartPBINT              ;turn off portb irupts
        move    #$0000,x:uartRXC                ;rx_cnt = 0
        bfclr   #uartRXPB,x:uartSR              ;reset parity check bit
        rti


;****************************************************************************
; Routine:      uart_TIMER0_ISR
; Purpose:      This routine is the interrupt service routine for timer0.
;               When the value in the timer0 count register goes to zero,
;               an interrupt occurs and this routine is invoked.  This
;               routine handles the UART receive operation.  Upon completion
;               of the UART reception, the received data is put into
;               uartDR and the RDRF flag is asserted high (RDRF = 1).
; Parameters:   N/A
; Notes:        DO NOT INVOKE DIRECTLY
;
;               ** the RDRF flag must be high (RDRF = 1) before the contents
;                  of uartDR is valid
;
;               ** if uartDR is not read and the RDRF flag is not cleared
```

```
;                    (RDRF = 0) before the reception of the next serial data,
;                    the ORUN flag of the uartSR will be set and the data in
;                    uartDR is lost
;*************************************************************************
uart_TIMER0_ISR
        lea     (sp)+                    ;advance stack pointer
        move    x0,x:(sp)+               ;push x0 onto stack
        move    y0,x:(sp)                ;push y0 onto stack

; DETERMINE RECEIVE STATE
        move    x:uartRXC,x0             ;x0 = rx count
        cmp     #0,x0                    ;if x0 == 0
        beq     uartSAMP0                ;goto SAMP0
        cmp     #1,x0                    ;if x0 == 1
        beq     uartSAMP1                ;goto SAMP1

; RECEIVE DATA BITS 1-8/PARITY BIT STATE
        move    x:uartRXR,y0             ;y0 = previous sample
        bftsth  #uartPBRX,x:uartPBD      ;sample rx pin, set carry
        ror     y0                       ;rotate carry into y0
        bge     uartNoToggle             ;if rx=0, goto NoToggle
        bfchg   #uartRXPB,x:uartSR       ;else toggle parity check bit

uartNoToggle
        cmp     #10,x0                   ;if x0 == 10
        beq     uartSAMP10               ;goto uartSAMP10

        move    y0,x:uartRXR             ;current sample = y0
        cmp     #9,x0                    ;if x0 == 9
        beq     uartSAMP9                ;goto uartSAMP9
        bra     uartRXend                ;goto uartRXend

; RECEIVE DATA BIT 8 STATE
uartSAMP9
        brset   #uartPTYP,x:uartCR,uartRXend ;if PBIT=1 goto uartRXend
        bra     uartChkDSIZ              ;else goto ChkDSIZ

; RECEIVE PARITY BIT STATE
uartSAMP10
        move    x:uartRXR,y0             ;y0 = previous sample
        brclr   #uartRXPB,x:uartSR,uartChkDSIZ ;if RXPB=0 goto ChkDSIZ
        bfset   #uartPERR,x:uartSR       ;else set parity error bit
        bra     uartChkDSIZ              ;goto ChkDSIZ

; RECEIVE FIRST START BIT OVERSAMPLE STATE
uartSAMP0
        bfclr   #uartENABLET0,x:uartTCR01   ;disable timer0
        move    #uartBR/4-10,x:uartTCT0      ;timer0 cnt reg = qrate
        bfset   #uartENABLET0,x:uartTCR01   ;enable timer0 with irupts

; RECEIVE SECOND START BIT OVERSAMPLE STATE
uartSAMP1
        brclr   #uartPBRX,x:uartPBD,uartRXend ;if rx pin=0 goto uartRXend
```

```
        bra     uartRXDone                      ;goto RXDone

; CHECK DATA SIZE CONTROL BIT
uartChkDSIZ
        brset   #uartDSIZ,x:uartCR,uart16In     ;if DSIZ=1 goto 16In
        move    #8,x0                           ;else x0=shift amount (8)
        lsrr    y0,x0,y0                        ;shift sample into low byte
        bra     uartEndSamp                     ;goto EndSamp

; DATA SIZE IS 16 BITS
uart16In
        bfchg   #uartRXHI,x:uartSR              ;test and toggle RXHI flag
        bcc     uartRXDone                      ;if RXHI=0 goto RXDone

; END OF SAMPLING, STORE SAMPLE IN INPUT BUFFER
uartEndSamp
        brclr   #uartRDRF,x:uartSR,uartNoORUN   ;if RDRF=0 goto NoORUN
        bfset   #uartORUN,x:uartSR              ;else set overrun bit

; NO OVERRUN ERROR, SIGNAL THAT DATA HAS BEEN RECEIVED
uartNoORUN
        move    y0,x:uartDR                     ;inbuf = current sample
        bfset   #uartRDRF,x:uartSR              ;RDRF flag = 1
;---------------------------------------------------------------------------
;               STEP 4  If desired add user code to uart_TIMER0_ISR that
;                       will be executed upon completion of a serial data
;                       receive.  Add your code here.
;---------------------------------------------------------------------------

; CONFIGURE TO WAIT FOR NEXT START BIT
uartRXDone
        bfclr   #uartENABLET0,x:uartTCR01       ;disable timer0
        move    #uartPBIRUPT,x:uartPBINT        ;set up port b irupts

uartRXend
        incw    x0                              ;increment sample cnt
        move    x0,x:uartRXC                    ;update sample count
        pop     y0                              ;restore y0
        pop     x0                              ;restore x0
        rti

;***************************************************************************
; Routine:      uart_Transmit (tdata)
; Purpose:      This user-callable routine can be invoked to transmit data
;               via the software UART.  The data to be transmitted is passed
;               to the routine on the software stack.
; Parameters:   tdata = transmit data
; Notes:        This routine should be invoked as follows:
;
;                       move    #tdata,reg      ;reg = tdata
;                       lea     (sp)+           ;advance sp
;                       move    reg,x:(sp)      ;push reg onto stack
;                       jsr     uart_Transmit   ;invoke transmit routine
```

```
;                      pop                    ;pop passed parameter
;
;              where    tdata = immediate operand transmit data
;                       reg = core register
;
;              ** the TDRE bit in uartSR must be high (TDRE = 1) before
;                 invoking this funciton
;*****************************************************************************
uart_Transmit
        lea     (sp)+                         ;advance stack pointer
        move    x0,x:(sp)                     ;push x0 onto stack
        move    x:(sp-3),x0                   ;x0 = transmit data
        move    x0,x:uartTXR                  ;uartTXR = x0
        bfclr   #uartTDRE,x:uartSR            ;TDRE = 0
        jsr     uart_Start_TX                 ;invoke Start_TX routine
        pop     x0                            ;restore x0
        rts


;*****************************************************************************
; Routine:      uart_Start_TX
; Purpose:      This routine initiates a UART serial transmit by modifying
;               the appropriate peripheral and UART register space.
; Parameters:   N/A
; Notes:        DO NOT INVOKE DIRECTLY
;*****************************************************************************
uart_Start_TX
        bfclr   #uartPBTX,x:uartPBD           ;drive tx pin low
        move    #uartBR,x:uartTCT1            ;timer1 count reg = rate
        bfset   #uartENABLET1,x:uartTCR01     ;enable timer1 with irupts
        move    #$0000,x:uartTXC              ;reset transmit count
        bfclr   #uartTXPB,x:uartSR            ;reset transmit parity
        rts


;*****************************************************************************
; Routine:      uart_TIMER1_ISR
; Purpose:      This routine is the interrupt service routine for timer1.
;               When the value in the timer1 count register goes to zero,
;               an interrupt occurs and this routine is invoked.  This
;               routine handles the UART transmit operation.
; Parameters:   N/A
; Notes:        DO NOT INVOKE DIRECTLY
;*****************************************************************************
uart_TIMER1_ISR
        lea     (sp)+                         ;advance stack pointer
        move    x0,x:(sp)+                    ;push x0 onto stack
        move    y0,x:(sp)                     ;push y0 onto stack

; DETERMINE TRANSMIT STATE
        move    x:uartTXC,y0                  ;y0 = transmit count
        cmp     #8,y0                         ;if y0 = 8
        beq     uartTRANS8                    ;goto TRANS8
        cmp     #9,y0                         ;if y0 = 9
        beq     uartTRANS9                    ;goto TRANS9
```

```
        cmp     #10,y0                  ;if y0 = 10
        beq     uartEndTrans            ;goto EndTrans

; TRANSMIT DATA BITS 1-8 STATE
        move    x:uartTXR,x0            ;x0 = previous transmit word
        ror     x0                      ;rotate LSB into carry
        move    x0,x:uartTXR            ;update tx register
        bcs     uartTXHigh              ;if carry set, goto TXHigh

; WRITE A ZERO
uartTXLow
        bfclr   #uartPBTX,x:uartPBD     ;drive output pin low
        bra     uartNextTX              ;goto NextTX

; WRITE A ONE
uartTXHigh
        bfset   #uartPBTX,x:uartPBD     ;drive output pin high
        bfchg   #uartTXPB,x:uartSR      ;toggle the parity send bit
        bra     uartNextTX              ;goto NextTX

; END OF STOP BIT/TRANSMIT STOP BIT STATE
uartTRANS9
        brclr   #uartPTYP,x:uartCR,uartEndTrans ;if PBIT = 0, goto EndTrans
        bra     uartTXHigh              ;else goto uartTXHigh

; TRANSMIT PARITY/STOP BIT STATE
uartTRANS8
        brclr   #uartPTYP,x:uartCR,uartTXHigh  ;if PBIT = 0, goto EndTrans
        brset   #uartTXPB,x:uartSR,uartTXHigh  ;if TXPB = 1, goto TXHigh
        bra     uartTXLow               ;else goto TXLow

; GET READY TO TRANSMIT NEXT BIT
uartNextTX
        incw    y0                      ;increment transmit count
        move    y0,x:uartTXC            ;update transmit count
        bra     uartTXEnd               ;goto TXEnd

; TRANSMIT IS OVER
uartEndTrans
        bfclr   #uartENABLET1,x:uartTCR01  ;disable timer1
        brclr   #uartDSIZ,x:uartCR,uartTXDone  ;if DSIZ=0 goto TXDone
        bfchg   #uartTXHI,x:uartSR      ;else test and toggle TXHI
        bcs     uartTXDone              ;if TXHI=1 goto TXDone
        jsr     uart_Start_TX           ;invoke Start_TX
        bra     uartTXEnd               ;goto TXEnd

; SIGNAL THAT DATA HAS BEEN TRANSMITTED
uartTXDone
        bfset   #uartTDRE,x:uartSR      ;TDRE flag = 1
;----------------------------------------------------------------------------
;            STEP 5  If desired add user code to uart_TIMER1_ISR that
;                    will be executed upon completion of a serial data
;                    transmit.  Add your code here.
```

```
;------------------------------------------------------------------------------
uartTXEnd
        pop     y0                              ;restore y0
        pop     x0                              ;restore x0
        rti



;**************************
;* ********************** *
;* * UART X-Memory Space * *
;* ********************** *
;**************************
        org     x:uartXBASE
uartSR          ds      1                       ;UART status register
uartCR          ds      1                       ;UART control register
uartRXC         ds      1                       ;number reads on receive pin
uartTXC         ds      1                       ;number writes on transmit pin
uartRXR         ds      1                       ;UART receive shift register
uartTXR         ds      1                       ;UART transmit shift register
uartDR          ds      1                       ;UART data register



;*****************************************
;* ************************************* *
;* * UART Interrupt Vector Table Entries * *
;* ************************************* *
;*****************************************
        org     p:$0014
        jsr     uart_GPIO_ISR                   ;GPIO interrupt

        org     p:$0018
        jsr     uart_TIMER0_ISR                 ;Timer0 interrupt

        org     p:$001A
        jsr     uart_TIMER1_ISR                 ;Timer1 interrupt
;------------------------------------------------------------------------------
```

A.3   uartapp.asm FILE LISTING

```
;-------------------------------------------------------------------------
;
;                 (C) Motorola, Inc. 1996 -- All rights reserved.
;
; Module:        uartapp.asm
;
; Version:          1.0
;
; Purpose:       The purpose of this module is to provide an example
;                application of the software module used to emulate a
;                hardware UART on the DSP56800.
;
;                This module will configure the DSP56L811 for receiving
;                a series of UART data blocks from a host device.  The
;                blocks are formatted as UART packets with 8-bit data and
;                parity check bit.  The first byte of the block serves as
;                a header to specify the number of UART packets to follow.
;                The packets should appear across the serial transfer line as
;                follows:
;
;                # of packets,packet[0],packet[1],...,packet[# of packets - 1]
;
;                After receiving all of the packets, as specified in the
;                block header, a response is sent to the host device.  This
;                response is an echo of the block header if the transfer
;                was successful, else the response is a zero.  If the
;                response is non-zero, the block of UART packets received
;                are then echoed back to the host.
;
; Modification
;      History: None
;
; How to run
; this code:     This code includes the file 'uart.asm' which contains
;                the UART emulation software for the 56L811.  The following
;                description lists the steps necessary prior to executing
;                this code.  The user can use a search of each step to find
;                the point in this code that needs modifications.  For
;                instance, do a text search on STEP 0 to find the point of
;                the module that needs modification for step 0.
;
;                STEP 1  Modify the PLL setup code to work with the clock
;                        for the 56L811.
;
;                STEP 2  Make sure the relative path of the module 'uart.asm'
;                        is correct.  For example:
;
;                        include 'uart.asm'         ;same dir as uartapp.asm
;                        include 'c:\code\uart.asm'   ;in 'c:\code' dir
;
```

```
;                    STEP 3  Go to the module 'uart.asm' and follow the steps
;                            in the "How to run this code" section.
;
;                    STEP 4  Assemble this file with the 56800 assembler.
;
;-----------------------------------------------------------------------------


; ***** ADDRESSES ***** ;
START           equ    $0080                    ;start of program
BUF             equ    $0400                    ;array addr for UART transfers
BCR             equ    $FFF9                    ;Bus Control Reg
IPR             equ    $FFFB                    ;Interrupt Priority Reg
PCR0            equ    $FFF2                    ;PLL Control Reg 0
PCR1            equ    $FFF3                    ;PLL Control Reg 1


; ***** RESET VECTOR ***** ;
        org    p:$0000
        jmp    START


;-----------------------------------------------------------------------------
; MAIN PROGRAM
;-----------------------------------------------------------------------------
        org    p:START

; Initialize BCR for zero wait states
        move   #0,x:BCR


;-----------------------------------------------------------------------------
;                STEP 1  Modify the PLL setup code to work with the clock
;                        for the 56L811.
;-----------------------------------------------------------------------------
; Initialize PLL (Multily 20MHz by 2) set YD=1
        move    #$0020,x:PCR0

; Delay to meet the 10ms lock spec
        move    #$1fff,lc
        do      lc,delay1
        nop
delay1
        move    #$1fff,lc
        do      lc,delay2
        nop
delay2

; Set PLLE to enable PLL
        bfset   #$4080,x:PCR1

; Set stack pointer to first location after page 0
        move    #$40,sp

; Enable all levels of interrupts
```

```
            bfset   #$0100,sr
            bfclr   #$0200,sr


;-----------------------------------------------------------------------------
; Configure the UART by invoking uart_Config_Com.  Set the UART control
; register for 8-bit transfers with even parity checking enabled.
;-----------------------------------------------------------------------------
            jsr     uart_Config_Com             ;configure UART
            bfset   #uartPTYP,x:uartCR          ;PTYP = 1 (even parity)
            bfclr   #uartDSIZ,x:uartCR          ;DSIZ = 0 (8-bit data)


;-----------------------------------------------------------------------------
; Begin the main loop
;-----------------------------------------------------------------------------
loop    move    #BUF,r0                         ;r0 = addr of UART data


;-----------------------------------------------------------------------------
; Read the header byte from the host that contains the number of bytes that
; will follow.  Polling of the RDRF bit is used to determine if UART packet
; transfer is complete.
;-----------------------------------------------------------------------------
header  brclr   #uartRDRF,x:uartSR,header    ;if RDRF = 0, goto header
            move    x:uartDR,y0                 ;y0 = number of bytes
            move    y0,x0                       ;x0 = number of bytes
            bfclr   #uartRDRF,x:uartSR          ;clear RDRF flag in uartSR


;-----------------------------------------------------------------------------
; Reading data bytes from the host.  Polling of the RDRF bit is used to
; determine if UART packet transfer is complete.
;-----------------------------------------------------------------------------
rxwait  brclr   #uartRDRF,x:uartSR,rxwait    ;if RDRF = 0, goto rxwait
            move    x:uartDR,y1                 ;y1 = received UART data
            move    y1,x:(r0)+                  ;BUF[r0] = y1
            bfclr   #uartRDRF,x:uartSR          ;clear RDRF flag in uartSR
            decw    y0                          ;repeat for all data
            bgt     rxwait


;-----------------------------------------------------------------------------
; Check the status register to see if a parity error occurred during the
; UART transfer.  If a parity error has occurred, clear x0 which will be
; echoed back to the host as a response.
;-----------------------------------------------------------------------------
            bftstl  #uartPERR,x:uartSR          ;if PERR = 0
            bcs     no_err                      ;   goto no_err
            clr     x0                          ;report error to host


;-----------------------------------------------------------------------------
; Check the status register to see if an overrun error occurred during the
; UART transfer.  If an overrun error has occurred, clear x0 which will be
; echoed back to the host as a response.
;-----------------------------------------------------------------------------
no_err  bftstl  #uartORUN,x:uartSR              ;if ORUN = 0
            bcs     txwait1                      ;   goto txwait1
```

```
        clr    x0                          ;report error to host

;-------------------------------------------------------------------------
; Echo the value in register x0 back to the host.  If the header was
; received successfully and no error occurred during the transfer, this
; register will contain the original header from the host.  This provides
; a response of the transfer status to the host.  Polling of the TDRE bit
; in the status register is used to determine if the transmit data register
; is empty before starting a new UART transmit.
;-------------------------------------------------------------------------
txwait1 brclr  #uartTDRE,x:uartSR,txwait1   ;if TDRE = 0, goto txwait1
        lea    (sp)+                       ;advance sp
        move   x0,x:(sp)                   ;push x0
        jsr    uart_Transmit               ;invoke transmit
        pop                                ;readjust sp for parm


;-------------------------------------------------------------------------
; Echo the values received x0 back to the host.
;-------------------------------------------------------------------------
        move   #BUF,r0                     ;r0 = addr of UART data
txloop  cmp    #0,x0                       ;if (x0 <= 0) then
        ble    loop                        ;goto top of loop
txwait2 brclr  #uartTDRE,x:uartSR,txwait2   ;if TDRE = 0, goto txwait2
        move   x:(r0)+,y0                  ;get value from buffer
        lea    (sp)+                       ;advance sp
        move   y0,x:(sp)                   ;push buffer value
        jsr    uart_Transmit               ;invoke transmit
        pop                                ;readjust sp for parm
        decw   x0
        bra    txloop

;-------------------------------------------------------------------------
; Include the UART software module
;
;              STEP 2  Make sure the relative path of the module 'uart.asm'
;                      is correct.
;-------------------------------------------------------------------------
        include 'uart.asm'
```